**Министерство транспорта Российской Федерации**

**Федеральное государственное автономное образовательное учреждение высшего образования «Российский университет транспорта»**

**((РУТ)МИИТ)**

**Институт управления и цифровых технологий (ИУЦТ)**

**Кафедра «Цифровые технологии управления транспортными средствами»**

**(ЦТУТП)**

**Технологии хранения данных**

**Курсовая работа**

**на тему «Музыкальный магазин (Музыкальный интернет сервис)»**

Выполнил: Толстиков А. А.

Группа: УВП – 212

Проверил: старший преподаватель Дутова Е. А.

**Москва 2023 г.**

Оглавление

[**1. Задание** 3](#_Toc138087470)

[**2. Описание предметной области «Музыкальный магазин (Музыкальный интернет сервис)»** 4](#_Toc138087471)

[**3. ER-модели базы данных «Ветеринарная клиника»** 5](#_Toc138087472)

[3.1 ER-модель в нотации Чена 5](#_Toc138087473)

[3.2 ER-модель в нотации IDEF1X 5](#_Toc138087474)

[3.3 Описание схем ER-модели базы данных 5](#_Toc138087475)

[**4. Запросы на языке SQL на создание таблиц в СУБД PostgreSQL** 6](#_Toc138087476)

[4.1 Заполненная база данных 8](#_Toc138087477)

[**5. Запросы к базе данных на языках SQL, реляционной алгебры, реляционных исчислений** 10](#_Toc138087478)

[5.1 Запросы на языке SQL 10](#_Toc138087479)

[5.2 Запросы на языке реляционной алгебры и реляционных исчислений 10](#_Toc138087480)

[**6. Стек курсовой работы** 13](#_Toc138087481)

[**7. Демонстрация кода программы** 13](#_Toc138087482)

[7.1 Маппинг-классы 13](#_Toc138087483)

[7.2 Запросы, реализованные при помощи Hibernate 13](#_Toc138087484)

[7.3 Результат выполнения запросов 14](#_Toc138087485)

[**8. Приложение А. Пример запросов на заполнение таблиц данными на языке SQL** 14](#_Toc138087486)

[**9. Приложение Б. Код программы** 14](#_Toc138087487)

# **1. Задание**

Разработать приложение на одном из языков высокого уровня (C++, C#, Java, Python и др.), которое позволяет работать с базой данных. Приложение должно иметь визуальный интерфейс, позволяющий вводить, корректировать и удалять информацию из таблиц базы данных. Приложение может быть как локальным, так и веб-приложением. Приложение должно выполнять поисковые запросы, которые были реализованы в лабораторных работах. Приложение должно позволять вводить параметры запроса, а не использовать конкретные константы запросов, которые были в запросах, приведенных в лабораторных работах. Возможна реализация интерфейса, который позволит вводить произвольный запрос на языке SQL и показывать результат выполнения запроса в виде таблицы. Предметная область для базы данных выбирается в соответствии с заданием на выполнение лабораторных работ. Схема базы данных должна быть спроектирована по алгоритмам проектирования реляционной модели базы данных (алгоритмы: Фэджина, Делобеля-Кейси, Бернштейна). Для реализации проекта выбирается схема реляционной базы данных, полученная по одному из алгоритмов проектирования. При выполнении курсовой работы используется такая же система управления базами данных как при выполнении лабораторных работ.

# **2. Описание предметной области «Музыкальный магазин (Музыкальный интернет сервис)»**

Для лабораторных работ была выбрана предметная область «Музыкальный магазин (Музыкальный интернет сервис)». Данная предметная область представляет собой сайт, который содержит в себе информацию о музыке, актерах, альбомах и плейлистах. Данные таблицы содержат такие поля как: время трека, жанр, его название. Каждый трек имеет своего автора, у которого есть такие поля как: дата рождения, псевдоним/имя/, краткая биография; также каждый трек имеет альбом, у которого есть такие поля как: количество исполнителей в альбоме, время, название, количество треков. Пользователь может добавить каждый трек в свой личный плейлист, у плейлиста есть такие поля как: количество прослушиваний, категория, создатель плейлиста, название плейлиста.  Все таблицы в базе данных связаны при помощи вторичных ключей, которые позволяют представить всю базу данных в виде определенной структуры, соответствующей определенной нормальной форме. База данных, хранящая в себе информацию по предметной области «Ветеринарная клиника» соответствует 3 нормальной форме, что говорит о том, что она соответствует и 1 и 2 нормальной форме.

# **3. ER-модели базы данных «Ветеринарная клиника»**

## 3.1 ER-модель в нотации Чена

Изображение выглядит как диаграмма, линия, шаблон, оригами

Автоматически созданное описание

## 3.2 ER-модель в нотации IDEF1X

Изображение выглядит как диаграмма, линия, текст, Параллельный

Автоматически созданное описание

## 3.3 Описание схем ER-модели базы данных

Данная ER-модель базы данных включает в себя следующие таблицы с полями:

user(**login**, name, surname, password, email, phone number, subscribe status)

playlist(**id**, name, shortinfo , creator, audition rate, gender)

musical\_track(**id**, name, time, genre, album\_id)

album(**id**, track\_count, number\_of\_singers, time, name, creator\_id)

performer(**id**, name, short info, audition rate, born)

Каждая таблица связана с другой при помощи вторичных ключей. Вторичные ключи выделены в списке выше при помощи подчеркивания. Таким образом, таблица пользователей связана с таблицей плейлистов при помощи связи один-ко-многим. Это означает, что у одного пользователя может быть плейлистов, то есть одной сущности пользователя могут соответствовать несколько записей в сущности плейлист. Плейлист связан с музыкальным треком один-ко-многим. Это означает, что плейлист может содержать несколько треков, каждый из которых описывает отдельная запись в таблице музыкальный трек. В свою очередь сущность музыкальный трек имеет связь с сущностью альбом для определения к какому альбому принадлежит музыкальный трек. Исполнитель также связан с сущностью альбом для определения, какой альбом выпустил данный исполнитель. Данные схемы подтверждают, что база данных «Ветеринарная клиника» находится в 3, а значит и в 1 и 2 нормальной форме.

