

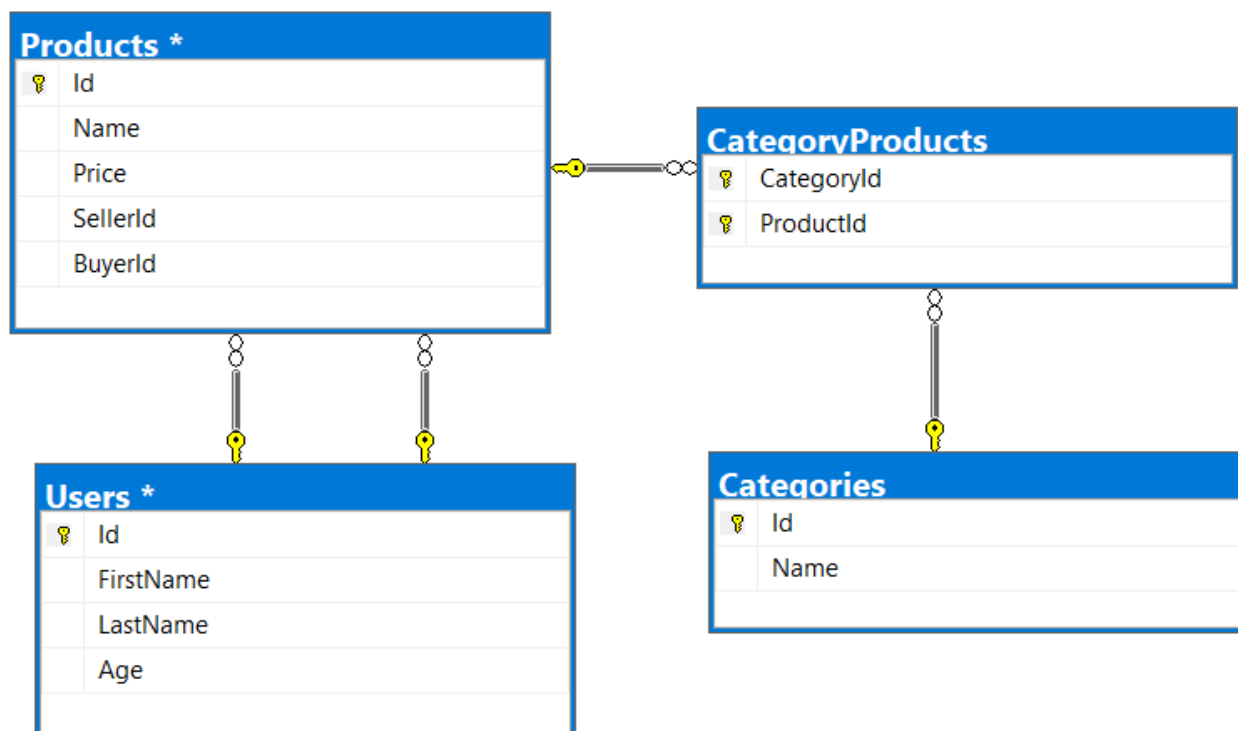
# Exercises: XML Processing

This document defines the **exercise assignments** for the ["Databases Advanced – EF Core" course @ Software University](#).

## Product Shop 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** and **age** (optional).
- Products have an **id**, **nam**, **price**, **buyerId** (optional) and **sellerId** as IDs of users.
- Categories have an **id** and **name**.
- Using Entity Framework 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**

## 1. 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 string with 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 string with 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 in the database. Just skip the record and continue.

Your method should return string with 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 category or product id, doesn't exists, skip the whole entry!

Your method should return string with message `"Successfully imported {CategoryProducts.Count}"`;

