

SQL APPLICATION FOR SECONDARY SCHOOL LEAVING EXAMINATION

Lorentz JÄNTSCHI

Technical University of Cluj-Napoca, Romania, <http://lori.academicdirect.ro>

Mariana MARCU

*Hidromecanica Schooling Group,
Braşov, Romania*

Sorana BOLBOACĂ

*Iuliu Haţieganu Medicine and Pharmacy University,
Cluj-Napoca, Romania*

Abstract

This paper main objective is to present SQL power of use for a case of data management with heterogeneous data classification. The article is based on a PHP application developed to manage a real situation of secondary school leaving examination in August 2003 at Braşov city, Romania at exams center no. 29, where the first author was president of the examination committee. The paper describes all used queries that was used for classifications, room repartitions, probes and investigation subject envelopes and finally, describe de final classification algorithm.

Keywords: SQL application, MySQL database server, PHP programming.

INTRODUCTION

Main information about the secondary school leaving examination is required [1]. The examination is organized into Exam Centers, where participate candidates form more than one school (in presented case 12 schools) with varied specializations (in our case 12 specializations). Note that one school has in most of cases candidates with more than one specialization (in our case from 1 to 5). For example, produce 5 rows a query like:

```
SELECT DISTINCT `pl` FROM `candidati`  
WHERE `sc` LIKE 'N. Titulescu'
```

School-leaving examination is organized by probes. All candidates that were studied in Romanian language in school must sustain 6 probes. For the candidates that were studied in mother tongue in school must sustain 8 probes. Probes are 2 (or three, respectively) eliminatory and are viva voce and the candidates obtains a qualifying result for every probe (one from: admitted, discarded and not present). The 'a' probe is obligatory for all candidates and is Romanian language and literature. The 'c' probe is for candidates with mother tongue in school. According to candidate specialization for the last three probes, the candidates has option to sustain at every probe the exam at one discipline from a list of one or more disciplines at partial free choosing (as example, if candidate specialization is philology, the candidate can sustain 'b' probe at one modern language, and at 'e' probe must sustain the exam at the other one language according to his school studies. For example, in our case, for 'b' probe, we have three modern languages, that result from select phrase:

```
SELECT DISTINCT `pb` FROM `candidati`
```

After the eliminatory probes (2 or 3) all candidates are allowed to participate at rest of the probes (4 or 5). These probes are noted with marks from 10.00 to 1.00.

To promote school-leaving examination, candidate must pass the eliminatory probes (with admitted qualification) and must have mark of at least 5.00 for every other ones probes. More, two decimals chopped average mean of all noted with marks probes must be at least 6.00.

A database to store all required information is necessary. More, with PHP [2] in conjunction with MySQL [3] a set of programs to process efficiently the information the task is fast and efficient.[4]

DATABASE DESIGN

A database called Bac2003 with a set of 16 tables was created (see fig. 1).

Because a set of varied number from 2 to 6 Exam Center (EC) are goes to one Zonal Center of Evaluation (ZCE) where all hand writes exam papers are evaluated and marked ('a' and 'c' probes, and some disciplines from 'e' and 'f' probes) the database are designed to manage all exam results from more than one EC (as example, all EC depended at one ZCE).

The table 'ex' is used for both user management and EC classification. It store in 'id' field the identification number of EC (in our case from 29 to 34), in field 'nume' the name of operator (usually one of the EC committee members) and his password. Note that for this table 'id' is not a primary key, but using a SELECT DISTINCT SQL phrase we have a column that act as primary key for 'ex', 'candidati' and 'scoli' tables. More, from here it result as consequence that for an EC we can have more than one operator. There exists a super user for user management, with a blank 'id' key. A script for user management was created. The script use three SQL phrases, one for SELECT, one for INSERT and one for UPDATE in table 'ex'. For example, the phrase which and add a new operator using users.php program at \$HTTP_POST_VARS['id'] EC with \$HTTP_POST_VARS['nume'] name and \$HTTP_POST_VARS['password'] password is:

```
INSERT INTO `ex` VALUES (  
$HTTP_POST_VARS['nume'], $HTTP_POST_VARS['id'],  
PASSWORD($HTTP_POST_VARS['password']))
```

