МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ  
Учреждение образования «БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ   
ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет информационных технологий

Кафедра информационных систем и технологий

Специальность 1-40 05 01 «Информационные системы и технологии»

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА КУРСОВОГО ПРОЕКТА**

по дисциплине «Базы данных»

Тема: «Реализация базы данных музыкальная площадка с использованием технологии мультимедийные типы данных и средств диагностики»

**Исполнитель**

студент 2 курса 3 группы \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ У. Э. Эльзарович

подпись, дата

**Руководитель**

ассистент \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Н. И. Уласевич

(должность, уч. звание) (подпись, дата)

Допущен(а) к защите \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_

дата, подпись

Курсовой проект защищен с оценкой

Руководитель \_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_ Н. И. Уласевич

подпись дата инициалы и фамилия

Минск 2022

**Содержание**

[Введение 4](#_Toc133712800)

[1 Анализ требований к программному средству 5](#_Toc133712801)

[1.1 Аналитический обзор аналогов 5](#_Toc133712802)

[1.1.1 Аналог Spotify 5](#_Toc133712803)

[1.1.2 Аналог ipmtorgi.by 6](#_Toc133712804)

[1.2 Разработка функциональных требований, определение вариантов использования 7](#_Toc133712805)

[1.3 Вывод 9](#_Toc133712807)

[2 Разработка архитектуры проекта 10](#_Toc133712808)

[2.1 Обобщенная структура управлением приложения 10](#_Toc133712809)

[2.2 Диаграммы UML, взаимосвязь всех компонентов. 10](#_Toc133712810)

[3 Разработка модели базы данных 13](#_Toc133712811)

[3.1 Создание необходимых объектов 13](#_Toc133712812)

[3.1.1 Представления базы данных 14](#_Toc133712813)

[3.1.2 Индексы базы данных 14](#_Toc133712814)

[3.1.3 Триггеры базы данных 15](#_Toc133712815)

[3.2 Описание используемой технологии 16](#_Toc133712816)

[3.3 Вывод 17](#_Toc133712817)

[4 Установка, настройка и использование Oracle 12c 19](#_Toc133712818)

[4.1 Установка Oracle 12c 19](#_Toc133712819)

[4.2 Создание таблиц 19](#_Toc133712820)

[4.3 Создание ролей для разграничения 20](#_Toc133712821)

[4.4 Создание пакетов процедур для базы данных 22](#_Toc133712822)

[4.4.1 Выборка данных из таблиц 23](#_Toc133712823)

[4.4.3 Заполнение таблиц 100 000 строк 25](#_Toc133712825)

[4.4.4 Добавление данных в таблицы 25](#_Toc133712826)

[4.4.5 Удаление данных в таблицы 27](#_Toc133712827)

[4.4.6 Изменение данных в таблицы 28](#_Toc133712828)

[4.4.7 Дополнительные функции 29](#_Toc133712829)

[4.5 Описание процедур экспорта и импорта 30](#_Toc133712830)

[4.6 Вывод 33](#_Toc133712831)

[5 Тестирование 34](#_Toc133712832)

[5.1 Тестирование производительности базы данных 34](#_Toc133712833)

[5.2 Вывод 35](#_Toc133712834)

[Заключение 42](#_Toc133712839)

[Список литературных источников 43](#_Toc133712840)

[Приложение А 44](#_Toc133712841)

# Введение

Музыка является древнейшим искусством. В таком искусстве есть только одно ограничение – фантазия композитора. С помощью музыки можно развеять тоску, задуматься над бытием или отношением к жизни, загрустить или просто отдохнуть. Именно поэтому музыку считают неотъемлемой частью нашей жизни.

Современную жизнь довольно тяжело представить без музыки. Она буквально повсюду, треки люди слушают в общественном транспорте или в торговых центрах, они ходят на концерты или создают свои собственные музыкальные коллекции. Такая популяризация музыки в жизни общества легко объяснима, ведь она дарит всем слушателям уникальную возможность снять стресс, избавиться от излишнего эмоционального напряжения. Некоторые психологи даже используют музыкальную терапию в своей практике, она помогает их клиентам бороться с депрессиями или апатиями.  
Сегодня каждый пользователь может [слушать музыку](https://muzonov.net/) прямо в режиме онлайн со специализированных музыкальных сайтов. Подобных ресурсов в сети интернет достаточно много, каждый из них предлагает свои уникальные условия для непосредственного взаимодействия с пользователями. Так, например, для работы с возможностями одних платформ обязательно нужна предварительная регистрация, а на других сайтах можно свободно работать и без нее. У каждого музыкального web ресурса, безусловно, имеются свои определенные достоинства и нюансы.  
Целью данной работы являлась разработка реляционной базы данных для музыкальной площадки. Эта база данных должна составлялась для обеспечения клиента доступом к музыкальным композициям, имеющимся на ней. Так же было необходимо разработать соответствующее приложение, для демонстрации её работы.

База данных — это упорядоченный набор структурированной информации или данных, которые обычно хранятся в электронном виде в компьютерной системе. Реляционная база данных – это набор данных с предопределенными связями между ними. Эти данные организованны в виде набора таблиц, состоящих из столбцов и строк. В качестве СУБД для базы данных была выбрана Oracle 12c, в связи с ее высокой производительностью и надежностью.

Целью данной курсовой работы является создание базы данных музыкальная площадка, которая будет эффективной и надежной. Работа направлена на улучшение процесса прослуивания композиций именна в нашей площадки.

Основными задачами является реализация следующих функций базы данных:

* Управление информацией о композициях.
* Управление информацией о исполнителях.
* Анализ данных о прослушиваемых композициях.
* Управление избранными композициях.

1. Анализ требований к программному средству

## Аналитический обзор аналогов

Музыка играет огромную роль в жизни человека. Одной из главных ее функций является объединять людей, поскольку язык музыки понятен без перевода. Значимость музыки заключается в ее способности обогащать жизнь человека и общества в целом.

Изобретение радио и кинематографа позволило перейти на следующий уровень инженерных разработок – телевидения, которое стало играть важную роль в продвижении музыки и популяризации артистов. В начале 1980-х гг. музыкальное телевидение затмило важность радио. А сегодня стриминговые сервисы стали еще популярнее и удобнее телевидения.

Одним из основных трендов музыкальных приложений остается курируемая музыка. Это так называемые «радиостанции», создающиеся с минимальных использованием компьютерных алгоритмов. В составлении плейлистов участвуют люди – музыкальные эксперты, знаменитости и т.д.

При разработке своего приложения я обратила внимания на самые успешные примеры популярнейших музыкальных площадок. Рассмотрим их ниже.

### 1.1.1 Аналог Spotify

Сегодня Spotify – крупнейший в мире музыкальный стриминговый сервис. Он располагает библиотекой c 35 миллионами песен и имеет 160 миллионов активных пользователей. Интерфейс этого сервиса можно увидеть на рисунке 1.1.

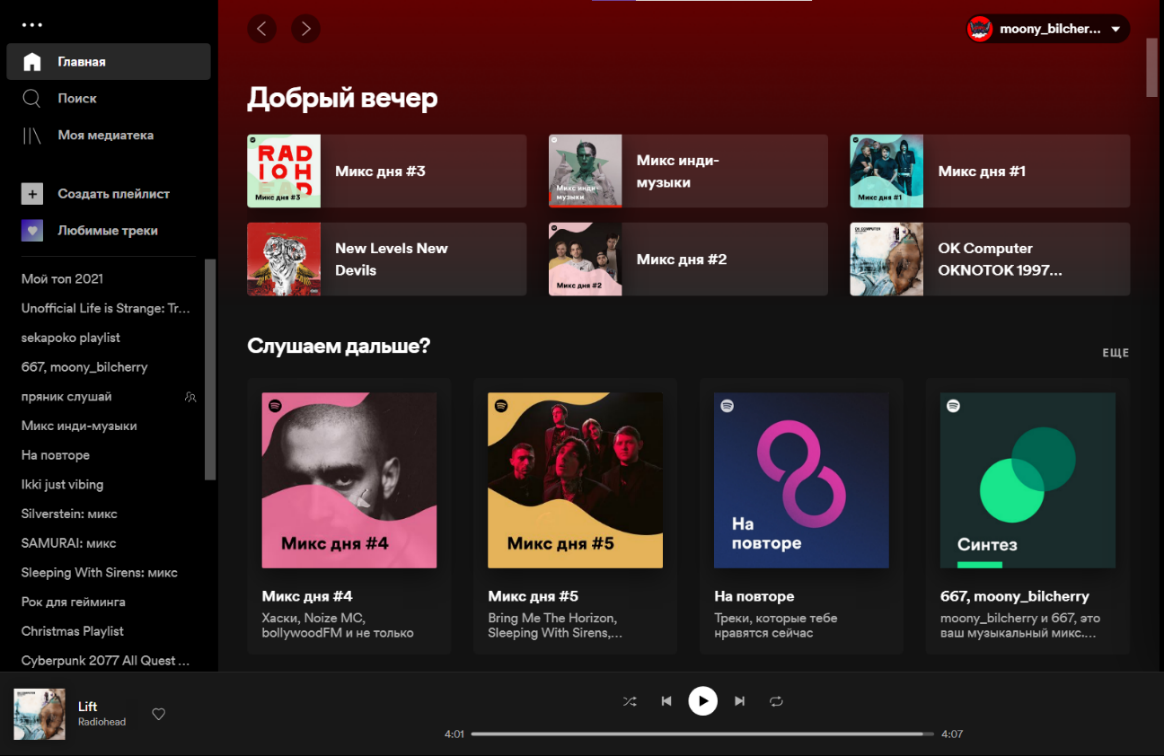


Рисунок 1.1 – Домашняя страница Spotify

С домашней страницы в Spotify пользователь может воспользоваться поиском, попасть в свою библиотеку, обратиться к недавно прослушанным плейлистам и открыть настойки своего аккаунта. Также Spotify по умолчанию имеет темный интерфейс

### 1.1.2 Аналог ipmtorgi.by

По популярности Apple Music идёт следом за Spotify, с 56 миллионами пользователей и большей по размеру библиотекой в 45 миллионов песен. Рассмотрим эту площадку на рисунке 1.2.

Apple Music на домашней странице имеет доступ к расширенному поиску. Пользователь получает рекомендации на основании своих прослушиваний и так же первым делом может обратить свое внимание на новые релизы от знакомых исполнителей. В боковом меню можно перейти к списку известных исполнителей, альбомов и песен.

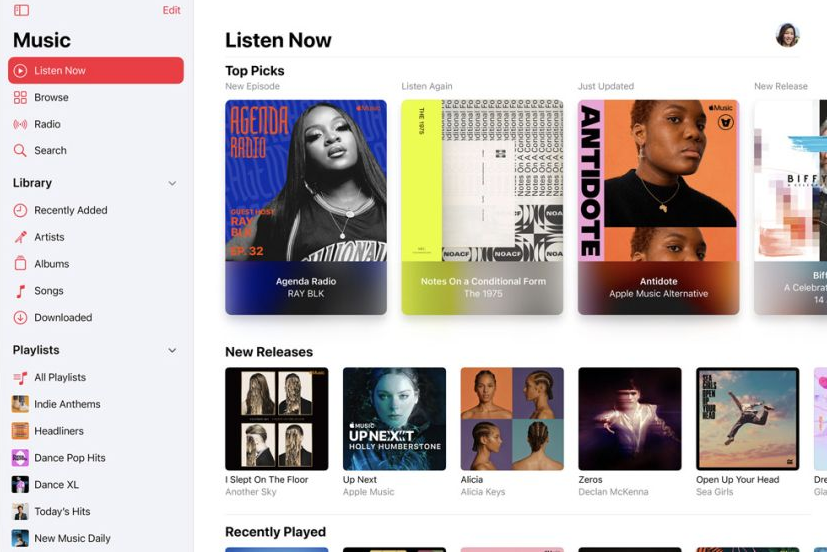


Рисунок 1.2 – Интерфейс сервиса ipmtorgi.by

Разобрав два приведенных выше сервиса для стриминга музыки, можно сформировать общий функционал необходимый для работы приложения.

Прежде всего, естественно, база данных должна корректно работать с изображениями и аудио файлами. Обязательно должен быть поиск песен по исполнителю, альбому и названию песни, также пользователь должен быть способен сохранять в личный плейлист понравившиеся ему песню.

Также стоит ввести простейшее меню настроек, чтобы пользователь мог изменить свой пароль или логин по желанию.

## 1.2 Разработка функциональных требований, определение вариантов использования

Проект предполагает создание базы данных для музыкальной площадки с использованием технологий мультимедийные типы данных и средств диагностики в СУБД Oracle 12c. База данных должна быть реализована с помощью соответствующих процедур для доступа к данным.

Важной частью проекта является обеспечение безопасности базы данных и защита ее от несанкционированного доступа. Для этого необходимо использовать механизмы аутентификации и авторизации, а также применить ролевую модель доступа к данным

Важным требованием к базе данных является импорт данных из XML файлов и экспорт данных в формат XML. Также необходимо протестировать производительность базы данных на таблице, содержащей не менее 100 000 строк, и внести изменения в структуру в случае необходимости.

Для подробного описания применяемых системных пакетов, утилит и технологий в базе данных проведен анализ функций пользователей и их ролей в системе.

В проекте присутствуют три вида пользователей: пользователь (user), автор (author) и админ (admin). Для каждого пользователя определены их роль и функции в системе, а также созданы соответствующие процедуры и запросы для доступа к данным.

Для более наглядного представления проекта и описания ролей пользователей можно использовать диаграмму UML. Например, диаграмма вариантов использования (use case diagram), представленная на рисунке 1.2, может помочь определить функциональные требования к базе данных для каждого пользователя. На диаграмме определены основные действия, которые пользователи выполняют в системе, и связи между ними.

## 1.3 Вывод

Итак, в результате проведенного аналитического обзора аналогов музыкальных площадок и сервисов были определены ключевые характеристики и функциональные возможности, которые должны быть реализованы в нашей системе. Были выявлены требования к базе данных, включая импорт и экспорт данных в формате XML, а также проведено тестирование производительности на таблице с более чем 100 000 строками. Также были определены роли пользователей и функциональные требования к системе в зависимости от этих ролей, и для этого была разработана UML-диаграмма вариантов использования системы. В целом, разработка базы данных для музыкальной площадки с использованием технологии мультемидийные типы данных и средств диагностики является важным шагом для обеспечения эффективной работы и удобства использования для всех ее пользователей.

1. Разработка архитектуры проекта

## 2.1 Обобщенная структура управлением приложения

Для обеспечения управления аукционной площадкой с использованием базы данных необходимо разработать удобный и интуитивно понятный интерфейс, который позволит пользователям взаимодействовать с базой данных и эффективно управлять данными. Это может включать в себя разработку оптимизированных запросов для вставки, обновления и удаления данных, а также механизмов для извлечения и обработки информации из базы данных.

