МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждение образования «БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет Информационных технологий
Кафедра Информационные системы и технологии
Специальность 6-05-0612-01 Программная инженерия (профилизация Программное обеспечение информационных технологий)

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА К КУРСОВОЙ РАБОТЕ НА ТЕМУ:

« Реализация базы данных игровой платформы с использованием технологии In-Memory »

Выполнил студент Стахейко Антон Алексеевич
(Ф.И.О.)

Руководитель работы ст. преп. Нистюк О. А.
(учен. степень, звание, должность, Ф.И.О., подпись)

Зав. кафедрой ст. преп., к.т.н Блинова Е.А.
(учен. степень, звание, должность, Ф.И.О., подпись)

Курсовая работа защищена с оценкой

Минск 2025

Содержание

Введение

Цель данного проекта состоит в создании реляционной базы данных для игровой платформы с использованием технологии In-Memory, обеспечивая доступ пользователю ко всем товарам.

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.

База данных разрабатывается с расчётом на масштабируемость и высокую производительность. Поскольку система предназначена для обслуживания большого количества одновременных пользователей, в ней применяется технология In-Memory. Она обеспечивает хранение наиболее часто используемых данных непосредственно в оперативной памяти сервера, что позволяет существенно сократить время отклика и повысить скорость обработки запросов. Такой подход особенно эффективен при работе с динамическими данными Использование In-Memory-технологии способствует повышению масштабируемости системы и обеспечивает стабильную работу под высокой нагрузкой.

1 Постановка задачи

1.1 Анализ аналогичных решений

Интернет-магазины прочно вошли в повседневную жизнь, предлагая широкий выбор товаров и услуг, доступных прямо из дома. Поэтому при разработке базы данных для такого сервиса важно тщательно проанализировать и объективно оценить функциональные возможности, применяемые технологии и прочие ключевые аспекты.

После проведения анализа требуется составить конкретный перечень функциональных требований к базе данных и создать диаграмму вариантов использования. Основная задача проекта заключается в разработке архитектуры базы данных и проведении тестирования.

Основные требования к реализации:

– Определение ролей (Администратор, Разработчик, Пользовтель);

– Возможность добавления, удаления и редактирования видеоигры (Разрабочик);

– Поиск видеоигр по категориям и критериям (Пользователь);

– Добавление видеоигры в корзину или избранное (Пользователь);

– Возможность приобрести или скачать игру (Пользователь);

– Манипуляция с записями пользователей и записями о играх (Администратор)

– Взаимодействие с базой данных при помощи хранимых процедур и функций.

1.1.1 Аналог Steam

Steam является одной из ведущих платформ для цифровой дистрибуции видеоигр и программного обеспечения. Steam стал популярен по причине что он открыл этот рынок и сразу завоевал признание клиентво, и продолжает удерживать его до сих пор. Интерфейс Steam представлен на рисунке 1.1.

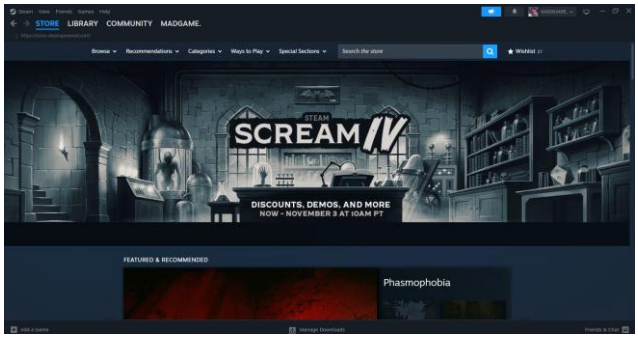


Рисунок 1.1 – Интерфейс сервиса Steam

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 предлагает широкий ассортимент игр, включая как классические хиты, так и самые новые релизы. Сайт обеспечивает удобный поиск по различным категориям и ценовым диапазонам.

Уникальной особенностью Epic Games Store является возможность быстрой регистрации через социальные сети, что существенно упрощает процесс начала работы с сервисом и снижает порог входа для новых пользователей. Кроме того, платформа регулярно предлагает бесплатные игры и эксклюзивные проекты, что повышает её привлекательность среди широкой аудитории игроков. Пример интерфейса данного сервиса представлен на рисунке 1.2.

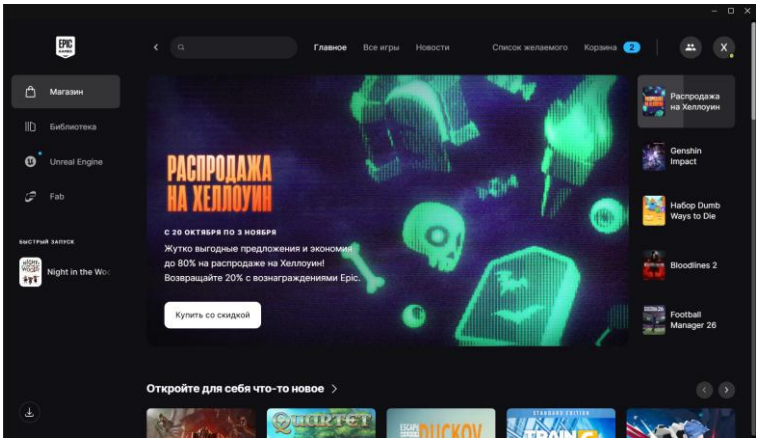


Рисунок 1.2 – Интерфейс сервиса Epic Games Store

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.

1.1.3 GameJolt

GameJolt — это онлайн-платформа, ориентированная на независимых разработчиков и сообщество игроков. Она объединяет в себе магазин игр, социальную сеть и площадку для публикации творческих проектов. На Game Jolt пользователи могут не только скачивать и запускать игры, но и делиться своими работами, вести блоги, участвовать в обсуждениях и конкурсах. Пример интерфейса GameJolt педставлен на рисунке 1.3.

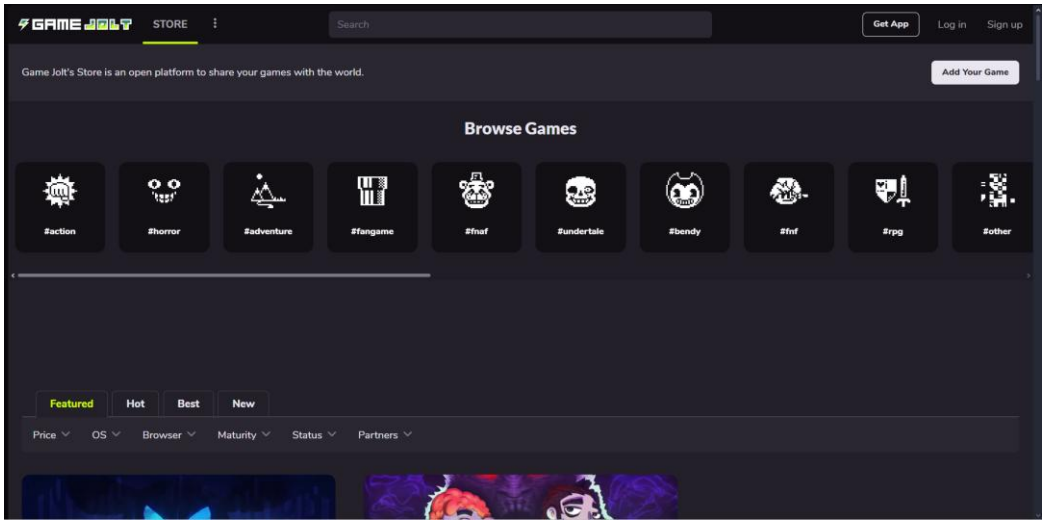


Рисунок 1.2 – Интерфейс сервиса GameJolt

Поскольку на платформе активно создаются и обновляются пользовательские данные — посты, комментарии, ленты активности, личные сообщения, — важно обеспечить не только сохранность информации, но и высокую скорость обработки большого количества мелких запросов. Поэтому в подобных системах часто используется гибридный подход, сочетающий реляционные и нереляционные решения. Основная часть данных, требующих строгой согласованности и транзакционности (пользователи, игры, публикации, связи между ними), обычно хранится в реляционной СУБД, например, PostgreSQL. Она обеспечивает надёжность, поддержку сложных связей и индексов, а также удобство при построении аналитических запросов.

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 игровой платформы функциональные требования могут включать функции для хранения информации о файлах видеоиграх и страничек игр, возможность поиска товаров по различным категориям и критериям, создание и удаление товаров, возможность купить товар и получить ссылку на скачивание, а также функции оценки товаров, и анализа статистики.

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;

– разработать функционал для получения статистики.

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

– Разработчик

– Администратор

Каждая из перечисленных ролей обладает определённым набором прав и возможностей. Гость имеет доступ к просмотру общей информации и каталога игр без возможности совершения каких-либо действий. Пользователь может регистрироваться в системе, приобретать и загружать игры, а также управлять своим профилем. Разработчик получает доступ к функционалу публикации и обновления игр, а также к просмотру статистики. Администратор отвечает за управление системой в целом, включая контроль пользователей, контента и корректности работы сервиса. На основе предоставленного списка ролей необходимо построить варианты использования. Варианты использования изображены на рисунке 1.4.

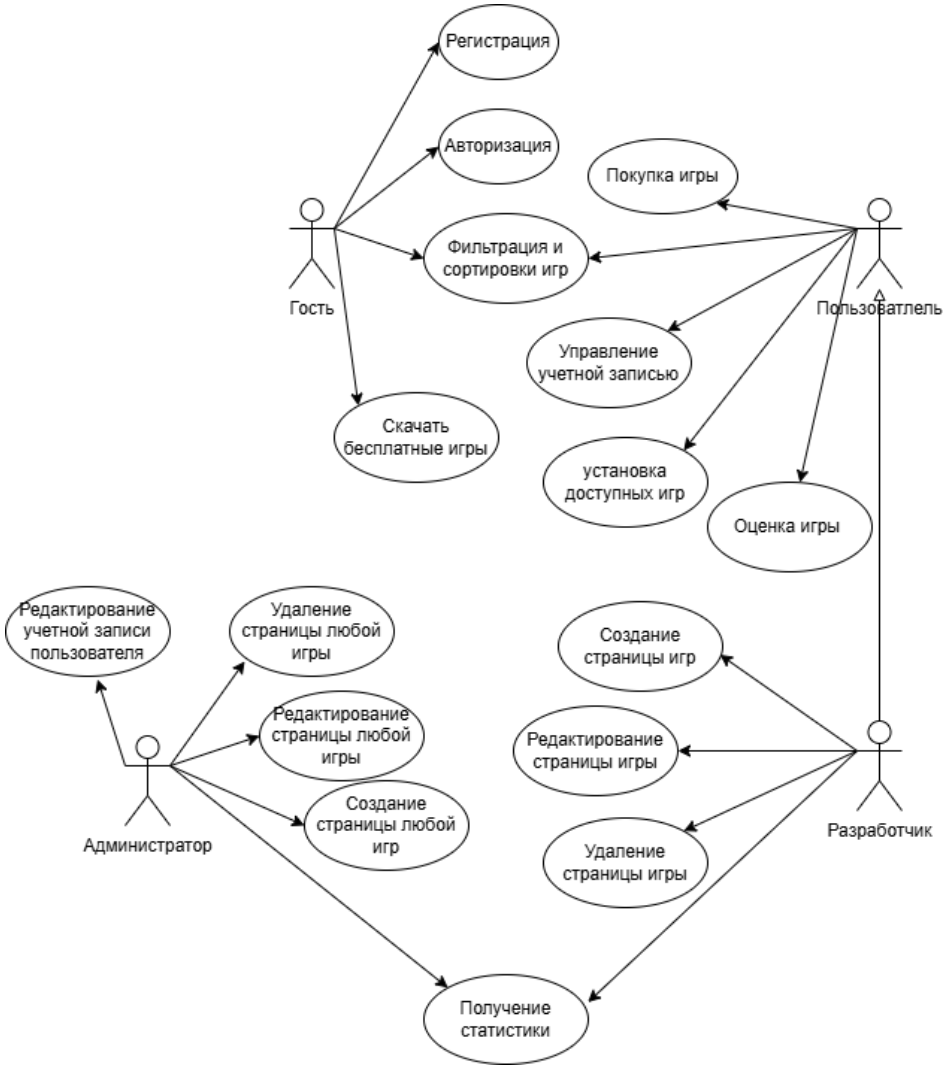


Рисунок 1.3 – UML диаграмма вариантов использования

В начале работы с приложением пользователь является гостем. Ему будет доступна просмотр катологa доступных игр, фильтрация и сортировка, и возможность скачать бесплатные. После регистрации пользователь становится зарегистрированным пользователем.