## 2. 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 5. 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
<pre>&lt;?xml version="1.0" encoding="utf-16"?&gt; &lt;Products&gt;   &lt;Product&gt;     &lt;name&gt;TRAMADOL HYDROCHLORIDE&lt;/name&gt;     &lt;price&gt;516.48&lt;/price&gt;   &lt;/Product&gt;   &lt;Product&gt;     &lt;name&gt;Allopurinol&lt;/name&gt;</pre>

```

    <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. 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 **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**. Take top **5** records.

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

users-sold-products.xml
<pre> &lt;?xml version="1.0" encoding="utf-16"?&gt; &lt;Users&gt;   &lt;User&gt;     &lt;firstName&gt;Almire&lt;/firstName&gt;     &lt;lastName&gt;Ainslee&lt;/lastName&gt;     &lt;soldProducts&gt;       &lt;Product&gt;         &lt;name&gt;olio activ mouthwash&lt;/name&gt;         &lt;price&gt;206.06&lt;/price&gt;       &lt;/Product&gt;       &lt;Product&gt;         &lt;name&gt;Acnezzol Base&lt;/name&gt;         &lt;price&gt;710.6&lt;/price&gt;       &lt;/Product&gt;       &lt;Product&gt;         &lt;name&gt;ENALAPRIL MALEATE&lt;/name&gt;         &lt;price&gt;210.42&lt;/price&gt;       &lt;/Product&gt;     &lt;/soldProducts&gt;   &lt;/User&gt;... &lt;/Users&gt; </pre>

## Query 7. 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.

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

categories-by-products.xml
<pre> &lt;?xml version="1.0" encoding="utf-16"?&gt; &lt;Categories&gt;   &lt;Category&gt; </pre>

```

<name>Garden</name>
<count>23</count>
<averagePrice>709.94739130434782608695652174</averagePrice>
<totalRevenue>16328.79</totalRevenue>
</Category>
<Category>
  <name>Adult</name>
  <count>22</count>
  <averagePrice>704.41</averagePrice>
  <totalRevenue>15497.02</totalRevenue>
</Category>
...
</Categories>

```

## Query 8. 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
<pre> &lt;Users&gt;   &lt;count&gt;54&lt;/count&gt;   &lt;users&gt;     &lt;User&gt;       &lt;firstName&gt;Cathee&lt;/firstName&gt;       &lt;lastName&gt;Rallings&lt;/lastName&gt;       &lt;age&gt;33&lt;/age&gt;       &lt;SoldProducts&gt;         &lt;count&gt;9&lt;/count&gt;         &lt;products&gt;           &lt;Product&gt;             &lt;name&gt;Fair Foundation SPF 15&lt;/name&gt;             &lt;price&gt;1394.24&lt;/price&gt;           &lt;/Product&gt;           &lt;Product&gt;             &lt;name&gt;IOPE RETIGEN MOISTURE TWIN CAKE NO.21&lt;/name&gt;             &lt;price&gt;1257.71&lt;/price&gt;           &lt;/Product&gt;           &lt;Product&gt;             &lt;name&gt;ESIKA&lt;/name&gt;             &lt;price&gt;879.37&lt;/price&gt;           &lt;/Product&gt;           &lt;Product&gt;             &lt;name&gt;allergy eye&lt;/name&gt;             &lt;price&gt;426.91&lt;/price&gt;           &lt;/Product&gt;         &lt;/products&gt;       &lt;/SoldProducts&gt;     &lt;/User&gt;     ...   &lt;/users&gt; &lt;/Users&gt; </pre>

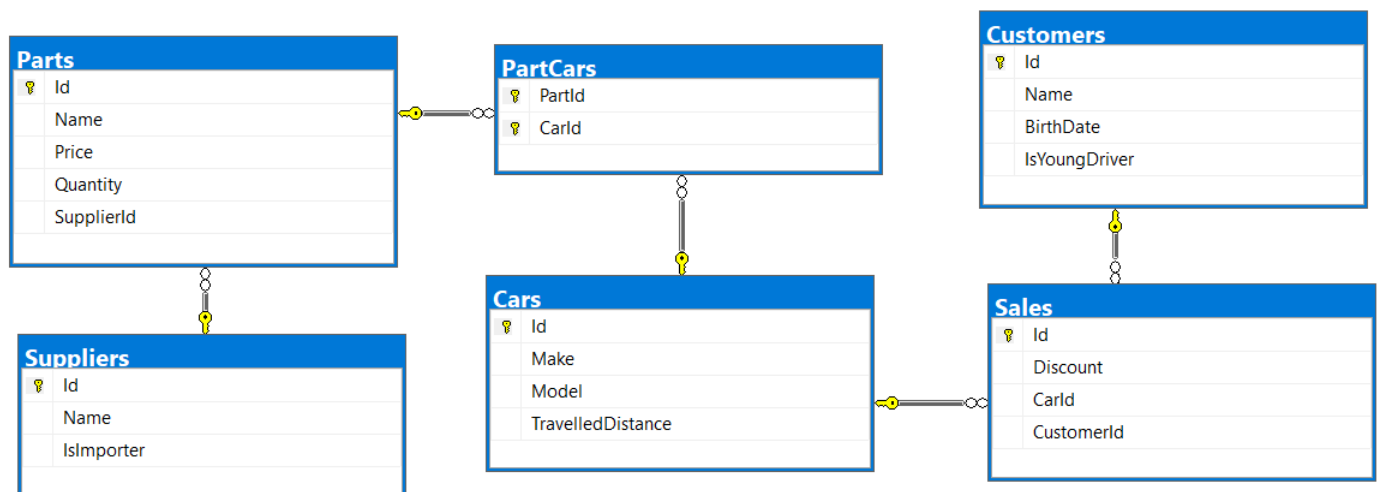
# Car Dealer Database

## 1. 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**
- **Sale** has **car**, **customer** and **discount percentage**

A **price** of a car is formed by **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
- Each **sale** has **one customer** and a **customer** can buy **many cars**

## 2. 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 string with 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 string with 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 part id doesn't exist, skip the part record.

Your method should return string with 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 string with 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 string with message `"Successfully imported {sales.Count}"`;

## 3. 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. Cars With Distance

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

Get all **cars** with distance 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
<pre>&lt;?xml version="1.0" encoding="utf-16"?&gt; &lt;cars&gt;   &lt;car&gt;     &lt;make&gt;BMW&lt;/make&gt;     &lt;model&gt;1M Coupe&lt;/model&gt;     &lt;travelled-distance&gt;39826890&lt;/travelled-distance&gt;</pre>

