Databases Advanced Exam – 01 April 2023

Exam problems for the Databases Advanced - Entity Framework course @ SoftUni.
Submit your solutions in the **SoftUni Judge** system (delete all **bin/obj** and **packages** folders) here.

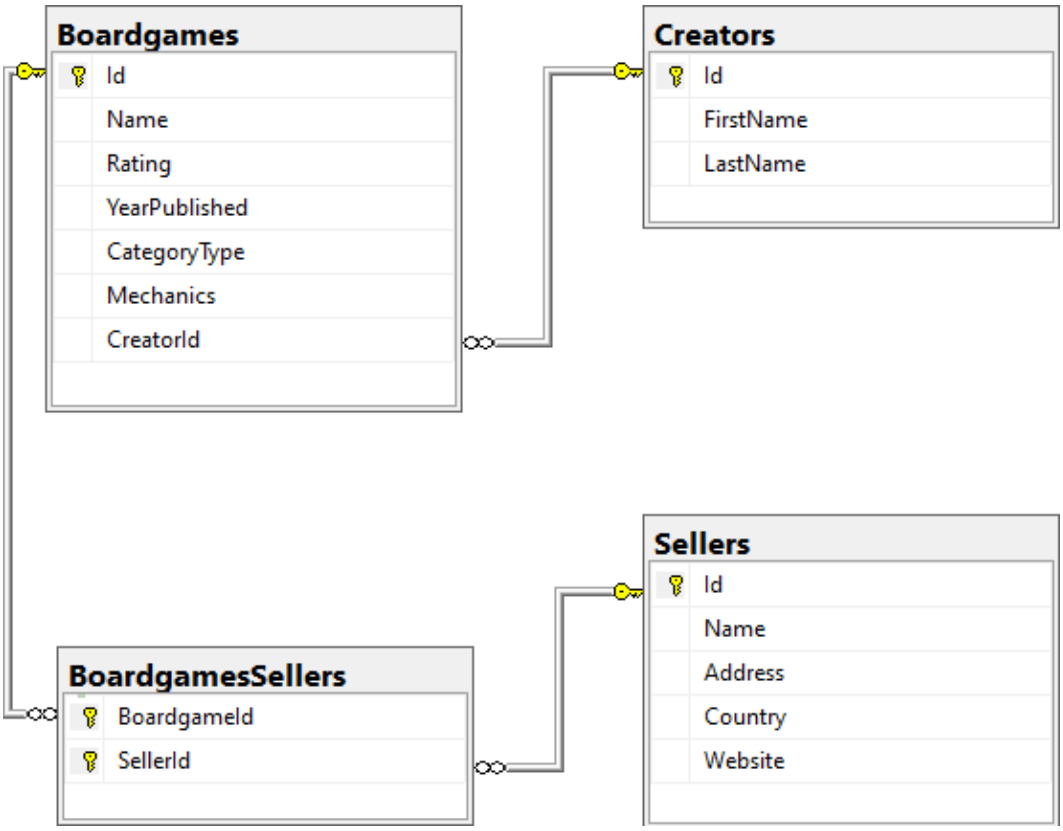
Before submitting your solutions in the **SoftUni Judge** system, delete all **bin/obj** and **packages** folders. If the **zip** file is still too large, you can delete the **ImportResults**, **ExportsResults** and **Datasets** folders too.

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**.

Boardgames



1. Project Skeleton Overview

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

- **Data** – contains the **BoardgamesContext** class, **Models** folder, which contains the **entity classes** and the **Configuration** class with **connection string**
- **DataProcessor** – contains the **Serializer** and **Deserializer** classes, 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

2. Model Definition (50 pts)

The application needs to store the following data:

Boardgame

- **Id** – integer, **Primary Key**
- **Name** – text with length **[10…20]** (**required**)
- **Rating** – **double** in range [**1…10.00**] (**required**)
- **YearPublished** – integer in range [**2018…2023**] (**required**)
- **CategoryType** – enumeration of type **CategoryType**, with possible values **(Abstract**, **Children**, **Family**, **Party, Strategy)** (**required**)
- **Mechanics** – text (**string**, **not** an array) (**required**)
- **CreatorId** – **integer**, **foreign key** (**required**)
- **Creator** – **Creator**
- **BoardgamesSellers** – collection of type **BoardgameSeller**