Роль Пользователь получает возможность осуществления покупки платных игр, оставление отзыва, и возможность скачать все доступные пользователю игры, так же пользователь может редактировать свою учетную запись.

Роль Разработчика имеет все возможности пользователя и получает возможность создать страничку своей игры, а также редактировать и удалять уже существующие видеоигры. Он является управляющим контентом площадки, так же просмотр статистики опубликованных игр.

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, а также информацию о пользователях, заказах и их статусах. Была также выявлена необходимость в функциях по 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.

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

2 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.

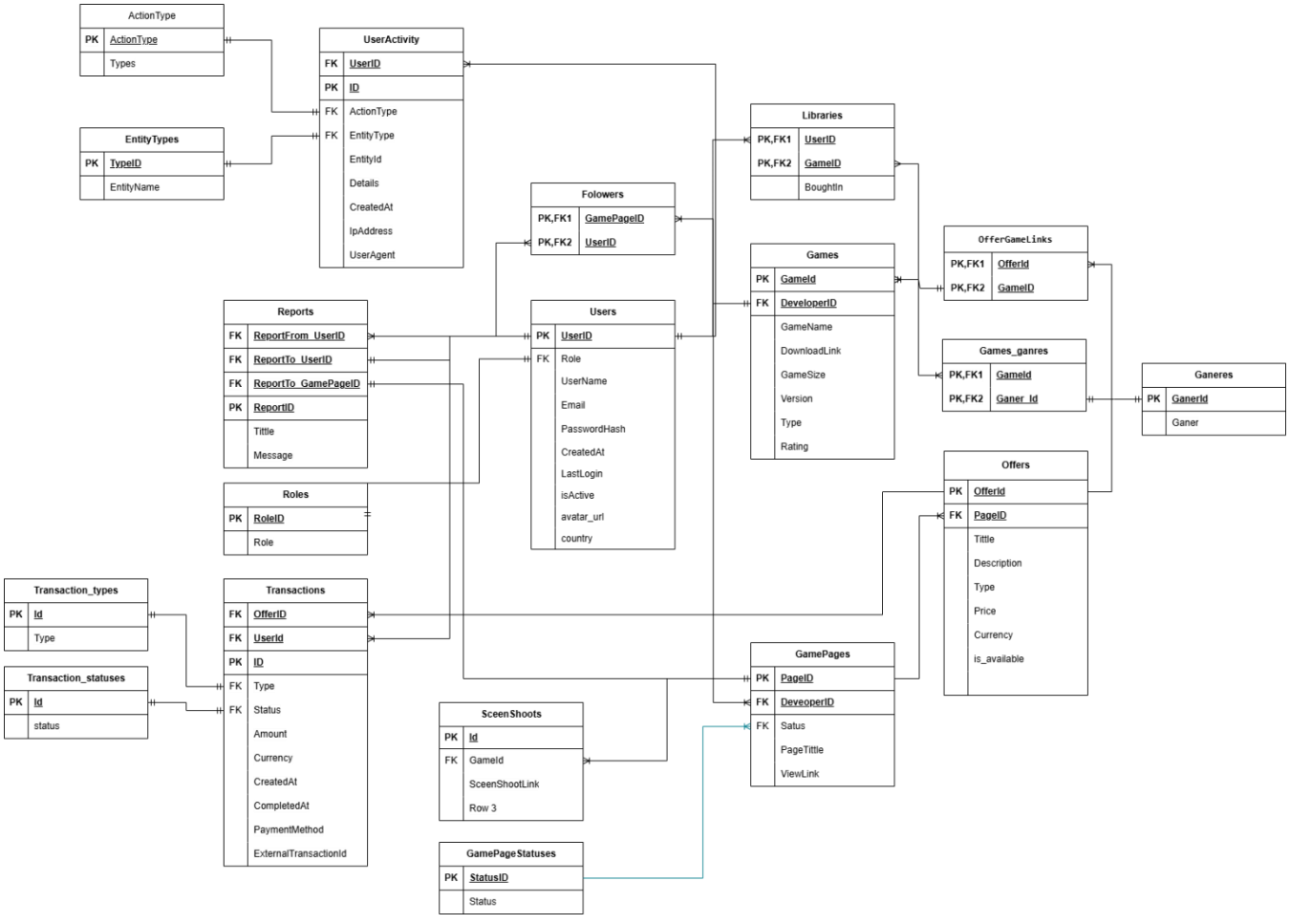


Рисунок 2.1 – Схема базы данных

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,

Folowers, UserActivity, Libraries, Games, GamePages, Reports и Transctions через внешние ключи user_id. Также видно, что таблица Games связана с таблицами Offers через дополнительную таблицы для организации связей "многие-ко-многим". Кроме того, таблица Screenshots связана с таблицей GamePages через внешний ключ PageId. Связи между таблицами Users и Roles и таблицами GamePages и Screenshots представляют собой связи "один-ко-многим".

2.2 Описание информационных объектов и ограничений целостности

Для реализации базы данных было разработано 17 таблиц. В структуру схемы базы данных для проекта входят следующие таблицы: ActionType, EntityTypes, UserActivity, Libraries, Folowers, Reports, Users, Games, OfferGameLinks, Roles, Games_ganres, Ganeres, Transaction_types, Transaction_statuses, Transactions, Offers, GamePages, GamePageStatuses и SceenShoots. Ниже приведено описание каждой таблицы, включающее типы данных столбцов и ограничения целостности.

Первой размариваемой таблицей является ActionTypes, она является вспомогательной таблицей для другой, и все что в себе хранит это типы действий, которые могут быть совершенны пользователем, для использования этих данных в других таблицах. Структура таблицы представлена в таблице 2.1

Имя столбца	Тип атрибута	Пояснения	Тип данных
ActionType	PK	Индефикатор типа	NUMBER
Type		Имя типа	VARCHAR2(125)

Таблица 2.1 – Структура таблицы ActionTypes

Следующей рассматриваемой таблицей является таблица UserActivity, эта таблица автоматически заполняется в следствие вызова разных процедур пользователей. К примеру, при скачивание какой-нибудь игры, данные об этой операции будут занесены в эту таблицы. Нужна она для сбора статистики о пользователях и играх. Структура приведена в таблице 2.2

Имя стобца	Тип атрибута	Пояснения	Тип данных
ID	PK	Индефикатор записи	NUMBER
UserID	FK	Индефикатор пользователя	NUMBER
ActionType	FK	Индефикатора типа события	NUMBER

EntityType	FK	Индефикатор на обьект события	NUMBER
EntityID		Индификатор обьекта события	NUMBER
Details		Дополнительнбные детали	NVARCHAR2(255)
CreatedAt		Дата когда совершенно событие	TIMESTAMP WITH TIME ZONE
IpAddress		IP-адресс откуда пришло событие	VARCHAR2(45)
UserAgent		Источник собтия(браузер, приложени)	VARCHAR(512)

Таблица 2.2 – Структура таблицы UserActivity

Следящая рассматриваемая таблица — это EntityTypes. Эта таблица так же является вспомогательной для UserActivity, так как действия, совершаемые пользователем могут распространяться на разные сущность, должен быть способ определить над какой сущностью было совершенно действие, для этого и была введена эта таблица, структура ее представлена в таблицу 2.3

Имя стобца	Тип атрибута	Пояснения	Тип данных
TypeID	PK	Индефикатор типа	NUMBER
EntityName		Имя сущности	VARCHAR(125)

Таблица 2.3 – Структура таблицы EntityTypes

Следующей таблицей является таблица Folowers, она нужна чтоб пользователи могли оставлять оценки и комментарии на страницах игр, структура ее приведена в таблице 2.4

Имя столбца	Тип атрибута	Пояснения	Тип данных
GamePageId	PK,FK	Индефикатор страницы игры	NUMBER
UserId	PK,FK	Индефикатор пользователя	NUMBER
Rating		Оценка поставленная пользователем	NUMBER(1)
Comment		Коментарий оставленный пользователем	NVARCAHR(2000)

Таблица 2.4 – Структура таблицы Folowers

Следующая рассматриваемое таблица так же является вспомогательной. Roles - эта таблица нужна чтоб указать какой пользователь принадлежит какой роле, и в зависимости от этого наложить ограничения на его учетную запись или на оборот выдать привилегии. Структура предоставлена в таблице 2.5

Имя столбца	Тип атрибута	Пояснения	Тип данных
RoleId	PK	Идентификатор роли	NUMBER
Role		Название роли	NVARCHAR2(255)

Таблица 2.5 – Структура таблицы Roles

Следующая рассматриваемое таблица Users. Это одна из самых главных таблиц, в ней хранятся данные от зарегистрированных пользователей. Структура таблицы приведена в таблице 2.6.

Имя столбца	Тип атрибута	Пояснения	Тип данных
userID	PK	Идентификатор пользователя	NUMBER
UserName		Имя пользователя	NVARCHAR2(255)
Email		E-mail пользователя	NVARCHAR2(255)
PasswordHash		Хеш пароля пользователя	NVARCHAR2(255)
CreatedAt		Дата создани аккаунта	TIMESTAMP WITH TIME ZONE
LastLogin		Время последнего захода	TIMESTAMP WITH TIME ZONE
isActive		Активен ли пользователь	NUMBER(1)
Avatar_uri		Путь к аватарке пользователя	NVARCHAR2(255)
Role	FK	Идентификатор роли пользователя	NUMBER
Country		Страна пользователя	NVARCHAR2(255)

Таблица 2.6 – Структура таблицы Users

Следующая таблица - Geners, предназначена для хранить жанров, которые используются в страницах игр, структура приведена в таблице 2.7.

