Your Source For Technical Trends, Tips, And Best Practices From Pythian Experts

Subscribe

# MySQL Query Best Practices

*by Edwin Wang* | April 25, 2016

*Posted in:* *MySQL*, *Technical Track*

*Tags:* *MySQL*

Using MySQL Workbench to Connect Through ProxySQL
150 views
While Upgrading from GI 12c to 19c, +asm1(19c) Cannot Detect +asm2(12c)
136 views
Still have binlog_error_action as IGNORE_ERROR?
123 views
Datascape Podcast Episode 41 - Choosing a Public Cloud for Oracle Database
105 views

All-Time Highest Viewed Posts

Build an E-Business Suite 12.1.3 Sandbox In VirtualBox in One Hour
253,302 views
DUPLICATE from ACTIVE Database Using RMAN, a Step-by-Step Guide
202,958 views
Quick Install Guide for Oracle 10g Release 2 on Mac OS X Leopard & Snow Leopard
199,921 views
How to Install Oracle 12c RAC: A Step-by-Step Guide
179,919 views
Step-by-Step Installation of an EBS 12.2 Vision Instance
177,060 views

You can get many returns from a Google search for “MySQL Query Best Practices” or “MySQL Query Optimization.” The drawback is that too many rules can provide confusing or even conflicting advice. After doing some research and tests, I outlined the essential and important ones below:

**1) Use proper data types**

**1.1) Use the smallest data types if possible**

MySQL tries to load as much data as possible into memory (innodb-buffer-pool, key-buffer), so a small data type means more rows of data in memory, thus improving performance. Also, small data sizes reduces disk i/o.

**1.2) Use Fixed-length Data Types if Possible**

MySQL can calculate quickly the position of a fixed-length column in a specific row of a table.

disk space required.

In practice, if the column data size varies a lot, then use a flexible-length data type (e.g., varchar); if the data length is short or length barely changes, then use a fixed data type.

**1.3) Use not null unless there is reason not to**

It is harder for MySQL to optimize queries that refer to nullable columns, because they make indexes, index statistics, and value comparisons more complicated. A nullable column uses more storage space and requires special processing inside MySQL.

When a nullable column is indexed, it requires an extra byte per entry and can even cause a fixed-size index (e.g., an index on a single integer column) to be converted to a variable-sized one in MyISAM.

**2)Use indexes smartly**

**2.1) Make primary keys short and on meaningful fields**

A shorter primary key will benefit your queries, because the smaller your primary key, the smaller the index, and the less pages in the cache. In addition, a numeric type is prefered because numeric types are stored in a much more compact format than character formats and so it will make primary key shorter.

Another reason to make primary key shorter, is because we usually use primary key to join with the other tables.

It is a good idea to use a primary key on a meaningful field, because MySQL uses a cluster index on a primary key. We usually just need the info from primary key, and especially when joined with other fields wouldn't change, otherwise it might affect all the tables using this as foreign key when you have to change the primary key.

**2.2) Index on the search fields only when needed**

Usually we add indexes on the fields that frequently show up in a where clause — that is the purpose of indexing. But while an index will benefit reads, it can make writes slower (inserting/updating), so index only when you need it and index smartly.

**2.3) Index and use the same data types for join fields**

MySQL can do joins on different data types, but the performance is poor as it has to convert from one type to the other for each row. Use the same data type for join fields when possible.

**2.4) Use a composite index if your query has has more than one field in the where clause**

When the query needs to search on multiple columns of a table, it might be a good idea to create a compound index for those columns. This is because with composite index on multiple columns, the search will be able to narrow down the result set by the first column, then the second, and so on.

Please note that the order of the columns in the composite index affects the performance, so put the columns in the order of the efficiency of narrowing down the search.

**2.5) Covering index for most commonly used fields in results**

In some cases, we can put all the required fields into an index (i.e., a covering index) with only some of the fields in the index used for searching and the others for data only. This way, MySQL only need to access the index and there is no need to search in another table.

the string already covers most of the unique values, it is good to just index the prefix part.

**2.7) Avoid over-indexing**

Don't index on the low cardinality values, MySQL will choose a full table scan instead of use index if it has to scan the index more than 30%.

If a field already exists in the first field of a composite index, you may not need an extra index on the single field. If it exists in a composite index but not in the leftmost field, you will usually need a separate index for that field only if required.

Bear in mind that indexing will benefit in reading data but there can be a cost for writing (inserting/updating), so index only when you need it and index smartly.

## 3) Others

### 3.1) Avoid SELECT *

There are many reasons to avoid select * from… queries. First, it can waste time to read all the fields if you don't need all the columns. Second, even if you do need all columns, it is better to list the all the field names, to make the query more readable. Finally, if you alter the table by adding/removing some fields, and your application uses select * queries, you can get unexpected results.

### 3.2) Prepared Statements

Prepared Statements will filter the variables you bind to them by default, which is great for protecting your application against SQL injection attacks.

When the same query is being used multiple times in your application, you can assign different values to the same prepared statement, yet MySQL will only have to parse it once.

select count from a table, since the first method will return a result once it gets one row of the required data, while the second one will have to count on the whole table/index.

**3.4) Use select limit [number]**

Select… limit [number] will return the only required lines of rows of data. Including the limit keyword in your SQL queries can have performance improvements.

**3.5) Be careful with persistent connections**

Persistent connections can reduce the overhead of re-creating connections to MySQL. When a persistent connection is created, it will stay open even after the script finishes running. The drawback is that it might run out of connections if there are too many connections remaining open but in sleep status.

**3.6) Review your data and queries regularly**

MySQL will choose the query plan based on the statistics of the data in the tables. When the data size changes, the query plan might change, and so it is important to check your queries regularly and to make optimizations accordingly. Check regularly by:

3.6.1) EXPLAIN your queries

3.6.2) Get suggestions with PROCEDURE ANALYSE()

3.6.3) Review slow queries

**Interested in working with Edwin? Schedule a tech call.**

1 Comment. Leave New

**AT** February 25, 2019 5:30 am

PROCEDURE ANALYSE() is deprecated as of MySQL 5.7.18, and is removed in MySQL 8.0.

Reply

## Leave A Reply

Your email address will not be published. Required fields are marked *

Comment

Name *

Email *

Website

POST COMMENT

ABOUT CAREERS CLIENTS RESOURCES BLOG CONTACT

About Pythian
Our Experts
Leadership Team
Board of Directors
Partners
Events
Newsroom

Data & Analytics
Enterprise Data Platform
Big Data
Data Warehouses
Advanced Analytics
Custom Analytics Services
Blockchain

Cloud Services
Cloud Strategy
Migrate to Cloud
Managed Cloud
DevOps
Kubernetes

IT Infrastructure
Remote DBA Services
Database Managed Services
Managed Infrastructure

Technologies
Amazon Web Services
Google Cloud Platform
Microsoft Azure
Oracle
Oracle EBS
SQL Server
Hadoop
Cassandra
MySQL
MongoDB