Query Optimization With MySQL 8.0 and MariaDB 10.3: The Basics

Jaime Crespo

Percona Live Europe 2018

-Frankfurt, 5 Nov 2018-

dbahire.com/pleu18

Agenda

1. Introduction
2. Query Profiling
3. Access Types and Basic Indexing Techniques
4. Multi-Column Indexing
5. Break
6. Joins
7. Subqueries
8. FULLTEXT, Geodata Search and JSON support
9. Conclusions

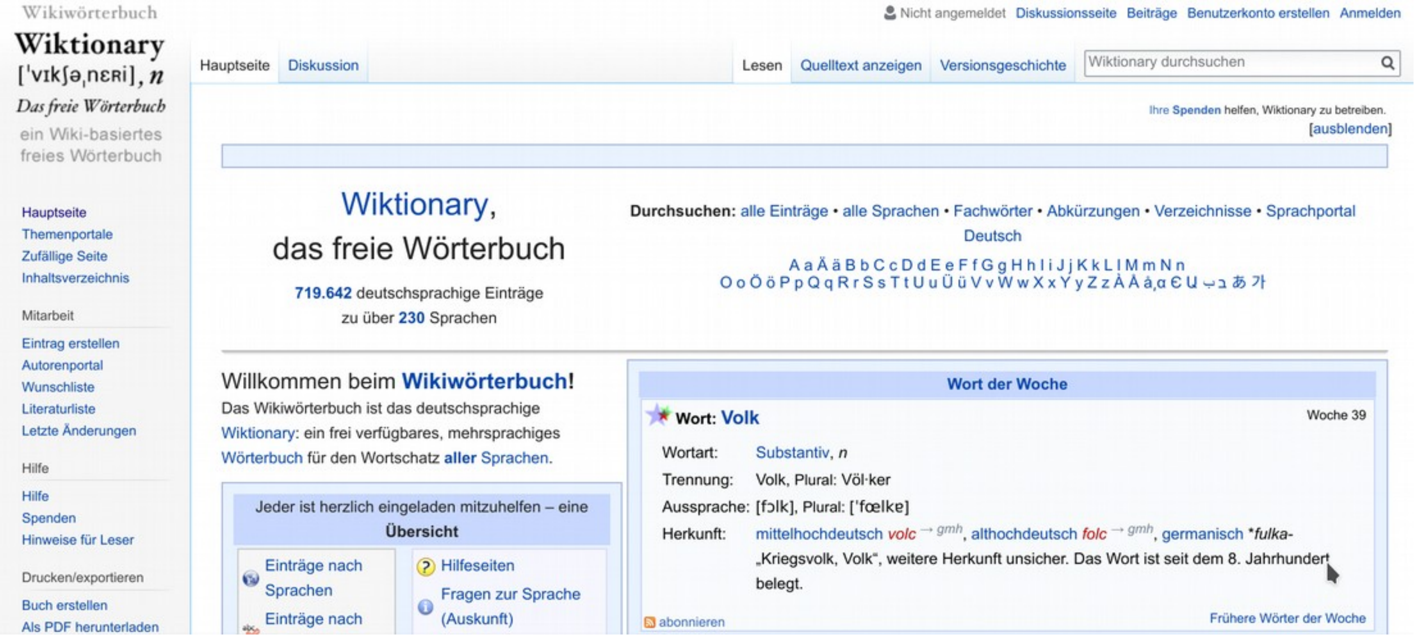
Query Optimization with MySQL 8.0 and MariaDB 10.3: The Basics

INTRODUCTION

This is me fighting bad query performance

- Sr. Database Administrator at Wikimedia Foundation
- Used to work as a trainer for Oracle (MySQL), as a Consultant (Percona) and as a Freelance administrator (DBAHire.com)

Example Application (I)



Ihre Spenden helfen, Wiktionary zu betreiben.
[ausblenden]

Wiktionary,
das freie Wörterbuch

719.642 deutschsprachige Einträge
zu über 230 Sprachen

Durchsuchen: alle Einträge • alle Sprachen • Fachwörter • Abkürzungen • Verzeichnisse • Sprachportal Deutsch

A a Ä ä B b C c D d E e F f G g H h I i J j K k L l M m N n
O o Ö ö P p Q q R r S s T t U u Ü ü V v W w X x Y y Z z À Å å‚α Є Ա ب ב あ 가

Willkommen beim **Wikiwörterbuch!**

Das Wikiwörterbuch ist das deutschsprachige Wiktionary: ein frei verfügbares, mehrsprachiges Wörterbuch für den Wortschatz **aller** Sprachen.

Jeder ist herzlich eingeladen mitzuhelfen – eine
Übersicht

Einträge nach Sprachen
Einträge nach

Hilfeseiten
Fragen zur Sprache (Auskunft)

Wort der Woche

Wort: Volk

Woche 39

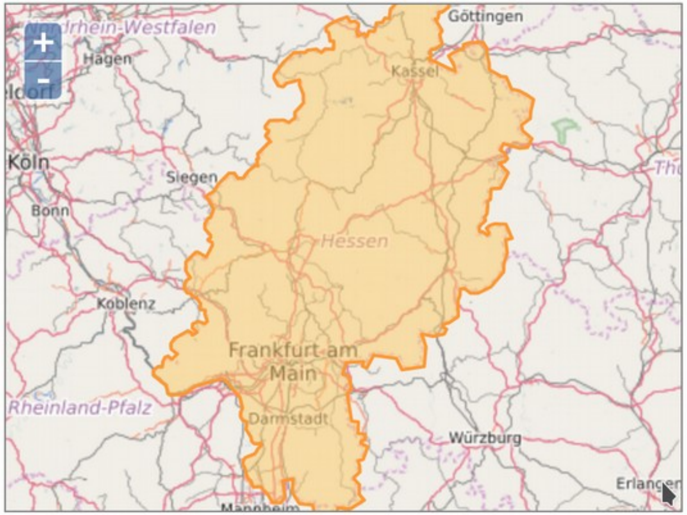
Wortart:	Substantiv, *n*
Trennung:	Volk, Plural: Völ·ker
Aussprache:	[fɔlk], Plural: ['fœlkɐ]
Herkunft:	mittelhochdeutsch *volc* → gmh, althochdeutsch *folc* → gmh, germanisch **fulka-* „Kriegsvolk, Volk“, weitere Herkunft unsicher. Das Wort ist seit dem 8. Jahrhundert belegt.

abonnieren

Frühere Wörter der Woche

- Wiktionary (and all Wikimedia projects’ data) is licensed under the Creative Commons BY-SA-3.0 License and is Copyright its Contributors

Example Application (II)



- OSM Database is licensed under the Open DataBase License and is Copyright OpenStreetMap Contributors

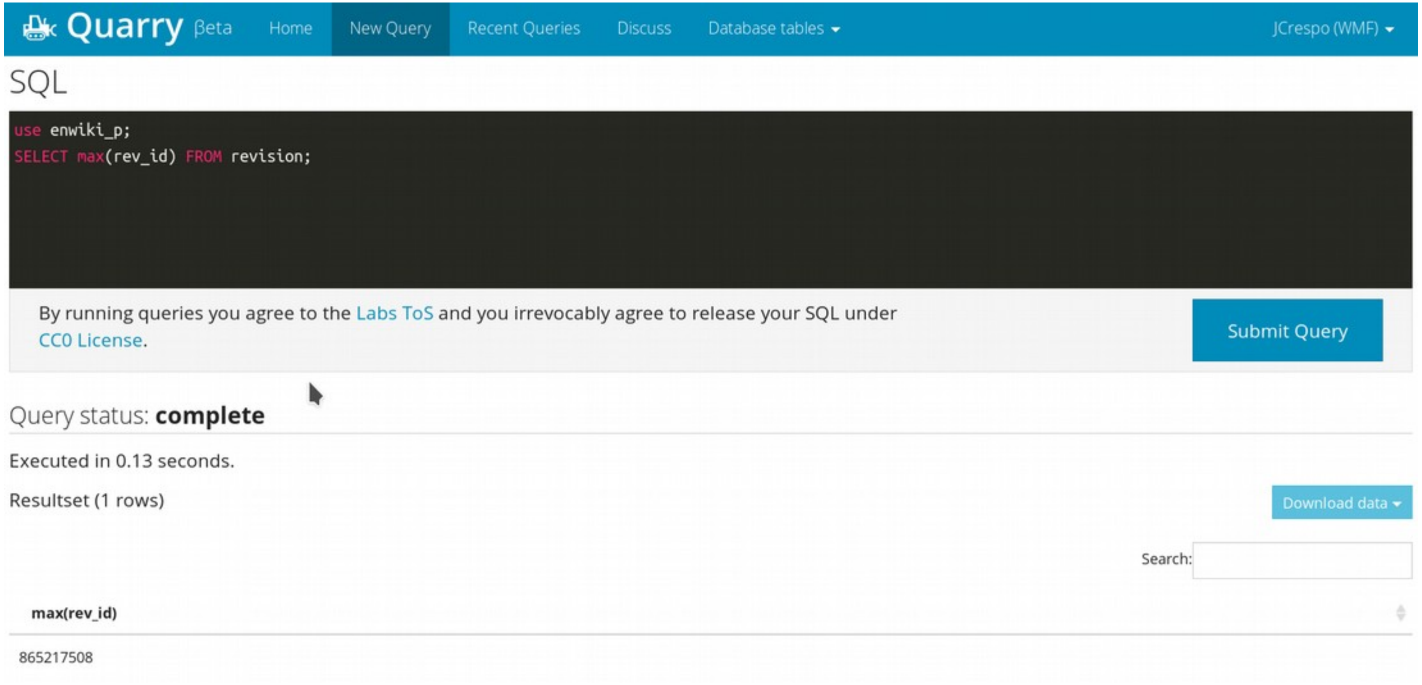
Install the example databases

- Downloads and instructions at:
  http://dbahire.com/pleu18
  - Requirements: a MySQL 8.0 or MariaDB 10.3 installation
  - The Wiktionary and OSM extracts
- Import them by doing:

```
$ tar xf <file> && cd <dir> && myloader -d .
```

Alternative: Query the Live Database

- https://quarry.wmflabs.org/



```
use enwiki_p;
SELECT max(rev_id) FROM revision;
```

By running queries you agree to the Labs ToS and you irrevocably agree to release your SQL under CC0 License.

Query status: **complete**

Executed in 0.13 seconds.

Resultset (1 rows)

```
max(rev_id)
865217508
```

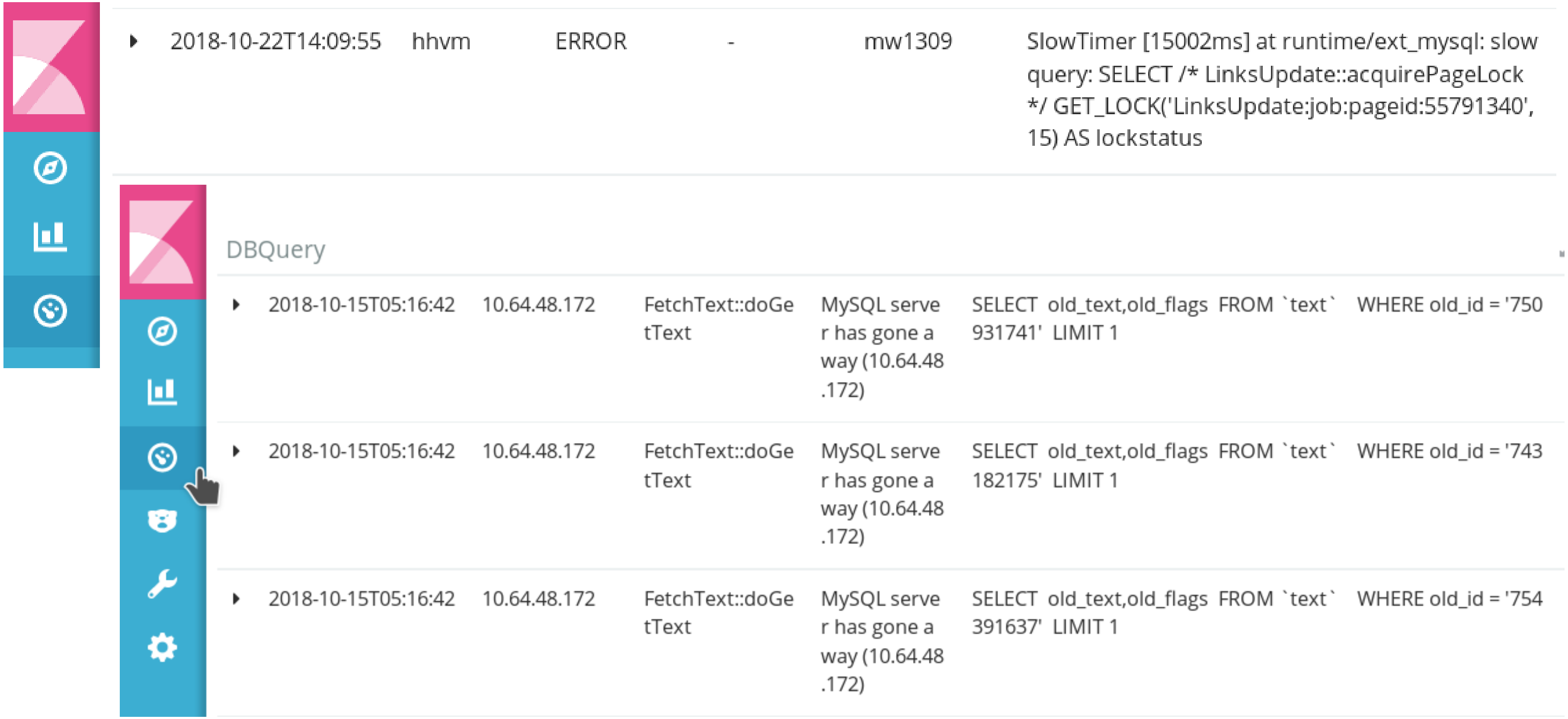
Query Optimization with MySQL 8.0 and MariaDB 10.3: The Basics

QUERY PROFILING

Which Queries Should I Optimize First?

- Tools:
  - Application-level profiling
  - General monitoring metrics (CPU, IO, etc.)
  - Aggregated PROCESSLIST
  - pt-query-digest
  - PERFORMANCE_SCHEMA
- They are not exclusive to each other

Application-Level profiling

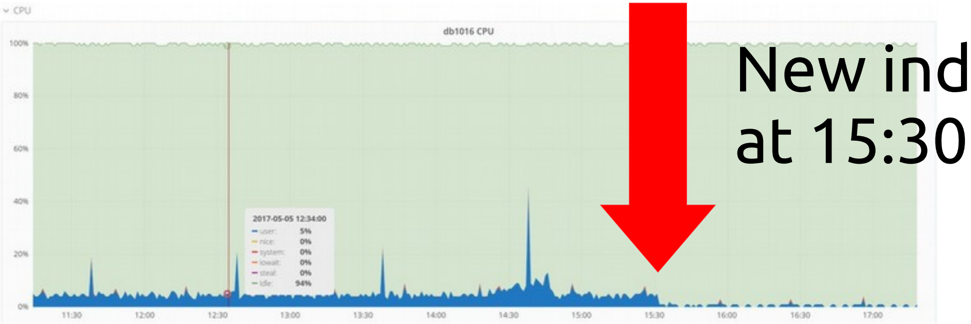


```
2018-10-22T14:09:55   hhvm   ERROR   -   mw1309   SlowTimer [15002ms] at runtime/ext_mysql: slow
                                                  query: SELECT /* LinksUpdate::acquirePageLock
                                                  */ GET_LOCK('LinksUpdate:job:pageid:55791340',
                                                  15) AS lockstatus
```

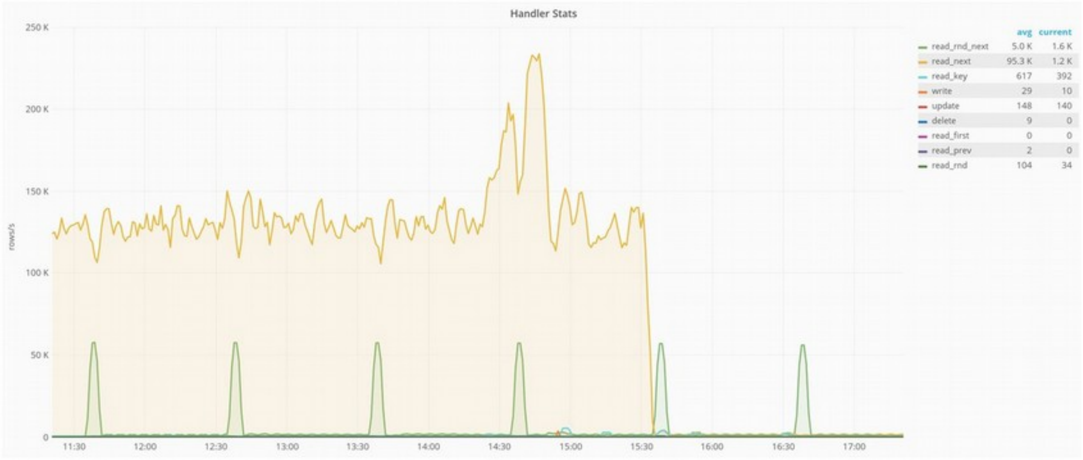
DBQuery

```
2018-10-15T05:16:42   10.64.48.172   FetchText::doGe   MySQL serve   SELECT  old_text,old_flags  FROM `text`   WHERE old_id = '750
                                     tText             r has gone a  931741'  LIMIT 1
                                                       way (10.64.48
                                                       .172)
2018-10-15T05:16:42   10.64.48.172   FetchText::doGe   MySQL serve   SELECT  old_text,old_flags  FROM `text`   WHERE old_id = '743
                                     tText             r has gone a  182175'  LIMIT 1
                                                       way (10.64.48
                                                       .172)
2018-10-15T05:16:42   10.64.48.172   FetchText::doGe   MySQL serve   SELECT  old_text,old_flags  FROM `text`   WHERE old_id = '754
                                     tText             r has gone a  391637'  LIMIT 1
                                                       way (10.64.48
                                                       .172)
```

General Monitoring Metrics



New index added at 15:30h



pt-query-digest

- It is a 3rd party tool written in Perl, originally created by Baron Schwartz
- It requires activation of the slow log:
  - `SET GLOBAL slow_query_log = 1;`
  - `SET long_query_time = 0;`
- In Percona Server and MariaDB it can provide extra information:
  - `SHOW GLOBAL VARIABLES like 'log_slow_verbosity';`

Be careful with extra IO and latency!

pt-query-digest Execution (I)

```
# 1094.7s user time, 9.4s system time, 141.22M rss, 205.21M vsz
# Current date: Wed Jul  1 07:32:28 2015
# Hostname: db1018
# Files: STDIN
# Overall: 4.66M total, 640 unique, 53.47 QPS, 0.02x concurrency _________
# Time range: 2015-06-30 07:00:10 to 2015-07-01 07:11:37
# Attribute          total     min     max     avg     95%  stddev  median
# ============     ======= ======= ======= ======= ======= ======= =======
# Exec time          1320s     1us      3s   283us   332us     3ms   152us
# Lock time           238s       0    13ms    51us    93us    39us    52us
# Rows sent          5.02M       0   4.16k    1.13    1.96    8.69    0.99
# Rows examine       9.50M       0 232.93k    2.14    3.89  261.15    0.99
# Merge passes           0       0       0       0       0       0       0
# Query size         1.06G      17  67.89k  243.89  511.45  368.99  192.76
# Boolean:
# Filesort       8% yes,  91% no
# Full scan     94% yes,   5% no
# Priority que   3% yes,  96% no
# Tmp table     29% yes,  70% no
# Tmp table on   1% yes,  98% no
```

Actual execution on Wikipedia production servers

pt-query-digest Execution (II)

```
# Profile
# Rank Query ID                 Response time  Calls   R/Call V/M   Item
# ==== ======================== ============== ======= ====== ===== ============
#    1 0xSANITIZED              242.2765 18.4%  691005 0.0004  0.00 SELECT revision page user
#    2 0xSANITIZED              204.7052 15.5%   80863 0.0025  0.01 SELECT revision page user
#    3 0xSANITIZED              162.8476 12.3% 1025179 0.0002  0.00 SELECT page
#    4 0xSANITIZED               68.1164  5.2%   93928 0.0007  0.01 SELECT revision page user
#    5 0xSANITIZED               66.8302  5.1%  354562 0.0002  0.00 SELECT page revision
#    6 0xSANITIZED               57.0374  4.3%  211631 0.0003  0.00 SELECT page revision
#    7 0xSANITIZED               44.0751  3.3%    6925 0.0064  0.07 SELECT page categorylinks
category
#    8 0xSANITIZED               35.0655  2.7%    9689 0.0036  0.00 SELECT text
#    9 0xSANITIZED               29.4363  2.2%  152259 0.0002  0.00 SELECT page
#   10 0xSANITIZED               24.1864  1.8%  176927 0.0001  0.00 SELECT msg_resource
#   11 0xSANITIZED               23.7016  1.8%  144807 0.0002  0.00 SELECT page_restrictions
#   12 0xSANITIZED               16.6547  1.3%   10135 0.0016  0.03 SELECT revision
#   13 0xSANITIZED               15.0564  1.1%  263809 0.0001  0.00 SET
```

pt-query-digest Execution (III)

```
# Query 1: 7.93 QPS, 0.00x concurrency, ID 0xSANITIZED at byte 1553864032
# This item is included in the report because it matches -limit.
# Scores: V/M = 0.00
# Time range: 2015-06-30 07:00:10 to 2015-07-01 07:11:37
# Attribute    pct   total     min     max     avg     95%  stddev  median
# ============ === ======= ======= ======= ======= ======= ======= =======
# Count         14  691005
# Exec time     18    242s   163us    91ms   350us   348us   563us   301us
# Lock time     26     63s    47us     7ms    91us   103us    14us    84us
# Rows sent     12 657.18k       0       1    0.97    0.99    0.16    0.99
# Rows examine   6 657.18k       0       1    0.97    0.99    0.16    0.99
# Query size    31 345.42M     501     749  524.16  537.02    9.22  511.45
# String:
# Databases    itwiki (225976/32%), enwiktiona... (219461/31%)... 15 more
# Hosts
# Users        wikiuser
# Query_time distribution
#   1us
#  10us
# 100us  ################################################################
#   1ms  #
#  10ms  #
# 100ms
#    1s
#  10s+
# Tables
#    SHOW TABLE STATUS FROM `enwiktionary` LIKE 'revision'\G
#    SHOW CREATE TABLE `enwiktionary`.`revision`\G
#    SHOW TABLE STATUS FROM `enwiktionary` LIKE 'page'\G
#    SHOW CREATE TABLE `enwiktionary`.`page`\G
#    SHOW TABLE STATUS FROM `enwiktionary` LIKE 'user'\G
#    SHOW CREATE TABLE `enwiktionary`.`user`\G
# EXPLAIN /*!50100 PARTITIONS*/
SELECT /* Revision::fetchFromConds SANITIZED */ * FROM `revision`
INNER JOIN `page` ON ((page_id = rev_page)) LEFT JOIN `user` ON
((rev_user != 0) AND (user_id = rev_user))  WHERE page_namespace = '0' AND
page_title = 'SANITIZED' AND (rev_id=page_latest)  LIMIT 1\G
```

PERFORMANCE_SCHEMA

- Monitoring schema (engine) enabled by default since MySQL 5.6
  - performance_schema = 1 (it is not dynamic)
- Deprecates the old query profiling
- It is way more user-friendly when combined with the SYS schema/ps_helper (a set of views and stored procedures created by Mark Leith)
  - Included by default since 5.7.7

Installation of the SYS Schema <5.7/MariaDB

```
$ git clone https://github.com/MarkLeith/mysql-sys.git

Cloning into 'mysql-sys'...

remote: Counting objects: 926, done.

remote: Compressing objects: 100% (73/73), done.

remote: Total 926 (delta 35), reused 6 (delta 2)

Receiving objects: 100% (926/926), 452.19 KiB | 225.00 KiB/s, done.

Resolving deltas: 100% (584/584), done.

$ cd mysql-sys/

$ mysql < sys_56.sql
```