Имя столбца	Тип атрибута	Пояснения	Тип данных
genreId	PK	Идентификатор жанра	NUMBER
genre		Название жанра	NVARCHAR2(255)

Таблица 2.7 – Структура таблицы Geners

Следующая таблица - Reports, предназначена для хранения данных о жалобах пользователей на других пользователей или станицы игр, для последующих действий от лица пользователя с правами администратора, структура приведена в таблице 2.8

Имя стобца	Тип атрибута	Пояснения	Тип данных
ReportID	PK	Идентификатор разработчика	NUMBER
ReportFrom_UserId	FK	От кого жалоба	NUMBER
ReportTo_UserID	FK	На какого пользователя жалоба	NUMBER
ReportTo_GamePageID	FK	На какую страницу игры жалоба	NUMBER
Tittle		Заголовок	NVARCHAR2(255)
Message		Сообшение	NVARCHAR2(2000)

Таблица 2.8 – Структура таблицы Reports

Следующая таблица OfferGameLinks, нужна для связи опции покупки и страницы игры, нужно это так как на одной странице, может быть, несколько предложений о покупке, для этого и была создана эта таблица, Структура таблицы приведена в таблице 2.9.

Имя столбца	Тип атрибута	Пояснения	Тип данных
OfferId	PK FK	Идентификатор предложения	NUMBER
GameID	PK FK	Индефикатор игры	NUMBER

Таблица 2.9 – Структура таблицы OfferGameLinks

Следующая таблица так же является одной из основных. Games - таблица, предназначенная для хранения данных о когда-либо опубликованных игр, особенность заключается в том, если файлы игры когда-либо были опубликованы, записи о них не могут уже быть удалены, сделано это для того, если игра была приобретена пользователем, нельзя допустить чтоб в одни момент даже из-за случайности данные о ней были потеряны. Структура приведена в таблице 2.10.

Имя стобца	Тип атрибута	Пояснения	Тип данных
GameID	PK	Идентификатор игры	NUMBER
DeveloperID	FK	Индефикатор разработчика	NUMBER

GameName		Название игры	NVARCHAR2(512)
DownloadLink		Ссылка для скачивания	VARCHAR(512)
GameSize		Размер игры(байт)	NUMBER(20)
Version		Дата выхода видеоигры	VARCHAR2(20)
type		Тип контента(игра, DLC)	NVARCHAR2(125)

Таблица 2.10 – Структура таблицы Games

Следующая таблица Screenshots предназначена для хранения ссылок для получения скриншотов, используемых в странице игры. Структрура приведена в таблице 2.11.

Имя стобца	Тип атрибута	Пояснения	Тип данных
id	PK	Идентификатор скриншота	NUMBER
gameid	FK	Идентификатор видеоигры	NUMBER
screenshotLink		Скриншот	VARCHAR(512)

Таблица 2.11 – Структура таблицы Screenshots

Следующая таблица предназначена для создания предложений о покупке, особенность заключается в том что на одно странице, может быть, несколько предложений, как и в предложении может быть несколько игр. Структура приведена в таблице 2.12.

Имя стобца	Тип атрибута	Пояснения	Тип данных
OfferId	PK	Идентификатор предложения	NUMBER
PageID	FK	Идентификатор страницы на которой это предложение	NUMBER
Tittle		Заголовок предложения	NVARCHAR2(512)
Description		Описание	NVARCHAR2(1000)
Price		Ценна	NUMBER(10,2)
Currency		Валюта	NVARCHAR2(6)

Таблица 2.12 – Структура таблицы Offers

Следующая таблица так же является одной из основных, она предназначена для хранения данных, о страницах игр, которые может увидеть пользователь, основана логика заключается в том, что игра не привязана к конкретной странице, страница разрабатывается отдельно от публикации

фалов игры, такая архитектурная является более гибкой и масштабируемой. Структура приведена в таблице 2.13.

Имя стобца	Тип атрибута	Пояснения	Тип данных
PageID	PK	Идентификатор страницы	NUMBER
developerId	FK	Идентификатор разработчика	NUMBER
Status	Fk	ID стаутса страницы	NUMBER
PageTittle		Заголовок страницы	NVARCHAR2(512)
ViewLink		Сылка на заглавное изображен пользователя	NVARCHAR2(512)

Таблица 2.12 – Структура таблицы GamePages

Следующая таблица является вспомогательной, и необходима для хранения в каком статусе может находиться страница игры. Структура таблицы приведена в таблице 2.13.

Имя стобца	Тип атрибута	Пояснения	Тип данных
StatusID	PK	ID статуса	INTEGER
Status		Имя статуса	VARCHAR(512)

Таблица 2.13 – Структура таблицы GamePagesStatuses

Следующая таблица так же является достаточно важная, она предназначена для хранения финансовых транзакций, производимых пользователями, изменять или заполнять ее не может никакой пользователь, она заполняется исключительно автоматически в процессе работы остальных процедур, сделано это для гарантии безопасности и предотвращения не правомерного использования. Структура таблиц приведена в таблице 2.14.

Имя стобца	Тип атрибута	Пояснения	Тип данных
ID	PK	ID записи	NUMBER
OfferID	FK	Идентификатор предложения	NUMBER
UserID	FK	Индефикатор пользователя	NUMBER
TYPE	FK	Тип транзакции	NUMBER
Status	FK	Статус транзакции	NUMBER
Amount		Ценна	NUMBER(10,2)
Currency		Валюта	NVARCHAR2(6)

CreatedAt		Время когда транзакция началась	TIMESTAMP WITH TIME ZONE
CompletedAt		Время когда тарнзакция завершилась	TIMESTAMP WITH TIME ZONE
PaymentMethod		Способ оплаты	NVARCHAR2(125)
ExternalTransactionId		Id строннего метода оплаты	NVARCHAR2(512)

Таблица 2.14 – Структура таблицы Transactions

Следующая таблица так же является служебной, и предношена для хранения типа производимой транзакции (пополнение средств, покупка и т.д.), структура приведена в таблице 2.15

Имя стобца	Тип атрибута	Пояснения	Тип данных
ID	PK	Идентификатор типа	NUMBER
Type		Имя типа	VARCHAR(125)

Таблица 2.15 – Структура таблицы TransctionType

Следующая таблица предназначена для хранения приобретенных пользователем игр, структура приведена в таблице 2.16.

Имя стобца	Тип атрибута	Пояснения	Тип данных
userId	PK FK	Идентификатор пользователя	BIGINT
gameId	PK FK	Идентификатор видеоигры	BIGINT
BoughtIn		Дата покупки	TIMESTAMP WITH TIME ZONE

Таблица 2.16 – Структура таблицы Libraries

Последняя таблица так же служебная и предназначена для хранения статусов, в которых может находиться транзакции производимая пользователем. Структура приведена в таблице 2.17

Имя столбца	Тип атрибута	Пояснения	Тип данных
id	PK	Идентификатор	INTEGER
status		Имя статуса	VARCHAR(125)

Таблица 2.17 – Структура таблицы Transction_statuses

Таким образом, была определена структура таблиц и их ограничений.

2.3 Разработка процедур базы данных

Все процедуры можно разбить на несколько общих групп, допустим по пользователям которым они предназначаются и сделать пару общих процедур.

В таблице ниже представлены процедуры предназначенные для пользователя “Гость”

Имя процедуры	Входные данные	Пояснения
register_guest	Имя пользователя, Пароль, Email	Процедура предназначена для создания новой учетной записи.
login_guest	Email, Пароль	Процедура должна найти пользователя согласно входным данные, проверить их и вернуть id пользователя
TryDownload	Id пердложения	Процедура должна произвести попытка получить ссылку для установки если игра беспалая
get_game_pages_filtered	Параметры для фильтрации если нужны	Процедура предназначена для формирования списка доступных игры и фильтровать их

Таблица 2.18 – Примерные процедуры для гостя

В таблице ниже представлены примерные процедуры для авторизованного пользователя, у него в свою очередь привилегий больше, чем у гостя.

Имя процедуры	Входные данные	Пояснения
update_profile	Id_пользоватлея, Ссылка на новый аватар, Страна	Процедура предназначена для изменения второстепенных данных о пользователе
update_password	Id_пользоватлея, Старый пароль, Новый пароль	Процедура должна проверить соответствие данных в нормальном случаее изменить пароль
update_nickname	Id_пользоватлея, Новый никнейм,	Процедура должна изменить некнейм пользователя.
update_email	Id_пользоватлея, Новый email	Процедура должна изменить email пользователя
add_balance_transaction	Id_пользоватлея, Сумма пополнения, Способ	Процеудра должна совершить транзакцию по пополнению внутреннего баланса пользоателя
purchase_game_pending	Id_пользователя, Id_предложения,	Процеудра должна начать тарнзакции покупки какогото предложения

	Способ покупке(по карте или внутриний баланс)	
complete_purchase	Id_транзакции	В случае если на предыдушем этапе все прошло хорошо транзакция фиксируется этой процедурой.
get_user_library	Id_Пользователя	Процедура должна вернуть библиотеку пользователя
download_library_game	ID_пользователя ID_игры	Процеудра должна проверить пренадлежит ли игра пользователю, в случае успеха вернуть ему ссылку на скачивание

Таблица 2.19 – Примерные процедуры для авторизованного пользователя

В таблице ниже предстваленны ключевые процедуры которые должны быть у пользователя имеюшего статус разарботчика.

Имя процедуры	Входные данные	Пояснения
add_game	Id_Разработчика, Имя игры, Сылека на скачивание, Размер игры	Процеудра должна добавлять записи о новой игре.
get_all_games	Id_Разработчика	Процедура должна вернуть все опубликованные игры разработчиком
create_game_page	Id_Разработчика Заголовок страницы Ссылка на содержимо	Процедура должна создать базовые данные для страницы игры
add_offer	Id_Разработчика Id_страницы Заголовок предложения ценна	Процеудра должна создать предложения для страницы игры

Таблица 2.20 – Примерные процедуры для Разработчика

Также, разумеется, для разработчика необходимо реализовать процедуры для изменения уже созданных предложений игр, их удаления, а также вспомогательные процедуры, обеспечивающие корректную работу функционала управления контентом. Данные процедуры позволяют поддерживать актуальность информации об играх и оперативно вносить изменения при необходимости.

У админимтратора в свою роль должны быть похожие процеудуры напроцедуры других пользоватлей только без ограничений и исключений, в разумных пределах, исключение администратор единственный кто может получить доступ к таблице транзакций через процедуры, но исключительно прочитать их, не меняя ничего.

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 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 17 таблиц, каждая из которых выполняет свою уникальную функцию и содержит определенные данные. Основные таблицы включают в себя: Users, Genres, GamePages, Offers, transactions, Games.

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 базы данных.

3.1 Табличные пространства базы данных

Для любой базы данных первое что необходимо сделать это разработать табличные пространства, так как именно они используются в процессе создания всех остальных объектов в БД, неправильно настроенное табличное пространство может привести что объекты БД не смогут каретной выполнять свои задачи, или БД не сможет обслуживать большое количество данных.

