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:**

«База данных интернет-магазина видеоигр с применением технологии Stand By-серверов СУБД»

Выполнил студент Н.А. Турчинович

(Ф.И.О.)

Руководитель проекта асс. Н.И. Уласевич

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

Консультант: асс. Н.И. Уласевич.

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

Нормоконтролер: асс. Н.И. Уласевич.

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

Курсовой проект защищен с оценкой

Содержание

[Введение 5](#_Toc166901231)

[1 Аналитический обзор литературы по теме проекта 6](#_Toc166901232)

[1.1 Постановка задачи 6](#_Toc166901233)

[1.1.1 Аналог Steam 6](#_Toc166901234)

[1.1.2 Epic Games Store 7](#_Toc166901235)

[1.2 Функциональные требования 8](#_Toc166901236)

[1.3. Определение вариантов использования 9](#_Toc166901237)

[1.4 Вывод 10](#_Toc166901238)

[2 Анализ и проектирование архитектуры проекта. 11](#_Toc166901239)

[2.1 UML схема базы данных 11](#_Toc166901240)

[2.2 Описание информационных объектов и ограничений целостности 11](#_Toc166901241)

[2.3 Вывод 14](#_Toc166901242)

[3 Проектирование и разработка базы данных 15](#_Toc166901243)

[3.1 Табличные пространства базы данных 15](#_Toc166901244)

[3.2 Создание основных ролей и пользователей базы данных 16](#_Toc166901245)

[3.3 Вывод 18](#_Toc166901246)

[4 Разработка необходимых объектов 19](#_Toc166901247)

[4.1 Создание таблиц 19](#_Toc166901248)

[4.2 Создание триггеров 20](#_Toc166901249)

[4.3 Создание индексов 21](#_Toc166901250)

[4.4 Создание представлений 21](#_Toc166901251)

[4.5 Создание процедур 22](#_Toc166901252)

[4.6 Создание функций 23](#_Toc166901253)

[4.7 Создание пакетов процедур и функций для базы данных 23](#_Toc166901254)

[4.8 Создание пользовательских типов 24](#_Toc166901255)

[4.9 Вывод 25](#_Toc166901256)

[5 Описание процедур экспорта и импорта 26](#_Toc166901257)

[5.1 Экспорт данных в JSON 26](#_Toc166901258)

[5.2 Импорт данных из JSON 27](#_Toc166901259)

[5.3 Вывод 28](#_Toc166901260)

[6 Использование технологии Stand By-серверов СУБД 29](#_Toc166901261)

[6.1 Настройка Stand By 29](#_Toc166901262)

[6.2 Проверка работы Stand By 30](#_Toc166901263)

[6.3 Вывод 31](#_Toc166901264)

[7 Тестирование производительности базы данных 32](#_Toc166901265)

[7.1 Тестирование производительности 32](#_Toc166901266)

[7.2 Вывод 33](#_Toc166901267)

[Заключение 34](#_Toc166901268)

[Список используемых источников 35](#_Toc166901269)

[Приложение А Листинг создания таблиц 36](#_Toc166901270)

[Приложение Б Листинг создания триггеров 40](#_Toc166901271)

[Приложение В Листинг создания пакетов 42](#_Toc166901272)

[Приложение Г Листинг процедур экспорта 49](#_Toc166901273)

# Введение

Цель данного проекта состоит в создании реляционной базы данных для интернет-магазина видеоигр, обеспечивая доступ пользователю ко всем товарам, имеющимся в нем. Также требуется использовать подходящую технологию Stand By сервера для обеспечения безопасности данных в базе данных.

База данных представляет собой совокупность взаимосвязанных данных, обычно сохраняемых в электронном виде в компьютерной системе. Она используется для хранения, организации и управления как структурированными, так и неструктурированными данными. Реляционная база данных является наиболее распространенной формой организации, где данные представлены в виде таблиц, состоящих из строк и столбцов, где каждый столбец представляет атрибут, а каждая строка – кортеж или запись.

В данном проекте для управления базой данных была выбрана СУБД "Oracle" благодаря ее высокой надежности и производительности, что обеспечивает эффективное хранение, обработку и управление данными.

Для обеспечения стабильности приложения используется технология Stand By сервера, которая предполагает наличие дополнительного, резервного сервера. Эта технология позволяет создать резервную копию данных, которая синхронизируется с основной базой данных в реальном времени, что гарантирует минимальные потери данных в случае сбоя основного сервера.

Пояснительная записка содержит информацию о сопоставимых продуктах, структуре и реализации проекта, а также инструкции по использованию приложения.

1. Аналитический обзор литературы по теме проекта

## Постановка задачи

Сегодня интернет-магазины стали неотъемлемой частью нашей повседневной жизни, предоставляя доступ к огромному ассортименту товаров и услуг прямо из уютной обстановки дома. В связи с этим, для создания базы данных интернет-магазина необходимо тщательно изучить и объективно оценить возможности функционала, применяемых технологий и других аспектов.

После проведения анализа требуется составить конкретный перечень функциональных требований к базе данных и создать диаграмму вариантов использования. Основная задача проекта заключается в разработке архитектуры приложения с понятным для любого пользователя интерфейсом, построении базы данных и проведении тестирования готового продукта.

Основные требования к реализации:

* Определение ролей (Издатель, Пользователь);
* Возможность добавления, удаления и редактирования видеоигры (Издатель);
* Поиск видеоигр по категориям и критериям (Пользователь);
* Добавление видеоигры в корзину или избранное (Пользователь);
* Возможность добавление товара в корзину или избранное (Пользователь);
* Покупка видеоигры (Пользователь)
* Взаимодействие с базой данных при помощи хранимых процедур и функций.

### 1.1.1 Аналог Steam

Steam [1] является одной из ведущих платформ для цифровой дистрибуции видеоигр и программного обеспечения. Основным преимуществом этой платформы является ее удобство и простота использования. Пример интерфейса данного сервиса представлен на рисунке 1.1.

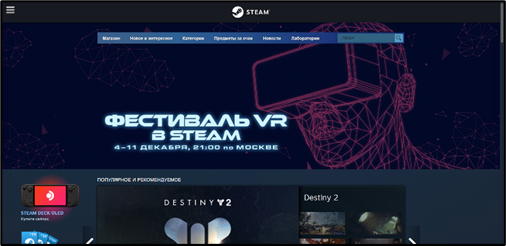


Рисунок 1.1 – Интерфейс сервиса Steam

Ключевая функциональность платформы Steam – это доступ к широкому ассортименту компьютерных игр. Пользователи могут легко найти игры по жанру, разработчику, цене или популярности. Кроме того, Steam предоставляет персонализированные рекомендации на основе игровой истории пользователя, а также список игр, рекомендованных другими игроками и кураторами платформы.

Пользователи могут создавать собственные списки желаемых игр и делиться ими с друзьями или сохранять игры для последующего приобретения. Steam также предоставляет пользователям обширную информацию о каждой игре, включая описание, отзывы пользователей и технические характеристики.

Другая важная функция Steam – это возможность использовать платформу как социальную сеть для игроков. Пользователи могут добавлять друзей, просматривать их списки игр и рекомендации, общаться через чат и форумы.

Платформа использует множество баз данных в своей архитектуре, включая системы управления базами данных для хранения информации о играх, разработчиках, отзывах и т.д. Эти системы также используются для хранения информации о пользовательских аккаунтах, достижениях и настройках.

В общем, Steam использует масштабируемые и высокопроизводительные базы данных для обеспечения быстрого доступа к огромному объему информации о играх и обработки данных для предоставления персонализированных рекомендаций и других функций платформы.

### 1.1.2 Epic Games Store

Epic Games Store [3] предлагает широкий ассортимент игр, включая как классические хиты, так и самые новые релизы. Сайт обеспечивает удобный поиск по различным категориям и ценовым диапазонам, а его дизайн прост и дружелюбен для пользователей.

Однако, среди недостатков можно выделить иногда высокую загруженность сайта, что может затруднить опыт пользователя, а также не всегда легкость в поиске конкретных игр. Уникальной особенностью Epic Games Store является возможность быстрой регистрации через социальные сети. Пример интерфейса данного сервиса представлен на рисунке 1.2.

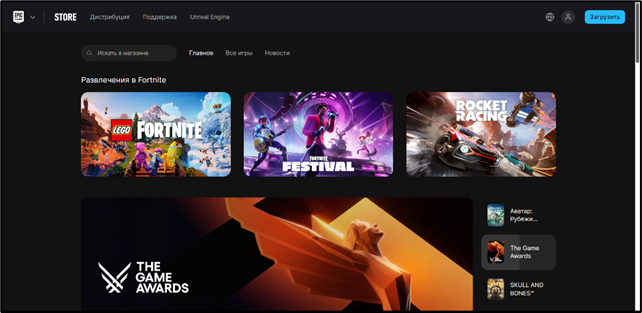


Рисунок 1.2 – Интерфейс сервиса Epic Games Store

Epic Games Store, также опирается на разнообразные базы данных в своей архитектуре. Для хранения информации о пользователях, включая данные об учетных записях, платежах и покупках, они, вероятно, используют реляционную базу данных, такую как MySQL. При этом для эффективного управления метаданными о продуктах, такими как описания, изображения и технические характеристики игр, Epic Games Store может прибегать к использованию NoSQL базы данных, например, MongoDB или DynamoDB.

Кроме того, Epic Games Store, вероятно, воспользуется облачной инфраструктурой, такой как Amazon Web Services или Microsoft Azure, для обеспечения высокой доступности и масштабируемости своей платформы. Это обеспечивает надежность сервиса и возможность масштабирования в соответствии с увеличивающимся потоком пользователей и объемом данных.

В общем, Epic Games Store использует современные и масштабируемые базы данных, чтобы обеспечить эффективный доступ к разнообразной информации о продуктах и управление данными о пользователях и их покупках. Однако конкретные детали об архитектуре и используемых технологиях могут быть доступны только внутри компании Epic Games.

## 1.2 Функциональные требования

Функциональные требования базы данных определяют способы обработки данных и предоставления пользователю определенной роли необходимой функциональности. Это включает в себя указание на способы хранения и структурирования данных, методы поиска и выборки данных, процессы обновления информации и механизмы защиты данных. Кроме того, такие требования могут охватывать интеграцию базы данных с другими системами и программным обеспечением. Например, для интернет-магазина видеоигр функциональные требования могут включать функции для хранения информации о видеоиграх и изображениях, возможность поиска товаров по различным категориям и критериям, создание и удаление товаров, управление товарами в корзине покупателя, а также функции оценки товаров.

По результатам рассмотренных аналогов основные задачи для базы данных интернет-магазина видеоигр:

* разработать процедуры и функции для обработки информации и действий пользователя;
* разработать функционал добавления и удаления товаров из корзины;
* разработать структурную модель базы данных;
* разработать функционал оценки видеоигры;
* разработать функционал для покупки видеоигры;
* разработать функционал редактирования/удаления пользователя/товара.
* разработать функционал поиска товаров.

## 1.3. Определение вариантов использования

Помимо функциональных требований, важно также определить роли пользователей и их варианты использования системы. Варианты использования описывают, как пользователи будут взаимодействовать с системой в зависимости от своих ролей. Варианты использования обычно представляются в виде UML диаграмм, которые позволяют наглядно отобразить взаимодействие между пользователями и системой.

В зависимости от роли пользователя, он может иметь доступ к различным функциям системы. В данном проекте роли пользователей будут следующими:

* Гость
* Пользователь
* Издатель
* Администратор

На основе предоставленного списка ролей необходимо построить варианты использования. Варианты использования изображены на рисунке 1.4.

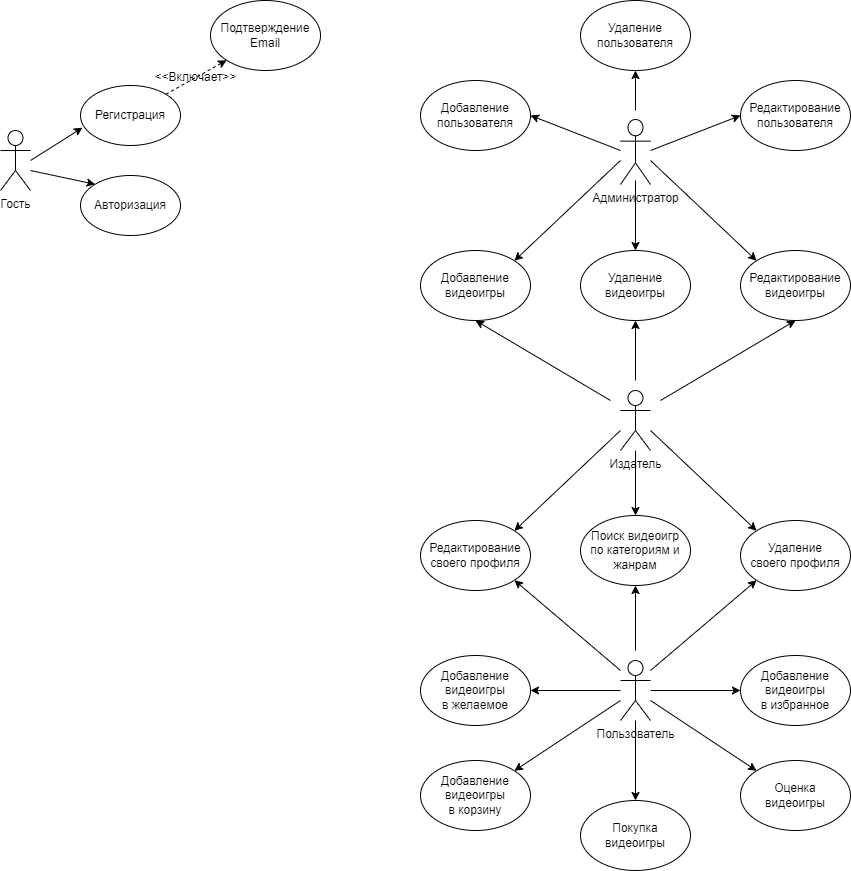


Рисунок 1.3 – UML диаграмма вариантов использования

В начале работы с приложением пользователь является гостем. Ему будет доступна только просмотр всей информации без возможности добавления видеоигры в корзину. После регистрации пользователь становится зарегистрированным пользователем.

Роль Пользователь получает возможность осуществления поиска товаров по различным критериям, добавления видеоигр в список желаемого, корзину и избранное, покупке и оценке товара.

Роль Издатель получает возможность добавлять новые видеоигры на площадку, а также редактировать и удалять уже существующие видеоигры. Он является управляющим контентом площадки.

