Improving MySQL/MariaDB query performance through optimizer tuning

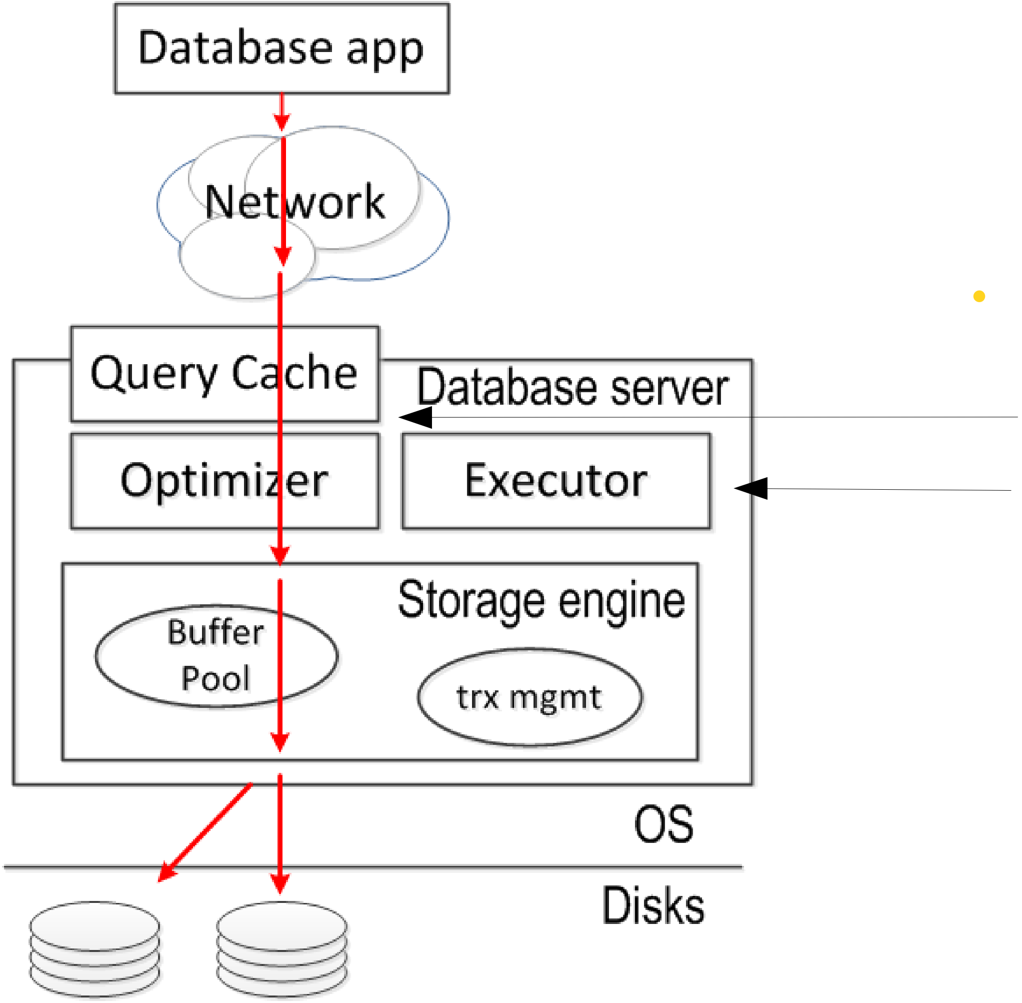
Sergey Petrunya
Timour Katchaounov
{psergey,timour}@montyprogram.com

What is an optimizer

and why do I need to tune it

Why does one need optimizer tuning

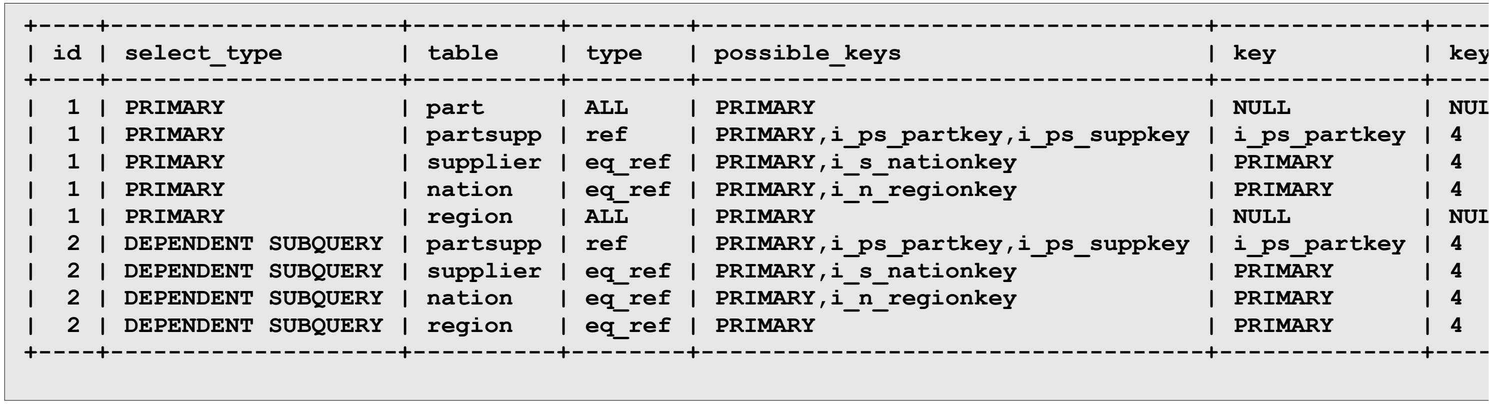
- Database performance is affected by many factors



- One of them is the **query optimizer** (and query executor, together **query processor**)

What is a query optimizer ?

- Converts SQL into execution instructions
- Like GPS gives driving directions from A to B



```
+----+--------------------+----------+--------+-----------------------------------+--------------+---
| id | select_type        | table    | type   | possible_keys                     | key          | key
+----+--------------------+----------+--------+-----------------------------------+--------------+---
|  1 | PRIMARY            | part     | ALL    | PRIMARY                           | NULL         | NUL
|  1 | PRIMARY            | partsupp | ref    | PRIMARY,i_ps_partkey,i_ps_suppkey | i_ps_partkey | 4
|  1 | PRIMARY            | supplier | eq_ref | PRIMARY,i_s_nationkey             | PRIMARY      | 4
|  1 | PRIMARY            | nation   | eq_ref | PRIMARY,i_n_regionkey             | PRIMARY      | 4
|  1 | PRIMARY            | region   | ALL    | PRIMARY                           | NULL         | NUL
|  2 | DEPENDENT SUBQUERY | partsupp | ref    | PRIMARY,i_ps_partkey,i_ps_suppkey | i_ps_partkey | 4
|  2 | DEPENDENT SUBQUERY | supplier | eq_ref | PRIMARY,i_s_nationkey             | PRIMARY      | 4
|  2 | DEPENDENT SUBQUERY | nation   | eq_ref | PRIMARY,i_n_regionkey             | PRIMARY      | 4
|  2 | DEPENDENT SUBQUERY | region   | eq_ref | PRIMARY                           | PRIMARY      | 4
+----+--------------------+----------+--------+-----------------------------------+--------------+---
```

Can the optimizer work poorly?

Yes, just like your GPS

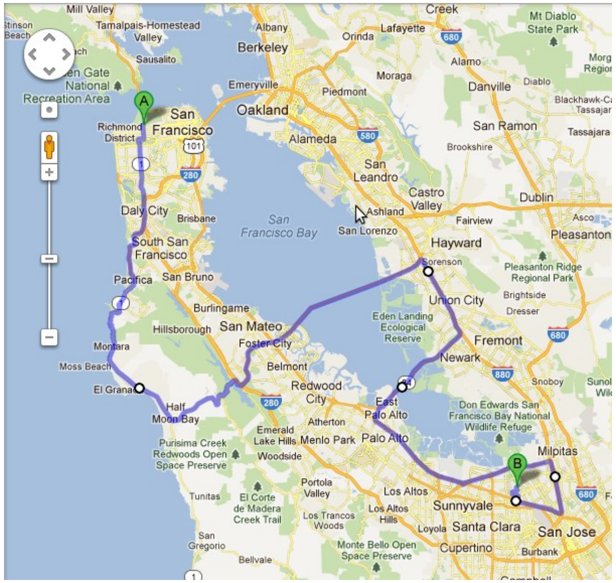
It can produce a sub-optimal query plan (compare to GPS's directions)

Unneeded CPU/IO usage in the database is the same extra mileage

How can one tell if they have a problem in query optimizer?

- Like with the GPS:
  - Knowledge of possible routes
  - Common sense

This is what this tutorial is about



Start from a database performance problem

- Database response time is too high
  - or the server denies connections altogether
    - because running queries have occupied all @@max_connections threads.
- Monitoring (cacti, etc) shows high CPU/ disk utilization
- Is this an optimizer problem?
  - Maybe.

Signs that it's caused by the optimizer

- Only certain kinds of queries are affected
  - Unlike concurrency/disk/binlog problems that cause slowdown for all queries
- These queries can be caught with
  - Slow query log
    - pt-query-digest
  - SHOW PROCESSLIST
  - SHOW PROFILE

Optimizer problems in Slow query log

- Percona server/MariaDB:
  --log_slow_verbosity=query_plan

```
# Thread_id: 1  Schema: dbt3sf10  QC_hit: No
# Query_time: 2.452373  Lock_time: 0.000113  Rows_sent: 0  Rows_examined: 1500000
# Full_scan: Yes  Full_join: No  Tmp_table: No  Tmp_table_on_disk: No
# Filesort: No  Filesort_on_disk: No  Merge_passes: 0
SET timestamp=1333385770;
select * from customer where c_acctbal < -1000;
```

- Look for big Query_time (>> Lock_time)
- Look for Rows_examined >> Rows_sent
  - Although there are “legitimate” cases for this
- Query_plan members (violet), if present, also show *possible* problem areas

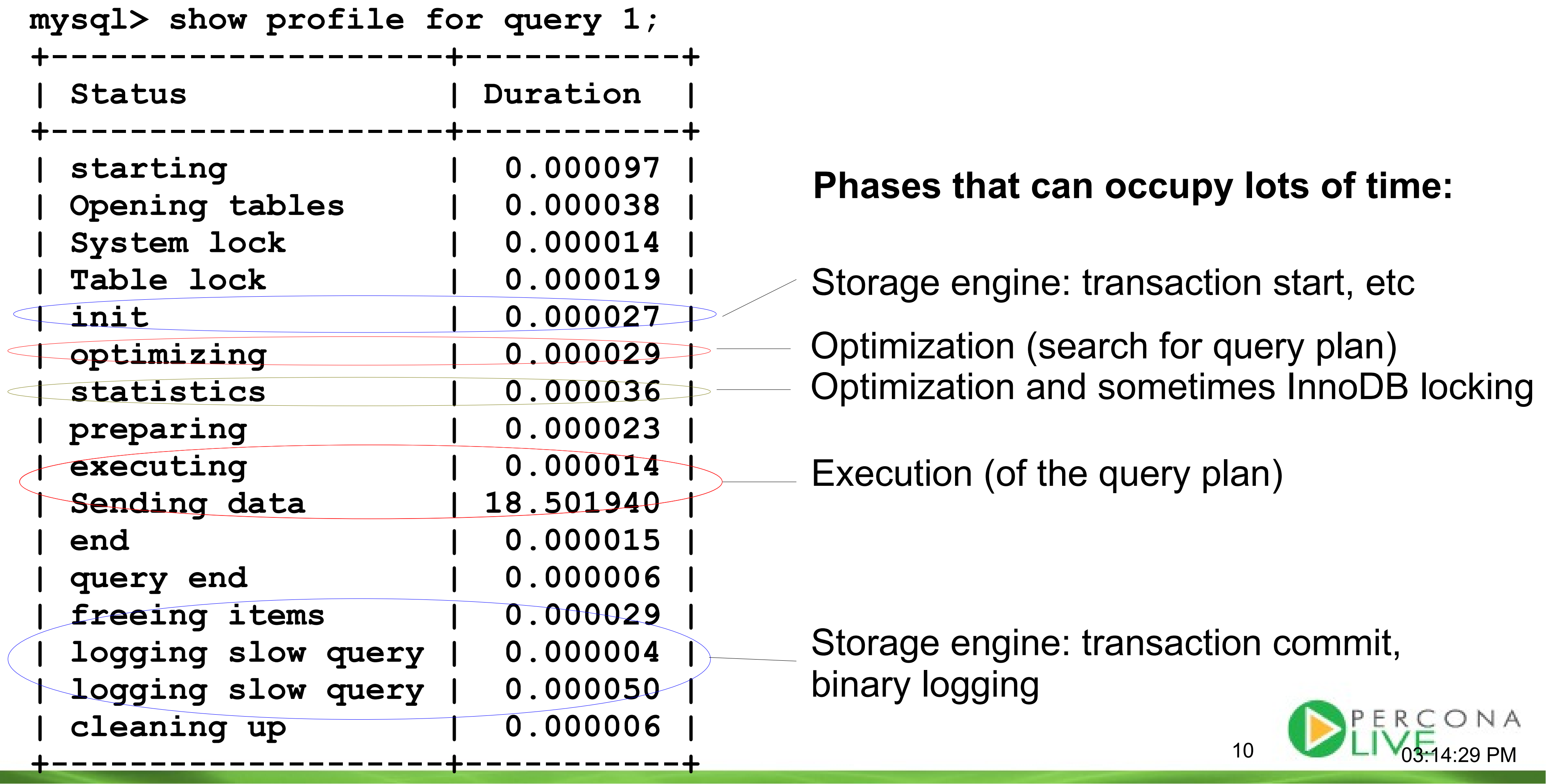
SHOW PROCESSLIST;

```
MariaDB [dbt3sf10]> show processlist;
+----+------+-----------+----------+---------+------+---------------+--------
| Id | User | Host      | db       | Command | Time | State         | Info
+----+------+-----------+----------+---------+------+---------------+--------
|  1 | root | localhost | dbt3sf10 | Query   |  176 | Sending data  | select
|  2 | root | localhost | dbt3sf10 | Query   |    0 | NULL          | show pr
|  3 | root | localhost | dbt3sf10 | Query   |   19 | freeing items | insert
|  4 | root | localhost | dbt3sf10 | Sleep   |   77 |               | NULL
|  5 | root | localhost | dbt3sf10 | Query   |    4 | Updating      | update
|  6 | root | localhost | dbt3sf10 | Query   |    5 | statistics    | select
```

- States to look for:
  - optimizing
  - executing
  - Statistics
  - Sending data

SHOW PROFILE

- Uses the same set of states
- Shows where exactly the query has spent time



```
mysql> show profile for query 1;
+--------------------+------------+
| Status             | Duration   |
+--------------------+------------+
| starting           |   0.000097 |
| Opening tables     |   0.000038 |
| System lock        |   0.000014 |
| Table lock         |   0.000019 |
| init               |   0.000027 |
| optimizing         |   0.000029 |
| statistics         |   0.000036 |
| preparing          |   0.000023 |
| executing          |   0.000014 |
| Sending data       |  18.501940 |
| end                |   0.000015 |
| query end          |   0.000006 |
| freeing items      |   0.000029 |
| logging slow query |   0.000004 |
| logging slow query |   0.000050 |
| cleaning up        |   0.000006 |
+--------------------+------------+
```

Phases that can occupy lots of time:

Storage engine: transaction start, etc

Optimization (search for query plan)

Optimization and sometimes InnoDB locking

Execution (of the query plan)

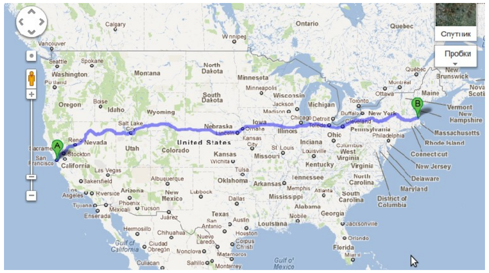
Storage engine: transaction commit, binary logging

Global observations summary

- A query [pattern] that
  - Spends a lot of time in optimization/execution phase
  - Examines many more records than you think it should
  - Uses Full_scan, Full_join, etc
  - .. or all of the above
- Is a candidate for further investigation

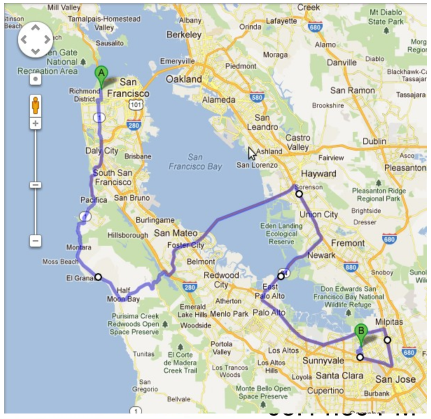
Reasons why a query is “heavy”

- Some queries are inherently hard to compute



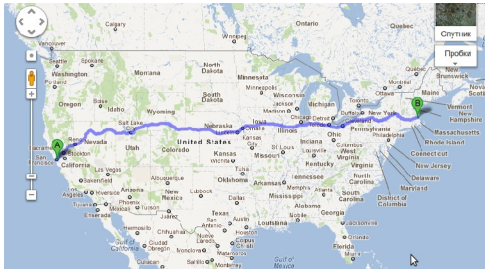
- Some get a plan which looks good but turns out bad due to circumstances that were not accounted for

