МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ

«ГОМЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ ИМЕНИ П. О. СУХОГО»

Факультет автоматизированных и информационных систем

Кафедра «Информатика»

Дисциплина «Избранные главы информатики»

ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ

«Разработка модели предметной области с использованием ENTITY FRAMEWORK»

Вариант № 17

Выполнил:

студент группы ИП-31, Соловьев Д.С.

Принял:

доцент Асенчик О.Д.

Гомель 2018

**1. Цель работы:**

Ознакомиться с возможностями ENTITY FRAMEWORK для моделирования предметной области, выборки, вставки, удаления и обновления данных объектов, связанных с таблицами базы данных заданной предметной области.

**2. Ход работы и результаты.**

Классы:

Тип комнаты (напр. Стандарт, Люкс и т.п.):

|  |
| --- |
| using System.Collections.Generic;  namespace lab1\_ef  {  public class RoomType  {  public int Id { get; set; }  public string Name { get; set; }  public virtual ICollection<Room> Rooms { get; set; }  public override string ToString()  {  return Name;  }  }  } |

Комната:

|  |
| --- |
| using System;  using System.Collections.Generic;  namespace lab1\_ef  {  public class Room  {  public int RoomId { get; set; }  public int? RoomTypeId { get; set; }  public string RoomNo { get; set; }  public int Capacity { get; set; }  public string Description { get; set; }  public double Cost { get; set; }  public DateTime CostDate { get; set; }  public virtual RoomType RoomType { get; set; }  public virtual ICollection<Client> Clients { get; set; }  }  } |

Услуга (напр. Уборка, Обед в номер и т.п.):

|  |
| --- |
| using System.Collections.Generic;  namespace lab1\_ef  {  public class ServiceType  {  public int ServiceTypeId { get; set; }  public string Name { get; set; }  public string Description { get; set; }  public double Cost { get; set; }  public ICollection<Service> Services { get; set; }  }  } |

Услуга (оказанная):

|  |
| --- |
| namespace lab1\_ef  {  public class Service  {  public int ServiceId { get; set; }  public int? ClientId { get; set; }  public int? EmployeeId { get; set; }  public int? ServiceTypeId { get; set; }  public virtual Client Client { get; set; }  public virtual Employee Employee { get; set; }  public virtual ServiceType ServiceType { get; set; }  }  } |

Сотрудник:

|  |
| --- |
| using System.Collections.Generic;  namespace lab1\_ef  {  public class Employee  {  public int EmployeeId { get; set; }  public string Name { get; set; }  public ICollection<Service> Services { get; set; }  }  } |

Клиент:

|  |
| --- |
| using System;  using System.Collections.Generic;  using System.Linq;  namespace lab1\_ef  {  public class Client  {  public int ClientId { get; set; }  public int? RoomId { get; set; }  public string Name { get; set; }  public string Passport { get; set; }  public DateTime OccupancyDate { get; set; }  public DateTime DepartureDate { get; set; }  public virtual Room Room { get; set; }  public ICollection<Service> Services { get; set; }  public ICollection<Employee> ServeEmployees { get  { return Services?.Where(s => s.Client == this).Select(s => s.Employee).ToList(); } }  }  } |

Контекст данных:

|  |
| --- |
| using Microsoft.EntityFrameworkCore;  namespace lab1\_ef  {  public class HotelContext : DbContext  {  public DbSet<Client> Clients { get; set; }  public DbSet<Employee> Employees { get; set; }  public DbSet<Room> Rooms { get; set; }  public DbSet<RoomType> RoomTypes { get; set; }  public DbSet<Service> Services { get; set; }  public DbSet<ServiceType> ServiceTypes { get; set; }  protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)  {  optionsBuilder.UseSqlServer  (@"Server=(localdb)\mssqllocaldb;Database=helloappdb;Trusted\_Connection=True;");  }  }  } |

