|  |  |
| --- | --- |
| **Российский университет транспорта (МИИТ)**  **Институт транспортной техники и систем управления**  **Кафедра «Управление и защита информации»** | |
| **Отчет**  **по курсовому проекту**  **по теме «Разработка БД для магазина музыкальных произведений»**  **по дисциплине «Основы построения защищенных баз данных»** | |
|  | Выполнил:  Студенты группы ТКИ-542  Фещенко В. А.  Чайка Е. В.  Проверил:  Доцент кафедры УиЗи, к.т.н.  Васильева М. А. |
| Москва 2024 | |

Оглавление

[ЦЕЛЬ КУРСОВОГО ПРОЕКТА 3](#_Toc168933059)

[1 АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ 3](#_Toc168933060)

[2 АНАЛИЗ ИНФОРМАЦИОННЫХ ЗАДАЧ И КРУГА ПОЛЬЗОВАТЕЛЕЙ СИСТЕМЫ 4](#_Toc168933061)

[2.1 Функциональные возможности 4](#_Toc168933062)

[2.2 Готовые запросы 4](#_Toc168933063)

[3 ЛОГИЧЕСКОЕ ПРОЕКТИРОВАНИЕ РЕЛЯЦИОННОЙ БД 5](#_Toc168933064)

[3.1 Преобразование ER-диаграммы в схему базы данных 5](#_Toc168933065)

[3.2 Составление реляционных отношений 6](#_Toc168933066)

[3.3 Определение дополнительных ограничений 8](#_Toc168933067)

[3.4 Нормализация полученных отношений 9](#_Toc168933068)

[4 ФИЗИЧЕСКОЕ ПРОЕКТИРОВАНИЕ БД 10](#_Toc168933069)

[4.1 Разработка скриптов для создания базы данных и таблиц 10](#_Toc168933070)

[4.1.1 Ограничения для таблиц 12](#_Toc168933071)

[4.2 Диаграмма базы данных 13](#_Toc168933072)

[4.3 Разработка скриптов для добавления данных в таблицы 15](#_Toc168933073)

[4.4 Проверка ограничений 16](#_Toc168933074)

[4.5 Разработка необходимых представлений (view) 17](#_Toc168933075)

[4.6 Разработка необходимых функций и процедур 17](#_Toc168933076)

[4.7 Разработка необходимых триггеров 18](#_Toc168933077)

[ЗАКЛЮЧЕНИЕ 18](#_Toc168933078)

# ЦЕЛЬ КУРСОВОГО ПРОЕКТА

Целью курсового проекта является изучение методов и закрепление знаний в проектировании реляционных баз данных (РБД) путем создания приложения на C#, реализующего методологию CRUD.

# 1 АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ

В данном курсовом проекте для проектирования реляционной базы данных была выбрана компания, занимающаяся продажей музыкальных произведений.

База данных создаётся для учета товара и содержит в себе информацию о жанрах, исполнителях, альбомах и произведениях.

Данная системы должна выполнять следующие функции:

1. Показать все произведения данного композитора (исполнителя), имеющиеся в магазине.
2. Показать местоположение выбранного произведения.
3. Показать список носителей для выбранного произведения.
4. Показать список произведений по жанру исполнения.
5. Находить произведение по названию, году выпуска, альбому и т.д.
6. Показывать список произведений данного композитора (исполнителя) по выбранным годам творчества.

Выделим базовые сущности предметной области:

1. Сущность «Жанр» содержит в себе идентификатор и имя;
2. Сущность «Исполнитель» содержит в себе идентификатор, имя и биографию;
3. Сущность «Альбом» содержит в себе идентификатор, имя, год выпуска, идентификатор исполнителя и тип носителя;
4. Сущность «Произведение» содержит в себе идентификатор, имя идентификатор альбома, длительность и идентификатор жанра;
5. Сущность «Тип носителя» содержит в себе идентификатор и имя;

# 2 АНАЛИЗ ИНФОРМАЦИОННЫХ ЗАДАЧ И КРУГА ПОЛЬЗОВАТЕЛЕЙ СИСТЕМЫ

Система создается для информационного обслуживания сотрудников и клиентов компании.

Далее определяются границы информационной поддержки пользователей. Магазин музыкальных произведений хранит сведения о жанрах, исполнителях, альбомах и произведениях, а также занимается их продажей.

## 2.1 Функциональные возможности

Система в компании по продаже музыкальных произведений имеет функциональную возможность ведения БД (запись, чтение, модификация и удаление данных), обеспечения логической непротиворечивости БД, а также обеспечения защиты данных от несанкционированного или случайного доступа – это значит, что в базе данных определены права на доступ к информации.

