Building and Execution of Queries for Educational Process Management System

Ara Arakelyan
Yerevan State University
Yerevan, Armenia
e-mail: ara.arakelyan@ysu.am

Ani Balasanyan
Yerevan State University
Yerevan, Armenia
e-mail: anibalasanyan@yahoo.com

ABSTRACT
The main participants of university's educational process need a lot of information for their daily activities related to the students, exams, modules, students' assessments etc. Thus, the educational process management system must provide a way of retrieving the requested accessible information easily and quickly. The purpose of this work is to design and implement a subsystem for the system called IAMUniver, which will allow the end users to create queries and execute them in order to retrieve necessary information. IAMUniver is now being used in the faculty of Informatics and Applied Mathematics of YSU on a test environment.

Keywords
Query, abstract table, graphical language, translation

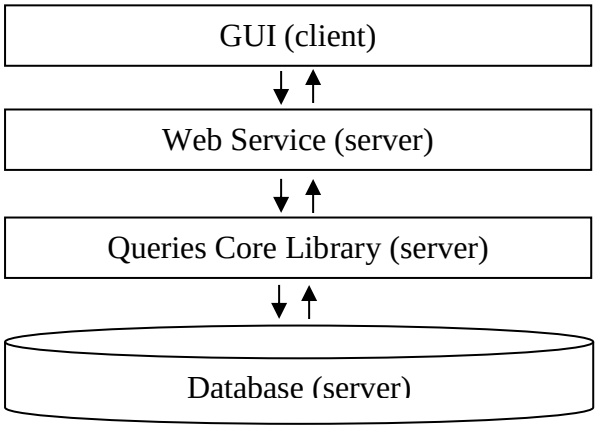
1. INTRODUCTION
The subsystem described in this paper has been designed taking into account the following requirements:

- Expandable - it must be possible to build new queries, execute them and share them with other users.
- Easy to use - it must not be mandatory for the end users to have technical skills or to know details of system implementation in order to build queries or execute them.

In order to meet the mentioned requirements, it has been decided to provide an ability of building queries using special graphical language, which is based on the usage of abstract tables. Rows of those abstract tables are such entities, which are used by the end users during their everyday work. Query building and execution subsystem provides the following main functionalities:

- Building new query via graphical language or via SQL.
- Executing selected query and seeing the result as a table.
- Exporting the result of query execution for further analysis using third party tools, such as Microsoft Excel.
- Saving created queries, editing or deleting existing ones.

2. ARCHITECTURE OF SUBSYSTEM
Query building and execution subsystem consists of the following layers:



- Database - is responsible for providing data required for execution of queries as well as for keeping all shared queries. This subsystem only adds few tables and stored procedures to the existing database (designed using Microsoft SQL Server [1]) used by the whole system.
- Queries Core Library - is .NET assembly [2] responsible for implementation of business logic related to the query building and execution.
- Web Service - is responsible for interaction between the client and server parts of the subsystem and has been developed using WCF Framework [3].
- GUI - is responsible for allowing the end users to build queries, execute them and see the results, export results of query execution etc. It is just part of an existing WPF [4] application used by the whole system.

2.1. Graphical User Interface and Abstract Tables Language
Query building and execution subsystem is represented by one section in GUI of the whole system. The name of this section is Queries and it allows the end users to perform the following operations:

1. Create graphical queries using graphical language based on abstract tables.
2. Create advanced queries using SQL language [1]. Only the users familiar with database schema could create such queries.
3. Save queries locally in a client device or share them by saving them in a database. Queries are saved in XML format via usage of .NET XmlSerializer class [5].
4. Search the required queries among the saved ones.
5. Delete the found queries or open them for viewing or editing.
6. Execute the found queries or currently creating/editing queries and see the results of execution as a table.
7. Export the results of query execution in CSV format.

