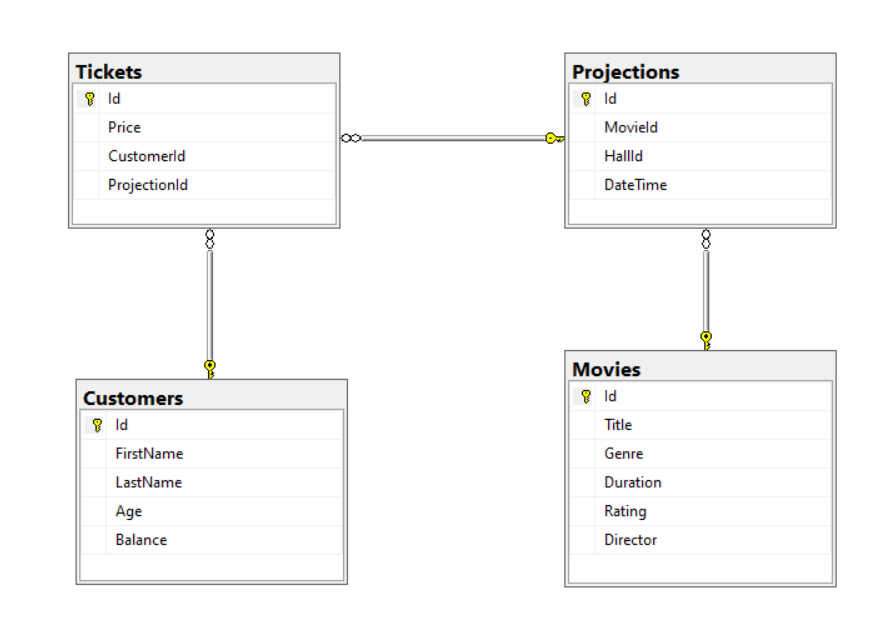
# Databases Advanced Exam – 11 April 2021

Exam problems for the [Databases Advanced - Entity Framework course @ SoftUni](https://softuni.bg/courses/entity-framework-core). Submit your solutions in the **SoftUni judge** system (delete all "**bin**"/"**obj**" and "**packages**" folders).

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.

# Cinema



## Project Skeleton Overview

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

* Data – contains the CinemaContext class, Models folder which contains the **entity classes** and the **Configuration** class with **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)

***Note: Foreign key navigation properties are required!***

The application needs to store the following data:

### Movie

* Id – integer, **Primary Key**
* Title – **text with length [3, 20]** **(required)**
* Genre – **enumeration** of type Genre, with possible values **(****Action, Drama, Comedy, Crime, Western, Romance, Documentary, Children, Animation, Musical)** **(required)**
* Duration – **TimeSpan (required)**
* Rating – double **in the range** [1,10] (required)
* Director – **text with length [3, 20]** **(required)**
* Projections - collection of type Projection

### Projection

* Id – integer, **Primary Key**
* MovieId – integer, Foreign key (required)
* Movie – the Projection’s **Movie**
* **DateTime -** DateTime **(required)**
* Tickets - collection of type Ticket

### Customer

* Id – integer, **Primary Key**
* FirstName – **text with length [3, 20]** **(required)**
* LastName – **text with length [3, 20]** **(required)**
* Age – **integer** **in the range [12, 110]** **(required)**
* **Balance** - **decimal** (non-negative, minimum value: **0.01**) **(required)**
* Tickets - collection of type Ticket

### Ticket

* Id – integer, **Primary Key**
* Price – **decimal** (non-negative, minimum value: **0.01**) **(required)**
* CustomerId – integer, Foreign key (required)
* Customer – the Ticket’s **Customer**
* ProjectionId – integer, Foreign key (required)
* Projection – the Ticket’s **Projection**

## 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**. Usage of **Data Transfer Objects** 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:**

**За функционалността на приложението трябва да създадете няколко метода, които манипулират базата данни. Скелетът на проекта вече ви предоставя тези методи, в класа на Deserializer. Използването на обекти за прехвърляне на данни е по избор. Използвайте предоставените JSON и XML файлове, за да попълните базата данни с данни. Импортирайте цялата информация от тези файлове в базата данни. Нямате право да променяте предоставените JSON и XML файлове. Ако даден запис не отговаря на изискванията от първия раздел, отпечатайте съобщение за грешка:**

|  |
| --- |
| **Error message** |
| Invalid Data! |

### JSON Import

#### Import Movies

Using the file movies.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 if Rating is not between 1 and 10, a **Title/Genre/Duration/Rating/Director** is missing, or they exceed required the min and max length), **do not** import any part of the entity and **append an error message** to the **method output**.
* If a movie with this **title** already exists, do not import it and append an error message.
* **Durations** will always be in the format "**c**". Do not forget to use **CultureInfo.InvariantCulture**!
* When **printing** the output **Ratings** should be in format "**f2**".