# **4. Запросы на языке SQL на создание таблиц в СУБД PostgreSQL**

CREATE TABLE user(

  login INTEGER NOT NULL,

  name VARCHAR,

  surname VARCHAR,

  password VARCHAR NOT NULL,

  email VARCHAR NOT NULL,

  phone number VARCHAR

  subscribe status VARCHAR NOT NULL,

  PRIMARY KEY (login)

            FOREIGN KEY(login user) REFERENCES playlist (id)  ON DELETE CASCADE

);

CREATE TABLE playlist (

  id INTEGER NOT NULL,

  name VARCHAR NOT NULL,

  short info VARCHAR,

  creator VARCHAR NOT NULL,

  audition rate INTEGER NOT NULL,

  PRIMARY KEY (id)

             FOREIGN KEY(id\_track) REFERENCES musical track(id) ON DELETE CASCADE

);

CREATE TABLE musical track (

  id INTEGER NOT NULL,

  name VARCHAR NOT NULL,

  time INTEGER NOT NULL,

  genre VARCHAR NOT NULL,

  PRIMARY KEY (id),

  FOREIGN KEY (album\_id) REFERENCES album (id) ON DELETE CASCADE

  FOREIGN KEY (performer\_id) REFERENCES performer (id) ON DELETE CASCADE

);

CREATE TABLE performer (

  id INTEGER NOT NULL,

  name VARCHAR NOT NULL,

  short info VARCHAR,

  audition rate INTEGER NOT NULL,

  born VARVHAR NOT NULL,

  PRIMARY KEY (id),

);

CREATE TABLE album (

  id INTEGER NOT NULL,

  track count INTEGER NOT NULL,

  number of singers INTEGER NOT NULL

  time INTEGER,

  name VARCHAR NOT NULL,

  PRIMARY KEY (id),

FOREIGN KEY(performer\_id) REFERENCES performer(id) ON DELETE CASCADE

);

## 4.1 Заполненная база данных

1. Таблица user

Изображение выглядит как текст, чек, Шрифт, документ

Автоматически созданное описание

2. Таблица playlist

Изображение выглядит как текст, снимок экрана, число, Шрифт

Автоматически созданное описание

3. Таблица musical\_track

Изображение выглядит как текст, число, снимок экрана, Шрифт

Автоматически созданное описание

4. Таблица album

Изображение выглядит как текст, число, Шрифт, снимок экрана

Автоматически созданное описание

5. Таблица performer

Изображение выглядит как текст, снимок экрана, число, Шрифт

Автоматически созданное описание

# **5. Запросы к базе данных на языках SQL, реляционной алгебры, реляционных исчислений**

## 5.1 Запросы на языке SQL

1. Запрос на названия и время треков если время больше 70.

SELECT \* FROM musical\_track WHERE musical\_track.time>70

2. Запрос, позволяющий вывести имена исполнителей с определенным названием альбома.

SELECT performer.name FROM performer JOIN album ON performer.id = album.creator\_id WHERE album.name = “EP”.

3. Запрос выводящий все треки указанного альбома и его исполнителя

SELECT album.name, musical\_track.name, performer.name FROM album JOIN musical\_track ON album.id = musical\_track.album\_id

JOIN performer ON album.creator\_id = performer.id WHERE album.name =”EP”

4. Запрос, выводящий все треки с жанром рэп и названия их альбомов.

SELECT musical\_track.name FROM musical track JOIN album on album.id =musical\_track.album\_id WHERE musical\_track.name = ‘Rap’.

5. Запрос, выводящий имена исполнителей чье кол-во прослушиваний больше 10000

SELECT perfomer.name FROM performer WHERE performer.audition\_rate >10000

6. Запрос, позволяющий добавить трек

INSERT INTO musical track VALUES(6, '666PRADA', 125, 'Rap', 1)

7. Запрос, позволяющий удалить плейлист с определенным создателем

DELETE FROM playslist WHERE creator = ‘Tylor’

8. Запрос, позволяющий изменить имя пользователя с определенным id

UPDATE user SET name = ‘Vanya’ WHERE user.login = 100000

## 5.2 Запросы на языке реляционной алгебры и реляционных исчислений

1. Запрос на названия и время треков если время больше 70.

S1 = π id (σ time > 70 Musical\_track)

Range Musical\_track M

GET S (M.client\_id) : (M.name > 70)

2. Запрос, позволяющий вывести имена исполнителей с определенным названием альбома.

