MySQL Feature Enhancement

Ante Pimentel (212166187)

Dusan Birtasevic (210451383)

Francis Okoyo (213811336)

Hashim Al-Helli (212172359)

Richard Van (212918652)

York University

Table of Contents

Abstract

This report proposes an enhancement to MySQL version 8. We begin by introducing our proposed enhancement ‘Full Outer Join Operation’, followed by an overview on MySQL and general approaches to implementing the operation. The following section deals with the two approaches we considered for implementing the ‘Full Outer Join Operation’, which are “Combining existing features”, and “Source Code Level Implementation”. We also discuss the advantages and disadvantages as well as possible risks and limitations in this section. The next section covers any impacts on subsystems, and then a depiction of a use case, followed by a discussion on testing methods to verify correctness and functionality. The report concludes by highlighting lessons learned and mentioning some future improvements.

1 Introduction and Overview

When using a query based database management system DBMS, it's important to be able to issue commands using as few queries as possible. Doing this allows for fast and easy interactions with a DBMS as well as a more user friendly experience with said DBMS. MySQL is an open source relational database management system (RDBMS) with lots of functionality and features that have been added over the years of its existence. Despite its large plethora of features, it lacks features that would greatly improve user experience. One such feature is the "FULL OUTER JOIN".

Full outer join is a method of joining to tables such that elements in both tables are represented in one; including terms that do not exist in either. Below is a visual depiction of a full outer join operation using two tables A and B containing student numbers, courses and departments.

Table A	
Student Number	Department
12	Computer Science
38	Math
24	Math
80	Computer Science

Table B	
Student Number	Course
80	MATH 1000
12	EECS 2000
40	EECS 1000
38	MATH 1500

A outer join B on Student Number			
Student Number	Departmnet	Student Number	Course
12	Computer Science	12	EECS 2000
38	Math	38	MATH 1500
80	Computer Science	80	Math 1000
24	Math	<null>	<null>
<null>	<null>	40	EECS 1000

From the table above we see that student "24" is a part of the math department but is not enrolled in any courses. Also, student "40" is enrolled in "EECS 1000" but is

not listed as a part of any department. Situations like these should not occur, therefore members of a university/college registration department may perform a full outer join operation to verify that all students enrolled in a course must be listed as a part of a department and vice versa. Full outer join operations are also useful for inventory tracking, purchase order verification and etc.

Currently if a client wants to perform a full outer join operation they must, emulate it using existing features “LEFT JOIN”, “UNION”, “RIGHT JOIN”. The problem with this approach is that it takes a lot of statements and different feature calls to make it work. Why do that when you could just have one feature do it for you? In the following sections, we will highlight the advantages of the addition of the enhancement.

2 Proposed enhancement

As stated in the introduction, the enhancement we are proposing is the addition of a FULL OUTER JOIN statement in MySQL version 8. This addition would improve the functionality of table operations in the RDBMS

2.1 Stakeholders

The main stakeholders are the clients because they are the many that will benefit from the enhancement. The developers are certainly also stakeholders due to the fact that they must implement this feature. It is also important for them to endure the happiness of their consumers/clients. And of course the owners are indirectly involved.

3 Methods of implementation

In this section, we are going to describe to you the 2 best ways of implementing this features, as well as their benefits and drawbacks. The two methods we’ll be talking about are feature combinations, and ground up (from scratch) implementation.

3.1 Implementation Approach 1: Combine Existing Features

This approach simulates full outer join using pre-existing commands in MySQL. Given a table A and B, we take (LEFT JOIN B on A) and unionize it with (RIGHT JOIN B on A). this leads to some code of the form shown in **figure 1** below. This is what would have to be done by a user every time he/she wants to perform a full join. To implement approach one, we must take the code in **figure 1** and store it within the MySQL source directory and update the dictionary file and resolver to recognize a “FULL OUTER JOIN” and call this code.

```
SELECT  column_names  FROM table-name1
LEFT JOIN table2 ON column_name1=
column_name2
 UNION ALL
SELECT  column_names FROM table_name1
 RIGHT JOIN table_name2 ON table_column1 =
table_column2  WHERE condition
```

Figure 1 : emulated FULL OUTER JOIN

3.1-1 Advantages and Disadvantages

Implementation of this approach is very fast in terms of development time and is less prone to error due to its reuse of existing well tested features. With that being said, it is highly dependent on these statements and if their behavior were to ever change in the future, it may negatively affect the new feature.

