Combining the procedural and the set-based approaches in the teaching of SQL SELECT statements in the introductory databases course

Philip Petrov, Daniel Djolev
Assistant-professor in Technical University - Sofia, Bulgaria, philip@abv.bg
Part-time assistant in Technical University – Sofia, Bulgaria, daniel.djolev@abv.bg

***Abstract**: We present the two popular approaches in the teaching of SELECT statements in the introductory database course – the procedural approach and the set-based approach. Our study shows that the procedural approach comes very natural to the students and they tend to prefer it; however it leads them to a "dead end" on very complicated SELECT queries, especially when there are two or more aggregate functions involved. Our tests showed that a sudden switch to the set-based approach for the complicated queries only is not giving excellent results. Therefore we propose introduction of the set-based approach earlier in the course and teaching it in parallel with the procedural one.*
***Keywords:** SQL, SELECT, teaching, aggregate functions, join, MySQL, relational databases.*

1. INTRODUCTION

While there are many books for teaching SQL, only few describe methods for solving complicated problems. Most of the authors seem to be focused on showing and describing the different features of the SQL language and they show different solutions to many problems; however they do not provide much information about the path (the steps that the student must go through) from the problem description to its final solution. It is our understanding that the best for a person to effectively memorize and understand is to reach the solutions himself and rediscover the general conclusions on his own. The role of the teacher should be to provide the heuristic methods to students and to error-correct them if needed. As mentioned in [5] "*Studies show that advanced database skills are important for students to be prepared for today's highly competitive job market*". Giving "ready answers" to common problems is not very effective because it does not prepare students for working in unknown areas like when they need to provide custom solutions to real-life problems when they start to work. There are cases in the practice when a person must write a select query with many nested subqueries and tens of joins – there aren't many such examples in textbooks.

The author in [3] writes that "*Despite their simple appearances, queries involving set comparison are very difficult to formulate in relational query languages*". While introductory courses are usually not working on problems with too complicated set operations, they must eventually reach few at the end because this will be the connecting point with the Advanced SQL course which will be studied later. Given that requirement, our focus is to provide methods to simplify the solving of problems with two or more aggregate functions over different sets and make them solvable even for the introductory course.

We chose MySQL in our exercises for few reasons [4], the most important of which is its big popularity in the practice. That was also a big step-forward towards teaching SQL compared to the program of the "Computer Systems and Technologies" Databases course in Technical University - Sofia from few years ago when Microsoft Access was used in the lab exams, because as [7] states "*Learners are learning how to use Mi-*

crosoft Access rather than learning how to write SQL". As the time showed, this change was evaluated as very positive both by the students and the professors and MySQL was approved as the better system for teaching SQL.

2. THE PROCEDURAL APPROACH

What we call a "procedural approach" in the teaching of SQL is the following four step procedure that the student follows after reading the problem description:

- Find what information is needed in the final result set, then write it down in the beginning of the statement after the SELECT clause;
- Join all the tables that are involved in the FROM clause;
- Write the restrictive conditions in the WHERE clause;
- Finish the query by extending it with the clauses GROUP BY, ORDER BY, HAVING and LIMIT corresponding to the problem description.

In general the most important thing that the students must be able to do in order to solve a problem with that approach is to be able to clearly separate the joining and the restrictive conditions. Due to our experience the majority of the issues are coming from incorrect grouping in complicated queries and incorrect joins (especially when there are null values and OUTER JOIN has to be used). Let's look into a simple example db:

- *theaters(id, name, city)* – cinemas in different cities with unique id (PK);
- *screens(no, theater_id, type)* – each cinema has many screens of different types(for example normal, VIP, deluxe). The screens in each cinema are numbered 1, 2, 3, etc. The PK is the combination (no, theater_id);
- *movie(id, name)* – each movie has a name and an unique id;
- *shows(time, screen_no, theater_id, movie_id, visitors)* – there are projections at a different time on different screens and each one has a number of visitors.

We need to solve the following problem: "*Make a list with the names of the cinemas, the numbers of the screens and the times of the projections for the movie Fast And Furious 7 in screens of type VIP and deluxe. Sort the list in alphabetical order first by the name of the cinema and then by the numbers of their screens*". Solving that problem with the procedural approach is straight-forward:

- The problem description asks for "*names of the cinemas, the numbers of the screens and the times of the projections*", so we put these columns at the start of the query: *SELECT theaters.name, screens.no, shows.time*;
- The involved tables are theaters, screens and shows, so we join them:
  ```
  FROM cinemas
  JOIN screens ON theaters.id = screens.theater_id
  JOIN shows ON (screens.no,          screens.theater_id) =
                (shows.screen_no, shows.theater_id)
  ```
- The restrictive conditions are "*for the movie Fast And Furious 7*" and "*screens of type VIP and deluxe*" - we add these conditions in WHERE clause:
  ```
  WHERE screen.type IN ("VIP", "deluxe")  AND
        shows.movie_id IN(
           SELECT movie.id FROM movie
           WHERE name="Fast and furious 7"
        );
  ```

- The reminder of the description is about sorting the result set:
  ORDER BY theater.name, screen.no;

