Database Performance-Tuning Concepts

Database Performance-Tuning Concepts:

- Goal: execute queries as fast as possible
- Procedures designed to reduce response time of database system
- Database performance begins with good database design!

1) Identify performance bottlenecks (other than DBMS):

- Memory
- CPU
- I/O (throughput): hard disk, network and OS

2) DBMS Sources of Problems:

- Poor database design
- Bad SQL Coding Practices
- Poor or nonexistent indexes
- Server variables not tuned properly (more advanced topic, via DBA--Modifying server variables can be done by modifying the configuration files, or, for some variables, issuing a SET command.)
- Hardware and/or network bottlenecks (use system/network log files)

Schema Guidelines:

- Inefficient schema another way to kill performance
- Use smallest data types necessary (e.g., Do you really need that BIGINT?)
- Normalize **first**, denormalize, though **only** in **special** cases

Indexing Guidelines:

- Create indexes on columns that are expected to be searched. For example,
  - Last name
  - SSN
- Create indexes for attributes used in:
  - select or join criteria
  - where
  - having
  - order by, or group by clauses
  - aggregate functions (e.g., max(), min(), avg(), etc.)
  - built-in functions (e.g., concat(), format(), etc.)
- Declare pks and fks (automatically indexed), so optimizer can use indexes (in joins)
- Declare indexes in join columns other than pks or fks (when used)
- Do **NOT** index small tables or tables w/low sparsity (attributes w/few values)
- **Avoid** indexing the following types of attributes:
  - frequently updated
  - will retrieve 25% or more of the records from a table
  - contain long character strings

Indexing Issues:

- Poor or missing index slows down system
- Ensure good selectivity on field (what is being searched most frequently)
- Look for covering index opportunities (other fields that may be indexed)
- On multi-column indexes, pay attention to order of fields in index (example ahead)
- Remove redundant indexes for faster write performance (e.g., uid and ssn, only need one)

Example:
INDEX `idx_Home_Owner` (`ssn` ASC) ,
INDEX `idx_Home_Agent` (`aid` ASC) ,

CONSTRAINT `fk_Home_Owner`FOREIGN KEY (ssn) REFERENCES owner (ssn) ON DELETE CASCADE ON UPDATE CASCADE,

CONSTRAINT `fk_Home_Agent`FOREIGN KEY (aid) REFERENCES agent (aid) ON DELETE CASCADE ON UPDATE CASCADE

CASCADE: Delete or update row from parent table (owner and agent) and automatically delete or update matching rows in child (home) table.
http://dev.mysql.com/doc/refman/5.1/en/innodb-foreign-key-constraints.html

The INFORMATION_SCHEMA STATISTICS Table: provides information about table indexes.
http://dev.mysql.com/doc/refman/5.0/en/statistics-table.html

Syntax:
```
SELECT * FROM INFORMATION_SCHEMA.STATISTICS
  WHERE table_name = 'tbl_name'
  AND table_schema = 'db_name';
```

Or…

```
SHOW INDEX
  FROM tbl_name
  FROM db_name;
```

Example:
```
SELECT * FROM INFORMATION_SCHEMA.STATISTICS
  WHERE table_name = 'movies'
  AND table_schema = 'dvd_collection';
```

Common Index Problem:
```
CREATE TABLE tag (
tag_id INT NOT NULL AUTO_INCREMENT
, tag_text VARCHAR(50) NOT NULL
, PRIMARY KEY (tag_id)
) ENGINE=MyISAM;

CREATE TABLE product (
product_id INT NOT NULL AUTO_INCREMENT
, name VARCHAR(100) NOT NULL
// more fields...
, PRIMARY KEY (product_id)
) ENGINE=MyISAM;

CREATE TABLE product_tag (
product_id INT NOT NULL
, tag_id INT NOT NULL
, PRIMARY KEY (product_id, tag_id)
) ENGINE=MyISAM;
```

// This query uses index on product_tag

```
SELECT p.name
, COUNT(*) as tags
FROM product_tag pt
INNER JOIN product p
ON pt.product_id = p.product_id
GROUP BY p.name;
```

// This query does not because index order prohibits it

```
SELECT t.tag_text
, COUNT(*) as products
FROM product_tag pt
INNER JOIN tag t
ON pt.tag_id = t.tag_id
GROUP BY t.tag_text;
```

Solution to index problem:

```
CREATE INDEX idx_tag
ON product_tag (tag_id);
```

// or... create a covering index:

```
CREATE INDEX idx_tag_prod
ON product_tag (tag_id, product_id);
```

