Advanced MySQL Query Tuning

Alexander Rubin
July 21, 2013
About Me

My name is Alexander Rubin

• Working with MySQL for over 10 years
  – Started at MySQL AB, then Sun Microsystems,
  – then Oracle (MySQL Consulting)
  – Joined Percona recently

• Helping customers improve MySQL performance
  – performance tuning
  – full text search
  – high availability
  – Reporting, database infrastructure scale-outs
  – Big data
Agenda

- Indexes
  - How B-tree works
  - Range scans

- Queries
  - Temporary Tables and Filesort in MySQL
  - GROUP BY Optimizations
  - ORDER BY Optimizations

- Social Graph Example
How to Deal with Slow Performance

Indexes
MySQL Index Types: B-Tree

Default index type (except for MEMORY tables)

- When you add index (except for MEMORY) MySQL will use B-Tree
- Support equality and “range” operations

http://en.wikipedia.org/wiki/B-tree
MySQL Index Types: B-Tree

Equality search: `select * from table where id = 12`

- Scan thru the tree and go directly to 1 leaf
- Stop

http://en.wikipedia.org/wiki/B-tree
MySQL Index Types: B-Tree

Range: `select * from table where id in (6, 12, 18)`

- Scan thru the tree and visit many leafs/nodes
CREATE TABLE City (  
    ID int(11) NOT NULL AUTO_INCREMENT,  
    Name char(35) NOT NULL DEFAULT '' ,  
    CountryCode char(3) NOT NULL DEFAULT '' ,  
    District char(20) NOT NULL DEFAULT '' ,  
    Population int(11) NOT NULL DEFAULT ' 0 ' ,  
    PRIMARY KEY ( ID ) ,  
    KEY CountryCode ( CountryCode )  
) Engine=InnoDB ;
Indexes: Example

• MySQL will use 1 (best) index

mysql> explain select * from City where ID = 1;
+-------+-------+---------------+---------+---------+-------+------+-------+
| table | type  | possible_keys | key     | key_len | ref   | rows | Extra |
+-------+-------+---------------+---------+---------+-------+------+-------+
| City  | const | PRIMARY       | PRIMARY | 4       | const | 1    |       |
+-------+-------+---------------+---------+---------+-------+------+-------+

mysql> explain select * from City where CountryCode = 'USA';
+-------+-------+---------------+-------------+---------+-------+------+------------+
| table | type  | possible_keys | key         | key_len | ref   | rows | Extra      |
+-------+-------+---------------+-------------+---------+-------+------+------------+
| City  | ref   | CountryCode   | CountryCode | 3       | const | 274  | Using where|
+-------+-------+---------------+-------------+---------+-------+------+------------+
Combined Indexes: Example

- Leftmost part of combined index

```sql
mysql> alter table City add key comb(CountryCode, District, Population), drop key CountryCode;
```
Combined Indexes: Example

- Leftmost part of combined index

```sql
mysql> explain select * from City
    where CountryCode = 'USA'\G
```

```
******************** 1. row ****************
  table: City
  type: ref
  possible_keys: comb
    key: comb
    key_len: 3
    ref: const
  rows: 273
```

Uses first field from the comb key
Combined Indexes: Example

- Key_len = total size (in bytes) of index parts used

Index: comb(CountryCode, District, Population)

Explain:

key: comb
key_len: 3

Fields:

CountryCode char(3)
District char(20)
Population int(11)

3 -> Char(3) -> First field is used
Combined Indexes: Example

- 2 Leftmost Fields

```sql
mysql> explain select * from City
where CountryCode = 'USA' and District = 'California'
 далек
 ********************** 1. row ******************
 table: City
 type: ref
 possible_keys: comb
 key: comb
 key_len: 23
 ref: const,const
 rows: 68

Uses 2 first fields from the comb key
CountryCode = 3 chars
District = 20 chars
Total = 23
```
Combined Indexes: Example

- 3 Leftmost Fields

```
mysql> explain select * from City
where CountryCode = 'USA' and District = 'California'
and population > 10000
G
************************** 1. row **************************
table: City
type: range
possible_keys: comb
key: comb
key_len: 27
ref: NULL
rows: 68
```

Uses **all** fields from the comb key
- CountryCode = 3 chars/bytes
- District = 20 chars/bytes
- Population = 4 bytes (INT)
- Total = 27
Combined Indexes: Example

