# Databases Advanced Retake Exam – 11 April 2023

Exam problems for the [Databases Advanced - Entity Framework course @ SoftUni](https://softuni.bg/trainings/3966/entity-framework-core-february-2023).   
Submit your solutions in the **SoftUni Judge** system (delete all **bin**/**obj** and **packages** folders) [here](https://judge.softuni.org/Contests/3989/CSharp-DB-Advanced-Retake-Exam-11-April-2023).

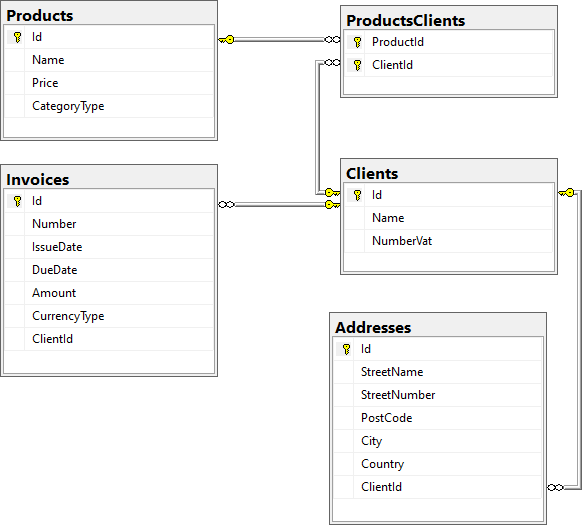
Before submitting your solutions in the **SoftUni Judge** system, delete all **bin**/**obj** and **packages** folders. If the **zip** file is still too large, you can delete the **ImportResults**, **ExportsResults** and **Datasets** folders too.

Your task is to create a **database application**, using **Entity Framework Core,** using the **Code First** approach. Design the **domain models** and **methods** for manipulating the data, as described below.

**NOTE:** Don't forget that it's a good practice when implementing a **collection** to write your code orientied towards the **interface**, not the implementation.

**NOTE**: If you want to use AutoMapper, don't forget to go to the **methods** of the **Deserializer** and/or **Serializer** classes, in which you want to use automapping, and initialize the **MapperConfiguration**.

# Invoices



## Project Skeleton Overview

You are given a **project skeleton**, which includes the following folders:

* **Data** – contains the **InvoicesContext** class, **Models** folder which contains the **entity classes,**and the **Configuration**class with **the connection string**
* DataProcessor – contains the Deserializer and Serializerclasses, which are used for **importing** and **exporting** data
* Datasets – contains the .json and .xml files for the import part
* ImportResults – contains the **import** results you make in the Deserializer class
* ExportResults – contains the **export** results you make in the Serializer class

## Model Definition (50 pts)

The application needs to store the following data:

### Product

* Id – **integer**, **Primary Key**
* Name – **text** with length **[9…30]** **(required)**
* **Price** – **decimal** in range **[5.00…1000.00]** **(required)**
* CategoryType– **enumeration** of type CategoryType, with possible values **(**ADR, Filters, Lights, Others, Tyres**)** **(required)**
* ProductsClients – collection of type ProductClient

### Address

* Id – **integer**, **Primary Key**
* StreetName – **text** with length **[10…20]** **(required)**
* StreetNumber – **integer (required)**
* PostCode – **text** **(required)**
* City – **text** with length **[5…15]** **(required)**
* Country – **text** with length **[5…15]** **(required)**
* ClientId– integer, foreign key (required)
* Client– Client

### Invoice

* Id – **integer**, **Primary Key**
* Number – **integer** in range **[1,000,000,000…1,500,000,000]** **(required)**
* IssueDate – **DateTime** **(required)**
* DueDate – **DateTime** **(required)**
* Amount – decimal (required)
* CurrencyType – **enumeration** of type CurrencyType, with possible values **(**BGN, EUR, USD**)** **(required)**
* ClientId– integer, foreign key (required)
* Client– Client

### Client

* Id – integer, **Primary Key**
* Name – **text** with length **[10…25]** **(required)**
* NumberVat – **text** with length **[10…15]** **(required)**
* Invoices – collection of type Invoicе
* Addresses – collection of type Address
* ProductsClients – collection of type ProductClient

### ProductClient

* ProductId– integer, Primary Key, foreign key (required)
* Product– Product
* ClientId– integer, Primary Key, foreign key (required)
* Client – Client

## Data Import (25pts)

For the functionality of the application, you need to create several methods that manipulate the database. The **project skeleton** already provides you with these methods, inside the Deserializer class.

