Overview

SQL Optimizations
- Early NULL filtering
- Index condition Pushdown

YourSQL

Conclusions
Overview

SQL Optimizations
- Early NULL filtering
- Index condition Pushdown

YourSQL

Conclusions
SQL Optimizations

In relational tables, different search strategies work for different data.

- Let’s take the following table:
  \[
  T(\text{ID} \text{ integer primary key,} \\
  \text{Name} \text{ varchar}, \text{Indexed} \\
  \text{Birthday} \text{ date});
  \]

- Consider the following queries:
  1. Find Bob’s birthday
     \[\text{Pretty fast, Single lookup}\]
  2. Find birthdays of people with name like “Z%”
     \[\text{Still fast, Range scan}\]
  3. Find birthdays of people with names starting from M-Z
     \[\text{Lots of accesses, Probably Random seeks}\]

SQL Optimizations

- The search strategies become even more important in join queries.

- Supported types of joins in MySQL 5.7:
  - system
  - const
  - `eq_ref`
  - ref
  - fulltext
  - ref_or_null
  - index_merge
  - range

  - `eq_ref`
    One row is read from this table for each combination of rows from the previous tables. Other than the system and const types, this is the best possible join type. It is used when all parts of an index are used by the join and the index is a PRIMARY KEY OR UNIQUE NOT NULL index.

  - `ref`
    All rows with matching index values are read from this table for each combination of rows from the previous tables. ref is used if the join uses only a leftmost prefix of the key or if the key is not a PRIMARY KEY OR UNIQUE INDEX (in other words, if the join cannot select a single row based on the key value). If the key that is used matches only a few rows, this is a good join type.

Source: http://dev.mysql.com/doc/refman/5.7/en/explain-output.html#explain_type
SQL Optimizations

- Consider the query:

```sql
SELECT table1.pk, table2.col_int_key, table1.col_int_key, table2.pk
FROM C table1 JOIN C table2
    ON table2.`col_int_key`  = table1.`col_int_key`
WHERE table2.`pk`;
```

- Table 1 is accessed first, then Table 2 with ‘ref’.

- Optimization: If a row of table1 has a NULL in table1.col_int_key, no access to table2 is required. → Early NULLs filtering

This is seen in EXPLAIN:

<table>
<thead>
<tr>
<th>id</th>
<th>select_type</th>
<th>table</th>
<th>type</th>
<th>possible_keys</th>
<th>key</th>
<th>key_len</th>
<th>ref</th>
<th>rows</th>
<th>filtered</th>
<th>Extra</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>SIMPLE</td>
<td>table1</td>
<td>ALL</td>
<td>col_int_key</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
<td>20</td>
<td>100.00</td>
<td>Using where</td>
</tr>
<tr>
<td>1</td>
<td>SIMPLE</td>
<td>table2</td>
<td>ref</td>
<td>col_int_key</td>
<td>col_int_key</td>
<td>5</td>
<td>test.table1.col_int_key</td>
<td>100.00</td>
<td>Using where</td>
<td></td>
</tr>
</tbody>
</table>

Source:https://bugs.mysql.com/bug.php?id=53793
SQL Optimizations

- Consider the query:
  ```sql
  SELECT table1.pk, table2.col_int_key, table1.col_int_key, table2.pk
  FROM C table1 JOIN C table2
  ON table2.`col_int_key` = table1.`col_int_key`
  WHERE table2.`pk`;
  ```

- Table 1 is accessed first, then Table 2 with 'ref'.

- Optimization: If a row of table1 has a NULL in table1.col_int_key, no access to table2 is required. → Early NULLs filtering

This is seen in EXPLAIN:

<table>
<thead>
<tr>
<th>id</th>
<th>select_type</th>
<th>table</th>
<th>type</th>
<th>possible_keys</th>
<th>key</th>
<th>key_len</th>
<th>ref</th>
<th>rows</th>
<th>filtered</th>
<th>Extra</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>SIMPLE</td>
<td>table1</td>
<td>ALL</td>
<td>col_int_key</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
<td>20</td>
<td>100.00</td>
<td>Using where</td>
</tr>
<tr>
<td>1</td>
<td>SIMPLE</td>
<td>table2</td>
<td>ref</td>
<td>col_int_key</td>
<td>col_int_key</td>
<td>5</td>
<td>test.table1.col_int_key 3</td>
<td>100.00</td>
<td>Using where</td>
<td></td>
</tr>
</tbody>
</table>

Source: https://bugs.mysql.com/bug.php?id=53793
Optimizing even more...

- Can we do more such filtering? Yes, in form of Index Condition Pushdown (ICP).

- Without ICP → the storage engine traverses the index to locate rows in the base table and returns them to the MySQL server which evaluates the WHERE condition for the rows.

- With ICP → if parts of the WHERE condition can be evaluated by using only fields from the index, the MySQL server pushes this part of the WHERE condition down to the storage engine.

- ICP requires preindexing.
How good are these optimizations?

