# Entity Framework Exam – 9 April 2017

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 at [judge](https://softuni.bg/courses/databases-advanced-entity-framework).

Your task is to create a database application using **Entity Framework** with **Code First** approach. Design the domain models and utility functions for manipulating the data, as described below. Add **navigation properties** as you see fit.

# Planet Hunters

Create an application that stores information about discoveries of exoplanets (planets outside our solar system). It needs to keep track of astronomers and telescopes working on a project as well as historical data about old observations. Any newly discovered objects must be stored along with information about who discovered it and when – a team of astronomers who first observed it (named pioneers) and any people who later confirmed the discovery (named observers).

## Problem 1. Model Definition (50 pts)

Create **two** separate **projects** in the same solution – one for models and one for the data connection.

An **astronomer** can make many **discoveries** and observe many **discoveries**. Conversely, a **discovery** can be made by many **astronomers** and can be **observed** (confirmed) by many astronomers. A single **discovery** may include many **planets** and **stars**. Each **planet** or **star** holds information about its host **star system**.

The application needs to store information about the following:

##### Astronomer

* **First Name** – text with max length 50 (**required**)
* **Last Name** – text with max length 50 (**required**)
* **Many pioneering Discoveries made** – a collection of discoveries
* **Many observations of Discoveries made** – a collection of discoveries

##### Discovery

* **Date Made** – **DATE** when the discovery was announced (**required**)
* **Telescope used** (**required**)
* **Many Stars**
* **Many Planets**
* **Pioneers** – a collection of astronomers
* **Observers** – a collection of astronomers

##### Telescope

* **Name** – text with max length 255 (**required**)
* **Location** – text with max length 255 (**required**)
* **Mirror Diameter** – size of primary reflector in meters, floating point number. **Cannot** be zero or negative.

##### Star System

* **Name** – text with max length 255 (**required**)
* **Many Stars**
* **Many Planets**

##### Star

* **Name** – text with max length 255 (**required**)
* **Temperature** – effective temperature in Kelvin, integer (**required**). **Cannot** be lower than 2400K.
* **Host Star System** (**required**)

##### Planet

* **Name** – text with max length 255 (**required**)
* **Mass** – represented in relative Earth masses, floating point number. **Cannot** be zero or negative (**required**)
* **Host Star System** (**required**)

Any information/relation **not marked** as required may be **null**. Any information that has additional requirements needs to be **validated**.

## Problem 2. Data Import (25pts)

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 import functionality.** Use **Data Transfer Objects** as needed.

Create a **new project** inside your solution that would handle **importing (where the actual deserialization will happen)**. 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 (5 pts)

#### Import Astronomers

Using the file **astronomers.json**, create method for importing the data from that file into the database. Print information about each imported object in the format described below.

#### Example

|  |
| --- |
| **astronomers.json** |
| [  {  "FirstName": "John",  "LastName": "Gizis"  },  {  "FirstName": "David",  "LastName": "Monet"  },  …  ] |
| **Output** |
| Record John Gizis successfully imported.  Record David Monet successfully imported.  … |

#### Import Telescopes

Using the file **telescopes.json**, create a method that imports the data from that file into the database. Print information about each imported object in the format described below.

#### Example

|  |
| --- |
| **telescopes.json** |
| [  {  "Name": "TRAPPIST",  "Location": "Chile",  "MirrorDiameter": 0.6  },  {  "Name": "La Silla",  "Location": "Chile",  "MirrorDiameter": 3.6  },  …  ] |
| **Output** |
| Record TRAPPIST successfully imported.  Record La Silla successfully imported.  … |

#### Import Planets

Using the file **planets.json**, create method for importing the data from that file into the database. Print information about each imported object in the format described below. If there is **no** star system with given name simply **add it** to database.

#### Example

|  |
| --- |
| **planets.json** |
| [  {  "Name": "Trappist-1 b",  "Mass": 0.85,  "StarSystem": "Trappist-1",  },  {  "Name": "Trappist-1 c",  "Mass": 1.38,  "StarSystem": "Trappist-1"  },  …  ] |
| **Output** |
| Record Trappist-1 b successfully imported.  Record Trappist-1 successfully imported.  Record Trappist-1 c successfully imported.  … |

### XML Import (5 pts)

#### Import Stars

Using the file **stars.xml**, create method for importing the data from that file into the database. Print information about each imported object in the format described below. If there is **no** star system with given name simply **add it** to database.

