# Exercises: XML Processing

This document defines the **exercise assignments** for the [Databases Advanced - Entity Framework course @ SoftUni](https://softuni.bg/trainings/4540/entity-framework-core-june-2024)  
You can check your solutions in [Judge](https://judge.softuni.org/Contests/3929/Extensible-Markup-Language-XML)

# 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**, **FirstName** (optional), **LastName** and **Age**(optional).
* **Products** have an **Id**, **Name**, **Price**, **BuyerId**(optional) and **SellerId** as **Id**s of **Users**.
* **Categories** have an **Id** and **Name**.
* Using Entity Framework and Code First create a database, following the above description.



* **Users** should have **many Products sold** and **many Products bought**.
* **Products** should have **many Categories.**
* **Categories** should have **many Products.**
* **CategoryProducts** should **map Products** and **Categories.**

## Import Data

### Query 1. Import Users

**NOTE:** You will need method **public static string ImportUsers(ProductShopContext context, string inputXml)** and **public StartUp** class.

Import the users from the provided file "**users.xml"**.

Your method should return a string with the following message:  
**$"Successfully imported {users.Count}";**

### Query 2. Import Products

**NOTE:** You will need method **public static string ImportProducts(ProductShopContext context, string inputXml)** and **public StartUp** class.

Import the products from the provided file "**products.xml**".

Your method should return a string with the following message:  
**$"Successfully imported {products.Count}";**

### Query 3. Import Categories

**NOTE:** You will need method **public static string ImportCategories(ProductShopContext context, string inputXml)** and **public StartUp** class.

Import the categories from the provided file "**categories.xml**".

Some of the names will be null, so you don't have to add them to the database. Just skip the record and continue.

Your method should return a string with the following message:  
**$"Successfully imported {categories.Count}";**

### Query 4. Import Categories and Products

**NOTE:** You will need method **public static string ImportCategoryProducts(ProductShopContext context, string inputXml)** and **public StartUp** class.

Import the categories and products ids from the provided file "**categories-products.xml**". If provided **CategoryId** or **ProductId** doesn't exist, skip the whole entry!

Your method should return a string with the message:  
**$"Successfully imported {categoryProducts.Count}";**

## Query and Export Data

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

### Query 5. Export Products In Range

**NOTE:** You will need method **public static string GetProductsInRange(ProductShopContext context)** and **public StartUp** class.

Get all products in a specified **price range** between **500** and **1000** (inclusive). Order them by price (from lowest to highest). Select only the **product name**, **price** and the **full name** **of the buyer**. Take top **10** records.

**Return** the list of suppliers **to XML** in the format provided below.

|  |
| --- |
| **products-in-range.xml** |
| <?xml version="1.0" encoding="utf-16"?>  <Products>  <Product>  <name>TRAMADOL HYDROCHLORIDE</name>  <price>516.48</price>  </Product>  <Product>  <name>Allopurinol</name>  <price>518.5</price>  <buyer>Wallas Duffyn</buyer>  </Product>  <Product>  <name>Parsley</name>  <price>519.06</price>  <buyer>Brendin Predohl</buyer>  </Product>  …  </Products> |

### Query 6. Export Sold Products

**NOTE:** You will need method **public static string GetSoldProducts(ProductShopContext context)** and **public StartUp** class.

Get all users who have **at least 1 sold item**. Order them by **the last name**, then by **the first name**. Select the person's **first** and **last name**. For each of the **sold products**, select the product's **name** and **price**. Take top **5** records.

**Return** the list of suppliers **to XML** in the format provided below.

|  |
| --- |
| **users-sold-products.xml** |
| <?xml version="1.0" encoding="utf-16"?>  <Users>  <User>  <firstName>Almire</firstName>  <lastName>Ainslee</lastName>  <soldProducts>  <Product>  <name>Ampicillin</name>  <price>674.63</price>  </Product>  <Product>  <name>Strattera</name>  <price>658.54</price>  </Product>  <Product>  <name>Aspergillus repens</name>  <price**>**1231.42**</**price>  </Product>  </soldProducts>  </User>  ...  </Users> |

### Query 7. Export Categories By Products Count

**NOTE:** You will need method **public static string** **GetCategoriesByProductsCount(ProductShopContext context)** and **public** **StartUp** class.

Get **all** **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). Order them by the **number of products** (**descending**), then by total revenue (**ascending**).

