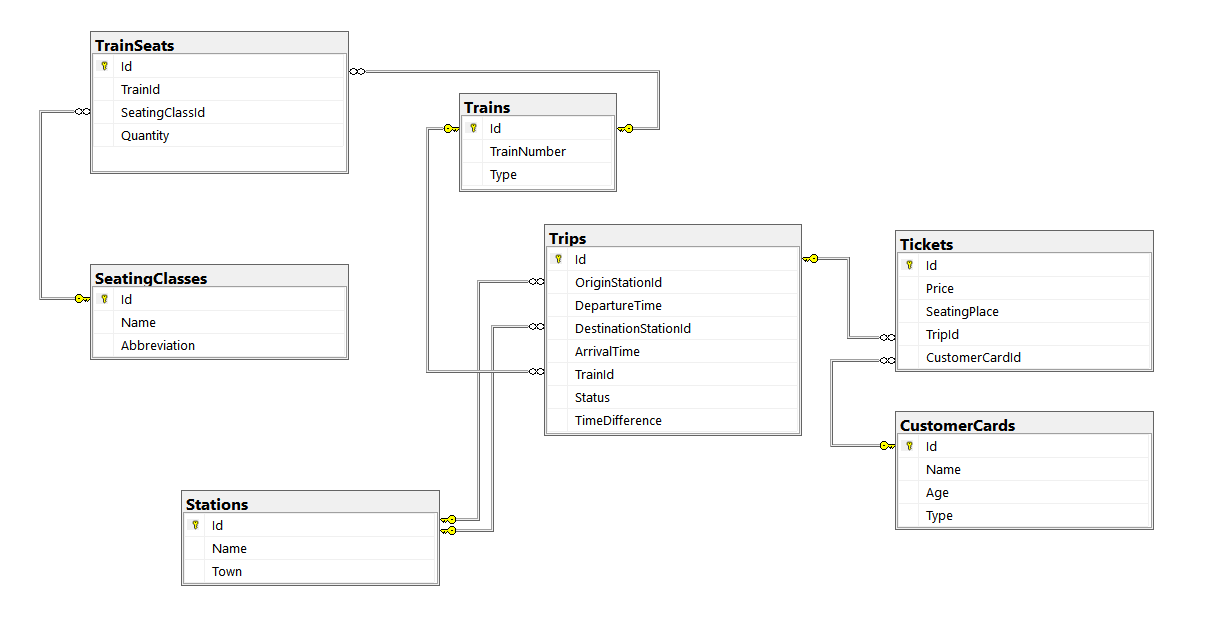
# Entity Framework Core: Exam Preparation 2

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

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.

# Stations

Create an application which models a **train transport system**. The main idea is to have **stations** between which **trains** travel. Each **travel** between two stations is calleda **trip**. People can buy **tickets** for a specific **trip**. Some people might not be registered while others may have a **personal card** (in other words – some tickets may have **buyer’s information** while others will not). For more details about the **models** and their **relations** read further.



## Project Skeleton Overview

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

* Stations.App – contains the Startup class, which is the **entry point of the application**. Also contains an AutoMapperprofile, which you can configure if you choose to use AutoMapper in your app.
* Stations.Data – contains the StationsDbContext class and the **connection string**
* Stations.Models – contains the **entity classes**
* Stations.DataProcessor – contains the Serializer and Deserializerclasses, which are used for **importing** and **exporting** data

## Problem 1. Model Definition (50 pts)

Every **trip** has an origin **station** and destination **station**, it also includes information about which **train** travelled between those two stations. Each train may have **seats**, which are grouped in **seating classes** (**First class**, **Second class** and so on). For each trip, you may buy a **ticket**. the **ticket** keeps information about the trip and **may** include information about the **customer** who bought it through their **customer card** (**if** they have one). The **customer** **card** contains basic person data.