В функциональность музыкальной площадки может входить возможность быстрого и удобного поиска композиций, пользователей, ролей, плейлистов, албомов. Также пользователи должны иметь возможность просмотривать композиции,добавлять и создавать плейлисты на сайте.

## 2.2 Диаграммы UML, взаимосвязь всех компонентов.

Диаграммы UML - это унифицированный язык моделирования, который позволяет разработчикам и аналитикам описывать структуру, поведение и взаимодействие компонентов системы. В контексте разработки базы данных для музыкальной площадки, использование диаграмм UML позволяет детально определить связи между всеми компонентами системы и обеспечить ее эффективное функционирование. В данном разделе будет рассмотрено применение диаграмм UML для описания взаимосвязи всех компонентов базы данных музыкальной площадки. В диаграмме, представленной на рисунке 2.1, описаны таблицы, образующие базу данных музыкальной площадки. Диаграмма таблиц помогает понять, каким образом компоненты взаимодействуют друг с другом и какие данные они обмениваются.

Для удобства визуализации структуры базы данных была создана диаграмма данных, которая отображает все таблицы и связи между ними. Диаграмма позволяет легко ориентироваться в структуре базы данных и понимать, какие данные хранятся в каждой таблице, а также как они связаны друг с другом. Такая диаграмма является важным инструментом для разработки, тестирования и сопровождения базы данных. Для музыкальной площадки важно отобразить все основные таблицы, которые отвечают за хранение данных о ролях, пользовотелей, композиций, плейлистах, исполнителях, альбомах, прослушанных и сохраненных песнях всех основных сущностей музыкальной площадки. Кроме того, диаграмма данных должна также отражать ограничения и свойства таблиц, такие как первичные и внешние ключи, ограничения целостности данных и другие ограничения, которые могут быть установлены для каждой таблицы. Это поможет разработчикам и администраторам баз данных легко понимать, какие ограничения установлены и как они взаимодействуют друг с другом.

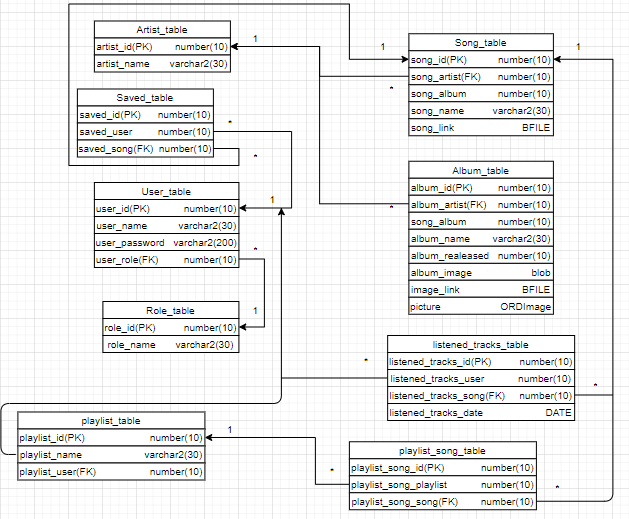


Рисунок 2.1 – Диаграмма базы данных музыкальной площадки

Таким образом, диаграмма таблиц для музыкальной площадки отражает структуру базы данных, предназначенной для хранения информации о ролях, пользовотелей, композиций, плейлистах, исполнителях, альбомах, прослушанных и сохраненных песнях. Она помогает визуализировать связи между данными, хранящимися в разных таблицах, и понять, как они взаимодействуют друг с другом.

**2.3 Описание информационных объектов**

Данная база данных описывает работу музыкальной площадки и состоит из 9 таблиц: role\_table, user\_table, artist\_table, album\_table, song\_table, saved\_table, playlist\_table, playlist\_song\_table и listened\_tracks\_table. Каждая из таблиц содержит информацию, необходимую для эффективного функционирования площадки, включая данные о продуктах, заказах, клиентах, категориях товаров, сотрудниках и аукционах. С помощью этой базы данных администраторы могут отслеживать процесс торговли и управлять заказами, клиентами и товарами на платформе.

Таблица " song\_table " является центральной таблицей в базе данных, которая хранит информацию о продуктах, которые продает компания. Эта таблица хранит информацию об отдельных песнях, доступных в системе, включая исполнителя, создавшего песню, альбом, к которому она относится, и название песни.. Данная таблица состоит из следующих столбцов:

ROLE\_TABLE: Эта таблица имеет два столбца, role\_id и role\_name. Столбец role\_id является автоматически генерируемым первичным ключом, а столбец role\_name хранит название роли. На этой таблице нет ограничений внешнего ключа или других ограничений.

Tаблица 2.1- role\_table

|  |  |
| --- | --- |
| role\_id | NUMBER, PRIMARY KEY |
| role\_name | VARCHAR2, NOT NULL |

USER\_TABLE: Эта таблица имеет четыре столбца, user\_id, user\_login, user\_password и user\_role. Столбец user\_id является автоматически генерируемым первичным ключом, а столбцы user\_login и user\_password хранят учетные данные пользователя. Столбец user\_role является внешним ключом, который ссылается на столбец role\_id в таблице ROLE\_TABLE. Эта таблица обеспечивает соблюдение ссылочной целостности между таблицами пользователей и ролей.

Tаблица 2.2- user\_table

|  |  |
| --- | --- |
| user\_id | NUMBER, PRIMARY KEY |
| user\_login | VARCHAR2, NOT NULL |
| user\_password | VARCHAR2, NOT NULL |
| user\_role | NUMBER, NOT NULL |

ARTIST\_TABLE: Эта таблица имеет два столбца, artist\_id и artist\_name. Столбец artist\_id является автоматически генерируемым первичным ключом, а столбец artist\_name хранит имя исполнителя. На этой таблице нет огрчений внешнего ключа или других ограничений.

Tаблица 2.3- artist\_table

|  |  |
| --- | --- |
| artist\_id | NUMBER, PRIMARY KEY |
| artist\_name | VARCHAR2, NOT NULL |

ALBUM\_TABLE: Эта таблица имеет пять столбцов, album\_id, album\_artist, album\_name, album\_released и album\_image. Столбец album\_id является автоматически генерируемым первичным ключом, а столбец album\_artist является внешним ключом, который ссылается на столбец artist\_id в таблице ARTIST\_TABLE. Столбцы album\_name и album\_released хранят название и дату выпуска альбома соответственно. Столбец album\_image хранит изображение обложки альбома. Эта таблица обеспечивает соблюдение ссылочной целостности между таблицами альбомов и исполнителей.

Tаблица 2.4-album\_table

|  |  |
| --- | --- |
| album\_id | NUMBER, PRIMARY KEY |
| album\_artist | VARCHAR2, NOT NULL |
| album\_name | VARCHAR2(30) NOT NULL |
| album\_released | NUMBER(10) NOT NULL |
| album\_image | BLOB |
| image\_link | BFILE |
| picture | ORDImage NOT NULL |

SONG\_TABLE: Эта таблица имеет четыре столбца, song\_id, song\_artist, song\_album и song\_name. Столбец song\_id является автоматически генерируемым первичным ключом, а столбцы song\_artist и song\_album являются внешними ключами, которые ссылается на столбцы artist\_id и album\_id в таблицах ARTIST\_TABLE и ALBUM\_TABLE соответственно. Столбец song\_name хранит название песни. Эта таблица обеспечивает соблюдение ссылочной целостности мду таблицами песен, исполнителей и альбомов.

Tаблица 2.5- song\_table

|  |  |
| --- | --- |
| song\_id | NUMBER, PRIMARY KEY |
| song\_artist | NUMBER(10) NOT NULL |
| song\_album | NUMBER(10) NOT NULL |
| song\_name | VARCHAR2(30) NOT NULL |
| song\_link | BFILE |

SAVED\_TABLE: Эта таблица имеет три столбца, saved\_id, saved\_user и saved\_song. Столбец saved\_id является автоматическиенерируемым первичным ключом, а столбцы saved\_user и saved\_song являются внешними ключами, которые ссылается на столбцы user\_id и song\_id в таблицах USER\_TABLE и SONG\_TABLE соответственно. Эта таблица обеспечивает соблюдение ссылочной целостности между таблицами сохраненных песен, пользователей и песен.

Tаблица 2.5- save\_table

|  |  |
| --- | --- |
| saved\_id | NUMBER(10), PRIMARY KEY |
| saved\_user | NUMBER(10) NOT NULL |
| saved\_song | NUMBER(10) NOT NULL |

PLAYLIST\_TABLE: Эта таблица имеет три столбца, playlist\_id, playlist\_name и playlist\_user. Столбец playlist\_id является автоматически генерируемым первичным ключом, а столбец playlist\_name хранит название плейлиста. Столбец playlist\_user является внешним ключом, который ссылается на столбц user\_id в таблице USER\_TABLE. Эта таблица обеспечивает соблюдение ссылочной целостности между таблицами плейлистов и пользователей.

Tаблица 2.7- playlist\_table

|  |  |
| --- | --- |
| playlist\_id | NUMBER(10), PRIMARY KEY |
| playlist\_name | VARCHAR2(30) NOT NULL |
| playlist\_user | NUMBER(10) NOT NULL |

PLAYLIST\_SONG\_TABLE: Эта таблица имеет три столбца, playlist\_song\_id, playlist\_song\_playlist и playlist\_song\_song. Столбец playlist\_song\_id является автоматически генерируемым первичным ключом, а столбцы playlist\_song\_playlist и playlist\_song\_song являются внешними ключами, которые ссылается на столбцы playlist\_id и song\_id в таблицах PLAYLIST\_TABLE и SONG\_TABLE соответственно. Эта таблица обеспечивает соблюдение ссылочной целостности между таблицами плейлистов, песен и связующей таблицы.

Tаблица 2.8- playlist\_song\_table

|  |  |
| --- | --- |
| playlist\_song\_id | NUMBER(10), PRIMARY KEY |
| playlist\_song\_playlist | NUMBER(10), NOT NULL |
| playlist\_song\_song | NUMBER(10), NOT NULL |

LISTENED\_TRACKS\_TABLE: Эта таблица имеет четыре столбца, listened\_tracks\_id, listened\_tracks\_user, listened\_tracks\_song и listened\_tracks\_date. Столбец listened\_tracks\_id является автоматически генерируемым первичным ключом, а столбцы listened\_tracks\_user и listened\_song являются внешними ключами, которые ссылается на столбцы user\_id и song\_id в таблицах USER\_TABLE и SONG\_TABLE соответственно. Столбец listened\_date хранит дату, когда пользователь прослушал песню. Эта таблица обеспечивает отслеживание прослушиваний песен пользователями и свывает таблицы пользователей и песен через внешние ключи.

Tаблица 2.9- listened\_tracks\_table

|  |  |
| --- | --- |
| listened\_tracks\_id | NUMBER(10), PRIMARY KEY |
| listened\_tracks\_user | NUMBER(10), NOT NULL |
| listened\_tracks\_song | NUMBER(10), NOT NULL |
| listened\_tracks\_date | DATE NOT NULL |

Таким образом, база данных музыкальной площадки содержит информацию о продуктах, заказах, клиентах, категориях, сотрудниках, аукционах и ставках на аукционах. Эти таблицы связаны между собой и обеспечивают хранение, обработку и управление данными, связанными с работой музыкальной площадки.

**2.4 Вывод**

Разработка архитектуры проекта является важным этапом для создания функционального и эффективного приложения. Описание информационных объектов является неотъемлемой частью этого процесса, так как позволяет определить, какие данные будут использоваться в приложении, и как они будут храниться и обрабатываться. В данной реализации базы данных музыкальной площадки были описаны объекты, такие как роли, пользовотели, композиции, плейлисты, исполнители, альбомы, прослушанные и сохраненные песни, что помогло определить структуру и функциональность приложения.

3 Разработка модели базы данных

## 3.1 Создание необходимых объектов

Для музыкальной площадки необходимо создать несколько таблиц, которые будут хранить данные о пользователях, ролях, композициях, плейлистах и сохраненных и прослушанных песнях.

Для музыкальной площадки необходимо создать следующие таблицы:

Role\_table- таблица, содержащая список продуктов на складе и их количество. В этой таблице будут храниться название товаров в наличии на складе, которые могут быть выставлены на аукцион в качестве лота.

User\_table- таблица, содержащая данные о заказах. Заказом в контексте данной площадки можно считать выигрышную ставку на товар, которая переходит в статус продаваемой. В этой таблице будут храниться информация о пользователе, такая как имя, фотография, электронная почта, дата рождения и т.д.

Artist\_table – таблица, содержащая информацию о товарах в заказе: идентификатор товара в заказе, идентификатор заказа, идентификатор заявки на участие в аукционе, количество товаров и цену за единицу товара.

Album\_table– таблица, содержащая информацию о категориях товаров: идентификатор категории, название и описание.

Song\_table – таблица, содержащая информацию о сотрудниках: идентификатор сотрудника, имя, должность (продавец или менеджер) и контактную информацию.

Saved\_table - таблица, содержащая информацию об оценках пользователей музыкальных треков. В этой таблице будут храниться данные об оценках, которые пользователи выставляют трекам.

Playlist\_table – таблица, содержащая информацию об аукционах: идентификатор аукциона, название, дату, время, местоположение и идентификатор продавца.

Playlist\_song\_table – таблица, содержащая информацию о товарах, проданных на аукционах: идентификатор товара на аукционе, идентификатор аукциона, идентификатор товара, начальную цену.

Listened\_tracks\_table – таблица, содержащая информацию о заявках на участие в аукционах: идентификатор заявки на участие в аукционе, идентификатор товара на аукционе, идентификатор покупателя, сумму ставки, дату ставки и флаг, указывающий, является ли заявка выигранной.

Для эффективного использования базы данных в проекте, необходимо создать индексы на столбцах, используемых в запросах с поиском данных. Например, создать индексы на столбцах customer\_email в таблице Customers, чтобы быстро находить информацию о клиенте по его электронной почте.

### 3.1.1 Представления базы данных

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

В данном проекте были создано несколько представлений рассмотрим 3 из них:

* Most\_listened\_compositions, которое содержит информацию о прослушиваемыемых композициях;
* User\_role\_view, в этом представлении отображаются логин пользователя, пароль пользователя и имя роли для всех пользователей;
* ARTIST & ALBUM, в этом представлении отображаются идентификатор альбома, имя исполнителя, название альбома, дата выпуска альбома.

Данное представление с названием User\_role\_view объединяет данные из нескольких таблиц в одной таблице, чтобы получить информацию о логине пользователя, пароли пользователя и имя роли для всех пользователей. Оно будет на листинге 3.1. Остальные представления будут аналогичны, только будут работать с другими таблицами.

|  |
| --- |
| CREATE VIEW user\_role\_view AS SELECT user\_table.user\_login, user\_table.user\_password, role\_table.role\_name  FROM user\_table LEFT JOIN role\_table ON user\_table.user\_role = role\_table.role\_id; |

Листинг 3.1 – Представление User\_role\_view

Таким образом, представление User\_role\_view упрощает доступ к данным о отображении логина пользователя, пароля пользователя и имя роли для всех пользователей, объединяя данные из нескольких таблиц в одной таблице.