Example Usage: Discovering Unused Indexes

```
mysql (osm) > SELECT * FROM sys.schema_unused_indexes LIMIT 5;
+---------------+-------------------+---------------------------------+
| object_schema | object_name       | index_name                      |
+---------------+-------------------+---------------------------------+
| osm           | acls              | acls_k_idx                      |
| osm           | changeset_tags    | changeset_tags_id_idx           |
| osm           | current_nodes     | current_nodes_timestamp_idx     |
| osm           | current_nodes     | current_nodes_tile_idx          |
| osm           | current_relations | current_relations_timestamp_idx |
+---------------+-------------------+---------------------------------+
5 rows in set (0.04 sec)

mysql-5.7.8 (osm) > SELECT * FROM current_nodes WHERE tile = 100;
...

mysql-5.7.8 (osm) > SELECT * FROM sys.schema_unused_indexes LIMIT 5;
+---------------+-------------------+---------------------------------+
| object_schema | object_name       | index_name                      |
+---------------+-------------------+---------------------------------+
| osm           | acls              | acls_k_idx                      |
| osm           | changeset_tags    | changeset_tags_id_idx           |
| osm           | current_nodes     | current_nodes_timestamp_idx     |
| osm           | current_relations | current_relations_timestamp_idx |
| osm           | current_relations | changeset_id                    |
+---------------+-------------------+---------------------------------+
5 rows in set (0.03 sec)
```

With enough activity, it can help us clean up our schema

Example Usage: Slow Queries (ordered by server time)

```
mysql (osm) > SELECT * FROM sys.statement_analysis LIMIT 10\G
```

```
*************** 1. row ***************
           query: SELECT `way_id` AS
`id` , `v` FROM `way_tags` WHERE `v` LIKE ?
              db: osm
       full_scan: *
      exec_count: 15
       err_count: 0
      warn_count: 0
   total_latency: 7.83 s
     max_latency: 1.33 s
     avg_latency: 521.84 ms
    lock_latency: 17.94 ms
       rows_sent: 6779
   rows_sent_avg: 452
   rows_examined: 20152155
rows_examined_avg: 1343477
   rows_affected: 0
rows_affected_avg: 0
      tmp_tables: 0
 tmp_disk_tables: 0
     rows_sorted: 0
sort_merge_passes: 0
          digest:
21f90695b1ebf20a5f4d4c1e5e860f58
      first_seen: 2014-11-01 17:04:51
       last_seen: 2014-11-01 17:05:22
```

Example Usage: Top Queries Creating Temporary Tables

```
mysql (osm) > SELECT * FROM sys.statements_with_temp_tables
WHERE db = 'osm' LIMIT 10\G
*************************** 1. row ***************************
                   query: SELECT ? AS TYPE , `node_id` A ... gs` WHERE `k` = ? AND `v` = ?
                      db: osm
              exec_count: 11
           total_latency: 7.57 s
       memory_tmp_tables: 11
         disk_tmp_tables: 0
avg_tmp_tables_per_query: 1
  tmp_tables_to_disk_pct: 0
              first_seen: 2014-11-01 17:33:55
               last_seen: 2014-11-01 17:34:45
                  digest: 5e6e82799b7c7c0e5c57cfe63eb98d5d
```

Example Usage: Top Queries Creating Temporary Tables (cont.)

```
mysql (osm) > SELECT DIGEST_TEXT FROM performance_schema.events_statements_summary_by_digest WHERE
digest = '5e6e82799b7c7c0e5c57cfe63eb98d5d'\G
*************************** 1. row ***************************
DIGEST_TEXT: SELECT ? AS TYPE , `node_id` AS `id` FROM `node_tags` WHERE `k` = ? AND `v` = ? UNION
SELECT ? AS TYPE , `way_id` AS `id` FROM `way_tags` WHERE `k` = ? AND `v` = ? UNION SELECT ? AS
TYPE , `relation_id` AS `id` FROM `relation_tags` WHERE `k` = ? AND `v` = ?
1 row in set (0.00 sec)

mysql-5.7.8 (osm) > EXPLAIN SELECT 'node' as type, node_id as id FROM node_tags WHERE k='amenity'
and v='cafe' UNION SELECT 'way' as type, way_id as id FROM way_tags WHERE k='amenity' and v='cafe'
UNION SELECT 'relation' as type, relation_id as id FROM relation_tags WHERE k='amenity' and
v='cafe';
+----+--------------+---------------+------------+------+---------------+------+---------+------+---------+----------+-----------------+
| id | select_type  | table         | partitions | type | possible_keys | key  | key_len | ref  | rows    | filtered | Extra           |
+----+--------------+---------------+------------+------+---------------+------+---------+------+---------+----------+-----------------+
|  1 | PRIMARY      | node_tags     | NULL       | ALL  | NULL          | NULL | NULL    | NULL |  851339 |     0.00 | Using where     |
|  2 | UNION        | way_tags      | NULL       | ALL  | NULL          | NULL | NULL    | NULL | 1331016 |     0.00 | Using where     |
|  3 | UNION        | relation_tags | NULL       | ALL  | NULL          | NULL | NULL    | NULL |   63201 |     0.00 | Using where     |
| NULL | UNION RESULT | <union1,2,3>  | NULL       | ALL  | NULL          | NULL | NULL    | NULL |    NULL |     NULL | Using temporary |
+----+--------------+---------------+------------+------+---------------+------+---------+------+---------+----------+-----------------+
4 rows in set, 1 warning (0.01 sec)
```

Query Optimization with MySQL 8.0 and MariaDB 10.3: The Basics

ACCESS TYPES AND BASIC INDEXING TECHNIQUES

EXPLAIN

- Essential to understand the execution plan of our queries
  - Works on SELECTs, INSERTs, UPDATEs, REPLACEs, DELETEs and connections
  - Fully documented on: https://dev.mysql.com/doc/refman/8.0/en/explain-output.html

EXPLAIN Example

```
MariaDB [dewiktionary]> EXPLAIN SELECT * FROM page
WHERE page_title = 'German';
+------+-------------+-------+------+---------------
+------+---------+------+--------+-------------+
| id   | select_type | table | type | possible_keys |
key  | key_len | ref  | rows   | Extra       |
+------+-------------+-------+------+---------------
+------+---------+------+--------+-------------+
|    1 | SIMPLE      | page  | ALL  | NULL          |
NULL | NULL    | NULL | 778885 | Using
+------+-------------+-------+------+-
+------+---------+------+--------+----
1 row in set (0.00 sec)
```

Difficult to see something

EXPLAIN Example (vertical format)

```
MariaDB [dewiktionary]> EXPLAIN SELECT * FROM page WHERE
page_title = 'German'\G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: page
         type: ALL
possible_keys: NULL
          key: NULL
      key_len: NULL
          ref: NULL
         rows: 778885
        Extra: Using where
1 row in set (0.00 sec)
```

Use \G for vertical formatting

EXPLAIN Example (id)

```
MariaDB [dewiktionary]> EXPLAIN SELECT * FROM page WHERE
page_title = 'German'\G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: page
         type: ALL
possible_keys: NULL
          key: NULL
      key_len: NULL
          ref: NULL
         rows: 778885
        Extra: Using where
1 row in set (0.00 sec)
```

Indicates nesting level, not execution order

EXPLAIN Example (select_type)

```
MariaDB [dewiktionary]> EXPLAIN SELECT * FROM page WHERE
page_title = 'German'\G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: page
         type: ALL
possible_keys: NULL
          key: NULL
      key_len: NULL
          ref: NULL
         rows: 778885
        Extra: Using where
1 row in set (0.00 sec)
```

Not a subquery or a UNION

EXPLAIN Example (table)

```
MariaDB [dewiktionary]> EXPLAIN SELECT * FROM page WHERE
page_title = 'German'\G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: page
         type: ALL
possible_keys: NULL
          key: NULL
      key_len: NULL
          ref: NULL
         rows: 778885
        Extra: Using where
1 row in set (0.00 sec)
```

Table scanned for this step

EXPLAIN Example (type)

```
MariaDB [dewiktionary]> EXPLAIN SELECT * FROM page WHERE
page_title = 'German'\G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: page
         type: ALL
possible_keys: NULL
          key: NULL
      key_len: NULL
          ref: NULL
         rows: 778885
        Extra: Using where
1 row in set (0.00 sec)
```

All rows are read for this table (FULL TABLE SCAN)

EXPLAIN Example (rows)

```
MariaDB [dewiktionary]> EXPLAIN SELECT * FROM page WHERE
page_title = 'German'\G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: page
         type: ALL
possible_keys: NULL
          key: NULL
      key_len: NULL
          ref: NULL
         rows: 778885
        Extra: Using where
1 row in set (0.00 sec)
```

Estimated number of rows to be read (all table rows)

How to improve performance?

```
MariaDB [dewiktionary]> SELECT * FROM page
WHERE page_title = 'German';
1 row in set (0.10 sec)
```

- Let's add an index on page.page_title:

```
MariaDB [dewiktionary]> ALTER TABLE page ADD INDEX
page_title (page_title);
Query OK, 0 rows affected (22.70 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

Index creation results (type)

```
MariaDB [dewiktionary]> EXPLAIN SELECT * FROM page WHERE
page_title = 'German'\G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: page
         type: ref
possible_keys: page_title
          key: page_title
      key_len: 257
          ref: const
         rows: 1
        Extra: Using index condition
1 row in set (0.00 sec)
```

type: ref means that an equality comparison will be checked against an index and several results could be returned

Index creation results (possible_keys and key)

```
MariaDB [dewiktionary]> EXPLAIN SELECT * FROM page WHERE
page_title = 'German'\G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: page
         type: ref
possible_keys: page_title
          key: page_title
      key_len: 257
          ref: const
         rows: 1
        Extra: Using index condition
1 row in set (0.00 sec)
```

index(es) that the optimizer considered potentially useful, and final index chosen

Index creation results (ref)

```
MariaDB [dewiktionary]> EXPLAIN SELECT * FROM page WHERE
page_title = 'German'\G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: page
         type: ref
possible_keys: page_title
          key: page_title
      key_len: 257
          ref: const
         rows: 1
        Extra: Using index
1 row in set (0.00 sec)
```

Index is compared with a constant, not with another table

Index creation results (rows)

```
MariaDB [dewiktionary]> EXPLAIN SELECT * FROM page WHERE
page_title = 'German'\G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: page
         type: ref
possible_keys: page_title
          key: page_title
      key_len: 257
          ref: const
         rows: 1
        Extra: Using index condit
1 row in set (0.00 sec)
```

Only 1 row read. In this case, estimation is exact (thanks to index dive)

Index creation results (query time)

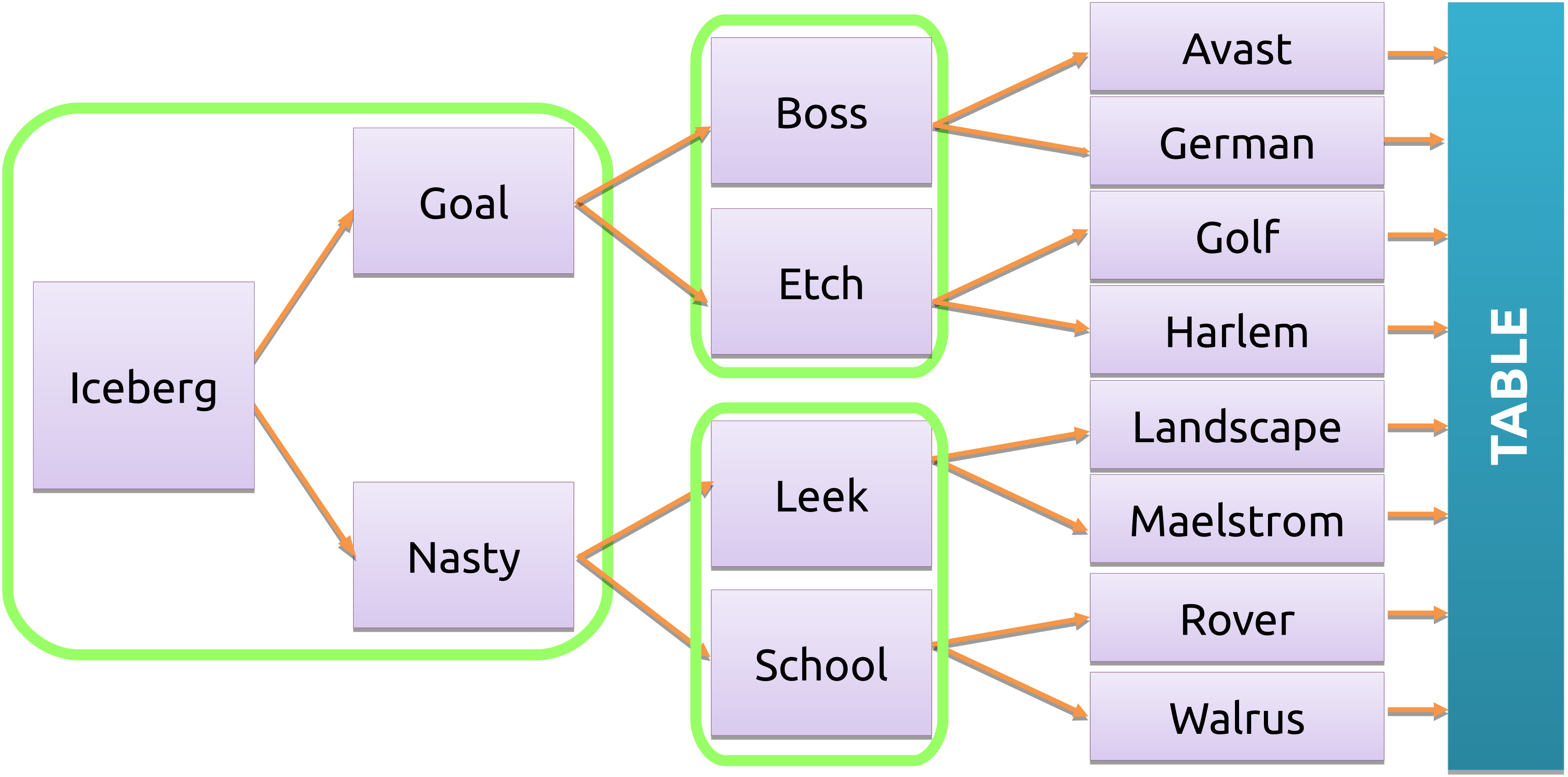
```
MariaDB [dewiktionary]> SELECT * FROM page
WHERE page_title = 'German';
1 row in set (0.00 sec)
```

Query time has been reduced substantially

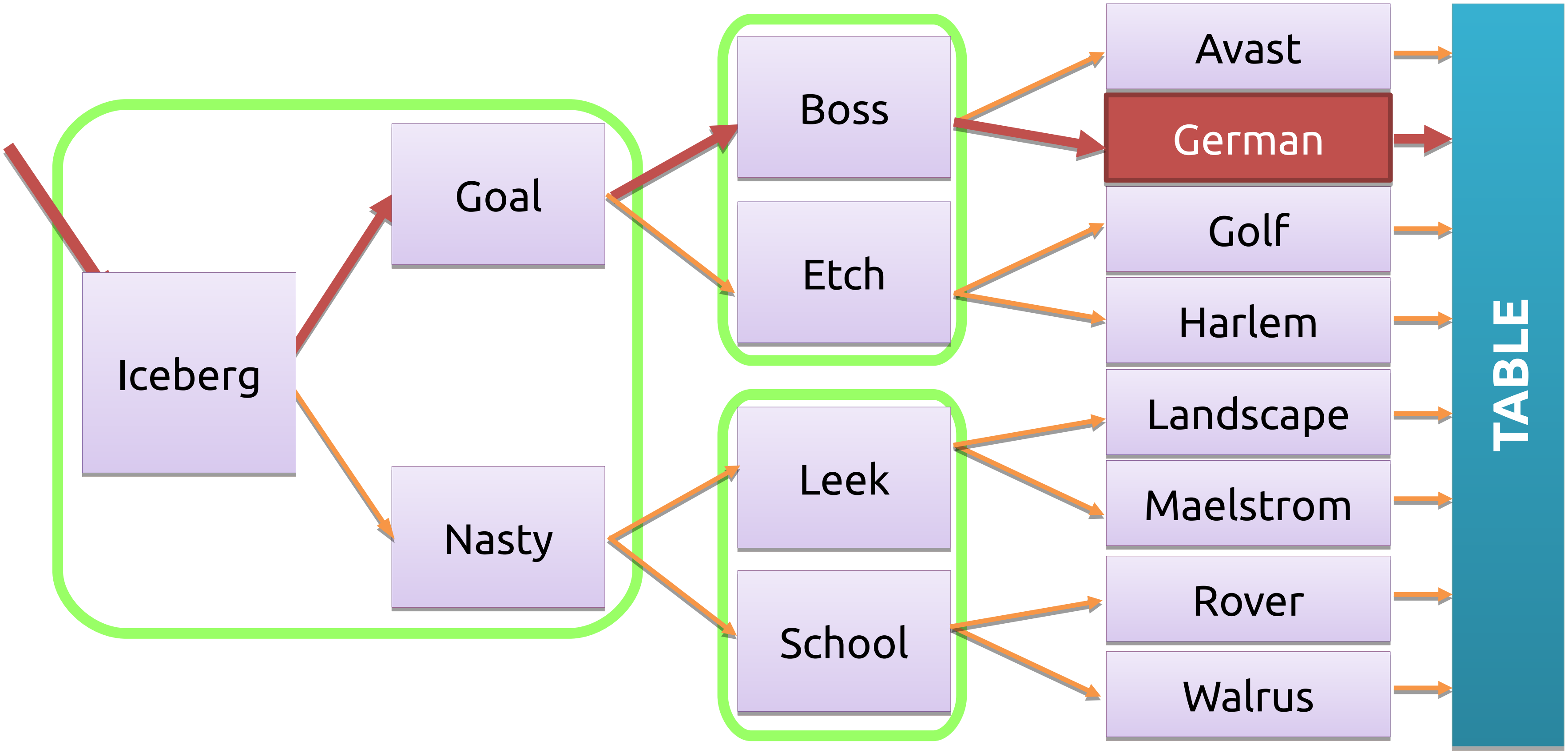
Types of indexes

- BTREE
  - B-TREE in MyISAM, B+TREE in InnoDB
- HASH
  - Only available for MEMORY and NDB
- FULLTEXT
  - Inverted indexes in MyISAM and InnoDB
- SPATIAL
  - RTREEs in MyISAM and InnoDB

Finding “German” with a BTREE



Finding “German” with a BTREE



Do indexes always work? (1/2)

- Can we use an index to make this query faster?

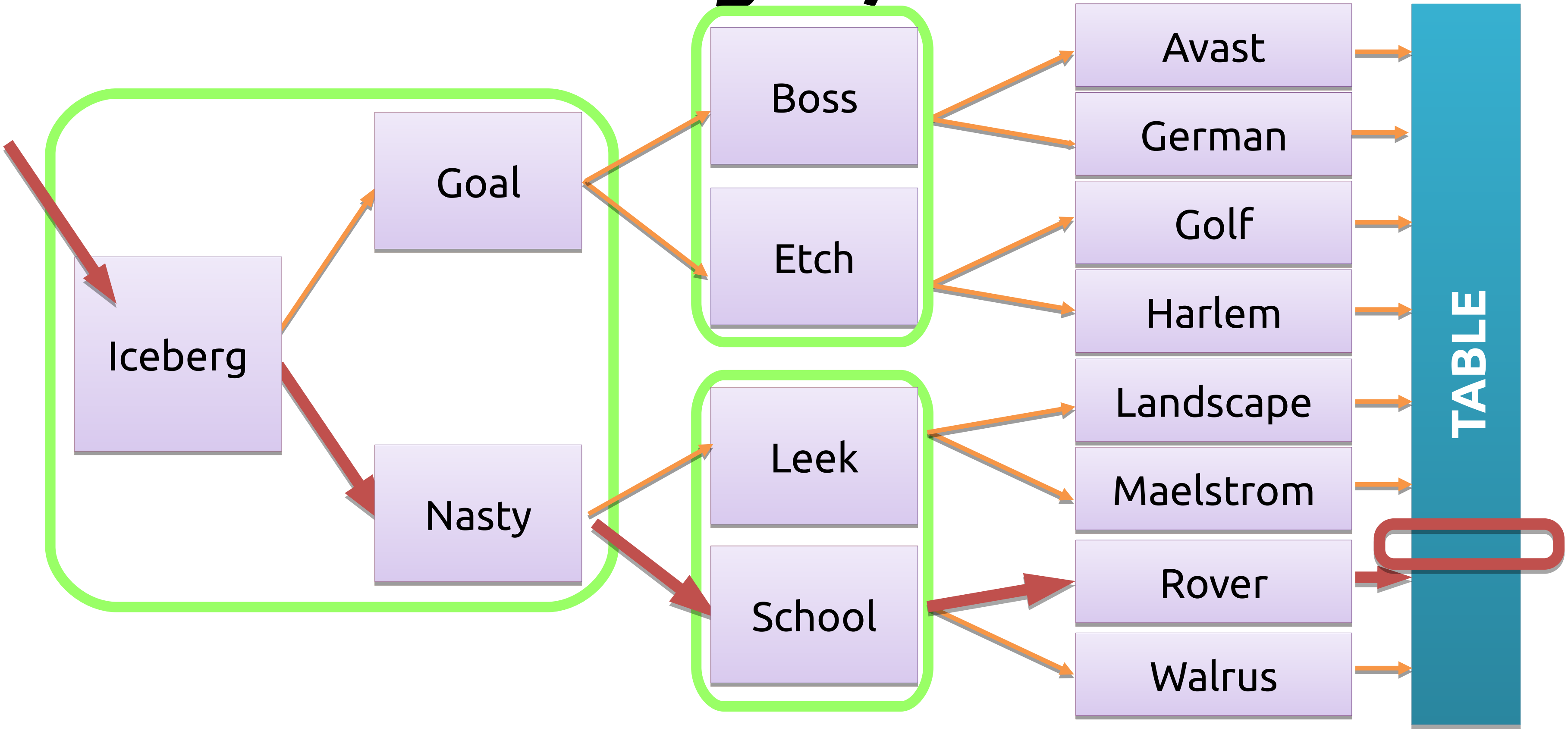
```
SELECT * FROM page WHERE page_title like 'Spa%';
```

It is a range

```
MariaDB [dewiktionary]> EXPLAIN SELECT * FROM page WHERE
page_title like 'Spa%'\G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: page
         type: range
possible_keys: page_title
          key: page_title
      key_len: 257
          ref: NULL
         rows: 563
        Extra: Using index condition
1 row in set (0.00 sec)
```

Despite not being an equality, we can use the index to find the values quickly

BTREE Indexes can be used for ranges, too



Do indexes always work? (2/2)

- What about this other query?

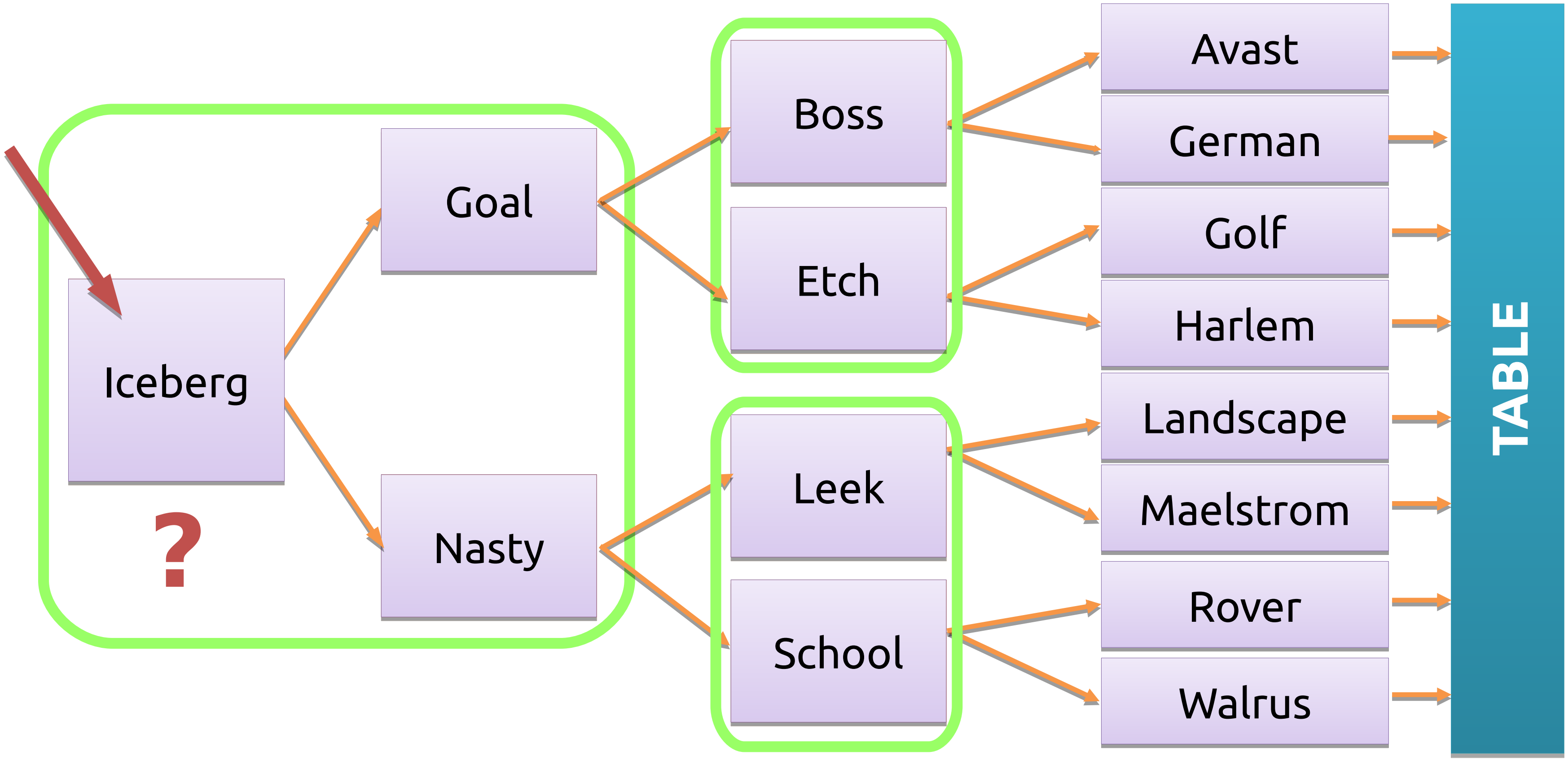
```
SELECT * FROM page WHERE page_title like '%erman';
```

Let's check with EXPLAIN