Ролям Пользователь и Издатель доступно изменение данных собственного профиля, а также его удаление. К примеру, они могут изменить свои имена или пароли.

Роль Администратор заключается в управлении площадкой и мониторинге действий пользователей. Администратор имеет доступ к просмотру информации о всех пользователях, в том числе их действиях на площадке. При необходимости Администратор может редактировать информацию о видеоиграх и пользователях. Администратору доступен функционал смены ролей пользователей.

## 1.4 Вывод

Проведен аналитический обзор аналогов интернет-магазинов видеоигр, существующих на рынке, для определения основных характеристик и функциональных возможностей, необходимых в разрабатываемой базе данных. Этот обзор помог выявить требования к базе данных и роли пользователей, а также варианты использования приложения в зависимости от этих ролей.

На основе анализа были выделены функциональные требования базы данных. Это включает в себя хранение информации о видеоиграх, их описаниях, изображениях, ценах, наличии на складе, а также информацию о пользователях, заказах и их статусах. Была также выявлена необходимость в функциях по управлению инвентарем, обработке платежей и генерации отчетов.

Для удобства проектирования и понимания системы была разработана UML-диаграмма, на которой отображены основные функции, доступные для каждой из ролей пользователей. Например, администратор системы имеет доступ к функциям управления инвентарем и пользователями, в то время как обычные пользователи могут осуществлять поиск, просмотр информации о продуктах и совершать заказы.

Этот аналитический подход позволил учесть разнообразные потребности пользователей и создать базу данных, которая будет эффективно поддерживать функционирование интернет-магазина видеоигр, обеспечивая удобство использования и надежность операций.

1. Анализ и проектирование архитектуры проекта.

## 2.1 UML схема базы данных

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

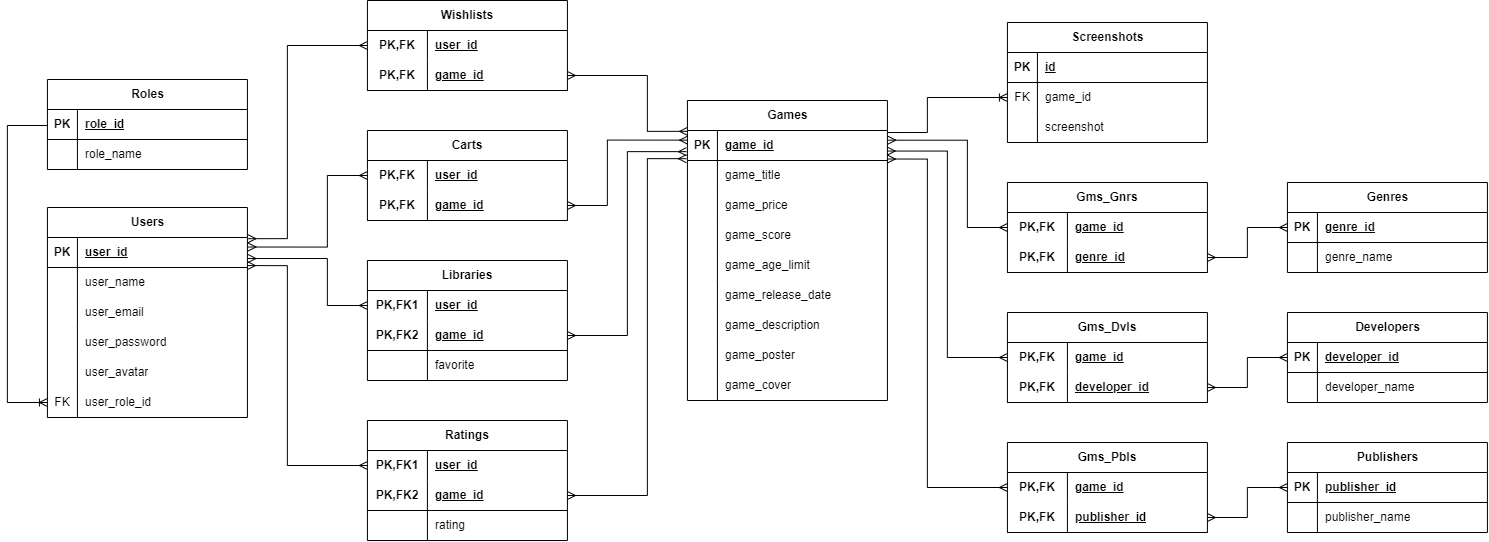


Рисунок 2.1 – Схема базы данных

Таким образом, схема показывает связи между таблицами и полями, а также типы отношений между ними, такие как связи "один-ко-многим" и "многие-ко-многим". Например, таблица Users связана с таблицами Roles, Wishlists, Carts, Libraries и Ratings через внешние ключи user\_id. Также видно, что таблица Games связана с таблицами Genres, Developers и Publishers через дополнительные таблицы для организации связей "многие-ко-многим". Кроме того, таблица Screenshots связана с таблицей Games через внешний ключ game\_id. Связи между таблицами Users и Roles и таблицами Games и Screenshots представляют собой связи "один-ко-многим".

## 2.2 Описание информационных объектов и ограничений целостности

Для реализации базы данных было разработано 14 таблиц. В структуру схемы базы данных для проекта входят следующие таблицы: Roles, Users, Wishlists, Carts, Libraries, Ratings, Games, Screenshots, Genres, Developers, Publishers и дополнительные Gms\_Gnrs, Gms\_Dvls, Gms\_Pbls. Ниже приведено описание подробное описание каждой таблицы, включающее типы данных столбцов и ограничения целостности. Описание представлено в таблице 2.2.

Таблица 2.2 − Описание ограничений целостности

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Roles | role\_id | PK | Идентификатор роли | INTEGER |
| role\_name |  | Название роли | NVARCHAR2(255) |
| Users | user\_id | PK | Идентификатор пользователя | INTEGER |
| user\_name |  | Имя пользователя | NVARCHAR2(255) |
| user\_email |  | E-mail пользователя | NVARCHAR2(255) |
| user\_password |  | Хеш пароля пользователя | NVARCHAR2(255) |
| user\_avatar |  | Аватар пользователя | BLOB |
| user\_role\_id | FK | Идентификатор роли пользователя | INTEGER |
| Genres | genre\_id | PK | Идентификатор жанра | INTEGER |
| genre\_name |  | Название жанра | NVARCHAR2(255) |
| Developers | developer\_id | PK | Идентификатор разработчика | INTEGER |
| developer\_name |  | Название разработчика | NVARCHAR2(255) |
| Publishers | publisher\_id | PK | Идентификатор издателя | INTEGER |
| publisher\_name |  | Название издателя | NVARCHAR2(255) |
| Games | game\_id | PK | Идентификатор видеоигры | INTEGER |
| game\_title |  | Название видеоигры | NVARCHAR2(255) |
| game\_price |  | Стоимость видеоигры | NUMBER(15, 2) |
| game\_score |  | Оценка критиков | NUMBER(3, 0) |
| game\_age\_limit |  | Возрастное ограничение | NUMBER(2, 0) |
| game\_release\_date |  | Дата выхода видеоигры | DATE |
| game\_description |  | Описание видеоигры | NVARCHAR2(2000) |
| game\_poster |  | Постер видеоигры | BLOB |
| game\_cover |  | Обложка видеоигры | BLOB |
| Screenshots | id | PK | Идентификатор скриншота | INTEGER |
| game\_id | FK | Идентификатор видеоигры | INTEGER |
| screenshot |  | Скриншот | BLOB |
| Gms\_Gnrs | game\_id | PK FK | Идентификатор видеоигры | INTEGER |
| genre\_id | PK FK | Идентификатор жанра | INTEGER |
| Gms\_Dvls | game\_id | PK FK | Идентификатор видеоигры | INTEGER |
| developers\_id | PK FK | Идентификатор разработчика | INTEGER |
| Gms\_Pbls | game\_id | PK FK | Идентификатор видеоигры | INTEGER |
| publisher\_id | PK FK | Идентификатор издателя | INTEGER |
| Wishlists | user\_id | PK FK | Идентификатор пользователя | INTEGER |
| game\_id | PK FK | Идентификатор видеоигры | INTEGER |
| Carts | user\_id | PK FK | Идентификатор пользователя | INTEGER |
| game\_id | PK FK | Идентификатор видеоигры | INTEGER |
| Libraries | user\_id | PK FK | Идентификатор пользователя | INTEGER |
| game\_id | PK FK | Идентификатор видеоигры | INTEGER |
| favorite |  | Добавлена ли игра в избранное | CHAR(1) |
| Ratings | user\_id | PK FK | Идентификатор пользователя | INTEGER |
| game\_id | PK FK | Идентификатор видеоигры | INTEGER |
| rating |  | Пользовательская оценка видеоигры | NUMBER(2, 0) |

Таким образом, была определена структура таблиц и их ограничений.

## 2.3 Вывод

Анализ и проектирование модели базы данных представляют собой важный и ответственный этап разработки любой информационной системы. На этом этапе закладывается фундамент, от которого во многом зависит эффективность, масштабируемость и надежность будущей системы. В ходе выполнения данного этапа была разработана UML-схема базы данных, отражающая структуру и взаимосвязи её компонентов.

UML-схема базы данных представляет собой графическое отображение структурных элементов базы данных, таких как таблицы, поля и связи между ними. Эта схема наглядно демонстрирует, каким образом таблицы взаимодействуют друг с другом, какие типы данных хранятся в полях, и какие отношения существуют между различными сущностями. На схеме указаны все основные компоненты базы данных, что позволяет лучше понять её структуру и упрощает дальнейшую разработку и поддержку системы.

Для реализации базы данных была разработана структура, состоящая из четырнадцати таблиц, каждая из которых выполняет свою уникальную функцию и содержит определенные данные. Основные таблицы включают в себя: Roles, Users, Genres, Developers, Publishers, Games.

Кроме этих основных таблиц, в структуру базы данных включены и другие таблицы, обеспечивающие дополнительные функции и хранящие вспомогательную информацию. Каждая из таблиц связана с другими таблицами через внешние ключи, что обеспечивает целостность данных и позволяет эффективно управлять информацией.

Таким образом, разработанная UML-схема базы данных и структура таблиц обеспечивают четкую и логичную организацию данных, что является основой для создания надежной и масштабируемой системы. Внимательное проектирование и анализ на данном этапе позволяют предусмотреть возможные сложности и минимизировать риски, связанные с разработкой и эксплуатацией базы данных.

# 3 Проектирование и разработка базы данных

## 3.1 Табличные пространства базы данных

Для интернет-магазина видеоигр потребуется создать несколько табличных пространств, которые будут хранить связанные таблицы.

Пространство TS\_USER необходимо для хранения данных таблиц, связанных с пользователями. К таким таблицам относятся Users и Roles. Пример скрипта создания табличного пространства будет в листинге 3.1. Остальные табличные пространства будут аналогичны.

|  |
| --- |
| ---------- USERS TABLESPACE ----------  CREATE TABLESPACE TS\_USER  DATAFILE 'D:\F5STORETS\TS\_USER.DBF'  SIZE 100M  REUSE  ;  ---------- GAMES TABLESPACE ----------  CREATE TABLESPACE TS\_GAME  DATAFILE 'D:\F5STORETS\TS\_GAME.DBF'  SIZE 100M  REUSE  ;  ---------- LISTS TABLESPACE ----------  CREATE TABLESPACE TS\_LIST  DATAFILE 'D:\F5STORETS\TS\_LIST.DBF'  SIZE 100M  REUSE  ; |

Листинг 3.1 – Табличные пространства

В пространстве TS\_GAME хранятся данные, связанные с информацией о видеоиграх, такие как жанры, разработчики, издатели, снимки экрана, а также таблицы, образующие связи "многие-ко-многим".

Пространство TS\_LIST хранит данные списков, такие как корзины, библиотеки и списки желаемых видеоигр пользователей.

Для каждого табличного пространства были созданы новые файлы для хранения информации, а при создании пользователя разработчику была выделена квота на добавление данных в табличные пространства пример в листинге 3.2.

|  |
| --- |
| CREATE USER DEVELOPER  IDENTIFIED BY F5STOREDBDeveloper  QUOTA UNLIMITED ON TS\_USER  QUOTA UNLIMITED ON TS\_GAME  QUOTA UNLIMITED ON TS\_LIST; |

Листинг 3.2 – Выделение квоты на пространства

Таким образом, разработчик не имеет лимита на размещение данных в созданных табличных пространствах.

## 3.2 Создание основных ролей и пользователей базы данных

Для проекта было выделено несколько основных ролей, участвующих в разработке базы данных для интернет-магазина видеоигр с ролями и привилегиями для пользователей и ограничениями доступа к базе данных.

В нем определены такие пользователи, как Administrator и Developer, каждый из которых имеет свои роли и привилегии.

