



# Bend SQL to Your Will With EXPLAIN

Ligaya Turmelle  
MySQL Support Engineer  
[ligaya@mysql.com](mailto:ligaya@mysql.com)



# MySQL Basics

- MySQL is the world's most popular open source database software, with over 100 million copies of its software downloaded or distributed throughout its history.
- a key part of LAMP (Linux, Apache, MySQL, PHP / Perl / Python), the fast-growing open source enterprise software stack.
- Site: <http://www.mysql.com/>
- Download: <http://dev.mysql.com/downloads/>
- Online Manual:  
<http://dev.mysql.com/doc/refman/5.1/en/index.html>

# Objectives

- So what are we hoping you will learn
  - > How to get EXPLAIN output
  - > How to read the output of EXPLAIN
  - > What does it mean? What can we do to speed things up?
  - > Briefly discuss indexing techniques
  - > Briefly discuss query optimization techniques

# Start at the beginning

- Syntax:
  - EXPLAIN [EXTENDED] SELECT *select\_options*
  - > Basically you just put an EXPLAIN before a SELECT statement.
- What does EXPLAIN do?
  - > displays information from the optimizer about the query execution plan.
    - EXTENDED with SHOW WARNINGS
- Works only with SELECT statements
  - > Some statements can be converted to SELECTS
    - but the statement has to “touch” all the same columns

# Output

```
mysql> EXPLAIN SELECT c.Name FROM City c WHERE  
District='Florida'\G
```

```
***** 1. row *****  
      id: 1  
select_type: SIMPLE  
      table: c  
      type: ref      key: District      ref: const  
      Extra: Using where
```

# EXPLAIN output

- Each row provides information about one table
- EXPLAIN output columns

id	key_len
select_type	ref
table	rows
type	filtered (new to 5.1)
possible_keys	Extra
key	

# id

- The SELECT identifier.
- Only if there are subqueries, derived tables or unions is this incremented
  - > Derived table is a subquery in the FROM clause that created a temporary table
- Number reflects the order that the SELECT was done in

```
mysql> EXPLAIN SELECT * FROM (SELECT * FROM COUNTRY) AS Country1 LIMIT 5;
```

id	select_type	table	type	possible_keys	key	key_len
1	PRIMARY	<derived2>	ALL	NULL	NULL	NULL
2	DERIVED	COUNTRY	ALL	NULL	NULL	NULL

# select\_type

- The type of SELECT
- Can have lots of values and each means something different
  - > SIMPLE – normal SELECT
  - > PRIMARY – outermost SELECT
  - > DERIVED - subquery in the FROM clause that is placed in a temporary table (table is derived from the subquery).
  - > UNION – second or more SELECT statement in a UNION
  - > SUBQUERY – a SELECT that is not in the FROM clause
    - UNION and SUBQUERY can have DEPENDANT if they use the outer SELECT.

# table

- Displays the name or alias of the table used
- Read it down and you see the order of the tables in a JOIN which may or may not be the same as the order given in the query.
- Derived tables will be noted and numbered in the listing

```
mysql> EXPLAIN SELECT * FROM (SELECT * FROM COUNTRY) AS Country1 LIMIT 5;
```

id	select_type	table	type	possible_keys	key	key_len
1	PRIMARY	<derived2>	ALL	NULL	NULL	NULL
2	DERIVED	COUNTRY	ALL	NULL	NULL	NULL

```
2 rows in set (0.02 sec)
```

# type

- The access type for the SELECT query
- Various methods:

best system/const

| eq\_ref

| ref

| ref\_or\_null

| index\_merge

| range

| index

worst ALL

# system/const

- Best types
- system
  - > From an in memory table
  - > Has only one row of data
- const
  - > optimizer knows that it can get at most one row from the table.
  - > Primary key or unique key

```
mysql> EXPLAIN SELECT * FROM Country WHERE Code='USA';
```

id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
1	SIMPLE	Country	const	PRIMARY	PRIMARY	3	const	1	

