Top Five MySQL Query Tuning Tips
Janis Griffin, Database Performance Evangelist
Who Am I

- Senior DBA / Performance Evangelist for SolarWinds
  - Janis.Griffin@solarwinds.com
  - Twitter® - @DoBoutAnything
  - Current – 25+ Years in Oracle®, SQL Server®, ASE, and now MySQL®
  - DBA and Developer

- Specialize in Performance Tuning

- Review Database Performance for Customers and Prospects

- Common Question – How do I tune it?
• Challenges Of Tuning
• Top Five Tuning Tips
  • Monitor Wait Time
    • Find the right SQL statements to work on
    • Get Baseline Metrics
  • Review the Execution Plan
    • Know which Optimizer Features are being used
  • Gather Object Information
    • Review Table, Column, Index, and Constraint Information
    • Understand Column Selectivity, and Statistics
  • Find the Driving Table
    • Consider SQL Diagramming
  • Engineer out the Stupid
Challenges Of Tuning

• SQL Tuning is Hard
  • Who should tune – DBA or Developer?
  • Which SQL to tune

• Requires Expertise in Many Areas
  • Technical – Plan, Data Access, SQL Design
  • Business – What is the Purpose of SQL?

• Tuning Takes Time
  • Large Number of SQL Statements
  • Each Statement is Different

• Low Priority in Some Companies
  • Vendor Applications
  • Focus on Hardware or System Issues

• Never Ending

© 2017 SolarWinds Worldwide, LLC. All rights reserved.
Consider these tips as building blocks for your ‘Tuning’ toolbox.
1. Monitor Wait Time

Focus on Total Wait Time

Identify Wait-Time at every step and rank bottlenecks by user impact.

- Understand the total time a Query spends in Database
- Measure time while Query executes
- MySQL helps by providing Wait Events and Thread States
• Greatly improved.
• More information / more instrumentation
• Current / historical events at all levels
  • Statements
  • Stages
  • Waits
• Performance_Schema consumers now ON by default
• Storage Engine now defaults to INNODB
• 5.7 Performance_Schema
  • 35 new tables – 5.7 has 87, 5.6 had 52, 5.5 had 17
  • Reduced overhead and footprint
  • 1005 Instruments (5.7.10)
    • New ones for transactions, metadata locks, memory usage
• SYS Schema – now provided by default
  • 100 views – User, Host, IO, InnoDB, Memory Summaries
  • Wait analysis and statement level
MySQL Wait Time Data
INSERT INTO wta_data SELECT current_timestamp, t.processlist_id AS conn_id, t.processlist_user AS user, t.processlist_host AS host, t.processlist_db AS db,
LOWER(t.processlist_state) AS state, t.processlist_info AS current_sql, MD5(t.processlist_info) AS current_sql_md5,
t.processlist_command AS command, w.event_id AS wait_event_id, w.end_event_id AS wait_event_end_id,
w.event_name AS wait_event_name, w.operation AS wait_operation, w.object_schema AS wait_schema, w.object_name AS wait_object_name,
w.object_type AS wait_type, w.index_name AS wait_index, s.event_name AS statement_event, s.sql_text AS statement_sql,
MD5(s.sql_text) AS statement_sql_md5, CASE WHEN RIGHT(s.digest_text, 3) = '...' THEN 1 ELSE 0 END AS statement_digest_truncated,
s.digest AS statement_digest, s.event_id AS statement_event_id, s.end_event_id AS statement_event_end_id,
conn.attr_value AS program_name, trx.trx_state, trx.trx_operation_state, trx.blocking_id
FROM performance_schema.threads AS t  LEFT JOIN performance_schema.events_waits_current AS w ON w.thread_id = t.thread_id AND w.end_event_id IS NULL AND w.event_name <> 'idle'
LEFT JOIN performance_schema.events_statements_current AS s ON s.thread_id = t.thread_id AND s.digest IS NOT NULL
LEFT JOIN (SELECT processlist_id, GROUP_CONCAT(attr_value ORDER BY attr_name DESC SEPARATOR '~^~') attr_value
  FROM performance_schema.session_connect_attrs WHERE attr_name IN ('program_name','_client_name') GROUP BY processlist_id ) conn
ON t.processlist_id = conn.processlist_id
LEFT JOIN (SELECT trx.trx_mysql_thread_id processlist_id, blocking trx.trx_mysql_thread_id blocking_id, trx.trx_state, trx.trx_operation_state
  FROM information_schema.innodb_trx)
  LEFT JOIN (SELECT requesting_trx_id, MIN(blocking_trx_id) blocking_trx_id FROM information_schema.innodb_lock_waits
GROUP BY requesting_trx_id) lock_waits  ON trx.trx_id=lock_waits.requesting_trx_id
LEFT OUTER JOIN information_schema.innodb_trx blocking_trx ON lock_waits.blocking_trx_id=blocking_trx.trx_id
WHERE t.processlist_id IS NOT NULL AND t.instrumented = 'YES' AND t.processlist_id <> connection_id()
AND (t.processlist_info IS NOT NULL OR s.sql_text IS NOT NULL)
AND (t.PROCESSLIST_COMMAND <> 'Sleep')
select w.sql_text, w.wait_operation, w.time_in_seconds, tot.tot_time from

    (select substr(current_sql, 1, 60) sql_text, wait_operation, count(*) time_in_seconds from wta_data w group by substr(current_sql, 1, 60), wait_operation) w

    (select substr(current_sql, 1,60) sql_text, count(*) tot_time from wta_data group by substr(current_sql, 1,60)) tot

where w.wait_operation is not null
and w.sql_text = tot.sql_text

order by tot.tot_time, time_in_seconds;
Almost 16 hours spent on 'waiting for table metadata lock'

100% of time on 'sending data'

Over 20 hours spent!
Sending data

While your query is in this state, MySQL tends to perform large amounts of disk access (reads). Sending data will often be the largest bar in DPA as queries tend to spend a lot of time in this wait (state).

