# Exercises: XML Processing

This document defines the **exercise assignments** for the ["Databases Advanced – Entity Framework" course @ Software University.](https://softuni.bg/trainings/1442/databases-advanced-entity-framework-october-2016)

## Create Students XML Document

Create a XML document students.xml, 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).

## Add XML Namespace

Explore <http://en.wikipedia.org/wiki/Uniform_Resource_Identifier> to learn more about URI, URN and URL definitions. Add default namespace for the students "urn:students".

## Create XSD Schema

Create a **XSD Schema** **students.xsd** for the students XML document. You may generate the XSD schema with Visual Studio or other tool. Add a new optional element in the XSD schema: **endorsements** (author, author email, endorsement date, endorsement text). Validate your XML by the XSD schema.

## \*\* XSL Stylesheet

Write a **XSL stylesheet** to visualize the students as HTML. Test it in your favourite browser.

\* Some browsers might have security restrictions that prevent XSL rendering from local files. In this case, use different browser or upload your files to a Web server.

## Catalog of Musical Albums in XML Format

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>.

## Extract Album Names

Write a program that extracts **all album names** from **catalog.xml**

## Extract All Artists Alphabetically

Write a program that extracts **all artists** in alphabetical order from **catalog.xml**. Keep the artists in a SortedSet<string> to avoid duplicates and to keep the artist in alphabetical order.

## Extract Artists and Number of Albums

Write a program that extracts **all different artists**, which are found in the **catalog.xml**. For each artist print the **number of albums** in the catalogue. Use a Dictionary<string, int> (use the artist name as key and the number of albums as value in the dictionary).

## Delete Albums

Write a program to **delete** from catalog.xml all albums having price > 20. Save the result in a new file cheap-albums-catalog.xml.

## Old Albums

Write a program, which extract from the file catalog.xml the titles and prices for all albums, published 5 years ago or earlier.

## \* Directory Contents as XML

Write a program to traverse given directory and write to a **XML file** its contents together with all subdirectories and files. Use tags <file> and <dir> with attributes. Sample output:

|  |
| --- |
| <?xml version="1.0" ?>  <root-dir path="C:\Example">  <dir name="docs">  <file name="tutorial.pdf" />  <file name="TODO.txt" />  <file name="Presentation.pptx" />  </dir>  <dir name="photos">  <dir name="birthday-4-march">  <file name="friends.jpg" />  <file name="the\_cake.jpg" />  <file name="baloons.jpg" />  </dir>  <dir name="travel">  <file name="IMG24152.jpg" />  </dir>  </dir>  </root-dir> |

\* Hint: search in Internet for "*directory traversal C# recursion*".

## \* XML Schema Validation

Using Visual Studio generate an **XSD schema** for the file catalog.xml. Write a C# program that takes an **XML file** and an **XSD file** (schema) and **validates the XML** file against the schema. Test it with valid XML catalogs and invalid XML catalogs.

## \* XML to HTML through XSL Stylesheet

Create an **XSL stylesheet**, which transforms the file catalog.xml into HTML document, formatted for viewing in a standard Web-browser.

Write a C# program to apply the XSLT stylesheet transformation on the file catalog.xml using the class XslTransform.

## \*\*\* XML to HTML through XQuery

Read some tutorial about the **XQuery** language. Implement the XML to HTML transformation with XQuery (instead of XSLT). Download some **open source XQuery library for .NET** and execute the XQuery to transform the catalog.xml to HTML.

# Product Shop Database

In the next exercises you will be required to **use the models** from [previous exercise for JSON processing](https://softuni.bg/downloads/svn/DB-Fundamentals/DB-Advanced-EntityFramework/Oct-2016/08.%20DB-Advanced-EntityFramework-JSON-Processing/08.%20DB-Advanced-EntityFramework-JSON-Processing-Exercises.zip).

## Seed the Database

**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.

#### Query 1 - Products In Range

Get all products in a specified **price range** (e.g. 500 to 1000) which have **no buyer**. Order them by price (from lowest to highest). Select only the **product name**, **price** and the **full name** **of the seller**. Export the result to XML.

|  |
| --- |
| **products-in-range.xml** |
| <?xml version="1.0" encoding="utf-8"?>  <products>  <product name="TRAMADOL HYDROCHLORIDE" price="516.46" seller="Christine Gomez" />  <product name="Allopurinol" price="518.50" seller="Kathy Gilbert" />  <product name="Parsley" price="519.06" seller="Jacqueline Perez" />  ...  </products> |

#### Query 2 - Successfully Sold Products

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

|  |
| --- |
| **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>  <buyer-first-name>Anna</buyer-first-name>  <buyer-last-name>Parker</buyer-last-name>  </product>  <product>  <name>Warfarin Sodium</name>  <price>1379.79</price>  <buyer-first-name>Brandon</buyer-first-name>  <buyer-last-name>Fuller</buyer-last-name>  </product>  ...  </sold-products>  </user>  ...  </users> |

#### Query 3 - 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> |

#### Query 4 - 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

In the next exercises you will be required to **use the models** from [previous exercise for JSON processing](https://softuni.bg/downloads/svn/DB-Fundamentals/DB-Advanced-EntityFramework/Oct-2016/08.%20DB-Advanced-EntityFramework-JSON-Processing/08.%20DB-Advanced-EntityFramework-JSON-Processing-Exercises.zip).

## Car Dealer 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%).

## Car Dealer 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 1 – Ordered Customers

Get all **customers** ordered by their **birth date ascending**. If two customers are born on the same date **first print those who are not young drivers** (e.g. print experienced drivers first). **Export** the list of customers **to XML** in the format provided below.

|  |
| --- |
| **ordered-customers.xml** |
| <?xml version="1.0" encoding="utf-8"?>  <customers>  <customer>  <id>29</id>  <name>Louann Holzworth</name>  <birth-date>1960-10-01T00:00:00</birth-date>  <is-youn-driver>false</is-young-driver>  </customer>  <customer>  <id>28</id>  <name>Donnetta Soliz</name>  <birth-date>1963-10-01T00:00:00</birth-date>  <is-youn-driver>false</is-young-driver>  </customer>  ...  </customers> |

#### Query 2 – Cars from make Toyota

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

|  |
| --- |
| **toyota-cars.xml** |
| <?xml version="1.0" encoding="utf-8"?>  <cars>  <car id="117" make="Toyota" model="Camry Hybrid" travelled-distance="954775807" />  <car id="112" make="Toyota" model="Camry Hybrid" travelled-distance="92275807" />  ...  </cars> |

#### Query 3 – 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> |

#### Query 4 – 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> |

#### Query 5 – 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> |

#### Query 6 – 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>  <car make="Mercedes" model="W124" travelled-distance="2147647" />  <customer-name>Carri Knapik</customer-name>  <discount>0.2</discount>  <price>254.96999999999997</price>  <price-with-discount>203.97599999999997</price-with-discount>  </sale>  ...  </sales> |