MySQL Optimization Profiling:

```
mysql> show processlist;
+----+---------+----------------+-------+---------+------+-------+------------------+
| Id | User    | Host           | db    | Command | Time | State | Info             |
+----+---------+----------------+-------+---------+------+-------+------------------+
|  1 | username | localhost:2234 | mysql | Query   |    0 | NULL  | show processlist |
+----+---------+----------------+-------+---------+------+-------+------------------+
```

```
mysql> show status;
+-----------------------------------+------------+
| Variable_name                     | Value      |
+-----------------------------------+------------+
| Aborted_clients                   | 0          |
| Aborted_connects                  | 0          |
| Binlog_cache_disk_use             | 0          |
| Binlog_cache_use                  | 0          |
| Bytes_received                    | 390        |
| Bytes_sent                        | 17164      |
...
```

EXPLAIN statement:

- synonym for DESCRIBE
- or as a way to obtain information about how MySQL executes a SELECT statement:

```
mysql> explain select user from user where host='localhost';
+----+-------------+-------+------+---------------+---------+---------+-------+------+--------------------------+
| id | select type | table | type | possible keys | key     | key len | ref   | rows | Extra                    |
+----+-------------+-------+------+---------------+---------+---------+-------+------+--------------------------+
|  1 | SIMPLE      | user  | ref  | PRIMARY       | PRIMARY | 180     | const |    1 | Using where; Using index |
+----+-------------+-------+------+---------------+---------+---------+-------+------+--------------------------+
```

Optimizing Queries with EXPLAIN:

When EXPLAIN is used with an explainable statement, MySQL displays information from the optimizer about the statement execution plan. That is, MySQL explains how it would process the statement, including information about how tables are joined and in which order. For information about using EXPLAIN to obtain execution plan information, see Section 8.8.2, "EXPLAIN Output Format".

https://dev.mysql.com/doc/refman/5.7/en/using-explain.html

The EXPLAIN statement provides information about how MySQL executes statements. EXPLAIN works with SELECT, DELETE, INSERT, REPLACE, and UPDATE statements.

EXPLAIN returns a row of information for each table used in the SELECT statement. It lists the tables in the output in the order that MySQL would read them while processing the statement.

https://dev.mysql.com/doc/refman/5.7/en/explain-output.html

Table 8.1 EXPLAIN Output Columns (MySQL 5.7)

Column	JSON Name	Meaning
`id`	`select_id`	The `SELECT` identifier
`select_type`	None	The `SELECT` type
`table`	`table_name`	The table for the output row
`partitions`	`partitions`	The matching partitions
`type`	`access_type`	The join type
`possible_keys`	`possible_keys`	The possible indexes to choose
`key`	`key`	The index actually chosen
`key_len`	`key_length`	The length of the chosen key
`ref`	`ref`	The columns compared to the index
`rows`	`rows`	Estimate of rows to be examined
`filtered`	`filtered`	Percentage of rows filtered by table condition
`Extra`	None	Additional information

Horizontal partitioning use when there is a regular need to access or to isolate a subset of the "parent" table's rows. Technique may be effective to meet security, distribution, and performance optimization needs. **Note:** MySQL only supports horizontal (row) partitioning.
https://dev.mysql.com/doc/refman/5.7/en/partitioning-overview.html

http://www.vertabelo.com/blog/technical-articles/everything-you-need-to-know-about-mysql-partitions

Horizontal Partitioning Benefits:

- Main table has narrow rows:
  - More records fit into single data page
  - Fewer reads from memory/disk to get same number of records
- Less frequently queried data doesn't take up memory
- More possibilities for indexing and using different storage engines (e.g., MyISAM vs. InnoDB)

Example:

```
CREATE TABLE Users (
user_id INT NOT NULL AUTO_INCREMENT
, email VARCHAR(80) NOT NULL
, display_name VARCHAR(50) NOT NULL
, password CHAR(41) NOT NULL
, first_name VARCHAR(25) NOT NULL
, last_name VARCHAR(25) NOT NULL
, address VARCHAR(80) NOT NULL
, city VARCHAR(30) NOT NULL
, province CHAR(2) NOT NULL
, postcode CHAR(7) NOT NULL
, interests TEXT NULL
, bio TEXT NULL
, signature TEXT NULL
, skills TEXT NULL
, company TEXT NULL
, PRIMARY KEY (user_id)
, UNIQUE INDEX (email)
) ENGINE=InnoDB;
```