Resolved by

DBAs, storage admins, network admins

Solutions

If this is one of your top waits, consider the following:

- Optimize SQL to reduce Sending Data wait time.
  1. On the Trends page for the instance, click the Waits tab.
  2. In the Top Waits graph, drill down to a day and time that have significant amounts of time spent in the sending data wait state.
  3. Click sending data on the graph to see the SQL statements most affected by this wait state.
  4. Click each SQL and compare Rows Examined to Rows Affected or Sent statistics in the SQL Data tab.
  5. If Rows Examined is more than ten times larger than Rows Affected or Sent, consider the following:
     - Use summary tables where possible to limit the number of rows processed.
     - Rewrite complicated queries to assist in processing fewer rows.
     - Evaluate WHERE clauses to ensure you process only rows that are required.

- If the number of rows retrieved hasn't changed but the time spent in Sending Data has increased, this could indicate slow I/O performance.

- This can indicate slow network speeds between the MySQL server and the application. This is prevalent in cases where the SQL returns a lot of rows back to the application.
### WTA Benefits – Blocking Issues

**Note:** Data reflects activity for sessions accessing tables managed by the InnoDB (or an InnoDB-based) storage engine.

<table>
<thead>
<tr>
<th>SPID</th>
<th>Caused</th>
<th>Waited</th>
<th>User</th>
<th>Program Name</th>
<th>Machine</th>
<th>SQL</th>
</tr>
</thead>
<tbody>
<tr>
<td>1804091 (blocker)</td>
<td>235</td>
<td>0</td>
<td>root</td>
<td>No Program Name</td>
<td>localhost</td>
<td>Details</td>
</tr>
<tr>
<td>1803966 (blocker and waiter)</td>
<td>54</td>
<td>10</td>
<td>root</td>
<td>No Program Name</td>
<td>localhost</td>
<td>SELECT s_quantity, s_distr_06, s_data INTO out_s_quantity, tmp_s_distr, tmp_s_data FROM stock WHERE s_lid = sl_id AND s_w_lid = w_lid FOR UPDATE</td>
</tr>
<tr>
<td>1804026 (blocker and waiter)</td>
<td>30</td>
<td>10</td>
<td>root</td>
<td>No Program Name</td>
<td>localhost</td>
<td>Details</td>
</tr>
<tr>
<td>1804032 (blocker and waiter)</td>
<td>30</td>
<td>10</td>
<td>root</td>
<td>No Program Name</td>
<td>localhost</td>
<td>Details</td>
</tr>
<tr>
<td>1804092 (blocker and waiter)</td>
<td>28</td>
<td>10</td>
<td>root</td>
<td>No Program Name</td>
<td>localhost</td>
<td>Details</td>
</tr>
<tr>
<td>1803911 (blocker and waiter)</td>
<td>9</td>
<td>9</td>
<td>root</td>
<td>No Program Name</td>
<td>localhost</td>
<td>Details</td>
</tr>
<tr>
<td>1804034 (waiter)</td>
<td>10</td>
<td>0</td>
<td>root</td>
<td>No Program Name</td>
<td>localhost</td>
<td>Details</td>
</tr>
<tr>
<td>1804050 (waiter)</td>
<td>10</td>
<td>0</td>
<td>root</td>
<td>No Program Name</td>
<td>localhost</td>
<td>Details</td>
</tr>
<tr>
<td>1804024 (waiter)</td>
<td>10</td>
<td>0</td>
<td>root</td>
<td>No Program Name</td>
<td>localhost</td>
<td>Details</td>
</tr>
<tr>
<td>1803908 (waiter)</td>
<td>5</td>
<td>0</td>
<td>root</td>
<td>No Program Name</td>
<td>localhost</td>
<td>Details</td>
</tr>
</tbody>
</table>

**SQL Statements Executed by Blocking Session 1804091 while Holding the Lock | MYLAB:3306**

**September 17, 2015 - 8:00AM to 3:52PM**

```
4290438705
update_stock
```

© 2017 SolarWinds Worldwide, LLC. All rights reserved.
• Get baseline metrics.
  • How long does it take now?
  • What is acceptable? (Ten seconds? Two minutes? One hour?)
  • Record ‘rows examined’ and ‘rows sent’ for tuning comparison

• Collect wait or thread states.
  • Locking / blocking (system lock)
  • I/O problem (sending data)
  • Calculating statistics (statistics)
  • Network slowdown (writing to net)
  • May be multiple issues
  • All have different resolutions

© 2017 SolarWinds Worldwide, LLC. All rights reserved.
SELECT dayofweek (payment_date) AS day_of_week,
    staff.first_name, staff.last_name,
    city.city, country.country, category.name AS category,
    SUM(payment.amount) AS total_sales
FROM payment
INNER JOIN rental ON payment.rental_id = rental.rental_id
INNER JOIN inventory ON rental.inventory_id = inventory.inventory_id
INNER JOIN film ON inventory.film_id = film.film_id
INNER JOIN film_category ON film.film_id = film_category.film_id
INNER JOIN category ON film_category.category_id = category.category_id
INNER JOIN store ON store.store_id = inventory.store_id
INNER JOIN address ON address.address_id = store.address_id
INNER JOIN city ON city.city_id = address.city_id
INNER JOIN country ON country.country_id = city.country_id
INNER JOIN staff ON staff.staff_id = rental.staff_id
WHERE quarter(rental_date) = ?
    AND return_date IS NOT ?
GROUP BY dayofweek(payment_date), staff.first_name, staff.last_name, city.city, country.country, category.name
ORDER BY total_sales DESC
WTA – Baseline Metrics