Инициализатор:

|  |
| --- |
| using System;  using System.Linq;  namespace lab1\_ef  {  public static class Initializer  {  static string[] malenameusual = {"Александр", "Алексей", "Анатолий", "Андрей", "Антон", "Аркадий",  "Артём", "Богдан", "Борис", "Вадим", "Валентин", "Валерий", "Василий", "Виктор", "Виталий", "Владимир",  "Владислав", "Вячеслав", "Гавриил", "Геннадий", "Георгий", "Глеб", "Григорий", "Даниил", "Данила", "Денис",  "Дмитрий", "Евгений", "Егор", "Кирилл", "Иван", "Игорь", "Илья", "Иннокентий", "Лев", "Леонид", "Максим",  "Матвей", "Михаил", "Никита", "Николай", "Олег", "Павел", "Пётр", "Роман", "Ростислав", "Руслан", "Семён",  "Святослав", "Сергей", "Станислав", "Степан", "Тимофей", "Тимур", "Фёдор", "Филипп", "Эдуард", "Юрий", "Яков", "Ярослав" };  static string[] patronimusual = { "Александрович", "Алексеевич", "Анатольевич", "Андреевич", "Антонович",  "Аркадьевич", "Артёмович", "Богданович", "Борисович", "Вадимович", "Валентинович", "Валерьевич",  "Васильевич", "Викторович", "Витальевич", "Владимирович", "Владиславович", "Вячеславович", "Гавриилович",  "Геннадьевич", "Георгиевич", "Глебович", "Григорьевич", "Даниилович", "Данилович", "Денисович", "Дмитриевич",  "Евгеньевич", "Егорович", "Кириллович", "Иванович", "Игоревич", "Ильич", "Иннокентьевич", "Львович",  "Леонидович", "Максимович", "Матвеевич", "Михайлович", "Никитич", "Николаевич", "Олегович", "Павлович",  "Петрович", "Романович", "Ростиславович", "Русланович", "Семёнович", "Святославович", "Сергеевич", "Станиславович",  "Степанович", "Тимофеевич", "Тимурович", "Фёдорович", "Филиппович", "Эдуардович", "Юрьевич", "Яковлевич", "Ярославович" };  static string[] lastnameusual = { "Смирнов", "Иванов", "Кузнецов", "Новиков", "Морозов", "Петров", "Павлов",  "Семёнов", "Богданов", "Воробьёв", "Тарасов", "Белов", "Киселёв", "Макаров", "Андреев", "Ковалёв", "Ильин",  "Гусев", "Титов", "Кузьмин", "Кудрявцев", "Баранов", "Куликов", "Алексеев", "Степанов", "Яковлев", "Сорокин",  "Сергеев", "Романов", "Захаров", "Борисов", "Королёв", "Герасимов", "Пономарёв", "Григорьев", "Лазарев", "Ершов",  "Никитин", "Соболев", "Рябов", "Цветков", "Данилов", "Журавлёв", "Николаев", "Крылов", "Максимов", "Сидоров",  "Осипов", "Белоусов", "Федотов", "Дорофеев", "Егоров", "Матвеев", "Бобров", "Дмитриев", "Анисимов", "Антонов",  "Тимофеев", "Никифоров", "Веселов", "Филиппов", "Марков", "Большаков", "Суханов", "Миронов", "Ширяев", "Александров",  "Коновалов", "Шестаков", "Казаков", "Громов", "Фомин", "Давыдов", "Мельников", "Щербаков", "Блинов", "Колесников",  "Афанасьев", "Власов", "Исаков", "Тихонов", "Аксёнов", "Родионов", "Котов", "Зуев", "Панов", "Рыбаков", "Абрамов",  "Воронов", "Мухин", "Архипов", "Трофимов", "Горшков", "Овчинников", "Панфилов", "Копылов", "Лобанов", "Лукин", "Беляков",  "Потапов", "Некрасов", "Хохлов", "Жданов", "Наумов", "Шилов", "Воронцов", "Ермаков", "Дроздов", "Игнатьев", "Савин",  "Логинов", "Сафонов", "Капустин", "Кириллов", "Моисеев", "Елисеев", "Кошелев", "Костин", "Горбачёв", "Орехов",  "Ефремов", "Исаев", "Евдокимов", "Калашников", "Кабанов", "Носков", "Юдин", "Кулагин", "Лапин", "Прохоров", "Нестеров",  "Харитонов", "Агафонов", "Муравьёв", "Ларионов", "Федосеев", "Зимин", "Пахомов", "Шубин", "Игнатов", "Филатов", "Крюков",  "Рогов", "Кулаков", "Терентьев", "Молчанов", "Владимиров", "Артемьев", "Гурьев", "Зиновьев", "Гришин", "Кононов", "Дементьев",  "Ситников", "Симонов", "Мишин", "Фадеев", "Комиссаров", "Мамонтов", "Носов", "Гуляев", "Шаров", "Устинов", "Вишняков",  "Евсеев", "Лаврентьев", "Брагин", "Константинов", "Корнилов", "Авдеев", "Зыков", "Бирюков", "Шарапов", "Никонов", "Щукин",  "Дьячков", "Одинцов", "Сазонов", "Якушев", "Красильников", "Гордеев", "Самойлов", "Князев", "Беспалов", "Уваров", "Шашков",  "Бобылёв", "Доронин", "Белозёров", "Рожков", "Самсонов", "Мясников", "Лихачёв", "Буров", "Сысоев", "Фомичёв", "Русаков",  "Стрелков", "Гущин", "Тетерин", "Колобов", "Субботин", "Фокин", "Блохин", "Селиверстов", "Пестов", "Кондратьев", "Силин",  "Меркушев", "Лыткин", "Туров" };  public static void Initialize(HotelContext db)  {  db.Database.EnsureCreated();    // check only by this dbset  if (db.Clients.Any())  {  return;  }  int roomsCnt = 100;  int clientsCnt = 70; // must be less than rooms cnt  int employeesCnt = 100;  int servicesCnt = 100;  Random rand = new Random(DateTime.Now.Millisecond);  string[] roomTypes = { "Люкс", "Стандарт", "Эконом" };  string[] serviceTypes = { "Уборка", "Еда в номер", "Пополнение мини-бара" };  double[] sTypesCosts = { 70, 50, 30 }; // same size as for serviceTypes  string[] passSeries = { "AB", "BM", "HB", "KH", "MP", "MC", "KB", "PP", "ВМ" }; // + 7 digits  // firstly, fill constants  // fill room types  foreach (var rt in roomTypes)  db.RoomTypes.Add(new RoomType { Name = rt });  db.SaveChanges();  // fill serv types  for (int i = 0; i < roomTypes.Length; i++)  db.ServiceTypes.Add(new ServiceType { Name = serviceTypes[i],  Cost = sTypesCosts[i], Description = rand.Next() % 2 == 1 ? "SomeDescription" : null });  db.SaveChanges();  // fill rooms  var rTypes = db.RoomTypes.ToList();  for (int i = 0; i < roomsCnt; i++)  {  var value = new Room  {  RoomType = rTypes[rand.Next(roomTypes.Length)],  Capacity = 2 + rand.Next(3), // 2, 3, 4  Cost = 75 + rand.NextDouble() % 75,  RoomNo = $"{(i + 1)}",  CostDate = DateTime.Now - new TimeSpan(rand.Next() % 20, 0, 0, 0),  Description = rand.Next() % 2 == 1 ? "Room Description" : null  };  db.Rooms.Add(value);  }  db.SaveChanges();  // fill clients  var rooms = db.Rooms.ToList();  for (int i = 0; i < clientsCnt && i < rooms.Count; i++)  {  db.Clients.Add(new Client  {  Name = $"{lastnameusual[rand.Next(lastnameusual.Length)]} " +  $"{malenameusual[rand.Next(malenameusual.Length)]} " +  $"{patronimusual[rand.Next(patronimusual.Length)]}",  Passport = $"{passSeries[rand.Next() % passSeries.Length]}{rand.Next(10000000)}",  Room = rooms[i],  DepartureDate = DateTime.Now + new TimeSpan(2, 0, 0, 0),  OccupancyDate = DateTime.Now - new TimeSpan(7 \* rand.Next(3), 0, 0, 0)  });  }  db.SaveChanges();  // fill employees  for (int i = 0; i < employeesCnt; i++)  {  db.Employees.Add(new Employee  {  Name = $"{lastnameusual[rand.Next(lastnameusual.Length)]} " +  $"{malenameusual[rand.Next(malenameusual.Length)]} " +  $"{patronimusual[rand.Next(patronimusual.Length)]}"  });  }  db.SaveChanges();  // fill services  var sTypes = db.ServiceTypes.ToList();  var employees = db.Employees.ToList();  var clients = db.Clients.ToList();  for (int i = 0; i < servicesCnt; i++)  {  db.Services.Add(new Service  {  ServiceType = sTypes[rand.Next(db.ServiceTypes.Count())],  Employee = employees[rand.Next(db.Employees.Count())],  Client = clients[rand.Next(db.Clients.Count())]  });  }  db.SaveChanges();  }  }  } |

