

Setup Exam INFDTB

Steps that can be done **BEFORE** the exam

Running Docker-Compose:

<https://docs.docker.com/compose/gettingstarted/>

You need to have Docker Engine and Docker Compose on your machine. You can either:

- Install Docker Engine and Docker Compose as standalone binaries

OR

- Install Docker Desktop which includes both Docker Engine and Docker Compose

Define services in a Compose file by Creating a file called docker-compose.yaml in a directory (for instance your project directory) and use the following (this file has been provided previously and can also be found in the Exam assignment):

```
docker-compose.yaml
1  version: '3'
2
3  services:
4    postgres:
5      image: postgres
6      command: -c shared_buffers=256MB -c max_connections=200
7      ports:
8        - 5432:5432
9      environment:
10       POSTGRES_HOST_AUTH_METHOD: trust
11      volumes:
12        - pgdata:/var/lib/postgresql/data
13        - ./scripts:/scripts
14      networks:
15        - mynetwork
16      logging:
17        driver: none
18
19    pgadmin:
20      image: dpage/pgadmin4
21      environment:
22        PGADMIN_DEFAULT_EMAIL: admin@ad.min
23        PGADMIN_DEFAULT_PASSWORD: admin
24        PGADMIN_LISTEN_PORT: 80
25      ports:
26        - '8081:80'
27      volumes:
28        - pgadmin-data:/var/lib/pgadmin
29        - ./scripts:/scripts
30      depends_on:
31        - "postgres"
32      networks:
33        - mynetwork
34      logging:
35        driver: none
36
37  volumes:
38    pgdata:
39    pgadmin-data:
40
41  networks:
42    mynetwork:
43      driver: bridge
44
```

Build and run your app with Compose from the chosen directory, start up your containers by running : **docker compose up -d**

You may use **MS Visual Studio**, or **VS Code**

It is advisable to install and use **.Net 7**

It is advisable to create and clone on your machine **TWO** different git repositories in two different folders.

Create **TWO** different projects (one for the Linq Questions section and one for the Model Question section) following these steps every time:

Creating app:

Visual Studio

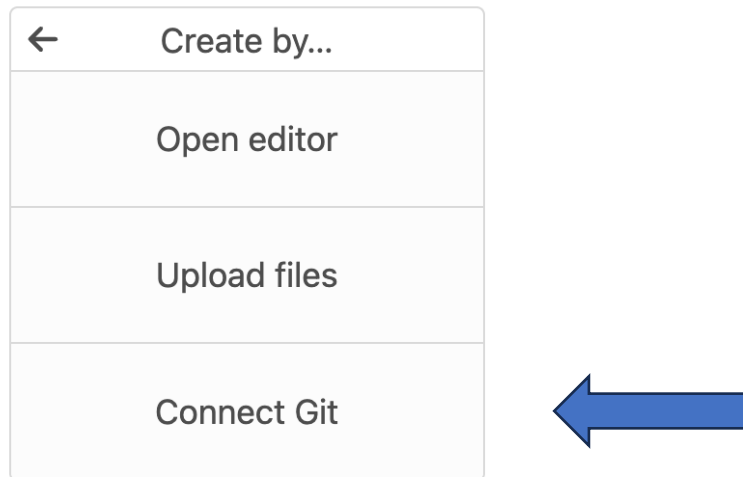
- a. Create a console app with logical name (say DBExam)
File -> New -> Project -> Console App (c#) -> next [ProjectName]-> Finish
- b. Add the following NuGet Packages (right click in solution explorer and goto Manage
NuGet Packages OR Tools-> NuGet Package Manager-> Manage NuGet Package)
Microsoft.EntityFrameworkCore
Microsoft.EntityFrameworkCore.Tools
Microsoft.EntityFrameworkCore.Design
Npgsql.EntityFrameworkCore.PostgreSQL

Visual Studio Code / CLI

- a. Create a console app with logical name (say DBExam)
`dotnet new console -o DBExam`
`cd DBExam`
- b. Install dotnet-ef (only needed once):
`dotnet tool install --global dotnet-ef`
- c. Add the following NuGet Packages
these steps are NOT needed when using the provided csproj file
`dotnet add package Microsoft.EntityFrameworkCore`
`dotnet add package Microsoft.EntityFrameworkCore.Tools`
`dotnet add package Microsoft.EntityFrameworkCore.Design`
`dotnet add package Npgsql.EntityFrameworkCore.PostgreSQL`
- c. Open project directory in VS Code (you may add packages via terminal in vs code)

DURING the Exam

1. In CodeGrade connect each of these git repositories to the two exam sections: one for the Linq Questions section and one for the Model Question section.