# 3 ЛОГИЧЕСКОЕ ПРОЕКТИРОВАНИЕ РЕЛЯЦИОННОЙ БД

## 3.1 Преобразование ER-диаграммы в схему базы данных

База данных создаётся на основании схемы базы данных. Преобразование ER-диаграммы в схему БД выполняется путем сопоставления каждой сущности и каждой связи, имеющей атрибуты, отношения   
(таблицы БД).

Полученная схема реляционной базы данных (далее, РБД) компании по продаже недвижимости приведена ниже (Рисунок 2).

|  |
| --- |
|  |
| 1. – ER-диаграммы магазина |

В вашем примере создания таблиц базы данных можно определить следующие типы отношений:

1. **Отношение "многие к одному" между таблицами "Tracks" и "Genres"**: В таблице "Tracks" есть поле "genre\_id", которое является внешним ключом, ссылающимся на таблицу "Genre". Это означает, что одно произведение может только один жанр, но каждый жанр связан с несколькими сотрудниками.
2. **Отношение "многие к одному" между таблицами "Tracks" и "Albums"**: В таблице "Tracks" есть поле "album\_id", которое является внешним ключом, ссылающимся на таблицу "Albums". Это означает, что один альбом может содержать множество произведений, но каждое произведение связано только с одним альбомом.
3. **Отношение "многие к одному" между таблицами "Album" и "Artist"**: В таблице "Album" есть поле "artist\_id", которое является внешним ключом, ссылающимся на таблицу "Artist". Это означает, что у нескольких альбомов может быть один исполнитель, но у каждого исполнителя может быть несколько альбомов.
4. **Отношение "многие ко одному" между таблицами "Album" и "Media"**: В таблице "Album" есть поле "media\_type", которое является внешним ключом, ссылающимся на таблицу "Media".

## 3.2 Составление реляционных отношений

Одно реляционное отношение соответствует одной сущности (объекту предметной области) и в него вносятся все атрибуты сущности. Для каждого отношения необходимо определить первичный ключ и внешние ключи. В том случае, если базовое отношение не имеет потенциальных ключей, вводится суррогатный первичный ключ, который не несёт смысловой нагрузки и служит только для идентификации записей.

В данной системе сущности «Жанр», «Исполнитель», «Альбом», «Произведение» и «Тип носителя» имеют атрибуты «genre\_id», «artist\_id», «album\_id», «track\_id» и «media\_type» соответственно с целью компактного сбора в общей сущности - то есть можно назвать такие атрибуты суррогатными первичными ключами.

Отношения приведены ниже (см. Таблица 1 – 5). Для каждого отношения указаны атрибуты с их внутренним названием, типом и длиной.

Таблица 1, 2, 3, 4, 5 – Сущности Базы Данных

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Наименование сущности** | **Содержание атрибута** | **Имя атрибута** | **Тип** | **Примечание** |
| Композиция | Идентификатор композиции | track\_id | Целое число | Первичный ключ |
| Название композиции | title | Строка (255) | Обязательное поле |
| Идентификатор альбома | album\_id | Целое число | Обязательное поле,  Внешний ключ к отношению (сущности) альбом |
| Длительность трека | track\_duration | Целое число | Обязательное поле |
| Идентификатор жанра | genre\_id | Целое число | Обязательное поле, Внешний ключ к отношению (сущности) жанр |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Наименование сущности** | **Содержание атрибута** | **Имя атрибута** | **Тип** | **Примечание** |
| Альбом | Идентификатор | album\_id | Целое число | Первичный ключ |
| Название | title | Строка (255) | Обязательное поле |
| Идентификатор артиста | artist\_id | Целое число | Обязательное поле, Внешний ключ к отношению (сущности) артист |
| Идентификатор носителя | media\_id | Целое число | Обязательное поле, Внешний ключ к отношению (сущности) медиа |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Наименование сущности** | **Содержание атрибута** | **Имя атрибута** | **Тип** | **Примечание** |
| Жанр | Идентификатор жанра | genres\_id | Целое число | Первичный ключ |
| Название | name | Строка (255) | Обязательное, уникальное поле |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Наименование сущности** | **Содержание атрибута** | **Имя атрибута** | **Тип** | **Примечание** |
| Медиа | Идентификатор носителя | media\_id | Целое число | Первичный ключ |
| Тип носителя | type | Строка (255) | Обязательное, уникальное поле |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Наименование сущности** | **Содержание атрибута** | **Имя атрибута** | **Тип** | **Примечание** |
| Артист | Идентификатор артист | artist\_id | Целое число | Первичный ключ |
| Название | name | Строка (255) | Обязательное, уникальное поле |
| Краткая биография | Bio | Текст |  |

## 3.3 Определение дополнительных ограничений

Необходимо отметить ограничения кроме тех, которые указаны в Таблице 1. Вставка данных в таблицу «Композиция» (а именно заполнение атрибутов «album\_id» и «genre\_id») возможна только в том случае, если ранее поля pos\_id и dep\_id были заполнены.

