# Database Applications Exam (March 2015) – Media

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

## Part I – Query Existing Database, Import and Export Data

You are given a **MS SQL Server database "Diablo"** holding users, games, items, characters and statistics, available as а **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. To test your EF data model, **list all characters' names**.

3 score

### Export Characters and Players as JSON

Write a **C# application** based on your EF data model for **exporting all characters along with the names of the players who played them** in the following JSON format:

|  |
| --- |
| **characters.json** |
| [  { "name":"Amazon", "playedBy":[ "bindbawdy", "admincuttle", ... ] },  { "name":"Assassin", "playedBy":[ "rotoriginally", "brooklyndecent", ... ] },  { "name":"Barbarian", "playedBy":[ "Petya", "slushyinflate", ... ] },  { "name":"Crusader", "playedBy":[ "nakov", "VGeorgiev", "unfriendlyfurther", "Pesho", "baleremuda", "Alex", ... ] },  ...  ] |

Write the output in a JSON file named **characters.json**. The code indentation in the JSON file is not important.

4 score

Order the **characters** **alphabetically**.

2 score

For better performance, ensure your program executes a **single DB query** and retrieves from the database only the required data (without unneeded rows and columns).

4 score

### Export Finished Games as XML

Write a **C# application** based on your EF data model for **exporting all finished games and their players** in a XML file named finished-games.xml in the following XML format:

|  |
| --- |
| **finished-games.xml** |
| <?xml version="1.0" encoding="utf-8"?>  <games>  <game name="Ablajeck" duration="5">  <users>  <user username="potterssnarky" ip-address="129.74.12.234" />  <user username="hurdlesurmise" ip-address="81.91.116.195" />  </users>  </game>  <game name="Ablajeck" duration="9">  <users>  <user username="advisersspry" ip-address="100.97.183.157" />  </users>  </game>  ...  <game name="Amsterdam" duration="7">  <users>  <user username="VGeorgiev" ip-address="74.212.145.183" />  <user username="posteriorflowery" ip-address="105.26.72.120" />  <user username="hurdlesurmise" ip-address="81.91.116.195" />  </users>  </game>  <game name="Apple" duration="6">  <users />  </game>  ...  <game name="Safflower">  <users>  <user username="Stamat" ip-address="180.166.72.243" />  </users>  </game>  ...  </games> |

**Each game** should have a **name** and **duration** (may be missing). Each **user** should have a **username** and **IP address**. Use an XML parser by choice. Attach the **duration** **attribute** only if the duration is not null.

6 score

For better performance, ensure your program executes a **single DB query** and retrieves from the database only the required data (without unneeded rows and columns).

4 score

**Sort by name and duration!!!!**

### Import Users and Their Games from XML

Write a **C# application** based on your EF data model for **importing users and their games**.

The input comes from an XML file users-and-games.xml in the following format:

|  |
| --- |
| **users-and-games.xml** |
| <?xml version="1.0" encoding="utf-8"?>  <users>  <user first-name="Jose" last-name="Diaz" username="jdiaz0" is-deleted="2" ip-address="103.167.134.170" registration-date="24/06/2015">  <games>  <game>  <game-name>Japhette orchid</game-name>  <character name="Druid" cash="9567.00" level="30" />  <joined-on>13/11/2011</joined-on>  </game>  <game>  <game-name>Amsterdam</game-name>  <character name="Druid" cash="9208.00" level="32" />  <joined-on>10/12/2014</joined-on>  </game>  ...  </games>  </user>  <user last-name="Morris" username="bmorrisc" email="mmorrisc@craigslist.org" is-deleted="1" ip-address="239.14.112.90" registration-date="17/06/2014">  <games>  <game>  <game-name>Moscow</game-name>  <character name="Crusader" cash="7548.00" level="29" />  <joined-on>20/12/2015</joined-on>  </game>  ...  </games>  </user>  </users> |

The input XML holds a sequence of requests given in the **<user>…</user>** elements.

The user attributes **first-name**, **last-name** and **email** are optional.

The **game elements** describe **existing games** where the users should be added with their respective **character**, **level**, **cash** and the **date they joined**.

Running the program for the first time should yield the following results:

|  |
| --- |
| **Successfully added user jdiaz0**  **User jdiaz0 successfully added to game Japhette orchid**  **User jdiaz0 successfully added to game Amsterdam**  **User jdiaz0 successfully added to game Wellington**  **User jdiaz0 successfully added to game Bucharest**  **User jdiaz0 successfully added to game Boston**  **User jdiaz0 successfully added to game Milan**  **Successfully added user rlane1**  **User rlane1 successfully added to game Star of Bethlehem**  **...** |

Running the program again should produce the following results:

|  |
| --- |
| **User jdiaz0 already exists**  **User rlane1 already exists**  **...** |

Users with those **usernames** have already been imported and the program should not allow duplicate imports.

Your program should correctly **parse the input XML**.

5 score

Your program should correctly **import users**.

7 score

Your program should correctly **add users to the games**. If a game **fails to add**, neither the user, nor any other of his games should be added.

12 score

### Part II – EF Code First: Define Data Model, Import and Export Data

You are assigned to define a **code first data model in EF** and write a few data-driven applications in C# for importing data, querying data and exporting data from the database.