Программа:

|  |
| --- |
| using System;  using System.Collections.Generic;  using System.Linq;  namespace lab1\_ef  {  class Program  {  static void Main(string[] args)  {  HotelContext db = new HotelContext();  Initializer.Initialize(db);  bool stay = true;  while (stay)  {  Console.Clear();  Console.WriteLine("======================================");  Console.WriteLine("1 > Получить все виды услуг");  Console.WriteLine("2 > Получить все виды услуг + фильтр");  Console.WriteLine("3 > Получить комнаты + группировка по полю + вывод результата");  Console.WriteLine("4 > Выборка данных из двух полей двух таблиц, связанных отношением «один-ко-многим»{0}"+  "5 > Выборка данных из двух таблиц, связанных отношением «один-ко-многим» + фильтр{0}" +  "6 > Вставка данных в таблицы на стороне отношения «Один»{0}" +  "7 > Вставка данных в таблицы на стороне отношения «Многие»{0}" +  "8 > Удаление данных из таблицы на стороне отношения «Один»{0}" +  "9 > Удаление данных из таблицы на стороне отношения «Многие»{0}" +  "0 > Обновление удовлетворяющих условию записей в любой из таблиц базы данных{0}", Environment.NewLine);  Console.WriteLine("{0}ESCAPE > Выход", Environment.NewLine);  switch(Console.ReadKey(true).Key)  {  case ConsoleKey.D1:  Console.Clear();  OutputServiceTypes(db.ServiceTypes.ToList());  break;  case ConsoleKey.D2:  Console.Clear();  OutputServiceTypes(FilterServices(db.ServiceTypes).ToList());  break;  case ConsoleKey.D3:  Console.Clear();  OutputRooms(GroupRoomsByCapacity(db.Rooms));  break;  case ConsoleKey.D4:  Console.Clear();  JoinTwoTables(db.Services.ToList(), db.ServiceTypes.ToList());  break;  case ConsoleKey.D5:  Console.Clear();  Console.WriteLine("COST < 60 {0}", Environment.NewLine);  JoinFilterTwoTables(db.Services.ToList(), db.ServiceTypes.ToList());  break;  case ConsoleKey.D6:  Console.Clear();  OutputServiceTypes(db.ServiceTypes.ToList());  Console.WriteLine("{0}{0}", Environment.NewLine);  AddServiceType(db, new ServiceType { Name = $"TestType {DateTime.Now}", Cost = 500 });  Console.WriteLine("{0}{0}", Environment.NewLine);  OutputServiceTypes(db.ServiceTypes.ToList());  break;  case ConsoleKey.D7:  Console.Clear();  JoinTwoTables(db.Services.ToList(), db.ServiceTypes.ToList());  Console.WriteLine("{0}{0}", Environment.NewLine);  AddService(db, new Service { ServiceType = db.ServiceTypes.FirstOrDefault  (st => st.Name.Contains("TestType"))});  Console.WriteLine("{0}{0}", Environment.NewLine);  JoinTwoTables(db.Services.ToList(), db.ServiceTypes.ToList());  break;  case ConsoleKey.D8:  {  var value = db.ServiceTypes.FirstOrDefault(st => st.Name.Contains("TestType"));  if (value != null)  {  Console.Clear();  Console.WriteLine("{0}{0}", Environment.NewLine);  OutputServiceTypes(db.ServiceTypes.ToList());  DeleteServiceType(db, value);  Console.WriteLine("{0}{0}", Environment.NewLine);  OutputServiceTypes(db.ServiceTypes.ToList());  }  else  Console.WriteLine("Нет данных для удаления");  }  break;  case ConsoleKey.D9:  {  var value = db.Services.FirstOrDefault(st => st.ServiceType.Name.Contains("TestType"));  if (value != null)  {  Console.Clear();  Console.WriteLine("{0}{0}", Environment.NewLine);  JoinTwoTables(db.Services.ToList(), db.ServiceTypes.ToList());  DeleteService(db, value);  Console.WriteLine("{0}{0}", Environment.NewLine);  JoinTwoTables(db.Services.ToList(), db.ServiceTypes.ToList());  }  else  Console.WriteLine("Нет данных для удаления");  }  break;  case ConsoleKey.D0:  Console.Clear();  var client = new Client { Name = "TESTCLIENT", Passport = "AAA" };  Console.WriteLine($"{client.Name} ({client.Passport})");  AddClient(db, client);  EditClient(db, client);  Console.WriteLine($"{client.Name} ({client.Passport})");  break;  case ConsoleKey.Escape:  stay = false;  break;  default:  continue;  }  Console.ReadKey();  }  }  static void OutputServiceTypes(List<ServiceType> data)  {  foreach(var e in data)  Console.WriteLine($"{e.Name} [{e.Cost}] " +  $"{(string.IsNullOrWhiteSpace(e.Description) ? "" : $"{Environment.NewLine} {e.Description}")}");  }    static void