Например, Administrator получает полные права администратора базы данных через ALL PRIVILEGES, в то время как Developer получает ограниченную роль и соответствующие привилегии для работы с данными. Пример администратора базы данных показана в листинге 3.1.

|  |
| --- |
| ---------- ROLE ----------  CREATE ROLE ADMINISTRATOR\_ROLE;  GRANT  ALL PRIVILEGES  TO ADMINISTRATOR\_ROLE;  ---------- USER ----------  CREATE USER ADMINISTRATOR  IDENTIFIED BY F5STOREDBAdministrator;  GRANT ADMINISTRATOR\_ROLE TO ADMINISTRATOR; |

Листинг 3.1 – Пользователь Administrator

Developer имеет привилегии для создания таблиц, представлений, процедур и индексов. Пример создания роли вместе с пользователем и выданными привилегиями будет на листинге 3.2.

|  |
| --- |
| ---------- ROLE ----------  CREATE ROLE DEVELOPER\_ROLE;  GRANT  CONNECT,  CREATE TYPE,  CREATE ROLE,  CREATE USER,  CREATE TABLE,  CREATE VIEW,  CREATE TRIGGER,  CREATE PROCEDURE,  CREATE SEQUENCE,  CREATE SYNONYM,  CREATE PUBLIC SYNONYM,  DROP PUBLIC SYNONYM,  CREATE ANY DIRECTORY  TO DEVELOPER\_ROLE;  ---------- USER ----------  CREATE USER DEVELOPER  IDENTIFIED BY F5STOREDBDeveloper  QUOTA UNLIMITED ON TS\_USER  QUOTA UNLIMITED ON TS\_GAME  QUOTA UNLIMITED ON TS\_LIST  ;  GRANT DEVELOPER\_ROLE TO DEVELOPER;  GRANT CONNECT TO DEVELOPER WITH ADMIN OPTION; |

Листинг 3.2 – Пользователь Developer

Пользователем Developer были так же созданы пользователи типа Client и Publisher для представления им доступа к соответствующему функционалу и дальнейшего наглядного примера использования. Пользователи играют ключевую роль в выполнении запросов и вызова процедур Пример создания пользователей и выдача привилегий будет на листинге 3.3.

|  |
| --- |
| ---------- ROLES ----------  ---------- VIEW ACCESS ROLE ----------  CREATE ROLE VIEW\_ACCESS\_ROLE;  GRANT SELECT ON V\_ROLES TO VIEW\_ACCESS\_ROLE;  GRANT SELECT ON V\_USERS TO VIEW\_ACCESS\_ROLE;  GRANT SELECT ON V\_GENRES TO VIEW\_ACCESS\_ROLE;  GRANT SELECT ON V\_DEVELOPERS TO VIEW\_ACCESS\_ROLE;  GRANT SELECT ON V\_PUBLISHERS TO VIEW\_ACCESS\_ROLE;  GRANT SELECT ON V\_GAMES TO VIEW\_ACCESS\_ROLE;  GRANT SELECT ON V\_SCREENSHOTS TO VIEW\_ACCESS\_ROLE;  GRANT SELECT ON V\_GMS\_GNRS TO VIEW\_ACCESS\_ROLE;  GRANT SELECT ON V\_GMS\_DVLS TO VIEW\_ACCESS\_ROLE;  GRANT SELECT ON V\_GMS\_PBLS TO VIEW\_ACCESS\_ROLE;  GRANT SELECT ON V\_WISHLISTS TO VIEW\_ACCESS\_ROLE;  GRANT SELECT ON V\_CARTS TO VIEW\_ACCESS\_ROLE;  GRANT SELECT ON V\_LIBRARIES TO VIEW\_ACCESS\_ROLE;  GRANT SELECT ON V\_RATINGS TO VIEW\_ACCESS\_ROLE;  ---------- PUBLISHER ROLE ----------  CREATE ROLE PUBLISHER\_ROLE;  GRANT  CONNECT  TO PUBLISHER\_ROLE;  GRANT EXECUTE ON USERS\_SELF\_UTILS TO PUBLISHER\_ROLE;  GRANT EXECUTE ON GAMES\_VIEW\_UTILS TO PUBLISHER\_ROLE;  GRANT EXECUTE ON GAMES\_MANAGEMENT\_UTILS TO PUBLISHER\_ROLE;  ---------- CLIENT ROLE ----------  CREATE ROLE CLIENT\_ROLE;  GRANT  CONNECT  TO CLIENT\_ROLE;  GRANT EXECUTE ON USERS\_SELF\_UTILS TO CLIENT\_ROLE;  GRANT EXECUTE ON GAMES\_VIEW\_UTILS TO CLIENT\_ROLE;  ---------- USERS ----------  ---------- PUBLISHER USER ----------  CREATE USER PUBLISHER  IDENTIFIED BY F5STOREDBPublisher  ;  GRANT VIEW\_ACCESS\_ROLE TO PUBLISHER;  GRANT PUBLISHER\_ROLE TO PUBLISHER;  ---------- CLIENT USER ----------  CREATE USER CLIENT  IDENTIFIED BY F5STOREDBClient  ;  GRANT CLIENT\_ROLE TO CLIENT;  GRANT VIEW\_ACCESS\_ROLE TO CLIENT; |

Листинг 3.3 – Создание пользователей разработчиком

В целом, была создана безопасная и эффективная база данных для интернет-магазина видеоигр, где различные пользователи имеют различные уровни доступа и привилегии в соответствии с их ролями и функциями.

## 3.3 Вывод

Проект базы данных для магазина видеоигр включает создание нескольких табличных пространств, предназначенных для хранения информации о пользователях, продуктах и т.д.

Каждое табличное пространство имеет свой собственный файл для хранения информации, и для разработчика была выделена квота на добавление данных в эти пространства.

В рамках создания базы данных также определены роли и пользователи. Основные пользователи Administrator и Developer имеют различные роли с различными привилегиями и выполняют собственные задачи.

В целом, создана безопасная и эффективная база данных, в которой различные пользователи имеют различные уровни доступа и привилегии в соответствии с их ролями и функциями.

4 Разработка необходимых объектов

## 4.1 Создание таблиц

В рамках проекта необходимо создать 14 таблиц, в которых будут храниться данные о пользователях, видеоиграх и прочих объектах.

Для магазина видеоигр были созданы следующие таблицы.

ROLES – таблица, содержащая роли пользователей. Здесь хранится информация о ролях, которые могут быть присвоены пользователям. Каждая роль имеет уникальный идентификатор (ROLE\_ID) и название (ROLE\_NAME).

USERS – это таблица, где хранятся данные о пользователях. Здесь содержится информация, такая как идентификатор пользователя (USER\_ID), его имя (USER\_NAME), электронная почта (USER\_EMAIL), хеш пароля (USER\_PASSWORD) и аватар (USER\_AVATAR). Также в таблице присутствует связь с таблицей ROLES через поле USER\_ROLE\_ID, определяющее роль пользователя.

GENRES – таблица, предназначенная для хранения жанров видеоигр. Здесь содержатся различные жанры, каждый из которых имеет название (GENRE\_NAME) и уникальный идентификатор.

Таблицы DEVELOPERS и PUBLISHERS содержат информацию о различных игровых студиях, разрабатывающих видеоигры.

GAMES – таблица, содержащая информацию о видеоиграх. Каждая игра имеет уникальный идентификатор (GAME\_ID), название (GAME\_TITLE), цену (GAME\_PRICE) и прочую информацию.

Связь между таблицами GAMES и GENRES, DEVELOPERS, PUBLISHERS реализована через дополнительные таблицы GMS\_GNRS, GMS\_DVLS, GMS\_PBLS, позволяющие организовать связь "многие-ко-многим".

Таблицы WISHLISTS, CARTS, LIBRARIES и RATINGS отображают пользовательские списки желаемого, корзины, библиотеки и оценки купленных видеоигр.

При создании таблицы указывается ее название, перечисляются поля, ограничения целостности, а также табличное пространство, в котором будут храниться данные таблицы. Пример создания таблицы будет на листинге 4.1.

|  |
| --- |
| ---------- USERS TABLE ----------  CREATE TABLE USERS (  USER\_ID INTEGER GENERATED ALWAYS AS IDENTITY(START WITH 1 INCREMENT BY 1),  USER\_NAME NVARCHAR2(255) NOT NULL,  USER\_EMAIL NVARCHAR2(255) NOT NULL UNIQUE,  USER\_PASSWORD NVARCHAR2(255) NOT NULL,  USER\_AVATAR BLOB,  USER\_ROLE\_ID INTEGER NOT NULL,    CONSTRAINT USRS\_PK PRIMARY KEY (USER\_ID),  CONSTRAINT USRS\_USER\_ROLE\_ID\_FK FOREIGN KEY (USER\_ROLE\_ID) REFERENCES ROLES(ROLE\_ID)  )  TABLESPACE TS\_USER  ; |

Листинг 4.1 – Создание таблицы USERS

Таблицы являются главной частью базы данных. Все последующие созданные объекты будут использовать данные базовых таблиц.

## 4.2 Создание триггеров

Триггер (trigger) в базе данных представляет собой особый вид процедур, которые срабатывают по запускающему их событию. В рамках данного проекта использовано четыре триггера для валидации некоторых значений при вставке и обновлении данных в различных таблицах: USERS\_VALIDATION\_TRIGGER, GAMES\_VALIDATION\_TRIGGER, LIBRARIES\_VALIDATION\_TRIGGER и RATINGS\_VALIDATION\_TRIGGER. Каждый триггер является before-триггером, срабатывающим при вставке или изменении значений и рассматривает каждую измененную строку по отдельности. Пример создания триггера GAMES\_VALIDATION\_TRIGGER, проверяющего корректность данных в таблице GAMES, представлен в листинге 4.2.

|  |
| --- |
| ---------- GAMES VALIDATION TRIGGER ----------  CREATE OR REPLACE TRIGGER GAMES\_VALIDATION\_TRIGGER  BEFORE INSERT OR UPDATE ON GAMES  FOR EACH ROW  BEGIN  IF :NEW.GAME\_PRICE < 0 THEN  RAISE\_APPLICATION\_ERROR(-20000, 'Game price cannot be negative');  END IF;  IF :NEW.GAME\_SCORE NOT BETWEEN 0 AND 100 THEN  RAISE\_APPLICATION\_ERROR(-20000, 'Game score could be between 0 and 100');  END IF;  IF :NEW.GAME\_AGE\_LIMIT NOT BETWEEN 0 AND 25 THEN  RAISE\_APPLICATION\_ERROR(-20000, 'Game age limit could be between 0 and 25');  END IF;  IF :NEW.GAME\_RELEASE\_DATE > SYSDATE THEN  RAISE\_APPLICATION\_ERROR(-20000, 'Game release date cannot be in the future');  END IF;  END; |

Листинг 4.2 – Создание триггера GAMES\_VALIDATION\_TRIGGER

Использование триггеров сделало возможным создание сложных ограничений целостности для таблиц. Это решение позитивно сказалось на качестве хранимых значений в базе данных.

## 4.3 Создание индексов

Индекс представляет собой объект базы данных, который используется для ускорения поиска данных. Последовательный поиск данных может занимать много времени при большом количестве записей в таблице. 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, так как они имеют оптимизированную структуру для поиска.

Для того, чтобы быстро находить данные по их названию или описанию, были созданы индексы:

Индексы ROLES\_ROLE\_NAME\_INDEX, USERS\_USER\_NAME\_INDEX, GENRES\_GENRE\_NAME\_INDEX, DEVELOPERS\_DEVELOPER\_NAME\_INDEX, PUBLISHERS\_PUBLISHER\_NAME\_INDEX используются оптимизации поиска по названию объектов в соответствующих таблицах.

Индексы для таблицы GAMES, такие как GAMES\_GAME\_TITLE\_INDEX, GAME\_GAME\_RELEASE\_DATE\_INDEX, GAME\_GAME\_DESCRIPTION\_INDEX, используются для оптимизации поиска видеоигр по критериям.

Пример создания индекса приведен в листинге 4.3.

|  |
| --- |
| ---------- GAMES TABLE INDEXES ----------  CREATE INDEX GAMES\_GAME\_TITLE\_INDEX ON GAMES(GAME\_TITLE); |

Листинг 4.3 – Индекс GAMES\_GAME\_TITLE\_INDEX

Использование индексов ускорило работу некоторых запросов к таблицам, что позитивно сказалось на опыте работы с базой данных.

## 4.4 Создание представлений

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. Представления дают возможность пользователям просматривать содержимое таблиц без непосредственного обращения к таблицам, а также позволяют скрыть некоторые конфиденциальные данные, либо, наоборот, выдать больше данных, чем позволяют исходные таблицы.

В данном проекте было решено создать представления для каждой из 14 таблиц. Данное решение позволит пользователям не ограничиваться получением данных только лишь из процедур или функций, но и даст им возможность использовать собственные select-запросы к представлениям. Названия всех представлений содержат префикс "V\_", а также название исходной таблицы. Ниже приведен список некоторых представлений:

* V\_ROLES содержит информацию о всех ролях.
* V\_USERS содержит информацию о всех пользователях, скрывая поле захешированного пароля.
* V\_GAMES содержит информацию о видеоиграх.
* V\_GENRES, V\_DEVELOPERS и V\_PUBLISHERS содержат дополнительные данные для видеоигр.
* V\_WISHLISTS, V\_CARTS, V\_LIBRARIES и V\_RATINGS содержат пользовательскую информацию связанную с играми.

Представление USER\_V было создано для того, чтобы получить полную информацию о пользователях, за исключением поля пароля. Оно представлено в листинге 4.4. Остальные представления аналогичны, но в работе используют другие таблицы.

|  |
| --- |
| ---------- USERS VIEW ----------  CREATE VIEW V\_USERS  AS  SELECT  USER\_ID ID,  USER\_NAME NAME,  USER\_EMAIL EMAIL,  USER\_ROLE\_ID ROLE\_ID,  USER\_AVATAR AVATAR  FROM  USERS  ; |

Листинг 4.4 – Представление USER\_V

Использование представлений позволило скрыть от пользователей внутреннюю структуру таблиц в базе данных, что повысило ее безопасность.

## 4.5 Создание процедур

Процедура в PL/SQL – это модуль подпрограммы, состоящий из группы операторов PL/SQL, которые можно вызывать по имени. Каждая процедура в PL/SQL имеет собственное уникальное имя, по которому к ней можно обращаться и вызывать. Этот модуль подпрограммы в Oracle хранится как объект базы данных. Пример процедуры представлен в листинге 4.5.

|  |
| --- |
| ---------- INSERT 100K GAMES PROCEDURE ----------  CREATE OR REPLACE PROCEDURE INSERT\_100K\_GAMES  IS  BEGIN  FOR i IN 0..100000 LOOP  GAMES\_MANAGEMENT\_UTILS.CREATE\_GAME(  '100K Test Game #' || i,  (1.67 \* i + 1.9),  MOD(10 \* i + i, 100),  21,  TO\_DATE('01.01.1978', 'DD.MM.YYYY') + numtodsinterval(i\*4, 'hour'),  '100K Test Game #' || i || ' Full Long Description'  );  END LOOP;  END INSERT\_100K\_GAMES; |

Листинг 4.5 – Процедура INSERT\_100K\_GAMES

В зависимости от того, какую задачу необходимо выполнить, следует использовать хранимую процедуру или функцию. Хранимые процедуры могут использоваться для выполнения сложных операций над данными, таких как массовые изменения в таблицах, а также для оптимизации производительности приложения. Функции же наиболее полезны в случаях, когда требуется выполнить вычисление на основе данных в базе данных, например, для подсчета статистики или фильтрации данных.