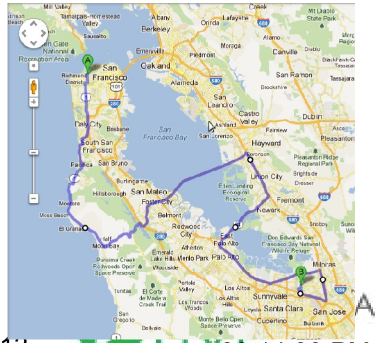
- And some are just bad query plan choices



Know thy optimizer

- How to tell which case you're dealing with?
  - Need to know “the distance”
    - The lower bound of amount of effort one needs to expend to arrive at the destination
    - Easy in geography, not so easy in SQL
  - Need to know “the roads”
    - Available table access strategies
    - How they are composed into query plans





Optimizer troubleshooting

From simple to complex queries

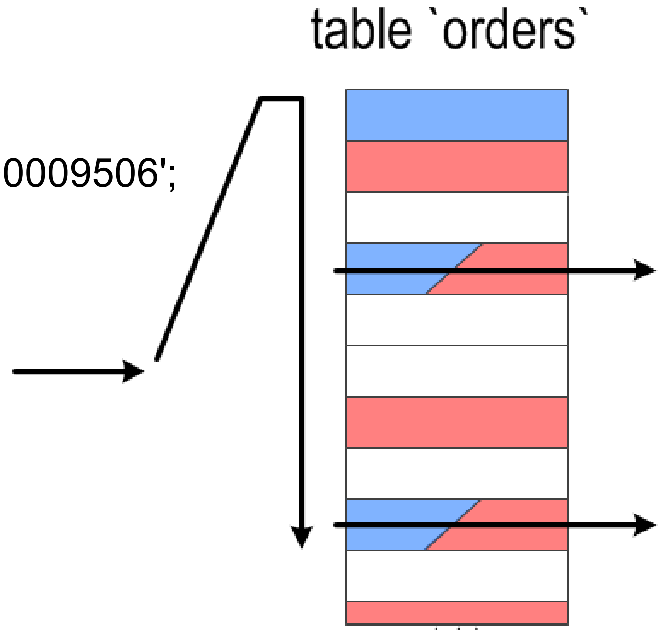
Let's take a simple, slow query

select * from orders
where o_orderDate='1992-06-06' and o_clerk='Clerk#000009506';

```
+----+-------------+--------+------+---------------+------+---------+------+----------+-------------+
| id | select_type | table  | type | possible_keys | key  | key_len | ref  | rows     | Extra       |
+----+-------------+--------+------+---------------+------+---------+------+----------+-------------+
|  1 | SIMPLE      | orders | ALL  | NULL          | NULL | NULL    | NULL | 15084733 | Using where |
+----+-------------+--------+------+---------------+------+---------+------+----------+-------------+
```

```
# Thread_id: 2  Schema: dbt3sf10  QC_hit: No
# Query_time: 41.129178  Lock_time: 0.000174  Rows_sent: 2  Rows_examined: 15000000
# Full_scan: Yes  Full_join: No  Tmp_table: No  Tmp_table_on_disk: No
# Filesort: No  Filesort_on_disk: No  Merge_passes: 0
use dbt3sf10;
SET timestamp=1333432984;
select * from orders where o_orderDate='1992-06-06' and o_clerk='Clerk#000009506';
```

- Full scan examined lots of rows
- But only a few (two) of them “have contributed to the result set”
  - It was unnecessary to read the rest
- Query plan is poor, it seems.



Status variables

```
MariaDB [dbt3sf10]> flush status;
Query OK, 0 rows affected (0.00 sec)

MariaDB [dbt3sf10]> select * from orders where o_orderDate='1992-06-06' and o_clerk='Clerk#000009506';
...

MariaDB [dbt3sf10]> show status like 'Handler%';
+----------------------------+----------+
| Variable_name              | Value    |
+----------------------------+----------+
| Handler_commit             | 1        |
| Handler_delete             | 0        |
| Handler_discover           | 0        |
| Handler_icp_attempts       | 0        |
| Handler_icp_match          | 0        |
| Handler_mrr_init           | 0        |
| Handler_mrr_key_refills    | 0        |
| Handler_mrr_rowid_refills  | 0        |
| Handler_prepare            | 0        |
| Handler_read_first         | 0        |
| Handler_read_key           | 0        |
| Handler_read_next          | 0        |
| Handler_read_prev          | 0        |
| Handler_read_rnd           | 0        |
| Handler_read_rnd_deleted   | 0        |
| Handler_read_rnd_next      | 15000001 |
| Handler_rollback           | 0        |
| Handler_savepoint          | 0        |
| Handler_savepoint_rollback | 0        |
| Handler_tmp_update         | 0        |
| Handler_tmp_write          | 0        |
| Handler_update             | 0        |
| Handler_write              | 0        |
+----------------------------+----------+
23 rows in set (0.01 sec)
```

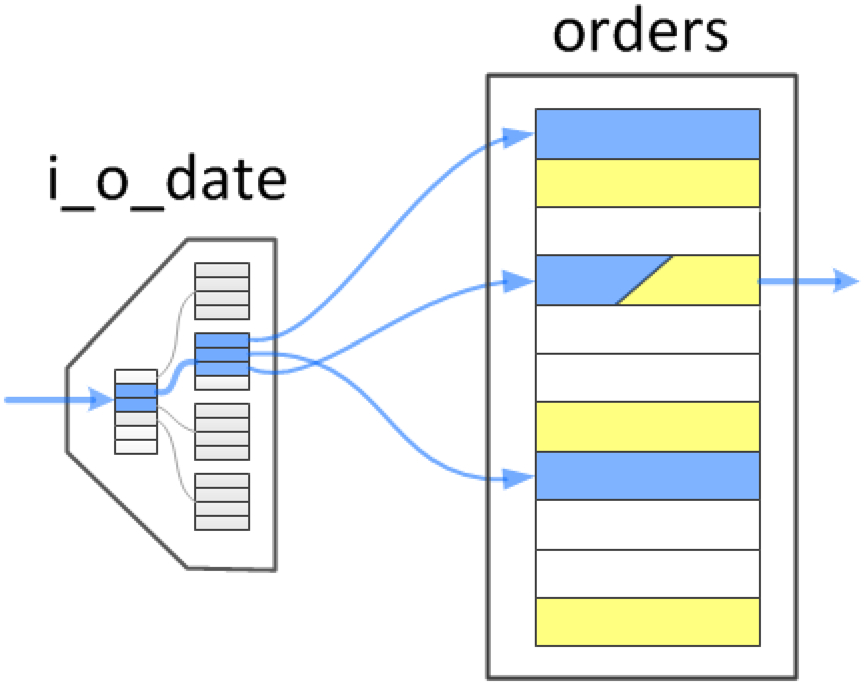
Indeed, 15M sequential reads

With index, we'll read fewer rows

alter table orders add key i_o_orderdate (o_orderdate);
select * from orders
where o_orderDate='1992-06-06' and o_clerk='Clerk#000009506';

```
+----+-------------+--------+------+---------------+---------------+---------+-------+------+-------------+
| id | select_type | table  | type | possible_keys | key           | key_len | ref   | rows | Extra       |
+----+-------------+--------+------+---------------+---------------+---------+-------+------+-------------+
|  1 | SIMPLE      | orders | ref  | i_o_orderdate | i_o_orderdate | 4       | const | 6303 | Using where |
+----+-------------+--------+------+---------------+---------------+---------+-------+------+-------------+
```

- The index allows to only read records with i_o_orderdate='Clerk..'
  - (At the cost of having to read index entries also)
- Still not perfect: we're reading records that have the wrong o_orderDate
- Let's check how it executes:



Status variables

```
MariaDB [dbt3sf10]> flush status;

MariaDB [dbt3sf10]> select * from orders where o_orderDate='1992-06-06' and o_clerk='Clerk#000009506';
... (0.39 sec)

MariaDB [dbt3sf10]> show status like 'Handler%';
+----------------------------+-------+
| Variable_name              | Value |
+----------------------------+-------+
| Handler_commit             | 1     |
| Handler_delete             | 0     |
| Handler_discover           | 0     |
| Handler_icp_attempts       | 0     |
| Handler_icp_match          | 0     |
| Handler_mrr_init           | 0     |
| Handler_mrr_key_refills    | 0     |
| Handler_mrr_rowid_refills  | 0     |
| Handler_prepare            | 0     |
| Handler_read_first         | 0     |
| Handler_read_key           | 1     |
| Handler_read_next          | 6122  |
| Handler_read_prev          | 0     |
| Handler_read_rnd           | 0     |
| Handler_read_rnd_deleted   | 0     |
| Handler_read_rnd_next      | 0     |
| Handler_rollback           | 0     |
| Handler_savepoint          | 0     |
| Handler_savepoint_rollback | 0     |
| Handler_tmp_update         | 0     |
| Handler_tmp_write          | 0     |
| Handler_update             | 0     |
| Handler_write              | 0     |
+----------------------------+-------+
23 rows in set (0.00 sec)
```

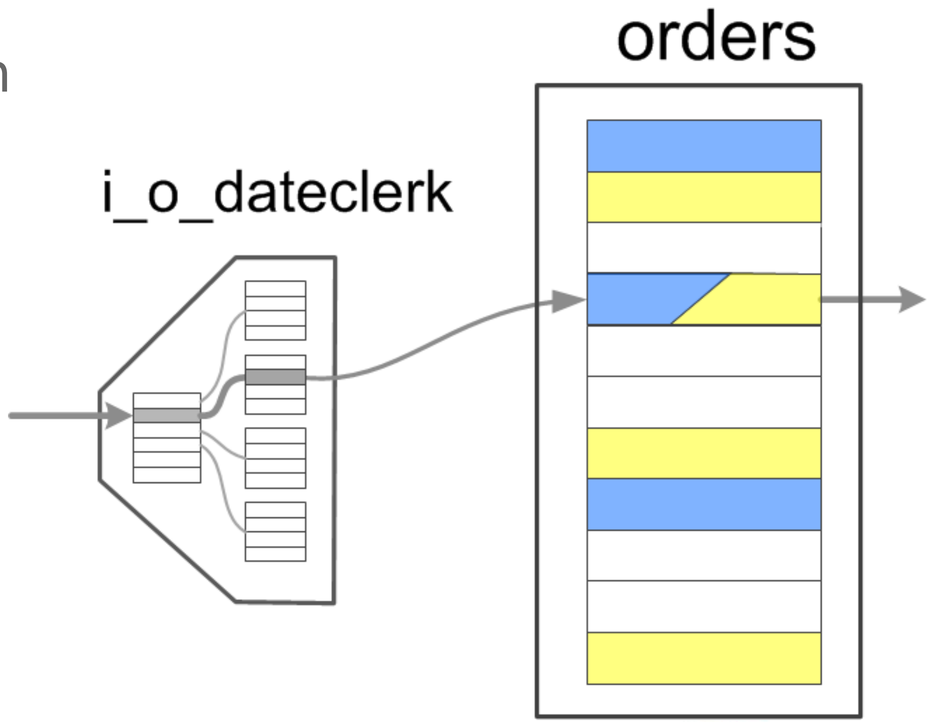
1 index lookup
6K forward index reads

Can have an even better index

```
alter table orders add key i_o_date_clerk (o_orderdate, o_clerk);
select * from orders
where o_orderDate='1992-06-06' and o_clerk='Clerk#000009506';
```

```
+--+-----------+------+----+----------------------------+--------------+-------+-----------+----+---------------------+
|id|select_type|table |type|possible_keys               |key           |key_len|ref        |rows|Extra                |
+--+-----------+------+----+----------------------------+--------------+-------+-----------+----+---------------------+
| 1|SIMPLE     |orders|ref |i_o_orderdate,i_o_date_clerk|i_o_date_clerk|20     |const,const|   1|Using index condition|
+--+-----------+------+----+----------------------------+--------------+-------+-----------+----+---------------------+
```

- Added a wide index covering both columns
- Now, only records that match the whole WHERE will be read
  - Perfect!



Status variables

```
MariaDB [dbt3sf10]> select * from orders  where o_orderDate='1992-06-06' and o_clerk='Clerk#000009506';
... (0.00 sec)

MariaDB [dbt3sf10]> show status like 'Handler%';
+----------------------------+-------+
| Variable_name              | Value |
+----------------------------+-------+
| Handler_commit             | 1     |
| Handler_delete             | 0     |
| Handler_discover           | 0     |
| Handler_icp_attempts       | 0     |
| Handler_icp_match          | 0     |
| Handler_mrr_init           | 0     |
| Handler_mrr_key_refills    | 0     |
| Handler_mrr_rowid_refills  | 0     |
| Handler_prepare            | 0     |
| Handler_read_first         | 0     |
| Handler_read_key           | 1     |
| Handler_read_next          | 0     |
| Handler_read_prev          | 0     |
| Handler_read_rnd           | 0     |
| Handler_read_rnd_deleted   | 0     |
| Handler_read_rnd_next      | 0     |
| Handler_rollback           | 0     |
| Handler_savepoint          | 0     |
| Handler_savepoint_rollback | 0     |
| Handler_tmp_update         | 0     |
| Handler_tmp_write          | 0     |
| Handler_update             | 0     |
| Handler_write              | 0     |
+----------------------------+-------+
23 rows in set (0.00 sec)
```

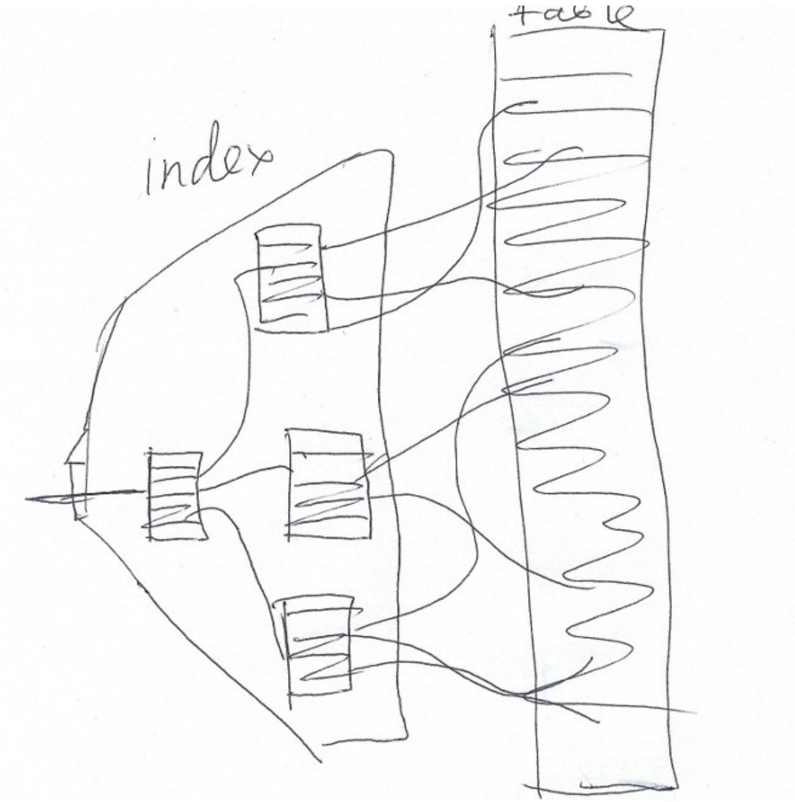
1 index lookup, and that's it

- Let's try to generalize..

