Delia Alexandrina Mitrea, Călin Cenan, Paulina Mitrea

Relational Databases, SQL and Specific Data Structures, in Practice

Database Laboratory Guide - 2nd year

Technical University of Cluj-Napoca, 2017

Table of Contents

Introduction

The laboratory works described in this book are addressed to the students of the 2nd year, English section.

The main objectives of these works are the following:

- To understand what a database is;
- To learn what kind of structure a database should have, in order to keep the database consistent over time;
- To learn how to extract the desired information from a database by using queries;
- To learn how to effectively build a database within a Database Management System (DBMS) by creating the corresponding tables; defining relationships between the tables; defining constraints and referential integrity rules at the database level; creating user-defined views. In the corresponding practical exercises, Microsoft SQL Server, MySQL Workbench and JSON data structures will be used.

Also, as the Internet applications and databases are highly developed nowadays, during the laboratories the students study *how to build web applications* that communicate with databases. In this context, the students learn how to work with a web server (Apache), and they also study the HTML language, a server side scripting language (PHP).

The laboratory works guide the students to correctly assimilate the practical knowledge about the databases, to develop the corresponding technical skills, and also to learn the basic notions about the HTML and PHP.

All the assimilated knowledge is practiced again in the context of an individual assignment at the end of the term, when the students have to build a small web-site on their own, by implementing a basic web interface and also the most important operations performed with the database: data insertion, deletion, visualization, and update.

The software tools that are used during the database laboratories are the following:

- Microsoft SQL Server, together with SQL Server Management Studio
- MySQL Workbench
- Wamp (Apache + MySQL + PHP).
- HTML and PHP editor (Eclipse, NuSphere, PhpEdit**,** Notepad++**)**
- SQL interactive tutorial: http://www.sqlzoo.net/
- SQL Server- SQL Server Books Online

Laboratory no. 1

I. Brief theory reminder:

Definitions: A *database* defines *an organized collection of data* in an *optimum manner*; so, a **DATABASE** consists of an organized collection of data for one or more users, typically *in digital form*. A **DATABASE** must achieve the following properties [1]:

- *consistency*, *easy and fast access to the data*, *comprehensibility*, *no redundancy*, *minimum storage space*
- *utility*: to store the data of various activity domains; to be able to stand at the basics of building dynamic web-sites, or environments for technological design; to be able to store large and very large amounts of data for automatic decision making

The **DATA** consist of facts/objects represented according to some conventions, gathered from the real life through observations and measurements The **INFORMATION** is the result of the data interpretation by a certain subject (including computers), who gets the capacity of making decisions. **DATA** are transposed into **INFORMATION** only after they are processed through interaction with a system capable of interpreting them.

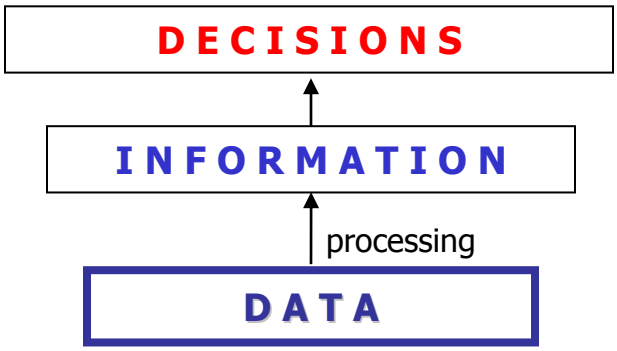

Figure 1. Data versus Information

A **DATABASE MANAGEMENT SYSTEM** or **DBMS** is a software designed to assist in maintaining and utilizing large collections of data. An alternative for using a DBMS is to use ad hoc approaches **that do not carry over from one application to another:** store data in files and write application-specific code to manage it.

All **DBMS** performs **3 main functions**:

- data definition
- data manipulation
- user interface.

Many **other functions** that can be carried out:

- data security
- data integrity
- data access sharing; data access control
- data recovery.

The ease with which information can be obtained from a database usually determines the value of the database from the user's point of view. Relational database systems allow a rich variety of questions to be posed easily, while older database systems do not have this facility, this feature having a great contribution to their popularity. Questions referring to the data stored within a DBMS are called **queries**. The DBMS offers **the query language**, a specialized language, in which queries can be formulated. The query language support constitutes a very attractive feature of the relational model [1]. The **Structured Query Language (SQL)** was firstly employed by IBM, when developing System-R, the pioneering relational DBMS. Along the years, SQL became the query language most widely used for creating, manipulating and querying relational DBMSs.

The basic structure of an SQL query is:

SELECT *select_list* [INTO *new_table*]
FROM *table_list*
WHERE $condition_1$ AND $condition_2$ AND... AND $condition_n$
GROUP BY $field_1$, $field_2$,..., fieldn
HAVING *group_condition*
ORDER BY $field_1$, $field_2$,..., $field_n$ [ASC|DESC]

The *select_list* has the following form [2]:
SELECT [ALL | DISTINCT] [TOP n [PERCENT] [WITH TIES]] {* |{table_name |view_name|table_alias}.* |{column_name|expression} [[AS] column_alias]} [,n]

In Microsoft SQL Server, TOP ***n*** [PERCENT] returns only the first ***n*** tuples of the result. If the PERCENT option is also specified, then only the first ***n*** percents of the result tuples are provided. The value of ***n*** must be between *0* and ***100***. WITH TIES specifies that if there exist multiple tuples with the same values of the ORDER BY criteria and they are among the last candidates, then these tuples will be included in the result. In MySQL, a similar intention could be expressed by the ***LIMIT m,n*** clause, placed after the ORDER BY clause, specifying how many tuples must be retained from the final ordered list (***n***), starting from the index m ($m \geq 0$).

The table_list of the FROM clause contains a list of table names, views, respectively table or view aliases, separated by comma, or connected through the JOIN operator. If the *table_list* consists of comma separated items, then the JOIN conditions

should be specified in the WHERE clause. Otherwise, the WHERE clause usually contains only search conditions.

II. Assignments:

1.) Read about databases in http://en.wikipedia.org/wiki/Database
2.) Read more about the SQL SELECT Phrase at:
http://msdn.microsoft.com/en-us/library/ms189499%28v=sql.100%29.aspx
3.) Solve the queries from the interactive web-site:
http://www.sqlzoo.net

Sections: 0.) Select basics; 1.) Select name; 2.) Select from World; 3.) Select from Nobel

- Use the following options: Microsoft SQL Server; MySQL

Laboratory no. 2

I. Brief theory reminder:

The structure of the SQL SELECT phrase is given below:

SELECT *select_list, AGG_function(field$_i$)*
FROM *table_list*
WHERE *condition$_1$* AND *condition$_2$* AND... AND *condition$_n$*
GROUP BY field$_1$, field$_2$,..., fieldn
HAVING *group_condition*
ORDER BY *field$_1$, field$_2$,..., field$_n$* [ASC|DESC]

- **The JOIN operation**

Table_list in the FROM clause usually has one of the forms:

((table$_1$ JOIN table$_2$ ON join_cond$_1$) JOIN table$_3$ ON join_cond$_2$)...JOIN table$_n$ ON join_cond$_{n-1}$ - explicit JOIN

OR

table$_1$, table$_2$,...., table$_n$
WHERE join_cond$_1$ AND join_cond$_2$ AND....AND join_cond$_n$
- implicit JOIN

The JOIN (CROSS JOIN) operation is equivalent with a cartesian product between two tables (relations). Either INNER JOIN or OUTER JOIN can be performed. In the case of INNER JOIN (specified as JOIN) we retain only those tuples of the cartezian product that satisfy the join condition [4]. Usually, the join condition has the form of an arithmetic comparison, involving specific operators, such as ("=", "<", ">", "≤", "≥", "< >", "!="). The OUTER JOIN operation will be discussed in *Laboratory no. 3*.

- **Aggregate functions**

The aggregate functions perform a specific computation upon a set of values and return a single value as the result. These functions are usually employed together with the GROUP BY clause, in order to compute aggregate values for each group of tuples. The aggregate functions *AGG_function(field$_i$)* can be specified only within the SELECT list. in the principal SELECT clause, or in a SELECT clause nested within the HAVING clause. The main aggregate functions available in SQL are [2]:

- AVG([ALL | DISTINCT] expression) - computes the arithmetic mean of the values of a group; expression should be a *numeric expression*
- COUNT({[ALL | DISTINCT] expression | *}) - computes the number of the elements in the group
- MAX([ALL | DISTINCT] expression) - computes the maximum value within a group

- MIN([ALL | DISTINCT] expression) - computes the minimum value within a group
- SUM([ALL | DISTINCT] expression) - computes the sum of all the values of a group
- STD([ALL | DISTINCT] expression) - computes the standard deviation of a group

In the specification above, *expression* can have any type, except *uniqueidentifier*, *text*, *image* or *ntext*. *Expression* cannot contain aggregate functions or nested SELECT phrases.

- **SELECT within SELECT**

The *nested queries (SELECT within SELECT)* are those queries that contain a secondary SELECT phrase within the main SELECT phrase. The second SELECT phrase (subquery) can be introduced using set operators, such as IN/NOT IN, EXCEPT, EXTISTS/NOT EXISTS, UNION or INTERSECT. The *IN/NOT IN operator* allows us to test whether a value is in a given set of elements/ or does not belong to that set of elements. EXISTS/ NOT EXISTS is another operator, such as IN, which tests if a set is nonempty/empty. EXCEPT stands for the set difference operator [1].

The subquery can also be connected to the main query by using the construction (op ANY) or (op ALL), where op is one of the arithmetic comparison operators, belonging to {<,<=,=,<>,>=,>}. In both cases, *op ANY* and *op ALL*, the comparison is taken out repeatedly with each value returned by the subquery. In the case when the ANY quantifier is used, at least one comparison should return *true*, so that the final result to be *true*. In the case when the ALL quantifier is used, all the comparisons should return *true*, so that the final result to be *true* (if at least one comparison returns *false*, then the final result will be *false*). SOME is also available, but it is just a synonym for ANY [2].

II. Assignments:

Continue working within the interactive web-site: http://www.sqlzoo.net, by considering the following sections**:**

4. Select within Select
5. SUM and COUNT
6. The JOIN operation

Read all the indications, study the model queries and find the appropriate solutions for the unsolved queries. Solve the additional two queries:

a. Using the *world* table, show the region that contains the country which has the minimum population from all the existing countries.
b. Using the *world* table, for each region, show the country that has the maximum population

Laboratory no. 3

I. Brief theory reminder:

NULL values in SQL indicate the lack of the value for a certain field or expression (the value is unknown or inaplicabile). There is the same NULL for all the data types.

In order to check if a field or expression has NULL value, the *IS NULL* operator can be used. For example, given the table Teacher(id, dept, name, phone, mobile) if we want to return the id of those tuples for which the mobile has a NULL value, we use the following phrase:

SELECT name
FROM Teacher
Where mobile *IS NULL*

NULL values are different from 0 and from the empty string.

In order to avoid the existance of the NULL values within the result sets, Transact SQL employs the ISNULL function, having the following form [2]:

ISNULL(original_expression, replacing_expression)

If *original_expression* in not NULL, the function returns the value of the expression. Otherwise, if *original_expression* is NULL, the value of the *replacing_expression* is returned.

The MySQL equivalent of the ISNULL function is COALESCE (original_expression, replacing_expression). In MySQ, the COALESCE function can also take three arguments:

$$coalesce(x, y, z) = \begin{cases} x \text{ if } x \text{ is not NULL,} \\ y \text{ if } x \text{ is NULL and } y \text{ is not NULL,} \\ z \text{ if } x \text{ and } y \text{ are NULL but } z \text{ is not NULL,} \\ \text{NULL if } x \text{ and } y \text{ and } z \text{ are all NULL} \end{cases}$$

When NULL values occur in the database, we can use OUTER JOIN, having the following forms:

- Left (Outer) Join – A case of JOIN operation, for which all the not null values in the left hand part relation are considered. From the right hand

part relation, we take only those values satisfying the JOIN condition, otherwise the NULL value is provided instead.

- *Right (Outer) Join* - A case of JOIN operation, for which all the not null values in the right hand part relation are considered. From the left hand part relation, we take only those values satisfying the JOIN condition, otherwise the NULL value is provided instead.
- *Full (Outer) Join* - A case of JOIN operation, for which all the not null values in both the left hand part and right hand part relations are considered. For those values that do not have a meci, the NULL value is provided in the other part.