## 4.6 Создание функций

Функция в PL/SQL – это модуль подпрограммы, состоящий из группы операторов PL/SQL, которые можно вызывать по имени. В отличие от процедур, функции должны заканчиваться оператором RETURN, возвращая вычисленное значение к точке вызова функции. Каждая функция в PL/SQL имеет собственное уникальное имя, по которому к ней можно обращаться и вызывать. Этот модуль подпрограммы в Oracle хранится как объект базы данных. Пример функции вычисления хеша приведен в листинге 4.6.

|  |
| --- |
| ---------- GET HASH FUNCTION ----------  CREATE OR REPLACE FUNCTION GET\_HASH(  in\_text IN NVARCHAR2  )  RETURN NVARCHAR2  IS  BEGIN  RETURN SYS.DBMS\_CRYPTO.HASH(UTL\_RAW.CAST\_TO\_RAW(in\_text), DBMS\_CRYPTO.HASH\_SH1);  END GET\_HASH; |

Листинг 4.6 – Функция GET\_HASH

Данная функция используется других процедурах и функциях для вычисления хеша пароля и последующего его сохранения в базу данных.

## 4.7 Создание пакетов процедур и функций для базы данных

Для управления данными через приложение пользователи и администраторы будут использовать хранимые процедуры и функции. Хранимая процедура представляет собой набор SQL-инструкций, который компилируется один раз и хранится на сервере. Функция также представляет собой набор SQL-инструкций, но возвращает значение, которое может быть использовано внутри другой инструкции SQL. Для упорядочивания функций и процедур были созданы пакеты. Можно выделить следующие пакеты: USERS\_UTILS – пакет для управления данными пользователей, ROLES\_UTILS – пакет для управления данными ролей, GAMES\_UTILS – пакет для управления данными видеоигр. Каждый из пакетов группирует в себе процедуры и функции, выполняющие действия с определенной группой объектов.

Каждый пакет состоит из двух частей – спецификации и реализации (тела). Пример создания спецификации пакета приведен в листинге 4.7.

|  |
| --- |
| ---------- USERS SELF UTILS PACKAGE SPECIFICATION ----------  CREATE OR REPLACE PACKAGE USERS\_SELF\_UTILS IS  PROCEDURE REGISTER(  in\_name USERS.USER\_NAME%TYPE,  in\_email USERS.USER\_EMAIL%TYPE,  in\_password NVARCHAR2  );    PROCEDURE LOGIN(  in\_email USERS.USER\_EMAIL%TYPE,  in\_password NVARCHAR2  );    PROCEDURE DELETE\_ACCOUNT(  in\_email USERS.USER\_EMAIL%TYPE,  in\_password NVARCHAR2  );  END USERS\_SELF\_UTILS; |

Листинг 4.7 – Часть пакета USERS\_SELF\_UTILS

Аналогичные пакеты были созданы для работы с остальными таблицами базы данных. Использование пакетов позволило более гибко распределить процедуры и функции, логически группируя их.

## 4.8 Создание пользовательских типов

Для работы процедур и функций были созданы пользовательские типы, представляющие их сложные возвращаемые значения. Пользовательские типы позволяют пользователям создавать свои собственные типы данных, которые могут быть использованы для моделирования сложных данных, соответствующих требованиям их приложений. Эти типы данных могут быть скалярными или составными, а также могут включать методы и функции для работы с данными. Пример создания пользовательского типа представлен в листинге 4.8.

|  |
| --- |
| CREATE OR REPLACE TYPE GAME\_RECORD IS OBJECT (  ID NUMBER,  TITLE NVARCHAR2(255),  PRICE NUMBER,  SCORE NUMBER,  AGE\_LIMIT NUMBER,  RELEASE\_DATE DATE,  DESCRIPTION NVARCHAR2(2000),  POSTER BLOB,  COVER BLOB  ); |

Листинг 4.8 – Пользовательский тип GAME\_RECORD

Аналогичные типы данных были созданы для представления всех остальных таблиц базы данных, с которыми могут работать процедуры и функции.

## 4.9 Вывод

В рамках данного раздела базы данных были разработаны и созданы различные объекты, предназначенные для эффективного хранения, обработки и обеспечения безопасности данных интернет-магазина видеоигр. Среди ключевых элементов используются таблицы для хранения различных типов данных, таких как информация о продуктах, заказах, пользователях и других аспектах бизнеса.

Для обеспечения целостности данных были созданы триггеры, которые активируются при вставке или изменении значений в определенных таблицах. Эти триггеры выполняют валидацию данных и применяют необходимые правила, чтобы гарантировать корректность информации, сохраняемой в базе данных.

Для обеспечения безопасного доступа пользователей к данным были разработаны представления, которые предоставляют удобный интерфейс для работы с данными, фильтруя или скрывая конфиденциальную информацию в зависимости от прав доступа пользователя.

Для управления изменениями данных, вносимыми пользователями, были разработаны хранимые процедуры и функции. Они предоставляют удобные интерфейсы для выполнения операций над данными, таких как добавление новых продуктов, обновление информации о заказах и управление аккаунтами пользователей, не требуя напрямую взаимодействовать с исходными таблицами базы данных.

Для организации и группировки логически связанных объектов были созданы пакеты. Они позволяют упорядочить и сгруппировать хранимые процедуры, функции и другие элементы базы данных под единым названием, что облегчает их использование и поддержку.

В целом, эти объекты и механизмы созданы для обеспечения эффективного управления данными интернет-магазина видеоигр, обеспечивая их целостность, безопасность и удобство использования для пользователей.

5 Описание процедур экспорта и импорта

## 5.1 Экспорт данных в JSON

Для экспорта данных в формат JSON было разработано несколько процедур. К примеру, процедура EXPORT\_GAMES\_TO\_JSON преобразует каждую запись из таблицы GAMES в JSON-объект и сохраняет полученные объекты в виде JSON-массива в файл GAMES.JSON. Код данной процедуры представлен в листинге 5.1.

|  |
| --- |
| ---------- EXPORT GAMES TO JSON PROCEDURE ----------  CREATE OR REPLACE PROCEDURE EXPORT\_GAMES\_TO\_JSON  IS  JSON\_FILE UTL\_FILE.FILE\_TYPE;  JSON\_DATA NVARCHAR2(32000);  BEGIN  SELECT  '['  ||  LISTAGG(JSON\_OBJECT(  'ID' IS GAME\_ID,  'TITLE' IS GAME\_TITLE,  'PRICE' IS GAME\_PRICE,  'SCORE' IS GAME\_SCORE,  'AGE\_LIMIT' IS GAME\_AGE\_LIMIT,  'RELEASE\_DATE' IS GAME\_RELEASE\_DATE,  'DESCRIPTION' IS GAME\_DESCRIPTION,  'POSTER' IS GAME\_POSTER,  'COVER' IS GAME\_COVER  ), ',')  ||  ']'  INTO JSON\_DATA  FROM GAMES  ;  JSON\_FILE := UTL\_FILE.FOPEN('JSON\_EXPORT\_DIR', 'GAMES.JSON', 'W');  UTL\_FILE.PUT\_LINE(JSON\_FILE, JSON\_DATA);  UTL\_FILE.FFLUSH(JSON\_FILE);  UTL\_FILE.FCLOSE(JSON\_FILE);  EXCEPTION  WHEN OTHERS THEN BEGIN  UTL\_FILE.FCLOSE(JSON\_FILE);  RAISE;  END;  END EXPORT\_GAMES\_TO\_JSON; |

Листинг 5.1 – Процедура EXPORT\_GAMES\_TO\_JSON

Формат записи в JSON-файле представлен в листинге 5.2.

|  |
| --- |
| [{"ID":1, "TITLE": "Test game", "PRICE": 16, "SCORE": 67, "AGE\_LIMIT": 18, "RELEASE\_DATE": "2023-05-12T00:00:00", "DESCRIPTION": "Description of the game", "POSTER" :null, "COVER": null}] |

Листинг 5.2 –Запись в файле

Аналогичные процедуры экспорта были созданы и для остальных таблиц базы данных.

## 5.2 Импорт данных из JSON

Для импорта JSON-данных в базу данных были созданы процедуры, использующие встроенные в Oracle типы данных и функции для работы с форматом JSON. Среди таких типов и функций можно выделить JSON\_ARRAY\_T и JSON\_VALUE. Для импортирования полученных данных в таблицы используется конструкция MERGE INTO, позволяющая обновить существующие или добавить новые строки в таблицу. Пример процедуры IMPORT\_GAMES\_FROM\_JSON, позволяющей внести данные из файла GAMES.JSON в таблицу GAMES, представлен в листинге 5.3.

|  |
| --- |
| ---------- IMPORT GAMES FROM JSON PROCEDURE ----------  CREATE OR REPLACE PROCEDURE IMPORT\_GAMES\_FROM\_JSON  IS  JSON\_FILE UTL\_FILE.FILE\_TYPE;  JSON\_DATA NVARCHAR2(32000);  JSON\_OBJS JSON\_ARRAY\_T;  JSON\_OBJ NVARCHAR2(32000);  BEGIN  JSON\_FILE := UTL\_FILE.FOPEN('JSON\_EXPORT\_DIR', 'GAMES.JSON', 'R');  UTL\_FILE.GET\_LINE(JSON\_FILE, JSON\_DATA);  JSON\_OBJS := JSON\_ARRAY\_T(JSON\_DATA);    FOR i IN 0..JSON\_OBJS.GET\_SIZE-1 LOOP  JSON\_OBJ := JSON\_OBJS.GET(i).TO\_STRING();  MERGE INTO GAMES g  USING (  SELECT  JSON\_VALUE(JSON\_OBJ, '$.ID') AS GAME\_ID,  JSON\_VALUE(JSON\_OBJ, '$.TITLE') AS GAME\_TITLE,  JSON\_VALUE(JSON\_OBJ, '$.PRICE') AS GAME\_PRICE,  JSON\_VALUE(JSON\_OBJ, '$.SCORE') AS GAME\_SCORE,  JSON\_VALUE(JSON\_OBJ, '$.AGE\_LIMIT') AS GAME\_AGE\_LIMIT,  JSON\_VALUE(JSON\_OBJ, '$.RELEASE\_DATE') AS GAME\_RELEASE\_DATE,  JSON\_VALUE(JSON\_OBJ, '$.DESCRIPTION') AS GAME\_DESCRIPTION,  JSON\_VALUE(JSON\_OBJ, '$.POSTER') AS GAME\_POSTER,  JSON\_VALUE(JSON\_OBJ, '$.COVER') AS GAME\_COVER  FROM DUAL  ) parsed  ON (g.GAME\_ID = parsed.GAME\_ID)  WHEN NOT MATCHED THEN  INSERT (GAME\_ID, GAME\_TITLE, GAME\_PRICE, GAME\_SCORE, GAME\_AGE\_LIMIT, GAME\_RELEASE\_DATE, GAME\_DESCRIPTION, GAME\_POSTER, GAME\_COVER)  VALUES (  parsed.GAME\_ID,  parsed.GAME\_TITLE,  parsed.GAME\_PRICE,  parsed.GAME\_SCORE,  parsed.GAME\_AGE\_LIMIT,  TO\_DATE(parsed.GAME\_RELEASE\_DATE, 'YYYY-MM-DD"T"HH24:MI:SS'),  parsed.GAME\_DESCRIPTION,  parsed.GAME\_POSTER,  parsed.GAME\_COVER  )  ;  END LOOP;    UTL\_FILE.FCLOSE(JSON\_FILE);  EXCEPTION  WHEN OTHERS THEN BEGIN  UTL\_FILE.FCLOSE(JSON\_FILE);  RAISE;  END;  END IMPORT\_GAMES\_FROM\_JSON; |

Листинг 5.3 – Процедура IMPORT\_GAMES\_FROM\_JSON

Аналогичные процедуры были созданы для импорта всех таблиц базы данных из соответствующих JSON-файлов.

## 5.3 Вывод

В рамках данного раздела были созданы процедуры, позволяющие экспортировать и импортировать данные из таблиц в формат JSON. Подобная работа с данными может использоваться для резервного сохранения данных, их модификации вне базы данных и многих других сценариях. К тому же, выбор популярного, простого и широко используемого формата JSON позволяет использовать экспортированные из базы данные во многих современных приложениях.

6 Использование технологии Stand By-серверов СУБД

Технология Stand By-серверов представляет собой средство обеспечения высокой доступности и защиты данных в базе данных. Эта технология использует концепцию создания и поддержания резервных копий главной базы данных. Резервные базы данных могут быть расположены в другом географическом регионе или в том же дата-центре, что и основная база данных. Основные цели использования данной технологии включают повышение отказоустойчивости, обеспечение защиты данных и минимизацию времени простоя в случае отказа основной базы данных. Данная технология в СУБД Oracle получила название Data Guard.

## 6.1 Настройка Stand By

Настройка Stand By-сервера требует некоторых особых настроек основной базы. Примером таких настроек являются archivelog и force logging. Эти настройки необходимо включить с помощью запросов alter database archivelog и alter database force logging.

Так же на основной и резервной базах данных должны быть созданы специальные файлы журналов – standby redo logs. Они создаются с помощью запросов alter database.

Для основного и резервного серверов были подготовлены файлы pfilePROD.ora и pfileSTAN.ora – файлы параметров для серверов. В них указываются параметры, используемые при работе данных серверов в режимах standby и primary. Параметры, используемые для настройки серверов, приведены в листинге 6.1.

|  |
| --- |
| db\_name='orcl'  db\_unique\_name='orclprod'  log\_archive\_config='dg\_config=(orclprod,orclstan)'  log\_archive\_dest\_1='SERVICE=orclstan LGWR ASYNC VALID\_FOR=(ONLINE\_LOGFILES,PRIMARY\_ROLE) db\_unique\_name=orclstan'  log\_archive\_dest\_2='LOCATION=C:\OracleDB\oradata\ORCL\archive VALID\_FOR=(ALL\_LOGFILES,ALL\_ROLES) db\_unique\_name=orclprod'  log\_archive\_dest\_state\_1=ENABLE  log\_archive\_dest\_state\_2=ENABLE  fal\_client='orclprod'  fal\_server='orclstan'  standby\_file\_management='AUTO' |

Листинг 6.1 – Параметры серверов баз данных

Для правильного сетевого подключения между серверами были подготовлены файлы hostsRROD, hostsSTAN, listenerSTAN.ora и tnsnamesBOTH.ora, представляющие настройки служб для корректного и удобного обнаружения серверов в сети. Пример конфигурации listener на резервном сервере представлен в листинге 6.2.

|  |
| --- |
| SID\_LIST\_LISTENER =  (SID\_LIST =  (SID\_DESC =  (SID\_NAME = CLRExtProc)  (ORACLE\_HOME = C:\WINDOWS.X64\_193000\_db\_home)  (PROGRAM = extproc)  (ENVS = "EXTPROC\_DLLS=ONLY:C:\WINDOWS.X64\_193000\_db\_home\bin\oraclr19.dll")  )  (SID\_DESC =  (GLOBAL\_DBNAME = orclstan)  (ORACLE\_HOME = C:\WINDOWS.X64\_193000\_db\_home)  (SID\_NAME = orcl)  )  )  LISTENER =  (DESCRIPTION\_LIST =  (DESCRIPTION =  (ADDRESS = (PROTOCOL = TCP)(HOST = oracledb.mshome.net)(PORT = 1521))  (ADDRESS = (PROTOCOL = IPC)(KEY = EXTPROC1521))  )  ) |

Листинг 6.2 – Конфигурация listener резервной базы данных

Далее необходимо создать резервные копии основной базы данных. Для этого используется утилита командной строки rman. С ее помощью создается резервная копия основной базы данных, после чего созданные файлы необходимо передать на резервный сервер.