3.2 Implementation Approach 2: Source Code Level Implementation

3.2-1 Join Algorithms

There are three common methods that can be used to perform the join operation. These methods include but aren't limited to: nested loop join, the sort-merge join and the hash join.

Nested loop: uses two nested loops to join two tables of data. It must loop through and compare all elements in each table. As a result, this algorithm is Very slow and expensive to use, especially for large tables.

Sort-merge: sorts the two tables of data before performing alternating steps through both tables and builds the output table. Sort-merge is much faster than the nested loop algorithm but must sort two tables of data before merging them, so sort-merge is still slow for large tables.

Hash join: stores each table's relations in a hash table using hash keys derived for each row of the table. It then loops through the hash table and builds the output table based on the hash keys.

The best and most efficient option out of these 3 is the hash join method. It's very fast compared to the other 2 methods; even with very large data sets. Below is an example of the hash join algorithm.

1. Given tables **A** and **B**, scan through **A** and create a hash table from **A** using hash keys based on the **join condition** (defined by the client)
2. Scan through table **B** row by row, computing the hash key for each row
3. Compare **A** and **B** according to the hash key
    - If the hash keys match, store rows from each table into "**Matching**" table
    - If the hash keys do not match:
        - For keys in **A** but not in **B**, store rows with those keys into the "**A not B**" table

- For keys in **B** but not in **A**, store rows with those keys into the ”**B not A**” table

4. Append tables “**A not B**” and “**B not A**” to the ”**Matching**” table.
5. Return the ”**Matching**” table.

Figure 2 : Hash Join algorithm

3.2-2 Advantages and Disadvantages

To better visualize the algorithm consider the following example. A computer parts company deals with several computer manufacturers to purchase replacement parts. The company’s warehouse manager wishes to verify recent purchases so he performs a full outer join operation on a table of suppliers, and a table of purchase orders:

Table: Suppliers	
supplier_id	supplier_name
10002	Microsoft
10003	Apple
10000	IBM
10001	Asus

Table: Orders		
order_id	supplier_id	order_date
500125	10000	2017/11/21
500126	10001	2017/11/22
500127	10004	2017/11/23

Table: Hash Table	
supplier_id	supplier_name
10000	IBM
10001	Asus
10002	Microsoft
10003	Apple

Table: Matching		
supplier_id	supplier_name	order_date
10000	IBM	2017/11/21
10001	Asus	2017/11/22

Table: Suppliers not Matching		
supplier_id	supplier_name	order_date
10002	Microsoft	<null>
10003	Apple	<null>

Table: Orders not Matching		
supplier_id	supplier_name	order_date
<null>	<null>	2017/11/23

Table: Results		
supplier_id	supplier_name	order_date
10000	IBM	2017/11/21
10001	Asus	2017/11/22
10002	Microsoft	<null>
10003	Apple	<null>
<null>	<null>	2017/11/23

After analyzing the results table the warehouse manager sees that no purchases have been made from microsoft or apple. However the final entry in the table attracts his attention. On november the 23rd a purchase was made but it does not coincide with any of the known suppliers. The warehouse manager forwards this information to the accounting department for further investigation.

The benefit of implementing a solution from scratch is that it ensures that the feature is embedded within the system and is readily available to all clients. Despite its relatively fast runtime, hash join requires a large amount of memory to perform its task efficiently; it needs to store the entire hash table in memory. Hash join is also more difficult to implement compared to the sort-merge and nested loop join algorithms.