II. Assignments:

Continue working within the interactive web-site: http://www.sqlzoo.net.

Consider the following sections:

7. More JOIN operations
8. Using NULL
9. Self Join

Read all the indications, study the model queries and find the appropriate solutions for the unsolved queries.

III. Additional reading:

- the JOIN operator in SQL : http://msdn.microsoft.com/en-us/library/ms191517.aspx
- using JOINS (Inner Join, Outer Join, Cross Join, Self Join, joining three or more tables): http://msdn.microsoft.com/en-us/library/ms191472.aspx
- NULL values and Joins: http://msdn.microsoft.com/en-us/library/ms190409.aspx
- Create and Drop, Insert and Delete, Date and Time, Functions, http://www.sqlzoo.net

Laboratory no. 4

1. The relational data model

The relational data model *stands at the basics of the majority of commercial DBMS*, that exist and appear nowadays. An important **advantage of the relational data models and of the relational DBMS** is that they possess *high level data manipulation languages **(DML)*** simple, but very powerful, called **Relational Languages (RL)**.

The RL features are the following:

- Ability to allow the definition of new relations based on the existing ones
- They allow *the development*, within relational DBMS, of some *flexible and friendly interfaces*, having the possibility to be directly explored by much larger user categories, compared with the case of the network and relational databases

The *entity-relationship (ER) data model* allows us to describe the data involved in a real-world enterprise in terms of objects and their relationships and is widely used to develop an initial database design. The ER model is important primarily for its role in database design, as it provides useful concepts that allow us to move from an informal description of what users want from their database to a more detailed, and precise description that can be implemented in a DBMS. Within the larger context of the overall design process, the ER model is used in a phase called *conceptual database design* [3].

The conceptual database design is translated into the logical database design, which consists of tables and connections between tables. The table corresponds to the mathematical notion of relationship. The connections between the tables can be of type one-to-many (1-n), for example *between students and groups*, respectively many-to-many (m-n), for example *between students and disciplines*. The one-to-many relationships are realized directly, through the propagation of the primary key of the parent table in the form of a foreign key in the child table, while the many-to-many relationships need an intermediate table in order to be achieved [1].

Thus, in the case of the many-to-many relationship defined between students an disciplines, an intermediate table, students_disciplines (student_id, discipline_id) should be created.

10. Database creation in Microsoft SQL Server, Management Studio

Microsoft SQL Server, Management Studio, provides a friendly interface, in order to build the database at logical level.

In order to build a database, perform right click in Object Explorer, type the name of the database, select the owner „sa”, as shown in Fig. 2, then click ok.

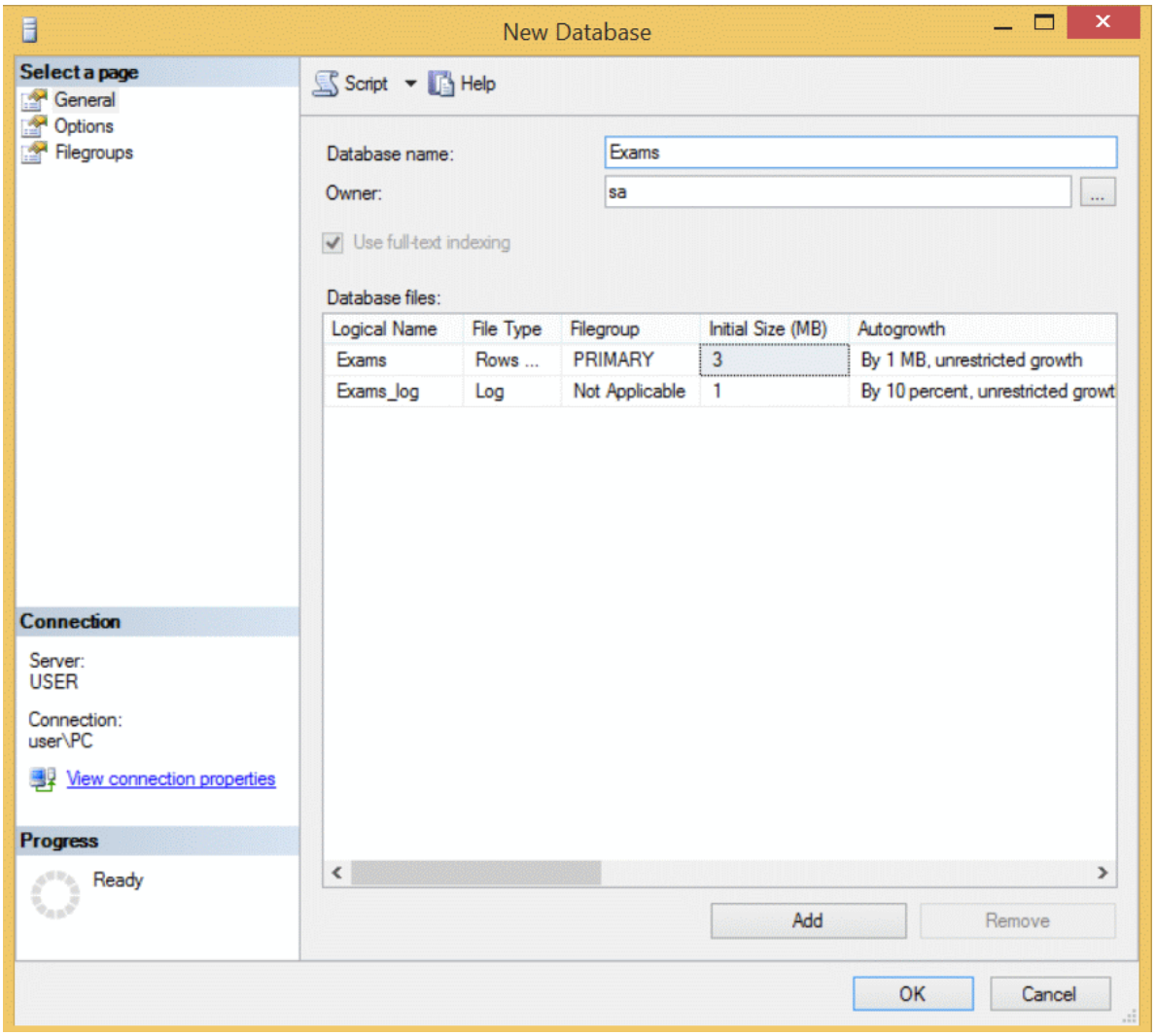

Figure 2. Database creation in Microsoft SQL Server

In order to build the database tables, go to the Tables section, within the current database, perform right click and choose New Table. Then, in the Table Design mode, define the table columns with the associated data types and with their properties, as indicated in Fig. 3.

The primary key can be set by selecting the desired field, then by pressing the primary key button within above the toolbar. You can also set the primary key by performing right click on the desired fileld and selecting the corresponding option from the context menu.

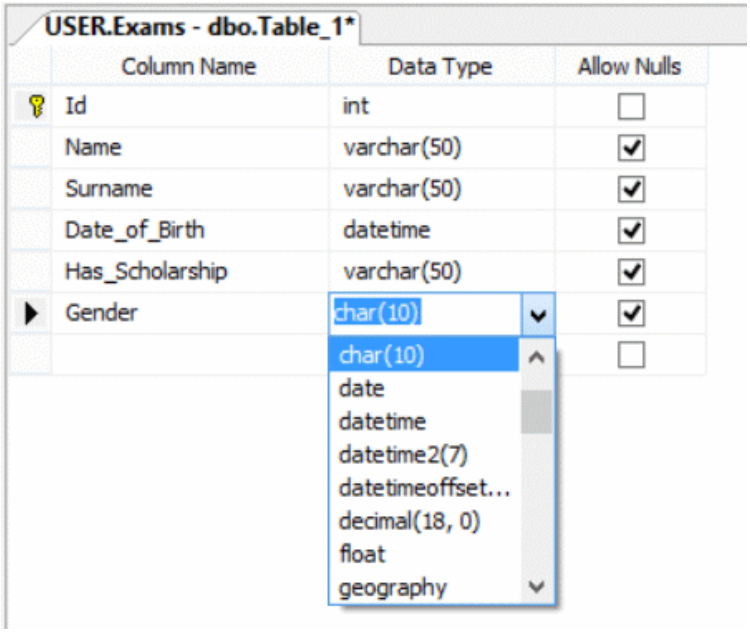

Figure 3. The creation of a new Table in Microsoft SQL Server, Management Studio

You can set the *Identity* property on Yes, meaning that for the corresponding column, successive values of a certian type (e.g. *int*) will be automatically generated. This should be done as shown in Fig. 4, within the „Column Properties" window, displayed below the table structure, in *Table Design* mode.

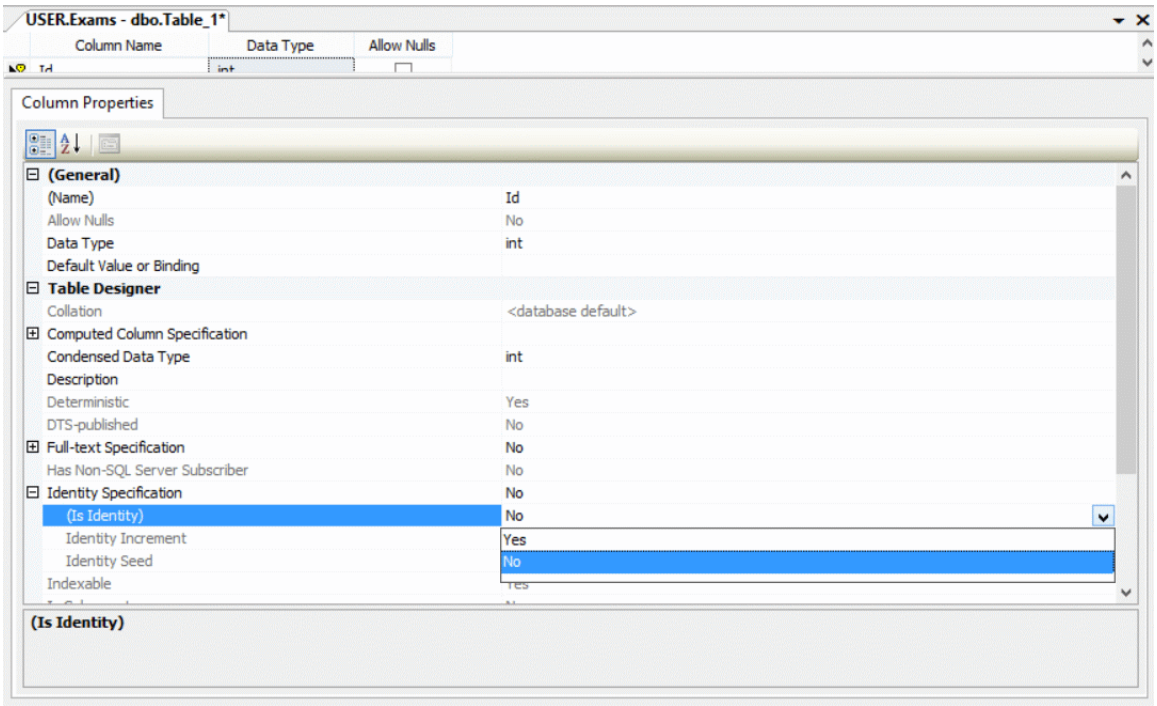

Figure 4. Setting the Identity property

After defining the columns and setting their properties, press the Save button and choose the table name (e.g. Students), as shown in Figure 5.

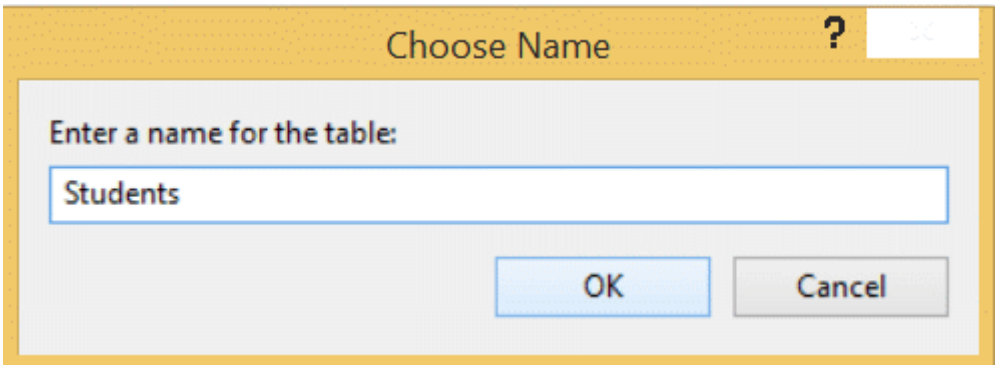

Figure 5. Saving the table under a certain name, in Microsoft SQL Server, Management Studio

User defined domain constraints (Check constraints) can also be imposed, from the *Constraints* section, attached to each table. By performing *right click* on *Constraints* and selecting *New Constraint* from the context menu, a new check constraint will be created. The user only has to write the condition, by editing the expression from the General section. The corresponding options: *Check existing data on creation (checks if the existing table data violates the constraint), Enforce for Inserts and Updates (verifies the constraint whenever a row of this table is inserted or updated), Enforce for Replication (verifies the constraint whenever a replication operation takes place within this table),* can be set within the Table Designer section. The corresponding screen is depicted in Fig.6.