С помощью утилиты rman и удаленного подключения основного сервера к резервному необходимо восстановить резервную базу данных из копии основной.

В результате выполнения данных действий основная база данных начинает работать в режиме primary, а резервная – в режиме standby.

## 6.2 Проверка работы Stand By

Для проверки работы серверов СУБД можно выполнить select-запрос к представлению v$database на обоих серверах. Результат запроса для основного сервера представлен на рисунке 6.3.

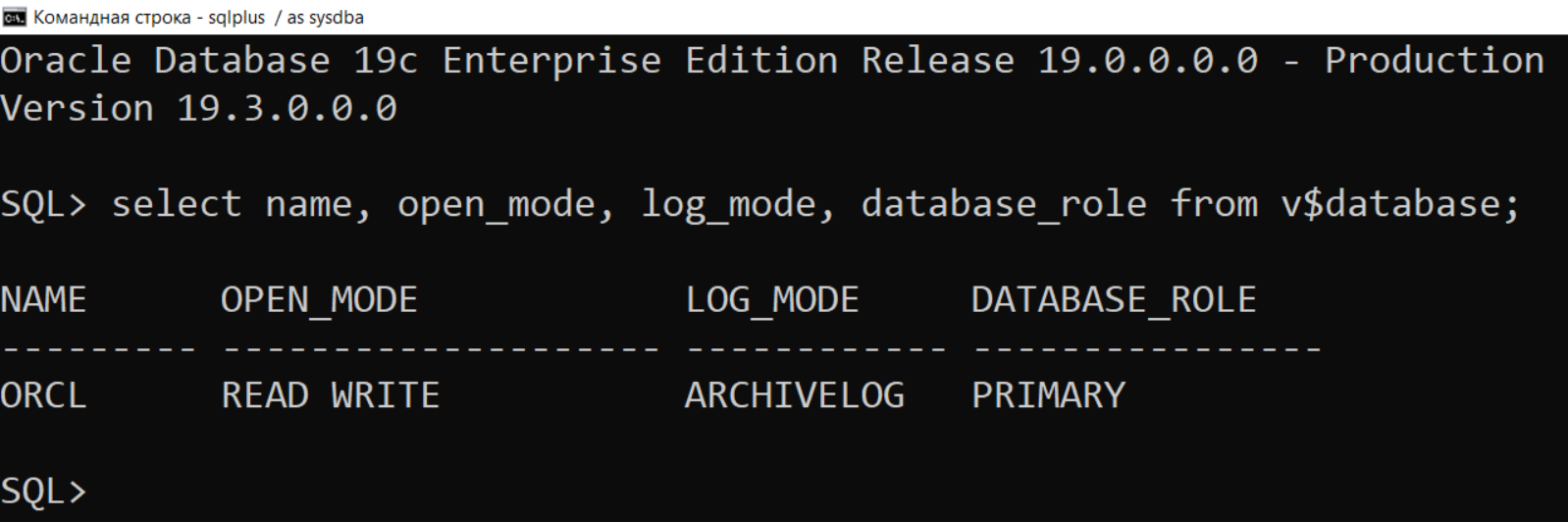


Рисунок 6.3 – Выполнение запроса на основном сервере

Результат выполнения аналогичного select-запроса на резервном сервере представлен на рисунке 6.4.

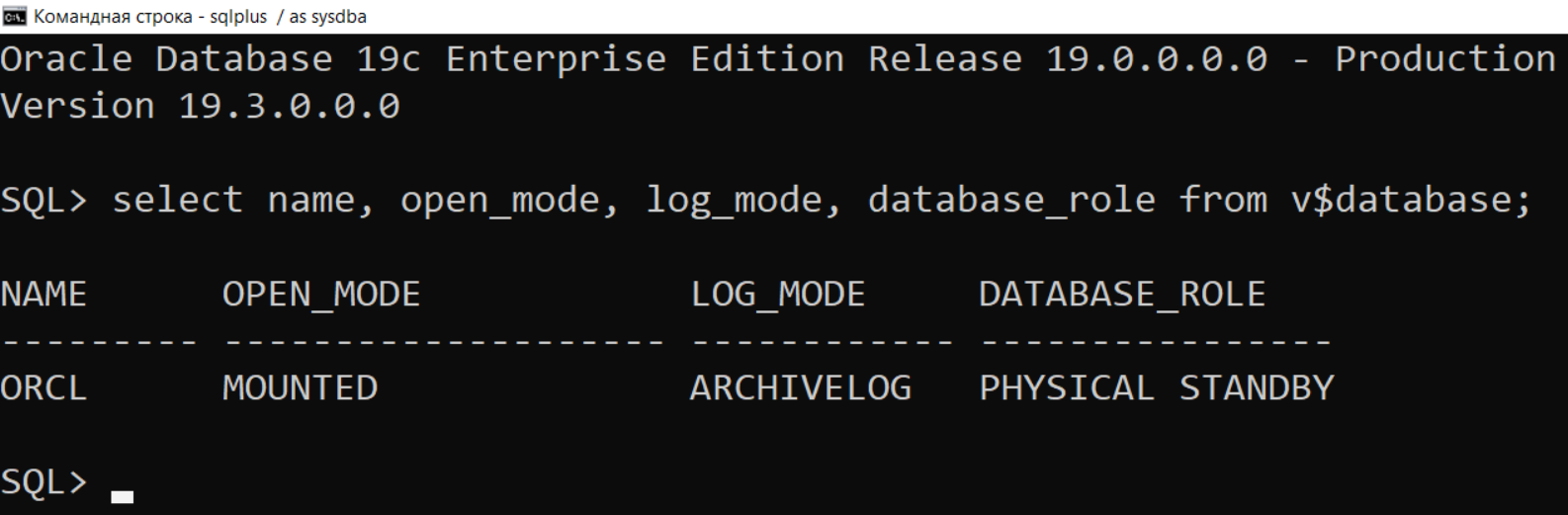


Рисунок 6.4 – Выполнение запроса на резервном сервере

Как можно заметить, основной сервер имеет роль primary, а резервный сервер – роль standby. Из этого можно сделать вывод, что настройка серверов прошла успешно.

## 6.3 Вывод

В рамках данного раздела был настроен резервный сервер для базы данных с применением технологии Stand By-серверов СУБД. В результате итоговая база данных состоит из двух серверов – основного, с которым работают пользователи, и резервного, который пользователям недоступен. Резервный сервер находится в состоянии ожидания, поддерживая у себя актуальные данные из журналов повтора основного сервера. Такое решение позволит быстро открыть резервный сервер для пользователей в случае отказа основного сервера, тем самым минимизировав время просто базы данных и обеспечив стабильную работу приложений в случае критической ситуации.

7 Тестирование производительности базы данных

## 7.1 Тестирование производительности

Для тестирования производительности базы данных было решено заполнить таблицу GAMES. Процедура INSERT\_100K\_GAMES, представленная в листинге 4.5, при выполнении добавит в таблицу GAMES 100000 строк. Время выполнения данной процедуры представлено на рисунке 7.1.



Рисунок 7.1 – Время выполнения процедуры

Результат выборки всех данных из таблицы представлен на рисунке 7.2.

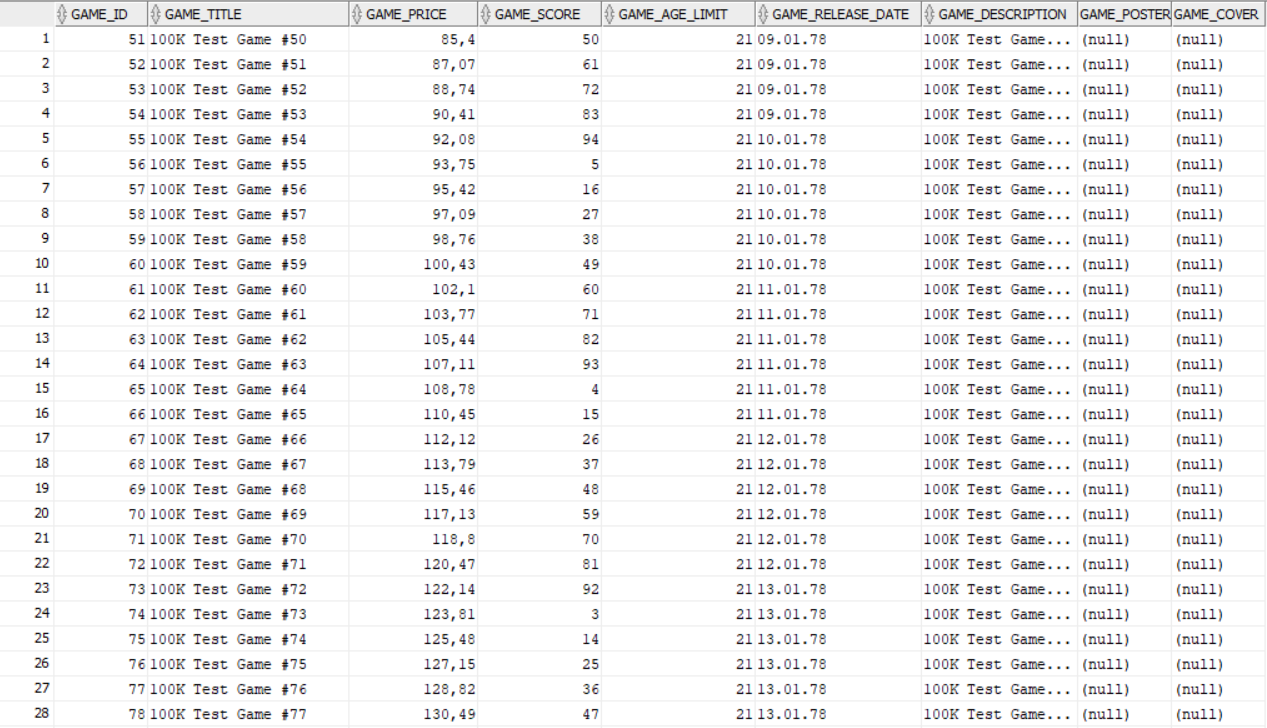


Рисунок 7.2 – Выборка всех данных из таблицы

Время выполнения запроса к таблице без использования дополнительных индексов представлено на рисунке 7.3.



Рисунок 7.3 – Время выполнения процедуры

Время выполнение аналогичного запроса после создания индексов для столбцов данной таблицы представлено на рисунке 7.4.



Рисунок 7.4 – Время выполнения процедуры

Как можно заметить, после создания индексов выполнение запросов заметно ускорилось, что говорит о важности индексов при работе с базой данных.

## 7.2 Вывод

В рамках данного раздела была протестирована производительность базы данных. Для этого в одну из таблиц было вставлено 100,000 записей и проведены замеры времени выполнения одинаковых запросов с использованием дополнительных индексов и без них. По результатам тестирования выяснилось следующее: база данных стабильно работает при большом количестве записей в ней и использование индексов полезно при работе с большими объемами данных.

Добавление индексов значительно сократило время выполнения запросов, что подтверждает их важность для оптимизации работы базы данных. Эти результаты демонстрируют необходимость и эффективность использования индексов для обеспечения высокой производительности системы при увеличении объема данных.

Таким образом, исследование показало, что индексы являются неотъемлемым инструментом для поддержания и улучшения производительности базы данных в условиях роста объема информации. Эти данные подчеркивают важность правильного проектирования структуры базы данных с учетом возможного увеличения объема данных, а также необходимость регулярного анализа и оптимизации производительности с использованием индексов.

Кроме того, результаты тестирования подчеркивают важность стратегического подхода к управлению базой данных. Это включает в себя не только создание и использование индексов, но и регулярное мониторинг производительности, своевременную оптимизацию запросов и обновление структуры базы данных в соответствии с растущими требованиями и объемами данных. Таким образом, правильное управление базой данных и использование индексов играют ключевую роль в обеспечении ее эффективной и стабильной работы в долгосрочной перспективе.

# Заключение

В результате работы была успешно реализована база данных для интернет-магазина видеоигр, включающая управление информацией о пользователях, видеоиграх, жанрах, корзинах, оценках товаров и другими аспектами. Эта база данных охватывает все ключевые элементы, необходимые для полноценного функционирования интернет-магазина видеоигр, обеспечивая надежное хранение и оперативное управление данными.

Были использованы различные возможности базы данных Oracle, начиная от простых операций вставки и удаления до сложных триггеров и процедур. Применение широкого спектра функций Oracle позволило создать эффективную и гибкую систему управления данными. Были разработаны триггеры для автоматизации определённых задач и процедуры для выполнения сложных операций с данными.

Важным аспектом проекта стало обеспечение безопасности данных с помощью шифрования паролей и применения технологии Stand By-серверов СУБД. Для защиты данных пользователей была внедрена система шифрования паролей, что гарантирует сохранность конфиденциальной информации. Технология Stand By-серверов обеспечила высокий уровень отказоустойчивости и непрерывность работы базы данных.

Была разработана иерархия процедур управления данными, позволяющая каждому пользователю получать доступ только к необходимым данным. Разработка иерархии прав доступа позволила точно распределить роли и обязанности между пользователями системы, обеспечивая их доступ только к тем данным, которые необходимы для выполнения их задач.

Проведено тестирование производительности базы данных на большом объеме данных, что позволило оптимизировать ее структуру и проанализировать планы запросов к таблицам. Тестирование на высоких нагрузках помогло выявить и устранить узкие места в структуре базы данных, что значительно повысило её производительность и эффективность обработки запросов.

Использование технологий базы данных в этом проекте значительно упростило и ускорило процессы управления данными магазина. Благодаря современным технологиям, интегрированным в проект, удалось существенно сократить время обработки данных и улучшить общую производительность системы, что в конечном итоге повысило удовлетворённость пользователей и эффективность работы интернет-магазина.

