Содержание

[Введение 3](#_Toc153923918)

[1 Постановка задача 4](#_Toc153923919)

[1.1 Программное средство «Spotify» 4](#_Toc153923920)

[1.2 Программное средство «Яндекс.Музыка» 5](#_Toc153923921)

[1.3 Программное средство «Apple Music» 5](#_Toc153923922)

[1.4 Определение цели и задачи 6](#_Toc153923923)

[2 Проектирование базы данных ………………………………………7](#_Toc153923924)

[2.1 Вывод по проектированию модели базы данных 7](#_Toc153923925)

[3 Разработка объектов базы данных 8](#_Toc153923926)

[3.1 Разработка таблиц базы данных 8](#_Toc153923927)

[3.2 Индексы 11](#_Toc153923928)

[3.3 Процедуры 11](#_Toc153923929)

[3.4 Триггеры 12](#_Toc153923930)

[3.5 Пользователи 12](#_Toc153923931)

[3.6 Вывод по разработке объектов базы данных 13](#_Toc153923932)

[4 Описание процедур импорта и экспорта 14](#_Toc153923933)

[4.1 Описание процедуры экспорта данных 14](#_Toc153923934)

[4.2 Описание процедуры импорта данных 15](#_Toc153923935)

[4.3 Вывод по разделу 15](#_Toc153923936)

[5 Тестирование производительности 16](#_Toc153923937)

[6 Описание технологии и ее применения в базе данных 18](#_Toc153923938)

[7 Краткое описание приложения для демонстрации 19](#_Toc153923939)

[8 Руководство пользователя 20](#_Toc153923940)

[Заключение 22](#_Toc153923941)

[Список используемых источников 23](#_Toc153923942)

[ПРИЛОЖЕНИЕ А 24](#_Toc153923943)

[ПРИЛОЖЕНИЕ Б 26](#_Toc153923944)

[ПРИЛОЖЕНИЕ В 30](#_Toc153923945)

[ПРИЛОЖЕНИЕ Г 31](#_Toc153923946)

# **Введение**

В современном мире музыка занимает важное место в жизни людей. Желание всегда иметь возможность слушать любимые композиции, привело к появлению большого количества приложений для поиска и потребления музыкального контента. А любое приложение требует наличия качественно спроектированной базы данных. Проектирование необходимо произвести таким образом, чтобы конечные данные соответствовали общим требованиям к информации в базе данных.

Функционально должны быть выполнены следующие задачи:

* определение ролей (администратор, пользователь),
* управление авторами (добавление, удаление, изменение),
* управление пользователями (добавление, удаление, изменение),
* управление песнями (добавление, удаление, изменение),
* управление плейлистами (добавление, удаление, изменение),
* прослушивание композици,
* добавление композиций в избранные.

Цель данного курсового проекта – создание базы данных для хранения музыки, альбомов различных авторов, плейлистов пользователей, ознакомление с технологией полнотекстового поиска в базе данных.

Также необходимо уделить особое внимание вопросам безопасности. Когда речь идет о личной информации пользователей, ее сохранность и конфиденциальность становятся приоритетом. В связи с этим, в проекте будет использоваться шифрование, обеспечивающие защиту от легкого подбора паролей.

# **Постановка задача**

В настоящее время существует большое количество программных продуктов для прослушивания музыки, а программируемые системы облегчают работу с базами данных.

Для того чтобы окончательно определиться с постановкой задачи курсового проекта, необходимо проанализировать прототипы программных средств выбранной темы.

1. **Программное средство «****Spotify»**

На рисунке 1.1 предствлен скриншот веб-приложения «Spotify».

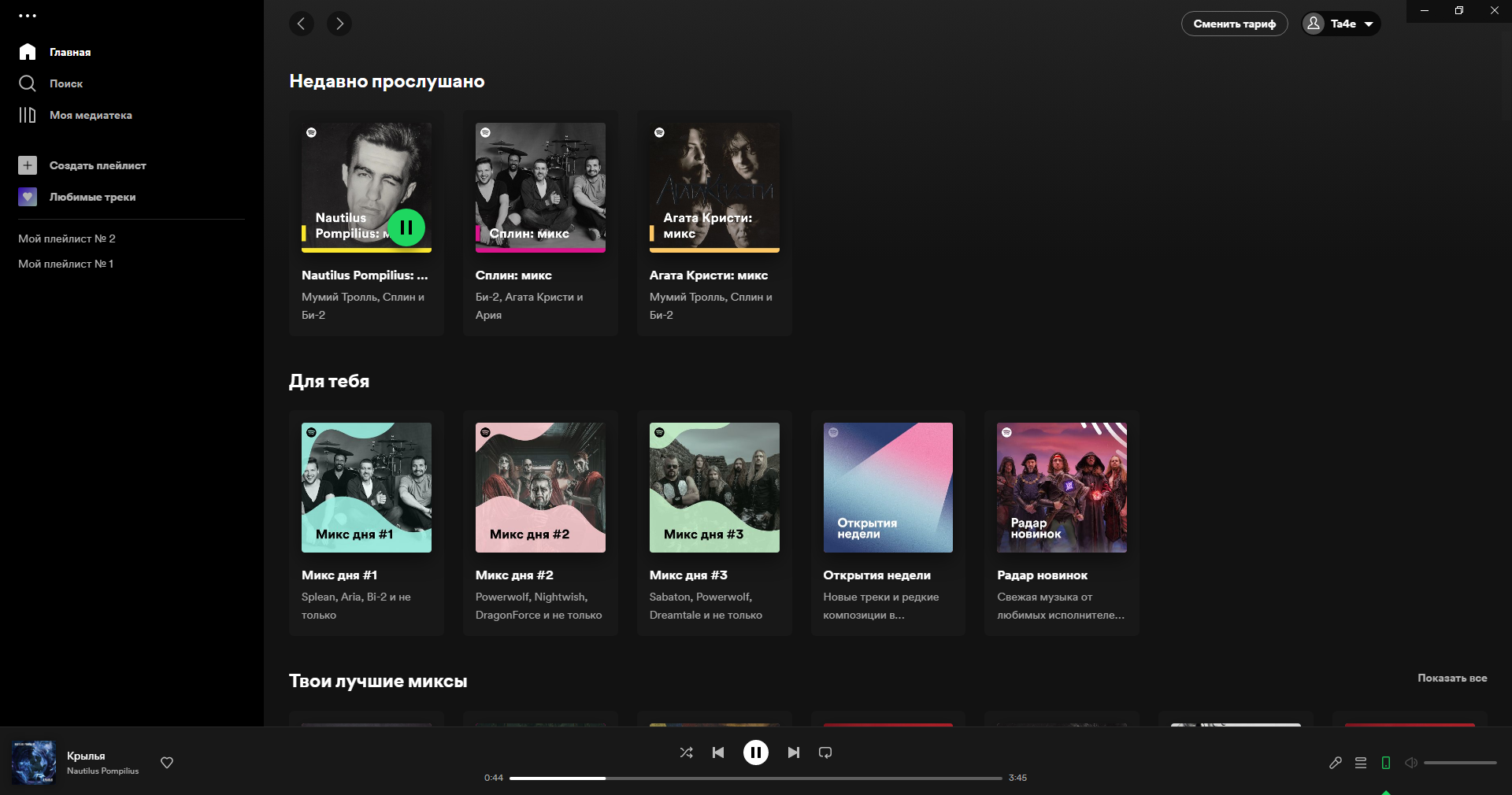


Рисунок 1.1 – Скриншот аналога веб-приложения «Spotify»