Seller

- **Id** – integer, Primary Key
- **Name** – **text** with length **[5…20]** (**required**)
- **Address** – **text** with length **[2…30]** (**required**)
- **Country** – **text** (**required**)
- **Website** – a **string** (**required**). First four characters are "**www.**", followed by upper and lower **letters**, **digits** or '**-**' and the last three characters are "**.com**".
- **BoardgamesSellers** – collection of type **BoardgameSeller**

Creator

- **Id** – integer, **Primary Key**
- **FirstName** – **text with length [2, 7] (required)**
- **LastName** – **text with length [2, 7] (required)**
- **Boardgames** – collection of type **Boardgame**

BoardgameSeller

- **BoardgameId** – integer, **Primary Key**, **foreign key** (**required**)
- **Boardgame** – **Boardgame**
- **SellerId** – integer, **Primary Key**, **foreign key** (**required**)
- **Seller** – **Seller**

3. 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 **Data Transfer Objects** 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 Creators

Using the file "**`creators.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 **creator** entity (such as **invalid first and last names**), **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 **boardgame** entity (such as invalid or null or empty **name**, **publishing year is** invalid, **rating is** invalid), **do not import it (only the boardgame itself, not the whole creator info)** and **append an error message to the method output**.

Success message
`Successfully imported creator – {creatorFirstName} {creatorLastName} with {boardgamesCount} boardgames.`

Example

creators.xml

```
<?xml version='1.0' encoding='UTF-8'?>
<Creators>
  <Creator>
    <FirstName>Debra</FirstName>
    <LastName>Edwards</LastName>
    <Boardgames>
      <Boardgame>
        <Name>4 Gods</Name>
        <Rating>7.28</Rating>
        <YearPublished>2017</YearPublished>
        <CategoryType>4</CategoryType>
        <Mechanics>Area Majority / Influence, Hand Management, Set Collection,
Simultaneous Action Selection, Worker Placement</Mechanics>
      </Boardgame>
      <Boardgame>
        <Name>7 Steps</Name>
        <Rating>7.01</Rating>
        <YearPublished>2015</YearPublished>
        <CategoryType>4</CategoryType>
        <Mechanics>Action Queue, Hand Management, Push Your Luck, Set
Collection</Mechanics>
      </Boardgame>
      …
    </Boardgames>
  </Creator>
```

```
…
</Creators>
```

Output
Invalid data! Invalid data! Invalid data! Invalid data! Invalid data! Invalid data! Invalid data! Invalid data! Invalid data! Successfully imported creator - Debra Edwards with 4 boardgames. Invalid data! Invalid data! …

Upon **correct import logic**, you should have imported **19 creators** and **81 boardgames**.

JSON Import

Import Sellers

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

Constraints

- If any validation errors occur (such as invalid **name**, missing or invalid **country, website and/or address**), **do not** import any part of the entity and **append an error message** to the **method output**.
- Take only the unique boardgames.
- If a **boardgame** does **not exist** in the database, **append an error message** to the **method output** and **continue** with the next **boardgame**.

Success message
Successfully imported seller - {**sellerName**} with {**boardgamesSellersCount**} boardgames.

Example

sellers.json

```
[
 {
    "Name": "6am",
    "Address": "The Netherlands",
    "Country": "Belgium",
    "Website": "www.6pm.com",
    "Boardgames": [
       1,
       105,
       1,
       5,
       15
    ]
  },
  {
```

```
    "Name": "Asurion, LLC",
    "Address": "P.O. Box 234,  38-54",
    "Country": "Belgium",
     "Website": "www.asurion-llc.com",
    "Boardgames": [
       1,
       85,
       81,
       80,
       5,
       9
    ]
  },
  …
]
```

Output

```
Invalid data!
Invalid data!
Successfully imported seller - Asurion, LLC with 5 boardgames.
Successfully imported seller - Bedsure with 6 boardgames.
Invalid data!
Invalid data!
Invalid data!
...
```

Upon **correct import logic**, you should have imported **11 sellers** and **59 boardgames**.

