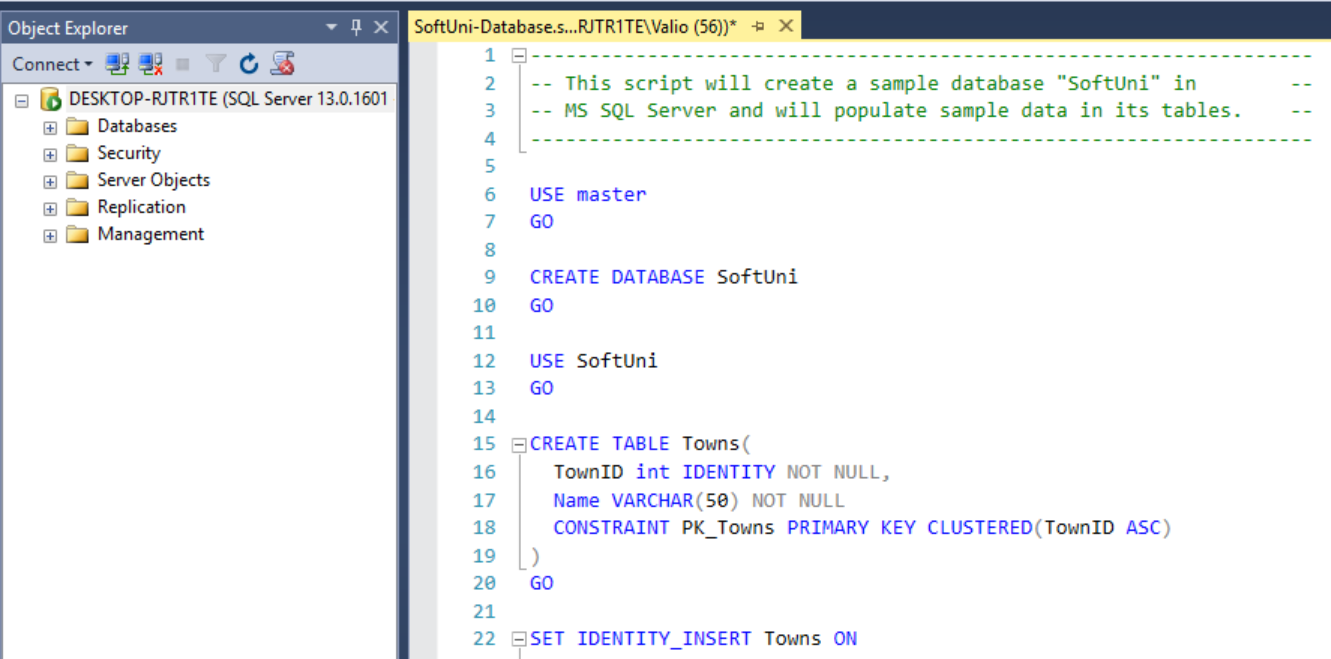
# Exercises: Introduction to Entity Framework

This document defines the **exercise assignments** for the ["Databases Advanced – Entity Framework" course @ Software University.](https://softuni.bg/trainings/1442/databases-advanced-entity-framework-october-2016) Please submit your solutions in [Judge.](https://judge.softuni.bg/)