В качестве одного из аналогов возьмем программу «Spotify». «Spotify» - [стриминговый сервис](https://ru.wikipedia.org/wiki/%D0%9F%D0%BE%D1%82%D0%BE%D0%BA%D0%BE%D0%B2%D0%BE%D0%B5_%D0%BC%D1%83%D0%BB%D1%8C%D1%82%D0%B8%D0%BC%D0%B5%D0%B4%D0%B8%D0%B0#%D0%A1%D1%82%D1%80%D0%B8%D0%BC%D0%B8%D0%BD%D0%B3%D0%BE%D0%B2%D1%8B%D0%B5_%D1%81%D0%B5%D1%80%D0%B2%D0%B8%D1%81%D1%8B), позволяющий легально прослушивать музыкальные композиции, [аудиокниги](https://ru.wikipedia.org/wiki/%D0%90%D1%83%D0%B4%D0%B8%D0%BE%D0%BA%D0%BD%D0%B8%D0%B3%D0%B0) и [подкасты](https://ru.wikipedia.org/wiki/%D0%9F%D0%BE%D0%B4%D0%BA%D0%B0%D1%81%D1%82%D0%B8%D0%BD%D0%B3), не скачивая их на устройство. Доступен в виде [веб-сайта](https://ru.wikipedia.org/wiki/%D0%92%D0%B5%D0%B1-%D1%81%D0%B0%D0%B9%D1%82), [приложений](https://ru.wikipedia.org/wiki/%D0%9C%D0%BE%D0%B1%D0%B8%D0%BB%D1%8C%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%B8%D0%BB%D0%BE%D0%B6%D0%B5%D0%BD%D0%B8%D0%B5) для всех операционных систем, смартфонов, смарт-устройств и медиасистем автомобилей.

­˗ Огромный каталог музыки с более чем 70 миллионами треков доступен для потоковой передачи.

­˗ Высокое качество звука с возможностью стриминга в формате Lossless.

˗ Возможность создания и редактирования плейлистов.

Однако у данного музыкального сервиса имеются и недостатки:

˗ Наличие рекламы в бесплатной версии сервиса.

­˗ Отсутствие возможности покупки музыки для хранения на устройствах, что ограничивает пользователей, которые хотят иметь физический доступ к своей музыке.

И всё же, данный музыкальный сервис является одним из самых популярных в мире, удобство и большой набор функций позволяет ему занимать лидирующие позиции на рынке музыкальных приложений.

1. **Программное средство «Яндекс.Музыка»**

На рисунке 1.2 предствлен скриншот веб-приложения «Яндекс.Музыка».

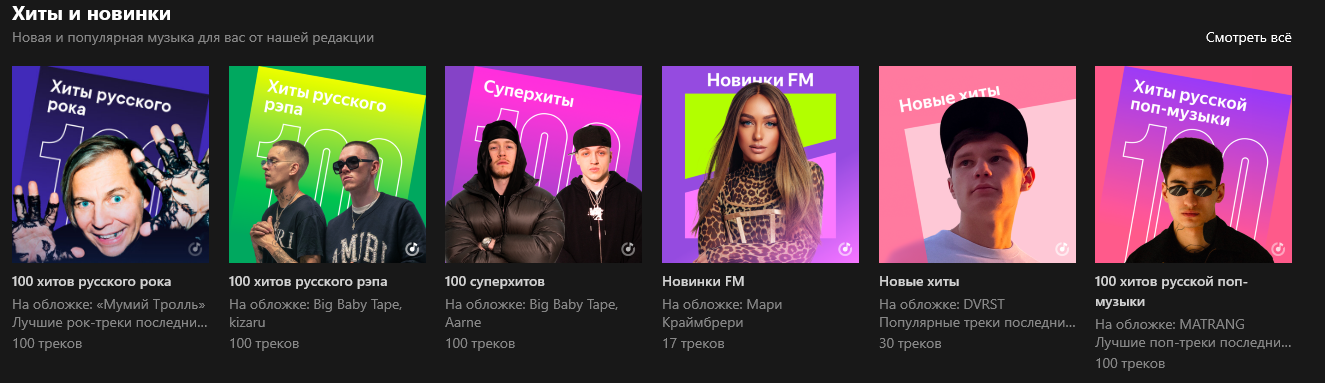


Рисунок 1.2 – Скриншот аналога веб-приложения «Яндекс.Музыка»

Следующая программа для анализа – «Яндекс.Музыка».  
Яндекс.Музыка – ещё один популярный стриминговый сервис музыки. Функционал схож с «Spotify».

˗ Огромная библиотека музыки, включая русскую и зарубежную музыку, подкасты и радио.

˗ Персональные рекомендации на основе прослушиваемых пользователем треков.

˗ Совместимость с голосовыми помощниками, такими как Алиса и Siri.

Но кроме положительный есть так же отрицательные моменты.

˗ Низкое качество звука в бесплатной версии приложения.

˗ Отсутствие возможности покупки музыки для хранения на устройствах, что ограничивает пользователей, которые хотят иметь физический доступ к своей музыке.

1. **Программное средство «Apple Music»**

На рисунке 1.3 представлен скриншот веб-приложения «Apple Music».