A workaround method exists if there is not enough memory to perform hash join operation on the input. The grace hash join algorithm can separate the input tables into partitions and perform hash join on each partition before merging the partitions. This makes hash join the preferred algorithm for implementing the JOIN operation.

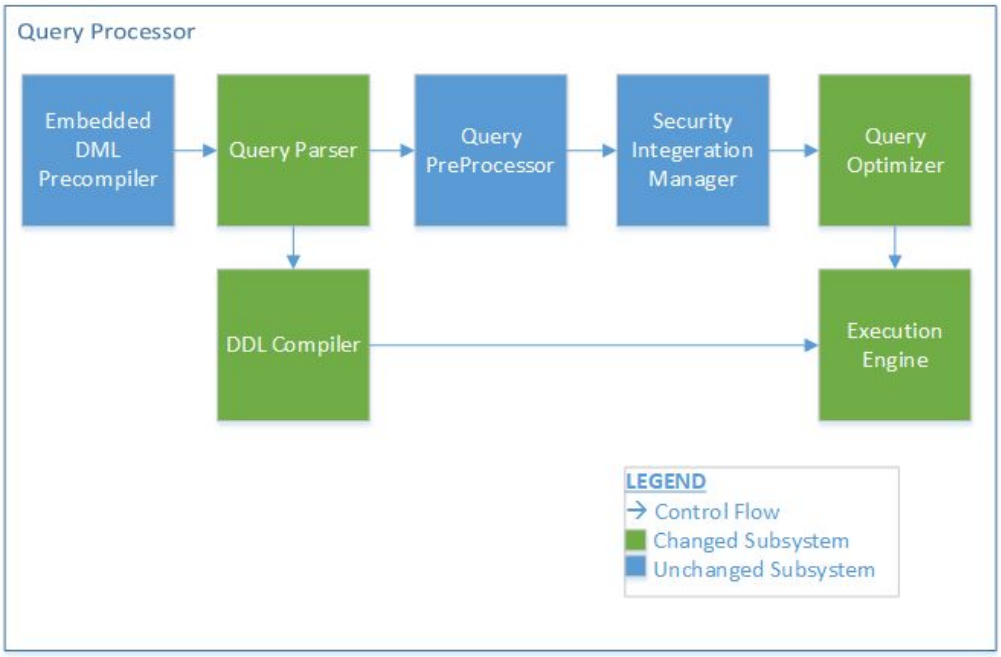
4 Potentially Impact on SubSystems

Excluding test files, the following files and directories are affected (Results found using ‘grep’ command on MySQL source code directory):

- ./storage/ndb/tools/ndbinfo_sql.cpp
- ./rapid/plugin/x/src/admin_cmd_handler.cc
- ./sql/sql_base.cc
- ./sql/item_cmpfunc.h
- ./sql/aggregate_check.h
- ./sql/sql_planner.cc
- ./sql/sql_select.cc

- ./sql/sql_parse.cc
- ./sql/item.h
- ./sql/sql_optimizer.h
- ./sql/item_cmpfunc.cc
- ./sql/sql_optimizer.cc
- ./sql/sql_resolver.cc
- ./sql/dd/impl/system_views/statistics.cc
- ./sql/dd/impl/system_views/routines.cc
- ./sql/dd/impl/system_views/tables.cc
- ./sql/table.h
- ./sql/opt_sum.cc
- ./sql/sql_executor.cc

The Systems that will require changes all reside in the Query Processing layer, these subsystems are : DML Compiler, Query Parser, Query Optimizer, and Execution Engine. These components are highlighted in green in the figure below.



It can be seen from the diagram that architecture of MySQL will not be altered thus leading to preservation of concurrency and all other features will run just as fine since no architectural alteration is made such as adding new subcomponent and so on.