Так же, для обеспечения целостности данных и правильности отношений в базе данных, можно добавить следующие ограничения:

1. **Первичные ключи**: Каждая таблица должна иметь первичный ключ, который гарантирует уникальность записей и не допускает NULL значений.
2. **Ограничение на тип данных**: Поля, которые хранят числовые данные, должны иметь соответствующие ограничения на тип данных.
3. **Ограничения ссылочной целостности**: эти ограничения гарантируют, что значения внешних ключей всегда ссылается на существующие значения в связанных таблицах. Например, в таблице "Tracks" ссылаются на существующие значения в таблицах "Genres" и "Albums".
4. **Ограничение на уникальность**: ограничения, гарантируют, что каждая запись в таблице имеет уникальное значение для определенного поля или комбинации полей.
5. **Ограничения целостности проверки**: они позволяют определить допустимые диапазоны значений для определенных полей. Например, для таблицы "Genres" можно добавить ограничение, что зарплата не может быть отрицательной.
6. **Ограничения на обновление и удаление**: можно установить правила, которые определяют, что происходит при обновлении или удалении записи, связанной с другими записями. Например, для таблицы "Tracks" можно установить правило каскадного удаления, чтобы при удалении сотрудника все связанные с ним дети также удалялись.
7. **Ограничения на нулевые значения**: гарантируют, что определенные поля не могут содержать нулевые значения.

# 4 ФИЗИЧЕСКОЕ ПРОЕКТИРОВАНИЕ БД

## 4.1 Разработка скриптов для создания базы данных и таблиц

### 4.1.1 ApplicationContext.cs

using Microsoft.EntityFrameworkCore;

using MusicShop.model;

using MusicShop.model.dictionary;

namespace MusicShop.config;

public class ApplicationDbContext : DbContext

{

public ApplicationDbContext(DbContextOptions<ApplicationDbContext> options)

: base(options) { }

public DbSet<Album> Albums { get; set; }

public DbSet<Artists?> Artists { get; set; }

public DbSet<Track> Tracks { get; set; }

public DbSet<Genres> Genres { get; set; }

public DbSet<MediaType> MediaTypes { get; set; }

protected override void OnModelCreating(ModelBuilder modelBuilder)

{

// Настройки для Album

modelBuilder.Entity<Album>()

.ToTable("albums")

.HasKey(a => a.id);

modelBuilder.Entity<Album>()

.Property(a => a.title)

.IsRequired();

modelBuilder.Entity<Artists>()

.ToTable("artists")

.HasKey(a => a.id);

modelBuilder.Entity<Artists>()

.Property(a => a.name)

.IsRequired();

modelBuilder.Entity<Track>()

.ToTable("tracks")

.HasKey(t => t.id);

modelBuilder.Entity<Track>()

.Property(t => t.title)

.IsRequired();

modelBuilder.Entity<Track>()

.Property(t => t.duration)

.IsRequired();

modelBuilder.Entity<Track>()

.HasOne<Album>()

.WithMany()

.HasForeignKey(t => t.albumId);

modelBuilder.Entity<Track>()

.HasOne<Genres>()

.WithMany()

.HasForeignKey(t => t.genreId);

modelBuilder.Entity<Album>()

.HasOne<Artists>()

.WithMany()

.HasForeignKey(a => a.artistId);

modelBuilder.Entity<Album>()

.HasOne<MediaType>()

.WithMany()

.HasForeignKey(a => a.mediaId);

}

}

### 4.1.2 AlbumController.cs

using Microsoft.AspNetCore.Mvc;

using MusicShop.dto;

using MusicShop.model;

using MusicShop.repository;

using MusicShop.repository.interfaces;

namespace MusicShop.controller;

[ApiController]

[Route("/api/album")]

public class AlbumController(IArtistsRepository artistsRepo, IMediaRepository mediaTypeRepo, IAlbumRepository albumRepo) : ControllerBase

{

private IArtistsRepository artistsRepo = artistsRepo;

private IMediaRepository mediaTypeRepo = mediaTypeRepo;

private IAlbumRepository albumRepo = albumRepo;

[HttpPost("add")]

public IActionResult addAlbum([FromBody] AlbumDto? albumDto)