Table 'sc' stores the names of schools ('name' field, candidate key), EC number ('ex' field, foreign key) and primary key ('id' field, auto increment). Table 'abs' contain only 'nume' field that stores school graduation years (from

2003 descending) which act as primary key also. Table 'pg' store only two values to discriminate the study curricula of the candidates (candidates from 2003 school graduation year with 12 graduation years has one type of curricula, and the all others, i.e. previously graduated candidates and 2003 graduated candidates with 13 graduation years). Table 'pl' store profiles of studies (such as philology, economics, and so on, 12 distinct profiles in our case). The 'id' field is primary key and 'nume' field is the candidate key. The tables 'pa', 'pb', 'pc', 'pas', 'pcs', 'pd', 'pe' and 'pf' contains the names and the primary keys for the disciplines by probes, every table storing all possible disciplines of one probe. The table c contain the labels and primary keys of qualification marks (i.e. 'adm.' for admitted, 'res.' for not admitted, and 'nep.' for not presented).

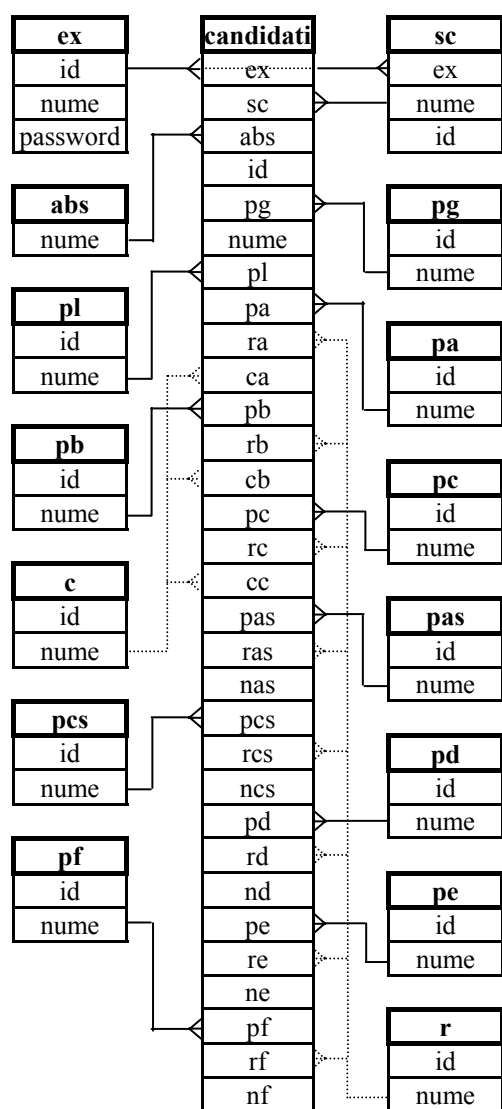


Fig.1. Bac2003 database architecture

Table 'candidati' is designed to store all information about candidates. A script to store the primary information (candidate name) was designed and called candidats.php. The candidats.php program makes INSERT candidates in table 'candidati' successively for an EC by provenience schools. The required information is candidate's names and inserted information is composed from 'id' (primary key), 'nume' (candidate first and second name, not necessary candidate key), 'ex' (EC, foreign key from 'ex' table) and 'sc' (provenience school, foreign key from 'sc' table).

The second step is to fill all others information about candidates into candidate records (defined by 'id' key). A set of PHP programs was prepared for this task. The program query.php retrieves from 'candidati' table for selected candidates a set (one or more fields) of information and, for every field (excepting 'nas', 'ncs', 'nd', 'ne', 'nf' fields) create a combo box with all possible values from primary tables (see fig. 1). Then, the candidate's registration procedure is fast and easy. More, the program can be executed any time for updates or eventually corrections. The program do.php verifies the updates and for not empty values makes UPDATE at 'candidati' table.

PROBLEMS AND SOLUTIONS

The problems that appear in school-leaving examination procedure are produced by the complexity of examination methodology.

First, the candidates must provide a list with options for every partial free selection examination probe (for example, the candidates with humanistic profiles, i.e. philology, social sciences, and so on can choose one of two options: Geography or History for 'd' probe in the mean while candidates with realistic profiles don't have options, the examination at 'd' probe is from Mathematics, but depending on profile, graduation year, and type of study curricula, resulting 6 types of subjects of Mathematics). More, for all probes, the control key for correct options is candidate profile, graduation year and study curricula.