- Can’t use combined index – not a leftmost part

```sql
mysql> explain select * from City where District = 'California' and population > 10000\G
*************************** 1. row ***************************
table: City
  type: ALL
possible_keys: NULL
  key: NULL
key_len: NULL
  ref: NULL
rows: 3868
```

Does not have the `CountryCode` in the where clause = *can’t use comb index*
Covered Index: Example

- Covered index = cover all fields in query

```
select name from City where CountryCode = 'USA'
and District = 'Alaska' and population > 10000
```

```
mysql> alter table City add key cov1(CountryCode, District, population, name);
```

Uses **all** fields in the query in particular order:
1. Where part
2. Group By/Order (not used now)
3. Select part (here: **name**)

www.percona.com
Covered Index: Example

- Explain

mysql> explain select name from City where CountryCode = 'USA' and District = 'Alaska' and population > 10000

*************************** 1. row ***********

table: City
  type: range
possible_keys: cov1
  key: cov1
key_len: 27
  ref: NULL
rows: 1
Extra: Using where; Using index

Using index = covered index is used
MySQL will only use index
Will not go to the data file
Order of Fields in Index

Range and “const” scans: use “effective” cardinality

- select * from City where district = 'California' and population > 30000
- Index (district, population) in this order
- Rule of thumb: “Const” first, “Range” second
  - Depends on query
Order of Fields in Index: Example

```sql
mysql> alter table City add key comb1(district, population);
```

```sql
mysql> explain select name from City where district = 'California' and population > 10000\G
```

************** 1. row ****************

```
table: City
type: range
possible_keys: comb1
  key: comb1
  key_len: 24
  ref: NULL
  rows: 68
```

Good: Index is used to restrict rows
Key_len = 24 – both fields used
Order of Fields in Index: Example

mysql> alter table City add key comb2(population, district);

mysql> explain select name from City where district = 'California' and population > 3000\G

<table>
<thead>
<tr>
<th>table: City</th>
</tr>
</thead>
<tbody>
<tr>
<td>type: ALL</td>
</tr>
<tr>
<td>possible_keys: comb2</td>
</tr>
<tr>
<td>key: NULL</td>
</tr>
<tr>
<td>key_len: NULL</td>
</tr>
<tr>
<td>ref: NULL</td>
</tr>
<tr>
<td>rows: 4162</td>
</tr>
<tr>
<td>Extra: Using where</td>
</tr>
</tbody>
</table>

BAD! MySQL decided not to use index at all
Why?
MySQL can only use “population” part
Too many cities with population > 3000
**Simplified BTree Scan Example I**

\[ \text{Comb1}(\text{district}, \text{population}) \]

1. Go “directly”* to the district (CA)
2. Do range scan by population starting with “CA”

*via index scan
**Simplified BTree Scan Example II**

\[ \text{Combl}(\text{population}, \text{district}) \]

1. Do range scan by population
2. For each scanned index record
   Check for correct district (CA)
3. = Only use “population” part of the index
```
mysql> alter table City
add key comb2(population, District);

explain select name from City where District = 'California' and population > 1000000\G

***************************** 1. row *****************************

table: City
  type: range
possible_keys: comb2
  key: comb2
  key_len: 4
  ref: NULL
rows: 237
extra: Using where
```

**Uses Index**

**BUT:**
key_len = 4 (INT)
Only population part is used
How to Deal with Slow Performance

Queries
Complex Slow Queries

... Group By ...
... Order By ...
Select distinct ...

Temporary tables

Filesort
GROUP BY Queries
GROUP BY and Temporary Tables

How many cities in each country?

```sql
mysql> explain select CountryCode, count(*) from City group by CountryCode
```
```
id: 1
select_type: SIMPLE
table: City
type: ALL
possible_keys: NULL
key: NULL
key_len: NULL
ref: NULL
rows: 4079
Extra: Using temporary; Using filesort
```

Temporary tables are slow!
Temporary Tables: Theory
Temporary Tables, I

Main performance issues

- MySQL can create temporary tables when query uses:
  - GROUP BY
  - Range + ORDER BY
  - Some other expressions

- 2 types of temporary tables
  - MEMORY
  - On-disk
Temporary Tables, II

• First, MySQL tries to create temporary table in memory
• MySQL configuration variables:
  • `tmp_table_size`
    • maximum size for in Memory temporary tables
  • `max_heap_table_size`
    • Sets the maximum size for MEMORY tables
Temporary Tables III