**Return** the list of suppliers **to XML** in the format provided below.

|  |
| --- |
| **categories-by-products.xml** |
| <?xml version="1.0" encoding="utf-16"?>  <Categories>  <Category>  <name>Garden</name>  <count>23</count>  <averagePrice>800.150869</averagePrice>  <totalRevenue>18403.47</totalRevenue>  </Category>  <Category>  <name>Weapons</name>  <count>22</count>  <averagePrice>671.073181</averagePrice>  <totalRevenue>14763.61</totalRevenue>  </Category>  ...  </Categories> |

### Query 8. Export Users and Products

**NOTE:** You will need method **public static string GetUsersWithProducts(ProductShopContext context)** and **public StartUp** class.

Select users who have **at least 1 sold product**. Order them by the **number of sold products** (from highest to lowest). Select only their **first** and **last name**, **age, count** of sold products and for each product - **name** and **price** sorted by price (descending). Take top **10** records.

Follow the format below to better understand how to structure your data.

**Return** the list of suppliers **to XML** in the format provided below.

|  |
| --- |
| **users-and-products.xml** |
| <Users>  <count>54</count>  <users>  <User>  <firstName>Dale</firstName>  <lastName>Galbreath</lastName>  <age>31</age>  <SoldProducts>  <count>9</count>  <products>  <Product>  <name>Fair Foundation SPF 15</name>  <price>1394.24</price>  </Product>  <Product>  <name>Finasteride</name>  <price>1374.01</price>  </Product>  <Product>  <name>EMEND</name>  <price>1365.51</price>  </Product>  ...  </Users> |

# Car Dealer

## Setup Database

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

* **Cars** have **Make, Model**, **TraveledDistance** in kilometers.
* **Parts** have **Name**, **Price** and **Quantity.**
* **Supplier** has a **Name** and info whether they supply **imported parts.**
* **Customer** has a **Name**, **BirthDate** and info whether they are **a young driver** (young driver is a driver that has **less than 2 years of experience**. Those customers get **an additional 5% off** for the sale.).
* **Sale** has **a Car**, **Customer**and a **Discount** **percentage.**

A **Price of a Car** is formed by **the total price** of **its Parts**.



* 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 to only one **Customer**.
* **A Customer** can buy **many Cars.**

## Import Data

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

### Query 9. Import Suppliers

**NOTE:** You will need method **public static string ImportSuppliers(CarDealerContext context, string inputXml)** and **public StartUp** class.

Import the suppliers from the provided file "**suppliers.xml**".

Your method should return a string with the following message:  
**$"Successfully imported {suppliers.Count}";**

### Query 10. Import Parts

**NOTE:** You will need method **public static string ImportParts(CarDealerContext context, string inputXml)** and **public StartUp** class.

Import the parts from the provided file "**parts.xml**". If the **supplierId** doesn't exist, skip the record.

Your method should return a string with the message:  
**$"Successfully imported {parts.Count}";**

### Query 11. Import Cars

**NOTE:** You will need method **public static string ImportCars(CarDealerContext context, string inputXml)** and **public StartUp** class.

Import the cars from the provided file "**cars.xml**". Select unique car part ids. If the **partId** doesn't exist, skip the **Part** record.

Your method should return a string with the following message:  
**$"Successfully imported {cars.Count}";**

### Query 12. Import Customers

**NOTE:** You will need method **public static string ImportCustomers(CarDealerContext context, string inputXml)** and **public StartUp** class.

Import the customers from the provided file "**customers.xml**".

Your method should return a string with the following message:  
**$"Successfully imported {customers.Count}";**

### Query 13. Import Sales

**NOTE:** You will need method**public static string ImportSales(CarDealerContext context, string inputXml)** and **public StartUp** class.

Import the sales from the provided file "**sales.xml**". If car doesn't exist, skip whole entity.

Your method should return a string with the following message:  
**$"Successfully imported {sales.Count}";**

## 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.

### Query 14. Export Cars With Distance

**NOTE:** You will need method **public static string GetCarsWithDistance(CarDealerContext context)** and **public StartUp** class.

Get all **cars** with a distance of more **than** 2,000,000. Order them by make, then by model alphabetically. Take top 10 records.