### 3.1.2 Индексы базы данных

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

Для оптимизации запросов к таблицам базы данных были созданы группы индексов практически для каждого столбца таблицы. Они позволяют ускорять поиск по данным, таким как название песни, имя артиста, название альбома базы данных и др. Пример индексов для таблицы product представлен на листинге 3.2.

|  |
| --- |
| -- Song\_table  CREATE INDEX song\_name\_idx ON song\_table (song\_name);  CREATE INDEX song\_artist\_idx ON song\_table (song\_artist);  CREATE INDEX album\_name\_idx ON song\_table (album\_name); |

Листинг 3.2 – Индексы базы данных для таблицы Song\_table.

Кроме индексов song\_name\_idx, song\_artist\_idx, album\_name\_idx в базе данных также присутствуют индексы на полях других таблиц, таких как Album\_table, Artist\_table, User\_table, playlist\_table и других.

Индекс album\_name\_idx на таблице Album\_table был создан для ускорения поиска пользователя по его названию.

Индекс saved\_table на таблице Saved\_table может использоваться для быстрого поиска сохрненных песень по их названию.

Индекс user\_login\_idx на таблице User\_table был создан для ускорения поиска учетных записей пользователя по его логину.

В целом, использование индексов позволяет существенно ускорить операции поиска, сортировки и фильтрации данных в базе данных, особенно в случае большого объема данных. Однако создание индексов может занять дополнительное время при добавлении или изменении данных в таблицах, поэтому необходимо сбалансировать количество и тип индексов для оптимальной производительности базы данных.

### 3.1.3 Триггеры базы данных

Триггер базы данных — это объект базы данных, который выполняет некоторое действие автоматически при определенных событиях в таблице или представлении базы данных. Триггер может быть запрограммирован на срабатывание при вставке, обновлении или удалении строк в таблице.

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

В данном случае были созданы триггеры для таблиц базы данных, которые проверяет, что дата прослушивания песни не находится в будущем и записывают данные с типом XMLType в специальные таблицы (например, Album\_table \_XML). Эти таблицы затем используются процедурами, которые используют пакет UTL\_FILE для записи данных в XML-файл.

Каждый триггер содержит код, который выполняется перед или после операции insert, update или delete на определенной таблице базы данных. Код триггера получает доступ к измененным данным через предопределенные ссылки на старые и новые значения (old и new), которые могут использоваться для формирования XML-данных. Затем, при помощи функции insert, данные вставляются в соответствующую таблицу XML. Пример триггера реагирующего на операцию update для таблицы Listened\_tracks\_table будет представлен на листинге 3.3.

|  |
| --- |
| CREATE OR REPLACE TRIGGER trg\_listened\_t\_update  AFTER UPDATE ON listened\_tracks\_table  FOR EACH ROW  DECLARE  v\_xml XMLType;  BEGIN  SELECT XMLType(  '<listened\_track>' ||  '<listened\_track\_id>' || :new.listened\_tracks\_id || '</listened\_track\_id>' ||  '<listened\_track\_user>' || :new.listened\_tracks\_user || '</listened\_track\_user>' ||  '<listened\_track\_song>' || :new.listened\_tracks\_song || '</listened\_track\_song>' ||  '<listened\_track\_date>' || :new.listened\_tracks\_date || '</listened\_track\_date>' ||  '</listened\_track>'  )  INTO v\_xml  FROM dual;  UPDATE listened\_tracks\_table\_XML SET xml\_data = v\_xml WHERE EXTRACTVALUE(xml\_data, '/listened\_track/listened\_track\_id/text()') = :new.listened\_tracks\_id;  END; |

Листинг 3.3 – Скрипт триггера trg\_listened\_t\_update

Второй триггер проверяет, что артист песни, добавляемой в плейлист, разрешен для пользователя, создавшего плейлист. Триггер выполняется до того, как какие-либо строки будут вставлены в таблицу. Если артист песни не разрешен для пользователя, создавшего плейлист, то триггер вызывает ошибку. Триггер объявляет две переменные: v\_song\_artist и v\_playlist\_user. Триггер использует оператор SELECT для получения исполнителя песни и пользователя списка воспроизведения из таблиц song\_table и playlist\_table соответственно. Триггер использует оператор IN для проверки наличия исполнителя песни в списке пользователей, которым разрешено добавлять песни в список воспроизведения. Скрипт триггера будет представлен на листинге 3.4.

|  |
| --- |
| CREATE OR REPLACE TRIGGER TRG\_CHECK\_PLAYLIST\_SONG  BEFORE INSERT ON playlist\_song\_table  FOR EACH ROW  DECLARE  v\_song\_artist NUMBER(10);  v\_playlist\_user NUMBER(10);  BEGIN  SELECT song\_artist INTO v\_song\_artist  FROM song\_table  WHERE song\_id = :NEW.playlist\_song\_song;  SELECT user\_id INTO v\_playlist\_user  FROM playlist\_table  WHERE playlist\_id = :NEW.playlist\_song\_playlist;  IF v\_song\_artist NOT IN (SELECT user\_role FROM user\_table WHERE user\_id = v\_playlist\_user) THEN  RAISE\_APPLICATION\_ERROR(-20001, 'The song artist is not allowed in this playlist.');  END IF;  END; |

Листинг 3.4 – Скрип триггера trg\_check\_playlist\_song

Таким образом, триггер выполняют валидацию данных перед их вставкой или обновлением в таблицу, обеспечивая целостность данных.

## 3.2 Описание используемой технологии

Oracle предоставляет несколько типов данных для хранения мультимедийных объектов, таких как изображения, звуковые файлы и видео. Некоторые из этих типов данных включают в себя:

ORDImage - это тип данных, который используется для хранения изображений в Oracle. Он позволяет хранить изображения в различных форматах, таких как JPEG, GIF, BMP и PNG. ORDImage также предоставляет множество методов для работы с изображениями, таких как изменение размера, поворот, наложение текста и т.д.

BLOB - это тип данных, который используется для хранения двоичных данных в Oracle. Он может использоваться для хранения мультимедийных объектов, таких как изображения, звуковые файлы и видео. BLOB может хранить данные до 4 гигабайт.

BFILE - это тип данных, который используется для хранения ссылок на файлы, которые находятся вне базы данных Oracle. BFILE позволяет хранить мультимедийные объекты в файлах, которые могут быть доступны извне базы данных. BFILE может хранить ссылки на файлы до 4 гигабайт.

Для работы с мультимедийными типами данных в Oracle, можно использовать различные инструменты и технологии, такие как Oracle Multimedia, Oracle Text и Oracle Spatial. Oracle Multimedia предоставляет набор инструментов для работы с мультимедийными типами данных, включая возможность создания и изменения изображений, аудио и видео файлов. Oracle Text позволяет выполнять поиск по мультимедийным объектам, а Oracle Spatial позволяет выполнять пространственный анализ мультимедийных объектов.

Для реализации технологии мультимедийных типов данных в базе данных реализованы столбцы таблицы TA\_TOUR, хранящие в себе изображения туров, а также ссылку на изображение, код для создания таблицы представлен на рисунке 2.2.

|  |
| --- |
| CREATE TABLE album\_table (  album\_id NUMBER(10) GENERATED AS IDENTITY(START WITH 1 INCREMENT BY 1),  album\_artist NUMBER(10) NOT NULL,  album\_name VARCHAR2(30) NOT NULL,  album\_released NUMBER(10) NOT NULL,  album\_image BLOB ,  image\_link BFILE,  picture ORDImage NOT NULL,  CONSTRAINT album\_pk PRIMARY KEY (album\_id),  CONSTRAINT album\_artist\_fk FOREIGN KEY (album\_artist) REFERENCES artist\_table(artist\_id)); |

Листинг 3.5 – Скрип создания таблицы Album\_table

Процедура, которая вставляет альбом в таблицу Album\_table. Процедура принимает пять параметров, таких как исполнитель альбома (pAlbum\_artist), название альбома (pAlbum\_name), год выпуска альбома (pAlbum\_released), каталог, в котором хранится изображение альбома (pOraDir), имя файла изображения альбома (pFileName), количество мест (pSeats\_count), путь к директории (pOraDir) и имя файла (pFileName). Процедура сначала создает объект BFILE для файла изображения альбома. Затем он создает объект BLOB для хранения изображения альбома. Затем он вставляет строку в таблицу Album\_table. После вставки данных в таблицу Album\_table, происходит загрузка изображения из файла в переменную l\_blob с помощью функции DBMS\_LOB.loadblobfromfile(). Перед этим файл открывается с помощью функции DBMS\_LOB.fileopen() и закрывается с помощью функции DBMS\_LOB.fileclose(). Затем происходит COMMIT для сохранения изменений в базе данных.

В случае возникновения ошибки, происходит вывод сообщения об ошибке с помощью функции DBMS\_OUTPUT.PUT\_LINE().

|  |
| --- |
| CREATE OR REPLACE PROCEDURE INSERT\_ALBUM(pAlbum\_artist NUMBER,pAlbum\_name VARCHAR2,  pAlbum\_released NUMBER,pOraDir VARCHAR2,pFileName VARCHAR2) IS  l\_bfile BFILE;  l\_blob BLOB;  l\_dest\_offset INTEGER := 1;  l\_src\_offset INTEGER := 1;  l\_lobmaxsize CONSTANT INTEGER := DBMS\_LOB.LOBMAXSIZE;  l\_img ORDImage;  BEGIN  l\_bfile := BFILENAME(pOraDir, pFileName);  l\_img := ORDSYS.ORDImage(l\_blob);  INSERT INTO album\_table(album\_id,album\_artist,album\_name,album\_released,album\_image,image\_link,picture)  VALUES(NULL,pAlbum\_artist,pAlbum\_name,pAlbum\_released,empty\_blob(),l\_bfile,l\_img)  RETURNING album\_image INTO l\_blob;  DBMS\_LOB.fileopen(l\_bfile, DBMS\_LOB.file\_readonly);  DBMS\_LOB.loadblobfromfile(l\_blob, l\_bfile, l\_lobmaxsize, l\_dest\_offset, l\_src\_offset);  DBMS\_LOB.fileclose(l\_bfile);  COMMIT;  EXCEPTION  WHEN OTHERS THEN  DBMS\_OUTPUT.PUT\_LINE('CODE ERROR: '||SQLCODE||' MESSAGE: '||SQLERRM);  END; |

Листинг 3.6 – Скрип создания процедуры INSERT\_ALBUM

На рисунке 3.1 представлен код процедуры для добавления значений в таблицу Album\_table. При выборке значений из таблицы данные представлены в определённом типе, для просмотра непосредственно загруженных изображений, необходимо дважды нажать на иконку редактирования значения в ячейке, а после выбрать пункт Image.

Результатом данный действий будет открытое окно с выводом значения столбца типа blob.

Результат SELECT-запроса представлен на рисунке 3.1 и 3.2.

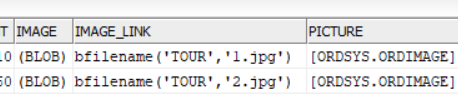


Рисунок 3.1 – Результат SELECT-запроса

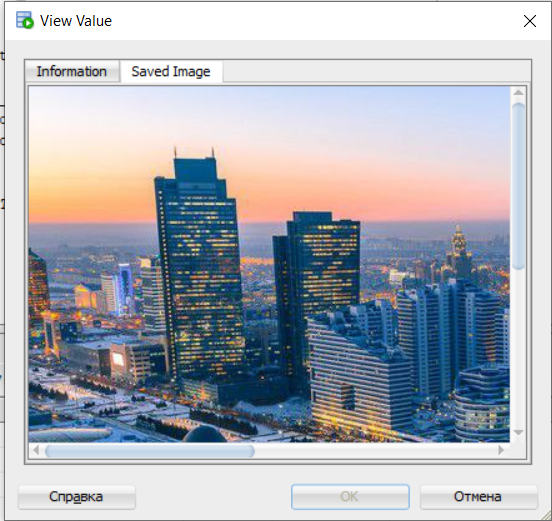


Рисунок 3.2 – Результат SELECT-запроса

Несмотря на то, что Oracle предоставляет очень богатый функционал для разработчика, его отличительной чертой является также ограниченность в бесплатном вспомогательном программном обеспечении, в отличие от MS SQL. Из очень ограниченного круга средств мой выбор упал на Database Performance Analyzer, разработанный компанией SolarWinds, поскольку с ним предоставляется 14-тидневный период оценки ПО, в рамках которого я был ознакомлен с его функционалом.

Программное обеспечение SolarWinds Database Performance Analyzer – это инструмент мониторинга производительности SQL-сервера, поддерживающий также и сервер Oracle 12c, помогающий выявлять наиболее опасные для конечных пользователей проблемы.

Благодаря многомерному анализу SolarWinds Database Performance Analyzer помогает выявлять корневые причины неполадок. Продукт превосходит обычные решения для наблюдения за состоянием сервера; это единый инструмент администраторов, разработчиков и менеджеров баз данных, который позволяет экономично и эффективно совершенствовать работу серверов SQL. Интерфейс SolarWinds Database Performance Analyzer изображен на рисунке 4.5.

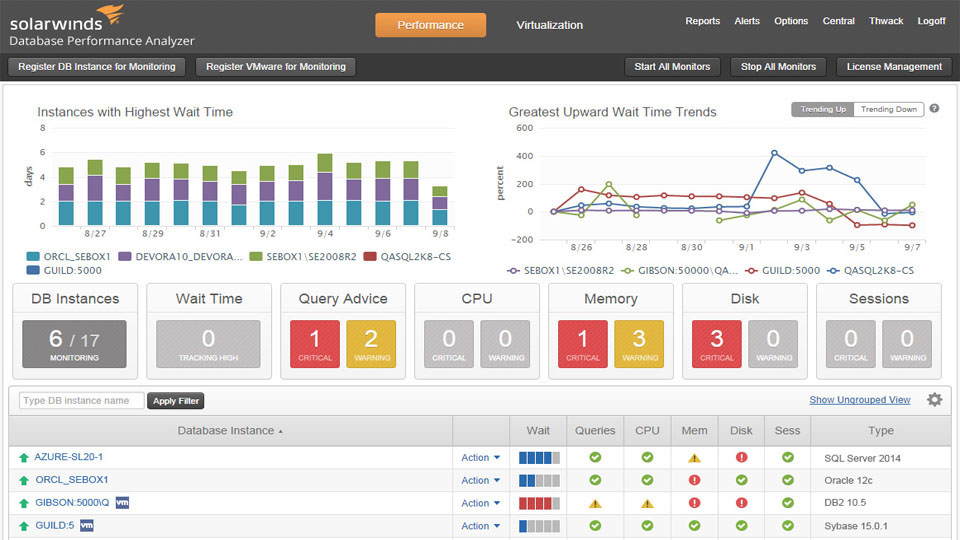


Рисунок 3.3– Интерфейс SolarWinds DPA

В горизонтальном блоке плиток по центру рисунка 4.5 можно увидеть, что DPA позволяет отслеживать следующие показатели: количество отслеживаемых баз данных, количество БД с критическим временем задержки, очереди запросов, состояния процессоров, состояние памяти, состояние дисков, сессии.