MySQL temp table > tmp_table_size

OR

MySQL temp table > max_heap_table_size

= convert to MyISAM temporary table on disk
Temporary Tables

• MEMORY engine does not support BLOB/TEXT
  • select blob_field from table group by field1
  • select concat(...string>512 chars) group by field1
• Create on-disk temporary table right away

• Percona server uses the new MEMORY engine with BLOB/TEXT Support
• BUT: it is not used for the temp tables
Temporary Tables: Practice
6M rows, ~2G in size

CREATE TABLE ontime_2012 (  
YearD int(11) DEFAULT NULL,  
MonthD tinyint(4) DEFAULT NULL,  
DayofMonth tinyint(4) DEFAULT NULL,  
DayOfWeek tinyint(4) DEFAULT NULL,  
Carrier char(2) DEFAULT NULL,  
Origin char(5) DEFAULT NULL,  
DepDelayMinutes int(11) DEFAULT NULL,  
...  
) ENGINE=InnoDB DEFAULT CHARSET=latin1
http://www.transtats.bts.gov/DL_SelectFields.asp?Table_ID=236&DB_Short_Name=On-Time
GROUP BY Query Example

- Find maximum delay for flights on Sunday
- Group by airline

```sql
SELECT max(DepDelayMinutes), carrier, dayofweek
FROM ontime_2012
WHERE dayofweek = 7
GROUP BY Carrier
```
GROUP BY Query Example

```sql
select max(DepDelayMinutes), carrier, dayofweek
from ontime_2012
where dayofweek = 7
group by Carrier

   type: ALL
possible_keys: NULL
    key: NULL
key_len: NULL
  ref: NULL
rows: 4833086
Extra: Using where; Using temporary; Using filesort
```
Adding Index: Fixing Full Table Scan

```
mysql> alter table ontime_2012 add key (dayofweek);
```

```
explain select max(DepDelayMinutes), Carrier, dayofweek from ontime_2012 where dayofweek = 7
group by Carrier;
```

```
   type: ref
possible_keys: DayOfWeek
   key: DayOfWeek
key_len: 2
   ref: const
rows: 817258
Extra: Using where; Using temporary; Using filesort
```

Index is used = better
BUT: Large temporary table!
GROUP BY: Adding Covered Index

mysql> alter table ontime_2012
add key covered(dayofweek, Carrier, DepDelayMinutes);

explain select max(DepDelayMinutes), Carrier, dayofweek from ontime_2012 where dayofweek =7 group by Carrier\G

... possible_keys: DayOfWeek,covered
   key: covered
   key_len: 2
   ref: const
   rows: 905138
   Extra: Using where; Using index

• Called “tight index scan”
When Covered Indexes Aren’t Good …

mysql> explain select max(DepDelayMinutes), Carrier, dayofweek from ontime_2012
where dayofweek > 3 group by Carrier, dayofweek\G
...

<table>
<thead>
<tr>
<th>type:</th>
<th>range</th>
</tr>
</thead>
<tbody>
<tr>
<td>possible_keys:</td>
<td>covered</td>
</tr>
<tr>
<td>key:</td>
<td>covered</td>
</tr>
<tr>
<td>key_len:</td>
<td>2</td>
</tr>
<tr>
<td>ref:</td>
<td>NULL</td>
</tr>
<tr>
<td>rows:</td>
<td>2441781</td>
</tr>
<tr>
<td>Extra:</td>
<td>Using where; Using index; Using temporary; Using filesort</td>
</tr>
</tbody>
</table>
Converting query to Union

```sql
(SELECT max(DepDelayMinutes), Carrier, dayofweek
FROM ontime_2012
WHERE dayofweek = 3
GROUP BY Carrier, dayofweek) 
UNION
(SELECT max(DepDelayMinutes), Carrier, dayofweek
FROM ontime_2012
WHERE dayofweek = 4
GROUP BY Carrier, dayofweek)
```
### 1. row

<table>
<thead>
<tr>
<th>Table:</th>
<th>ontime_2012</th>
</tr>
</thead>
<tbody>
<tr>
<td>Key:</td>
<td>covered</td>
</tr>
</tbody>
</table>

Extra: Using where; Using index

### 2. row

<table>
<thead>
<tr>
<th>Table:</th>
<th>ontime_2012</th>
</tr>
</thead>
<tbody>
<tr>
<td>Key:</td>
<td>covered</td>
</tr>
</tbody>
</table>