Use a new database "**Movies**". Do not modify the "**Diablo**" database.

### EF Code First: Movies

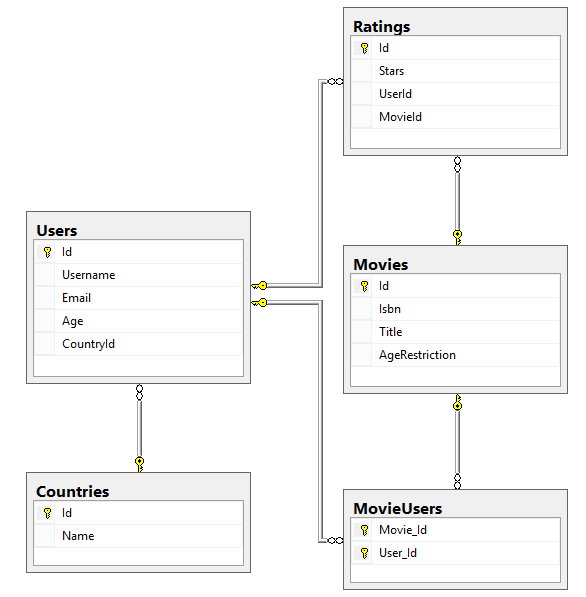
Create an **Entity Framework (EF) code first data model** holding **users** and **movies**. Users can have **multiple favourite movies** and can **give ratings** to movies (from **0 to 10**).

* **Countries** have **id** and **name**
* **Users** have **id**, **username** (mandatory, minimum 5 characters), **email** (optional) and **age** (optional)
* **Movies** have **id**, **isbn** (mandatory), **title** (mandatory, from 2 to 100 characters) and **age** **restriction** (either Child, Teen or Adult)
* **Ratings** are given by users for movies and can be in the range **0..10**

Configure the following relations:

* **Users** can have only **one country**. **Countries** can have **many users**.
* **Users** can have **many favourite movies**.
* **Users** can give **ratings** to **many movies**. A user cannot rate a given movie more than once, though.

Put **navigation properties** on **both sides** of the relations.



20 score

Seed the database by importing the provided JSON files (**countries.json**, **users.json**, **movies.json**, **users-and-favourite-movies.json**, **movie-ratings.json**).

10 score

### Query the Database

Query the newly created Code First database using Entity Framework. Export the results as JSON in the format specified.

#### Adult Movies

Export all **adult movies**. Order them by **title** and by **ratings received** as secondary criteria. Select each movie's **title** and the **number of ratings** received.

|  |
| --- |
| **adult-movies.json** |
| [  {  "title": "12 Years a Slave (2013)",  "ratingsGiven": 3  },  {  "title": "A Clockwork Orange (1971)",  "ratingsGiven": 5  },  {  "title": "Before Sunrise (1995)",  "ratingsGiven": 4  },  {  "title": "Ben-Hur (1959)",  "ratingsGiven": 3  },  ...  ] |

Correctly export **all adult movies** to JSON.

6 score

For better performance, ensure your program executes a **single DB query** and retrieves from the database only the required data (without unneeded rows and columns).

4 score

#### Rated Movies by User

Export **all movies rated by a given user** (e.g. user **jmeyery**). Select the user's **username** and **all movies** rated by him. **Order** the movies by **title** alphabetically. Select each movie's **title**, the **rating** given by the user and its **average rating**.

|  |
| --- |
| **rated-movies-by-jmeyery.json** |
| {  "username": "jmeyery",  "ratedMovies": [  {  "title": "Indiana Jones and the Last Crusade (1989)",  "userRating": 4,  "averageRating": 5.25  },  {  "title": "Kill Bill: Vol. 1 (2003)",  "userRating": 9,  "averageRating": 6.75  },  {  "title": "Mad Max: Fury Road (2015)",  "userRating": 8,  "averageRating": 5.0  },  ...  ]  } |

Correctly export **all rated movies** by **"jmeyery"** to JSON with the required data.

7 score

For better performance, ensure your program executes a **single DB query** and retrieves from the database only the required data (without unneeded rows and columns).

4 score

#### Top 10 Favourite Movies

Export the **top 10** most **favourite** (being **favourite to most users**) **teen** movies. Order them by the **number of users they are favourite to** (descending), then by **title** (ascending). Select each movie's **isbn** and **title**. Select also each user's **username** who favourited the movie.

|  |
| --- |
| **top-10-favourite-movies.json** |
| [  {  "isbn": "660097222-X",  "title": "Prisoners (2013)",  "favouritedBy": [  "tmurphy1",  "tgrant4",  "mgeorgeh",  "dburket",  "bhamilton1n",  "ksims1s"  ]  },  {  "isbn": "653288544-4",  "title": "Amores Perros (2000)",  "favouritedBy": [  "tmurphy1",  "tgrant4",  "jhowarda",  "tbennettb",  "jgibson1j"  ]  },  ...  ] |

Correctly export **the top 10 most favourite movies**.

8 score

For better performance, ensure your program executes a **single DB query** and retrieves from the database only the required data (without unneeded rows and columns).

4 score

## Exam Information

You are allowed to use any resources at your disposal, 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**.