Во вкладке Alerts (Оповещения), изображенной на рисунке 4.6, можно отслеживать статус ошибок и неисправностей, создавать группы оповещений, задать случаи, когда необходимо оповещать о неисправности по электронной почте.

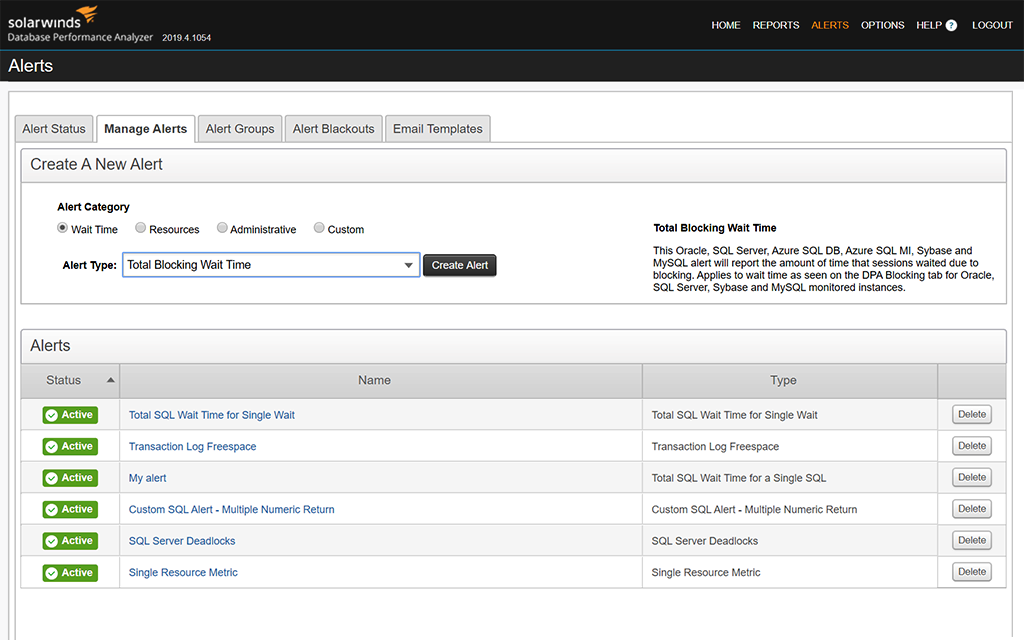


Рисунок 3.4– Вкладка Alerts

По итогам 14-тидневного периода пользования мною были отмечены следующие преимущества данного средства диагностики. Это ПО оказалось крайне полезным, поскольку оно предоставляет полную картину. За несколько секунд программа показывает текущий статус сеанса, так что администраторы баз данных могут мгновенно реагировать на проблемы производительности.

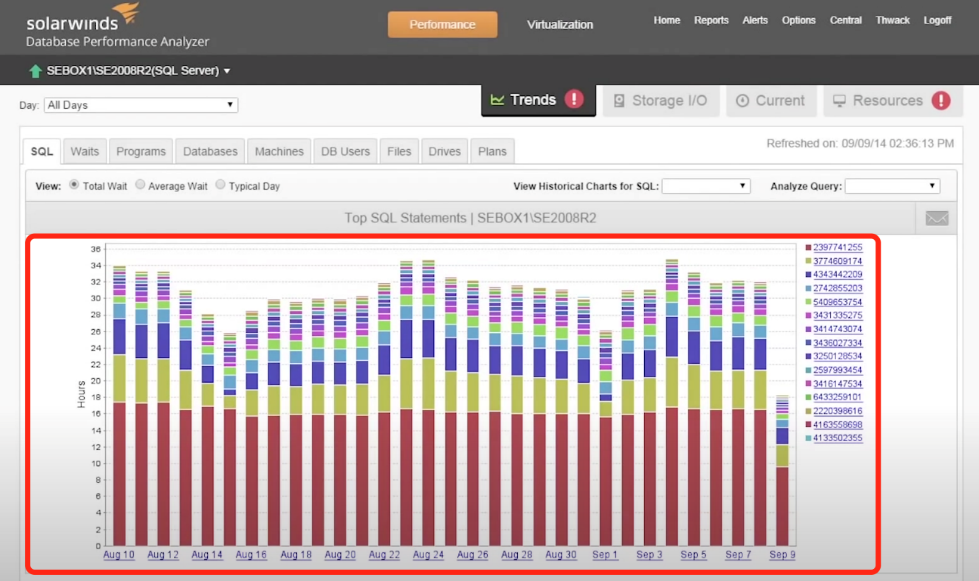


Рисунок 3.5– Диаграмма времени ожидания запросов

На первой вкладке подробной информации будет находиться временная диаграмма, отображающая время ожидания запросов, как показано на рисунке 4.7.

Также, как оказалось, SolarWinds DPA поддерживает все возможные серверы баз данных, что сводит на «нет» нужды в нескольких средствах для мониторинга базы данных. Эта система «все в одном» справится с любым типом сред: физической, виртуальной, облачной. Еще одной сильной стороной SolarWinds DPA оказалась графическая составляющая: имеющиеся графики очень информативны и интуитивно понятны.

## 3.3 Вывод

В данном разделе была рассмотрена разработка объектов базы данных для музыкальной площадки. Tакже была описана технология диагностики и мультемидийые типы данных базы данных. Технология диагностики базы данных является неотъемлемой частью работы с базами данных, так как позволяет контролировать состояние системы, выявлять проблемы и решать их до того, как они приведут к серьезным последствиям.

4 Установка, настройка и использование Oracle 12c

## 4.1 Установка Oracle 12c

Oracle 12c - это реляционная база данных, выпущенная компанией Oracle Corporation. Она предлагает множество функций, таких как масштабируемость, высокую производительность, защиту данных, поддержку многопоточности и транзакционность. Одной из особенностей Oracle 12c является использование контейнерной архитектуры, где база данных разделена на отдельные контейнеры (CDB) и содержащиеся в них базы данных подчинены этим контейнерам (PDB).

После установки Oracle 12c на сервер, была произведена конфигурация сервера для оптимальной работы с базой данных. Затем была создана база данных с названием auctions\_pdb, которая будет использоваться в дальнейшем для хранения данных музыкального приложения.

## 4.2 Создание таблиц

В данном разделе мы создадим таблицы для нашей базы данных. Но перед тем, как приступить к созданию таблиц, нам нужно создать табличное пространство.

Табличное пространство - это механизм, который помогает связать объекты базы данных, такие как таблицы, индексы и представления, с файловой системой. Оно позволяет логически разделять объекты базы данных на разные физические устройства или диски, что может улучшить производительность работы с базой данных. Скрипт для создания табличных пространств будет представлен на листинге 4.1.

|  |
| --- |
| CREATE TABLESPACE TS\_MAIN  DATAFILE 'C:\app\Kursach\TS\_MAIN.dbf'  SIZE 500M  AUTOEXTEND ON  EXTENT MANAGEMENT LOCAL;  CREATE TEMPORARY TABLESPACE TS\_TEMP  TEMPFILE 'C:\app\Kursach\с.dbf'  SIZE 100M  AUTOEXTEND ON  EXTENT MANAGEMENT LOCAL; |

Листинг 4.1 – Cкрипт для создания табличных пространств

Для базы данных создано 2 табличных пространство: TS\_MAIN и Auctions\_Temp. Auctions\_Temp в свою очередь является временным табличным пространством, в котором будут храниться временные объекты.

Каждая таблица будет содержать свои поля (столбцы) и ограничения (constraints), которые определяют правила для хранения и изменения данных. Например, ограничение FOREIGN KEY определяет связь между двумя таблицами, а ограничение PRIMARY KEY определяет уникальный идентификатор для каждой записи в таблице.

Кроме того, в базе данных будут присутствовать связи между таблицами. Одна из основных связей - это связь "один ко многим" (one-to-many), которая определяет отношение одной записи в таблице к нескольким записям в другой таблице. Например, у артиста может быть множество композиций, которые он создал. Для этого мы добавим в таблицу Song\_table внешний ключ (FOREIGN KEY) на таблицу Artist\_table, который будет указывать на идентификатор заказанного товара. На листинге 4.2 представлен код создания таблицы Song\_table.

|  |
| --- |
| CREATE TABLE song\_table (  song\_id NUMBER(10) GENERATED AS IDENTITY(START WITH 1 INCREMENT BY 1),  song\_artist NUMBER(10) NOT NULL,  song\_album NUMBER(10) NOT NULL,  song\_name VARCHAR2(30) NOT NULL,  song\_link BFILE,  CONSTRAINT song\_pk PRIMARY KEY (song\_id),  CONSTRAINT song\_artist\_fk FOREIGN KEY (song\_artist) REFERENCES artist\_table(artist\_id),  CONSTRAINT song\_album\_fk FOREIGN KEY (song\_album) REFERENCES album\_table(album\_id)  ); |

Листинг 4.2 – Cкрипт создание таблицы Song\_table

Таким образом, было описано создание табличного пространства для базы данных, а также таблиц, которые будут храниться в этих пространствах. Было создано два табличных пространства: TS\_MAIN, TS\_TEMP.

## Создание ролей для разграничения

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

Будут созданы три роли для разграничения доступа к базе данных: ROLE\_USER, ROLE\_AUTHOR и ROLE\_ADMIN. Это позволит ограничить доступ к определенным функциям базы данных и предотвратить несанкционированный доступ к конфиденциальной информации. Роли ROLE\_USER и ROLE\_AUTHOR являются основными, в то время как роль manager\_role будет иметь больший набор привелегий, по сравнению с ROLE\_ADMIN.

Роль ROLE\_ADMIN имеет полный доступ ко всей базе данных и может выполнять любые операции, в том числе создавать и изменять таблицы, индексы, представления и триггеры. Выданные привилегии роли ROLE\_ADMIN можно увидеть на листинге 4.3.

|  |
| --- |
| grant execute on INSERT\_ALBUM on ROLE\_ADMIN;  grant execute on PROCEDURE INSERT\_USER on ROLE\_ADMIN;  grant execute on PROCEDURE INSERT\_ROLE on ROLE\_ADMIN;  grant execute on PROCEDURE INSERT\_ARTIST on ROLE\_ADMIN;  grant execute on PROCEDURE INSERT\_SONG on ROLE\_ADMIN;  grant execute on PROCEDURE INSERT\_SAVED on ROLE\_ADMIN;  grant dba to ROLE\_ADMIN; |

Листинг 4.3 – Привилегии, выданные роли ROLE\_ADMIN

Роль manager\_role имеет права на выполнение различных функций и процедур, связанных с управлением пользователями, товарами и аукционами в базе данных. Эта роль может выполнять функции добавления, редактирования и удаления товаров в ассортименте, а также обновления информации о пользователях. Выданные привилегии роли manager\_role можно увидеть на листинге 4.4.

|  |
| --- |
| GRANT SELECT,INSERT,UPDATE,DELETE ON artist\_table on ROLE\_ AUTHOR;  GRANT SELECT,INSERT,UPDATE,DELETE ON album\_tabla on ROLE\_ AUTHOR;  GRANT SELECT,INSERT,UPDATE,DELETE ON song\_tabla on ROLE\_ AUTHOR;  grant execute on PROCEDURE INSERT\_ARTIST on ROLE\_AUTHOR;  grant execute on PROCEDURE INSERT\_SONG on ROLE\_AUTHOR;  grant execute on PROCEDURE search\_songs\_by\_name ON ROLE\_AUTHOR;  grant execute on PROCEDURE search\_users\_by\_name ON ROLE\_AUTHOR; |

Листинг 4.4 – Привилегии, выданные роли ROLE\_AUTHOR

Роль customer может делать ставки, просматривать каталог товаров и видеть некоторую информацию о продавце. Кроме того, customer имеет возможность редактировать некоторую информацию о себе. Выданные привилегии роли customer\_role можно увидеть на листинге 4.5.

|  |
| --- |
| grant execute on PROCEDURE search\_songs\_by\_name on ROLE\_USER;  grant execute on PROCEDURE search\_users\_by\_name on ROLE\_USER;  grant execute on PROCEDURE search\_songs\_by\_name on ROLE\_USER;  grant execute on PROCEDURE search\_users\_by\_name on ROLE\_USER; |

Листинг 4.5 – Привилегии, выданные роли ROLE\_USER

Таким образом, были созданы три роли для ограничения доступа к базе данных: ROLE\_USER, ROLE\_AUTHOR, ROLE\_AUTHOR. Каждая роль имеет определенный набор привилегий, который позволяет пользователю выполнять определенные функции в базе данных. Роль admin\_role имеет наибольшие привилегии и может выполнять любые операции в базе данных, в то время как роль customer\_role имеет ограниченный набор привилегий, который позволяет пользователю только просматривать каталог товаров и делать ставки. Роль manager\_role имеет средний уровень привилегий и может выполнять функции, связанные с управлением пользователями и лотами в базе данных.

## 4.4 Создание пакетов процедур для базы данных

Для управления данными через приложение пользователи и администраторы используют 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.

Написанные в ходе разработки курсового проекта процедуры и функции можно разбить на несколько категорий:

1. Выборка данных из таблиц;
2. Выборка данных по поисковому запросу;
3. Заполнение таблиц 100 000 строк;
4. Добавление данных в таблицы;
5. Удаление данных из таблиц;
6. Изменение данных в таблицах;
7. Дополнительные функции.

Отличие функций от процедур состоит в том, что функции возвращают значение, которое может быть использовано в других SQL-запросах, а процедуры не возвращают значение. Кроме того, функции могут быть использованы в выражениях SQL, например, для вычисления значения поля в запросе SELECT.

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.

### 4.4.1 Выборка данных из таблиц

Для вывода данных из таблиц были написаны следующие процедуры и функции: SELECT\_ROLES, SELECT\_ROLES, SELECT\_ARTISTS, SELECT\_SONGS, SELECT\_SAVED, SELECT\_ALBUMS, USER\_ROLE\_VIEW, ARTIST\_ALBUM\_VIEW, SELECT\_PLAYLISTS, SELECT\_PLAYLIST\_SONGS, SELECT\_LISTENED\_TRACKS. Основная их задача – выборка данных из всех основных таблиц базы данных. Ниже будут описание каждой функции.

SELECT\_USERS, SELECT\_ROLES, SELECT\_PLAYLISTS - функции для выборки списка пользователей, треков и всех плейлистов.

USER\_ROLE\_VIEW, ARTIST\_ALBUM\_VIEW - функции для получения информации об определенном пользователе, треке или плейлисте.

SELECT\_PLAYLISTS, SELECT\_PLAYLIST\_SONGS - функции для получения связанных записей между пользователями и треками.

