МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждение образования «БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ

ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет Информационных технологий

Кафедра Информационные системы и технологии

Специальность 1–40 05 01 Информационные системы и технологии

Специализация 1–40 05 01-03 Информационные системы и технологии (издательско-полиграфический комплекс)

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

**К КУРСОВОМУ ПРОЕКТУ НА ТЕМУ:**

«Реализация базы данных музыкального сервиса с использованием мультимедийных типов данных»

Выполнил студент Пригодич Вера Валерьевна

(Ф.И.О.)

Руководитель проекта асс. Сазонова Д. В.

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

Заведующий кафедрой к.т.н., доц. Смелов В.В .

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

Консультант: асс. Сазонова Д. В.

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

Нормоконтролер: асс. Сазонова Д. В.

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

Курсовой проект защищен с оценкой

Минск 2023

# Содержание

[Введение 3](#_Toc134792606)

[1 Аналитический обзор литературы 4](#_Toc134792607)

[1.1 Аналоги 4](#_Toc134792608)

[1.1.1 Яндекс Музыка 4](#_Toc134792609)

[1.1.2 Spotify 5](#_Toc134792610)

[1.1.3 Apple Music 5](#_Toc134792611)

[1.2 Вывод по разделу 6](#_Toc134792612)

[2 Проектирование 7](#_Toc134792613)

[2.1 Таблицы базы данных 7](#_Toc134792614)

[2.2 Пользователи базы данных 9](#_Toc134792615)

[2.3 Представления базы данных 11](#_Toc134792616)

[2.4 Триггеры базы данных 11](#_Toc134792617)

[2.5 Хранимые процедуры 12](#_Toc134792618)

[2.5.1 Общий пакет 12](#_Toc134792619)

[2.5.2 Пакет клиента 13](#_Toc134792620)

[2.5.3 Пакет администратора 14](#_Toc134792621)

[2.5.4 Процедуры экспорта и импорта таблиц в XML 14](#_Toc134792622)

[2.6 Вывод по разделу 15](#_Toc134792623)

[3 Тестирование и улучшение производительности 16](#_Toc134792624)

[3.1 Вывод по разделу 17](#_Toc134792625)

[4 Описание технологии 18](#_Toc134792626)

[4.1 Вывод по разделу 18](#_Toc134792627)

[5 Руководство пользователя 19](#_Toc134792628)

[5.1 Выводы по разделу 28](#_Toc134792629)

[Заключение 29](#_Toc134792630)

[Список используемых источников 30](#_Toc134792631)

[Приложение А 31](#_Toc134792632)

[Приложение Б 33](#_Toc134792633)

[Приложение В 40](#_Toc134792634)

[Приложение Г 54](#_Toc134792635)

[Приложение Д 63](#_Toc134792636)

## Введение

Целью данной работы являлась разработка реляционной базы данных для музыкального (стримингового) сервиса. 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 организованная структура, предназначенная для хранения информации, систематизированная таким образом, чтобы эти материалы могли быть найдены и обработаны с помощью электронной вычислительной машины. Реляционная база данных — база данных, основанная на реляционной модели данных. В качестве СУБД для базы данных была выбрана Oracle 12c.

В основной части будут затронуты все аспекты разработки проекта и обоснованы некоторые технические приёмы, к которым приходилось прибегнуть, с целью реализации работы веб-сервера с базой данных.

Для обеспечения безопасности пользователей приложения в моем курсовом проекте используется технология шифрования паролей от аккаунта перед записью их в базу данных. А также для обеспечения работы приложения использовались мультимедийные типы данных при хранении изображений и аудио файлов.

Основные требования к приложению:

* управление информацией о композициях;
* дополнение информации;
* реализация ролей администратора и рядового пользователя;
* поиск аудиозаписей по альбому и исполнителю;
* загрузка аудиозаписей на платформу администратором;
* взаимодействие с базой данных при помощи хранимых процедур.

В пояснительной записке вы сможете найти краткую информацию о похожих продуктах, архитектуре, реализации проекта, руководстве пользователя.

## 1 Аналитический обзор литературы

## 1.1 Аналоги

В современном мире музыка стала неотъемлемой частью жизни большинства людей. С развитием технологий и интернета появилось множество музыкальных сервисов, которые предоставляют возможность слушать и скачивать музыку в любое время и в любом месте.

### Яндекс Музыка

Данное приложение представляет собой музыкальный стриминговый сервис, созданный компанией Яндекс. Приложение содержит каталог музыки, однако полный доступ к каталогу можно получить исключительно по платной подписке.

Данный функционал позволяет:

– Создавать плейлисты;

– Для каждого пользователя создаётся отдельный плейлист куда добавляются треки, которые ему понравились;

– Возможность создания собственных плейлистов;

– Возможность прослушивания потока музыки, где нейросеть сама будет подбирать музыку по твоему вкусу;

– Добавлять исполнителей и их альбомы в понравившееся;

– Обширный каталог плейлистов составленных по жанру; исполнителю стране и так далее;

– Чарт музыки с самыми популярными трэками на площадке;

– Также предоставляется библиотека подкастов и аудиокниг.

Интерфейс данного приложения представлен на рисунке 1.1

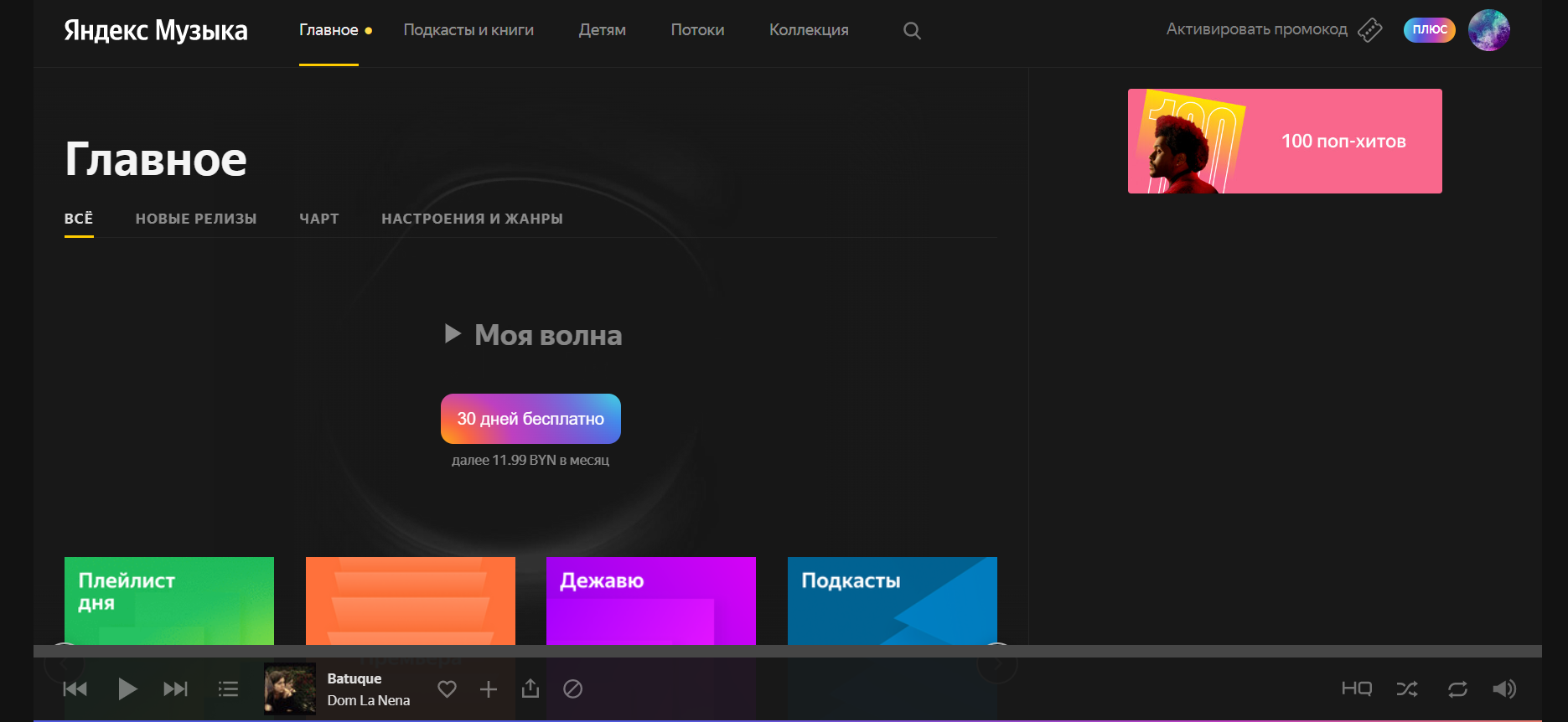


Рисунок 1.1 – Домашняя сстраница Яндекс Музыки

Так же были проанализированы основные сущности базы данных данного приложения, такие как:

− Track — трек (аудио композиция);

− Album — альбом (сборник треков);

− Artist — исполнитель (исполнитель или композитор трека, альбома);

− Genre — музыкальный жанр;

− Label — лейбл звукозаписи;

### Spotify

Spotify является самым известным музыкальным стриминговым сервисом на данный момент. Его функционал примерно такой же, как и у Яндекс музыки. Отличия имеются, однако они не значительны в контексте нашего курсового проекта. Интерфейс данного приложения представлен ниже на рисунке 1.2

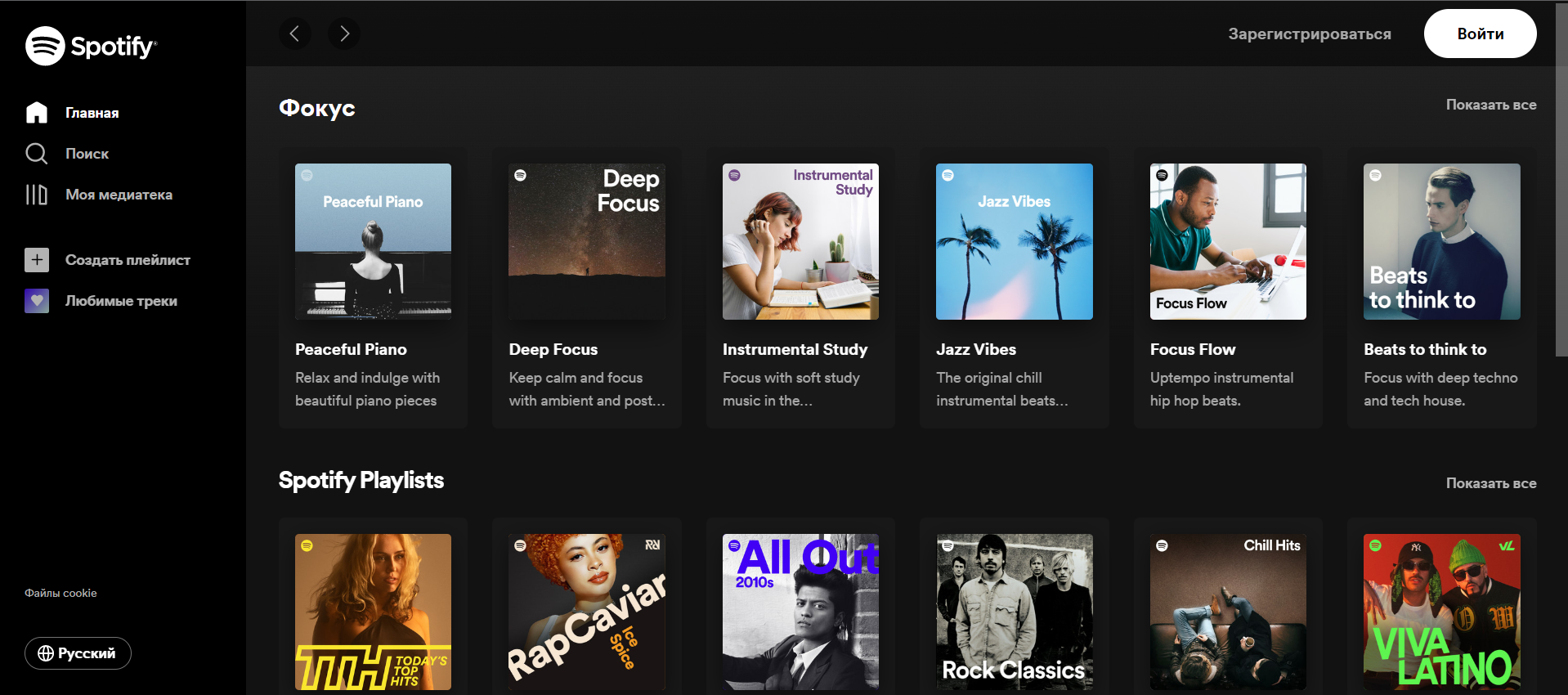


Рисунок 1.2 – Домашняя страница Spotify

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

### Apple Music

По популярности Apple Music идёт следом за Spotify, с 56 миллионами пользователей и большей по размеру библиотекой. Рассмотрим эту площадку на рисунке 1.3.

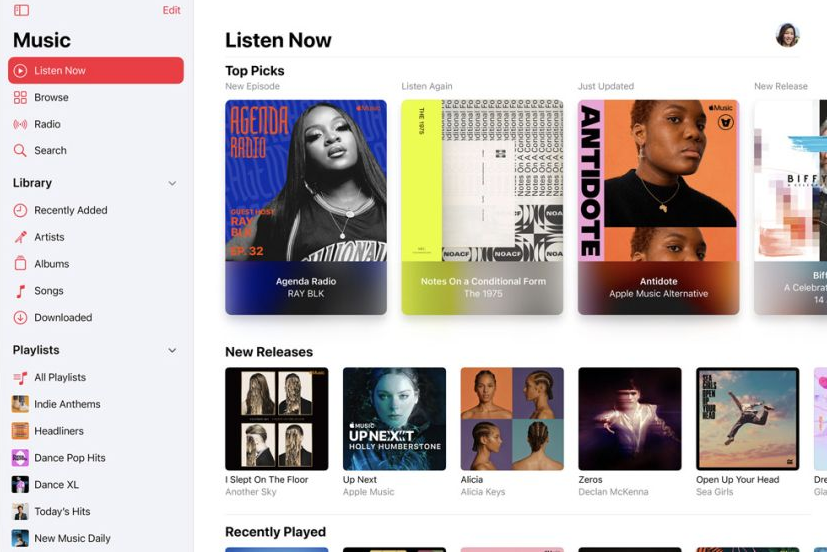


Рисунок 1.3 – Домашняя страница Apple Music

Разобрав приведённые выше сервисы 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 Вывод по разделу

В данном разделе был представлен аналитический обзор литературы. Были рассмотрены такие сервисы, как Яндекс Музыка, Spotify и Apple Music.

В результате анализа было выявлено, что данные сервисы обладают сходным функционалом, таким как: возможность прослушивания музыки онлайн, создание персональных плейлистов, предоставление рекомендаций пользователю, наличие платной подписки и т.д.

По результатам первого раздела мы получили понимание о предметной области разработки, сформировали требования к разрабатываемой базе данных.

## 2 Проектирование

При разработке приложения для курсового проекта была использована база данных Oracle 12c.

## 2.1 Таблицы базы данных

Одним из ключевых моментов при проектировании и создании базы данных является грамотный анализ предметной области базы данных. Далее необходимо составить такую модель данных, которая будет правильно отражать то, как с этими с данными в общем, и этой моделью, в частности, подразумевается взаимодействовать.

Результатом корректно проведённого анализа, проектирования, и разработки, является модель, способная предоставить функционально все необходимые возможности для конкретного типа пользователя.

На рисунке 2.1 отображена модель базы данных, полученная в ходе анализа предметной области и необходимого функционала. Между сущностями, представленными таблицами, также настроены связи, отражающие их взаимосвязь.

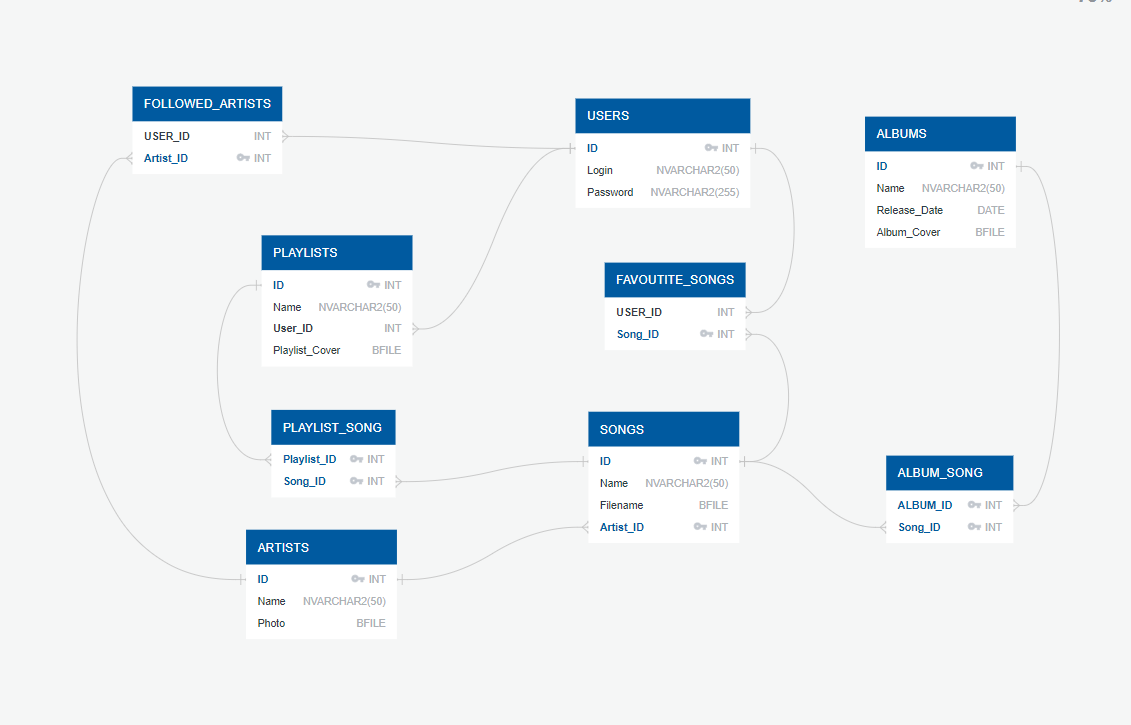


Рисунок 2.1 – Схема базы данных

Для реализации базы данных стримингового сервиса было разработано 7 таблиц, диаграмма со структурой связей которых представлена в приложении А.