## Import the SoftUni Database

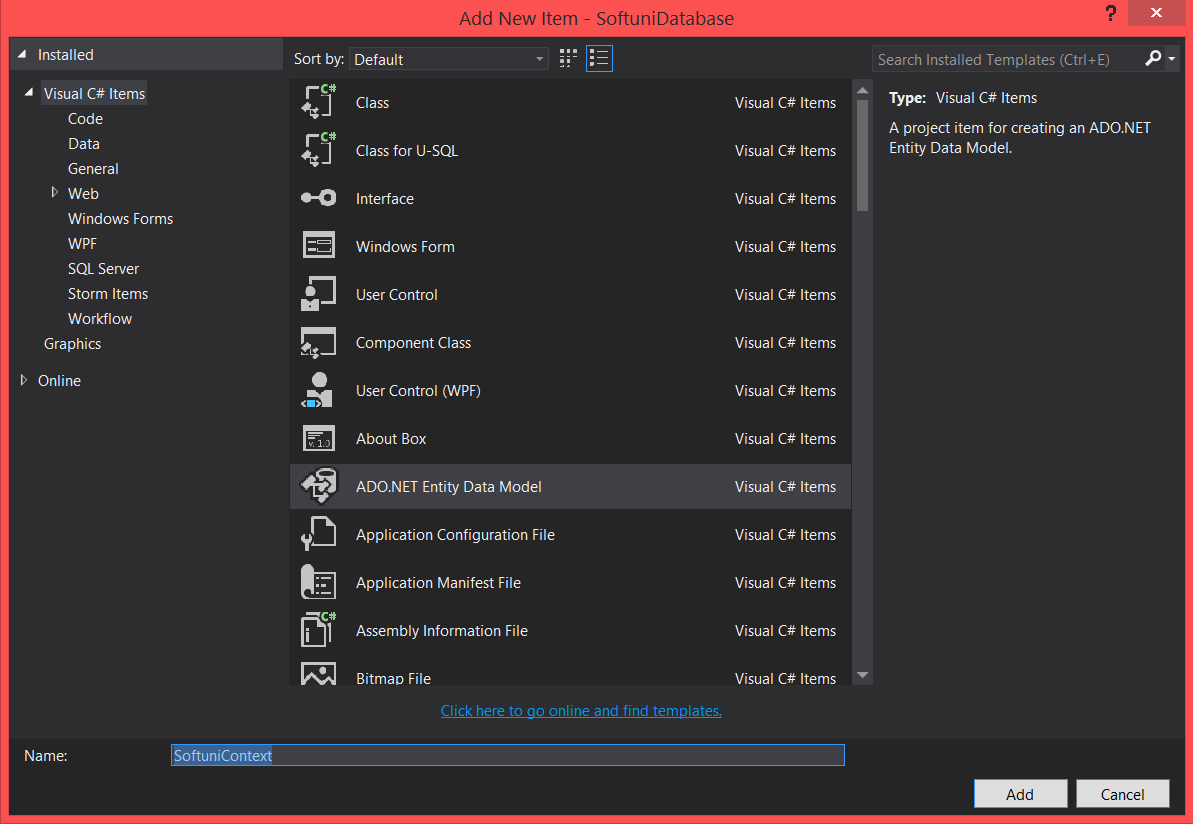
Import the SoftUni database into SQL Management Studio by **executing** the provided **.sql** script.