2. Download and add to the respective sections (Linq Questions; Model Question) the files available in Teams.
3. Content of the provided csproj files should replace the respective ones in the respective project.
4. **Run the migrations** (Tools/NuGet Package Manager/PMC ► CLI Terminal) and make sure Database created by looking into PgAdmin.

Visual Studio
Add-Migration migrationName Update-Database

Visual Studio Code / CLI
dotnet ef migrations add migrationName dotnet ef database update

Answers can be provided by changing the relevant methods (ONLY) in the file **Solution.cs** (Linq Questions section).

And by changing (ONLY) the file **Model.cs** for the Model Question section.

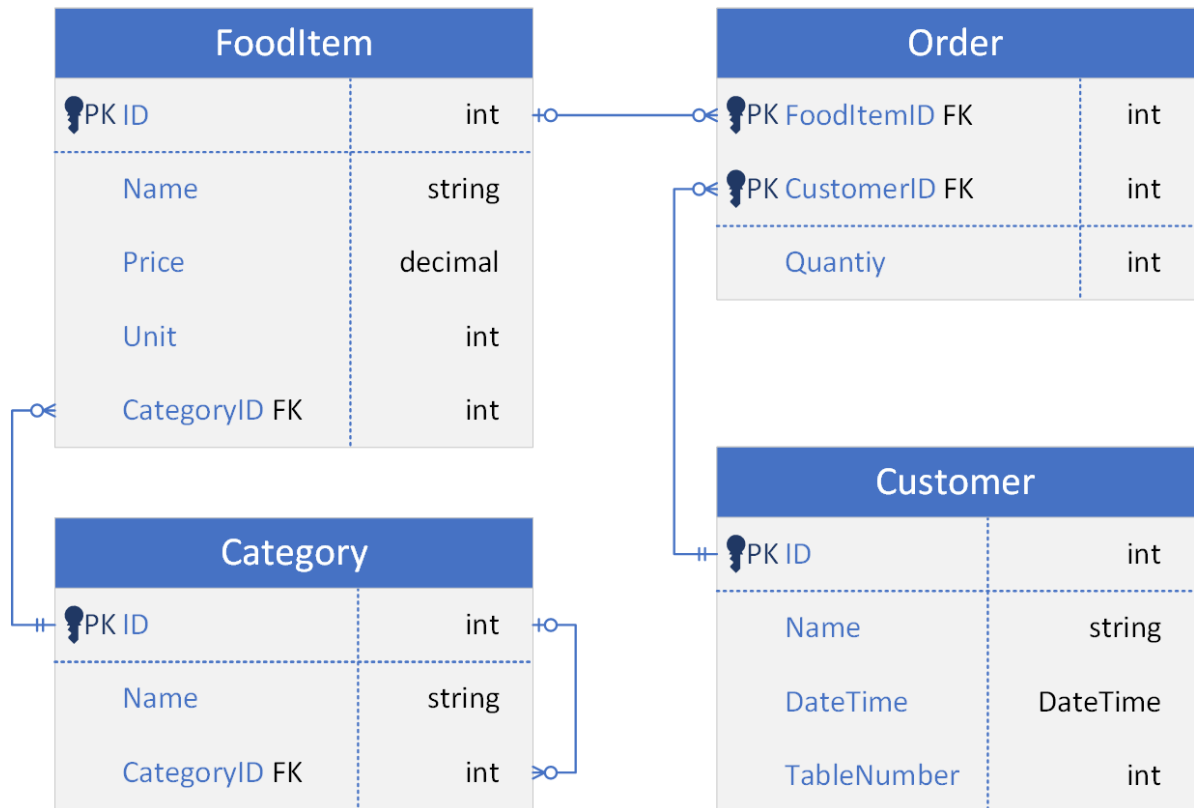
Other changes will not affect the result.

Submission in CodeGrade can happen an indefinite number of times using one of these three ways: using the online editor, uploading the relevant files (Solution.cs, Model.cs), or with a git push.

Sample Exam INFDTB (2023-24)

The case study: (Restaurant)

- This case study is about a restaurant dine out facility
- Following data is recorded:
- The Restaurant menu is organised into Food Categories, and Sub categories. Where in each category/Sub category there are some food items.
- There is no fixed configuration of tables/chairs. However two or more customers visiting together maybe allocated a table, in order to calculate collective bill if required.
- A customer may order one or more Food Item(s).