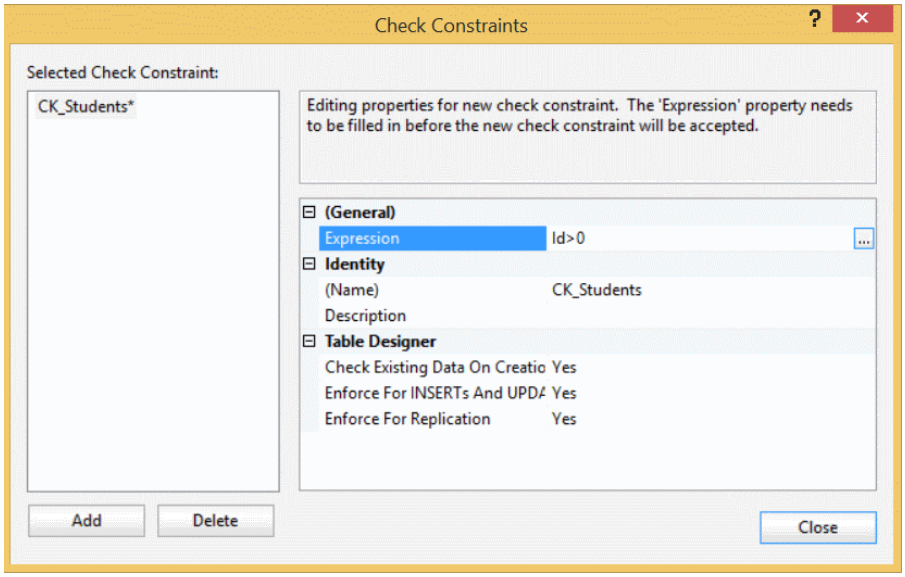

Figure 6. Creating user defined domain constraints using the facilities of Microsoft SQL Server, Management Studio

You can build the database diagram, by right click on Diagrams -> New diagram. A wizard will be launch that allows the user to add the desired tables to the diagram. Within the diagram, the relations can be created by drag and drop, from the primary key to the foreign key.

The referential integrity constraints can be imposed visually in the following manner: within the diagram, select the desired connection between two tables; in the right hand part, you can visualize the Properties Window, as shown in Figure 7 (you can make it visible from View/Properties Window); in the Properties Window, within the section *INSERT And UPDATE Specification*, set the following values for the *Delete Rule* and/or for the *Update Rule: No Action* (prohibits the operation in the primary key table, when there are records in the foreign key table); *Cascade* (propagates the operation within the related table); *Set Null* (sets the Null value for the field corresponding to the foreign key in the related table); *Set Default* (sets a default value for the field corresponding to the foreign key in the related table).

You can edit (insert or update) the data within a table, manually, by right click, and then „Edit top 200 rows". You can visualize the data inserted in a table, by right click, then „Select top 1000 rows".

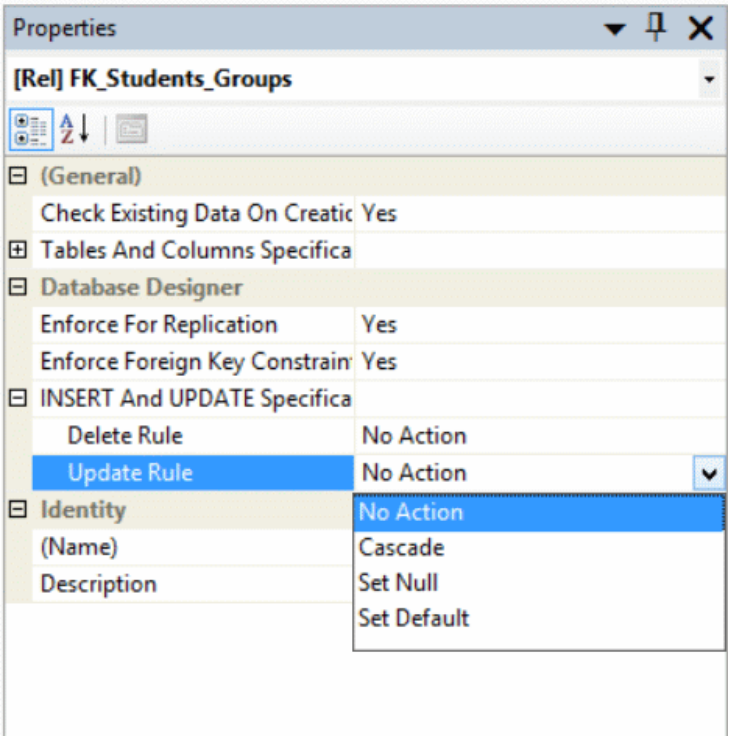

Figure 7. Setting the referential integrity rules in Microsoft SQL Server, Management Studio

After completing the database creation, you can save your work by performing a backup of your database on the disk. For this purpose, perform right click on the database name and from the context menu choose „Tasks/Back Up".

11. Database creation by using SQL statements

In order to create new tables, to modify the table defintion or to remove tables from the database, you can use the SQL statements CREATE/ALTER/DROP TABLE.

- The CREATE TABLE statement has the following syntax [2]:

CREATE TABLE *table_name*
(Col_1_name $type_1$, [DEFAULT $expr_1$] [IDENTITY(*seed*, *increment*)] [$Constraint_1$],
Col_2_name $type_2$, [DEFAULT $expr_2$] [IDENTITY(*seed*, *increment*)] [$Constraint_2$],
.....,
Col_n_name $type_n$, [DEFAULT $expr_n$] [IDENTITY(*seed*, *increment*)] [$Constraint_n$]),
Table_Constraint

$Constraint_i$ is a column constraint that can take one of the following forms:

- **NOT NULL** - Ensures that a column cannot have a NULL value
- **UNIQUE** - Ensures that all values in a column are different
- **PRIMARY KEY** – Ensures that the current column will be the primary key of the table
- **FOREIGN KEY REFERENCES *related_table_name (pk_col_name)***
  ON DELETE CASCADE | NO ACTION
  ON UPDATE CASCADE | NO ACTION

  - Ensures that the current column will be a foreign key, referencing the primary key in the related table of the table
- **CHECK** *condition* - Ensures that all values in a column satisfy a specific condition
- **INDEX** - Used to create and retrieve data from the database very quickly

Table_constraint refers to constraints specified at table level and can have the following forms:

- **PRIMARY KEY (pk_col_1, pk_col_2, ..., pk_col_n)**
- **FOREIGN KEY(*col_name*) REFERENCES *related_table_name (pk_col_name)***
  ON DELETE CASCADE | NO ACTION
  ON UPDATE CASCADE | NO ACTION
- **CHECK** *condition*

This statement can also have the following form:

CREATE TABLE *new*_table_name **AS**
SELECT col_name_1, col_name_2,..., col_name_n
FROM *existing_table_name*
WHERE *condition*;

The above statement creates *a new table* having the same structure as *the existing table* and also copies the data from the existing table into the new table.

- The ALTER TABLE statement has the following syntax:

ALTER TABLE *table_name*
ADD *column$_1$_name type$_1$, column$_2$_name type$_2$,…, colum$_n$_name type$_n$* |
ADD CONSTRAINT *constraint_name constraint_definition* |
ALTER *column_name new_type*

In order to insert new data within the database tables, to delete or modify the data in these tables, the Transact SQL statements INSERT, DELETE, UPDATE can be employed.

- The DROP TABLE statement has the following syntax:

DROP TABLE *table_name*

Above, table_name reprsents the name of the table that will be removed from the database.

In order to add (insert) new data to a certain table, to modify (update) these data, respectively to delete (remove) these data from the table, the following statements exist:

- The INSERT statement has the following syntax:

INSERT [INTO] *table_name* (*col_name$_1$, col_name$_2$,...,col_name$_n$*) VALUES *(val$_1$, val$_2$,..., val$_n$*)
The items *col_name$_1$, col_name$_2$, ..., col_name$_n$* represent the names of the columns where the data will be inserted, while *val$_1$, val$_2$,..., val$_n$* are the values that will be inserted in the corresponding columns. The data types of *val$_1$, val$_2$,..., val$_n$* should correspond to the data types of the columns.

- The general form of the UPDATE statement is provided below:

UPDATE *table_name*
SET *col_name$_1$=expression$_1$, col_name$_2$=expression$_2$, col_name$_n$=expression$_n$*
[FROM *table_name$_1$', table_name$_2$',..., table_name$_n$'*]
[WHERE *conditions*]

The items *expression$_1$, expression$_2$,..., expression$_n$* can be constant values, calculated expressions, or even sub-queries. The items *table_name$_1$', table_name$_2$', ..., table_name$_n$'* represent additional relations(tables) that can be used in order to achieve some specific conditions.

- The DELETE statement has the following form:

DELETE from *table_name*
[FROM *table_name$_1$', table_name$_2$',..., table_name$_n$'*]
[WHERE *conditions*]

The list *table_name$_1$', table_name$_2$', ..., table_name$_n$'* has the same meaning as in the previous case.

II. Assignments:

1.) In Microsoft SQL Server, Management Studio, create the "Exams" database having the following structure (*tables in the database*):

Students – fields: Stud_Id (int), Name (varchar), Surname (varchar), Gender(char), Date_of_Birth (datetime), Group_Id(int),
Groups – fields: Gr_Id (int), Gr_name (varchar), Year_of_Study (int), Comments (varchar)
Teachers – fields: Teacher_Id (int), Name (varchar), Surname (varchar), Date_of_Birth (datetime), degree (varchar)
Disciplines – fields: Discipline_Id (int), Disc_name (varchar), Discipline_Type (varchar – hardware or software)
Marks – fields: Student_Id (int), Teacher_Id (int), Disc_Id (int), Mark (int), Date_of_Exam (datetime)

Choose the **primary keys** of the tables (Stud_Id, Gr_Id, Teacher_Id, Discipline_Id). Modify (alter) the structure(design) of the Students table by adding a new field, "Has_scholarship", that can take the values "yes" or "no". Build the SQL Server Diagram of this database. Establish **one-to-many and many-to-many** relationships between the tables [2].

2.) Using Microsoft SQL Server, Management Studio, impose restrictions on the created database:

a. **Make the "Name" field in the Disciplines table unique** (manually or using the Alter Table or Create Unique Index statement)
b. **Check Constraints:**
    - using the Check clause within the Alter Table statement, make sure that, from now on, the group number from the Student Table will be situated between 1 and 5.
    - also, make sure that the Mark field in the Marks table is always a positive number, >0

Referential Integrity Restrictions:

c. **Every foreign key value from the child table must be contained also in the parent table** (tables Students – Groups, Marks – Students, Marks – Teachers, Marks – Disciplines)

d. **Referential Integrity (RI) Rules:** set cascade on delete, restrict on insert and update - manually as well as using Transact-SQL code within the Alter Table statement

3.) Use the appropriate Transact SQL statements in order to create or drop tables, insert, delete and update data from the database. Write the appropriate Transact SQL-statements in order to drop the table Groups, then to create it again, then to alter it in order to add a new column – Group_description, varchar(100). Create a new table Teachers/Disciplines in order to make direct correspondence between the teachers and the disciplines they teach. Insert data in all the tables from the database.