В структуру базы данных входят следующие таблицы: USERS, ARTISTS, PLAYLISTS, SONGS, ALBUMS, PLAYLIST\_SONG, ARTIST\_SONG. Далее представлено описание каждой из них.

Таблица USERS используется для хранения данных о пользователе, которые представляют собой имя пользователя и его пароль. Её структура представлена в таблице 2.1.

Таблица 2.1 – Структура таблицы USERS

|  |  |  |
| --- | --- | --- |
| Столбец | Тип, ограничение целостности | Описание |
| ID | int, primary key | Идентификатор пользователя |
| Login | unique | Логин пользователя |
| Password | nvarchar2(255), not null | Хеш пароля |

Таблица ARTISTS используется для хранения данных об исполнителях, которые представляют собой уникальный идентификатор исполнителя, его имя и изображение. Её структура представлена в таблице 2.2.

Таблица 2.2 – Структура таблицы ARTISTS

|  |  |  |
| --- | --- | --- |
| Столбец | Тип, ограничение целостности | Описание |
| ID | int, primary key | Идентификатор исполнителя |
| Name | nvarchar2(50), not null | Имя исполнителя |
| Photo | bfile | Изображение исполнителя |

Таблица ALBUMS используется для хранения данных об альбомах, которые представляют собой уникальный идентификатор альбома, его название, дату выпуска и файл обложки альбома. Её структура представлена в таблице 2.3.

Таблица 2.3 – Структура таблицы ALBUMS

|  |  |  |
| --- | --- | --- |
| Столбец | Тип, ограничение целостности | Описание |
| ID | int, primary key | Идентификатор альбома |
| Name | nvarchar2(50), not null | Название альбома |
| Release\_Date | date, not null | Дата выпуска альбома |
| Album\_Cover | blob, not null | Обложка альбома |

Таблица SONGS используется для хранения данных о треках, которые представляют собой уникальный идентификатор трека, его название, айдиофайл и альбом. Её структура представлена в таблице 2.4.

Таблица 2.4 – Структура таблицы SONGS

|  |  |  |
| --- | --- | --- |
| Столбец | Тип, ограничение целостности | Описание |
| ID | int, primary key | Идентификатор песни |
| Name | nvarchar2(50), not null | Название песни |
| Filename | blob, not null | Аудиофайл трека |
| Artist\_ID | int, foreign key | Идентификатор исполнителя |

Таблица ALBUM\_SONG используется для связи многие ко многим альбома и песни. Её структура представлена в таблице 2.5.

Таблица 2.5 – Структура таблицы ALBUM\_SONG

|  |  |  |
| --- | --- | --- |
| Столбец | Тип, ограничение целостности | Описание |
| Album\_ID | int, foreign key, primary key | Идентификатор альбома |
| Song\_ID | int, foreign key, primary key | Идентификатор песни |

Таблица PLAYLIST используется для хранения данных о плейлистах, которые представляют собой уникальный идентификатор плейлиста, его название, пользователь, которому принадлежит плейлист и обложка плейлиста. Её структура представлена в таблице 2.6.

Таблица 2.6 – Структура таблицы PLAYLISTS

|  |  |  |
| --- | --- | --- |
| Столбец | Тип, ограничение целостности | Описание |
| ID | int, primary key | Идентификатор плейлиста |
| Name | nvarchar2(50), not null | Название плейлиста |
| User\_ID | int, foreign key | Идентификатор пользователя |
| Playlist\_Cover | bfile | Обложка плейлиста |

Таблица PLAYLIST\_SONG используется для связи многие ко многим плейлиста и песни. Её структура представлена в таблице 2.7.

Таблица 2.7 – Структура таблицы PLAYLIST\_SONG

|  |  |  |
| --- | --- | --- |
| Столбец | Тип, ограничение целостности | Описание |
| Playlist\_ID | int, foreign key, primary key | Идентификатор плейлиста |
| Song\_ID | int, foreign key, primary key | Идентификатор песни |

Таблица FAVOURITE\_SONG используется для хранения понравившихся пользователю песен. Её структура представлена в таблице 2.8.

Таблица 2.8 – Структура таблицы PLAYLIST\_SONG

|  |  |  |
| --- | --- | --- |
| Столбец | Тип, ограничение целостности | Описание |
| User\_ID | int, foreign key, primary key | Идентификатор пользователя |
| Song\_ID | int, foreign key, primary key | Идентификатор песни |

Таблица FOLLOWED\_ARTISTS используется для хранения подписок пользователей на исполнителей. Её структура представлена в таблице 2.9.

Таблица 2.9 – Структура таблицы PLAYLIST\_SONG

|  |  |  |
| --- | --- | --- |
| Столбец | Тип, ограничение целостности | Описание |
| User\_ID | int, foreign key, primary key | Идентификатор пользователя |
| Artist\_ID | int, foreign key, primary key | Идентификатор исполнителя |

Таким образом, было разработано 9 таблиц базы данных для хранения информации о пользователях, исполнителях, альбомах, песнях, плейлистах, а также информацию об отмеченных пользователями песнях и отслеживаемых ими исполнителях.

## 2.2 Пользователи базы данных

База данных содержит 2 основных пользователя: администратора и клиента. Пользователи базы данных могут взаимодействовать с базой данных при помощи процедур, описанных в разделе 2.3

Каждый пользователь имеет профиль. Профиль администратора представлен на листинге 2.1

CREATE PROFILE ADMIN\_PROFILE LIMIT

PASSWORD\_LIFE\_TIME 260

SESSIONS\_PER\_USER 5

FAILED\_LOGIN\_ATTEMPTS 10

PASSWORD\_LOCK\_TIME 60

CONNECT\_TIME 720

Листинг 2.1 – Профиль администратора

Профиль клиента представлен на листинге 2.2

CREATE PROFILE CLIENT\_PROFILE LIMIT

PASSWORD\_LIFE\_TIME 260

SESSIONS\_PER\_USER 100

FAILED\_LOGIN\_ATTEMPTS 10

PASSWORD\_LOCK\_TIME 60

CONNECT\_TIME 360

Листинг 2.2 – Профиль клиента

Для каждого пользователя создается его собственная роль. Роль – поименованный набор привилегий. Роли выдаются привилегии на пакеты, в которых содержаться процедуры, предназначенные для вызова. Создание роли клиента и выдача ей привилегий представлена на листинге 2.3

CREATE ROLE CLIENT\_ROLE;

GRANT CONNECT TO CLIENT\_ROLE;

GRANT EXECUTE ON GENERAL\_PACKAGE TO CLIENT\_ROLE;

GRANT EXECUTE ON CLIENT\_PACKAGE TO CLIENT\_ROLE;

GRANT CLIENT\_ROLE TO CLIENT;

Листинг 2.3 – Роль клиента

После создания роли и назначения ей привилегий, необходимо создать пользователя и присвоить ему роль, созданную ранее. Создание пользователя *CLIENT* приведено на листинге 2.4.

CREATE USER CLIENT IDENTIFIED BY CLIENT

PROFILE CLIENT\_PROFILE;

Листинг 2.4 – Создание пользователя *CLIENT*

Остальные пользователи базы данных создаются по тому же принципу, но их ролям присваивается привилегия на вызов процедур и функций из других пакетов.

## 2.3 Представления базы данных

Представление – это специфический образ таблицы или набора таблиц, определенный оператором SELECT.

Для удобства и повышения производительности при обращении с базой данных были разработаны следующие представления.

Представление ARTIST\_ALBUM\_SONG\_VIEW, которое соединяет таблицы ARTISTS, ALBUMS, SONGS и ALBUM\_SONG по их взаимосвязям. Это представление включает в себя информацию об артистах, альбомах и песнях. Реализация этого представления приведена в листинге 2.5.

CREATE OR REPLACE VIEW ARTIST\_ALBUM\_SONG\_VIEW AS

SELECT ARTISTS.ARTIST\_ID, ARTISTS.ARTIST\_NAME,

ALBUMS.ALBUM\_ID, ALBUMS.ALBUM\_NAME, ALBUMS.ALBUM\_RELEASE\_DATE, ALBUMS.ALBUM\_COVER,

SONGS.SONG\_ID, SONGS.SONG\_NAME, SONGS.SONG\_FILE

FROM ALBUMS JOIN ALBUM\_SONG ON ALBUMS.ALBUM\_ID = ALBUM\_SONG.ALBUM\_ID

JOIN SONGS ON ALBUM\_SONG.SONG\_ID = SONGS.SONG\_ID

JOIN ARTISTS ON SONGS.ARTIST\_ID = ARTISTS.ARTIST\_ID;

Листинг 2.5 – Представление ARTIST\_ALBUM\_SONG\_VIEW

Аналогичным образом создаются представления PLAYLIST\_USER\_VIEW, PLAYLIST\_SONG\_VIEW и FAVS\_USER\_SONG\_ARTIST\_VIEW, которые также объединяют соответствующие таблицы базы данных.

Представление PLAYLIST\_USER\_VIEW содержит информацию о пользователях и их плейлистах. Представление PLAYLIST\_SONG\_VIEW содержит информацию о песнях и плейлистах, к которым они принадлежат.

Представление FAVS\_USER\_SONG\_ARTIST\_VIEW включает информацию о песнях и артистах, которые отметил пользователь как избранные.

Представление FAVORITE\_SONG\_ID\_COUNT\_VIEW позволяет узнать, сколько пользователей добавило каждую песню в качестве любимой. Представление TOP\_FAVOURITE\_SONGS использует эту информацию, чтобы пользователи могли получить информацию о наиболее популярных песнях.

Аналогичным образом, представление TOP\_FOLLOWED\_ARTISTS\_VIEW позволяет получить информации о наиболее популярных артистах, за которыми следят пользователи.

Таким образом, созданные представления облегчают доступ к информации в базе данных, объединяя необходимые таблицы и отображая только необходимые столбцы данных.

## 2.4 Триггеры базы данных

Триггер — хранимая процедура особого типа, которую пользователь не вызывает непосредственно, а исполнение которой обусловлено действием по модификации данных: добавлением INSERT, удалением DELETE строки в заданной таблице, или изменением UPDATE данных в определённом столбце заданной таблицы реляционной базы данных.

Для хеширования пароля пользователя был создан триггер *hash*\_*password*. Данный триггер срабатывает до добавления или изменения записи в таблице USERS. Реализация этого триггера приведена в листинге 2.6.

CREATE OR REPLACE TRIGGER HASH\_PASSWORD

BEFORE INSERT OR UPDATE OF USER\_PASSWORD ON USERS FOR EACH ROW

BEGIN

-- SHA256

:NEW.USER\_PASSWORD := DBMS\_CRYPTO.HASH(UTL\_RAW.CAST\_TO\_RAW(:NEW.USER\_PASSWORD), 4);

END;

Листинг 2.6 – Триггер hash\_password

Данный тригер применяется к каждой строке на столбце "USER\_PASSWORD". Реализация триггера осуществляется при помощи встроенной функции хэширования SHA256, предоставляемой пакетом DBMS\_CRYPTO.

## 2.5 Хранимые процедуры

Хранимая процедура – объект базы данных, представляющий собой набор SQL-инструкций, который компилируется один раз и хранится на сервере. Хранимые процедуры, созданные в данном курсовом проекте, содержат обработку исключений для того, чтобы ограничить пользователя от возможных ошибок.

Для управления данными, пользователи вызывают процедуры и функции, которые располагаются в пакетах. Пакет Oracle PL/SQL – это объект БД, который группирует логически связанные типы, элементы и подпрограммы. В базе данных используется три пакета: general\_package, client\_package и admin\_package описание которых представлено ниже.

### 2.5.1 Общий пакет

Пакет general\_package содержит 14 процедур, общих для всех пользователей базы данных, реализация этого пакета приведена в приложении Б.

Процедура get\_artist\_by\_name необходима для выборки из базы данных исполнителей по имени.

Процедура get\_artist\_by\_id необходима для выборки из базы данных исполнителей по идентификатору исполнителя.

Процедура get\_all\_artists необходима для выборки из базы данных всех исполнителей.

Процедура get\_all\_songs необходима для выборки из базы данных всех песен.

Процедура get\_song\_by\_id необходима для выборки из базы данных песен по идентификатору.

Процедура get\_song\_by\_artist необходима для выборки из базы данных песен по исполнителю.

Процедура get\_all\_albums необходима для выборки из базы данных всех альбомов.

Процедура get\_album\_by\_id необходима для выборки из базы данных альбомов по идентификатору.

Процедура get\_songs\_by\_album необходима для выборки песен из базы данных по альбому.

Процедура get\_albums\_by\_song необходима для получения из базы данных, в каких альбомах есть песня.

Процедура get\_artist\_followers\_amount необходима для получения количества подписчиков исполнителя.

Процедура get\_top\_followed\_artists необходима для получения исполнителей, отсортированных по количеству подписчиков.

### 2.5.2 Пакет клиента

Пакет client\_package содержит 16 процедур, предназначенных для взаимодействия с базой данной на уровне клиента, реализация этого пакета приведена в приложении В.

Процедура register\_user необходима для добавления нового пользователя в базу данных.

Процедура log\_in\_user необходима для авторизации пользователя по логину и паролю. Данная процедура вычисляет хеш для входного параметра пароля и сравнивает его с хранимым в базе данных паролем.

Процедура delete\_user необходима для удаления пользователя из базы данных.

Процедура update\_user необходима для изменения информации пользователя в базе данных.

Процедура create\_playlistпозволяют пользователю создавать новый плейлист.

Процедура get\_playlist\_by\_userнеобходима для получения всех плейлистов пользователя.

Процедура update\_playlist необходима для изменения информации о плейлисте в базе данных.

Процедура delete\_playlist необходима для удаления информации о плейлисте из базы данных.

Процедура add\_song\_to\_playlist необходима для добавления песни в плейлист.

Процедура delete\_song\_from\_playlist необходима для удаления песни из плейлиста.

Процедура add\_song\_to\_favourite необходима для добавления песни в понравившееся.

Процедура delete\_song\_from\_favourite необходима для удаления песни из понравившегося.

Процедура get\_user\_favourites необходима для получения понравившихся пользователю песен.

Процедура follow\_artist необходима для подписки пользователя на исполнителей.

Процедура unfollow\_artist необходима для отмены подписки пользователя на исполнителей.

Процедура get\_followed\_artist необходима для получения исполнителей, на которых подписан пользователь.

### 2.5.3 Пакет администратора

Пакет admin\_package содержит 12 процедур, предназначенных для взаимодействия с базой данной на уровне администратора, реализация этого пакета приведена в приложении Г.

Процедура delete\_user необходима для удаления пользователя из базы данных.

Процедура create\_artist необходима для добавления нового исполнителя в базу данных.

Процедура update\_artist необходима для изменения данных исполнителя в базе данных.

Процедура delete\_artist необходима для удаления исполнителя из базы данных.

Процедура create\_album необходима для добавления нового альбома в базу данных.

Процедура update\_album необходима для изменения данных альбома в базе данных.

Процедура delete\_album необходима для удаления альбома из базы данных.

Процедура create\_song необходима для добавления новой песни в базу данных.

Процедура update\_song необходима для изменения данных песни в базе данных.

Процедура delete\_song необходима для удаления песни из базы данных.

Процедура add\_song\_to\_album необходима для добавления песни в альбом.

Процедура delete\_song\_from\_album необходима для удаления песни из альбома.

### 2.5.4 Процедуры экспорта и импорта таблиц в XML

Для реализации обоих процедур потребовалось создать директорий с желаемым расположением при подключении под пользователем SYS к инстансу и выдать пользователю, владеющему базой данных, гран на чтение и запись из этого директория.

В рамках задания нужно было разработать процедуру, экспортирующую данные из таблицы в XML документ. С этой целью разработана процедура export\_users\_to\_xml.

Также нужно было написать процедуру, которая бы, наоборот, импортировала существующие данные из XML документа в таблицу базы данных. Для этого была разработана процедура import\_USERS\_from\_xml.

Реализация данных процедур приведена в приложении Д.

## 2.6 Вывод по разделу

В данном разделе были спроектированы основные объекты базы данных. Были подробно описаны таблицы с информацией о типе данных полей, ограничений целостности и описание данного поля, приведена UML диаграмма базы данных. Также был описан порядок создания пользователей, их профилей и ролей, а также создания представлений и триггеров. Разработанные процедуры были разделены на 3 пакета и выданы соответствующим ролям.

## 3 Тестирование и улучшение производительности

Производительность базы данных — это способность базы данных обрабатывать запросы и транзакции быстро и эффективно. Высокая производительность базы данных может быть достигнута с помощью эффективного использования индексов, хорошей организации структуры базы данных и оптимизации запросов. Хорошая производительность базы данных важна для обеспечения эффективной работы системы и удовлетворения потребностей пользователей.

Для тестирования производительности в таблицу ALBUMS было добавлено 100000 строк. Для данной задачи мы создали анонимный блок, и так как работаем именно с действующей базой данных, то можем использовать готовую процедуру для вставки данных в таблицу.

Разработанный анонимный блок позволяет добавить большое количество строк за одно выполнение (листинг 3.1).

BEGIN

FOR i IN 1 .. 100000

LOOP

insert into ALBUMS (ALBUM\_NAME, ALBUM\_RELEASE\_DATE) VALUES ('ALBUM' || i, TO\_DATE('2003/05/03 21:02:44', 'yyyy/mm/dd hh24:mi:ss'));

END LOOP;

END;

Листинг 3.1– Анонимный блок для заполнения таблицы большим количеством данных

Был сделан запрос, который запрашивает песню с определённым названием. После получения плана запроса его стоимость была 171, что видно на рисунке 3.1.

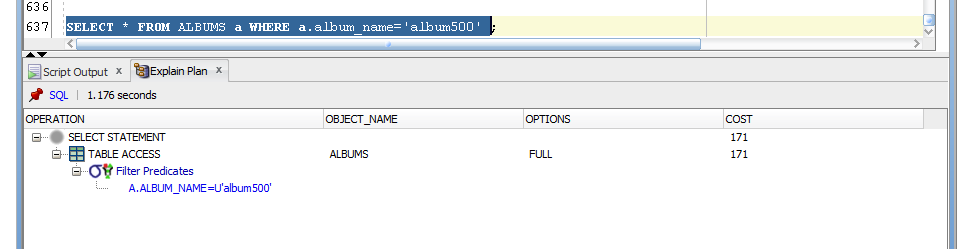


Рисунок 3.1– Стоимость заброса до добавления индекса

После был добавлен пространственный индекс на таблицу ALBUMS, после чего стоимость запроса стала 2, что мы видим на рисунке 3.2.