```
MariaDB [dewiktionary]> EXPLAIN SELECT * FROM page WHERE
page_title like '%erman'\G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: page
         type: ALL
possible_keys: NULL
          key: NULL
      key_len: NULL
          ref: NULL
         rows: 778885
        Extra: Using where
1 row in set (0.00 sec)
```

No index can be used for filtering. A full table scan is performed.

BTREE Index


Iceberg
?
Goal
Nasty
Boss
Etch
Leek
School
Avast
German
Golf
Harlem
Landscape
Maelstrom
Rover
Walrus
TABLE

Btree indexes usage (I)

- Filtering (“WHERE”)
  - Equality (operator '=')
  - Ranges (BETWEEN … AND, >, <, >=, <=, like 'prefix%')
  - “EXISTS” operators: IN, OR on the same column
- Ordering
  - ORDER BY (indexed columns)
  - GROUP BY (indexed columns)

Btree indexes usage (II)

- Returning values directly from the index
  - Covering index
  - Functions like max(), min(), etc.
- Reduced contention (writes)
  - Locking less rows, increasing concurrency

type: const

```
MariaDB [dewiktionary]> EXPLAIN SELECT * FROM revision WHERE
rev_id = 2\G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: revision
         type: const
possible_keys: PRIMARY
          key: PRIMARY
      key_len: 4
          ref: const
         rows: 1
        Extra:
1 row in set (0.00 sec)
```

'const' is a special case of 'ref', when the index can assure that only 1 result can be returned (equality + primary key or unique key). It is faster.

type: NULL

```
MariaDB [dewiktionary]> EXPLAIN SELECT * FROM revision WHERE
rev_id = -1\G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: NULL
         type: NULL
possible_keys: NULL
          key: NULL
      key_len: NULL
          ref: NULL
         rows: NULL
        Extra: Impossible WHERE noticed after reading const
tables
1 row in set (0.00 sec)
```

'NULL' is not really a plan, just an optimization that allows immediately discarding impossible conditions

type: ref_or_null

```
MariaDB [osm]> EXPLAIN SELECT * FROM nodes WHERE tile = 1 or
tile is null\G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: nodes
         type: ref_or_null
possible_keys: nodes_tile_idx
          key: nodes_tile_idx
      key_len: 5
          ref: const
         rows: 2
        Extra: Using index condition; Using where
1 row in set (0.00 sec)
```

Equivalent to 'ref', but also takes into account NULL values

type: range (using IN / OR)

```
MariaDB [dewiktionary]> EXPLAIN SELECT * FROM page WHERE
page_title IN ('German', 'English', 'Spanish')\G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: page
         type: range
possible_keys: page_title
          key: page_title
      key_len: 257
          ref: NULL
         rows: 4
        Extra: Using index condition
1 row in set (0.00 sec)
```

Despite being a range, its execution is very different from ranges using like, between or inequality operators

Is this a bug? (1/2)

```
MariaDB [dewiktionary]> EXPLAIN SELECT * FROM page WHERE
page_namespace > 200\G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: page
         type: ref
possible_keys: name_title
          key: name_title
      key_len: 4
          ref: const
         rows: 50
        Extra:
1 row in set (0.00 sec)
```

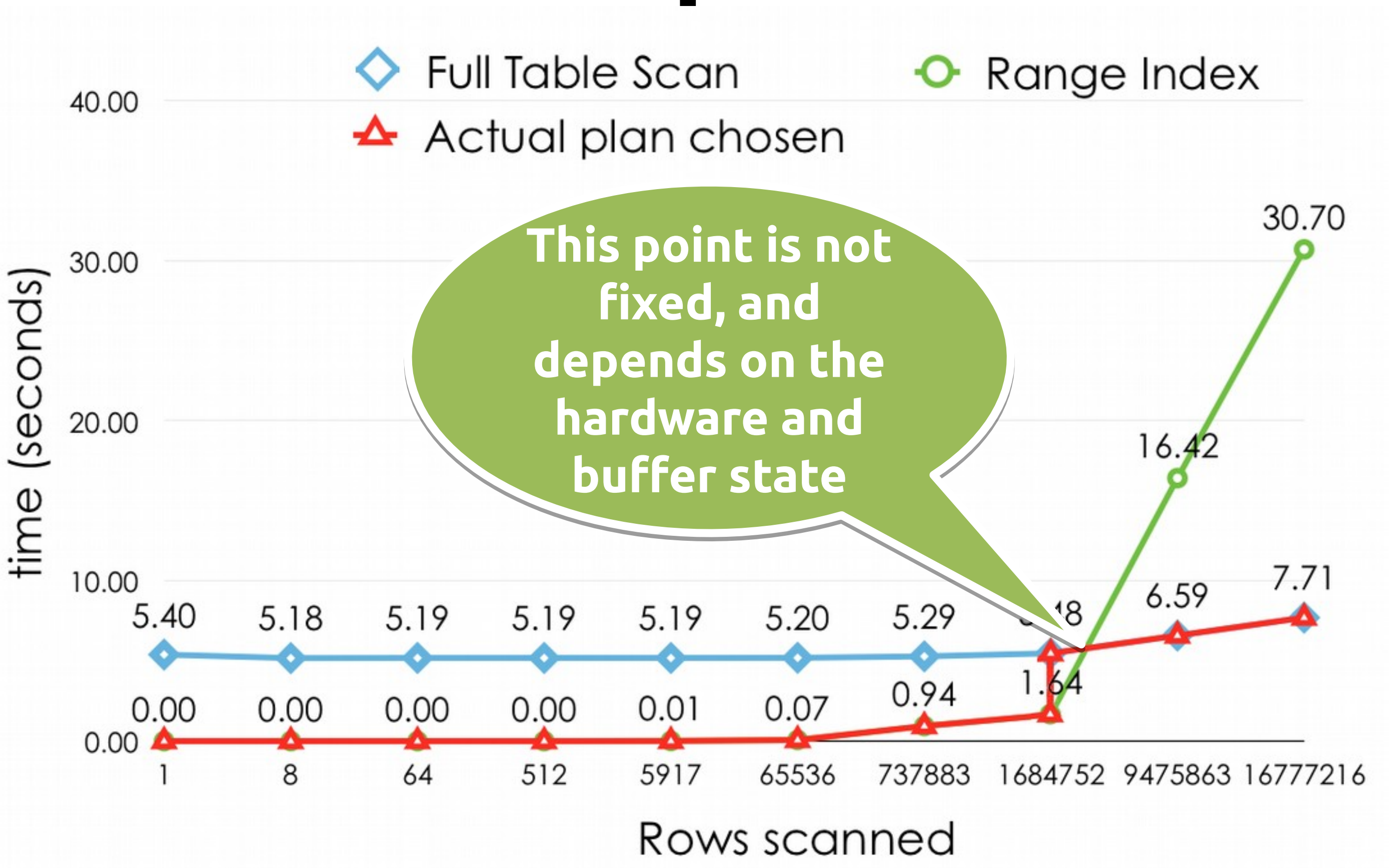
An index is used to return pages with namespace>200

Is this a bug? (2/2)

```
MariaDB [dewiktionary]> EXPLAIN SELECT * FROM page WHERE
page_namespace < 200\G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: page
         type: ALL
possible_keys: name_title
          key: NULL
      key_len: NULL
          ref: NULL
         rows: 778885
        Extra: Using where
1 row in set (0.00 sec)
```

The index is not used with ns=0

Using an index is sometimes suboptimal



This point is not fixed, and depends on the hardware and buffer state

What index should we add to make this query faster?

```
MariaDB [dewiktionary]> EXPLAIN SELECT * FROM revision WHERE
left(rev_timestamp, 6) = '201809'\G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: revision
         type: ALL
possible_keys: NULL
          key: NULL
      key_len: NULL
          ref: NULL
         rows: 1376651
        Extra: Using where
1 row in set (0.00 sec)
```

The table has already an index on rev_timestamp

```
MariaDB [dewiktionary]> SHOW CREATE TABLE revision\G
*************************** 1. row ***************************
       Table: revision
Create Table: CREATE TABLE `revision` (
...
  KEY `rev_timestamp` (`rev_timestamp`),
```

We need to rewrite the query

```
MariaDB [dewiktionary]> EXPLAIN SELECT * FROM revision WHERE
rev_timestamp >= '201509' and rev_timestamp < '201510'\G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: revision
         type: range
possible_keys: rev_timestamp
          key: rev_timestamp
      key_len: 14
          ref: NULL
         rows: 24588
        Extra: Using index condition
1 row in set (0.00 sec)
```

This transformation is not trivial or even possible in all cases

```
MariaDB [dewiktionary]> EXPLAIN SELECT * FROM revision WHERE
substr(rev_timestamp, 5, 2) = '09'\G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: revision
         type: ALL
possible_keys: NULL
          key: NULL
      key_len: NULL
          ref: NULL
         Rows: 1376651
        Extra: Using where
1 row in set (0.00 sec)
```

Can you think of a way to improve this query?

Indexes for Ordering

```
MariaDB [dewiktionary]> EXPLAIN SELECT * FROM page ORDER BY
page_touched DESC LIMIT 10\G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: page
         type: ALL
possible_keys: NULL
          key: NULL
      key_len: NULL
          ref: NULL
         rows: 778885
        Extra: Using filesort
1 row in set (0.00 sec)
```

"Using filesort" indicates that an ordering is needed before returning the results

If that is frequent, we can create an index on page_touched...

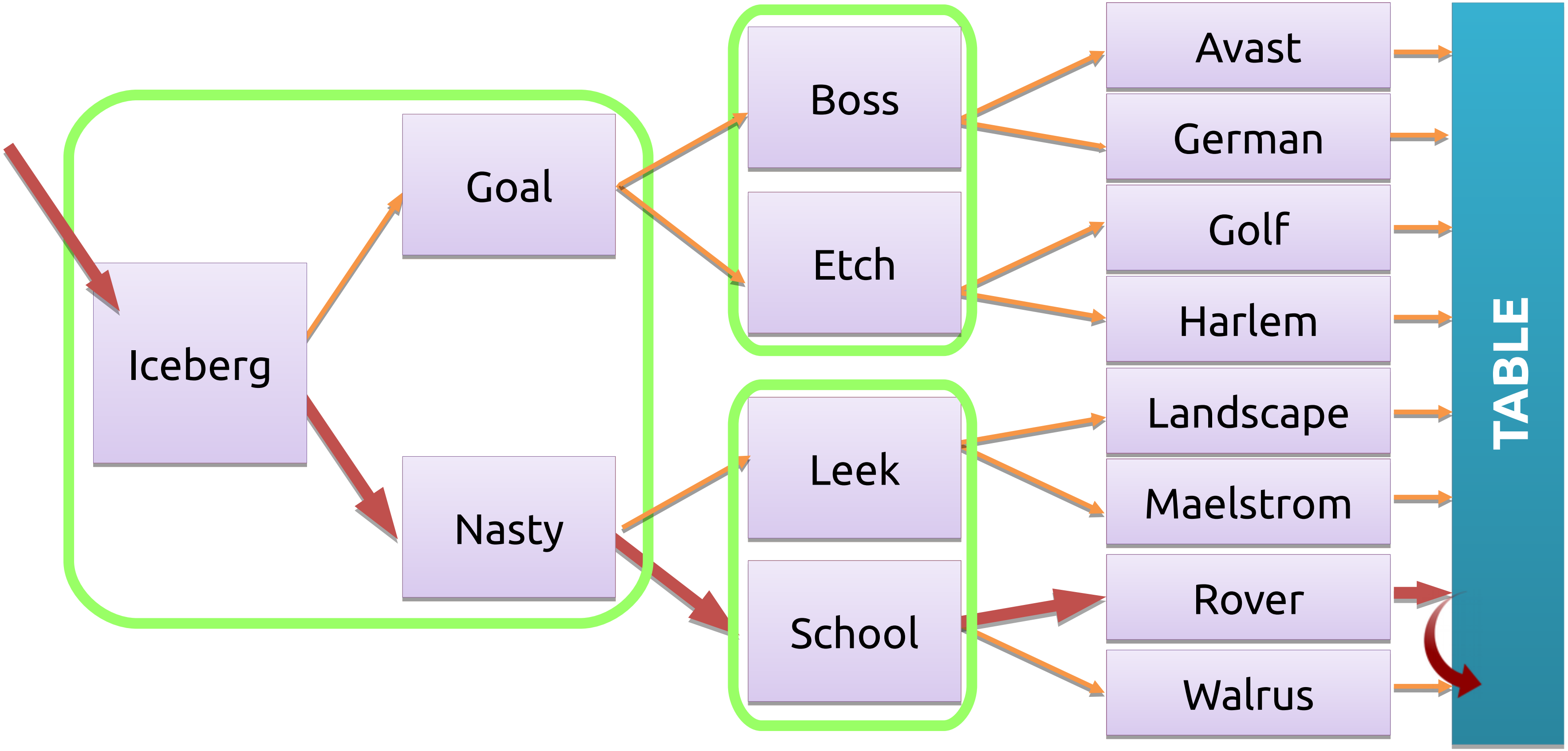
```
[dewiktionary]> ALTER TABLE page ADD INDEX page_page_touched(page_touched);
Query OK, 0 rows affected (3.03 sec)
Records: 0  Duplicates: 0  Warnings: 0

[dewiktionary]> EXPLAIN SELECT * FROM page ORDER BY page_touched DESC LIMIT
10\G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: page
         type: index
possible_keys: NULL
          key: page_page_touched
      key_len: 16
          ref: NULL
         rows: 10
        Extra:
1 row in set (0.00 sec)
```

The index does not produce any advantage for filtering

However, it is very effective by helping to avoid the sort phase

It can return data in index order faster



Trick: ORDER BY NULL avoids filesort

```
MariaDB [dewiktionary]> EXPLAIN SELECT * FROM revision GROUP BY substr(rev_timestamp, 5,
2) = '09'\G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: revision
         type: ALL
possible_keys: NULL
          key: NULL
      key_len: NULL
          ref: NULL
         rows: 1376651
        Extra: Using temporary; Using filesort
1 row in set (0.00 sec)
```

There is no good index in this case

```
MariaDB [nlwiktionary]> EXPLAIN SELECT * FROM revision GROUP BY substr(rev_timestamp, 5,
2) = '09' ORDER BY NULL\G
*************************** 1. row ***************************
...
         rows: 1376651
        Extra: Using temporary
1 row in set (0.00 sec)
```

The advantage is not too big, but it avoids the filesort

Does LIMIT improve the performance? (I)

```
MariaDB [dewiktionary]> ALTER TABLE page ADD INDEX page_touched (page_touched); […]
MariaDB [dewiktionary]> EXPLAIN SELECT * FROM page ORDER BY page_touched\G
*************************** 1. row ***************************
         type: ALL
possible_keys: NULL
          key: NULL
      key_len: NULL
          ref: NULL
         rows: 783231
        Extra: Using filesort
MariaDB [dewiktionary]> EXPLAIN SELECT * FROM page ORDER BY page_touched LIMIT 10\G
*************************** 1. row ***************************
         type: index
possible_keys: NULL
          key: page_page_touched
      key_len: 16
          ref: NULL
         rows: 10
        Extra:
1 row in set (0.00 sec)
```

In some cases it can be essential to allow effective usage of the indexes

Does LIMIT improve the performance? (II)

```
MariaDB [dewiktionary]> EXPLAIN SELECT * FROM revision ORDER BY rev_comment\G
*************************** 1. row ***************************
[...]    type: ALL
possible_keys: NULL
          key: NULL
      key_len: NULL
          ref: NULL
         rows: 817636
        Extra: Using filesort
1 row in set (0.00 sec)

MariaDB [dewiktionary]> EXPLAIN SELECT * FROM revision ORDER BY rev_comment LIMIT 10\G
*************************** 1. row ***************************
[…]     table: revision
         type: ALL
possible_keys: NULL
          key: NULL
      key_len: NULL
          ref: NULL
         rows: 817636
        Extra: Using filesort
1 row in set (0.00 sec)
```

In other cases, it has no effect on the scanned rows (just on the returned ones)

Does LIMIT improve the performance? (III)

```
MariaDB [nlwiktionary]> EXPLAIN SELECT * FROM page ORDER BY page_title LIMIT 100\G
*************************** 1. row ***************************
         type: index
possible_keys: NULL
          key: page_title
      key_len: 257
          ref: NULL
         rows: 100
        Extra:
1 row in set (0.00 sec)

MariaDB [nlwiktionary]> EXPLAIN SELECT * FROM page ORDER BY page_title LIMIT 10000, 100\G
*************************** 1. row ***************************
         type: ALL
possible_keys: NULL
          key: NULL
      key_len: NULL
          ref: NULL
         rows: 90956
        Extra: Using filesort
1 row in set (0.00 sec)
```

In this case, performance will vary depending on the offset (not ideal)

Indexes and GROUP BY (no indexes)

```
MariaDB [dewiktionary]> EXPLAIN SELECT rev_page, count(*) FROM
revision IGNORE INDEX(rev_page_id, page_timestamp,
page_user_timestamp) GROUP BY rev_page\G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: revision
         type: ALL
possible_keys: NULL
          key: NULL
      key_len: NULL
          ref: NULL
         rows: 1376651
        Extra: Using temporary; Using filesort
1 row in set (0.00 sec)
```

Without indexes, a temporary table is created to order results

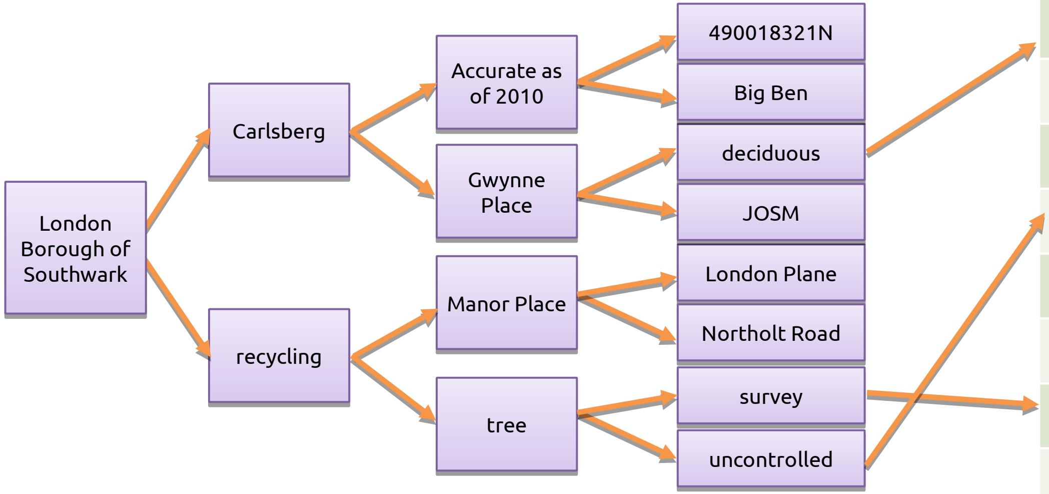
Indexes and GROUP BY (rev_page_id)

```
MariaDB [dewiktionary]> EXPLAIN SELECT rev_page, count(*) FROM
revision GROUP BY rev_page\G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: revision
         type: index
possible_keys: NULL
          key: rev_page_id
      key_len: 8
          ref: NULL
         rows: 1376651
        Extra: Using index
1 row in set (0.00 sec)
```

The index does not produce any advantage for filtering (there is no WHERE clause)

However, thanks to it we avoid a sort and a temporary table

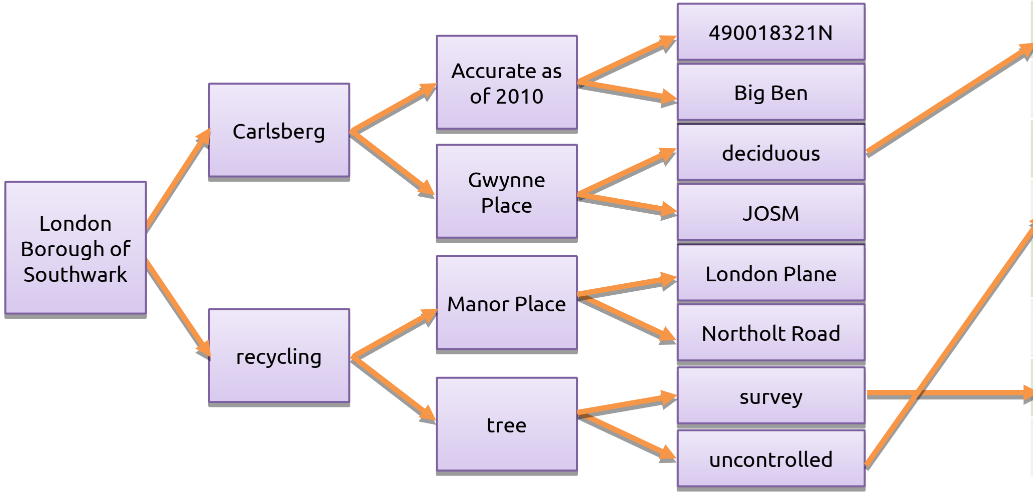
type: index, loose index scan and covering index (1/3)



node_id	version	k	v
234234344545	1	leaves	deciduous
234234344546	3	name	Northolt Road
234234344548	5	gps:latitude	490018321N
234234344549	6	access	uncontrolled
234234344550	1	editor	JOSM
234234344551	9	name	Big Ben
234234344552	1	source	survey
234234344557	1	name	London Plane

With 'type:index', all rows are read in index order (full index scan)

type: index, loose index scan and covering index (2/3)



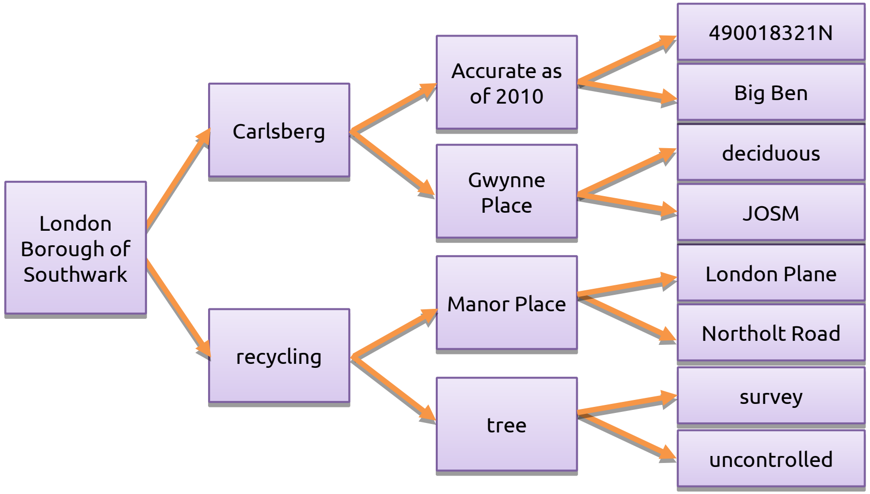
node_id	version	k	v
234234344545	1	leaves	deciduous
~~234234344546~~	3	name	Northolt Road
~~234234344548~~	5	gps:latitude	490018321N
234234344549	6	access	uncontrolled
~~234234344550~~	1	editor	JOSM
~~234234344551~~	9	name	Big Ben
234234344552	1	source	survey
~~234234344557~~	1	name	London Plane

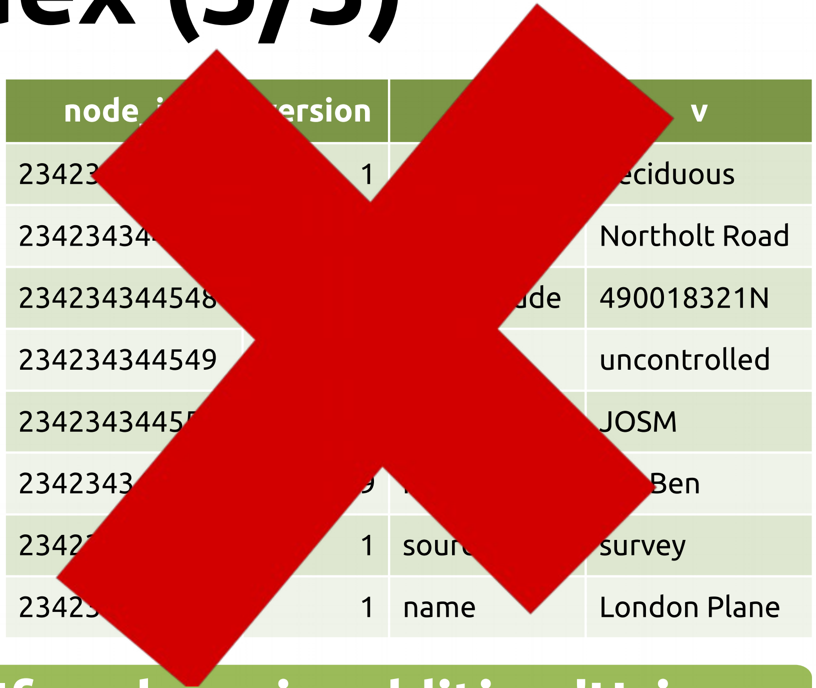
If we have in addition 'Using index for group-by' we have the loose index scan optimization

Loose Index Scan Example