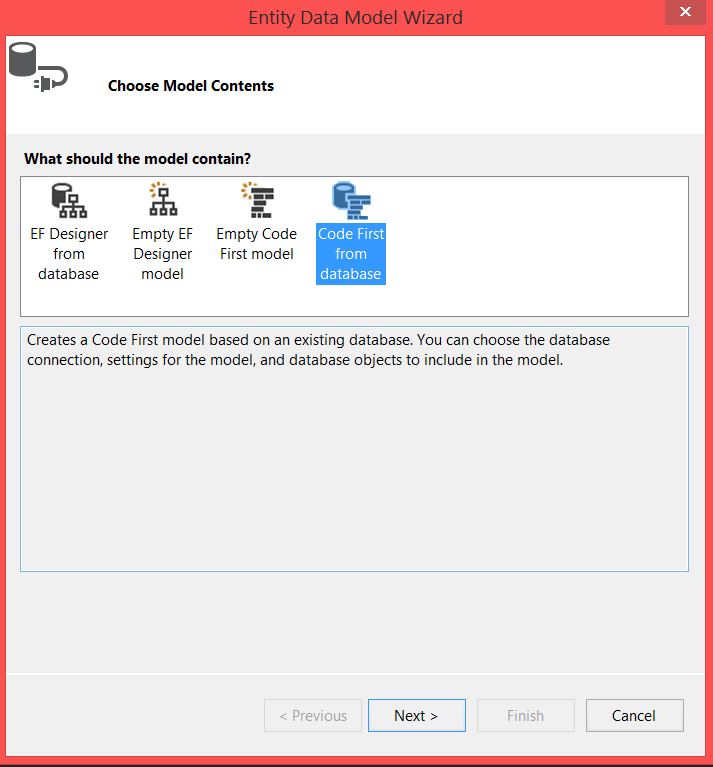
****

## Database First

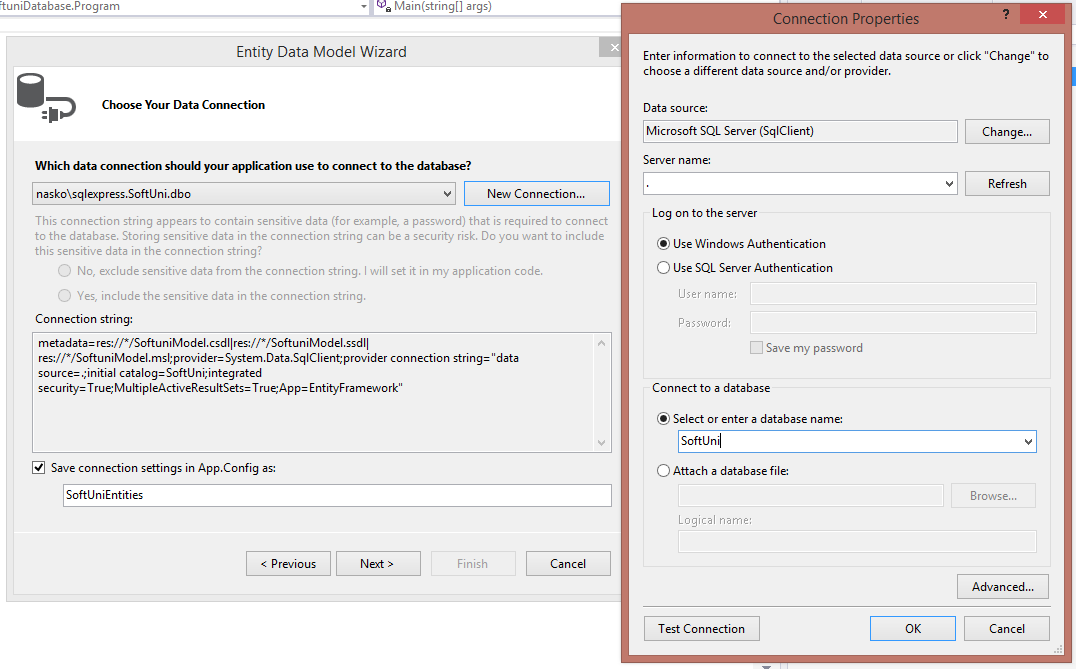
Model the existing database by using Database First.

First create a new empty **ConsoleApplication** and after it is created right click on the project and add a new item. In the newly appeared window select **ADO.NET Entity Data Model** and call it **SoftuniContext** as shown below:

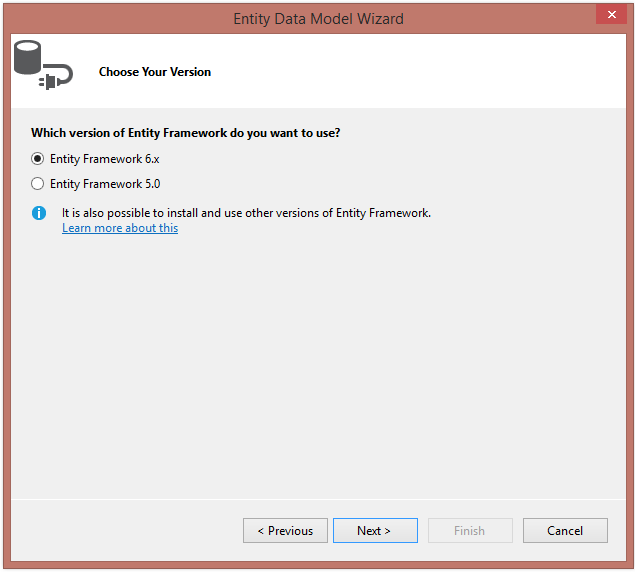




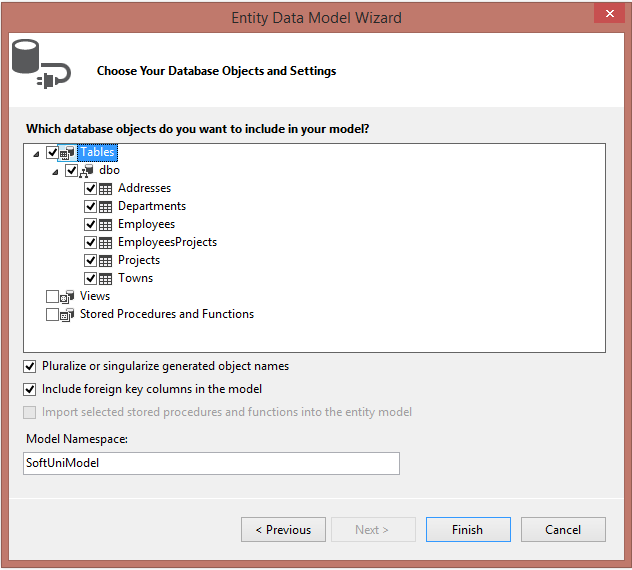
Choose your **server name** (SQL management studio connection) and the **database** you wish to model.



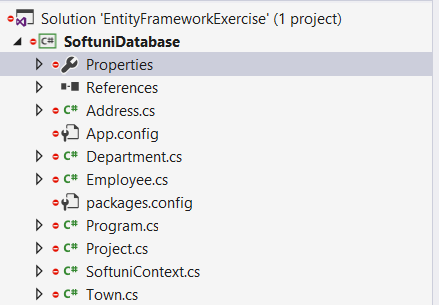
Select the latest stable Entitiy Framework version.



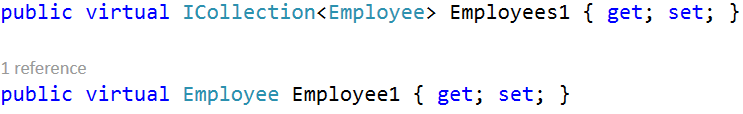
Select the tables the desired tables from the target database. Exclude any views and stored procedures.

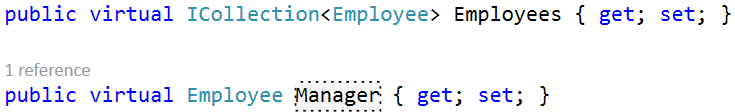


Click finish. The result should be all the tables that are in your table, generated as classes:

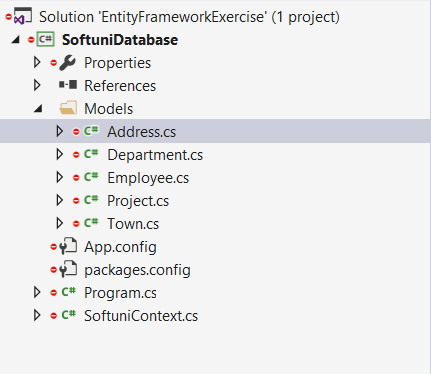


Entity Framework has successfully **mapped the database schema to C# classes**. However, it isn't quite good with names (See the **Employee** class) - e.g. **Employees1**, **Employee1**. Edit the properties and give them more **proper names**.





Finally, before we begin working with the data, please create a new folder in the project called Models and insert your models in it. (Don’t forget to change the namespace, once the models are moved).



## Employees full information

Let's start writing C# queries! Your task is to extract **all employees** print their **first**, **last** and **middle** name, their **job** **title** and **salary** separated with a space. (**Test in judge)**

SoftuniContext context = new SoftuniContext();

using (context)

{

var employees = context.Employees

.Select(e => new

{

FirstName = e.FirstName,

LastName = e.LastName,

MiddleName = e.MiddleName,

JobTitle = e.JobTitle,

Salary = e.Salary

});

foreach (var employee in employees)