SELECT\_LISTENED\_TRACKS. - функции для выборки плейлистов и треков, которые то прослушиваются.

На листинге 4.6 будет функция SELECT\_USERS, которая предназначена для выборки всех пользователей из таблицы Users.

|  |
| --- |
| CREATE OR REPLACE PROCEDURE SELECT\_USERS  AS  pUser\_login user\_table.user\_login%TYPE;  pUser\_password user\_table.user\_password%TYPE;  pUser\_role user\_table.user\_role%TYPE;  CURSOR USER\_CURSOR IS  SELECT user\_login, user\_password, user\_role  FROM user\_table;  BEGIN  OPEN USER\_CURSOR;  LOOP  FETCH USER\_CURSOR INTO pUser\_login, pUser\_password, pUser\_role;  EXIT WHEN USER\_CURSOR%NOTFOUND;  IF USER\_CURSOR%NOTFOUND THEN  RAISE\_APPLICATION\_ERROR(-20003, 'Data is not found');  END IF;  DBMS\_OUTPUT.PUT\_LINE(pUser\_login || ' ' || pUser\_password || ' ' || pUser\_role);  END LOOP;  CLOSE USER\_CURSOR;  EXCEPTION  WHEN OTHERS  THEN DBMS\_OUTPUT.PUT\_LINE('CODE ERROR: '||SQLCODE||' MESSAGE: '||SQLERRM);  END; |

Листинг 4.6 – Функция SELECT\_USERS

Все остальные функции и процедуры будут аналогичны, также предназначены для выборки данных из различных таблиц базы данных.

### 4.4.3 Заполнение таблиц 100 000 строк

Для заполнения таблицы Role\_table была разработана функция insert\_roles, которая вставляет 100000 строк в таблицу. Функция представлена на листинге 4.8.

|  |
| --- |
| CREATE OR REPLACE PROCEDURE insert\_roles AS  BEGIN  FOR i IN 1..100000 LOOP  INSERT INTO role\_table (  role\_name  ) VALUES (  'Role ' || i  );  END LOOP;  COMMIT;  END; |

Листинг 4.8 – Функция заполнения таблицы Role\_table

Функция insert\_ roles была создана для заполнения таблицы Role\_table 100000 строками. В теле функции используется цикл FOR, который проходит по значениям от 1 до 100000 и для каждого значения выполняет вставку новой строки в таблицу Role\_table с именем ' Role ' и порядковым номером из цикла. Процедура не имеет выходного параметра и не принимает аргументов.

### 4.4.4 Добавление данных в таблицы

Для добавления значений в таблицы были разработаны процедуры, являющиеся оболочками для вставки значений в определённые таблицы базы данных. На листинге 4.9 показана процедура INSERT\_LISTENED\_TRACKS, добавляющая новые строки в таблицу LISTENED\_TRACKS.

|  |
| --- |
| C CREATE OR REPLACE PROCEDURE INSERT\_LISTENED\_TRACKS(  pListened\_tracks\_user NUMBER,  pListened\_tracks\_song NUMBER,  pListened\_tracks\_date DATE  ) IS  BEGIN  INSERT INTO listened\_tracks\_table(listened\_tracks\_user, listened\_tracks\_song, listened\_tracks\_date) VALUES(pListened\_tracks\_user, pListened\_tracks\_song, pListened\_tracks\_date);  COMMIT;  EXCEPTION  WHEN OTHERS THEN  DBMS\_OUTPUT.PUT\_LINE('CODE ERROR: '||SQLCODE||' MESSAGE: '||SQLERRM);  END; |

Листинг 4.9 – Процедура добавления прослушанных треков

Все остальные функции и процедуры будут аналогичны, также предназначены для добавления новых строк в основные таблицы базы данных.

### 4.4.6 Изменение данных в таблицы

Для изменения значений в таблицы по id были разработаны процедуры, являющиеся оболочками для обновления строк в определённые таблицы базы данных. На листинге 4.10 показана процедура UPDATE\_ROLE, добавляющая новые строки в таблицу Role\_table

|  |
| --- |
| create or replace procedure UPDATE\_ROLE(pId\_role number, pRole\_name nvarchar2)  is  begin  UPDATE ROLE\_TABLE SET ROLE\_NAME = pRole\_name  WHERE ROLE\_ID = pId\_role;  EXCEPTION  WHEN OTHERS  THEN DBMS\_OUTPUT.PUT\_LINE('CODE ERROR: '||SQLCODE||' MESSAGE: '||SQLERRM);  end; |

Листинг 4.10 – Процедура обновления товара

Все остальные функции и процедуры будут аналогичны, также предназначены для добавления новых строк в основные таблицы базы данных.

### 4.4.6 Удаление данных из таблицы

Для удаления строк из таблицы по id были разработаны процедуры, являющиеся оболочками для обновления строк в определённые таблицы базы данных. На листинге 4.11 показана процедура delete\_product, добавляющая новые строки в таблицу Products.

|  |
| --- |
| CREATE OR REPLACE PROCEDURE update\_product (  p\_id IN NUMBER,  p\_name IN VARCHAR2,  p\_desc IN VARCHAR2,  cat\_id IN NUMBER,  stock\_qty IN NUMBER,  p\_price IN NUMBER  ) AS  BEGIN  UPDATE Products  SET product\_name = p\_name,  product\_description = p\_desc,  category\_id = cat\_id,  stock\_quantity = stock\_qty,  price = p\_price  WHERE product\_id = p\_id;  END; |

Листинг 4.10 – Процедура удаления товара

Все остальные функции и процедуры будут аналогичны, также предназначены для удаления строк из основных таблиц базы данных.

### 4.4.7 Дополнительные функции

Дополнительные функции в базе данных могут быть полезны для решения различных задач, которые не решаются стандартными запросами.

Процедура search\_playlists\_by\_name позволяет получить плейлист по его имени, что также может быть полезно при использовании базы данных. Функция представлена на листинге 4.12.

|  |
| --- |
| CREATE OR REPLACE PROCEDURE search\_playlists\_by\_name(  p\_search\_term IN VARCHAR2,  p\_results OUT SYS\_REFCURSOR  )  AS  BEGIN  OPEN p\_results FOR  SELECT \*  FROM playlist\_table  WHERE playlist\_name LIKE '%' || p\_search\_term || '%';  END; |

Листинг 4.12 – Процедура search\_playlists\_by\_name

Процедура register\_user позволяет зарегестрироваться на сайте. Процедура представлена на листинге 4.13.

|  |
| --- |
| C CREATE OR REPLACE PROCEDURE register\_user  (p\_user\_login IN user\_table.user\_login%TYPE,  p\_user\_password IN user\_table.user\_password%TYPE)  IS  cnt NUMBER;  BEGIN  SELECT COUNT(\*) INTO cnt FROM user\_table WHERE UPPER(user\_login) = UPPER(p\_user\_login);  IF (cnt = 0) THEN  INSERT INTO user\_table(user\_login, user\_password, user\_role) VALUES(UPPER(p\_user\_login), encryption\_password(UPPER(p\_user\_password)), 1);  COMMIT;  ELSE  RAISE\_APPLICATION\_ERROR(-20001, 'This login is already taken');  END IF;  END register\_user; |

Листинг 4.13 – Процедура mark\_ register\_user.

В целом, эти функции могут быть полезны для повышения безопасности хранения паролей пользователей, а также для получения дополнительной информации в базе данных.

## 4.5 Описание процедур экспорта в XML

Для базы данных была предусмотрена функция экспорта данных таблиц базы данных в xml файлы, хранящиеся на сервере.

В начале были созданы соответствующие таблицы для каждой из таблиц базы данных, состоящие из одно столбца типа XMLType. В Листинге 4.14. приведён пример создание данной таблицы.

|  |
| --- |
| CREATE TABLE role\_table\_XML (xml\_data XMLType) XMLTYPE COLUMN xml\_data STORE AS BINARY XML; |

Листинг 4.14 – Таблица role\_table\_XML

Данные таблицы, содержащие xml данные, используются триггерами таблиц базы данных при удалении, изменении, добавлении строк в оригинальные таблицы. В листинге 4.16. представлен триггер, который записывает данные параллельно в таблицу role\_table \_XML при их добавлении в таблицу role\_table.

|  |
| --- |
| CREATE OR REPLACE TRIGGER trg\_role\_table  AFTER INSERT ON role\_table  FOR EACH ROW  DECLARE  v\_xml XMLType;  BEGIN  SELECT XMLType(  '<role>' ||  '<role\_id>' || :new.role\_id || '</role\_id>' ||  '<role\_name>' || :new.role\_name || '</role\_name>' ||  '</role>'  )  INTO v\_xml  FROM dual;  INSERT INTO role\_table\_XML (xml\_data) VALUES (v\_xml);  END; / |

Листинг 4.15 – Триггер trg\_ role\_table \_insert

Для экспорта и сохранения данных в xml формате на сервере базы данных была разработана процедура SAVE\_role\_table\_XML, которая для записи в файл использует пакет функций UTL\_FILE. Процедура представлена на листинге 4.16.

|  |
| --- |
| CREATE OR REPLACE PROCEDURE SAVE\_role\_table\_XML AS  v\_xml XMLType;  v\_file UTL\_FILE.file\_type;  BEGIN  v\_file := UTL\_FILE.FOPEN('XML\_DIR', 'role.xml', 'W');  UTL\_FILE.PUT\_LINE(v\_file, '<?xml version="1.0" encoding="utf-8"?>');  UTL\_FILE.PUT\_LINE(v\_file, '<roles>');    FOR rec IN (SELECT xml\_data FROM role\_table\_XML)  LOOP  v\_xml := rec.xml\_data;  UTL\_FILE.put\_line(v\_file, v\_xml.getClobVal());  END LOOP;    UTL\_FILE.PUT\_LINE(v\_file, '</roles>');  UTL\_FILE.FCLOSE(v\_file);  END; / |

Листинг 4.16 – Процедура SAVE\_role\_table\_XML

Таким образом, пользователи базы данных могут легко экспортировать данные и сохранять данные в формате XML, что делает управление базой данных более удобным и эффективным.

## Вывод

В данном разделе были рассмотрены основные этапы установки, настройки и использования Oracle 12c. Были описаны процедуры создания таблиц, ролей для разграничения доступа к базе данных и пакетов процедур для выполнения различных операций с данными.

Были также представлены процедуры экспорта данных в формате Xml, а также были проведены тесты производительности базы данных. В результате тестирования было установлено, что база данных Oracle 12c обладает высокой производительностью и способна быстро обрабатывать запросы на получение данных.

Итак, можно сделать вывод, что Oracle 12c является мощной и надежной системой управления базами данных, которая может быть использована для хранения и обработки больших объемов данных. Правильная установка и настройка Oracle 12c а также оптимизация запросов, позволят обеспечить высокую производительность и эффективность работы с базой данных.

5 Тестирование

## 5.1 Тестирование производительности базы данных

Тестирование производительности является важным этапом разработки, поскольку позволяет определить, насколько хорошо база данных может обрабатывать запросы и как быстро она может возвращать результаты.

Для тестирования производительности базы данных была выбрана таблица Products, содержащая 100000 строк. Для получения выборки данных использовался запрос, который представлен на листинге 5.1.

|  |
| --- |
| select \* from role\_table where role\_name='Admin' |

Листинг 5.1 – Запрос к таблице role\_table

Запрос к таблице Products показывает, что его стоимость выполнения с Total Cost равным 103 говорит о том, что выполнение запроса требует значительных затрат времени и ресурсов, особенно при сканировании всей таблицы и применении фильтра. Время выполнения запроса составляет 0,89 с. Результаты запроса будут представлены на рисунке 5.2.

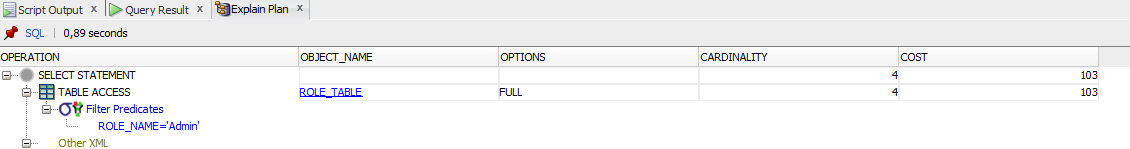


Рисунок 5.2 – Результат выполнения запроса без индекса idx\_product\_name

Для ускорения данного процесса можно создать индекс на поле product\_name, так как именно по этому полю выполняется фильтрация. После создания индекса, можно повторить запрос и сравнить стоимость с предыдущим запросом. Результат будет представлен на рисунке 5.3

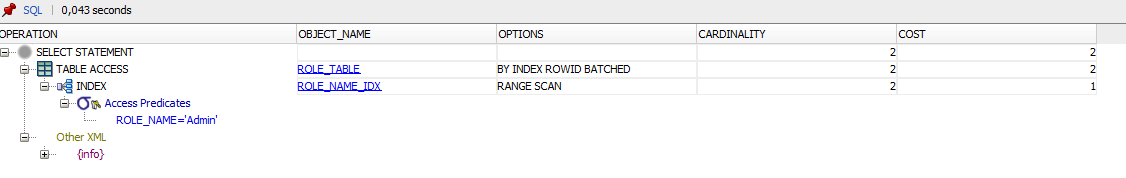


Рисунок 5.3 – Результат выполнения запроса с индексом idx\_product\_name

После создания индекса на поле role\_name запрос к таблице Role\_table имеет стоимость выполнения с Total Cost равным 2, что является значительным улучшением по сравнению с предыдущим результатом, составляющим 238. Время выполнения запроса сократилось до 0,043 с. Результаты запроса также будут представлены на рисунке 5.3.

Тестирование показало, что создание индекса на поле role\_name значительно улучшило производительность запроса к таблице Role\_table. Стоимость выполнения запроса уменьшилась с 103 до 2. При работе с большими объемами данных каждая миллисекунда может иметь значение. Поэтому создание индексов на полях, по которым выполняются частые запросы, может значительно повысить производительность базы данных.

## 5.2 Вывод

В данном разделе было рассмотрено важное понятие тестирования производительности базы данных. Для проведения тестирования была выбрана таблица Role\_table с большим количеством данных. Проведенный тест показал, что создание индекса на поле role\_name значительно улучшило производительность запроса к таблице Role\_table, сократив время выполнения запроса и уменьшив стоимость выполнения запроса. Выводом является то, что создание индексов на полях, по которым выполняются частые запросы, может значительно повысить производительность базы данных, что особенно важно при работе с большими объемами данных.

# Заключение

База данных является ключевым элементом любой современной организации, обеспечивая надежное хранение и управление информацией. В данной работе была поставлена задача разработки базы данных для электронной музыкальной площадки с использованием технологии диагностики и мониторинга.

На начальном этапе проектирования был проведён предметный анализ области аукционных площадок, были выделены некоторые основные объекты необходимые для корректной работы проекта.