```
MariaDB [dewiktionary]> EXPLAIN SELECT rev_page,
max(rev_timestamp) FROM revision GROUP BY rev_page\G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: revision
         type: range
possible_keys: NULL
          key: page_timestamp
      key_len: 4
          ref: NULL
         rows: 80980
        Extra: Using index for group-by
1 row in set (0.00 sec)
```

type: index, loose index scan and covering index (3/3)





node_i	ersion		v
23423	1		eciduous
23423434			Northolt Road
234234344548		de	490018321N
234234344549			uncontrolled
2342343445			JOSM
2342343	9		Ben
2342	1	sour	survey
23423	1	name	London Plane

If we have in addition 'Using index' we have the covering index optimization

Covering Index Example (1/3)

```
MariaDB [dewiktionary]> ALTER TABLE revision DROP INDEX rev_page_id, drop index page_timestamp,
drop index page_user_timestamp;
Query OK, 0 rows affected (0.04 sec)
Records: 0  Duplicates: 0  Warnings: 0

MariaDB [dewiktionary]> EXPLAIN SELECT count(DISTINCT rev_user) FROM revision WHERE rev_page =
31579\G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: revision
         type: ALL
possible_keys: NULL
          key: NULL
      key_len: NULL
          ref: NULL
         rows: 1376651
        Extra: Using where
1 row in set (0.00 sec)

MariaDB [dewiktionary]> SELECT count(DISTINCT rev_user) FROM revision WHERE rev_page = 31579\G
*************************** 1. row ***************************
count(DISTINCT rev_user): 1
1 row in set (3.57 sec)
```

Let's start with no indexes

Covering Index Example (2/3)

```
MariaDB [dewiktionary]> ALTER TABLE revision ADD INDEX revision_rev_page(rev_page);
Query OK, 0 rows affected (12.38 sec)
Records: 0  Duplicates: 0  Warnings: 0


MariaDB [dewiktionary]> EXPLAIN SELECT count(DISTINCT rev_user) FROM revision WHERE rev_page =
31579\G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: revision
         type: ref
possible_keys: revision_rev_page
          key: revision_rev_page
      key_len: 4
          ref: const
         rows: 4038
        Extra:
1 row in set (0.00 sec)

MariaDB [dewiktionary]> SELECT count(DISTINCT rev_user) FROM revision WHERE rev_page = 31579\G
*************************** 1. row ***************************
count(DISTINCT rev_user): 1
1 row in set (0.04 sec)
```

Adding an index on rev_page increases the speed due to improved filtering

Covering Index Example (3/3)

```
MariaDB [dewiktionary]> ALTER TABLE revision ADD INDEX revision_rev_page_rev_user(rev_page,
rev_user);
Query OK, 0 rows affected (13.20 sec)
Records: 0  Duplicates: 0  Warnings: 0

MariaDB [dewiktionary]> EXPLAIN SELECT count(DISTINCT rev_user) FROM revision WHERE rev_page = 31579\
G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: revision
         type: ref
possible_keys: revision_rev_page,revision_rev_page_rev_u
          key: revision_rev_page_rev_user
      key_len: 4
          ref: const
         rows: 4038
        Extra: Using index
1 row in set (0.00 sec)

MariaDB [dewiktionary]> SELECT count(DISTINCT rev_user) FROM revision WHERE rev_page = 31579\G
*************************** 1. row ***************************
count(DISTINCT rev_user): 1
1 row in set (0.00 sec)
```

rev_page, rev_user does not increase the index selectiveness, but allows returning results directly from the index

The speed difference can be huge

Query Optimization with MySQL 8.0 and MariaDB 10.3: The Basics

MULTI-COLUMN INDEXES

In many cases, conditions are applied on more than one column

```
MariaDB [dewiktionary]> EXPLAIN SELECT * FROM revision WHERE
rev_page = 31579 and rev_timestamp < '2008'\G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: revision
         type: ALL
possible_keys: NULL
          key: NULL
      key_len: NULL
          ref: NULL
         rows: 1376651
        Extra: Using where
1 row in set (0.00 sec)
```

Assuming there were no previously created indexes, which would the optimal one be?

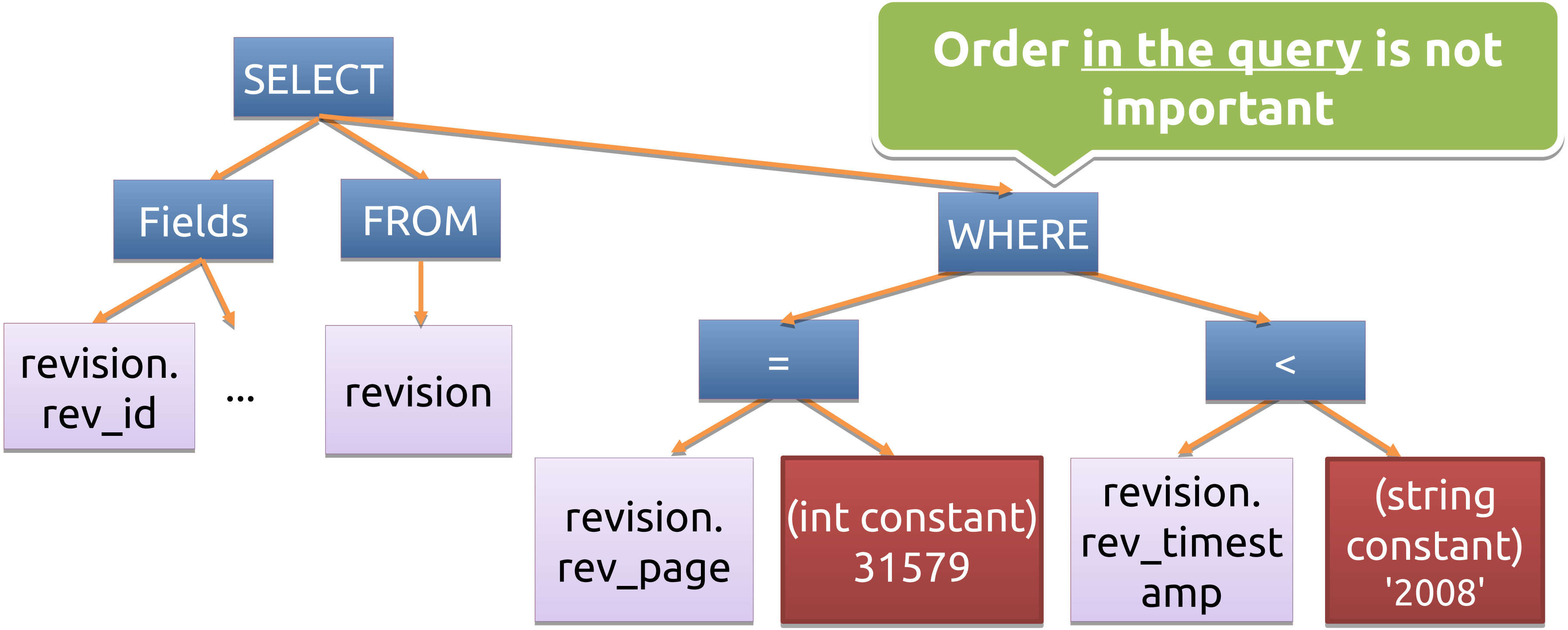
Options for indexes

- 1 index on column (rev_page)
- 1 index on column (rev_timestamp)
- 2 indexes, 1 on (rev_page) and another on (rev_timestamp)
- 1 multi-column index on (rev_page, rev_timestamp)
- 1 multi-column index on (rev_timestamp,rev_page)

Are these last 2 different from each other? Would it depend on the query order?

A brief reminder about query parsing

```
SELECT * FROM revision WHERE rev_page = 31579 and rev_timestamp < '2008'
```



Order in the query is not important

Index on (rev_page)

```
MariaDB [dewiktionary]> ALTER TABLE revision ADD INDEX rev_page (rev_page);
Query OK, 0 rows affected (11.87 sec)
Records: 0  Duplicates: 0  Warnings: 0

MariaDB [dewiktionary]> EXPLAIN SELECT * FROM revision WHERE rev_page = 31579 and
rev_timestamp < '2008'\G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: revision
         type: ref
possible_keys: rev_page
          key: rev_page
      key_len: 4
          ref: const
         rows: 4038
        Extra: Using where
1 row in set (0.00 sec)
```

Query time improves significantly with this index

Fewer rows are scanned

Adding (rev_timestamp)

```
MariaDB [dewiktionary]> ALTER TABLE revision ADD INDEX rev_timestamp
(rev_timestamp);
Query OK, 0 rows affected (17.01 sec)
Records: 0  Duplicates: 0  Warnings: 0

MariaDB [dewiktionary]> EXPLAIN SELECT * FROM revision WHERE rev_page =
31579 and rev_timestamp < '2008'\G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: revision
         type: ref
possible_keys: rev_page,rev_timestamp
          key: rev_page
      key_len: 4
          ref: const
         rows: 4038
        Extra: Using where
1 row in set (0.01 sec)
```

In general, only one index can be used per table access

rev_page is preferred over rev_timestamp

Forcing the use of (rev_timestamp)

```
MariaDB [dewiktionary]> EXPLAIN SELECT * FROM revision FORCE
INDEX(rev_timestamp) WHERE rev_page = 31579 and rev_timestamp
< '2008'\G
*******************
           id: 1
  select_type: SIMPLE
        table: revision
         type: range
possible_keys: rev_timestamp
          key: rev_timestamp
      key_len: 14
          ref: NULL
         rows: 688325
        Extra: Using index condition; Using where
1 row in set (0.00 sec)
```

Forcing the index is worse than type:ALL!

It is a range access

A lot more accessed rows

Adding (rev_page, rev_timestamp)

```
MariaDB [dewiktionary]> ALTER TABLE revision ADD INDEX
rev_page_rev_timestamp(rev_page, rev_timestamp);
Query OK, 0 rows affected (14.02 sec)
Records: 0  Duplicates: 0  Warnings: 0

MariaDB [dewiktionary]> EXPLAIN SELECT * FROM revision WHERE rev_page =
31579 and rev_timestamp < '2008'\G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: revision
         type: range
possible_keys: rev_page,rev_timestamp,rev_page_rev_timestamp
          key: rev_page_rev_timestamp
      key_len: 18
          ref: NULL
         rows: 530
        Extra: Using index condition
1 row in set (0.00 sec)
```

Reduced number of rows scanned

Is (rev_timestamp, rev_page) a better option?

```
MariaDB [dewiktionary]> ALTER TABLE revision ADD INDEX rev_timestamp_rev_page
(rev_timestamp, rev_page);
Query OK, 0 rows affected (16.80 sec)
Records: 0  Duplicates: 0  Warnings: 0

MariaDB [dewiktionary]> EXPLAIN SELECT * FROM revision WHERE rev_page = 31579 and
rev_timestamp < '2008'\G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: revision
         type: range
possible_keys: rev_page,rev_timestamp,rev_page_rev_timestamp,rev_timestamp_rev_page
          key: rev_page_rev_timestamp
      key_len: 18
          ref: NULL
         rows: 530
        Extra: Using index condition
1 row in set (0.00 sec)
```

Previous index is still preferred, why?

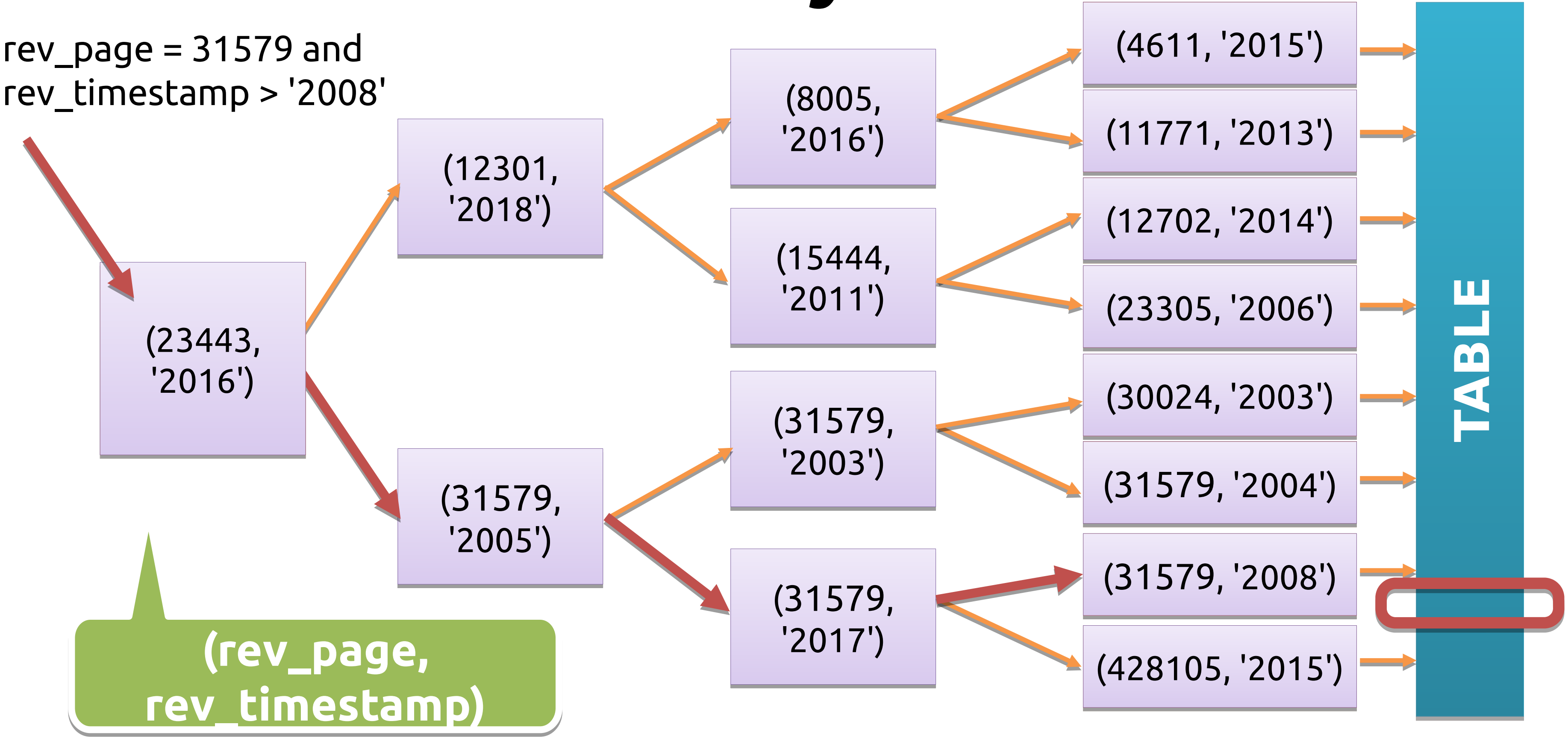
Forcing (rev_timestamp, rev_page)

```
MariaDB [dewiktionary]> EXPLAIN SELECT * FROM revision FORCE
INDEX(rev_timestamp_rev_page) WHERE rev_page = 31579 and
rev_timestamp < '2008'\G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: revision
         type: range
possible_keys: rev_timestamp_rev_page
          key: rev_timestamp_rev_page
      key_len: 18
          ref: NULL
         rows: 688325
        Extra: Using index condition
1 row in set (0.00 sec)
```

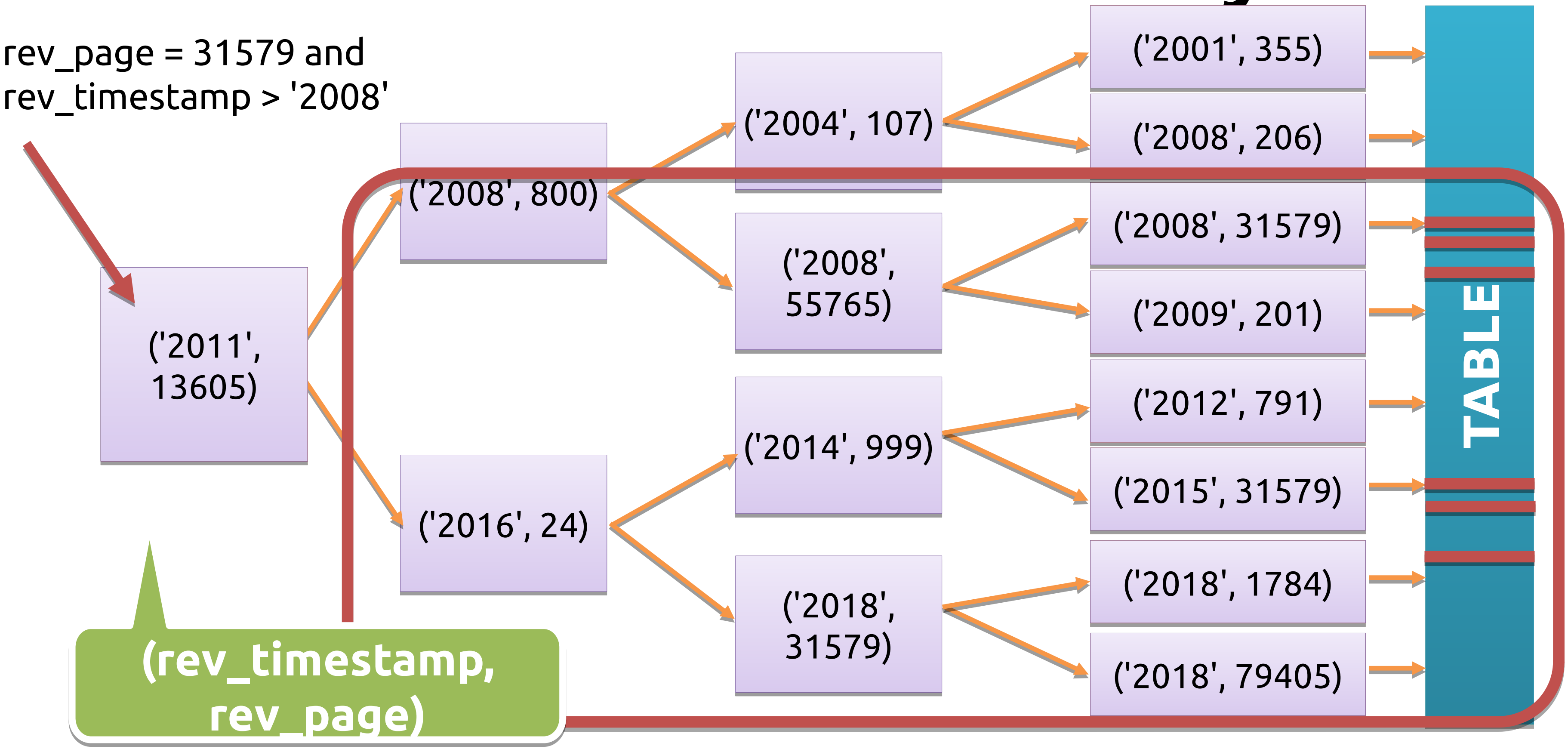
Only the 1st column* is being used effectively for filtering

A compound index produces a single tree ordered by the 2 values

rev_page = 31579 and
rev_timestamp > '2008'



The alternative index cannot be used in such an effective way



Order and column selection

- Range access using >, <, >=, <=, BETWEEN can only be filtered once effectively, at the end of an index
- When selecting indexes, prefer columns with high cardinality (very selective)
    - The optimal index can depend on the constants used

Can we filter and sort at the same time using indexes?

```
MariaDB [dewiktionary]> EXPLAIN SELECT * FROM revision WHERE
rev_comment='' ORDER BY rev_timestamp ASC\G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: revision
         type: ALL
possible_keys: NULL
          key: NULL
      key_len: NULL
          ref: NULL
         rows: 817636
        Extra: Using where; Using filesort
1 row in set (0.00 sec)
```

This query is slow because a) the full table scan

b) Required sort after filtering

Adding an index on (rev_comment, rev_timestamp)

```
MariaDB [dewiktionary]> ALTER TABLE revision ADD INDEX
rev_comment_rev_timestamp (rev_comment, rev_timestamp);
Query OK, 0 rows affected (16.19 sec)
Records: 0  Duplicates: 0  Warnings: 0

MariaDB [dewiktionary]> EXPLAIN SELECT * FROM revision WHERE rev_comment=''
ORDER BY rev_timestamp ASC\G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: revision
         type: ref
possible_keys: rev_comment_rev_times
          key: rev_comment_rev_timesta
      key_len: 769
          ref: const
         rows: 266462
        Extra: Using index condition; Using where
1 row in set (0.00 sec)
```

Both ‘type: ALL’ and ‘filesort’ have disappeared

This is not always possible

```
MariaDB [dewiktionary]> EXPLAIN SELECT * FROM revision WHERE
rev_len > 5500 ORDER BY rev_timestamp ASC\G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: revision
         type: range
possible_keys: rev_len_rev_timest
          key: rev_len_rev_timest
      key_len: 5
          ref: NULL
         rows: 38744
        Extra: Using index condition; Using filesort
1 row in set (0.00 sec)
```

The range makes impossible to use the index optimally for the ORDER BY: either we filter (rev_len) or sort (rev_timestamp)

A Strange Game. The Only Winning Move is Not to Play

```
mysql (osm) > SELECT * FROM nodes FORCE INDEX(version_idx)
WHERE version < 15 ORDER BY changeset_id;
/* type: range, Using filesort */
2859673 rows in set (30.58 sec)

mysql (osm) > SELECT * FROM nodes FORCE
INDEX(changeset_id_idx) WHERE version < 15 ORDER BY
changeset_id;
/* type: index */
2859673 rows in set (30.92 sec)

mysql (osm) > SELECT * FROM nodes WHERE version < 15 ORDER BY
changeset_id;
/* type: ALL, Using filesort */
2859673 rows in set (16.54 sec)
```

DESC Indexes (new in 8.0)

- Regular indexes could also be used to scan rows in inverse order
  - It had a penalty
  - It improves certain queries combining DESC and ASC order

```
mysql-8.0 [dewiktionary]> ALTER TABLE page ADD INDEX
`page_redirect_namespace_len_DESC`
(`page_is_redirect`,`page_namespace`,`page_len` DESC);
Query OK, 0 rows affected, 1 warning (3.96 sec)
Records: 0  Duplicates: 0  Warnings: 1
```

Descending Indexes in Action

```
mysql-8.0 [dewiktionary]> EXPLAIN SELECT * FROM page WHERE
page_is_redirect = 1 ORDER BY page_namespace, page_len DESC\G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: page
         type: ref
possible_keys:
page_redirect_namespace_len,page_redirect_namespace_len_DESC
          key: page_redirect_namespace_len
      key_len: 1
          ref: const
         rows: 2404
        Extra: Using where
1 row in set (0.00 sec)
```

Filesort avoided

Index Condition Pushdown

```
MariaDB [dewiktionary]> ALTER TABLE revision ADD INDEX
rev_len_rev_timestamp(rev_len, rev_timestamp);
Query OK, 0 rows affected (18.11 sec)
Records: 0  Duplicates: 0  Warnings: 0