```

</car>
<car>
  <make>BMW</make>
  <model>E67</model>
  <travelled-distance>476830509</travelled-distance>
</car>
<car>
  <make>BMW</make>
  <model>E88</model>
  <travelled-distance>27453411</travelled-distance>
</car>
...
</cars>

```

## Query 15. 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 **travelled distance descending**.

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

bmw-cars.xml
<pre> &lt;cars&gt;   &lt;car id="7" model="1M Coupe" travelled-distance="39826890" /&gt;   &lt;car id="16" model="E67" travelled-distance="476830509" /&gt;   &lt;car id="5" model="E88" travelled-distance="27453411" /&gt;   ... &lt;/cars&gt; </pre>

## Query 16. 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
<pre> &lt;?xml version="1.0" encoding="utf-16"?&gt; &lt;suppliers&gt;   &lt;suplier id="2" name="VF Corporation" parts-count="3" /&gt;   &lt;suplier id="5" name="Saks Inc" parts-count="2" /&gt;   ... &lt;/suppliers&gt; </pre>

## Query 17. 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 **travelled distance** and for the **parts** get only **name** and **price** and sort all parts by price (descending). Sort all cars by travelled distance (**descending**) then by model (**ascending**). Select top 5 records.

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

cars-and-parts.xml
<pre>&lt;?xml version="1.0" encoding="utf-16"?&gt; &lt;cars&gt;   &lt;car make="Opel" model="Astra" travelled-distance="516628215"&gt;     &lt;parts&gt;       &lt;part name="Master cylinder" price="130.99" /&gt;       &lt;part name="Water tank" price="100.99" /&gt;       &lt;part name="Front Right Side Inner door handle" price="100.99" /&gt;     &lt;/parts&gt;   &lt;/car&gt;   ... &lt;/cars&gt;</pre>

## Query 18. 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**.

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

customers-total-sales.xml
<pre>&lt;?xml version="1.0" encoding="utf-16"?&gt; &lt;customers&gt;   &lt;customer full-name="Taina Achenbach" bought-cars="1" spent-money="5588.17" /&gt;   &lt;customer full-name="Johnette Derryberry" bought-cars="1" spent-money="2694.84" /&gt;   &lt;customer full-name="Jimmy Grossi" bought-cars="1" spent-money="2366.38" /&gt;   ... &lt;/customers&gt;</pre>

## Query 19. 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**.

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

sales-discounts.xml
<pre>&lt;?xml version="1.0" encoding="utf-16"?&gt; &lt;sales&gt;   &lt;sale&gt;     &lt;car make="BMW" model="M5 F10" travelled-distance="435603343" /&gt;     &lt;discount&gt;30.00&lt;/discount&gt;     &lt;customer-name&gt;Hipolito Lamoreaux&lt;/customer-name&gt;     &lt;price&gt;707.97&lt;/price&gt;     &lt;price-with-discount&gt;495.58&lt;/price-with-discount&gt;   &lt;/sale&gt;   ... &lt;/sales&gt;</pre>