{

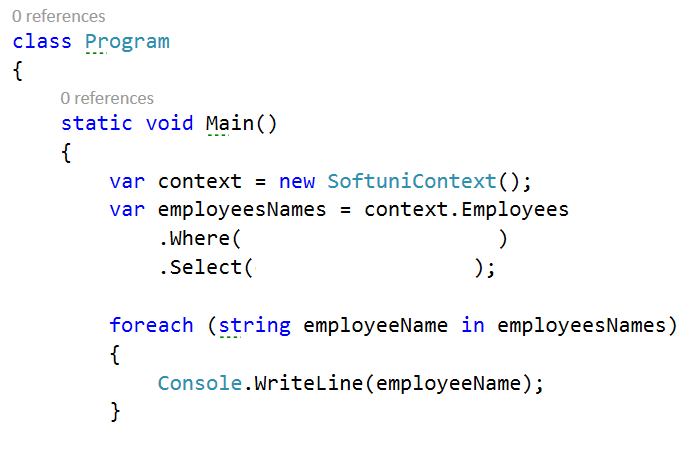
Console.WriteLine(string.Join(" ", employee.FirstName, employee.LastName, employee.MiddleName, employee.JobTitle, employee.Salary));

}

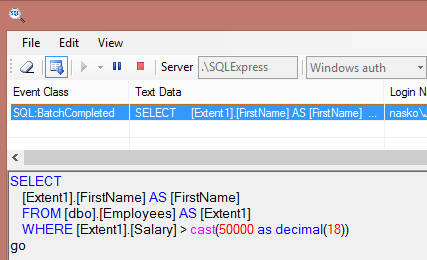
}

## Employees with Salary Over 50 000

Your task is to extract **all employees** with **salary** over **50000**. Create a new context and use lambdas to build your query. Make sure the query returns **only the** **first names** of those employees.



Use Express Profiler and check if the made query by Entity Framework is correct (there is only one query, but there may be more that are performed by EF for checks).



SoftuniContext context = new SoftuniContext();

using (context)

{

var employees = context.Employees

.Where(e=> e.Salary > 50000)

.Select(e => new

{

FirstName = e.FirstName,

});

foreach (var employee in employees)

{

Console.WriteLine(employee.FirstName);

}

}

## Employees from Seattle

Extract all employees from the **Research and Development** department. Order them by **salary** (in ascending order), then by **first name** (in descending order). Print only their **first name**, **last name**, **department name** and **salary** in the format shown below:

SoftuniContext context = new SoftuniContext();

using (context)

{

var employees = context.Employees

.Where(e=> e.Department.Name == "Research and Development")

.OrderBy(e=> e.Salary)

.ThenByDescending(e=> e.FirstName)

.Select(e => new

{

e.FirstName,

e.LastName,

DepartmentName = e.Department.Name,

e.Salary

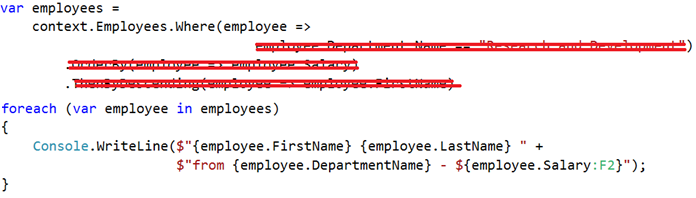
});

foreach (var employee in employees)

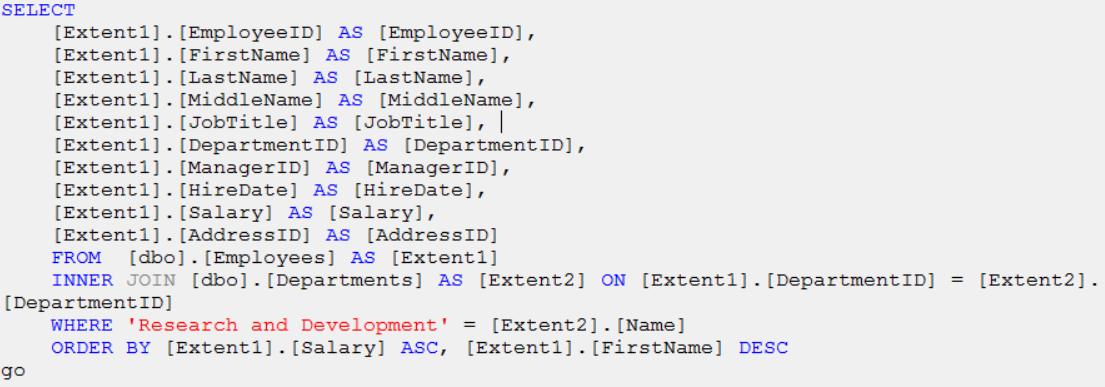
{

Console.WriteLine($"{employee.FirstName} {employee.LastName} from {employee.DepartmentName} - ${employee.Salary:F2}");