4.) Homework:

Solve the following SQL queries:

1. Select the names and surnames of the students from the 3-rd group.
2. Select the name, surname, date of birth and group for those students whose name begin with the 'A' letter

III. Additional reading:

- ***SQL Server Books Online : Create Table, Drop Table, Alter Table, Insert, Delete, Update***

http://msdn.microsoft.com/en-us/library/ms174979%28v=sql.105%29.aspx
http://msdn.microsoft.com/en-us/library/ms174335%28v=sql.105%29.aspx

***Transact SQL Data Types*:**

http://msdn.microsoft.com/en-us/library/ms187752%28v=sql.105%29.aspx

Laboratory no. 5

I. Brief theory reminder

1. The query languages:
   - specialized languages for asking questions, or ***queries***, that *involve the data in a database* [1]
   - The two theoretical instruments that stand at the basics of the commercial query languages are:
     - **The relational algebra** - queries are specified *operationally*
       - set of operators
       - each query describes a step-by-step procedure for computing the desired answer
     - **The relational calculus** - queries are specified non-procedurally, *declaratively*

2. The join operation:
   - one of the most useful operations in relational algebra
   - most commonly used way to combine information from two or more relations
   - can be defined as a cross-product followed by selections and projections (joins arise much more frequently in practice than plain cross-products)
   - important to recognize joins and implement them without materializing the underlying cross-product
   - the most general version of the join operation accepts a *join condition* ***c*** and a pair of relation instances as arguments, and returns a relation instance as the result
     - *the join condition* is identical to a *selection condition* in form:

       $$R \blacktriangleright\blacktriangleleft_c S = \sigma_c (R \times S)$$

       being defined to be a cross-product *followed by* a selection
     - *the condition c* can (and typically *does*) refer to attributes of both *R* and *S*
     - *the reference* to an attribute of a relation, say *R*, can be by position (of form *R.i*) or by name (of form *R.name*)
     - a special case of the join operation $R \blacktriangleright\blacktriangleleft S$ is when *join condition* consists solely of equalities (connected by $\wedge$) of form *R.name*1 = *S.name*2, that is, equalities between two fields in *R* and *S*
     - obviously, there is some redundancy in retaining both attributes in result*; the* join operation is refined by doing an additional projection in which *S.name*2 is dropped
   - further, a special case of the join operation $R \blacktriangleright\blacktriangleleft S$ is an equijoin in which equalities are specified on *all* fields having the same name in *R* and *S*
     - here, we can simply omit the join condition
     - the default is that join condition is a collection of equalities on all common fields
     - the result is guaranteed not to have two fields with the same name
     - if the two relations have no attributes in common, $R \blacktriangleright\blacktriangleleft S$ is simply the cross-product

3. SQL Queries

In SQL, the relational queries are solved using the SELECT phrase (statement), described at Laboratory no. 2. Recall the basic syntax of the SELECT phrase [1]:

SELECT *select_list, AGG_function(field$_i$)*
FROM *table_list*
WHERE *condition$_1$* AND *condition$_2$* AND... AND *condition$_n$*
GROUP BY field$_1$, field$_2$,..., field$_n$
HAVING *group_condition*
ORDER BY *field$_1$, field$_2$,..., field$_n$* [ASC|DESC]

II. Assignments:

Solve the following SQL queries in Microsoft SQL Server, Management Studio, using the Exams database and the SELECT statement:

(Inner) Join operator

a. Name, surname and group for students who sustained the exam with the teacher named 'Popovici'.

Select In Select

b. Find those students whose marks at Databases are greater than Popescu Ion's Mark at the same discipline.

Order By, Top

c. Find the first three students (name, surname and group) with the highest marks at Computer Programming.

Group By, Aggregate Functions

d. Name, surname and average mark (for all the exams sustained) for each student from group 3021 (use Group By, AVG function).

e. Name, description and number of students for each group (Group By clause, COUNT function).

Outer Join, Aggregate functions:

f. Students' name, surname and number of exams sustained. For students that didn't sustain any exam, the *0* value should appear.

In/Not In Operator; **Select within Select**

g. Find the teachers that teach both Databases and Computer Programming (***clue*** - those teachers that appear both in the following result sets: teachers that teach Databases and teachers that teach Computer Programming)

h. Name, surname and group for students who didn't sustain the exam at Databases.(***clue:*** those students do not appear in the Exams table; the Student_id from the Students table doesn't appear in the Exams table)

Homework:

Solve the same query in two ways:

- Using Exists/ Not exists function
- Using the Count aggregate function and the Group By clause

1.) Find the students who didn't sustain the exam with the teacher named 'Ionescu Vasile'
2.) Find the pairs student/teacher that didn't sustain any exam together.

III. Additional reading

- **The Transact-SQL SELECT phrase:**

  https://docs.microsoft.com/en-us/sql/t-sql/queries/select-transact-sql

Laboratory no. 6

I. Brief theory reminder:

- **SQL Views**

A view is a virtual table for which the data does not exist physically (is not stored) within the database, unless the view is indexed. Only the definition of that view, in the form of an SQL query, is stored in association with the view. However, after creation, the view can be treated like an usual table [1].

In order to create a new view in the database, the CREATE VIEW statement exists in SQL, having the following syntax:

CREATE VIEW *View_name* AS
SELECT_statement

In order to modify the view definition, the ALTER VIEW statement can be employed:

ALTER VIEW *View_name* AS
modified_SELECT_statement

In order to remove the view definition from the database, the DROP VIEW statement can be used:

DROP VIEW *View_name*

The INSERT, UPDATE and DELETE statements *can be also applied upon the views* in some conditions, in order to modify or delete the data in the underlying tables. They have the same syntax as the INSERT, UPDATE and DELETE statements, performed on tables, presented within *Laboratory no. 4*.

In the general case, a view is considered updatable if the reference to the columns (fields) that should be modified is clearly specified and no confusion occurs. Thus, in SQL server, a view is updatable in the following conditions [2]:

- The view refers at least one basic table, not being exclusively based on one or more expressions
- The SELECT list of the definition query does not contain any aggregate function or one of the options: DISTINCT or TOP. The nested queries within the FROM clause can contain aggregate functions, but the corresponding derived values should not be the subject of modification.
- The SELECT phrase does not contain the GROUP BY or UNION clauses.
- If the operation is INSERT, then the view should not contain columns derived (obtained) from an expression, function, etc. within the SELECT list.

- If the operation is DELETE, then the SELECT phrase which defines the view can refer a single basic table in the FROM clause.
- The name of a column should not appear multiple times within the SELECT list of the query that defines the view.

In Microsoft SQL Server - Management Studio, *the views* can be also created by using the *View Designer*, a friendly environment that allows the definition of the views in visual manner. In order to access the view designer for view creation, right click on the *views* section within your database, then, from the context menu, select *new view*. In order to modify the view definition with *View Designer*, select the *Design* option from the context menu.

- ***Control structures* and *batch files* in Transact SQL:**

Transact SQL contains the following control structures [2]:

Statement	Description
BEGIN…END	Defines a statement block.
BREAK	Generates the exit from the current WHILE loop.
CONTINUE	Restarts the execution of a WHILE loop
GOTO label	Continues the execution with the statement following after the label specified after GOTO.
IF…ELSE	Conditional execution and an alternative, for the case when the test condition takes the value FALSE.
RETURN	Unconditioned exit from the current procedure.
WAITFOR	Establishes a delay for the execution of a statement.
WHILE	Repeats a statement sequence as long as the specified condition is true.

In Transact SQL, a batch file represents a group of one or more statements, sent in the same time from an application to SQL Server, in order to be executed. SQL Server compiles the statements of a batch into a single executable unit, called an execution plan. The statements are then executed at once. Thus, a batch file constitutes an independent compilation unit, having its own context, which is not influenced and does not influence the context of other batch files. The variables declared inside a BATCH are visible only within this batch. If syntax errors occur, which hinder the correct compilation of the batch and the building of the execution plan, then none of the batch statements will be executed. The batch file ends with the keyword GO.

The following rules work upon a batch file:

- A batch file can contain only one of the following statements: CREATE DEFAULT, CREATE PROCEDURE, CREATE RULE, CREATE TRIGGER or CREATE VIEW. None of these statements can be combined with another similar statement within the same batch.
- If a table is modified within a batch, then the new columns of this batch cannot be referred within the same batch.
- If an EXECUTE statement is the first within a batch, then the keyword EXECUTE can be omitted.

II. Assignments:

1. **Create the following views in your database (*Create View* Transact-SQL statement):**
   - Student's name, surname and group, discipline name, date of exam, mark obtained at that exam.
   - Teacher's name and surname, discipline name that is taught by the teacher. For the teachers that don't teach any discipline, the 'no discipline' string will be displayed.
   - Names and surnames for the teachers that didn't sustain any exam with the students from the group 3021.
   - Names and surnames for the students that sustained the exam at Databases, but didn't sustain the exam at Computer Programming.
   - Names and surnames for teachers that teach Mathematics (the discipline name must contain the word 'Mathematics').
2. **Modify (alter) the first view in order to display only the marks obtained by the students at the exam at databases.**
3. **Update the last created view, as a mistake has been made in the corresponding data: the teacher called Popescu Maria, who teaches mathematics, is actually named Pop Maria.**
4. **Read in SQL Server Books online about Transact SQL statements (BEGIN…END, IF…ELSE, CASE, WHILE, WAITFOR, GOTO, GO) and about batch files. Create a batch file in order to display the names and surnames of the students that sustain the exam at 'Computer Programming'. If nobody sustained this exam, a corresponding message should be displayed. Use the IF…ELSE Transact SQL Statement.**

III. Additional reading

http://msdn.microsoft.com/en-us/library/ms190706%28v=sql.105%29.aspx
http://msdn.microsoft.com/en-us/library/ms187956(SQL.90).aspx
http://msdn.microsoft.com/en-us/library/ms187956%28v=sql.105%29.aspx
http://msdn.microsoft.com/en-us/library/ms180800%28v=sql.105%29.aspx
http://msdn.microsoft.com/en-us/library/ms189826(SQL.90).aspx
http://msdn.microsoft.com/en-us/library/ms175502%28v=sql.105%29.aspx

Laboratory no. 7

HTML, PHP Basics

I. Brief theory reminder:

1. Client-Server Systems

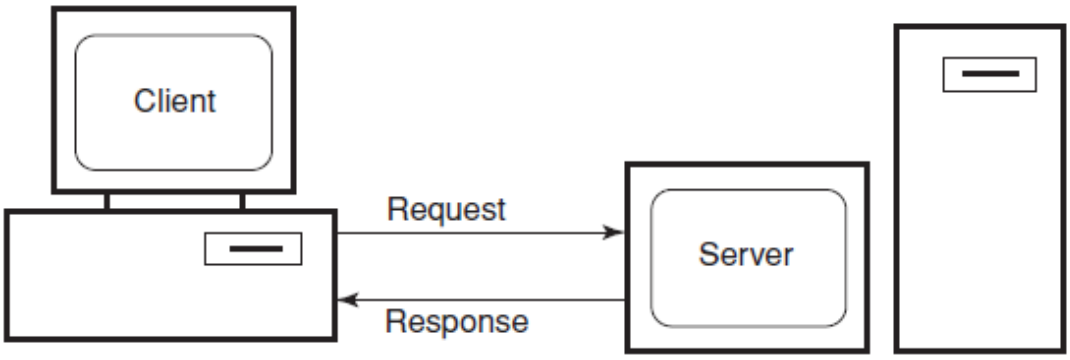

Figure 8. The client-server architecture

The client-server architecture (Figure 8) corresponds to a network communication model frequently employed, involving the participation of two entities: the client and the server. Usually, there are multiple clients in the network which communicate with the server. Sometimes, the same client might be connected to multiple servers.
Each client can address requests to the server, for some services, such as applications access, storage, file sharing, and/or direct access to the server's computing power; the server processes the request, then replies to the client, providing him the required service, or a corresponding message when that service is not available. In its simplest form, the Internet is also based on the client/server model (architecture): the Web server serves simultaneously many users, with web pages and/or website data [1].