OutputRooms(List<Room> data)  {  foreach (var e in data)  Console.WriteLine($"[{e.RoomNo}] {e.RoomType} ({e.Capacity} чел)");  }  static void OutputRooms(IQueryable<IGrouping<int, Room>> data)  {  foreach (var group in data)  {  Console.WriteLine("{0} {1} {0}", new string('\*', 15), group.Key);  OutputRooms(group.ToList());  }  }  static IQueryable<ServiceType> FilterServices(IQueryable<ServiceType> data)  {  bool stay = true;  IQueryable<ServiceType> result = data;    while (stay)  {  Console.WriteLine("Фильтр по:");  Console.WriteLine("1 > Description");  Console.WriteLine("2 > Cost");  Console.WriteLine();  Console.WriteLine("4 > Сброс");  Console.WriteLine("5 > Стоп");  switch (Console.ReadKey(true).Key)  {  case ConsoleKey.D1:  {  Console.WriteLine("1 > Пуст");  Console.WriteLine("2 > Не пуст");  var k = Console.ReadKey(true).Key;  if (k == ConsoleKey.D1)  result = result.Where(o => string.IsNullOrWhiteSpace(o.Description));  else if (k == ConsoleKey.D2)  result = result.Where(o => !string.IsNullOrWhiteSpace(o.Description));  else  Console.WriteLine("Выбран неверный пункт");  }  break;  case ConsoleKey.D2:  {  Console.WriteLine("1 > Меньше");  Console.WriteLine("2 > Больше");  var k = Console.ReadKey(true).Key;  if (k == ConsoleKey.D1 || k == ConsoleKey.D2)  {  Console.WriteLine("Значение: ");  if (double.TryParse(Console.ReadLine(), out double value))  {  if (k == ConsoleKey.D1)  result = result.Where(o => o.Cost <= value);  else  result = result.Where(o => o.Cost >= value);  }  else  Console.WriteLine("Ошибка при преобразовании значения");  }  else  Console.WriteLine("Выбран неверный пункт");  }  break;  case ConsoleKey.D4:  result = data;  break;  case ConsoleKey.D5:  stay = false;  break;  }  Console.Clear();  }  return result;  }  static IQueryable<IGrouping<int, Room>> GroupRoomsByCapacity(IQueryable<Room> data)  {  var value = from item in data  group item by item.Capacity;  return value;  }  static void JoinTwoTables(List<Service> services, List<ServiceType> serviceType)  {  var result = services.Join(serviceType,  p => p.ServiceTypeId, q => q.ServiceTypeId,  (p, q) => new  {  ServiceName = q.Name,  ClientName = p.Client?.Name,  p.Client?.Room?.RoomNo,  q.Cost  });  // вывод тут же - работать с анонимными типами между методами тот еще гемор  foreach (var item in result)  Console.WriteLine($"{item.ServiceName} для {item.ClientName}(комната #{item.RoomNo}) [{item.Cost}]");  }    static void JoinFilterTwoTables(List<Service> services, List<ServiceType> serviceType)  {  var result = services.Join(serviceType,  p => p.ServiceTypeId, q => q.ServiceTypeId,  (p, q) => new  {  ServiceName = q.Name,  ClientName = p.Client?.Name,  p.Client?.Room?.RoomNo,  q.Cost  });  result = result.Where(r => r.Cost < 60);  // вывод тут же - работать с анонимными типами между методами тот еще гемор  foreach (var item in result)  Console.WriteLine($"{item.ServiceName} для {item.ClientName}(комната #{item.RoomNo}) [{item.Cost}]");  }    static void AddServiceType(HotelContext db, ServiceType serviceType)  {  db.ServiceTypes.Add(serviceType);  db.SaveChanges();  }  static void AddService(HotelContext db, Service service)  {  db.Services.Add(service);  db.SaveChanges();  }  static void DeleteServiceType(HotelContext db, ServiceType serviceType)  {  var value = db.ServiceTypes.Where(st => st.ServiceTypeId == serviceType.ServiceTypeId);  db.ServiceTypes.RemoveRange(value);  db.SaveChanges();  }  static void DeleteService(HotelContext db, Service service)  {  var value = db.Services.Where(sv => sv.ServiceId == service.ServiceId).ToList();  db.Services.RemoveRange(value?.ToList());  db.SaveChanges();  }  static void AddClient(HotelContext db, Client client)  {  db.Clients.Add(client);  db.SaveChanges();  }  static void EditClient(HotelContext db, Client client)  {  Client value = db.Clients.Where(c => c.ClientId == client.ClientId).First();  value.Name = "Соловьев Дмитрий Сергеевич";  value.Passport = "HB1234567";  value.OccupancyDate = DateTime.Now;  value.OccupancyDate = value.OccupancyDate + new TimeSpan(7,0,0,0);  db.SaveChanges();  }  }  } |