Horizontal partitioning:

```
CREATE TABLE Users (
user_id INT NOT NULL AUTO_INCREMENT
, email VARCHAR(80) NOT NULL
, display_name VARCHAR(50) NOT NULL
, password CHAR(41) NOT NULL
, PRIMARY KEY (user_id)
, UNIQUE INDEX (email)
) ENGINE=MyISAM;

CREATE TABLE UserExtra (
user_id INT NOT NULL
, first_name VARCHAR(25) NOT NULL
, last_name VARCHAR(25) NOT NULL
, address VARCHAR(80) NOT NULL
, city VARCHAR(30) NOT NULL
, province CHAR(2) NOT NULL
, postcode CHAR(7) NOT NULL
, interests TEXT NULL
, bio TEXT NULL
, signature TEXT NULL
, skills TEXT NULL
, company TEXT NULL
, PRIMARY KEY (user_id)
) ENGINE=InnoDB;
```

BENCHMARK(count,expr): It may be used to time how quickly MySQL processes the expression. The result value is always 0.

For example, benchmark(number of times to run, function_name);

Benchmark generating 1 million randomized md5, sha1 encrypted values
(**Note:** Intel Core i7 @2.3GHz, 8GB RAM):

```
mysql> select benchmark(1000000,sha1(md5(rand())));
+-------------------------------------+
| benchmark(1000000,sha1(md5(rand())) |
+-------------------------------------+
|                                   0 |
+-------------------------------------+
1 row in set (1.50 sec)
```

Benchmark generating 1 million randomized md5, sha2 encrypted values:

```
mysql> select benchmark(1000000,sha2(md5(rand()), 512));
+-----------------------------------------+
| benchmark(1000000,sha2(md5(rand()), 512)) |
+-----------------------------------------+
|                                       0 |
+-----------------------------------------+
1 row in set (1.89 sec)
```

Note: The **MD5** algorithm is a widely used hash function producing a 128-bit hash value. Although MD5 was initially designed to be used as a cryptographic hash function, it has been found to suffer from extensive vulnerabilities. It can still be used as a checksum to verify data integrity, but only against unintentional corruption.

SHA-1: The Secure Hash Algorithm (SHA) was developed by NIST and is specified in the Secure Hash Standard (SHS, FIPS 180). SHA-1 is a revision to this version and was published in 1994. SHA-1 produces a 160-bit (20 byte) message digest. Although slower than MD5, this larger digest size makes it stronger against brute force attacks.

MD5: MD5 was designed by Ronald Rivest in 1991 to replace an earlier hash function MD4. Its 128 bit (16 byte) message digest makes it a faster implementation than SHA-1, though, less secure.

Stress testing Web applications:

http://west-wind.com/presentations/webstress/webstress.htm

http://jakarta.apache.org/jmeter/

mysqlslap (most flexible tool for benchmarking MySQL databases)**:**

One of the interesting new tools in MySQL 5.1.4 is mysqlslap, a load emulator that lets you see how well a particular query set or table engine performs under high-load conditions.

mysqlslap is a diagnostic program designed to emulate client load for a MySQL server and to report the timing of each stage. It works as if multiple clients are accessing the server. *mysqlslap* is available as of MySQL 5.1.4.

A query that consumes too many database resources may be the result of:

- designing tables incorrectly
- choosing the wrong table type, or
- creating an inefficient query

When a query eats up a lot of database resources, it can negatively affect other application components. By using mysqlslap to stress test a server in a **non-public environment**, you will discover these errors sooner, allowing you to avoid a database meltdown once your application goes live.

Examples:
The --auto-generate-sql switch creates a table, executes an INSERT query and includes dummy data, executes a select query to retrieve the dummy data, and then drops the table.

```
mysqlslap -u username -p --port=3306 --auto-generate-sql
```

```
mysqlslap -u username -p --port=3306 --concurrency=5 --iterations=5 --create="CREATE TABLE MyTable (id int(2), name varchar(10)); INSERT INTO MyTable VALUES (12, 'John Doe');" --query="select name from MyTable where id=12;" --delimiter=";"
```

...or using file… (**Note:** when using file, remove ending semicolon in file.)

```
mysqlslap -u username -p --port=3306 --concurrency=5 --iterations=5 --create="create.sql" --query="query.sql" --delimiter=";"
```

`--concurrency`
Number of clients/connections to simulate when issuing SELECT statement
(without --concurrency, simulates single thread)

`--concurrency=1,2,4,8,100`
(run same test increasing concurrency for each run

`--create`
The file or string containing the statement to use for creating the table
(e.g., `--create="CREATE TABLE MyTable (id int(2), name varchar(10)); INSERT INTO MyTable VALUES (12, 'John Doe');"`)