Extra: Using where; Using index

### 3. row

<table>
<thead>
<tr>
<th>Id:</th>
<th>NULL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Select type:</td>
<td>UNION RESULT</td>
</tr>
<tr>
<td>Table:</td>
<td>&lt;union1,2&gt;</td>
</tr>
<tr>
<td>Type:</td>
<td>ALL</td>
</tr>
<tr>
<td>Possible keys:</td>
<td>NULL</td>
</tr>
<tr>
<td>Key:</td>
<td>NULL</td>
</tr>
<tr>
<td>Key_len:</td>
<td>NULL</td>
</tr>
<tr>
<td>Ref:</td>
<td>NULL</td>
</tr>
<tr>
<td>Rows:</td>
<td>NULL</td>
</tr>
</tbody>
</table>

Extra: Using temporary
GROUP BY: Loose index scan

- Loose index scan:
  - considers only a *fraction of the* keys in an index
- Following rules apply:
  - The query is over a single table.
  - The GROUP BY names only columns that form a leftmost prefix of the index and no other columns.
  - The only aggregate functions used in the select list (if any) are MIN() and MAX(), same column
Loose index scan example

mysql> alter table ontime_2012 add key loose_index_scan
    (Carrier, dayofweek, DepDelayMinutes);

mysql> explain select max(DepDelayMinutes), Carrier, dayofweek
    from ontime_2012 where dayofweek > 5 group by Carrier, dayofweek\G

...
GROUP BY: Tight index scan

- Full index scan or a range index scan
- Can work if loose index scan can’t be used
- Allow to scan index and **AVOID creating tmp table**
- Find ALL keys => group
- = Covered index
Loose index scan vs. tight index scan
Table: ontime_2012, 6M rows, data: 2G, index: 210M

CREATE TABLE ontime_2012 (  
    YearD int(11) DEFAULT NULL,  
    MonthD tinyint(4) DEFAULT NULL,  
    DayOfMonth tinyint(4) DEFAULT NULL,  
    DayOfWeek tinyint(4) DEFAULT NULL,  
    Carrier char(2) DEFAULT NULL,  
    Origin char(5) DEFAULT NULL,  
    DepDelayMinutes int(11) DEFAULT NULL,  
    ...  
   KEY loose_index_scan (Carrier,DayOfWeek,DepDelayMinutes),  
   KEY covered (DayOfWeek,Carrier,DepDelayMinutes)  
) ENGINE=InnoDB DEFAULT CHARSET=latin1
select max(DepDelayMinutes) as ddm, Carrier, dayofweek from ontime_2012
where dayofweek = 5  group by Carrier, dayofweek

```
table: ontime_2012
type: range
possible_keys: covered
  key: loose_index_scan
key_len: 5
  ref: NULL
rows: 201
Extra: Using where; Using index for group-by
```

```
Carrier, DayOfWeek, DepDelayMinutes
1606 | AA | 7 |
...
30 rows in set (0.00 sec)
```
Loose index scan vs. tight index scan

Loose index scan

```sql
select max(DepDelayMinutes) as ddm, Carrier, dayofweek from ontime_2012
where dayofweek > 5 group by Carrier, dayofweek;
```

table: ontime_2012
  type: range
possible_keys: covered
  key: loose_index_scan
key_len: 5
ref: NULL
rows: 213
Extra: Using where; Using index for group-by;

mysql> select ...  
+-----+---------+-----------+
| ddm | Carrier | dayofweek |
|-----+---------+-----------|
| 1606 | AA      | 7         |
+-----+---------+-----------+
30 rows in set (0.00 sec)
Loose index scan vs. tight index scan

Loose index scan

```sql
select max(DepDelayMinutes) as ddm, Carrier, dayofweek from ontime_2012
where dayofweek > 5 group by Carrier, dayofweek order by ddm desc;
```

<table>
<thead>
<tr>
<th>table: ontime_2012</th>
</tr>
</thead>
<tbody>
<tr>
<td>type: range</td>
</tr>
<tr>
<td>possible_keys: covered</td>
</tr>
<tr>
<td>key: loose_index_scan</td>
</tr>
<tr>
<td>key_len: 5</td>
</tr>
<tr>
<td>ref: NULL</td>
</tr>
<tr>
<td>rows: 213</td>
</tr>
<tr>
<td>Extra: Using where; Using index for group-by; Using temporary; Using filesort</td>
</tr>
</tbody>
</table>

```
mysql> select ...
+----------------+---------+-----------+
| ddm  | Carrier | dayofweek |
+----------------+---------+-----------+
| 1606 | AA      |         7 |
```