JSON импортиране на филми с помощта на файла movies.json, импортирайте данните от този файл в базата данни. Печат на информация за всеки импортиран обект във формата, описан по-долу. Ограничения,

ако се появят грешки при проверка (например ако оценка не е между 1 и 10, title/Genre/продължителност/оценка/директор или те надвишават изисква мин и максимална дължина), не импортирайте част от обекта и добавите съобщение за грешка на метода изходния метод.

Ако вече съществува филм с това заглавие, не го импортирайте и прикани съобщение за грешка. Продължителността винаги ще бъде във формат "c". Не забравяйте да използвате културатаInfo.invariantКултура!

При отпечатване на изходните оценки трябва да бъде във формат "f2".

|  |
| --- |
| **Success message** |
| Successfully imported {**movie title**} with genre {**movie genre**} and rating {**movie rating**}! |

##### Example

|  |
| --- |
| **movies.json** |
| [  {  "Title": "Little Big Man",  "Genre": "Western",  "Duration": "01:58:00",  "Rating": 28,  "Director": "Duffie Abrahamson"  },  {  "Title": "Gui Si (Silk)",  "Genre": "Drama",  "Duration": "02:21:00",  "Rating": 9,  "Director": "Perl Swyne"  },  {  "Title": "F",  "Genre": "Drama",  "Duration": "01:08:00",  "Rating": 2,  "Director": "Emiline Newby"  },  {  "Title": "F/X2 (a.k.a. F/X 2 - The Deadly Art of Illusion)",  "Genre": "Action",  "Duration": "01:57:00",  "Rating": 7,  "Director": "Sheppard Cescoti"  },  ...  ] |
| **Output** |
| **Invalid data!**  **Successfully imported Gui Si (Silk) with** **genre Drama and rating 9.00!**  **Invalid data!**  **Invalid data!**  **Successfully imported Prey, The (La proie) with genre Action and rating 5.00!**  **Successfully imported SIS with genre Action and rating 10.00!**  **...** |

Upon **correct import logic**, you should have imported **21** **movies**.

### XML Import

#### Import Projections

Using the file **projections.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 (such as invalid **movie**), **do not import** **any part of the entity** and **append an error message to the method output**.
* Dates will **always** be in the format: "yyyy-MM-dd HH:mm:ss". Do not forget to use **CultureInfo.InvariantCulture**!
* In the output **Projection Datetime** is in format "**MM/dd/yyyy**".

С помощта на файловите прогнози.xml импортирайте данните от файла в базата данни. Печат на информация за всеки импортиран обект във формата, описан по-долу.

Ограничения Ако има грешки при проверка (например невалиден филм), не импортирайте част от обекта и добавяйте съобщение за грешка към изходния метод. Датите винаги ще бъдат във формат: "гггг-ММ-дд HH:mm:ss". Не забравяйте да използвате културатаInfo.invariantКултура! В изхода проекцията Datetime е във формат "ММ/дд/гг".

|  |
| --- |
| **Success message** |
| Successfully imported projection {**movie title**} on {**projection datetime**}! |

##### Example

|  |
| --- |
| **projections.xml** |
| <Projections>  <Projection>  <MovieId>38</MovieId>  <DateTime>2019-04-27 13:33:20</DateTime>  </Projection>  <Projection>  <MovieId>6</MovieId>  <DateTime>2019-05-12 05:51:29</DateTime>  </Projection>  <Projection>  <MovieId>21</MovieId>  <DateTime>2019-05-03 16:56:12</DateTime>  </Projection>  <Projection>  <MovieId>10</MovieId>  <DateTime>2019-05-01 00:11:21</DateTime>  </Projection>  <Projection>  <MovieId>40</MovieId>  <DateTime>2019-04-26 08:56:57</DateTime>  </Projection>  ...  </Projections> |
| **Output** |
| **Invalid data!**  **Successfully imported projection Trojan Eddie on** **05/12/2019!**  **Successfully imported projection Gloriously Wasted on 05/03/2019!**  **Successfully imported projection Fahrenhype 9/11 on 05/01/2019! Invalid data!**  **...** |

Upon **correct import logic**, you should have imported **32 projections**.

#### Import Customers Tickets

Using the file **customers-tickets.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 (such invalid **names**, **age**, **balance**, etc.), **do not import** **any part of the entity** and **append an error message to the method output**.
* If there are any validation errors in **tickets** (such invalid **price**), **do not import the ticket entity itself** and **append an error message to the method output**.

|  |
| --- |
| **Success message** |
| Successfully imported customer {**customer first name**} {**customer last name**} with bought tickets: {**tickets count**}! |