4. Data Export (25 pts)

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

JSON Export

Export Sellers With Most Boardgames

Select the **top 5 sellers** that have **at least one boardgame** that **their year of publishing** is greater or equal to the **given year** and **their rating** is smaller or equal to the **given rating. Select** them with their **boardgames** who meet the **same criteria** (their year of publishing is greater or equals the given year and the rating is smaller or equal to the given rating). For each **seller**, export their **name, website** and their **boardgames.** For each **boardgame**, export their **name**, **rating, mechanics** and **category** type**.** Order the **boardgames** by **rating** (**descending**), then by **name** (**ascending**). Order the **sellers** by **all boardgames** (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.ExportSellersWithMostBoardgames(context, year, rating)`

```
[
  {
    "Name": "Bedsure",
    "Website": "www.bedsure.com",
    "Boardgames": [
      {
        "Name": "The Fog of War",
```

```
        "Rating": 9.65,
        "Mechanics": "Grid Movement, Hand Management, Rock-Paper-Scissors, Time
Track, Variable Player Powers",
        "Category": "Strategy"
      },
      {
        "Name": "Capital Lux",
        "Rating": 7.58,
        "Mechanics": "Grid Movement, Tile Placement",
        "Category": "Abstract"
      },
      {
        "Name": "King's Road",
        "Rating": 7.48,
        "Mechanics": "Card Drafting, End Game Bonuses, Memory, Set Collection,
Simultaneous Action Selection",
        "Category": "Strategy"
      },
      {
        "Name": "Imperial Struggle",
        "Rating": 7.19,
        "Mechanics": "Card Drafting, Dice Rolling, Drafting, Set Collection,
Simultaneous Action Selection",
        "Category": "Family"
      },
      {
        "Name": "Nerdy Inventions",
        "Rating": 7.1,
        "Mechanics": "Hand Management, Pattern Building",
        "Category": "Abstract"
      },
      {
        "Name": "Star Wars: Rebellion",
        "Rating": 6.19,
        "Mechanics": "Action Queue, Modular Board",
        "Category": "Abstract"
      }
    ]
  },
  …
]
```


XML Export

Export Creators with Their Boardgames

Export all **creators** that have created at least **one** boardgame. For each **creator**, export their **name** and **boardgames count**. For each **boardgame**, export their **full name** and **year of publishing.** Order the **boardgames** by **name** (**ascending**). Order the **creators** by **boardgames 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.ExportCreatorsWithTheirBoardgames(context)`
`<?xml version="1.0" encoding="utf-16"?>`

```xml
<Creators>
  <Creator BoardgamesCount="6">
    <CreatorName>Cade O'Neill</CreatorName>
    <Boardgames>
      <Boardgame>
        <BoardgameName>Bohnanza: The Duel</BoardgameName>
        <BoardgameYearPublished>2019</BoardgameYearPublished>
      </Boardgame>
      <Boardgame>
        <BoardgameName>Great Western Trail</BoardgameName>
        <BoardgameYearPublished>2018</BoardgameYearPublished>
      </Boardgame>
      <Boardgame>
        <BoardgameName>Indulgence</BoardgameName>
        <BoardgameYearPublished>2021</BoardgameYearPublished>
      </Boardgame>
      <Boardgame>
        <BoardgameName>Risk Europe</BoardgameName>
        <BoardgameYearPublished>2018</BoardgameYearPublished>
      </Boardgame>
      <Boardgame>
        <BoardgameName>The Grimm Forest</BoardgameName>
        <BoardgameYearPublished>2022</BoardgameYearPublished>
      </Boardgame>
      <Boardgame>
        <BoardgameName>Whitehall Mystery</BoardgameName>
        <BoardgameYearPublished>2023</BoardgameYearPublished>
      </Boardgame>
    </Boardgames>
  </Creator>
   …
</Creators>
```