At the beginning of the school-leaving examination period, it must displayed a list with all candidates and there options alphabetically sorted. Once we have 'Bac2003' database, a SELECT command is enough:

```
SELECT * FROM 'candidati' WHERE 'ex' LIKE '29'
ORDER BY 'nume' ASC
```

To format data on page, following script additions in PHP program are proper:

```
<style>.lori{text-align:center;font: Bold 8pt. Times New
Roman;}</style>
```

```
<br clear=all style='mso-special-character:line-break; page-break-
before:always'>
```

where style are used to display data in a table and page break are inserted after n=20 candidates.

For the first day of school-leaving examination (SLE), a list with all candidates which must sustain 'a' eliminator probe must be provided, in same alphabetically order for time scheduling and room repartitions:

```
SELECT 'id', 'nume', 'ex', 'sc', 'abs', 'pg', 'pl', 'pa' FROM
'candidati' WHERE 'ex' LIKE '29' AND 'ra' LIKE " ORDER
BY 'nume' ASC
```

Also, for verification, a list with candidates that has 'a' probe recognized from previous sessions, but sorter first by school and second by name is very useful:

```
SELECT 'id', 'nume', 'ex', 'sc', 'abs', 'pg', 'pl', 'pa', 'ra', 'ca'
FROM 'candidati' WHERE 'ex' LIKE '29'
AND 'ra' LIKE 'd' ORDER BY 'sc' ASC, 'nume' ASC
```

At the end of 'a' probe, after the updating of the database, a list with admitted candidates is:

```
SELECT 'id', 'nume', 'ex', 'sc', 'abs', 'pg', 'pl', 'pa', 'ra', 'ca'
FROM 'candidati' WHERE 'ex' LIKE '29'
```

```
AND 'ra' LIKE " AND 'ca' LIKE 'adm.' ORDER BY nume'
```

The not present candidates are:

```
SELECT 'id', 'nume', 'ex', 'sc', 'abs', 'pg', 'pl', 'pa', 'ra', 'ca'
FROM 'candidati' WHERE 'ex' LIKE '29'
```

```
AND 'ra' LIKE " AND 'ca' LIKE 'nep.' ORDER BY 'nume'
```

The not admitted candidates are:

```
SELECT 'id', 'nume', 'ex', 'sc', 'abs', 'pg', 'pl', 'pa', 'ra', 'ca'
FROM 'candidati' WHERE 'ex' LIKE '29'
```

```
AND 'ra' LIKE " AND 'ca' LIKE 'res.' ORDER BY 'nume'
```

Before the beginning of 'b' probe, a list with all candidates that must sustain this probe, ordered by discipline and by name must be generated:

```
SELECT 'id', 'nume', 'ex', 'sc', 'abs', 'pg', 'pl', 'pb' FROM
'candidati' WHERE 'ex' LIKE '29' AND 'ca' LIKE 'adm.' AND
'rb' LIKE " ORDER BY 'pb', 'nume'
```

For verifications, the candidates with 'b' probe recognized from previous sessions are obtained from:

```
SELECT 'id', 'nume', 'ex', 'sc', 'abs', 'pg', 'pl', 'pb', 'rb', 'cb'
FROM 'candidati' WHERE 'ex' LIKE '29'
AND 'rb' LIKE 'd' ORDER BY 'nume' ASC
```

After updating the database with 'b' probe results, the list of all candidates that can sustain following probes is:

```
SELECT * FROM 'candidati' WHERE 'ex' LIKE '29'
AND 'ca' LIKE 'adm.' AND 'cb' LIKE 'adm.'
ORDER BY 'nume' ASC
```

For candidates with mother tongue studies the 'c' probe is necessary and eliminatory:

```
SELECT 'id', 'nume', 'ex', 'sc', 'abs', 'pg', 'pl', 'pc' FROM
'candidati' WHERE 'ex' LIKE '29' AND 'pc' NOT LIKE "
AND 'rc' LIKE " ORDER BY 'pc', 'nume'
```