# Список используемых источников

1. Работа с файлами в Oracle [Электронный ресурс]. – Режим доступа: https://docs.oracle.com/cd/F49540\_01/DOC/server.815/a68001/utl\_file.htm. – Дата доступа: 01.04.2024.
2. Работа с JSON в Oracle [Электронный ресурс]. – Режим доступа: <https://docs.oracle.com/en/database/oracle/oracle-database/21/adjsn/query-json-data.html#GUID-119E5069-77F2-45DC-B6F0-A1B312945590>. – Дата доступа: 10.04.2024.
3. Функция хеширования в Oracle [Электронный ресурс]. – Режим доступа:

<https://docs.oracle.com/en/database/oracle/oracle/database/23/arpls/DBMS_CRYPTO.html> – Дата доступа: 23.04.2024.

1. Технология Oracle Data Guard [Электронный ресурс]. – Режим доступа: <https://docs.oracle.com/en/database/oracle/oracle-database/19/sbydb/getting-started-with-oracle-data-guard.html>. – Дата доступа: 01.05.2024.
2. Настройка Oracle Stand By [Электронный ресурс]. – Режим доступа:

<https://docs.oracle.com/en/solutions/configure-standby-db/index.html>. – Дата доступа: 01.05.2024.

# Приложение А Листинг создания таблиц

|  |
| --- |
| ---------- EXECUTE AS DEVELOPER ----------  ---------- USERS DATA ----------  ---------- ROLES TABLE ----------  CREATE TABLE ROLES (  ROLE\_ID INTEGER GENERATED BY DEFAULT ON NULL AS IDENTITY(START WITH 1 INCREMENT BY 1),  ROLE\_NAME NVARCHAR2(255) NOT NULL,    CONSTRAINT RLS\_PK PRIMARY KEY (ROLE\_ID)  )  TABLESPACE TS\_USER  ;  ---------- USERS TABLE ----------  CREATE TABLE USERS (  USER\_ID INTEGER GENERATED BY DEFAULT ON NULL AS IDENTITY(START WITH 1 INCREMENT BY 1),  USER\_NAME NVARCHAR2(255) NOT NULL,  USER\_EMAIL NVARCHAR2(255) NOT NULL UNIQUE,  USER\_PASSWORD NVARCHAR2(255) NOT NULL,  USER\_AVATAR BLOB,  USER\_ROLE\_ID INTEGER NOT NULL,    CONSTRAINT USRS\_PK PRIMARY KEY (USER\_ID),  CONSTRAINT USRS\_USER\_ROLE\_ID\_FK FOREIGN KEY (USER\_ROLE\_ID) REFERENCES ROLES(ROLE\_ID)  )  TABLESPACE TS\_USER  ;  ---------- GAMES DATA ----------  ---------- GENRES TABLE ----------  CREATE TABLE GENRES (  GENRE\_ID INTEGER GENERATED BY DEFAULT ON NULL AS IDENTITY(START WITH 1 INCREMENT BY 1),  GENRE\_NAME NVARCHAR2(255) NOT NULL,    CONSTRAINT GNRS\_PK PRIMARY KEY (GENRE\_ID)  )  TABLESPACE TS\_GAME  ;  ---------- DEVELOPERS TABLE ----------  CREATE TABLE DEVELOPERS (  DEVELOPER\_ID INTEGER GENERATED BY DEFAULT ON NULL AS IDENTITY(START WITH 1 INCREMENT BY 1),  DEVELOPER\_NAME NVARCHAR2(255) NOT NULL,    CONSTRAINT DVLS\_PK PRIMARY KEY (DEVELOPER\_ID)  )  TABLESPACE TS\_GAME  ;  ---------- PUBLISHERS TABLE ----------  CREATE TABLE PUBLISHERS (  PUBLISHER\_ID INTEGER GENERATED BY DEFAULT ON NULL AS IDENTITY(START WITH 1 INCREMENT BY 1),  PUBLISHER\_NAME NVARCHAR2(255) NOT NULL,    CONSTRAINT PBLS\_PK PRIMARY KEY (PUBLISHER\_ID)  )  TABLESPACE TS\_GAME  ;  ---------- GAMES TABLE ----------  CREATE TABLE GAMES (  GAME\_ID INTEGER GENERATED BY DEFAULT ON NULL AS IDENTITY(START WITH 1 INCREMENT BY 1),  GAME\_TITLE NVARCHAR2(255) NOT NULL,  GAME\_PRICE NUMBER(15, 2) NOT NULL,  GAME\_SCORE NUMBER(3, 0) NOT NULL,  GAME\_AGE\_LIMIT NUMBER(2, 0) NOT NULL,  GAME\_RELEASE\_DATE DATE NOT NULL,  GAME\_DESCRIPTION NVARCHAR2(2000) NOT NULL,  GAME\_POSTER BLOB,  GAME\_COVER BLOB,    CONSTRAINT GMS\_PK PRIMARY KEY (GAME\_ID)  )  TABLESPACE TS\_GAME  ;  ---------- SCREENSHOTS TABLE ----------  CREATE TABLE SCREENSHOTS (  ID INTEGER GENERATED BY DEFAULT ON NULL AS IDENTITY(START WITH 1 INCREMENT BY 1),  GAME\_ID INTEGER NOT NULL,  SCREENSHOT BLOB NOT NULL,    CONSTRAINT SNSH\_PK PRIMARY KEY(ID),  CONSTRAINT SNSH\_GAME\_ID\_FK FOREIGN KEY (GAME\_ID) REFERENCES GAMES(GAME\_ID)  )  TABLESPACE TS\_GAME  ;  ---------- GMS\_GNRS TABLE ----------  CREATE TABLE GMS\_GNRS (  GAME\_ID INTEGER NOT NULL,  GENRE\_ID INTEGER NOT NULL,    CONSTRAINT GMS\_GNRS\_PK PRIMARY KEY (GAME\_ID, GENRE\_ID),  CONSTRAINT GMS\_GNRS\_GAME\_ID\_FK FOREIGN KEY (GAME\_ID) REFERENCES GAMES(GAME\_ID),  CONSTRAINT GMS\_GNRS\_GENRE\_ID\_FK FOREIGN KEY (GENRE\_ID) REFERENCES GENRES(GENRE\_ID)  )  TABLESPACE TS\_GAME  ;  ---------- GMS\_DVLS TABLE ----------  CREATE TABLE GMS\_DVLS (  GAME\_ID INTEGER NOT NULL,  DEVELOPER\_ID INTEGER NOT NULL,    CONSTRAINT GMS\_DVLS\_PK PRIMARY KEY (GAME\_ID, DEVELOPER\_ID),  CONSTRAINT GMS\_DVLS\_GAME\_ID\_FK FOREIGN KEY (GAME\_ID) REFERENCES GAMES(GAME\_ID),  CONSTRAINT GMS\_DVLS\_DEVELOPER\_ID\_FK FOREIGN KEY (DEVELOPER\_ID) REFERENCES DEVELOPERS(DEVELOPER\_ID)  )  TABLESPACE TS\_GAME  ;  ---------- GMS\_PBLS TABLE ----------  CREATE TABLE GMS\_PBLS (  GAME\_ID INTEGER NOT NULL,  PUBLISHER\_ID INTEGER NOT NULL,    CONSTRAINT GMS\_PBLS\_PK PRIMARY KEY (GAME\_ID, PUBLISHER\_ID),  CONSTRAINT GMS\_PBLS\_GAME\_ID\_FK FOREIGN KEY (GAME\_ID) REFERENCES GAMES(GAME\_ID),  CONSTRAINT GMS\_PBLS\_PUBLISHER\_ID\_FK FOREIGN KEY (PUBLISHER\_ID) REFERENCES PUBLISHERS(PUBLISHER\_ID)  )  TABLESPACE TS\_GAME  ;  ---------- LISTS DATA ----------  ---------- WISHLISTS TABLE ----------  CREATE TABLE WISHLISTS (  USER\_ID INTEGER NOT NULL,  GAME\_ID INTEGER NOT NULL,    CONSTRAINT WLST\_PK PRIMARY KEY(USER\_ID, GAME\_ID),  CONSTRAINT WLST\_USER\_ID\_FK FOREIGN KEY (USER\_ID) REFERENCES USERS(USER\_ID),  CONSTRAINT WLST\_GAME\_ID\_FK FOREIGN KEY (GAME\_ID) REFERENCES GAMES(GAME\_ID)  )  TABLESPACE TS\_LIST  ;  ---------- CARTS TABLE ----------  CREATE TABLE CARTS (  USER\_ID INTEGER NOT NULL,  GAME\_ID INTEGER NOT NULL,    CONSTRAINT CRTS\_PK PRIMARY KEY(USER\_ID, GAME\_ID),  CONSTRAINT CRTS\_USER\_ID\_FK FOREIGN KEY (USER\_ID) REFERENCES USERS(USER\_ID),  CONSTRAINT CRTS\_GAME\_ID\_FK FOREIGN KEY (GAME\_ID) REFERENCES GAMES(GAME\_ID)  )  TABLESPACE TS\_LIST  ;  ---------- LIBRARIES TABLE ----------  CREATE TABLE LIBRARIES (  USER\_ID INTEGER NOT NULL,  GAME\_ID INTEGER NOT NULL,  FAVORITE CHAR(1) NOT NULL,    CONSTRAINT LBRS\_PK PRIMARY KEY(USER\_ID, GAME\_ID),  CONSTRAINT LBRS\_USER\_ID\_FK FOREIGN KEY (USER\_ID) REFERENCES USERS(USER\_ID),  CONSTRAINT LBRS\_GAME\_ID\_FK FOREIGN KEY (GAME\_ID) REFERENCES GAMES(GAME\_ID)  )  TABLESPACE TS\_LIST  ;  ---------- RATINGS TABLE ----------  CREATE TABLE RATINGS (  USER\_ID INTEGER NOT NULL,  GAME\_ID INTEGER NOT NULL,  RATING NUMBER(2, 0) NOT NULL,    CONSTRAINT RTGS\_PK PRIMARY KEY(USER\_ID, GAME\_ID),  CONSTRAINT RTGS\_USER\_ID\_FK FOREIGN KEY (USER\_ID) REFERENCES USERS(USER\_ID),  CONSTRAINT RTGS\_GAME\_ID\_FK FOREIGN KEY (GAME\_ID) REFERENCES GAMES(GAME\_ID)  )  TABLESPACE TS\_LIST  ; |

# Приложение Б Листинг создания триггеров

|  |
| --- |
| ---------- EXECUTE AS DEVELOPER ----------  ---------- USERS VALIDATION TRIGGER ----------  CREATE OR REPLACE TRIGGER USERS\_VALIDATION\_TRIGGER  AFTER INSERT OR UPDATE ON USERS  FOR EACH ROW  BEGIN  IF NOT REGEXP\_LIKE(:NEW.USER\_EMAIL, '^[a-zA-Z0-9!#$%''\\*\+-/=\?^\_`\{|\}~]+@[a-zA-Z0-9.\_%-]+\.[a-zA-Z]{2,4}$') THEN  RAISE\_APPLICATION\_ERROR(-20000, 'Bad user email format');  END IF;  END;  /  ---------- GAMES VALIDATION TRIGGER ----------  CREATE OR REPLACE TRIGGER GAMES\_VALIDATION\_TRIGGER  AFTER INSERT OR UPDATE ON GAMES  FOR EACH ROW  BEGIN  IF :NEW.GAME\_PRICE < 0 THEN  RAISE\_APPLICATION\_ERROR(-20000, 'Game price cannot be negative');  END IF;  IF :NEW.GAME\_SCORE NOT BETWEEN 0 AND 100 THEN  RAISE\_APPLICATION\_ERROR(-20000, 'Game score could be between 0 and 100');  END IF;  IF :NEW.GAME\_AGE\_LIMIT NOT BETWEEN 0 AND 25 THEN  RAISE\_APPLICATION\_ERROR(-20000, 'Game age limit could be between 0 and 25');  END IF;  IF :NEW.GAME\_RELEASE\_DATE > SYSDATE THEN  RAISE\_APPLICATION\_ERROR(-20000, 'Game release date cannot be in the future');  END IF;  END;  /  ---------- LIBRARIES VALIDATION TRIGGER ----------  CREATE OR REPLACE TRIGGER LIBRARIES\_VALIDATION\_TRIGGER  AFTER INSERT OR UPDATE ON LIBRARIES  FOR EACH ROW  BEGIN  IF :NEW.FAVORITE NOT IN ('Y', 'N') THEN  RAISE\_APPLICATION\_ERROR(-20000, 'Library favorite could be Y or N');  END IF;  END;  /  ---------- RATINGS VALIDATION TRIGGER ----------  CREATE OR REPLACE TRIGGER RATINGS\_VALIDATION\_TRIGGER  AFTER INSERT OR UPDATE ON RATINGS  FOR EACH ROW  BEGIN  IF :NEW.RATING NOT BETWEEN 0 AND 10 THEN  RAISE\_APPLICATION\_ERROR(-20000, 'User game rating could be between 0 and 10');  END IF;  END;  / |