Создание табличных пространств находится в файле “01_create_pdb_and_tablespaces.sql” в нем же находится и создание PDB, данный скрипт должен выполняться от роли SYS, строчки касавшиеся создания табличных представлены представленны в листинге 3.1

```
create tablespace gs_data
    datafile '/opt/oracle/oradata/ORCLCDB/gsdb/gs_data01.dbf'
    size 500m
    autoextend on  next 50m
    maxsize  UNLIMITED
    EXTENT MANAGEMENT LOCAL;


create tablespace gs_index
    datafile '/opt/oracle/oradata/ORCLCDB/gsdb/gs_index01.dbf'
    size 250m
    autoextend on next 100m
    maxsize UNLIMITED
    EXTENT MANAGEMENT LOCAL;


Create TEMPORARY tablespace gs_temp
    tempfile '/opt/oracle/oradata/ORCLCDB/gsdb/gs_temp01.dbf'
    size 200M
```

```
    AUTOEXTEND ON NEXT 50M
    MAXSIZE 1G
    EXTENT MANAGEMENT LOCAL UNIFORM SIZE 1M;


create UNDO tablespace gs_undo
    datafile '/opt/oracle/oradata/ORCLCDB/gsdb/gs_undo01.dbf'
    SIZE 250M
    AUTOEXTEND ON NEXT 50M
    maxsize 1G;
```

Листинг 3.1 – Табличные пространства

Всего были разработаны 4 табличных пространств:

- gs_data - основное табличное пространство для всех объектов;
- gs_index - табличное пространство для индексов;
- gs_temp - табличное пространство для временных данных;
- gs_undo - для откатов;

3.2 Создание основных ролей и пользователей базы данных

Следующим важным этапом, является разработка профилей безопастности и пользователей, которые будут подключатся в БД извне. Всего было определено 4 пользователя и соответствующие для них профили с разными уровнями безопасности и доступным функционалом.

- Гость (GUEST) - не авторизованный пользователь, доступны только процедуры для чтения данных и скачивания бесплатных игр.
- Авторизованный пользователь (USER_APP) - Пользователь со своим профилем, может редактировать свой профиль, и покупать платные игры.
- Разработчик (DEVELOPER) - Пользователь имеющий статус разработчика может загружать файлы своих игр, создавать странички и предложения.
- Администратор (ADMIN) - Администратор базы данных, пользователь имеющий больше всех привилегий и может обращаться к любым объектам

Скрипты для создания профилей безопасности и пользователей реализованы в файле “03_create_users.sql”, и должен выполняться от лица пользователя “app_user”. Фрагменты файла кусающиеся создание профилей и пользователей представлены в листинге 3.2.

```
CREATE PROFILE GUEST_PROFILE LIMIT
    SESSIONS_PER_USER      10
    FAILED_LOGIN_ATTEMPTS  UNLIMITED
    PASSWORD_LIFE_TIME     UNLIMITED
    PASSWORD_REUSE_TIME    1
    PASSWORD_REUSE_MAX     UNLIMITED;

CREATE PROFILE USER_PROFILE LIMIT
    SESSIONS_PER_USER      3
    FAILED_LOGIN_ATTEMPTS  5
    PASSWORD_LIFE_TIME     180
    PASSWORD_REUSE_TIME    30
    PASSWORD_REUSE_MAX     5;

CREATE PROFILE DEVELOPER_PROFILE LIMIT
    SESSIONS_PER_USER      10
    FAILED_LOGIN_ATTEMPTS  5
    PASSWORD_LIFE_TIME     90
    PASSWORD_VERIFY_FUNCTION ORA12C_VERIFY_FUNCTION;

CREATE PROFILE ADMIN_PROFILE LIMIT
    SESSIONS_PER_USER      1
    FAILED_LOGIN_ATTEMPTS  3
    PASSWORD_VERIFY_FUNCTION ORA12C_STRONG_VERIFY_FUNCTION
    PASSWORD_LIFE_TIME     60;


CREATE USER GUEST
    IDENTIFIED by guest_password
    PROFILE GUEST_PROFILE
    DEFAULT TABLESPACE GS_DATA
    TEMPORARY TABLESPACE GS_TEMP
    ACCOUNT UNLOCK;


CREATE USER USER_APP
    IDENTIFIED BY user_password
    PROFILE USER_PROFILE
    DEFAULT TABLESPACE GS_DATA
    TEMPORARY TABLESPACE GS_TEMP
    ACCOUNT UNLOCK;


CREATE USER DEVELOPER
```

```
    IDENTIFIED BY TEMP_password1
    DEFAULT TABLESPACE GS_DATA
    TEMPORARY TABLESPACE GS_TEMP
    PROFILE DEVELOPER_PROFILE
    ACCOUNT UNLOCK;

CREATE USER ADMIN
    IDENTIFIED BY TEMP_ADM_password12
    DEFAULT TABLESPACE GS_DATA
    TEMPORARY TABLESPACE GS_TEMP
    PROFILE ADMIN_PROFILE
    ACCOUNT UNLOCK;
```

Листинг 3.2 – Создание профилей и пользователей

Таже для пользователь были разработаны соответствующие им роли, чтоб наложить ограничения, но возможность использования объектов БД. Скипты создания ролей представлены в листинге 3.3.

```
     CREATE ROLE ROLE_GUEST;

     CREATE ROLE ROLE_USER;

     CREATE ROLE ROLE_DEVELOPER;

     CREATE ROLE ROLE_ADMIN;


     GRANT CREATE SYNONYM TO ROLE_GUEST;
     GRANT CREATE SYNONYM TO ROLE_USER;

     GRANT CREATE SESSION TO ROLE_ADMIN;
     GRANT ALTER USER TO ROLE_ADMIN;
     GRANT CREATE USER TO ROLE_ADMIN;
     GRANT DROP USER TO ROLE_ADMIN;
     GRANT GRANT ANY PRIVILEGE TO ROLE_ADMIN;
     GRANT GRANT ANY ROLE TO ROLE_ADMIN;
     GRANT CREATE ANY TABLE TO ROLE_ADMIN;
     GRANT DROP ANY TABLE TO ROLE_ADMIN;
     GRANT CREATE ANY PROCEDURE TO ROLE_ADMIN;
     GRANT ALTER ANY PROCEDURE TO ROLE_ADMIN;
     GRANT DROP ANY PROCEDURE TO ROLE_ADMIN;
     GRANT UNLIMITED TABLESPACE TO ROLE_ADMIN;
```

Листинг 3.3 – Создание ролей

3.3 Вывод

Проект базы данных игровой платформы включает создание нескольких табличных пространств, предназначенных для хранения информации о пользователях, продуктах, транзакциях и других сущностях системы. Каждое табличное пространство имеет собственный файл для хранения данных, что позволяет оптимизировать управление ресурсами и повысить производительность базы данных. Для разработчика была выделена отдельная квота на добавление данных в данные пространства, что обеспечивает контроль использования дискового пространства.

В рамках создания базы данных также были определены роли и пользователи. Основные пользователи — гость, авторизованный пользователь, разработчик и администратор — обладают различными ролями, набором привилегий и выполняют собственные задачи в системе. Такое разделение прав доступа повышает уровень безопасности и снижает риск несанкционированных изменений данных.

В итоге была создана безопасная и эффективная основа для базы данных, обеспечивающая целостность, защищённость и масштабируемость системы. Реализованная модель доступа позволяет гибко управлять правами пользователей и упрощает дальнейшее развитие и сопровождение игровой платформы.

4 Разработка объектов базы данных

4.1 Создание таблиц

На этапе планирования было определённо, что для корректного функционирования проекта, на том уровне на котором он запланирован, необходимо создать 19 таблиц, для хранения как основной информации, так и вспомогательной.

Подробнее необходимые таблицы и их структура была описана в разделе 2.2.

Ниже представлен пример создания таблицы пользователей в листинге 4.1

```
CREATE TABLE Users (
    UserID           NUMBER GENERATED ALWAYS AS IDENTITY,
    Email            NVARCHAR2(255) NOT NULL UNIQUE,
    NickName         NVARCHAR2(255)  NOT NULL UNIQUE,
    PasswordHash     NVARCHAR2(255) NOT NULL,
    RoleID           NUMBER        NOT NULL,
    CreatedAt             TIMESTAMP  WITH  TIME  ZONE     DEFAULT
CURRENT_DATE NOT NULL,
    LastLogIn        TIMESTAMP WITH TIME ZONE,
    IsActive         NUMBER(1) DEFAULT 1,
    Avatar_uri       NVARCHAR2(255),
    Country          NVARCHAR2(255),
    Balance NUMBER(10,2) DEFAULT 0,


    CONSTRAINT PK_User PRIMARY KEY(UserID)
        USING INDEX TABLESPACE GS_INDEX,

    CONSTRAINT FK_RoleID FOREIGN KEY (RoleID)
        REFERENCES Roles(RoleId)
)
TABLESPACE GS_DATA ;
```

Листинг 4.1 – Создание таблицы пользователей

Все скрипты создания таблиц опредлены в рамках файла “02_create_tables.sql”, и должны исполняться от лица пользователя “app_user” - это пользователь, которому принадлежат все объекты базы данных направленные на реализацию бизнес-логики, такой подход был вобран введу большого количества таблиц, и в целях упростить дальнейшую разработку. Далее все объекты БД будут создаваться именно от его имени.

4.2 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.

Эти особенности использовались в рамках разработки БД в моментах, когда было очень трудно в рамках входных данных процедур добраться до необходимых, из-за чего код становился мало читаемым, в такие моменты создавались представления, ниже приведен пример скрипта создания представления которое используется в процедурах связанных с предложениями (offers) чтобы объединить жанры всех доступных в предложении игр.

```
CREATE OR REPLACE VIEW OfferGamesWithGenres AS
SELECT
    o.OfferId,
    gp.PageID,
    gp.PageTittle,
    gp.DeveloperId,
    gp.ViewLink,
    g.GameID,
    g.GameName,
    g.DownloadLink,
    g.type AS GameType,
    gg.Ganer_ID,
    o.Price
FROM Offers o
JOIN GamePages gp ON gp.PageID = o.PageID
JOIN OfferGameLinks ogl ON ogl.OfferId = o.OfferId
JOIN Games g ON g.GameID = ogl.GameID
JOIN Games_ganers gg ON gg.GameID = g.GameID;
```

Листинг 4.2 – Создание представления

4.3 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.

Решение использовать функции пришло в момент, когда код процедур становился слишком сложным, и нарушался принцип единной ответственности, в такие моменты часть составного функционала переносилась либо в другую процедуру, либо функцию.