2. The structure of the Database Web Applications

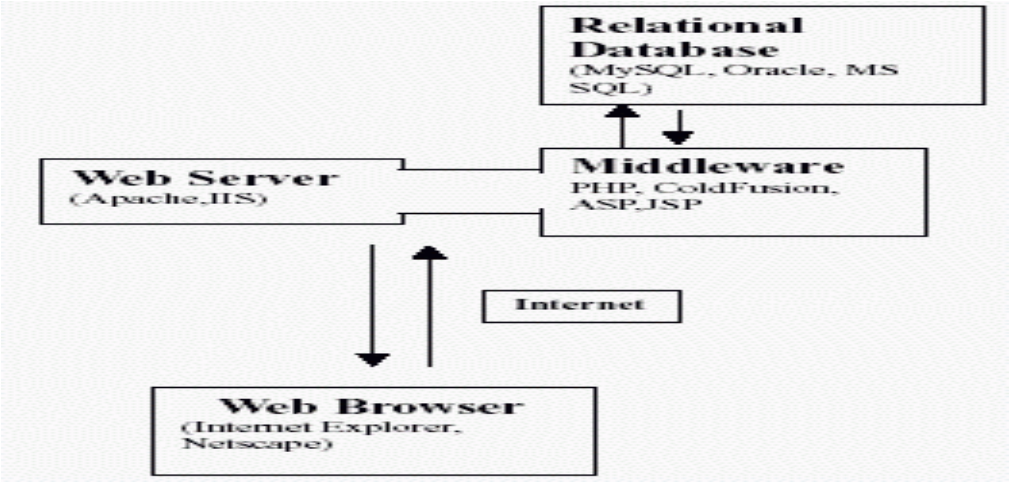

Figure 9. The architecture of the web applications that communicate with the databases

The architecture of the web applications that communicate with databases is depicted in Figure 9. This is a client-server architecture, in which the client (represented by the client's web browser) communicates with the web server. There are two servers which are part of the architecture of the database web applications: *the relational database server* and of *the web server*. The web server communicates with the relational database server through the middleware, which enables the web server to send specific requests (e.g. SQL queries) to the database server. The middleware is generated by scripts written in a specific *scripting language* (e.g. PHP, ColdFusion, ASP, JSP). The web server communicates through the Internet with the web browsers installed at the clients. Concerning the database servers, a MySQL server is a good choice, due to the fact that it provides high speed and increased flexibility. Other usually employed database servers are Microsoft SQL Server, Oracle, IBM DB2, PostgreSQL. Regarding the practical work, some specific packages, like WAMP (Windows – Apache, MySQL and PHP), LAMP (Linux – Apache, MySQL and PHP) and MoWeS Portable (Portable Modular Webserver System) exist, integrating Apache, MySQL and a PHP interpreter and running under both Windows and Linux.

In the early days of the Internet, the websites were just static things that required each page to be created manually. For example, in the case of a web application that managed a sales catalogue on the web, the web application programmer had to create manually a separate page for every item in the catalogue. Nowadays, after the development of the web application databases, the web application programmer only has to create a standard template page for the catalog item (product) and this page is modified during the runtime, "on the fly" , being populated with the appropriate data from the database.

3. The HTML language

HTML (HyperText Markup Language) is nowadays the most employed language for the creation of the webpages. Some extensions of HTML appeared during the last years,

such as HAML (HTML Abstraction Markup Language) and XHTML (Extensible Hypertext Markup Language), which involves XML.
The structure of an HTML page is provided in Figure 10. The page content is described within the beginning html tag, "<html>" and the ending html tag, "</html>". A head section is usually present, having a title sub-section, specifying the title which appears in the browser upper bar, and eventually other tags (e.g. <script>, <meta>). The main part of the html page is that specified between the <body>, </body> tags. These tags, together with the <html>, </html> tags are compulsory for every web page [5].

```
<html>
<head><title>Page title</title></head>
<body>
  ……
</body>
</html>
```

Figure 10. The basic structure of an HTML page

Thus, HTML is a web page description language based on tags. We usually have the beginning tag, <tag>, respectively the closing tag, </tag>.

3.1. The HTML tags:

The basic html tags are described below:

** tag**
- display the text in a standard font
- tag attributes
- text

** tag**
- stands for Paragraph; used to break up text into paragraphs
- tag has attribute which can be added to it - align option
- Text
- can use the tag or the
- hardly ever necessary to use the align="left" attribute as nearly all browsers automatically align text to the left

**
 tag**
- want to leave a space after your paragraphs
- should use the
 (break) tag
- there is no end tag
-
 =
 =
 =
- Tags are case insensitive

<img> tag

- Images are added to pages
- must use the *src* attribute to choose the image to insert
- can either be a relative reference or a direct reference
- can resize images inside the browser using two other image attributes: width and height
- Alt tells the browser what the alternative text for an image should be if the browser has images turned off

<a> tag

- used when creating hyperlinks and bookmarks; stands for anchor
- need to use the href attribute
- to make a piece of text or an image into a hyperlink you contain it in:
- Link

3.2. HTML forms

The HTML forms contain specific elements (controls) which are used in order to collect the user input. They are described between the beginning tag, <form>, respectively the ending tag, </form>. An example of a *form*, which collects input data about students, is provided below:

```
<form method=POST action=”MyFile.php”>

Student name: <input type=”text” name=”st_name”></input><br>
Student surname: <input type=”text” name=”st_surname”></input><br>
Gender :<input type=”radio” name=”gender” value=”M” checked>M</input>
        <input type=”radio” name=”gender” value=”F”>F</input><br>
Specialization: <select name=”specialization”>
<option name=”automation” value=”Automation”>Automation</option>
<option name=”cs” value=”Computer Science”>Computer Science</option>
</select><br>
<input type=”submit” name=”OK” value=”OK”></input>

</form>
```

The form described above contains, firstly, the form definition tag, in which the method (“POST”), respectively the action to be performed (“MyFile.php”) are specified. Then, two input controls of type *text* are defined, which enable the user to provide the name and the surname of the student. A radio button is also defined, which enables the specification of the student’s gender. For this purpose, two input elements, having the same name, “gender”, but different values, “M”, respectively “F” are declared. In order to allow the user to specify his

specialization, an input control of type “select”, having two options is inserted, as well, within the form.
At the end, there is a submit button, which enables the transmission of the parameters from the current page to the php page, “myfile.php”, initially mentioned.
There are also other input elements (such as the *checkboxes*), which can be defined within the html forms, as shown in the appendix. More sophisticated html interfaces can be created using the jQuery and jQuery mobile *javascript* libraries.

4. The PHP language

PHP (Hypertext Pre-Processor) represents a scripting language that can be embedded in a HTML page. It is a general-purpose scripting language, able to perform data processing and communication with various databases [6].
While embedded into HTML, the PHP script is delimited by the *beginning tag*, “<?php”, respectively by the ending tag, “?php>”, or “?>”.

4.1. Your first, “Hello World”, PHP page

Your first HTML-PHP page should look like:

```
<html>
    <head>
        <title>My First HTML-PHP page</title>
    </head>
    <body>

        <?php
            echo "Hello World!";
        ?>

    </body>
</html>
```

The above file must be saved under the name “MyFirstFile.php” using the “.php” extension. In order to execute the PHP files within the browser, you can use the MoWeS Portable environment, which contains an Apache server, a MySQL database server and a PHP interpreter. Launch the Apache and MySQL Servers by running the “mowes.exe” file from the “portable” folder. Place your PHP file into the “www” sub-folder of the “portable”. In the “www” folder, create your own sub-folder “myfolder” and copy “MyFirstFile.php” there. In order to visualize the file in browser, go, for example, in Mozilla Firefox and in the address bar type:

```
http://localhost/myfolder/MyFirstFile.php
```

4.2. Parameter transmission from HTML to PHP

The parameter transmission from HTML to PHP can be performed using one of the methods GET or POST. The method can be specified within the form definition. The GET method can also be specified within hyperlinks, as for example:

```
<a href="mypage.php?par1=val1&par2=val2">My page</a>
```

In the above example, the parameters *par1* and *par2* are transmitted to "mypage.php".

Assume that we transmit the parameters through the POST method, through the html form described at 3.2. In this case, MyFile.php should look like:

```
<html>
    <head>
        <title>HTML-PHP parameter transmission</title>
    </head>
    <body>

        <?php

        $name=$_POST["st_name"];
        $surname=$_POST["st_surname"];
        $gender=$_POST["gender"];
        $specialization=$_POST["specialization"];

        echo("Student name:".$name."<br>");
        echo("Student surname:".$st_surname."<br>");
        echo("Gender:".$gender."<br>");
        echo("Specialization:".$specialization."<br>");

        ?>

    </body>
</html>
```

The values of the parameters are obtained using the predefined $_POST associative array, automatically created by the system when performing parameter transmission from HTML to PHP. The indexes of this array are the names of the html input elements, specified within the HTML form. Then, the "echo" statement generates the display of the values for the newly created variables. The "." symbol stands for the string concatenation operator. The "
" tag generates a line break event (the movement of the cursor on a new line).

II. Assignments:

1. Read in HTML tutorial about the HTML page structure and about the basic HTML tags.
2. **HTML images:** Build a HTML page with the title "Technical University of Cluj-Napoca" and insert the Technical University's symbol, using the "logo.gif" file in the current folder. Use the **<img src>** HTML tag. Display the image in the upper side of the page, bellow the title, in the center.
3. **HTML Hyperlinks:** Insert, in the same page, bellow the image, a Hyperlink to the Computer Science Department of the Technical University of Cluj site; URL: http://www.cs.utcluj.ro.Use the **<a href>** HTML tag. Change the hyperlink colors: active link should be red, visited link should be green and the usual color of the link should be orange.
4. **HTML forms and parameter transmission; Insert, in the HTML page, forms in order to input data about students:**
    a. Insert text-boxes for student's names, student's surnames, student's place of birth and student's address.
    b. Use select lists – one for student's gender (Male or Female) and another group for student's date of birth (day, month, year).
    c. Use radio-buttons, in order to specify whether the student is in years 1-3, or in years 4-5.
    d. Use a check box, in order to specify whether the student takes scholarship or not.
    e. Use a "Submit" button in order to submit the form data and a "Clear" button in order to clear the form at user's request.

Parameter transmission: use the clauses ACTION= "myfile.php" and METHOD = POST inside of the <form> tag, just like in the example from the tutorial. See the example above and the PHP Manual – Dealing with Forms.

5. Arrange the data from the HTML page, using an HTML table. Put each interface element (e.g. text-box) and its label in a distinct row. Put the label in the left column and the interface element in the right column. Make the borders of your table invisible. Make the first column of your table of width 30%. Vertically align the text in your table in center and horizontally – on the left side. Use the documentation in the appendix and the HTML manual.

6. Create a .php file named "MyFile.php". Dysplay the world "Hello!" using the php statement " echo". Format the world using the HTML tags: make it ***bold***.

Put the html and php files in a folder called “work”. Move this folder in the “www” subfolder of *mowes portable*. Run the .php file in browser, using the following address:

http://localhost/work/MyFile.php

7. **Data processing and display with PHP:**
   Getting the parameters: In “MyFile.php”, use the $_POST[name_from_input_tag] expression, in order to get the data from the previous HTML page. Assign the values to PHP variables. Display the data on the screen in the following format:
   Name: Popescu
   Surname: Ion
   Gender: Male
   Date of Birth: Day, Month, Year
   Years of study: 1-3, or 4-5
   Takes scholarship (or Doesn’t take scholarship).
   Age: calculate age in years, based on the Date of Birth

III. Additional reading

HTML Tutorial: http://www.2createawebsite.com/build/html.html
PHP Tutorial: http://www.php.net/manual/en/

Laboratory no. 8

HTML, PHP Basics; Connections to Databases

I. **Brief theory reminder**:

1. The PHP language

Unlike *the javscript language*, which represents client-side scripting and it is executed on the client machine, PHP is a server-side scripting language, meaning that the PHP code is executed on the server, generating HTML code that is sent back to the client. PHP is inspired from C, Java and Perl, being easy to learn. It corresponds to both the imperative and object-oriented programming paradigms, having an Object-Oriented version, as well.

The main *PHP data-types* are: *integer*, *floating point number*, *boolean*, *string*, *object*, *array*, *null* and *resource*. The numbers in PHP are usually represented as a 32-bit signed integers. *The integer variables* can be assigned using decimal (positive and negative), octal and hexadecimal notations. The *real numbers* can be specified using floating point notation, or two forms of scientific notation. There is a *native boolean type* in PHP. Also, *non-zero values* can be interpreted as *true values*, while *the zero value* is interpreted as *false*. *The null data type* represents a variable that has no value, the only value in the *null data type* being *null*. *Arrays* support both numeric and string indices, and are heterogeneous. Arrays can contain elements of any type that PHP can handle, including resources, objects, and even other arrays.

In PHP, *the variables* do not have to be declared. PHP variables are created in the moment of the assignment and the same variable can change its type during runtime. The PHP variable names are prefixed with the "$" symbol.

If, for example, we have the following PHP statement:

```
$num=101;
```

It means that the variable named "$num", of numeric type, is created.

Unlike function and class names, the variable names in PHP are case sensitive. In PHP, each statement has to be terminated by ";". There are three types of comments in PHP: "/* */" serves as a block comment, while "//" and "#" are used for inline comments.