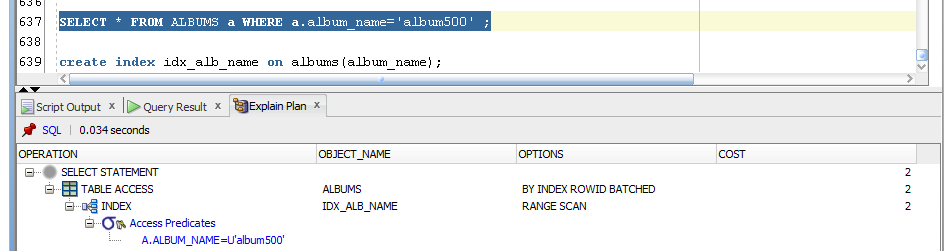


Рисунок 3.2– Стоимость заброса после добавления индекса

Соответственно мы можем сделать вывод, что индексы могут улучшить производительность базы данных за счет ускорения поиска и обработки данных. Они создаются для определенных столбцов и позволяют ускорить поиск строк в таблице по этим столбцам. Индексы также могут ускорить сортировку и группировку данных

## 3.1 Вывод по разделу

В данном разделе было проведено нагрузочное тестирование для проверки наличия проблем с производительностью и масштабируемостью при различном уровне нагрузки. В нашем случае в качестве тестирования производительности было протестировано заполнение одной из таблиц 100000 строк. После добавления индекса, можно сделать вывод, что тестирование прошло успешно.

## 4 Описание технологии

В качестве основной технологии было использование мультимедийных типов данных в базах данных. Мультимедийные типы данных — это типы данных, которые используются для хранения мультимедийных файлов, таких как аудио, видео, изображения и т.д. Эти файлы обычно больше по размеру, чем текстовые файлы или файлы таблиц, поэтому они требуют специальных типов данных для хранения.

Для работы с данными большого объема СУБД Oracle предоставляет типы данных BLOB, CLOB, NCLOB и BFILE. Здесь LOB означает Large OBject, или большой объект, и далее по тексту термины LOB и "большой объект" взаимозаменяемы. По сути, большой объект – это абстрактный тип для манипуляции данными большого объема внутри БД, а типы BLOB, CLOB, NCLOB и BFILE – его конкретные реализации.

BLOB (Binary Large OBject) – это тип данных, который используется в базах данных для хранения больших бинарных объектов, таких как изображения, видео и аудиофайлы. BLOB может хранить любые бинарные данные, независимо от их размера. CLOB (Character Large OBject) – это тип данных, который используется для хранения больших текстовых объектов. CLOB может хранить до нескольких мегабайт текста, в отличие от типа данных VARCHAR, который обычно ограничен размером в несколько сотен килобайт. NCLOB (National Character Large OBject) – это тип данных, который используется в базах данных для хранения больших текстовых объектов, которые содержат символы национальных алфавитов. NCLOB хранит текст в кодировке Unicode, которая поддерживает большое количество различных языков и алфавитов.

LOB можно управлять с помощью интерфейса вызовов Oracle, OCI или пакета PL / SQLDBMS\_LOB. Вы можете писать функции, включая методы для типов объектов, которые могут содержать LOBs, для управления частями LOBs.

В данном курсовом проекте технология мультимедийных типов данных отражена в необходимости хранить изображения, являющиеся обложками музыкальных альбомов, плейлистов и фотографиями исполнителей, а также аудиофайлы, без которых все приложение потеряло бы смысл.

Для хранения в поле изображений и аудиофайлов в СУБД применяется тип BLOB. Тип BLOB представляет из себя массив двоичных данных, поэтому процесс сводился к разбиению мультимедийных файлов на массив битов при загрузке их в базу данных, и к сборке изображения или аудиофайла из массива битов на обратном пути.

## 4.1 Вывод по разделу

В данном разделе была подробна описана технология мультимедийных типов данных, описана её важность при разработке базы данных, разработаны все необходимые процедуры для реализации данной технологии.

## 5 Руководство пользователя

Разработанный в рамках курсового проекта музыкальный сервис поддерживает два вида ролей у пользователя: администратор и клиент.

Смысл аккаунта клиента сводится к поиску песен, прослушиванию их, добавлению в плейлист, подписки на исполнителей, добавление в понравившееся песен.

Администратор же может делать то же, что и пользователь, с расширенным функционалом: администратор может добавлять, изменять и удалять исполнителей, альбомы и песни, а также имеет доступ к окну администратора (через меню «Settings») со списком пользователей и кнопками для запуска импорта и экспорта XML.

Роль администратора дополняет функционал обычного пользователя, поэтому будет уместно в рамках руководства осветить возможности аккаунта администратора.

При запуске приложения открывается окно авторизации (рисунок 5.1), где зарегистрированному пользователю следует ввести свой логин и пароль.

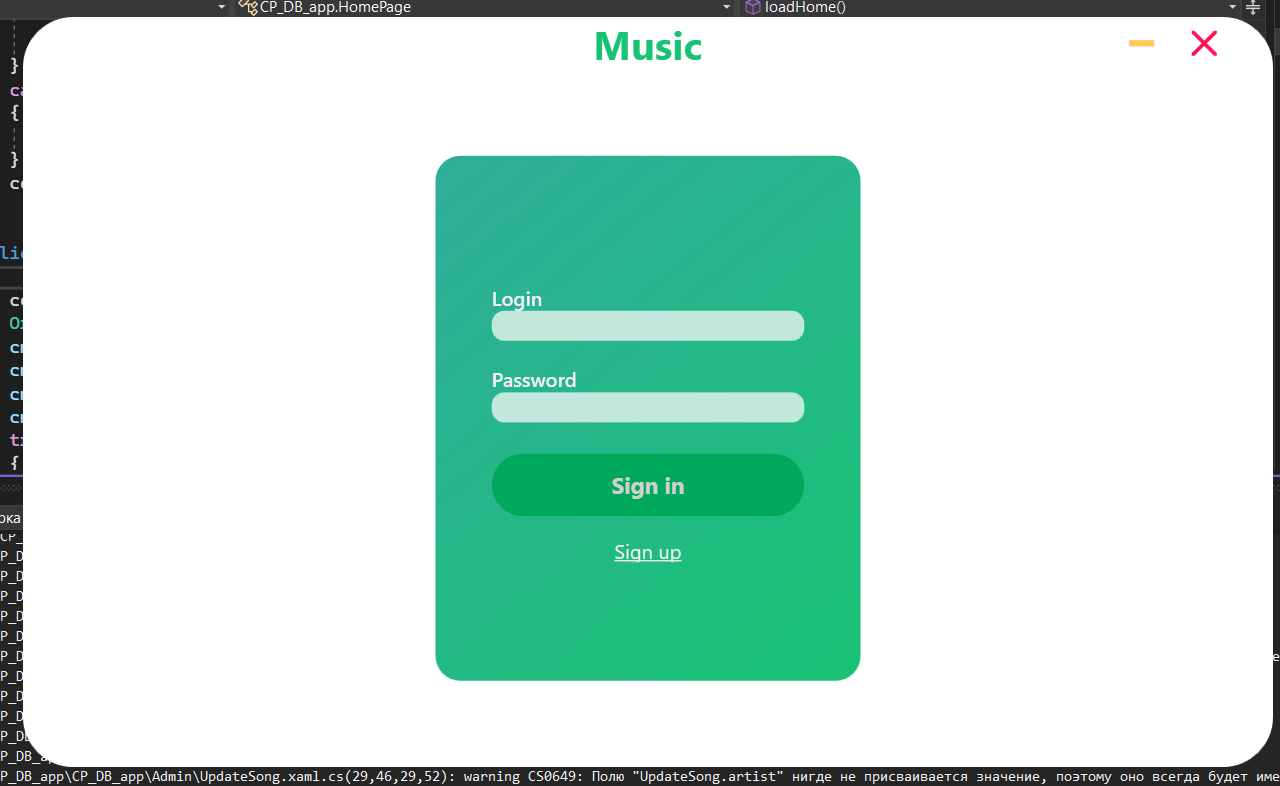


Рисунок 5.1 – Окно авторизации

При неправильном вводе пароля пользователю выводится сообщение о неправильном пароле (рисунок 5.2).

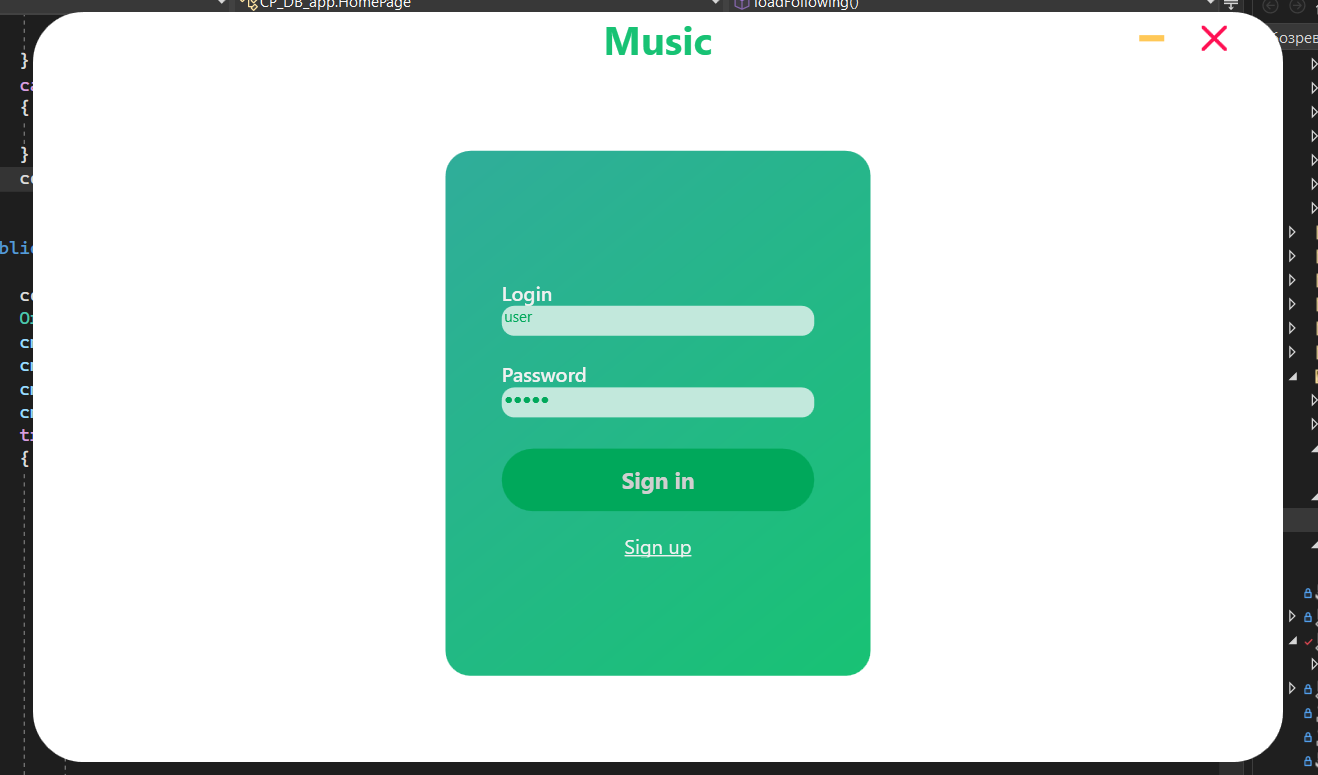


Рисунок 5.2 – Окно авторизации, неверный логин или пароль

При успешной авторизации пользователь будет перенаправлен на главную страницу приложения.

Неавторизованный пользователь не может зайти на главную страницу, пока не авторизуется. Для этого следует нажать на кнопку «Sign Up» в окне авторизации. Окно авторизации изображено на рисунке 5.3.

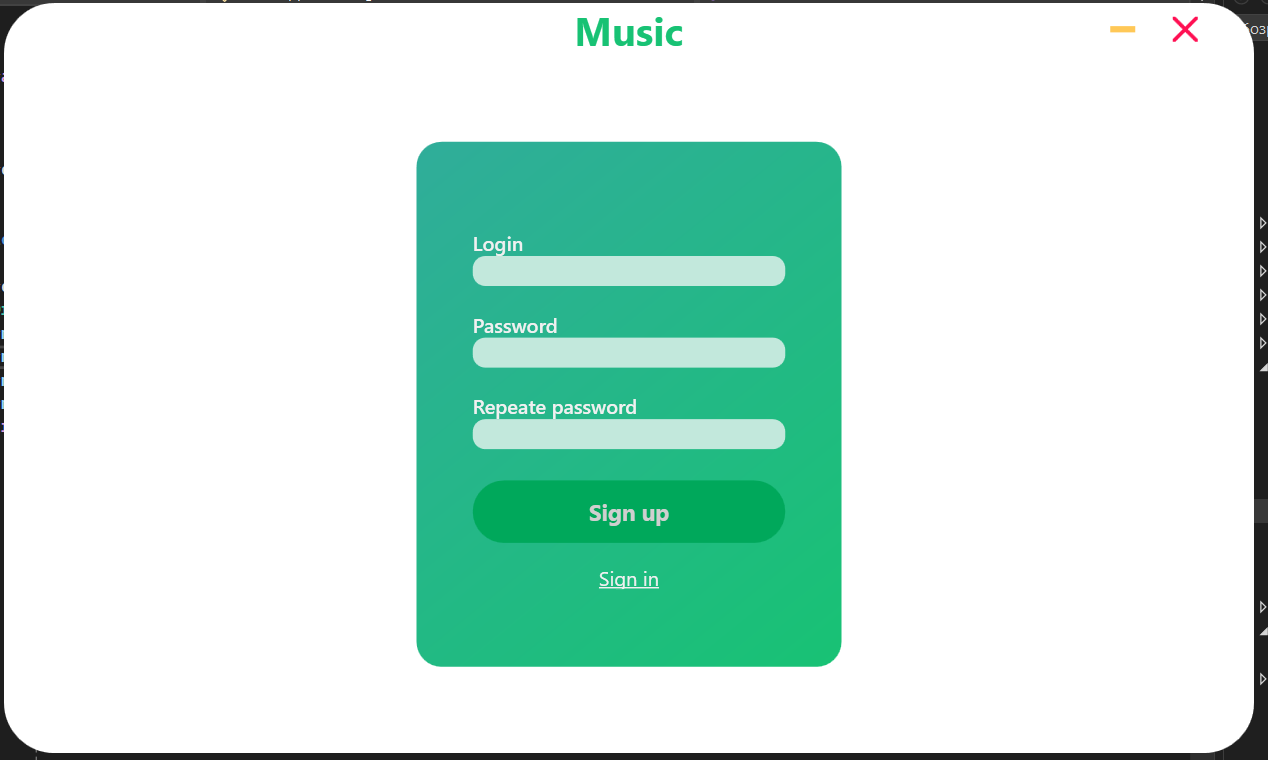


Рисунок 5.3 – Окно регистрации

Если в этом окне ввести логин пользователя, уже существующего в базе данных, пользователь увидит следующую ошибку (рисунок 5.4).

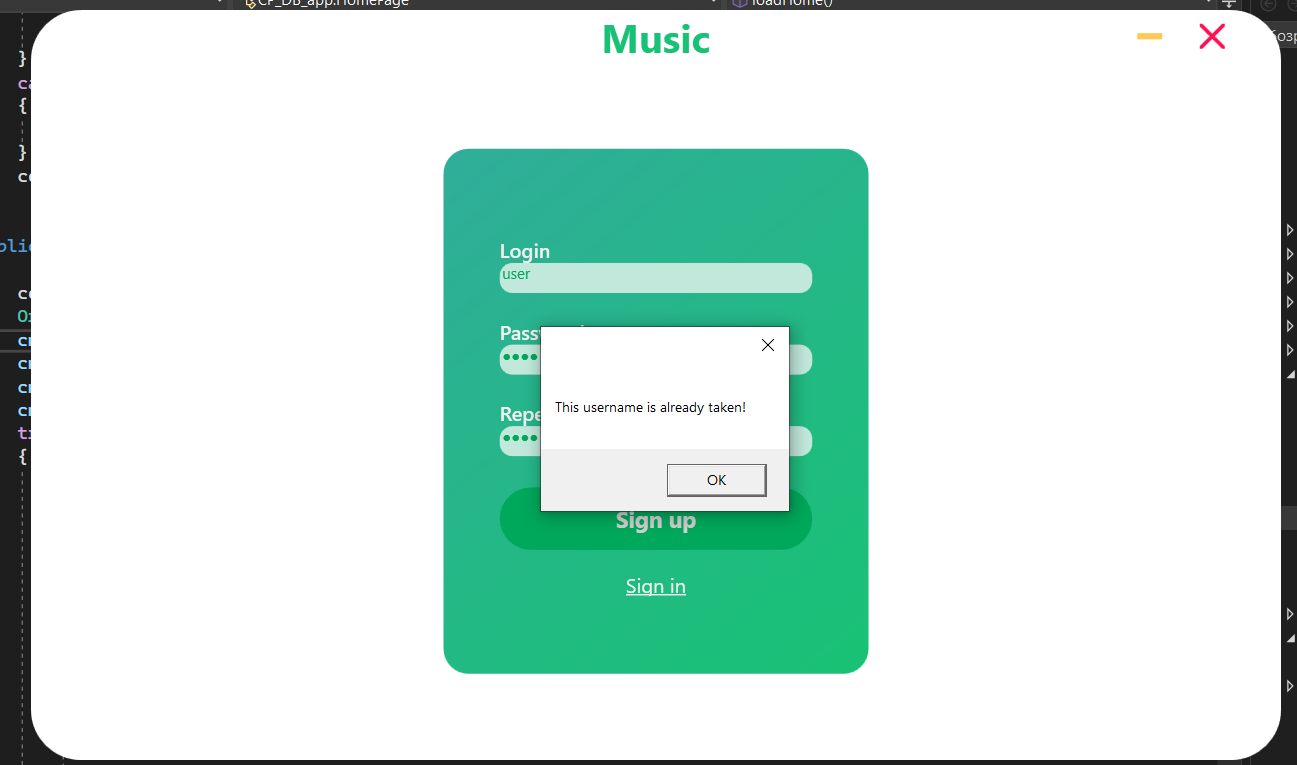


Рисунок 5.4 – Ввод занятого логина

Также в рамках данного окна обрабатывается ошибка при введении несовпадающих паролей.

