| **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  Ефремов Д. Ю.  Волочинский И. О.  Проверил:  Доцент кафедры УиЗи, к.т.н.  Васильева М. А. |
| Москва 2024 | |

Оглавление

[ЦЕЛЬ КУРСОВОГО ПРОЕКТА 3](#_gjdgxs)

[1 АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ 3](#_30j0zll)

[2 АНАЛИЗ ИНФОРМАЦИОННЫХ ЗАДАЧ И КРУГА ПОЛЬЗОВАТЕЛЕЙ СИСТЕМЫ 4](#_1fob9te)

[2.1 Функциональные возможности 4](#_3znysh7)

[3 ЛОГИЧЕСКОЕ ПРОЕКТИРОВАНИЕ РЕЛЯЦИОННОЙ БД 4](#_2et92p0)

[3.1 Преобразование ER-диаграммы в схему базы данных 4](#_tyjcwt)

[3.2 Составление реляционных отношений 6](#_3dy6vkm)

[3.3 Определение дополнительных ограничений 9](#_1t3h5sf)

[4 ФИЗИЧЕСКОЕ ПРОЕКТИРОВАНИЕ БД 11](#_4d34og8)

[4.1 Разработка скриптов для создания базы данных и таблиц 11](#_2s8eyo1)

[4.1.1 ApplicationContext.cs 11](#_17dp8vu)

[**4.1.2 WorkerController.cs 12**](#)

[**4.1.3 WorkerDto.cs 14**](#)

[**4.1.4 Discipline.cs 14**](#)

[**4.1.5 Lesson.cs 15**](#)

[**4.1.6 Worker.cs 15**](#)

[**4.1.7 IDisciplineRepository.cs 16**](#)

[**4.1.8 ILessonRepository.cs 17**](#)

[**4.1.9 IWorkerRepository.cs 17**](#)

[4.1.10 WorkerRepoTest.cs 18](#_3rdcrjn)

[4.2 Диаграмма классов 22](#_26in1rg)

[4.3 Результат работы CRUD для таблицы артистов 23](#_pnujkwt1vijx)

[**ЗАКЛЮЧЕНИЕ 24**](#)

# 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 кафедра университета

База данных создаётся для учета работников кафедры и содержит в себе информацию о именах сотрудников, фамилии, отчестве, должности, дисциплинах и лекциях

Данная системы должна выполнять следующие функции:

1. Показать работника кафедры
2. Удалить работника кафедры
3. Создать работника кафедры
4. Отредактировать данные работника кафедры

Выделим базовые сущности предметной области:

1. Сущность «Сотрудник» содержит в себе идентификатор, имя, фамилию и отчество;
2. Сущность «Дисциплина» содержит в себе идентификатор, имя и идентификаторы сотрудника лектора и его помощника;
3. Сущность «Лекция» содержит в себе идентификатор, время начала, время конца, идентификатор дисциплины;
4. Сущность «Научная исследовательская работа» содержит в себе идентификатор, идентификатор выполняющего эту работу, идентификатор дисциплины;

# 2 АНАЛИЗ ИНФОРМАЦИОННЫХ ЗАДАЧ И КРУГА ПОЛЬЗОВАТЕЛЕЙ СИСТЕМЫ

Система создается для информационного обслуживания сотрудников кафедры.

Далее определяются границы информационной поддержки пользователей.

## 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 "Worker" и "Discipline"**: В таблице "Discipline" есть поле "professor\_id", которое является внешним ключом, ссылающимся на таблицу "Worker". Это означает, что одно один профессор может вести несколько дисциплин.
2. **Отношение "многие к одному" между таблицами "Worker" и "Discipline"**: В таблице "Discipline" есть поле "helper\_id", которое является внешним ключом, ссылающимся на таблицу "Worker". Это означает, что один человек может быть помощникам в разных дисциплинах.
3. **Отношение "многие к одному" между таблицами "Discipline" и "Lesson"**: В таблице "Lesson" есть поле "discipline\_id", которое является внешним ключом, ссылающимся на таблицу "Discipline". Это означает, что у одной дисциплины может быть множество лекций.
4. **Отношение "многие ко одному" между таблицами "Scientific\_research\_work" и "Discipline"**: В таблице "Scientific\_research\_work" есть поле "discipline\_id", которое является внешним ключом, ссылающимся на таблицу "Discipline". Это означает, что может быть множество научных исследовательских работ по одной дисциплине.
5. **Отношение "многие ко одному" между таблицами "Scientific\_research\_work" и "Worker"**: В таблице "Scientific\_research\_work" есть поле "implementer\_id", которое является внешним ключом, ссылающимся на таблицу "Worker". Это означает, что один человек может работать над несколькими научными исследовательскими работами

