# Database Applications Exam (March 2015) – Geography

Your exam consists of several parts, explained below. You may work independently on each exam part. Submit your solutions as single **ZIP file holding the full source code**, without any libraries and compiled binaries.

## Part I – Existing Database: Import, Queries, Export

You are given a **MS SQL Server database "Geography"** holding continents, countries, currencies, monasteries and rivers, available as SQL script. Your task is to write a few data-driven applications in C# for importing data, querying data and exporting data from the database.

### Entity Framework Mappings (Database First)

Create an **Entity Framework (EF) data model** of the existing database (map the database tables to C# classes). Use the "**database first**" model in EF.

5 score

### Export Monasteries by Country as XML

Write a **C# application** based on your EF data model for exporting all monasteries by country in a XML file named monasteries.xml in the following XML format:

|  |
| --- |
| **monasteries.xml** |
| <?xml version="1.0" ?>  <monasteries>  <country name="Bulgaria">  <monastery>Bachkovo Monastery “Virgin Mary”</monastery>  <monastery>Rila Monastery “St. Ivan of Rila”</monastery>  <monastery>Troyan Monastery “Holy Mother's Assumption”</monastery>  </country>  <country name="Bhutan">  <monastery>Taktsang Palphug Monastery</monastery>  </country>  …  </monasteries> |

Order the countries **alphabetically** and the monasteries in each country also alphabetically. Exclude all countries with no monasteries. For better performance, ensure your program executes a single DB query.

15 score

### Export Rivers as JSON

Write a **C# application** based on your EF data model for exporting all rivers along with their countries in a JSON file named rivers.json in the following JSON format:

|  |
| --- |
| **rivers.json** |
| [  { "river-name": "Nile", "river-length": 6650, "countries": ["Burundi","Democratic Republic of the Congo","Egypt","Eritrea","Ethiopia","Kenya","Rwanda","South Sudan","Sudan","Tanzania","Uganda"]},  { "river-name": "Amazon", "river-length": 6650, "countries": ["Bolivia","Brazil","Colombia","Ecuador","Guyana","Peru","Venezuela"]},  …  ] |

Order the **rivers** by **length** (from the longest) and the countries for each river **alphabetically**. Include also the rivers with no countries (if any). For better performance, ensure your program executes a single DB query.

15 score

### Import Rivers from XML

Write a **C# application** based on your EF data model for **importing into the DB a set of rivers** given in the XML file rivers.xml. The rivers come in the following XML format:

|  |
| --- |
| **rivers.xml** |
| <?xml version="1.0" ?>  <rivers>  <river>  <name>Maritsa</name>  <length>480</length>  <outflow>Aegean Sea</outflow>  <countries>  <country>Bulgaria</country>  <countries>  </river>  <river>  <name>Madre de Dios</name>  <length>1130</length>  <drainage-area>125000</drainage-area>  <average-discharge>4915</average-discharge>  <outflow>Beni River</outflow>  <countries>  <country>Peru</country>  <country>Bolivia</country>  <countries>  </river>  …  </rivers> |

The name, length and outflow elements are **mandatory**. The drainage-area, average-discharge and countries elements are **optional**.

You should **parse the XML** and throw an exception in case of incorrect data, e.g. when a required element is missing or an invalid value is given. The size of the XML file will be less than **10 MB**. Use an XML parser by choice.

8 score

You should correctly **import the rivers into the DB**.

7 score

You should correctly import the **countries for each river** into the DB.

5 score

### Entity Framework Mappings (Code First)

Create an **Entity Framework (EF) data model** of the existing database (map the database tables to C# classes). Use the "**code first**" model in EF.

5 score

### EF Migrations: Add Entities for Mountains and Peaks

You need to store **mountains** and **peaks** in the database. Each **mountain** has a **name** and may belong to **multiple countries**. Each **peak** has a **name**, **elevation** and **mountain**.

Modify your EF data models to **add the new entities** (mountains and peaks). Enable the **automatic migrations in EF** and ensure the database schema is correctly migrated to support the new EF data model.

Submit an **entity relationship diagram** (PNG or JPG image file) of all your tables in the database.

10 score

### Import Mountains and Peaks from JSON

Write a **C# application** based on your EF code first data model for **importing into the DB a set of mountains and peaks** given in the JSON file mountains.json. The mountains and peaks come in the following JSON format:

|  |
| --- |
| **mountains.json** |
| [  { "mountainName" : "Andes",  "peaks" : [  { "peakName" : "Aconcagua", "elevation" : 6962 },  { "peakName" : "Monte Pissis", "elevation" : 6793 },  { "peakName" : "Ojos del Salado", "elevation" : 6893 } ],  "countries" : [ "Argentina", "Chile"] },  { "mountainName" : "Unknown Mountains" },  { },  { "peaks" : [ { "peakName" : "Incorrect peak" } ],  "mountainName" : "Strange Mountain" },  { "mountainName" : "Caucasus",  "peaks" : [  { "peakName" : "Dykh-Tau", "elevation" : 5205 },  { "peakName" : "Elbrus", "elevation" : 5642 } ],  "countries" : [ "Russia", "Georgia"] }  ] |

The properties mountainName, peakName and elevation are **mandatory**. The properties peaks and countries are **optional**.

You should **parse the JSON** and throw an exception in case of incorrect data, e.g. when a required element is missing or an invalid value is given. The size of the JSON file will be less than **10 MB**. Use a JSON parser by choice.

5 score

You should correctly **import the mountains into the DB**.

3 score

You should correctly import the **countries for each mountain** into the DB.

3 score

You should correctly import the **peaks for each mountain** into the DB.

4 score

Insert each mountain in a **separate transaction**. A mountain should either be inserted correctly along with all its peaks and countries, or no part of it should be inserted at all. Print as **output** a single line for each mountain from the input JSON: either "*Mountain <name> imported*" or "*Error: <error message>*". Error messages should describe briefly the problem (as free text) and may optionally include exception stack-trace. Sample output:

|  |
| --- |
| Mountain Andes imported  Mountain Unknown Mountains imported  Error: Missing mountain name  Error: Missing peak elevation  Mountain Caucasus imported |

5 score

### Rivers by Country Query

Write a **C# application** based on your EF data model for **listing all rivers that pass through a given set of countries**. The application should process multiple queries and produce an **XML output**. Each query might have an optional attribute max-results="*xxx*", that limits the results up to *xxx* items.

The input comes from an XML file rivers-query.xml in the following format:

|  |
| --- |
| **rivers-query.xml** |
| <?xml version="1.0" ?>  <queries>  <query max-results="2">  <country>Brazil</country>  </query>  <query max-results="20">  <country>Mexico</country>  <country>Afganistan</country>  </query>  <query>  <country>Bulgaria</country>  <country>Romania</country>  <country>Austria</country>  </query>  </rivers> |

The **output** should be printed on the console in the following **XML format**:

|  |
| --- |
| <?xml version="1.0" encoding="utf-8" ?>  <results>  <rivers total-count="6" listed-count="2">  <river name="Amazon" />  <river name="Madeira" />  </rivers>  <rivers total-count="0" listed-count="0" />  <rivers total-count="1" listed-count="1">  <river name="Danube" />  </rivers>  </rivers> |

The query results should come in the same order like the queries in the input. Display in the output the **total count** of matched rivers and the **listed count**: the smaller of total count and max results.

30 score

## Exam Information

You are allowed to use any resources you have, e.g. Internet, software, existing code.

You are not allowed to get help from other people. Skype, ICQ, FB, email, talks, phone calls, etc. are forbidden.

Exam time: **6 hours**.