**NOTE:** Usage of DataTransferObjects and **AutoMapper** is **optional**.

Use the provided **JSON** and **XML** files to populate the database with data. Import all the information from those files into the database.

You are **not allowed** to modify the provided **JSON** and **XML** files.

**If a record does not meet the requirements from the first section, print an error message:**

|  |
| --- |
| **Error message** |
| Invalid data! |

### XML Import

#### Import Clients

Using the file **"clients.xml"**, import the data from the file into the database. Print information about each imported object in the format described below.

##### Constraints

* If there are **any validation errors** for the **client** entity (such as **invalid name or vat number**), **do not** import any part of the entity and **append an error message** to the **method output**.
* If there are **any validation errors** for the **address** entity (such as invalid or null or empty **street name**, **invalid street number, invalid or missing post code, city or country**), **do not import it (only the address itself, not the whole client info)** and **append an error message to the method output**.

|  |
| --- |
| **Success message** |
| Successfully imported {**clientName**}. |

##### Example

|  |
| --- |
| **clients.xml** |
| <?xml version="1.0" encoding="UTF-8" ?>  <Clients>  <Client>  <Name>LiCB</Name>  <NumberVat>BG5464156654654654</NumberVat>  <Addresses>  <Address>  <StreetName>Gnigler strasse</StreetName>  <StreetNumber>57</StreetNumber>  <PostCode>5020</PostCode>  <City>Salzburg</City>  <Country>Austria</Country>  </Address>  </Addresses>  </Client>  …  </Clients> |
| **Output** |
| Invalid data!  Invalid data!  Invalid data!  Invalid data!  Invalid data!  Successfully imported client SPEDOX,SRO.  ... |

Upon **correct import logic**, you should have imported **29 clients**.

### JSON Import

#### Import Invoices

Using the file "**invoices.json**", import the data from the file into the database. Print information about each imported object in the format described below.

##### Constraints

* If there are any validation errors (such as invalid **issue or due date, due date is before issue date, invalid amount, currency type or client**), **do not import** **any part of the entity** and **append an error message to the method output**.

**NOTE**: Do not forget to use **CultureInfo.InvariantCulture.**

|  |
| --- |
| **Success message** |
| Successfully imported invoice with number {**invoiceNumber**}. |

##### Example

|  |
| --- |
| **invoices.json** |
| [  {  "Number": 1427940691,  "IssueDate": "2022-08-29T00:00:00",  "DueDate": "2022-10-28T00:00:00",  "Amount": 913.13,  "CurrencyType": 1,  "ClientId": 1  },  {  "Number": 142796902,  "IssueDate": "2022-08-31T00:00:00",  "DueDate": "2022-10-30T00:00:00",  "Amount": 891.76,  "CurrencyType": 2,  "ClientId": 2  },  {  "Number": 1427940690,  "IssueDate": "2022-09-05T00:00:00",  "DueDate": "2022-11-04T00:00:00",  "Amount": 704.48,  "CurrencyType": 3,  "ClientId": 3  },  …  ] |
| **Output** |
| Successfully imported invoice with number 1427940691.  Invalid data!  Invalid data!  Invalid data! |

Upon **correct import logic**, you should have imported **55 invoices**.

#### Import Products

Using the file "**products.json**", import the data from the file into the database. Print information about each imported object in the format described below.

