Guide To Using mysqlidxchk

The purpose of mysqlidxchk is to check MySQL databases and tables for indexes which are not being used. MySQL does not currently have an intrinsic method or metric to track which indexes are or are not used. Therefore, a script like mysqlidxchk is needed to discover this information easily.

mysqlidxchk can be used in the scope of an entire schema, or in the scope of a select number of databases. Either way, mysqlidxchk requires one or more MySQL slow or general logs (or a "raw" log) in order to compare the indexes used by the queries in the logs to all available indexes in the given scope. From this comparison, mysqlidxchk reports which indexes are not used (and, with the **--show-used command line option**, which indexes are used).

Therefore, the extent to which mysqlidxchk can accurately report which indexes are not used is directly related to the extent to which the queries in the given logs are representative of how the databases are used.

In this guide we will consider a number of topics. First, we'll consider some practical uses for mysqlidxchk. Then, we'll look more closely at how mysqlidxchk works. Finally, I will present a short demonstration of mysqlidxchk in action.

Practical Uses For mysqlidxchk

The idea for mysqlidxchk was generously shared with me by Peter Zaitsev, who was previously employed as Manager of High Performance Group at MySQL AB but is now an independent **MySQL consultant**. Hence, mysqlidxchk's most obvious practical use is for MySQL consultants who wish to streamline a database schema. Among a multitude of other tasks, a consultant will eventually want to remove any unnecessary and unused indexes from the database schema. With a general log and the --databases command line option, a consultant can limit mysqlidxchk to only the databases that they are interested in.

Developers can use mysqlidxchk to check for unused indexes—indexes which they had created earlier but have since abandoned or changed. In the haste of development sometimes we (or at least I) forget about those "other" indexes. As developers, we already know the queries which are ran against the database, therefore we don't really need a MySQL general or slow log. Instead, we can write our own SQL statements to a "raw" log and use mysqlidxchk with that to check for forgotten indexes. A "raw" log is simply a text file that contains semi-colon new-line (;\n) terminated MySQL statements.

In certain cases, system/database administrators can also use mysqlidxchk to check for unused indexes. Administrators would probably use a combination of slow and general logs and check every database in the entire schema. For MySQL servers with many databases and tables, this would be practically impossible by hand, but mysqlidxchk can do it in a few seconds.

Finally, hackers can use mysqlidxchk to expose the unused indexes of some third-party application. For example, I often times talk with people who are fixing and optimizing the database schema and queries of a third-party app. With mysqlidxchk these hackers can see what extraneous indexes the third-party app has left lurking in the schema and remove them.

How mysqlidxchk Works

At over 900 lines, mysqlidxchk is not a small script, yet it seems to do a relatively simple task: find unused indexes. After I began writing mysqlidxchk, I realized that it had to do a lot "background" work to insure that it was truly helpful. Otherwise, due to the lack of certain features and functionalities in MySQL, mysqlidxchk would fail far too easily.

Log Parsing & USE

The first and most important task is log parsing. The log parsing routines in mysqlidxchk were taken from mysqlsla v1.4 and modified. The first problem to overcome in log parsing was retaining **USE statements** for each query. As we already know, you cannot **EXPLAIN** a query until you've selected the appropriate database.

For **general logs** it is easy to find and retain the USE statements for every query because every connection either connects with a database specified or issues an "Init DB" command. Since all connections are uniquely numbered, mysqlidxchk keeps track of every connection's database usage and saves that information with the connection's queries. In short: for general logs mysqlidxchk always knows which queries use which databases.

Slow logs are a different. Slow logs do not always contain USE statements and they do not contain any other information that would allow mysqlidxchk to easily discover a query's database. Therefore, for slow logs mysqlidxchk must often rely on "database discovery" which is discussed below. Alternatively, one could manually add USE statements to a slow log if necessary.

The final kind of "log" file that mysqlidxchk can parse is what I call a raw log. A raw log is simply a text file containing SQL statements that are semi-colon new-line terminated. A single SQL statements can span multiple lines, but it must end with a ";\n". In other words, the next SQL statement has to start on the next line down. For raw logs, one must add the appropriate USE statements *before* the queries that are to be used with that database. For example, here is a short raw log:

```
USE db_foo;
SELECT * FROM table WHERE col = 1;
USE db_bar;
UPDATE table SET col = 'hello' WHERE col = 'goodbye';
```

In this example, the SELECT statement is used with database db_foo and the UPDATE is used with database db_bar. A USE statement remains in effect for all queries below it until another USE statement is found.