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

The application needs to store the following data:

### Station

* Id – integer, **Primary Key**
* Name – text with **max length** **50** (**required, unique**)
* Town– text with **max length** **50** (**required**)
* TripsTo – Collection of type Trip
* TripsFrom – Collection of type Trip

### Train

* Id – integer, **Primary Key**
* TrainNumber **– text with max length 10 (required, unique)**
* Type **–** TrainType **enumeration with possible values: "HighSpeed", "LongDistance" or "Freight" (optional)**
* TrainSeats – Collection of type TrainSeat
* Trips – Collection of type Trip

### SeatingClass

* Id – integer, **Primary Key**
* Name – text with **max length 30** (**required, unique**)
* Abbreviation – text with an **exact** length of **2** (**no more, no less**), (**required, unique**)

### TrainSeat

* Id – integer, **Primary Key**
* TrainId **–** integer (required)
* Train **– train whose seats will be described (required)**
* SeatingClassId **– integer (required)**
* SeatingClass **– class of the seats (required)**
* Quantity **– how many seats of given class total for the given train (required, non-negative)**

### Trip

* Id – integer, **Primary Key**
* OriginStationId – integer **(required)**
* OriginStation – station from which the trip begins (**required**)
* DestinationStationId – integer **(required)**
* DestinationStation **–** station **where the trip ends (required)**
* DepartureTime – date and time of departure from origin station (**required**)
* ArrivalTime– date and time of arrival at destination station, must be after departure time (**required**)
* TrainId – integer **(required)**
* Train– train used for that particular trip (**required**)
* Status– TripStatus **enumeration** with possible values: "**OnTime**", "**Delayed**" and "**Early**" (**default**: "**OnTime**")
* TimeDifference **– time(span) representing how late or early a given train was (optional)**

### Ticket

* Id – integer, **Primary Key**
* Price – decimal value of the ticket (**required, non-negative**)
* SeatingPlace **– text with max length of 8 which combines seating class abbreviation plus a positive integer (required)**
* TripId – integer **(required)**
* Trip – the trip for which the ticket is for (**required**)
* CustomerCardId– **integer (optional)**
* CustomerCard **– reference to the ticket’s buyer**

### CustomerCard

* Id – integer, **Primary Key**
* Name – text with **max length** **128** (**required**)
* Age– integer **between** **0 and 120**
* Type– CardType enumeration with values: "**Normal**", "**Pupil**", "**Student**", "**Elder**", "**Debilitated**" (default: **Normal**)
* **BoughtTickets** – Collection of type Ticket

Any information which has additional requirements needs to be **validated**.

## Problem 2. Data Import (35pts)

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 Stations.DataProcessor project inside your solution. Use **Data Transfer Objects** as needed.

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 format. |

### JSON Import (15 pts)

#### 1.Import Stations

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

* If the town name is not given, insert it with the same value as the station name.
* If any other validation error occurs (such as **long** **station** **or** **town name**) proceed as described above.

##### Example

|  |
| --- |
| **stations.json** |
| [  {  "Name": "Sofia"  },  {  "Name": "Sofia Sever",  "Town": "Sofia"  },  …  ] |
| **Output** |
| Record Sofia successfully imported.  Record Sofia Sever successfully imported.  … |

#### 2.Import Seating Classes

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

If any validation error occurs proceed as described above.

##### Example

|  |
| --- |
| **classes.json** |
| [  {  Name:"First class",   Abbreviation: "FC",  },  {  Name:"Second class",   Abbreviation: "SC",  },  …  ] |
| **Output** |
| Record First class successfully imported.  Record Second class successfully imported.  … |

#### 3.Import Trains

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

##### Constraints

* Train type will always be a valid type (if not null). If it’s **null**, use the **default type** (**HighSpeed**).
* If there is no **seat class** with given name and abbreviation – **skip** thewhole entity.
* If there is seat class with same name but different abbreviation or the quantity is negative – **ignore** the whole entity.
* **Any** of the properties **may** **be** not given (null) in the json file (including the quantity of seats per train) – if the specific property is required and it has null value the whole entity is considered **invalid**.
* If any validation error occurs proceed as described above.
* There will be **only one** seat class with same name **per** train:

|  |
| --- |
| [  {  "Seats": [  {  Name:"First class",   …  },  {  Name:" First class",   …  }  ]  }  ] |

**The input above will not be presented!**

##### Example

|  |
| --- |
| **trains.json** |
| [  {  "TrainNumber": "KB20012",  "Type": "HighSpeed",  "Seats": [  {  Name:"First class",   Abbreviation: "FC",  Quantity:50  },  {  Name:"Business class",   Abbreviation: "BC",  Quantity:44  }  ]  },  …  ] |
| **Output** |
| Record KB20012 successfully imported.  … |

#### 4.Import Trips

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

If there is any violation of the requirements mentioned in the first section or some of the stations/trains do not exist **ignore** the whole entity and **continue** with next one.