30 rows in set (0.00 sec)
Loose index scan vs. tight index scan

Temporary table

```sql
select max(DepDelayMinutes) as ddm, Carrier, dayofweek from ontime_2012
ignore index(loose_index_scan) where dayofweek > 5
group by Carrier, dayofweek
```

table: ontime_2012
  type: range
  key: covered
  key_len: 2
  ref: NULL
  rows: 2302412
Extra: Using where; Using index;
  Using temporary; Using filesort

1. Range scan
2. Temp table and filesort of 2M rows

```
mysql> select ...
+-----------------------------+
| ddm  | Carrier | dayofweek |
+-----------------------------+
| 1606 | AA      | 7         |
... 30 rows in set (1.30 sec)
```
Loose index scan vs. tight index scan

Tight Index Scan

```sql
select max(DepDelayMinutes) as ddm, Carrier, dayofweek from ontime_2012
ignore index(loose_index_scan) where dayofweek = 5
```

group by Carrier, dayofweek

<table>
<thead>
<tr>
<th>table: ontime_2012</th>
</tr>
</thead>
<tbody>
<tr>
<td>type: ref</td>
</tr>
<tr>
<td>key: covered</td>
</tr>
<tr>
<td>key_len: 2</td>
</tr>
<tr>
<td>ref: NULL</td>
</tr>
<tr>
<td>rows: 2302412</td>
</tr>
</tbody>
</table>

Extra: Using where; Using index;

```
mysql> select ...
+-------------------------------+
| ddm  | Carrier | dayofweek |
+-------------------------------+
| 1606 | AA      | 7         |
... 30 rows in set (0.37 sec)
```

DayOfWeek, Carrier, DepDelayMinutes

1. Covered index
2. No temp table
3. BUT: have to scan 2M rows
Loose index scan vs. tight index scan

Results

Loose index scan
Tight Index Scan
Temporary table
Where loose index scan is not supported

- AVG() + Group By – loose index scan is not supported

```
mysql> explain select avg(DepDelayMinutes) as ddm, Carrier, dayofweek
   from ontime_2012 where dayofweek >5  group by Carrier, dayofweek \G

+--------+---------+-----------+
| ddm    | Carrier | dayofweek |
+--------+---------+-----------+
| 10.8564| AA      | 6         |
+--------+---------+-----------+
```

1. No loose index scan
2. Filter by key
3. Group by filesort

```
mysql> select ...
+--------+---------+-----------+
| ddm    | Carrier | dayofweek |
+--------+---------+-----------+
| 10.8564| AA      | 6         |
+--------+---------+-----------+
```

30 rows in set (1.39 sec)
ORDER BY and filesort
ORDER BY and filesort

Find 10 cities in the US with the largest population

mysql> explain select district, name, population from City where CountryCode = 'USA' order by population desc limit 10\G

<table>
<thead>
<tr>
<th>table</th>
<th>City</th>
</tr>
</thead>
<tbody>
<tr>
<td>type</td>
<td>ALL</td>
</tr>
<tr>
<td>possible_keys</td>
<td>NULL</td>
</tr>
<tr>
<td>key</td>
<td>NULL</td>
</tr>
<tr>
<td>key_len</td>
<td>NULL</td>
</tr>
<tr>
<td>ref</td>
<td>NULL</td>
</tr>
<tr>
<td>rows</td>
<td>4079</td>
</tr>
<tr>
<td>Extra</td>
<td>Using where; Using filesort</td>
</tr>
</tbody>
</table>
Fixing Filesort: Adding Index

```sql
mysql> alter table City
    add key my_sort2 (CountryCode, population);

mysql> explain select district, name, population from City
where CountryCode = 'USA' order by population desc limit 10\G
```

```
table: City
  type: ref
    key: my_sort2
key_len: 3
    ref: const
rows: 207
Extra: Using where
```
Sorting and Limit

mysql> alter table ontime_2012 add key (DepDelayMinutes);
Query OK, 0 rows affected (38.68 sec)

mysql> explain select * from ontime_2012
where dayofweek in (6,7) order by DepDelayMinutes desc
limit 10\G

  type: index
possible_keys: DayOfWeek,covered
  key: DepDelayMinutes
  key_len: 5
  ref: NULL
  rows: 24
  Extra: Using where

10 rows in set (0.00 sec)