Total Wait Time for SQL Sales Quarterly Contest | GRID12C:3306 | March 20, 2017 to May 2, 2017

Executions

Rows Affected or Sent

Rows Examined

Saturday - April 22, 2017

Wait  sending data
Average Time (This Wait)  14.84 (msecs)
Average Time (All Waits)  15.13 (msecs)
Contribution of This Wait  97%
SQL Executions  1,430

SQL Text
SELECT day_of_week, payment_date AS day_of_week, staff . first_name, staff . last_name, city, city, country . country, category AS category, SUM (payment . amount) AS total_sales FROM payment INNER JOIN rental ON payment . rental_id = rental . rental_id INNER JOIN inventory ON rental . inventory_id = inventory . inventory_id
2. Review the Execution Plan

- Explain
  - select_type | table        | type   | possible_keys                          | key
  - SIMPLE      | store        | index  | PRIMARY,idx_fk_address_id              | idx_fk_address_id
  - SIMPLE      | address      | eq_ref | PRIMARY,idx_fk_city_id                 | PRIMARY
  - SIMPLE      | city         | eq_ref | PRIMARY,idx_fk_country_id              | PRIMARY
  - SIMPLE      | country      | eq_ref | PRIMARY                                 | PRIMARY
  - SIMPLE      | payment      | ALL    | fx_payment_rental                       | NULL
  - SIMPLE      | rental       | eq_ref | PRIMARY,idx_fk_inventory_id,idx_fk_staff_id | PRIMARY
  - SIMPLE      | staff        | eq_ref | PRIMARY                                 | PRIMARY
  - SIMPLE      | inventory    | eq_ref | PRIMARY,idx_fk_film_id,idx_store_id_film_id | PRIMARY
  - SIMPLE      | film         | eq_ref | PRIMARY                                 | PRIMARY
  - SIMPLE      | film_category| ref    | PRIMARY,fk_film_category_category       | PRIMARY
  - SIMPLE      | category     | eq_ref | PRIMARY                                 | PRIMARY

Tip: Try to avoid using table aliases—they don’t translate in plan.

- Explain Extended
- Explain FORMAT=JSON
- Optimizer Trace 5.6.3+
- MySQL Workbench
**EXPLAIN Cheat Sheet**


### Syntax: EXPLAIN <SELECT STATEMENT>

<table>
<thead>
<tr>
<th>System</th>
<th>Table has 1 value (a system table)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Const</td>
<td>Evaluates if not using PRIMARY or UNIQUE KEY</td>
</tr>
<tr>
<td>Eq_ref</td>
<td>Evaluates if not using PRIMARY or UNIQUE KEY</td>
</tr>
<tr>
<td>Ref</td>
<td>JOINS using any pseudo-JOIN using KEY that isn't</td>
</tr>
<tr>
<td>Fulltext</td>
<td>FULLTEXT index used</td>
</tr>
<tr>
<td>Ref_or_null</td>
<td>as for Ref, with an entire row for NULL values</td>
</tr>
<tr>
<td>Index_merge</td>
<td>Index merge optimization used; keys and key/or row values are TRUE</td>
</tr>
<tr>
<td>Unique_subquery</td>
<td>Subquery on a PRIMARY or UNIQUE KEY of table</td>
</tr>
<tr>
<td>Index_subquery</td>
<td>Subquery on a non-unique KEY of one table</td>
</tr>
<tr>
<td>Range</td>
<td>Key compared with &lt;=, &lt;=, &lt;, &gt;, &gt;, IS NULL, &lt;=, &lt;, &gt;=, BETWEEN, or IN(exists)</td>
</tr>
<tr>
<td>Index</td>
<td>Full scan of the INDEX</td>
</tr>
<tr>
<td>Full</td>
<td>Full scan of the table</td>
</tr>
</tbody>
</table>

### Extra

<table>
<thead>
<tr>
<th>Distinct</th>
<th>Stale after first row match</th>
</tr>
</thead>
<tbody>
<tr>
<td>Full scan on NULL key</td>
<td>No index lookup on subquery</td>
</tr>
<tr>
<td>Impossible WHERE noticed after reading consists</td>
<td>WHERE clauses always false</td>
</tr>
<tr>
<td>Tables</td>
<td>WHERE clauses always false</td>
</tr>
<tr>
<td>No tables</td>
<td>No FROM or FROM DUAL</td>
</tr>
<tr>
<td>Not exists</td>
<td>Stale after first row match for each row set from previous tables</td>
</tr>
<tr>
<td>Range checks for each record</td>
<td>No good index; there might be one after values from previous tables are known</td>
</tr>
</tbody>
</table>

### Syntax of for...in... WHERE...in... WHERE...in...

- **Using (...)**
  - filterset
  - index
  - index for group-by
  - intersection
  - union
  - index_merge
  - temporary
  - union
  - index_merge

### Pythian

- 1-877-PYTHIAN
- www.pythian.com
SHOWS optimizer transformation of query

- Can help you write better SQL
JSON format shows extended and partition information

```sql
mysql> EXPLAIN FORMAT=JSON
-> SELECT student.fname, student.lname, registration.signup_date
-> FROM student
-> INNER JOIN registration
-> ON student.student_id = registration.student_id
-> INNER JOIN class
-> ON registration.class_id = class.class_id
-> WHERE class.name = 'SQL TUNING'
-> AND registration.signup_date BETWEEN DATE_SUB(@BillDate, INTERVAL 1 DAY) and @BillDate
-> AND registration.cancelled = 'N'\G

EXPLAIN: {
  "query_block": {
    "select_id": 1,
    "nested_loop": {
      "table": {
        "table_name": "student",
        "access_type": "ALL",
        "possible_keys": [
          "PRIMARY",
          "pk_student"
        ],
        "rows": 8707,
        "filtered": 100
      }
    },
    "table": {
      "table_name": "registration",
      "access_type": "ref",
      "possible_keys": [
        "PRIMARY",
        "pk_registration"
      ],
      "key": "PRIMARY",
      "used_key_parts": [
        "student_id"
      ],
      "key_length": 4,
      "ref": {
        "table": "class",
        "student_id": "student.id"
      },
      "rows": 4,
      "filtered": 100,
      "attached_condition": "(("cosu".registration.cancelled = 'N')
                        and ("cosu".registration.signup_date
                            between coalesce((@BillDate) - INTERVAL 1 day) and (@'BillDate')))
```
OPTIMIZER_TRACE