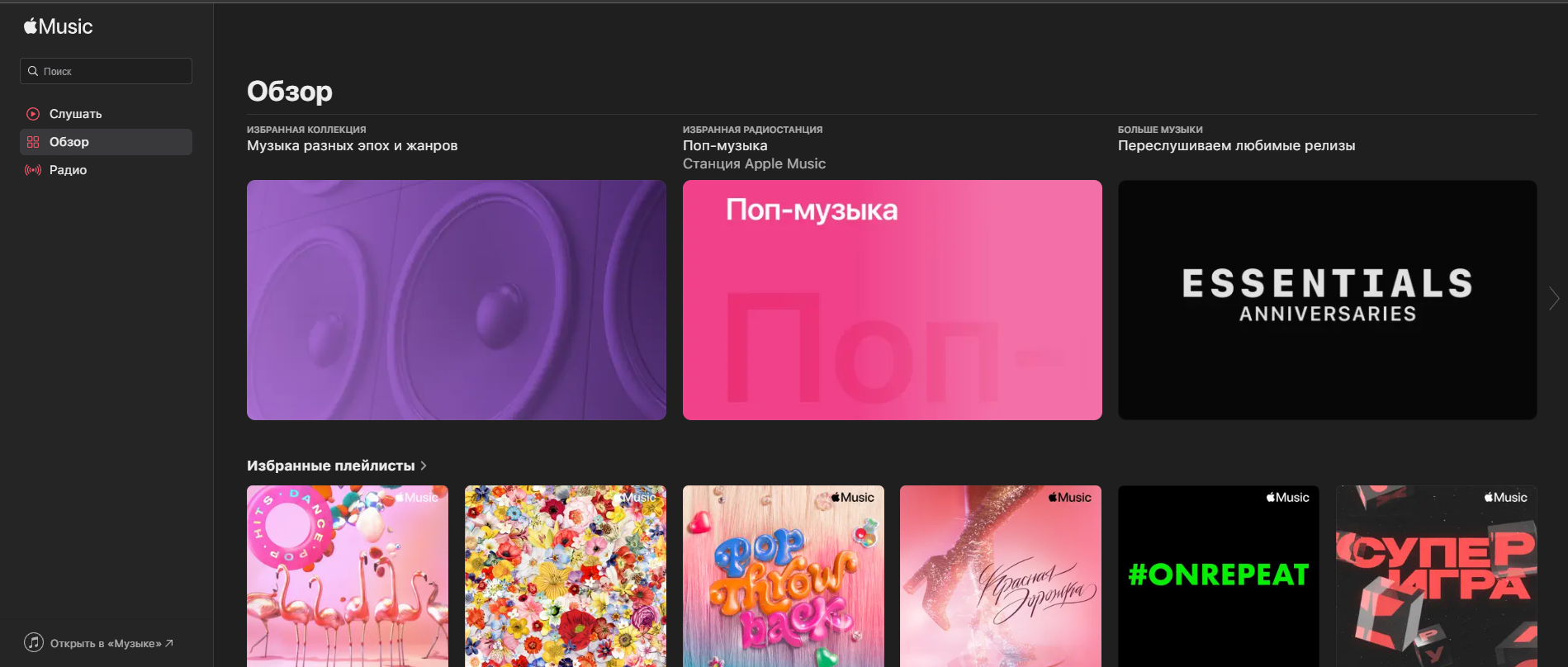


Рисунок 1.3 – Скриншот аналога веб-приложения «Apple Music»

Apple Music - музыкальный стриминговый сервис, предлагаемый Apple. Сервис позволяет получить доступ к огромной библиотеке музыки, включая тысячи альбомов и плейлистов, а также оригинальные программы и шоу, созданные специально для Apple Music Оригинальный контент: Apple Music предлагает различный контент, включая концерты, интервью и документальные фильмы.

1. **Определение цели и задачи**

На рынке, на данный момент, присутствует достаточное количество различных решений для прослушивания музыки, но создания более легких и быстрых решений является приоритетной задачей, которую можно выполнить.

Задача курсовой работы: разработать базу данных, которая предоставит полный спектр различных решений для работы с музыкой, а также выполнить тестирование данной базы данных,

Должны быть выполнены следующие требования:

˗ база данных должна быть реализована в СУБД Oracle 12c.

˗ доступ к данным должен осуществляться только через соответствующие процедуры;

˗ должен быть проведен импорт данных из JSON файлов, экспорт данных в формат JSON;

˗ необходимо протестировать производительность базы данных на таблице, содержащей не менее 100 000 строк, и внести изменения в структуру в случае необходимости. Необходимо проанализировать планы запросов к таблице;

˗ применить технологию базы данных согласно выбранной теме: подробно описать применяемые системные пакеты, утилиты или технологии; показать применение указанной технологии в базе данных.

# **Проектирование базы данных**

Архитектура базы данных в данном проекте реализована в третей нормальной форме. Такой подход обеспечивает ясное разделение ответственности между различными сущностями и позволяет оптимизировать процесс работы приложения.

Диаграмма базы данных, спроектированной в ходе разработки представлена на рисунке 2.1.

|  |
| --- |
|  |

Рисунок 2.1 – Диаграмма базы данных

Таблицы SONG\_USER, PLAYLIST\_SONG, ALBUM\_USER и ALBUM\_SONG являются ассоциативными таблицами обеспечивают возможность создания связи многие ко многим.

Логическая структура таблиц позволяет быстро и легко получать нужную информацию, а также расширять функциональность системы без пересмотра всей архитектуры, что позволит ускорить масштабирование системы.

Диаграмма вариантов использования для пользователя администратора, автора и гостя представлены в приложении Г.

## **Вывод по проектированию модели базы данных**

Архитектура базы данных в проекте следует третьей нормальной форме, обеспечивая эффективное управление данными и поддерживая гибкие связи многие ко многим через ассоциативные таблицы. Введение ролевой таблицы улучшает контроль доступа, а логическая структура таблиц обеспечивает легкость получения информации и готовность к расширению функциональности системы.

# **Разработка объектов базы данных**

1. **Разработка таблиц базы данных**

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

В базе данных представлены 11 таблиц: USERS, ROLES, SONGS, SONG\_USER, PLAYLIST, PLAYLIST\_SONG, LISTENING\_HISTORY, GENRES, ALBUMS, ALBUM\_USER, ALBUM\_SONG.

В данной главе будут описаны все таблицы, которые были использованы в курсовой работе.

Таблица USERS хранит информацию о профилях пользователей. Состоит из следующих столбцов (таблица 3.1):

Таблица 3.1 – Столбцы таблицы USERS

|  |  |  |
| --- | --- | --- |
| Наименование | Описание | Тип |
| USER\_ID | Уникальный идентификатор профиля, первичный ключ | NUMBER |
| USER\_ROLE | Описание роли пользователя | NVARCHAR2(64) |
| USER\_NAME | Имя пользователя | VARCHAR2(50) |
| USER\_LOGIN | Логин пользователя | VARCHAR2(50) |
| USER\_PASS | Зашифрованный пароль пользователя | VARCHAR2(100) |

Таблица ROLES содержит роли, которые определены в базе данных. Состоит из столбцов (таблица 3.2):

Таблица 3.2 – Столбцы таблицы ROLES

|  |  |  |
| --- | --- | --- |
| Наименование | Описание | Тип |
| ROLE\_NAME | Уникальный идентификатор роли, первичный ключ | NVARCHAR2(64) |

Таблица SONGS хранит содержимое, связанное с песнями. Состоит из следующих столбцов (таблица 3.3):

Таблица 3.3 – Столбцы таблицы SONGS

|  |  |  |
| --- | --- | --- |
| Наименование | Описание | Тип |
| SONG\_ID | Уникальный идентификатор песни, первичный ключ | NUMBER |
| GENRE\_ID | ID жанра песни | NUMBER |
| SONG | Путь к файлу | VARCHAR2(200) |
| SONG\_COVER | Обложка песни | VARCHAR2(200) |
| SONG\_NAME | Название песни | VARCHAR2(50) |
| LISTENING\_COUNTER | Количество прослушиваний | NUMBER |
| AUTHOR\_ID | ID автора песни | NUMBER |

Таблица SONG\_USER является вспомогательной таблицей, нужна для связи многие ко многим таблиц SONGS и USERS. Состоит из следующих столбцов (таблица 3.4):

Таблица 3.4 – Столбцы таблицы SONG\_USER

|  |  |  |
| --- | --- | --- |
| Наименование | Описание | Тип |
| SONG\_ID | Идентификатор песни | NUMBER |
| USER\_ID | Идентификатор пользователя | NUMBER |

Таблица PLAYLIST представляет плейлисты, которые может создавать пользователь. Состоит из следующих столбцов (таблица 3.5):

Таблица 3.5 – Столбцы таблицы PLAYLIST

|  |  |  |
| --- | --- | --- |
| Наименование | Описание | Тип |
| PLAYLIST\_ID | Уникальный идентификатор плейлиста, первичный ключ | NUMBER |
| PLAYLIST\_NAME | Название плейлиста | VARCHAR2(50) |
| PLAYLIST\_COVER | Путь к обложке плейлиста | VARCHAR2(50) |
| USER\_ID | Пользователь, который является владельцем данного плейлиста | NUMBER |

