Database Systems – Practical Exam – September 2014

Problem 1. Database First (10 points)

In **company**, information about different **employees** and **projects** is held. Each employee has **first** and **last name** (both between 5 and 20 letters, inclusive), **year salary**, **manager** and **department**. Each **department** has **unique name** between 10 and 50 letters, inclusive. Each employee works on many projects. Each **project** has **name** between 5 and 50 letters, inclusive, and **employees** working on it. Each employee also has **starting** and **ending date** for each of his/her **projects**. Every single piece of information is **mandatory**, except **manager** – some employees do not have one. Additionally, every employee has to report every day when he/she got to work. Each **report** has **time** of reporting.

Design a database schema "**Company**" in SQL Server to keep the information about **employees**, **departments**, **projects** and **reports**. Ensure data integrity is fulfilled.

Provide a **SQL script for your database schema** (without the data), a backup (.bak file) from the database and a **simple documentation of the relationships** between the tables. Additionally, provide a **picture of the design diagram** describing your created database.

Problem 2. Sample Data (15 points)

Use C# to implement an application for generating random sample data in the company. Create at least:

- **100 departments**
- **5 000 employees** – each employee has department, 95% of them have managers and their salary is between $50 000 and $200 000, inclusive. There must be no cycles in the management tree – example for a cycle is Pesho's manager is Gosho, Gosho's manager is Ivan and Ivan's manager is Pesho.
- **1 000 projects** – on each project there are working between 2 and 20 employees, inclusive – average of 5. Ending date is always after starting date (you don't say) and ending date may be in the future.
- **250 000 reports** – add average of 50 reports per employee.

* You are free to use an ORM framework by choice or plain ADO.NET.
* Adding a lot of data may be slow, depending on your implementation, so be patient. Consider turning off the automatic change tracking in Entity Framework, if you use it.
* All text can be random string with valid length depending on the field.

Problem 3. SQL Queries (18 points)

Write three queries in native SQL:

1. Get the full name (first name + " " + last name) of every employee and its salary, for each employee with salary between $100 000 and $150 000, inclusive. Sort the results by salary in ascending order.
2. Get all departments and how many employees there are in each one. Sort the result by the number of employees in descending order.
3. Get each employee's full name (first name + " " + last name), project's name, department's name, starting and ending date for each employee in project. Additionally get the number of all reports, which time of reporting is between the start and end date. Sort the results first by the employee id, then by the project id. (This query is slow, be patient!)

Provide three .sql files with the three queries in them.

Problem 4. Performance Tuning (7 points)

Create a "cache" table for the third query from problem 3 and save all the results there for future querying. Provide the .sql file, containing the stored procedures which creates the table and inserts the data. You should write two stored procedures – one for creating the table and one for updating the data.

Problem 5. Code First (15 points)

In different **cities**, **cars** from different **manufacturers and dealers** are sold. Each **manufacturer** has name (mandatory with maximum of 10 symbols). Each **car** has manufacturer, model (maximum of 11 symbols), transmission type, year, price and dealer. All fields are mandatory. **Transmission type** can be manual or automatic. **Dealers** have name (mandatory with maximum of 50 symbols) and cities, in which they sold cars. Each **city** has name (mandatory with maximum of 10 symbols). Each city and each manufacturer is unique by their name.

Design a database schema "**Cars**" using Code First approach with Entity Framework.

Problem 6. Import from JSON File (15 points)

Write a **C# program** to **parse a number of JSON files** with name format "data.*number*.json" holding a set of cars' information in the format given below and **insert them into the database**:

data.0.json

```
[
   {
      "Year":2014,
      "TransmissionType":0,
      "ManufacturerName":"Mazda",
      "Model":"6 Skyactiv",
      "Price":60000.0,
      "Dealer":{
         "Name":"Star Motors",
         "City":"Sofia"
      }
   },
   {
      "Year":2008,
      "TransmissionType":1,
      "ManufacturerName":"BMW",
      "Model":"320i Cabrio",
      "Price":30000.0,
      "Dealer":{
         "Name":"MM Auto",
         "City":"Sofia"
      }
   },
   {
      "Year":2003,
      "TransmissionType":0,
      "ManufacturerName":"Renaut",
      "Model":"Clio 1.4",
      "Price":5500.0,
      "Dealer":{
         "Name":"T Motopc",
         "City":"Sofia"
      }
   },
```

```
   {
      "Year":1997,
      "TransmissionType":0,
      "ManufacturerName":"Opel",
      "Model":"Tigra",
      "Price":5000.0,
      "Dealer":{
         "Name":"TA Motopc",
         "City":"Sofia"
      }
   }
]
```

* You are free to use a JSON parser by choice (or to parse the JSON without using a parser).

