Database Optimization for Web Developers

Little things that make a big difference
About Me

- 15 years experience in web development and administration (systems, databases, networks)
- Various jobs: Web Developer, Chief Architect, Manager of IT, Director of Technical Operations, Database Consultant
- Mostly worked with MySQL, as a developer, DBA, and administrator
- Member of the Utah Open Source Foundation (UTOS) core team
- Help organize the OpenWest Conference
- Organizer of the Ski PHP Conference
What Will We Talk About?

- Query optimizations
- Schema optimizations
- Server optimizations (briefly)
What Will Be Expected of You?
Assumptions

● Recent version of MySQL, Percona Server, or MariaDB
● InnoDB (or XtraDB) storage engine
● Demo database
MyISAM’s “table lock” problem

<table>
<thead>
<tr>
<th>Read Queue</th>
<th>Write Queue</th>
</tr>
</thead>
<tbody>
<tr>
<td>SELECT (0.1s)</td>
<td></td>
</tr>
<tr>
<td>SELECT (3.5s)</td>
<td></td>
</tr>
<tr>
<td>SELECT (121s)</td>
<td></td>
</tr>
<tr>
<td>SELECT (0.0s)</td>
<td>UPDATE (0.0s)</td>
</tr>
<tr>
<td>SELECT (1.2s)</td>
<td></td>
</tr>
<tr>
<td>SELECT (5.1s)</td>
<td></td>
</tr>
</tbody>
</table>
### MultiVersion Concurrency Control

<table>
<thead>
<tr>
<th>(Time T1)</th>
<th>(Time T2)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1: a</td>
<td>1: a</td>
</tr>
<tr>
<td>2: b</td>
<td>2: b</td>
</tr>
<tr>
<td>3: c</td>
<td>3: c</td>
</tr>
<tr>
<td>4: d</td>
<td>4: d</td>
</tr>
<tr>
<td>5: e</td>
<td>5: h</td>
</tr>
<tr>
<td>6: f</td>
<td>6: f</td>
</tr>
<tr>
<td>7: g</td>
<td>7: g</td>
</tr>
<tr>
<td>8: h</td>
<td>8: h</td>
</tr>
<tr>
<td>9: i</td>
<td>9: i</td>
</tr>
<tr>
<td>10: j</td>
<td>10: j</td>
</tr>
</tbody>
</table>

Query 1 at time T1:
```
SELECT * FROM tbl
```

Query 2 at time T2:
```
UPDATE tbl SET v='h' WHERE k=5
```

Even if Query 1 gets to row 5 after time T2, the database will return ‘e’ for row 5. The data will be consistent from when the query began.*

* with some caveats, depending on the server’s isolation level setting
Tools of the Trade

- EXPLAIN
- EXPLAIN EXTENDED
- SHOW KEYS FROM
- `pt-query-digest` (advanced)
EXPLAIN <query>

- ID – the order this table will be viewed
- SELECT_TYPE – what type of select query is happening on this table, usually SIMPLE
- TABLE – the table involved
- TYPE – how table is used, often const, ref, or all
- POSSIBLE_KEYS – list of keys that could be used
EXPLAIN <query>

- **KEY** – the key that will actually be used
- **KEY_LEN** – the length (in bytes) it will use
- **REF** – what columns (or constants) are being compared to the key
- **ROWS** – how many rows will be examined
- **EXTRA** – additional information
SHOW KEYS FROM <table>

- TABLE – the table
- NON_UNIQUE – 0 for unique, 1 for non-unique
- KEY_NAME – the name of the key
- SEQ_IN_INDEX – for multi-column keys, the sequence in which this column occurs
- COLUMN_NAME – the column
- COLLATION – how the index is sorted, A means ascending, or NULL
SHOW KEYS FROM <table>

- **CARDINALITY** – how many unique values
- **SUB_PART** – only relevant for partial indexes
- **PACKED** – only relevant for packed indexes
- **NULL** – is **NULL** an option for this column
- **INDEX_TYPE** – usually **BTREE**
- **COMMENT** – information about the index
- **INDEX_COMMENT** – comments provided when the index was created
What Do Keys Do?