The candidates with recognized results at 'c' probe from previous sessions are:

```
SELECT 'id', 'nume', 'ex', 'sc', 'abs', 'pg', 'pl', 'pc', 'rc', 'cc'
FROM 'candidati' WHERE 'ex' LIKE '29' AND 'pc' NOT
LIKE " AND 'rc' LIKE 'd' ORDER BY 'pc' ASC, 'nume'
```

After the finishing of viva voice eliminatory probes, all accepted candidates for following probes are:

```
SELECT * FROM 'candidati' WHERE 'ex' LIKE '29' AND
'ca' LIKE 'adm.' AND 'cb' LIKE 'adm.' AND
(cc' LIKE 'adm.' OR 'pc' LIKE ") ORDER BY 'nume'
```

Concordant with 'a', 'b' and 'c' results and previous sessions results, the list of candidates for 'as' probe are:

```
SELECT 'id', 'nume', 'ex', 'sc', 'abs', 'pg', 'pl', 'pas' FROM
'candidati' WHERE 'ex' LIKE '29' AND 'ca' LIKE 'adm.' AND
'cb' LIKE 'adm.' AND 'ra' LIKE " AND
(cc' LIKE 'adm.' OR 'pc' LIKE ") ORDER BY 'nume'
```

Room's repartition of candidates for 'as' probe is obtained from limit clause:

```
SELECT 'id', 'nume', 'ex', 'sc', 'abs', 'pg', 'pl', 'pas' FROM
'candidati' WHERE 'ex' LIKE '29' AND 'ca' LIKE 'adm.' AND
'cb' LIKE 'adm.' AND 'ra' LIKE " AND
(cc' LIKE 'adm.' OR 'pc' LIKE ")
ORDER BY 'pg', 'nume' LIMIT 0,20
```

and so on (LIMIT 20, 20, ...) until all candidates are fetched (for rooms by 20 places).

For all following probes, the lists of disciplines for which exam subjects are necessary are required. The list for 'cs' disciplines is:

```
SELECT DISTINCT 'pcs' FROM 'candidati' WHERE 'ex'
LIKE '29' AND 'ca' LIKE 'adm.' AND 'cb' LIKE 'adm.' AND
(cc' LIKE 'adm.' OR 'pc' LIKE ") AND 'rcs' LIKE " AND 'pcs'
NOT LIKE " ORDER BY 'pcs' DESC
```

Analogue, for 'd', 'f' and 'e' probe disciplines lists are obtained from previous select phrase by replacing of 'cs' with 'd', 'e' and 'f' respectively.

Room's repartitions for 'cs' mother tongue writes probe:

```
SELECT 'id', 'nume', 'ex', 'sc', 'abs', 'pg', 'pl', 'pcs' FROM
'candidati' WHERE 'ex' LIKE '29' AND 'ca' LIKE 'adm.' AND
'cb' LIKE 'adm.' AND 'cc' LIKE 'adm.'
AND 'rcs' LIKE " ORDER BY 'pl' ASC, 'nume' ASC
```

Candidates with 'cs' recognized probe:

```
SELECT 'id', 'nume', 'ex', 'sc', 'abs', 'pg', 'pl', 'pcs', 'rcs', 'ncs'
FROM 'candidati' WHERE 'ex' LIKE '29' AND
'ca' LIKE 'adm.' AND 'cb' LIKE 'adm.' AND
'cc' LIKE 'adm.' AND 'rc' LIKE 'd' ORDER BY 'pl', 'nume'
```

Candidates for 'd' probe:

```
SELECT 'id', 'nume', 'ex', 'sc', 'abs', 'pg', 'pl', 'pd' FROM
'candidati' WHERE 'ex' LIKE '29' AND 'rd' LIKE " AND 'ca'
LIKE 'adm.' AND 'cb' LIKE 'adm.' AND
(cc' LIKE 'adm.' OR 'pc' LIKE ")
ORDER BY 'pd' ASC, 'pg' ASC, 'nume' ASC
```

Candidates for 'f' probe are split in two, because 'EdF' and 'EdF-v' probes (sport) are practically:

```
SELECT 'id', 'nume', 'ex', 'sc', 'abs', 'pg', 'pl', 'pf' FROM
'candidati' WHERE 'ex' LIKE '29' AND 'rf' LIKE " AND 'ca'
LIKE 'adm.' AND 'cb' LIKE 'adm.' AND
(cc' LIKE 'adm.' OR 'pc' LIKE ") AND (pf LIKE 'EdF%')
ORDER BY 'pf' ASC, 'pg' ASC, 'nume' ASC
```

for sport and for all others are replaced ('pf LIKE 'EdF%') clause with ('pf NOT LIKE 'EdF%').