Если пользователь успешно вошел в свой аккаунт, ему открывается домашняя страница приложения (рисунок 5.5).

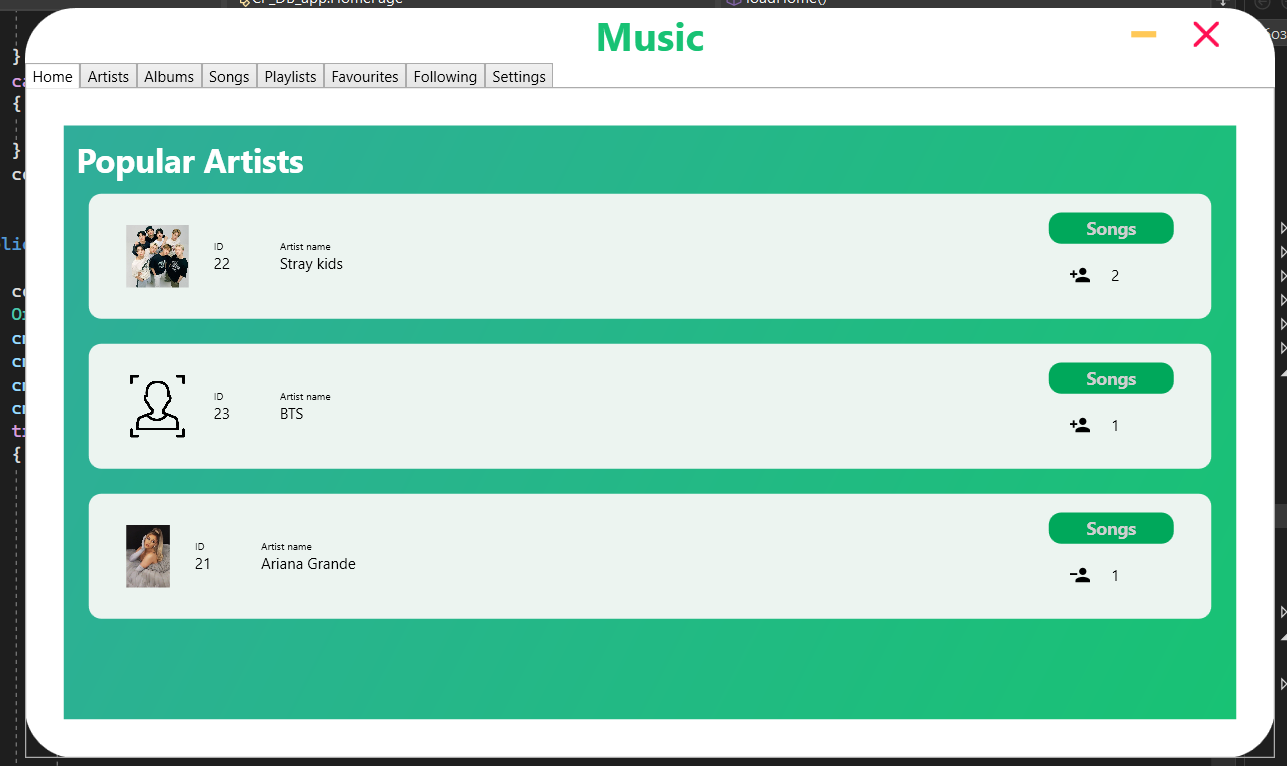


Рисунок 5.5 – Домашняя страница приложения

Попав на домашнюю страницу, по умолчанию открывается вкладка «Home», на которой представлен топ исполнителей по количеству подписок. Если обратить внимание на меню, в нем также имеются вкладки «Artists», «Albums», «Songs», «Playlists», «Favourites», «Following» и «Settings». Данные вкладки доступны всем пользователям.

Вкладка «Artists» содержит список исполнителей, состоящий из плиток с кнопками. На этой вкладке можно осуществлять поиск по имени исполнителя, воспользовавшись текстовый полем вверху. Вкладка «Artists» изображена на рисунке 5.6.

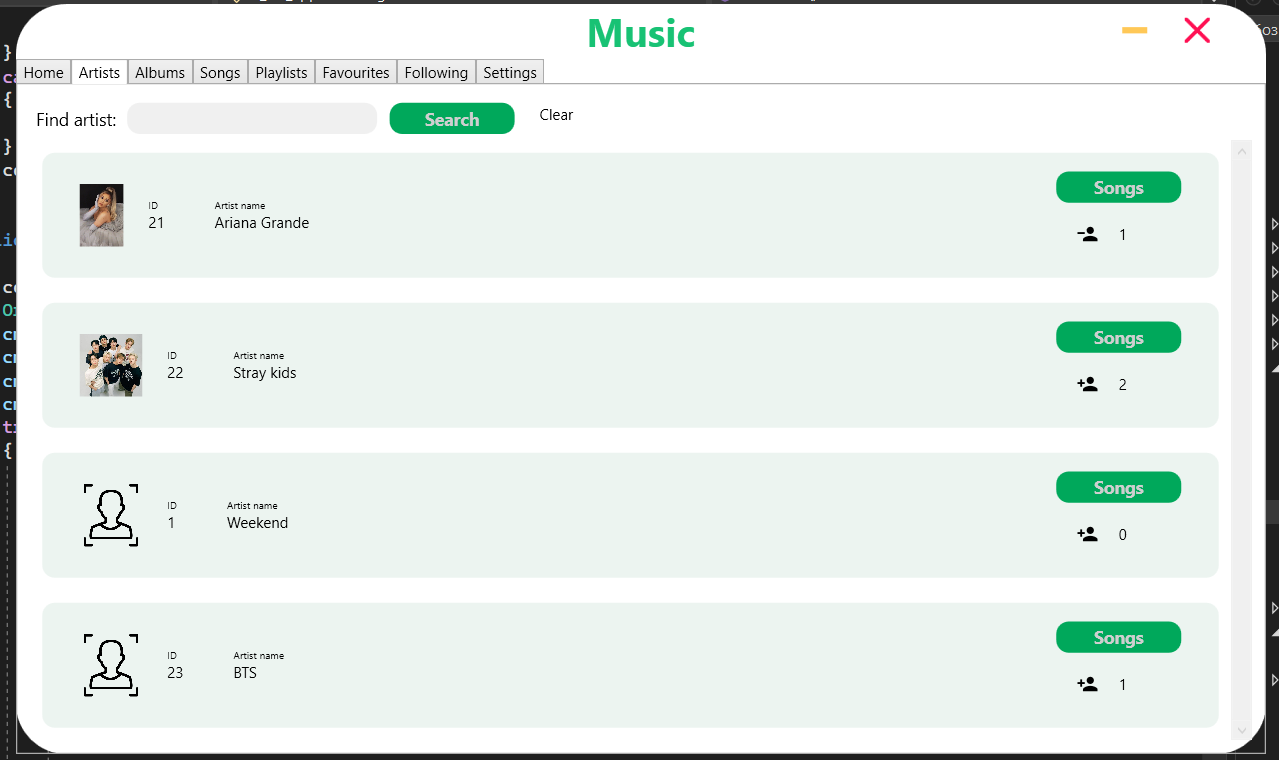


Рисунок 5.6 – Вкладка «Artists»

Всем пользователям доступны кнопки управления подпиской на исполнителя, представленные иконками. Администратору доступны кнопки управления существующими исполнителями, а также кнопка добавления нового исполнителя (рисунок 5.7).

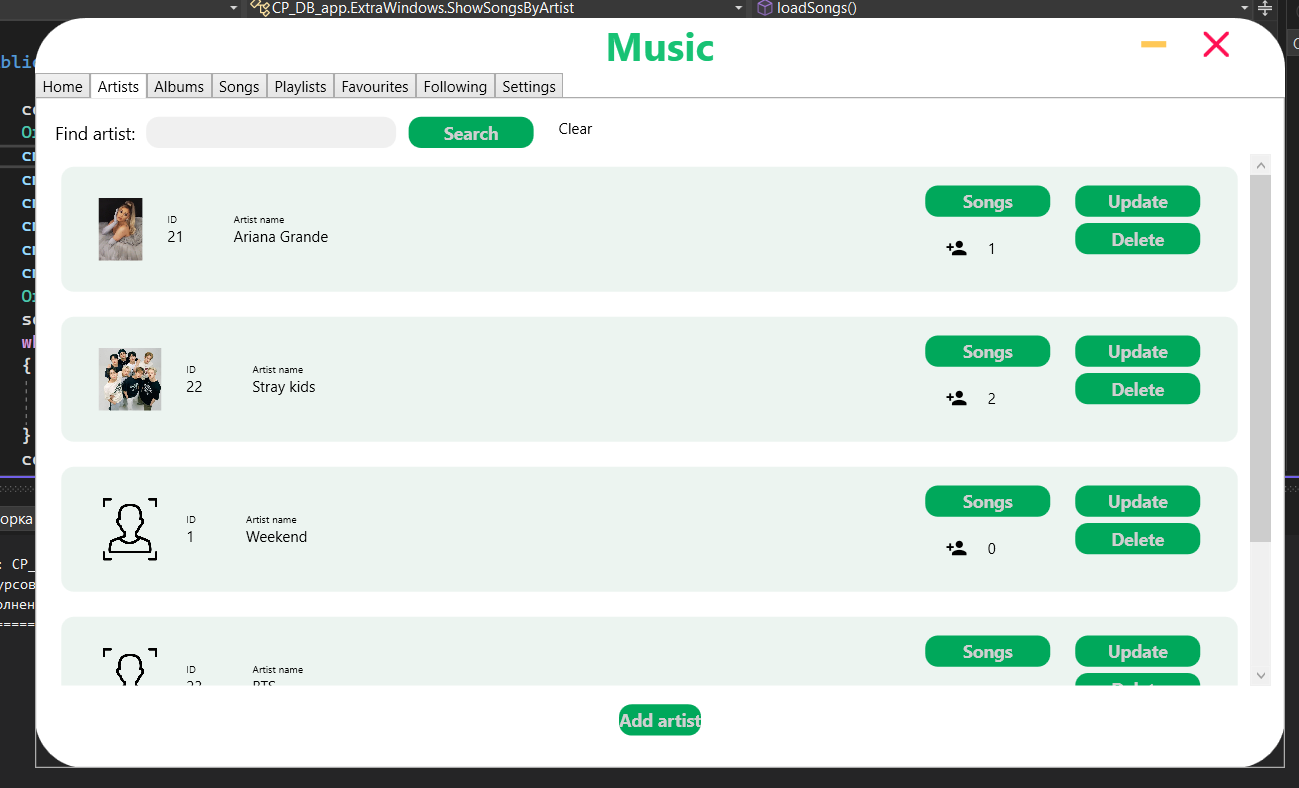


Рисунок 5.7 – Вкладка «Artists» администратора

При нажатии на кнопку «Add artist» откроется новое окно для добавления нового исполнителя (рисунок 5.8).

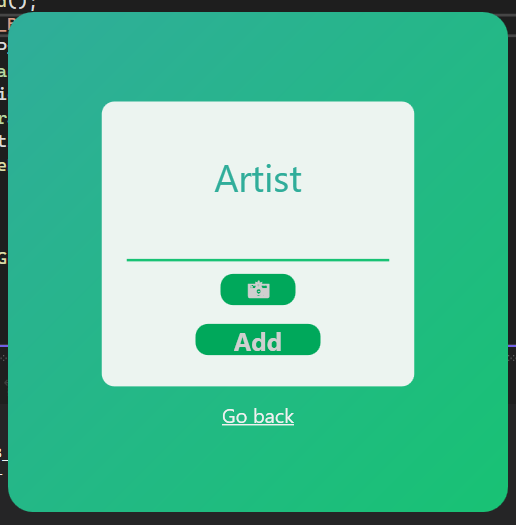


Рисунок 5.8 – Окно добавления нового исполнителя

В данном окне необходимо ввести имя исполнителя и выбрать изображение исполнителя. Если администратор введет имя уже существующего исполнителя, он увидит ошибку, представленную на рисунке 5.9.

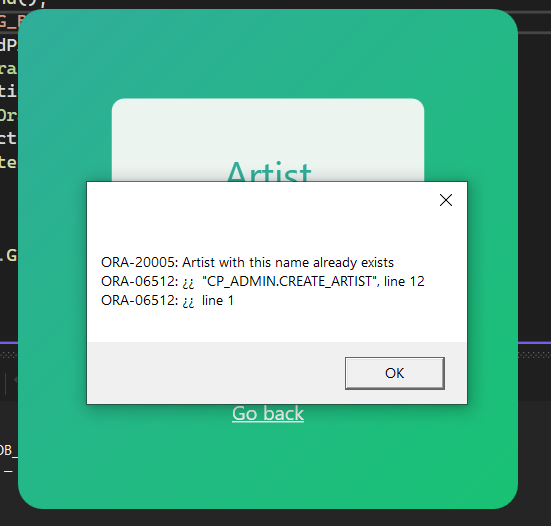


Рисунок 5.9 – Ввод занятого имени исполнителя

Также любой пользователь может нажать на кнопку «Songs» напротив имени исполнителя, и тогда попадет на окно со списком всех имеющихся песен этого исполнителя. Это окно можно увидеть на рисунке 5.10.

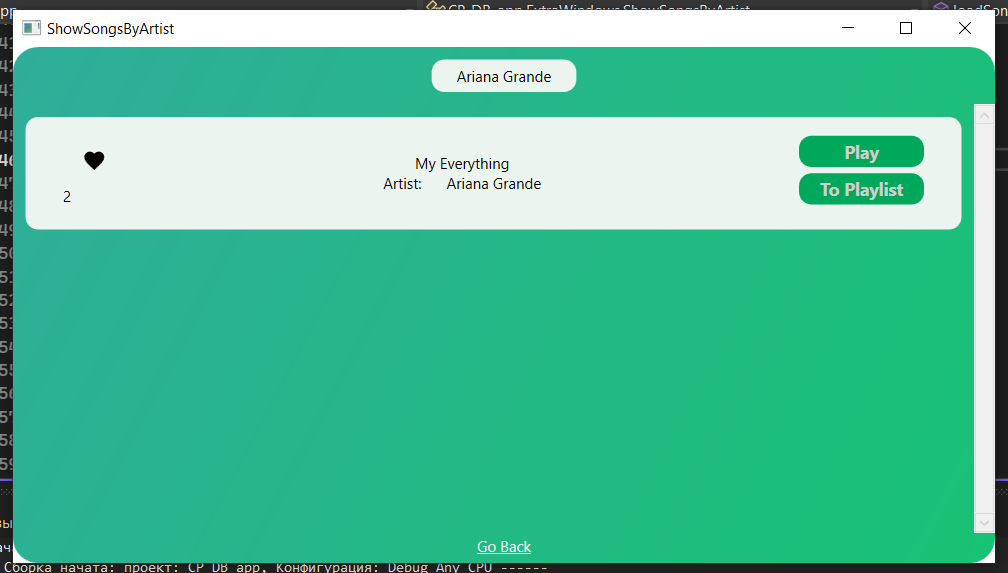


Рисунок 5.10 – Список песен исполнителя

Иконка в виде сердца обозначает кнопку для добавления песни в понравившееся и удаления из него. Счетчик снизу обозначает сколько пользователей отметили эту песню как понравившуюся.

Администратор имеет аналогичные предыдущей вкладке кнопки для добавления, удаления и редактирования песен (рисунок 5.11).

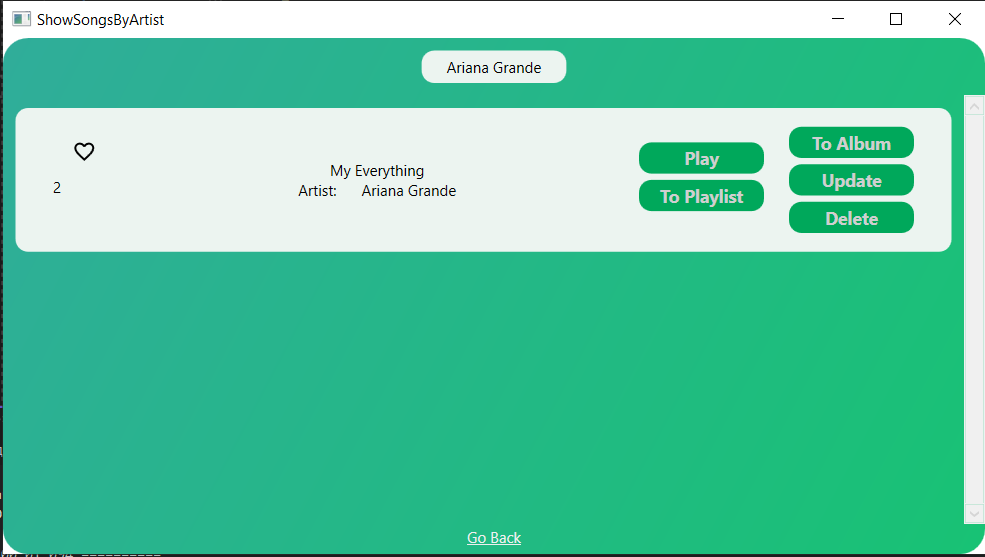


Рисунок 5.11 – Список песен исполнителя (администратор)

Вкладка «Albums» содержит список альбомов, состоящий из плиток с кнопками. На этой вкладке можно осуществлять поиск по названию альбома, воспользовавшись текстовый полем вверху. Вкладка «Albums» изображена на рисунке 5.12.