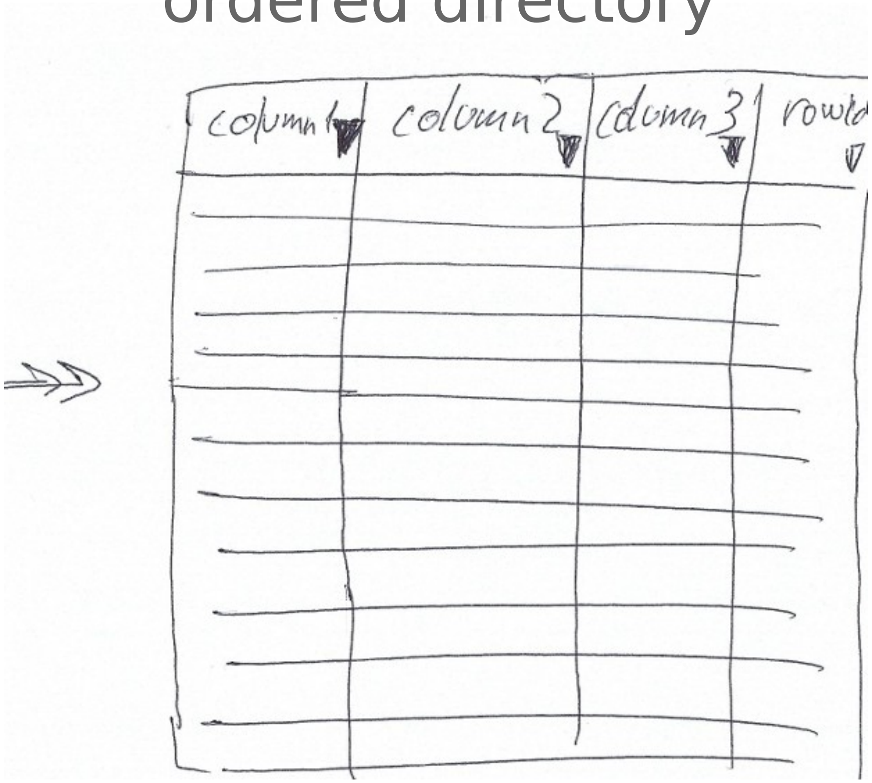
Index fundamentals

ALTER TABLE tbl ADD INDEX(column1, column2, column3)

On the physical level, it's a B-TREE



On a logical level: an ordered directory

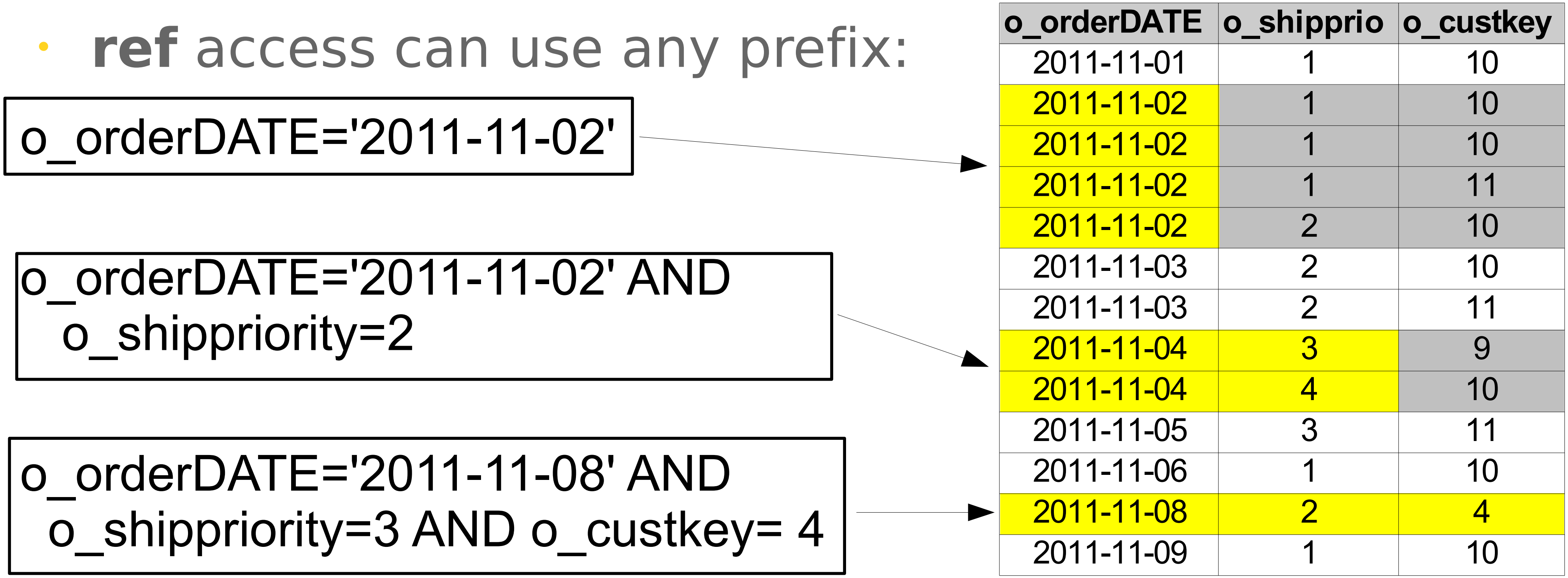


column1	column2	column3	rowid

Querying data through index: 'ref'

alter table orders add index(o_orderDATE, o_shippriority, o_custkey)

- **ref** access can use any prefix:



o_orderDATE='2011-11-02'

o_orderDATE='2011-11-02' AND
o_shippriority=2

o_orderDATE='2011-11-08' AND
o_shippriority=3 AND o_custkey= 4

o_orderDATE	o_shipprio	o_custkey
2011-11-01	1	10
2011-11-02	1	10
2011-11-02	1	10
2011-11-02	1	11
2011-11-02	2	10
2011-11-03	2	10
2011-11-03	2	11
2011-11-04	3	9
2011-11-04	4	10
2011-11-05	3	11
2011-11-06	1	10
2011-11-08	2	4
2011-11-09	1	10

- Not allowed:

o_shipprioirity=2

o_shipprioirity=2

Ref(const) access summary

- Uses equalities to make index lookups
- #rows estimate is usually precise
- ANALYZE will not help

EXPLAIN: type=ref, ref=const, key_len

```
+----+-------------+--------+------+---------------+---------------+---------+-------+------+-------------+
| id | select_type | table  | type | possible_keys | key           | key_len | ref   | rows | Extra       |
+----+-------------+--------+------+---------------+---------------+---------+-------+------+-------------+
|  1 | SIMPLE      | orders | ref  | i_o_orderdate | i_o_orderdate | 4       | const | 6303 | Using where |
+----+-------------+--------+------+---------------+---------------+---------+-------+------+-------------+
```

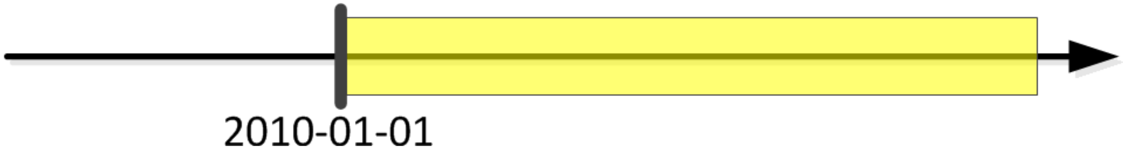
- Status variables:
  - 1 x Handler_read_key
  - N x Handler_read_next
- Estimates and control.. will talk later

Range access

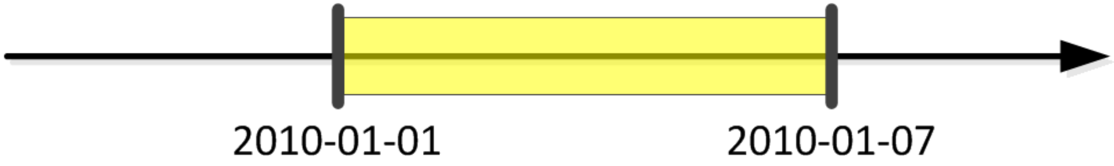
Range access

- Can use equality and non-equality comparisons
- Think of index as a directed axis.
- and “ranges” on it where the WHERE condition is TRUE:

… WHERE o_orderDATE > '2010-01-01'

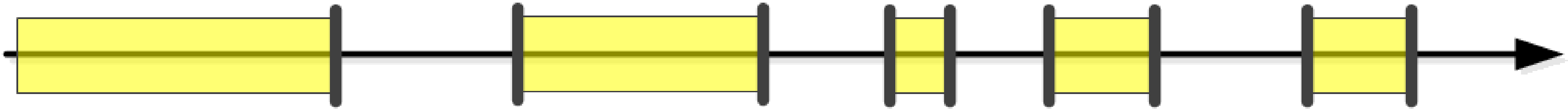


… WHERE o_orderDATE > '2010-01-01' AND o_orderDATE < '2010-01-07'



Range access (2)

- Arbitrarily deep AND/OR formulas are supported
- The optimizer will split out the “useful” parts of WHERE
- And produce a list of disjoint ranges to be scanned:



- “Useful” conditions compare key with constant:

tbl.key>const

tbl.key=const

tbl.key BETWEEN c1 AND c2

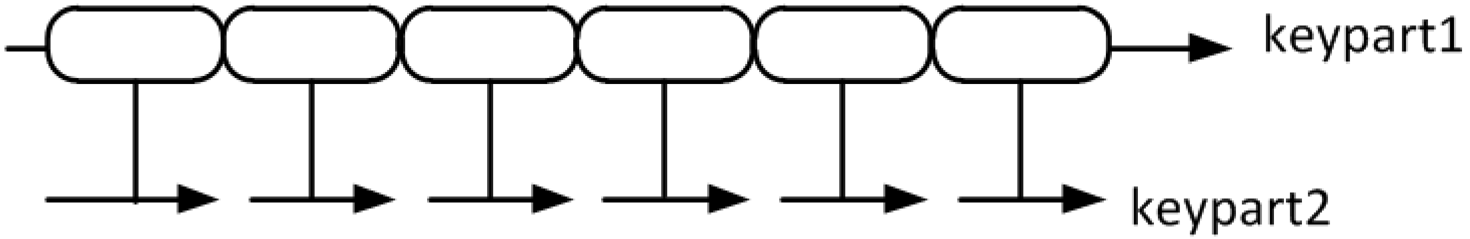
tbl.key IS NULL

tbl.key LIKE 'foo%'

tbl.key IN (const1, const2 ...)

Range access for multi-part keys

- Much more complex
- Ranges over (keypart1, keypart2, …) tuple space
  - Think of it as axis with sub-axis at each point:

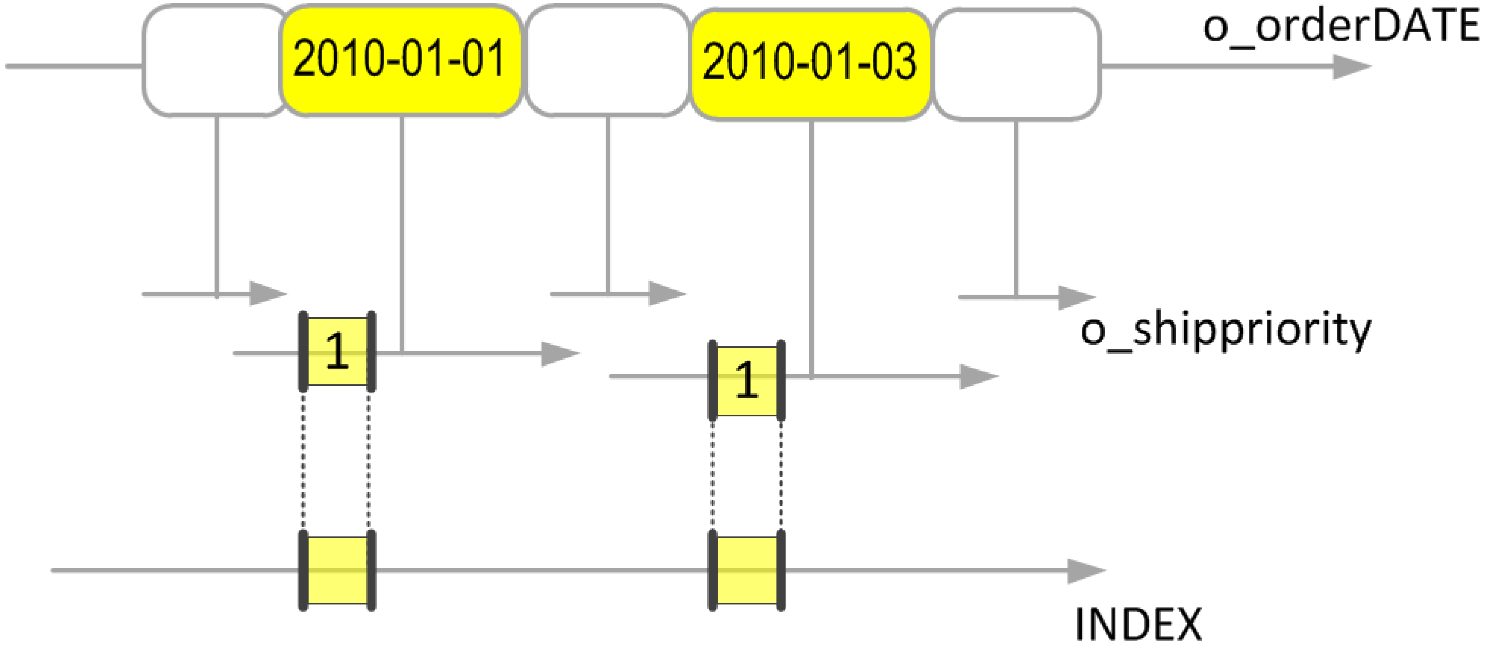


  - Range building rules:
    - Number of ranges is a function of WHERE, not of data in the table
    - Do not use knowledge about domain density (e.g. no knowledge like “there is no integers between 1 and 2”)

Multi-key-part range #1

- Equality on 1st key part? Can put condition on 2nd key part into sub-axis

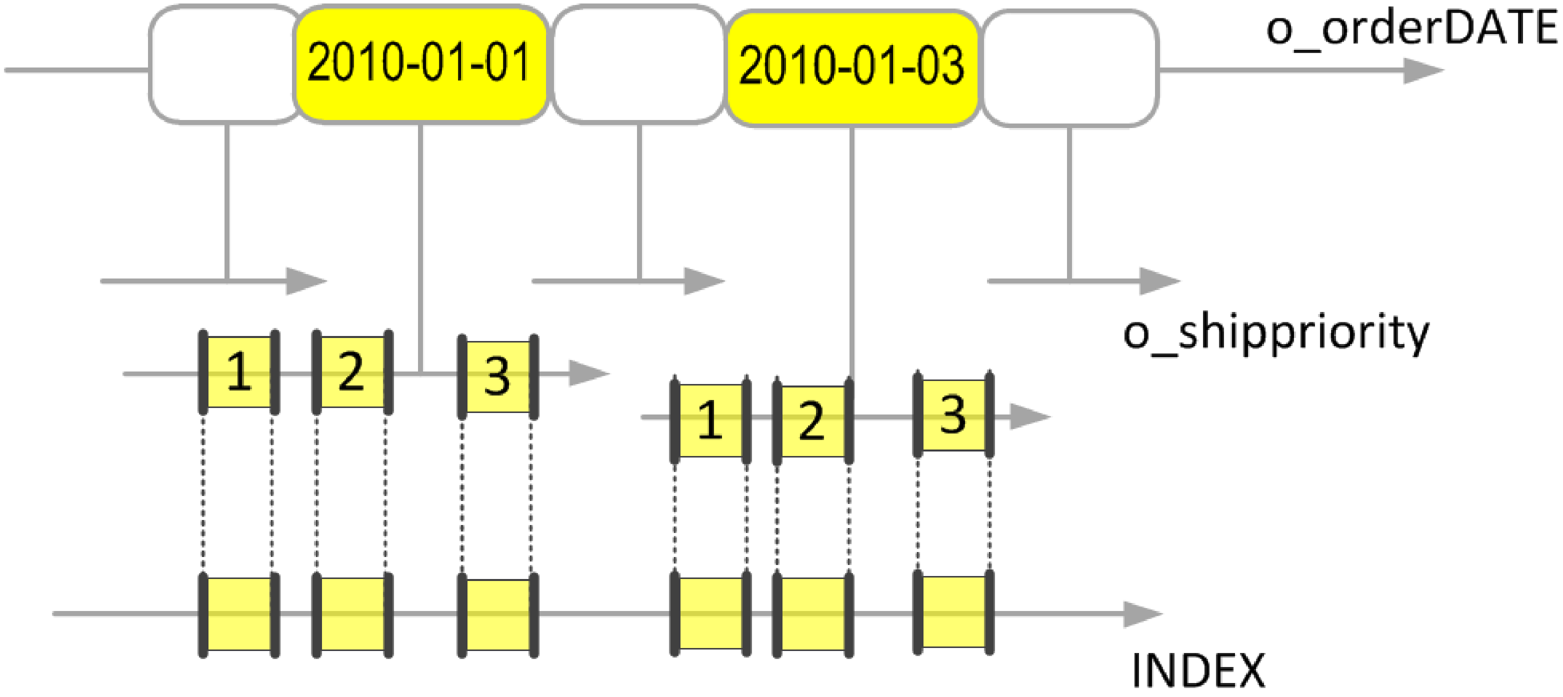
```
alter table orders add key i_o_date_clerk (o_orderdate, o_clerk);
select * from orders
where o_orderDATE IN ('2010-01-01', '2010-01-03') AND o_shippriority=1
```



Multi-key-part range #2

- This can cause big number of ranges. That's ok

```
select * from orders
where
  o_orderDATE IN ('2010-01-01', '2010-01-03') AND
  (o_shippriority =1 OR o_shippriority=2 OR o_shippriority=3)
```