Выполнение задания:

Создать с использованием .NET Core Entity Framework Core консольное приложение, содержащее набор классов, моделирующих предметную область, и осуществляющих генерацию и заполнение тестовыми наборами записей базу данных MS SQL Server или Sqlite. Для этого необходимо создать:

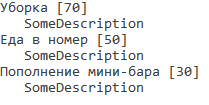
* Классы моделирующие не менее чем три таблицы базы данных согласно вашему варианту. Перечень таблиц предварительно согласовывается с преподавателем. Одна из таблиц обязательно должна находиться на стороне отношения «многие» связи с другой таблицей в схеме базы данных.
* Класс контекста данных.

3.2. Выполнить, используя объекты Entity Framework и LINQ:

1. Выборку всех данных из таблицы, стоящей в схеме базы данных на стороне отношения «один» – 1 шт.

Выполняет функция OutputServiceTypes :

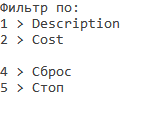
|  |
| --- |
| static void OutputServiceTypes(List<ServiceType> data)  {  foreach(var e in data)  Console.WriteLine($"{e.Name} [{e.Cost}] " +  $"{(string.IsNullOrWhiteSpace(e.Description) ? "" : $"{Environment.NewLine} {e.Description}")}");  } |

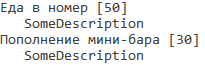


1. Выборку данных из таблицы, стоящей в схеме базы данных на стороне отношения «один», отфильтрованные по определенному условию, налагающему ограничения на одно или несколько полей – 1 шт.

Выполняют функции OutputServiceTypes и FilterServices :

|  |
| --- |
| static void OutputServiceTypes(List<ServiceType> data)  {  foreach(var e in data)  Console.WriteLine($"{e.Name} [{e.Cost}] " +  $"{(string.IsNullOrWhiteSpace(e.Description) ? "" : $"{Environment.NewLine} {e.Description}")}");  }  static IQueryable<ServiceType> FilterServices(IQueryable<ServiceType> data)  {  bool stay = true;  IQueryable<ServiceType> result = data;    while (stay)  {  Console.WriteLine("Фильтр по:");  Console.WriteLine("1 > Description");  Console.WriteLine("2 > Cost");  Console.WriteLine();  Console.WriteLine("4 > Сброс");  Console.WriteLine("5 > Стоп");  switch (Console.ReadKey(true).Key)  {  case ConsoleKey.D1:  {  Console.WriteLine("1 > Пуст");  Console.WriteLine("2 > Не пуст");  var k = Console.ReadKey(true).Key;  if (k == ConsoleKey.D1)  result = result.Where(o => string.IsNullOrWhiteSpace(o.Description));  else if (k == ConsoleKey.D2)  result = result.Where(o => !string.IsNullOrWhiteSpace(o.Description));  else  Console.WriteLine("Выбран неверный пункт");  }  break;  case ConsoleKey.D2:  {  Console.WriteLine("1 > Меньше");  Console.WriteLine("2 > Больше");  var k = Console.ReadKey(true).Key;  if (k == ConsoleKey.D1 || k == ConsoleKey.D2)  {  Console.WriteLine("Значение: ");  if (double.TryParse(Console.ReadLine(), out double value))  {  if (k == ConsoleKey.D1)  result = result.Where(o => o.Cost <= value);  else  result = result.Where(o => o.Cost >= value);  }  else  Console.WriteLine("Ошибка при преобразовании значения");  }  else  Console.WriteLine("Выбран неверный пункт");  }  break;  case ConsoleKey.D4:  result = data;  break;  case ConsoleKey.D5:  stay = false;  break;  }  Console.Clear();  }  return result;  } |