MariaDB [dewiktionary]> SET optimizer_switch='index_condition_pushdown=on'; EXPLAIN
SELECT * FROM revision WHERE rev_timestamp < '2008' AND rev_len > 10000\G
Query OK, 0 rows affected (0.00 sec)
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: revision
         type: range
possible_keys: rev_timestamp,rev_timestamp_
          key: rev_len_rev_timestamp
      key_len: 5
          ref: NULL
         rows: 180270
        Extra: Using index condition
1 row in set (0.00 sec)
```

Index condition pushdown (ICP) enables the engines to use extra parts of the index while avoiding costly row movements to and from the SQL layer

ICP optimizations

- Differences in execution time are more significant when the extra column condition is very selective (getting 5x the original performance)
- ICP is ignored when using covering Index, potentially making the performance worse

Use of "Handler_*" statistics

- They are post-execution statistics at row level
  - Unlike EXPLAIN's "rows" column, they are exact, not a guess
  - They allow comparison of query execution performance in a deterministic way, independently of the execution time

"Handler" Stats (indexed)

```
mysql> SHOW SESSION STATUS like 'Hand%';
+----------------------------+----------+
| Variable_name              | Value    |
+----------------------------+----------+
| Handler_commit             | 80       |
| Handler_delete             | 0        |
| Handler_discover           | 0        |
| Handler_external_lock      | 166      |
| Handler_mrr_init           | 0        |
| Handler_prepare            | 0        |
| Handler_read_first         | 23       |
| Handler_read_key           | 736212   |
| Handler_read_last          | 0        |
| Handler_read_next          | 22208001 |
| Handler_read_prev          | 0        |
| Handler_read_rnd           | 665215   |
| Handler_read_rnd_next      | 14223297 |
| Handler_rollback           | 0        |
| Handler_savepoint          | 0        |
| Handler_savepoint_rollback | 0        |
| Handler_update             | 66970    |
| Handler_write              | 2869409  |
+----------------------------+----------+
18 rows in set (0.00 sec)
```

Number of times that the first entry of an index was read. It may indicate the number of full index scans

Number of times a row has been retrieved using an index

Next row has been requested in index order (typical for index scans or ranges)

"Handler" Stats (unindexed)

```
mysql> SHOW SESSION STATUS like 'Hand%';
+----------------------------+----------+
| Variable_name              | Value    |
+----------------------------+----------+
| Handler_commit             | 80       |
| Handler_delete             | 0        |
| Handler_discover           | 0        |
| Handler_external_lock      | 166      |
| Handler_mrr_init           | 0        |
| Handler_prepare            | 0        |
| Handler_read_first         | 23       |
| Handler_read_key           | 736212   |
| Handler_read_last          | 0        |
| Handler_read_next          | 22208001 |
| Handler_read_prev          | 0        |
| Handler_read_rnd           | 665215   |
| Handler_read_rnd_next      | 14223297 |
| Handler_rollback           | 0        |
| Handler_savepoint          | 0        |
| Handler_savepoint_rollback | 0        |
| Handler_update             | 66970    |
| Handler_write              | 2869409  |
+----------------------------+----------+
18 rows in set (0.00 sec)
```

A row has been requested in a specific position (typical for joins or order by without indexes)

Request to read the next row in “table order” (typical for full table scans)

Insertions in SELECTS may indicate temporary tables

Comparing statistics of the previous indexes (no indexes)

```
MariaDB [dewiktionary]> FLUSH STATUS;
Query OK, 0 rows affected (0.00 sec)

MariaDB [dewiktionary]> SELECT * FROM revision IGNORE INDEX(rev_page, rev_timestamp, rev_page_rev_timestamp,
rev_timestamp_rev_page) WHERE rev_page = 31579 and rev_timestamp < '2008';
530 rows in set (0.71 sec)

MariaDB [dewiktionary]> SHOW STATUS like 'Hand%';
+-----------------------------+---------+
| Variable_name               | Value   |
+-----------------------------+---------+
| Handler_commit              | 1       |
| Handler_delete              | 0       |
…
| Handler_read_first          | 0       |
| Handler_read_key            | 0       |
| Handler_read_last           | 0       |
| Handler_read_next           | 0       |
| Handler_read_prev           | 0       |
| Handler_read_rnd            | 0       |
| Handler_read_rnd_deleted    | 0       |
| Handler_read_rnd_next       | 1430043 |
| Handler_rollback            | 0       |
…
| Handler_update              | 0       |
| Handler_write               | 0       |
+-----------------------------+---------+
27 rows in set (0.00 sec)
```

Typical result for a full table scan

Index on (rev_page)

```
MariaDB [dewiktionary]> SHOW STATUS like 'Hand%';
+----------------------------+-------+
| Variable_name              | Value |
+----------------------------+-------+
| Handler_commit             | 1     |
| Handler_delete             | 0     |
…
| Handler_read_first         | 0     |
| Handler_read_key           | 1     |
| Handler_read_last          | 0     |
| Handler_read_next          | 4038  |
| Handler_read_prev          | 0     |
| Handler_read_rnd           | 0     |
| Handler_read_rnd_deleted   | 0     |
| Handler_read_rnd_next      | 0     |
| Handler_rollback           | 0     |
…
| Handler_update             | 0     |
| Handler_write              | 0     |
+----------------------------+-------+
27 rows in set (0.01 sec)
```

Using the index, request the first row with rev_page=31579

Then, scan them one by one in index order

Index on (rev_timestamp)

```
MariaDB [dewiktionary]> SHOW STATUS like 'Hand%';
+----------------------------+---------+
| Variable_name              | Value   |
+----------------------------+---------+
| Handler_commit             | 1       |
| Handler_delete             | 0       |
…
| Handler_read_first         | 0       |
| Handler_read_key           | 1       |
| Handler_read_last          | 0       |
| Handler_read_next          | 199153  |
| Handler_read_prev          | 0       |
| Handler_read_rnd           | 0       |
| Handler_read_rnd_deleted   | 0       |
| Handler_read_rnd_next      | 0       |
| Handler_rollback           | 0       |
…
+----------------------------+---------+
27 rows in set (0.00 sec)
```

ICP will be explained later, let's ignore it for now

Using the index, request the first row where rev_timestamp<2008

Then, scan them one by one in index order (more are matched)

Index on (rev_page, rev_timestamp)

```
MariaDB [dewiktionary]> SHOW STATUS like 'Hand%';
+----------------------------+-------+
| Variable_name              | Value |
+----------------------------+-------+
| Handler_commit             | 1     |
| Handler_delete             | 0     |
…
| Handler_read_first         | 0     |
| Handler_read_key           | 1     |
| Handler_read_last          | 0     |
| Handler_read_next          | 530   |
| Handler_read_prev          | 0     |
| Handler_read_rnd           | 0     |
| Handler_read_rnd_deleted   | 0     |
| Handler_read_rnd_next      | 0     |
| Handler_rollback           | 0     |
…
| Handler_update             | 0     |
| Handler_write              | 0     |
+----------------------------+-------+
25 rows in set (0.00 sec)
```

With both conditions covered, we can find the actual first row that matches the condition using the index

Rows scanned == Rows returned

Index on (rev_timestamp, rev_page), no ICP

```
MariaDB [dewiktionary]> SHOW STATUS like 'Hand%';
+----------------------------+----------+
| Variable_name              | Value    |
+----------------------------+----------+
| Handler_commit             | 1        |
| Handler_delete             | 0        |
…
| Handler_read_first         | 0        |
| Handler_read_key           | 1        |
| Handler_read_last          | 0        |
| Handler_read_next          | 452539   |
| Handler_read_prev          | 0        |
| Handler_read_rnd           | 0        |
| Handler_read_rnd_deleted   | 0        |
| Handler_read_rnd_next      | 0        |
| Handler_rollback           | 0        |
…
| Handler_update             | 0        |
| Handler_write              | 0        |
+----------------------------+----------+
27 rows in set (0.00 sec)
```

Assuming no ICP, exact same results as with (rev_timestamp). The extra column does not help. Also, EXPLAIN's row count was very off.

Index on (rev_timestamp, rev_page), with ICP

```
MariaDB [dewiktionary]> SHOW STATUS like 'Hand%';
+----------------------------+--------+
| Variable_name              | Value  |
+----------------------------+--------+
| Handler_commit             | 1      |
| Handler_delete             | 0      |
…
| Handler_icp_attempts       | 452539 |
| Handler_icp_match          | 530    |
…
| Handler_read_first         | 0      |
| Handler_read_key           | 1      |
| Handler_read_last          | 0      |
| Handler_read_next          | 530    |
| Handler_read_prev          | 0      |
| Handler_read_rnd           | 0      |
| Handler_read_rnd_deleted   | 0      |
| Handler_read_rnd_next      | 0      |
| Handler_rollback           | 0      |
…
| Handler_update             | 0      |
| Handler_write              | 0      |
+----------------------------+--------+
27 rows in set (0.00 sec)
```

ICP reduces the number of ‘ENGINE API calls’ significantly, although making it work more internally

Redundant Indexes

- Creating all 4 previous indexes in production is not a great idea
  - "Left-most index prefix" allows, for example `(rev_page, rev_timestamp)` doing everything you can do with `(rev_page)`
  - If two indexes have equal selectivity, MySQL chooses the shortest one

"Left-most index" Example

```
MariaDB [dewiktionary]> EXPLAIN SELECT * FROM revision FORCE
INDEX(rev_page_rev_timestamp) WHERE rev_page = 31579\G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: revision
         type: ref
possible_keys: rev_page_rev_timestamp
          key: rev_page_rev_time
      key_len: 4
          ref: const
         rows: 4038
        Extra:
1 row in set (0.00 sec)
```

Only the first column is used

Duplicate Indexes

- It is very easy to create indexes with the same exact definition (same columns and ordering)
  - Set a convention for index naming (e.g `tablename_column1_column2_idx`) – MySQL does not allow 2 indexes with the same identifier
  - Since MySQL 5.6, an warning is thrown if a duplicate index is created:

Still a warning on 8.0

```
Duplicate index 'page_random2' defined on the table
'dewiktionary.page'. This is deprecated and will be disallowed
in a future release.
```

pt-duplicate-index-checker

```
$ pt-duplicate-key-checker h=localhost,u=root,D=dewiktionary
[…]
# rev_timestamp is a left-prefix of rev_timestamp_rev_page
# Key definitions:
#   KEY `rev_timestamp` (`rev_timestamp`),
#   KEY `rev_timestamp_rev_page` (`rev_timestamp`,`rev_page`)
# Column types:
#    `rev_timestamp` binary(14) not null default '\0\0\0\0\0\0\0\0\0
#    `rev_page` int(10) unsigned not null
# To remove this duplicate index, execute:
ALTER TABLE `dewiktionary`.`revision` DROP INDEX `rev_timestamp`;

# rev_page is a left-prefix of rev_page_rev_timestamp
# Key definitions:
#   KEY `rev_page` (`rev_page`),
#   KEY `rev_page_rev_timestamp` (`rev_page`,`rev_timestamp`),
# Column types:
#    `rev_page` int(10) unsigned not null
#    `rev_timestamp` binary(14) not null default '\0\0\0\0\0\0\0\0\0\0\0\0\0\0'
# To remove this duplicate index, execute:
ALTER TABLE `dewiktionary`.`revision` DROP INDEX `rev_page`;

#  ############################################# Summary of indexes  ########################################
# Size Duplicate Indexes   47836942
# Total Duplicate Indexes  4
# Total Indexes            285
```

Simple tool to check redundant and duplicate indexes

Invisible Indexes (new in 8.0)

- Before removing a (believed) redundant or duplicate index, make it “invisible” to prevent long-running outages:

```
mysql-8.0 [dewiktionary]> ALTER TABLE page ALTER INDEX
page_len INVISIBLE;
Query OK, 0 rows affected (0.04 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql-8.0 [dewiktionary]> SELECT * FROM page FORCE
INDEX(page_len);
ERROR 1176 (42000): Key 'page_len' doesn't exist in table
'page'
```

"OR"-style conditions over the same column

```
MariaDB [dewiktionary]> EXPLAIN SELECT * FROM revision WHERE
rev_page = 790 OR rev_page = 795 OR rev_page = 1024\G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: revision
         type: range
possible_keys: rev_page,rev_page_rev_timestamp
          key: rev_page
      key_len: 4
          ref: NULL
         rows: 201
        Extra: Using index condition; Using where
1 row in set (0.01 sec)
```

Equivalent to:
SELECT * FROM revision WHERE rev_page IN (790, 795, 1024)

Handlers on "IN" / "OR" conditions over the same column

```
+----------------------------+-------+
| Variable_name              | Value |
+----------------------------+-------+
| Handler_commit             | 1     |
| Handler_delete             | 0     |
…
| Handler_prepare            | 0     |
| Handler_read_first         | 0     |
| Handler_read_key           | 3     |
| Handler_read_last          | 0     |
| Handler_read_next          | 201   |
| Handler_read_prev          | 0     |
| Handler_read_rnd           | 0     |
| Handler_read_rnd_deleted   | 0     |
| Handler_read_rnd_next      | 0     |
| Handler_rollback           | 0     |
…
| Handler_update             | 0     |
| Handler_write              | 0     |
+----------------------------+-------+
25 rows in set (0.00 sec)
```

Despite identifying themselves as “range”s, the execution is slightly different, one index dive (similar to a ref) is done per value. This can be an issue in conditions with thousands of items.

"OR"-style conditions over different columns

- We cannot use a single index efficiently for both conditions
  - We can scan both conditions separately and mix the results, discarding duplicates
  - Or use an index for one condition and not for the other
  - Index merge allows the use of two indexes for a single table access simultaneously

Index Merge Example

```
MariaDB [dewiktionary]> EXPLAIN SELECT * FROM revision WHERE
rev_page = 790 or rev_timestamp < '2004'\G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: revision
         type: index_merge
possible_keys: rev_page,rev_timestamp,rev_page_rev_timestamp,
rev_timestamp_rev_page
          key: rev_page,rev_timestar
      key_len: 4,14
          ref: NULL
         rows: 190
        Extra: Using sort_union(rev_page,rev_timestamp); Using
where
1 row in set (0.00 sec)
```

Both indexes are used, then combined using the "union" operation

INDEX_MERGE Issues

- Sometimes it is faster to to execute the sentence using UNION:
  - This is especially true with (UNION ALL) since MySQL 5.7, if you do not care or expect duplicates
- There are also intersection merges, but multi-column indexes are preferred

Disabling optimizer features (I)

- The optimizer_switch variable allows enabling and disabling globally or per session many query optimizer features:

```
MariaDB [nlwiktionary]> SHOW VARIABLES like 'optimizer_switch'\G
*************************** 1. row ***************************
Variable_name: optimizer_switch
Value:
index_merge=on,index_merge_union=on,index_merge_sort_union=on,index_merge_intersection=
on,index_merge_sort_intersection=off,engine_condition_pushdown=off,index_condition_push
down=on,derived_merge=on,derived_with_keys=on,firstmatch=on,loosescan=on,materializatio
n=on,in_to_exists=on,semijoin=on,partial_match_rowid_merge=on,partial_match_table_scan=
on,subquery_cache=on,mrr=off,mrr_cost_based=off,mrr_sort_keys=off,outer_join_with_cache
=on,semijoin_with_cache=on,join_cache_incremental=on,join_cache_hashed=on,join_cache_bk
a=on,optimize_join_buffer_size=off,table_elimination=on,extended_keys=on,exists_to_in=o
n,orderby_uses_equalities=on,condition_pushdown_for_derived=on,split_materialized=on
1 row in set (0.00 sec)
```

Disabling optimizer features (II)

```
MariaDB [dewiktionary]> SET optimizer_switch='index_merge_sort_union=off';
Query OK, 0 rows affected (0.00 sec)

MariaDB [dewiktionary]> EXPLAIN SELECT * F
790 or rev_timestamp < '2004'\G
*************************** 1. row *******
           id: 1
  select_type: SIMPLE
        table: revision
         type: ALL
possible_keys:
rev_page,rev_timestamp,rev_page_rev_timestamp,rev_timestamp_rev_page
          key: NULL
      key_len: NULL
          ref: NULL
         rows: 1376651
        Extra: Using where
1 row in set (0.00 sec)
```

This will only have effect for the current session.

What happens if we have two ranges?

- As seen previously, we cannot use efficiently two range types on the same table access. Alternatives:
  - Use only one index for the most selective column
  - Use index condition pushdown to get an advantage
  - Change one of the two ranges into a discrete "IN" comparison/bucketize with a new column
  - Use quadtrees or R-TREEs (spatial indexing)

Example of Bucketizing (I)

```
MariaDB [dewiktionary]> ALTER TABLE revision DROP INDEX
rev_len_rev_timestamp;
MariaDB [dewiktionary]> EXPLAIN SELECT count(*) FROM revision
WHERE rev_timestamp < '2008' AND rev_len > 5500\G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: revision
         type: ALL
possible_keys: rev_timestamp,rev_timestamp_rev_page
          key: NULL
      key_len: NULL
          ref: NULL
         rows: 1376651
        Extra: Using where
1 row in set (0.00 sec)
```

Looks like only an index on (rev_timestamp) or (rev_len) would be useful as we have 2 ranges.

Example of Bucketizing (II)

```
MariaDB [dewiktionary]> ALTER TABLE revision ADD
rev_len_cat int;
Query OK, 0 rows affected (0.01 sec)
Records: 0  Duplicates: 0  Warnings: 0

MariaDB [dewiktionary]> UPDATE revision set rev_len_cat =
IF(rev_len < 10000, rev_len div 1000, 10);
Query OK, 1430042 rows affected (21.18 sec)
Rows matched: 1430042  Changed: 1430042  Warnings: 0

MariaDB [dewiktionary]> ALTER TABLE revision ADD INDEX
rev_len_cat_rev_timestamp (rev_len_cat, rev_timestamp);
Query OK, 0 rows affected (7.77 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

8.0/10.3
instant
alter-nice!

Example of Bucketizing (III)

```
MariaDB [dewiktionary]> EXPLAIN SELECT count(*) FROM revision WHERE
rev_timestamp < '2008' AND rev_len > 5500 AND rev_len_cat IN (5, 6,
7, 8, 9, 10)\G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: revision
         type: range
possible_keys:
rev_timestamp,rev_timestamp_rev_page,rev_len_cat_rev_timestamp
          key: rev_len_cat_rev_timestamp
      key_len: 19
          ref: NULL
         rows: 77601
        Extra: Using where
1 row in set (0.00 sec)
```

We did some transformations to both the structure and the query.

Even better: Functional Indexes (I)

```
MariaDB [dewiktionary]> ALTER TABLE revision
DROP INDEX rev_len_cat_rev_timestamp,
DROP COLUMN rev_len_cat,
ADD COLUMN rev_len_cat_virtual int as (IF(rev_len < 10000,
rev_len div 1000, 10)) VIRTUAL,
ADD INDEX rev_len_cat_virtual_rev_timestamp
(rev_len_cat_virtual, rev_timestamp);
Query OK, 1430042 rows affected (1 min 42.35 sec)
Records: 1430042  Duplicates: 0  Warnings: 0
```

Indexing virtual columns, new in 10.3

Storing the extra content may be a waste of resources, lets make it a virtual column

Even better: Functional Indexes (II)

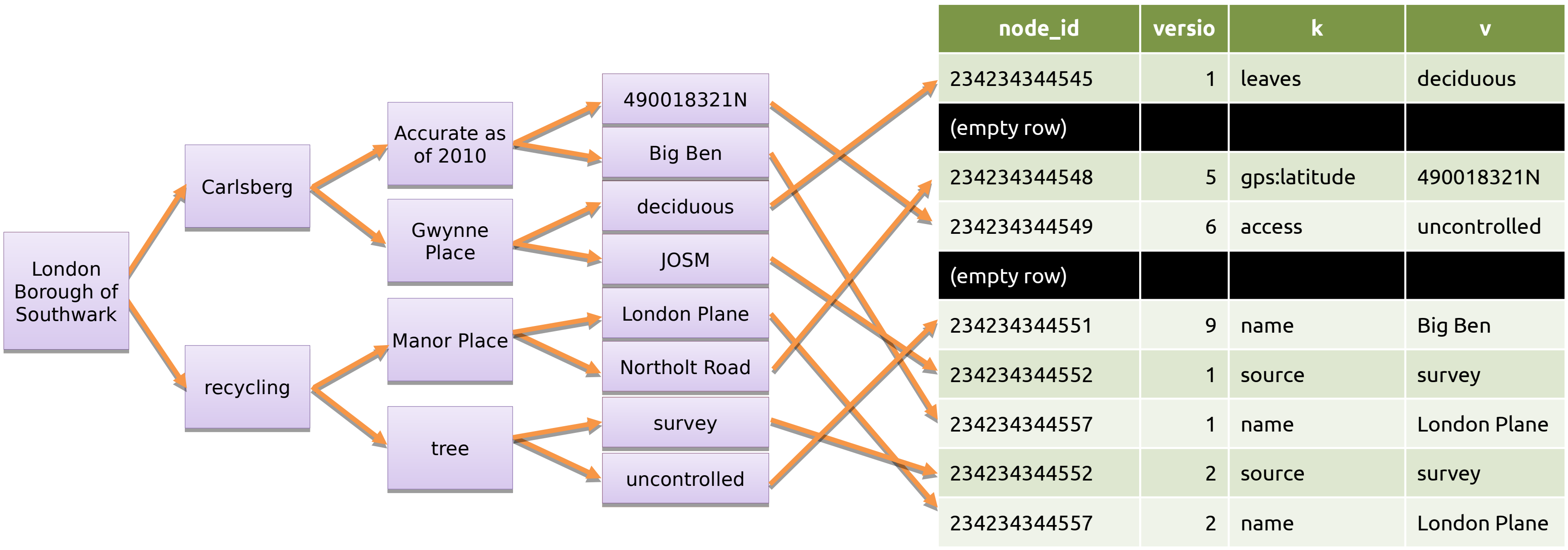
```
MariaDB [dewiktionary]> EXPLAIN SELECT count(*) FROM revision WHERE
rev_timestamp < '2008' AND rev_len > 5500 AND rev_len_cat_virtual IN
(5, 6, 7, 8, 9, 10)\G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: revision
         type: range
possible_keys:
rev_timestamp,rev_timestamp_rev_page,rev_len_cat_virtual_rev_timestamp
          key: rev_len_cat_virtual_rev_timestamp
      key_len: 19
          ref: NULL
         rows: 78649
        Extra: Using where
1 row in set (0.01 sec)
```

We did some transformations to both the structure and the query.