- Stored in information_schema.optimizer_trace
- Default is 16k of Memory
- set optimizer_trace_max_mem_size=1000000;
- set optimizer_trace_features="greedy_search=OFF";
  - Not as verbose

```
mysql> set optimizer_trace="enabled=on";
Query OK, 0 rows affected (0.03 sec)

mysql> SELECT student.fname, student.lname, registration.signup_date -> FROM student
  -> INNER JOIN registration
  -> ON student.student_id = registration.student_id
  -> INNER JOIN class
  -> ON registration.class_id = class.class_id
  -> WHERE class.name = 'SQL TUNING'
  -> AND registration.signup_date
  -> BETWEEN DATE_SUB(@BillDate, INTERVAL 1 DAY) and @BillDate
  -> AND registration.cancelled = 'N'
...
11 rows in set (2 min 7.76 sec)

mysql> select * from information_schema.optimizer_trace
...
```

```
TRACE: {  
  "scope": {  
    "join_preparation": {  
      "transformations_to_nested_joins": {  
        "transformations": [  
          "JOIN_condition_to_WHERE",
          "condition_processing": {  
            "condition": "WHERE",
            "transformation": "equality_propagation",
            "transformation": "constant_propagation",
            "transformation": "trivial_condition_removal",
            "table_dependencies": [  
              "rows_estimation": {  
                "cable": "student",
                "cable_scan": {  
                  "rows": 8707,
                  "cost": 161
                }
              },
              "cable": "registration",
              "cable_scan": {  
                "rows": 78054,
                "cost": 228
              }
            ]
          }
        ]
      }
    }
  }
```

```
MISSING_BYTES_Beyond_MAX_MEM_SIZE: 9146
INSUFFICIENT_PRIVILEGES: 0
1 row in set (0.00 sec)

mysql> set optimizer_trace="enabled=off";
Query OK, 0 rows affected (0.00 sec)
```
```
SELECT student.fname, student.lname, registration.signup_date
FROM student
INNER JOIN registration
ON student.student_id = registration.student_id
INNER JOIN class
ON registration.class_id = class.class_id
WHERE class.name = 'SQL TUNING'
AND registration.signup_date BETWEEN DATE_SUB(@billiard, INTERVAL 1 YEAR)
AND registration.cancelled = 'N';
```
Examine the Execution Plan

- Find Expensive Operators
  - Examine IO costs, rows examined vs. rows sent
  - Look for full table or clustered index scans

- Review the Predicate Information
  - Know how parameters are being interpreted
    - Review the data types
    - Implicit conversions?
  - Know which step filtering predicate is applied

- Review Temporary and Filesort activities in Extra column
  - Try to fix them

```sql
SELECT registration.signup_date, class.name, count(*) num_students_per_class
FROM student
INNER JOIN registration
ON student.student_id = registration.student_id
INNER JOIN class
ON registration.class_id = class.class_id
WHERE class.name = 'SQL TUNING'
AND registration.signup_date BETWEEN DATE_SUB(@BillDate, INTERVAL 3 YEAR_MONTH) AND @BillDate
AND registration.cancelled = 'N'
GROUP BY registration.signup_date,
class.name
ORDER BY registration.signup_date, class.name, count(*) desc
```
3. Gather Object Information

- Table Definition
  - Is it really a table or is it a view?
  - Get size of tables
    - Quick way: `mysqlshow --status database {table} {column}`

```
mysqlshow --status csu
Database: csu

+-------------------------------+--------+-----------------+-----------------+-----------------+----------------+-----------------+----------------+
| Name      | Rows  | Avg_row_length | Data_length    | Max_data_length | Index_length   | Data_free       |
+-------------------------------+--------+-----------------+-----------------+-----------------+----------------+-----------------+----------------+
| class     | 1000  | 163            | 16384          | 0               | 16384          | 0               |
| registration | 80494 | 95             | 3686400        | 0               | 4227072        | 6291456         |
| student   | 9489  | 277            | 2637824        | 0               | 212992         | 4194304         |
+-------------------------------+--------+-----------------+-----------------+-----------------+----------------+-----------------+----------------+
```

- Examine Columns in Where Clause
  - Cardinality of columns / data skew
- Review the ‘Selected’ Columns
  - Use of ‘*’ or scalar functions
Review Indexes and Constraints

• Know Existing Indexes
  • Columns those indexes contain
  • If multi-column, know the left leading column
  • Cardinality

• Make sure the optimizer can use the index
  • Functions on indexed columns can turn off index
    • Consider rewriting
  • Look for implicit conversions
    • Get sample parameter values

• Review existing keys and constraints
  • Know Multi-Table Relationships (ERD)
    • Primary key and foreign definitions
  • Check and not null constraints

• Tip: Keys and constraints help create better execution plans
• No Relationships?
4. Find the Driving Table

• Need to know the size of the actual data sets of each step
  • In Joins (Right, Left, Outer)
  • What are the filtering predicates
  • When is each filtering predicate applied
    • Try to filter earlier rather than later

• Compare size of final result set with data examined
  • Find the driving table
    • To reduce rows examined
Who registered yesterday for SQL Tuning?