S1 = π album.id (σ name = “EP” Album)

S2 = π id (Performer) ⋈ S1

S3 = π performer.name (Performer) ⋈ S2

Range Performer P

Range Album A

GET S (P.name) : ∃P(A.name = ”EP” & P.id = A.performer\_id)

3. Запрос, выводящий все треки указанного альбома и его исполнителя

S1 = π album.id (σ name = “NATIVE” Album)

S2 = π performer.name (Performer) ⋈ S1

S3 = π musical\_track.name (Album) ⋈ S1

S4 = S2 ⋈ S3

Range Performer.name

Range Album.name

Range Musical\_track.name

GET S (P.name, A.name, M.name) : ∃P∃A∃M(A.name = “NATIVE” & P.name = A.performer\_id &A.id = M.album\_id)

4. Запрос, выводящий все треки с жанром рэп и названия их альбомов.

S1 = π musical\_track.name (σ genre = “RAP” Musical track)

S2 = π album.name (Album) ⋈ S1

Range A Album

Range M Musical track

GET S (A.name, M.name) : ∃A∃M(M.genre = “RAP” & M.id = A.musical\_track.id)

5. Запрос, выводящий имена исполнителей чье кол-во прослушиваний больше 10000

S1 = π id(σ audition\_rate > 10000 Performer)

S2 = π name (Performer) ⋈ S1

Range Performer P

GET S (P.name) : (P.audition\_rate > 10000)

6. Запрос, позволяющий добавить клиента

S1 = “6” x “666PRADA” x 125 x “Rap” x “1”

Musical track= (Musical track) S1

P.id = “6”

P.name = “666PRADA”

P.time = 125

P.genre = “Rap”

P.album\_id = “+79159897643”

PUT S(Client)

7. Запрос, позволяющий удалить плейлист с определенным создателем S1 = σ creator = ”Tylor” User

Playlist= Playlist – S1

HOLD S(Playlist) : P.creator = “Tylor”

DELETE S

8. Запрос, позволяющий изменить имя пользователя с определенным id

S1 = σ login= 100000 User

S2 = π name, surname, subscribe\_status, phone number, password, S1

S3 = S2 x (100000)

User = (User – S1) U S3

HOLD S (Pet) : P.pet\_id = “1” S.gender = мужской

UPDATE S

# **6. Стек курсовой работы**

При выполнении данной курсовой работы было принято решение использовать СУБД PostgreSQL, язык программирования Java. Для связи реляционной базы данных и объектно-ориентированного языка программирования было принято решение использовать ОРМ Hibernate. Это позволило преобразовать таблицы базы данных в отдельные сущности при помощи так называемого маппинга. Для каждой таблицы базы данных были созданы маппинг-классы. Это позволило работать с сущностями базы данных как с отдельными объектами классов, что является объектно-ориентированным подходом в программировании. Это упрощает работу с базой данных. Запросы к базе данных были написаны на языке HQL, который является языком ОРМ Hibernate. Это позволяет один раз написать запрос и использовать его для объектов определенного типа (сущность из базы данных) не ограниченное количество раз.

# **7. Демонстрация кода программы**

## 7.1 Маппинг-классы

Изображение выглядит как текст, снимок экрана

Автоматически созданное описаниеИзображение выглядит как текст, снимок экрана

Автоматически созданное описание

Изображение выглядит как текст, снимок экрана, документ

Автоматически созданное описание

Изображение выглядит как текст, снимок экрана

Автоматически созданное описание

Изображение выглядит как текст, снимок экрана

Автоматически созданное описание

## 7.2 Запросы, реализованные при помощи Hibernate

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Изображение выглядит как текст, снимок экрана, программное обеспечение, дисплей

Автоматически созданное описание

Изображение выглядит как текст, снимок экрана

Автоматически созданное описание

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

## 7.3 Результат выполнения запросов

Изображение выглядит как текст, снимок экрана, Шрифт, число

Автоматически созданное описание

Изображение выглядит как текст, Шрифт, линия, снимок экрана

Автоматически созданное описание

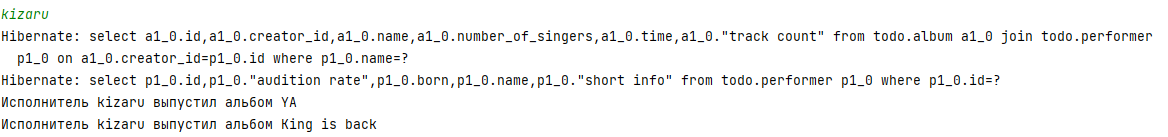
Изображение выглядит как текст, Шрифт, снимок экрана, линия

Автоматически созданное описание Изображение выглядит как текст, Шрифт, снимок экрана, алгебра

Автоматически созданное описание

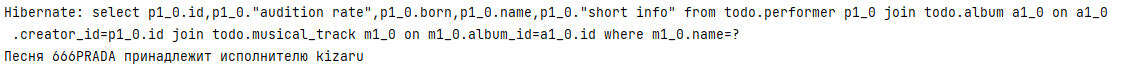
Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание



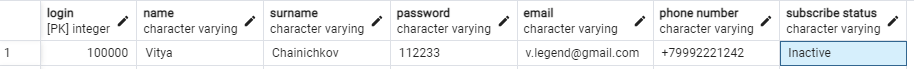
Изображение выглядит как текст, снимок экрана, Шрифт, информация

Автоматически созданное описание



# **8. Приложение А. Пример запросов на заполнение таблиц данными на языке SQL**

INSERT INTO user (name, surname, password, email, phone number, subscribe status) VALUES ('Vitya', ' Chainichkov ', '112233', 'v.legend@gmail.com', '+79992221242 ', 'Inactive');



INSERT INTO playlist (name, short info, creator, audition rate) VALUES ('Best car', 'music for driving', 'Davidych', 10422), ('Popsa', 'about jail', 'MishaKrug', 92487);