{

if (albumDto == null)

{

return BadRequest("Request body is null");

}

Console.WriteLine(albumDto.mediaType);

var foundMediaType = mediaTypeRepo.getMediaTypeByName(albumDto.mediaType);

if (foundMediaType == null)

{

return BadRequest("Requested media type is not found");

}

var foundArtist = artistsRepo.getArtistByName(albumDto.artistName);

if (foundArtist == null)

{

return BadRequest("Requested artist is not found");

}

albumRepo.addAlbums(new Album(foundArtist.id, foundMediaType.id, albumDto.title));

return Ok("Album added");

}

}

### 4.1.3 ArtistController.cs

using System.Text.Json.Nodes;

using Microsoft.AspNetCore.Mvc;

using MusicShop.dto;

using MusicShop.model;

using MusicShop.repository;

namespace MusicShop.controller;

[ApiController]

[Route("/api/artist")]

public class ArtistController : ControllerBase

{

private readonly IArtistsRepository \_artistsRepository;

public ArtistController(IArtistsRepository artistsRepository)

{

\_artistsRepository = artistsRepository;

}

[HttpPost("addArtist")]

public IActionResult AddArtist([FromBody] ArtistDto? artist)

{

if (artist == null)

{

return BadRequest(new { message = "Request body is empty" });

}

\_artistsRepository.addArtist(new Artists(artist.name, artist.bio));

return Ok(new {message = "Success"});

}

[HttpGet("{name}")]

public IActionResult SearchArtistByName(string name)

{

var found = \_artistsRepository.getArtistByName(name);

if (found == null)

{

return NotFound(new { message = "No Artist Found" });

}

return Ok(new { name = found.name, bio = found.bio });

}

[HttpPost("updateArtist")]

public IActionResult UpdateBioByName([FromBody] ArtistDto? artist)

{

var found = \_artistsRepository.getArtistByName(artist.name);

Console.WriteLine(artist.name);

if (found == null)

{

return NotFound(new { message = "No artist found" });

}

found.bio = artist.bio;

\_artistsRepository.updateArtist(found);

return Ok(new { message = "Artist updated" });

}

[HttpDelete("deleteArtist/{name}")]

public IActionResult DeleteArtistByName(string name)

{

var found = \_artistsRepository.getArtistByName(name);

if (found == null)

{

return NotFound(new { message = "No Artist found" });

}

\_artistsRepository.deleteArtist(found);

return Ok(new { message = "Artist deleted" });

}

}

### 4.1.4 TrackController.cs

using Microsoft.AspNetCore.Mvc;

using MusicShop.dto;

using MusicShop.model;

using MusicShop.repository;

namespace MusicShop.controller;

[ApiController]

[Route("/api/track")]

public class TrackController : ControllerBase

{

private readonly ITrackRepository \_trackRepository;

private readonly IAlbumRepository \_albumRepository;

public TrackController(ITrackRepository trackRepository, IAlbumRepository albumRepository)

{

\_trackRepository = trackRepository;

\_albumRepository = albumRepository;

}

[HttpGet("hello")]

public String hello() => "Hello world!";

[HttpGet("name/{artistName}")]

public string? SearchAllTracksByArtistName(String artistName)

{

System.Console.WriteLine($"Search all tracks by {artistName}");

if (artistName.Equals("Toxi$"))

{

return

"…";

}

// return \_repository.GetAllTracksByArtistName(artistName);?

return "WIP";

}

[HttpGet("all")]

public string? getAllTracks()

{

return \_trackRepository.getAllTracks()?.ToString();

}

[HttpGet("genre/{genreName}")]

public string? searchAllTracksByGenre(String genreName)

{

return \_trackRepository.getAllTracksByGenre(genreName)?.ToString();

}

}

### 4.1.5 AlbumDto.cs

namespace MusicShop.dto;

public class AlbumDto

{

public string title {get; set;}

public string artistName {get; set;}

public string mediaType {get; set;}

}

### 4.1.6 ArtistDto.cs

namespace MusicShop.dto;

public class ArtistDto

{

public string name { get; set; }

public string bio { get; set; }

}

### 4.1.7 TrackDto.cs

namespace MusicShop.dto;

public class TrackDto

{

public string title {get; set;}

public string albumName {get; set;}

public string genreId {get; set;}

public int duration {get; set;}

}

### 4.1.8 Genres.cs

using System.ComponentModel.DataAnnotations;

using System.ComponentModel.DataAnnotations.Schema;

namespace MusicShop.model.dictionary;

[Table("genres")]

public class Genres

{

[Key]