Graphical queries consist of three main parts. The first part is the selection from the list of predefined abstract tables: Students, Lecturers, Groups, Professions, Faculties, Universities, Curriculums, Modules, Exams and Exam Assessments. The second part is the selection of columns from the list of selected abstract tables' columns. The third part is filter rows, which could be connected to each other using AND/OR operations. Each filter row contains the name of some selected abstract table, column name from that table, comparison operation and the value to which column content must be compared during query execution.

2.2. Security
In order not to allow query building and execution subsystem to make any changes in the data related to the educational process, this subsystem connects to the database (during query execution) in context of such user, which has read-only permissions to the entire database. This particularly eliminates the possibility of executing advanced que-

ries, which contains SQL commands of data modification. This subsystem also takes care of allowing only the authorized users to edit or delete the shared queries.

3. QUERY EXECUTION

Execution of advanced queries is trivial, because in that case we already have an SQL query that must be executed. For execution of graphical queries first of all we need to translate them into SQL queries [1]. In this section we will describe the algorithm of query translation from graphical queries into SQL queries.

3.1. Necessary Notations and Definitions

Let us formalize the notion of graphical query and the notion of graphical condition used in filtering part of graphical query.

Let $AbsTables = \{R_1, \dots, R_h\}$ be a fixed set of abstract tables used in query building graphical language. Also let $Columns(R_i)$ be a columns set of abstract table R_i, $i = 1, \dots, h$ and $AbsColumns = \bigcup_{j=1}^{h} Columns(R_j)$. Each column $C \in AbsColumns$ has a data type assigned to it, which will be denoted by $Type(C)$ and could be $String$, $Number$ or $DateTime$.

***Definition* 1.** The triple $\langle C, Op, V \rangle$ is called a graphical condition, where $C \in AbsColumns$, $Op \in \{=, \neq, <, >, \leq, \geq, StartsWith, EndsWith, Contains\}$ is comparison operation such, that $Op \in \{=, \neq, <, >, \leq, \geq\}$ in cases when $Type(C) \neq String$ and V is a constant value of type $Type(C)$.

***Definition* 2.** The pair $\langle S, W \rangle$ is called a graphical query, where $S = \langle C_1, \dots, C_k \rangle, C_i \in AbsColumns, i = 1, \dots, k, k > 0$ is a tuple of one or more selected columns from the selected abstract tables and W is a condition of the query, which is a tuple of the following form: $W = \langle U_1, O_1, \dots, U_{m-1}, O_{m-1}, U_m \rangle$, U_i is a graphical condition and $O_j \in \{AND, OR\}$ is an operation of combining conditions, $i = 1, \dots, m-1, j = 1, \dots, m, m \geq 0$.

During work of query translation algorithm several fixed mappings are used, which are presented below:

- $Map: AbsColumns \rightarrow DBColumns$, where $DBColumns$ is a set of pairs of database table and its column. This mapping shows from where information of the given abstract column must be retrieved.
- $Priority: DBTables \rightarrow N$, where $DBTables$ is a set of database tables and N is a set of natural numbers. This mapping is used for determining order in which some database tables must be joined during work of query translation algorithm.
- $Cond: DBTables \times DBTables \rightarrow SQLCond$, where $SQLCond$ is a set of conditions of SQL language. This mapping is used for determining condition used when joining two database tables during work of query translation algorithm.
- $Path: DBTable \times DBTables \rightarrow DBTables'$, where $DBTables'$ is a set of tuples of database tables. This mapping is used for determining intermediate tables, with the aid of which two database tables must be joined.

3.2. Query Translation Algorithm

Now let us present the algorithm of translation from graphical queries to SQL queries.

Query translation Algorithm.

Input: Graphical query $\langle S_0, W_0 \rangle$, where $S_0 = \langle C_1, \dots, C_k \rangle$, $W_0 = \langle U_1, O_1, \dots, U_{m-1}, O_{m-1}, U_m \rangle$ and $U_i = \langle C_i', Op_i, V_i \rangle$, $i = 1, \dots, m,\ k > 0, m \geq 0$.

Output: SQL query.