## 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 «Лекция», «Работник», «Дисциплина», «Научная исследовательская работа» имеют атрибуты «id», 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** |
| --- | --- | --- | --- | --- |
| Работник | Идентификатор сотрудника | id | Целое число | Первичный ключ |
| Имя сотрудника | name | Строка (255) | Обязательное поле |
| Фамилия сотрудника | lastname | Строка (255) | Обязательное поле, |
| Отчество сотрудника | surname | Строка (255) | Необязательное поле |
| Тип сотрудника (должность) | type | Строка (255) | Обязательное поле, |

| **Наименование сущности** | **Содержание атрибута** | **Имя атрибута** | **Тип** | **Примечание** |
| --- | --- | --- | --- | --- |
| Дисциплина | Идентификатор | id | Целое число | Первичный ключ |
| Название | name | Строка (255) | Обязательное поле |
| Идентификатор  профессора | professor\_id | Целое число | Обязательное поле, Внешний ключ к отношению (сущности) работник |
| Идентификатор помощника | hellper\_id | Целое число | Обязательное поле, Внешний ключ к отношению (сущности) работник |

| **Наименование сущности** | **Содержание атрибута** | **Имя атрибута** | **Тип** | **Примечание** |
| --- | --- | --- | --- | --- |
| Лекция | Идентификатор | id | Целое число | Первичный ключ |
| Время начала | name | Время | Обязательное поле |
| Идентификатор  дисциплины | discipline\_id | Целое число | Обязательное поле, Внешний ключ к отношению (сущности) дисциплина |
| Время конца | hellper\_id | Время | Обязательное поле |

| **Наименование сущности** | **Содержание атрибута** | **Имя атрибута** | **Тип** | **Примечание** |
| --- | --- | --- | --- | --- |
| Научная исследовательская работа | Идентификатор | id | Целое число | Первичный ключ |
| Идентификатор  работника | implementer\_id | Целое число | Обязательное поле,  Внешний ключ к отношению (сущности) работник |
| Идентификатор  дисциплины | discipline\_id | Целое число | Обязательное поле, Внешний ключ к отношению (сущности) дисциплина |
| Тема работы | theme | Строка (255) | Обязательное поле |

## 3.3 Определение дополнительных ограничений

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 "Scientific\_research\_work" ссылаются на существующие значения в таблицах "Worker" и "Discipline".
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 "Lesson" можно добавить ограничение, что конец лекции не может быть раньше ее начала.
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 "Discipline" можно установить правило каскадного удаления, чтобы при удалении дисциплины все связанные с ней лекции 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 Cathedra.model;

using Microsoft.EntityFrameworkCore;

namespace Cathedra.config;

public class ApplicationDbContext : DbContext

{

public ApplicationDbContext(DbContextOptions<ApplicationDbContext> options)

: base(options) { }

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

public DbSet<Lesson> Lessons { get; set; }

public DbSet<Discipline> Disciplines { get; set; }

protected override void OnModelCreating(ModelBuilder modelBuilder)

{

// Настройки для Album

modelBuilder.Entity<Worker>()

.ToTable("worker")

.HasKey(w => w.id);

modelBuilder.Entity<Worker>()

.Property(w => w.name)

.IsRequired();

modelBuilder.Entity<Lesson>()

.ToTable("lesson")

.HasKey(l => l.id);

modelBuilder.Entity<Lesson>()

.Property(l => l.disciplineId)

.IsRequired();

modelBuilder.Entity<Discipline>()

.ToTable("discipline")

.HasKey(d => d.id);

modelBuilder.Entity<Discipline>()

.Property(d => d.name)