SELECT s.fname, s.lname, r.signup_date
FROM student s
INNER JOIN registration r
    ON s.student_id = r.student_id
INNER JOIN class c
    ON r.class_id = c.class_id
WHERE c.name  = 'SQL TUNING'
AND r.signup_date BETWEEN
    DATE_SUB('@BillDate', INTERVAL 1 DAY) and '@BillDate'
AND r.cancelled = 'N';
“SQL Tuning” by Dan Tow

- Great book that teaches SQL Diagramming
- http://www.singingsql.com

```sql
select count(*) from registration r
where r.signup_date BETWEEN DATE_SUB('@BillDate', INTERVAL 1 DAY) and '@BillDate' AND r.cancelled = 'N';
```

4,228 / 79,981 * 100 = 5.2%

```sql
select count(0) from class where name = 'SQL TUNING'
```

2 / 1,000 * 100 = .2%
ALTER TABLE registration ADD FOREIGN KEY (student_id) REFERENCES student(student_id);
ALTER TABLE registration ADD FOREIGN KEY (class_id) REFERENCES class(class_id);

SELECT table_name, 
    column_name,constraint_name, 
    referenced_table_name,referenced_column_name 
FROM information_schema.key_column_usage 
WHERE table_schema = 'csu' AND table_name = 'registration' 
AND referenced_column_name IS NOT NULL;

mysql> source cons_vw
+-----------------+------------+----------------+-----------------+-----------------+----------------+ 
| table_name      | column_name | constraint_name | referenced_table_name | referenced_column_name | 
| registration    | class_id    | registration_ibfk_2 | class            | class_id         | 
| registration    | student_id  | registration_ibfk_3 | student         | student_id       |
+-----------------+------------+----------------+-----------------+-----------------+----------------+ 
2 rows in set (0.01 sec)
Any Change?

F Keys added
CREATE INDEX class_nm ON class(name);
Did We Improve It?

Average Wait Time per Execution for SQL Student Billing | IMDB_SLAVE.3306
April 7, 2016 to April 8, 2016

- **SQL Name**: Student Billing
- **Wait Time**: 8 hours 45 minutes 10 seconds
- **Total Wait Time for Time Period**: 1 hour 37 minutes 53 seconds
- **% of Total Wait Time**: 6%
- **Average (seconds)**: 0.07253065

Executions:
- **Saturday - April 9: 3AM-4AM**
  - Executions: 434,437

Rows Affected or Sent:
- **Saturday - April 9: 3AM-4AM**
  - Rows Affected or Sent: 6,015,249

Rows Examined:
- **Saturday - April 9: 3AM-4AM**
  - Rows Examined: 12,899,372
Try to reduce Logical IO (rows examined)

Adding indexes is not always the right thing to do!
  - Consider Insert, Update, and Delete activity

Covering index
  CREATE INDEX cidx_registration ON registration(class_id, signup_date, cancelled);
  - Include all columns in
  - Reads only the index – doesn’t go back to table

Partial index
  ALTER TABLE class ADD INDEX (description(3));
  - description varchar(200)
  SELECT * FROM class WHERE description LIKE 'SQL%';
Create an index on a Virtual Column in MySQL 5.7.7

Kind of like a function index
Persist the data in an index NOT the table

```
SELECT signup_dayofweek, count(*)
FROM vregistration r
WHERE r.cancelled = 'N'
GROUP BY signup_dayofweek
HAVING count(*) < 10000;
```

```
ALTER TABLE vregistration ADD COLUMN signup_dayofweek tinyint
GENERATED ALWAYS AS (dayofweek(signup_date)) VIRTUAL;
CREATE INDEX signup_dayofweek ON vregistration(signup_dayofweek);
```

Consider other specialized index types

- Fulltext
- Spatial
- Hash for memory engine
- Beyond scope of this presentation
CREATE INDEX cidx_registration ON registration(class_id, signup_date, cancelled);
Is the Covering Index Working?

- Can’t cover it because of range

```
```

AND r.signup_date BETWEEN DATE_SUB('@BillDate', INTERVAL 1 DAY) and '@BillDate'
```
Did We Improve It?

Average Wait Time per Execution for SQL Student Billing - IMDB_SLAVE:3306 | April 15, 2016 to April 16, 2016

- **Covered Index added**

No improvement?
• Look for performance inhibitors
  • Cursor or row by row processing
  • Parallel query processing.
    • Not always bad, but…
    • Percona_Alex_Rubin_increasing-slow-query-performance-with-parallel-query-execution
• Hard-coded hints
  • logicalread.solarwinds.mysql-query-hints-improve-performance
• Nested views
  • Also, sub-queries
• Abuse of Wild Cards (*) or No Where Clause
• Code-based SQL Generators (e.g. PHP generator, LINQ; nHibernate)
• Implicit data conversions
• Non-SARG-able / scalar functions
  • Select… where upper(first_name) = ‘JANIS’
Data Types

- **Signup Date** is a DATE not TIMESTAMP.
- **Change between clause to ‘=’**

```
AND r.signup_date BETWEEN
    DATE_SUB('(@BillDate', INTERVAL 1 DAY) and '@BillDate'
----------
AND r.signup_date =
    DATE_SUB ('(@BillDate', INTERVAL ? DAY )
```
Fix the Query

```sql
SELECT s.fname, s.lname, r.signup_date
FROM student s
INNER JOIN registration r
  ON s.student_id = r.student_id
INNER JOIN class c
  ON r.class_id = c.class_id
WHERE c.name = 'SQL TUNING'
AND r.signup_date = DATE_SUB(@BillDate, INTERVAL 1 DAY)
AND r.cancelled = 'N';
```

**Visual Explain**

- **Access Type:** ref
- **Cost Hints:** Low-medium - Low if number of matching rows is small.
- **Key Index:** cidx_registration
- **Used Key Parts:** class_id, signup_date, cancelled
- **Possible Keys:** PRIMARY, pk_registration.

**Spans:**
- 2 rows
  - 1 row
  - 1 row

**Nodes:**
- Non-Unique Key Lookup
- Unique Key Lookup
Did Performance Improve?
Another Case Study – Sales Quarterly Contest