##### Constraints

* If there are any validation errors (such as invalid product name, invalid price or category type), **do not import** **any part of the entity** and **append an error message to the method output**.
* Take only unique clients.
* If a **client** does **not exist** in the database, **append an error message** to the **method** **output** and **continue** with the next **client**.

|  |
| --- |
| **Success message** |
| Successfully imported product - {**productName**} with {**clientsCount**} clients. |

##### Example

|  |
| --- |
| **products.json** |
| [  {  "Name": "ADR plate",  "Price": 14.97,  "CategoryType": 1,  "Clients": [  1,  105,  1,  5,  15  ]  },  {  "Name": "ADR light",  "Price": 21.25,  "CategoryType": 1,  "Clients": [  1,  85,  81,  80,  5,  9  ]  },  …  ] |
| **Output** |
| Invalid data!  Successfully imported product - ADR plate with 3 clients.  Invalid data!  Invalid data!  Invalid data!  Successfully imported product - ADR light with 3 clients. |

Upon **correct import logic**, you should have imported **91 products** with **137 clients**.

## Data Export (25 pts)

**Use the provided methods in the** Serializer class**.** Usage of **Data Transfer Objects and Automapper** is **optional**.

### JSON Export

#### Export Products With Most Clients

Select the **top** **5 products** that havebeen **sold to at least one client, where the client's name is at least as long as the given number. Select** them with their **clients** who meet the **same criteria** (the client's name is at least as long as the given number). For each **product**, export their **name, price, category type** and their **clients.** For each **client**, export their **name** and **vat number.** Order the **clients** by **name** (**ascending**). Order the **products** by **all** **clients** (meeting above condition) **count** (**descending**), then by **name** (**ascending**).

**NOTE**: You **may** need to **call** **.ToArray()** function **before the selection** in order to **detach entities from the database** and **avoid runtime errors** (**EF Core bug**).

##### Example

|  |
| --- |
| **Serializer.ExportProductsWithMostClients(context, nameLength)** |
| [  {  "Name": "MAHLE KX400KIT",  "Price": 26.13,  "Category": "Tyres",  "Clients": [  {  "Name": "BTS GMBH CO KG",  "NumberVat": "DE814592224"  },  {  "Name": "DPS EUROPE AB",  "NumberVat": "SE556488676901"  },  {  "Name": "FREIGHTS PC",  "NumberVat": "EL801106064"  },  {  "Name": "KAMEEN LOGISTIC KG",  "NumberVat": "ATU75339778"  },  …  },  …  ] |

### XML Export

#### Export Clients with Their Invoices

Export all **clients** that have at least one issued **invoices**, issued after the given date. For each **client**, export their **name, vat number** and **invoices count**. For each **invoice**, export its **number, amount, currency** and **due date.** Order the **invoices** by **due** by **issue date** (**ascending**), then **date** (**descending**). Order the **clients** by **invoices count** (**descending**), then by **name** (**ascending**).

**NOTE**: You **may** need to **call** **.ToArray()** function **before the selection,** in order to **detach entities from the database** and **avoid runtime errors** (**EF Core bug**).

**NOTE**: Do not forget to use **CultureInfo.InvariantCulture.** Use formatting (**"d"**).

##### Example

|  |
| --- |
| Serializer.ExportClientsWithTheirInvoices(context, date) |
| <?xml version="1.0" encoding="utf-16"?>  <Clients>  <Client InvoicesCount="9">  <ClientName>SPEDOX,SRO</ClientName>  <VatNumber>SK2023911087</VatNumber>  <Invoices>  <Invoice>  <InvoiceNumber>1063259096</InvoiceNumber>  <InvoiceAmount>167.22</InvoiceAmount>  <DueDate>02/19/2023</DueDate>  <Currency>EUR</Currency>  </Invoice>  <Invoice>  <InvoiceNumber>1427940691</InvoiceNumber>  <InvoiceAmount>913.13</InvoiceAmount>  <DueDate>10/28/2022</DueDate>  <Currency>EUR</Currency>  </Invoice>  …  </Invoices>  …  </Client>  …  </Clients> |