}



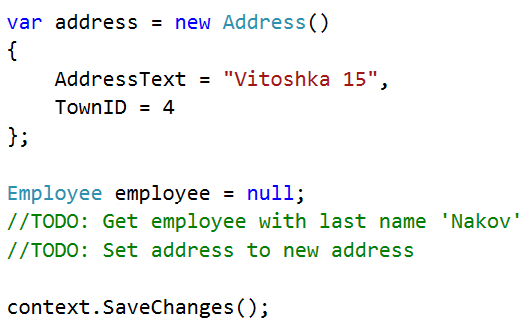
Use Express Profiler and check if the made query by Entity Framework is correct (there is only one query).



**Test in judge**

## Adding a New Address and Updating Employee

Create a new address with **text** "**Vitoshka 15**" and **TownId** **4**. Set that address to the employee with last name “Nakov”.



The above code should successfully **insert a new address** in the database and **set it as Nakov's new address**.

Then order by descending all the employees by their Address’ Id, take 10 rows and from them, take the AddressText. Print the results each on a new line:

**Test in judge**

SoftuniContext context = new SoftuniContext();

using (context)

{

var address = new Address()

{

AddressText = "Vitoshka 15",

TownID = 4

};

var nakov = context.Employees.First(e => e.LastName == "Nakov");

nakov.Address = address;

context.SaveChanges();

var employees = context.Employees

.OrderByDescending(e => e.Address.AddressID)

.Take(10)

.Select(e => e.Address.AddressText);

foreach (var employee in employees)

{

Console.WriteLine(employee);

}

}

## Delete Project by Id

Let's delete a project by id(2). Sounds simple:



However, …



The project is **referenced** by the junction (many-to-many) table **EmployeesProjects**. Therefore we cannot safely delete it. First, we need to remove any references to that row in the **Projects** table.

This is done by removing the project from all employees who reference it. **Tip**: Get those employees from the **Employee** navigation property.



Then take 10 projects, select their names and print them on the console each on a new line.

**Test in judge.**

var project = context.Projects.Find(2);

var employees = project.Employees;

foreach (var employee in employees)

{

employee.Projects.Remove(project);

}

context.SaveChanges();

context.Projects.Remove(project);

context.SaveChanges();

var projects = context.Projects

.Take(10)

.Select(p => p.Name);

foreach (var proj in projects)

{

Console.WriteLine(proj);

}

## Find employees in period

Find all employees who have **projects** started in the time period **2001 - 2003** (inclusive). Print each employee's **first name**, **last name** and **manager’s first name** and each of their projects' **name**, **start date**, **end date**. Here is the format:

“first Name lastName managerFirstName”

“—projectName projectStart projectEnd”

CultureInfo.DefaultThreadCurrentCulture = CultureInfo.InstalledUICulture;

SoftuniContext context = new SoftuniContext();

using (context)

{

var employees = context.Employees

.Where(e => e.Projects.Count(p => p.StartDate.Year >= 2001 & p.StartDate.Year <= 2003) > 0).Take(30);

foreach (var employee in employees)

{

Console.WriteLine($"{employee.FirstName} {employee.LastName}

{employee.Manager.FirstName}");

foreach (var project in employee.Projects)

{

Console.WriteLine($"--{project.Name} {project.StartDate} {project.EndDate}" );

}

}

}

**Test in judge**

## Addresses by town name

Find all addresses, **ordered** by the **number of employees** who live there (descending), then by **town name** (ascending). Take only the **first 10 addresses**. For each address print it in the format

“addressText, townName – numberOfEmployees employees”

**Test in judge**

SoftuniContext context = new SoftuniContext();

using (context)

{

var addresses = context.Addresses

.OrderByDescending(a => a.Employees.Count)

.ThenBy(t => t.Town.Name)

.Take(10);

foreach (var address in addresses)

{

Console.WriteLine($"{address.AddressText}, {address.Town.Name} - {address.Employees.Count} employees");