Таблица PLAYLIST\_SONG является вспомогательной таблицей, нужна для связи многие ко многим таблиц PLAYLIST и SONG. Состоит из следующих столбцов (таблица 3.6):

Таблица 3.6 – Столбцы таблицы PLAYLIST\_SONG

|  |  |  |
| --- | --- | --- |
| Наименование | Описание | Тип |
| PLAYLIST\_ID | Идентификатор плейлиста | NUMBER |
| SONG\_ID | Идентификатор песни | NUMBER |

Таблица LISTENING\_HISTORY хранит информацию о песнях, которые были прослушаны пользователем. Состоит из следующих столбцов (таблица 3.7):

Таблица 3.7 – Столбцы таблицы LISTENING\_HISTORY

|  |  |  |
| --- | --- | --- |
| Наименование | Описание | Тип |
| USER\_ID | Идентификатор пользователя | NUMBER |
| SONG\_ID | Идентификатор песни | NUMBER |
| AUDITION\_DATE | Дата прослушивания песни | DATE |

Таблица GENRES содержит жанры песен. Состоит из следующих столбцов (таблица 3.8):

Таблица 3.8 – Столбцы таблицы GENRES

|  |  |  |
| --- | --- | --- |
| Наименование | Описание | Тип |
| GENRE\_ID | Уникальный идентификатор жанра, первичный ключ | NUMBER |
| GENRE\_NAME | Название жанра | VARCHAR2(50) |

Таблица ALBUMS содержит альбомы авторов. Состоит из следующих столбцов (таблица 3.9):

Таблица 3.9 – Столбцы таблицы ALBUMS

|  |  |  |
| --- | --- | --- |
| Наименование | Описание | Тип |
| ALBUM\_ID | Уникальный идентификатор альбома, первичный ключ | NUMBER |
| ALBUM\_COVER | Путь к обложке альбома | VARCHAR2(100) |
| REALEASE\_DATE | Дата релиза альбома | DATE |
| ALBUM\_NAME | Название альбома | VARCHAR2(50) |

Таблица ALBUM\_USER является вспомогательной таблицей, нужна для связи многие ко многим таблиц ALBUM и USER (таблица 3.10):

Таблица 3.10 – Столбцы таблицы ALBUM\_USER

|  |  |  |
| --- | --- | --- |
| Наименование | Описание | Тип |
| ALBUM\_ID | Идентификатор альбома | NUMBER |
| USER\_ID | Идентификатор пользователя | NUMBER |

Таблица ALBUM\_SONG является вспомогательной таблицей, нужна для связи многие ко многим таблиц ALBUM и SONG (таблица 3.11):

Таблица 3.11 – Столбцы таблицы ALBUM\_SONG

|  |  |  |
| --- | --- | --- |
| Наименование | Описание | Тип |
| ALBUM\_ID | Идентификатор альбома | NUMBER |
| SONG\_ID | Идентификатор песни | NUMBER |

Листинг SQL-кода для создания таблиц находится в приложении А.

## **Индексы**

Индекс — объект базы данных, создаваемый с целью повышения производительности поиска данных.

Таблицы в базе данных могут иметь большое количество строк, которые хранятся в произвольном порядке, и их поиск по заданному критерию путём последовательного просмотра таблицы строка за строкой может занимать много времени.

В связи с оптимизацией запросов и улучшением производительности был создан уникальный индекс для столбца USER\_NAME таблицы USERS листинг 3.1.

|  |
| --- |
| Create unique index fastSearch On USERS(USER\_NAME); |

Листинг 3.1 – Индекс для USERS

Данный индекс позволит ускорить выборку пользователей по USER\_NAME.

## **Процедуры**

Процедуры, представленные ниже, обеспечивают базовые операции управления данными в базе данных, такие как вставка, обновление и удаление. Эти процедуры охватывают ключевые аспекты системы, включая управление пользователями, плейлистами, песнями и с альбомами.

Листинг PL/SQL-кода для создания процедур находится в приложении Б.

Процедуры, разработанные в рамках курсового проекта:

* create\_user – создание пользователя.;
* update\_user – обновление данных пользователя.;
* delete\_user – позволяет удалить пользователя.;
* create\_song\_user – добавляет выбранную песню в избранные.;
* create\_album – позволяет авторам создавать альбомы.;
* add\_song\_to\_album – позволяет авторам добавлять авторам свои песни в альбом.;
* ALBUM\_BELONGS\_TO\_USER – Это позволяет системе определить, принадлежит ли песня автору или нет.;
* REMOVE\_SONG\_FROM\_ALBUM – удаляет песню из альбома.;
* GET\_USER\_BY\_NAME – позволяет получить пользователя по его имени.;
* ADD\_SONG – позволяет добавить песню в общий список песен.;
* update\_song – обеспечивает возможность удаления песни, удаляя связанные записи.;
* DELETE\_ALL\_SONG – позволяет удалить все песни из базы данных, а так же из таблиц, где данные песни использовались.;
* DELETE\_SONG\_BY\_ID – позволяет удалить нужную песню из базы данных по ID, а так же из таблиц, где песня использовалась.;

## **Триггеры**

Триггеры базы данных служат важной цели в системе управления базами данных. Триггер update\_release\_date\_trigger был разработан для отслеживания добавления новых песен в альбомы. Его главная цель – изменить дату релиза альбома.

Пример создания триггера приводится в листинге 3.2.

|  |
| --- |
| CREATE OR REPLACE TRIGGER update\_release\_date\_trigger AFTER INSERT ON ALBUM\_SONG FOR EACH ROW DECLARE BEGIN  UPDATE ALBUMS  SET RELEASE\_DATE = SYSDATE  WHERE ALBUM\_ID = :NEW.ALBUM\_ID;  COMMIT; END; |

Листинг 3.2 – Создания триггера update\_release\_date\_trigger

Каждый раз при добавлении новой песни в таблицу ALBUM\_SONG триггер выполняет обновление даты релиза альбома. Это сделано для того, чтобы пользователь знал актуальность альбома, ведь дата релиза альбома будет считаться с момента добавления новой песни в альбом.

## **Пользователи**

Пользователь базы данных — это физическое или юридическое лицо, которое имеет доступ к БД и пользуется услугами информационной системы для получения информации. На каждом этапе развития БД (проектирование, реализация, эксплуатация, модернизация и развитие, полная реорганизация) с ней связаны разные категории пользователей.

При проектировании базы данных было создано 3 типа пользователей: пользователь, осуществляющий доступные ему операции, автор, который имеет возможность создавать, редактировать и добавлять песни в альбом, а также администратор имеющий полный доступ ко всем объектам базы данных.

Создание пользователей представлено в листинге 3.3.

|  |
| --- |
| CREATE USER BASE\_USER IDENTIFIED BY 123456 PROFILE USERS\_PROFILE;  CREATE USER SONG\_WRITER IDENTIFIED BY 123456 PROFILE USERS\_PROFILE; |

Листинг 3.3 – Создание пользователей «Пользователь» и «Автор»

Далее для созданных пользователей базы данных были выданы необходимые привилегии для успешного выполнения их задач.