Изображение выглядит как текст, Шрифт, снимок экрана, линия

Автоматически созданное описание

INSERT INTO performer(name, short info, audition rate, born VALUES ('OG BUDA', 'Freak', 21521, '1995'), ('kizaru', 'Russian spainboy', 2521518, '1990')

Изображение выглядит как текст, Шрифт, снимок экрана, линия

Автоматически созданное описание

INSERT INTO musical track (name, time, genre) VALUES('Hate', 97, 'techno'),( 'Hope',125, 'trans'),( 'Playa',85, 'rap');

Изображение выглядит как текст, Шрифт, снимок экрана, число

Автоматически созданное описание

Insert INTO album (track count, number of singers, time, name) VALUES (12,2,1065, 'YA'), (1,1,98, 'EP'), (5,1,950, 'Idiot'), (9,1,897, 'MISS'),

Изображение выглядит как текст, Шрифт, число, линия

Автоматически созданное описание

# **9. Приложение Б. Код программы**

import utils.AbstractJPA;  
import utils.HibernateUtil;  
  
import java.util.Scanner;  
  
public class AddMusicalTrack extends AbstractJPA {  
 Scanner scanner = new Scanner(System.*in*);  
  
 public void add() throws Exception {  
 Session session = HibernateUtil.*getSessionFactory*().openSession();  
 session.getTransaction().begin();  
 try {  
 MusicalTrack musicalTrack = new MusicalTrack();  
 System.*out*.print("Введите название трека:");  
 musicalTrack.setName(scanner.nextLine());  
 System.*out*.print("Введите длительность трека:");  
 musicalTrack.setTime(Integer.*parseInt*(scanner.nextLine()));  
 System.*out*.print("Введите жанр трека:");  
 musicalTrack.setGenre(scanner.nextLine());  
 System.*out*.print("Введите номер альбома к которому принадлежит трек трека:");  
 Integer album =Integer.*parseInt*(scanner.nextLine());  
 musicalTrack.setAlbumId(album);  
 session.persist(musicalTrack);  
 session.getTransaction().commit();  
 session.close();  
 HibernateUtil.*close*();  
 }  
 catch (Exception e) {  
 System.*out*.println(e.getMessage());  
 throw new Exception("Укажите правильное значение для номер альбома или длительности трека");  
  
 }  
 }  
}

package doing;  
  
import entity\_try.Playlist;  
import utils.AbstractJPAPers;  
  
import java.util.Scanner;  
  
public class DeletePlaylist extends AbstractJPAPers {  
 Scanner scanner = new Scanner(System.*in*);  
  
 public void deleteByCreator() {  
 init();  
 System.*out*.println("Введите id создателя, чей плейлист хотите удалить");  
 String id = scanner.nextLine();  
 Playlist playlist = entityManager.getReference(Playlist.class, id);  
 entityManager.remove(playlist);  
 close();  
  
 }  
}

package doing;  
  
import entity\_try.Album;  
import entity\_try.MusicalTrack;  
import entity\_try.Performer;  
import jakarta.persistence.TypedQuery;  
import org.hibernate.Session;  
import utils.HibernateUtil;  
  
import java.util.List;  
  
public class SearchQuerry {  
 public static List<MusicalTrack> trackTime(int time){  
 try(Session session = HibernateUtil.*getSessionFactory*().openSession()) {  
 TypedQuery<MusicalTrack> typedQuery = session.createQuery("SELECT m FROM MusicalTrack m WHERE m.time >:time",MusicalTrack.class);  
 typedQuery.setParameter("time",time);  
 return typedQuery.getResultList();  
 }  
 }  
 public static List<Album> performerAs(String name){  
 try(Session session = HibernateUtil.*getSessionFactory*().openSession()) {  
 TypedQuery<Album> typedQuery = session.createQuery("SELECT a FROM Album a JOIN Performer p ON a.performerByCreatorId.id = p.id WHERE p.name =:name", Album.class);  
 typedQuery.setParameter("name",name);  
 return typedQuery.getResultList();  
 }  
 }  
 public static List<MusicalTrack> musicalTrackAs(String name){  
 try(Session session = HibernateUtil.*getSessionFactory*().openSession()) {  
 TypedQuery<MusicalTrack> typedQuery = session.createQuery("SELECT m FROM MusicalTrack m JOIN Album a ON m.albumByAlbumId.id = a.id JOIN Performer p ON a.performerByCreatorId.id = p.id WHERE p.name =:name", MusicalTrack.class);  
 typedQuery.setParameter("name",name);  
 return typedQuery.getResultList();  
 }  
 }  
 public static List<Performer> musicalTrackBy(String name) {  
 try (Session session = HibernateUtil.*getSessionFactory*().openSession()) {  
 TypedQuery<Performer> typedQuery = session.createQuery("SELECT p FROM Performer p JOIN Album a ON a.performerByCreatorId.id = p.id JOIN MusicalTrack m ON m.albumByAlbumId.id = a.id WHERE m.name =:name", Performer.class);  
 typedQuery.setParameter("name", name);  
 return typedQuery.getResultList();  
 }  
 }  
}

package doing;  
  
import entity\_try.Userinfo;  
import utils.AbstractJPAPers;  
  
import java.util.Scanner;  
  
public class UpdateUserInfo extends AbstractJPAPers {  
 Scanner scanner = new Scanner(System.*in*);  
  