MyISAM Internals

Index (part of revision.MYI)

Data (revision.MYD)



London Borough of Southwark

Carlsberg

recycling

Accurate as of 2010

Gwynne Place

Manor Place

tree

490018321N

Big Ben

deciduous

JOSM

London Plane

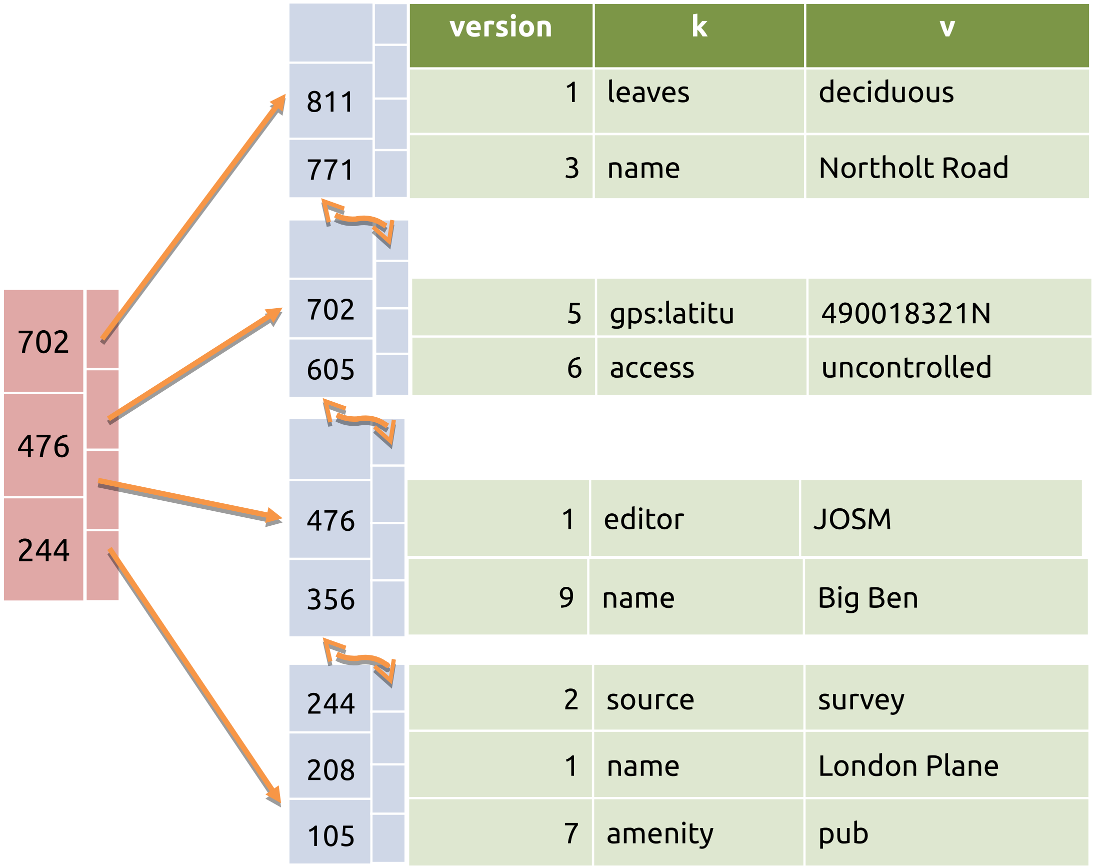
Northolt Road

survey

uncontrolled

node_id	versio	k	v
234234344545	1	leaves	deciduous
(empty row)			
234234344548	5	gps:latitude	490018321N
234234344549	6	access	uncontrolled
(empty row)			
234234344551	9	name	Big Ben
234234344552	1	source	survey
234234344557	1	name	London Plane
234234344552	2	source	survey
234234344557	2	name	London Plane

InnoDB Internals (PRIMARY)

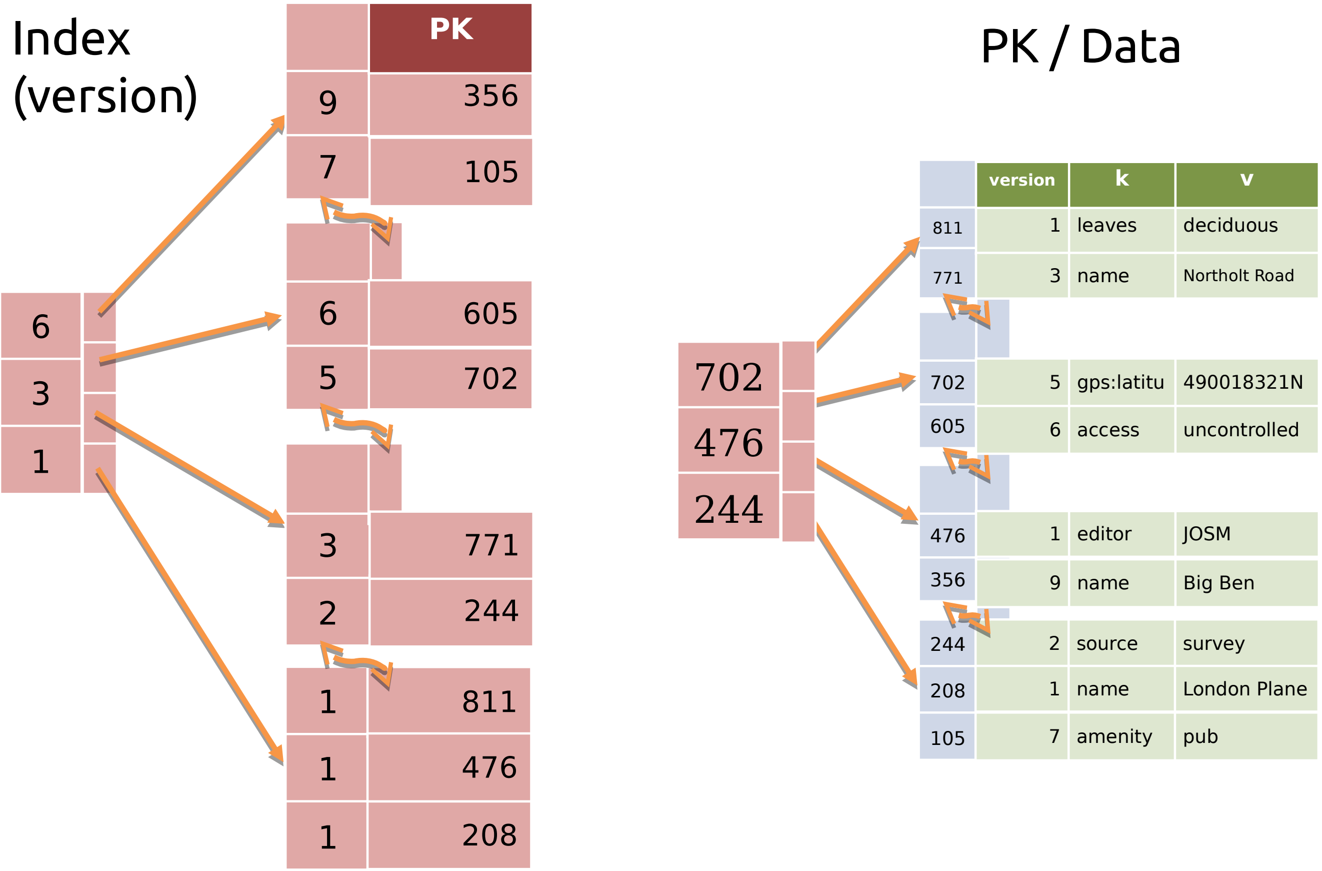


	version	k	v
811	1	leaves	deciduous
771	3	name	Northolt Road
702	5	gps:latitu	490018321N
605	6	access	uncontrolled
476	1	editor	JOSM
356	9	name	Big Ben
244	2	source	survey
208	1	name	London Plane
105	7	amenity	pub

Data clustered always using the primary key

InnoDB Internals (Secondary)

Secondary indexes contain the primary key value



6
3
1

	PK
9	356
7	105
6	605
5	702
3	771
2	244
1	811
1	476
1	208

702
476
244

	version	k	v
811	1	leaves	deciduous
771	3	name	Northolt Road
702	5	gps:latitu	490018321N
605	6	access	uncontrolled
476	1	editor	JOSM
356	9	name	Big Ben
244	2	source	survey
208	1	name	London Plane
105	7	amenity	pub

Consequences of using InnoDB (I)

- Every table should have a primary key
  - If one is not defined, MySQL will choose an available NOT NULL unique key
  - If that is not possible, an internal 6-byte row identifier will be generated (not user-accesible)

Consequences of using InnoDB (II)

- Inserting in primary key order is much faster
  - Less fragmentation/page-split
  - Usage of "batch" mode, improving insert speed
- Using auto-increment keys as primary keys can be a good idea for InnoDB

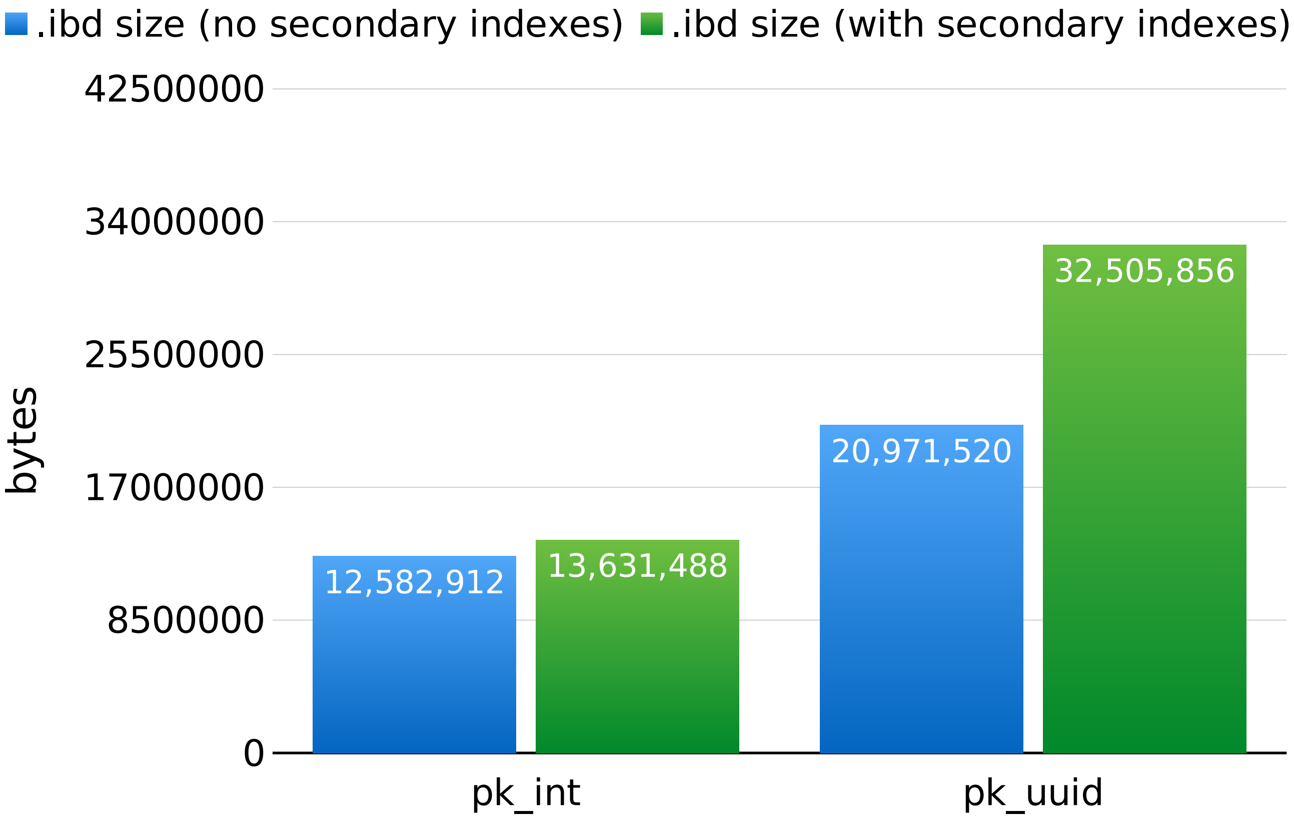
Consequences of using InnoDB (III)

- A very long primary key may increment substantially the size of secondary keys
  - Int or bigint types are recommended instead of UUIDs or other long strings

Differences in size

```
mysql (osm) > CREATE TABLE
pk_int (id int PRIMARY KEY
auto_increment,
a int,
b int,
c int,
d int);
Query OK, 0 rows affected
(0.16 sec)

mysql (osm) > CREATE TABLE
pk_uuid (id char(36)
PRIMARY KEY,
a int,
b int,
c int,
d int);
Query OK, 0 rows affected
(0.04 sec)
```



Extended primary key optimization

- As the primary key is part of all secondary keys, this can be used “for free”:
  - For row filtering (since MySQL 5.6)
  - To return results in primary key order
  - To avoid reading data from the table (covering index)

Extended Primary Key Example

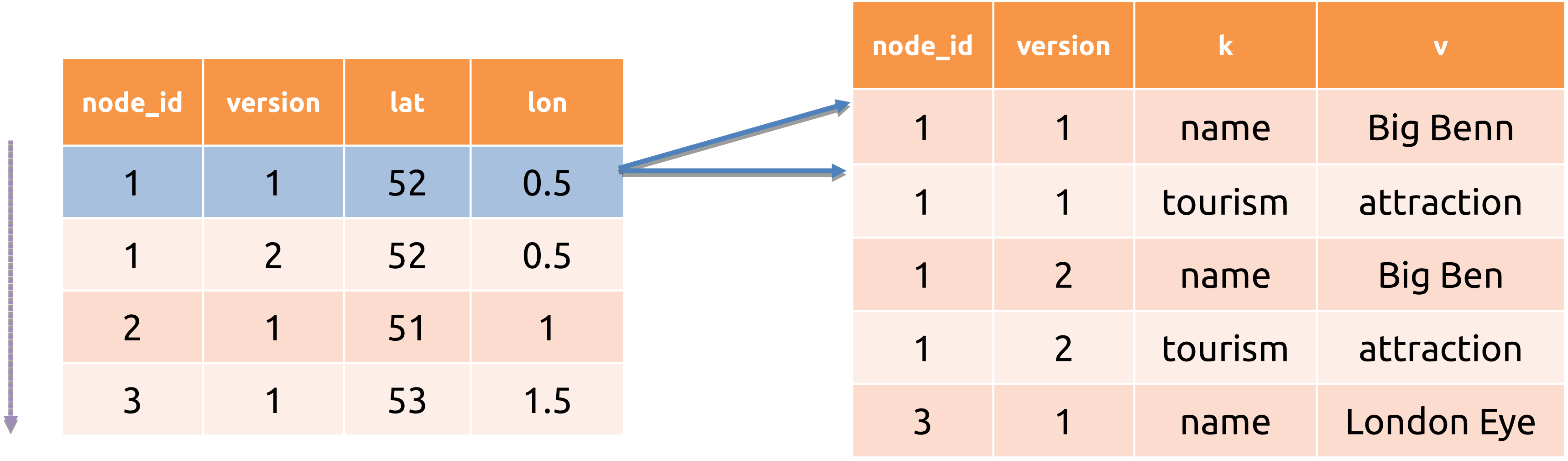
```
mysql (osm) > EXPLAIN SELECT node_id FROM nodes WHERE
changeset_id = 24284 and node_id <> 146472942\G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: nodes
         type: range
possible_keys: PRIMARY,changeset_id_idx
          key: changeset_id_idx
      key_len: 16
          ref: NULL
         rows: 50
        Extra: Using where; Using index
1 row in set (0.07 sec)
```

Query Optimization with MySQL 8.0 and MariaDB 10.3: The Basics

JOINS

(Block) Nested Loop Join

- Until MySQL 5.5 there was only one algorithm to execute a JOIN:



node_id	version	lat	lon
1	1	52	0.5
1	2	52	0.5
2	1	51	1
3	1	53	1.5

node_id	version	k	v
1	1	name	Big Benn
1	1	tourism	attraction
1	2	name	Big Ben
1	2	tourism	attraction
3	1	name	London Eye

Extra Access type: eq_ref

```
mysql (osm) > EXPLAIN SELECT * FROM nodes JOIN node_tags
                  USING(node_id, version) WHERE node_tags.v= 'Big Ben'\G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: node_tags
         type: ref
possible_keys: PRIMARY,v_idx
          key: v_idx
      key_len: 767
          ref: const
         rows: 1
        Extra: Using where; Using index
*************************** 2. row ***********************
           id: 1
  select_type: SIMPLE
        table: nodes
         type: eq_ref
possible_keys: PRIMARY,version_idx
          key: PRIMARY
      key_len: 16
          ref: osm.node_tags.node_id,osm.node_tags.version
         rows: 1
        Extra: NULL
2 rows in set (0.00 sec)
```

eq_ref is similar to ref, but allows faster JOINS because, by using a unique key, it only has to search one row for each previous result

JOIN Optimization

- Two main goals:
  - Perform an effective filtering on each table access, if possible using indexes
  - Perform the access in the most efficient table order
- When joining 3 or more tables in a star schema, the "covering index" strategy can have a huge impact

Example: optimize this JOIN (I)

```
SELECT n.node_id, n.latitude, n.longitude
  FROM way_nodes w_n
  JOIN way_tags w_t
    ON w_n.way_id = w_t.way_id and
       w_n.version = w_t.version
  JOIN nodes n
    ON w_n.node_id = n.node_id
  JOIN node_tags n_t
    ON n.node_id = n_t.node_id and
       n.version = n_t.version
 WHERE w_t.k = 'building' and
       n_t.k = 'entrance' and
       n_t.v = 'main';
```

We start without secondary indexes

Example: optimize this JOIN (II)

```
************************ 1. row ************************
           id: 1
  select_type: SIMPLE
        table: w_t
         type: index
possible_keys: PRIMARY
          key: PRIMARY
      key_len: 783
          ref: NULL
         rows: 1335702
        Extra: Using where; Using index
************************ 2. row ************************
           id: 1
  select_type: SIMPLE
        table: w_n
         type: ref
possible_keys: PRIMARY
          key: PRIMARY
      key_len: 16
          ref: osm.w_t.way_id,osm.w_t.version
         rows: 3
        Extra: NULL
************************ 3. row ************************
           id: 1
  select_type: SIMPLE
        table: n_t
         type: ref
possible_keys: PRIMARY
          key: PRIMARY
      key_len: 8
          ref: osm.w_n.node_id
         rows: 1
        Extra: Using where
************************ 4. row ************************
           id: 1
  select_type: SIMPLE
        table: n
         type: eq_ref
possible_keys: PRIMARY
          key: PRIMARY
      key_len: 16
          ref: osm.w_n.node_id,osm.n_t.version
         rows: 1
        Extra: Using index
4 rows in set (0.01 sec)
```

```
mysql (osm) > SELECT …
858 rows in set (9.00 sec)
```

Example: optimize this JOIN (III)

```
mysql (osm) > ALTER TABLE way_tags
              ADD INDEX k_idx(k);
Query OK, 0 rows affected (4.80 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

Creating an index on way_tags.k

Example: optimize this JOIN (IV)

```
*********************** 1. row ***********************
           id: 1
  select_type: SIMPLE
        table: w_t
         type: ref
possible_keys: PRIMARY,k_idx
          key: k_idx
      key_len: 767
          ref: const
         rows: 452274
        Extra: Using where; Using index
*********************** 2. row ***********************
           id: 1
  select_type: SIMPLE
        table: w_n
         type: ref
possible_keys: PRIMARY
          key: PRIMARY
      key_len: 16
          ref: osm.w_t.way_id,osm.w_t.version
         rows: 3
        Extra: NULL
*********************** 3. row ***********************
           id: 1
  select_type: SIMPLE
        table: n_t
         type: ref
possible_keys: PRIMARY
          key: PRIMARY
      key_len: 8
          ref: osm.w_n.node_id
         rows: 1
        Extra: Using where
*********************** 4. row ***********************
           id: 1
  select_type: SIMPLE
        table: n
         type: eq_ref
possible_keys: PRIMARY
          key: PRIMARY
      key_len: 16
          ref: osm.w_n.node_id,osm.n_t.version
         rows: 1
        Extra: NULL
4 rows in set (0.00 sec)
```

```
mysql (osm) > SELECT …
858 rows in set (8.58 sec)
```

It seems like the index is not very useful

Example: optimize this JOIN (V)

```
mysql (osm) > ALTER TABLE node_tags
              ADD INDEX k_idx(k);
Query OK, 0 rows affected (2.82 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

The order does not seem to be adequate, let's try adding an index to start by accessing node_tags

Example: optimize this JOIN (VI)

```
********************* 1. row *********************
           id: 1
  select_type: SIMPLE
        table: w_t
         type: ref
possible_keys: PRIMARY,k_idx
          key: k_idx
      key_len: 767
          ref: const
         rows: 452274
        Extra: Using where; Using index
********************* 2. row *********************
           id: 1
  select_type: SIMPLE
        table: w_n
         type: ref
possible_keys: PRIMARY
          key: PRIMARY
      key_len: 16
          ref: osm.w_t.way_id,osm.w_t.version
         rows: 3
        Extra: NULL
********************* 3. row *******
           id: 1
  select_type: SIMPLE
        table: n_t
         type: ref
possible_keys: PRIMARY,k_idx
          key: PRIMARY
      key_len: 8
          ref: osm.w_n.node_id
         rows: 1
        Extra: Using where
********************* 4. row *******
           id: 1
  select_type: SIMPLE
        table: n
         type: eq_ref
possible_keys: PRIMARY
          key: PRIMARY
      key_len: 16
          ref: osm.w_n.node_id,osm.n_t.version
         rows: 1
        Extra: NULL
4 rows in set (0.00 sec)
```

```
mysql (osm) > SELECT …
858 rows in set (7.33 sec)
```

It keeps using the wrong order, even if we delete the w_t.k_idx index

Example: optimize this JOIN (VII)

```
SELECT STRAIGHT_JOIN n.node_id, n.latitude, n.longitude
  FROM node_tags n_t
  JOIN nodes n
    ON n.node_id = n_t.node_id and
       n.version = n_t.version
  JOIN way_nodes w_n
    ON w_n.node_id = n.node_id
  JOIN way_tags w_t
    ON w_n.way_id = w_t.way_id and
       w_n.version = w_t.version
 WHERE w_t.k = 'building' and
       n_t.k = 'entrance' and
       n_t.v = 'main';
```

Let's see why rewriting it into this query

Example: optimize this JOIN (VIII)

```
********************** 1. row **********************
           id: 1
  select_type: SIMPLE
        table: n_t
         type: ref
possible_keys: PRIMARY,k_idx
          key: k_idx
      key_len: 767
          ref: const
         rows: 2390
        Extra: Using index condition; Using where
********************** 2. row **********************
           id: 1
  select_type: SIMPLE
        table: n
         type: eq_ref
possible_keys: PRIMARY
          key: PRIMARY
      key_len: 16
          ref: osm.n_t.node_id,osm.n_t.version
         rows: 1
        Extra: NULL
```

```
********************** 3. row **********************
           id: 1
  select_type: SIMPLE
        table: w_n
         type: ALL
possible_keys: PRIMARY
          key: NULL
      key_len: NULL
          ref: NULL
         rows: 3597858
        Extra: Using where; Using join buffer (Block
Nested Loop)
********************** 4. row **********************
           id: 1
  select_type: SIMPLE
        table: w_t
         type: eq_ref
possible_keys: PRIMARY
          key: PRIMARY
      key_len: 783
          ref: osm.w_n.way_id,osm.w_n.version,const
         rows: 1
        Extra: Using where; Using index
4 rows in set (0.00 sec)
```

There is no index on w_n that would allow efficient access

Example: optimize this JOIN (IX)

```
mysql (osm) > ALTER TABLE way_nodes
              ADD INDEX node_id_idx(node_id);
Query OK, 0 rows affected (17.77 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

Example: optimize this JOIN (X)

```
********************** 1. row **********************
           id: 1
  select_type: SIMPLE
        table: n_t
         type: ref
possible_keys: PRIMARY,k_idx
          key: k_idx
      key_len: 767
          ref: const
         rows: 2390
        Extra: Using index condition; Using where
********************** 2. row **********************
           id: 1
  select_type: SIMPLE
        table: n
         type: eq_ref
possible_keys: PRIMARY
          key: PRIMARY
      key_len: 16
          ref: osm.n_t.node_id,osm.n_t.version
         rows: 1
        Extra: NULL
```

Now it starts with the right table (without STRAIGHT_JOIN)

```
********************** 3. row **********************
           id: 1
  select_type: SIMPLE
        table: w_n
         type: ref
possible_keys: PRIMARY,node_id_idx
          key: node_id_idx
      key_len: 8
          ref: osm.n_t.node_id
         rows: 1
        Extra: Using index
********************** 4. row **********************
           id: 1
  select_type: SIMPLE
        table: w_t
         type: eq_ref
possible_keys: PRIMARY
          key: PRIMARY
      key_len: 783
          ref: osm.w_n.way_id,osm.w_n.version,const
         rows: 1
        Extra: Using where; Using index
4 rows in set (0.04 sec)
```

```
mysql (osm) > SELECT …
858 rows in set (0.73 sec)
```

Example: optimize this JOIN (XI)

```
************************* 1. row *************************
           id: 1
  select_type: SIMPLE
        table: n_t
         type: ref
possible_keys: PRIMARY,k_idx,k_v_i
          key: k_v_idx
      key_len: 1534
          ref: const,const
         rows: 900
        Extra: Using where; Using index
************************* 2. row *************************
           id: 1
  select_type: SIMPLE
        table: n
         type: eq_ref
possible_keys: PRIMARY
          key: PRIMARY
      key_len: 16
          ref: osm.n_t.node_id,osm.n_t.version
         rows: 1
        Extra: NULL
```

```
************************* 3. row *************************
           id: 1
  select_type: SIMPLE
        table: w_n
         type: ref
possible_keys: PRIMARY,node_id_idx
          key: node_id_idx
      key_len: 8
          ref: osm.n_t.node_id
         rows: 1
        Extra: Using index
************************* 4. row *************************
           id: 1
  select_type: SIMPLE
        table: w_t
         type: eq_ref
possible_keys: PRIMARY
          key: PRIMARY
      key_len: 783
          ref: osm.w_n.way_id,osm.w_n.version,const
         rows: 1
        Extra: Using where; Using index
4 rows in set (0.00 sec)
```