1. Index is sorted
2. Scan the **whole table** in the order of the index
3. Filter results
4. Stop after finding 10 rows matching the "where" condition
If Index points to the beginning of the table (physically) = fast
As it stops after 10 rows (LIMIT 10)
If Index points to the end of table (physically) or random = slower
Much more rows to scan (and skip)
Example: Social Graph
Social Graph Example: FlockDB

- Source_id = <YOU>
- Destination_id = IDs of people following <YOU>
- Position = date (unix_timestamp) when <Destination_id> added <YOU>
- State = \{0, 1, 2, 3\} 0 = normal, 2 = removed, 3 = archived (Example)

https://blog.twitter.com/2010/introducing-flockdb
CREATE TABLE social_graph (  
source_id bigint(20) unsigned NOT NULL,  
destination_id bigint(20) unsigned NOT NULL,  
position bigint(20) unsigned NOT NULL,  
state tinyint(4) NOT NULL,  
...
PRIMAY KEY (source_id,destination_id)  
) ENGINE=InnoDB DEFAULT CHARSET=utf8
Social Graph Example: Query

```sql
mysql> explain select * from social_graph where source_id = 927 and state in (0) order by position desc limit 100\G

select_type: SIMPLE
  table: social_graph
  type: ref
possible_keys: PRIMARY
  key: PRIMARY
key_len: 8
    ref: const
    rows: 957874
Extra: Using where; Using filesort
1 row in set (0.00 sec)

Time:
100 rows in set (0.20 sec)
```

Celebrity!
Almost 1M followers! 😊
Social Graph Example: Query

```sql
mysql> alter table social_graph add key src_state_position(source_id, state, position);

mysql> select * from social_graph
where source_id = 927 and state in (0)
order by position desc
limit 100;

...  

100 rows in set (0.00 sec)
```
mysql> explain select * from social_graph where source_id = 927 and state in (0) order by position desc limit 100\G

table: social_graph
  type: ref
possible_keys: PRIMARY,src_state_position
    key: src_state_position
    key_len: 9
      ref: const,const
    rows: 957874
Extra: Using where; Using index
mysql> explain select * from where source_id = 927
and state in (0,2) order by position desc limit 100\G

select_type: SIMPLE
table: social_graph
type: ref
possible_keys: PRIMARY,src_state_position
    key: src_state_position
key_len: 8
    ref: const
rows: 923251
Extra: Using where; Using filesort

100 rows in set (0.54 sec)
Social Graph Example: New Index

```sql
mysql> alter table social_graph
   add key src_position(source_id, position);

mysql> explain select * from social_graph where source_id = 927 and state in (0,2) order by position desc limit 100\G

select_type: SIMPLE
    table: social_graph
    type: range
possible_keys: PRIMARY,src_state_position,src_position
    key: src_position
    key_len: 8
    ref: NULL
    rows: 923922
    Extra: Using where

100 rows in set (0.00 sec)
```

Scan by source_id, correct order
Filter out “wrong” state(s)
Stop after finding 100 rows
mysql> explain select * from social_graph where source_id = 927 and state in (1,2) order by position desc limit 100\G

<table>
<thead>
<tr>
<th>select_type:</th>
<th>SIMPLE</th>
</tr>
</thead>
<tbody>
<tr>
<td>table:</td>
<td>social_graph</td>
</tr>
<tr>
<td>type:</td>
<td>range</td>
</tr>
</tbody>
</table>

Use proper index
Scan 28 rows is faster even with filesort

| possible_keys: | PRIMARY,src_state_position,src_position |
|               | key: src_state_position |
|               | key_len: 9 |
|               | ref: NULL |
|               | rows: 28 |
| Extra:        | Using where; Using filesort |

100 rows in set (0.00 sec)
Social Graph Example: New Index

mysql> explain select * from social_graph
ignore index (src_state_position)
where source_id = 927 and state in (1,2) order by position desc limit 100\G

select_type: SIMPLE
  table: social_graph
  type: ref
possible_keys: PRIMARY,src_position
  key: src_position
  key_len: 8
    ref: const
    rows: 923922
Extra: Using where

27 rows in set (0.54 sec)

+-------+----------+
| state | count(*) |
+-------+----------+
|     0 |   456156 |
|     1 |       27 |
|     2 |        1 |
+-------+----------+

- Will stop only after “discarding” 400K rows.
- 27 rows returned
- filtering + filesort is faster in this case
Questions?

Thank you!