 public Integer searchLogin() throws Exception {  
 System.*out*.println("Введите login пользователя, чье имя хотите сменить:");  
 Integer login = Integer.*parseInt*(scanner.nextLine());  
 try {  
 return login;  
 } catch (Exception e) {  
 if (login < 100000) {  
 System.*out*.println(e.getMessage());  
 throw new Exception("Слишком маленькое значение 'login'");  
 }  
 else {  
 System.*out*.println(e.getMessage());  
 throw new Exception("Укажите правильное значение 'login' ");  
 }  
 }  
 }  
 public void newInfo(Integer id){  
 init();  
 Userinfo newUser = entityManager.find(Userinfo.class,id);  
  
 if(newUser == null){  
 System.*out*.println("Пользователь с данным id не найден.");  
 return;  
 }  
 System.*out*.println("Введите новое имя пользователя:");  
 newUser.setName(scanner.nextLine());  
 System.*out*.println("Введите новую фамилию пользователя(Enter если хотите пропустить данный пункт):");  
 String surname ="" + scanner.nextLine();  
 if (surname.equals("")) newUser.setSurname(null);  
 else newUser.setSurname(surname);  
 System.*out*.println("Введите новый пароль:");  
 newUser.setPassword(scanner.nextLine());  
 System.*out*.println("Введите новую эл.почту пользователя(Enter если хотите пропустить данный пункт):");  
 String mail = "" + scanner.nextLine();  
 if (mail.equals("")) newUser.setEmail(null);  
 else newUser.setEmail(mail);  
 System.*out*.println("Введите новый номер телефона(Enter если хотите пропустить данный пункт):");  
 String phone = scanner.nextLine();  
 if (phone.equals("")) newUser.setPhoneNumber(null);  
 else newUser.setPhoneNumber(phone);  
 newUser.setSubscribeStatus("Inactive");  
 System.*out*.println("Информация о пользователе успешно изменена");  
 close();  
  
 }  
}

package entity\_try;  
  
import jakarta.persistence.\*;  
  
import java.util.Objects;  
  
@Entity  
@Table(name = "album", schema = "todo", catalog = "music\_shop")  
public class Album {  
 @GeneratedValue(strategy = GenerationType.*IDENTITY*)  
 @Id  
 @Column(name = "id", nullable = false)  
 private short id;  
 @Basic  
 @Column(name = "track count", nullable = false)  
 private int trackCount;  
 @Basic  
 @Column(name = "number\_of\_singers", nullable = false)  
 private int numberOfSingers;  
 @Basic  
 @Column(name = "time", nullable = true)  
 private Integer time;  
 @Basic  
 @Column(name = "name", nullable = false, length = 32)  
 private String name;  
 @Basic  
 @Column(name = "creator\_id", nullable = false,insertable=false, updatable=false)  
 private int creatorId;  
  
 @ManyToOne  
 @JoinColumn(name = "creator\_id", referencedColumnName = "id")  
 private Performer performerByCreatorId;  
  
 public short getId() {  
 return id;  
 }  
  
 public void setId(short id) {  
 this.id = id;  
 }  
  
 public int getTrackCount() {  
 return trackCount;  
 }  
  
 public void setTrackCount(int trackCount) {  
 this.trackCount = trackCount;  
 }  
  
 public int getNumberOfSingers() {  
 return numberOfSingers;  
 }  
  
 public void setNumberOfSingers(int numberOfSingers) {  
 this.numberOfSingers = numberOfSingers;  
 }  
  
 public Integer getTime() {  
 return time;  
 }  
  
 public void setTime(Integer time) {  
 this.time = time;  
 }  
  
 public String getName() {  
 return name;  
 }  
  
 public void setName(String name) {  
 this.name = name;  
 }  
  
 public int getCreatorId() {  
 return creatorId;  
 }  
  
 public void setCreatorId(int creatorId) {  
 this.creatorId = creatorId;  
 }  
  
 @Override  
 public boolean equals(Object o) {  
 if (this == o) return true;  
 if (o == null || getClass() != o.getClass()) return false;  
 Album album = (Album) o;  
 return id == album.id && trackCount == album.trackCount && numberOfSingers == album.numberOfSingers && creatorId == album.creatorId && Objects.*equals*(time, album.time) && Objects.*equals*(name, album.name);  
 }  
  
 @Override  
 public int hashCode() {  
 return Objects.*hash*(id, trackCount, numberOfSingers, time, name, creatorId);  
 }  
}

package entity\_try;  
  
import jakarta.persistence.\*;  
  
import java.util.Objects;  
  
@Entity  
@Table(name = "musical\_track", schema = "todo", catalog = "music\_shop")  
public class MusicalTrack {  
 @GeneratedValue(strategy = GenerationType.*IDENTITY*)  
 @Id  
 @Column(name = "id", nullable = false)  
 private int id;  
 @Basic  
 @Column(name = "name", nullable = false, length = 32)  
 private String name;  
 @Basic  
 @Column(name = "time", nullable = false)  
 private int time;  
 @Basic  
 @Column(name = "genre", nullable = true, length = 16)  
 private String genre;  
 @Basic  
 @Column(name = "album\_id", nullable = true,insertable=false, updatable=false)  
 private Integer albumId;  
 @ManyToOne  
 @JoinColumn(name = "album\_id", referencedColumnName = "id")  
 private Album albumByAlbumId;  
  