Database Discovery

In cases where mysqlidxchk has queries with no database (no corresponding USE statement), it will try to discover which database the query belongs to. This process of database discovery is actually quite successful, except in one case. First, lets look at how mysqlidxchk does database discovery.

One of the first things that mysqlidxchk does when ran is read the entire **information schema** (a.k.a. metadata, data dictionary, system catalog). Actually, the current version of mysqlidxchk does not read

MySQL's INFORMATION_SCHEMA database; it uses **SHOW statements** to discover every index in every table in every database (unless the --databases option is used, then it only discovers the given databases).

Knowing every table in every database allows mysqlidxchk to find a query's database by comparing the tables that the query uses to the tables in each database. For example, suppose you have a database called "foo" with tables "t1, t2, t3". Now suppose mysqlidxchk encounters a query: "SELECT * FROM t1, t2 WHERE t1.col = 'happy';". mysqlidxchk will see that every table in the query is found in database foo. Therefore, it assigns the query to USE foo.

As previously mentioned there is one case in which database discovery fails. You can probably already guess it. The case is when there are two or more databases with identically named tables. A case like this can happen on a shared server with, for example, many different installations of **WordPress**. In such a case, the result of mysqlidxchk's database discovery is undefined. mysqlidxchk will probably pick the first matching database that it finds. There is no clean solution to this problem that I am aware of.

In general though, database discovery works well. It even works to find "missing" databases. If mysqlidxchk has a query which uses a nonexistent database, it will employ database discovery to find the query's missing database. Conversely, if mysqlidxchk cannot find the missing database, it will discard all queries which use the nonexistent database.

See the **mysqlidxchk Documentation** for two command line options related to database discovery: --no-db-discovery and --no-discovery-report.

UPDATE Statements

Unless the --ignore-update command line is used, mysqlidxchk converts UPDATE statements into SELECT statements in order to EXPLAIN them. The method by which mysqlidxchk does this is simple: it converts "UPDATE table_reference SET column_reference WHERE etc." to "SELECT * FROM table_reference WHERE etc." Database discovery applies to UPDATE statements, too.

Showing Used Index

The --show-used option causes mysqlidxchk to also show which indexes are used. At present, that is about all it does except for one more thing. The indexes that are marked as used will have a number in parenthesis beside them. This number is the number of *unique* (i.e. abstracted) queries which use that index. I am open to **suggestions** about what else you would like mysqlidxchk to say about used indexes?