Below you may find sample data for each entity/table stored in database, the snippet also give away information about column names and data types. This is dummy data generated randomly so data and result may differ for each student.





Categories

	ID [PK] integer	Name text	CategoryID integer
1	1	Beverages	[null]
2	2	Starter	[null]
3	3	Sea Food	[null]
4	4	Vegetarian	[null]
5	5	Meat	[null]
6	6	Dessert	[null]
7	7	Milk Shake	1
8	8	Hot Drinks	1
9	9	Soft Drinks	1
10	10	Sea Food Starter	2
11	11	Vegetarian Starter	2
12	12	Meat Starter	2

FoodItems

	ID [PK] integer	Name text	CategoryID integer	Price numeric	Unit text
1	1	Tea	8	2.50	250ml
2	2	Espresso	8	3	30ml
3	3	Cappuccino	8	3.45	100ml
4	4	Chinotto	9	3.50	300ml
5	5	Water	9	1.65	1Liter
6	6	Fritto misto	10	9.50	350g
7	7	Samosa	11	5.50	100g
8	8	Corn Kebab	11	6.50	100g
9	9	Chicken Tikka	11	10.50	300g
10	10	Coniglio in agrodolce	5	15.55	250g
11	11	Spinach	4	12	[null]
12	12	Aubergine	4	9.50	[null]
13	13	Broccoli	4	8	[null]
14	14	Melanzane alla parmigiana	4	12.55	[null]
15	15	Meatballs	5	12	[null]
16	16	Chicken Fried Rice	5	13.50	[null]
17	17	Vitello Tonnato	5	20	[null]
18	18	Chocolate Coffee Truffle	6	9	[null]
19	19	Apple Pie	6	8.50	[null]
20	20	Brownies	6	9	[null]
21	21	Cannoli	6	7.25	[null]
22	22	Sebadas	6	8.75	[null]
23	23	Pardula	6	3	[null]
24	24	Mango Lassi	7	2	[null]

Customers

	ID [PK] integer 	Name text 	DateTime timestamp with time zone 	TableNumber integer 
22	22	Jeffie Kleinmintz	2022-05-20 20:26:00+00	8
23	23	Gardiner Vyvyan	2022-07-31 17:45:00+00	3
24	24	Alfie Oldam	2021-12-23 00:43:00+00	4
25	25	Teodor Andric	2022-05-20 20:26:00+00	5
26	26	Ave Lovel	2022-08-06 20:14:00+00	3
27	27	Alayne Dowe	2022-02-18 15:39:00+00	7

Orders

	CustomerID [PK] integer 	FoodItemID [PK] integer 	Quantity integer 
6	3	13	1
7	4	18	2
8	5	11	2
9	6	12	1
10	6	14	1
11	7	5	2
12	7	11	2
13	7	24	1
14	8	21	1

Exercise 1 [1.0 Points]:

Write a query to list down all Dishes (FoodItems projected into Dishes) containing the given name within the minimum and maximum given prices (included).

Method signature:

```
public static IQueryable<Dish> Q1(ExamContext db, string name, decimal minPrice, decimal maxPrice)
```

The record **Dish** can be found in the file **DataFormats.cs**

```
public record Dish
{
    public Dish(string name, decimal price, string? unit) {
        Name = name;
        Price = price;
        Unit = unit;
    }

    public string Name { get; set; } = null!;
    public decimal Price { get; set; }
    public string? Unit { get; set; } = null!;
}
```

Sample Result:

```
Minimum price (randomly generated): 1.5
Maximum price: 28.5
Correct number of elements: True -> found: 5 | correct: 5
Correct elements retrieved: True
Retrieved records:
Dish { Name = Melanzane alla parmigiana, Price = 12.55, Unit = }
Dish { Name = Meatballs, Price = 12, Unit = }
Dish { Name = Fritto misto, Price = 9.50, Unit = 350g }
Dish { Name = Samosa, Price = 5.50, Unit = 100g }
Dish { Name = Mango Lassi, Price = 2, Unit = }
```

Exercise 2 [1.0 Points]:

Write a query to list down all FoodItems and CategoryName projected into DishAndCategory ordered by a Customer (CustomerID given as parameter).