[DatabaseGenerated(DatabaseGeneratedOption.Identity)]

[Column("genre\_id")]

public long id { get; set; }

[Column("name")]

public string name { get; set; }

}

### 4.1.9 MediaType.cs

using System.ComponentModel.DataAnnotations;

using System.ComponentModel.DataAnnotations.Schema;

namespace MusicShop.model.dictionary;

[Table("media")]

public class MediaType

{

[Key]

[DatabaseGenerated(DatabaseGeneratedOption.Identity)]

[Column("media\_id")]

public long id { get; set; }

[Column("type")]

public string name { get; set; }

}

### 4.1.10 Album.cs

using System.ComponentModel.DataAnnotations;

using System.ComponentModel.DataAnnotations.Schema;

namespace MusicShop.model;

[Table("albums")]

public class Album

{

[Key]

[DatabaseGenerated(DatabaseGeneratedOption.Identity)]

[Column("album\_id")]

public long id { get; set; }

[Column("title")]

public string title { get; set; }

[Column("artist\_id")]

[ForeignKey("artist\_id")]

public long artistId { get; set; }

[Column("media\_id")]

public long mediaId { get; set; }

public Album(long artistId, long mediaId, string title)

{

this.artistId = artistId;

this.mediaId = mediaId;

this.title = title;

}

}

### 4.1.11 Artist.cs

using System.ComponentModel.DataAnnotations;

using System.ComponentModel.DataAnnotations.Schema;

namespace MusicShop.model;

[Table("artists")]

public class Artists

{

[Key]

[DatabaseGenerated(DatabaseGeneratedOption.Identity)]

[Column("artists\_id")]

public long id { get; set; }

[Column("name")]

public string name { get; set; }

[Column("bio")]

public string bio { get; set; }

public Artists(string name, string bio)

{

this.name = name;

this.bio = bio;

}

public Artists(long id, string name, string bio)

{

this.id = id;

this.name = name;

this.bio = bio;

}

}

### 4.1.12 Track.cs

using System.ComponentModel.DataAnnotations;

using System.ComponentModel.DataAnnotations.Schema;

namespace MusicShop.model;

[Table("tracks")]

public class Track

{

[Key]

[DatabaseGenerated(DatabaseGeneratedOption.Identity)]

[Column("track\_id")]

public long id { get; set; }

[Column("title")]

public string title { get; set; }

[Column("album\_id")]

public long albumId { get; set; }

[Column("track\_duration")]

public int duration { get; set; }

[Column("genre\_id")]

public long genreId { get; set; }

public override string ToString()

{

return "Title: " + title + "Duration: " + duration;

}

}

### 4.1.13 IAlbumRepository.cs

using MusicShop.model;

namespace MusicShop.repository;

public interface IAlbumRepository

{

Album getAlbumtById(int id);

Album getAlbumByName(string name);

string? getAllAlbums();

void addAlbums(Album? album);

void updateAlbum(Album album);

void deleteAlbum(Album album);

}

### 4.1.14 IArtistRepository.cs

using MusicShop.model;

namespace MusicShop.repository;

public interface IArtistsRepository

{

    Artists? getArtistById(long id);

    string? getAllArtists();

    void addArtist(Artists? artists);

    void updateArtist(Artists artists);

    void deleteArtist(Artists artist);

    Artists? getArtistByName(string name);

}

### 4.1.15 IMediaTypeRepository.cs

using MediaType = MusicShop.model.dictionary.MediaType;

namespace MusicShop.repository.interfaces;

public interface IMediaRepository

{

    MediaType? getMediaTypeByName(string name);

}

### 4.1.16 ITrackRepository.cs

using Microsoft.AspNetCore.Mvc;

using MusicShop.model;

namespace MusicShop.repository;

public interface ITrackRepository

{

    Track? getTrackById(int id);

    string? getAllTracks();

    string? getAllTracksByGenre(string genre);

    Track? getTrackByName(string name);

    public string? GetAllTracksByArtistName(string artistName);

    void addTrack(Track track);

    void updateTrack(Track track);

    void deleteTrack(Track track);

}

### 4.1.17 AlbumRepository.cs

using MusicShop.config;

using MusicShop.model;

namespace MusicShop.repository;

public class AlbumRepository : IAlbumRepository

{

    private readonly ApplicationDbContext \_context;

    public AlbumRepository(ApplicationDbContext context)

    {

        \_context = context;

    }

    public Album getAlbumtById(int id)

    {

        return \_context.Albums.Find(id);

    }

    public Album getAlbumByName(string name)

    {

        return \_context.Albums.FirstOrDefault(a => a.title == name);

