Query Processing and Optimization

Nabin Giri (700676332)

MSc. Computer Science

University of Central Missouri

Santosh Raj Bista (700688704)

MSc. Computer Science

University of Central Missouri

Sai Wang (700678494)

MSc. Computer Science

University of Central Missouri

***Abstract—*Query Optimization is an important process in the database. Depending on the complexity of database and nature of the query, processing time can vary largely from a fraction of seconds to hours. The purpose of query optimization is to find the way to process a given query with the minimum access time/cost involved in accessing the physical data stored on the disk and communication cost associated with the transmission of data among nodes in distributed database systems.**

**The main agenda of this project was to implement various methodologies of writing efficient queries and comparing them with the usual queries to find the cost difference between them.**

1. *Introduction*

A query is an inquiry into the database using the SELECT statement. A query is used to extract data from the database in readable format according to the user request. It can be as simple as "finding the address of a person with SS# 123-45-6789," or more complex like "finding the average salary of all the employed married men in Warrensburg between the ages 30 to 39, that earn less than their wives." Queries results are generated by accessing relevant database data and manipulating it in a way that yields the requested information (1). Depending on the complexity, size of database and nature of query, processing time can vary largely from fraction of seconds to hours.

1. *Database Used*

We are using *parking* database for our project and it has three tables: location, day and violation. The database was created using dataset (Parking Violation) from Kaggle (2) which has 132,850 rows and 16 columns. Data has been cleaned: noise & missing values in the data has been removed and has been processed to create respective tables.

We have imported the cleaned dataset into respective tables from csv file format. We are using MYSQL as it is open source relational database management system developed by Oracle and we are using MYSQL Workbench to run and analyze our queries execution time.

1. *Ways to Optimize Queries*

We have tested five different ways for optimizing the queries while retrieving the information from MYSQL

Workbench. In the experiment the simple query cost metrics are: Execution time, Number of rows examined, and Number of rows returned (3). None of these metrics is a perfect way to measure query cost, but they reflect roughly how much data MYSQL must access internally to execute a query and translate approximately into how fast the query runs

1. Index

The SQL standard does not provide any way for the database user or administrator to control what indices are created and maintained in the database system. Indices are not required for correctness since they are redundant data structures. However, indices are important for efficient processing of transactions, including both update transactions and queries (4).

Consider below query to retrieve street\_id and value which is greater than 9130.

select street\_id from parking.location where street\_id > '9130';

We have used indexing in the parking.location street\_id.

|  |  |
| --- | --- |
| Query Type | Cost |
| Without using Index | Rows Examined: 3068 rows & Execution Cost: 284.60 |
| With Index | Rows Examined: 967 rows & Execution Cost: 195.71 |

Fig.1. Comparing cost for execution of query with and without index.

1. Union and Union All

UNION is used to combine the result from multiple SELECT statements into a single result set. The default behavior for UNION is that duplicate rows are removed from the result.

Union all is like union, but it does not remove duplicate rows.

We have three tables on our database; day, location and violation. All three tables have different data in them. If we use Union and Union All them, they are going to give us below results.

/\*Union QUERY\*/

select row\_id, object\_id from parking.day a

union select row\_id, address\_id from parking.location b

union select row\_id, violation\_code from parking.violation c;

/\*UNION ALL QUERY\*/

select row\_id, object\_id from parking.day x

union all select row\_id, address\_id from parking.location y

union all select row\_id, violation\_code from parking.violation z;

|  |  |
| --- | --- |
| Query Type | Cost |
| UNION | Rows Examined: 18408 rows |
| UNION ALL | Rows Examined: 9204 rows |

Fig.2. Comparing Cost for execution of query with UNION and UNION ALL.

1. Table order in Join Clause

In INNER JOIN the order of table in query does not make difference in cost because inner join is optimized automatically by MYSQL query optimizer engine.

We performed three different inner join queries with different table order.