- It's like the phone book's white pages
- Partial index use
- Multi-column indexes
- Can you look up by first name?
- Writing to the index is costly
Let's Look At Our Schema

- pagestats – 1 hour's page view statistics from the Wikimedia Foundation
- properties – ID/name mappings for the various Wikimedia properties
- languages – ID/name mappings for the various languages served
- For simplicity, language ID #1 is English, and property ID #1 is Wikipedia
- Other tables are for specific test cases
SHOW KEYS FROM pagestats

<table>
<thead>
<tr>
<th>Non_unique</th>
<th>Key_name</th>
<th>Seq_in_index</th>
<th>Column_name</th>
<th>Cardinality</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>PRIMARY</td>
<td>1</td>
<td>id</td>
<td>6929464</td>
</tr>
<tr>
<td>1</td>
<td>language_property</td>
<td>1</td>
<td>language_id</td>
<td>13</td>
</tr>
<tr>
<td>1</td>
<td>language_property</td>
<td>2</td>
<td>property_id</td>
<td>2370</td>
</tr>
<tr>
<td>1</td>
<td>language_property</td>
<td>3</td>
<td>requests</td>
<td>439362</td>
</tr>
<tr>
<td>1</td>
<td>property_id</td>
<td>1</td>
<td>property_id</td>
<td>13</td>
</tr>
<tr>
<td>1</td>
<td>page_hash</td>
<td>1</td>
<td>page_hash</td>
<td>6929464</td>
</tr>
<tr>
<td>1</td>
<td>page</td>
<td>1</td>
<td>page</td>
<td>6929464</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>id</th>
<th>language_id</th>
<th>property_id</th>
<th>page_hash</th>
<th>page</th>
<th>requests</th>
<th>size</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td></td>
<td></td>
<td>3</td>
</tr>
</tbody>
</table>
 Queries – Things to Note

- SQL_NO_CACHE – we'll use this flag in our SQL queries to tell MySQL not to use the query cache, if it exists.
- IGNORE INDEX (index_name) – we'll use this parameter to see how queries would perform without a particular index.
- STRAIGHT_JOIN – we’ll use this parameter to force a specific JOIN order.
SELECT SQL_NO_CACHE COUNT(*)
FROM pagestats
    IGNORE INDEX (property_id, language_property)
WHERE property_id = 2;
01-not-enough-indexes/01.sql

id: 1
select_type: SIMPLE
table: pagestats
type: ALL
possible_keys: NULL
key: NULL
key_length: NULL
ref: NULL
rows: 9682433
filtered: 100.00
Extra: Using where
SELECT SQL_NO_CACHE COUNT(*)
FROM pagestats
WHERE property_id=2;
id: 1

select_type: SIMPLE

table: pagestats

type: ref

possible_keys: property_id

key: property_id

key_len: 4

ref: const

rows: 72830

filtered: 100.00

Extra: Using index
SELECT SQL_NO_CACHE COUNT(*)
FROM pagestats
WHERE requests % 16 = 0;
id: 1
select_type: SIMPLE
table: pagestats
type: index
possible_keys: NULL
key: language_property
key_len: 12
ref: NULL
rows: 9682433
filtered: 100.00
Extra: Using where; Using index
SELECT SQL_NO_CACHE *
FROM pagestats
WHERE language_id=1 AND property_id=1
ORDER BY size DESC
LIMIT 20
02-multi-column/02.sql

SELECT SQL_NO_CACHE *
FROM pagestats
WHERE language_id=1
    AND property_id=1
ORDER BY requests DESC
LIMIT 20
### Too Many Indexes?

<table>
<thead>
<tr>
<th>Non_unique</th>
<th>Key_name</th>
<th>Seq_in_index</th>
<th>Column_name</th>
<th>Cardinality</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>PRIMARY</td>
<td>1</td>
<td>id</td>
<td>6929464</td>
</tr>
<tr>
<td>1</td>
<td>language_property</td>
<td>1</td>
<td>language_id</td>
<td>13</td>
</tr>
<tr>
<td>1</td>
<td>language_property</td>
<td>2</td>
<td>property_id</td>
<td>2370</td>
</tr>
<tr>
<td>1</td>
<td>language_property</td>
<td>3</td>
<td>requests</td>
<td>439362</td>
</tr>
<tr>
<td>1</td>
<td>property_id</td>
<td>1</td>
<td>property_id</td>
<td>13</td>
</tr>
<tr>
<td>1</td>
<td>page_hash</td>
<td>1</td>
<td>page_hash</td>
<td>6929464</td>
</tr>
<tr>
<td>1</td>
<td>dupe_index</td>
<td>1</td>
<td>property_id</td>
<td>146</td>
</tr>
<tr>
<td>1</td>
<td>dupe_index</td>
<td>2</td>
<td>page_hash</td>
<td>146</td>
</tr>
</tbody>
</table>
# Good Index vs. Bad Index