Ниже представлен пример создания функции преобразующая жанры конкретной игры в json объект.

```
CREATE OR REPLACE FUNCTION get_game_genres_json(p_game_id
NUMBER)
RETURN CLOB IS
    v_clob CLOB := '[]';
BEGIN
    SELECT JSON_ARRAYAGG(
                JSON_OBJECT(
                    'genre_id' VALUE g.genreId,
                    'genre' VALUE g.genre
                )
            )
    INTO v_clob
    FROM Games_ganers gg
    JOIN Geners g ON gg.Ganer_ID = g.genreId
    WHERE gg.GameID = p_game_id;

    RETURN NVL(v_clob, '[]');
END get_game_genres_json;
```

Листинг 4.3 – Создание функции

4.4 Создание пакетов процедур

Для управления данными пользователи и администраторы будут использовать хранимые процедур. 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: enums_pkg – пакет, который используется, когда нужно обратиться к таблицам, которые хранят типы объектов, guest_pkg – пакет, с процедурами для гостя, user_pkg – пакет, выдаваемый в использование авторизованным пользователям, developer_pkg – пакет, хранящий процедуры для разработчиков, admin_pkg – пакет,

храняший процедуры для адменистратора, stat_pkg – пакет, с процедурами для аналитических отчетов. Каждый из пакетов группирует в себе процедуры, выполняющие действия опредленной направленности.

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

```
CREATE OR REPLACE PACKAGE guest_pkg IS

    PROCEDURE get_game_pages_filtered(
        p_developer_id IN NUMBER,
        p_title_search IN NVARCHAR2,
        p_genre_id     IN NUMBER,
        p_order_by    IN NVARCHAR2,  -- 'TITLE' или 'PRICE'
        p_order_dir    IN NVARCHAR2,  -- 'ASC' или 'DESC'
        p_response     OUT CLOB
    );


    PROCEDURE TryDownload(
        p_offer_id  IN NUMBER,
        p_response  OUT CLOB
    );

    PROCEDURE register_guest(
        p_username IN NVARCHAR2,
        p_password IN NVARCHAR2,
        p_email    IN NVARCHAR2,
        p_response OUT CLOB
    );

    PROCEDURE login_guest(
        p_email    IN NVARCHAR2,
        p_password IN NVARCHAR2,
        p_response OUT CLOB
    );

END guest_pkg;
```

Листинг 4.4 – Создание пакетов

4.4 Создание синоноимов

Синоним (synonym) — это объект базы данных Oracle, представляющий собой альтернативное имя для другого объекта (таблицы, представления, последовательности, пакета, процедуры и т. д.). Синонимы используются для упрощения доступа к объектам, скрытия их реального имени и схемы, а также

для обеспечения гибкости при изменении структуры базы данных, позволяя клиентскому коду обращаться к объекту через единое логическое имя.

Так как все объекты создаются от одного пользователя, и разрешение на них предоставляется другим, синонимы тут очень хорошо подходят чтоб упростить запись вызова объекта, нет необходимости обрушатся к схеме объекта. Пример создания синонимов представлен в листинге 4.5.

```
-- синоним от соединения гостя
CREATE or REPLACE SYNONYM guest_pkg FOR app_user.guest_pkg;

-- синоним от соединения пользоватлея
CREATE or REPLACE SYNONYM guest_pkg FOR app_user.user_pkg;

-- синоним от соединения пользоватлея
CREATE or REPLACE SYNONYM guest_pkg FOR app_user.developer_pkg;

-- синоним от соединения администратора
CREATE or REPLACE SYNONYM guest_pkg FOR app_user.admin_pkg;
```

Листинг 4.5 – Создание синонимов

4.5 Вывод

В итоге выполнения данного этапа были разработаны самые таблицы по разработанной схеме в предыдущем разделе, и необходимые процедуры для взаимодействия пользователей с базой данных, а также были разработаны вспомогательные функции, упрощающие работу с базой данных, а так же синонимы для пользователей, что украшает работу с пакетами разработанными другим пользователем. По итогам выполнения мы получили полностью рабочую базу данных с гибкой архитектурой.

5 Описание процедур экпорта и импорта данных

5.1 Процедура импорта данных из JSON-файла

В файле import_useractivity_from_json.sql определен анонимный блок предназначеный для импорта данных из json файла в таблицу UserActivity. В этом блоке реализованы функции чтения парсера, валидации и записи данных в таблицу.

Процедура открывает файл USERACTIVITY_EXPORT.json в указанной дериктории JSON_EXPORT_DIR используя пакет UTL_FILE.

Чтение содержимого: Весь текст файла последовательно считывается и конкатенируется в переменную типа CLOB.

Парсинг JSON: с помощью оператора JSON_TABLE содержимое JSON-массива преобразуется в реляционный формат. Для каждого элемента массива извлекаются соответствующие поля.

Валидация и вставка: Извлеченные данные проходят проверку адекватности данных, при помаши регулярных выражения, а также проверяется ссылочная целостность, если произойдет исключение целостности данных, строка с этими данными будет пропущена, и процедура продолжит пытаться вставить остальные данные.

Завершение транзакции: каждые 100 строк происходит фиксация изменений, а также в конце записи финальный коммит.

Листинг содержимого файла import_useractivity_from_json.sql представлен в листинге.

5.2 Процедура экспорта данных в JSON-файла

Процедура экспорта определена в файле export_useractivity_to_json.sql, там реализован анонимный блок, который сохранит данные из таблицы useractivity в Json файл, и удалит данные из таблицы. Такой метод можно использовать как для передачи данных, так и для резервных копий.

Происходит формирование JSON вручную: в связи с требованиями к специфическому формату вывода, JSON формируется путем прямого построения строки. Процедура открывает файл на запись и начинает формировать JSON-массив.

Далее, итерация по записям: данные из таблицы useractivity выбираются в курсор. Для каждой записи формируется отдельный JSON-объект, поля которого соответствуют структуре таблицы. Значения дат и временных меток предварительно конвертируются в строковый формат ISO 8601 (YYYY-MM-DD"T"HH24:MI:SS) для обеспечения совместимости.

Обработка NULL-значений: Для числового поля IpAddress , UserAgent, Details используется функция NVL для корректной подстановки значения null в JSON, если в базе данных хранится NULL.

Синтаксическое оформление: Процедура обеспечивает правильную расстановку запятых между объектами в массиве и корректное закрытие JSON-структуры.

Содержимое файла представлено в листинге.

В результате выполнения в каталоге JSON_EXPORT_DIR будет создан файл USERACTIVITY_EXPORT.json, содержащий полный дамп таблицы заказов в формате JSON.

5.3 Вывод по разделу

В рамках данного раздела были успешно разработаны и реализованы два ключевых механизма для обеспечения мобильности данных системы управления компьютерным клубом: процедура импорта import_useractivity_from_json и процедура экспорта export_useractivity_to_json.

Реализованные процедуры демонстрируют эффективное использование возможностей СУБД Oracle для работы с форматом JSON и файловой системой сервера. Подход, основанный на хранимых процедурах, полностью удовлетворяет требованию задания о предоставлении доступа к данным только через соответствующие процедуры. Это обеспечивает несколько важных преимуществ: безопасность операций за счёт выполнения в контексте контролируемых прав доступа к каталогам и системным пакетам; целостность данных благодаря встроенной в процедуру импорта проверке внешних ключей, что предотвращает появление некорректных ссылок; а также надежность, поскольку обработка исключений гарантирует, что в случае сбоя система не останется в противоречивом состоянии, а все занятые ресурсы, такие как открытые файлы, будут корректно освобождены.

6 Тестирование производительности

6.1 Тестирование процедур

Для тестирования производительности базы данных необходимо располагать существенным набором данных, либо хотя бы их имитация. Для решения этой проблемы был разработал файл с sql запросами которые вставят базовые тестовые записи в таблицы, такие как, 5 разработчиков, 5 обычных пользователь, и каждому разработчику по 25 игр, к каждой игре одна страница, и на одно странице от 1 до 3 предложений в котором могут влачатся от 1 до 3 игр разработчика. После чего был разработан анонимный блок, который имитировал большую активность обычных пользователей скачиванием бесплатных игр. Анонимный блок выполнял итерацию 100 000 раз, выполняя запрос на получения ссылки скачивания случайной игры от лица случайного пользователя, вследствие чего, заполнялась таблица активностей пользователей.

После добавления данных таблица активностей пользователей получилась самой тяжеловесной, и процедуру которые создают отчеты на ее основе не самые простые, оперируя этими фактами было принято решения протестировать производительность именно с этими процедурами и таблицей.

Выполнение запроса по активности конкретного пользователя заняло 2 секунды, что достаточно много, но на это и был расчет с учетом плана использование технологии in-memory, для был найден пользоватлеь который совершил очень много автивностей, и было собрана вся его статистика, ниже приведены скриншоты времени выполнения процедуры, и анализ запроса при момоши модуля DBMS_XPLAN.DISPLAY_CURSOR.

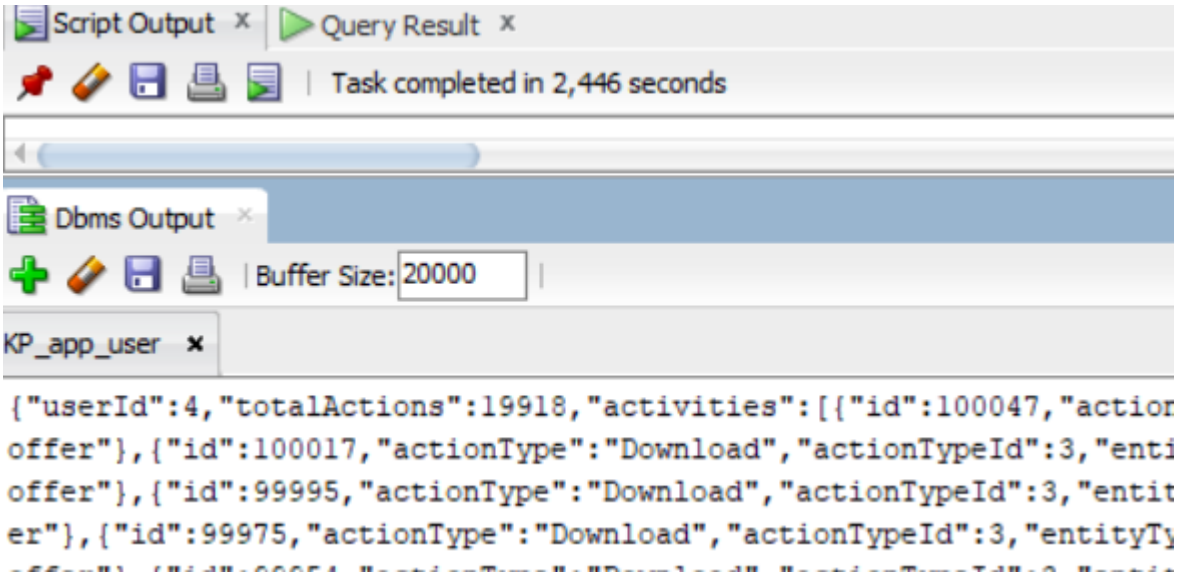
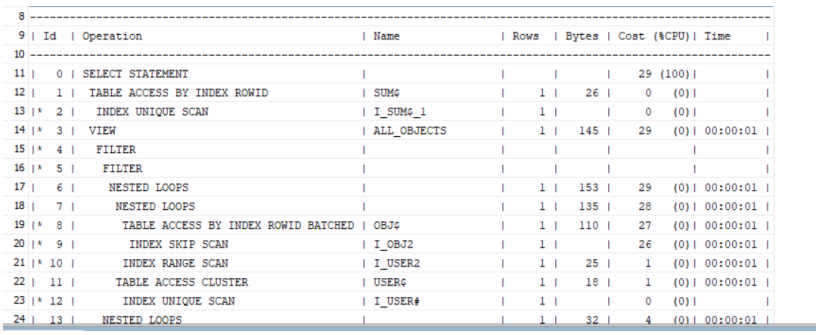


Рисунок 6.1 – Результат работы процеудуры

Ниже представлен рисунок демонстрирующий анализ плана выполнения запроса.



Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time
0	SELECT STATEMENT				29 (100)	
1	TABLE ACCESS BY INDEX ROWID	SUM$	1	26	0 (0)	
* 2	INDEX UNIQUE SCAN	I_SUM$_1	1		0 (0)	
* 3	VIEW	ALL_OBJECTS	1	145	29 (0)	00:00:01
* 4	FILTER					
* 5	FILTER					
6	NESTED LOOPS		1	153	29 (0)	00:00:01
7	NESTED LOOPS		1	135	28 (0)	00:00:01
* 8	TABLE ACCESS BY INDEX ROWID BATCHED	OBJ$	1	110	27 (0)	00:00:01
* 9	INDEX SKIP SCAN	I_OBJ2	1		26 (0)	00:00:01
* 10	INDEX RANGE SCAN	I_USER2	1	25	1 (0)	00:00:01
11	TABLE ACCESS CLUSTER	USER$	1	18	1 (0)	00:00:01
* 12	INDEX UNIQUE SCAN	I_USER#	1		0 (0)	
13	NESTED LOOPS		1	32	4 (0)	00:00:01

Рисунок 6.2 – Анализ выполнения процедуры

Из положительного можно заметить, что большинство доступов к данным происходят через индексы, а если не через индексы через представления, а также ценна операций достаточно низкая.

6.2 Вывод по разделу

Проведённое тестирование производительности базы данных в СУБД Oracle позволило получить объективные данные о её эффективности при работе с большими объёмами информации. Разработанная процедура генерации тестовых данных успешно создала 100 000 записей.

Ключевым этапом тестирования стал анализ планов выполнения запросов на репрезентативной выборке данных. Как показало исследование типичного запроса на анализ статистики, система демонстрирует высокую производительность благодаря грамотно спроектированной индексации. Оптимизатор запросов корректно выбирает стратегию доступа по индексу.

Полученные результаты свидетельствуют о том, что реляционная реализация базы данных на платформе Oracle полностью соответствует требованиям производительности. Система эффективно масштабируется до значительных объёмов данных, поддерживает быстрый отклик на оперативные запросы и обеспечивает стабильную работу при выполнении как пакетных операций (импорт/экспорт, генерация данных), так и транзакционных запросов пользователей.

Однако основана оптимизация должна реализовываться технологией in-memory, что должно еще значительнее ускорить работу базы данных.

7 Описание технологии и ее применение в базе данных

7.1 Использование технологии In-Memmory

В рамках данного курсового проекта необходимо изучить и использовать технологию In-Memmory. Суть этой технологии заключается в том, что позволяет ускорять аналитические и транзакционные запросы за счёт хранения данных в оперативной памяти в формате, оптимизированном для чтения.

Основная идея: обычно Oracle хранит данные на диске и кеширует их в буферах памяти (SGA). In-Memory позволяет хранить таблицы или их части полностью в оперативной памяти, используя столбцатый формат (columnar format) для быстрых аналитических вычислений.

Перед использованием технологии следует проверить, поддерживает ли база данных эту технологию, так как Oracle требует специальную лицензию для использования in-memmory, сделать это можно при помаши скрипта, приведенного в листинге 7.1

```
SELECT parameter, value
FROM v$option
WHERE parameter = 'In-Memory Column Store';
```

Листинг 7.1 – скрипт для проверки лицензии in-memmory

В результате селект запроса получены следующие данные.

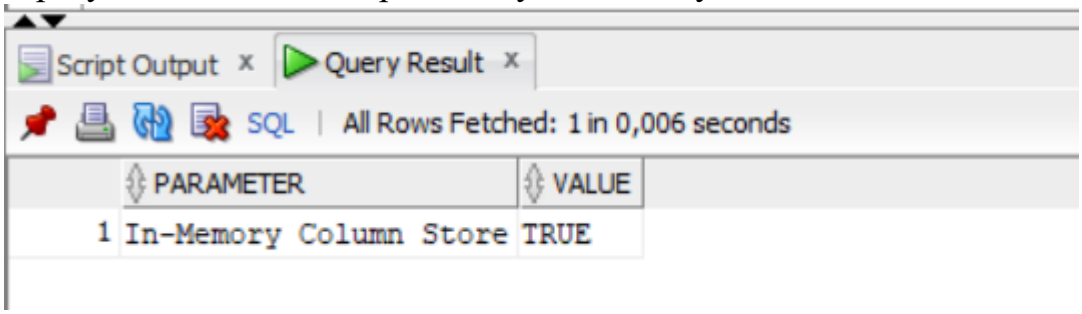


Рисунок 7.2 – результат селект запроса

В результате селект запроса можно сделать вывод, что базза данных полностью поддерживает технологию in-Memmory.

Зная это можно приступать к выделению памяти для in_memmory, но стоит произвести еще пару проверок, так как без них неправильная настройка in-memmory, может привести к тому что база данных перестанет запускаться. Основана проблема заключается в том, что при старте инстанса, память выделяется для SGA и Inmemmory, и если им в сумме не достаточно памяти или она ограничена max_target_size, инстанс не сомжет запустится.

Ниже приведён sql скрипт который может проверить доступную память, по нему можно проверь можно сейчас выделять память для in-memmory или нет.

```
SELECT name, value FROM v$parameter
WHERE name IN ('sga_target','sga_max_size','inmemory_size');
```

Листинг 7.3 – скрипт для проверки выделяемой памяти для инстанса

Результат работы скрипта представлен в рисунке 7.4

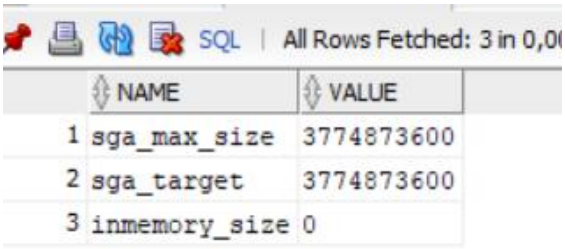


Рисунок 7.4 – результат селект запроса

В результате запроса можено сделать вывод, что сейчас выделять память под in-memmory, нельзя, иначе при перезапуска база данных не запустится, в сумме sga_target и inmemory_size должны быть не больше чем sga_max_size. В данном случаае есть два вариант либо увеличить sga_max_size, либо уменьшить sga_target.

Ниже приведены скрит который изменить размеры выделяемы для sga.

```
ALTER SYSTEM SET SGA_MAX_SIZE = 6G SCOPE=SPFILE;
ALTER SYSTEM SET SGA_TARGET = 3G SCOPE=SPFILE;
```

Листинг 7.5 – скрипт для изменения выделяемой памяти

После выполнения скрипта необходимо перезапустить базу данных, результат перезапуска базы данных представлен на рисунке 7.6

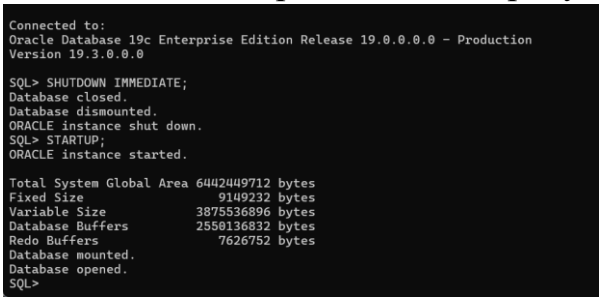


Рисунок 7.6 – результат перезапуска БД

В процессе перезапуска можно заметить, что выделяемая память действительно изменилась, теперь база данных готова к выделению памяти под in-memmory. Скрипт выделения памяти для in-memory предвставлен в листинге 7.7

```
ALTER SYSTEM SET INMEMORY_SIZE = 2560M SCOPE=SPFILE;
```

Листинг 7.7 – скрипт для выделения памяти

Данный скрипт выделит для in-memory, память в размере 2,5 гб. После успешного выполнения скрипта базу данных снова нужно перезапустить. Результат перезапуска представлен на рисунке 7.7.

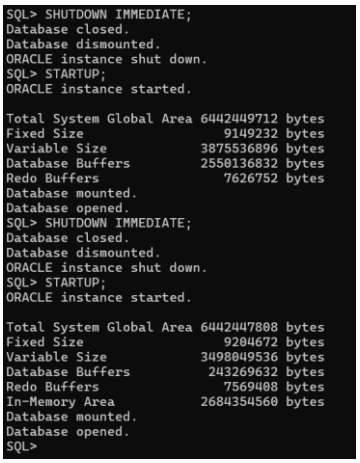


Рисунок 7.8 – Результат перезагурзки

В результате перезагрузки можно обратить внимание что теперь действительно память выделяется для in-memory.

После перезагрузки БД необходимо указать какие таблицы поместить в in-memory, их приоритет и какие коленки, самое главная таблица, которая должна находиться там это USERACTIVITY, так как она взаимодействует во всех процедурах, связанных со статистикой, и в ней хранится очень много данных, а так же нужно поместить таблицы но с меньшими преорететатами. Скрипт настройки таблицы USERACTIVITY пердставлен в листинге 7.9

```
-- UserActivity - самая важная таблица для аналитики
ALTER TABLE UserActivity INMEMORY
MEMCOMPRESS FOR QUERY HIGH
PRIORITY HIGH;
```

Рисунок 7.9 – Скрипт настройки таблицы

Скрипты настройки остальных таблиц находятся в приложениях.

7.2 Сравнительный анализ производительности

После загрузки таблиц в ОЗУ, была проведена проверка что таблица действительно находится там, делалось это при помаши sql запроса, результат его находится в рисунке 7.10.

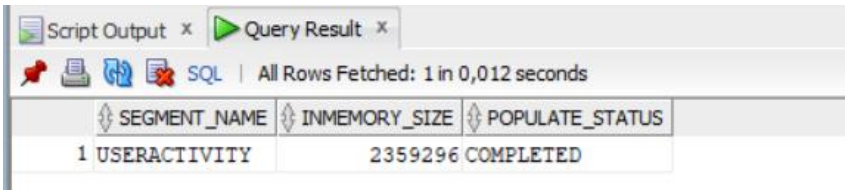


Рисунок 7.10 – результат работы запроса

Теперь мы точно знаем, что таблица UsersActivity находится в ОЗУ, теперь мы можем провести повторные замеры просизвоотельности той же самой процедуры с теми же самыми входными данными. Новый результат работы процедуры представлен на рисунке 7.11.