.IsRequired();

modelBuilder.Entity<Discipline>()

.Property(d => d.name)

.IsRequired();

modelBuilder.Entity<Discipline>()

.HasOne<Worker>()

.WithMany()

.HasForeignKey(d => d.professorId);

modelBuilder.Entity<Discipline>()

.HasOne<Worker>()

.WithMany()

.HasForeignKey(d => d.helperId);

modelBuilder.Entity<Lesson>()

.HasOne<Discipline>()

.WithMany()

.HasForeignKey(d => d.disciplineId);

}

}

### 4.1.2 WorkerController.cs

using Cathedra.dto;

using Cathedra.model;

using Cathedra.repository;

using Microsoft.AspNetCore.Mvc;

namespace Cathedra.controller;

[ApiController]

[Route("/api/worker")]

public class WorkerController : ControllerBase

{

private readonly IWorkerRepository \_workerRepository;

public WorkerController(IWorkerRepository workerRepository)

{

\_workerRepository = workerRepository;

}

[HttpPost("create")]

public IActionResult addWorker([FromBody] WorkerDto? worker)

{

if (worker == null)

{

return BadRequest("Request body is empty");

}

\_workerRepository.addWorker(new Worker(worker.name, worker.lastname, worker.type));

return Ok(new {status = "Worker added"});

}

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

public IActionResult findWorker(string name)

{

var found = \_workerRepository.getWorkerByName(name);

return Ok(new {name = found.name, lastname = found.lastName, type = found.type});

}

[HttpPut("update")]

public IActionResult updateWorker([FromBody] WorkerDto? worker)

{

var workerForUpdate = \_workerRepository.getWorkerByNameAndLastName(worker.name, worker.lastname);

workerForUpdate.type = worker.type;

\_workerRepository.updateWorker(workerForUpdate);

return Ok(new {status = "Worker updated"});

}

[HttpDelete("delete")]

public IActionResult deleteWorker([FromBody] WorkerDto? worker)

{

\_workerRepository.deleteWorker(\_workerRepository.getWorkerByNameAndLastName(worker.name, worker.lastname));

return Ok(new {status = "Worker deleted"});

}

}

### 4.1.3 WorkerDto.cs

namespace Cathedra.dto;

public class WorkerDto

{

public String name { get; set; }

public String lastname { get; set; }

public String type {get; set;}

}

### 4.1.4 Discipline.cs

using System.ComponentModel.DataAnnotations;

using System.ComponentModel.DataAnnotations.Schema;

namespace Cathedra.model;

[Table("discipline")]

public class Discipline

{

[Key]

[DatabaseGenerated(DatabaseGeneratedOption.Identity)]

[Column("discipline\_id")]

public long id { get; set; }

[Column("name")]

public string name { get; set; }

[Column("professor\_id")]

public long professorId { get; set; }

[Column("helper\_id")]

public long helperId { get; set; }

}

### 4.1.5 Lesson.cs

using System.ComponentModel.DataAnnotations;

using System.ComponentModel.DataAnnotations.Schema;

namespace Cathedra.model;

[Table("lesson")]

public class Lesson

{

[Key]

[DatabaseGenerated(DatabaseGeneratedOption.Identity)]

[Column("lesson\_id")]

public long id { get; set; }

[Column("discipline\_id")]

public long disciplineId { get; set; }

[Column("start\_time")]

public DateTime startTime { get; set; }

public Lesson(long disciplineId, DateTime startTime)

{

this.disciplineId = disciplineId;

this.startTime = startTime;

}

public Lesson(long id, long disciplineId, DateTime startTime)

{

this.id = id;

this.disciplineId = disciplineId;

this.startTime = startTime;

}

}

### 4.1.6 Worker.cs

using System.ComponentModel.DataAnnotations;

using System.ComponentModel.DataAnnotations.Schema;

namespace Cathedra.model;

[Table("worker")]

public class Worker

{

[Key]

[DatabaseGenerated(DatabaseGeneratedOption.Identity)]

[Column("worker\_id")]

public long id { get; set; }

[Column("name")]

public string name { get; set; }

[Column("last\_name")]

public string lastName { get; set; }

[Column("type")]

public string type { get; set; }

public Worker(string name, string lastName, string type)

