Need for Speed: MySQL Indexing

Percona Live 2013, November 11 – 12, London

Oli Sennhauser

Senior MySQL Consultant at FromDual GmbH

oli.sennhauser@fromdual.com
About FromDual GmbH

- FromDual provides neutral and independent:
  - Consulting for MySQL, Percona Server, MariaDB
  - Support for all MySQL and Galera Cluster
  - Remote-DBA Services
  - MySQL Training
- Oracle Silver Partner (OPN)

www.fromdual.com
Our customers
MySQL and Indexing

MySQL documentation says:

*The best way to improve the performance of SELECT operations is to create indexes on one or more of the columns that are tested in the query.*

Great! But:

*Unnecessary indexes waste space and waste time to determine which indexes to use. You must find the right balance to achieve fast queries using the optimal set of indexes.*

... hmm so we have to think a bit... :-(
What is an Index?

- Adams, Douglas: The Hitchhiker's Guide to the Galaxy?

- Sennhauser, Oli, Uster?
What is an Index technically?
MySQL uses indexes:

- To enforce uniqueness (PRIMARY KEY, UNIQUE KEY)
- To fast access and filter rows (WHERE)
- To perform joins fast (JOIN)
- To find MIN() and MAX() values
- For sorting and grouping (ORDER BY, GROUP BY)
- To avoid joins by using covering indexes
- To enforce FOREIGN KEY Constraints (FOREIGN KEY)
### WHERE clause 1

```
SELECT *
FROM customers
WHERE name = 'No Clue of MySQL LLC';
```

### SHOW CREATE TABLE customers

```
SHOW CREATE TABLE customers\G
```

```
CREATE TABLE `customers` (  
  `customer_id` smallint(5) unsigned,
  `name` varchar(64) DEFAULT NULL
) ENGINE InnoDB DEFAULT CHARSET=utf8;
```

### EXPLAIN

```
EXPLAIN
SELECT *
FROM customers
WHERE name = 'No Clue of MySQL LLC';
```

```
+-----------+------+---------------+------+-------+-------------+
| table     | type | possible_keys | key  | rows  | Extra       |
|------------+------|---------------+-------|-------|-------------|
| customers  | ALL   | NULL          | NULL  | 31978  | Using where |
```

---

**Note:** The output of `EXPLAIN` shows the execution plan for the `SELECT` statement, indicating that the query will use the index if available. The `Using where` flag indicates that the WHERE clause is used to eliminate rows from the table before applying the table scan.
How to create and Index?

ALTER TABLE …

- ADD PRIMARY KEY (id);
- ADD UNIQUE KEY (uuid);
- ADD FOREIGN KEY (customer_id)
  REFERENCES customers (customer_id);
- ADD INDEX (last_name, first_name);
- ADD INDEX pre_ind (hash(8));
- ADD FULLTEXT INDEX (last_name, first_name);
WHERE clause 2

Alter Table customers
ADD INDEX (name);
### JOIN clause

```sql
EXPLAIN SELECT *
FROM customers AS c
JOIN orders AS o ON c.customer_id = o.customer_id
WHERE c.name = 'No Clue of MySQL LLC';
```

<table>
<thead>
<tr>
<th>table</th>
<th>type</th>
<th>possible_keys</th>
<th>key</th>
<th>key_len</th>
<th>ref</th>
<th>rows</th>
</tr>
</thead>
<tbody>
<tr>
<td>c</td>
<td>ref</td>
<td>PRIMARY,name</td>
<td>name</td>
<td>67</td>
<td>const</td>
<td>1</td>
</tr>
<tr>
<td>o</td>
<td>ALL</td>
<td></td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
</tr>
</tbody>
</table>

Gain: 450 ms → 6 ms

```sql
ALTER TABLE orders
ADD INDEX (customer_id);
```

<table>
<thead>
<tr>
<th>table</th>
<th>type</th>
<th>possible_keys</th>
<th>key</th>
<th>key_len</th>
<th>ref</th>
<th>rows</th>
</tr>
</thead>
<tbody>
<tr>
<td>c</td>
<td>ref</td>
<td>PRIMARY,name</td>
<td>name</td>
<td>67</td>
<td>const</td>
<td>1</td>
</tr>
<tr>
<td>o</td>
<td>ref</td>
<td>customer_id</td>
<td>customer_id</td>
<td>3</td>
<td>c.customer_id</td>
<td>8</td>
</tr>
</tbody>
</table>
For sorting/grouping tables