For 'e' probe, is possible to have practical tests, such as in our case. The practical disciplines are 'StFor' and 'Instr' so the previous select must be updated as follows:

- for 'StFor' (picture): 'f' replaced with 'e' in 'pf' and 'rf' and 'EdF%' replaced with 'StFor';
- for 'Instr' (music): 'f' replaced with 'e' in 'pf' and 'rf' and 'EdF%' replaced with 'Instr';
- for all others: 'f' replaced with 'e' in 'pf' and 'rf' and ('pf NOT LIKE 'EdF%') replaced with ((pf NOT LIKE 'StFor') OR (pf NOT LIKE 'Instr'));

RESULTS AND DISCUSSION

To make required lists, all that we have to do is to submit queries to the SQL server (in our case MySQL) and to fetch the results. One solution is to create a drop down list with all SQL phrases for selection and more, for room repartition to create two input boxes for start value and limit value from LIMIT clause. Probably is the best option.

Because the program was created and completed in school-leaving examination period from 17 august to 29 august 2003 and the real situation was not known before the designed interface is adaptive one that allow user to construct SQL phrases. Note that is not a perfect one, because do not consider the 'c' mother tongue possibly eliminated candidates (conditional OR in SELECT). Anyway, for a small number of mother tongue candidates it works perfectly.

The database management was makes on PHPTriad under Windows XP, because the unavailability of internet connection, and after the school-leaving examination the database and the programs was ported into a FreeBSD operating system with Apache web server. The entry can be found at the address:

http://academicdirect.ro/virtual_library/exams/Bac2003/

The entry interface is password restricted, as we described below (fig. 2).

Fig. 2. Bac2003 entry point

The action allowed (fig. 2) is to select the EC (C.E. noted in figure) and the action (select, update, add schools, add candidates, add profile, make means). For select action, a menu like in fig. 3 is displayed.

Fig. 3. SELECT phrase construction

First list is field list (<select multiple name='fields[]' size='\$cols_n'>) a header message follow, then filter list (up to 5 filters), then ordering criterions (up to 5, order is strict), a footer message, a drop down list to draw supplementary columns (from 0 to 4) in resulted table for supplementary information like time scheduling, room, marks, exam paper serial number. Both filter and order drop down lists contain all fields from 'candidati' table and was retrieved using another SQL phrases:

SHOW COLUMNS FROM 'candidati' and
SHOW TABLES FROM 'Bac2003'

The update interface is designed as in fig. 4:

Fig. 4. UPDATE phrase construction

FINAL REPORTS

At the end of school-leaving examination period, the final results must be provided. A registration book leaf with all candidates ordered by name containing probe results is the final document of examination. The document it contain also average mean of the candidates for the candidates which obtain at least minimum requirement marks for all probes. To succeed the school-leaving examination the average mean must be at least 6.00. More, a qualifying

label are assigned to every candidate, one from 'reusit' (for succeeded school-leaving examination candidates), 'respins' (for not succeeded school-leaving examination candidates) and 'neprezentat' (for not present candidates at one or more probes). The average mean and qualification label require a program to computes. A program called media.php was build. The program manages also the registration book leaf extracts by provenience schools (for diplomas elaboration). So, the select phrase are again from user interface constructed (fig. 5):

Fig. 5. Means and qualifying labels presetting interface

The SELECT phrase is constructing dependent of the filter (page break filter) selection:

SELECT * FROM 'candidati' ORDER BY \$sc `nume`

where \$sc can have a empty value or \$sc="sc", "".

MySQL query and fetch sequence is:

```
$q=mysql_query($query);
while($r=mysql_fetch_array($q)){
    ...
}
```

where \$query it contain the SELECT phrase as string, \$q is MySQL handler for fetching management and \$r is an object oriented array with elements constructed from MySQL database. Every fetching produce a updating of \$r values, that is \$r['name'] is candidate name, \$r['sc'] is provenience school and so on.