{

this.name = name;

this.lastName = lastName;

this.type = type;

}

}

### 4.1.7 IDisciplineRepository.cs

using Microsoft.AspNetCore.Mvc;

using Cathedra.model;

namespace Cathedra.repository;

public interface IDisciplineRepository

{

Discipline? getDisciplineById(int id);

string? getAllDisciplines();

Discipline? getDisciplineByName(string name);

void addDiscipline(Discipline discipline);

void updateDiscipline(Discipline discipline);

void deleteDiscipline(Discipline discipline);

}

### 4.1.8 ILessonRepository.cs

using Cathedra.model;

namespace Cathedra.repository;

public interface ILessonsRepository

{

Lesson? getLessonById(long id);

string? getAllLessons();

void addLesson(Lesson? lesson);

void updateLesson(Lesson lesson);

void deleteLesson(Lesson lesson);

Lesson? getLessonByName(long name);

}

### 4.1.9 IWorkerRepository.cs

using Cathedra.dto;

using Cathedra.model;

namespace Cathedra.repository;

public interface IWorkerRepository

{

Worker getWorkertById(int id);

Worker getWorkerByName(string name);

string? getAllWorkers();

void addWorker(Worker? worker);

void updateWorker(Worker worker);

void deleteWorker(Worker worker);

Worker? getWorkerByNameAndLastName(string name, string lastName);

}

### 4.1.10 WorkerRepoTest.cs

using Cathedra.config;

using Cathedra.model;

using Cathedra.repository;

using Microsoft.EntityFrameworkCore;

using NUnit.Framework;

namespace Cathedra.test;

[TestFixture]

public class WorkerRepoTest

{

private DbContextOptions<ApplicationDbContext> \_options;

private ApplicationDbContext \_context;

private WorkerRepository \_repository;

[SetUp]

public void Setup()

{

\_options = new DbContextOptionsBuilder<ApplicationDbContext>()

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

.Options;

\_context = new ApplicationDbContext(\_options);

\_repository = new WorkerRepository(\_context);

}

[TearDown]

public void Teardown()

{

\_context.Database.EnsureDeleted();

\_context.Dispose();

}

[Test]

public void AddWorker\_ShouldAddWorker\_WhenWorkerDoesNotExist()

{

var worker = new Worker ( "Test name", "Test lastname", "test type");

\_repository.addWorker(worker);

var result = \_context.Worker.FirstOrDefault(w => w.name == "Test name" && w.lastName == "Test lastname");

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

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

}

[Test]

public void AddWorker\_ShouldNotAddDuplicateWorker()

{

var worker = new Worker("Test name","Test lastname", "Test type");

\_context.Worker.Add(worker);

\_context.SaveChanges();

// \_repository.addWorker(worker);

var count = \_context.Worker.Count(a => a.name == "Test name" && a.lastName == "Test lastname");

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

}

[Test]

public void GetWorkerByName\_ShouldReturnWorker\_WhenWorkerExists()

{

var worker = new Worker("Test name", "Test lastname", "Test type");

\_context.Worker.Add(worker);

\_context.SaveChanges();

var result = \_repository.getWorkerByNameAndLastName("Test name", "Test lastname");

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

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

}

[Test]

public void GetWorkerByName\_ShouldReturnNull\_WhenWorkerDoesNotExist()

{

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

Assert.That(result, Is.Null);

}

[Test]

public void UpdateWorker\_ShouldUpdateArtistDetails()

{

var worker = new Worker("Test name", "Test lastname", "Test type");

\_context.Worker.Add(worker);

\_context.SaveChanges();

worker.type = "Updated type";

\_repository.updateWorker(worker);

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

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

}

[Test]

public void DeleteWorker\_ShouldRemoveWorker()

{

var worker = new Worker("Test name", "Test lastname", "Test type");

\_context.Worker.Add(worker);

\_context.SaveChanges();

\_repository.deleteWorker(worker);

var result = \_context.Worker.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 классов  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 кафедры в университете позволит эффективно управлять информацией о дисциплинах, что способствует повышению эффективности работы кафедры и улучшению организованности.