If status is not specified use the **default** one.

##### Constraints

* Status will always be a valid value or null
* Time Difference may be null or be represented in format “hh\:mm”
* Arrival/Departure date may be null or in format “dd/MM/yyyy HH:mm”
* Always check for the existence of trains and statons.
* Always check if **departure time** is before the **arrival** one.
* Any of the properties might be null – if the property is considered required consider the **whole entity** as invalid.

##### Example

|  |
| --- |
| **trips.json** |
| [  {  "Train": "KB20012",  "OriginStation": "Sofia",  "DestinationStation": "Sofia Sever",  "DepartureTime": "27/12/2016 12:00",  "ArrivalTime": "27/12/2016 12:30",  "Status": "OnTime",  },  …  ] |
| **Output** |
| Trip from Sofia to Sofia Sever imported.  … |

### XML Import (5 pts)

#### 5.Import Person Cards

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

If any of the model requirements is violated continue with the next entity.

##### Constraints

* Card type will be valid value or null (if null set default value)
* There will be no any other missing(null) elements

##### Example

|  |
| --- |
| **cards.xml** |
| <?xml version="1.0" encoding="utf-8"?>  <Cards>  <Card>  <Name>John Levit</Name>  <Age>25</Age>  <CardType>Normal</CardType>  </Card>  <Card>  <Name>Ana Keanig</Name>  <Age>19</Age>  <CardType>Student</CardType>  </Card>  …  </Cards> |
| **Output** |
| Record John Levit successfully imported.  Record Ana Keanig successfully imported. |

#### 6.Import Tickets

Using the file **tickets.xml**, import the data from the file into the database.

Find the trip with the specified data (by origin/destination station and departure time). Assume there will be exactly one or zero trips available (there will be no trips with same departure time and origin/destination station). If there is existing trip use the train referenced there for the next part.

The **seat** must be a combination between seating class **abbreviation** and **integer** number (e.g: “**FC42**” - where “**FC**” stands for “**First Class**” and “**42**” is **number** of the specific **seat**) – this means that you should check if there is a **class** with same abbreviation and then check **if** the **train** has **any** **seats** from that class. Last, the seat number must be positive and **below** **or** **equal** to the **quantity** of seats specified for that train.

If the format of the seat doesn’t match the above criteria, the entity is invalid.

##### Constraints

* Ticket’s price will always be a valid number (**not null**)
* Ticket’s seat will not be null
* If there is no existing trip with matching properties or there is **no card** with matching name in the database, **ignore** the whole entity.
* Departure time will be a valid datetime in the format: “dd/MM/yyyy HH:mm”
* Cards will not have duplicate names
* If any validation error occurs, proceed as described above.

##### Example

|  |
| --- |
| **tickets.xml** |
| <?xml version="1.0" encoding="utf-8"?>  <Tickets>  <Ticket price="3.53" seat="SC12">  <Trip>  <OriginStation>Sofia</OriginStation>  <DestinationStation>Sofia Sever</DestinationStation>  <DepartureTime>27/12/2016 12:00</DepartureTime>  </Trip>  </Ticket>  <Ticket price="3.28" seat="FC1">  <Trip>  <OriginStation>Sofia</OriginStation>  <DestinationStation>Sofia Sever</DestinationStation>  <DepartureTime>27/12/2016 12:00</DepartureTime>  </Trip>  <Card Name="Amber Hosh" />  </Ticket>  </Tickets>  … |
| **Output** |
| Ticket from Sofia to Sofia Sever departing at 27/12/2016 12:00 imported.  Ticket from Sofia to Sofia Sever departing at 27/12/2016 12:00 imported.  … |

## Problem 3. Data Export (15pts)

For the functionality of the application, you need to create several methods that manipulate the database. Create these methods inside the **data layer** of your solution. **Database query methods will be assessed separately from export functionality.** Use **Data Transfer Objects** as needed.