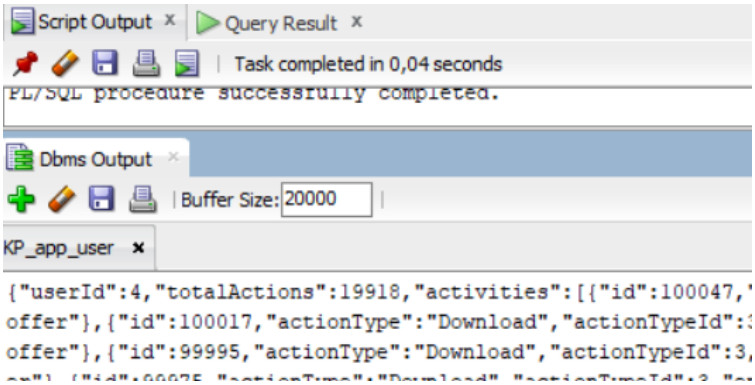


Рисунок 7.10 – результат работы процедуры

Можем заметить, что разница во времени выполнения действительно существенная, что говорит о том, что in-memory действительно ускоряет запросы и весьма существенно.

7.3 Вывод по разделу

В ходе работы была исследована технология in-memory, предназначенная для хранения данных в оперативной памяти хоста. Данная технология была применена для оптимизации аналитических и SELECT-запросов в базе данных. В процессе исследования были выявлены как сильные, так и слабые стороны данной технологии. Одним из её существенных недостатков является риск некорректного включения без предварительных проверок, что может привести к невозможности запуска базы данных. В таком

случае восстановление работоспособности требует прямого редактирования файла spfile, что при недостаточной осторожности может повлечь дополнительные негативные последствия.

Также под сомнение ставится целесообразность использования данной технологии в рассматриваемом проекте, поскольку в СУБД Oracle уже реализовано множество встроенных механизмов оптимизации запросов, работающих в фоновом режиме. В результате прирост производительности не всегда оказывается существенным. Кроме того, технологии кэширования и хранения данных в оперативной памяти часто реализуются на уровне серверной части приложения, что снижает необходимость применения in-memory на уровне базы данных.

Таким образом, использование технологии in-memory требует тщательного анализа архитектуры системы, нагрузки и доступных ресурсов. Её применение оправдано преимущественно в системах с высокой долей сложных аналитических запросов и строгими требованиями к производительности, тогда как в иных случаях предпочтительнее использование стандартных средств оптимизации базы данных.

Заключение

В ходе выполнения курсового проекта были достигнуты все поставленные цели и задачи. В результате была спроектирована и реализована полностью функционирующая база данных игровой платформы, обладающая высокой производительностью, гибкой и масштабируемой архитектурой. Структура базы данных разрабатывалась с учётом современных требований к проектированию информационных систем, что позволило обеспечить целостность данных, удобство расширения и сопровождения, а также высокий уровень надёжности.

В процессе разработки были реализованы основные принципы структуризации проекта и обеспечения безопасности данных, включая разграничение доступа, использование ролей и контроль целостности. Особое внимание было уделено оптимизации работы базы данных: изучена и внедрена технология In-Memory, которая была использована для ускорения выполнения как аналитических, так и транзакционных запросов. Применение данной технологии позволило значительно сократить время обработки данных и повысить общую эффективность системы.

Практическая значимость данной работы заключается в формировании обоснованных рекомендаций по выбору типа системы управления базами данных в зависимости от требований проекта, а также в приобретении практических навыков проектирования, развертывания и оптимизации баз данных. Полученные знания и опыт могут быть использованы при разработке реальных информационных систем и послужить основой для дальнейшего углублённого изучения технологий управления данными.

Список используемых источников

1. Oracle Database Online Documentation 23c [Электронный ресурс] – Режим доступа:https://docs.oracle.com/en/database/oracle/oracle-database/23/index.html – Дата обращения: 19.09.2025.
2. Документация по in-memmoy в oralce [Электронный ресурс] – режим доступа:https://www.oracle.com/technetwork/ru/database/options/database-in-memory-ds-2210927-ru.pdf
3. Документация по JSON в Oracle Database [Электронный ресурс] – Режим доступа:https://docs.oracle.com/en/database/oracle/oracle-database/23/adjsn – Дата обращения: 19.09.2025.
4. Управление доступом в Oracle Database [Электронный ресурс] – Режим доступа:https://docs.oracle.com/en/database/oracle/oracle-database/23/dbseg – Дата обращения: 19.09.2025.
5. IEEE Transactions on Knowledge and Data Engineering [Электронный ресурс] – Режим доступа: https://ieeexplore.ieee.org – Дата обращения: 19.09.2025.

ПРИЛОЖЕНИЕ А. Use Case диаграмма

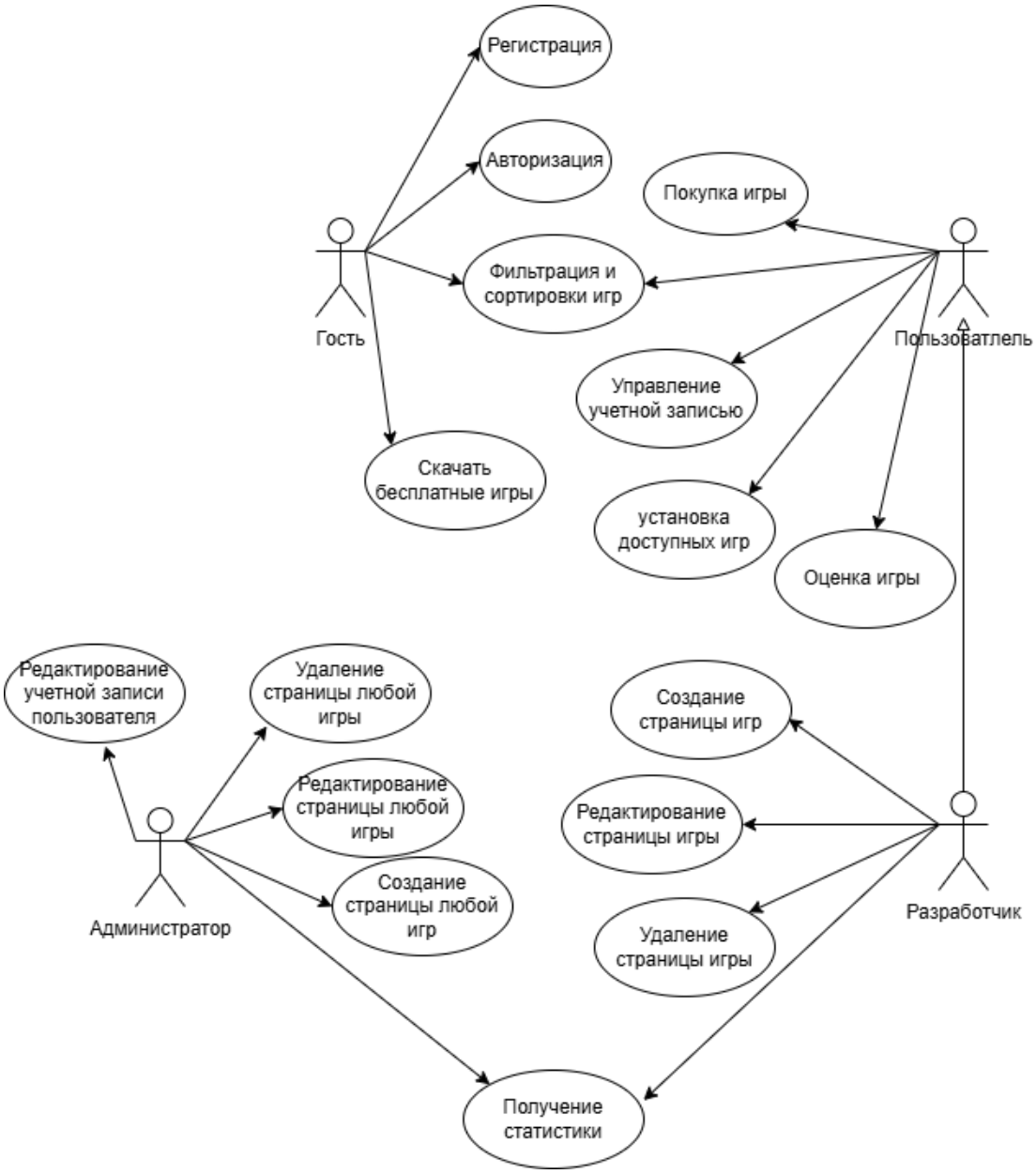


Рисунок 1.1 – Диаграмма вариантов использования

ПРИЛОЖЕНИЕ Б. Структура базы данных.