 public int getId() {  
 return id;  
 }  
  
 public void setId(int id) {  
 this.id = id;  
 }  
  
 public String getName() {  
 return name;  
 }  
  
 public void setName(String name) {  
 this.name = name;  
 }  
  
 public int getTime() {  
 return time;  
 }  
  
 public void setTime(int time) {  
 this.time = time;  
 }  
  
 public String getGenre() {  
 return genre;  
 }  
  
 public void setGenre(String genre) {  
 this.genre = genre;  
 }  
  
 public Integer getAlbumId() {  
 return albumId;  
 }  
  
 public void setAlbumId(Integer albumId) {  
 this.albumId = albumId;  
 }  
  
 @Override  
 public boolean equals(Object o) {  
 if (this == o) return true;  
 if (o == null || getClass() != o.getClass()) return false;  
 MusicalTrack that = (MusicalTrack) o;  
 return id == that.id && time == that.time && Objects.*equals*(name, that.name) && Objects.*equals*(genre, that.genre) && Objects.*equals*(albumId, that.albumId);  
 }  
  
 @Override  
 public int hashCode() {  
 return Objects.*hash*(id, name, time, genre, albumId);  
 }  
  
 public Album getAlbumByAlbumId() {  
 return albumByAlbumId;  
 }  
  
 public void setAlbumByAlbumId(Album albumByAlbumId) {  
 this.albumByAlbumId = albumByAlbumId;  
 }  
  
 @Override  
 public String toString() {  
 return "MusicalTrack{" +  
 "id=" + id +  
 ", name='" + name + '\'' +  
 ", time=" + time +  
 ", genre='" + genre + '\'' +  
 ", albumId=" + albumId +  
 '}';  
 }  
}

package entity\_try;  
  
import jakarta.persistence.\*;  
  
import java.util.Collection;  
import java.util.Objects;  
  
@Entity  
@Table(name = "performer", schema = "todo", catalog = "music\_shop")  
public class Performer {  
 @GeneratedValue(strategy = GenerationType.*IDENTITY*)  
 @Id  
 @Column(name = "id", nullable = false)  
 private int id;  
 @Basic  
 @Column(name = "name", nullable = false, length = 32)  
 private String name;  
 @Basic  
 @Column(name = "short info", nullable = true, length = 150)  
 private String shortInfo;  
 @Basic  
 @Column(name = "audition rate", nullable = false)  
 private long auditionRate;  
 @Basic  
 @Column(name = "born", nullable = false, length = 4)  
 private String born;  
 @OneToMany(mappedBy = "performerByCreatorId")  
 private Collection<Album> albumsById;  
  
 public int getId() {  
 return id;  
 }  
  
 public void setId(int id) {  
 this.id = id;  
 }  
  
 public String getName() {  
 return name;  
 }  
  
 public void setName(String name) {  
 this.name = name;  
 }  
  
 public String getShortInfo() {  
 return shortInfo;  
 }  
  
 public void setShortInfo(String shortInfo) {  
 this.shortInfo = shortInfo;  
 }  
  
 public long getAuditionRate() {  
 return auditionRate;  
 }  
  
 public void setAuditionRate(long auditionRate) {  
 this.auditionRate = auditionRate;  
 }  
  
 public String getBorn() {  
 return born;  
 }  
  
 public void setBorn(String born) {  
 this.born = born;  
 }  
  
 @Override  
 public boolean equals(Object o) {  
 if (this == o) return true;  
 if (o == null || getClass() != o.getClass()) return false;  
 Performer performer = (Performer) o;  
 return id == performer.id && auditionRate == performer.auditionRate && Objects.*equals*(name, performer.name) && Objects.*equals*(shortInfo, performer.shortInfo) && Objects.*equals*(born, performer.born);  
 }  
  
 @Override  
 public int hashCode() {  
 return Objects.*hash*(id, name, shortInfo, auditionRate, born);  
 }  
  
 public Collection<Album> getAlbumsById() {  
 return albumsById;  
 }  
  
 public void setAlbumsById(Collection<Album> albumsById) {  
 this.albumsById = albumsById;  
 }  
  
 @Override  
 public String toString() {  
 return "Performer{" +  
 "id=" + id +  
 ", name='" + name + '\'' +  
 ", shortInfo='" + shortInfo + '\'' +  
 ", auditionRate=" + auditionRate +  
 ", born='" + born + '\'' +  
 '}';  
 }  
}

package entity\_try;  
  
import jakarta.persistence.\*;  
  
import java.util.Objects;  
  
@Entity  
@Table(name = "playlist", schema = "todo", catalog = "music\_shop")  
public class Playlist {  
 @GeneratedValue(strategy = GenerationType.*IDENTITY*)  
 @Id  
 @Column(name = "id", nullable = false)  
 private int id;  
 @Basic  
 @Column(name = "name", nullable = false, length = 32)  
 private String name;  
 @Basic  
 @Column(name = "short info", nullable = true, length = 64)  
 private String shortInfo;  
 @Basic  
 @Column(name = "creator", nullable = false, length = 16)  
 private String creator;  
 @Basic  
 @Column(name = "audition rate", nullable = false)  
 private long auditionRate;  
  
  
 public int getId() {  
 return id;  
 }  
  
 public void setId(int id) {  
 this.id = id;  
 }  
  
 public String getName() {  
 return name;  
 }  
  
 public void setName(String name) {  
 this.name = name;  
 }  
  
 public String getShortInfo() {  
 return shortInfo;  
 }  
  
 public void setShortInfo(String shortInfo) {  
 this.shortInfo = shortInfo;  
 }  
  
 public String getCreator() {  
 return creator;  
 }  
  
 public void setCreator(String creator) {  
 this.creator = creator;  
 }  
  
 public long getAuditionRate() {  
 return auditionRate;  
 }  
  
 public void setAuditionRate(long auditionRate) {  
 this.auditionRate = auditionRate;  
 }  
  