You should correctly **import the cars, manufacturers, dealers and cities into the DB**. Keep in mind that the cities and manufacturers are unique. If some city or manufacturer already exists in the database, it should be reused (no duplicates are allowed).

* You are free to use an ORM framework by choice (in code first).

Problem 7. Application Queries (20 points)

Implement a **C# program** for **searching for cars by given conditions**. It should be able to process a sequence of queries from the XML file **queries.xml** in the following format:

queries.xml

```
<?xml version="1.0"?>
<Queries xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <Query OutputFileName="Result0.xml">
    <OrderBy>Id</OrderBy>
    <WhereClauses>
      <WhereClause PropertyName="City" Type="Equals">Sofia</WhereClause>
      <WhereClause PropertyName="Year" Type="GreaterThan">1999</WhereClause>
    </WhereClauses>
  </Query>
</Queries>
```

Each query has different output file, provided in the "OutputFileName" attribute.

The queries can have two types of parameters - ordering and where clauses.

Ordering the results can be done by the following fields: Id, Year, Model, Price, Manufacturer (its name) and Dealer (its name).

Where clauses can be one or many. If many where clauses are provided, use "and" as logical operator between them. Where clauses have two attributes for defining the query – "PropertyName" and "Type". "PropertyName" defines by which car property the search should be done. "Type" defines the type of comparer to use in the search. The following list gives you the available property types and comparers for each of them:

- Id (number) -> Equals, GreaterThan, LessThan
- Year (number) -> Equals, GreaterThan, LessThan
- Price (floating point number) -> Equals, GreaterThan, LessThan
- Model (text) -> Equals, Contains
- Manufacturer (its name, text) -> Equals, Contains

- Dealer (its name, text) -> Equals, Contains
- City (its name) -> Equals

"GreaterThan" and "LessThan" does not include equal elements.

Write the results in the corresponding XML files in the following format:

```
<?xml version="1.0"?>
<Cars xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <Car Manufacturer="Mazda" Model="6 Skyactiv" Year="2014" Price="60000.00">
    <TransmissionType>manual</TransmissionType>
    <Dealer Name="Star Motors">
      <Cities>
        <City>Sofia</City>
      </Cities>
    </Dealer>
  </Car>
  <Car Manufacturer="BMW" Model="320i Cabrio" Year="2008" Price="30000.00">
    <TransmissionType>automatic</TransmissionType>
    <Dealer Name="MM Auto">
      <Cities>
        <City>Sofia</City>
      </Cities>
    </Dealer>
  </Car>
  <Car Manufacturer="Renaut" Model="Clio 1.4" Year="2003" Price="5500.00">
    <TransmissionType>manual</TransmissionType>
    <Dealer Name="TA Моторс">
      <Cities>
        <City>Sofia</City>
        <City>Kyustendil</City>
      </Cities>
    </Dealer>
  </Car>
</Cars>
```

Implement the search functionality correctly. Ensure it works fast enough for millions of records in the DB, and SQL injection is not possible.

* You are free to use an ORM framework by choice or plain ADO.NET.

* You are free to use XML parsers by choice, but take into account the performance.

Evaluation Criteria

The evaluation criteria are as follows:

- Correct and complete fulfillment of the requirements.
- Good technical design and appropriate use of technologies.
- High-quality programming code – correctness, readability, maintainability.
- Performance – highly-efficient code.

Other Terms

During the exam you are allowed to use any teaching materials, lectures, books, existing source code, and other paper or Internet resources.

The Telerik Academy Anti-cheat client should be turned on during the entire exam.

Direct or indirect communication with anybody in class or outside is forbidden. This includes, but does not limit to, technical conversations with other students, using mobile phones, chat software (Skype, ICQ, etc.), email communication, posting in forums, folder synchronization software (like Dropbox), etc.

Exam Duration

Students are allowed to work up to 8 hours.