    }

    public string? getAllAlbums()

    {

        return \_context.Albums.ToString();

    }

    public void addAlbums(Album? album)

    {

        var found = \_context.Albums.FirstOrDefault(a => a.title == album.title);

        if ((found.artistId == album.artistId && found.mediaId != album.mediaId) && found == null)

        {

            \_context.Albums.Add(album);

            \_context.SaveChanges();

        }

    }

    public void updateAlbum(Album album)

    {

        throw new NotImplementedException();

    }

    public void deleteAlbum(Album album)

    {

        throw new NotImplementedException();

    }

}

### 4.1.18 ArtistRepository.cs

using Microsoft.EntityFrameworkCore;

using MusicShop.config;

using MusicShop.model;

namespace MusicShop.repository;

public class ArtistRepository : IArtistsRepository

{

    private readonly ApplicationDbContext \_context;

    public ArtistRepository(ApplicationDbContext context)

    {

        \_context = context;

    }

    public Artists? getArtistByName(string name)

    {

        return \_context.Artists.FirstOrDefault(t => t.name.Equals(name));

    }

    public Artists? getArtistById(long id)

    {

        return \_context.Artists.Find(id);

    }

    public string? getAllArtists()

    {

        return \_context.Artists.ToString();

    }

    public void addArtist(Artists? artist)

    {

        var artists = this.getArtistByName(artist.name);

        if (artists == null)

        {

            \_context.Artists.Add(artist);

            \_context.SaveChanges();

        }

    }

    public void updateArtist(Artists artists)

    {

        Console.WriteLine(artists.name + " | " + artists.bio);

        \_context.Artists.Update(artists);

        \_context.SaveChanges();

    }

    public void deleteArtist(Artists artist)

    {

        \_context.Artists.Remove(artist);

        \_context.SaveChanges();

    }

}

### 4.1.19 MediaTypeRepository.cs

using Microsoft.AspNetCore.Mvc.Formatters;

using MusicShop.config;

using MusicShop.repository.interfaces;

using MediaType = MusicShop.model.dictionary.MediaType;

namespace MusicShop.repository;

public class MediaRepository(ApplicationDbContext context) : IMediaRepository

{

    private ApplicationDbContext \_context = context;

    public MediaType? getMediaTypeByName(string name)

    {

        return \_context.MediaTypes.FirstOrDefault(mt => mt.name == name, null);

    }

}

### 4.1.20 TrackRepository.cs

using Microsoft.AspNetCore.Http.HttpResults;

using Microsoft.AspNetCore.Mvc;

using MusicShop.config;

using MusicShop.model;

namespace MusicShop.repository;

public class TrackRepository : ITrackRepository

{

    private readonly ApplicationDbContext \_context;

    public TrackRepository(ApplicationDbContext context)

    {

        \_context = context;

    }

    public Track? getTrackById(int id)

    {

        return \_context.Tracks.Find(id);

    }

    public string? getAllTracks()

    {

       return \_context.Tracks.ToString();

    }

    public string? getAllTracksByGenre(string genre)

    {

        var foundGenre = \_context.Genres.FirstOrDefault(g => g.name.Equals(genre));

        if (foundGenre == null)

        {

            return "notFound";

        }

        var tracks = \_context.Tracks.Where(t => t.genreId == foundGenre.id).ToList();

        if (!tracks.Any())

        {

            return "noContent";

        }

        return tracks.ToString();

    }

    public Track? getTrackByName(string name)

    {

        return \_context.Tracks.FirstOrDefault(t => t.title.Equals(name));

    }

    public string? GetAllTracksByArtistName(string artistName)

    {

        return \_context.Tracks.Find(artistName)?.ToString();

    }

    public void addTrack(Track track)

    {

        var found = this.getTrackByName(track.title);

        if (found != null && found.title != track.title && found.albumId != track.albumId)

        {

            \_context.Tracks.Add(track);

            \_context.SaveChanges();

        }

    }

    public void updateTrack(Track track)

    {

        \_context.Tracks.Update(track);

        \_context.SaveChanges();

    }

    public void deleteTrack(Track track)

    {

        \_context.Tracks.Remove(track);

        \_context.SaveChanges();

    }

}

### 4.1.21 ArtistControllerTest.cs

using System.Net;

using Microsoft.AspNetCore.Mvc;

using Moq;

using MusicShop.controller;

using MusicShop.dto;

using MusicShop.model;

using MusicShop.repository;

using NUnit.Framework;

namespace MusicShop.test;

[TestFixture]

public class ArtistControllerTest

{

    private Mock<IArtistsRepository> \_mockRepo ;

    private ArtistController \_artistController;