 @Override  
 public boolean equals(Object o) {  
 if (this == o) return true;  
 if (o == null || getClass() != o.getClass()) return false;  
 Playlist playlist = (Playlist) o;  
 return id == playlist.id && auditionRate == playlist.auditionRate && Objects.*equals*(name, playlist.name) && Objects.*equals*(shortInfo, playlist.shortInfo) && Objects.*equals*(creator, playlist.creator);  
 }  
  
 @Override  
 public int hashCode() {  
 return Objects.*hash*(id, name, shortInfo, creator, auditionRate);  
 }  
  
  
}

package entity\_try;  
  
import jakarta.persistence.\*;  
  
import java.util.Objects;  
  
@Entity  
@Table(name = "userinfo", schema = "todo", catalog = "music\_shop")  
public class Userinfo {  
 @GeneratedValue(strategy = GenerationType.*IDENTITY*)  
 @Id  
 private int login;  
 @Basic  
 @Column(name = "name", nullable = true, length = 16)  
 private String name;  
 @Basic  
 @Column(name = "surname", nullable = true, length = 32)  
 private String surname;  
 @Basic  
 @Column(name = "password", nullable = false, length = 32)  
 private String password;  
 @Basic  
 @Column(name = "email", nullable = true, length = 64)  
 private String email;  
 @Basic  
 @Column(name = "phone number", nullable = true, length = 16)  
 private String phoneNumber;  
 @Basic  
 @Column(name = "subscribe status", nullable = true, length = 16)  
 private String subscribeStatus;  
  
  
 public int getLogin() {  
 return login;  
 }  
  
 public void setLogin(int login) {  
 this.login = login;  
 }  
  
 public String getName() {  
 return name;  
 }  
  
 public void setName(String name) {  
 this.name = name;  
 }  
  
 public String getSurname() {  
 return surname;  
 }  
  
 public void setSurname(String surname) {  
 this.surname = surname;  
 }  
  
 public String getPassword() {  
 return password;  
 }  
  
 public void setPassword(String password) {  
 this.password = password;  
 }  
  
 public String getEmail() {  
 return email;  
 }  
  
 public void setEmail(String email) {  
 this.email = email;  
 }  
  
 public String getPhoneNumber() {  
 return phoneNumber;  
 }  
  
 public void setPhoneNumber(String phoneNumber) {  
 this.phoneNumber = phoneNumber;  
 }  
  
 public String getSubscribeStatus() {  
 return subscribeStatus;  
 }  
  
 public void setSubscribeStatus(String subscribeStatus) {  
 this.subscribeStatus = subscribeStatus;  
 }  
  
 @Override  
 public boolean equals(Object o) {  
 if (this == o) return true;  
 if (o == null || getClass() != o.getClass()) return false;  
 Userinfo userinfo = (Userinfo) o;  
 return login == userinfo.login && Objects.*equals*(name, userinfo.name) && Objects.*equals*(surname, userinfo.surname) && Objects.*equals*(password, userinfo.password) && Objects.*equals*(email, userinfo.email) && Objects.*equals*(phoneNumber, userinfo.phoneNumber) && Objects.*equals*(subscribeStatus, userinfo.subscribeStatus);  
 }  
  
 @Override  
 public int hashCode() {  
 return Objects.*hash*(login, name, surname, password, email, phoneNumber, subscribeStatus);  
 }  
  
  
}

package utils;  
  
import jakarta.persistence.EntityManager;  
import org.hibernate.Session;  
  
public class AbstractJPA extends HibernateUtil {  
 protected EntityManager entityManager;  
  
  
 Session session = HibernateUtil.*getSessionFactory*().openSession();  
 public void initOne(){  
 session.getTransaction().begin();  
 }  
 public void commitOne(){  
 session.getTransaction().commit();  
 }  
  
 public void closeA(){  
 session.close();  
 HibernateUtil.*close*();  
 }  
}

package utils;  
  
import jakarta.persistence.EntityManager;  
import jakarta.persistence.EntityManagerFactory;  
import jakarta.persistence.Persistence;  
  
public class AbstractJPAPers {  
 protected EntityManager entityManager;  
  
 public void init(){  
 EntityManagerFactory entityManagerFactory = Persistence.*createEntityManagerFactory*("Persistence");  
 entityManager = entityManagerFactory.createEntityManager();  
 entityManager.getTransaction().begin();  
 }  
  
 public void close(){  
 if(entityManager.getTransaction().isActive()){  
 entityManager.getTransaction().commit();  
 }  
 entityManager.getEntityManagerFactory().close();  
 entityManager.close();  
 }  
}

package utils;  
  
import org.hibernate.SessionFactory;  
import org.hibernate.cfg.Configuration;  
  
import java.io.File;  
  
public class HibernateUtil {  
 protected static final SessionFactory *sessionFactory* = *initSessionFactory*();  
  
 protected static SessionFactory initSessionFactory(){  
 try {  
 return new Configuration().configure(new File("src\\main\\resources\\hibernate.cfg.xml")).buildSessionFactory();  
 }  
 catch(Throwable ex){  
 System.*err*.println(("Initial SessionFactory creation failed." + ex));  
 throw new ExceptionInInitializerError(ex);  
 }  
 }  
 public static SessionFactory getSessionFactory(){  
 if(*sessionFactory* == null){  
 *initSessionFactory*();  
 }  
 return *sessionFactory*;}  
  