## **Вывод по разработке объектов базы данных**

Все рассмотренные объекты вместе создают сложную, но гибкую и эффективную структуру, которая обеспечивает правильное функционирование и управление данными в системе. Благодаря хорошо спроектированным объектам базы данных система способна обеспечивать надежность, своевременное быстродействие и целостность данных.

Важно отметить, что разработанные объекты базы данных спроектированы с учетом различных ролей пользователей, таких как обычный пользователь, автор и администратор. Каждому пользователю предоставлены соответствующие привилегии для выполнения своих задач в системе.

Система охватывает не только основные операции управления данными, но также предоставляет дополнительные функциональности, такие как возможность добавления песни в избранные, создание и редактирование альбомов, а также удобный поиск пользователей по имени.

Таким образом, разработанные объекты базы данных обеспечивают не только структуру для хранения данных, но и функциональность, способствующую эффективному управлению и взаимодействию с системой.

# **Описание процедур импорта и экспорта**

В данной курсовой работе реализованы процедуры экспорта и импорта данных из JSON файла в базу данных таблицы PLAYLIST и наоборот.

1. **Описание процедуры экспорта данных**

В представленном блоке кода описана процедура EXPORT\_JSON\_PLAYLIST. Эти процедуры позволяют экспортировать и импортировать данные из таблицы USERS в формат JSON.

Код процедура импорта данных в формат JSON представлена в листинге 4.1.

|  |
| --- |
| CREATE OR REPLACE PROCEDURE EXPORT\_JSON\_PLAYLIST IS  v\_file UTL\_FILE.FILE\_TYPE;  v\_cursor SYS\_REFCURSOR;  v\_row PLAYLIST%ROWTYPE;  v\_json CLOB; BEGIN  v\_file := UTL\_FILE.FOPEN('UTL\_DIR', 'JSON\_PLAYLIST.json', 'W', 32767); -- Increase buffer size  OPEN v\_cursor FOR SELECT \* FROM PLAYLIST;  v\_json := '[';  LOOP  FETCH v\_cursor INTO v\_row;  EXIT WHEN v\_cursor%NOTFOUND;  IF v\_json != '[' THEN  v\_json := v\_json || ',';  END IF;  v\_json := v\_json || '{';  v\_json := v\_json || '"PLAYLIST\_ID":' || v\_row.PLAYLIST\_ID ||',';  v\_json := v\_json || '"PLAYLIST\_NAME":"' || COALESCE(REPLACE(v\_row.PLAYLIST\_NAME, '"', '\"'), 'NULL') || '",';  v\_json := v\_json || '"PLAYLIST\_COVER":"' || COALESCE(REPLACE(v\_row.PLAYLIST\_COVER, '"', '\"'), 'NULL') || '",';  v\_json := v\_json || '"USER\_ID":' || NVL(TO\_CHAR(v\_row.USER\_ID), 'NULL') || '';  v\_json := v\_json || '}';  END LOOP;  CLOSE v\_cursor;  v\_json := v\_json || ']';  UTL\_FILE.PUT\_LINE(v\_file, v\_json);  UTL\_FILE.FCLOSE(v\_file); EXCEPTION  WHEN OTHERS THEN  DBMS\_OUTPUT.PUT\_LINE('Error: ' || SQLCODE || ' - ' || SQLERRM);  RAISE; END; |

Листинг 4.1 – Процедура экспорта

Вызвав данную процедуру, мы создадим файл заполненный данными из таблицы PLAYLIST в формате JSON.

## **Описание процедуры импорта данных**

Процедуру импорта IMPORT\_JSON\_PLAYLIST рассмотрим также на таблице PLAYLIST. Код процедуры экспорта представлена в листинге 4.2.

|  |
| --- |
| CREATE OR REPLACE PROCEDURE IMPORT\_JSON\_PLAYLIST IS TYPE rec\_tab\_type IS TABLE OF PLAYLIST%ROWTYPE;  rec\_tab rec\_tab\_type;  v\_user\_exists NUMBER; BEGIN  SELECT PLAYLIST\_ID, PLAYLIST\_NAME, PLAYLIST\_COVER, USER\_ID  BULK COLLECT INTO rec\_tab  FROM JSON\_TABLE(  BFILENAME('UTL\_DIR', 'JSON\_PLAYLIST.json'),  '$[\*]' COLUMNS (  PLAYLIST\_ID NUMBER PATH '$.PLAYLIST\_ID',  PLAYLIST\_NAME VARCHAR2(50) PATH '$.PLAYLIST\_NAME',  PLAYLIST\_COVER VARCHAR2(50) PATH '$.PLAYLIST\_COVER',  USER\_ID NUMBER PATH '$.USER\_ID' ));  FOR i IN 1..rec\_tab.COUNT  LOOP -- Check if the user exists  SELECT COUNT(\*) INTO v\_user\_exists FROM USERS WHERE USER\_ID = rec\_tab(i).USER\_ID;  IF v\_user\_exists > 0 THEN  BEGIN INSERT INTO PLAYLIST (PLAYLIST\_ID, PLAYLIST\_NAME, PLAYLIST\_COVER, USER\_ID) VALUES (rec\_tab(i).PLAYLIST\_ID, rec\_tab(i).PLAYLIST\_NAME, rec\_tab(i).PLAYLIST\_COVER, rec\_tab(i).USER\_ID); EXCEPTION  WHEN DUP\_VAL\_ON\_INDEX THEN  DBMS\_OUTPUT.PUT\_LINE('Skipping duplicate record with PLAYLIST\_ID ' || rec\_tab(i).PLAYLIST\_ID);  END; ELSE  DBMS\_OUTPUT.PUT\_LINE('Skipping playlist with PLAYLIST\_ID ' || rec\_tab(i).PLAYLIST\_ID || ' because the corresponding user does not exist.'); END IF;  END LOOP; COMMIT; EXCEPTION WHEN OTHERS THEN  DBMS\_OUTPUT.PUT\_LINE('Error: ' || SQLCODE || ' - ' || SQLERRM); RAISE; END; |

Листинг 4.2 – Процедура экспорта

Обе процедуры содержат механизмы обработки исключений для обработки ошибок, связанных с работой с файлами и другими возможными ошибками.

4.3 Вывод по разделу

В данном разделе мы рассмотрели вопрос о наполнении таблиц данными (импорте) и выгрузке данных из таблицы (экспорте), предложив решение на основе использования хранимых процедур.

# **Тестирование производительности**

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

Для начала, нужно заполнить таблицу большим количеством данных, в моем примере это 100 000 строк (рисунок 5.1).



Рисунок 5.1 –Заполнение таблицы большим количеством данных;

Ниже на рисунке 5.2 представлен результат статистического анализа и компиляции SQL Developer до изменения таблицы

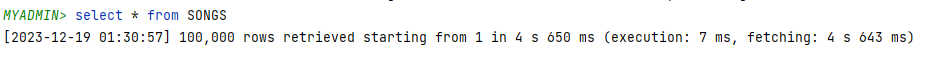


Рисунок 5.2 – Статический анализ перед изменением таблицы

Как можно увидеть, затраченное время достаточно велико. Для оптимизации поиска была изменена таблицы путем добавления индекса, код которого представлен на рисунке 5.3.