Or… `--create="create.sql"`

`--delimiter`
Delimiter to use in SQL statements

`--engine=myisam`
Storage engine to use for creating table (MyISAM default)

`--iterations`
Number of times to run/repeat tests

`--number-char-cols`
Number of VARCHAR columns to use if --auto-generate-sql is specified

`--number-int-cols`
Number of INT columns to use if --auto-generate-sql is specified

`--number-of-queries`
Limit each client to approximately this number of queries

--query
File or string containg the SELECT statement to use for retrieving data
(e.g., --query="select name from MyTable where id=12;")

Or… --query="query.sql"

Examples:

```
C:\mysql\bin>mysqlslap -u username -p --port=3306 --auto-generate-sql -v
Enter password: *******
Benchmark
        Average number of seconds to run all queries: 0.235 seconds
        Minimum number of seconds to run all queries: 0.235 seconds
        Maximum number of seconds to run all queries: 0.235 seconds
        Number of clients running queries: 1
        Average number of queries per client: 0

C:\mysql\bin>mysqlslap -u username -p --port=3306 --auto-generate-sql -v --
concurrency=100
Enter password: *******
Benchmark
        Average number of seconds to run all queries: 26.250 seconds
        Minimum number of seconds to run all queries: 26.250 seconds
        Maximum number of seconds to run all queries: 26.250 seconds
        Number of clients running queries: 100
        Average number of queries per client: 0

C:\mysql\bin>mysqlslap -u username -p --port=3306 --auto-generate-sql -v --
concurrency=100 --iterations=5
Enter password: *******
Benchmark
        Average number of seconds to run all queries: 20.240 seconds
        Minimum number of seconds to run all queries: 18.391 seconds
        Maximum number of seconds to run all queries: 22.688 seconds
        Number of clients running queries: 100
        Average number of queries per client: 0

C:\mysql\bin>mysqlslap -u username -p --port=3306 --auto-generate-sql -v --
concurrency=100 --iterations=5 --number-of-queries=1000
Enter password: *******
Benchmark
        Average number of seconds to run all queries: 16.334 seconds
        Minimum number of seconds to run all queries: 16.015 seconds
        Maximum number of seconds to run all queries: 16.593 seconds
        Number of clients running queries: 100
        Average number of queries per client: 10

C:\mysql\bin>mysqlslap -u username -p --port=3306 --concurrency=5 --iterations=5 --
query=query.sql --create=create.sql  --delimiter=";"
Enter password: *******
Benchmark
        Average number of seconds to run all queries: 0.278 seconds
        Minimum number of seconds to run all queries: 0.062 seconds
        Maximum number of seconds to run all queries: 1.063 seconds
        Number of clients running queries: 5
        Average number of queries per client: 1

mysqlslap -u username -p --port=3306 --concurrency=5 --iterations=5 --query=query.sql --
create=create.sql concurrency=100 --iterations=5 --number-of-queries=1000 --delimiter=";"
```

Using larger tables:
The default behavior of mysqlslap when using the –auto-generate-sql switch is to create a two-column table with one integer column and one character column. If this isn't representative of the kind of tables you typically use, you can adjust these settings to include more integer and/or character columns, with the –number-char-cols and –number-int-cols switches:

```
mysqlslap -u username -p --port=3306 --auto-generate-sql --concurrency=100 --number-of-queries=1000 --number-char-cols=4 --number-int-cols=7

mysqlslap -u username -p --port=3306 --auto-generate-sql --concurrency=100 --number-char-cols=4

mysqlslap -u username -p --port=3306 --auto-generate-sql --concurrency=100 --number-int-cols=5
```

//comparing table engines…

```
mysqlslap -u username -p --port=3306 --auto-generate-sql -v --concurrency=100 --number-of-queries=1000 --engine=innodb
```

//roughly twice as fast…

```
mysqlslap -u username -p --port=3306 --auto-generate-sql -v --concurrency=100 --number-of-queries=1000 --engine=myisam
```

//more -vvv's more verbose (up to three)

1)

```
mysqlslap -u username -p --port=3306 --auto-generate-sql -v --concurrency=100 --number-of-queries=1000 --engine=myisam
```

2)

```
mysqlslap -u username -p --port=3306 --auto-generate-sql -vv --concurrency=100 --number-of-queries=1000 --engine=myisam
```

3)

```
mysqlslap -u username -p --port=3306 --auto-generate-sql -vvv --concurrency=100 --number-of-queries=1000 --engine=myisam
```