PHP also provides *control structures*, such as: *if-else*, *elseif/else-if*, *while*, *do-while*, *for*, *foreach*, *break*, *continue*, *switch*, *declare*, *return*, *goto*.

2. Connection to databases form PHP

PHP has many predefined functions that allow the programmer to perform connections to the databases, to send SQL statements and queries to the database

server. Thus, there are specific libraries, integrated in the PHP packages, such as "php_mysql.dll", "php_mysqli.dll", "php_mssql.dll", "php_pgsql.dll, "php_sqlite.dll", "php_odbc.dll" that allow the communication with various types of Database Management Systems (DBMS): MySql, Microsoft SQL Server, PostrgreSQL, SQL Lite.

In order to perform the connection to a MySQL server and to select a MySQL database, there exist the following predefined PHP functions [6]:

```
resource mysql_connect ( [string $servername [, string $username [,
string $password [, bool $new_link]]]] )

bool mysql_select_db ( string $database_name [, resource
$link_identifier] )
```

In the newer versions of PHP (beginning with PHP 5.0.0), the *mysqli_connect* function is implemented, which performs both the connection to the database and database selection:

```
resource mysqli_connect (string $servername, string $username, string
$password, string $database_name);
```

PHP also provides the possibility to perform *persistent database connections.*
Persistent database connections represent links, which do not close when the execution the script ends. In the moment when a persistent connection is requested, PHP checks if there exists already an identical persistent connection (remained open from an earlier connection) - and if this connection exists, it uses it. If the connection does not exist, it creates it. An 'identical' connection represents a connection opened to the same host, having the same username and password. In order to perform persistent database connections, the following predefined PHP function can be employed:

```
resource mysql_pconnect ( [string $servername [, string $username [,
string $password [, bool $new_link]]]] )
```

In order to send various statements to the database server, the *mysql_query*, respectively *mysqli_query* statements exist.

Below is an example of PHP code, performing a connection to the database server and database selection, then the selection of table data and the display of these data on the screen:

```
<?php

# Connect
$conn= mysqli_connect('localhost', 'username', 'password',
‘someDatabase’) or die('Could not connect: ' . mysqli_error($conn));

# Perform database query
$query = "SELECT * from someTable";
$result = mysqli_query($conn, $query) or die('Query failed:
'.mysqli_error());

# Browse (or filter) through rows and display the desired information
while ($row = mysqli_fetch_array($result, MYSQLI_ASSOC)) {
    echo $row[‘field_name’];
}

?>
```

The first statement performs the connection to the database server and database selection. Then, a query is sent to the MySQL database using the *mysqli_query* predefined function. This function returns a result set, consisting of the records (rows) that satisfy the query. Afterwards, a *while loop* is employed: at each iteration, each record (row) of the result set is fetched into an array, $row, using the *mysqli_fetch_array* predefined function. The second argument of this function, MYSQLI_ASSOC, defines the type of the array, $row, as an associative array. Thus, each field of the current row can be referred through its name from the database ($row[‘field_name’]).

Besides the SQL SELECT statements, other types of SQL statements (e.g. *INSERT*, *UPDATE* or *DELETE*) can be addressed as well from PHP using the *mysqli_query* function. Below an example of INSERT statement performance against MySQL is provided:

```
<?php

$st_name=”John”;
$st_surname=”Brown”;

$q=”Insert into Students(Name, Surname) values(‘$st_name’, ‘$st_surname’)”;
mysqli_query ($q) or die (“Query error”);

?>
```

Equivalent functions for database server connection, database selection, query sending and result set manipulation also exist for other types of databases, besides MySQL, such as Microsoft SQL Server (prefix: “mssql”), PostgreSQL (prefix: “pgsql”), SQL Lite(prefix: “sqlite”), or generic database connections, through Open Database Connectivity – ODBC (prefix: “odbc”). However, *MySQL* is usually recommended for web application development.

MySQL is integrated in packages like WAMP, LAMP and MoWeS portable. In order to work with MySQL databases, the *PhpMyAdmin* interface can be used (for WAMP and LAMP), or the MySQLWorkbench application can be employed as well.

II. Assignments:

1. Use the string concatenation operation in PHP, in order to obtain a new string from student's name and surname, separated by a space character. Write on the screen the following phrase: The student name is …. (specify student's name and surname using the string built previously).
2. Read in "Php Manual" about Control Structures.
   In the ".php" page, use an array in order to specify the main subjects that are being studied by the students. Display these subjects on the screen using a bulleted list and a cycle (for, while, or do…while). Then, use an ordered (numbered) list in order to display three rules that you consider very important for students.
3. Rebuild the Exams database in the MySQL. Use the MySql Workbench environment of MoWeS Portable.
4. Modify the "test1_mysql.php" file from the laboratory folder in order to connect to your own database "Exams", to select data from the Students table and to display on the screen the students' names, surnames and group name, within an html table. Use the mysqli and mysqli_query statements. Read in the PHP manual about connections to databases: ordinary and persistent connections.
5. Change the Students table in order to contain the following fields: Name (varchar), Surname (varchar), Gender (Char), Date_of_Birth (Datetime), Year_of_study (varchar), Scholarship (char –'y' or 'n'). Use the form designed in the .html page that you've created during the former laboratory (Laboratory no. 7), and the mechanism of parameter transmission, in order to Insert data from the .html file into the Students table of the Exams database, using the SQL INSERT statement.
6. Use a first page, "select.php", containing two select lists, in order to choose the desired student by name and surname. Transmit the data about the selected student in a second page, "update_student.php". There, build a form in order to modify the data about the student, then update the corresponding data into the database, using the MySQL Update statement.

III. Additional reading:

Help: PHP, HTML, MySQL documentation:

http://www.2createawebsite.com/build/html.html
http://www.php.net/manual/en/
https://www.tutorialspoint.com/mysql/
http://www.mysqltutorial.org/

Laboratory no. 9

HTML, PHP Web Sites

I. Brief theory reminder:

According to its name, a *web site* represents a site (location) on the World Wide Web. *A web site* consists of a collection of web pages (e.g. of type ".html", ".php", ".asp"). The navigation among the web pages of the same web-site occurs due mainly to hyperlinks, to pressing the form submit buttons or to *redirect* actions.
We can use *html framesets* in order to create web sites. An *html frameset* can be defined in the following manner:

```
<HTML>
<HEAD>
<TITLE>A Basic Example of Frames</TITLE>
</HEAD>

<FRAMESET COLS="20%, 60%, 20%">
     <FRAME NAME="left_frame"   SRC="framea.html">
     <FRAME NAME="middle_frame" SRC="frameb.html">
     <FRAME NAME="right_frame"  SRC="framec.html">
</FRAMESET>

</HTML>
```

The example above defines a frameset containing three vertical frames: the left frame, (named "*left_frame*"), the middle frame (named "*middle_frame*") and the right frame (named "*right_frame*"). The left frame occupies 20% of the columns, the middle frame occupies 60% and the right frame occupies 20%, as well. The initial sources for each of the three frames are web pages (e.g. *html pages*) that have to be defined in separate files, named appropriately ("framea.html", "frameb.html" and "framec.html"). Frequently, the web programmers want to perform hyperlinks from one frame to the other, for example to define a hyperlink, in the left frame, that opens the desired page in the middle frame. The corresponding hyperlink should be described in the following manner, within the html code corresponding to the left frame:

```
<a href=”student_data.php” target=”middle_frame”>Student data</a>
```

As you can notice, the *target* keyword is used in order to specify the name of the frame where the file "student_data.php" should be opened. If we specify *target=parent*, then the file referred through the *href* attribute will open as an independent web page.

Suppose we have a web-site that communicate with a database, containing HTML and PHP pages. Then, the connection to the database should be performed from every PHP file that requires data from the database. For more convenience, we should create a single file, *init.php*, where the *mysqli* function is called with the appropriate arguments, as shown in *Laboratory no. 8*. The init.php file should be invoked from every PHP file that needs to perform a database connection, by using the keyword "require", as indicated below:

```
require init.php
```

The *require* statement will copy the code of init.php inside the page that invokes this statement. The PHP statement *include* has the same effect. However, there is a small difference between *require* and *include*: when using the *require* statement, if the required file is missing, the execution stops and an error is reported; if the file is missing in the case of the *include* statement, the execution continues with the next statement. There are also the *require_once* and *include_once* statements: they are identical to the require and include statements, but in the case of require_once, or include_once, PHP will verify if the file has already been included, and in that case, will not require (include) it again [6].

II. Assignments:

1. Read in PHP Documentation (manual) about the require( ) and include( ) statements. Build an extra-file, called "init.php" and write code for database connection here. In each ".php" file use the "require" statement in order to include "init.php".

2. Read the Web Site definition from:
   http://www.webopedia.com/TERM/W/web_site.htm

3. Build a Web Site corresponding to your own domain, containing multiple .html and .php pages. You will have "index.html" as the first page, consisting of an authentication form. If the login is correct, the user must access the second page, "main.php". The page "main.php" consists of a frameset (describe the frameset, using the <frameset> html tag). The frameset will contain a left frame with a menu, a middle frame in order to display the information required in the left frame and a right frame displaying the author's (your name) [5].

   The menu in the left frame will have to enable the following operations with the database: Insertion, Update, Deletion and Selection (visualization) of data for each entity (e.g.: Students, Disciplines, Teachers, Exams).

   All these operations will be done through ".php" pages displayed in the middle frame. The menu in the left frame will contain the links to these pages; in order to display these pages in the middle frame, you use the "target" keyword (e.g. target="middleFrame").

The interface should look like the “model.jpg” file, placed in the laboratory folder.

III. Additional reading:

- **HTML, PHP documentation:**
  http://www.2createawebsite.com/build/html.html
  http://www.php.net/manual/en/
- **HTML framesets:**
  http://www.w3schools.com/tags/tag_frameset.asp

Laboratory no. 10
Working with NoSQL structures (JSON, XML) in JavaScript

I. Brief Theory Reminder

- **The XML and JSON data structures**

In some situations, it is more efficient to some data on the local device, without using a database server. This is valuable in the case when we have limited resources, as for example for mobile applications. In this case, we can use NoSQL data structures, such as those defined in JavaScript Object Notation (JSON), respectively in eXtensible Markup Language (XML). The advantage in the case of the XML structures is that they are both human and machine readable, while the JSON structures are lighter (less complex) than XML, so less processing time is needed in their case [7].
The XML data structure consists of tags, describing both the entity sets, entities and the attributes, an example of an XML data structure for the *employees* entity set being provided below:

```
<employees>
    <employee>
        <firstName>John</firstName>
        <lastName>Brown</lastName>
    </employee>
    <employee>
        <firstName>Michael</firstName>
         <lastName>Smith</lastName>
    </employee>
</employees>
```

Within the first tag, the entity name, <*employees*>, is specified. Then, each entity is specified between separates tags, <employee>…</employee>. For each entity, there are separate tags with each attribute name, the corresponding values of the attributes being mentioned between these tags.
The JSON data structure for the same entity set, *employees*, is given below. It consists of nested square brackets and braces in order to specify the entities and the attributes:

```
{"employees": [
   {"firstName":"John",  "lastName":"Brown"},
   {"firstName":"Michael", "lastName":"Smith"},
   {"firstName":"Pete",  "lastName":"Jones"}
 ]}
```

In the case of the JSON structure, the name of the entity set, *employees*, is firstly mentioned, the entire entity set is specified between square brackets, each entity being specified between braces (accolades), with each attribute and attribute value mentioned between quotes (“”).
The client-side script is generated on the client’s computer. The web server retrieves the page and sends it as it is to the web browser. The web browser then processes the code embedded within the page (normally HTML and JavaScript) and displays the corresponding page.
In JavaScript, an XML structure is defined as shown below. First, a text variable containing the XML specification for the *employees* entity set is created. Then, a new DOMParser variable is created. The DOM structure models XML as a set of node objects. The nodes can be accessed with JavaScript or other programming languages. At the end, the “firstName” attribute for the first entity is displayed within the HTML page, inside the section (div) called “demo”. The text variable, containing the “<employees>” structure, can also be written to the local storage, by using the statement: *localStorage.setItem(‘employee’, text)*.

```
<html>
<body>

<p id="demo"></p>

<script>
var text, parser, xmlDoc;

text = “<employees><employee>”+
        “<firstName>John</firstName>”+
        “<lastName>Brown</lastName>”+
       “</employee><employee>”+
        “<firstName>Michael</firstName>”+
        “<lastName>Smith</lastName>”+
        “</employee></employees>”

parser = new DOMParser();
xmlDoc = parser.parseFromString(text,"text/xml");

document.getElementById("demo").innerHTML =
xmlDoc.getElementsByTagName("firstName")[0].childNodes[0]
.nodeValue;
</script>

</body>
</html>
```

```
//Storing data:
var myObj = {"employees": [
    {"firstName":"John",  "lastName":"Brown"},
    {"firstName":"Michael", "lastName":"Smith"},
    {"firstName":"Pete",  "lastName":"Jones"}
 ]} ;

var myJSON = JSON.stringify(myObj);
localStorage.setItem("testJSON", myJSON);

//Retrieving data:
var text = localStorage.getItem("testJSON");
var obj = JSON.parse(text);
document.getElementById("demo").innerHTML = obj.name;
```

