# Exercises: XML Processing

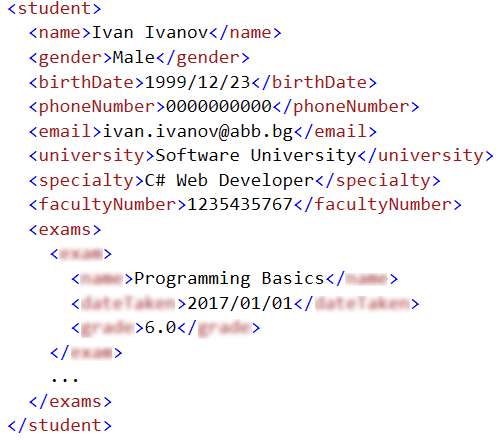
This document defines the **exercise assignments** for the ["Databases Advanced – EF Core" course @ Software University](https://softuni.bg/trainings/1741/databases-advanced-entity-framework-october-2017).

# XML Format

## Students XML

Create a XML document students.xml in your favorite editor, which contains structured description of students. For each student you should enter information for his **name**, **gender**, **birth date** (in ISO 8601 format), **phone number** (optional), **email**, **university**, **specialty**, **faculty number** (optional) and a list of taken **exams** (exam name, date taken, grade). Use only elements (tags).

Create a new file students.xml and insert some xml with a structure similar to this one:



## Catalog of Musical Albums

Create a XML file catalog.xml representing a catalog of musical **albums**. For each album you should define name, artist, year, producer, price and a list of songs. Each song should be described by title and duration.

Hint: You can take sample data from <https://gist.github.com/jasonbaldridge/2597611>.

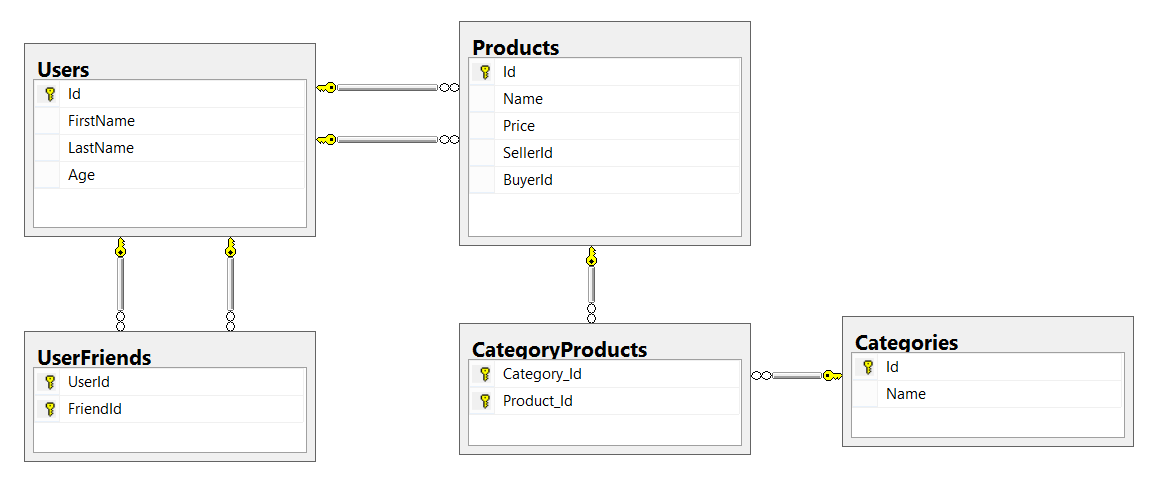
# Product Shop Database

## Setup Database

A products shop holds **users**, **products** and **categories** for the products. Users can **sell** and **buy** products.

* Users have an **id**, **first** **name** (optional) and **last** **name** (at least 3 characters) and **age** (optional).
* Products have an **id**, **name** (at least 3 characters), **price**, **buyerId** (optional) and **sellerId** as IDs of users.
* Categories have an **id** and **name** (from **3** to **15** characters)

Using Entity Framework Code First create a database following the above description.



Configure the following relations in your EF models:

* **Users** should have **many products sold** and **many products bought**.
* **Products** should have **many categories**
* **Categories** should have **many products**
* **Users** should have **many friends** (i.e. users)

## Import Data

**Import** the data from the provided files (**users.xml**, **products.xml**, **categories.xml**).

Import the **users** first. When importing products, randomly **select the buyer** and **seller** from the existing users. Leave out some **products** that have **not been sold** (i.e. buyer is null).