1. ***LET*** $T \coloneqq \langle\,\rangle$ be an empty list of database tables, $TC \coloneqq \langle\,\rangle$ be an empty list of database table columns and $S \coloneqq$ "", $F \coloneqq$ "", $W \coloneqq$ "" be empty strings.
2. ***FOR*** $i \coloneqq 1$ ***TO*** k ***DO***
   LET $\langle t, c \rangle \coloneqq Map(C_i)$
   IF $t \notin T$ ***THEN*** add t at the end of the list T
   IF $c \notin TC$ ***THEN*** add c at the end of the list TC
3. ***FOR*** $i \coloneqq 1$ ***TO*** m ***DO***
   LET $\langle t, c \rangle \coloneqq Map(C_i')$
   IF $t \notin T$ ***THEN*** add t at the end of the list T
4. ***FOR EACH*** c ***IN*** TC ***DO***
   IF $S =$ "" ***THEN*** $S \coloneqq S + c$ ***ELSE*** $S \coloneqq S + ',' + c$
5. $S \coloneqq$ "$SELECT\ DISTINCT$ " $+ S$
6. ***FOR EACH*** t ***IN*** T ***DO***
   FOR EACH t' ***IN*** T ***DO***
   IF $t \neq t'$ ***THEN***
   FOR EACH t'' ***IN*** $Path(t, t')$ ***DO***
   IF $t'' \notin T$ ***THEN*** add t'' at the end of the list T
7. Sort list T in ascending order of weights of its elements, where weight function is $Priority$.
8. ***LET*** $TCount$ be count of elements in list T.
9. $F \coloneqq T[1]$
10. ***FOR*** $i \coloneqq 2$ ***TO*** $TCount$ ***DO***
    $F \coloneqq F +$ "$\ JOIN\ $" $+ T[i] +$ "$\ ON\ $" $+ SQLCond(T[i-1], T[i])$
11. $F \coloneqq$ "$\ FROM\ $" $+ F$
12. ***FOR*** $i \coloneqq 1$ ***TO*** m ***DO***
    LET $\langle t, c \rangle \coloneqq Map(C_i')$
    IF $i > 1$ ***THEN*** $W \coloneqq W + ' ' + O_i + ' '$
    IF $Op_i \in \{=, \neq, <, >, \leq, \geq\}$ ***THEN***
    $W \coloneqq W + c + Op_i + V_i$
    ELSE
    IF $Op_i = StartsWith$ ***THEN***
    $W \coloneqq W + c +$ "$\ LIKE\ $" $+ (V + '\%')$
    IF $Op_i = EndsWith$ ***THEN***
    $W \coloneqq W + c +$ "$\ LIKE\ $" $+ ('\%' + V)$
    IF $Op_i = Contains$ ***THEN***
    $W \coloneqq W + c +$ "$\ LIKE\ $" $+ ('\%' + V + '\%')$
13. $W \coloneqq$ "$\ WHERE\ $" $+ W$
14. ***RETURN*** $(S + F + W)$

3.3. Future Work

It's planned to make some further enhancements in graphical query language, two of which are listed below:

- Ability to use aggregation functions (sum, max, min etc.) in graphical condition and also in column selection part of the query.
- Ability to specify expression (that could contain references to abstract columns) as last component of graphical condition instead of just using constant values.

REFERENCES

[1] L. Davidson, J. Moss, *Pro SQL Server 2012 Relational Database Design and Implementation*, Apress, 2012.

[2] A. Troelsen, *Pro C# 5.0 and the .Net 4.5 Framework*, Apress, 2012.

[3] J. Lowy, *Programming WCF Services, 3rd edition*, O'Reilly Media, 2010.

[4] C. Sells, I. Griffiths, *Programming WPF, 2nd Edition*, O'Reilly Media, 2007.

[5] S. Livingstone, S. Fraser, *Beginning C# XML: Essential XML Skills for C# Programmers*, Wrox Press, 2002.