}

}

## Employee with id 147 sorted by project names

Get an **employee with id 147.** Print only his/her **first name**, **last name**, **job title** and **projects** (print only their names). The projects should be **ordered** **by** **name** (ascending). Format of the output:

“first Name lastName job Title”

“projectName”

SoftuniContext context = new SoftuniContext();

using (context)

{

var employees = context.Employees

.Where(e => e.EmployeeID == 147)

.Select(e => new

{

e.FirstName,

e.LastName,

e.JobTitle,

e.Projects

});

foreach (var e in employees)

{

Console.WriteLine($"{e.FirstName} {e.LastName} {e.JobTitle}");

foreach (var project in e.Projects.OrderBy(p=> p.Name))

{

Console.WriteLine($"{project.Name}");

}

}

**Test in judge**

## Departments with more than 5 employees

Find **all departments** with more than **5 employees**. Order them by **employee count** (ascending).Print the **department name**, **manager’s first name** and first name, last name and job title of every **employee**. Format of the output:

“department.Name manager.FirstName”

“employee1.FirstName employee1.LastName employee1.JobTitle”

…

**Test in judge**

SoftuniContext context = new SoftuniContext();

using (context)

{

var departments = context.Departments

.Where(d => d.Employees.Count > 5)

.OrderBy(d => d.Employees.Count);

foreach (var department in departments)

{

Console.WriteLine($"{department.Name} {department.Employee.FirstName}");

foreach (var employee in department.Employees)

{

Console.WriteLine($"{employee.FirstName} {employee.LastName} {employee.JobTitle}");

}

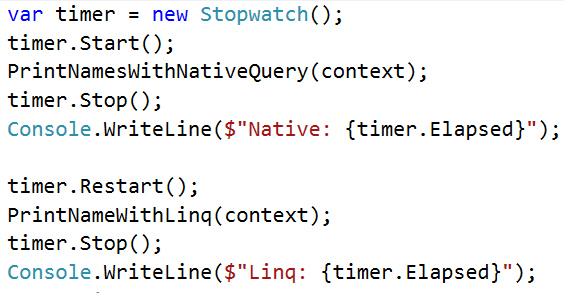
}

}

## \*Native SQL Query

Find all **employees** who have projects with **start date** in the year **2002**. Select only their **first name**. Solve this task by using both **LINQ query** and **native SQL query** through the context.

Measure the difference in performance in both cases with a **Stopwatch**.



**Tip**: Use the **context.Database.SqlQuery<T>()** method for executing native SQL queries.

static void Main(string[] args)

{

SoftuniContext context = new SoftuniContext();

StringBuilder content = new StringBuilder();

using (context)

{

context.Addresses.Count();

var timer = new Stopwatch();

timer.Start();

PrintNameWithLinq(context);

timer.Stop();

content.AppendLine($"Linq: {timer.Elapsed}");

timer.Reset();

timer.Start();

PrintNamesWithNativeQuery(context);

timer.Stop();

content.AppendLine($"Native: {timer.Elapsed}");

timer.Reset();

}

}

private static void PrintNameWithLinq(SoftuniContext context)

{

var employeesNames =

context.Employees

.Where(employee =>

employee.Projects.Count(project => project.StartDate.Year == 2002) != 0)

.Select(employee => employee.FirstName).GroupBy(s => s);

foreach (var s in employeesNames)

{

}

}

private static void PrintNamesWithNativeQuery(SoftuniContext context)

{

string query = "SELECT em.FirstName FROM Employees em " +

"JOIN EmployeesProjects emproj " +

"ON emproj.EmployeeId = em.EmployeeId " +

"JOIN Projects proj " +

"ON emproj.ProjectId = proj.ProjectId AND YEAR(proj.StartDate) = 2002 " +

"GROUP BY em.FirstName";

var result = context.Database.SqlQuery<string>(query);

foreach (var f in result)

{

}

}

}

}

## \* Play with the SQL Server Profiler

Your task is to use the [SQL Server ExpressProfiler](https://expressprofiler.codeplex.com/) to view all your queries from the previous homework task.

## \* Explore the Full Source Code of Entity Framework

Your task is to download (clone the repository) and explore the full source code of Entity Framework. You can find it on <http://entityframework.codeplex.com>. Do not submit anything for this problem.