A Short Demonstration of mysqlidxchk

Here follows two mock databases and some queries I created to demonstrate mysqlidxchk in action. First we have the databases: hacking and hacking2. hacking contains tables a, b, c. hacking2 contains tables x, y, z. The first two tables of each database (a, b and x, y) contain one column which is indexed. The last tables contain three columns which are indexed in various ways. Here are the indexes for all tables:

```
mysql> SHOW INDEX FROM a; SHOW INDEX FROM b; SHOW INDEX FROM c;
+-------+------------+-----------+--------------+-------------+
| Table | Non_unique | Key_name  | Seq_in_index | Column_name |
+-------+------------+-----------+--------------+-------------+
| a     |          1 | idx_Ta_C1 |            1 | col_1       |
+-------+------------+-----------+--------------+-------------+

+-------+------------+-----------+--------------+-------------+
| Table | Non_unique | Key_name  | Seq_in_index | Column_name |
+-------+------------+-----------+--------------+-------------+
| b     |          1 | idx_Tb_C1 |            1 | col_1       |
+-------+------------+-----------+--------------+-------------+

+-------+------------+-------------+--------------+-------------+
| Table | Non_unique | Key_name    | Seq_in_index | Column_name |
+-------+------------+-------------+--------------+-------------+
| c     |          1 | idx_Tc_C1   |            1 | col_1       |
| c     |          1 | idx_Tc_C2   |            1 | col_2       |
| c     |          1 | idx_Tc_C1_2 |            1 | col_1       |
| c     |          1 | idx_Tc_C1_2 |            2 | col_2       |
| c     |          1 | idx_Tc_C2_1 |            1 | col_2       |
| c     |          1 | idx_Tc_C2_1 |            2 | col_1       |
+-------+------------+-------------+--------------+-------------+

mysql> SHOW INDEX FROM x; SHOW INDEX FROM y; SHOW INDEX FROM z;
+-------+------------+-----------+--------------+-------------+
| Table | Non_unique | Key_name  | Seq_in_index | Column_name |
+-------+------------+-----------+--------------+-------------+
| x     |          1 | idx_Tx_C1 |            1 | col_1       |
+-------+------------+-----------+--------------+-------------+

+-------+------------+-----------+--------------+-------------+
| Table | Non_unique | Key_name  | Seq_in_index | Column_name |
+-------+------------+-----------+--------------+-------------+
| y     |          1 | idx_Ty_C1 |            1 | col_1       |
+-------+------------+-----------+--------------+-------------+

+-------+------------+-------------+--------------+-------------+
| Table | Non_unique | Key_name    | Seq_in_index | Column_name |
+-------+------------+-------------+--------------+-------------+
| z     |          1 | idx_Tz_C3_2 |            1 | col_3       |
| z     |          1 | idx_Tz_C3_2 |            2 | col_2       |
| z     |          1 | idx_Tz_C1_2 |            1 | col_1       |
| z     |          1 | idx_Tz_C1_2 |            2 | col_2       |
| z     |          1 | idx_Tz_C1   |            1 | col_1       |
+-------+------------+-------------+--------------+-------------+
```

Now, here are some queries from a raw log for these databases and tables:

```
USE hacking;
SELECT * FROM c;
SELECT * FROM a AS apple, b AS banana
   WHERE apple.col_1 = banana.col_1;

USE hacking2;
SELECT * FROM x, y, z LIMIT 2;
SELECT z.col_3 FROM x, z WHERE z.col_1 = x.col_1;
SELECT bar.col_3 FROM x foo, z bar WHERE bar.col_1 = foo.col_1;
SELECT x.col_1, y.col_1, z.* FROM x, y, z
   WHERE x.col_1 = y.col_1 and x.col_1 = z.col_1;

USE hacking;
SELECT cat.*, a.*, boy.*
FROM
   c AS cat,
   b boy,
   a
WHERE
   a.col_1 = cat.col_2 AND cat.col_1 NOT IN (2,4, 6, 8);

USE hacking;
UPDATE a SET col_1 = 2 WHERE col_1 IN (2, 4, 6);
UPDATE b AS boy SET col_1 = NULL WHERE 1;

USE hacking2;
UPDATE LOW_PRIORITY z zoot SET col_1 + 1, col_2 = 3 WHERE col_3 IS NOT NULL LIMIT 2;
UPDATE IGNORE x, y AS yak, z zar SET col_1 = 0 WHERE zar.col_1 IS NULL;
```

After inserting some random data into the tables I ran mysqlidxchk:

```
# ./mysqlidxchk --raw test_raw --databases hacking,hacking2 --show-used
Reading raw log 'test_raw'.
11 total valid queries, 11 unique.
Only using databases: hacking hacking2

Database: hacking
        Table: a
                Index: idx_Ta_C1            used (3)
        Table: b
                Index: idx_Tb_C1            used (3)
        Table: c
                Index: idx_Tc_C1            NOT used
                Index: idx_Tc_C1_2          NOT used
                Index: idx_Tc_C2            NOT used
                Index: idx_Tc_C2_1          NOT used
Database: hacking2
        Table: foo
                Index: idx_Tx_C1            used (1)
        Table: x
                Index: idx_Tx_C1            used (4)
        Table: y
                Index: idx_Ty_C1            used (3)
        Table: z
                Index: idx_Tz_C1            NOT used
                Index: idx_Tz_C1_2          used (1)
                Index: idx_Tz_C3_2          used (1)
```

And that is all there is to it. If this were a real database, I would probably drop all those unused indexes.

For now mysqlidxchk is very simple with just one report. However, its one report is quite helpful in streamlining one more crucial part of your databases and tables: indexes.

(Doc rev: Mar 31 2007)

Last updated: 2009-09-30