```
SELECT dayofweek (payment_date) AS day_of_week, staff.first_name, staff.last_name,
city.city, country.country, category.name AS category, SUM(payment.amount) AS total_sales
FROM payment
INNER JOIN rental ON payment.rental_id = rental.rental_id
INNER JOIN inventory ON rental.inventory_id = inventory.inventory_id
INNER JOIN film ON inventory.film_id = film.film_id
INNER JOIN film_category ON film.film_id = film_category.film_id
INNER JOIN category ON film_category.category_id = category.category_id
INNER JOIN store ON store.store_id = inventory.store_id
INNER JOIN address ON address.address_id = store.address_id
INNER JOIN city ON city.city_id = address.city_id
INNER JOIN country ON country.country_id = city.country_id
INNER JOIN staff ON staff.staff_id = rental.staff_id
WHERE quarter (rental_date) = ?
AND return_date IS NOT ?
GROUP BY dayofweek (payment_date), staff.first_name, staff.last_name, city.city, country.country, category.name
ORDER BY total_sales DESC
```
### Review the Execution Plan

#### Explain

<table>
<thead>
<tr>
<th>select_type</th>
<th>table</th>
<th>type</th>
<th>possible_keys</th>
<th>key</th>
</tr>
</thead>
<tbody>
<tr>
<td>SIMPLE</td>
<td>store</td>
<td>index</td>
<td>PRIMARY,idx_fk_address_id</td>
<td>idx_fk_address_id</td>
</tr>
<tr>
<td>SIMPLE</td>
<td>address</td>
<td>eq_ref</td>
<td>PRIMARY,idx_fk_city_id</td>
<td>PRIMARY</td>
</tr>
<tr>
<td>SIMPLE</td>
<td>city</td>
<td>eq_ref</td>
<td>PRIMARY,idx_fk_country_id</td>
<td>PRIMARY</td>
</tr>
<tr>
<td>SIMPLE</td>
<td>country</td>
<td>eq_ref</td>
<td>PRIMARY</td>
<td>PRIMARY</td>
</tr>
<tr>
<td>SIMPLE</td>
<td>payment</td>
<td>ALL</td>
<td>fk_payment_rental</td>
<td>NULL</td>
</tr>
<tr>
<td>SIMPLE</td>
<td>rental</td>
<td>eq_ref</td>
<td>PRIMARY,idx_fk_inventory_id,idx_fk_staff_id</td>
<td>PRIMARY</td>
</tr>
<tr>
<td>SIMPLE</td>
<td>staff</td>
<td>eq_ref</td>
<td>PRIMARY</td>
<td>PRIMARY</td>
</tr>
<tr>
<td>SIMPLE</td>
<td>inventory</td>
<td>eq_ref</td>
<td>PRIMARY,idx_fk_film_id,idx_store_id_film_id</td>
<td>PRIMARY</td>
</tr>
<tr>
<td>SIMPLE</td>
<td>film</td>
<td>eq_ref</td>
<td>PRIMARY</td>
<td>PRIMARY</td>
</tr>
<tr>
<td>SIMPLE</td>
<td>film_category</td>
<td>ref</td>
<td>PRIMARY,idx_fk_film_category_category</td>
<td>PRIMARY</td>
</tr>
<tr>
<td>SIMPLE</td>
<td>category</td>
<td>eq_ref</td>
<td>PRIMARY</td>
<td>PRIMARY</td>
</tr>
</tbody>
</table>

#### Explain Extended
- Explain FORMAT=JSON
- Optimizer Trace 5.6.3+
- MySQL Workbench

---

**Tip:** Try to avoid using table aliases—they don’t translate in plan.

<table>
<thead>
<tr>
<th>ref</th>
<th>rows</th>
<th>filtered</th>
<th>Extra</th>
</tr>
</thead>
<tbody>
<tr>
<td>NULL</td>
<td>2</td>
<td>100.00</td>
<td>Using index; Using temporary; Using filesort</td>
</tr>
<tr>
<td>store.address_id</td>
<td>1</td>
<td>100.00</td>
<td>NULL</td>
</tr>
<tr>
<td>address.city_id</td>
<td>1</td>
<td>100.00</td>
<td>NULL</td>
</tr>
<tr>
<td>city.country_id</td>
<td>1</td>
<td>100.00</td>
<td>NULL</td>
</tr>
<tr>
<td>NULL</td>
<td>532160</td>
<td>100.00</td>
<td>Using where; Using join buffer (Block Nested Loop)</td>
</tr>
<tr>
<td>payment.rental_id</td>
<td>1</td>
<td>90.00</td>
<td>Using where</td>
</tr>
<tr>
<td>rental.staff_id</td>
<td>1</td>
<td>100.00</td>
<td>NULL</td>
</tr>
<tr>
<td>rental.inventory_id</td>
<td>1</td>
<td>50.00</td>
<td>Using where</td>
</tr>
<tr>
<td>inventory.film_id</td>
<td>1</td>
<td>100.00</td>
<td>Using index</td>
</tr>
<tr>
<td>inventory.film_id</td>
<td>2</td>
<td>100.00</td>
<td>Using index</td>
</tr>
<tr>
<td>film_category.category_id</td>
<td>1</td>
<td>100.00</td>
<td>NULL</td>
</tr>
</tbody>
</table>
FILTERING PREDICATES:

```sql
select avg(cnt) from (select quarter(rental_date), count(*) cnt
                         from rental
                         where return_date is not null group by quarter(rental_date)) a;

select 125557.5000 / 8918218 * 100 = 1.4%
```

JOIN RESULTS EXAMPLE FOR CITY TO ADDRESS:

```sql
select avg(cnt) from (select city_id, count(*) cnt
                         from address group by city_id) a;
```

64 rows
Drive the Query by Rental

- Create virtual column on rental_date

```
CREATE INDEX rental_qtr_idx ON rental(quarter_date, return_date);
```
Driving by Payment – Not by Rental

<table>
<thead>
<tr>
<th>select_type</th>
<th>table</th>
<th>type</th>
<th>possible_keys</th>
<th>key</th>
<th>key_len</th>
<th>rows</th>
</tr>
</thead>
<tbody>
<tr>
<td>SIMPLE</td>
<td>payment</td>
<td>ALL</td>
<td>fk_payment_rental, pay_idx</td>
<td>NULL</td>
<td>NULL</td>
<td>332160</td>
</tr>
<tr>
<td>SIMPLE</td>
<td>rental</td>
<td>eq_ref</td>
<td>PRIMARY, idx_fk_inventory_id, idx_fk_staff_id, rental_qtr_idx</td>
<td>PRIMARY</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>SIMPLE</td>
<td>store</td>
<td>index</td>
<td>PRIMARY, idx_fk_address_id</td>
<td>idx_fk_address_id</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>SIMPLE</td>
<td>staff</td>
<td>eq_ref</td>
<td>PRIMARY</td>
<td>PRIMARY</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>SIMPLE</td>
<td>inventory</td>
<td>eq_ref</td>
<td>PRIMARY, idx_fk_film_id, idx_store_id_film_id</td>
<td>PRIMARY</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>SIMPLE</td>
<td>film</td>
<td>eq_ref</td>
<td>PRIMARY</td>
<td>PRIMARY</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>SIMPLE</td>
<td>address</td>
<td>eq_ref</td>
<td>PRIMARY, idx_fk_city_id</td>
<td>PRIMARY</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>SIMPLE</td>
<td>city</td>
<td>eq_ref</td>
<td>PRIMARY, idx_fk_country_id</td>
<td>PRIMARY</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>SIMPLE</td>
<td>country</td>
<td>eq_ref</td>
<td>PRIMARY</td>
<td>PRIMARY</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>SIMPLE</td>
<td>film_category</td>
<td>ref</td>
<td>PRIMARY, fk_film_category_category</td>
<td>PRIMARY</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>SIMPLE</td>
<td>category</td>
<td>eq_ref</td>
<td>PRIMARY</td>
<td>PRIMARY</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
ALTER TABLE payment ADD COLUMN payment_dayofweek TINYINT GENERATED ALWAYS AS (dayofweek(payment_date)) VIRTUAL;

CREATE INDEX pay_idx ON payment(rental_id, payment_dayofweek, amount);

SELECT `payment_dayofweek`, day_of_week, staff.first_name, staff.last_name, city.city, country.country, category.name AS category, SUM(payment.amount) AS total_sales
FROM payment
INNER JOIN rental ON payment.rental_id = rental.rental_id
INNER JOIN inventory ON rental.inventory_id = inventory.inventory_id
INNER JOIN film ON inventory.film_id = film.film_id
INNER JOIN film_category ON film.film_id = film_category.film_id
INNER JOIN category ON film_category.category_id = category.category_id
INNER JOIN store ON store.store_id = inventory.store_id
INNER JOIN address ON address.address_id = store.address_id
INNER JOIN city ON city.city_id = address.city_id
INNER JOIN country ON country.country_id = city.country_id
INNER JOIN staff ON staff.staff_id = rental.staff_id
WHERE quarter(rental_date) = ?
AND return_date IS NOT ?
GROUP BY `payment_dayofweek`, staff.first_name, staff.last_name, city.city, country.country, category.name
ORDER BY total_sales DESC
### Still Driving by Payment Using Index

- Can’t use virtual column on Rental

<table>
<thead>
<tr>
<th>select_type</th>
<th>table</th>
<th>type</th>
<th>possible_keys</th>
<th>key</th>
<th>key_len</th>
<th>rows</th>
</tr>
</thead>
<tbody>
<tr>
<td>SIMPLE</td>
<td>payment</td>
<td>index</td>
<td>fk_payment_rental,pay_idx</td>
<td>pay_idx</td>
<td>12</td>
<td>332160</td>
</tr>
<tr>
<td>SIMPLE</td>
<td>rental</td>
<td>eq_ref</td>
<td>PRIMARY,idx_fk_inventory_id,idx_fk_staff_id,rental_qtr_idx</td>
<td>PRIMARY</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>SIMPLE</td>
<td>store</td>
<td>index</td>
<td>PRIMARY,idx_fk_address_id</td>
<td>idx_fk_address_id</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>SIMPLE</td>
<td>staff</td>
<td>eq_ref</td>
<td>PRIMARY</td>
<td>PRIMARY</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>SIMPLE</td>
<td>inventory</td>
<td>eq_ref</td>
<td>PRIMARY,idx_fk_film_id,idx_store_id_film_id</td>
<td>PRIMARY</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>SIMPLE</td>
<td>film</td>
<td>eq_ref</td>
<td>PRIMARY</td>
<td>PRIMARY</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>SIMPLE</td>
<td>address</td>
<td>eq_ref</td>
<td>PRIMARY,idx_fk_city_id</td>
<td>PRIMARY</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>SIMPLE</td>
<td>city</td>
<td>eq_ref</td>
<td>PRIMARY,idx_fk_country_id</td>
<td>PRIMARY</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>SIMPLE</td>
<td>country</td>
<td>eq_ref</td>
<td>PRIMARY</td>
<td>PRIMARY</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>SIMPLE</td>
<td>film_category</td>
<td>ref</td>
<td>PRIMARY,fk_film_category_category</td>
<td>PRIMARY</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>SIMPLE</td>
<td>category</td>
<td>eq_ref</td>
<td>PRIMARY</td>
<td>PRIMARY</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
select day_of_week, sales.first_name, sales.last_name, sales.city, sales.country, sales.category, sum(pay.amount) total_sales
from
  (select rental.rental_id, rental.staff_id, staff.first_name, staff.last_name, city.city, country.country, category.name category
   from rental
   INNER JOIN inventory ON rental.inventory_id = inventory.inventory_id
   INNER JOIN film ON inventory.film_id = film.film_id
   INNER JOIN film_category ON film.film_id = film_category.film_id
   INNER JOIN category ON film_category.category_id = category.category_id
   INNER JOIN store ON store.store_id = inventory.store_id
   INNER JOIN address ON address.address_id = store.address_id
   INNER JOIN city ON city.city_id = address.city_id
   INNER JOIN country ON country.country_id = city.country_id
   INNER JOIN staff ON staff.staff_id = rental.staff_id
   WHERE quarter_date = 1
   AND return_date is not null) sales
inner join
  (select payment_dayofweek day_of_week, rental_id, staff_id, payment.amount from payment) pay
on pay.rental_id = sales.rental_id and pay.staff_id = sales.staff_id
group by day_of_week, sales.first_name, sales.last_name, sales.city, sales.country, sales.category
order by day_of_week, total_sales desc;
### Query Uses Both Virtual Columns and Indexes

<table>
<thead>
<tr>
<th>select_type</th>
<th>table</th>
<th>type</th>
<th>possible_keys</th>
<th>key</th>
<th>key_len</th>
<th>rows</th>
</tr>
</thead>
<tbody>
<tr>
<td>SIMPLE</td>
<td>rental</td>
<td>range</td>
<td>PRIMARY,idx_fk_inventory_id,idx_fk_staff_id,rental_qtr_idx</td>
<td>rental_qtr_idx</td>
<td>8</td>
<td>106952</td>
</tr>
<tr>
<td>SIMPLE</td>
<td>payment</td>
<td>ref</td>
<td>idx_fk_staff_id,idx_payment_rental,pay_idx</td>
<td>pay_idx</td>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>SIMPLE</td>
<td>score</td>
<td>index</td>
<td>PRIMARY,idx_fk_address_id</td>
<td>idx_fk_address_id</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>SIMPLE</td>
<td>staff</td>
<td>eq_ref</td>
<td>PRIMARY</td>
<td>idx_fk_address_id</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>SIMPLE</td>
<td>inventory</td>
<td>eq_ref</td>
<td>PRIMARY,idx_fk_film_id,idx_store_id_film_id</td>
<td>PRIMARY</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>SIMPLE</td>
<td>film</td>
<td>eq_ref</td>
<td>PRIMARY</td>
<td>PRIMARY</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>SIMPLE</td>
<td>address</td>
<td>eq_ref</td>
<td>PRIMARY,idx_fk_city_id</td>
<td>PRIMARY</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>SIMPLE</td>
<td>city</td>
<td>eq_ref</td>
<td>PRIMARY,idx_fk_country_id</td>
<td>PRIMARY</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>SIMPLE</td>
<td>country</td>
<td>eq_ref</td>
<td>PRIMARY</td>
<td>PRIMARY</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>SIMPLE</td>
<td>film_category</td>
<td>ref</td>
<td>PRIMARY,idx_fk_film_category_category</td>
<td>PRIMARY</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>SIMPLE</td>
<td>category</td>
<td>eq_ref</td>
<td>PRIMARY</td>
<td>PRIMARY</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

- Using where; Using MRR; Using temporary; Using filesort
- Multi-Range Read Optimization
  - Using a range scan on a secondary index can result in many random disk accesses
    - when the table is large and not stored in the storage engine's cache.
  - MySQL tries to reduce random disk access for range scans by first scanning the index collecting the keys for relevant rows.
Sloppy Coding – Need to Fix Query

Example SQL Text:

```sql
SELECT CONCAT(customer.last_name, ' ', customer.first_name) AS customer,
       address.phone,
       film.title,
       rental_date
FROM rental
INNER JOIN customer
ON rental.customer_id = customer.customer_id
INNER JOIN address
ON customer.address_id = address.address_id
INNER JOIN inventory
ON rental.inventory_id = inventory.inventory_id
INNER JOIN film
ON inventory.film_id = film.film_id
WHERE rental.return_date IS NULL
AND rental_date + INTERVAL film.rental_duration DAY > CURRENT_DATE() - 30
AND customer.last_name like 'griffin%'
```
Changed the Code: last_name like ‘griffin%’
There are a lot of challenges in Query Tuning
If you remember the Top 5 Tips, they should take you a long way

1. Monitor Wait Time
   - Look at wait events, record baseline metrics

2. Review the Execution Plan
   - Look for expensive steps, know what’s optimizer features are supporting the plan

3. Gather Object Information
   - For expensive objects, know what the optimizer knows

4. Find the Driving Table
   - Consider SQL Diagramming techniques

5. Engineer out the Stupid
• Try Database Performance Analyzer FREE for 14 days
• Improve root cause of slow performance
  • Quickly identify root cause of issues that impact end-user response time
  • See historical trends over days, months, and years
  • Understand impact of VMware® performance
  • Agentless architecture with no dependence on Oracle® Packs; installs in minutes

www.solarwinds.com/dpa-download/
The SolarWinds, SolarWinds & Design, Orion, and THWACK trademarks are the exclusive property of SolarWinds Worldwide, LLC or its affiliates, are registered with the U.S. Patent and Trademark Office, and may be registered or pending registration in other countries. All other SolarWinds trademarks, service marks, and logos may be common law marks or are registered or pending registration. All other trademarks mentioned herein are used for identification purposes only and are trademarks of (and may be registered trademarks) of their respective companies.