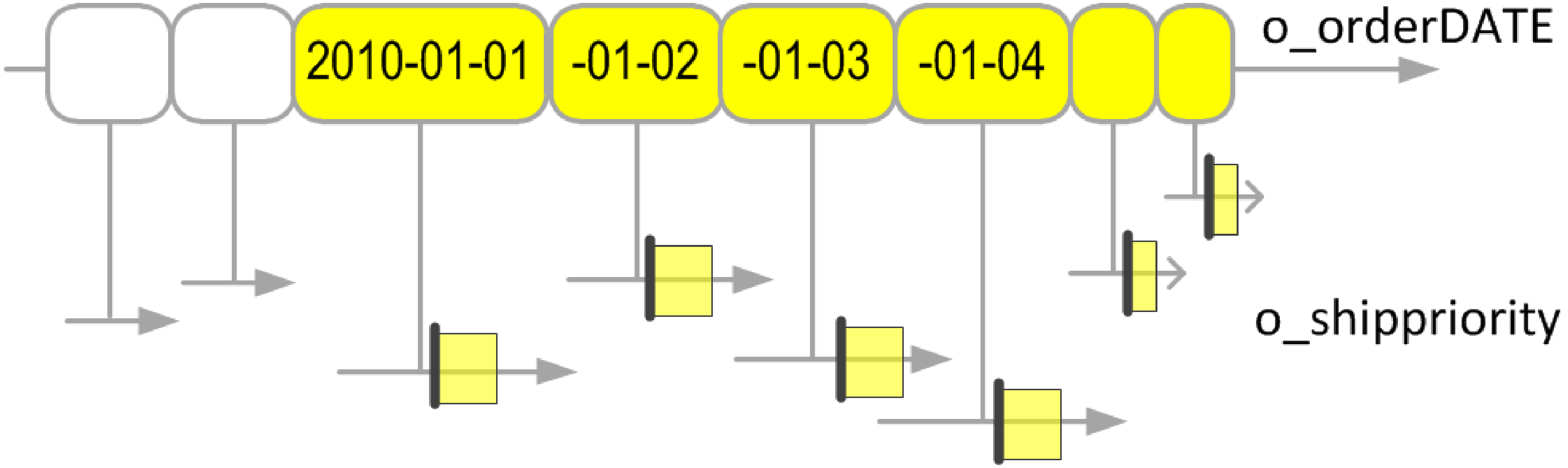


03:14:33 PM

Multi-key-part range #3

- Non-equality on 1st keypart:

```
select * from orders
where o_orderDATE >= '2010-01-01' AND o_shippriority>3
```

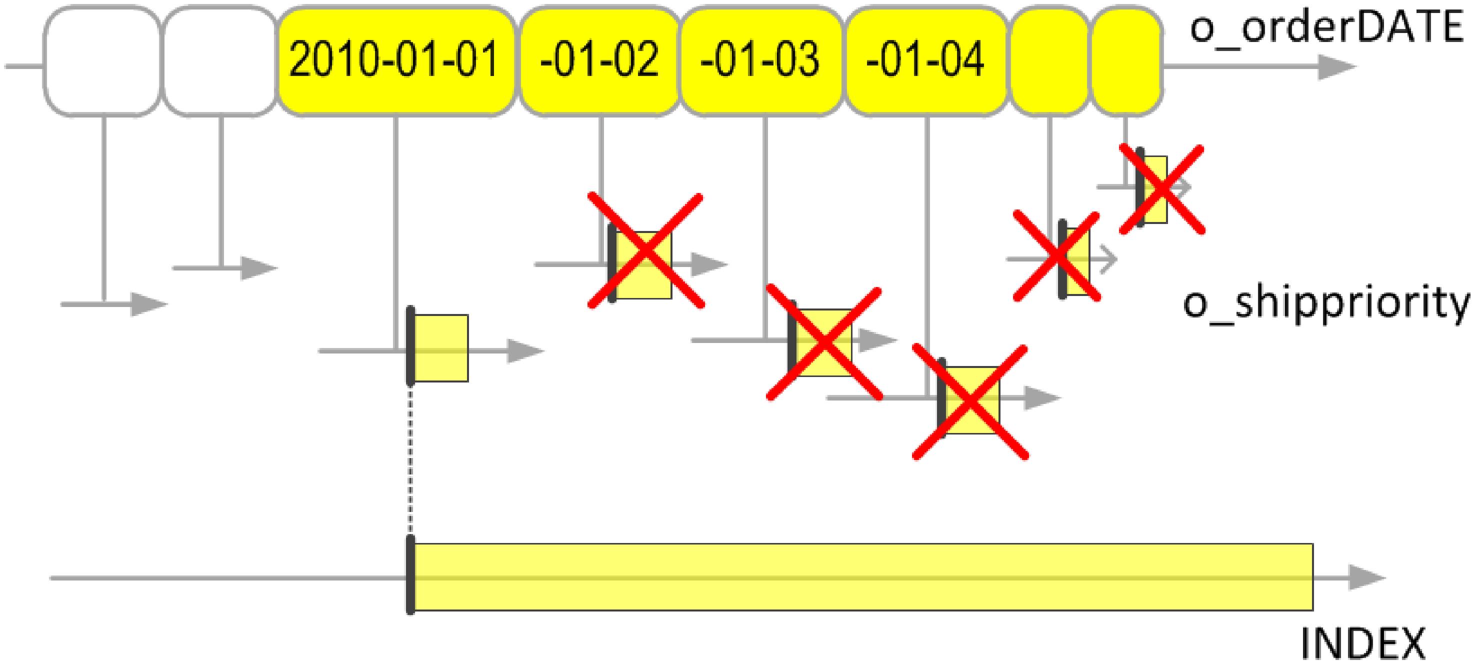


Multi-key-part range #3

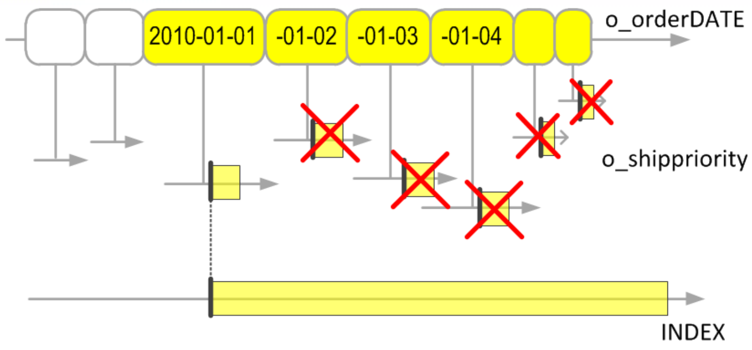
- Non-equality on 1st keypart:

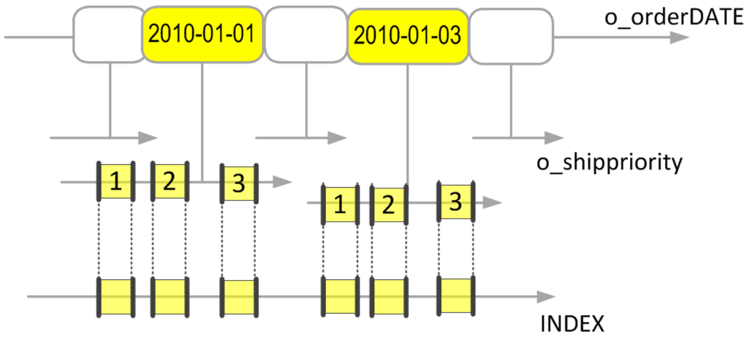
select * from orders
where o_orderDATE >= '2010-01-01' AND o_shippriority>3

- Can attach the second only to the edge “cell”:



Multi-key-part range summary





- keyparts for which queries have equalities should go first
- multiple keyparts with non-equality ranges will not reduce the number of rows read

  e.g. `t.keypart1 BETWEEN … AND t.keypart2 BETWEEN …`
  won't take advantage of condition on keypart2
- When keypartX is followed by other keyparts, then

  `keypartX IN (1,... N)` will work better than `keypartX BETWEEN 1 AND N`

Can I look at the ranges it has constructed?

- Normally, no. But there is a last-resort method:
  - Get a really small subset of your data onto separate server
  - Get a debug build there. Start it with –debug
  - Run your query with FORCE INDEX:

```
explain select * from orders force index(i_o_datepriokey)
where  (o_orderDATE IN ('2010-02-01','2010-02-03') AND o_shippriority < 3) OR
           (o_orderDATE >= '2010-03-10' AND o_shippriority < 4)
```

- Go into mysqld.trace
  - Grep for “query:” until you find the query
  - Grep for “>print_quick”

Ranges in mysqld.trace

mysqld.trace

```
T@10   : | | query: explain select * from orders force index(i_o_datepriokey) where  o_orderDATE
IN ('2010-02-01','2010-02-03') AND o_shippriority < 3 OR (o_orderDATE > '2010-03-10' AND o
_shippriority < 4)
T@10   : | | >PROFILING::set_query_source
T@10   : | | <PROFILING::set_query_source
T@10   : | | >mysql_parse

...

T@10   : | | | | | | | | | | >print_quick
quick range select, key i_o_datepriokey, length: 9
  2010-02-01/NULL < X < 2010-02-01/3
  2010-02-03/NULL < X < 2010-02-03/3
  2010-03-10 < X
other_keys: 0x0:
```

- EXPLAIN only shows key_len which is max # keyparts used

```
+----+-------------+--------+-------+-----------------+-----------------+---------+------+------+--
| id | select_type | table  | type  | possible_keys   | key             | key_len | ref  | rows | E
+----+-------------+--------+-------+-----------------+-----------------+---------+------+------+--
|  1 | SIMPLE      | orders | range | i_o_datepriokey | i_o_datepriokey | 9       | NULL |    3 | U
+----+-------------+--------+-------+-----------------+-----------------+---------+------+------+--
```

Range access estimates

```
+----+-------------+--------+-------+-----------------+-----------------+---------+------+------+--
| id | select_type | table  | type  | possible_keys   | key             | key_len | ref  | rows | E
+----+-------------+--------+-------+-----------------+-----------------+---------+------+------+--
|  1 | SIMPLE      | orders | range | i_o_datepriokey | i_o_datepriokey | 9       | NULL |    3 | U
+----+-------------+--------+-------+-----------------+-----------------+---------+------+------+--
```

- Are obtained by doing dives into the index
  - Dive at the start of the range
  - Dive at the end
  - Estimate the difference
- Usually are precise (not more than 2x wrong)
- Are not affected/do not need ANALYZE TABLE
- Cost you disk IO!
  - Indexes that are never worth using will still be probed!
  - Check your buffer pool + EXPLAINs to see if this happens!
    - Quick fix: IGNORE INDEX (unusable_index)

Range access: Status variables

```
+----------------------------+-------+
| Variable_name              | Value |
+----------------------------+-------+
| Handler_commit             | 1     |
| Handler_delete             | 0     |
| Handler_discover           | 0     |
| Handler_icp_attempts       | 9     |
| Handler_icp_match          | 9     |
| Handler_mrr_init           | 0     |
| Handler_mrr_key_refills    | 0     |
| Handler_mrr_rowid_refills  | 0     |
| Handler_prepare            | 0     |
| Handler_read_first         | 0     |
| Handler_read_key           | 9     |
| Handler_read_next          | 9     |
| Handler_read_prev          | 0     |
| Handler_read_rnd           | 0     |
| Handler_read_rnd_deleted   | 0     |
| Handler_read_rnd_next      | 0     |
| Handler_rollback           | 0     |
| Handler_savepoint          | 0     |
| Handler_savepoint_rollback | 0     |
| Handler_tmp_update         | 0     |
| Handler_tmp_write          | 0     |
| Handler_update             | 0     |
| Handler_write              | 0     |
+----------------------------+-------+
23 rows in set (0.05 sec)
```

Increments

- index lookup counter

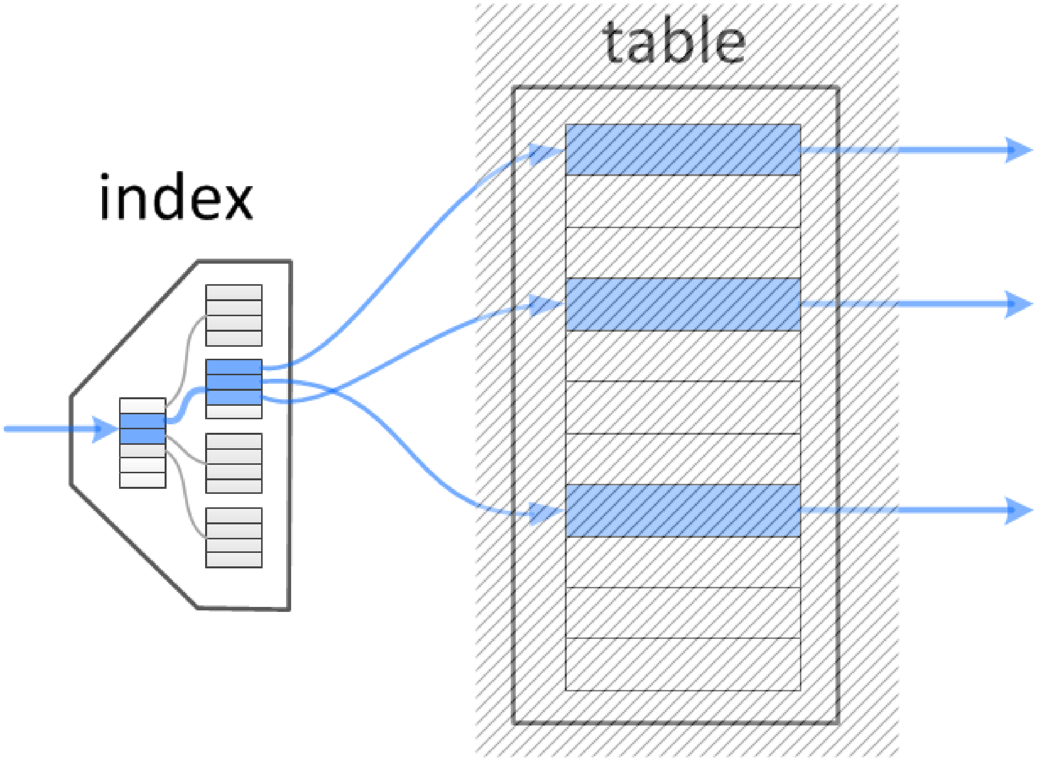
- walk-forward-in-index counter

Index-only scans

Index-only scans

- Besides lookups, indexes have another use

- If the index has all columns you need, it is possible to avoid accessing table rows altogether
- This is called “index only” scan
- Can be applied to **ref**, and **range** accesses.



Index-only read example

```
explain select sum(o_totalprice)
from orders
where o_orderdate between '1995-01-01' and '1995-01-07';
```

```
+--+-----------+------+-----+-------------+-------------+-------+----+----+---------------------+
|id|select_type|table |type |possible_keys|key          |key_len|ref |rows|Extra                |
+--+-----------+------+-----+-------------+-------------+-------+----+----+---------------------+
| 1|SIMPLE     |orders|range|i_o_orderdate|i_o_orderdate|4      |NULL|4358|Using index condition|
+--+-----------+------+-----+-------------+-------------+-------+----+----+---------------------+
```

```
alter table orders add key o_date_price (o_orderDATE, o_totalprice);
# re-run the EXPLAIN:
```

```
alter table orders add key o_date_price (o_orderDATE, o_totalprice);
explain select sum(o_totalprice) from orders where o_orderdate between '1995-01-01' and '1995-02-01';
+--+-----------+------+-----+-------------+------------+-------+----+-----+------------------------+
|id|select_type|table |type |possible_keys|key         |key_len|ref |rows |Extra                   |
+--+-----------+------+-----+-------------+------------+-------+----+-----+------------------------+
| 1|SIMPLE     |orders|range|...          |o_date_price|4      |NULL|39424|Using where; Using index|
+--+-----------+------+-----+-------------+------------+-------+----+-----+------------------------+
```

- scale=1, in-mem: from 0.2 to 0.1 sec

Index-only reads and status variables

- Surprisingly, no difference

```
select sum(o_totalprice)
from orders use index(o_date_price)
where o_orderdate between '1995-01-01' and
                          '1995-02-01';
+----------------------------+-------+
| Variable_name              | Value |
+----------------------------+-------+
| Handler_commit             | 1     |
| Handler_delete             | 0     |
| Handler_discover           | 0     |
| Handler_icp_attempts       | 0     |
| Handler_icp_match          | 0     |
| Handler_mrr_init           | 0     |
| Handler_mrr_key_refills    | 0     |
| Handler_mrr_rowid_refills  | 0     |
| Handler_prepare            | 0     |
| Handler_read_first         | 0     |
| Handler_read_key           | 1     |
| Handler_read_next          | 20137 |
| Handler_read_prev          | 0     |
| Handler_read_rnd           | 0     |
| Handler_read_rnd_deleted   | 0     |
| Handler_read_rnd_next      | 0     |
| Handler_rollback           | 0     |
| Handler_savepoint          | 0     |
| Handler_savepoint_rollback | 0     |
| Handler_tmp_update         | 0     |
| Handler_tmp_write          | 0     |
| Handler_update             | 0     |
| Handler_write              | 0     |
+----------------------------+-------+
```

```
select sum(o_totalprice)
from orders use index(i_o_orderdate)
where o_orderdate between '1995-01-01' and
                          '1995-02-01';
+----------------------------+-------+
| Variable_name              | Value |
+----------------------------+-------+
| Handler_commit             | 1     |
| Handler_delete             | 0     |
| Handler_discover           | 0     |
| Handler_icp_attempts       | 20137 |
| Handler_icp_match          | 20137 |
| Handler_mrr_init           | 0     |
| Handler_mrr_key_refills    | 0     |
| Handler_mrr_rowid_refills  | 0     |
| Handler_prepare            | 0     |
| Handler_read_first         | 0     |
| Handler_read_key           | 1     |
| Handler_read_next          | 20137 |
| Handler_read_prev          | 0     |
| Handler_read_rnd           | 0     |
| Handler_read_rnd_deleted   | 0     |
| Handler_read_rnd_next      | 0     |
| Handler_rollback           | 0     |
| Handler_savepoint          | 0     |
| Handler_savepoint_rollback | 0     |
| Handler_tmp_update         | 0     |
| Handler_tmp_write          | 0     |
| Handler_update             | 0     |
| Handler_write              | 0     |
+----------------------------+-------+
```

Index-only reads and status variables

- Let's look at InnoDB “lower-level” counters

```
show status like 'Innodb%';
+----------------------------------+--------+
| Variable_name                    | Value  |
+----------------------------------+--------+
...
| Innodb_buffer_pool_read_requests | 413168 | +68929
| Innodb_buffer_pool_reads         | 10034  |
| Innodb_buffer_pool_wait_free     | 0      |
...
| Innodb_rows_inserted             | 0      |
| Innodb_rows_read                 | 181237 | +20137
| Innodb_rows_updated              | 0      |
+----------------------------------+--------+
```

```
show status like 'Innodb%';
+----------------------------------+--------+
| Variable_name                    | Value  |
+----------------------------------+--------+
...
| Innodb_buffer_pool_read_requests | 422426 | +2566
| Innodb_buffer_pool_reads         | 10034  |
| Innodb_buffer_pool_wait_free     | 0      |
...
| Innodb_rows_inserted             | 0      |
| Innodb_rows_read                 | 241651 | +20138
| Innodb_rows_updated              | 0      |
+----------------------------------+--------+
```

- These are global non FLUSHable counters, so we manually calculate increment and show it on the right
- _rows_read are the same, too
- _buffer_pool_reads are different
- A bit sense-less counting
  - Have plans to fix in MariaDB.