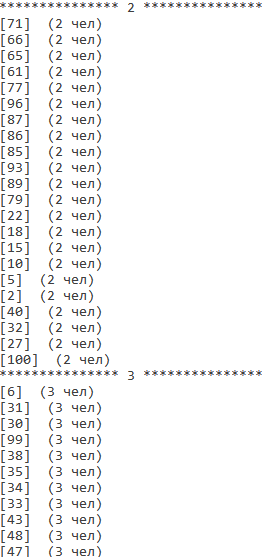




1. Выборку данных, сгруппированных по любому из полей данных с выводом какого-либо итогового результата (min, max, avg, сount или др.) по выбранному полю из таблицы, стоящей в схеме базы данных нас стороне отношения «многие» – 1 шт.

Выполняют функции OutputRooms и GroupRoomsByCapacity:

|  |
| --- |
| static void OutputRooms(IQueryable<IGrouping<int, Room>> data)  {  foreach (var group in data)  {  Console.WriteLine("{0} {1} {0}", new string('\*', 15), group.Key);  OutputRooms(group.ToList());  }  }  static void OutputRooms(List<Room> data)  {  foreach (var e in data)  Console.WriteLine($"[{e.RoomNo}] {e.RoomType} ({e.Capacity} чел)");  }  static IQueryable<IGrouping<int, Room>> GroupRoomsByCapacity(IQueryable<Room> data)  {  var value = from item in data  group item by item.Capacity;  return value;  } |



1. Выборку данных из двух полей двух таблиц, связанных между собой отношением «один-ко-многим» – 1 шт.

Выполняет функция JoinTwoTables:

|  |
| --- |
| static void JoinTwoTables(List<Service> services, List<ServiceType> serviceType)  {  var result = services.Join(serviceType,  p => p.ServiceTypeId, q => q.ServiceTypeId,  (p, q) => new  {  ServiceName = q.Name,  ClientName = p.Client?.Name,  p.Client?.Room?.RoomNo,  q.Cost  });  // вывод тут же - работать с анонимными типами между методами тот еще гемор  foreach (var item in result)  Console.WriteLine($"{item.ServiceName} для {item.ClientName}(комната #{item.RoomNo}) [{item.Cost}]");  } |

1. Выборку данных из двух таблиц, связанных между собой отношением «один-ко-многим» и отфильтрованным по некоторому условию, налагающему ограничения на значения одного или нескольких полей – 1 шт.

static void JoinTwoTablesAndFilterIt(List<Service> services, List<TypeOfService> typesOfServices)

{

var result =

services.Join(

typesOfServices,

p => p.TypeOfServiceID,

q => q.TypeOfServiceID,

(p, q) => new

{

ID = p.ServiceID,

Name = q.Name,

CustomerName = p.CustomerName,

Price = q.Price

});

result =

from res in result

where res.Name.Length < 8

select res;

Console.WriteLine(String.Format("|{0,5}|{1,10}|{2,10}|{3,10}|", "ID", "Type", "Customer", "Price"));

foreach (var item in result)

Console.WriteLine(String.Format("|{0,5}|{1,10}|{2,10}|{3,10}|", item.ID, item.Name, item.CustomerName, item.Price));

}



1. Вставку данных в таблицы, стоящей на стороне отношения «Один» – 1 шт.

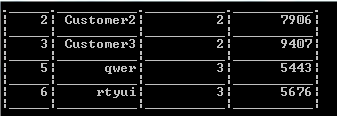
static void AddService(SpaSalonDB db, Service service)

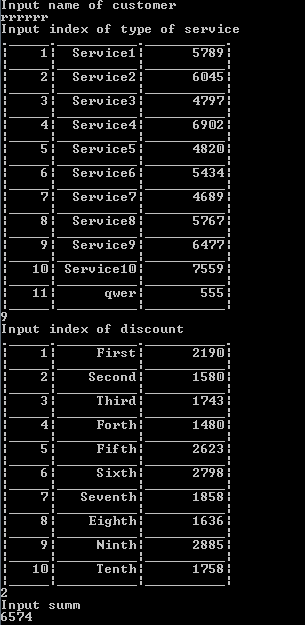
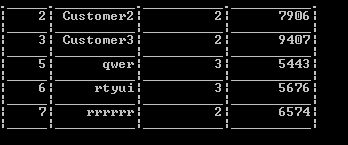
{

db.Servises.Add(service);

db.SaveChanges();