##### Example

|  |
| --- |
| **customers-tickets.xml** |
| <Customers>  <Customer>  <FirstName>Randi</FirstName>  <LastName>Ferraraccio</LastName>  <Age>20</Age>  <Balance>59.44</Balance>  <Tickets>  <Ticket>  <ProjectionId>1</ProjectionId>  <Price>7</Price>  </Ticket>  <Ticket>  <ProjectionId>1</ProjectionId>  <Price>15</Price>  </Ticket>  <Ticket>  <ProjectionId>1</ProjectionId>  <Price>12.13</Price>  </Ticket>  <Ticket>  <ProjectionId>1</ProjectionId>  <Price>11</Price>  </Ticket>  <Ticket>  <ProjectionId>1</ProjectionId>  <Price>9.13</Price>  </Ticket>  <Ticket>  <ProjectionId>1</ProjectionId>  <Price>9.13</Price>  </Ticket>  </Tickets>  </Customer>  <Customer>  <FirstName>Duff</FirstName>  <LastName>Honig</LastName>  <Age>89</Age>  <Balance>82.17</Balance>  <Tickets>  <Ticket>  <ProjectionId>25</ProjectionId>  <Price>12.13</Price>  </Ticket>  <Ticket>  <ProjectionId>25</ProjectionId>  <Price>11</Price>  </Ticket>  <Ticket>  <ProjectionId>25</ProjectionId>  <Price>9.13</Price>  </Ticket>  <Ticket>  <ProjectionId>3</ProjectionId>  <Price>9.13</Price>  </Ticket>  <Ticket>  <ProjectionId>3</ProjectionId>  <Price>9.13</Price>  </Ticket>  <Ticket>  <ProjectionId>3</ProjectionId>  <Price>9.13</Price>  </Ticket>  </Tickets>  </Customer>  <Customer>  <FirstName>Bondy</FirstName>  <LastName>Linsay</LastName>  <Age>15</Age>  <Balance>230.20</Balance>  <Tickets>  <Ticket>  <ProjectionId>3</ProjectionId>  <Price>12.13</Price>  </Ticket>  <Ticket>  <ProjectionId>3</ProjectionId>  <Price>11</Price>  </Ticket>  <Ticket>  <ProjectionId>3</ProjectionId>  <Price>9.13</Price>  </Ticket>  <Ticket>  <ProjectionId>3</ProjectionId>  <Price>7</Price>  </Ticket>  </Tickets>  </Customer>  ...  </Customers> |
| **Output** |
| **Successfully imported customer Randi Ferraraccio with bought tickets: 6!**  **Successfully imported customer Duff Honig with bought tickets: 6!**  **Successfully imported customer Bondy Linsay with bought tickets: 4!**  **...** |

Upon **correct import logic**, you should have imported **79 customers and 168 tickets**.

## Data Export (25 pts)

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

### JSON Export

#### Export Top Movies

The given method in the project skeleton receives movie rating. Export **top 10** **movies** which have rating more or equal to the given and **have at least one projection with sold tickets**. For each **movie**, export its **name**, **rating formatted to the second digit, total incomes formatted same way** and **customers.** For each **customer**, export its **first name**, last **name** and **balance formatted to the second digit.** Order the customers by balance (**descending by the formatted string, not the balance itselft**), then by first name (**ascending**) and last name (**ascending**). Take first 10 records ordered by **rating** (**descending**), then by **total incomes** (**descending**).

##### Example

|  |
| --- |
| **Serializer.ExportTopMovies(context, rating)** |
| [  {  "MovieName": "SIS",  "Rating": "10.00",  "TotalIncomes": "58.26",  "Customers": [  {  "FirstName": "Ray",  "LastName": "MacDearmid",  "Balance": "77.69"  },  {  "FirstName": "Ray",  "LastName": "MacDearmid",  "Balance": "77.69"  },  {  "FirstName": "Garry",  "LastName": "Blackeby",  "Balance": "123.82"  },  {  "FirstName": "Garry",  "LastName": "Blackeby",  "Balance": "123.82"  },  {  "FirstName": "Garry",  "LastName": "Blackeby",  "Balance": "123.82"  },  {  "FirstName": "Garry",  "LastName": "Blackeby",  "Balance": "123.82"  }  ]  },  ...  ] |

### XML Export

#### Export Top Customers

Use the method provided in the project skeleton, which receives customer age. Export customers with age above or equal to the given. For each **customer**, export their **first name**, **last name, spent money for tickets (formatted to the second digit) and spent time (**in format: "**hh\:mm\:ss**"**).** Take first **10** records and order the result by **spent money** in **descending order.**

##### Example

|  |
| --- |
| Serializer.TopCustomers(context, age) |
| <Customers>  <Customer FirstName="Marjy" LastName="Starbeck">  <SpentMoney>82.65</SpentMoney>  <SpentTime>12:10:00</SpentTime>  </Customer>  <Customer FirstName="Jerrie" LastName="O\'Carroll">  <SpentMoney>67.13</SpentMoney>  <SpentTime>13:20:00</SpentTime>  </Customer>  <Customer FirstName="Randi" LastName="Ferraraccio">  <SpentMoney>63.39</SpentMoney>  <SpentTime>06:36:00</SpentTime>  </Customer>  ...  </Customers> |