Далее была реализована инфраструктура базы данных, в частности табличные пространства, профили безопасности, роли, пользователи, привелегии. В контексте этой инфраструктуры были спроектированы и созданы на основе ранее проведённого анализа. Также были созданы вспомогательные элементы для функционирования базы данных, такие как индексы, триггеры, представления, функции и процедуры.

Тестирование базы данных было проведено при использовании большого объема данных, и результаты были положительными. Были реализованы процедуры для импорта и экспорта данных в формат XML, что обеспечило удобство использования и управления данными, а также их сохранность.

В контексте базы данных были полностью реализованы поставленные задачи, такие как:

* Управление информацией о композициях.
* Управление информацией о исполнителях.
* Анализ данных о прослушиваемых композициях.
* Управление избранными композициях.

В целом, результаты работы говорят о том, что база данных полностью соответствует поставленным требованиям и может успешно использоваться в рамках музыкальной площадки.

# Список литературных источников

1. Чарльз Ким и Сэм М. Аллапати. Oracle Database 11g [Электронный ресурс] / – Дата доступа: 18.04.2023.

2. Стивен Фейерштейн. Oracle PL/SQL для профессионалов [Электронный ресурс] / – Дата доступа: 18.04.2023.

3 Use SQL Developer to Monitor Database Perfomance [Электронный ресурс] / Режим доступа: <https://docs.oracle.com/en/cloud/paas/exadata-express-cloud/csdbp/use-sql-developer-monitor-database-performance-ee> – Дата доступа: 18.04.2023.

# Приложение А

|  |
| --- |
| -- PROFILE PROFILE\_CLIENT --  CREATE PROFILE ROLE\_USER LIMIT  PASSWORD\_LIFE\_TIME UNLIMITED  SESSIONS\_PER\_USER 3  FAILED\_LOGIN\_ATTEMPTS 5  PASSWORD\_LOCK\_TIME 1  PASSWORD\_REUSE\_TIME 365  PASSWORD\_GRACE\_TIME DEFAULT  CONNECT\_TIME 120  IDLE\_TIME 30;  -- PROFILE PROFILE\_ANALYST --  CREATE PROFILE ROLE\_AUTHOR LIMIT  PASSWORD\_LIFE\_TIME 30  SESSIONS\_PER\_USER 2  FAILED\_LOGIN\_ATTEMPTS 3  PASSWORD\_LOCK\_TIME 30  PASSWORD\_REUSE\_TIME UNLIMITED  PASSWORD\_GRACE\_TIME 7  CONNECT\_TIME 300  IDLE\_TIME 60;  -- PROFILE ROLE\_ADMIN --  CREATE PROFILE PROFILE\_ADMIN LIMIT  PASSWORD\_LIFE\_TIME 1  SESSIONS\_PER\_USER 1  FAILED\_LOGIN\_ATTEMPTS 2  PASSWORD\_LOCK\_TIME UNLIMITED  PASSWORD\_REUSE\_TIME 30  PASSWORD\_GRACE\_TIME 1  CONNECT\_TIME UNLIMITED  IDLE\_TIME UNLIMITED;  CREATE TABLESPACE TS\_MAIN  DATAFILE 'C:\app\Kursach\TS\_MAIN.dbf'  SIZE 500M  AUTOEXTEND ON  EXTENT MANAGEMENT LOCAL;  CREATE TEMPORARY TABLESPACE TS\_TEMP  TEMPFILE 'C:\app\Kursach\TS\_TEMP.dbf'  SIZE 100M  AUTOEXTEND ON  EXTENT MANAGEMENT LOCAL;  -------------------------LISTENED\_TRACKS-------------------------  CREATE TABLE listened\_tracks\_table (  listened\_tracks\_id NUMBER(10) GENERATED AS IDENTITY(START WITH 1 INCREMENT BY 1),  listened\_tracks\_user NUMBER(10) NOT NULL,  listened\_tracks\_song NUMBER(10) NOT NULL,  listened\_tracks\_date DATE NOT NULL,  CONSTRAINT listened\_tracks\_pk PRIMARY KEY (listened\_tracks\_id),  CONSTRAINT listened\_tracks\_user\_fk FOREIGN KEY (listened\_tracks\_user) REFERENCES user\_table(user\_id),  CONSTRAINT listened\_tracks\_song\_fk FOREIGN KEY (listened\_tracks\_song) REFERENCES song\_table(song\_id)  );  -------------------------ROLE-------------------------  CREATE TABLE role\_table (  role\_id NUMBER(10) GENERATED AS IDENTITY(START WITH 1 INCREMENT BY 1),  role\_name VARCHAR2(30) NOT NULL,  CONSTRAINT role\_pk PRIMARY KEY (role\_id)  );  INSERT INTO role\_table(role\_name) VALUES('USER');  INSERT INTO role\_table(role\_name) VALUES('OWNER');  SELECT \* FROM role\_table;  DROP TABLE role\_table;  -------------------------USER-------------------------  CREATE TABLE user\_table (  user\_id NUMBER(10) GENERATED AS IDENTITY(START WITH 1 INCREMENT BY 1),  user\_login VARCHAR2(30) NOT NULL,  user\_password VARCHAR2(200) NOT NULL,  user\_role NUMBER(10) NOT NULL,  CONSTRAINT user\_pk PRIMARY KEY (user\_id),  CONSTRAINT user\_role\_fk FOREIGN KEY (user\_role) REFERENCES role\_table(role\_id)  );  SELECT \* FROM user\_table;  select count(\*) from user\_table;  DROP TABLE user\_table;  -------------------------ARTIST-------------------------  CREATE TABLE artist\_table (  artist\_id NUMBER(10) GENERATED AS IDENTITY(START WITH 1 INCREMENT BY 1),  artist\_name VARCHAR2(30) NOT NULL,  CONSTRAINT artist\_pk PRIMARY KEY (artist\_id)  );  INSERT INTO artist\_table(artist\_name) VALUES('Polyphia');  INSERT INTO artist\_table(artist\_name) VALUES('Radiohead');  INSERT INTO artist\_table(artist\_name) VALUES('Arctic Monkeys');  INSERT INTO artist\_table(artist\_name) VALUES('Pink Floyd');  INSERT INTO artist\_table(artist\_name) VALUES('Placebo');  SELECT \* FROM artist\_table;  DROP TABLE artist\_table;  -------------------------ALBUM-------------------------  CREATE TABLE album\_table (  album\_id NUMBER(10) GENERATED AS IDENTITY(START WITH 1 INCREMENT BY 1),  album\_artist NUMBER(10) NOT NULL,  album\_name VARCHAR2(30) NOT NULL,  album\_released NUMBER(10) NOT NULL,  album\_image BLOB ,  image\_link BFILE,  picture ORDImage NOT NULL,  CONSTRAINT album\_pk PRIMARY KEY (album\_id),  CONSTRAINT album\_artist\_fk FOREIGN KEY (album\_artist) REFERENCES artist\_table(artist\_id)  );  alter table album\_table add constraint fk\_artist\_album foreign key (album\_artist) references artist\_table (artist\_id) on delete cascade;  INSERT INTO album\_table(album\_artist, album\_name, album\_released) VALUES(1, 'New Levels New Devils', 2018);  INSERT INTO album\_table(album\_artist, album\_name, album\_released) VALUES(2, 'OK Computer', 1997);  INSERT INTO album\_table(album\_artist, album\_name, album\_released) VALUES(2, 'In Rainbows', 2007);  INSERT INTO album\_table(album\_artist, album\_name, album\_released) VALUES(2, 'Kid A', 2000);  INSERT INTO album\_table(album\_artist, album\_name, album\_released) VALUES(3, 'AM', 2013);  INSERT INTO album\_table(album\_artist, album\_name, album\_released) VALUES(3, 'Favourite Worst Nightmare', 2007);  INSERT INTO album\_table(album\_artist, album\_name, album\_released) VALUES(4, 'The Wall', 1979);  INSERT INTO album\_table(album\_artist, album\_name, album\_released) VALUES(4, 'Wish You Were Here', 1975);  INSERT INTO album\_table(album\_artist, album\_name, album\_released) VALUES(5, 'Loud Like Love', 2013);  INSERT INTO album\_table(album\_artist, album\_name, album\_released) VALUES(5, 'Meds', 2013);  SELECT \* FROM album\_table;  delete from album\_table;  commit;  DROP TABLE album\_table;  -------------------------SONG-------------------------  CREATE TABLE song\_table (  song\_id NUMBER(10) GENERATED AS IDENTITY(START WITH 1 INCREMENT BY 1),  song\_artist NUMBER(10) NOT NULL,  song\_album NUMBER(10) NOT NULL,  song\_name VARCHAR2(30) NOT NULL,  song\_link BFILE,  CONSTRAINT song\_pk PRIMARY KEY (song\_id),  CONSTRAINT song\_artist\_fk FOREIGN KEY (song\_artist) REFERENCES artist\_table(artist\_id),  CONSTRAINT song\_album\_fk FOREIGN KEY (song\_album) REFERENCES album\_table(album\_id)  );  /  SELECT \* FROM song\_table;  delete from song\_table;  commit;  DROP TABLE song\_table;  ------------------------ SAVED  CREATE TABLE save\_table (  saved\_id NUMBER(10) GENERATED AS IDENTITY(START WITH 1 INCREMENT BY 1),  saved\_user NUMBER(10) NOT NULL,  saved\_song NUMBER(10) NOT NULL,  CONSTRAINT saved\_pk PRIMARY KEY (saved\_id),  CONSTRAINT saved\_user\_fk FOREIGN KEY (saved\_user) REFERENCES user\_table(user\_id),  CONSTRAINT saved\_song\_fk FOREIGN KEY (saved\_song) REFERENCES song\_table(song\_id)  );  -------------------------PLAYLIST-------------------------  CREATE TABLE playlist\_table (  playlist\_id NUMBER(10) GENERATED AS IDENTITY(START WITH 1 INCREMENT BY 1),  playlist\_name VARCHAR2(30) NOT NULL,  playlist\_user NUMBER(10) NOT NULL,  CONSTRAINT playlist\_pk PRIMARY KEY (playlist\_id),  CONSTRAINT playlist\_user\_fk FOREIGN KEY (playlist\_user) REFERENCES user\_table(user\_id)  );  SELECT \* FROM playlist\_table;  delete from playlist\_table;  commit;  DROP TABLE playlist\_table;  -------------------------PLAYLIST\_SONG-------------------------  CREATE TABLE playlist\_song\_table (  playlist\_song\_id NUMBER(10) GENERATED AS IDENTITY(START WITH 1 INCREMENT BY 1),  playlist\_song\_playlist NUMBER(10) NOT NULL,  playlist\_song\_song NUMBER(10) NOT NULL,  CONSTRAINT playlist\_song\_pk PRIMARY KEY (playlist\_song\_id),  CONSTRAINT playlist\_song\_playlist\_fk FOREIGN KEY (playlist\_song\_playlist) REFERENCES playlist\_table(playlist\_id),  CONSTRAINT playlist\_song\_song\_fk FOREIGN KEY (playlist\_song\_song) REFERENCES song\_table(song\_id)  );  SELECT \* FROM playlist\_song\_table;  delete from playlist\_song\_table;  commit;  DROP TABLE playlist\_song\_table;  ------------------- SYNONYM -------------------  CREATE SYNONYM role\_t FOR role\_table;  CREATE SYNONYM user\_t FOR user\_table;  CREATE SYNONYM artist\_t FOR artist\_table;  CREATE SYNONYM album\_t FOR album\_table;  CREATE SYNONYM song\_t FOR song\_table;  CREATE SYNONYM saved\_t FOR saved\_table;  CREATE SYNONYM playlist\_t FOR playlist\_table;  CREATE SYNONYM playlist\_song\_t FOR playlist\_song\_table;  CREATE SYNONYM listened\_t FOR listened\_tracks\_table;  DROP PUBLIC SYNONYM role\_t;  DROP PUBLIC SYNONYM user\_t;  DROP PUBLIC SYNONYM artist\_t;  DROP PUBLIC SYNONYM album\_t;  DROP PUBLIC SYNONYM song\_t;  DROP PUBLIC SYNONYM saved\_t;  DROP PUBLIC SYNONYM playlist\_t;  DROP PUBLIC SYNONYM playlist\_song\_t;  DROP PUBLIC SYNONYM listened\_t;  ------------------- MATERIALIZED VIEW -------------------  CREATE MATERIALIZED VIEW playlist\_view  BUILD IMMEDIATE  REFRESH ON DEMAND  ENABLE QUERY REWRITE  AS SELECT \* FROM playlist\_table;  CREATE MATERIALIZED VIEW playlist\_song\_view  BUILD IMMEDIATE  REFRESH ON DEMAND  ENABLE QUERY REWRITE  AS SELECT \* FROM playlist\_song\_table;  CREATE MATERIALIZED VIEW listened\_tracks\_view  BUILD IMMEDIATE  REFRESH ON DEMAND  ENABLE QUERY REWRITE  AS SELECT \* FROM listened\_tracks\_table;  DROP MATERIALIZED VIEW playlist\_view;  DROP MATERIALIZED VIEW playlist\_song\_view;  DROP MATERIALIZED VIEW listened\_tracks\_table;  ------------------- INDEX -------------------  CREATE INDEX role\_name\_idx ON role\_table (role\_name);  select \* from role\_table where role\_name='Admin'  drop  CREATE INDEX user\_login\_idx ON user\_table (user\_login);  CREATE INDEX user\_role\_idx ON user\_table (user\_role);  CREATE INDEX artist\_name\_idx ON artist\_table (artist\_name);  CREATE INDEX album\_name\_idx ON album\_table (album\_name);  CREATE INDEX album\_artist\_idx ON album\_table (album\_artist);  CREATE INDEX song\_name\_idx ON song\_table (song\_name);  CREATE INDEX song\_artist\_idx ON song\_table (song\_artist);  CREATE INDEX album\_name\_idx ON song\_table (album\_name);  CREATE INDEX saved\_user\_idx ON saved\_table (saved\_user);  CREATE INDEX saved\_song\_idx ON saved\_table (saved\_song);  CREATE INDEX playlist\_name\_idx ON playlist\_table (playlist\_name);  CREATE INDEX playlist\_user\_idx ON playlist\_table (playlist\_user);  CREATE INDEX playlist\_song\_playlist\_idx ON playlist\_song\_table (playlist\_song\_playlist);  CREATE INDEX playlist\_song\_song\_idx ON playlist\_song\_table (playlist\_song\_song);  CREATE INDEX listened\_tracks\_user\_idx ON listened\_tracks\_table (listened\_tracks\_user);  CREATE INDEX listened\_tracks\_song\_idx ON listened\_tracks\_table (listened\_tracks\_song);  CREATE INDEX listened\_tracks\_date\_idx ON listened\_tracks\_table (listened\_tracks\_date);  ------------------- VIEW -------------------  CREATE VIEW role\_view AS SELECT role\_id, role\_name FROM role\_table;  CREATE VIEW user\_view AS SELECT user\_id, user\_login, user\_role FROM user\_table;  CREATE VIEW artist\_view AS SELECT artist\_id, artist\_name FROM artist\_table;  CREATE VIEW album\_view AS SELECT album\_id, album\_artist, album\_name, album\_released FROM album\_table;  CREATE VIEW song\_view AS SELECT song\_id, song\_artist, song\_album, song\_name FROM song\_table;  CREATE VIEW saved\_view AS SELECT saved\_id, saved\_user, saved\_song FROM saved\_table;  CREATE VIEW playlist\_view AS SELECT playlist\_id, playlist\_name, playlist\_user FROM playlist\_table;  CREATE VIEW playlist\_song\_view AS SELECT playlist\_song\_id, playlist\_song\_playlist, playlist\_song\_song FROM playlist\_song\_table;  CREATE VIEW listened\_tracks\_view AS SELECT listened\_tracks\_id, listened\_tracks\_user, listened\_tracks\_song, listened\_tracks\_date FROM listened\_tracks\_table;  ------------------- SYNONYM VIEW -------------------  CREATE SYNONYM role\_synonym FOR role\_view;  CREATE SYNONYM user\_synonym FOR user\_view;  CREATE SYNONYM artist\_synonym FOR artist\_view;  CREATE SYNONYM album\_synonym FOR album\_view;  CREATE SYNONYM song\_synonym FOR song\_view;  CREATE SYNONYM saved\_synonym FOR saved\_view;  CREATE SYNONYM playlist\_synonym FOR playlist\_view;  CREATE SYNONYM playlist\_song\_synonym FOR playlist\_song\_view;  CREATE SYNONYM listened\_tracks\_synonym FOR listened\_tracks\_view;  /  ------------------- PROCEDURE INSERT -------------------  /  CREATE OR REPLACE PROCEDURE INSERT\_ALBUM(pAlbum\_artist NUMBER,pAlbum\_name VARCHAR2,  pAlbum\_released NUMBER,pOraDir VARCHAR2,pFileName VARCHAR2) IS  l\_bfile BFILE;  l\_blob BLOB;  l\_dest\_offset INTEGER := 1;  l\_src\_offset INTEGER := 1;  l\_lobmaxsize CONSTANT INTEGER := DBMS\_LOB.LOBMAXSIZE;  l\_img ORDImage;  BEGIN  l\_bfile := BFILENAME(pOraDir, pFileName);  l\_img := ORDSYS.ORDImage(l\_blob);  INSERT INTO album\_table(album\_id,album\_artist,album\_name,album\_released,album\_image,image\_link,picture)  VALUES(NULL,pAlbum\_artist,pAlbum\_name,pAlbum\_released,empty\_blob(),l\_bfile,l\_img)  RETURNING album\_image INTO l\_blob;  DBMS\_LOB.fileopen(l\_bfile, DBMS\_LOB.file\_readonly);  DBMS\_LOB.loadblobfromfile(l\_blob, l\_bfile, l\_lobmaxsize, l\_dest\_offset, l\_src\_offset);  DBMS\_LOB.fileclose(l\_bfile);  COMMIT;  EXCEPTION  WHEN OTHERS THEN  DBMS\_OUTPUT.PUT\_LINE('CODE ERROR: '||SQLCODE||' MESSAGE: '||SQLERRM);  END; |