Index-only reads summary

- Can be used with any index-based access methods
- Allows to skip accessing table records
  - Some savings for CPU-bound loads
  - Huge savings for IO-bound loads
  - Extra columns in the index a cost, though
    - Index is bigger
    - [Potentially] more frequently updated
- EXPLAIN shows “Using index”
- Counting in for Handler_xxx and Innodb_row_reads status variables
  - Watch lower-level counters.

Index Condition Pushdown

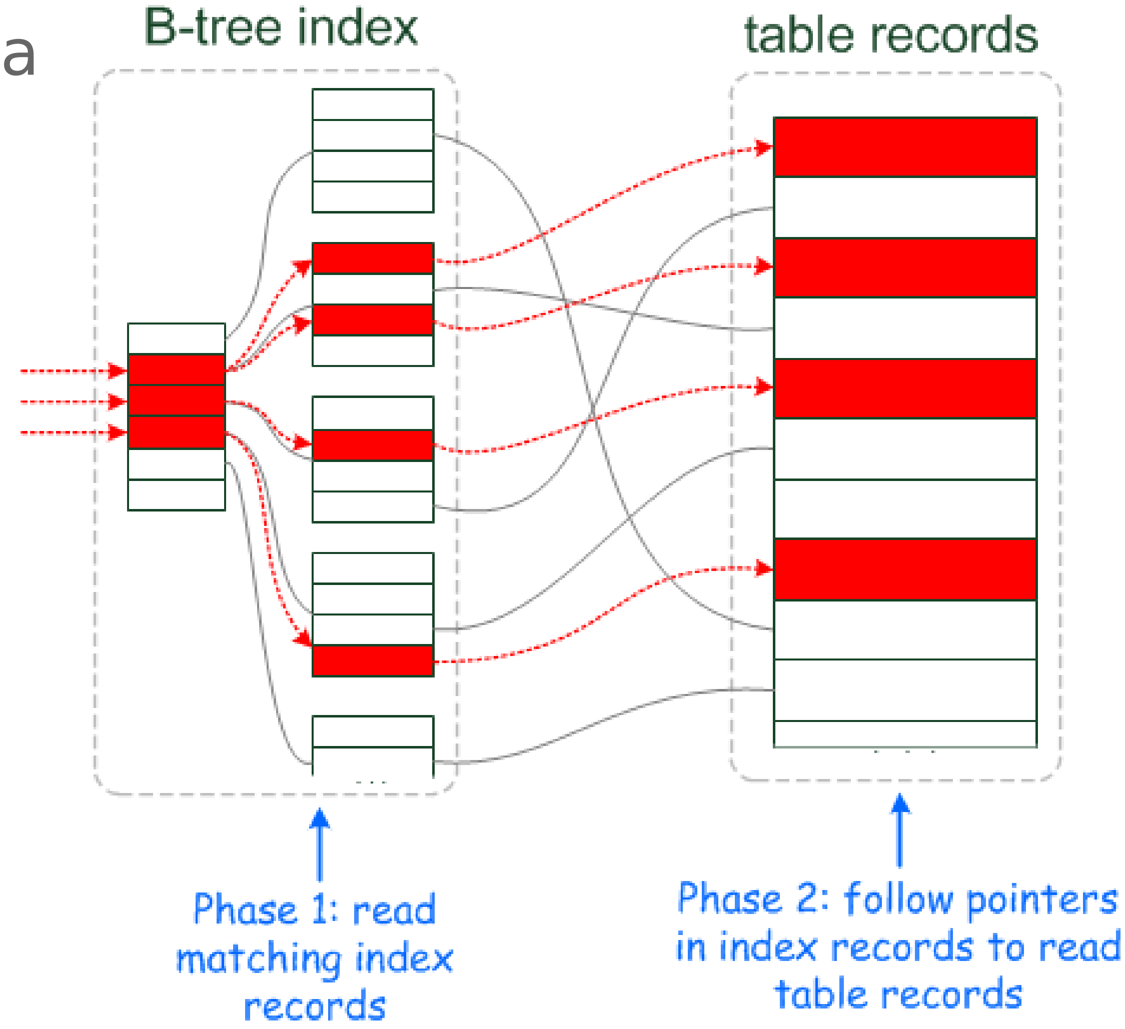
MySQL 5.6+, MariaDB 5.3+

Index Condition Pushdown idea

A non-index only read is a two-step process:

1. Read index
2. Read record

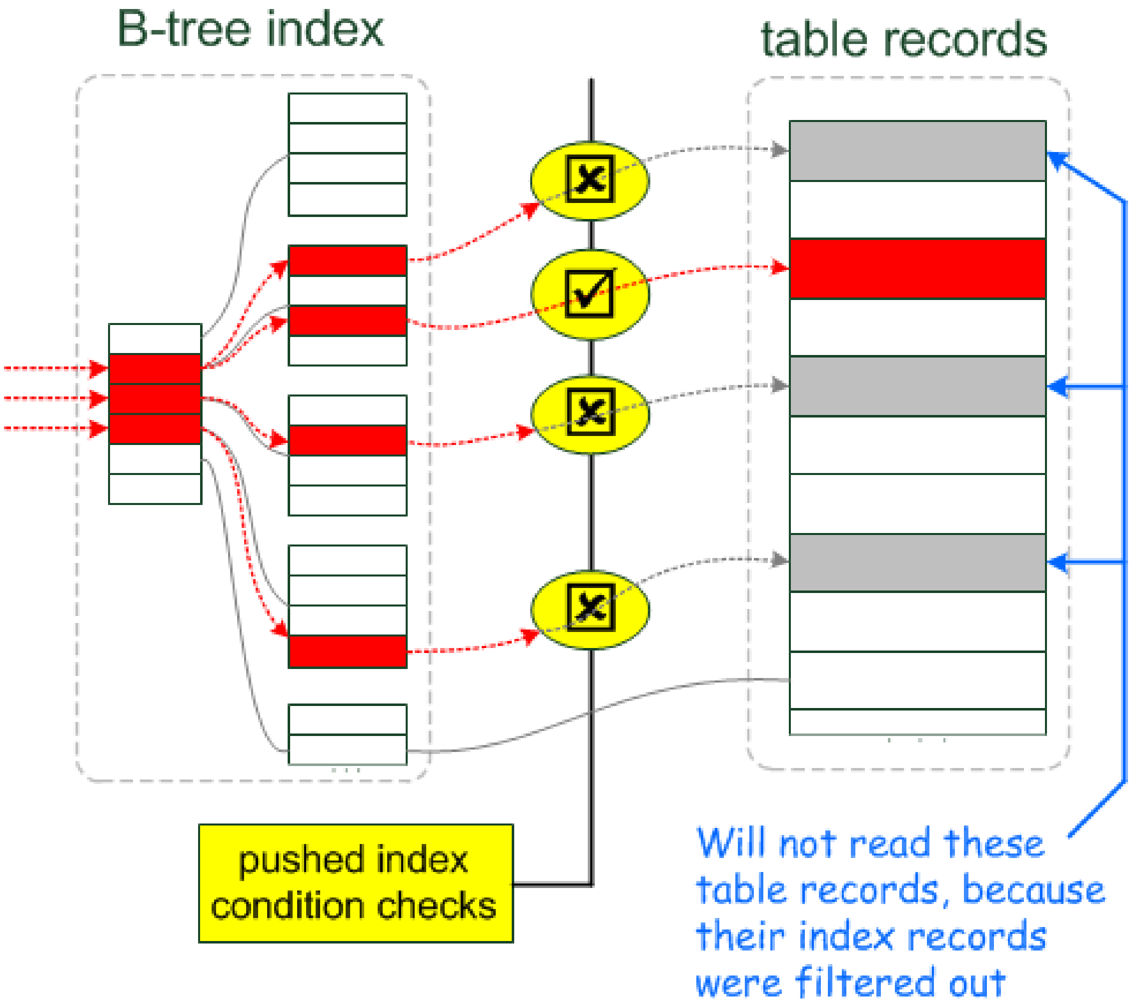
3. Check the WHERE condition.


B-tree index
table records
Phase 1: read matching index records
Phase 2: follow pointers in index records to read table records

Index Condition Pushdown idea

Index Condition Pushdown

1. Read index
2. Check condition on index columns
3. Read record
4. Check the WHERE condition


B-tree index
table records
pushed index condition checks
Will not read these table records, because their index records were filtered out

Index condition pushdown example

- Start without ICP

```
alter table orders add key i_o_clerk(o_clerk);
select o_clerk, o_orderkey, o_shippriority
from orders
where o_clerk IN ('Clerk#000000001', 'Clerk#00000201' , 'Clerk#000003001') and
         dayofweek(o_orderDATE)=1;
```

```
# MariaDB 5.2.x/ MySQL 5.5.x
+--+-----------+------+-----+-------------+---------+-------+----+----+-----------+
|id|select_type|table |type |possible_keys|key      |key_len|ref |rows|Extra      |
+--+-----------+------+-----+-------------+---------+-------+----+----+-----------+
| 1|SIMPLE     |orders|range|i_o_clerk    |i_o_clerk|16     |NULL|2978|Using where|
+--+-----------+------+-----+-------------+---------+-------+----+----+-----------+
```

```
# MariaDB 5.3+, MySQL 5.6+
+--+-----------+------+-----+-------------+---------+-------+----+----+----------------------------------+
|id|select_type|table |type |possible_keys|key      |key_len|ref |rows|Extra                             |
+--+-----------+------+-----+-------------+---------+-------+----+----+----------------------------------+
| 1|SIMPLE     |orders|range|i_o_clerk    |i_o_clerk|16     |NULL|2978|Using index condition; Using where|
+--+-----------+------+-----+-------------+---------+-------+----+----+----------------------------------+
```

Counters

alter table orders add key i_o_clerk(o_clerk);
select o_clerk, o_orderkey, o_shippriority
from orders
where o_clerk IN ('Clerk#000000001', 'Clerk#00000201' , 'Clerk#000003001') and
 dayofweek(o_orderDATE)=1;

```
# MariaDB 5.3+, MySQL 5.6+
+--+-----------+------+-----+-------------+---------+-------+----+----+-------------------------------------+
|id|select_type|table |type |possible_keys|key      |key_len|ref |rows|Extra                                |
+--+-----------+------+-----+-------------+---------+-------+----+----+-------------------------------------+
| 1|SIMPLE     |orders|range|i_o_clerk    |i_o_clerk|16     |NULL|2978|Using index condition; Using where|
+--+-----------+------+-----+-------------+---------+-------+----+----+-------------------------------------+
```

```
+----------------------------+-------+
| Variable_name              | Value |
+----------------------------+-------+
| Handler_commit             | 1     |
| Handler_delete             | 0     |
| Handler_discover           | 0     |
| Handler_icp_attempts       | 2979  |
| Handler_icp_match          | 2979  |
| Handler_mrr_init           | 0     |
| Handler_mrr_key_refills    | 0     |
| Handler_mrr_rowid_refills  | 0     |
| Handler_prepare            | 0     |
| Handler_read_first         | 0     |
| Handler_read_key           | 3     |
| Handler_read_next          | 2979  |
| Handler_read_prev          | 0     |
| Handler_read_rnd           | 0     |
...
```

```
MariaDB [(none)]> show status like 'Innodb%';
+---------------------------------+-------+
| Variable_name                   | Value |
+---------------------------------+-------+
| Innodb_buffer_pool_read_requests | 32103 | + 24416
| Innodb_rows_read                | 3406  | + 2979
```

Index condition pushdown example

- Now, add an index that ICP could use:

```
alter table orders add key i_o_clerk_date(o_clerk, o_orderDATE);
select o_clerk, o_orderkey, o_shippriority
from orders
where o_clerk IN ('Clerk#000000001', 'Clerk#00000201' , 'Clerk#000003001') and
        dayofweek(o_orderDATE)=1;
```

```
+--+-----------+------+-----+------------------------+--------------+-------+----+----+---------------------+
|id|select_type|table |type |possible_keys           |key           |key_len|ref |rows|Extra                |
+--+-----------+------+-----+------------------------+--------------+-------+----+----+---------------------+
| 1|SIMPLE     |orders|range|i_o_clerk_date,i_o_clerk|i_o_clerk_date|16     |NULL|2978|Using index condition|
+--+-----------+------+-----+------------------------+--------------+-------+----+----+---------------------+
```

Counters

```
alter table orders add key i_o_clerk_date(o_clerk, o_orderDATE);
select o_clerk, o_orderkey, o_shippriority
from orders
where o_clerk IN ('Clerk#000000001', 'Clerk#00000201' , 'Clerk#000003001') and
        dayofweek(o_orderDATE)=1;
```

```
+--+-----------+------+-----+---------------------+------------+-------+----+----+--------------------+
|id|select_type|table |type |possible_keys        |key         |key_len|ref |rows|Extra               |
+--+-----------+------+-----+---------------------+------------+-------+----+----+--------------------+
| 1|SIMPLE     |orders|range|i_o_clerk_date,i_o_clerk|i_o_clerk_date|16  |NULL|2978|Using index condition|
+--+-----------+------+-----+---------------------+------------+-------+----+----+--------------------+
```

```
+-----------------------------+-------+
| Variable_name               | Value |
+-----------------------------+-------+
| Handler_commit              | 2     |
| Handler_delete              | 0     |
| Handler_discover            | 0     |
| Handler_icp_attempts        | 2979  |
| Handler_icp_match           | 427   |  = 1/7th
| Handler_mrr_init            | 0     |
| Handler_mrr_key_refills     | 0     |
| Handler_mrr_rowid_refills   | 0     |
| Handler_prepare             | 0     |
| Handler_read_first          | 0     |
| Handler_read_key            | 3     |
| Handler_read_next           | 427   |  = 1/7th
| Handler_read_prev           | 0     |
| Handler_read_rnd            | 0     |
```

```
MariaDB [(none)]> show status like 'Innodb%';
+-----------------------------------+-------+
| Variable_name                     | Value |
+-----------------------------------+-------+
| Innodb_buffer_pool_read_requests  | 7687  | +2585
| Innodb_rows_read                  | 427   | +427
```

Index Condition Pushdown optimization … or lack thereof

```
alter table orders add key i_o_clerk_date(o_clerk, o_orderDATE);
select o_clerk, o_orderkey, o_shippriority
from orders
where o_clerk IN ('Clerk#000000001', 'Clerk#00000201' , 'Clerk#000003001') and
        dayofweek(o_orderDATE)=1;
```

- Range selectivity on 1st keypart is (or, should be) the same for both

INDEX (o_clerk)

- Index entries are smaller

INDEX (o_clerk, o_orderDATE);

- ICP is applicable
  - But is condition dayofweek(...)=1 selective?

Current optimizer actions:

- #1. Choose an index to use
- #2. If possible, use IndexConditionPushdown with it
  - => Possibility of ICP doesn't affect index choice.

Index Condition Pushdown optimization … or lack thereof

What to do, then?