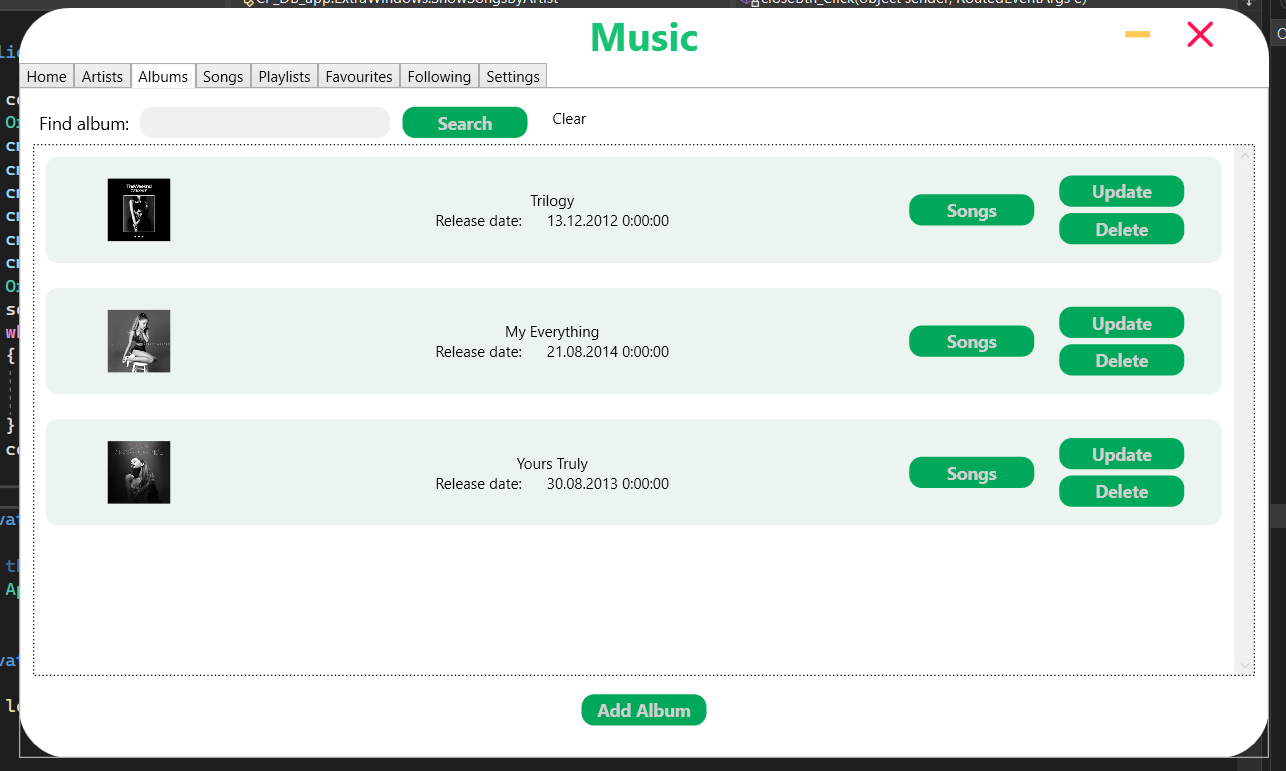


Рисунок 5.12 – Вкладка «Albums» (администратор)

Вкладка «Songs» содержит список всех песен, состоящий из плиток с кнопками. На этой вкладке можно осуществлять поиск по названию песни, воспользовавшись текстовый полем вверху.

Любой пользователь может нажать на кнопку «Play» внутри плитки песни, и тогда попадет на окно с плеером (рисунок 5.13).

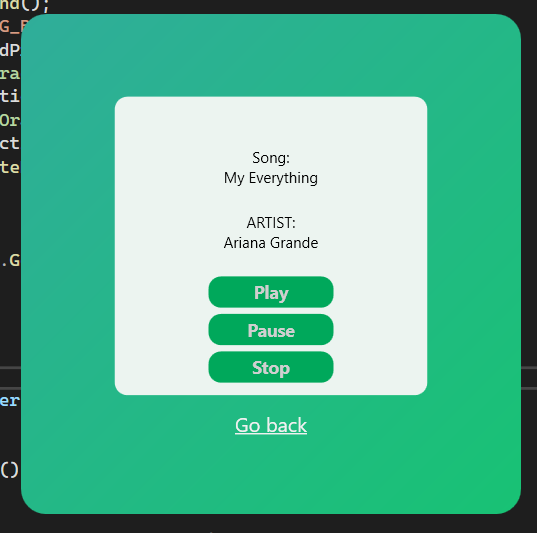


Рисунок 5.13 – Плеер

Плеер содержит информацию о текущей песне и три кнопки для управления песней. По нажатию на кнопку «Play» песня запустится.

По нажатию на кнопку «To Playlist» на плитке песни в любом месте приложения откроется окно добавления песни в плейлист (рисунок 5.14).

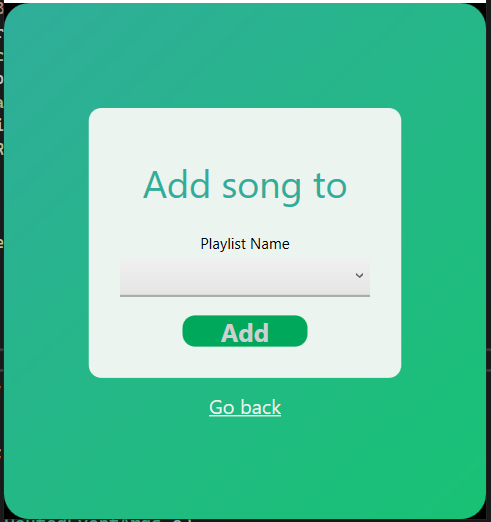


Рисунок 5.14 – Окно добавления песни в плейлист

Аналогичным образом работает кнопка, доступная администратору, «To Album», вызывая окно добавления песни в альбом (рисунок 5.15).

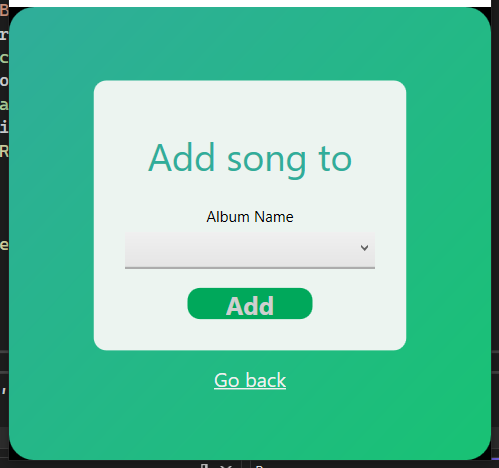


Рисунок 5.15 – Окно добавления песни в альбом

Вкладка «Playlists» содержит список всех плейлистов, состоящий из плиток с кнопками (рисунок 5.16).

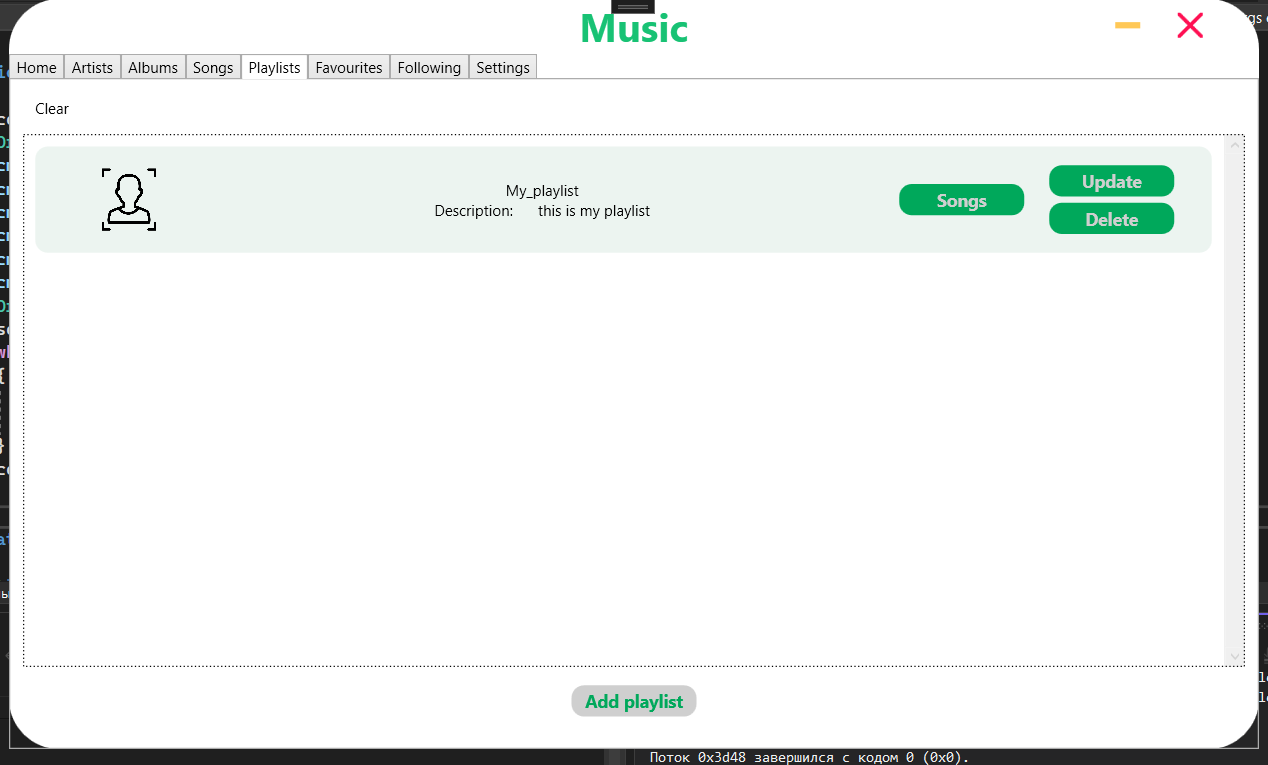


Рисунок 5.16 – Вкладка «Playlists»

Кнопки редактирования и удаления плейлиста доступны всем пользователям.

Вкладка «Favourites» содержит список всех песен, которые пользователь отметил как понравившиеся (рисунок 5.17).

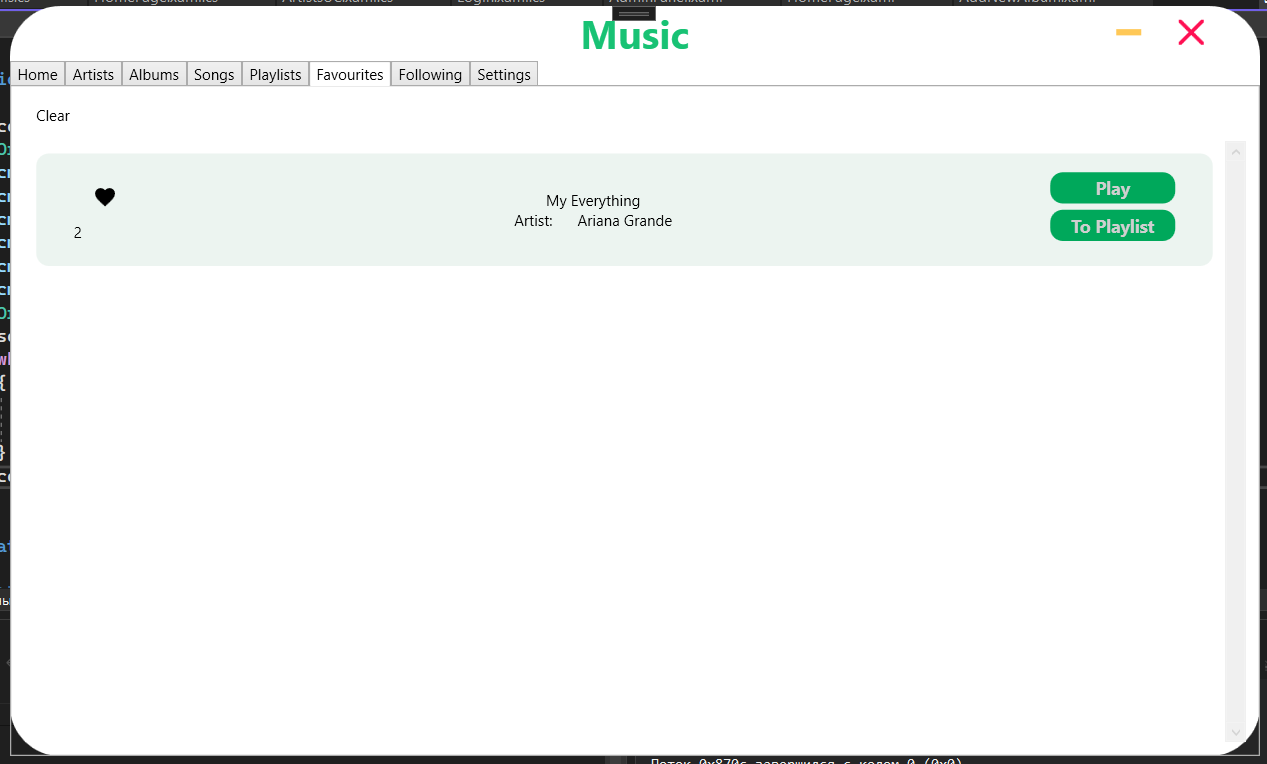


Рисунок 5.17 – Вкладка «Favourites»

Вкладка «Following» содержит список всех исполнителей, на которые подписался пользователь (рисунок 5.18).

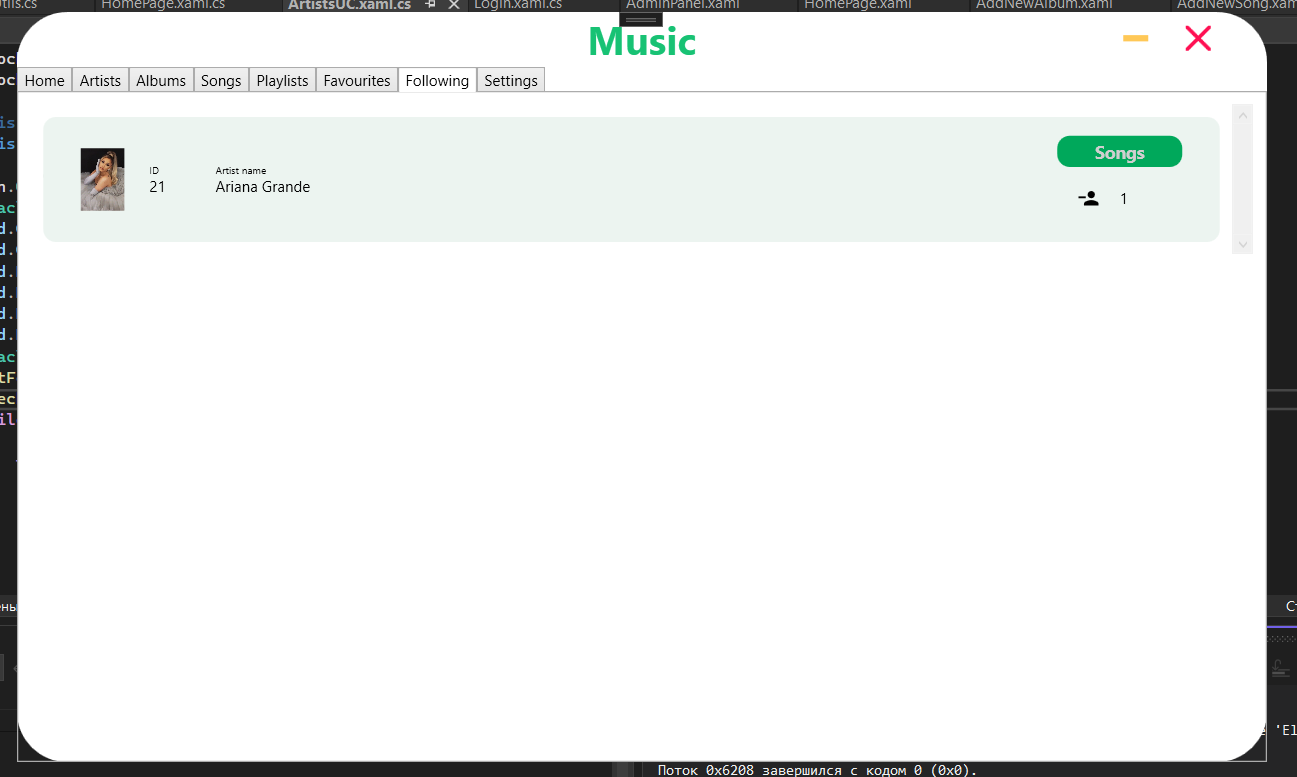


Рисунок 5.18 – Вкладка «Following»

Перейдя на вкладку «Settings», любой пользователь может выйти из аккаунта, изменить свой логин или свой пароль. Любой пользователь кроме администратора может удалить свой аккаунт по желанию. При этом надо иметь в виду, плейлисты этого пользователя также будут удалены.

## 5.1 Выводы по разделу

В данном разделе был описан весь базовый функционал пользователя и администратора приложения. Было проведено тестирование программного средства, которое показало его эффективность и правильную работу. Также было составлено руководство пользователя, которое описывает все необходимые действия для работы с программным средством.

## Заключение

Базы данных — это системы хранения и организации информации, которые используются для хранения, организации и доступа к большим объемам данных. Они часто используются для хранения структурированной информации, такой как списки клиентов, продажи, сотрудники и т.д.

В современном мире большинство организаций используют базы данных, также большинство приложений используют различные базы данных для хранения информации.

В процессе решения поставленной задачи была достигнута поставленная цель по созданию базы данных сервиса по прослушиванию музыки. В данной работе использовалось СУБД Oracle21c, а также .NET приложение с использованием технологии WPF для операционной системы Windows. При разработке курсового проекта использовались объекты: таблицы, представления, хранимые процедуры, триггеры, индексы.

В функционал созданной базы данных включает, добавление, удаление и редактирование исполнителей, песен, плейлистов, альбомов, а также управление подписками на исполнителей и понравившимися песнями.

Была освоена технология хранения мультимедийных типов данных и изучены различные их типы. Была проведена оптимизация работы базы данных, а также организован импорт и экспорт данных в XML формат.

В результате база данных успешно прошла тестирование при использовании большого количества данных.

В соответствии с полученным результатом работы можно сделать вывод, что разработанная база данных работает корректно.

## Список используемых источников

1 Data Cartridge Developer's Guide [Электронный ресурс] / Режим доступа: https://docs.oracle.com/en/database/oracle/oracle-database/18/addci/workingwith-multimedia-data-types.html – Дата доступа: 10.03.2023.