Method signature:

```
public static IQueryable<DishAndCategory> Q2(ExamContext db, int customerId)
```

The record **DishAndCategory** can be found in the file **DataFormats.cs**

```
public record DishAndCategory : Dish
{
    public DishAndCategory(string name, decimal price, string? unit, string categoryName) :
        base(name, price, unit) {
        CategoryName = categoryName;
    }

    public string CategoryName { get; set; } = null!;
}
```

Sample Result:

```
CustomerID: 90
Correct number of elements: True -> found: 2 | correct: 2
Correct elements retrieved: True
Retrieved records:
DishAndCategory { Name = Water, Price = 1.65, Unit = 1Liter, CategoryName = Soft Drinks }
DishAndCategory { Name = Spinach, Price = 12, Unit = , CategoryName = Vegetarian }
```

Exercise 3 [1.5 Points]:

Write a query to list down the bills: FoodItems, Order Quantity, Unit Price, Total, for the first "number" of Customers (*ordered in a descending way based on Total*).

Returns an IQueryable<CustomerBill> which will let fetch exactly "number" bills.

If for instance number: 3, it will fetch the first 3 bills ordered (descending) by Total.

Method signature:

```
public static IQueryable<CustomerBill> Q3(ExamContext db, int number)
```

The record **CustomerBill** can be found in the file **DataFormats.cs**

```
public record BillItem : Dish
{
    public BillItem(string name, decimal price, string? unit, int quantity) : base(name, price, unit) {
        Quantity = quantity;
    }
    public int Quantity { get; set; }
}

public record CustomerBill {
    public int CustomerID { get; set; }
    public List<BillItem>? Bill { get; set; }
    public decimal Total { get; set; }
}
```

Sample Result:

```
Correct number of elements: True -> found: 4 | correct: 4
Correct elements retrieved: True
Retrieved records:
Customer: 36
BillItem { Name = Samosa, Price = 5.50, Unit = 100g, Quantity = 2 }
BillItem { Name = Vitello Tonnato, Price = 20, Unit = , Quantity = 2 }
BillItem { Name = Sebadas, Price = 8.75, Unit = , Quantity = 2 }
Total: 68.50
Customer: 41
BillItem { Name = Espresso, Price = 3, Unit = 30ml, Quantity = 2 }
BillItem { Name = Vitello Tonnato, Price = 20, Unit = , Quantity = 2 }
BillItem { Name = Brownies, Price = 9, Unit = , Quantity = 2 }
Total: 64
Customer: 76
BillItem { Name = Fritto misto, Price = 9.50, Unit = 350g, Quantity = 2 }
BillItem { Name = Melanzane alla parmigiana, Price = 12.55, Unit = , Quantity = 2 }
BillItem { Name = Apple Pie, Price = 8.50, Unit = , Quantity = 2 }
Total: 61.10
Customer: 38
BillItem { Name = Coniglio in agrodolce, Price = 15.55, Unit = 250g, Quantity = 2 }
BillItem { Name = Vitello Tonnato, Price = 20, Unit = , Quantity = 1 }
BillItem { Name = Chocolate Coffee Truffle, Price = 9, Unit = , Quantity = 1 }
Total: 60.10
```


Exercise 4 [1.5 Points]:

Write a query to list down Dishes (FoodItems projected into Dishes) that are **NOT** sold at a given table. Ordering according to the dish price.

Method signature:

```
public static IQueryable<Dish> Q4(ExamContext db, int tableNumber)
```

The record **Dish** can be found in the file **DataFormats.cs**

Sample Result:

```
Table number: 8
Correct number of elements: True -> found: 8 | correct: 8
Correct elements retrieved: True
Retrieved records:
Dish { Name = Pardula, Price = 3, Unit = }
Dish { Name = Espresso, Price = 3, Unit = 30ml }
Dish { Name = Cappuccino, Price = 3.45, Unit = 100ml }
Dish { Name = Samosa, Price = 5.50, Unit = 100g }
Dish { Name = Cannoli, Price = 7.25, Unit = }
Dish { Name = Fritto misto, Price = 9.50, Unit = 350g }
Dish { Name = Aubergine, Price = 9.50, Unit = }
Dish { Name = Meatballs, Price = 12, Unit = }
```

Exercise 5 [2 Points]:

Produce a categorised list containing Dishes (FoodItems projected into Dishes) belonging to each sub category,

HINT: A sub category is the one whose [Main] CategoryID is not null.

HINT: find first those (sub) categories and the corresponding MainCategory (CategoryID attribute).