# Приложение В Листинг создания пакетов

|  |
| --- |
| ---------- EXECUTE AS DEVELOPER ----------  ---------- GAMES MANAGEMENT UTILS PACKAGE SPECIFICATION ----------  CREATE OR REPLACE PACKAGE GAMES\_MANAGEMENT\_UTILS IS  PROCEDURE CREATE\_GENRE(  in\_genre\_name GENRES.GENRE\_NAME%TYPE  );    PROCEDURE UPDATE\_GENRE(  in\_genre\_id GENRES.GENRE\_ID%TYPE,  in\_new\_genre\_name GENRES.GENRE\_NAME%TYPE  );    PROCEDURE DELETE\_GENRE(  in\_genre\_id GENRES.GENRE\_ID%TYPE  );    PROCEDURE CREATE\_DEVELOPER(  in\_developer\_name DEVELOPERS.DEVELOPER\_NAME%TYPE  );    PROCEDURE UPDATE\_DEVELOPER(  in\_developer\_id DEVELOPERS.DEVELOPER\_ID%TYPE,  in\_new\_developer\_name DEVELOPERS.DEVELOPER\_NAME%TYPE  );    PROCEDURE DELETE\_DEVELOPER(  in\_developer\_id DEVELOPERS.DEVELOPER\_ID%TYPE  );    PROCEDURE CREATE\_PUBLISHER(  in\_publisher\_name PUBLISHERS.PUBLISHER\_NAME%TYPE  );    PROCEDURE UPDATE\_PUBLISHER(  in\_publisher\_id PUBLISHERS.PUBLISHER\_ID%TYPE,  in\_new\_publisher\_name PUBLISHERS.PUBLISHER\_NAME%TYPE  );    PROCEDURE DELETE\_PUBLISHER(  in\_publisher\_id PUBLISHERS.PUBLISHER\_ID%TYPE  );    PROCEDURE CREATE\_GAME(  in\_title GAMES.GAME\_TITLE%TYPE,  in\_price GAMES.GAME\_PRICE%TYPE,  in\_score GAMES.GAME\_SCORE%TYPE,  in\_age\_limit GAMES.GAME\_AGE\_LIMIT%TYPE,  in\_release\_date GAMES.GAME\_RELEASE\_DATE%TYPE,  in\_description GAMES.GAME\_DESCRIPTION%TYPE  );    PROCEDURE UPDATE\_GAME(  in\_game\_id GAMES.GAME\_ID%TYPE,  in\_new\_title GAMES.GAME\_TITLE%TYPE,  in\_new\_price GAMES.GAME\_PRICE%TYPE,  in\_new\_score GAMES.GAME\_SCORE%TYPE,  in\_new\_age\_limit GAMES.GAME\_AGE\_LIMIT%TYPE,  in\_new\_release\_date GAMES.GAME\_RELEASE\_DATE%TYPE,  in\_new\_description GAMES.GAME\_DESCRIPTION%TYPE  );    PROCEDURE DELETE\_GAME(  in\_game\_id GAMES.GAME\_ID%TYPE  );    PROCEDURE ADD\_GENRE(  in\_game\_id GAMES.GAME\_ID%TYPE,  in\_genre\_id GENRES.GENRE\_ID%TYPE  );    PROCEDURE REMOVE\_GENRE(  in\_game\_id GAMES.GAME\_ID%TYPE,  in\_genre\_id GENRES.GENRE\_ID%TYPE  );    PROCEDURE ADD\_DEVELOPER(  in\_game\_id GAMES.GAME\_ID%TYPE,  in\_developer\_id DEVELOPERS.DEVELOPER\_ID%TYPE  );    PROCEDURE REMOVE\_DEVELOPER(  in\_game\_id GAMES.GAME\_ID%TYPE,  in\_developer\_id DEVELOPERS.DEVELOPER\_ID%TYPE  );    PROCEDURE ADD\_PUBLISHER(  in\_game\_id GAMES.GAME\_ID%TYPE,  in\_publisher\_id PUBLISHERS.PUBLISHER\_ID%TYPE  );    PROCEDURE REMOVE\_PUBLISHER(  in\_game\_id GAMES.GAME\_ID%TYPE,  in\_publisher\_id PUBLISHERS.PUBLISHER\_ID%TYPE  );  END GAMES\_MANAGEMENT\_UTILS;  /  ---------- GAMES MANAGEMENT UTILS PACKAGE BODY ----------  CREATE OR REPLACE PACKAGE BODY GAMES\_MANAGEMENT\_UTILS IS  ---------- CREATE GENRE PROCEDURE ----------  PROCEDURE CREATE\_GENRE(  in\_genre\_name GENRES.GENRE\_NAME%TYPE  )  IS  BEGIN  INSERT INTO GENRES(GENRE\_NAME)  VALUES (in\_genre\_name)  ;  END CREATE\_GENRE;  ---------- UPDATE GENRE PROCEDURE ----------  PROCEDURE UPDATE\_GENRE(  in\_genre\_id GENRES.GENRE\_ID%TYPE,  in\_new\_genre\_name GENRES.GENRE\_NAME%TYPE  )  IS  BEGIN  UPDATE GENRES  SET GENRE\_NAME = in\_new\_genre\_name  WHERE GENRE\_ID = in\_genre\_id  ;  END UPDATE\_GENRE;  ---------- DELETE GENRE PROCEDURE ----------  PROCEDURE DELETE\_GENRE(  in\_genre\_id GENRES.GENRE\_ID%TYPE  )  IS  BEGIN  DELETE FROM GENRES  WHERE GENRE\_ID = in\_genre\_id  ;  END DELETE\_GENRE;  ---------- CREATE DEVELOPER PROCEDURE ----------  PROCEDURE CREATE\_DEVELOPER(  in\_developer\_name DEVELOPERS.DEVELOPER\_NAME%TYPE  )  IS  BEGIN  INSERT INTO DEVELOPERS(DEVELOPER\_NAME)  VALUES (in\_developer\_name)  ;  END CREATE\_DEVELOPER;  ---------- UPDATE DEVELOPER PROCEDURE ----------  PROCEDURE UPDATE\_DEVELOPER(  in\_developer\_id DEVELOPERS.DEVELOPER\_ID%TYPE,  in\_new\_developer\_name DEVELOPERS.DEVELOPER\_NAME%TYPE  )  IS  BEGIN  UPDATE DEVELOPERS  SET DEVELOPER\_NAME = in\_new\_developer\_name  WHERE DEVELOPER\_ID = in\_developer\_id  ;  END UPDATE\_DEVELOPER;  ---------- DELETE DEVELOPER PROCEDURE ----------  PROCEDURE DELETE\_DEVELOPER(  in\_developer\_id DEVELOPERS.DEVELOPER\_ID%TYPE  )  IS  BEGIN  DELETE FROM DEVELOPERS  WHERE DEVELOPER\_ID = in\_developer\_id  ;  END DELETE\_DEVELOPER;  ---------- CREATE PUBLUSHER PROCEDURE ----------  PROCEDURE CREATE\_PUBLISHER(  in\_publisher\_name PUBLISHERS.PUBLISHER\_NAME%TYPE  )  IS  BEGIN  INSERT INTO PUBLISHERS(PUBLISHER\_NAME)  VALUES (in\_publisher\_name)  ;  END CREATE\_PUBLISHER;    ---------- UPDATE PUBLUSHER PROCEDURE ----------  PROCEDURE UPDATE\_PUBLISHER(  in\_publisher\_id PUBLISHERS.PUBLISHER\_ID%TYPE,  in\_new\_publisher\_name PUBLISHERS.PUBLISHER\_NAME%TYPE  )  IS  BEGIN  UPDATE PUBLISHERS  SET PUBLISHER\_NAME = in\_new\_publisher\_name  WHERE PUBLISHER\_ID = in\_publisher\_id  ;  END UPDATE\_PUBLISHER;  ---------- DELETE PUBLUSHER PROCEDURE ----------  PROCEDURE DELETE\_PUBLISHER(  in\_publisher\_id PUBLISHERS.PUBLISHER\_ID%TYPE  )  IS  BEGIN  DELETE FROM PUBLISHERS  WHERE PUBLISHER\_ID = in\_publisher\_id  ;  END DELETE\_PUBLISHER;  ---------- CREATE GAME PROCEDURE ----------  PROCEDURE CREATE\_GAME(  in\_title GAMES.GAME\_TITLE%TYPE,  in\_price GAMES.GAME\_PRICE%TYPE,  in\_score GAMES.GAME\_SCORE%TYPE,  in\_age\_limit GAMES.GAME\_AGE\_LIMIT%TYPE,  in\_release\_date GAMES.GAME\_RELEASE\_DATE%TYPE,  in\_description GAMES.GAME\_DESCRIPTION%TYPE  )  IS  BEGIN  INSERT INTO GAMES(GAME\_TITLE, GAME\_PRICE, GAME\_SCORE, GAME\_AGE\_LIMIT, GAME\_RELEASE\_DATE, GAME\_DESCRIPTION)  VALUES (in\_title, in\_price, in\_score, in\_age\_limit, in\_release\_date, in\_description)  ;  END CREATE\_GAME;    ---------- UPDATE GAME PROCEDURE ----------  PROCEDURE UPDATE\_GAME(  in\_game\_id GAMES.GAME\_ID%TYPE,  in\_new\_title GAMES.GAME\_TITLE%TYPE,  in\_new\_price GAMES.GAME\_PRICE%TYPE,  in\_new\_score GAMES.GAME\_SCORE%TYPE,  in\_new\_age\_limit GAMES.GAME\_AGE\_LIMIT%TYPE,  in\_new\_release\_date GAMES.GAME\_RELEASE\_DATE%TYPE,  in\_new\_description GAMES.GAME\_DESCRIPTION%TYPE  )  IS  BEGIN  UPDATE GAMES  SET  GAME\_TITLE = in\_new\_title,  GAME\_PRICE = in\_new\_price,  GAME\_SCORE = in\_new\_score,  GAME\_AGE\_LIMIT = in\_new\_age\_limit,  GAME\_RELEASE\_DATE = in\_new\_release\_date,  GAME\_DESCRIPTION = in\_new\_description  WHERE GAME\_ID = in\_game\_id  ;  END UPDATE\_GAME;  ---------- DELETE GAME PROCEDURE ----------  PROCEDURE DELETE\_GAME(  in\_game\_id GAMES.GAME\_ID%TYPE  )  IS  BEGIN  DELETE FROM GAMES  WHERE GAME\_ID = in\_game\_id  ;  END DELETE\_GAME;  ---------- ADD GENRE PROCEDURE ----------  PROCEDURE ADD\_GENRE(  in\_game\_id GAMES.GAME\_ID%TYPE,  in\_genre\_id GENRES.GENRE\_ID%TYPE  )  IS  BEGIN  INSERT INTO GMS\_GNRS(GAME\_ID, GENRE\_ID)  VALUES (in\_game\_id, in\_genre\_id)  ;  END ADD\_GENRE;  ---------- REMOVE GENRE PROCEDURE ----------  PROCEDURE REMOVE\_GENRE(  in\_game\_id GAMES.GAME\_ID%TYPE,  in\_genre\_id GENRES.GENRE\_ID%TYPE  )  IS  BEGIN  DELETE FROM GMS\_GNRS  WHERE  GAME\_ID = in\_game\_id  AND GENRE\_ID = in\_genre\_id  ;  END REMOVE\_GENRE;  ---------- ADD DEVELOPER PROCEDURE ----------  PROCEDURE ADD\_DEVELOPER(  in\_game\_id GAMES.GAME\_ID%TYPE,  in\_developer\_id DEVELOPERS.DEVELOPER\_ID%TYPE  )  IS  BEGIN  INSERT INTO GMS\_DVLS(GAME\_ID, DEVELOPER\_ID)  VALUES (in\_game\_id, in\_developer\_id)  ;  END ADD\_DEVELOPER;  ---------- REMOVE DEVELOPER PROCEDURE ----------  PROCEDURE REMOVE\_DEVELOPER(  in\_game\_id GAMES.GAME\_ID%TYPE,  in\_developer\_id DEVELOPERS.DEVELOPER\_ID%TYPE  )  IS  BEGIN  DELETE FROM GMS\_DVLS  WHERE  GAME\_ID = in\_game\_id  AND DEVELOPER\_ID = in\_developer\_id  ;  END REMOVE\_DEVELOPER;  ---------- ADD PUBLISHER PROCEDURE ----------  PROCEDURE ADD\_PUBLISHER(  in\_game\_id GAMES.GAME\_ID%TYPE,  in\_publisher\_id PUBLISHERS.PUBLISHER\_ID%TYPE  )  IS  BEGIN  INSERT INTO GMS\_PBLS(GAME\_ID, PUBLISHER\_ID)  VALUES (in\_game\_id, in\_publisher\_id)  ;  END ADD\_PUBLISHER;  ---------- REMOVE PUBLISHER PROCEDURE ----------  PROCEDURE REMOVE\_PUBLISHER(  in\_game\_id GAMES.GAME\_ID%TYPE,  in\_publisher\_id PUBLISHERS.PUBLISHER\_ID%TYPE  )  IS  BEGIN  DELETE FROM GMS\_PBLS  WHERE  GAME\_ID = in\_game\_id  AND PUBLISHER\_ID = in\_publisher\_id  ;  END REMOVE\_PUBLISHER;  END GAMES\_MANAGEMENT\_UTILS;  / |