```
mysql (osm) > SELECT ...
858 rows in set (0.02 sec)
```

An index on (k,v) is even better

“New” JOIN methods

- MySQL 5.6 added:
  - Batch Key Access
- MariaDB has since 5.3:
  - Batch Key Access
  - Hash Joins
  - Slightly modified versions of the above ones (with “incremental” buffers to join 3 or more tables)

Multi-range read

- This optimization orders results obtained from a secondary key in primary key/physical order before accessing the rows
  - It may help execution time of queries when disk-bound
  - It requires tuning of the read_rnd_buffer_size (size of the buffer used for ordering the results)
- BKA JOINs are based on the mrr optimization

MRR Example (I)

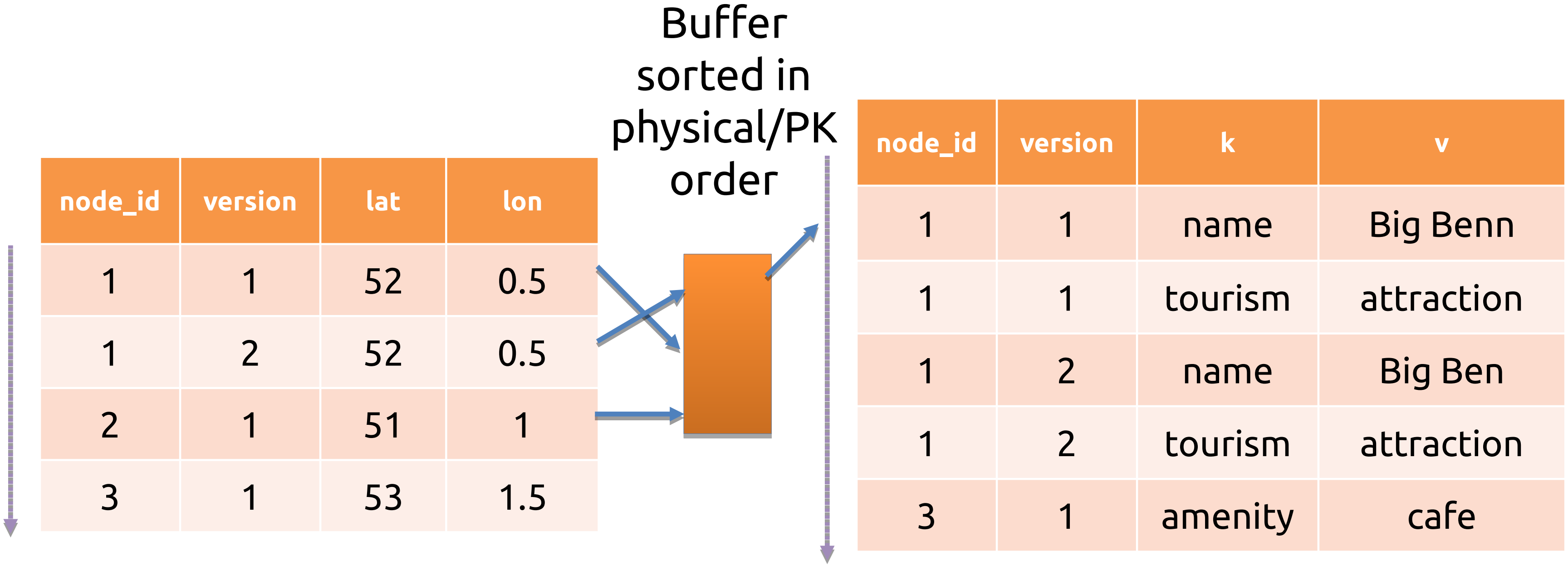
```
mysql (osm) > EXPLAIN SELECT * FROM nodes WHERE timestamp >= '2013-07-
01 00:00:00' AND timestamp < '2014-01-01 00:00:00'\G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: nodes
         type: range
possible_keys: nodes_timestamp_idx
          key: nodes_timestamp_idx
      key_len: 5
          ref: NULL
         rows: 429684
        Extra: Using index condition; Using MRR
1 row in set (0.02 sec)
```

MRR example (II)

```
[restart]
mysql> SET optimizer_switch='mrr=off';
mysql> SELECT * FROM nodes WHERE timestamp >= '2013-07-01 00:00:00' AND timestamp <
'2014-01-01 00:00:00';
205617 rows in set (5.16 sec)
mysql> SELECT * FROM nodes WHERE timestamp >= '2013-07-01 00:00:00' AND timestamp <
'2014-01-01 00:00:00';
205617 rows in set (0.60 sec)
[restart]
mysql> SET read_rnd_buffer_size=50 * 1024 * 1024;
mysql> SELECT * FROM nodes WHERE timestamp >= '2013-07-01 00:00:00' AND timestamp <
'2014-01-01 00:00:00';
205617 rows in set (2.39 sec)
mysql> SELECT * FROM nodes WHERE timestamp >= '2013-07
'2014-01-01 00:00:00';
205617 rows in set (0.73 sec)
```

“Cold” results are significantly better with mrr (but it can impact negatively, too)

Batch Key Access



Buffer sorted in physical/PK order

node_id	version	lat	lon
1	1	52	0.5
1	2	52	0.5
2	1	51	1
3	1	53	1.5

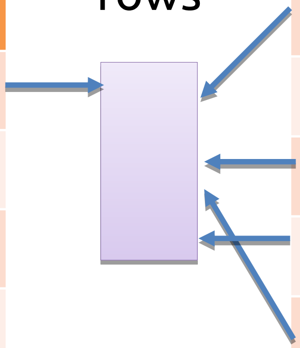
node_id	version	k	v
1	1	name	Big Benn
1	1	tourism	attraction
1	2	name	Big Ben
1	2	tourism	attraction
3	1	amenity	cafe

Hash Joins

- Only work for equi-joins

node_id	version	lat	lon
1	1	52	0.5
1	2	52	0.5
2	1	51	1
3	1	53	1.5

Hash table for faster access to rows



node_id	version	k	v
1	1	name	Big Benn
1	1	tourism	attraction
1	2	name	Big Ben
1	2	tourism	attraction
3	1	amenity	cafe

MySQL Configuration

- BKA requires changes of default optimizer configuration:

```
mysql-8.0 (osm) > SET optimizer_switch= 'mrr=on';
mysql-8.0 (osm) > SET optimizer_switch= 'mrr_cost_based=off';
mysql-8.0 (osm) > SET optimizer_switch= 'batch_key_access=on';
```

  - Additionally, configuring the join_buffer_size adequately

MariaDB configuration

```
mariadb-10.3 (osm) > SET optimizer_switch = 'join_cache_incremental=on';
mariadb-10.3 (osm) > SET optimizer_switch = 'join_cache_hashed=on';
mariadb-10.3 (osm) > SET optimizer_switch = 'join_cache_bka=on';
```

- Enabled by default

```
mariadb-10.3 (osm) > SET join_cache_level = 3 (for hash joins)
mariadb-10.3 (osm) > SET join_cache_level = 5 (for BKA)
```

- Also, configure join_buffer_size appropriately.
- Hash joins, like BKA, are highly dependent on disk-bound DBs to be effective due to the extra overhead

Nested Loop Join (cold buffers buffer_pool=100MB, join_buffer=4M)

```
mariadb-10.3 (osm) > EXPLAIN SELECT changeset_id,
count(*) FROM changesets JOIN nodes on
changesets.id = nodes.changeset_id GROUP BY
visible\G
******************** 1. row ********************
           id: 1
  select_type: SIMPLE
        table: changesets
         type: index
possible_keys: PRIMARY
          key: PRIMARY
      key_len: 8
          ref: NULL
         rows: 69115
        Extra: Using index; Using temporary; Using
filesort
******************** 2. row ********************
           id: 1
  select_type: SIMPLE
        table: nodes
         type: ref
possible_keys: changeset_id
          key: changeset_id
      key_len: 8
          ref: osm.changesets.id
         rows: 19
        Extra:
2 rows in set (0.00 sec)

mariadb-10.3 (osm) > SELECT visible, count(*) FROM
changesets JOIN nodes on changesets.id =
nodes.changeset_id GROUP BY visible;
+---------+----------+
| visible | count(*) |
+---------+----------+
|       1 |  2865312 |
+---------+----------+
1 row in set (32.86 sec)
```

Hash Join (cold buffers, buffer_pool=100M, join_buffer=4M)

```
mariadb-10.3 (osm) > EXPLAIN SELECT
changeset_id, count(*) FROM changesets JOIN
nodes on changesets.id = nodes.changeset_id
GROUP BY visible\G
******************* 1. row *******************
           id: 1
  select_type: SIMPLE
        table: changesets
         type: index
possible_keys: PRIMARY
          key: PRIMARY
      key_len: 8
          ref: NULL
         rows: 69115
        Extra: Using index; Using temporary;
Using filesort
```

```
******************* 2. row *******************
           id: 1
  select_type: SIMPLE
        table: nodes
         type: hash_ALL
possible_keys: changeset_id
          key: #hash#changeset_id
      key_len: 8
          ref: osm.changesets.id
         rows: 2781732
        Extra: Using join buffer (flat, BNLH join)
2 rows in set (0.00 sec)

mariadb-10.3 (osm) > SELECT visible, count(*) FROM
changesets JOIN nodes on changesets.id =
nodes.changeset_id GROUP BY visible;
+---------+----------+
| visible | count(*) |
+---------+----------+
|       1 |  2865312 |
+---------+----------+
1 row in set (6.66 sec)
```

Query Optimization with MySQL 8.0 and MariaDB 10.3: The Basics

SUBQUERIES

Access types: unique_subquery/index_subquery

```
mysql-8.0 (osm) > EXPLAIN SELECT *
FROM node_tags WHERE v = 'Big Ben'
and node_id NOT IN (SELECT node_id
FROM nodes WHERE tile < 100000000)\G
************** 1. row **************
           id: 1
  select_type: PRIMARY
        table: node_tags
         type: ref
possible_keys: v_idx
          key: v_idx
      key_len: 767
          ref: const
         rows: 1
        Extra: Using where; Using
index
************** 2. row **************
           id: 2
  select_type: DEPENDENT SUBQUERY
        table: nodes
         type: index_subquery
possible_keys: PRIMARY,nodes_tile_idx
          key: PRIMARY
      key_len: 8
          ref: func
         rows: 1
        Extra: Using where
2 rows in set (0.00 sec)
```

Unique subquery is similar, but using a unique or primary key

Subqueries in MySQL

- MySQL versions traditionally had very bad press regarding subquries
  - It was common to recommend rewriting them (when possible) into JOINS
- Since MySQL 5.6, its query execution plans have improved significantly

Lazy Materialization of derived tables

- Option available since MySQL 5.6
  - Improves the execution time of EXPLAIN (it no longer needs to execute subqueries)
  - Derived tables can be indexed automatically at execution time to improve its performance

Derived Table Example

```
mysql-5.5.40 (osm) > EXPLAIN SELECT count(*) FROM (SELECT * FROM nodes WHERE VISIBLE = 1) n JOIN changesets ON
n.changeset_id = changesets.id;
+----+-------------+------------+--------+---------------+---------+---------+----------------+---------+-------------+
| id | select_type | table      | type   | possible_keys | key     | key_len | ref            | rows    | Extra       |
+----+-------------+------------+--------+---------------+---------+---------+----------------+---------+-------------+
|  1 | PRIMARY     | <derived2> | ALL    | NULL          | NULL    | NULL    | NULL           | 2865312 |             |
|  1 | PRIMARY     | changesets | eq_ref | PRIMARY       | PRIMARY | 8       | n.changeset_id |       1 | Using index |
|  2 | DERIVED     | nodes      | ALL    | NULL          | NULL    | NULL    | NULL           | 2865521 | Using where |
+----+-------------+------------+--------+---------------+---------+---------+----------------+---------+-------------+
3 rows in set (1.42 sec)

mysql-8.0 (osm) > EXPLAIN SELECT count(*) FROM (SELECT * FROM nodes WHERE VISIBLE = 1) n JOIN changesets ON
n.changeset_id = changesets.id;
+----+-------------+------------+-------+---------------+-------------+---------+-------------------+---------
+-------------+
| id | select_type | table      | type  | possible_keys | key         | key_len | ref               | rows    | Extra
    |
+----+-------------+------------+-------+---------------+------  -----+---------+-------------------+---------
+-------------+
|  1 | PRIMARY     | changesets | index | PRIMARY       | PRIMARY     8         | NULL              |   70917 | Using
index |
|  1 | PRIMARY     | <derive                                  to_key0>                          .id |      40 | NULL
    |
|  2 | DERIVED     | nodes                                                                          | 2853846 | Using
where |
+----+-------------+---------                                                                    --+---------
+-------------+
3 rows in set (0.00 sec)
```

Subquery is not executed

Auto-generated index

A Common 5.5 Performance Problem

```
mysql-5.5.40 (osm) > EXPLAIN SELECT * FROM nodes
                       WHERE nodes.changeset_id IN (
                         SELECT changesets.id
                          FROM changesets
                          JOIN users
                            ON changesets.user_id = users.id and users.display_name = 'Steve');
+----+--------------------+------------+--------+-------------------------------+----------------------+---------+-------+---------+-------------+
| id | select_type        | table      | type   | possible_keys                 | key                  | key_len | ref   | rows    | Extra       |
+----+--------------------+------------+--------+-------------------------------+----------------------+---------+-------+---------+-------------+
|  1 | PRIMARY            | nodes      | AI L   | NULL                          | NULL                 | NULL    | NULL  | 2865521 | Using where |
|  2 | DEPENDENT SUBQUERY | users      | cons   PRIMARY,users_display_name_idx | users_display_name_idx | 767   | const |       1 | Using index |
|  2 | DEPENDENT SUBQUERY | changesets | eq_ref |                               | PRIMARY              | 8       | func  |       1 | Using where |
+----+--------------------+------------+--------+----                -----------+----------------------+---------+-------+---------+-------------+
3 rows in set (0.00 sec)

mysql-5.5.40 (osm) > SELECT ...;
+-----------+-----------+-----------+--------------+-------
| node_id   | latitude  | longitude | changeset_id | visi
+-----------+-----------+-----------+--------------+-------
|     99890 | 515276425 |  -1497621 |          552 |
|    109174 | 515364532 |  -1457329 |         1875 |
|    276538 | 515324296 |  -2094688 |          810 |        1
|    442987 | 515449207 |  -1275650 |         1941 |        1 | 20                                             1 |
|    442988 | 515449741 |  -1272860 |         1941 |        1 | 2006-01                          |         1 |
|    498803 | 515438432 |  -1269436 |         2171 |        1 | 2006-02-03 21:55:17 | 2062268628 |         1 |
| 138212838 | 513010180 |  -1699929 |      7757299 |        1 | 2011-04-03 18:14:14 | 2062220563 |         6 |
+-----------+-----------+-----------+--------------+---------+---------------------+------------+---------+
7 rows in set (2.60 sec)
```

This means that the subquery is executed almost 3 million times

Semijoin Optimization

- The only way to execute certain IN subqueries was to execute them with poor strategy
  - This forced rewriting of certain queries into JOINS or scalar subqueries, when possible
- There are now several additional options (many automatic):
  - Convert to a JOIN
  - Materialization (including index creation)
  - FirstMatch
  - LooseScan
  - Duplicate Weedout

The Previous Query is Not a Problem in 5.6+/MariaDB 5.3+

```
mysql-8.0 (osm) > EXPLAIN SELECT * FROM nodes
                          WHERE nodes.changeset_id IN (
                              SELECT changesets.id
                                FROM change sets
                                JOIN users
                                  ON changesets.user_id = users.id and users.display_name = 'Steve');
+----+-------------+------------+-------+-----------------------------+----------------------+---------+--------------------+-------+-------------+
| id | select_type | table      | type  | possible_keys               | key                  | key_len | ref                | rows  | Extra       |
+----+-------------+------------+-------+-----------------------------+----------------------+---------+--------------------+-------+-------------+
|  1 | SIMPLE      | users      | const | PRIMARY,users_display_name_idx | users_display_name_idx | 767     | const              |     1 | Using index |
|  1 | SIMPLE      | changesets | ALL   | PRIMARY                     | NULL                 | NULL    | NULL               | 70917 | Using where |
|  1 | SIMPLE      | nodes      | ref   | changeset_id                | changeset_id         | 8       | osm.changesets.id  |    21 | NULL        |
+----+-------------+------------+-------+-----------------------------+----------------------+---------+--------------------+-------+-------------+
3 rows in set (0.00 sec)

mysql-8.0 (osm) > SELECT ...;
+-----------+-----------+-----------+--------------+---------+---------------------+------------+---------+
| node_id   | latitude  | longitude | changeset_id | visible | timestamp           | tile       | version |
+-----------+-----------+-----------+--------------+---------+---------------------+------------+---------+
|     99890 | 515276425 |  -1497621 |          552 |       1 | 2005-10-25 00:35:24 | 2062268512 |       1 |
...
| 138212838 | 513010180 |  -1699929 |      7757299 |       1 | 2011-04-03 18:14:14 | 2062220563 |       6 |
+-----------+-----------+-----------+--------------+---------+---------------------+------------+---------+
7 rows in set (0.02 sec)
```

Executed as a regular JOIN

First Match Strategy

```
mysql-8.0 (osm) > EXPLAIN SELECT * FROM changesets WHERE id IN (SELECT changeset_id FROM nodes)\G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: changesets
   partitions: NULL
         type: ALL
possible_keys: PRIMARY
          key: NULL
      key_len: NULL
          ref: NULL
         rows: 70917
     filtered: 100.00
        Extra: NULL
*************************** 2. row ***************************
           id: 1
  select_type: SIMPLE
        table: nodes
   partitions: NULL
         type: ref
possible_keys: changeset_id
          key: changeset_id
      key_len: 8
          ref: osm.changesets.id
         rows: 33
     filtered: 100.00
        Extra: Using index; FirstMatch(changesets)
2 rows in set, 1 warning (0.00 sec)
```

It is converting the ref into an eq_ref, shot-circuiting the execution

Enabling and disabling materialization, semijoin, etc

```
mysql-8.0 (osm) > SHOW VARIABLES like 'optimizer_switch'\G
*************************** 1. row ***************************
Variable_name: optimizer_switch
        Value: 
index_merge=on,index_merge_union=on,index_merge_sort_union=on,index_merge_in
tersection=on,engine_condition_pushdown=on,index_condition_pushdown=on,mrr=o
n,mrr_cost_based=on,block_nested_loop=on,batched_key_access=off,materializat
ion=on,semijoin=on,loosescan=on,firstmatch=on,duplicateweedout=on,subquery_m
aterialization_cost_based=on,use_index_extensions=on,condition_fanout_filter
=on,derived_merge=on
1 row in set (0.00 sec)
```

Query Optimization with MySQL 8.0 and MariaDB 10.3: The Basics

FULLTEXT, GEODATA SEARCH AND JSON SUPPORT

Fuzzy Search of “hotel/Grand Hotel/Hotel X/etc”

- “Typical” way to solve this:

```
mysql (osm) >
SELECT way_id as id, v
FROM way_tags
WHERE v like '%otel%';
2125 rows in set (3.19 sec)
```

Too slow

Let’s Add an Index

```
mysql (osm) > ALTER TABLE way_tags ADD INDEX(v);
Query OK, 0 rows affected (23.96 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql (osm) > SELECT ...;
2125 rows in set (2.75 sec)
```

Still slow, why?

```
mysql (osm) > EXPLAIN SELECT way_id as id, v FROM way_tags
WHERE v like '%otel%';

+----+.+-------+---------------+------+---------+------+---------+----------+--------------------------+
| id |.| type  | possible_keys | key  | key_len | ref  | rows    | filtered | Extra                    |
+----+.+-------+---------------+------+---------+------+---------+----------+--------------------------+
|  1 |.| index | NULL          | v    | 769     | NULL | 8964000 |    11.11 | Using where; Using index |
+----+.+-------+---------------+------+---------+------+---------+----------+--------------------------+
1 row in set, 1 warning (0.01 sec)
```

Fulltext Index

```
mysql (osm) > ALTER TABLE way_tags add FULLTEXT index(v);
Query OK, 0 rows affected (53.53 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql (osm) > SELECT 'way' as type, way_id as id, v FROM
way_tags WHERE MATCH(v) AGAINST ('+hotel*' IN BOOLEAN MODE);
1851 rows in set (0.04 sec)

mysql (osm) > EXPLAIN …;
```

```
+----+.+----------+---------------+------+---------+-----+------+----------+-----------------------------------+
| id |.| type     | possible_keys | key  | key_len | ref | rows | filtered | Extra                             |
+----+.+----------+---------------+------+---------+-----+------+----------+-----------------------------------+
|  1 |.| fulltext | v             | v    | 0       |     |    1 |   100.00 | Using where; Ft_hints: no_ranking |
+----+.+----------+---------------+------+---------+-----+------+----------+-----------------------------------+
1 row in set, 1 warning (0.00 sec)
```

Some fields are useless for FULTEXT

‘Newer’ Fulltext Optimizations

```
mysql-5.5 (osm) > EXPLAIN SELECT count(*) FROM way_tags_myisam WHERE MATCH(v) AGAINST('hotel');
+----+-------------+-----------------+----------+---------------+------+---------+------+------+-------------+
| id | select_type | table           | type     | possible_keys | key  | key_len | ref  | rows | Extra       |
+----+-------------+-----------------+----------+---------------+------+---------+------+------+-------------+
|  1 | SIMPLE      | way_tags_myisam | fulltext | v             | v    | 0       |      |    1 | Using where |
+----+-------------+-----------------+----------+---------------+------+---------+------+------+-------------+
1 row in set (0.00 sec)

mysql-5.5 (osm) > SHOW STATUS like 'Hand%';
+----------------------------+-------+
| Variable_name              | Value |
+----------------------------+-------+
| Handler_commit             | 0     |
| Handler_delete             | 0     |
...
| Handler_read_first         | 0     |
| Handler_read_key           | 0     |
| Handler_read_last          | 0     |
| Handler_read_next          | 425   |
| Handler_read_prev          | 0     |
| Handler_read_rnd           | 0     |
| Handler_read_rnd_next      | 0     |
...
| Handler_update             | 0     |
| Handler_write              | 0     |
+----------------------------+-------+
16 rows in set (0.00 sec)
```

Newer Fulltext Optimizations (cont.)