Query 2: TPC-H Q2.

```sql
SELECT s.acctbal, s.name, n.name, p.partkey, p.mfgr, s.address, s.phone, s_comment
FROM part, supplier, partsupp, nation, region
WHERE p.partkey = ps.partkey
  AND s.suppkey = ps.suppkey
  AND p.size = 15 AND p_type LIKE 'BRASS'
  AND s.nationkey = n.nationkey
  AND n.regionkey = r.regionkey
  AND r.name = 'EUROPE'
  AND (SELECT MIN(ps_supplycost)
          FROM partsupp, supplier, nation, region
          WHERE p.partkey = ps.partkey
            AND s.suppkey = ps.suppkey
            AND s.nationkey = n.nationkey
            AND n.regionkey = r.regionkey
            AND r.name = 'EUROPE')
ORDER BY s.acctbal DESC, n.name, s.name, p.partkey LIMIT 100;
```

<table>
<thead>
<tr>
<th>Join</th>
<th>Table</th>
<th>Access method</th>
<th>Key</th>
<th># of read requests</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>MySQL without ICP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>region</td>
<td>all</td>
<td>null</td>
<td>16</td>
</tr>
<tr>
<td>2</td>
<td>nation</td>
<td>ref</td>
<td>nation_fk</td>
<td>13</td>
</tr>
<tr>
<td>3</td>
<td>supplier</td>
<td>ref</td>
<td>supplier_fk</td>
<td>36,867</td>
</tr>
<tr>
<td>4</td>
<td>partsupp</td>
<td>ref</td>
<td>partsupp_fk</td>
<td>2,842,639</td>
</tr>
<tr>
<td>5</td>
<td>part</td>
<td>eq_ref</td>
<td>pk</td>
<td>651,525</td>
</tr>
<tr>
<td>Total</td>
<td></td>
<td></td>
<td></td>
<td>3,531,060</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Join</th>
<th>Table</th>
<th>Access method</th>
<th>Key</th>
<th># of read requests</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>MySQL with ICP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>part</td>
<td>ref</td>
<td>p.size &amp; p_type</td>
<td>245</td>
</tr>
<tr>
<td>2</td>
<td>partsupp</td>
<td>ref</td>
<td>pk</td>
<td>98,520</td>
</tr>
<tr>
<td>3</td>
<td>supplier</td>
<td>eq_ref</td>
<td>pk</td>
<td>45,679</td>
</tr>
<tr>
<td>4</td>
<td>nation</td>
<td>eq_ref</td>
<td>pk</td>
<td>5</td>
</tr>
<tr>
<td>5</td>
<td>region</td>
<td>all</td>
<td>null</td>
<td>4</td>
</tr>
<tr>
<td>Total</td>
<td></td>
<td></td>
<td></td>
<td>144,453</td>
</tr>
</tbody>
</table>
A one-line summary of paper: “We argue that early filtering may well take place at the earliest point possible—within a storage device.”
Overview

SQL Optimizations
- Early NULL filtering
- Index condition Pushdown

YourSQL

Conclusions
YourSQL

Components of the system:

- **ISC tasks.** User-programmable SSDs employed in YourSQL allow complex query operations to be offloaded in the form of ISC task.
- **Query planner optimized for ISC.**
- **Reorganized datapath for ISC.**

![Diagram](image-url)

**Figure 1:** Two database system architectures. (a) Traditional system. (b) YourSQL.
YourSQL - How is it different?

1. Selecting the target table

- In MySQL, no indexes, no ICP.

- “MySQL’s overly simplified target selection criteria (simply choose a table with preindexed filter columns), do not always guarantee I/O reduction since the amount of I/O reduction depends on filtering ratio.”

- In YourSQL, filtering depends on:
  - *limiting score*: a heuristic
  - *estimated filtering ratio*: fraction of relevant pages in one page set.

Figure 3: Selection of the early filtering target table.
YourSQL - Filtering Condition Pushdown

2. Reading from the table

- ISC filters evaluate the pushed conditions against the target table and store *match hints*.
- *Match hints* are a byte array whose element is set to one if the corresponding page satisfies filtering conditions.

![Figure 4: ISC filters.](image-url)
Figure 5: Execution timing diagram in YourSQL.  
Figure 6: Dell R720 server with PM1725 SSD.
YourSQL - Evaluation

TPC-H

Figure 8: TPC-H results.
YourSQL - Evaluation
Vs. MySQL

Table 6: Different levels of optimization.

<table>
<thead>
<tr>
<th>Scheme</th>
<th>Configuration</th>
</tr>
</thead>
<tbody>
<tr>
<td>Opt-P</td>
<td>Hardware filter</td>
</tr>
<tr>
<td>Opt-PS</td>
<td>Hardware filter + Software filter</td>
</tr>
<tr>
<td>Opt-PSH</td>
<td>Hardware filter + Software filter + HABP</td>
</tr>
</tbody>
</table>

Figure 9: Speed-ups of the top five accelerated queries with different optimization schemes.
Conclusions

(Comments)
1. Near data processing
   - Reaffirms the fundamental idiom:  
     “When operating on large datasets, do not move data from disk unless absolutely necessary.”
   - Can speed up some queries, but not all of them.
   - Requires specialized hardware.

2. I think it’s interesting, but too specialized.
   - What about unstructured data?
   - SSDs are still somewhat expensive.
Thank You :}