Рисунок 5.3 – Применение индекса к таблице SONGS

Ниже на рисунке 5.4 представлен результат статистического анализа и компиляции SQL Developer после применение индекса.



Рисунок 5.4 – Статический анализ после изменение таблицы

На основе использованных индексов можно сделать вывод о сокращении времени поиска данных. Однако, необходимо также учитывать, что эффективность индексов может варьироваться в зависимости от структуры данных, запросов и общей архитектуры базы данных (БД). Проблемы выявления узких мест производительности баз данных напрямую связаны с метриками, методами измерения производительности и выбором подходящей технологии.

Для более глубокого анализа производительности БД необходимо рассмотреть различные аспекты, такие как оптимизация запросов, эффективность использования индексов, объем данных, аппаратные ресурсы и конфигурация сервера базы данных. Тестирование производительности должно быть основано на реалистичных сценариях использования и репрезентативных данных, чтобы получить достоверные результаты.

Тестирование БД особенно важно для приложений с низким и средним уровнем сложности, где эффективное использование ресурсов базы данных напрямую влияет на общую производительность приложения. Правильно организованный процесс тестирования дает уверенность в том, что приложение способно обрабатывать запросы эффективно и предоставлять высокую производительность для конечного пользователя.

В результате использование индексов в базе данных не только сокращает время поиска данных, но и способствует оптимизации выполнения запросов. Это, в свою очередь, приводит к более отзывчивой системе, снижению времени отклика и повышению общей производительности приложения. С учетом регулярного мониторинга и оптимизации запросов, а также управления ресурсами, можно достичь устойчивой и эффективной работы базы данных в долгосрочной перспективе.

# **Описание технологии и ее применения в базе данных**

Oracle Text — это технология полнотекстового поиска в базе данных Oracle. Она предоставляет возможность эффективного выполнения поисковых запросов, включая поддержку подстановочных символов, синонимов, весовых коэффициентов и других продвинутых возможностей.

Чтобы не загромождать оперативную память, используется сжатие столбцов в памяти. Технология полнотекстового поиска в системе управления базами данных Oracle представляет собой комплексный механизм, спроектированный для эффективного и точного поиска информации в больших объемах текстовых данных. Эта технология играет ключевую роль в обеспечении высокой производительности и точности при выполнении запросов, связанных с текстовыми данными, что является неотъемлемой частью современных информационных систем.

Основой полнотекстового поиска в Oracle является использование индексов полнотекстового поиска (Full Text Indexes). Эти индексы позволяют быстро и эффективно находить соответствия между текстовыми запросами и содержимым базы данных. Ключевыми особенностями этой технологии являются интеллектуальная обработка текста, многозначность запросов и поддержка различных языков.

Создание индекса полнотекстового поиска можно увидеть на рисунке 6.1.



Рисунок 6.1 – создание полнотекстового индекса

Процесс полнотекстового поиска начинается с создания Full Text Index на столбце, содержащем текстовые данные. Этот индекс анализирует и индексирует каждый документ, извлекая ключевые слова и строя полнотекстовый индекс для оптимизации последующих запросов. При выполнении полнотекстового запроса Oracle использует этот индекс для поиска наиболее подходящих результатов.

Одной из важных особенностей технологии полнотекстового поиска в Oracle является поддержка различных методов сопоставления строк, включая возможность выполнения запросов с учетом синонимов, весового приоритета слов и других параметров, что увеличивает точность и гибкость поискового механизма.

В заключение, технология полнотекстового поиска в Oracle представляет собой мощный инструмент для обеспечения эффективного и точного поиска текстовой информации в базах данных. Ее основные принципы включают создание Full Text Indexes, интеллектуальную обработку текста и поддержку различных языков, что обеспечивает высокую производительность и удовлетворение требований современных информационных систем.

# **Краткое описание приложения для демонстрации**

Для демонстрации возможности прослушивания музыки было создано компактное веб-приложение, предоставляющее пользователям функционал входа в учетную запись или регистрации нового аккаунта.

После успешного входа в систему пользователь получает широкий доступ к коллекции песен, доступных в базе данных. Он может свободно прослушивать музыку, наслаждаясь различными жанрами и исполнителями. Этот функционал предоставляет уникальный опыт для любителей музыки, позволяя им находить и открывать новые композиции.

Важным элементом функционала приложения является отслеживание количества прослушиваний каждой песни. При каждом прослушивании трека система автоматически записывает эту информацию, позволяя пользователям и администраторам базы данных следить за популярностью конкретных композиций. Эта функция также вносит элемент социальной интерактивности, позволяя пользователям видеть, какие треки пользуются наибольшей популярностью в сообществе.

Для обеспечения удобства пользователей и администраторов создана специальная таблица, в которой фиксируется история прослушиваний. Для администратора базы данных эта таблица становится ценным инструментом для анализа популярности треков.

# **Руководство пользователя**

При первом посещении клиентской части вас встретит страница авторизации, где вы можете войти в учетную запись, вводя необходимые данные. После успешной регистрации вы можете войти в систему, используя свой login и пароль на странице авторизации. Она приведена на рисунке 8.1.

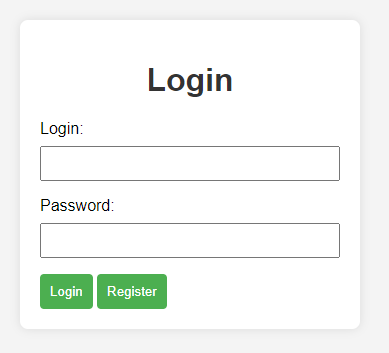


Рисунок 8.1 – Страница входа в аккаунт

Если же аккаунта нету, то его можно зарегистрировать, также во время регистрации у пользователя есть возможность определить себя как обычный пользователь или же как автор. Страница регистрации на рисунке 8.2.

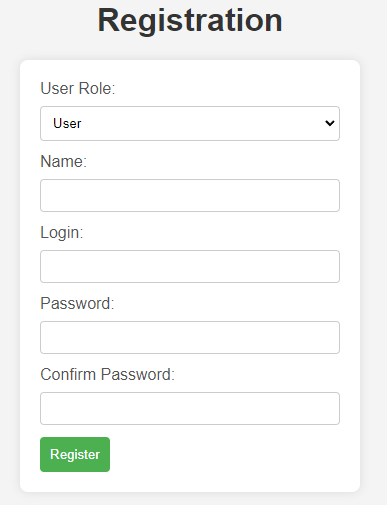


Рисунок 8.2 – Страница регистраци

После успешной авторизации пользователь увидит весь список песен, и сможет прослушать их. Страница со списком треков на рисунке 8.3.

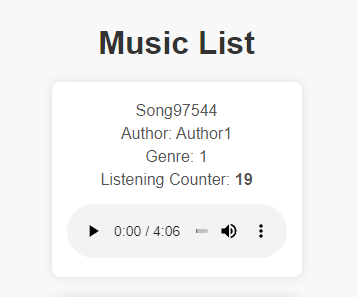


Рисунок 8.3 – Страница с песнями

Данное приложение простое в использовании и интуитивно понятно, также оно позволяет проверить функциональность некоторых функций, которые были определены в базе данных.

Заключение

В результате выполнения данного курсового проекта была разработана база

данных для хранения музыки, альбомов различных авторов, плейлистов пользователей.