# Приложение Г Листинг процедур экспорта

|  |
| --- |
| ---------- EXECUTE AS DEVELOPER ----------  ---------- EXPORT ROLES TO JSON PROCEDURE ----------  CREATE OR REPLACE PROCEDURE EXPORT\_ROLES\_TO\_JSON  IS  JSON\_FILE UTL\_FILE.FILE\_TYPE;  JSON\_DATA NVARCHAR2(32000);  BEGIN  SELECT  '['  ||  LISTAGG(JSON\_OBJECT(  'ID' IS ROLE\_ID,  'NAME' IS ROLE\_NAME  ), ',')  ||  ']'  INTO JSON\_DATA  FROM ROLES  ;  JSON\_FILE := UTL\_FILE.FOPEN('JSON\_EXPORT\_DIR', 'ROLES.JSON', 'W');  UTL\_FILE.PUT\_LINE(JSON\_FILE, JSON\_DATA);  UTL\_FILE.FFLUSH(JSON\_FILE);  UTL\_FILE.FCLOSE(JSON\_FILE);  EXCEPTION  WHEN OTHERS THEN BEGIN  UTL\_FILE.FCLOSE(JSON\_FILE);  RAISE;  END;  END EXPORT\_ROLES\_TO\_JSON;  /  ---------- EXPORT USERS TO JSON PROCEDURE ----------  CREATE OR REPLACE PROCEDURE EXPORT\_USERS\_TO\_JSON  IS  JSON\_FILE UTL\_FILE.FILE\_TYPE;  JSON\_DATA NVARCHAR2(32000);  BEGIN  SELECT  '['  ||  LISTAGG(JSON\_OBJECT(  'ID' IS USER\_ID,  'NAME' IS USER\_NAME,  'EMAIL' IS USER\_EMAIL,  'PASSWORD' IS USER\_PASSWORD,  'AVATAR' IS USER\_AVATAR,  'ROLE\_ID' IS USER\_ROLE\_ID  ), ',')  ||  ']'  INTO JSON\_DATA  FROM USERS  ;  JSON\_FILE := UTL\_FILE.FOPEN('JSON\_EXPORT\_DIR', 'USERS.JSON', 'W');  UTL\_FILE.PUT\_LINE(JSON\_FILE, JSON\_DATA);  UTL\_FILE.FFLUSH(JSON\_FILE);  UTL\_FILE.FCLOSE(JSON\_FILE);  EXCEPTION  WHEN OTHERS THEN BEGIN  UTL\_FILE.FCLOSE(JSON\_FILE);  RAISE;  END;  END EXPORT\_USERS\_TO\_JSON;  /  ---------- EXPORT GENRES TO JSON PROCEDURE ----------  CREATE OR REPLACE PROCEDURE EXPORT\_GENRES\_TO\_JSON  IS  JSON\_FILE UTL\_FILE.FILE\_TYPE;  JSON\_DATA NVARCHAR2(32000);  BEGIN  SELECT  '['  ||  LISTAGG(JSON\_OBJECT(  'ID' IS GENRE\_ID,  'NAME' IS GENRE\_NAME  ), ',')  ||  ']'  INTO JSON\_DATA  FROM GENRES  ;  JSON\_FILE := UTL\_FILE.FOPEN('JSON\_EXPORT\_DIR', 'GENRES.JSON', 'W');  UTL\_FILE.PUT\_LINE(JSON\_FILE, JSON\_DATA);  UTL\_FILE.FFLUSH(JSON\_FILE);  UTL\_FILE.FCLOSE(JSON\_FILE);  EXCEPTION  WHEN OTHERS THEN BEGIN  UTL\_FILE.FCLOSE(JSON\_FILE);  RAISE;  END;  END EXPORT\_GENRES\_TO\_JSON;  /  ---------- EXPORT DEVELOPERS TO JSON PROCEDURE ----------  CREATE OR REPLACE PROCEDURE EXPORT\_DEVELOPERS\_TO\_JSON  IS  JSON\_FILE UTL\_FILE.FILE\_TYPE;  JSON\_DATA NVARCHAR2(32000);  BEGIN  SELECT  '['  ||  LISTAGG(JSON\_OBJECT(  'ID' IS DEVELOPER\_ID,  'NAME' IS DEVELOPER\_NAME  ), ',')  ||  ']'  INTO JSON\_DATA  FROM DEVELOPERS  ;  JSON\_FILE := UTL\_FILE.FOPEN('JSON\_EXPORT\_DIR', 'DEVELOPERS.JSON', 'W');  UTL\_FILE.PUT\_LINE(JSON\_FILE, JSON\_DATA);  UTL\_FILE.FFLUSH(JSON\_FILE);  UTL\_FILE.FCLOSE(JSON\_FILE);  EXCEPTION  WHEN OTHERS THEN BEGIN  UTL\_FILE.FCLOSE(JSON\_FILE);  RAISE;  END;  END EXPORT\_DEVELOPERS\_TO\_JSON;  /  ---------- EXPORT PUBLISHERS TO JSON PROCEDURE ----------  CREATE OR REPLACE PROCEDURE EXPORT\_PUBLISHERS\_TO\_JSON  IS  JSON\_FILE UTL\_FILE.FILE\_TYPE;  JSON\_DATA NVARCHAR2(32000);  BEGIN  SELECT  '['  ||  LISTAGG(JSON\_OBJECT(  'ID' IS PUBLISHER\_ID,  'NAME' IS PUBLISHER\_NAME  ), ',')  ||  ']'  INTO JSON\_DATA  FROM PUBLISHERS  ;  JSON\_FILE := UTL\_FILE.FOPEN('JSON\_EXPORT\_DIR', 'PUBLISHERS.JSON', 'W');  UTL\_FILE.PUT\_LINE(JSON\_FILE, JSON\_DATA);  UTL\_FILE.FFLUSH(JSON\_FILE);  UTL\_FILE.FCLOSE(JSON\_FILE);  EXCEPTION  WHEN OTHERS THEN BEGIN  UTL\_FILE.FCLOSE(JSON\_FILE);  RAISE;  END;  END EXPORT\_PUBLISHERS\_TO\_JSON;  /  ---------- EXPORT GAMES TO JSON PROCEDURE ----------  CREATE OR REPLACE PROCEDURE EXPORT\_GAMES\_TO\_JSON  IS  JSON\_FILE UTL\_FILE.FILE\_TYPE;  JSON\_DATA NVARCHAR2(32000);  BEGIN  SELECT  '['  ||  LISTAGG(JSON\_OBJECT(  'ID' IS GAME\_ID,  'TITLE' IS GAME\_TITLE,  'PRICE' IS GAME\_PRICE,  'SCORE' IS GAME\_SCORE,  'AGE\_LIMIT' IS GAME\_AGE\_LIMIT,  'RELEASE\_DATE' IS GAME\_RELEASE\_DATE,  'DESCRIPTION' IS GAME\_DESCRIPTION,  'POSTER' IS GAME\_POSTER,  'COVER' IS GAME\_COVER  ), ',')  ||  ']'  INTO JSON\_DATA  FROM GAMES  ;  JSON\_FILE := UTL\_FILE.FOPEN('JSON\_EXPORT\_DIR', 'GAMES.JSON', 'W');  UTL\_FILE.PUT\_LINE(JSON\_FILE, JSON\_DATA);  UTL\_FILE.FFLUSH(JSON\_FILE);  UTL\_FILE.FCLOSE(JSON\_FILE);  EXCEPTION  WHEN OTHERS THEN BEGIN  UTL\_FILE.FCLOSE(JSON\_FILE);  RAISE;  END;  END EXPORT\_GAMES\_TO\_JSON;  /  ---------- EXPORT SCREENSHOTS TO JSON PROCEDURE ----------  CREATE OR REPLACE PROCEDURE EXPORT\_SCREENSHOTS\_TO\_JSON  IS  JSON\_FILE UTL\_FILE.FILE\_TYPE;  JSON\_DATA NVARCHAR2(32000);  BEGIN  SELECT  '['  ||  LISTAGG(JSON\_OBJECT(  'ID' IS ID,  'GAME\_ID' IS GAME\_ID,  'SCREENSHOT' IS SCREENSHOT  ), ',')  ||  ']'  INTO JSON\_DATA  FROM SCREENSHOTS  ;  JSON\_FILE := UTL\_FILE.FOPEN('JSON\_EXPORT\_DIR', 'SCREENSHOTS.JSON', 'W');  UTL\_FILE.PUT\_LINE(JSON\_FILE, JSON\_DATA);  UTL\_FILE.FFLUSH(JSON\_FILE);  UTL\_FILE.FCLOSE(JSON\_FILE);  EXCEPTION  WHEN OTHERS THEN BEGIN  UTL\_FILE.FCLOSE(JSON\_FILE);  RAISE;  END;  END EXPORT\_SCREENSHOTS\_TO\_JSON;  /  ---------- EXPORT GMS\_GNRS TO JSON PROCEDURE ----------  CREATE OR REPLACE PROCEDURE EXPORT\_GMS\_GNRS\_TO\_JSON  IS  JSON\_FILE UTL\_FILE.FILE\_TYPE;  JSON\_DATA NVARCHAR2(32000);  BEGIN  SELECT  '['  ||  LISTAGG(JSON\_OBJECT(  'GAME\_ID' IS GAME\_ID,  'GENRE\_ID' IS GENRE\_ID  ), ',')  ||  ']'  INTO JSON\_DATA  FROM GMS\_GNRS  ;  JSON\_FILE := UTL\_FILE.FOPEN('JSON\_EXPORT\_DIR', 'GMS\_GNRS.JSON', 'W');  UTL\_FILE.PUT\_LINE(JSON\_FILE, JSON\_DATA);  UTL\_FILE.FFLUSH(JSON\_FILE);  UTL\_FILE.FCLOSE(JSON\_FILE);  EXCEPTION  WHEN OTHERS THEN BEGIN  UTL\_FILE.FCLOSE(JSON\_FILE);  RAISE;  END;  END EXPORT\_GMS\_GNRS\_TO\_JSON;  /  ---------- EXPORT GMS\_DVLS TO JSON PROCEDURE ----------  CREATE OR REPLACE PROCEDURE EXPORT\_GMS\_DVLS\_TO\_JSON  IS  JSON\_FILE UTL\_FILE.FILE\_TYPE;  JSON\_DATA NVARCHAR2(32000);  BEGIN  SELECT  '['  ||  LISTAGG(JSON\_OBJECT(  'GAME\_ID' IS GAME\_ID,  'DEVELOPER\_ID' IS DEVELOPER\_ID  ), ',')  ||  ']'  INTO JSON\_DATA  FROM GMS\_DVLS  ;  JSON\_FILE := UTL\_FILE.FOPEN('JSON\_EXPORT\_DIR', 'GMS\_DVLS.JSON', 'W');  UTL\_FILE.PUT\_LINE(JSON\_FILE, JSON\_DATA);  UTL\_FILE.FFLUSH(JSON\_FILE);  UTL\_FILE.FCLOSE(JSON\_FILE);  EXCEPTION  WHEN OTHERS THEN BEGIN  UTL\_FILE.FCLOSE(JSON\_FILE);  RAISE;  END;  END EXPORT\_GMS\_DVLS\_TO\_JSON;  /  ---------- EXPORT GMS\_PBLS TO JSON PROCEDURE ----------  CREATE OR REPLACE PROCEDURE EXPORT\_GMS\_PBLS\_TO\_JSON  IS  JSON\_FILE UTL\_FILE.FILE\_TYPE;  JSON\_DATA NVARCHAR2(32000);  BEGIN  SELECT  '['  ||  LISTAGG(JSON\_OBJECT(  'GAME\_ID' IS GAME\_ID,  'PUBLISHER\_ID' IS PUBLISHER\_ID  ), ',')  ||  ']'  INTO JSON\_DATA  FROM GMS\_PBLS  ;  JSON\_FILE := UTL\_FILE.FOPEN('JSON\_EXPORT\_DIR', 'GMS\_PBLS.JSON', 'W');  UTL\_FILE.PUT\_LINE(JSON\_FILE, JSON\_DATA);  UTL\_FILE.FFLUSH(JSON\_FILE);  UTL\_FILE.FCLOSE(JSON\_FILE);  EXCEPTION  WHEN OTHERS THEN BEGIN  UTL\_FILE.FCLOSE(JSON\_FILE);  RAISE;  END;  END EXPORT\_GMS\_PBLS\_TO\_JSON;  /  ---------- EXPORT WISHLISTS TO JSON PROCEDURE ----------  CREATE OR REPLACE PROCEDURE EXPORT\_WISHLISTS\_TO\_JSON  IS  JSON\_FILE UTL\_FILE.FILE\_TYPE;  JSON\_DATA NVARCHAR2(32000);  BEGIN  SELECT  '['  ||  LISTAGG(JSON\_OBJECT(  'USER\_ID' IS USER\_ID,  'GAME\_ID' IS GAME\_ID  ), ',')  ||  ']'  INTO JSON\_DATA  FROM WISHLISTS  ;  JSON\_FILE := UTL\_FILE.FOPEN('JSON\_EXPORT\_DIR', 'WISHLISTS.JSON', 'W');  UTL\_FILE.PUT\_LINE(JSON\_FILE, JSON\_DATA);  UTL\_FILE.FFLUSH(JSON\_FILE);  UTL\_FILE.FCLOSE(JSON\_FILE);  EXCEPTION  WHEN OTHERS THEN BEGIN  UTL\_FILE.FCLOSE(JSON\_FILE);  RAISE;  END;  END EXPORT\_WISHLISTS\_TO\_JSON;  /  ---------- EXPORT CARTS TO JSON PROCEDURE ----------  CREATE OR REPLACE PROCEDURE EXPORT\_CARTS\_TO\_JSON  IS  JSON\_FILE UTL\_FILE.FILE\_TYPE;  JSON\_DATA NVARCHAR2(32000);  BEGIN  SELECT  '['  ||  LISTAGG(JSON\_OBJECT(  'USER\_ID' IS USER\_ID,  'GAME\_ID' IS GAME\_ID  ), ',')  ||  ']'  INTO JSON\_DATA  FROM CARTS  ;  JSON\_FILE := UTL\_FILE.FOPEN('JSON\_EXPORT\_DIR', 'CARTS.JSON', 'W');  UTL\_FILE.PUT\_LINE(JSON\_FILE, JSON\_DATA);  UTL\_FILE.FFLUSH(JSON\_FILE);  UTL\_FILE.FCLOSE(JSON\_FILE);  EXCEPTION  WHEN OTHERS THEN BEGIN  UTL\_FILE.FCLOSE(JSON\_FILE);  RAISE;  END;  END EXPORT\_CARTS\_TO\_JSON;  /  ---------- EXPORT LIBRARIES TO JSON PROCEDURE ----------  CREATE OR REPLACE PROCEDURE EXPORT\_LIBRARIES\_TO\_JSON  IS  JSON\_FILE UTL\_FILE.FILE\_TYPE;  JSON\_DATA NVARCHAR2(32000);  BEGIN  SELECT  '['  ||  LISTAGG(JSON\_OBJECT(  'USER\_ID' IS USER\_ID,  'GAME\_ID' IS GAME\_ID,  'FAVORITE' IS FAVORITE  ), ',')  ||  ']'  INTO JSON\_DATA  FROM LIBRARIES  ;  JSON\_FILE := UTL\_FILE.FOPEN('JSON\_EXPORT\_DIR', 'LIBRARIES.JSON', 'W');  UTL\_FILE.PUT\_LINE(JSON\_FILE, JSON\_DATA);  UTL\_FILE.FFLUSH(JSON\_FILE);  UTL\_FILE.FCLOSE(JSON\_FILE);  EXCEPTION  WHEN OTHERS THEN BEGIN  UTL\_FILE.FCLOSE(JSON\_FILE);  RAISE;  END;  END EXPORT\_LIBRARIES\_TO\_JSON;  /  ---------- EXPORT RATINGS TO JSON PROCEDURE ----------  CREATE OR REPLACE PROCEDURE EXPORT\_RATINGS\_TO\_JSON  IS  JSON\_FILE UTL\_FILE.FILE\_TYPE;  JSON\_DATA NVARCHAR2(32000);  BEGIN  SELECT  '['  ||  LISTAGG(JSON\_OBJECT(  'USER\_ID' IS USER\_ID,  'GAME\_ID' IS GAME\_ID,  'RATING' IS RATING  ), ',')  ||  ']'  INTO JSON\_DATA  FROM RATINGS  ;  JSON\_FILE := UTL\_FILE.FOPEN('JSON\_EXPORT\_DIR', 'RATINGS.JSON', 'W');  UTL\_FILE.PUT\_LINE(JSON\_FILE, JSON\_DATA);  UTL\_FILE.FFLUSH(JSON\_FILE);  UTL\_FILE.FCLOSE(JSON\_FILE);  EXCEPTION  WHEN OTHERS THEN BEGIN  UTL\_FILE.FCLOSE(JSON\_FILE);  RAISE;  END;  END EXPORT\_RATINGS\_TO\_JSON;  / |