First, the *myObj* variable is created, containing the JSON structure for the *employees* entity set. The values of the attributes of a certain entity can be accessed by indexing this array of objects, then by mentioning the attribute name.
For example, in order to access the value of the firstName attribute for the first entity, we write: myObj[0].firstName.
In order to store the JSON structure in the local storage, we first have to convert the myObj variable to a *string*, by using the statement *JSON.stringify(myObj)*. Then, the *localStorage.setItem* function can be called. In order to retrieve the data from the local storage, the function localStorage.getItem has to be employed, a string *text* being obtained. Then, an array *obj* is created, by using the *JSON.parse* statement.
Thus, the local storage and session storage can be performed in JavaScript due to the web storage API (Application Programming Interface), more specific to the properties *Window.localStorage* and *Window.sessionStorage* of the class *Window*. The first represents the permanent local storage, while the second represents the session(temporary) storage, that last only during a session. The format of the corresponding functions is provided below. The examples are provided for the *localStorage* property, but are identical for *sessionStorage*.

```
localStorage.setItem('key', 'value');
var data = localStorage.getItem('key');
localStorage.removeItem('key');
localStorage.clear();
```

- **Communication with the database server through AJAX calls**

As the local storage of the devices is usually limited, transfers between the local storage and the database on the server are necessary. These transfers can be performed through the JavaScript extensions AJAX and AngularJS. We will focus here on AJAX transfers.
AJAX (Asynchronous JavaScript and XML) is a set of web development techniques that can be employed on the client-side, in the context of web

application development. AJAX increases the interactivity of the web pages, their speed, functionality and usability. Due to this technology, the entire web page does not have to be reloaded each time the user requests a change. The retrieved data is usually formatted using XML and JSON. AJAX is asynchronous, meaning that XML or JSON data loading does not interfere with normal HTML and JavaScript page loading.

An example of an AJAX call performed through JavaScript is provided below. A request, using the "POST" method, for the file "myfile.php", residing on the server, is being performed. A data string, containing the names and values of the parameters, is transmitted through the POST method. In the case of *success*, a function(data) is defined, in order to process the data received from the server. In the case of *error*, another function is defined, in order to display an error message.

```
$.ajax({
type: "POST", //or "GET"
url: "http://server_name/folder/myfile.php",
data: dataString, //
dataString='param_1='+val_1+…+'param_n='+val_n
cache: false,
success: function(data) { /*data processing*/},
error:function() {
 alert ("Error while comunicating with the server!");
}
});
```

The code for "myfile.php" should look like:

```
<?php
header("Access-Control-Allow-Origin: *");
header("Content-Type: application/json; charset=UTF-8");

$conn = mysqli_connect("myServer", "myUser", "myPassword", "Northwind_DB");

$result = mysqli_query("SELECT CompanyName, City, Country FROM Customers
where Country='".$_POST['Country']."')";

$outp = "";
while($rs = mysqli_fetch_array($result, MYSQLI_ASSOC)) {
    if ($outp != "") {$outp .= ",";}
    $outp .= '{"Name":"'  . $rs["CompanyName"] . '",';
    $outp .= '"City":"'   . $rs["City"]        . '",';
    $outp .= '"Country":"'. $rs["Country"]     . '"}';
}
$outp ='{"records":['.$outp.']}';
$conn->close();

echo($outp);
?>
```

As we can notice, the output is returned in the form of a JSON structure.

II. Assignments

1. Create a JSON structure for the student data. Save it in the local storage. In the same web page, create a hyperlink for a second web page. Within the second web page, get the data from the local storage. Display the obtained student data using an html table.
2. Using an AJAX call, retrieve the data from the Exams database, from the students table. Display these data within the html page, using an html table.

III. Additional Reading

- XML and JSON structures:
https://www.w3schools.com/js/js_json_objects.asp
https://www.w3schools.com/xml/dom_intro.asp

- Local storage and session storage:
https://developer.mozilla.org/en/docs/Web/API/Window/localStorage
https://www.w3schools.com/html/html5_webstorage.asp

- AJAX and AngularJS:
https://www.w3schools.com/xml/ajax_intro.asp
https://www.w3schools.com/angular/

Laboratory no. 11 – 14
Individual assignment

I. Theoretical Considerations:

• Cascaded Style Sheets

Cascaded style sheets allow the user to personalize their html web interfaces. Usually, a separate file is created in order to define user styles (e.g. "mystyle.css"). This file can be invoked within the head of the html file, using the *link* tag (<link>), as shown below:

```
<head>
<link rel="stylesheet" type="text/css" href="mystyle.css" />
</head>
```

The styles defined within the html style sheets correspond to the elements of the html web page, such as hyperlinks, paragraphs, divisions (divs).
Examples of style definitions within an ".css" style sheet are provided below:

```
#standard {font-family:Verdana;color:#000000;font-size:x-small}

#link_a:link{color:#000000;text-decoration:none;font-size:x-
small;font-family:Verdana}
#link_a:active{color:#000000;text-decoration:none;font-size:x-
small;font-family:Verdana}
#link_a:visited{color:#000000;text-decoration:none;font-size:x-
small;font-family:Verdana}
#link_a:hover{color:#000000;text-decoration:none;font-size:x-
small;font-family:Verdana;font-weight=bold}
```

#

First, a standard style is defined, where the font family, color and size properties are set. Then, some styles are provided for a usual hyperlink, for an active hyperlink, for a visited hyperlink and for the situation when the mouse cursor is above the hyperlink.

The #standard style can be implemented, for example, within a html page section, or within a paragraph, in the following manner:

```
<div id=standard ….> </div>
```

In the hyperlink case, the html code will look like:

```
<a id=link_a href="my_second_page.html"> My Second Web Page</a>
```

• PHP Session Variables

In order to transmit values among the pages of a web site during a session, we use session variables. *Session* means the period devoted to the web-site runtime, beginning

with the moment that correspond to its opening within the browser, until the moment when the user closes the browser.

In PHP, the session variables can be implemented using the predefined associative array, $_SESSION[‘var_name’], where *var_name* is the name of the session variable.

A practical situation of employing session variables is that in which, after the authentication procedure, we want to transmit the user name among the pages of the web site. For this purpose, we create the following session variable, assuming that the PHP variable $uname contains the user name value:

```
session_start( );
$_SESSION[‘user_name’]=$uname;
```

The session_start() function should be called firstly, in order to enable the work with session variables.

In the other pages of the web site, we should check if the session variable containing the user name was set and otherwise, to forbid the user access within that page. For this purpose, the predefined PHP function *isset* should be employed:

```
if(isset($_SESSION[‘user_name’]))
    # display the web page
else
   # display an error message and prohibit the user access
```

II. Individual assignment:

1. Build your own website

a. Build the database in MySQL or Microsoft SQL Server

b. Build the authentication page

c. Build the other web-pages using html, php and eventually other scripting languages for web page construction.

2. Use *Session Variables*.

3. Build your own stylesheet (“.css” file) and integrate it in the “.html” page using the link tag, inside of the header.

Use different styles on different portions of text – e.g. – for the title use the font Verdana, color: red, size: 3 and for the other text use the font Arial, size 2.

Use different styles on different groups of links. Define colors for active link, for visited link, for normal link, and use a distinct color when the mouse cursor passes over the link (“hover”).

Models: “mystyle.css”, in the laboratory folder, on the network.

III.Additional reading:

MySQL Tutorial

http://dev.mysql.com/doc/refman/5.1/en/

Cascaded Style Sheets

http://www.w3schools.com/css/default.asp

PHP Session Variables

Appendix – Complements and theory reminder, in order to assist the achievement of individual assignments:

1. HyperText Markup Language:

- predominant markup language for web pages
- provides a means to describe the structure of text-based information in a document — by denoting certain text as headings, paragraphs, lists, and so on — and to supplement that text with interactive forms, embedded images, and other objects
- written in the form of labels (known as tags), surrounded by angle brackets
- can also describe, to some degree, the appearance and semantics of a document
- can include embedded scripting language code which can affect the behavior of web browsers and other HTML processors
- You can, of course, use a WYSIWYG (What You See Is What You Get) HTML editor to make websites but they have 3 main disadvantages:
    - They sometimes use excess code to create a look on a page which slows down loading times
    - They do not always create fully compatible code
    - Some WYSIWYG editors change any HTML code you enter by hand

1.1 HTML tags:

<tag>
There are two types of tag. Opening and closing tags. Closing tags are only different as they have a / before them:
</tag>

- Nearly all tags have a closing tag but a few do not (e.g.
).

Ex. :

```
<html>
    <head>
    <title>Untitled</title>
</head>

<body>
    …
</body>
</html>
```

Details about tags:

- <html>
    - beginning of an HTML document
- <head>
    - beginning of the header section - contains configuration options for the page
- <title>Untitled</title>
    - tells the browser what to display as title of page
    - appears in the title bar at the top of the browser

- </head>
  - end of the header section
- <body>
  - Everything between these is in the body of the page
  - This is where all text, images etc. are.
  - This is the most important part of the page.
- </body>
- </html>
  - end of the HTML document
- tag
  - display the text in a standard font
  - tag attributes
  - text

- tag
  - stands for Paragraph; used to break up text into paragraphs
  - tag has attribute which can be added to it - align option
  - Text
  - can use the tag or the
  - hardly ever necessary to use the align="left" attribute as nearly all browsers automatically align text to the left

-
 tag
  - want to leave a space after your paragraphs
  - should use the
 (break) tag
  - there is no end tag
  -
 =
 =
 =
  - Tags are case insensitive

- <img> tag
  - Images are added to pages
  - must use the *src* attribute to choose the image to insert
  - can either be a relative reference or a direct reference
  - can resize images inside the browser using two other image attributes: width and height
  - Alt tells the browser what the alternative text for an image should be if the browser has images turned off
- <a> tag
  - used when creating hyperlinks and bookmarks; stands for anchor
  - need to use the href attribute
  - to make a piece of text or an image into a hyperlink you contain it in:
  - Link
- <table> tag
  - used in order to define tables within HTML pages
  - define a table row: the <tr> tag

- define a table cell: the <td> tag
- a table definition should look like:
- <table>
  <tr><td>First row, first cell</td><td>First row, second cell</td></tr>
  <tr><td>Second row, first cell</td><td>Second row, second cell</td></tr>
  </table>

- tag
  - used in order to define unordered(bulleted) lists
- tag
  - used in order to define ordered (numbered) lists
- tag
  - used in order to define list items
- Example („<ul>" can be replaced by : „<ol>")

```
<ul>
 <li>First item</li>
 <li>Second item</li>
  ……….
</ul>
```

1.2 Hyperlinks:

- Web Page or Site <a href="http://www.webaddress.com/folder/page">
- Local Page <a href="pagename.html">
- Local Page In A Folder Level Below <a href ="foldername/pagename.html">
- Local Page In A Folder Level Above <a href ="../pagename.html">
- Open E-mail Program With E-mail Addressed <a href ="mailto:yourname@yourname.com">
- Bookmarked Section <a href="#bookmarkname">
- Bookmarked Section In Another Page <a href ="pagelocation.htm#bookmarkname">

1.3. Bookmarks:

- instead of changing the href variable use the name variable. For example:
-
- will create a bookmark called top in the text
- can then link to this using a standard hyperlink:
- Back To Top

1.4. Setting Up Forms:

- A form is an area that can contain form elements.
- Form elements are elements that allow the user to enter information in a form
- A form is defined with the <form> </form> tags

```
<form action="process.php" method="post">
```

1.4.1. Text Fields:

- when you want the user to type letters, numbers, etc. in a form

```
<form>
First name:
<input type="text" name="firstname">
<br>
Last name:
<input type="text" name="lastname">
</form>
```

