Queries - EXPLAIN SELECT

MySQL provides a critical performance-tuning tool in the form of the EXPLAIN SELECT command. As a general rule, an application should never be deployed without running its queries through this utility to verify that they are executing as expected. This tool specifically tells:

* How queries are using (or are failing to use) indexes
* The order in which tables are being joined

It shows exactly how MySQL is executing the query and gives clues about how the query performance can be improved.

For example, consider a database with a State table and a query to retrieve the state\_name based on the code, state\_cd.

mysql> SELECT state\_name FROM State WHERE state\_cd = 'CA';

+------------+

| state\_name |

+------------+

| California |

+------------+

1 row in set (0.00 sec)

To use EXPLAIN SELECT, simply prepend the EXPLAIN keyword to the query. MySQL won't execute the query; instead, it will produce output describing the plan for executing the query. For example:

mysql> EXPLAIN SELECT state\_name FROM State WHERE state\_cd = 'CA';

+-------+------+---------------+------+---------+------+------+------------+

| table | type | possible\_keys | key | key\_len | ref | rows | Extra |

+-------+------+---------------+------+---------+------+------+------------+

| State | ALL | NULL | NULL | NULL | NULL | 50 | where used |

+-------+------+---------------+------+---------+------+------+------------+

1 row in set (0.00 sec)

This output is simply a set of steps to be performed in order during the execution phase. Below is a list of the columns returned by EXPLAIN SELECT and what they mean:

**table**

The table to which the row of output refers. In queries with multiple tables, EXPLAIN SELECT will return a row for each table.

**type**

The join type. Possible join types are listed below, ranked fastest to slowest:

* **system** - The table is a system table with only one row. This is a special case of the const join type.
* **cost** - The table has at most one matching row, can be read once, and is treated as a constant for the remainder of query optimization. A const query is fast because the table is read only once.
* **eq\_ref** - No more than one row will be read from this table for each combination of rows from previous tables. This type is used when all columns of an index are used in the query, and the index is UNIQUE or a PRIMARY KEY.
* **ref** - All matching rows will be read from this table for each combination of rows from previous tables. This is used when an index is neither UNIQUE nor a PRIMARY KEY, or if a left subset of index columns is used in the query.
* **range** - Only rows in a given range will be retrieved from this table, using an index to select the rows.
* **index** - A full scan of the index will be performed for each combination of rows from previous tables. This is the same as an ALL join type except only the index is scanned.
* **ALL** - A full scan of the table will be performed for each combination of rows from previous tables. ALL joins should be avoided by adding an index.

**possible\_keys**

possible\_keys lists which indexes MySQL could use to find the rows in this table. When there are no relevant indexes, possible\_keys is NULL. This indicates that you can improve the performance of your query by adding an index.

**key**

key lists the actual index that MySQL chose. It is NULL if no index was chosen.

**key\_len**

key\_len lists the length, in bytes, of the index that MySQL chose. This can be used to determine how many parts of a multicolumn index MySQL chose to use.

**ref**

ref lists which columns or constants are used to select rows from this table.

**rows**

rows lists the number of rows that MySQL thinks it will have to examine from this table to execute the query.

**Extra**

Extra lists more information about how a query is resolved. Possible values are:

* **distinct** - After MySQL has found the first matching row, it will stop searching in this table.
* **not exists** - MySQL was able to do a left join optimization of the query.
* **range checked for each record (index map: #)** - MySQL was not able to identify a suitable index to use. For each combination of rows from the previous tables, it will look for an index to use. This is not ideal, but should be faster than using no index at all.
* **using filesort** - MySQL has to sort the rows before retrieving the data.
* **using index** - All needed information is available in the index, so MySQL doesn't need to read any data from the table.
* **using temporary** - MySQL has to create a temporary table to resolve the query. This occurs if you use ORDER BY and GROUP BY on different sets of columns.
* **where used** - The WHERE clause will be used to restrict the rows returned from this table.

A detailed example will help illustrate how to use EXPLAIN SELECT to optimize a query. Even though SELECT queries are referred to here, these guidelines apply to UPDATE and DELETE statements as well. INSERT statements do not need to be optimized unless they are INSERT...SELECT statements.

For this example, a State table is used, which includes data about all 50 U.S. states.

mysql> DESCRIBE State;

+------------+----------+------+-----+---------+-------+

| Field | Type | Null | Key | Default | Extra |

+------------+----------+------+-----+---------+-------+

| state\_id | int(11) | | | 0 | |

| state\_cd | char(2) | | | | |

| state\_name | char(30) | | | | |

+------------+----------+------+-----+---------+-------+

3 rows in set (0.00 sec)

To get the name for the state of California (the state matching the code CA):

SELECT state\_name FROM State WHERE state\_cd = 'CA';

Running EXPLAIN SELECT will explain how the query will be executed:

mysql> EXPLAIN SELECT state\_name FROM State where state\_cd = 'CA';

+-------+------+---------------+------+---------+------+------+------------+

| table | type | possible\_keys | key | key\_len | ref | rows | Extra |

+-------+------+---------------+------+---------+------+------+------------+

| State | ALL | NULL | NULL | NULL | NULL | 50 | where used |

+-------+------+---------------+------+---------+------+------+------------+

1 row in set (0.00 sec)

The join type ALL tells that MySQL will scan all rows in the State table to satisfy the query. In other words, MySQL will read each of the rows in the table and compare it to the WHERE clause criteria (state\_cd = 'CA'). The rows column explains that MySQL estimates it will have to read 50 rows to satisfy the query, which is expected since there are 50 states.

This performance can be improved. Since state\_cd is being used in a WHERE clause, put an index on it and rerun the EXPLAIN SELECT to check its impact on performance:

mysql> CREATE INDEX st\_idx ON State ( state\_cd );

.

.

mysql> EXPLAIN SELECT state\_name FROM State WHERE state\_cd = 'CA';

+-------+------+---------------+--------+---------+-------+------+------------+

| table | type | possible\_keys | key | key\_len | ref | rows | Extra |

+-------+------+---------------+--------+---------+-------+------+------------+

| State | ref | st\_idx | st\_idx | 2 | const | 1 | where used |

+-------+------+---------------+--------+---------+-------+------+------------+

The key column indicates that MySQL has decided to use the new index. Consequently, the processing of the query has been reduced from 50 rows to one.

The index on the state\_cd column provided MySQL some more information to be used during the optimization phase. MySQL uses the st\_idx index to find the rows that match the WHERE clause criteria. Because the index is sorted, MySQL can quickly locate the matching row. Each row in the index provides a pointer back to its corresponding row in the table. Once MySQL locates the rows in the index, it knows exactly which rows to read from the table to satisfy the query.

In the first (non-indexed) case, MySQL had to read each row in the table and compare it to the criteria to find the matching row. In the second (indexed) case, MySQL exploits the sorted index to locate the matching records, then read the matching row from the table a much faster operation.