 public static void close(){  
 *getSessionFactory*().close();  
 }  
  
}

import doing.AddMusicalTrack;  
import doing.DeletePlaylist;  
import doing.SearchQuerry;  
import doing.UpdateUserInfo;  
import entity\_try.Album;  
import entity\_try.MusicalTrack;  
import entity\_try.Performer;  
  
import java.util.List;  
import java.util.Scanner;  
  
public class Main {  
 public static void main(String[] args) throws Exception {  
 Scanner scanner = new Scanner(System.*in*);  
  
 while(true) {  
 System.*out*.println();  
 System.*out*.println("Kakoe действие вы хотите произвести?\n1)Добавить музыкальный трек\n2)Удалить плейлист по id пользователя" +  
 "\n3)Изменить данные пользователя с указанным id\n4)Вывести треки с временем больше указанного\n5)Вывести альбомы указанного исполнителя\n6)Вывести треки указанного исполнителя\n7)Вывести исполнителя указанного трека");  
 int change =Integer.*parseInt*(scanner.nextLine());  
 if(change == 1){  
 AddMusicalTrack addMusicalTrack = new AddMusicalTrack();  
 addMusicalTrack.add();  
 }  
 else if(change == 2){  
 DeletePlaylist deletePlaylist = new DeletePlaylist();  
 deletePlaylist.deleteByCreator();  
  
 }  
 else if(change == 3){  
 UpdateUserInfo updateUserInfo = new UpdateUserInfo();  
 updateUserInfo.newInfo(updateUserInfo.searchLogin());  
 }  
 else if(change == 4){  
 System.*out*.println("Введите время:");  
 int time = Integer.*parseInt*(scanner.nextLine());  
 List<MusicalTrack> musTracksTime = SearchQuerry.*trackTime*(time);  
 for(MusicalTrack i:musTracksTime){  
 System.*out*.println("Название:" + i.getName() + "-->" + i.getTime() + "сек");  
 }  
 }  
 else if(change == 5){  
 System.*out*.println("Введите альбомы какого исполнителя вы хотите узнать:");  
 String name = scanner.nextLine();  
 List<Album> albumSearch = SearchQuerry.*performerAs*(name);  
 for(Album i: albumSearch){  
 System.*out*.println("Исполнитель " + name + " выпустил альбом " + i.getName());  
 }  
 }  
 else if(change == 6){  
 System.*out*.println("Введите песни какого исполнителя вы хотите узнать:");  
 String name = scanner.nextLine();  
 List<MusicalTrack> musicTrackSearch = SearchQuerry.*musicalTrackAs*(name);  
 for(MusicalTrack i: musicTrackSearch){  
 System.*out*.println("Исполнитель " + name + " выпустил трек " + i.getName());  
 }  
  
 }  
 else if(change == 7){  
 System.*out*.println("Введите исполнителя какой песни вы хотите узнать:");  
 String name = scanner.nextLine();  
 List<Performer> performerTrack = SearchQuerry.*musicalTrackBy*(name);  
 for(Performer i: performerTrack){  
 System.*out*.println("Песня " + name + " принадлежит исполнителю " + i.getName());  
 }  
  
 }  
 else break;  
 }  
  
  
  
  
  
  
  
  
  
  
  
 }  
}

*<?*xml version="1.0" encoding="UTF-8" standalone="yes"*?>*<persistence xmlns="https://jakarta.ee/xml/ns/persistence"  
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
 xsi:schemaLocation="https://jakarta.ee/xml/ns/persistence https://jakarta.ee/xml/ns/persistence/persistence\_3\_0.xsd"  
 version="3.0">  
 <persistence-unit name="Persistence">  
  
 <class>entity\_try.Album</class>  
 <class>entity\_try.MusicalTrack</class>  
 <class>entity\_try.Performer</class>  
 <class>entity\_try.Playlist</class>  
 <class>entity\_try.Userinfo</class>  
  
 <properties>  
 <property name="hibernate.connection.url" value="jdbc:postgresql://localhost:5432/music\_shop"/>  
 <property name="hibernate.connection.driver\_class" value="org.postgresql.Driver"/>  
 <property name="hibernate.connection.username" value="postgres"/>  
 <property name="hibernate.connection.password" value="1905"/>  
 <property name="hibernate.show\_sql" value="true"/>  
 </properties>  
 </persistence-unit>  
</persistence>

*<?*xml version='1.0' encoding='utf-8'*?>*<!DOCTYPE hibernate-configuration PUBLIC  
 "-//Hibernate/Hibernate Configuration DTD//EN"  
 "http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd"*>*<hibernate-configuration>  
 <session-factory>  
 <property name="connection.url">jdbc:postgresql://localhost:5432/music\_shop</property>  
 <property name="connection.driver\_class">org.postgresql.Driver</property>  
 <property name="hibernate.dialect">org.hibernate.dialect.PostgreSQL10Dialect</property>  
 <property name="connection.username">postgres</property>  
 <property name="connection.password">1905</property>  
 <property name="hibernate.show\_sql">true</property>  
  
  
  
  
 <mapping class="entity\_try.Performer"/>  
 <mapping class="entity\_try.MusicalTrack"/>  
 <mapping class="entity\_try.Album"/>  
 <mapping class="entity\_try.Playlist"/>  
 <mapping class="entity\_try.Userinfo"/>  
  
  
 </session-factory>  
</hibernate-configuration>