- in most browsers, the width of the text field is 20 characters by default

1.4.2. Radio Buttons:

- used when you want the user to select one of a limited number of choices

```
<form>
<input type="radio" name="sex" value="male"> Male
<br>
<input type="radio" name="sex" value="female"> Female
</form>
```

- Note that only one option can be chosen

1.4.3. Checkboxes:

- used when you want the user to select one or more options of a limited number of choices

```
<form>
I have a bike:
<input type="checkbox" name="vehicle" value="Bike" />
<br />
I have a car:
<input type="checkbox" name="vehicle" value="Car" />
<br />
I have an airplane:
<input type="checkbox" name="vehicle" value="Airplane" />
</form>
```

1.4.4. Drop Down list:

```
<form>
<select name="cars">
<option value="volvo">Volvo</option>
<option value="saab">Saab</option>
```

```
<option value="fiat" selected="selected">Fiat</option>
<option value="audi">Audi</option>
</select>
</form>
```

1.4.5. Text-area:

- multi-line text input control; user can write text (an unlimited number of characters) in the text-area

```
<textarea rows="10" cols="30">
Initial text to be displayed in text-area.
</textarea>
```

1.4.6. Submit button:

```
<input type="submit" value="Submit">
<input type="button" value="Hello world!">
<input type=“reset" value=“Reset">
<input type=“hidden" value=“Hidden">
```

1.4.7. Putting all together:

```
 <form action="process.php" method="post">
…
<input name = …>
<select name = …>
<input type="submit" value = …">
<input type="button" value = …>
…
 </form>
```

2. PHP script:

2.1. Testing PHP scripts:

- write file
- upload it to your Web Server (e.g. “www” folder of *mowes portable*)
- using your browser, go to the URL of the script
  http://localhost/*...path from root of WebServer*…/script.php

2.2. Outputting Variables in php:

```
<?php
$welcome_text = "Hello and welcome to my website.";
print($welcome_text);
?>
```

- in php variable names begin with “$”

2.3. Formatting Text:

```
print("<font face=\“FontFace\" size=\“FontSize\" color\“FontColor\“
>Hello and welcome to my website.</font>");
```

2.4. *IF* Structure:

```
IF (something == something else)
{
 THEN Statement
} else {
 ELSE Statement
}
```

2.5. WHILE, DO WHILE:

```
while (expr)
   statement

do statement
while (expr)
```

2.6. FOR:

```
for (expr1; expr2; expr3)
statement
```

2.7. Arrays:

- $array[key] = value;
- should always use quotes around a string literal array index
  - For example, use $foo['bar'] and not $foo[bar]
- do not quote keys which are constants or variables, as this will prevent PHP from interpreting them
- It works because PHP automatically converts a bare string (unquoted string which does not correspond to any known symbol) into string
- PHP may in future define constants which, unfortunately for your code, have the same name

2.8. User-defined functions:

```
function name($arg1, $arg2, ..., $argn)
{
   echo "Example function.\n";
   return $retval;
}
```

2.9. Functions:

- All functions and classes in PHP have the global scope - they can be called outside a function even if they were defined inside and vice versa
- PHP does not support function overloading
- Function and classes names are case-insensitive
- support variable numbers of arguments to functions
- It is possible to call recursive functions
  - avoid recursive function/method calls with over 100-200 recursion levels as it can smash the stack and cause a termination of the current script

2.10. File include:

- When a file is included, the code it contains *inherits* the variable scope of the line on which the include occurs. Any variables available at that line in the calling file will be available within the called file, from that point forward. However, all functions and classes defined in the included file have the global scope.

2.11. Basic include() example:

- GeneralFunctions.php

```
function DataBaseConnect ($arg1, $arg2, ..., $argn)
{
}
```

- test.php

```
include 'GeneralFunctions.php';
DataBaseConnect ();
```

2.12. Predefined Variables:

- Server variables: *$_SERVER*
- Environment variables: *$_ENV*
- HTTP Cookies: *$_COOKIE*
- HTTP File upload variables: $_FILES
- Request variables: $_REQUEST
- Session variables: $_SESSION
- ODBC access several databases that have borrowed the semantics of the ODBC API
  - unified into a single set of ODBC functions

2.13. Connecting to databases:

```
bool mysql_connect (
    [string $servername
    [, string $username
    [, string $password
    [, bool $new_link]]]] )
bool mysql_select_db ( string $database_name
        [, resource $link_identifier] )

$conn = mysql_connect('MYSQLSERVER', 'sa', 'password');
mysql_select_db('my data-base', [$conn]);
```

2.14. Counting Rows

```
$num=mysql_num_rows($result);
int mysql_num_rows ( resource $result )
```

- will set the value of $num to be the number of rows stored in $result (the output you got from the database)
  - can then be used in a loop to get all the data and output

2.15. Setting Up The Loop

- $i is the number of times the loop has run and is used to make sure the loop stops at the end of the results so there are no errors

```
$i=0;
while ($i < $num)
{
CODE
$i++;
}
```

2.16. Assigning To Variables the Data extracted from the database

string mysql_result (resource $result, int $row, mixed $field)

- field can be the field's offset, the field's name or the field's table dot field's name (tablename.fieldname)

```
$variable=mysql_result($result,$i,"fieldname");
```

2.17. Recommended high-performance alternatives:

- mysql_fetch_row()
- mysql_fetch_array()
- mysql_fetch_object()

2.18. Combining The Script Format the output

```
echo "<b><center>Database Output</center></b><br><br>";
$i=0;
while ($i < $num) {
$first=mysql_result($result,$i,"first");
…
$web=mysql_result($result,$i,"web");
        echo "<b>$first $last</b><br>Phone: $phone<br>Mobile:
$mobile<br>Fax: $fax<br>E-mail: $email<br>Web:
$web<br><hr><br>";
        $i++;
        }
```

3. Normalization:

- process of taking entities and attributes that have been discovered and making them suitable for the relational database
- process does this by removing redundancies and shaping data in manner that the relational engine desires
- based on a set of levels, each of which achieving a level of correctness or adherence to a particular set of rules
- rules formally known as forms, normal forms
- First Normal Form(1NF)
  - which eliminates data redundancy and continue through to
- Fifth Normal Form (5NF)

- which deals with decomposition of ternary relationships

- each level of normalization indicates an increasing degree of adherence to the recognized standards of database design
- as you increase degree of normalization of your data, you'll naturally tend to create an increasing number of tables of decreasing width (fewer columns)

3.1. Why Normalize?

- eliminate data that's duplicated, chance it won't match when you need it
- avoid unnecessary coding needed to keep duplicated data in sync
- keep tables thin, increase number of values that will fit on a page (8K) – decrease number of reads that will be needed
- maximizing use of clustered indexes – allow for more optimum data access and joins
- lowering number of indexes per table - indexes are costly to maintain

3.1.1. Eliminating duplicated data

- any piece of data that occurs more than once in the database => increased probability for errors to happen

3.1.2. Eliminating anomalies

- INSERT
- DELETE
- UPDATE

⇒ Easy to keep database consistent;

3.2. Process of Normalization

- take entities that are complex and extract simpler entities from them
- continues until every table in database represents one thing (simple entity) and every column describes that thing

3.2.1. Categories of normalization steps

- entity and attribute shape
- relationships between attributes
- multi-valued and join dependencies in entities

3.2.2. Entity and attribute shape. First Normal Form

- all attributes must be atomic, that is, only a single value represented in a single attribute in a single instance of an entity
- all instances of an entity must contain the same number of values
- all instances of an entity must be different
- entities have a fixed number of attributes, and tables have a fixed number of columns
- entities should be designed such that every attribute has a fixed number of values associated with it

- example of a violation of this rule in entities that have several attributes with same base name suffixed (or prefixed) with a number, such as Payment1, Payment2, and so on

3.2.2.1. Programming Anomalies avoided by First Normal Form

- modifying lists in single column
- modifying multipart values
- dealing with a variable number of facts in an instance

3.2.2.2. Clues that design is not in First Normal Form

- string data that contains separator-type characters
- attribute names with numbers at the end
- tables with no or poorly defined keys

3.2.3. Second Normal Form:

- **relationships *between* non-key attributes and part of the primary key**
  - entity must be in First Normal Form.
  - each attribute must be a fact describing the entire key
  - technically relevant only when a composite key *(a key composed of two or more columns)* exists in the entity
  - Definition – A relation R is in the second normal form (FN2) if it is in FN1 and *every nonkey attribute* is totally dependent on every relationship key

3.2.4. Third Normal Form

- **relationships between non-key attributes**
  - entity must be in Second Normal Form.
  - non-key attributes *cannot describe* other non-key attributes
  - Definition: A relation R is in the third normal form (FN3) if it is in FN2 and none of the non-key attributes is not functionally dependent on another non-key attribute of the relation.

3.2.5. BCNF (Boyce Codd Normal Form)

- **relationships between non-key attributes and any key**
  - entity is in First Normal Form.
  - all attributes are fully dependent on a key
  - every determinant is a key
  - BCNF extends previous normal forms by saying that each entity might have many keys, and all attributes must be dependent on one of these keys
  - Third Normal Form table which does not have multiple overlapping candidate keys is guaranteed to be in BCNF
  - Third Normal Form table with *two or more overlapping candidate keys* may or may not be in BCNF

- Definition – A relation R is in the Boyce-Codd Normal Form (BCNF), if, for every functional dependency X→A from R, where A is an attribute that doesn't belong to X => X is a key, or includes a key from R**.**

3.2.6. Multivalue Dependencies. Fourth Normal Form (4NF).

- Third Normal Form is generally considered pinnacle of proper database design
- serious problems might still remain in logical design
- Definition: we say that there exists a **multi-value dependency of the attribute Z on Y**, or that Y performs a multi-determination on Z, Y→→Z, if, for every values x1, x2, y, z1, z2, where x1≠x2, z1≠z2, such that the tuples (x1,y,z1) and (x2,y,z2) belong to R, then also the tuples (x1, y, z2) and (x2, y, z1) belong to R.
- Fourth Normal Form-Definition: a relationship R is in the fourth normal form if, for every multivalue dependency, X→→Y, then X is a key or includes a key in R

→ Condition equivalent with the definition:

- entity must be in BCNF, and
- there must not be more than one multivalue dependency between an attribute and the key of the entity

→ Table is in 4NF if and only if, for every one of its non-trivial multivalued dependencies $X \rightarrow\rightarrow Y$, X is a super key, X is either candidate key or a superset thereof

- Fourth Normal Form violations:

- ternary relationships
- lurking multivalued attributes

3.2.7. Coupling dependency. Fifth Normal Form (5NF):

- not every ternary relationship can be broken down into two entities related to a third
- aim of 5NF is to ensure that any ternary relationships that still exist in 4NF, can be decomposed into entities without loss of information
- eliminates problems with update anomalies due to multivalve dependencies
- Definition: **Coupling Dependency** < == > Consider R(A1,A2,..,An) a relation schema and R1, R2, .., Rk subsets of {A1, A2,.., An}. There is a join dependency called *(R1, R2, …, Rk) if and only if any instantiation r of R is the result of coupling between its projections R1, R2,…,Rk,

  r = πR1(r) |><| πR2(r)….. |><| πRk(r)

  => *(R1, R2, R3) is a join dependency on the relation R.

- Definition: A relation is in **5NF** if and only if the coupling dependencies that exist in a relation are implied by a key of the relation.

3.3. Denormalization

- used primarily to improve performance in cases where over-normalized structures are causing overhead to query processor
- whether slightly slower (but 100 percent accurate) application is not preferable to a faster application of lower accuracy
- during logical modeling, we should never step back from our normalized structures to performance-tune our applications proactively

Bibliographic references:

[1] R. Ramakrishnan, J. Gehrke, “Database Management Systems, 3rd Edition”, McGraw Hill Higher Education, 2007
[2] R. Dollinger, „Utilizarea sistemului SQL Server (SQL 7.0, SQL 2000)”, Ed. Albastra, 2001
[3] G. Preda, *Aplicatii cu baze de date*, Editura Matrix Rom, Bucuresti, 2014
[4] SQL Server documentation - SQL Server Books Online, http://msdn2.microsoft.com/en-us/sql/
[5] HTML Tutorial- http://www.2createawebsite.com/build/html.html
[6] PHP Tutorial – http://www.php.net/manual/en/
[7] Java Script Tutorial- https://www.tutorialspoint.com/javascript/