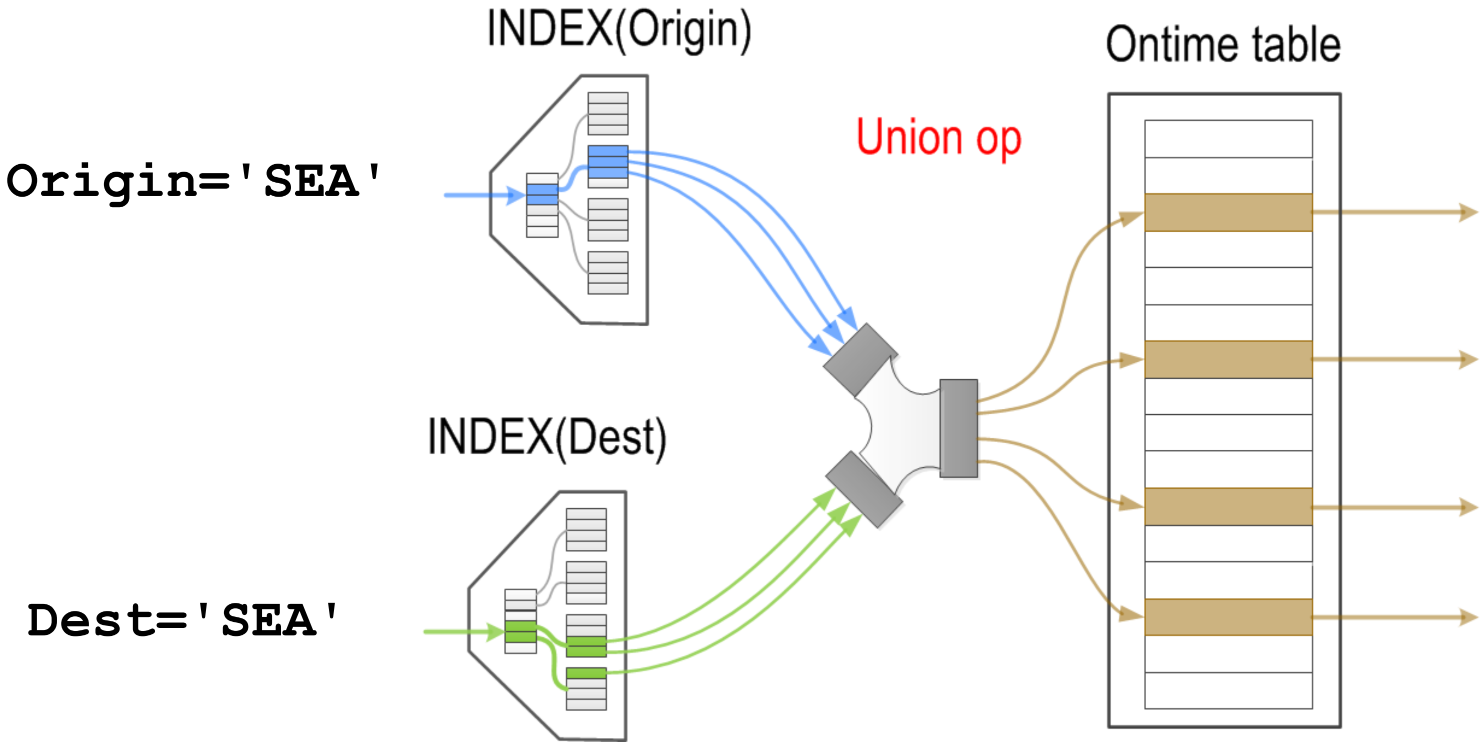
- USE/IGNORE INDEX hints
- Create only indexes such that ICP will be used
  - Include approriate extra columns at the end
  - In our example:
    - Create INDEX (o_clerk, o_orderDATE)
    - Don't create INDEX (o_clerk)

Index Merge

Index Merge Union

- ORed conditions on different columns cannot be resolved with any single index
- Example: airline on-time performance data, 15M flights:

```
select * from ontime where (Origin='SEA' or Dest='SEA');
```

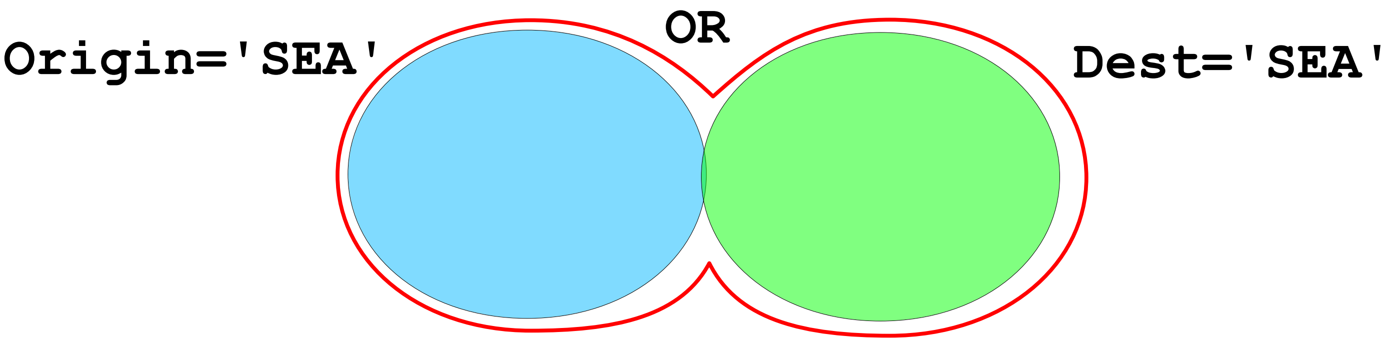


Index Merge union

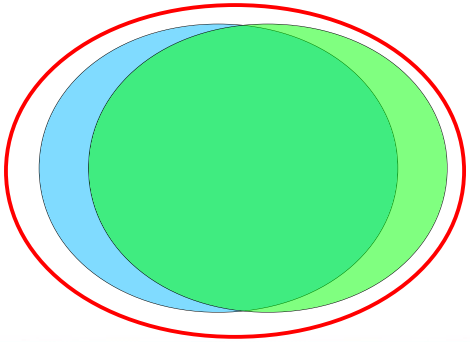
```
select * from ontime where (Origin='SEA' or Dest='SEA');
```

```
+--+-----------+------+-----------+-------------+-----------+-------+----+-----+------------------------------------+
|id|select_type|table |type       |possible_keys|key        |key_len|ref |rows |Extra                               |
+--+-----------+------+-----------+-------------+-----------+-------+----+-----+------------------------------------+
| 1|SIMPLE     |ontime|index_merge|Origin,Dest  |Origin,Dest|6,6    |NULL|92850|Using union(Origin,Dest); Using where|
+--+-----------+------+-----------+-------------+-----------+-------+----+-----+------------------------------------+
```

- Optimization is challenging:



v.s.



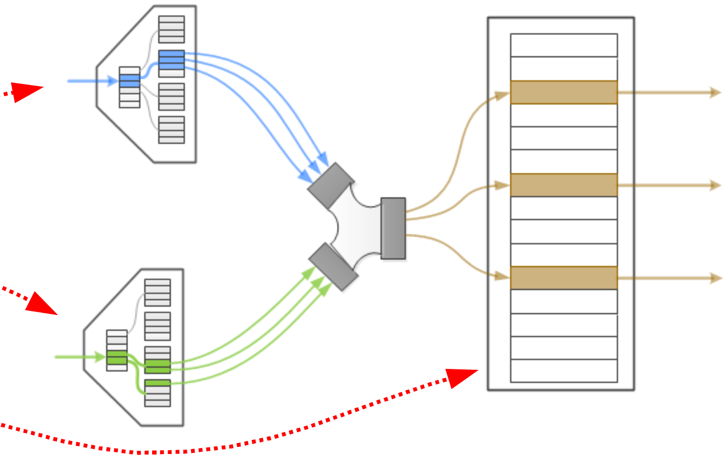
...no way to tell.

Current assumtion: pessimistic (top)

Index merge: statistics counters

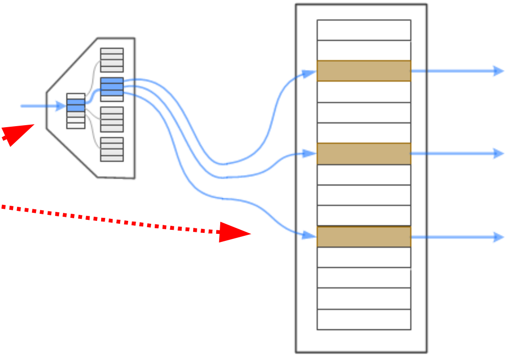
select count(*) from ontime where (Origin='SEA' or Dest='SEA'); -- 46412 rows

```
+-----------------------------+--------+
| Variable_name               | Value  |
+-----------------------------+--------+
...
| Handler_read_first          | 0      |
| Handler_read_key            | 2      |
| Handler_read_next           | 46412  |
| Handler_read_prev           | 0      |
| Handler_read_rnd            | 46412  |
| Handler_read_rnd_deleted    | 0      |
| Handler_read_rnd_next       | 0      |
...
| Innodb_rows_read            | 464120 | +92824
```



select count(*) from ontime where Origin='SEA' -- 23207 rows

```
+-----------------------------+--------+
| Variable_name               | Value  |
+-----------------------------+--------+
...
| Handler_read_first          | 0      |
| Handler_read_key            | 1      |
| Handler_read_next           | 23207  |
| Handler_read_prev           | 0      |
| Handler_read_rnd            | 0      |
| Handler_read_rnd_deleted    | 0      |
| Handler_read_rnd_next       | 0      |
...
| Innodb_rows_read            | 556943 | +23207
```



1.5X reads look like 3x!

Index merge properties

[MySQL, Percona, MariaDB 5.2x]

index_merge plans are removed from consideration if a range access is possible:

```
MySQL [ontime]> explain select * from ontime where (Origin='SEA' or Dest='SEA') and securitydelay=0;
+--+-----------+------+----+-------------------------+-------------+-------+-----+------+-----------+
|id|select_type|table |type|possible_keys            |key          |key_len|ref  |rows  |Extra      |
+--+-----------+------+----+-------------------------+-------------+-------+-----+------+-----------+
| 1|SIMPLE     |ontime|ref |Origin,Dest,SecurityDelay|SecurityDelay|5      |const|791546|Using where|
+--+-----------+------+----+-------------------------+-------------+-------+-----+------+-----------+
```

```
MySQL [ontime]> explain select * from ontime where (Origin='SEA' or Dest='SEA') and depdelay < 12*60;
+--+-----------+------+----+--------------------+----+-------+----+-------+-----------+
|id|select_type|table |type|possible_keys       |key |key_len|ref |rows   |Extra      |
+--+-----------+------+----+--------------------+----+-------+----+-------+-----------+
| 1|SIMPLE     |ontime|ALL |Origin,DepDelay,Dest|NULL|NULL   |NULL|1583093|Using where|
+--+-----------+------+----+--------------------+----+-------+----+-------+-----------+
```

Solutions:

- Upgrade to MariaDB 5.3+ :-)
- IGNORE INDEX(*$index_used_for_range*)
  - there is no “FORCE INDEX MERGE” hint

Index Merge/Union summary

- Used for ORs spanning multiple indexes
  - Each part of OR must have an index it could use
- Counter increments not quite comparable with single-index access methods
- Can be turned off globally with
  `set optimizer_switch='index_merge=off'`
- Can be turned off locally with
  `IGNORE INDEX(some_merged_index)`
- Can be blocked by potential ref/range accesses
  - Fix1: Upgrade to MariaDB 5.3
  - Fix2: `IGNORE INDEX(other_range_indexes)`
    `USE INDEX (index_merge_indexes)`

Index Merge/Intersection

- Like Index Merge/union, but for ANDs
- Aimed at cases when there is no composite index

select avg(arrdelay) from ontime where depdel15=1 and OriginState ='CA';

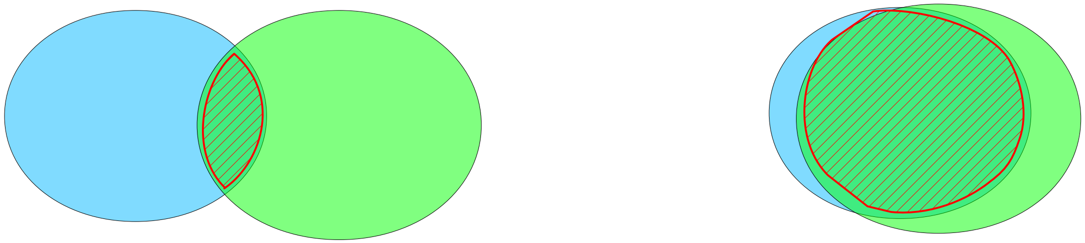
```
-----------+------+-----------+--------------------+--------------------+-------+----+-----+----------------------------------------
select_type|table |type       |possible_keys       |key                 |key_len|ref |rows |Extra
-----------+------+-----------+--------------------+--------------------+-------+----+-----+----------------------------------------
SIMPLE     |ontime|index_merge|OriginState,DepDel15|OriginState,DepDel15|3,5    |NULL|76952|Using intersect(OriginState,DepDel15);Us
-----------+------+-----------+--------------------+--------------------+-------+----+-----+----------------------------------------
```

- MySQL, MariaDB 5.2: only support equality conditions
- MariaDB 5.3: any range condition supported
  - Must be manually enabled:
    set optimizer_switch='index_merge_sort_intersection=on'
  - EXPLAIN: Using sort-intersect

Index Merge/intersect optimization

- Optimization challenges are similar to UNION

select avg(arrdelay) from ontime where depdel15=1 and OriginState ='CA';



- There is no way to force index_merge/intersect
- To disable
  - set optimizer_switch='index_merge_intersection=off
  - IGNORE INDEX(one_of_used_indexes)
- Tip: showeling through lots of data?

set optimizer_switch='index_merge_sort_intersection=on'
set sort_buffer_size=...

JOINs

Consider a join

```
select count(*)
from
  customer, orders
where
  c_custkey=o_custkey and c_acctbal < -500 and o_orderpriority='1-URGENT';
```

```
+--+-----------+--------+-----+-------------+-----------+-------+------------------+----+--------------------
|id|select_type|table   |type |possible_keys|key        |key_len|ref               |rows|Extra
+--+-----------+--------+-----+-------------+-----------+-------+------------------+----+--------------------
| 1|SIMPLE     |customer|range|...          |c_acctbal  |9      |NULL              |6802|Using where; Using i
| 1|SIMPLE     |orders  |ref  |i_o_custkey  |i_o_custkey|5      |customer.c_custkey|   7|Using where
+--+-----------+--------+-----+-------------+-----------+-------+------------------+----+--------------------
```

Nested loops join

```
select count(*)
from
  customer, orders
where
  c_custkey=o_custkey and c_acctbal < -500 and o_orderpriority='1-URGENT';
```

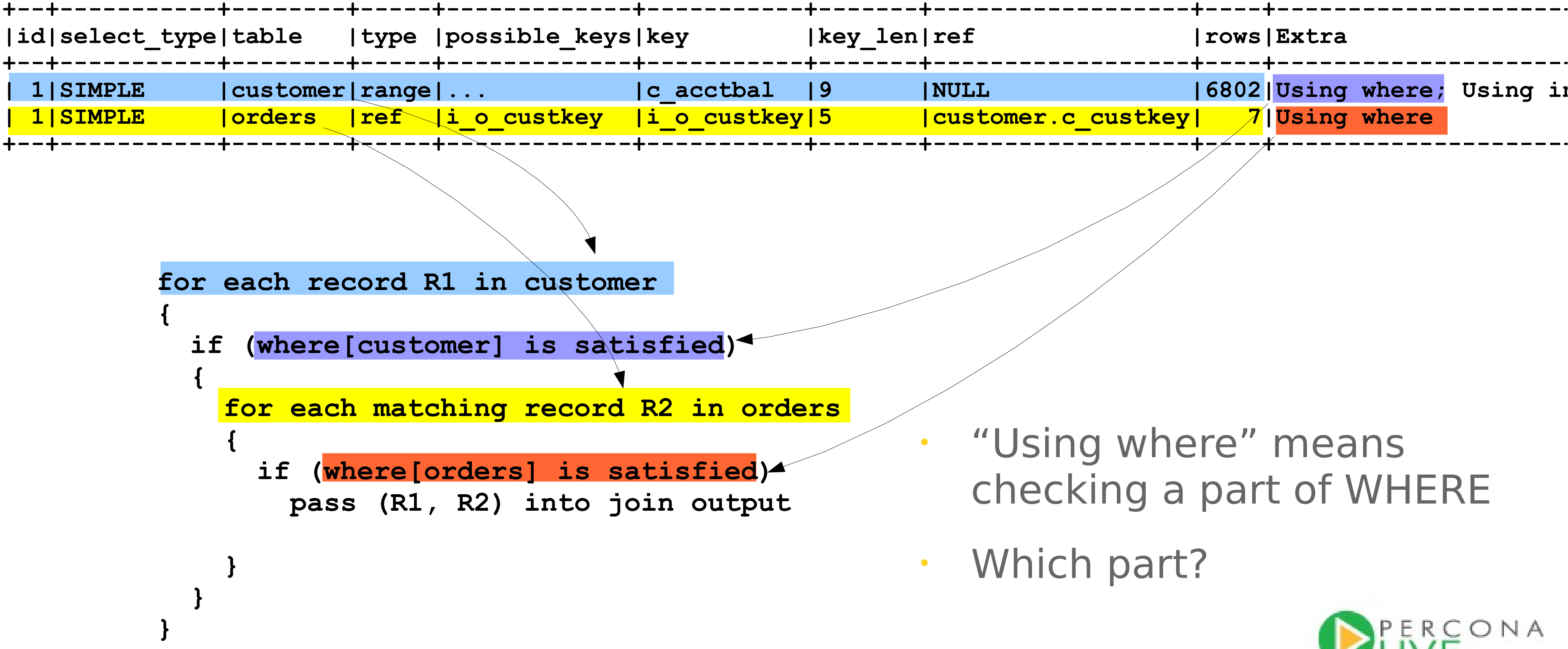
```
+--+-----------+--------+-----+-------------+-----------+-------+------------------+----+------------------
|id|select_type|table   |type |possible_keys|key        |key_len|ref               |rows|Extra
+--+-----------+--------+-----+-------------+-----------+-------+------------------+----+------------------
| 1|SIMPLE     |customer|range|...          |c_acctbal  |9      |NULL              |6802|Using where; Using i
| 1|SIMPLE     |orders  |ref  |i_o_custkey  |i_o_custkey|5      |customer.c_custkey|   7|Using where
+--+-----------+--------+-----+-------------+-----------+-------+------------------+----+------------------
```

```
for each record R1 in customer
{
    for each matching record R2 in orders
    {
        pass (R1, R2) into join output
    }
}
```