В процессе работы были рассмотрены вопросы проектирования базы данных, реализован алгоритм шифрования, осуществлена интеграция технологии полнотекстового поиска, а также выполнено тестирование производительности системы в условиях реального использования. Основной целью курсового проекта стало проектирование базы данных для возможной интеграции с приложением в будущем, которое помогло бы облегчить взаимодействие с базой данных посредством программного интерфейса.

При разработке выполнены следующие пункты:

* определение ролей (администратор, пользователь),
* управление авторами (добавление, удаление, изменение),
* управление пользователями (добавление, удаление, изменение),
* управление песнями (добавление, удаление, изменение),
* управление плейлистами (добавление, удаление, изменение),
* прослушивание композици,
* добавление композиций в избранные.

БД прошло тестирование при использовании большого количество данных. Также были реализованы процедуры для импорта, экспорта данных в формат JSON.

Таким образом, курсовой проект позволил улучшить практические навыки в использовании и разработке современных информационных систем, а также понимание применения технологии полнотекстового поиска.

Список используемых источников

1. Нистюк О. А. Курс лекций по базам данных, [Электронный ресурс]. – Режим доступа: https://diskstation.belstu.by:5001/. – Дата доступа: 05.09.2023.

2. MSDN сеть разработчиков в Microsoft [Электронный ресурс] / Режим доступа ­– URL: https://learn.microsoft.com/ru-ru/ – Дата доступа: 14.10.2023

URL: http://msdn.microsoft.com/library/rus/ – Дата доступа: 20.10.2023

3. Примеры импорта и экспорта JSON-документов [Электронный ресурс] / Режим доступа – URL: https://stackoverflow.com/questions/9672780/importing-json-data-from-json-file-to-sql-database ­ – Дата доступа: 30.10.2023

4. METANIT.COM Сайт о программировании [Электронный ресурс] / Режим доступа – URL: https://metanit.com – Дата доступа: 04.11.2023

5. Технология Oracle Text [Электронный ресурс] / Режим доступа – URL: <https://blogs.oracle.com/database/post/oracle-text-and-accented-characters/> – Дата доступа: 01.11.2023

ПРИЛОЖЕНИЕ А

Создание базы данных

|  |
| --- |
| CREATE TABLE ROLES (  ROLE\_NAME NVARCHAR2(64) PRIMARY KEY );  CREATE TABLE USERS (  USER\_ID INTEGER GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,  USER\_ROLE NVARCHAR2(64) NOT NULL,  USER\_NAME VARCHAR2(50) NOT NULL,  USER\_LOGIN VARCHAR2(50) NOT NULL UNIQUE,  USER\_PASS VARCHAR2(100) NOT NULL,  CONSTRAINT FK\_USER\_ROLE  FOREIGN KEY (USER\_ROLE)  REFERENCES ROLES (ROLE\_NAME) );  CREATE TABLE PLAYLIST (  PLAYLIST\_ID INTEGER GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,  PLAYLIST\_NAME VARCHAR2(50) NOT NULL,  PLAYLIST\_COVER VARCHAR2(50) NOT NULL,  USER\_ID INT NOT NULL,  FOREIGN KEY (USER\_ID) REFERENCES USERS(USER\_ID) ON DELETE CASCADE );  CREATE TABLE SONGS (  SONG\_ID INTEGER GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,  GENRE\_ID INT NOT NULL,  SONG VARCHAR2(200) NOT NULL,  SONG\_COVER VARCHAR2(200) NOT NULL,  SONG\_NAME VARCHAR2(50) NOT NULL,  LISTENING\_COUNTER INT NOT NULL,  AUTHOR\_ID INT,  CONSTRAINT FK\_SONG\_AUTHOR\_USER  FOREIGN KEY (AUTHOR\_ID) REFERENCES USERS(USER\_ID) ON DELETE CASCADE );  CREATE TABLE SONG\_USER (  SONG\_ID INT NOT NULL,  USER\_ID INT NOT NULL,   FOREIGN KEY (USER\_ID) REFERENCES USERS(USER\_ID) ON DELETE CASCADE,  FOREIGN KEY (SONG\_ID) REFERENCES SONGS(SONG\_ID) ON DELETE CASCADE );  CREATE TABLE PLAYLIST\_SONG (  PLAYLIST\_ID INT NOT NULL,  SONG\_ID INT NOT NULL,  FOREIGN KEY (SONG\_ID) REFERENCES SONGS(SONG\_ID) ON DELETE CASCADE,  FOREIGN KEY (PLAYLIST\_ID) REFERENCES PLAYLIST(PLAYLIST\_ID) ON DELETE CASCADE );  CREATE TABLE GENRES (  GENRE\_ID INTEGER GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,  GENRE\_NAME VARCHAR2(50) NOT NULL );  CREATE TABLE LISTENING\_HISTORY (  USER\_ID INT NOT NULL,  SONG\_ID INT NOT NULL,  AUDITION\_DATE DATE NOT NULL, --время  FOREIGN KEY (SONG\_ID) REFERENCES SONGS(SONG\_ID),  FOREIGN KEY (USER\_ID) REFERENCES USERS(USER\_ID) );  CREATE TABLE ALBUMS (  ALBUM\_ID INTEGER GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,  ALBUM\_COVER VARCHAR2(100) NOT NULL,  RELEASE\_DATE DATE NOT NULL, --дата создания альбома  ALBUM\_NAME VARCHAR2(50) NOT NULL );  CREATE TABLE ALBUM\_USER (  ALBUM\_ID INT NOT NULL,  USER\_ID INT NOT NULL,  FOREIGN KEY (ALBUM\_ID) REFERENCES ALBUMS(ALBUM\_ID) ON DELETE CASCADE,  FOREIGN KEY (USER\_ID) REFERENCES USERS(USER\_ID) ON DELETE CASCADE ) TABLESPACE ALEGRO\_PDB;  CREATE TABLE ALBUM\_SONG (  ALBUM\_ID INT NOT NULL,  SONG\_ID INT NOT NULL,  FOREIGN KEY (SONG\_ID) REFERENCES SONGS(SONG\_ID) ON DELETE CASCADE,  FOREIGN KEY (ALBUM\_ID) REFERENCES ALBUMS(ALBUM\_ID) ON DELETE CASCADE ); |