A variable called \$media was used to store the average mean value and a variable called \$c was used to store qualifying label. To avoid the errors produced by floating point approximations and to display the mean with first two decimals some artifices are applied:

```
$media=($r['nas']+$r['ncs']+$r['nd']+$r['ne']+$r['nf'])*100;
//collect 2 decimals in integer part of the sum
if ($r['ncs']) $media /=5; else $media /=4;
//make average mean from sum
if ( $r['nas'] < 5) $media = 0;
if ( ( $r['pcs'] ) && ( $r['ncs'] < 5 ) ) $media = 0;
if ( $r['nd'] < 5 ) $media = 0;
if ( $r['ne'] < 5 ) $media = 0;
if ( $r['nf'] < 5 ) $media = 0;
//mean are not applied for marks under 5.00
$media = (string) $media;
//string forced conversion (to "freeze" the value)
$media = substr($media,0,strpos($media,"."));
//extract the integer part
$media = (int) $media;
//int forced conversion (to print as example 9.1 in place of 9.10)
$c = "reusit";
//hypothesis
while ( $c == "reusit" ) {
    //proper qualification label assignment, false cycle
```

```

if( $r['ca'] == 'nep.')
{ $c = "neprezentat"; break; }
if( $r['ca'] == 'res.')
{ $c = "respins"; break; }
if( $r['cb'] == 'nep.')
{ $c = "neprezentat"; break; }
if($r['cb'] == 'res.')
{ $c = "respins"; break; }
if( !$r['nas'])
{ $c = "neprezentat"; break; }
//the marks for `pas` probe are published after the last probe
if(( $r['cc'] )&&( !$r['ncs'] ))
{ $c="neprezentat"; break; }
//the marks for `pcs` probe are published after the last probe
if( !$r['nd'])
{ $c="neprezentat"; break; }
if( $r['nd'] < 5 )
{ $c="respins"; break; }
if( !$r['nf'])
{ $c="neprezentat"; break; }
if( $r['nf'] < 5 )
{ $c="respins"; break; }
if( !$r['ne'])
{ $c="neprezentat"; break; }
if( $r['ne'] < 5 )
{ $c="respins"; break; }
if( $r['nas'] < 5 )
{ $c="respins"; break; }
if( $r['cc'] )&&( $r['ncs'] < 5 )
{ $c="respins"; break; }
//the maks results for `pas` and `pcs` probes
if( $media < 600 ) $c = "respins";
break; //while
} //the end of qualification label assignment
if( $media ) {
    $media = ( (float) $media ) / 100.0;
    $media = sprintf("%.2f", $media);
} else {
    $media = "-";
}
//the end of media preparation for printing

```

CONCLUSIONS

The school-leaving examination procedure is complex one, with a large set of variants, cases and possibilities. It requires that at least one member of examination committee that has good qualification in computer using and database management.

The presented application uses hardly the performances of SQL kernel from MySQL database server for multiple filters, multiple sorting and specific fetching of the records.

The PHP implementation of the programs shortens the implementation time at the most and makes an easy to run and update applications.

The using of SQL phrases in this paper instead of programs presentation was didactical scope, to understand the nature of the problem and solution.

ACKNOWLEDGMENTS

Thanks to Mr. Prof. Aurel TĂRĂU, from Bihor School Inspectorate District, Romania, from first author for advices and help in his first school-leaving examination presidency in June 2003.

REFERENCES

- [1] Ecaterina ANDRONESCU (Ministry), Annex no. 2 at MEC no. 4328 Directive from 30.08.2002 about Calendar and Methodology of Organizing and Unfolding of School-leaving Examination - 2003,
http://academicdirect.ro/virtual_library/exams/Bac2003/MetodologieBac2003.pdf
- [2] www.zend.com
- [3] www.mysql.org
- [4] Lorentz JÄNTSCHI, Automat Server Side Processing of Statistical Data, UNITECH'02 International Scientific Conference, November 21-22 2002, Gabrovo, Bulgaria, vol. "ISC UNITECH'02 GABROVO Proceedings", p. 185-189.