ORDER BY, GROUP BY

**EXPLAIN SELECT * FROM contacts AS c WHERE last_name = 'Sennhauser' ORDER BY last_name, first_name;**

<table>
<thead>
<tr>
<th>table</th>
<th>type</th>
<th>key</th>
<th>rows</th>
<th>Extra</th>
</tr>
</thead>
<tbody>
<tr>
<td>c</td>
<td>ref</td>
<td>last_name</td>
<td>1561</td>
<td>Using index condition; Using where; Using filesort</td>
</tr>
</tbody>
</table>

**Gain: 20 ms → 7 ms**

**ALTER TABLE contacts ADD INDEX (last_name, first_name);**

<table>
<thead>
<tr>
<th>table</th>
<th>type</th>
<th>key</th>
<th>rows</th>
<th>Extra</th>
</tr>
</thead>
<tbody>
<tr>
<td>contacts</td>
<td>ref</td>
<td>last_name_2</td>
<td>1561</td>
<td>Using where; Using index</td>
</tr>
</tbody>
</table>
Covering Indexes

**EXPLAIN**

```sql
SELECT customer_id, amount
FROM orders AS o
WHERE customer_id = 59349;
```

<table>
<thead>
<tr>
<th>table</th>
<th>type</th>
<th>key</th>
<th>rows</th>
<th>Extra</th>
</tr>
</thead>
<tbody>
<tr>
<td>o</td>
<td>ref</td>
<td></td>
<td>15</td>
<td></td>
</tr>
</tbody>
</table>

**ALTER TABLE** orders

```
ADD INDEX (customer_id, amount);
```

<table>
<thead>
<tr>
<th>table</th>
<th>type</th>
<th>key</th>
<th>rows</th>
<th>Extra</th>
</tr>
</thead>
<tbody>
<tr>
<td>o</td>
<td>ref</td>
<td>customer_id_2</td>
<td>15</td>
<td>Using index</td>
</tr>
</tbody>
</table>
Benefit of Covering Indexes

• Why are Covering Indexes beneficial

Abächerli | PK
Bucher | PK
Cathomas | PK
Fuchs | PK
Haas | PK
Kübler | PK
Sennhauser | PK
Vögeli | PK
Zürcher | PK

1, Cathomas, Marco, Bahnhofstrasse 3, 3034, Lausanne; 2, Abächerli, Hans, Hauptstrasse 13, 8001, Zürich; 3, Haas, Daniel, Rüttistrasse 11, 8500, Zürich; 4, Fuchs, Hans-Peter, Reservestrasse 3, 8610, Uster; 5, Kübler, Kobi, Tschuriplatz 1, 8000, Zürich; 6, Sennhauser, Oli, Rebenweg 6, 8610 Uster; 7, Vögeli, Fridolin, Peterstrasse 3, 8610, Uster; 8, Bucher, Walti, Peterstrasse 1, 8610, Uster
How-to find missing indexes?

- ER Diagram? :-(
  - Most of them rely to you business logic...
- How-to FIND?
- Slow Query Log
- MySQL Variables:
- Since v5.1 on-line!

<table>
<thead>
<tr>
<th>Variable_name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>log_queries_not_using_indexes</td>
<td>ON</td>
</tr>
<tr>
<td>long_query_time</td>
<td>0.250000</td>
</tr>
<tr>
<td>min_examined_row_limit</td>
<td>100</td>
</tr>
<tr>
<td>slow_query_log</td>
<td>ON</td>
</tr>
<tr>
<td>slow_query_log_file</td>
<td>slow.log</td>
</tr>
</tbody>
</table>
Indexes are not only good

- Indexes use space (Disk, hot data in RAM!)
- Indexes use time to maintain (CPU, RAM, I/O)
- Optimizer needs time to determine which indexes to use.
- Sometimes optimizer is completely confused and does wrong decisions if too many (similar) indexes are there.