2 Oracle-dba.ru [Электронный ресурс] / Режим доступа: https://oracledba.ru – Дата доступа: 27.03.2023.

3 Stackoverflow.com [Электронный ресурс] / Режим доступа: https://stackoverflow.com – Дата доступа: 08.05.2023

4 Официальный сайт Oracle [Электронный ресурс] / Foundation, Inc. https://www.oracle.com/database/database-vault/index.html– Дата доступа: 10.04.2023

## Приложение А

DROP TABLE "USERS";

DROP TABLE ARTISTS;

DROP TABLE ALBUMS;

DROP TABLE SONGS;

DROP TABLE ARTIST\_SONG;

DROP TABLE PLAYLISTS;

DROP TABLE PLAYLIST\_SONG;

SELECT \* FROM USERS;

DELETE FROM SONGS WHERE SONG\_ID = 3;

ALTER TABLE USERS MODIFY(USERS\_ID GENERATED AS IDENTITY (START WITH 3));

COMMIT;

BEGIN

FOR i IN 1 .. 100000

LOOP

insert into USERS (USER\_LOGIN, USER\_PASSWORD) VALUES ('USER+ ' || i, '1');

END LOOP;

END;

--1

CREATE TABLE "USERS"(

USER\_ID INT GENERATED BY DEFAULT AS IDENTITY,

USER\_LOGIN NVARCHAR2(50) UNIQUE,

USER\_PASSWORD NVARCHAR2(255) NOT NULL,

PRIMARY KEY(USER\_ID));

--2

CREATE TABLE ARTISTS(

ARTIST\_ID INT GENERATED BY DEFAULT AS IDENTITY,

ARTIST\_NAME NVARCHAR2(50) NOT NULL,

ARTIST\_PHOTO BLOB,

PRIMARY KEY(ARTIST\_ID));

--3

CREATE TABLE ALBUMS(

ALBUM\_ID INT GENERATED BY DEFAULT AS IDENTITY,

ALBUM\_NAME NVARCHAR2(50) NOT NULL,

ALBUM\_RELEASE\_DATE DATE,

ALBUM\_COVER BLOB,

PRIMARY KEY(ALBUM\_ID));

--4

CREATE TABLE SONGS(

SONG\_ID INT GENERATED BY DEFAULT AS IDENTITY,

SONG\_NAME NVARCHAR2(50) NOT NULL,

SONG\_FILE BLOB NOT NULL,

ARTIST\_ID INT,

FOREIGN KEY (ARTIST\_ID) REFERENCES ARTISTS(ARTIST\_ID) ON DELETE CASCADE,

PRIMARY KEY(SONG\_ID));

--5

CREATE TABLE ALBUM\_SONG(

ALBUM\_ID INT,

SONG\_ID INT,

FOREIGN KEY (ALBUM\_ID) REFERENCES ALBUMS(ALBUM\_ID) ON DELETE CASCADE,

FOREIGN KEY (SONG\_ID) REFERENCES SONGS(SONG\_ID) ON DELETE CASCADE,

PRIMARY KEY(ALBUM\_ID, SONG\_ID));

--6

CREATE TABLE PLAYLISTS(

PLAYLIST\_ID INT GENERATED BY DEFAULT AS IDENTITY,

PLAYLIST\_NAME NVARCHAR2(50) NOT NULL,

USER\_ID INT NOT NULL,

PLAYLIST\_DESCRIPTION NVARCHAR2(255),

PLAYLIST\_COVER BLOB,

FOREIGN KEY (USER\_ID) REFERENCES "USERS"(USER\_ID) ON DELETE CASCADE,

PRIMARY KEY(PLAYLIST\_ID));

--7

CREATE TABLE PLAYLIST\_SONG(

PLAYLIST\_ID INT,

SONG\_ID INT,

FOREIGN KEY (PLAYLIST\_ID) REFERENCES PLAYLISTS(PLAYLIST\_ID) ON DELETE CASCADE,

FOREIGN KEY (SONG\_ID) REFERENCES SONGS(SONG\_ID) ON DELETE CASCADE,

PRIMARY KEY(PLAYLIST\_ID, SONG\_ID));

--8

CREATE TABLE FAVOURITE\_SONGS (

USER\_ID INT,

SONG\_ID INT,

FOREIGN KEY (SONG\_ID) REFERENCES SONGS(SONG\_ID) ON DELETE CASCADE,

FOREIGN KEY (USER\_ID) REFERENCES USERS(USER\_ID) ON DELETE CASCADE,

PRIMARY KEY(USER\_ID, SONG\_ID));

--9

CREATE TABLE FOLLOWED\_ARTISTS (

USER\_ID INT,

ARTIST\_ID INT,

FOREIGN KEY (USER\_ID) REFERENCES USERS(USER\_ID) ON DELETE CASCADE,

FOREIGN KEY (ARTIST\_ID) REFERENCES ARTISTS(ARTIST\_ID) ON DELETE CASCADE,

PRIMARY KEY(USER\_ID, ARTIST\_ID));

## Приложение Б

CREATE OR REPLACE PACKAGE GENERAL\_PACKAGE IS

------------------------------------ARTISTS-------------------------------------

PROCEDURE GET\_ARTIST\_BY\_NAME (I\_ARTIST\_NAME IN ARTISTS.ARTIST\_NAME%TYPE,

O\_ARTIST\_CURS OUT SYS\_REFCURSOR);

PROCEDURE GET\_ARTIST\_BY\_ID (I\_ARTIST\_ID IN ARTISTS.ARTIST\_ID%TYPE,

O\_ARTIST\_CURS OUT SYS\_REFCURSOR);

PROCEDURE GET\_ALL\_ARTISTS (O\_ARTIST\_CURS OUT SYS\_REFCURSOR);

------------------------------------SONGS--------------------------------------

PROCEDURE GET\_ALL\_SONGS (O\_SONG\_CURS OUT SYS\_REFCURSOR);

PROCEDURE GET\_SONG\_BY\_ID (I\_SONG\_ID IN SONGS.SONG\_ID%TYPE,

O\_SONG\_CURS OUT SYS\_REFCURSOR);

PROCEDURE GET\_SONG\_BY\_ARTIST (I\_ARTIST\_ID IN ARTISTS.ARTIST\_ID%TYPE,

O\_SONG\_CURS OUT SYS\_REFCURSOR);

------------------------------------ALBUMS--------------------------------------

PROCEDURE GET\_ALL\_ALBUMS (O\_ALBUMS\_CURS OUT SYS\_REFCURSOR);

PROCEDURE GET\_ALBUM\_BY\_ID (I\_ALBUM\_ID IN ALBUMS.ALBUM\_ID%TYPE,

O\_ALBUMS\_CURS OUT SYS\_REFCURSOR);

PROCEDURE GET\_SONGS\_BY\_ALBUM (I\_ALBUM\_ID IN ALBUMS.ALBUM\_ID%TYPE,

O\_SONGS\_CURS OUT SYS\_REFCURSOR);

PROCEDURE GET\_ALBUMS\_BY\_SONG (I\_SONG\_ID IN SONGS.SONG\_ID%TYPE,

O\_ALBUMS\_CURS OUT SYS\_REFCURSOR);

---------------------------------FAVORITE\_SONG----------------------------------

PROCEDURE GET\_AMOUNT\_USERS\_BY\_FAVOURITES (I\_SONG\_ID IN SONGS.SONG\_ID%TYPE,

O\_USERS\_AMOUNT OUT NUMBER);

PROCEDURE GET\_TOP\_FAVORITE\_SONGS (O\_TOP\_SONGS\_CURS OUT SYS\_REFCURSOR);

-------------------------------FOLLOWED\_ARTISTS---------------------------------

PROCEDURE GET\_ARTIST\_FOLLOWERS\_AMOUNT (I\_ARTIST\_ID IN ARTISTS.ARTIST\_ID%TYPE,

O\_FOLLOWERS\_AMOUNT OUT NUMBER);

PROCEDURE GET\_TOP\_FOLLOWED\_ARTISTS (O\_TOP\_FOLLOWED\_CURS OUT SYS\_REFCURSOR);

END;

CREATE OR REPLACE PACKAGE BODY GENERAL\_PACKAGE IS

------------------------------------ARTISTS-------------------------------------

-------------------------GET\_ALL\_ARTISTS------------------------

PROCEDURE GET\_ALL\_ARTISTS

(O\_ARTIST\_CURS OUT SYS\_REFCURSOR)

IS

BEGIN

OPEN O\_ARTIST\_CURS FOR SELECT \* FROM ARTISTS;

END GET\_ALL\_ARTISTS;

-------------------------GET\_ARTIST\_BY\_NAME------------------------

PROCEDURE GET\_ARTIST\_BY\_NAME

(I\_ARTIST\_NAME IN ARTISTS.ARTIST\_NAME%TYPE,

O\_ARTIST\_CURS OUT SYS\_REFCURSOR)

IS

BEGIN

OPEN O\_ARTIST\_CURS FOR SELECT \* FROM ARTISTS WHERE ARTIST\_NAME = I\_ARTIST\_NAME;

IF O\_ARTIST\_CURS%NOTFOUND THEN

RAISE\_APPLICATION\_ERROR(-20017, 'Artist not found');

END IF;

END GET\_ARTIST\_BY\_NAME;

-------------------------GET\_ARTIST\_BY\_ID------------------------

PROCEDURE GET\_ARTIST\_BY\_ID

(I\_ARTIST\_ID IN ARTISTS.ARTIST\_ID%TYPE,

O\_ARTIST\_CURS OUT SYS\_REFCURSOR)

IS

BEGIN

OPEN O\_ARTIST\_CURS FOR SELECT \* FROM ARTISTS WHERE ARTIST\_ID = I\_ARTIST\_ID;

IF O\_ARTIST\_CURS%NOTFOUND THEN

RAISE\_APPLICATION\_ERROR(-20017, 'Artist not found');

END IF;

END GET\_ARTIST\_BY\_ID;

------------------------------------SONGS---------------------------------------

-------------------------GET\_ALL\_SONGS-------------------------

PROCEDURE GET\_ALL\_SONGS

(O\_SONG\_CURS OUT SYS\_REFCURSOR)

IS

BEGIN

OPEN O\_SONG\_CURS FOR SELECT \* FROM SONGS ;

END GET\_ALL\_SONGS;

-------------------------GET\_SONG\_BY\_ID-------------------------

PROCEDURE GET\_SONG\_BY\_ID

(I\_SONG\_ID IN SONGS.SONG\_ID%TYPE,

O\_SONG\_CURS OUT SYS\_REFCURSOR)

IS

BEGIN

OPEN O\_SONG\_CURS FOR SELECT \* FROM SONGS

WHERE SONG\_ID = I\_SONG\_ID;

IF O\_SONG\_CURS%NOTFOUND THEN

RAISE\_APPLICATION\_ERROR(-20019, 'Song not found');

END IF;

END GET\_SONG\_BY\_ID;

-------------------------GET\_SONG\_BY\_ARTIST-------------------------

PROCEDURE GET\_SONG\_BY\_ARTIST

(I\_ARTIST\_ID IN ARTISTS.ARTIST\_ID%TYPE,

O\_SONG\_CURS OUT SYS\_REFCURSOR)

IS

BEGIN

OPEN O\_SONG\_CURS FOR SELECT \* FROM SONGS

WHERE ARTIST\_ID = I\_ARTIST\_ID;

IF O\_SONG\_CURS%NOTFOUND THEN

RAISE\_APPLICATION\_ERROR(-20019, 'Song not found');

END IF;

END GET\_SONG\_BY\_ARTIST;

------------------------------------ALBUMS--------------------------------------

-------------------------GET\_ALL\_ALBUMS-------------------------

PROCEDURE GET\_ALL\_ALBUMS

(O\_ALBUMS\_CURS OUT SYS\_REFCURSOR)

IS

BEGIN

OPEN O\_ALBUMS\_CURS FOR SELECT \* FROM ALBUMS;

END GET\_ALL\_ALBUMS;

-------------------------GET\_ALBUM\_BY\_ID-------------------------

PROCEDURE GET\_ALBUM\_BY\_ID

(I\_ALBUM\_ID IN ALBUMS.ALBUM\_ID%TYPE,

O\_ALBUMS\_CURS OUT SYS\_REFCURSOR)

IS

BEGIN

OPEN O\_ALBUMS\_CURS FOR SELECT \* FROM ALBUMS WHERE ALBUM\_ID = I\_ALBUM\_ID;

IF O\_ALBUMS\_CURS%NOTFOUND THEN

RAISE\_APPLICATION\_ERROR(-20018, 'Album not found');

END IF;

END GET\_ALBUM\_BY\_ID;

-------------------------GET\_SONGS\_BY\_ALBUM-------------------------

PROCEDURE GET\_SONGS\_BY\_ALBUM

(I\_ALBUM\_ID IN ALBUMS.ALBUM\_ID%TYPE,

O\_SONGS\_CURS OUT SYS\_REFCURSOR)

IS

BEGIN

OPEN O\_SONGS\_CURS FOR SELECT SONG\_ID, SONG\_NAME, SONG\_FILE, ARTIST\_NAME

FROM ARTIST\_ALBUM\_SONG\_VIEW WHERE ALBUM\_ID = I\_ALBUM\_ID;

IF O\_SONGS\_CURS%NOTFOUND THEN

RAISE\_APPLICATION\_ERROR(-20018, 'Album not found');

END IF;

END GET\_SONGS\_BY\_ALBUM;

-------------------------GET\_ALBUMS\_BY\_SONG-------------------------

PROCEDURE GET\_ALBUMS\_BY\_SONG

(I\_SONG\_ID IN SONGS.SONG\_ID%TYPE,

O\_ALBUMS\_CURS OUT SYS\_REFCURSOR)

IS

BEGIN

OPEN O\_ALBUMS\_CURS FOR SELECT ALBUM\_ID, ALBUM\_NAME, ALBUM\_RELEASE\_DATE, ALBUM\_COVER

FROM ARTIST\_ALBUM\_SONG\_VIEW WHERE SONG\_ID = I\_SONG\_ID;

IF O\_ALBUMS\_CURS%NOTFOUND THEN

RAISE\_APPLICATION\_ERROR(-20019, 'Song not found');

END IF;

END GET\_ALBUMS\_BY\_SONG;

---------------------------------FAVORITE\_SONG----------------------------------

------GET\_AMOUNT\_USERS\_BY\_FAVOURITES---------

PROCEDURE GET\_AMOUNT\_USERS\_BY\_FAVOURITES

(I\_SONG\_ID IN SONGS.SONG\_ID%TYPE,

O\_USERS\_AMOUNT OUT NUMBER)

IS

BEGIN

SELECT COUNT(USER\_ID) INTO O\_USERS\_AMOUNT

FROM FAVS\_USER\_SONG\_ARTIST\_VIEW

WHERE SONG\_ID = I\_SONG\_ID;

END GET\_AMOUNT\_USERS\_BY\_FAVOURITES;

--------GET\_TOP\_FAVORITE\_SONGS------

PROCEDURE GET\_TOP\_FAVORITE\_SONGS

(O\_TOP\_SONGS\_CURS OUT SYS\_REFCURSOR)

IS

BEGIN

OPEN O\_TOP\_SONGS\_CURS FOR

SELECT \* FROM TOP\_FAVOURITE\_SONGS\_VIEW;

END GET\_TOP\_FAVORITE\_SONGS;

-------------------------------FOLLOWED\_ARTISTS---------------------------------

------GET\_ARTIST\_FOLLOWERS\_AMOUNT----------

PROCEDURE GET\_ARTIST\_FOLLOWERS\_AMOUNT

(I\_ARTIST\_ID IN ARTISTS.ARTIST\_ID%TYPE,

O\_FOLLOWERS\_AMOUNT OUT NUMBER)

IS

BEGIN

SELECT COUNT(\*) INTO O\_FOLLOWERS\_AMOUNT

FROM FOLLOWED\_ARTISTS

WHERE ARTIST\_ID = I\_ARTIST\_ID;

END GET\_ARTIST\_FOLLOWERS\_AMOUNT;

--------------GET\_TOP\_FOLLOWED\_ARTISTS---------------

PROCEDURE GET\_TOP\_FOLLOWED\_ARTISTS

(O\_TOP\_FOLLOWED\_CURS OUT SYS\_REFCURSOR)

IS

BEGIN

OPEN O\_TOP\_FOLLOWED\_CURS FOR

SELECT ARTIST\_ID FROM TOP\_FOLLOWED\_ARTISTS\_VIEW;

END GET\_TOP\_FOLLOWED\_ARTISTS;

END;

## Приложение В

CREATE OR REPLACE PACKAGE CLIENT\_PACKAGE IS

-------------------------------------USERS--------------------------------------

PROCEDURE REGISTER\_USER (I\_USER\_LOGIN IN USERS.USER\_LOGIN%TYPE,

I\_USER\_PASSWORD IN USERS.USER\_PASSWORD%TYPE);

PROCEDURE LOG\_IN\_USER (I\_USER\_LOGIN IN USERS.USER\_LOGIN%TYPE,

I\_USER\_PASSWORD IN USERS.USER\_PASSWORD%TYPE,

O\_USER\_ID OUT USERS.USER\_ID%TYPE,

O\_USER\_LOGIN OUT USERS.USER\_LOGIN%TYPE);

PROCEDURE DELETE\_USER (I\_USER\_LOGIN IN USERS.USER\_LOGIN%TYPE);

PROCEDURE UPDATE\_USER (I\_USER\_ID IN USERS.USER\_ID%TYPE,

I\_USER\_LOGIN IN USERS.USER\_LOGIN%TYPE,

I\_USER\_PASSWORD IN USERS.USER\_PASSWORD%TYPE);

-----------------------------------PLAYLISTS------------------------------------

PROCEDURE CREATE\_PLAYLIST (I\_PLAYLIST\_NAME IN PLAYLISTS.PLAYLIST\_NAME%TYPE,

I\_PLAYLIST\_DESCRIPTION IN PLAYLISTS.PLAYLIST\_DESCRIPTION%TYPE,

I\_USER\_ID IN USERS.USER\_ID%TYPE,

I\_PLAYLIST\_COVER IN PLAYLISTS.PLAYLIST\_COVER%TYPE);

