Department of Computer Engineering

University of Peradeniya

CO527 Advanced Database Systems

Lab Number :03
Topic : Query Optimization
Due Date : 03rd March 2022 before 11:55 PM

Objective :
At the end of this lab, you should be familiarized with ways to optimize the queries using *EXPLAIN*.

Submission :
Submit your report as E_1X_XXX_lab03.pdf. You can discuss among your classmates, but the submitted assignments should be your own work. Please note that committing plagiarism or assisting others to commit plagiarism will result in zero marks.

---

1. Introduction

When a *SELECT* query does not perform well as you think it should, use the *EXPLAIN* statement to ask the MySQL server for information about how the query optimizer processes the query. When you issue a query, the MySQL Query Optimizer tries to devise an optimal plan for query execution. *EXPLAIN* is one of the most powerful tools available for understanding and optimizing inefficient queries. You can use *EXPLAIN* by placing the keyword *EXPLAIN* in front of *SELECT, DELETE, INSERT, REPLACE*, and *UPDATE* query as of MySQL5.6.3. Earlier versions permit only SELECT statements for this. You can also add the keyword *EXTENDED* after *EXPLAIN* in your query for additional information about the execution plan.

2. Troubleshooting with EXPLAIN

We will be working on the same database *Company* used in previous labs but you may start by dropping all the indexes you created during the last lab.

1. Use *explain* to analyze the outputs of following two simple queries which use only one table access.

   I. SELECT * FROM departments WHERE deptname = 'Finance';
   II. SELECT * FROM departments WHERE deptno ='d002';

   What conclusions you can draw from the results?

2. *Explain* is especially important for join analysis because joins have much potential to increase the amount of server processing.

   Let's work on the following example to demonstrate how to use explain to analyze and optimize a poorly performing query that needs access of two tables. The query is to ask the question **"Which employees have worked for more than 4000 days for an assigned title"** and to display for each employee his/her first name and the number of days he/she has worked on a particular title. This question could be answered using only *titles* table except that to get each employee's first name rather than employee number, title information must be joined with employee information.

   Start by creating the initial tables *emplist* and *titleperiod* as follows. These derived tables need to contain only the columns involved in the query.

   I. create table emplist select emp_no, first_name from employees;
   II. create table titleperiod select emp_no, title, datediff(to_date, from_date) as period FROM titles;

   Now write the query that gives the desired information in the required format.

   Analyze the output of applying *EXPLAIN* to the above query explaining each value. Note that the tables are in their initial unindexed state.

   What could be the number of row combinations that MySQL would need to check?

3. Good columns to index are those that you typically use for searching, grouping, or sorting records. The query does not have any *GROUP BY* or *ORDER BY* clauses, but it does use columns for searching:

   - The query uses emplist.emp_no and titleperiod.emp_no to match records between tables.
   - The query uses titleperiod.period to cut down records that do not satisfy the condition.

I. Create indexes on the columns used to join the tables. In the emplist table, emp_no can be used as a primary key because it uniquely identifies each row.
II. In the titleperiod table, emp_no must be a non-unique index because multiple employees can share the same title:
III. Analyse the outputs of *EXPLAIN* After creating the indexes.

Is it possible to optimize the query execution further? If so, what can be done?

3. Query Rewriting Techniques

Using *explain* to analyze queries give you clues about ways the query might be improved. You can modify the query and then run *explain* again to see output changes. The following query rewriting techniques can be useful.

1. USE/FORCE INDEX - To force MySQL to use an index.
2. IGNORE INDEX - To tell MySQL to ignore an index.

Each of these options is used in the *FROM* clause, following the table name containing the index you want to control.

Ex:
SELECT first_name FROM emplist FORCE INDEX(PRIMARY) WHERE empno>1000;

If you want to force MySQL to join tables in a particular order, begin the query with *SELECT STRAIGHT_JOIN* rather than *SELECT*. This is an instruction to the MySQL query optimizer that the tables must be joined from left to right in the order they are listed in the query.

Assignment:

Write a brief analysis report (maximum 4 pages) summarizing the results of ***EXPLAIN*** used in the examples in the lab sheet and explaining how ***EXPLAIN*** helps in optimizing query execution.

References:

https://www.eversql.com/mysql-explain-example-explaining-mysql-explain-using-stackoverflow-data/

https://dev.mysql.com/doc/refman/5.7/en/using-explain.html

https://dzone.com/articles/how-to-optimize-mysql-queries-for-speed-and-perfor