Create a **new project** inside your solution that would handle data **export (where serialization would happen)**.

### JSON Export (5 pts)

#### Export All Delayed Trains

The given method in the project skeleton receives a **date** as string in format "**dd/MM/yyyy**". Export all trains which were part of trip with status: "**Delayed**" and also the trip’s departure time is **smaller or equal** to the given date. Display only unique trains alongside with how many times they have been late and their highest delay in format: "**HH:mm:ss**" (the **default** Timespan format). Order the trains by: count of how many times they have been late (descending), highest delay time (descending) and train number (ascending).

|  |
| --- |
| **delayed-trains-by-01-01-2017.json** |
| [  {  "TrainNumber": "PU17333",  "DelayedTimes": 2,  "MaxDelayedTime": "23:02:00"  },  {  "TrainNumber": "VT08003",  "DelayedTimes": 2,  "MaxDelayedTime": "15:51:00"  },  {  "TrainNumber": "BQ877549",  "DelayedTimes": 1,  "MaxDelayedTime": "21:39:00"  },  …  ] |

### XML Export (5 pts)

#### Export Cards by Type

Use the method provided in the project skeleton, which receives a **card type**. Your task is to export all tickets bought by people with the same card type. Order by **card name** in ascending order.

#### Example

|  |
| --- |
| **tickets-bought-with-card-type-Debilitated.xml** |
| <?xml version="1.0" encoding="utf-8"?>  <Cards>  <Card name="George Powell" type="Debilitated">  <Tickets>  <Ticket>  <OriginStation>Sofia</OriginStation>  <DestinationStation>Varna</DestinationStation>  <DepartureTime>24/05/2017 12:00</DepartureTime>  </Ticket>  <Ticket>  <OriginStation>Sofia</OriginStation>  <DestinationStation>Sagay</DestinationStation>  <DepartureTime>02/12/2016 12:20</DepartureTime>  </Ticket>  </Tickets>  </Card>  <Card name="Henry Moreno" type="Debilitated">  <Tickets>  <Ticket>  <OriginStation>Ajjah</OriginStation>  <DestinationStation>San Isidro</DestinationStation>  <DepartureTime>02/04/2016 12:33</DepartureTime>  </Ticket>  </Tickets>  </Card>  </Cards> |

#### Other case

|  |
| --- |
| **tickets-bought-with-card-type-Elder.xml** |
| <?xml version="1.0" encoding="utf-8"?>  <Cards>  <Card name="Jeremy Carroll" type="Elder">  <Tickets>  <Ticket>  <OriginStation>Sofia</OriginStation>  <DestinationStation>Varna</DestinationStation>  <DepartureTime>24/05/2017 22:00</DepartureTime>  </Ticket>  <Ticket>  <OriginStation>Huaian</OriginStation>  <DestinationStation>Chysky</DestinationStation>  <DepartureTime>13/08/2011 12:33</DepartureTime>  </Ticket>  <Ticket>  <OriginStation>Sofia</OriginStation>  <DestinationStation>Varna</DestinationStation>  <DepartureTime>24/05/2017 12:00</DepartureTime>  </Ticket>  <Ticket>  <OriginStation>Sofia</OriginStation>  <DestinationStation>Varna</DestinationStation>  <DepartureTime>24/05/2017 12:00</DepartureTime>  </Ticket>  </Tickets>  </Card>  <Card name="Michelle Sanders" type="Elder">  <Tickets>  <Ticket>  <OriginStation>Matriz de Camaragibe</OriginStation>  <DestinationStation>Chysky</DestinationStation>  <DepartureTime>23/11/2016 15:09</DepartureTime>  </Ticket>  </Tickets>  </Card>  <Card name="Wanda Ward" type="Elder">  <Tickets>  <Ticket>  <OriginStation>Sofia</OriginStation>  <DestinationStation>Sofia Sever</DestinationStation>  <DepartureTime>27/12/2016 12:00</DepartureTime>  </Ticket>  </Tickets>  </Card>  </Cards> |