- EXPLAIN shows the loops, from outer to inner

Nested loops join

```
select count(*)
from
  customer, orders
where
  c_custkey=o_custkey and c_acctbal < -500 and o_orderpriority='1-URGENT';
```



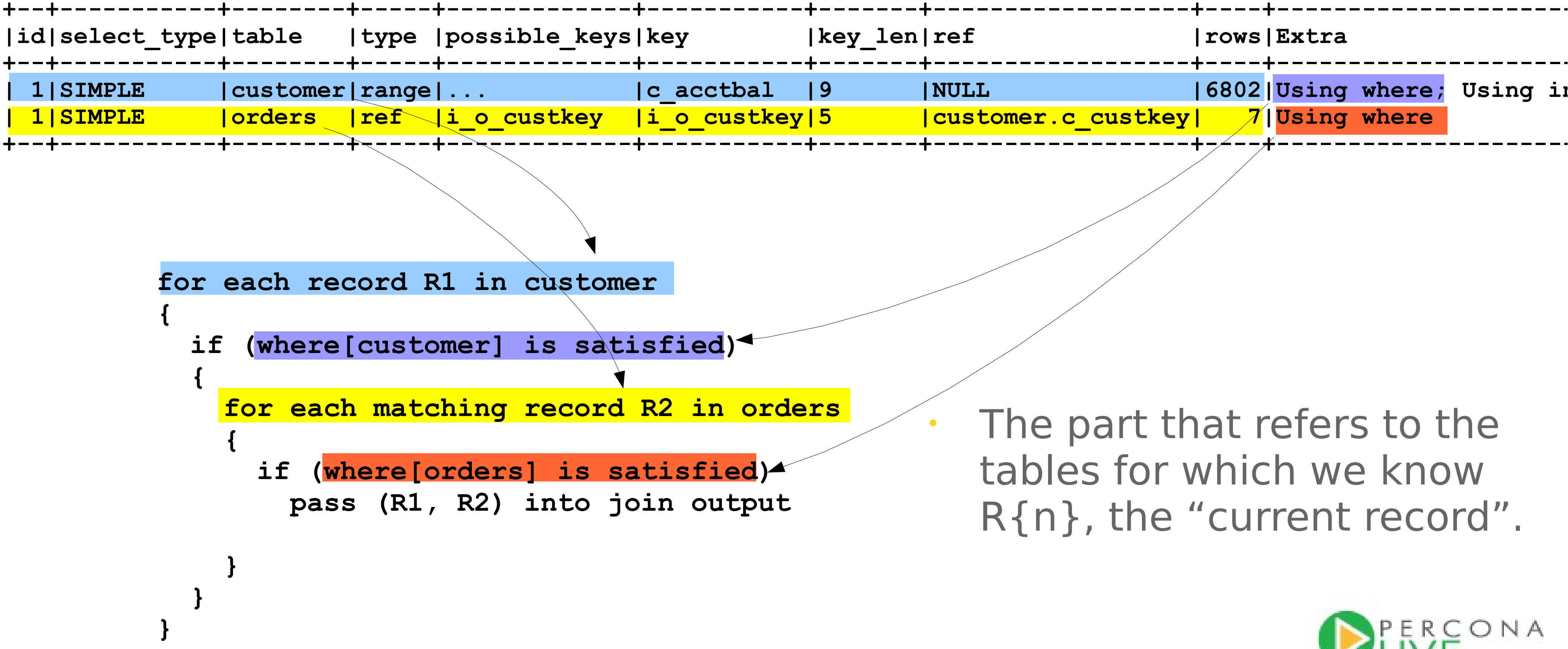
```
+--+-----------+--------+-----+-------------+-----------+-------+-----------------+----+-------------------
|id|select_type|table   |type |possible_keys|key        |key_len|ref              |rows|Extra
+--+-----------+--------+-----+-------------+-----------+-------+-----------------+----+-------------------
| 1|SIMPLE     |customer|range|...          |c_acctbal  |9      |NULL             |6802|Using where; Using i
| 1|SIMPLE     |orders  |ref  |i_o_custkey  |i_o_custkey|5      |customer.c_custkey|   7|Using where
+--+-----------+--------+-----+-------------+-----------+-------+-----------------+----+-------------------
```

```
for each record R1 in customer
{
  if (where[customer] is satisfied)
  {
    for each matching record R2 in orders
    {
      if (where[orders] is satisfied)
        pass (R1, R2) into join output

    }
  }
}
```

- “Using where” means checking a part of WHERE
- Which part?

Nested loops join

```
select count(*)
from
  customer, orders
where
  c_custkey=o_custkey and c_acctbal < -500 and o_orderpriority='1-URGENT';
```



```
+--+-----------+--------+-----+-------------+-----------+-------+------------------+----+---------------------
|id|select_type|table   |type |possible_keys|key        |key_len|ref               |rows|Extra
+--+-----------+--------+-----+-------------+-----------+-------+------------------+----+---------------------
| 1|SIMPLE     |customer|range|...          |c_acctbal  |9      |NULL              |6802|Using where; Using in
| 1|SIMPLE     |orders  |ref  |i_o_custkey  |i_o_custkey|5      |customer.c_custkey|   7|Using where
+--+-----------+--------+-----+-------------+-----------+-------+------------------+----+---------------------
```

```
for each record R1 in customer
{
  if (where[customer] is satisfied)
  {
    for each matching record R2 in orders
    {
      if (where[orders] is satisfied)
        pass (R1, R2) into join output

    }
  }
}
```

- The part that refers to the tables for which we know R{n}, the “current record”.

Nested loops join

```
select count(*)
from
  customer, orders
where
  c_custkey=o_custkey and c_acctbal < -500 and o_orderpriority='1-URGENT';
```

```
+--+-----------+--------+-----+-------------+-----------+-------+------------------+----+--------------------
|id|select_type|table   |type |possible_keys|key        |key_len|ref               |rows|Extra
+--+-----------+--------+-----+-------------+-----------+-------+------------------+----+--------------------
| 1|SIMPLE     |customer|range|...          |c_acctbal  |9      |NULL              |6802|Using where; Using i
| 1|SIMPLE     |orders  |ref  |i_o_custkey  |i_o_custkey|5      |customer.c_custkey|   7|Using where
+--+-----------+--------+-----+-------------+-----------+-------+------------------+----+--------------------
```

```
for each record R1 in customer   // 6802 loops
{
  if (where[customer] is satisfied)
  {
    for each matching record R2 in orders    // 7 loops
    {
      if (where[orders] is satisfied)
        pass (R1, R2) into join output

    }
  }
}
```

- “rows” shows how many rows are read from each table
- For `orders`, 7 rows will be read 6802 times

Nested loops join

```
select count(*)
from
  customer, orders
where
  c_custkey=o_custkey and c_acctbal < -500 and o_orderpriority='1-URGENT';
```

```
+--+-----------+--------+-----+-------------+-----------+-------+------------------+----+-----------------
|id|select_type|table   |type |possible_keys|key        |key_len|ref               |rows|Extra
+--+-----------+--------+-----+-------------+-----------+-------+------------------+----+-----------------
| 1|SIMPLE     |customer|range|...          |c_acctbal  |9      |NULL              |6802|Using where; Using i
| 1|SIMPLE     |orders  |ref  |i_o_custkey  |i_o_custkey|5      |customer.c_custkey|   7|Using where
+--+-----------+--------+-----+-------------+-----------+-------+------------------+----+-----------------
```

```
for each record R1 in customer   // 6802 loops
{
  if (where[customer] is satisfied)
  {
    for each matching record R2 in orders    // 7 loops
    {
      if (where[orders] is satisfied)
        pass (R1, R2) into join output

    }
  }
}
```

- Wait, what about this 'if'?
- It may reject some of the 6802 values of R1

Nested loops join

```
explain extended
select * from  customer, orders
where
  c_custkey=o_custkey and c_acctbal < 1000 and o_orderpriority='1-URGENT';
```

```
--+-----------+--------+----+-------------+-----------+-------+------------------+------+--------+-----------+
id|select_type|table   |type|possible_keys|key        |key_len|ref               |rows  |filtered|Extra      |
--+-----------+--------+----+-------------+-----------+-------+------------------+------+--------+-----------+
 1|SIMPLE     |customer|ALL |...          |NULL       |NULL   |NULL              |132987|   39.44|Using where|
 1|SIMPLE     |orders  |ref |i_o_custkey  |i_o_custkey|5      |customer.c_custkey|     7|  100.00|Using where|
--+-----------+--------+----+-------------+-----------+-------+------------------+------+--------+-----------+
```

```
for each record R1 in customer   // 6802 loops
{
  if (where[customer] is satisfied)
  {
    for each matching record R2 in orders   // 7 loops
    {
      if (where[orders] is satisfied)
        pass (R1, R2) into join output

    }
  }
}
```

- “filtered” column shows the percentage of records that are expected to pass the “if”.
- 100% is the most frequent (optimizer puts 100% when there are no reliable estimates for filtering)

Ref access

```
select count(*)
from
  customer, orders
where
  c_custkey=o_custkey and c_acctbal < -500 and o_orderpriority='1-URGENT';
```

```
+--+-----------+--------+-----+-------------+-----------+-------+------------------+----+-------------------
|id|select_type|table   |type |possible_keys|key        |key_len|ref               |rows|Extra
+--+-----------+--------+-----+-------------+-----------+-------+------------------+----+-------------------
| 1|SIMPLE     |customer|range|...          |c_acctbal  |9      |NULL              |6802|Using where; Using i
| 1|SIMPLE     |orders  |ref  |i_o_custkey  |i_o_custkey|5      |customer.c_custkey|   7|Using where
+--+-----------+--------+-----+-------------+-----------+-------+------------------+----+-------------------
```

- `orders` is accessed with **ref** access: index lookups on index **i_o_custkey** using value of customer.c_custkey
- This is similar to key=const ref access we've discussed earlier
  - But there is no “const”, the value of customer.c_custkey is different for each lookup.
- Each lookup is expected to produce 7 matching rows.
  - This information is from **Index Statistics**. We'll cover it later.

How good a join can/should be

Let's collect some data

```
select count(*)
from
  customer, orders
where
  c_custkey=o_custkey and c_acctbal < -500 and o_orderpriority='1-URGENT';
```

```
+--+-----------+--------+-----+-------------+-----------+-------+------------------+----+------------------
|id|select_type|table   |type |possible_keys|key        |key_len|ref               |rows|Extra
+--+-----------+--------+-----+-------------+-----------+-------+------------------+----+------------------
| 1|SIMPLE     |customer|range|...          |c_acctbal  |9      |NULL              |6802|Using where; Using i
| 1|SIMPLE     |orders  |ref  |i_o_custkey  |i_o_custkey|5      |customer.c_custkey|   7|Using where
+--+-----------+--------+-----+-------------+-----------+-------+------------------+----+------------------
```

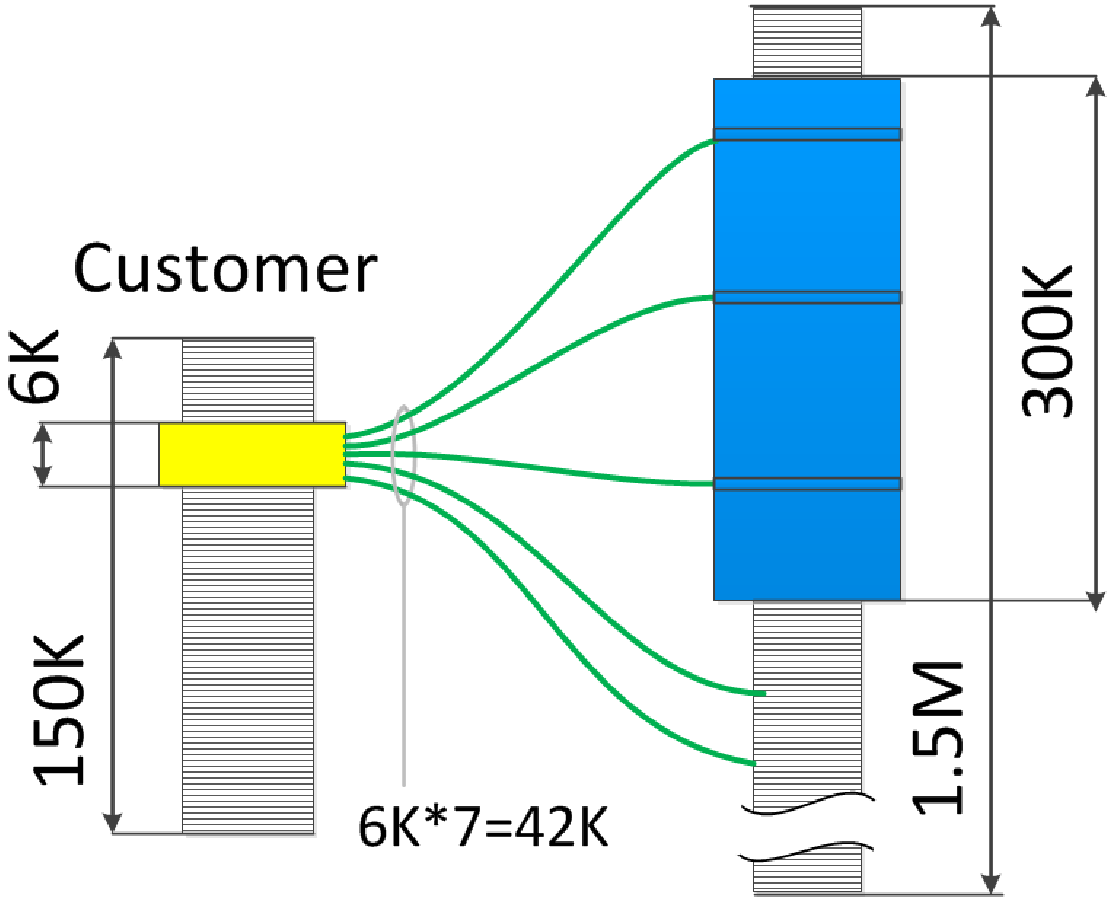
```
[explain] select count(*) from customer;   -- 150K

[explain] select count(*) from customer where c_acctbal < -500  -- 6802

[explain] select count(*) from orders -- 1.5M

[explain] select count(*) from orders where o_orderpriority='1-URGENT' - 300K
```

One customer (c_custkey) – 7 orders

- And use it to analyze the join:

Analyzing the join

```
select count(*) from customer, orders
where
 c_custkey=o_custkey and c_acctbal < 1000 and o_orderpriority='1-URGENT';
```

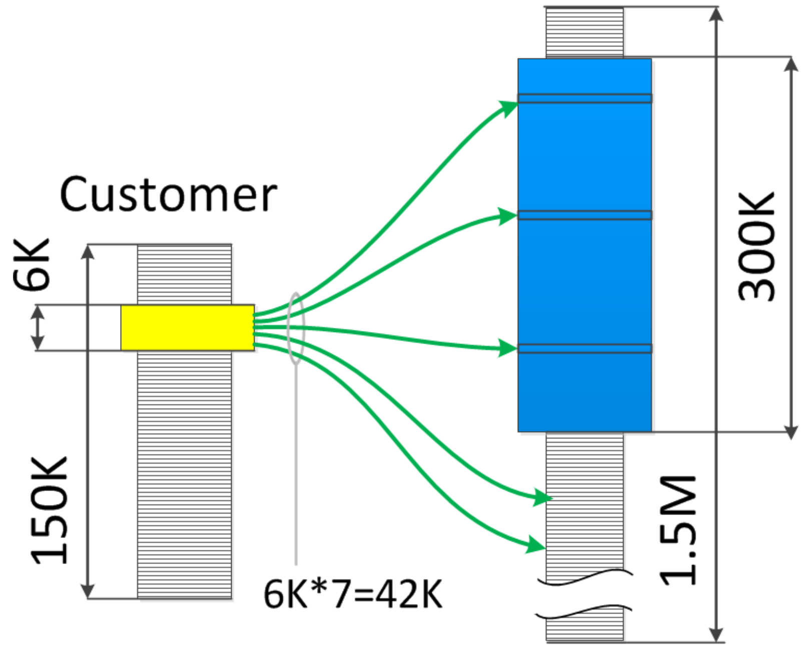


There are three ways to compute the join:

- Yellow → Green
- Blue → Green
- Yellow, Blue

Join plan analysis

```
+--+-----------+--------+-----+-------------+-----------+-------+------------------+----+------------------
|id|select_type|table   |type |possible_keys|key        |key_len|ref               |rows|Extra
+--+-----------+--------+-----+-------------+-----------+-------+------------------+----+------------------
| 1|SIMPLE     |customer|range|...          |c_acctbal  |9      |NULL              |6802|Using where; Using i
| 1|SIMPLE     |orders  |ref  |i_o_custkey  |i_o_custkey|5      |customer.c_custkey|   7|Using where
+--+-----------+--------+-----+-------------+-----------+-------+------------------+----+------------------
```



- We're using the Yellow->Green read
- The yellow part is read efficiently
  - Range access is used, it scans 6K rows
- The green part read efficiently
  - ref access, 6K lookups, each producing 7 rows on average
- This is probably better than the two other variants

Tracking query plan execution

- EXPLAINs are what the optimizer *expects * to happen
- What actually happens?
- MySQL has no EXPLAIN ANALYZE
- Possible solutions
  - Status variables
  - “userstat”
  - performance_schema?