5 Use Case

Below is a simple use case for the execution of the new syntax which has the form **SELECT column-names FROM table-name1 FULL OUTER JOIN table-name2 ON column-name1 = column-name2 WHERE condition**

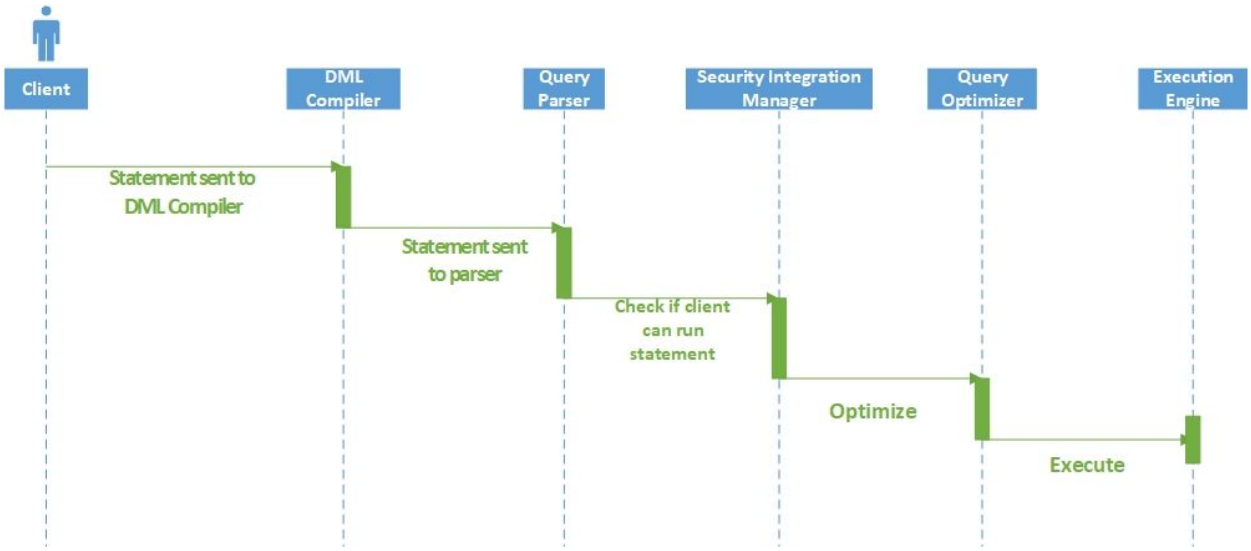


Figure 3: statement execution sequence diagram

Figure 3 shows the path of execution of a statement sent by a client. This is how statements were processed before and after the addition of the feature enhancement. Because of the nature of our enhancement, there are no new dependencies or sub components. The architecture of MySQL would remain exactly the same. Almost like it was never altered.

6 Testing Plans

6.1 Performance Testing Plan

Implementing a test plan for method 2 is relatively simple and can be done in any language. Implementing method 1 would involve changing sql source code and would be very difficult.

Since we can not test against method 1 we chose to instead test against the other variations of method 2, eg one of the other sorting algorithms Nested Loop. We took some old TA files from assignment 3 and decided to compare dependencies with Nested loop and Hash Sort.

Nested Loop	
Dependencies	27448
Runtime (ms)	64041

Hash Sort	
Dependencies	27448
Runtime (ms)	467

The results were conclusive, Hash Sort only took half a second to sort the data while Nested Loop took over a minute.

6.2 Integration Testing Plan

In order to verify the MySQL continues to function as expected we need Integration Testing. This works by combining individual modules/components and testing them as a group. Integration testing would be costly as there are a number of subsystems being modified. The best approach for this would be unit testing for the new syntax and old syntax to see if there are any adverse effects.

7 Lessons Learned

There were many lessons learned while doing the assignment. We have found that the limitations and features that are excluded missing in MySQL exist in other RDBMSs. One thing we can be certain about is that there is always room for improvement in a software system.

We learned that coming up with a new feature is not a simple task especially for without prior knowledge or experience with the system. The design of MySQL makes it easy to extend on it but addition of new features will have consequences ranging from implementation time, implement cost, or introduction of bugs to the system.