}



1. Вставку данных в таблицы, стоящей на стороне отношения «Многие» – 1 шт.:

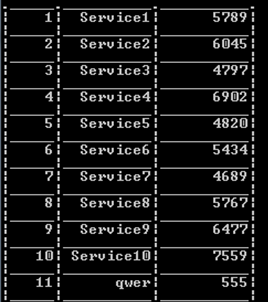
static void AddTypeOfService(SpaSalonDB db, TypeOfService typeOfService)

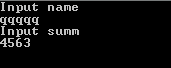
{

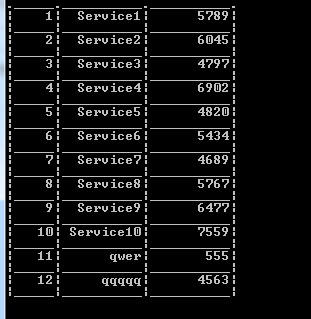
db.TypesOfServices.Add(typeOfService);

db.SaveChanges();

}







1. Удаление данных из таблицы, стоящей на стороне отношения «Один» – 1 шт.

static void DellService(SpaSalonDB db, int id)

{

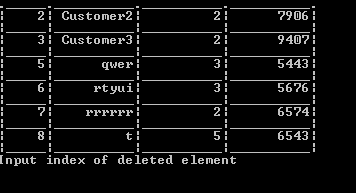
var tmp = db.Servises.Where(a => a.ServiceID == id);

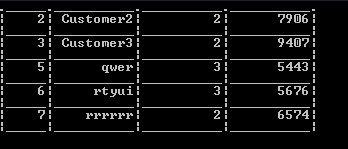
var oldValue = db.Servises.Where(a => a.ServiceID == id);

db.Servises.RemoveRange(oldValue);

db.SaveChanges();

}





1. Удаление данных из таблицы, стоящей на стороне отношения «Многие» – 1 шт.

static void DellTypeOfService(SpaSalonDB db, int id)

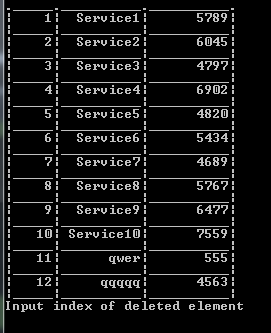
{

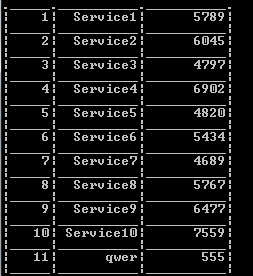
var oldValue = db.TypesOfServices.Where(a => a.TypeOfServiceID == id);

db.TypesOfServices.RemoveRange(oldValue);

db.SaveChanges();

}





1. Обновление удовлетворяющих определенному условию записей в любой из таблиц базы данных – 1 шт.

static void EditService(SpaSalonDB db, int id, Service service)

{

Service oldValue = db.Servises.Where(a => a.ServiceID == id).First();

oldValue.CustomerName = service.CustomerName;

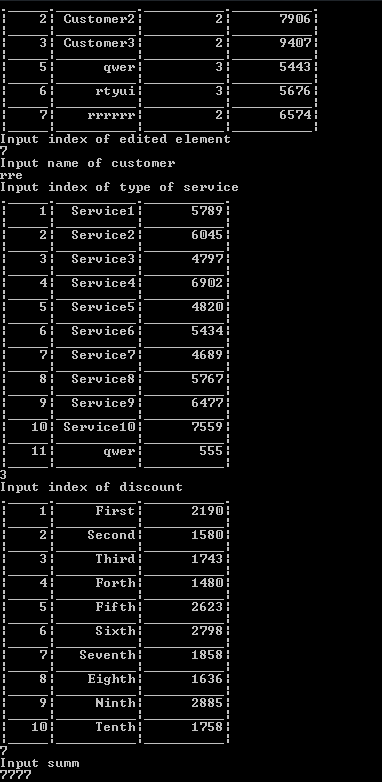
oldValue.DiscountType = service.DiscountType;

oldValue.Summ = service.Summ;

oldValue.Type = service.Type;

db.SaveChanges();

}



**3. Выводы.**

Ознакомился с возможностями ENTITY FRAMEWORK для моделирования предметной области, выборки, вставки, удаления и обновления данных объектов, связанных с таблицами базы данных заданной предметной области.