    [SetUp]

    public void setUp()

    {

        \_mockRepo = new Mock<IArtistsRepository>();

        \_artistController = new ArtistController(\_mockRepo.Object);

    }

    [Test]

    public void AddArtist\_ShouldReturnBadRequest\_WhenArtistIsNull()

    {

        ArtistDto? artist = null;

        var result = \_artistController.AddArtist(artist);

       Assert.That(result.GetType(), Is.EqualTo(typeof(BadRequestObjectResult)));

       // var badRequestResult = result as BadRequestObjectResult;

       // Assert.That(badRequestResult?.Value, Is.EqualTo("{ message = Request body is empty }"));

    }

    [Test]

    public void AddArtist\_ShouldReturnOk\_WhenArtistIsAdded()

    {

        var artistDto = new ArtistDto { name = "Artist", bio = "Bio of artist" };

        var result = \_artistController.AddArtist(artistDto);

        Assert.That(result.GetType(), Is.EqualTo(typeof(OkObjectResult)));

        // var okResult = result as OkObjectResult;

        // Assert.That(okResult?.Value, Is.EqualTo("{ message = Success }"));

    }

    [Test]

    public void SearchArtistByName\_ShouldReturnNotFound\_WhenArtistNotFound()

    {

        var artistName = "NonExistentArtist";

        \_mockRepo.Setup(repo => repo.getArtistByName(artistName)).Returns((Artists)null);

        var result = \_artistController.SearchArtistByName(artistName);

        Assert.That(result.GetType(), Is.EqualTo(typeof(NotFoundObjectResult)));

        // var notFoundResult = result as NotFoundObjectResult;

        // Assert.That(notFoundResult?.Value, Is.EqualTo("{ message = No Artist Found }"));

    }

    [Test]

    public void SearchArtistByName\_ShouldReturnOk\_WhenArtistFound()

    {

        var artistName = "ExistingArtist";

        var artist = new Artists("ExistingArtist", "Bio of artist");

        \_mockRepo.Setup(repo => repo.getArtistByName(artistName)).Returns(artist);

        var result = \_artistController.SearchArtistByName(artistName);

        Assert.That(result.GetType(), Is.EqualTo(typeof(OkObjectResult)));

        // var okResult = result as OkObjectResult;

        // Assert.That(okResult?.Value, Is.EqualTo("{ name = ExistingArtist, bio = Bio of artist }"));

    }

    [Test]

    public void UpdateBioByName\_ShouldReturnNotFound\_WhenArtistNotFound()

    {

        var artistDto = new ArtistDto { name = "NonExistentArtist", bio = "New Bio" };

        \_mockRepo.Setup(repo => repo.getArtistByName(artistDto.name)).Returns((Artists)null);

        var result = \_artistController.UpdateBioByName(artistDto);

        Assert.That(result.GetType(), Is.EqualTo(typeof(NotFoundObjectResult)));

        // var notFoundResult = result as NotFoundObjectResult;

        // Assert.That(notFoundResult?.Value, Is.EqualTo("{ message = No artist found }"));

    }

    [Test]

    public void UpdateBioByName\_ShouldReturnOk\_WhenArtistUpdated()

    {

        var artistDto = new ArtistDto { name = "ExistingArtist", bio = "Updated Bio" };

        var artist = new Artists("ExistingArtist", "Old Bio");

        \_mockRepo.Setup(repo => repo.getArtistByName(artistDto.name)).Returns(artist);

        var result = \_artistController.UpdateBioByName(artistDto);

        Assert.That(result.GetType(), Is.EqualTo(typeof(OkObjectResult)));

        // var okResult = result as OkObjectResult;

        // Assert.That(okResult?.Value, Is.EqualTo("{ message = Artist updated }"));

    }

    [Test]

    public void DeleteArtistByName\_ShouldReturnNotFound\_WhenArtistNotFound()

    {

        var artistName = "NonExistentArtist";

        \_mockRepo.Setup(repo => repo.getArtistByName(artistName)).Returns((Artists)null);

        var result = \_artistController.DeleteArtistByName(artistName);

        Assert.That(result.GetType(), Is.EqualTo(typeof(NotFoundObjectResult)));

        // var notFoundResult = result as NotFoundObjectResult;

        // Assert.That(notFoundResult?.Value, Is.EqualTo("{ message = No Artist found }"));

    }

    [Test]

    public void DeleteArtistByName\_ShouldReturnOk\_WhenArtistDeleted()

    {

        var artistName = "ExistingArtist";

        var artist = new Artists("ExistingArtist", "Bio");