# eq\_ref

- An index lookup that will only return one row
- Only used if both conditions are met
  - > All parts of a key are used by the JOIN
  - > Table contains unique, non-nullable key to JOIN on

```
mysql> EXPLAIN SELECT Country.name FROM Country JOIN City ON City.CountryCode = Country.Code;
+----+-----+-----+-----+-----+-----+-----+-----+
| id | select_type | table | type | possible_keys | key | key_len | ref |
+----+-----+-----+-----+-----+-----+-----+-----+
| 1 | SIMPLE | City | ALL | NULL | NULL | NULL | NULL |
| 1 | SIMPLE | Country | eq_ref | PRIMARY | PRIMARY | 3 | world.City.Countr |
+----+-----+-----+-----+-----+-----+-----+-----+
```



# ref\_or\_null

- Again similar to eq\_ref and ref but
  - > Allows null values
  - OR
  - > Allows null conditions

```
mysql> ALTER TABLE Country ADD INDEX (IndepYear);
```

```
Query OK, 239 rows affected (0.36 sec)
```

```
Records: 239 Duplicates: 0 Warnings: 0
```

```
mysql> EXPLAIN SELECT IndepYear FROM Country WHERE IndepYear = 1905 or IndepYear IS NULL;
```

```

+----+-----+-----+-----+-----+-----+-----+-----+-----+
| id | select_type | table | type | possible_keys | key | key_len | ref | rows |
+----+-----+-----+-----+-----+-----+-----+-----+-----+
| 1 | SIMPLE | Country | ref_or_null | IndepYear | IndepYear | 3 | const | 31 |
+----+-----+-----+-----+-----+-----+-----+-----+-----+

```

```
1 row in set (0.00 sec)
```

# index\_merge

- This is the only access type that allows you to use 2 separate indexes for a table with MySQL
- There are lots of rules of when it will be used and in what conditions – intersection, union
  - > <http://dev.mysql.com/doc/refman/5.0/en/index-merge-optimization.html>

```
mysql> EXPLAIN SELECT * from City where id = 5 or district = 'Michigan';
```

id	select_type	table	type	possible_keys	key	key_len	ref
1	SIMPLE	City	index_merge	PRIMARY, District	PRIMARY, District	4, 20	NULL
9	Using union(PRIMARY, District); Using where						

# range

- Access method for a range value in the where clause  
 (<, <=, >, >=, LIKE, IN or BETWEEN)
  - > With LIKE you can use the range access only if the first character is not a wild card character
  - > Not possible with hash indexes (MEMORY or NDB tables)

```
mysql> ALTER TABLE City ADD INDEX (population);
Query OK, 4079 rows affected (0.10 sec)
Records: 4079 Duplicates: 0 Warnings: 0
mysql> EXPLAIN SELECT * FROM City WHERE population>10000000;
```

id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
1	SIMPLE	City	range	Population	Population	4	NULL	2	Using where

# index

- Doing an index scan
- Seen when
  - > Getting data is slow...
  - > The data being sought is available through the index
- Different then “Using index” in Extra

```
mysql> EXPLAIN SELECT id FROM City;
```

id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
1	SIMPLE	City	index	NULL	PRIMARY	4	NULL	4079	Using index

# ALL

- Full table scan
- SLOW

```
mysql> ALTER TABLE City DROP INDEX Population;
```

```
Query OK, 4079 rows affected (0.06 sec)
```

```
Records: 4079 Duplicates: 0 Warnings: 0
```

```
mysql> EXPLAIN SELECT * FROM City WHERE Population > 10000000;
```

```

+----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| id | select_type | table | type | possible_keys | key | key_len | ref | rows | Extra |
+----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| 1 | SIMPLE | City | ALL | NULL | NULL | NULL | NULL | 4079 | Using where |
+----+-----+-----+-----+-----+-----+-----+-----+-----+-----+

```

# possible\_keys

- Provides a list of the available indexes or NULL that the optimizer considered for the query
- Gotcha
  - > If you have tables/column with different character sets, the indexes on the tables/column may not be available for lookup since it has to convert from one character set to another.