Randomly **generate categories** for each product from the existing categories.

## Query and Export Data

Write the below described queries and **export** the returned data to the specified **format**. Make sure that Entity Framework generates only a **single query** for each task.

### Products In Range

Get all products in a specified **price range** between 1000 and 2000 (inclusive) which have **buyer**. Order them by price (from lowest to highest). Select only the **product name**, **price** and the **full name** **of the buyer**. Export the result to XML.

|  |
| --- |
| **products-in-range.xml** |
| <?xml version="1.0" encoding="utf-8"?>  <products>  <product name="TYLENOL COLD MULTI-SYMPTOM DAYTIME" price="1010.81" buyer="Sandra Riley" />  <product name="Butalbital, Aspirin and Caffeine" price="1010.98" buyer=" Bennett" />  <product name="SEPHORA Acne-Fighting Mattifying Moisturizer" price="1019.28" buyer="Patricia Cooper" />  </products> |

### Sold Products

Get all users who have **at least 1 sold item**. Order them by **last name**, then by **first name**. Select the person's **first** and **last name**. For each of the **sold products**, select the product's **name** and **price**.

|  |
| --- |
| **users-sold-products.xml** |
| <?xml version="1.0" encoding="utf-8"?>  <users>  <user first-name="Carl" last-name="Daniels">  <sold-products>  <product>  <name>Peter Island Continous sunscreen kids</name>  <price>471.30</price>  </product>  <product>  <name>Warfarin Sodium</name>  <price>1379.79</price>  </product>  ...  </sold-products>  </user>  ...  </users> |

### Categories By Products Count

Get **all** **categories**. Order them by the **number of products** in that category (a product can be in many categories). For each category select its **name**, the **number of products**, the **average price of those products** and the **total revenue** (total price sum) of those products (regardless if they have a buyer or not).

|  |
| --- |
| **categories-by-products.xml** |
| <?xml version="1.0" encoding="utf-8"?>  <categories>  <category name="Sports">  <products-count>49</products-count>  <average-price>754.327755</average-price>  <total-revenue>36962.06</total-revenue>  </category>  <category name="Adult">  <products-count>46</products-count>  <average-price>905.283478</average-price>  <total-revenue>41643.04</total-revenue>  </category>  ...  </categories> |

### Users and Products

Get all users who have **at least 1 sold product**. Order them by the **number of sold products** (from highest to lowest), then by **last name** (ascending). Select only their **first** and **last name**, **age** and for each product - **name** and **price**.

Export the results to **XML**. Follow the format below to better understand how to structure your data.

|  |
| --- |
| **users-and-products.xml** |
| <?xml version="1.0" encoding="utf-8"?>  <users count="35">  <user first-name="Carl" last-name="Daniels" age="59">  <sold-products count="10">  <product name="Finasteride" price="1374.01" />  <product name="Peter Island Continous sunscreen kids" price="471.30" />  <product name="Warfarin Sodium" price="1379.79" />  <product name="Gilotrif" price="1454.77" />  <product name="Cold and Cough" price="218.14" />  ...  </sold-products>  </user>  <user last-name="Harris">  <sold-products count="9">  <product name="Clarins Paris Skin Illusion - 114 cappuccino" price="811.42" />  ...  </sold-products>  </user>  ...  </users> |

# Car Dealer Database

## Setup Database

A car dealer needs information about cars, their parts, parts suppliers, customers and sales.

* **Cars** have **make, model**, travelled distance in kilometers
* **Parts** have **name**, **price** and **quantity**
* Part **supplier** have **name** and info whether he **uses imported parts**
* **Customer** has **name**, **date of birth** and info whether he **is young driver** (Young driver is a driver that has **less than 2 years of experience**. Those customers get **additional 5% off** for the sale.)
* **Sale** has **car**, **customer** and **discount percentage**

A **price of a car** is formed by **total price of its parts**.

Using Entity Framework Code First create a database following the above description.



Configure the following relations in your EF models:

* A **car** has **many parts** and **one part** can be placed **in many cars**
* **One supplier** can supply **many parts** and each **part** can be delivered by **only one supplier**
* In **one sale**, only **one car** can be sold
* **Each sale** has **one customer** and **a customer** can buy **many cars**

## Import Data

Import data from the provided files (**suppliers.xml, parts.xml, cars.xml, customers.xml**).