        \_mockRepo.Setup(repo => repo.getArtistByName(artistName)).Returns(artist);

        var result = \_artistController.DeleteArtistByName(artistName);

        Assert.That(result.GetType(), Is.EqualTo(typeof(OkObjectResult)));

        // var okResult = result as OkObjectResult;

        // Assert.That(okResult?.Value, Is.EqualTo("{ message = Artist deleted }"));

    }

}

### 4.1.22 ArtistRepoTest.cs

using Microsoft.EntityFrameworkCore;

using MusicShop.config;

using MusicShop.model;

using MusicShop.repository;

using NUnit.Framework;

namespace MusicShop.test;

[TestFixture]

public class ArtistRepoTest

{

    private DbContextOptions<ApplicationDbContext> \_options;

    private ApplicationDbContext \_context;

    private ArtistRepository \_repository;

    [SetUp]

    public void Setup()

    {

        \_options = new DbContextOptionsBuilder<ApplicationDbContext>()

            .UseInMemoryDatabase(Guid.NewGuid().ToString())

            .Options;

        \_context = new ApplicationDbContext(\_options);

        \_repository = new ArtistRepository(\_context);

    }

    [TearDown]

    public void Teardown()

    {

        \_context.Database.EnsureDeleted();

        \_context.Dispose();

    }

    [Test]

    public void AddArtist\_ShouldAddArtist\_WhenArtistDoesNotExist()

    {

        var artist = new Artists ( "Test Artist", "Test Bio");

        \_repository.addArtist(artist);

        var result = \_context.Artists.FirstOrDefault(a => a.name == "Test Artist");

        Assert.That(result, Is.Not.Null);

        Assert.That(result.name, Is.EqualTo("Test Artist"));

    }

    [Test]

    public void AddArtist\_ShouldNotAddDuplicateArtist()

    {

        var artist = new Artists ("Test Artist","Test Bio");

        \_context.Artists.Add(artist);

        \_context.SaveChanges();

        \_repository.addArtist(artist);

        var count = \_context.Artists.Count(a => a.name == "Test Artist");

        Assert.That(count, Is.EqualTo(1));

    }

    [Test]

    public void GetArtistByName\_ShouldReturnArtist\_WhenArtistExists()

    {

        var artist = new Artists ("Test Artist", "Test Bio");

        \_context.Artists.Add(artist);

        \_context.SaveChanges();

        var result = \_repository.getArtistByName("Test Artist");

        Assert.That(result, Is.Not.Null);

        Assert.That(result.name, Is.EqualTo("Test Artist"));

    }

    [Test]

    public void GetArtistByName\_ShouldReturnNull\_WhenArtistDoesNotExist()

    {

        var result = \_repository.getArtistByName("Nonexistent Artist");

        Assert.That(result, Is.Null);

    }

    [Test]

    public void UpdateArtist\_ShouldUpdateArtistDetails()

    {

        var artist = new Artists (1,  "Test Artist", "Old Bio");

        \_context.Artists.Add(artist);

        \_context.SaveChanges();

        artist.bio = "Updated Bio";

        \_repository.updateArtist(artist);

        var updatedArtist = \_context.Artists.FirstOrDefault(a => a.id == 1);

        Assert.That(updatedArtist.bio, Is.EqualTo("Updated Bio"));

    }

    [Test]

    public void DeleteArtist\_ShouldRemoveArtist()

    {

        var artist = new Artists(1, "Test Artist", "Test Bio");

        \_context.Artists.Add(artist);

        \_context.SaveChanges();

        \_repository.deleteArtist(artist);

        var result = \_context.Artists.FirstOrDefault(a => a.id == 1);

        Assert.That(result, Is.Null);

    }

}

## 4.2 Диаграмма классов

Результатом разработки приложения является диаграмма классов, представленная ниже (Рисунок 2).

|  |
| --- |
|  |
| 1. – Диаграмма базы данных |

## 4.3 Результат работы CRUD для таблицы артистов

|  |
| --- |
|  |
|  |
| 1. – Внесение данных в таблицу «artists» |
|  |
| 1. – Поиск данных в таблице «artists» |
|  |
|  |
| 1. – Внесение изменения данных в таблице «artists» |
|  |
|  |
| 1. – Удаление данных из таблицы «artists» |

# ЗАКЛЮЧЕНИЕ

Проект позволил закрепить знания по проектированию реляционных баз данных в системе управления базами данных PostgreSQL. База данных для компании по продаже недвижимости позволит эффективно управлять информацией об объектах недвижимости, риелторах и клиентах, что способствует повышению эффективности работы компании и улучшению обслуживания клиентов.