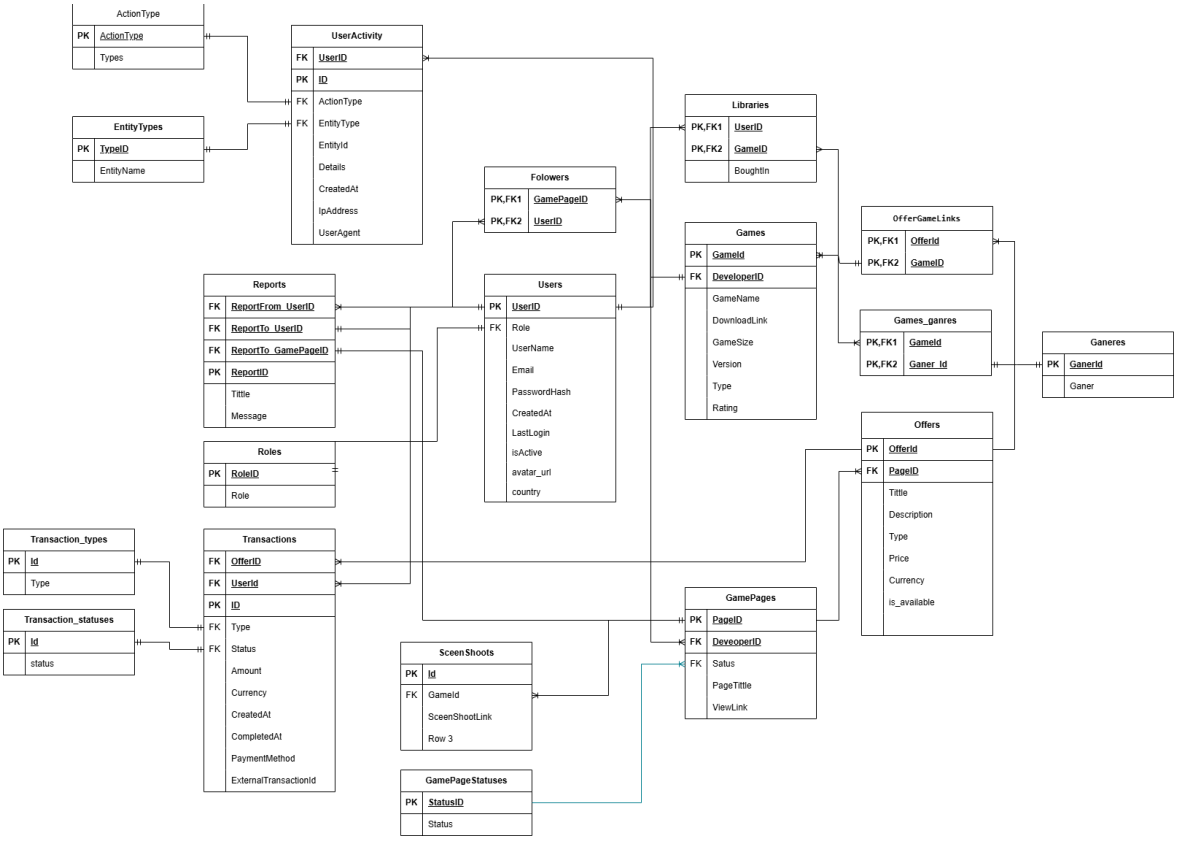


Рисунок 1.2 – Диаграмма базы данных

ПРИЛОЖЕНИЕ В. Процедура импорта.

```
DECLARE
    v_file_handle UTL_FILE.FILE_TYPE;
    v_json_content CLOB;
    v_line VARCHAR2(32767);
    v_json_data JSON_ARRAY_T;
    v_json_obj JSON_OBJECT_T;
    v_count NUMBER := 0;
    v_errors NUMBER := 0;

BEGIN
    -- Читаем JSON файл
    v_file_handle := UTL_FILE.FOPEN('JSON_EXPORT_DIR',
'USERACTIVITY_EXPORT.json', 'R', 32767);

    -- Читаем весь файл в CLOB
    BEGIN
        LOOP
            BEGIN
                UTL_FILE.GET_LINE(v_file_handle, v_line);
                v_json_content := v_json_content || v_line ||
CHR(10);
            EXCEPTION
                WHEN NO_DATA_FOUND THEN
                    EXIT;
            END;
        END LOOP;
    EXCEPTION
        WHEN NO_DATA_FOUND THEN
            NULL;
    END;

    UTL_FILE.FCLOSE(v_file_handle);

    -- Парсим JSON
    v_json_data := JSON_ARRAY_T(v_json_content);

    DBMS_OUTPUT.PUT_LINE('Найдено записей в JSON: ' ||
v_json_data.get_size());

    -- Обрабатываем каждую запись
    FOR i IN 0 .. v_json_data.get_size() - 1 LOOP
        DECLARE
```

```
            v_user_id NUMBER;
            v_action_type NUMBER;
            v_entity_type NUMBER;
            v_entity_id NUMBER;
            v_details NVARCHAR2(255);
            v_created_at TIMESTAMP WITH TIME ZONE;
            v_ip_address VARCHAR2(45);
            v_user_agent VARCHAR(512);
            v_created_at_str VARCHAR2(100);
        BEGIN
            v_json_obj := JSON_OBJECT_T(v_json_data.get(i));

            -- Извлекаем значения в переменные
            v_user_id := v_json_obj.get_number('UserID');
            v_action_type :=
v_json_obj.get_number('ActionType');
            v_entity_type :=
v_json_obj.get_number('EntityType');
            v_entity_id := v_json_obj.get_number('EntityID');

            -- Обрабатываем строковые поля с проверкой на NULL
            IF v_json_obj.get('Details').is_null THEN
                v_details := NULL;
            ELSE
                v_details := v_json_obj.get_string('Details');
            END IF;

            IF v_json_obj.get('CreatedAt').is_null THEN
                v_created_at := NULL;
            ELSE
                v_created_at_str :=
v_json_obj.get_string('CreatedAt');
                v_created_at_str :=
REGEXP_REPLACE(REGEXP_REPLACE(v_created_at_str, 'T', ' '), 'Z$',
'+00:00');
                v_created_at :=
TO_TIMESTAMP_TZ(v_created_at_str, 'YYYY-MM-DD HH24:MI:SS.FF
TZH:TZM');
            END IF;

            IF v_json_obj.get('IpAddress').is_null THEN
                v_ip_address := NULL;
            ELSE
                v_ip_address :=
v_json_obj.get_string('IpAddress');
            END IF;
```

```
            IF v_json_obj.get('UserAgent').is_null THEN
                v_user_agent := NULL;
            ELSE
                v_user_agent :=
v_json_obj.get_string('UserAgent');
            END IF;

            -- Вставляем данные в таблицу
            -- ВНИМАНИЕ: ID будет сгенерирован автоматически,
так как используется IDENTITY
            INSERT INTO UserActivity (
                UserID,
                ActionType,
                EntityType,
                EntityID,
                Details,
                CreatedAt,
                IpAddress,
                UserAgent
            ) VALUES (
                v_user_id,
                v_action_type,
                v_entity_type,
                v_entity_id,
                v_details,
                v_created_at,
                v_ip_address,
                v_user_agent
            );

            v_count := v_count + 1;

            -- Коммитим каждые 100 записей для
производительности
            IF MOD(v_count, 100) = 0 THEN
                COMMIT;
            END IF;

        EXCEPTION
            WHEN OTHERS THEN
                v_errors := v_errors + 1;
                DBMS_OUTPUT.PUT_LINE('Ошибка при импорте записи
' || i || ': ' || SQLERRM);
                -- Продолжаем обработку остальных записей
        END;
```

```
    END LOOP;

    -- Финальный коммит
    COMMIT;

    DBMS_OUTPUT.PUT_LINE('Импортировано успешно: ' || v_count ||
' записей');
    IF v_errors > 0 THEN
        DBMS_OUTPUT.PUT_LINE('Ошибок при импорте: ' ||
v_errors);
    END IF;

EXCEPTION
    WHEN UTL_FILE.INVALID_PATH THEN
        DBMS_OUTPUT.PUT_LINE('ОШИБКА: Файл
USERACTIVITY_EXPORT.json не найден в директории
JSON_EXPORT_DIR');
        DBMS_OUTPUT.PUT_LINE('Скопируйте файл в Docker
контейнер:');
        DBMS_OUTPUT.PUT_LINE('docker cp USERACTIVITY_EXPORT.json
<container>:/opt/oracle/json_export/');
        IF UTL_FILE.IS_OPEN(v_file_handle) THEN
            UTL_FILE.FCLOSE(v_file_handle);
        END IF;
    WHEN UTL_FILE.INVALID_OPERATION THEN
        DBMS_OUTPUT.PUT_LINE('ОШИБКА: Невозможно прочитать файл.
Проверьте права доступа.');
        IF UTL_FILE.IS_OPEN(v_file_handle) THEN
            UTL_FILE.FCLOSE(v_file_handle);
        END IF;
    WHEN OTHERS THEN
        DBMS_OUTPUT.PUT_LINE('ОШИБКА: ' || SQLERRM);
        IF UTL_FILE.IS_OPEN(v_file_handle) THEN
            UTL_FILE.FCLOSE(v_file_handle);
        END IF;
        ROLLBACK;
        RAISE;
END;
```

ПРИЛОЖЕНИЕ Г. Процедура экспорта.

```
DECLARE

    v_file_handle UTL_FILE.FILE_TYPE;

    v_json_data CLOB;

    v_file_path VARCHAR2(255) := 'USERACTIVITY_EXPORT.json'; --
Путь к файлу

    v_row_count NUMBER := 0;

    v_first_row BOOLEAN := TRUE;

    CURSOR c_user_activity IS

        SELECT

            ID,

            UserID,

            ActionType,

            EntityType,

            EntityID,

            Details,

            CreatedAt,

            IpAddress,

            UserAgent

        FROM UserActivity

        ORDER BY ID;

BEGIN

    -- Файл будет создан в /opt/oracle/json_export/

    v_file_handle := UTL_FILE.FOPEN('JSON_EXPORT_DIR',
'USERACTIVITY_EXPORT.json', 'W', 32767);
```

```
    UTL_FILE.PUT_LINE(v_file_handle, '[');
    FOR rec IN c_user_activity LOOP
        IF NOT v_first_row THEN
            UTL_FILE.PUT_LINE(v_file_handle, ',');
        END IF;
        v_first_row := FALSE;

        v_json_data := '{' ||
            '"ID":' || rec.ID || ',' ||
            '"UserID":' || rec.UserID || ',' ||
            '"ActionType":' || rec.ActionType || ',' ||
            '"EntityType":' || rec.EntityType || ',' ||
            '"EntityID":' || rec.EntityID || ',' ||
            '"Details":' ||
                CASE
                    WHEN rec.Details IS NULL THEN 'null'
                    ELSE '"' ||
REPLACE(REPLACE(REPLACE(rec.Details, '\', '\\'), '"', '\"'),
CHR(10), '\n') || '"'
                END || ',' ||
            '"CreatedAt":' ||
                CASE
                    WHEN rec.CreatedAt IS NULL THEN 'null'
                    ELSE '"' || TO_CHAR(rec.CreatedAt, 'YYYY-MM-
DD"T"HH24:MI:SS.FF3TZH:TZM') || '"'
                END || ',' ||
            '"IpAddress":' ||
                CASE
```

```
                    WHEN rec.IpAddress IS NULL THEN 'null'
                    ELSE '"' || REPLACE(REPLACE(rec.IpAddress,
'\', '\\'), '"', '\"') || '"'
                END || ',' ||
            '"UserAgent":' ||
                CASE
                    WHEN rec.UserAgent IS NULL THEN 'null'
                    ELSE '"' ||
REPLACE(REPLACE(REPLACE(rec.UserAgent, '\', '\\'), '"', '\"'),
CHR(10), '\n') || '"'
                END ||
            '}';


        UTL_FILE.PUT_LINE(v_file_handle, v_json_data);
        v_row_count := v_row_count + 1;
    END LOOP;
    UTL_FILE.PUT_LINE(v_file_handle, ']');
    UTL_FILE.FCLOSE(v_file_handle);
    DBMS_OUTPUT.PUT_LINE('Экспортировано строк: ' ||
v_row_count);
    IF v_row_count > 0 THEN
        DELETE FROM UserActivity;
        COMMIT;
        DBMS_OUTPUT.PUT_LINE('Данные успешно удалены из таблицы
UserActivity');
    ELSE
        DBMS_OUTPUT.PUT_LINE('Нет данных для экспорта');
    END IF;
```

```
    DBMS_OUTPUT.PUT_LINE('Экспорт завершен. Файл:
USERACTIVITY_EXPORT.json');

EXCEPTION

    WHEN UTL_FILE.INVALID_PATH THEN

        DBMS_OUTPUT.PUT_LINE('ОШИБКА: Неверный путь к файлу.
Убедитесь, что директория JSON_EXPORT_DIR существует.');

        DBMS_OUTPUT.PUT_LINE('Выполните:
@schema/setup_docker_directory.sql');

        DBMS_OUTPUT.PUT_LINE('И создайте директорию в Docker:
docker exec -it <container> mkdir -p /opt/oracle/json_export');

        IF UTL_FILE.IS_OPEN(v_file_handle) THEN

            UTL_FILE.FCLOSE(v_file_handle);

        END IF;

    WHEN UTL_FILE.INVALID_OPERATION THEN

        DBMS_OUTPUT.PUT_LINE('ОШИБКА: Невозможно выполнить
операцию с файлом. Проверьте права доступа.');

        IF UTL_FILE.IS_OPEN(v_file_handle) THEN

            UTL_FILE.FCLOSE(v_file_handle);

        END IF;

    WHEN OTHERS THEN

        DBMS_OUTPUT.PUT_LINE('ОШИБКА: ' || SQLERRM);

        IF UTL_FILE.IS_OPEN(v_file_handle) THEN

            UTL_FILE.FCLOSE(v_file_handle);

        END IF;

        RAISE;

END;
```