//saving reports

```
mysqlslap -u username -p --port=3306 --auto-generate-sql -v --auto-generate-sql --concurrency=100 --number-of-queries=1000 --engine=myisam >> output.txt

mysqlslap -u username -p --port=3306 --auto-generate-sql -v --auto-generate-sql --concurrency=100 --number-of-queries=1000 --engine=innodb >> output.txt
```

Or…

```
mysqlslap -u username -p --port=3306 --auto-generate-sql -v --auto-generate-sql --concurrency=100 --number-of-queries=1000 --engine=myisam --csv=output.csv

mysqlslap -u username -p --port=3306 --auto-generate-sql -v --auto-generate-sql --concurrency=100 --number-of-queries=1000 --engine=innodb --csv=output.csv
```

//with index on attribute "last"

```
C:\mysql\bin>mysqlslap -u username -p --port=3306 --concurrency=100 -v --iterations=10 --query=query.sql --create=create.sql  --delimiter=";"
Enter password: *******
Benchmark
        Average number of seconds to run all queries: 2.116 seconds
        Minimum number of seconds to run all queries: 2.101 seconds
        Maximum number of seconds to run all queries: 2.151 seconds
        Number of clients running queries: 100
        Average number of queries per client: 1
```

//with no index on attribute "last"

```
C:\mysql\bin>mysqlslap -u username -p --port=3306 --concurrency=100 -v --iterations=10 --query=query.sql --create=create.sql  --delimiter=";"
Enter password: *******
Benchmark
        Average number of seconds to run all queries: 2.325 seconds
        Minimum number of seconds to run all queries: 2.102 seconds
        Maximum number of seconds to run all queries: 3.202 seconds
        Number of clients running queries: 100
        Average number of queries per client: 1
```

mysqlcheck — A Table Maintenance Program

The *mysqlcheck* client performs table maintenance: It checks, repairs, optimizes, or analyzes tables.

Caution!

Always backup data before performing a database/table repair operations; under some circumstances the operation might cause data loss.

Syntax:

```
shell> mysqlcheck [options] db_name [tbl_name ...]
shell> mysqlcheck [options] --databases db_name ...
shell> mysqlcheck [options] --all-databases
```

1. Check table in database
2. Process all tables in the named databases
3. Check all tables in all databases

Options:

--analyze: Analyze the tables
--check: Check tables for errors
--auto-repair: If a checked table is corrupted, automatically fix it
--optimize: Optimize the tables
--verbose, -v: Verbose mode. Print information about the various stages of program operation.

Example:

```
mysql> use test;
Database changed
mysql> show tables;
+----------------+
| Tables_in_test |
+----------------+
| empinfo        |
| employee       |
| inventory      |
| log            |
| transaction    |
+----------------+
5 rows in set (0.17 sec)
```

show full tables; (shows views)
(May have to modify options from following command in version 5.5+):

```
C:\mysql\bin> mysqlcheck -u username -p --port=3306 --analyze --check --auto-repair --
optimize -v  --databases test
Enter password: *******
# Connecting to localhost...
test.employee                                     OK
test.inventory                                    OK
test.log                                          OK
test.transaction
note     : Table does not support optimize, doing recreate + analyze instead
status   : OK
```

Disconnecting from localhost...
demo mysqlupgrade with two servers (one old, one new with no user-defined databases)

References:

http://forums.mysql.com/read.php?24,92131,92131

http://dev.mysql.com/doc/refman/5.1/en/mysqlslap.html

http://blogs.techrepublic.com.com/howdoi/?p=133

http://dev.mysql.com/doc/refman/5.1/en/show.html

http://dev.mysql.com/doc/refman/5.5/en/explain.html

http://forge.mysql.com/wiki/ServerVariables

http://dev.mysql.com/doc/refman/5.0/en/mysql-benchmarks.html

http://www.scribd.com/doc/2602028/Benchmarking-and-Monitoring-Tools-of-the-Trade-Part-I-Presentation

http://dev.mysql.com/doc/refman/5.1/en/mysqlcheck.html

http://www.techrepublic.com/blog/howdoi/how-do-i-stress-test-mysql-with-mysqlslap/133

Bad database design:
http://www.simple-talk.com/sql/database-administration/ten-common-database-design-mistakes/

http://www.geekgirls.com/databases_from_scratch_3.htm

http://edn.embarcadero.com/article/40466

http://dev.mysql.com/doc/refman/5.0/en/statistics-table.html

http://dev.mysql.com/tech-resources/articles/mysql-query-cache.html