Including (Sub) Category and MainCategory (DishWithCategories records). **Self Join**

HINT: Don't forget to add a category even if there is no food item for the given category. **Outer Join**

Method signature:

```
public static IQueryable<DishWithCategories> Q5(ExamContext db)
```

The record **DishWithCategories** can be found in the file **DataFormats.cs**

```
public record DishWithCategories
{
    public string MainCategory { get; set; } = null!;
    public string CategoryName { get; set; } = null!;
    public Dish? Food { get; set; } = null!;
}
```

Sample Result:

```
Correct number of elements: True -> found: 6 | correct: 6
Correct elements retrieved: True
Retrieved records:
DishWithCategories { MainCategory = Beverages, CategoryName = Milk Shake, Food = Dish { Name = Mango Lassi, Price = 2, Unit = } }
DishWithCategories { MainCategory = Beverages, CategoryName = Hot Drinks, Food = Dish { Name = Tea, Price = 2.50, Unit = 250ml } }
DishWithCategories { MainCategory = Beverages, CategoryName = Hot Drinks, Food = Dish { Name = Espresso, Price = 3, Unit = 30ml } }
DishWithCategories { MainCategory = Beverages, CategoryName = Hot Drinks, Food = Dish { Name = Cappuccino, Price = 3.45, Unit = 100ml } }
DishWithCategories { MainCategory = Beverages, CategoryName = Soft Drinks, Food = Dish { Name = Chinotto, Price = 3.50, Unit = 300ml } }
DishWithCategories { MainCategory = Beverages, CategoryName = Soft Drinks, Food = Dish { Name = Water, Price = 1.65, Unit = 1liter } }
Correct number of elements: True -> found: 5 | correct: 5
Correct elements retrieved: True
Retrieved records:
DishWithCategories { MainCategory = Starter, CategoryName = Sea Food Starter, Food = Dish { Name = Fritto misto, Price = 9.50, Unit = 350g } }
DishWithCategories { MainCategory = Starter, CategoryName = Vegetarian Starter, Food = Dish { Name = Samosa, Price = 5.50, Unit = 100g } }
DishWithCategories { MainCategory = Starter, CategoryName = Vegetarian Starter, Food = Dish { Name = Corn Kebab, Price = 6.50, Unit = 100g } }
DishWithCategories { MainCategory = Starter, CategoryName = Vegetarian Starter, Food = Dish { Name = Chicken Tikka, Price = 10.50, Unit = 300g } }
DishWithCategories { MainCategory = Starter, CategoryName = Meat Starter, Food = }
```

Exercise 6 [1.5 Points]:

This method returns the **number of records changed** in the DB after these operations have been applied:
Create TWO customers and add them.

The Name of the customer, ID, TableNumber, and Datetime are almost (ID has to be unique) fully optional.

Each of the two customers will place TWO orders (per customer) for fooditems belonging to the two categories.

So customer one:

order1{customer1, firstCategory product1}; order2{customer1, secondCategory product2}

customer two:

order3{customer2, firstCategory product3}; order4{customer2, secondCategory product4}

One or both given categories might NOT exist, in this case make sure an order is then **not** placed, the two customers should be added anyways.

Method signature:

```
public static int Q6(ExamContext db, string firstCategory, string secondCategory)
```

Sample Result:

1)

```
Categories: <None>, Hot Drinks
records actually updated: 4
Correct number of customers Added
Correct number of orders Added
Correct number of orders Added for all categories!
Correct number of orders Added for category Hot Drinks
```

2)

```
Categories: Hot Drinks, Vegetarian
records actually updated: 6
Correct number of customers Added
Correct number of orders Added
Correct number of orders Added for all categories!
Correct number of orders Added for category Hot Drinks
Correct number of orders Added for category Vegetarian
```

Exercise 7 [1.5 Points]:

For this question it is advisable to use a second .Net Project

Create a new entity named MainCategory (table MainCategories), with the following fields:

ID: a numeric primary key.

Name: a string, do not allow empty values.

Modify Categories table, change the (now) *self-referencing* foreign key “CategoryID”, such that it references ID of table MainCategories, to indicate to which MainCategory each Category corresponds. Set the delete behaviour to CASCADE.

Create a new entity named Employee (table Employees), with the following fields:

ID: a numeric primary key.

Name: maximum up to 50 characters, do not allow empty values.

Modify Orders table, add EmployeeID as foreign key referencing Employees table to indicate who took care of certain Orders, Set the delete behaviour to CASCADE.