PROCEDURE GET\_PLAYLIST\_BY\_USER (I\_USER\_ID IN USERS.USER\_ID%TYPE,

O\_PLAYLIST\_CURS OUT SYS\_REFCURSOR);

PROCEDURE UPDATE\_PLAYLIST (I\_PLAYLIST\_ID IN PLAYLISTS.PLAYLIST\_ID%TYPE,

I\_PLAYLIST\_NAME IN PLAYLISTS.PLAYLIST\_NAME%TYPE,

I\_PLAYLIST\_DESCRIPTION IN PLAYLISTS.PLAYLIST\_DESCRIPTION%TYPE,

I\_USER\_ID IN USERS.USER\_ID%TYPE,

I\_PLAYLIST\_COVER IN PLAYLISTS.PLAYLIST\_COVER%TYPE);

PROCEDURE DELETE\_PLAYLIST (I\_PLAYLIST\_ID IN PLAYLISTS.PLAYLIST\_ID%TYPE);

PROCEDURE ADD\_SONG\_TO\_PLAYLIST (I\_SONG\_ID IN SONGS.SONG\_ID%TYPE,

I\_PLAYLIST\_ID IN PLAYLISTS.PLAYLIST\_ID%TYPE);

PROCEDURE DELETE\_SONG\_FROM\_PLAYLIST (I\_SONG\_ID IN SONGS.SONG\_ID%TYPE,

I\_PLAYLIST\_ID IN PLAYLISTS.PLAYLIST\_ID%TYPE);

---------------------------------FAVORITE\_SONG----------------------------------

PROCEDURE ADD\_SONG\_TO\_FAVOURITE (I\_SONG\_ID IN SONGS.SONG\_ID%TYPE,

I\_USER\_ID IN USERS.USER\_ID%TYPE);

PROCEDURE GET\_USER\_FAVOURITES (I\_USER\_ID IN USERS.USER\_ID%TYPE,

O\_SONGS\_CURS OUT SYS\_REFCURSOR);

PROCEDURE DELETE\_SONG\_FROM\_FAVOURITE (I\_SONG\_ID IN SONGS.SONG\_ID%TYPE,

I\_USER\_ID IN USERS.USER\_ID%TYPE);

-------------------------------FOLLOWED\_ARTISTS---------------------------------

PROCEDURE FOLLOW\_ARTIST (I\_ARTIST\_ID IN ARTISTS.ARTIST\_ID%TYPE,

I\_USER\_ID IN USERS.USER\_ID%TYPE);

PROCEDURE UNFOLLOW\_ARTIST (I\_ARTIST\_ID IN ARTISTS.ARTIST\_ID%TYPE,

I\_USER\_ID IN USERS.USER\_ID%TYPE);

PROCEDURE GET\_FOLLOWED\_ARTISTS (I\_USER\_ID IN USERS.USER\_ID%TYPE,

O\_ARTISTS\_CURS OUT SYS\_REFCURSOR);

END;

CREATE OR REPLACE PACKAGE BODY CLIENT\_PACKAGE IS

-------------------------------------USERS--------------------------------------

-------------------------REGISTER\_USER-------------------------

PROCEDURE REGISTER\_USER (I\_USER\_LOGIN IN USERS.USER\_LOGIN%TYPE,

I\_USER\_PASSWORD IN USERS.USER\_PASSWORD%TYPE)

IS

CNT NUMBER;

BEGIN

SELECT COUNT(\*) INTO CNT FROM USERS WHERE USER\_LOGIN = I\_USER\_LOGIN;

IF (CNT = 0) THEN

INSERT INTO USERS (USER\_LOGIN, USER\_PASSWORD)

VALUES (I\_USER\_LOGIN, I\_USER\_PASSWORD);

COMMIT;

ELSE

RAISE\_APPLICATION\_ERROR(-20001, 'This login is already taken');

END IF;

END REGISTER\_USER;

-------------------------LOG\_IN\_USER-------------------------

PROCEDURE LOG\_IN\_USER

(I\_USER\_LOGIN IN USERS.USER\_LOGIN%TYPE,

I\_USER\_PASSWORD IN USERS.USER\_PASSWORD%TYPE,

O\_USER\_ID OUT USERS.USER\_ID%TYPE,

O\_USER\_LOGIN OUT USERS.USER\_LOGIN%TYPE)

IS

CURSOR USER\_CURSOR IS SELECT USER\_ID, USER\_LOGIN FROM USERS

WHERE USERS.USER\_LOGIN = I\_USER\_LOGIN

AND USERS.USER\_PASSWORD = DBMS\_CRYPTO.HASH(UTL\_RAW.CAST\_TO\_RAW(I\_USER\_PASSWORD), 4);

BEGIN

OPEN USER\_CURSOR;

FETCH USER\_CURSOR INTO O\_USER\_ID, O\_USER\_LOGIN;

IF USER\_CURSOR%NOTFOUND THEN

RAISE\_APPLICATION\_ERROR(-20002, 'Incorrect login or password');

END IF;

CLOSE user\_cursor;

END LOG\_IN\_USER;

-------------------------DELETE\_USER-------------------------

PROCEDURE DELETE\_USER

(I\_USER\_LOGIN IN USERS.USER\_LOGIN%TYPE)

IS

CNT NUMBER;

BEGIN

SELECT COUNT(\*) INTO CNT FROM USERS WHERE USERS.USER\_LOGIN = I\_USER\_LOGIN;

IF (CNT != 0) THEN

DELETE FROM USERS WHERE USERS.USER\_LOGIN = I\_USER\_LOGIN;

COMMIT;

ELSE

RAISE\_APPLICATION\_ERROR(-20003, 'User is not found');

END IF;

END DELETE\_USER;

-------------------------UPDATE\_USER-------------------------

PROCEDURE UPDATE\_USER

(I\_USER\_ID IN USERS.USER\_ID%TYPE,

I\_USER\_LOGIN IN USERS.USER\_LOGIN%TYPE,

I\_USER\_PASSWORD IN USERS.USER\_PASSWORD%TYPE)

IS

CNT NUMBER;

BEGIN

SELECT COUNT(\*) INTO CNT FROM USERS WHERE USERS.USER\_ID = I\_USER\_ID;

IF (CNT != 0) THEN

UPDATE USERS SET USER\_LOGIN = I\_USER\_LOGIN,

USER\_PASSWORD = I\_USER\_PASSWORD

WHERE I\_USER\_ID = USER\_ID;

COMMIT;

ELSE

RAISE\_APPLICATION\_ERROR(-20003, 'User is not found');

END IF;

END UPDATE\_USER;

-----------------------------------PLAYLISTS------------------------------------

-------------------------CREATE\_PLAYLIST-------------------------

PROCEDURE CREATE\_PLAYLIST

(I\_PLAYLIST\_NAME IN PLAYLISTS.PLAYLIST\_NAME%TYPE,

I\_PLAYLIST\_DESCRIPTION IN PLAYLISTS.PLAYLIST\_DESCRIPTION%TYPE,

I\_USER\_ID IN USERS.USER\_ID%TYPE,

I\_PLAYLIST\_COVER IN PLAYLISTS.PLAYLIST\_COVER%TYPE)

IS

CNT NUMBER;

BEGIN

SELECT COUNT(\*) INTO CNT FROM PLAYLISTS

WHERE PLAYLIST\_NAME = I\_PLAYLIST\_NAME AND USER\_ID = I\_USER\_ID;

IF (CNT = 0) THEN

INSERT INTO PLAYLISTS (PLAYLIST\_NAME, PLAYLIST\_DESCRIPTION, USER\_ID, PLAYLIST\_COVER)

VALUES (I\_PLAYLIST\_NAME, I\_PLAYLIST\_DESCRIPTION, I\_USER\_ID, I\_PLAYLIST\_COVER);

COMMIT;

ELSE

RAISE\_APPLICATION\_ERROR(-20010, 'This user already has this playlist');

END IF;

END CREATE\_PLAYLIST;

-------------------------GET\_PLAYLIST\_BY\_USER-------------------------

PROCEDURE GET\_PLAYLIST\_BY\_USER

(I\_USER\_ID IN USERS.USER\_ID%TYPE,

O\_PLAYLIST\_CURS OUT SYS\_REFCURSOR)

IS

BEGIN

OPEN O\_PLAYLIST\_CURS FOR SELECT \* FROM PLAYLISTS WHERE USER\_ID = I\_USER\_ID;

IF O\_PLAYLIST\_CURS%NOTFOUND THEN

RAISE\_APPLICATION\_ERROR(-20020, 'Playlist not found');

END IF;

END GET\_PLAYLIST\_BY\_USER;

-------------------------UPDATE\_PLAYLIST-------------------------

PROCEDURE UPDATE\_PLAYLIST

(I\_PLAYLIST\_ID IN PLAYLISTS.PLAYLIST\_ID%TYPE,

I\_PLAYLIST\_NAME IN PLAYLISTS.PLAYLIST\_NAME%TYPE,

I\_PLAYLIST\_DESCRIPTION IN PLAYLISTS.PLAYLIST\_DESCRIPTION%TYPE,

I\_USER\_ID IN USERS.USER\_ID%TYPE,

I\_PLAYLIST\_COVER IN PLAYLISTS.PLAYLIST\_COVER%TYPE)

IS

CNT NUMBER;

BEGIN

SELECT COUNT(\*) INTO CNT FROM PLAYLISTS

WHERE PLAYLIST\_ID = I\_PLAYLIST\_ID;

IF (CNT != 0) THEN

UPDATE PLAYLISTS SET PLAYLIST\_NAME = I\_PLAYLIST\_NAME,

PLAYLIST\_DESCRIPTION = I\_PLAYLIST\_DESCRIPTION,

USER\_ID = I\_USER\_ID, PLAYLIST\_COVER = I\_PLAYLIST\_COVER

WHERE PLAYLIST\_ID = I\_PLAYLIST\_ID;

COMMIT;

ELSE

RAISE\_APPLICATION\_ERROR(-20020, 'Playlist not found');

END IF;

END UPDATE\_PLAYLIST;

-------------------------DELETE\_PLAYLIST-------------------------

PROCEDURE DELETE\_PLAYLIST

(I\_PLAYLIST\_ID IN PLAYLISTS.PLAYLIST\_ID%TYPE)

IS

CNT NUMBER;

BEGIN

SELECT COUNT(\*) INTO CNT FROM PLAYLISTS

WHERE PLAYLIST\_ID = I\_PLAYLIST\_ID;

IF (CNT != 0) THEN

DELETE FROM PLAYLISTS WHERE PLAYLIST\_ID = I\_PLAYLIST\_ID;

COMMIT;

ELSE

RAISE\_APPLICATION\_ERROR(-20020, 'Playlist not found');

END IF;

END DELETE\_PLAYLIST;

-------------------------ADD\_SONG\_TO\_PLAYLIST-------------------------

PROCEDURE ADD\_SONG\_TO\_PLAYLIST

(I\_SONG\_ID IN SONGS.SONG\_ID%TYPE,

I\_PLAYLIST\_ID IN PLAYLISTS.PLAYLIST\_ID%TYPE)

IS

CNT NUMBER;

BEGIN

SELECT COUNT(\*) INTO CNT FROM PLAYLIST\_SONG

WHERE PLAYLIST\_ID = I\_PLAYLIST\_ID AND SONG\_ID = I\_SONG\_ID;

IF (CNT = 0) THEN

INSERT INTO PLAYLIST\_SONG(PLAYLIST\_ID, SONG\_ID) VALUES (I\_PLAYLIST\_ID, I\_SONG\_ID);

COMMIT;

ELSE

RAISE\_APPLICATION\_ERROR(-20011, 'This playlist already has this song');

END IF;

END ADD\_SONG\_TO\_PLAYLIST;

-------------------------DELETE\_SONG\_FROM\_PLAYLIST-------------------------

PROCEDURE DELETE\_SONG\_FROM\_PLAYLIST

(I\_SONG\_ID IN SONGS.SONG\_ID%TYPE,

I\_PLAYLIST\_ID IN PLAYLISTS.PLAYLIST\_ID%TYPE)

IS

CNT NUMBER;

BEGIN

SELECT COUNT(\*) INTO CNT FROM PLAYLIST\_SONG

WHERE PLAYLIST\_ID = I\_PLAYLIST\_ID AND SONG\_ID = I\_SONG\_ID;

IF (CNT > 0) THEN

DELETE FROM PLAYLIST\_SONG WHERE PLAYLIST\_ID = I\_PLAYLIST\_ID AND SONG\_ID = I\_SONG\_ID;

COMMIT;

ELSE

RAISE\_APPLICATION\_ERROR(-20012, 'This playlist does not have has this song');

END IF;

END DELETE\_SONG\_FROM\_PLAYLIST;

---------------------------------FAVORITE\_SONG----------------------------------

-------------------------ADD\_SONG\_TO\_FAVOURITE-------------------------

PROCEDURE ADD\_SONG\_TO\_FAVOURITE

(I\_SONG\_ID IN SONGS.SONG\_ID%TYPE,

I\_USER\_ID IN USERS.USER\_ID%TYPE)

IS

CNT NUMBER;

BEGIN

SELECT COUNT(\*) INTO CNT FROM FAVOURITE\_SONGS

WHERE USER\_ID = I\_USER\_ID AND SONG\_ID = I\_SONG\_ID;

IF (CNT = 0) THEN

INSERT INTO FAVOURITE\_SONGS(USER\_ID, SONG\_ID) VALUES (I\_USER\_ID, I\_SONG\_ID);

COMMIT;

ELSE

RAISE\_APPLICATION\_ERROR(-20013, 'This user already has this song in favourite');

END IF;

END ADD\_SONG\_TO\_FAVOURITE;

-------------------------GET\_USER\_FAVOURITES-------------------------

PROCEDURE GET\_USER\_FAVOURITES

(I\_USER\_ID IN USERS.USER\_ID%TYPE,

O\_SONGS\_CURS OUT SYS\_REFCURSOR)

IS

BEGIN

OPEN O\_SONGS\_CURS FOR

SELECT SONG\_ID, SONG\_NAME, SONG\_FILE, ARTIST\_NAME

FROM FAVS\_USER\_SONG\_ARTIST\_VIEW

WHERE USER\_ID = I\_USER\_ID;

END GET\_USER\_FAVOURITES;

-------------------------DELETE\_SONG\_FROM\_FAVOURITE-------------------------

PROCEDURE DELETE\_SONG\_FROM\_FAVOURITE

(I\_SONG\_ID IN SONGS.SONG\_ID%TYPE,

I\_USER\_ID IN USERS.USER\_ID%TYPE)

IS

CNT NUMBER;

BEGIN

SELECT COUNT(\*) INTO CNT FROM FAVOURITE\_SONGS

WHERE USER\_ID = I\_USER\_ID AND SONG\_ID = I\_SONG\_ID;

IF (CNT > 0) THEN

DELETE FROM FAVOURITE\_SONGS WHERE USER\_ID = I\_USER\_ID AND SONG\_ID = I\_SONG\_ID;

COMMIT;

ELSE

RAISE\_APPLICATION\_ERROR(-20014, 'This user does not have this song in favourite');

END IF;

END DELETE\_SONG\_FROM\_FAVOURITE;

-------------------------------FOLLOWED\_ARTISTS---------------------------------

-------------------------FOLLOW\_ARTIST-------------------------

PROCEDURE FOLLOW\_ARTIST

(I\_ARTIST\_ID IN ARTISTS.ARTIST\_ID%TYPE,

I\_USER\_ID IN USERS.USER\_ID%TYPE)

IS

CNT NUMBER;

BEGIN

SELECT COUNT(\*) INTO CNT FROM FOLLOWED\_ARTISTS

WHERE USER\_ID = I\_USER\_ID AND ARTIST\_ID = I\_ARTIST\_ID;

IF (CNT = 0) THEN

INSERT INTO FOLLOWED\_ARTISTS(USER\_ID, ARTIST\_ID) VALUES (I\_USER\_ID, I\_ARTIST\_ID);

COMMIT;

ELSE

RAISE\_APPLICATION\_ERROR(-20015, 'This user already has followed this artist');

END IF;

END FOLLOW\_ARTIST;

-------------------------UNFOLLOW\_ARTIST-------------------------

PROCEDURE UNFOLLOW\_ARTIST

(I\_ARTIST\_ID IN ARTISTS.ARTIST\_ID%TYPE,

I\_USER\_ID IN USERS.USER\_ID%TYPE)

IS

CNT NUMBER;

BEGIN

SELECT COUNT(\*) INTO CNT FROM FOLLOWED\_ARTISTS

WHERE USER\_ID = I\_USER\_ID AND ARTIST\_ID = I\_ARTIST\_ID;

IF (CNT > 0) THEN

DELETE FROM FOLLOWED\_ARTISTS WHERE USER\_ID = I\_USER\_ID AND ARTIST\_ID = I\_ARTIST\_ID;

COMMIT;

ELSE

RAISE\_APPLICATION\_ERROR(-20016, 'This user is not following this artist');

END IF;

END UNFOLLOW\_ARTIST;

-------------------------GET\_FOLLOWED\_ARTISTS-------------------------

PROCEDURE GET\_FOLLOWED\_ARTISTS

(I\_USER\_ID IN USERS.USER\_ID%TYPE,

O\_ARTISTS\_CURS OUT SYS\_REFCURSOR)

IS

BEGIN

OPEN O\_ARTISTS\_CURS FOR

SELECT FOLLOWED\_ARTISTS.ARTIST\_ID FROM FOLLOWED\_ARTISTS WHERE FOLLOWED\_ARTISTS.USER\_ID = I\_USER\_ID;

END GET\_FOLLOWED\_ARTISTS;

END;

## Приложение Г

CREATE OR REPLACE PACKAGE ADMIN\_PACKAGE IS

-------------------------------------USERS--------------------------------------

PROCEDURE DELETE\_USER (I\_USER\_LOGIN IN USERS.USER\_LOGIN%TYPE);

------------------------------------ARTISTS-------------------------------------

PROCEDURE CREATE\_ARTIST (I\_ARTIST\_NAME IN ARTISTS.ARTIST\_NAME%TYPE,

I\_ARTIST\_PHOTO IN ARTISTS.ARTIST\_PHOTO%TYPE);

PROCEDURE UPDATE\_ARTIST (I\_ARTIST\_ID IN ARTISTS.ARTIST\_ID%TYPE,

I\_ARTIST\_NAME IN ARTISTS.ARTIST\_NAME%TYPE,

I\_ARTIST\_PHOTO IN ARTISTS.ARTIST\_PHOTO%TYPE);