Check status variables

```
MariaDB [dbt3sf1]> show status like 'Handler%';
+----------------------------+-------+
| Variable_name              | Value |
+----------------------------+-------+
| Handler_commit             | 1     |
| Handler_delete             | 0     |
| Handler_discover           | 0     |
| Handler_icp_attempts       | 0     |
| Handler_icp_match          | 0     |
| Handler_mrr_init           | 0     |
| Handler_mrr_key_refills    | 0     |
| Handler_mrr_rowid_refills  | 0     |
| Handler_prepare            | 0     |
| Handler_read_first         | 0     |
| Handler_read_key           | 6805  |
| Handler_read_next          | 75951 |
| Handler_read_prev          | 0     |
| Handler_read_rnd           | 0     |
| Handler_read_rnd_deleted   | 0     |
| Handler_read_rnd_next      | 0     |
| Handler_rollback           | 0     |
| Handler_savepoint          | 0     |
...
```

- Accesses to all tables are summed together!
- It is still possible to do analysis:
  - Run a “sub-join” with the 1st table; note the counters
  - Run a “sub-join” with the 1st and 2nd tables from the join order; note the counters; substract counters from #1
  - ...
- This is slow and painful.

More powerful: userstatv2

- Percona Server and MariaDB:

```
MariaDB [dbt3sf1]> show table_statistics;
+--------------+------------+-----------+--------------+-------------------------+
| Table_schema | Table_name | Rows_read | Rows_changed | Rows_changed_x_#indexes |
+--------------+------------+-----------+--------------+-------------------------+
| dbt3sf1      | customer   |      6805 |            0 |                       0 |
| dbt3sf1      | orders     |     69147 |            0 |                       0 |
+--------------+------------+-----------+--------------+-------------------------+
2 rows in set (0.00 sec)

MariaDB [dbt3sf1]> show index_statistics;
+--------------+------------+-------------+-----------+
| Table_schema | Table_name | Index_name  | Rows_read |
+--------------+------------+-------------+-----------+
| dbt3sf1      | customer   | c_acctbal   |      6805 |
| dbt3sf1      | orders     | i_o_custkey |     69147 |
+--------------+------------+-------------+-----------+
2 rows in set (0.00 sec)
```

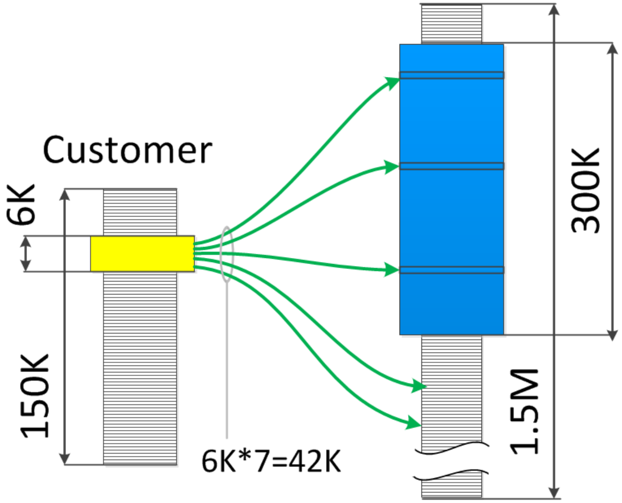
- Much better but self-joins are still painful

Recap

- Ok we now know
  - Single table access methods
    - ref(const)
    - Range
    - index_merge
  - Nested loop join algorithm
  - EXPLAIN
  - How to check if execution follows EXPLAIN

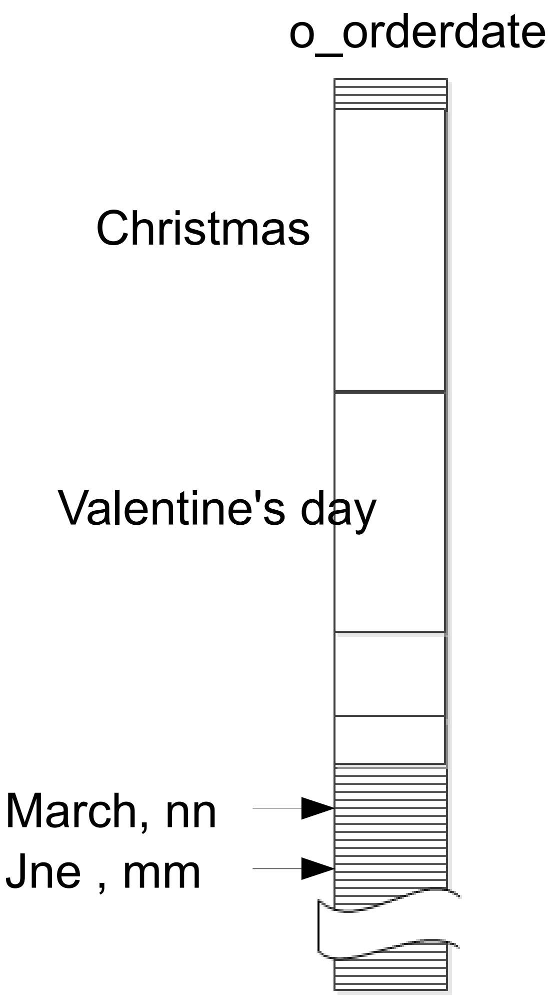
Typical reasons of poor join performance

- Poor access strategy for the 1st table
  - Highly-selective non-indexed conditions? Create an index!
- Joins on non-equality conditions
  - e.g. “same day” joins on DATETIME columns
  - Tip: Convert to equality join at all costs
- No index to do ref access for join
  - Cross-product joins inherently slow
- Correlated data (...)
- Wrong index statistics (...)



Non-uniform distributions

- Some data is highly non-uniform
- Performance depends on hitting the popular values
- No one-for-all solution



Index and Table Statistics

Database statistics and plan stability (InnoDB/XtraDB)

- Sample [8] random index leaf pages
- Table statistics (stored)
  - **rows** - estimated number of rows in a table
  - **clust_size** - cluster index (table/primary key) size in number of pages
  - **other_size** - other index (non primary key) size in number of pages
- Index statistics (stored)
  - **fields** - #fields in the index
  - **rows_per_key** - rows per 1 key value, per prefix fields ([1 column value], [2 columns value], [3 columns value], …)
  - **index_total_pages** – number of index pages
  - **index_leaf_pages** – number of leaf pages

Table statistics updates

- Statistics updated when:
  - ANALYZE TABLE tbl_name [, tbl_name] ...
  - SHOW TABLE STATUS, SHOW INDEX
  - Access to INFORMATION_SCHEMA.[TABLES| STATISTICS]
  - A table is opened for the first time
    (that is, after server restart)
  - A table has changed a lot
    (1/16th of the table updated/deleted/inserted)
  - When InnoDB Monitor is turned ON
  - Others (?)

Displaying statistics

- MySQL 5.5, MariaDB 5.3, and older (GA versions)
    - Issue SQL statements to count rows/keys
    - Indirectly, look at EXPLAIN for simple queries
- MariaDB 5.5, Percona Server 5.5 (using XtraDB) (GA versions)
    - information_schema.[innodb_index_stats, innodb_table_stats]
    - Read-only, always visible
- MySQL 5.6 (development)
    - mysql.[innodb_index_stats, innodb_table_stats]
    - User updatetable
    - Only available if innodb_analyze_is_persistent=ON
- MariaDB $NEXT (development)
    - MyISAM persistent updateable tables
    - + current MySQL and Percona mechanisms

Plan [in]stability

- Statistics may vary a lot (orders)

```
MariaDB [dbt3]> select * from information_schema.innodb_index_stats;
+------------+-----------------+--------------+    +---------------+
| table_name | index_name      | rows_per_key |    | rows_per_key  | error (actual)
+------------+-----------------+--------------+    +---------------+
| partsupp   | PRIMARY         | 3, 1         |    | 4, 1          | 25%
| partsupp   | i_ps_partkey    | 3, 0         | => | 4, 1          | 25%          (4)
| partsupp   | i_ps_suppkey    | 64, 0        |    | 91, 1         | 30%         (80)
| orders     | i_o_orderdate   | 9597, 1      |    | 1660956, 0    | 99%       (6234)
| orders     | i_o_custkey     | 15, 1        |    | 15, 0         |  0%         (15)
| lineitem   | i_l_receiptdate | 7425, 1, 1   |    | 6665850, 1, 1 | 99.9%  (23477)
+------------+-----------------+--------------+    +---------------+

MariaDB [dbt3]> select * from information_schema.innodb_table_stats;
+-----------------+----------+      +----------+
| table_name      | rows     |      | rows     |
+-----------------+----------+      +----------+
| partsupp        |  6524766 |      | 9101065  | 28%     (8000000)
| orders          | 15039855 | ==>  | 14948612 |  0.6% (15000000)
| lineitem        | 60062904 |      | 59992655 |  0.1% (59986052)
+-----------------+----------+      +----------+
```

Controlling statistics (GA versions)

- MySQL 5.5, MariaDB 5.3, and older
  - Manual tuning: optimizer hints, system variables
- MySQL 5.5, MariaDB 5.5 with InnoDB plugin
  - innodb_stats_on_metadata = OFF – update only on restart, ANALYZE
  - innodb_stats_sample_pages = 8 is default - increase precision
  - No way to “freeze” db statistics in all cases
- MariaDB 5.5, Percona Server 5.5 (using XtraDB)
  - Can “freeze” the current InnoDB statistics
  - innodb_use_sys_stats_table=ON – use I_S.INNODB_SYS_STATS
  - innodb_stats_auto_update=OFF – recalculate except for “first open” and “ANALYZE TABLE”
  - No manual control over statistics, only refresh by random sampling

Plan [in]stability

- Same query, different statistics

=>

- Different access methods and/or
- Different JOIN orders

=> different query plans

- Query performance may change a lot when statistics changes
- BEWARE WHEN BENCHMARKING THE OPTIMIZER

Controlling statistics (MySQL dev)

MySQL 5.6 (development version, public code)

- Persistent and user-updatetable InnoDB statistics
  - innodb_analyze_is_persistent = ON,
  - updated only on ANALYZE TABLE
- Control the precision of sampling [default 8]
  - innodb_stats_persistent_sample_pages,
  - innodb_stats_persistent_sample_pages
- No new statistics compared to older versions

Controlling statistics (MariaDB dev)

MariaDB $NEXT (development version, public code)

- Current MySQL and Percona InnoDB statistics
  +
- Engine-independent, persistent, user-updateable statistics
- Precise
- Additional statistics per column (even when there is no index):
  - min_value, max_value: minimum/maximum value per column
  - nulls_ratio: fraction of null values in a column
  - avg_length: average size of values in a column
  - avg_frequency: average number of rows with the same value

=> better query plans

Code: bzr branch lp:~maria-captains/maria/maria-5.5-mwl248

Resources on InnoDB statistics

From the InnoDB/XtraDB developers:

https://mysqlperformanceblog.com/doc/percona-server/5.5/diagnostics/innodb_stats.html

http://dev.mysql.com/doc/refman/5.6/en/innodb-other-changes-statistics-estimation.html

http://dev.mysql.com/doc/refman/5.6/en/innodb-parameters.html

http://dev.mysql.com/doc/refman/5.6/en/innodb-performance.html#innodb-persistent-stats

http://blogs.innodb.com/wp/2011/04/innodb-persistent-statistics-at-last/

http://dev.mysql.com/doc/refman/5.6/en/analyze-table.html

Optimizer differences between MySQL branches

Optimizer in different MySQL branches

- Base version: MySQL 5.5
  - Optimizer is not different from MySQL 5.1
- Percona Server X.Y
  - Closely follows MySQL X.Y
  - Some improvements in diagnostics (eg. userstats)
- MariaDB 5.3
  - Lots of new features
- MariaDB 5.5
  - For the most part, is a merge of Maria-5.3 with MySQL-5.5
  - But has a couple of small improvements
- MySQL 5.6
  - Lots of new features, intersection with MariaDB 5.3

Optimizer feature comparison (1)

Feature	MariaDB 5.3/5.5	MySQL 5.5	MySQL (5.6 dev)
Disk access optimizations			
Index Condition Pushdown (ICP)	**YES**	-	YES
Disk-sweep Multi-range read (DS-MRR)	**YES**	-	YES
DS-MRR with Key-ordered retrieval	**YES**	-	-
Index_merge / Sort_intersection	**YES**	-	-
Cost-based choice of range vs. index_merge	**YES**	-	-
ORDER BY .. LIMIT <small_limit>	**-**	**-**	YES
Use extended (hidden) primary keys for innodb/xtradb	**YES (5.5)**	-	-
Join optimizations			
Batched key access (BKA)	**YES**	-	YES
Block hash join	**YES**	-	-
User-set memory limits on all join buffers	**YES**	-	-
Apply early outer table ON conditions	**YES**	-	-
Null-rejecting conditions tested early for NULLs	**YES**	-	-

Optimizer feature comparison (2)

Feature	MariaDB 5.5	MySQL 5.5	MySQL (5.6 dev)
Subquery optimizations			
In-to-exists	**YES**	**YES**	YES
Semi-join	**YES**	-	-YES
Materialization	**YES**	-	YES
NULL-aware Materialization	**YES**	-	-
Cost choice of materialization vs in-to-exists	**YES**	-	-
Subquery cache	**YES**	-	-
Fast explain with subqueries	**YES**		-
Optimizations for derived tables / views			
Delayed materialization of derived tables / materialized views	**YES**	-	YES
Instant EXPLAIN for derived tables	**YES**	-	YES
Derived Table with Keys optimization	**YES**	-	YES
Fields of merge-able views and derived tables used in equality optimizations	**YES**	-	-

Optimizer feature comparison (3)

Feature	MariaDB 5.3/5.5	MySQL 5.5	MySQL (5.6 dev)
Execution control			
LIMIT ROWS EXAMINED rows_limit	**YES (5.5)**	-	-
Optimizer control (optimizer switch)			
Systematic control of all optimizer strategies	**YES**	-	partial
EXPLAIN improvements			
Explain for DELETE, INSERT, REPLACE, and UPDATE	-	-	YES
EXPLAIN in JSON format	-	-	YES
More detailed and consistent EXPLAIN for subqueries	**YES**	-	-

Optimizer switch (2)

MariaDB 5.5

Index merge:

index_merge=on,
index_merge_union=on,
index_merge_sort_union=on,
index_merge_intersection=on,
index_merge_sort_intersection=off

Condition pushdown:

engine_condition_pushdown=off,
index_condition_pushdown=on,

mrr=off,
mrr_cost_based=off,
mrr_sort_keys=off

MySQL 5.6

Index merge:

index_merge=on,
index_merge_union=on,
index_merge_sort_union=on,
index_merge_intersection=on

Condition pushdown:

engine_condition_pushdown=on,
index_condition_pushdown=on,

mrr=on,
mrr_cost_based=on

Optimizer switch (2)

MariaDB 5.5

Subqueries:
materialization=on,
semijoin=on,
loosescan=on,
firstmatch=on,
in_to_exists=on,
partial_match_rowid_merge=on,
partial_match_table_scan=on,
subquery_cache=on,

Derived tables:
derived_merge=on,
derived_with_keys=on

MySQL 5.6

Subqueries:
materialization=on,
semijoin=on,
loosescan=on,
firstmatch=on

Optimizer switch (3)

MariaDB 5.5

Joins
join_cache_bka=on,
join_cache_incremental=on,
join_cache_hashed=on,
outer_join_with_cache=on,
semijoin_with_cache=on,
optimize_join_buffer_size=off
+ other system variables

Other features
table_elimination=on,
extended_keys=off

MySQL 5.6

Joins:
batched_key_access=off
block_nested_loop=on,

Thank You!

Q & A