From a default WordPress install

<table>
<thead>
<tr>
<th>Non_unique</th>
<th>Key_name</th>
<th>Seq_in_index</th>
<th>Column_name</th>
<th>Cardinality</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>PRIMARY</td>
<td>1</td>
<td>ID</td>
<td>509</td>
</tr>
<tr>
<td>1</td>
<td>post_name</td>
<td>1</td>
<td>post_name</td>
<td>509</td>
</tr>
<tr>
<td>1</td>
<td>type_status_date</td>
<td>1</td>
<td>post_type</td>
<td>5</td>
</tr>
<tr>
<td>1</td>
<td>type_status_date</td>
<td>2</td>
<td>post_status</td>
<td>5</td>
</tr>
<tr>
<td>1</td>
<td>type_status_date</td>
<td>3</td>
<td>post_date</td>
<td>509</td>
</tr>
<tr>
<td>1</td>
<td>type_status_date</td>
<td>4</td>
<td>ID</td>
<td>509</td>
</tr>
<tr>
<td>1</td>
<td>post_parent</td>
<td>1</td>
<td>post_parent</td>
<td>84</td>
</tr>
<tr>
<td>1</td>
<td>post_author</td>
<td>1</td>
<td>post_author</td>
<td>11</td>
</tr>
</tbody>
</table>
SHOW KEYS FROM pagestats

<table>
<thead>
<tr>
<th>Non_unique</th>
<th>Key_name</th>
<th>Seq_in_index</th>
<th>Column_name</th>
<th>Cardinality</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>PRIMARY</td>
<td>1</td>
<td>id</td>
<td>6929464</td>
</tr>
<tr>
<td>1</td>
<td>language_property</td>
<td>1</td>
<td>language_id</td>
<td>13</td>
</tr>
<tr>
<td>1</td>
<td>language_property</td>
<td>2</td>
<td>property_id</td>
<td>2370</td>
</tr>
<tr>
<td>1</td>
<td>language_property</td>
<td>3</td>
<td>requests</td>
<td>439362</td>
</tr>
<tr>
<td>1</td>
<td>property_id</td>
<td>1</td>
<td>property_id</td>
<td>13</td>
</tr>
<tr>
<td>1</td>
<td>page_hash</td>
<td>1</td>
<td>page_hash</td>
<td>6929464</td>
</tr>
<tr>
<td>1</td>
<td>page</td>
<td>1</td>
<td>page</td>
<td>6929464</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>id</th>
<th>language_id</th>
<th>property_id</th>
<th>page_hash</th>
<th>page</th>
<th>requests</th>
<th>size</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td>3</td>
</tr>
</tbody>
</table>
Let's Talk About JOINs

- When you use JOIN query, you are combining data from two or more tables into a single result.
- If you do not specify a JOIN CONDITION, the result will be a Cartesian Join, which means that each row in the first table will match every row in the second table.
- JOIN CONDITIONS are specified using the ON or USING keywords.
- The order of the tables in the JOIN is important.
SELECT SQL_NO_CACHE STRAIGHT_JOIN
   p.name,
   SUM(s.requests),
   SUM(s.size)
FROM
   pagestats AS s
JOIN properties AS p ON p.id=s.property_id
GROUP BY
   s.property_id
SELECT SQL_NO_CACHE STRAIGHT_JOIN
  p.name,
  SUM(s.requests),
  SUM(s.size)
FROM properties AS p
JOIN pagestats AS s ON s.property_id=p.id
GROUP BY p.id
03-joins/*.sql EXPLAIN

01.sql

<table>
<thead>
<tr>
<th>table</th>
<th>type</th>
<th>key</th>
<th>key_len</th>
<th>ref</th>
<th>rows</th>
</tr>
</thead>
<tbody>
<tr>
<td>s</td>
<td>ALL</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
<td>7029800</td>
</tr>
<tr>
<td>p</td>
<td>eq_ref</td>
<td>PRIMARY</td>
<td>4</td>
<td>demo.s.property_id</td>
<td>1</td>
</tr>
</tbody>
</table>