**Return** the list of suppliers to **XML** in the format provided below.

|  |
| --- |
| **cars.xml** |
| <?xml version="1.0" encoding="utf-16"?>  <cars>  <car>  <make>BMW</make>  <model>1M Coupe</model>  <traveled-distance>39826890</traveled-distance>  </car>  <car>  <make>BMW</make>  <model>E67</model>  <traveled-distance>476830509</traveled-distance>  </car>  <car>  <make>BMW</make>  <model>E88</model>  <traveled-distance>27453411</traveled-distance>  </car>  ...  </cars> |

### Query 15. Export Cars from Make BMW

**NOTE**: You will need method **public static string GetCarsFromMakeBmw(CarDealerContext context)** and **public StartUp** class.

Get all **cars** from make **BMW** and **order them by model alphabetically** and by **traveled distance descending**.

**Return** the list of suppliers **to XML** in the format provided below.

|  |
| --- |
| **bmw-cars.xml** |
| <cars>  <car id="26" model="1M Coupe" traveled-distance="39826890" />  <car id="28" model="E67" traveled-distance="476830509" />  <car id="24" model="E88" traveled-distance="27453411" />  ...  </cars> |

### Query 16. Export Local Suppliers

**NOTE:** You will need method **public static string GetLocalSuppliers(CarDealerContext context)** and **public StartUp** class.

Get all **suppliers** that **do not import parts from abroad**. Get their **id**, **name** and **the number of parts they can offer to supply**.

**Return** the list of suppliers **to XML** in the format provided below.

|  |
| --- |
| **local-suppliers.xml** |
| <?xml version="1.0" encoding="utf-16"?>  <suppliers>  <supplier id="2" name="Agway Inc." parts-count="3" />  <supplier id="4" name="Airgas, Inc." parts-count="2" />  ...  </suppliers> |

### Query 17. Export Cars with Their List of Parts

**NOTE:** You will need method **public static string GetCarsWithTheirListOfParts(CarDealerContext context)** and **public StartUp** class.

Get all **cars along with their list of parts**. For the **car** get only **make, model** and **traveled distance** and for the **parts** get only **name** and **price** and sort all parts by price (descending). Sort all cars by traveled **distance** (descending) and then by the **model** (ascending). Select top 5 records.

**Return**the list of suppliers **to XML** in the format provided below.

|  |
| --- |
| **cars-and-parts.xml** |
| <?xml version="1.0" encoding="utf-16"?>  <cars>  <car make="Opel" model="Astra" traveled-distance="516628215">  <parts>  <part name="Tappet" price="300.29" />  <part name="Front Left Side Door Glass" price="100.92" />  <part name="Fan belt" price="10.99" />  </parts>  </car>  ...  </cars> |

### Query 18. Export Total Sales by Customer

**NOTE:** You will need method **public static string GetTotalSalesByCustomer(CarDealerContext context)** and **public StartUp** class.

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). **Don't forget that young drivers get a discount**!

**Return** the list of suppliers **to XML** in the format provided below.

|  |
| --- |
| **customers-total-sales.xml** |
| <?xml version="1.0" encoding="utf-16"?>  <customers>  <customer full-name="Emmitt Benally" bought-cars="1" spent-money="5044.10" />  <customer full-name="Garret Capron" bought-cars="1" spent-money="3327.10" />  <customer full-name="Cinthia Lasala" bought-cars="1" spent-money="3173.09" />  ...  </customers> |

### Query 19. Export Sales with Applied Discount

**NOTE:** You will need method **public static string GetSalesWithAppliedDiscount(CarDealerContext context)** and **public StartUp** class.

Get all **sales** with information about the **car**, **customer** and **price** of the sale **with and without discount**. **Ignore any young driver discount in this calculation!**

* **Retrieve Sales Data**
  + Fetch all sales including details about the car (make, model, and traveled distance), customer, and the prices
* **Discount Handling**
  + Convert the discount value **from decimal** (as defined in the data model) **to an integer percentage**
* **Price Calculation**
  + **Price:** Should be represented as a **decimal value**
  + **Price with Discount:** Convert the discount to a decimal and calculate the price after applying the discount. The result should be a decimal value rounded to four decimal places
* Formatting Requirements
  + **Price and Price with Discount:** Ensure these values are rounded to **four decimal places** when exporting

**Return** the list of suppliers **to XML** in the format provided below.

|  |
| --- |
| **sales-discounts.xml** |
| <?xml version="1.0" encoding="utf-16"?>  <sales>  <sale>  <car make="Opel" model="Omega" traveled-distance="109910837" />  <discount>30</discount>  <customer-name>Zada Attwoood</customer-name>  <price>330.97</price>  <price-with-discount>231.68</price-with-discount>  </sale>  ...  </sales> |