First import **suppliers**. When importing **parts** set to each part **random supplier** from already imported suppliers. Then, when importing cars add **between 10 and 20 random parts** to each car. Then import **all customers**. Finally, import **sales records** by **random** selecting a **car, customer** and the amount of **discount to be applied** (discounts can be 0%, 5%, 10%, 15%, 20%, 30%, 40% or 50%).

## Query and Export Data

Write the below described queries and **export** the returned data to the specified **format**. Make sure that Entity Framework generates only a **single query** for each task.

### Cars with Distance

Get all **cars** with distance more than 2,000,000. Order them by make, then by model alphabetically. **Export** the list of cars **to XML** in the format provided below.

|  |
| --- |
| **cars.xml** |
| <?xml version="1.0" encoding="utf-8"?>  <cars>  <car>  <make>Alfa Romeo</make>  <model>156</model>  <travelled-distance>92233720368547</travelled-distance>  </car>  <car>  <make>Alfa Romeo</make>  <model>156</model>  <travelled-distance>92036854775807</travelled-distance>  </car>  <car>  <make>Alfa Romeo</make>  <model>164</model>  <travelled-distance>92233775807</travelled-distance>  </car>  ...  </cars> |

### Cars from make Ferrari

Get all **cars** from make **Ferrari** and **order them by model alphabetically** and by **travelled distance descending**. **Export** the list of **cars to XML** in the format provided below.

|  |
| --- |
| **ferrari-cars.xml** |
| <?xml version="1.0" encoding="utf-8"?>  <cars>  <car id="232" model="250 GTO" travelled-distance="214747" />  <car id="237" model="250 GTO" travelled-distance="214747" />  ...  </cars> |

### Local Suppliers

Get all **suppliers** that **do not import parts from abroad**. Get their **id**, **name** and **the number of parts they can offer to supply**. **Export** the list of suppliers **to XML** in the format provided below.

|  |
| --- |
| **local-suppliers.xml** |
| <?xml version="1.0" encoding="utf-8"?>  <suppliers>  <suplier id="2" name="Agway Inc." parts-count="6" />  <suplier id="4" name="Airgas, Inc." parts-count="5" />  ...  </suppliers> |

### Cars with Their List of Parts

Get all **cars along with their list of parts**. For the **car** get only **make, model** and **travelled distance** and for the **parts** get only **name** and **price**. **Export** the list of **cars and their parts to XML** in the format provided below.

|  |
| --- |
| **cars-and-parts.xml** |
| <?xml version="1.0" encoding="utf-8"?>  <cars>  <car make="Opel" model="Omega" travelled-distance="2147483647" />  <parts>  <part name="Front Left Side Outer door handle" price="999.99" />  <part name="Gudgeon pin" price="44.99" />  <part name="Oil pump" price="100.19" />  <part name="Transmission pan" price="106.99" />  </parts>  </car>  <car make="Opel" model="Astra" travelled-distance="9223372036854775807" />  <parts>  <part name="Overflow tank" price="1200.99" />  ...  </parts>  </car>  ...  </cars> |

### Total Sales by Customer

Get all **customers** that have bought **at least 1 car** and get their **names**, **bought cars** **count** and **total spent money** on cars. **Order** the result list **by total spent money descending** then by **total bought cars** again in **descending** order. **Export** the list of customers **to** **XML** in the format provided below.

|  |
| --- |
| **customers-total-sales.xml** |
| <?xml version="1.0" encoding="utf-8"?>  <customers>  <customer full-name="Hipolito Lamoreaux" bought-cars="5" spent-money="8360.48" />  <customer full-name="Francis Mckim" bought-cars="4" spent-money="7115.50" />  <customer full-name="Johnette Derryberry" bought-cars="4" spent-money="5337.72" />  ...  </customer> |

### Sales with Applied Discount

Get all **sales** with information about the **car**, **customer** and **price** of the sale **with and without discount**. **Export** the list of sales **to XML** in the format provided below.

|  |
| --- |
| **sales-discounts.xml** |
| <?xml version="1.0" encoding="utf-8"?>  <sales>  <sale>  <car make="Peugeot" model="405" travelled-distance="92036854775807" />  <customer-name>Donnetta Soliz</customer-name>  <discount>0.3</discount>  <price>1402.53</price>  <price-with-discount>981.771</price-with-discount>  </sale>  <sale>  ...  </sales> |