Листинг 1 – infrastructure.sql

|  |
| --- |
|  |

Листинг 2 – triggers.sql

|  |
| --- |
| -------------------------------------  ----------- PROCEDURES ------------  -------------------------------------  ----insert  CREATE OR REPLACE PROCEDURE add\_product (  p\_id IN NUMBER,  p\_name IN VARCHAR2,  p\_desc IN VARCHAR2,  cat\_id IN NUMBER,  stock\_qty IN NUMBER,  p\_price IN NUMBER  ) AS  BEGIN  INSERT INTO Products (product\_id, product\_name, product\_description, category\_id, stock\_quantity, price)  VALUES (p\_id, p\_name, p\_desc, cat\_id, stock\_qty, p\_price);  END;  CREATE OR REPLACE PROCEDURE add\_order (  o\_id IN NUMBER,  o\_date IN DATE,  o\_total\_price IN NUMBER  ) AS  BEGIN  INSERT INTO Orders (order\_id, order\_date, total\_price)  VALUES (o\_id, o\_date, o\_total\_price);  END;  CREATE OR REPLACE PROCEDURE add\_order\_item (  oi\_id IN NUMBER,  o\_id IN NUMBER,  ab\_id IN NUMBER,  qty IN NUMBER,  price\_per\_item IN NUMBER  ) AS  BEGIN  INSERT INTO Order\_Items (order\_item\_id, order\_id, auction\_bid\_id, quantity, price\_per\_item)  VALUES (oi\_id, o\_id, ab\_id, qty, price\_per\_item);  END;  ----update  CREATE OR REPLACE PROCEDURE update\_product (  p\_id IN NUMBER,  p\_name IN VARCHAR2,  p\_desc IN VARCHAR2,  cat\_id IN NUMBER,  stock\_qty IN NUMBER,  p\_price IN NUMBER  ) AS  BEGIN  UPDATE Products  SET product\_name = p\_name,  product\_description = p\_desc,  category\_id = cat\_id,  stock\_quantity = stock\_qty,  price = p\_price  WHERE product\_id = p\_id;  END;  CREATE OR REPLACE PROCEDURE update\_order (  o\_id IN NUMBER,  o\_date IN DATE,  o\_total\_price IN NUMBER  ) AS  BEGIN  UPDATE Orders  SET order\_date = o\_date,  total\_price = o\_total\_price  WHERE order\_id = o\_id;  END;  CREATE OR REPLACE PROCEDURE update\_order\_item (  oi\_id IN NUMBER,  o\_id IN NUMBER,  ab\_id IN NUMBER,  qty IN NUMBER,  price\_per\_item IN NUMBER  ) AS  BEGIN  UPDATE Order\_Items  SET order\_id = o\_id,  auction\_bid\_id = ab\_id,  quantity = qty,  price\_per\_item = price\_per\_item  WHERE order\_item\_id = oi\_id;  END;  ----delete  CREATE OR REPLACE PROCEDURE delete\_product (p\_id IN NUMBER) AS  BEGIN  DELETE FROM Products WHERE product\_id = p\_id;  END;  CREATE OR REPLACE PROCEDURE delete\_order (o\_id IN NUMBER) AS  BEGIN  DELETE FROM Orders WHERE order\_id = o\_id;  END;  CREATE OR REPLACE PROCEDURE delete\_order\_item (oi\_id IN NUMBER) AS  BEGIN  DELETE FROM Order\_Items WHERE order\_item\_id = oi\_id;  END;  ------------------  ---- customer ----  ------------------  -- (размещение ставки клиентом)  CREATE OR REPLACE PROCEDURE place\_bid(  p\_customer\_id IN NUMBER,  p\_auction\_item\_id IN NUMBER,  p\_bid\_amount IN NUMBER  ) AS  v\_current\_bid\_amount NUMBER;  BEGIN  -- Получаем текущую максимальную ставку на данный товар  SELECT MAX(bid\_amount)  INTO v\_current\_bid\_amount  FROM Auction\_Bids  WHERE auction\_item\_id = p\_auction\_item\_id;    -- Проверяем, что новая ставка больше текущей максимальной ставки  IF p\_bid\_amount > v\_current\_bid\_amount THEN  -- Добавляем новую ставку в таблицу Auction\_Bids  INSERT INTO Auction\_Bids(auction\_bid\_id, auction\_item\_id, bidder\_id, bid\_amount, bid\_date, is\_win)  VALUES (auction\_bid\_id\_seq.NEXTVAL, p\_auction\_item\_id, p\_customer\_id, p\_bid\_amount, SYSDATE, 0);    COMMIT;  DBMS\_OUTPUT.PUT\_LINE('Your bid has been placed successfully.');  ELSE  DBMS\_OUTPUT.PUT\_LINE('Your bid amount must be greater than the current highest bid.');  END IF;  END place\_bid;  -- (получение списка всех выигранных ставок клиента)  CREATE OR REPLACE PROCEDURE get\_customer\_winning\_bids(  p\_customer\_id IN NUMBER,  p\_bids OUT SYS\_REFCURSOR  ) AS  BEGIN  OPEN p\_bids FOR  SELECT \*  FROM Auction\_Bids  WHERE bidder\_id = p\_customer\_id  AND is\_win = 1;  END get\_customer\_winning\_bids;  -- (получение списка всех ставок клиента)  CREATE OR REPLACE PROCEDURE get\_customer\_bids(  p\_customer\_id IN NUMBER,  p\_bids OUT SYS\_REFCURSOR  ) AS  BEGIN  OPEN p\_bids FOR  SELECT \*  FROM Auction\_Bids  WHERE bidder\_id = p\_customer\_id;  END get\_customer\_bids;  ----------------  ---- seller ----  ----------------  -- (добавление товара на аукцион)  CREATE OR REPLACE PROCEDURE add\_auction\_item(  p\_auction\_id IN NUMBER,  p\_product\_id IN NUMBER,  p\_starting\_price IN NUMBER  ) AS  BEGIN  -- Проверяем, что аукцион существует  IF NOT EXISTS(SELECT 1 FROM Auctions WHERE auction\_id = p\_auction\_id) THEN  DBMS\_OUTPUT.PUT\_LINE('Auction with ID ' || p\_auction\_id || ' does not exist.');  RETURN;  END IF;    -- Проверяем, что товар существует  IF NOT EXISTS(SELECT 1 FROM Products WHERE product\_id = p\_product\_id) THEN  DBMS\_OUTPUT.PUT\_LINE('Product with ID ' || p\_product\_id || ' does not exist.');  RETURN;  END IF;    -- Добавляем новый товар на аукцион  INSERT INTO Auction\_Items(auction\_item\_id, auction\_id, product\_id, starting\_price)  VALUES (auction\_item\_id\_seq.NEXTVAL, p\_auction\_id, p\_product\_id, p\_starting\_price);    COMMIT;  DBMS\_OUTPUT.PUT\_LINE('New auction item has been added successfully.');  END add\_auction\_item;  -- (определение выигрышной ставки и добавление её в заказы)  CREATE OR REPLACE PROCEDURE mark\_bid\_as\_winner(  p\_auction\_bid\_id IN NUMBER,  p\_quantity IN NUMBER  ) AS  v\_auction\_item\_id NUMBER;  v\_product\_id NUMBER;  v\_seller\_id NUMBER;  v\_bid\_amount NUMBER;  BEGIN  -- Получаем информацию о ставке  SELECT ai.auction\_item\_id, ai.product\_id, a.seller\_id, ab.bid\_amount  INTO v\_auction\_item\_id, v\_product\_id, v\_seller\_id, v\_bid\_amount  FROM Auction\_Bids ab  JOIN Auction\_Items ai ON ab.auction\_item\_id = ai.auction\_item\_id  JOIN Auctions a ON ai.auction\_id = a.auction\_id  WHERE ab.auction\_bid\_id = p\_auction\_bid\_id;    -- Помечаем ставку как выигрышную  UPDATE Auction\_Bids  SET is\_win = 1  WHERE auction\_bid\_id = p\_auction\_bid\_id;    -- Создаем новый заказ  INSERT INTO Orders(order\_id, order\_date, total\_price)  VALUES (order\_id\_seq.NEXTVAL, SYSDATE, v\_bid\_amount \* p\_quantity);    -- Добавляем запись в таблицу Order\_Items  INSERT INTO Order\_Items(order\_item\_id, order\_id, auction\_bid\_id, quantity, price\_per\_item)  VALUES (order\_item\_id\_seq.NEXTVAL, order\_id\_seq.CURRVAL, p\_auction\_bid\_id, p\_quantity, v\_bid\_amount);    -- Обновляем количество товара на складе  UPDATE Products  SET stock\_quantity = stock\_quantity - p\_quantity  WHERE product\_id = v\_product\_id;    -- Удаляем товар из таблицы Products, если его количество на складе стало равным 0  IF (SELECT stock\_quantity FROM Products WHERE product\_id = v\_product\_id) = 0 THEN  DELETE FROM Products WHERE product\_id = v\_product\_id;  END IF;    COMMIT;  DBMS\_OUTPUT.PUT\_LINE('Bid with ID ' || p\_auction\_bid\_id || ' has been marked as winner and a new order has been created.');  END mark\_bid\_as\_winner;  -- (получение списка всех товаров продавца)  CREATE OR REPLACE PROCEDURE get\_seller\_orders(  p\_seller\_id IN NUMBER,  p\_orders OUT SYS\_REFCURSOR  ) AS  BEGIN  OPEN p\_orders FOR  SELECT \*  FROM Orders o  JOIN Order\_Items oi ON o.order\_id = oi.order\_id  JOIN Auction\_Bids ab ON oi.auction\_bid\_id = ab.auction\_bid\_id  JOIN Auction\_Items ai ON ab.auction\_item\_id = ai.auction\_item\_id  JOIN Auctions a ON ai.auction\_id = a.auction\_id  WHERE a.seller\_id = p\_seller\_id;  END get\_seller\_orders;  -----------------  ---- manager ----  -----------------  -- (обновление информации о товаре)  CREATE OR REPLACE PROCEDURE update\_product(  p\_product\_id IN NUMBER,  p\_product\_name IN VARCHAR2,  p\_product\_description IN VARCHAR2,  p\_category\_id IN NUMBER,  p\_stock\_quantity IN NUMBER,  p\_price IN NUMBER  ) AS  BEGIN  -- Обновляем информацию о товаре  UPDATE Products  SET product\_name = p\_product\_name,  product\_description = p\_product\_description,  category\_id = p\_category\_id,  stock\_quantity = p\_stock\_quantity,  price = p\_price  WHERE product\_id = p\_product\_id;    COMMIT;  DBMS\_OUTPUT.PUT\_LINE('Product with ID ' || p\_product\_id || ' has been updated.');  END update\_product;  -- (удаление заказа)  CREATE OR REPLACE PROCEDURE delete\_order(  p\_order\_id IN NUMBER  ) AS  BEGIN  -- Удаляем запись из таблицы Order\_Items  DELETE FROM Order\_Items WHERE order\_id = p\_order\_id;    -- Удаляем запись из таблицы Orders  DELETE FROM Orders WHERE order\_id = p\_order\_id;    COMMIT;  DBMS\_OUTPUT.PUT\_LINE('Order with ID ' || p\_order\_id || ' has been deleted.');  END delete\_order;  --------------------  ------- XML --------  --------------------  -- products  CREATE OR REPLACE PROCEDURE SAVE\_PRODUCTS\_XML AS  v\_xml XMLType;  v\_file UTL\_FILE.file\_type;  BEGIN  v\_file := UTL\_FILE.FOPEN('XML\_DIR', 'products.xml', 'W');  UTL\_FILE.PUT\_LINE(v\_file, '<?xml version="1.0" encoding="utf-8"?>');  UTL\_FILE.PUT\_LINE(v\_file, '<products>');    FOR rec IN (SELECT xml\_data FROM Products\_XML)  LOOP  v\_xml := rec.xml\_data;  UTL\_FILE.put\_line(v\_file, v\_xml.getClobVal());  END LOOP;    UTL\_FILE.PUT\_LINE(v\_file, '</products>');  UTL\_FILE.FCLOSE(v\_file);  END;  /  --orders  CREATE OR REPLACE PROCEDURE SAVE\_ORDERS\_XML AS  v\_xml XMLType;  v\_file UTL\_FILE.file\_type;  BEGIN  v\_file := UTL\_FILE.FOPEN('XML\_DIR', 'orders.xml', 'W');  UTL\_FILE.PUT\_LINE(v\_file, '<?xml version="1.0" encoding="utf-8"?>');  UTL\_FILE.PUT\_LINE(v\_file, '<orders>');    FOR rec IN (SELECT xml\_data FROM Orders\_XML)  LOOP  v\_xml := rec.xml\_data;  UTL\_FILE.put\_line(v\_file, v\_xml.getClobVal());  END LOOP;    UTL\_FILE.PUT\_LINE(v\_file, '</orders>');  UTL\_FILE.FCLOSE(v\_file);  END;  /  --order\_items  CREATE OR REPLACE PROCEDURE SAVE\_ORDER\_ITEMS\_XML AS  v\_xml XMLType;  v\_file UTL\_FILE.file\_type;  BEGIN  v\_file := UTL\_FILE.FOPEN('XML\_DIR', 'order\_items.xml', 'W');  UTL\_FILE.PUT\_LINE(v\_file, '<?xml version="1.0" encoding="utf-8"?>');  UTL\_FILE.PUT\_LINE(v\_file, '<order\_items>');    FOR rec IN (SELECT xml\_data FROM Order\_Items\_XML)  LOOP  v\_xml := rec.xml\_data;  UTL\_FILE.put\_line(v\_file, v\_xml.getClobVal());  END LOOP;    UTL\_FILE.PUT\_LINE(v\_file, '</order\_items>');  UTL\_FILE.FCLOSE(v\_file);  END;  /  --employees  CREATE OR REPLACE PROCEDURE SAVE\_EMPLOYEES\_XML AS  v\_xml XMLType;  v\_file UTL\_FILE.file\_type;  BEGIN  v\_file := UTL\_FILE.FOPEN('XML\_DIR', 'employees.xml', 'W');  UTL\_FILE.PUT\_LINE(v\_file, '<?xml version="1.0" encoding="utf-8"?>');  UTL\_FILE.PUT\_LINE(v\_file, '<employees>');    FOR rec IN (SELECT xml\_data FROM Employees\_XML)  LOOP  v\_xml := rec.xml\_data;  UTL\_FILE.put\_line(v\_file, v\_xml.getClobVal());  END LOOP;    UTL\_FILE.PUT\_LINE(v\_file, '</employees>');  UTL\_FILE.FCLOSE(v\_file);  END;  /  --customers  CREATE OR REPLACE PROCEDURE SAVE\_CUSTOMERS\_XML AS  v\_xml XMLType;  v\_file UTL\_FILE.file\_type;  BEGIN  v\_file := UTL\_FILE.FOPEN('XML\_DIR', 'customers.xml', 'W');  UTL\_FILE.PUT\_LINE(v\_file, '<?xml version="1.0" encoding="utf-8"?>');  UTL\_FILE.PUT\_LINE(v\_file, '<customers>');    FOR rec IN (SELECT xml\_data FROM Customers\_XML)  LOOP  v\_xml := rec.xml\_data;  UTL\_FILE.put\_line(v\_file, v\_xml.getClobVal());  END LOOP;    UTL\_FILE.PUT\_LINE(v\_file, '</customers>');  UTL\_FILE.FCLOSE(v\_file);  END;  /  --categories  CREATE OR REPLACE PROCEDURE SAVE\_CATEGORIES\_XML AS  v\_xml XMLType;  v\_file UTL\_FILE.file\_type;  BEGIN  v\_file := UTL\_FILE.FOPEN('XML\_DIR', 'categories.xml', 'W');  UTL\_FILE.PUT\_LINE(v\_file, '<?xml version="1.0" encoding="utf-8"?>');  UTL\_FILE.PUT\_LINE(v\_file, '<categories>');    FOR rec IN (SELECT xml\_data FROM Categories\_XML)  LOOP  v\_xml := rec.xml\_data;  UTL\_FILE.put\_line(v\_file, v\_xml.getClobVal());  END LOOP;    UTL\_FILE.PUT\_LINE(v\_file, '</categories>');  UTL\_FILE.FCLOSE(v\_file);  END;  /  --auctions  CREATE OR REPLACE PROCEDURE SAVE\_AUCTIONS\_XML AS  v\_xml XMLType;  v\_file UTL\_FILE.file\_type;  BEGIN  v\_file := UTL\_FILE.FOPEN('XML\_DIR', 'auctions.xml', 'W');  UTL\_FILE.PUT\_LINE(v\_file, '<?xml version="1.0" encoding="utf-8"?>');  UTL\_FILE.PUT\_LINE(v\_file, '<auctions>');    FOR rec IN (SELECT xml\_data FROM Auctions\_XML)  LOOP  v\_xml := rec.xml\_data;  UTL\_FILE.put\_line(v\_file, v\_xml.getClobVal());  END LOOP;    UTL\_FILE.PUT\_LINE(v\_file, '</auctions>');  UTL\_FILE.FCLOSE(v\_file);  END;  /  --auction\_items  CREATE OR REPLACE PROCEDURE SAVE\_AUCTION\_ITEMS\_XML AS  v\_xml XMLType;  v\_file UTL\_FILE.file\_type;  BEGIN  v\_file := UTL\_FILE.FOPEN('XML\_DIR', 'auction\_items.xml', 'W');  UTL\_FILE.PUT\_LINE(v\_file, '<?xml version="1.0" encoding="utf-8"?>');  UTL\_FILE.PUT\_LINE(v\_file, '<auction\_items>');    FOR rec IN (SELECT xml\_data FROM Auction\_Items\_XML)  LOOP  v\_xml := rec.xml\_data;  UTL\_FILE.put\_line(v\_file, v\_xml.getClobVal());  END LOOP;    UTL\_FILE.PUT\_LINE(v\_file, '</auction\_items>');  UTL\_FILE.FCLOSE(v\_file);  END;  /  --auction\_bids  CREATE OR REPLACE PROCEDURE SAVE\_AUCTION\_BIDS\_XML AS  v\_xml XMLType;  v\_file UTL\_FILE.file\_type;  BEGIN  v\_file := UTL\_FILE.FOPEN('XML\_DIR', 'auction\_bids.xml', 'W');  UTL\_FILE.PUT\_LINE(v\_file, '<?xml version="1.0" encoding="utf-8"?>');  UTL\_FILE.PUT\_LINE(v\_file, '<auction\_bids>');    FOR rec IN (SELECT xml\_data FROM Auction\_Bids\_XML)  LOOP  v\_xml := rec.xml\_data;  UTL\_FILE.put\_line(v\_file, v\_xml.getClobVal());  END LOOP;    UTL\_FILE.PUT\_LINE(v\_file, '</auction\_bids>');  UTL\_FILE.FCLOSE(v\_file);  END;  / |