```
mysql-8.0 (osm) > EXPLAIN SELECT count(*) FROM way_tags WHERE MATCH(v) AGAINST('hotel');
+----+-------------+-------+------------+------+---------------+------+---------+------+------+----------+------------------------------+
| id | select_type | table | partitions | type | possible_keys | key  | key_len | ref  | rows | filtered | Extra                        |
+----+-------------+-------+------------+------+---------------+------+---------+------+------+----------+------------------------------+
|  1 | SIMPLE      | NULL  | NULL       | NULL | NULL          | NULL | NULL    | NULL | NULL |     NULL | Select tables optimized away |
+----+-------------+-------+------------+------+---------------+------+---------+------+------+----------+------------------------------+
1 row in set, 1 warning (0.00 sec)

mysql-8.0 (osm) > SHOW STATUS like 'Hand%';
+----------------------------+-------+
| Variable_name              | Value |
+----------------------------+-------+
| Handler_commit             | 1     |
| Handler_delete             | 0     |
...
| Handler_read_first         | 0     |
| Handler_read_key           | 0     |
| Handler_read_last          | 0     |
| Handler_read_next          | 0     |
| Handler_read_prev          | 0     |
| Handler_read_rnd           | 0     |
| Handler_read_rnd_next      | 0     |
...
| Handler_update             | 0     |
| Handler_write              | 0     |
+----------------------------+-------+
18 rows in set (0.00 sec)
```

It's counting directly from the FULLTEXT index

Open Issues and Limitations

- No postfix support (wildcards)
- Simple Ranking (and different from MyISAM)
- No stemming support
- Some multi-language limitations

More on FULLTEXT InnoDB support:
http://www.drdobbs.com/database/full-text-search-with-innodb/231902587

Alternatives

- Apache Lucene
  - Solr
  - Elasticsearch
- Sphinx
  - SphinxSE

Find the Closest Starbucks (Spatial Search)

You are here

```
mysql (osm) > SET @lat:=50.11745057; SET @lon:= 8.62699463;
mysql (osm) > SELECT n.node_id,
                     n.longitude/10000000 as longitude,
                     n.latitude/10000000 as latitude,
                     sqrt(pow((latitude/10000000 - @lat) * 111231.29, 2) +
                          pow((longitude/10000000 - @lon) * 71520.91, 2))
                     as `distance in metres`
                FROM nodes n
                JOIN node_tags n_t1
                  ON n.node_id = n_t1.node_id
                JOIN node_tags n_t2
                  ON n.node_id = n_t2.node_id
               WHERE
                     n_t1.k = 'amenity' and
                     n_t1.v = 'cafe' and
                     n_t2.k = 'name' and
                     n_t2.v like 'Starbucks%'
            ORDER BY `distance in metres` ASC
               LIMIT 1;
+------------+-----------+----------+--------------------+
| node_id    | longitude | latitude | distance in metres |
+------------+-----------+----------+--------------------+
| 3136353405 |    8.6523 |  50.1093 | 2023.6465372574328 |
+------------+-----------+----------+--------------------+
1 row in set, 36 warnings (0.81 sec)
```

This Query is Slow

```
mysql [osm]> EXPLAIN SELECT ...\G
**************** 1. row ***************
           Id: 1
  select_type: SIMPLE
        table: n_t1
         type: ALL
possible_keys: PRIMARY
          key: NULL
      key_len: NULL
          ref: NULL
         rows: 2210950
        Extra: Using where;
Using temporary; Using filesort
```

```
*********************** 2. row ***********************
           id: 1
  select_type: SIMPLE
        table: n_t2
         type: ref
possible_keys: PRIMARY
          key: PRIMARY
      key_len: 8
          ref: osm.n_t1.node_id
         rows: 1
        Extra: Using where
*********************** 3. row ***********************
           id: 1
  select_type: SIMPLE
        table: n
         type: ref
possible_keys: PRIMARY
          key: PRIMARY
      key_len: 8
          ref: osm.n_t1.node_id
         rows: 1
        Extra:
3 rows in set (0.00 sec)
```

Can We Optimize it?

- We could add a bounding box:

```
mysql (osm) > ALTER TABLE nodes ADD INDEX (latitude, longitude);

mysql (osm) > SELECT n.node_id,
       n.longitude/10000000 as longitude,
       n.latitude/10000000 as latitude,
       sqrt(pow((latitude/10000000 - @lat) * 111231.29, 2) +
            pow((longitude/10000000 - @lon) * 71520.91, 2))
            as `distance in metres`
       FROM nodes n
       JOIN node_tags n_t1
           ON n.node_id = n_t1.node_id
       JOIN node_tags n_t2
           ON n.node_id = n_t2.node_id
       WHERE
           n_t1.k = 'amenity' and
           n_t1.v = 'cafe' and
           n_t2.k = 'name' and
           n_t2.v like 'Starbucks%' and
           n.latitude BETWEEN ((@lat - 1000/111231.29) * 10000000)
                          AND ((@lat + 1000/111231.29) * 10000000) and
           n.longitude BETWEEN ((@lon - 1000/71520.91) * 10000000)
                          AND ((@lon + 1000/71520.91) * 10000000)
       ORDER BY `distance in metres` ASC
       LIMIT 1;
```

This is not a square, only an approximation

We Create an Index... and Force It

```
mysql-8.0 (osm) > EXPLAIN SELECT ...;
+----+-------------+-------+------------+------+-----                             ----------+----------+-----------------+
| id | select_type | table | partitions | type | pos                               ws       | filtered | Extra            |
|    |             |       |            |      | _ke                                        |          |                  |
+----+-------------+-------+------------+------+---                                ---------+----------+------------------+
|  1 | SIMPLE      | n_t1  | NULL       | ALL  | PRIMAR                          | 2210950 |     0.00 | Using where;     |
|    |             |       |            |      |                                 |         |          | Using temporary; |
|    |             |       |            |      |          |         |      |       |         |          | Using filesort   |
|  1 | SIMPLE      | n     | NULL       | ref  | PRIMARY, | PRIMARY | 8    | osm.n_t1.node_id |       1 |     5.00 | Using where      |
|    |             |       |            |      | latitude |         |      |                  |         |          |                  |
|  1 | SIMPLE      | n_t2  | NULL       | ref  | PRIMARY  | PRIMARY | 8    | osm.n_t1.node_id |       1 |     1.41 | Using where      |
+----+-------------+-------+------------+------+----------+---------+------+------------------+---------+----------+------------------+
3 rows in set, 1 warning (0.00 sec)

mysql-8.0 (osm) > EXPLAIN SELECT ... FROM nodes n FORCE INDEX(latitude) ...;
+----+-------------+-------+------------+-------+----------+----------+------+------------------+--------+----------+-----------------+
| id | select_type | table | partitions | type  | possible | key      | key_ | ref              | rows   | filtered | Extra           |
|    |             |       |            |       | _keys    |          | len  |                  |        |          |                 |
+----+-------------+-------+------------+-------+----------+----------+------+-----------------  +--------+----------+-----------------+
|  1 | SIMPLE      | n     | NULL       | range | latitude | latitude | 8    | NULL             | 949370 |    11.11 | Usin  here;     |
|    |             |       |            |       |          |          |      |                  |        |          | Usin   dex;     |
|    |             |       |            |       |          |          |      |                  |        |          | Usir   sort     |
|  1 | SIMPLE      | n_t1  | NULL       |                    | PRIMARY  | 8    | osm.n.n            |      1 |     1.41 |
|  1 | SIMPLE      | n_t2  | NULL       |                      IMARY  | 8    |                    |      1 |     1.
+----+-------------+-------+-----------                         -----+------+-                    -------+----
3 rows in set, 1 warning (0.00

Mysql-8.0 (osm) > SELECT ..                          .;
0 rows in set (0.26 se
```

MySQL ignores the newly created index, why?

Performance improvement is not great

Still many rows are examined

Most of the gain comes from the covering index, not the filtering

Multiple Range Scans Cannot Be Optimized with BTREE Indexes

- We need quadtrees or R-TREE Indexes for indexing in multiple dimensions
  - The latter are implemented in MySQL with the name “SPATIAL indexes”, as they only apply to GIS types
- Spatial indexing is available on MySQL since 5.7.5

Creating a Spatial Index

```
mysql (osm) > ALTER TABLE nodes
              ADD COLUMN coord GEOMETRY NOT NULL SRID 4326;
Query OK, 0 rows affected (0.01 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql (osm) > UPDATE nodes
              SET coord = point(longitude/10000000,
                                latitude/10000000);
Query OK, 20216490 rows affected (7 min 10.28 sec)
Rows matched: 20216490  Changed: 20216490  Warnings: 0

mysql (osm) > ALTER TABLE nodes add SPATIAL index(coord);
Query OK, 0 rows affected (1 hour 7 min 13.2[illegible] sec)
Records: 0  Duplicates: 0  Warnings: 0
```

Instant, yay!
(or create it
VIRTUAL)

This is new since
5.7

New Query

```
mysql> SET @area := ST_Envelope(linestring(POINT(@lon - 500/71520.91, @lat - 500/111231.29), POINT(@lon +
500/71520.91, @lat + 500/111231.29)));

mysql> SELECT n.node_id,
              x(n.coord) as longitude,
              y(n.coord) as latitude,
              st_distance(POINT(@lon, @lat), coord) as distance
       FROM nodes n
       JOIN node_tags n_t1
           ON n.node_id = n_t1.node_id
       JOIN node_tags n_t2
           ON n.node_id = n_t2.node_id
       WHERE
           n_t1.k = 'amenity' and
           n_t1.v = 'cafe' and
           n_t2.k = 'name' and
           n_t2.v like 'Starbucks%' and
           st_within(coord, @area)
       ORDER BY st_distance(POINT(@lon, @lat), coord) ASC
       LIMIT 1;
```

We can use any shape we want thanks to 5.6 improvements

Also substitute functions like Envelope with ST_Envelope since 5.7

Better Performance

```
mysql (osm) > SELECT ...;
+-----------+------------+------------+-----------------------+
| node_id   | longitude  | latitude   | distance              |
+-----------+------------+------------+-----------------------+
| XXXXXXXXX | XXXXXXXXXX | XXXXXXXXXX | 0.0014631428672541478 |
+-----------+------------+------------+-----------------------+
Empty set (0.11 sec)
```

This field used to be almost useless (wait for it)

```
mysql (osm) > EXPLAIN SELECT ...;
+----+---------+-------+-------+-------+----------+---------+------+---------+------+----------+-----------------+
| id | select  | table | parti | type  | possible | key     | key  | ref     | rows | filtered | Extra           |
|    | _type   |       | tions |       | _keys    |         | _len |         |      |          |                 |
+----+---------+-------+-------+-------+----------+---------+------+---------+------+----------+-----------------+
|  1 | SIMPLE  | n     | NULL  | range | PRIMARY  | coord   | 34   | NULL    |    2 |   100.00 | Using where;    |
|    |         |       |       |       | ,coord   |         |      |         |      |          | Using filesort  |
|  1 | SIMPLE  | n_t1  | NULL  | ref   | PRIMARY  | PRIMARY | 8    | osm.n.  |    3 |     1.41 | Using where     |
|    |         |       |       |       |          |         |      | node_id |      |          |                 |
|  1 | SIMPLE  | n_t2  | NULL  | ref   | PRIMARY  | PRIMARY | 8    | osm.n.  |    3 |     1.41 | Using where     |
|    |         |       |       |       |          |         |      | node_id |      |          |                 |
+----+---------+-------+-------+-------+----------+---------+------+---------+------+----------------------------+
3 rows in set (0.00 sec)
```

Better Filtering

```
mysql-8.0 (osm) > SHOW STATUS LIKE 'Hand%';
```

Not using the index:

```
+----------------------------+--------+
| Variable_name              | Value  |
+----------------------------+--------+
| Handler_commit             | 1      |
| Handler_delete             | 0      |
| Handler_discover           | 0      |
| Handler_external_lock      | 6      |
| Handler_mrr_init           | 0      |
| Handler_prepare            | 0      |
| Handler_read_first         | 1      |
| Handler_read_key           | 1914   |
| Handler_read_last          | 0      |
| Handler_read_next          | 1954   |
| Handler_read_prev          | 0      |
| Handler_read_rnd           | 1      |
| Handler_read_rnd_next      | 833426 |
| Handler_rollback           | 0      |
| Handler_savepoint          | 0      |
| Handler_savepoint_rollback | 0      |
| Handler_update             | 0      |
| Handler_write              | 1      |
+----------------------------+--------+
18 rows in set (0.00 sec)
```

Using the BTREE index:

```
+----------------------------+--------+
| Variable_name              | Value  |
+----------------------------+--------+
| Handler_commit             | 1      |
| Handler_delete             | 0      |
| Handler_discover           | 0      |
| Handler_external_lock      | 6      |
| Handler_mrr_init           | 0      |
| Handler_prepare            | 0      |
| Handler_read_first         | 0      |
| Handler_read_key           | 274    |
| Handler_read_last          | 0      |
| Handler_read_next          | 246540 |
| Handler_read_prev          | 0      |
| Handler_read_rnd           | 0      |
| Handler_read_rnd_next      | 0      |
| Handler_rollback           | 0      |
| Handler_savepoint          | 0      |
| Handler_savepoint_rollback | 0      |
| Handler_update             | 0      |
| Handler_write              | 0      |
+----------------------------+--------+
18 rows in set (0.00 sec)
```

Using the SPATIAL index:

```
+----------------------------+-------+
| Variable_name              | Value |
+----------------------------+-------+
| Handler_commit             | 1     |
| Handler_delete             | 0     |
| Handler_discover           | 0     |
| Handler_external_lock      | 6     |
| Handler_mrr_init           | 0     |
| Handler_prepare            | 0     |
| Handler_read_first         | 0     |
| Handler_read_key           | 522   |
| Handler_read_last          | 0     |
| Handler_read_next          | 5254  |
| Handler_read_prev          | 0     |
| Handler_read_rnd           | 259   |
| Handler_read_rnd_next      | 0     |
| Handler_rollback           | 0     |
| Handler_savepoint          | 0     |
| Handler_savepoint_rollback | 0     |
| Handler_update             | 0     |
| Handler_write              | 0     |
+----------------------------+-------+
18 rows in set (0.00 sec)
```

Geohash Functions

```
mysql (osm) > SELECT ST_GeoHash(@lon, @lat, 10);
+----------------------------+
| ST_GeoHash(@lon, @lat, 10) |
+----------------------------+
| u0yjh79gr9                 |
+----------------------------+
1 row in set (0.00 sec)
```

- Useful to index coordinates with a BTREE
  - It could be specially useful combined with indexed STORED columns (emulating quadtrees)

More on Geohashing: http://mysqlserverteam.com/geohash-functions/

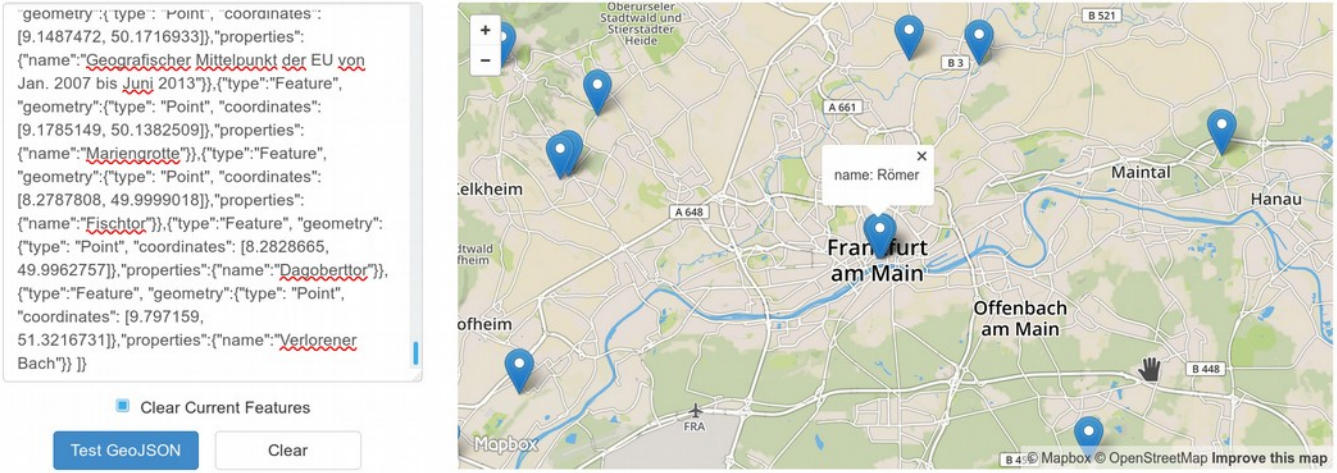
GeoJSON Functions

```
mysql (osm) > SELECT nm.v, ST_AsGeoJson(n.coord)
                FROM node_tags n_t
                JOIN nodes n USING (node_id, version)
                JOIN node_tags nm USING (node_id, version)
               WHERE n_t.k='tourism' AND
                     n_t.v='attraction' AND
                     nm.k='name';
+-----------------------+-----------------------------------------------------+
| v                     | ST_AsGeoJson(n.coord)                               |
+-----------------------+-----------------------------------------------------+
| Schießbuckel          | {"type": "Point", "coordinates": [8.4938556, 49.6351754]} |
| AKW Informationszentrum | {"type": "Point", "coordinates": [8.4175585, 49.7060227]} |
| Burg Gleiberg         | {"type": "Point", "coordinates": [8.6344314, 50.6150329]} |
| Brüderkirche          | {"type": "Point", "coordinates": [9.5042975, 51.3149351]} |
| Römer                 | {"type": "Point", "coordinates": [8.6816587, 50.1104684]} |
.
.
```

GeoJSON Functions (cont.)

```
$ mysql osm -B -e "SET NAMES utf8; SET SESSION group_concat_max_len = 10000; SELECT
CONCAT('{\"type\":\"FeatureCollection\", \"features\":[ ',
GROUP_CONCAT(CONCAT('{\"type\":\"Feature\", \"geometry\":', ST_AsGeoJson(n.coord),
',\"properties\":{\"name\":\"',nm.v,'\"}}')), ' ]}') FROM node_tags n_t JOIN nodes n USING
(node_id, version) JOIN node_tags nm USING (node_id, version) WHERE n_t.k='tourism' and
n_t.v='attraction' AND nm.k='name'"
```

http://geojsonlint.com/



```
"geometry":{"type": "Point", "coordinates":
[9.1487472, 50.1716933]},"properties":
{"name":"Geografischer Mittelpunkt der EU von
Jan. 2007 bis Juni 2013"}},{"type":"Feature",
"geometry":{"type": "Point", "coordinates":
[9.1785149, 50.1382509]},"properties":
{"name":"Mariengrotte"}},{"type":"Feature",
"geometry":{"type": "Point", "coordinates":
[8.2787808, 49.9999018]},"properties":
{"name":"Fischtor"}},{"type":"Feature", "geometry":
{"type": "Point", "coordinates": [8.2828665,
49.9962757]},"properties":{"name":"Dagoberttor"}},
{"type":"Feature", "geometry":{"type": "Point",
"coordinates": [9.797159,
51.3216731]},"properties":{"name":"Verlorener
Bach"}} ]}
```

Older Issues

- Before MySQL 8.0, SRID could be set and retrieved, but all operations were done in squared euclidean coordinates:

```
mysql-5.7.5 (osm) > SET @p1 := GeomFromText('POINT(8 50)', 4326);
Query OK, 0 rows affected (0.00 sec)

mysql-5.7.5 (osm) > SET @p2 := GeomFromText('POINT(7 50)', 4326);
Query OK, 0 rows affected (0.00 sec)

mysql-5.7.5 (osm) > SET @p3 := GeomFromText('POINT(8 51)', 4326);
Query OK, 0 rows affected (0.00 sec)

mysql-5.7.5 (osm) > SELECT srid(@p1);
+-----------+
| srid(@p1) |
+-----------+
|      4326 |
+-----------+
1 row in set (0.00 sec)
```

```
mysql-5.7.5 (osm) > SELECT st_distance(@p1, @p2);
+-----------------------+
| st_distance(@p1, @p2) |
+-----------------------+
|                     1 |
+-----------------------+
1 row in set (0.00 sec)

mysql-5.7.5 (osm) > SELECT st_distance(@p1, @p3);
+-----------------------+
| st_distance(@p1, @p3) |
+-----------------------+
|                     1 |
+-----------------------+
1 row in set (0.00 sec)
```

New 8.0 SRID Support !

```
mysql-8.0 (osm) > SET @p1 := ST_GeomFromText('POINT(8 50)',
4326);
Query OK, 0 rows affected (0.00 sec)

mysql-8.0 (osm) > SET @p2 := ST_GeomFromText('POINT(7 50)',
4326);
Query OK, 0 rows affected (0.00 sec)

mysql-8.0 (osm) > SET @p3 := ST_GeomFromText('POINT(8 51)',
4326);
Query OK, 0 rows affected (0.00 sec)

mysql-8.0 (osm) > SELECT ST_srid(@p1);
+--------------+
| ST_srid(@p1) |
+--------------+
|         4326 |
+--------------+
1 row in set (0.00 sec)
```

```
mysql-8.0 (osm) > SELECT st_distance(@p1, @p2);
+-----------------------+
| st_distance(@p1, @p2) |
+-----------------------+
|     110243.2689139409 |
+-----------------------
1 row in set (0.00 sec)

mysql-8.0 (osm) > SELECT
+-----------------------
| st_distance(@p1, @p3)
+-----------------------+
|    110592.12831394037 |
+-----------------------+
1 row in set (0.01 sec)
```

Answers in meters- The world is no longer flat!

I expected something around 71-72 Km, did I do something wrong?

JSON Native Data Type

- Since 5.7.8, MySQL allows columns defined with the JSON data type:

```
mysql-8.0 > CREATE TABLE json_test(id int PRIMARY KEY au
content JSON);
Query OK, 0 rows affected (0.03 sec)

mysql-8.0 > INSERT INTO json_test (content) VALUES ('{"type":
"correct_json"}');
Query OK, 1 row affected (0.00 sec)

mysql-8.0 > INSERT INTO json_test (content) VALUES ('{"type":
"incorrect_json}');
ERROR 3140 (22032): Invalid JSON text: "Missing a closing quotation
mark in string." at position 24 in value (or column) '{"type":
"incorrect_json}'.
```

MariaDB supports it only as a BLOB/LONGTEXT alias

They get validated on insert

JSON functions

- MySQL includes almost all functions to manipulate JSON that you may think of:
  - Validation test: `JSON_TYPE`
  - Object creation: `JSON_ARRAY, JSON_MERGE, ...`
  - Searching: `JSON_EXTRACT`
  - Modifying: `JSON_SET, JSON_INSERT, ...`

Indexing JSON

- JSON Columns cannot be indexed:

```
mysql [localhost] {msandbox} (test) > ALTER TABLE
json_test ADD INDEX(content);
ERROR 3152 (42000): JSON column 'content' cannot be
used in key specification.
```

- However, they can be compared with regular fields and use indexes thanks to virtual columns

Query Optimization with MySQL 8.0 and MariaDB 10.3: The Basics

CONCLUSIONS

8.0/10.3 Recently released

- They are mature enough to be used in production
- Latest release doesn’t have as many new features related to query optimization
  - But they have a lot of other features and quality of life improvements
  - They also fix long outstanding query performance bugs
  - MariaDB also catches up with MySQL in some areas

MySQL 8.0 New Features

- MySQL 5.6 seemed Percona Server-inspired
- MySQL 5.7 seemed MariaDB-inspired
- MySQL 8.0 seems Galera and Oracle DB-inspired
  - Competition is always good for consumer

PERCONA SERVER
MySQL™
MariaDB

Modern SQL is now part of MySQL!

- Window functions, recursive CTEs:

```
WITH RECURSIVE cte (cl_from, cl_type) AS
(
SELECT cl_from, cl_type FROM categorylinks WHERE cl_to = 'Database_management_systems'
-- starting category
UNION
SELECT categorylinks.cl_from, categorylinks.cl_type FROM cte JOIN page ON cl_from =
page_id JOIN categorylinks ON page_title = cl_to WHERE cte.cl_type = 'subcat' --
subcat addition on each iteration
)
SELECT page_title FROM cte JOIN page ON cl_from = page_id WHERE page_namespace = 0
ORDER BY page_title; -- printing only articles in the end, ordered by title
```

Scan Wikipedia categories, page links recursively!

Many Optimizer Advantages Have to Be Manually Enabled

- Modifying on a per-query basis:

  ```
  SET optimizer_switch='batched_key_access=on';
  SET join_cache_level=8; # for MariaDB
  ```

  in order to take advantage of them, making some of features useless unless you are fine-tuning

  - I expect that to change in the future

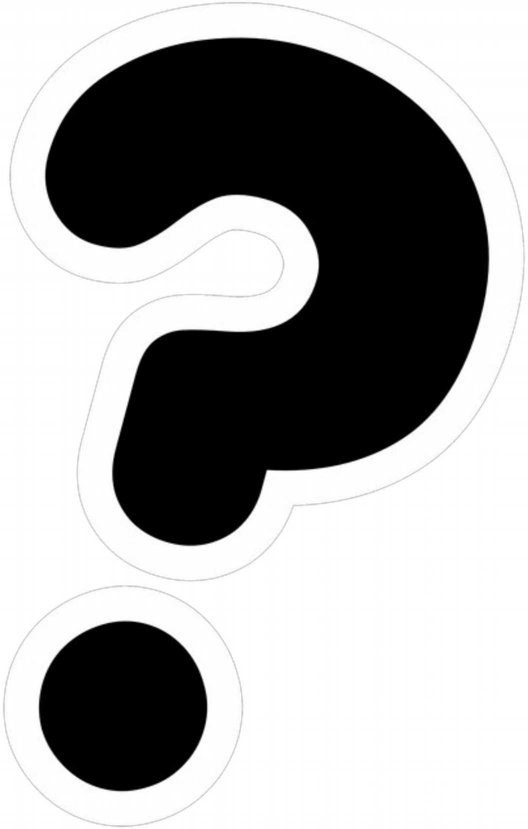
I Herby Declare MyISAM as Dead

- All major MyISAM-only features are now on MySQL 5.7/8.0
  - FULLTEXT
  - GIS
  - Transportable tables
- Previous blockers such as grant tables (new InnoDB data dictionary) and temporary tables are now in InnoDB format

Benchmarks

- Do not trust first party benchmarks
  - In fact, do not trust 3rd party benchmarks either
- Only care about the performance of your application running on your hardware

Q&A

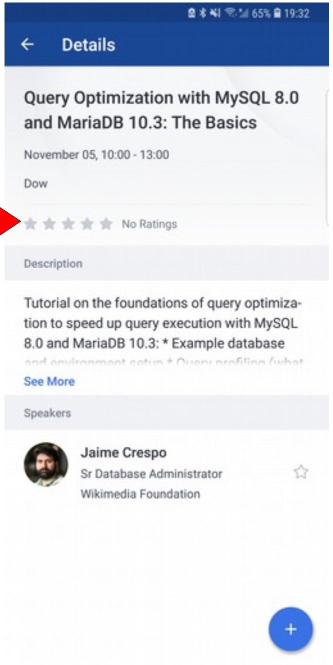


Not to Miss

- Operations track:
  TLS for MySQL at Large Scale:
  How we do relational data on-the-wire encryption at the Wikimedia Foundation
- Do you want to do query optimization for a website with 20 Billion views per month?
  https://wikimediafoundation.org/about/jobs/

Thank You for Attending!

- Do not forget, after the session finishes, to please login with your Percona Live app and “Rate This Session”
- Special thanks to in order by rand() to: Ariel Glenn, Manuel Arostegui, Morgan Tocker, Sean Pringle, David Hildebrandt, Bill Karwin, Domas Mituzas, Mark Callaghan, Shlomi Noach, Valerii Kravchuk, Miguel Ángel Nieto, Dimitri Kravtchuk, Olav Sandstå and the whole Wikimedia Team, and all people at the MariaDB, Percona and MySQL/Oracle teams, and the Percona Live Organization and Sponsors



Query Optimization with MySQL 8.0 and MariaDB 10.3: The Basics

November 05, 10:00 - 13:00

Dow

☆☆☆☆☆ No Ratings

Description

Tutorial on the foundations of query optimization to speed up query execution with MySQL 8.0 and MariaDB 10.3: * Example database and environment setup * Query profiling (what

See More

Speakers

Jaime Crespo
Sr Database Administrator
Wikimedia Foundation