#### Example

|  |
| --- |
| **stars.xml** |
| <?xml version="1.0" encoding="utf-8"?>  <Stars>  <Star>  <Name>Trappist-1</Name>  <Temperature>2550</Temperature>  <StarSystem>Trappist-1</StarSystem>  </Star>  <Star>  <Name>Proxima Centauri</Name>  <Temperature>3042</Temperature>  <StarSystem>Alpha Centauri</StarSystem>  </Star>  …  </Stars> |
| **Output** |
| Record Trappist-1 successfully imported.  Record Proxima Centauri successfully imported. |

#### Import Discoveries

Using the file **discoveries.xml**, create a method that imports the data from that file into the database. If any of the specified entities in a single discovery do not exist (astronomer, planet or star) you should **ignore** that discovery **and continue** with the next one. Assume that the telescope will always exist – there no need to validate it explicitly.

Keep in mind that you should find the entities by their **names**. If there are more the one entity matching given name pick the first **one**.

#### Example

|  |
| --- |
| **discoveries.xml** |
| <?xml version="1.0" encoding="utf-8"?>  <Discoveries>  <Discovery DateMade="2000-04-26" Telescope="TRAPPIST">  <Stars>  <Star>Trappist-1</Star>  </Stars>  <Planets />  <Pioneers>  <Astronomer>Gizis, John</Astronomer>  <Astronomer>Monet, David</Astronomer>  <Astronomer>Liebert, James</Astronomer>  </Pioneers>  <Observers />  </Discovery>  … |
| **Output** |
| Discovery (2004/04/26-TRAPPIST) with 1 star(s), 0 planet(s), 3 pioneer(s) and 0 observers successfully imported.  … |

## Problem 3. Data Export (25pts)

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 Planets

Write a method that takes in a **Telescope name** as a string **parameter** and creates the file **planets-by-{telescopeName}.json** (replace the part in brackets with the relevant name) that would contain the names and mass of all planets in the database, that were discovered by the given telescope. Order them descending by mass.

|  |
| --- |
| **planets-by-****TRAPPIST.json** |
| [  {  "Name": "Trappist-1 c",  "Mass": 1.38,  "Orbiting": [  "Trappist-1"  ]  },  {  "Name": "Trappist-1 b",  "Mass": 0.85,  "Orbiting": [  "Trappist-1"  ]  }  ] |

#### Export Astronomers

Write a method that takes in a **Star System’s name** as a string **parameter** and creates file **astronomers-of-{starSystemName}.json** (replace the part in brackets with the relevant name) that would contain the full names and involvement (discoverer or not) of all astronomers in the database, that are associated with the given Star System. Order them by last name. Make sure each Astronomer appears only once per role.

#### Example

|  |
| --- |
| **astronomers-of-****Alpha Centauri.json** |
| [  {  "Name": "Pedro Amado",  "Role": "pioneer"  },  {  "Name": "John Barnes",  "Role": "pioneer"  },  {  "Name": "Zaira Berdinos",  "Role": "pioneer"  },  {  "Name": "Nicolas de Lacaille",  "Role": "observer"  },  …  ] |

### XML Export (5 pts)

#### Export Stars

Write a method that creates the file **stars.xml** that would contain information about all stars (which have been discovered) in the database and details about their discovery, including:

* Name
* Temperature
* Star System Name
* Discovery Date
* Telescope Name
* List of all Astronomers involved in the discovery, with all pioneers tagged, ordered by name.

#### Example

|  |
| --- |
| **stars.xml** |
| <?xml version="1.0" encoding="utf-8"?>  <Stars>  <Star>  <Name>Trappist-1</Name>  <Temperature>2550</Temperature>  <StarSystem>Trappist-1</StarSystem>  <DiscoveryInfo DiscoveryDate="2000-04-26" TelescopeName="TRAPPIST" />  </Star>  <Star>  <Name>Proxima Centauri</Name>  <Temperature>3042</Temperature>  <StarSystem>Alpha Centauri</StarSystem>  <DiscoveryInfo DiscoveryDate="1915-10-12" TelescopeName="Union Observatory">  <Astronomer Pioneer="false">Joan Voute</Astronomer>  <Astronomer Pioneer="true">Robert Innes</Astronomer>  </DiscoveryInfo>  </Star>  …  </Stars> |

## Bonus Task: Migrate Database (10 pts)

For this task, you will need to **enable migrations**.

Augment the model to include Journals and Publications. The new entities contain the following information:

##### Journal

* **Name** – text with max length 50 (**required**)
* **Many Publications**

##### Publication

* **Release Date** – date when the publication was made (**required**)
* **Discovery** that the publication covers (**required**)

Add **navigation properties** as you see fit. Seed the database with at least one Journal and for every Discovery, create a Publication with release date matching the announcement date of the Discovery. Remove the **DateMade** property from Discovery as it is no longer needed. Record all changes and data transfer with migrations.