## Find Latest 10 Projects -

Write a program that prints information about **last 10 started projects**. Print **their name, description, start and end date** and **sort** **them by name** lexicographically. Format of the output:

“Name Description StartDate EndDate”

**Test in judge**

SoftuniContext context = new SoftuniContext();

StringBuilder content = new StringBuilder();

using (context)

{

var projects = context.Projects

.OrderByDescending(p => p.StartDate)

.Take(10)

.OrderBy(e => e.Name);

foreach (var project in projects)

{

//Console.WriteLine($"{project.Name} {project.Description} {project.StartDate} {project.EndDate}");

content.AppendLine($"{project.Name} {project.Description} {project.StartDate} {project.EndDate}");

Console.WriteLine(content);

}

}

## Increase Salaries

Write a program that increase salaries of all employees that are in the **Engineering**, **Tool Design**, **Marketing** or **Information Services** department by **12%**. Then **print first name, last name and salary** for those employees whose salary was increased. Format of the output:

“FirstName LastName $Salary”

**Test in judge**

SoftuniContext context = new SoftuniContext();

//StringBuilder content = new StringBuilder();

using (context)

{

var employees = context.Employees

.Where(

e =>

e.Department.Name == "Engineering" || e.Department.Name == "Tool Design" ||

e.Department.Name == "Marketing" || e.Department.Name == "Information Services");

foreach (var employee in employees)

{

employee.Salary \*= 1.12m;

Console.WriteLine($"{employee.FirstName} {employee.LastName} (${employee.Salary:F6})");

}

context.SaveChanges();

}

## Remove Towns

Write a program that **deletes town** which name is given as an input. Also, **delete all addresses** that are in those towns. Print on the console the number addresses that were deleted. You may also note that there are some Employees in that have addresses which may be a problem for deleting the addresses. So first off start by setting the AddressID of each employee in for the given address to null. After all of them are set to null, you may safely remove all the addresses from the context.Addresses and finally remove the given town. You may test this task locally, so that you can see what happens for more than 1 case of deletion.

SoftuniContext context = new SoftuniContext();

//StringBuilder content = new StringBuilder();

using (context)

{

string input = Console.ReadLine();

var town = context.Towns.FirstOrDefault(t=> t.Name == input);

Address[] townAdresses = town.Addresses.ToArray();

int n = 0;

foreach (Address townAddress in townAdresses)

{

Employee[] employeeAddresses = townAddress.Employees.ToArray();

foreach (var employeeAddress in employeeAddresses)

{

employeeAddress.AddressID = null;

}

}

context.Addresses.RemoveRange(townAdresses);

context.Towns.Remove(town);

context.SaveChanges();

Console.WriteLine($"{townAdresses.Length} address in {input} was deleted");

}

### Example

|  |  |
| --- | --- |
| **Input** | **Output** |
| Sofia | 1 address in Sofia was deleted |
| Seattle | 44 addresses in Seattle were deleted |

## Find Employees by First Name starting with ‘SA’

Write a program that finds all employees whose first name starts with ‘SA’. Print their first, last name, their job title and salary in the format given in the example below.

FirstName LastName – JobTitle - ($Salary)

**Test in judge**

SoftuniContext context = new SoftuniContext();

StringBuilder content = new StringBuilder();

using (context)

{

var employees = context.Employees

.Where(e => e.FirstName.StartsWith("SA"));

foreach (var employee in employees)

{

Console.WriteLine($"{employee.FirstName} {employee.LastName} - {employee.JobTitle} - (${employee.Salary})");

}

}

## First Letter

Use the **Gringotts database**. Write a program that prints all unique wizard first letters of their **first names** only if they have deposit of type Troll Chest. Order them alphabetically. Use DISTINCT for uniqueness.

GringottsContext context = new GringottsContext();

using (context)

{

var wizz = context.WizzardDeposits

.Where(d => d.DepositGroup == "Troll Chest")

.Select(e => e.FirstName.Substring(0, 1))

.Distinct()

.OrderBy(e => e);

foreach (var item in wizz)

{

Console.WriteLine(item);

}

}

### Example

|  |
| --- |
| **Output** |
| A |
| … |

**Test in judge**