# key

- The actual index that was used for the query or NULL
- key and possible\_keys are where you start looking at your indexing strategy
  - > We will briefly discuss optimizing your indexes a bit later in the talk.

```
mysql> EXPLAIN SELECT id FROM City;
```

id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
1	SIMPLE	City	index	NULL	PRIMARY	4	NULL	4079	Using index

# key\_len

- Shows the number of **bytes** MySQL will use from the index
- You can use this to see if the entire index or part of an index is being used for the query.
- Keep in mind that some character sets can use more than one byte per character. (Ex: UTF8 can use up to 3 bytes per character)

```
mysql> describe City;
```

Field	Type	Null	Key	Default	Extra
ID	int(11)	NO	PRI	NULL	auto_increment

← INT is 4 bytes

# ref

- Very different from the access type “ref”
- Show which columns or constants within the index that will be used to access the data

# ROWS

- Number of rows MySQL **expects** to find based on statistics
  - > Can be very different then what really shows up
- You can update these statistics with `ANALYZE TABLE` but remember that each storage engine updates it's statistics with varying levels of accuracy.
  - > Ex: InnoDB statistics are based on 10 random dives rather then on all the actual data

# filtered

- New – since 5.1.12
- indicates an estimated percentage of table rows that will be filtered by the table condition.
- rows shows the estimated number of rows examined
  - >  $\text{rows} \times \text{filtered} / 100$  shows the number of rows that will be joined with previous tables.
- displayed if you use EXPLAIN EXTENDED.

# Extra

- This is where all additional information is located
- Comments you need know about
  - > “Using index” - you are using a covering index (getting data from the index rather than the table)
  - > “Using filesort” - manual sorting was done rather than using an index for the sort
  - > “Using temporary” - at some point in the execution a temporary table was made. If necessary you want to keep it in RAM (tmp\_table\_size and max\_heap\_table\_size)
  - > “Using where” - filtering outside the storage engine.

# Indexing

- Know your data and queries
  - > You always have to keep the big picture in mind
  - > Data changes and the right index now – may not be the right index later
- Understand how indexes work in MySQL
  - > Leftmost prefix
  - > More indexes is not necessarily better
  - > MyISAM vs. InnoDB
  - > ANALYZE TABLE

# Indexing (con't)

- Shoot for the best EXPLAIN plan
  - > Use indexes for sorting
  - > Covering indexes
- Index hinting is possible though highly discouraged
  - > If you know **absolutely** that you should be using a specific index  
\_and\_ ANALYZE TABLE is not helping
  - > Requires that you maintain that query over time since data changes (can't set it and forget it)
  - > <http://dev.mysql.com/doc/refman/5.1/en/index-hints.html>
  - > also look into STRAIGHT\_JOIN

# Query Optimizing

- Use the slow query log
  - > You can set a maximum time for a query to run and if it goes over, log it (`long_query_time`)
    - In 5.1 micro time is available
  - > You can tell it to log all queries that do not use indexes (`log_queries_not_using_indexes`)
  - > <http://dev.mysql.com/doc/refman/5.0/en/slow-query-log.html>
- Don't be afraid of sub-selects, but use them wisely
  - > Can you write it as a JOIN and if so – does it run faster then the subselect?

# Query Optimizing (con't)

- Be willing to consider alternate methods of doing things
  - > Instead of one big query with lots of derived tables, use a stored procedure that builds intermittent MEMORY tables that you index and work with
- Use Query Profiling to get the nitty gritty
  - > Available from 5.0
  - > There is a **lot** of information available using profiling
  - > <http://dev.mysql.com/tech-resources/articles/using-new-query-profiler.html>

# Questions?



# Performance Tuning the MySQL Server

Ligaya Turmelle  
MySQL Support Engineer  
[ligaya@mysql.com](mailto:ligaya@mysql.com)