→ You must find the right balance to achieve fast queries using the optimal set of indexes.
Smaller indexes faster queries

- Better fit into memory (less I/O)
- Higher data density (rows/block)
- Less CPU cycles (to crawl through)
- Prefixed indexes:

```
ADD INDEX pre_ind (hash(8));
```
Avoid indexes

• Avoid redundant (and thus unnecessary) indexes

• How does it happen?
  • Developer 1: Creates a Foreign Key constraint → done
  • Developer 2: Ouu! Query is slow → Oli told me to create an index! → done
  • Developer 3: Ouu! Query is slow → Developer 2 is stupid! → Create and index → done

• Frameworks vs. Developer

• Upgrade process vs. Developer

• Avoid indexes which are not used / needed
How to find such indexes?

SHOW CREATE TABLE ...

mysql dump --no-data > structure_dump.sql

• Since MySQL 5.6: PERFORMANCE_SCHEMA
  • Percona Server / MariaDB: Userstats
  • http://fromdual.com/mysql-performance-schema-hints

```sql
SELECT object_schema, object_name, index_name,
FROM performance_schema.table_io_waits_summary_by_index_usage
WHERE index_name IS NOT NULL
  AND count_star = 0
ORDER BY object_schema, object_name;
```
Avoid partial redundant indexes

- INDEX (city, last_name, first_name)
- INDEX (city, last_name)
- INDEX (city)
- INDEX (last_name, city) ???
- INDEX (first_name, last_name) !!!
Bad selectivity

- Remove indexes with bad selectivity (≈ low cardinality)
- Candidates are:
  - status
  - gender
  - active
- How to find if field has bad selectivity?

Indexes (and Joins) are expensive!!!
- Break even between 15% and 66%
- Lets see if the MySQL optimizer knows about it... :-)

SELECT status, COUNT(*)
FROM orders
GROUP BY status;

<table>
<thead>
<tr>
<th>status</th>
<th>cnt</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>393216</td>
</tr>
<tr>
<td>1</td>
<td>262144</td>
</tr>
<tr>
<td>2</td>
<td>12</td>
</tr>
<tr>
<td>3</td>
<td>36</td>
</tr>
<tr>
<td>4</td>
<td>24</td>
</tr>
<tr>
<td>5</td>
<td>4</td>
</tr>
<tr>
<td>6</td>
<td>8</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Abscherli</th>
<th>PK</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bucher</td>
<td>PK</td>
</tr>
<tr>
<td>Cathomas</td>
<td>PK</td>
</tr>
<tr>
<td>Fuchs</td>
<td>PK</td>
</tr>
<tr>
<td>Hans</td>
<td>PK</td>
</tr>
<tr>
<td>Kuebler</td>
<td>PK</td>
</tr>
<tr>
<td>Mennhauser</td>
<td>PK</td>
</tr>
<tr>
<td>Vögele</td>
<td>PK</td>
</tr>
<tr>
<td>Zürcher</td>
<td>PK</td>
</tr>
</tbody>
</table>
Optimizer is wrong!

<table>
<thead>
<tr>
<th>table</th>
<th>type</th>
<th>possible_keys</th>
<th>key</th>
<th>rows</th>
</tr>
</thead>
<tbody>
<tr>
<td>orders</td>
<td>ref</td>
<td>status</td>
<td>status</td>
<td>12</td>
</tr>
</tbody>
</table>

SELECT * FROM orders WHERE status = 0;
1.43 s

<table>
<thead>
<tr>
<th>table</th>
<th>type</th>
<th>possible_keys</th>
<th>key</th>
<th>rows</th>
</tr>
</thead>
<tbody>
<tr>
<td>orders</td>
<td>ref</td>
<td>status</td>
<td>status</td>
<td>327469</td>
</tr>
</tbody>
</table>

SELECT status, COUNT(*)
FROM orders
GROUP BY status;