Combining the four components one after another produces the final query. As mentioned earlier, this is a very natural order for solving problems and the students tend to prefer to solve the problems that way – they simply translate the human readable text to the SQL language straight-forward - it is like "write what you read" method. Adding a single aggregate function will not complicate that approach a lot – based on our experience the only difficult thing for the students is to determine the column(s) for the GROUP BY clause. Of course it is possible for the teacher to "fool" the students by giving them unclear problem descriptions; however that should not be the correct intention when teaching, especially in the introductory course when students are still practicing the common problems and have not yet reach the step of mastering SQL. We have also found out that it is a good practice to introduce comments in the large SELECT queries (commenting is almost obligatory when teaching stored procedures later). It is also a good propedeutics for the hints [6] which will follow later in the optimization topics.

A bigger complication when using this approach appears when there are two or more aggregate functions involved in one SELECT query. Consider the following problem: "*Provide a list with the names of the theaters, the sum of all their visitors (from all their screens) for the last month and the count of their VIP screens, but only for theaters from the city named Sofia*". Using the procedural approach, one may start the query by writing *SELECT theaters.name, SUM(shows.visitors), COUNT(screens.*)* and this will end-up with an incorrect result at the end. The problem is that the two aggregate functions are working on completely different sets – the SUM function must sum all visitors from all screens while the COUNT must work with the VIPs. One solution in this particular case is to covert the COUNT to a SUM with a result from IF function as parameter:

SELECT theaters.name, SUM(shows.visitors), SUM(IF(screens.type="vip", 1, 0)) …

It is needless to say that this approach is not universal – it will work when one of the sets is a subset of the other. Given different relations will need a different "trick" to solve the problem. Our opinion is that this will overcomplicate the introductory database course and it will go beyond its primary target – to teach the students the SQL basics.

3. THE SET-BASED APPROACH

What we call a "set-based approach" are the following three steps:

- Split the problem to a set of simpler problems;
- Solve the smaller problems;
- Combine the solutions by intersecting their result sets.

It is obvious that this approach is a variation of the popular "divide and conquer" method for problem solving.

Following the last example we can split the problem description in the following three simpler problems, the solutions to which are pretty straight-forward:

- Provide a list with the names of the theaters from a city named "Sofia":

```
SELECT id AS theater_id, name AS theater_name
FROM theaters
WHERE city = "Sofia";
```

- Find the sum of all visitors for the last month for each theater:

```
SELECT theater_id, SUM(visitors) AS visitors_sum
FROM shows
WHERE time > DATE_SUB(NOW(), INTERVAL 1 MONTH)
GROUP BY theater_id;
```

- Count of the VIP screens for the theaters:

```
SELECT theater_id, COUNT(*) as vips_count
FROM screens
WHERE type = "VIP"
GROUP BY theater_id;
```

A special note should be taken: we always include the primary and foreign keys of the queries with no aggregate functions and the group by column of the aggregate queries – we will need them in order to join the results later. Making proper column aliasing also helps for better readability later when we combine the results:

```
SELECT sofia_th.theater_name, sum_all_th_vis.visitors_sum, th_vip_scrns.vips_count
FROM ( SELECT id AS theater_id, name AS theater_name
       FROM theaters
       WHERE city = "Sofia"
     ) AS sofia_th  /* ids and name of theaters from Sofia */
JOIN (  SELECT theater_id, SUM(visitors) AS visitors_sum
       FROM shows
       WHERE time > DATE_SUB(NOW(), INTERVAL 1 MONTH)
       GROUP BY theater_id
     ) AS sum_all_th_vis  /* ids of theaters and sum of visitors for the last month */
    ON sofia_th.theater_id = sum_all_th_vis.theater_id
JOIN (  SELECT theater_id, COUNT(*) as vips_count
       FROM screens
       WHERE type = "VIP"
       GROUP BY theater_id
     ) AS th_vip_scrns  /* ids of theaters and the count of their vip screens */
    ON sofia_th.theater_id = th_vip_scrns.theater_id;
```

The only eventual complication in the set-based approach is at the very beginning – it is sometimes hard for the students to correctly split the big problem in smaller parts. Here the teacher must emphasize on the minor details in the problem description [1] because they are the stumbling block which may lead to incorrect understanding.

4. A PROPOSAL FOR AN INTEGRATED APPROACH

What we have found out from our practice is that if we start teaching the students using the procedural approach and we switch to the set-based approach at the end of the SELECT statements topic (the only place where the complicated problems appear), we are losing the interest of many students. They start thinking that the new approach is very complicated and they tend to focus on the procedural approach only, even if they

know that it will result in lower assessment for them at the end of the course. The set-based approach is not complicated at all; however, the problems themselves are not simple and this is what is making students reject the approach itself. We think that the issue is coming from the fact that we introduce two new things at the same time - new problems and a new method. The mass student rejection to work by the new method is resulting in lower number of the highest grades in the end. Working only with the procedural approach is not giving better results too – the custom solutions seem to be too complicated for the introductory course. Therefore our proposal is to introduce the set-based approach to the students much earlier in the course as an alternative to the procedural one. Thus students can solve some simple problems in the two different ways. In our opinion, such methodology has three major benefits:

- The majority of the students will be comfortable in using the set-based approach with the simpler problems, so they will make transition to the harder problems more fluently – it will not be a "big jump" in the subject contents;
- Showing that two complex queries in SQL happen to have the same meaning, although they are very different from a syntactic point of view [2] is also an easy way to troubleshoot and detect errors. This is especially helpful when dealing with columns with possible null values which is one of the most common issues in complicated queries and according to our experience is often neglected by the students;
- That way the teachers can raise optimization questions (like the trivial "*which of the two solutions is the better one*"). The optimization topic is usually introduced much later in the course (or even in a separate advanced SQL course like the one in the Master's program in Technical University – Sofia); however this approach can be used as good propedeutics in the introductory course. The optimization must not be an emphasis here but only an additional viewpoint for the more talented students only.

5. TEST RESULTS

Our pilot study was conducted in 2014 and 2015 with students from specialty "Computer and Software Engineering" in Technical University – Sofia. The "Databases" subject is in the second semester in the second year of the Bachelor's degree and it is divided in two parts – the first part mainly covers database design and SELECT queries, and the second part is more focused on UPDATE, DELETE queries, transactions and stored procedures with lower emphasis on the database design topics. It is important to note here that we do not teach the database design and the SQL topics separately "one after another" but we integrate them. That approach is giving much better transition from the abstract data modeling to the practical SQL representation [8]. We are introducing some general rules for both "abstract design-to-SQL" transitions, the "SQL-to-abstract design" transitions and elements from the "SQL-to-relational algebra" [2] transitions. It was the good results from that integrated approach that influenced us for this study.

We compared the experimental group results from 2015 with the results of a control group from 2014 where the assistant in the lab exams was the same person (thus we are trying to minimize "the different teacher factor"). Our focus is only on the first test (it was performed in the middle of the course right after the end of the SELECT statements topics) because it will show the method differences more clearly than the final tests. For result comparison we used the Mann-Whitney U-test. We got a "z-value" of 1.73 which

is lower than the critical 1.96 and means that the experimental and the control groups must be considered statistically equal. If we compare the average points for the test (the range is from 0 to 20), the control group has average 8.79 and the experimental group has 11.50 which is an increase of more than 30%. The result is clearly showing that the new methodology affected only a small part of the experimental group. Only few of the students took benefit from the new approach while the rest of the group did not show any major difference. There was no surprise that it was again the "two or more aggregate functions over different sets in one query" problem which was the most difficult for the students in the experimental group; however we have much more students in the higher-end (16 points or more) which is considered to be from a very good to an excellent result. If we exclude these top students from the statistics in both the control and the experimental groups, we will get again statistically equal groups with the Mann-Whitney test (with a z-value of just 0.03!); however this time the averages are almost equal – 8.15 for the experimental group and a slightly higher 8.35 for the control group.

6. CONCLUSION

We expect that the introduction of the set-based approach earlier in the course in parallel with the procedural approach will produce much better results in a group of well-motivated students. In our practice the less motivated students tend to focus only on the procedural approach regardless if we introduce the set-based approach in parallel with it or not. Still we have managed to increase the excellent assessments in the test which is sustaining our optimistic hypothesis that the problems we introduce at the end of the SELECT statements topic are not overcomplicated for the introductory course and a better teaching method can help us to keep that material in the curriculum.

7. REFERENCES

[1] Caldeira, C. (2008), Teaching SQL: A Case Study. *ITiCSE '08 Proceedings of the 13th annual conference on Innovation and technology in computer science education* (ISBN: 978-1-60558-078-4), 340-340

[2] Ceri, S., Gottlob, G. (1985) Translating SQL into Relational Algebra: Optimization, Semantics, and Equivalence of SQL Queries. *IEEE Transactions on Software Engineering Vol.* SE-1I, No. 4, April 1985, 324-345;

[3] Dadashzadeh, M. (2003) A Simpler Approach in Set Comparison Queries in SQL. *Journal of Information Systems Education,* (ISSN 1055-3096) Vol. 14(4), 345-348;

[4] Denton, J. (2003) Selection and Use of MySQL in a Database Management Course. *Journal of Information Systems Education,* (ISSN 1055-3096) Vol. 14(4), 401-407;

[5] Hauser, K. (2007) Teaching Advanced SQL Skills: Text Bulk Loading. *Journal of Information Systems Education,* (ISSN 1055-3096) Vol. 18(4), 399-402;

[6] Lokhande, A.D., Shete, R.M. (2012) The Use of Hints in SQL-Nested Query Optimization. *Journal of Data Mining and Knowledge Discovery* (ISSN: 2229–6662 & ISSN: 2229–6670) Volume 3, Issue 1, 2012, 54-57;

[7] Renauld, K., Biljon, J. (2004) Teaching SQL —Which Pedagogical Horse for this Course?. *Key Technologies for Data Management*, 3112, 244–256;

[8] Watson, R. (2006) The Essential Skills of Data Modeling. *Journal of Information Systems Education,* (ISSN 1055-3096) Vol. 17(1), 39-41.