02.sql

<table>
<thead>
<tr>
<th>table</th>
<th>type</th>
<th>key</th>
<th>key_len</th>
<th>ref</th>
<th>rows</th>
</tr>
</thead>
<tbody>
<tr>
<td>p</td>
<td>index</td>
<td>PRIMARY</td>
<td>4</td>
<td>NULL</td>
<td>15</td>
</tr>
<tr>
<td>s</td>
<td>ref</td>
<td>property_id</td>
<td>4</td>
<td>demo.p.id</td>
<td>390544</td>
</tr>
</tbody>
</table>
$languages = []; foreach ($db->query("SELECT id, name FROM languages ORDER BY name") as $row) {
    $languages[$row['id']] = $row;
}

foreach ($languages as $id => $info) {
    $stmt = $db->query("SELECT COUNT(*) AS num_pages, SUM(requests) AS num_requests FROM pagestats WHERE language_id = $id");
    $row = $stmt->fetch(PDO::FETCH_ASSOC);
    $languages[$id]['num_pages'] = $row['num_pages'];
    $languages[$id]['num_requests'] = $row['num_requests'];
}
Rewrite this as a single JOIN

```php
$languages = [];
foreach ($db->query("SELECT id, name FROM languages ORDER BY name") as $row) {
    $languages[$row['id']] = $row;
}

foreach ($languages as $id => $info) {
    $stmt = $db->query("SELECT COUNT(*) AS num_pages, SUM(requests) AS num_requests FROM pagestats WHERE language_id = $id");
    $row = $stmt->fetch(PDO::FETCH_ASSOC);
    $languages[$id]['num_pages'] = $row['num_pages'];
    $languages[$id]['num_requests'] = $row['num_requests'];
}
```
Other kinds of indexes

- B-tree index
- Hash index
- Full-text index
  - MyISAM only until MySQL 5.6, MariaDB 10.0.3
- Spatial index
  - “Minimum bounding rectangle” before MySQL 5.6 or MariaDB 5.5
Make Your Own Index

- Hash column for long keys
- Grid index
- Use an approximate index, but add non-indexed conditions to drill down
05-own-index/*.sql

- 01.sql does not use any index, so it has to check every row in the table
- 02.sql uses an index on the page_hash column to speed up the lookup
  - The page_hash is the CRC32() of the page column
- 03.sql uses a partial index on the page column to speed up the lookup
- What are the pros/cons of 02 vs 03?
For ease of calculation, we’ll assume that each entry consists of the indexed value, and the primary key.

There are a total of 8,458,355 rows in the pagestats table.

1. How much memory does the page_hash index use?
2. How much memory does the page(12) prefix index use?
05-own-index/*.sql

1. For the page_hash, how many buckets have more than one distinct page name in them?
2. What is the highest number of distinct page names per page_hash?
3. For the page(12) prefix index, how many buckets have more than one distinct page name in them?
4. What is the highest number of distinct page name per page(12)?
BREAK TIME

3:00 - 3:30 pm
If you can’t index it, cache it!

- Do the expensive stuff less often
- Run reports for executives / marketing once a day
- Create a read-only cache table for searching
  - Don’t propagate inactive rows to the active cache; in other words, limit the number of rows to be searched
  - Rotate cache tables to keep the active cache table read-only
- In some cases, sharded cache tables are useful
Eliminate redundant queries

- Ensure that commonly-accessed data is only retrieved once
- Store relevant user information in their session (with expiration)
- Use memcached (or an equivalent fast data store)
- If you use an ORM, it may have ways to facilitate this
Batch writes

- Put the data into a queue
- Have a process that reads the queue and does a bulk insert into the database
- Redis has a built-in queue data type (LIST)
INSERT INTO test_writes
    (name, num)
VALUES
    ('test 1', 1),
    ('test 2', 2),
    ('test 3', 3),
    ('test 4', 4),
    ('test 5', 5);
INSERT INTO test_writes
  (id, num)
VALUES
  (1, 2),
  (2, 4),
  (3, 6),
  (4, 8),
  (5, 10),
  (6, 12)
ON DUPLICATE KEY UPDATE
  num=VALUES(num)
CREATE TEMPORARY TABLE test_writes_tmp
(id int unsigned not null primary key,
  num int unsigned not null);

INSERT INTO test_writes_tmp (id, num)
VALUES (1, 2), (2, 4), (3, 6), (4, 8),
  (5, 10), (6, 12);

UPDATE test_writes_tmp tmp JOIN
test_writes tw ON tw.id=tmp.id SET tw.
  num=tmp.num;
Batch writes performance

Using your scripting language of choice (Python, PHP, Perl, Bash, etc), do a performance test of the following batch write techniques. Pre-load the test_writes table with 100K rows, then update the num column to a different value. Experiment with updating different numbers of rows at a time.

