# Programming Applications with Databases

## Exercise Set 4

1. In the AdventureWorksLT database there is a table SalesLT.Customer with a ModifiedDate attribute. Create a trigger which ensures that in case of customer data modification the actual server date and time is taken.
   **[1p]**

2. In the AdventureWorksLT database there is a table SalesLT.Product with StandardCost and ListPrice attributes. Create a table which will hold the cost and price history including the date and time when the change occurred (ensure the table name is consistent with the whole schema). Then create a trigger which will register all changes (and only changes) in StandardCost and ListPrice values including the mentioned date and time. Finally, we want to get a report where we can see all costs and prices with periods of time when they were in effect – think what more is needed to get that kind of report (if anything).
   *Hint: consider the full lifecycle of the product from creating to deleting.*
   **[2p]**

3. The most common use case for INSTEAD OF triggers is operating on views. Understand and execute the example presented in `https://www.sqlservertutorial.net/sql-server-triggers/sql-server-instead-of-trigger/`. During classes present the whole scenario with explanations.
   **[2p]**

4. Using triggers implement the foreign key policy in the following extended version: having book and specimen one-to-many association ensure that book may have maximum 5 specimens.
   **[1p]**

5. Explain the concept and present appropriate example for recursive triggers (the one from the attached examples can be reused).
   *Hint: the following code can be used to check the recursive triggers status:*

   ```sql
   SELECT name AS 'Database name', is_recursive_triggers_on AS 'Recursive Triggers Enabled'
   FROM sys.databases
   ```

   **[1p]**

6. Consider the following tables: *Cache(ID, UrlAddress, LastAccess)*, *History(ID, UrlAddress, LastAccess)* and *Parameters(Name, Value)*. The meaning of fields of *Cache* and *History* tables is as follows:

   - *ID* — row identifier,
   - *UrlAddress* — the address of a website,
   - *LastAccess* — the time of the last visit with with an accuracy of a second.

   The *Parameters* table contains only one row with *Name* = *max_cache* and *Value* set to the maximum size of the *Cache* table. At the time of inserting of a new row to *Cache* table the following conditions should be met:

   - If the cache already contains the address of the website being inserted, the last access time should be only modified,
   - Otherwise, the size of the cache should be compared to the value in the *max_cache* parameter from the *Parameters* table. There are two cases:
     - If it's lower, the new row should be inserted.
     - Otherwise, determine a website for which the last access is the oldest (if there are more than one, pick any of them). The corresponding row should be transferred to the *History* table, but in case it refers to a website already present there, the last access time should be only modified.

   This task should be implemented using triggers.
   **[3p]**

*Paweł Rajba*