/\*INNER JOIN QUERY\*/

Select \* from parking.day d

inner join parking.location dl on d.row\_id = dl.row\_id

inner join parking.violation dv on d.row\_id = dv.row\_id

where d.row\_id = 8888677;

Select \* from parking.location dl

inner join parking.day d on dl.row\_id = d.row\_id

inner join parking.violation dv on dl.row\_id = dv.row\_id

where dl.row\_id = 8888677;

Select \* from parking.day d

inner join parking.violation dv on d.row\_id = dv.row\_id

inner join parking.location dl on d.row\_id = dl.row\_id

where d.row\_id = 8888677;

|  |  |
| --- | --- |
| Query Type | Cost |
| INNER JOIN | Execution Cost: 0.35 for all three queries. |

Fig.3.Cost for execution of queries of INNER JOIN.

/\*OUTER JOIN QUERY\*/

Select \* from parking.day d

left join parking.location dl on d.row\_id = dl.row\_id

left join parking.violation dv on d.row\_id = dv.row\_id

where d.row\_id = 8888677;

Select \* from parking.location dl

left join parking.day d on dl.row\_id = d.row\_id

left join parking.violation dv on dl.row\_id = dv.row\_id

where dl.row\_id = 8888677;

Select \* from parking.day d

left join parking.violation dv on d.row\_id = dv.row\_id

left join parking.location dl on d.row\_id = dl.row\_id

where d.row\_id = 8888677;

|  |  |
| --- | --- |
| Query Type | Cost |
| OUTER JOIN | Execution Cost: 0.70, 1.05 and 0.35 for queries top to bottom respectively. |

Fig.4. Cost for execution of queries of OUTER JOIN.

1. NOT IN and EQUALS TO

We are using NOT IN and EQUALS TO for the table parking.day where we have column day\_of\_week and it has all the days of week.

We performed queries to get only the value of Sunday on our table using both queries.

/\*NOT IN QUERY\*/

select day\_of\_week from parking.day where day\_of\_week NOT IN ('MONDAY' ,'TUESDAY',' WEDNESDAY',' THURSDAY', 'FRIDAY', 'SATURDAY' );

/\*EQUALS TO QUERY\*/

select day\_of\_week from parking.day where day\_of\_week = 'SUNDAY';

|  |  |
| --- | --- |
| Query Type | Cost |
| NOT IN | Execution Cost: 85.56 & Rows Examined: 406 rows. |
| EQUALS TO (=) | Execution Cost: 44.30 & Rows Examined: 400 rows |

Fig.5. Comparing Cost for execution of query with NOT IN and EQUALS TO.

1. Optimizing paging with LIMIT Clause

It is not a good idea to retrieve all the data for application and performing paging after that. It is better to only retrieve the rows which we need.

We can optimize paging with LIMIT Clause.

/\*LIMIT QUERY\*/

select \* from parking.violation

order by violation\_code

Limit 2060;

/\*LIMIT AND OFFSET QUERY\*/

select \* from parking.violation

order by violation\_code

Limit 2060,9;

|  |  |
| --- | --- |
| Query Type | Cost |
| LIMIT | Rows Sent to Client: 2060 |
| LIMIT and OFFSET | Rows Sent to Client: 9 |

Fig.6.Comparing Cost for execution of query with LIMIT and OFFSET.

1. *Conclusion*

Query optimization is required to tune the overall performance of the database. The project can be used as quick and easy guide to write queries. But it is always great idea to test query with real data on development server before deploying it on production because the optimization of query depends on database structure and environment in which the query is run. Query optimization has huge effect on performance of database due to which it should not be ignored

*Reference*

1. <https://en.wikipedia.org/wiki/Query_optimization>
2. <https://www.kaggle.com/arcgisopendata/dc-parking-violations>
3. <https://www.oreilly.com/library/view/highperformance-mysql/9780596101718/>
4. Database System Concepts sixth edition, Abraham Silberschatz, Henry F. Korth, S. Sudarshan, ISBN 978-0-07-352332-