1. Update one row at a time
2. Update one row at a time, in a single transaction
3. INSERT ... ON DUPLICATE KEY UPDATE
4. Create a temporary table, then do a multi-table UPDATE
Good to know...

The remainder of the information can be very helpful to know.

However, it requires more specialized tools, so we will not be doing any more exercises.
Replication: The Good Stuff

- Provides some redundancy in case of server failures
- Run long queries for reporting on a slave without slowing down the master
- Take backups from one of your slaves
- Spread load in read-heavy environments
- Allow database maintenance and upgrades without taking application down
Replication: Caveats

- This is by no means a cure-all
- One should not use replication slaves to solve concurrency issues without understanding the ramifications
- It’s possible that replication can increase the impact of your write queries
- Slave lag can cause major consistency issues if your app doesn't plan for it
- Slaves can get out of sync due to app or database bugs
- Slave promotion is not automatic
Percona Configuration Wizard

- Available at tools.percona.com/wizard

- Tell it a little about your server

- It will recommend a configuration for you
Tame Your Framework

- Do you really know what queries your ORM is making to your database server?
- Debug console can be useful
- `pt-query-digest` will allow you to analyze samples without changing your application
pt-query-digest

- Analyzes queries to find bottlenecks
- Many sources of query information:
  - Slow log
  - General log
  - Binary log
  - SHOW PROCESSLIST
  - tcpdump of MySQL traffic
pt-query-digest

tcpdump -x -nn -q -ttttt \
    port 3306 > mysql.pcap

pt-query-digest \
    --type tcpdump \
    mysql.pcap \
> mysql.log
<table>
<thead>
<tr>
<th>Rank</th>
<th>Query ID</th>
<th>Response time</th>
<th>Calls</th>
<th>R/Call</th>
<th>V/M</th>
<th>Item</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0xAA8DE031A809B98D</td>
<td>0.2933</td>
<td>31.1%</td>
<td>930</td>
<td>0.0003</td>
<td>SELECT mommy_options</td>
</tr>
<tr>
<td>2</td>
<td>0x0835F93CABD17C77</td>
<td>0.2000</td>
<td>21.2%</td>
<td>29</td>
<td>0.0069</td>
<td>SELECT mommy_posts</td>
</tr>
<tr>
<td>3</td>
<td>0x94ED99BB0975A13E</td>
<td>0.1085</td>
<td>11.5%</td>
<td>142</td>
<td>0.0008</td>
<td>SELECT mommy_postmeta</td>
</tr>
<tr>
<td>4</td>
<td>0x7A1D5F404164BF9F</td>
<td>0.0457</td>
<td>4.8%</td>
<td>94</td>
<td>0.0005</td>
<td>SELECT mommy_posts</td>
</tr>
<tr>
<td>Attribute</td>
<td>pct</td>
<td>total</td>
<td>min</td>
<td>max</td>
<td>avg</td>
<td>95%</td>
</tr>
<tr>
<td>---------------------</td>
<td>-----</td>
<td>-------</td>
<td>------</td>
<td>------</td>
<td>-------</td>
<td>-----</td>
</tr>
<tr>
<td>Count</td>
<td>51</td>
<td>930</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Exec time</td>
<td>31</td>
<td>293ms</td>
<td>188us</td>
<td>1ms</td>
<td>315us</td>
<td>403us</td>
</tr>
<tr>
<td>Rows affected</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Query size</td>
<td>42</td>
<td>81.93k</td>
<td>76</td>
<td>114</td>
<td>90.22</td>
<td>102.22</td>
</tr>
<tr>
<td>Warning count</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Databases</td>
<td>^@mysql_native_password</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Hosts</td>
<td>172.17.1.25</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Query_time distribution</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1us</td>
<td>#</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10us</td>
<td>#</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>100us</td>
<td>#</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1ms</td>
<td>#</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10ms</td>
<td>#</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>100ms</td>
<td>#</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1s</td>
<td>#</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10s+</td>
<td>#</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
CONCLUSION

◆ Know what your database is doing
◆ Use the readily available tools to get a better idea of where your bottlenecks are
◆ Break your problem down into its component parts
◆ Make sure you understand the ramifications of your proposed solutions
Thanks For Coming

@stevecougu

http://github.com/stevecougu