Листинг 3 – procedures.sql

|  |
| --- |
| -------------------------------------  ------------ FUNCTIONS ------------  -------------------------------------  -- (получение информации о товаре по ID)  CREATE OR REPLACE FUNCTION get\_product\_info\_by\_id(  p\_product\_id IN NUMBER  ) RETURN Products%ROWTYPE AS  v\_product\_info Products%ROWTYPE;  BEGIN  -- Получаем информацию о товаре по его ID  SELECT \*  INTO v\_product\_info  FROM Products  WHERE product\_id = p\_product\_id;    RETURN v\_product\_info;  END;  -- (получение информации о товаре по ID)  CREATE OR REPLACE FUNCTION get\_order\_info\_by\_id(  p\_order\_id IN NUMBER  ) RETURN Orders%ROWTYPE AS  v\_order\_info Orders%ROWTYPE;  BEGIN  -- Получаем информацию о заказе по его ID  SELECT \*  INTO v\_order\_info  FROM Orders  WHERE order\_id = p\_order\_id;    RETURN v\_order\_info;  END;  -- (получение информации о клиенте по ID)  CREATE OR REPLACE FUNCTION get\_customer\_info\_by\_id(  p\_customer\_id IN NUMBER  ) RETURN Customers%ROWTYPE AS  v\_customer\_info Customers%ROWTYPE;  BEGIN  -- Получаем информацию о покупателе по ID  SELECT \*  INTO v\_customer\_info  FROM Customers  WHERE customer\_id = p\_customer\_id;    RETURN v\_customer\_info;  END;  -- (получение информации о сотруднике по ID)  CREATE OR REPLACE FUNCTION get\_employee\_info\_id(  p\_employee\_id IN NUMBER  ) RETURN Employees%ROWTYPE AS  v\_employee\_info Employees%ROWTYPE;  BEGIN  -- Получаем информацию о продавце по его ID  SELECT \*  INTO v\_employee\_info  FROM Employees  WHERE employee\_id = p\_employee\_id;    RETURN v\_employee\_info;  END;  -- (получение информации о аукционе по ID)  CREATE OR REPLACE FUNCTION get\_auction\_bid\_info\_by\_id(  p\_auction\_bid\_id IN NUMBER  ) RETURN Auction\_Bids%ROWTYPE AS  v\_auction\_bid\_info Auction\_Bids%ROWTYPE;  BEGIN  -- Получаем информацию о заявке на участие в аукционе по ее ID  SELECT \*  INTO v\_auction\_bid\_info  FROM Auction\_Bids  WHERE auction\_bid\_id = p\_auction\_bid\_id;    RETURN v\_auction\_bid\_info;  END; |

Листинг 4 – functions.sql

|  |
| --- |
| -------------------------------------  ------------- INSERTS -------------  -------------------------------------  CREATE OR REPLACE PROCEDURE insert\_products AS  BEGIN  FOR i IN 1..100000 LOOP  INSERT INTO Products (  product\_id,  product\_name,  product\_description,  category\_id,  stock\_quantity,  price  ) VALUES (  i,  'Product ' || i,  'Product ' || i || ' description',  TRUNC(DBMS\_RANDOM.VALUE(1, 10)),  TRUNC(DBMS\_RANDOM.VALUE(1, 100)),  TRUNC(DBMS\_RANDOM.VALUE(1, 1000))  );  END LOOP;  COMMIT;  END;  CREATE OR REPLACE PROCEDURE insert\_orders AS  BEGIN  FOR i IN 1..10000 LOOP  INSERT INTO Orders (  order\_id,  order\_date,  total\_price  ) VALUES (  i,  SYSDATE,  TRUNC(DBMS\_RANDOM.VALUE(1, 10000))  );  END LOOP;  COMMIT;  END;  CREATE OR REPLACE PROCEDURE insert\_order\_items AS  BEGIN  FOR i IN 1..100000 LOOP  INSERT INTO Order\_Items (  order\_item\_id,  order\_id,  auction\_bid\_id,  quantity,  price\_per\_item  ) VALUES (  i,  TRUNC(DBMS\_RANDOM.VALUE(1, 100000)),  TRUNC(DBMS\_RANDOM.VALUE(1, 100000)),  TRUNC(DBMS\_RANDOM.VALUE(1, 10)),  TRUNC(DBMS\_RANDOM.VALUE(1, 100))  );  END LOOP;  COMMIT;  END;  CREATE OR REPLACE PROCEDURE insert\_customers AS  BEGIN  FOR i IN 1..10000 LOOP  INSERT INTO Customers (  customer\_id,  customer\_name,  customer\_email,  customer\_phone  ) VALUES (  i,  'Customer ' || i,  'customer' || i || '@example.com',  '555-555-' || LPAD(i, 4, '0')  );  END LOOP;  COMMIT;  END;  CREATE OR REPLACE PROCEDURE insert\_categories AS  BEGIN  FOR i IN 1..10 LOOP  INSERT INTO Categories (  category\_id,  category\_name,  category\_description  ) VALUES (  i,  'Category ' || i,  'Category ' || i || ' description'  );  END LOOP;  COMMIT;  END;  CREATE OR REPLACE PROCEDURE insert\_auctions AS  BEGIN  FOR i IN 1..10000 LOOP  INSERT INTO Auctions (  auction\_id,  auction\_name,  auction\_date,  auction\_time,  auction\_location,  seller\_id  ) VALUES (  i,  'Auction ' || i,  TRUNC(SYSDATE + DBMS\_RANDOM.VALUE(1, 30)),  LPAD(TRUNC(DBMS\_RANDOM.VALUE(0, 23.99)), 2, '0') || ':' || LPAD(TRUNC(DBMS\_RANDOM.VALUE(0, 59.99)), 2, '0'),  'Location ' || i,  TRUNC(DBMS\_RANDOM.VALUE(1, 100000))  );  END LOOP;  COMMIT;  END;  CREATE OR REPLACE PROCEDURE insert\_employees AS  BEGIN  FOR i IN 1..10000 LOOP  INSERT INTO Employees (  employee\_id,  employee\_name,  employee\_position,  employee\_contact\_info  ) VALUES (  i,  'Employee ' || i,  CASE WHEN MOD(i, 2) = 0 THEN 'seller' ELSE 'manager' END,  'employee' || i || '@example.com'  );  END LOOP;  COMMIT;  END;  CREATE OR REPLACE PROCEDURE insert\_auction\_items AS  BEGIN  FOR i IN 1..100000 LOOP  INSERT INTO Auction\_Items (  auction\_item\_id,  auction\_id,  product\_id,  starting\_price  ) VALUES (  i,  TRUNC(DBMS\_RANDOM.VALUE(1, 100000)),  TRUNC(DBMS\_RANDOM.VALUE(1, 100000)),  TRUNC(DBMS\_RANDOM.VALUE(1, 100))  );  END LOOP;  COMMIT;  END;  CREATE OR REPLACE PROCEDURE insert\_auction\_bids AS  BEGIN  FOR i IN 1..200000 LOOP  INSERT INTO Auction\_Bids (  auction\_bid\_id,  auction\_item\_id,  bidder\_id,  bid\_amount,  bid\_date,  is\_win  ) VALUES (  i,  TRUNC(DBMS\_RANDOM.VALUE(1, 100000)),  TRUNC(DBMS\_RANDOM.VALUE(1, 100000)),  TRUNC(DBMS\_RANDOM.VALUE(1, 1000)),  SYSDATE,  CASE WHEN MOD(i, 10) = 0 THEN 1 ELSE 0 END  );  END LOOP;  COMMIT;  END;  EXEC insert\_products;  EXEC insert\_orders;  EXEC insert\_order\_items;  EXEC insert\_employees;  EXEC insert\_categories;  EXEC insert\_customers;  EXEC insert\_auctions;  EXEC insert\_auction\_items;  EXEC insert\_auction\_bids; |

Листинг 5 – inserts.sql