ПРИЛОЖЕНИЕ Б

Создание процеду

|  |
| --- |
| CREATE OR REPLACE PROCEDURE create\_user (  p\_user\_role IN USERS.USER\_ROLE%TYPE,  p\_name IN USERS.USER\_NAME%TYPE,  p\_login IN USERS.USER\_LOGIN%TYPE,  p\_password IN VARCHAR2 ) IS  empty\_parameter\_ex EXCEPTION;  duplicate\_name\_login\_ex EXCEPTION;  PRAGMA EXCEPTION\_INIT(duplicate\_name\_login\_ex, -1); -- Код ошибки для дублирования  BEGIN  IF TRIM(p\_name) IS NULL OR TRIM(p\_login) IS NULL OR TRIM(p\_password) IS NULL THEN  RAISE empty\_parameter\_ex;  END IF;   -- Проверяем уникальность имени и логина  BEGIN  INSERT INTO USERS(USER\_ROLE, USER\_NAME, USER\_LOGIN, USER\_PASS)  VALUES (p\_user\_role, TRIM(p\_name), TRIM(p\_login), (SELECT hash\_password(TRIM(p\_password)) FROM DUAL));  COMMIT;  EXCEPTION  WHEN duplicate\_name\_login\_ex THEN  dbms\_output.put\_line('Name or login is not unique');  WHEN OTHERS THEN  dbms\_output.put\_line('Unexpected error');  END;  EXCEPTION  WHEN empty\_parameter\_ex THEN  dbms\_output.put\_line('Empty parameter');  WHEN OTHERS THEN  dbms\_output.put\_line('Error in create\_user procedure'); END;  CREATE OR REPLACE PROCEDURE update\_user (  p\_id IN USERS.USER\_ID%TYPE,  p\_user\_role IN USERS.USER\_ROLE%TYPE DEFAULT NULL,  p\_name IN USERS.USER\_NAME%TYPE DEFAULT NULL,  p\_login IN USERS.USER\_LOGIN%TYPE DEFAULT NULL,  p\_password IN VARCHAR2 DEFAULT NULL ) IS  v\_hashed\_password VARCHAR2(256); -- Используйте размер, подходящий для вашей функции hash\_password  v\_name\_exists NUMBER;  v\_login\_exists NUMBER;  v\_user\_exists NUMBER;  empty\_parameter\_ex EXCEPTION; BEGIN  IF TRIM(p\_id) IS NULL OR TRIM(p\_user\_role) IS NULL OR TRIM(p\_name) IS NULL OR TRIM(p\_login) IS NULL OR TRIM(p\_password) IS NULL THEN  RAISE empty\_parameter\_ex;  END IF;  -- Проверка существования пользователя по ID  SELECT COUNT(\*)  INTO v\_user\_exists  FROM USERS  WHERE USER\_ID = p\_id;   IF v\_user\_exists = 0 THEN  RAISE\_APPLICATION\_ERROR(-20000, 'Пользователь с указанным ID не найден');  END IF;   -- Проверка на изменение значений перед выполнением UPDATE  IF p\_user\_role IS NOT NULL OR p\_login IS NOT NULL OR p\_name IS NOT NULL OR p\_password IS NOT NULL THEN   -- Проверка уникальности имени пользователя  IF p\_name IS NOT NULL THEN  SELECT COUNT(\*)  INTO v\_name\_exists  FROM USERS  WHERE USER\_NAME = TRIM(p\_name) AND USER\_ID != p\_id;   IF v\_name\_exists > 0 THEN  RAISE\_APPLICATION\_ERROR(-20001, 'Имя пользователя уже используется');  END IF;  END IF;   -- Проверка уникальности логина  IF p\_login IS NOT NULL THEN  SELECT COUNT(\*)  INTO v\_login\_exists  FROM USERS  WHERE USER\_LOGIN = TRIM(p\_login) AND USER\_ID != p\_id;   IF v\_login\_exists > 0 THEN  RAISE\_APPLICATION\_ERROR(-20002, 'Логин уже используется');  END IF;  END IF;   -- Обработка хешированного пароля  IF p\_password IS NOT NULL THEN  v\_hashed\_password := hash\_password(TRIM(p\_password));  END IF;   -- Выполнение UPDATE  UPDATE USERS  SET USER\_ROLE = CASE WHEN p\_user\_role IS NOT NULL THEN TRIM(p\_user\_role) ELSE USER\_ROLE END,  USER\_LOGIN = CASE WHEN p\_login IS NOT NULL THEN TRIM(p\_login) ELSE USER\_LOGIN END,  USER\_NAME = CASE WHEN p\_name IS NOT NULL THEN TRIM(p\_name) ELSE USER\_NAME END,  USER\_PASS = CASE WHEN v\_hashed\_password IS NOT NULL THEN v\_hashed\_password ELSE USER\_PASS END  WHERE USER\_ID = p\_id;   DBMS\_OUTPUT.PUT\_LINE('Пользователь успешно обновлен');   ELSE  DBMS\_OUTPUT.PUT\_LINE('Нет изменений для обновления');  END IF; EXCEPTION  WHEN empty\_parameter\_ex THEN  dbms\_output.put\_line('Empty parameter');  WHEN OTHERS THEN  DBMS\_OUTPUT.PUT\_LINE('Ошибка в процедуре update\_user: ' || SQLCODE || ' - ' || SQLERRM);  RAISE; END;  --Удаление пользоваетля CREATE OR REPLACE PROCEDURE delete\_user (  p\_id IN USERS.USER\_ID%TYPE ) IS BEGIN  -- Удаление из таблицы SONG\_USER  DELETE FROM SONG\_USER WHERE USER\_ID = p\_id;   -- Удаление из таблицы PLAYLIST\_SONG  DELETE FROM PLAYLIST\_SONG WHERE PLAYLIST\_ID IN (SELECT PLAYLIST\_ID FROM PLAYLIST WHERE USER\_ID = p\_id);   -- Удаление из таблицы PLAYLIST  DELETE FROM PLAYLIST WHERE USER\_ID = p\_id;   -- Удаление из таблицы ALBUM\_USER  DELETE FROM ALBUM\_USER WHERE USER\_ID = p\_id;   -- Удаление из таблицы USERS  DELETE FROM USERS WHERE USER\_ID = p\_id;   COMMIT; END delete\_user;  create or replace PROCEDURE create\_song\_user (  p\_song\_id SONG\_USER.SONG\_ID%TYPE,  p\_user\_id SONG\_USER.USER\_ID%TYPE ) IS  empty\_parameter\_ex EXCEPTION; BEGIN  IF TRIM(p\_song\_id) IS NULL OR TRIM(p\_user\_id) IS NULL THEN  RAISE empty\_parameter\_ex;  END IF;   INSERT INTO Song\_User (SONG\_ID, USER\_ID) VALUES (p\_song\_id, p\_user\_id);  COMMIT; EXCEPTION  WHEN empty\_parameter\_ex THEN  dbms\_output.put\_line('Empty parameter');  WHEN OTHERS THEN  dbms\_output.put\_line('Error occurred: ' || SQLERRM);  RAISE; END create\_song\_user; |

ПРИЛОЖЕНИЕ В

Добавление 100 000 строк

|  |
| --- |
| CREATE OR REPLACE PROCEDURE FILL\_SONGS\_TABLE AS BEGIN  FOR i IN 1..100000  LOOP  DECLARE  v\_song\_name VARCHAR2(100);  v\_song\_file VARCHAR2(200);  v\_song\_cover VARCHAR2(200);  BEGIN  v\_song\_name := 'Song' || i;  v\_song\_file := 'D:\\УНИК\\Семестр 5\\урсач\\TestSearch\\public\\audio\\Dreamtale - Intro.mp3';  v\_song\_cover := 'D:\\УНИК\\Семестр 5\\урсач\\TestSearch\\public\\audio\\image.jpg';   INSERT INTO SONGS (GENRE\_ID, SONG, SONG\_COVER, SONG\_NAME, LISTENING\_COUNTER, AUTHOR\_ID)  VALUES (1, v\_song\_file, v\_song\_cover, v\_song\_name, 0, 53);  END;  END LOOP;  COMMIT; END FILL\_SONGS\_TABLE; alter session set "\_ORACLE\_SCRIPT" = true; |

ПРИЛОЖЕНИЕ Г

Диаграмма вариантов использования