<table>
<thead>
<tr>
<th>status</th>
<th>cnt</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>393216</td>
</tr>
<tr>
<td>1</td>
<td>262144</td>
</tr>
<tr>
<td>2</td>
<td>12</td>
</tr>
<tr>
<td>3</td>
<td>36</td>
</tr>
<tr>
<td>4</td>
<td>24</td>
</tr>
<tr>
<td>5</td>
<td>4</td>
</tr>
<tr>
<td>6</td>
<td>8</td>
</tr>
</tbody>
</table>

SELECT * FROM orders WHERE status = 0;
0.44 s

<table>
<thead>
<tr>
<th>table</th>
<th>type</th>
<th>I</th>
</tr>
</thead>
<tbody>
<tr>
<td>orders</td>
<td>ALL</td>
<td>I</td>
</tr>
</tbody>
</table>

5.6.12 (after analyze table)
InnoDB PK and SK

- InnoDB has
  - Primary Keys and
  - Secondary Keys
Clustered Index

• InnoDB: Data = Leaf of Primary Key
  • We call this an Index Clustered Table (IOT)
    → Data are sorted like PK (key is sorted)!
    → PK influences Locality of data (physical location)
• AUTO_INCREMENT ~= sorting by time!
• Good for many things
  • where hot data = recent data
• Bad for time series
  • Where hot data = per item data
### Example: InnoDB

<table>
<thead>
<tr>
<th>#</th>
<th>ts</th>
<th>v_id</th>
<th>xpos</th>
<th>ypos</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>17:30</td>
<td>#42</td>
<td>x, y</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>17:30</td>
<td>#43</td>
<td>x, y</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>17:30</td>
<td>#44</td>
<td>x, y</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2001</td>
<td>17:32</td>
<td>#42</td>
<td>x, y</td>
<td></td>
</tr>
<tr>
<td>2002</td>
<td>17:32</td>
<td>#43</td>
<td>x, y</td>
<td></td>
</tr>
<tr>
<td>2003</td>
<td>17:32</td>
<td>#44</td>
<td>x, y</td>
<td></td>
</tr>
</tbody>
</table>

**Q1:** Δ in rows?  ~ 2000 rows  
**A1:** 1 row ~ 100 byte

**Q2:** Δ in bytes?  ~ 200 kbyte

**Q3:** Default InnoDB block size?  default: 16 kbyte

**Q4:** Avg. # of rows of car #42 in 1 InnoDB block?  ~ 1

**A2:** 3 d and 720 pt/d → ~2000 pt ~ 2000 rec ~ 2000 blk

**Q5:** How long will this take and why (32 Mbyte)?

~ 2000 IOPS ~ 10s random read!!!

S: All in RAM or strong I/O system or …?
Q1: Avg. # of rows of car #42 in 1 InnoDB block?  ~ 120
Q2: How long will this take and why (320 kbyte)?
   ~ 1-2 IOPS ~ 10-20 ms sequential read!
S: Wow f=50 faster! Any drawbacks?
Index hints

- MySQL optimizer is sometimes wrong!
  - We have to help (= hint) him...
- Index hints are:
  - USE INDEX (ind1, ind2)
    - Only consider these indexes
  - FORCE INDEX (ind3)
    - Use this index without considering anything else
  - IGNORE INDEX (ind1, ind3)
    - Do NOT consider these indexes but everything else
- Hints should be used only as a last resort
MySQL Variables

- MySQL variables influencing index use
  - MyISAM: `key_buffer_size`
  - InnoDB: `innodb_buffer_pool_size / innodb_buffer_pool_instances`

- InnoDB Change Buffer
  - `innodb_change_buffer_max_size`
  - `innodb_change_buffering`

- Adaptive Hash Index (AHI)

- MySQL 5.6.3 / 5.5.14 index length 767 → 3072 bytes
  - `innodb_large_prefix`
We have time for some face-to-face talks...

- FromDual provides neutral and independent:
  - Consulting
  - Remote-DBA
  - Support for MySQL, Galera, Percona Server and MariaDB
  - Training

www.fromdual.com/presentations