PROCEDURE DELETE\_ARTIST (I\_ARTIST\_ID IN ARTISTS.ARTIST\_ID%TYPE);

------------------------------------ALBUMS--------------------------------------

PROCEDURE CREATE\_ALBUM (I\_ARTIST\_ID IN ARTISTS.ARTIST\_ID%TYPE,

I\_ALBUM\_NAME IN ALBUMS.ALBUM\_NAME%TYPE,

I\_ALBUM\_RELEASE\_DATE IN ALBUMS.ALBUM\_RELEASE\_DATE%TYPE,

I\_ALBUM\_COVER IN ALBUMS.ALBUM\_COVER%TYPE);

PROCEDURE UPDATE\_ALBUM (I\_ALBUM\_ID IN ALBUMS.ALBUM\_ID%TYPE,

I\_ALBUM\_NAME IN ALBUMS.ALBUM\_NAME%TYPE,

I\_ALBUM\_RELEASE\_DATE IN ALBUMS.ALBUM\_RELEASE\_DATE%TYPE,

I\_ALBUM\_COVER IN ALBUMS.ALBUM\_COVER%TYPE);

PROCEDURE DELETE\_ALBUM (I\_ALBUM\_ID IN ALBUMS.ALBUM\_ID%TYPE);

------------------------------------SONGS---------------------------------------

PROCEDURE CREATE\_SONG (I\_SONG\_NAME IN SONGS.SONG\_NAME%TYPE,

I\_SONG\_FILE IN SONGS.SONG\_FILE%TYPE,

I\_ARTIST\_ID IN ARTISTS.ARTIST\_ID%TYPE);

PROCEDURE UPDATE\_SONG (I\_SONG\_ID IN SONGS.SONG\_ID%TYPE,

I\_SONG\_NAME IN SONGS.SONG\_NAME%TYPE,

I\_SONG\_FILE IN SONGS.SONG\_FILE%TYPE,

I\_ARTIST\_ID IN ARTISTS.ARTIST\_ID%TYPE);

PROCEDURE DELETE\_SONG (I\_SONG\_ID IN SONGS.SONG\_ID%TYPE);

PROCEDURE ADD\_SONG\_TO\_ALBUM (I\_SONG\_ID IN SONGS.SONG\_ID%TYPE,

I\_ALBUM\_ID IN ALBUMS.ALBUM\_ID%TYPE);

PROCEDURE DELETE\_SONG\_FROM\_ALBUM (I\_SONG\_ID IN SONGS.SONG\_ID%TYPE,

I\_ALBUM\_ID IN ALBUMS.ALBUM\_ID%TYPE);

END;

CREATE OR REPLACE PACKAGE BODY ADMIN\_PACKAGE IS

-------------------------------------USERS--------------------------------------

-------------------------DELETE\_USER-------------------------

PROCEDURE DELETE\_USER

(I\_USER\_LOGIN IN USERS.USER\_LOGIN%TYPE)

IS

CNT NUMBER;

BEGIN

SELECT COUNT(\*) INTO CNT FROM USERS WHERE USERS.USER\_LOGIN = I\_USER\_LOGIN;

IF (CNT != 0) THEN

DELETE FROM USERS WHERE USERS.USER\_LOGIN = I\_USER\_LOGIN;

COMMIT;

ELSE

RAISE\_APPLICATION\_ERROR(-20003, 'User is not found');

END IF;

END DELETE\_USER;

------------------------------------ARTISTS-------------------------------------

-------------------------CREATE\_ARTIST-------------------------

PROCEDURE CREATE\_ARTIST

(I\_ARTIST\_NAME IN ARTISTS.ARTIST\_NAME%TYPE,

I\_ARTIST\_PHOTO IN ARTISTS.ARTIST\_PHOTO%TYPE)

IS

CNT NUMBER;

BEGIN

SELECT COUNT(\*) INTO CNT FROM ARTISTS WHERE I\_ARTIST\_NAME = ARTISTS.ARTIST\_NAME;

IF (CNT != 0) THEN

INSERT INTO ARTISTS (ARTIST\_NAME, ARTIST\_PHOTO) VALUES (I\_ARTIST\_NAME, I\_ARTIST\_PHOTO);

COMMIT;

ELSE

RAISE\_APPLICATION\_ERROR(-20005, 'Artist with this name already exists');

END IF;

END CREATE\_ARTIST;

-------------------------UPDATE\_ARTIST------------------------

PROCEDURE UPDATE\_ARTIST

(I\_ARTIST\_ID IN ARTISTS.ARTIST\_ID%TYPE,

I\_ARTIST\_NAME IN ARTISTS.ARTIST\_NAME%TYPE,

I\_ARTIST\_PHOTO IN ARTISTS.ARTIST\_PHOTO%TYPE)

IS

CNT NUMBER;

BEGIN

SELECT COUNT(\*) INTO CNT FROM ARTISTS WHERE I\_ARTIST\_ID = ARTISTS.ARTIST\_ID;

IF (CNT != 0) THEN

UPDATE ARTISTS SET ARTIST\_NAME = I\_ARTIST\_NAME,

ARTIST\_PHOTO = I\_ARTIST\_PHOTO

WHERE ARTIST\_ID = I\_ARTIST\_ID;

COMMIT;

ELSE

RAISE\_APPLICATION\_ERROR(-20017, 'Artist not found');

END IF;

END UPDATE\_ARTIST;

-------------------------DELETE\_ARTIST------------------------

PROCEDURE DELETE\_ARTIST

(I\_ARTIST\_ID IN ARTISTS.ARTIST\_ID%TYPE)

IS

CNT NUMBER;

BEGIN

SELECT COUNT(\*) INTO CNT FROM ARTISTS WHERE I\_ARTIST\_ID = ARTISTS.ARTIST\_ID;

IF (CNT != 0) THEN

DELETE FROM ARTISTS WHERE ARTIST\_ID = I\_ARTIST\_ID;

COMMIT;

ELSE

RAISE\_APPLICATION\_ERROR(-20017, 'Artist not found');

END IF;

END DELETE\_ARTIST;

------------------------------------ALBUMS--------------------------------------

-------------------------CREATE\_ALBUM-------------------------

PROCEDURE CREATE\_ALBUM

(I\_ARTIST\_ID IN ARTISTS.ARTIST\_ID%TYPE,

I\_ALBUM\_NAME IN ALBUMS.ALBUM\_NAME%TYPE,

I\_ALBUM\_RELEASE\_DATE IN ALBUMS.ALBUM\_RELEASE\_DATE%TYPE,

I\_ALBUM\_COVER IN ALBUMS.ALBUM\_COVER%TYPE)

IS

CNT NUMBER;

BEGIN

SELECT COUNT(\*) INTO CNT FROM ARTIST\_ALBUM\_SONG\_VIEW

WHERE ALBUM\_NAME = I\_ALBUM\_NAME AND ARTIST\_ID = I\_ARTIST\_ID;

IF (CNT = 0) THEN

INSERT INTO ALBUMS (ALBUM\_NAME, ALBUM\_RELEASE\_DATE, ALBUM\_COVER)

VALUES (I\_ALBUM\_NAME, I\_ALBUM\_RELEASE\_DATE, I\_ALBUM\_COVER);

COMMIT;

ELSE

RAISE\_APPLICATION\_ERROR(-20006, 'This artist already has this album');

END IF;

END CREATE\_ALBUM;

-------------------------UPDATE\_ALBUM------------------------

PROCEDURE UPDATE\_ALBUM

(I\_ALBUM\_ID IN ALBUMS.ALBUM\_ID%TYPE,

I\_ALBUM\_NAME IN ALBUMS.ALBUM\_NAME%TYPE,

I\_ALBUM\_RELEASE\_DATE IN ALBUMS.ALBUM\_RELEASE\_DATE%TYPE,

I\_ALBUM\_COVER IN ALBUMS.ALBUM\_COVER%TYPE)

IS

CNT NUMBER;

BEGIN

SELECT COUNT(\*) INTO CNT FROM ALBUMS WHERE ALBUM\_ID = I\_ALBUM\_ID;

IF (CNT != 0) THEN

UPDATE ALBUMS SET ALBUM\_NAME = I\_ALBUM\_NAME,

ALBUM\_RELEASE\_DATE = I\_ALBUM\_RELEASE\_DATE,

ALBUM\_COVER = I\_ALBUM\_COVER

WHERE ALBUM\_ID = I\_ALBUM\_ID;

COMMIT;

ELSE

RAISE\_APPLICATION\_ERROR(-20018, 'Album not found');

END IF;

END UPDATE\_ALBUM;

-------------------------DELETE\_ALBUM------------------------

PROCEDURE DELETE\_ALBUM

(I\_ALBUM\_ID IN ALBUMS.ALBUM\_ID%TYPE)

IS

CNT NUMBER;

BEGIN

SELECT COUNT(\*) INTO CNT FROM ALBUMS WHERE ALBUM\_ID = I\_ALBUM\_ID;

IF (CNT != 0) THEN

DELETE FROM ALBUMS WHERE ALBUM\_ID = I\_ALBUM\_ID;

COMMIT;

ELSE

RAISE\_APPLICATION\_ERROR(-20018, 'Album not found');

END IF;

END DELETE\_ALBUM;

------------------------------------SONGS---------------------------------------

-------------------------CREATE\_SONG-------------------------

PROCEDURE CREATE\_SONG

(I\_SONG\_NAME IN SONGS.SONG\_NAME%TYPE,

I\_SONG\_FILE IN SONGS.SONG\_FILE%TYPE,

I\_ARTIST\_ID IN ARTISTS.ARTIST\_ID%TYPE)

IS

CNT NUMBER;

BEGIN

SELECT COUNT(\*) INTO CNT FROM ARTIST\_ALBUM\_SONG\_VIEW

WHERE I\_SONG\_NAME = SONG\_NAME AND ARTIST\_ID = I\_ARTIST\_ID;

IF (CNT = 0) THEN

INSERT INTO SONGS (SONG\_NAME, SONG\_FILE, ARTIST\_ID) VALUES (I\_SONG\_NAME, I\_SONG\_FILE, I\_ARTIST\_ID);

COMMIT;

ELSE

RAISE\_APPLICATION\_ERROR(-20007, 'This artist already has this song');

END IF;

END CREATE\_SONG;

-------------------------UPDATE\_SONG-------------------------

PROCEDURE UPDATE\_SONG

(I\_SONG\_ID IN SONGS.SONG\_ID%TYPE,

I\_SONG\_NAME IN SONGS.SONG\_NAME%TYPE,

I\_SONG\_FILE IN SONGS.SONG\_FILE%TYPE,

I\_ARTIST\_ID IN ARTISTS.ARTIST\_ID%TYPE)

IS

CNT NUMBER;

BEGIN

SELECT COUNT(\*) INTO CNT FROM SONGS

WHERE I\_SONG\_ID = SONG\_ID;

IF (CNT != 0) THEN

UPDATE SONGS SET SONG\_NAME = I\_SONG\_NAME,

SONG\_FILE = I\_SONG\_FILE,

ARTIST\_ID = I\_ARTIST\_ID

WHERE I\_SONG\_ID = SONG\_ID;

COMMIT;

ELSE

RAISE\_APPLICATION\_ERROR(-20019, 'Song not found');

END IF;

END UPDATE\_SONG;

-------------------------DELETE\_SONG-------------------------

PROCEDURE DELETE\_SONG

(I\_SONG\_ID IN SONGS.SONG\_ID%TYPE)

IS

CNT NUMBER;

BEGIN

SELECT COUNT(\*) INTO CNT FROM SONGS

WHERE I\_SONG\_ID = SONG\_ID;

IF (CNT != 0) THEN

DELETE FROM SONGS WHERE SONG\_ID = I\_SONG\_ID;

COMMIT;

ELSE

RAISE\_APPLICATION\_ERROR(-20019, 'Song not found');

END IF;

END DELETE\_SONG;

-------------------------ADD\_SONG\_TO\_ALBUM-------------------------

PROCEDURE ADD\_SONG\_TO\_ALBUM

(I\_SONG\_ID IN SONGS.SONG\_ID%TYPE,

I\_ALBUM\_ID IN ALBUMS.ALBUM\_ID%TYPE)

IS

CNT NUMBER;

BEGIN

SELECT COUNT(\*) INTO CNT FROM ALBUM\_SONG

WHERE ALBUM\_ID = I\_ALBUM\_ID AND SONG\_ID = I\_SONG\_ID;

IF (CNT = 0) THEN

INSERT INTO ALBUM\_SONG(ALBUM\_ID, SONG\_ID) VALUES (I\_ALBUM\_ID, I\_SONG\_ID);

COMMIT;

ELSE

RAISE\_APPLICATION\_ERROR(-20008, 'This album already has this song');

END IF;

END ADD\_SONG\_TO\_ALBUM;

-------------------------DELETE\_SONG\_FROM\_ALBUM-------------------------

PROCEDURE DELETE\_SONG\_FROM\_ALBUM

(I\_SONG\_ID IN SONGS.SONG\_ID%TYPE,

I\_ALBUM\_ID IN ALBUMS.ALBUM\_ID%TYPE)

IS

CNT NUMBER;

BEGIN

SELECT COUNT(\*) INTO CNT FROM ALBUM\_SONG

WHERE ALBUM\_ID = I\_ALBUM\_ID AND SONG\_ID = I\_SONG\_ID;

IF (CNT > 0) THEN

DELETE FROM ALBUM\_SONG WHERE ALBUM\_ID = I\_ALBUM\_ID AND SONG\_ID = I\_SONG\_ID;

COMMIT;

ELSE

RAISE\_APPLICATION\_ERROR(-20009, 'This album does not have this song');

END IF;

END DELETE\_SONG\_FROM\_ALBUM;

## Приложение Д

--xml export

SELECT file\_name FROM dba\_data\_files;

CREATE OR REPLACE DIRECTORY EXPORT\_DATA AS 'C:/oracle\_db/xml/export';

CREATE OR REPLACE DIRECTORY IMPORT\_DATA AS 'C:/oracle\_db/xml/import';

CREATE OR REPLACE PROCEDURE export\_users\_to\_xml

IS

doc DBMS\_XMLDOM.DOMDocument;

xdata XMLTYPE;

CURSOR xmlcur IS

SELECT XMLELEMENT(

"USERS",

XMLAttributes('http://www.w3.org/2001/XMLSchema' AS "xmlns:xsi",

'http://www.oracle.com/Users.xsd' AS "xsi:nonamespaceSchemaLocation"),

XMLAGG(XMLELEMENT("User",

XMLELEMENT("login", USER.USER\_LOGIN),

xmlelement("password", USER.USER\_PASSWORD),

))) from Character;

BEGIN

open xmlcur;

LOOP

FETCH xmlcur INTO xdata;

EXIT WHEN xmlcur%notfound;

END LOOP;

CLOSE xmlcur;

doc := DBMS\_XMLDOM.NewDOMDocument(xdata);

DBMS\_XMLDOM.WRITETOFILE(doc, 'EXPORT\_DATA/characters.xml');

END;

CREATE OR REPLACE PROCEDURE import\_USERS\_from\_xml

IS

L\_CLOB CLOB;

L\_BFILE BFILE := BFILENAME('IMPORT\_DATA', 'characters.xml');

L\_DEST\_OFFSET INTEGER := 1;

L\_SRC\_OFFSET INTEGER := 1;

L\_BFILE\_CSID NUMBER := 0;

L\_LANG\_CONTEXT INTEGER := 0;

L\_WARNING INTEGER := 0;

P DBMS\_XMLPARSER.PARSER;

V\_DOC DBMS\_XMLDOM.DOMDOCUMENT;

V\_ROOT\_ELEMENT DBMS\_XMLDOM.DOMELEMENT;

V\_CHILD\_NODES DBMS\_XMLDOM.DOMNODELIST;

V\_CURRENT\_NODE DBMS\_XMLDOM.DOMNODE;

ch Administrator.Character%ROWTYPE;

BEGIN

DBMS\_LOB.CREATETEMPORARY (L\_CLOB, TRUE);

DBMS\_LOB.FILEOPEN(L\_BFILE, DBMS\_LOB.FILE\_READONLY);

DBMS\_LOB.LOADCLOBFROMFILE(DEST\_LOB => L\_CLOB, SRC\_BFILE => L\_BFILE, AMOUNT => DBMS\_LOB.LOBMAXSIZE,

DEST\_OFFSET => L\_DEST\_OFFSET, SRC\_OFFSET => L\_SRC\_OFFSET, BFILE\_CSID => L\_BFILE\_CSID,

LANG\_CONTEXT => L\_LANG\_CONTEXT, WARNING => L\_WARNING);

DBMS\_LOB.FILECLOSE(L\_BFILE);

COMMIT;

P := DBMS\_XMLPARSER.NEWPARSER;

DBMS\_XMLPARSER.PARSECLOB(P, L\_CLOB);

V\_DOC := DBMS\_XMLPARSER.GETDOCUMENT(P);

V\_ROOT\_ELEMENT := DBMS\_XMLDOM.Getdocumentelement(V\_DOC);

V\_CHILD\_NODES := DBMS\_XMLDOM.GETCHILDRENBYTAGNAME(V\_ROOT\_ELEMENT, '\*');

FOR i IN 0 .. DBMS\_XMLDOM.GETLENGTH(V\_CHILD\_NODES) - 1 LOOP

V\_CURRENT\_NODE := DBMS\_XMLDOM.ITEM(V\_CHILD\_NODES, i);

DBMS\_XSLPROCESSOR.VALUEOF(V\_CURRENT\_NODE,

'login/text()', ch.login);

DBMS\_XSLPROCESSOR.VALUEOF(V\_CURRENT\_NODE,

'PASSWORD/text()', ch.USER\_PASSWORD);

INSERT INTO CP\_ADMIN.USERS (USER\_LOGIN, USER\_PASSWORD)

VALUES (ch.login, ch.password);

END LOOP;

DBMS\_LOB.FREETEMPORARY(L\_CLOB);

DBMS\_XMLPARSER.FREEPARSER(P);

DBMS\_XMLDOM.FREEDOCUMENT(V\_DOC);

COMMIT;

END;

BEGIN

import\_users\_from\_xml();

end;