Tired of MySQL Making You Wait?

Alexander Rubin, *Principal Consultant, Percona*
Janis Griffin, *Database Evangelist, SolarWinds*

Who Am I?

- Senior DBA / Performance Evangelist for Solarwinds
  - Janis.Griffin@solarwinds.com 
  - Twitter - @DoBoutAnything
  - Current – 25+ Years in Oracle – and now MySQL ☺
  - DBA and Developer
- Specialize in Performance Tuning
- Review Database Performance for Customers and Prospects
- Common Question – How do I tune it?

Agenda

- Challenges of Tuning
- Wait time analytics using Performance / Information schemas
- Monitoring for performance utilizing DPA
- Explaining plan operations focusing on temporary tables and filesort
- Using indexes to optimize your queries
- Utilizing loose and tight index scans in MySQL

Challenges of Tuning

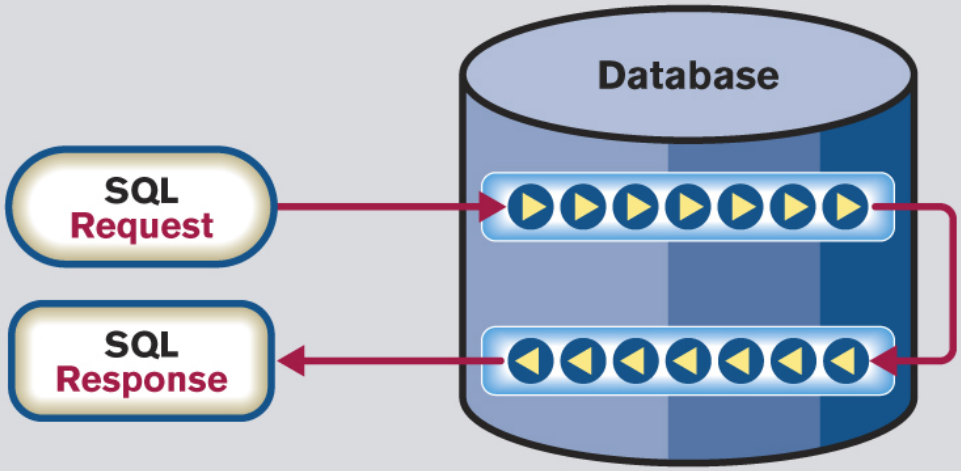
- SQL Tuning is Hard
- Requires Expertise in Many Areas
  - Technical – Plan, Data Access, SQL Design
  - Business – What is the Purpose of SQL?
- Tuning Takes Time
  - Large Number of SQL Statements
  - Each Statement is Different
- Low Priority in Some Companies
  - Vendor Applications
  - Focus on Hardware or System Issues
- Never Ending

Which SQL to Tune

- User / Batch Job Complaints
  - Known Poorly Performing SQL
- Queries Performing Most I/O
  - Rows Examined vs. Rows Affected or Sent
  - Table or Index Scans
- Queries Consuming CPU
- Highest Wait Times (DPA)
  - Performance Schema
  - Information Schema

Response Time Analysis (RTA)

Focus on Response Time



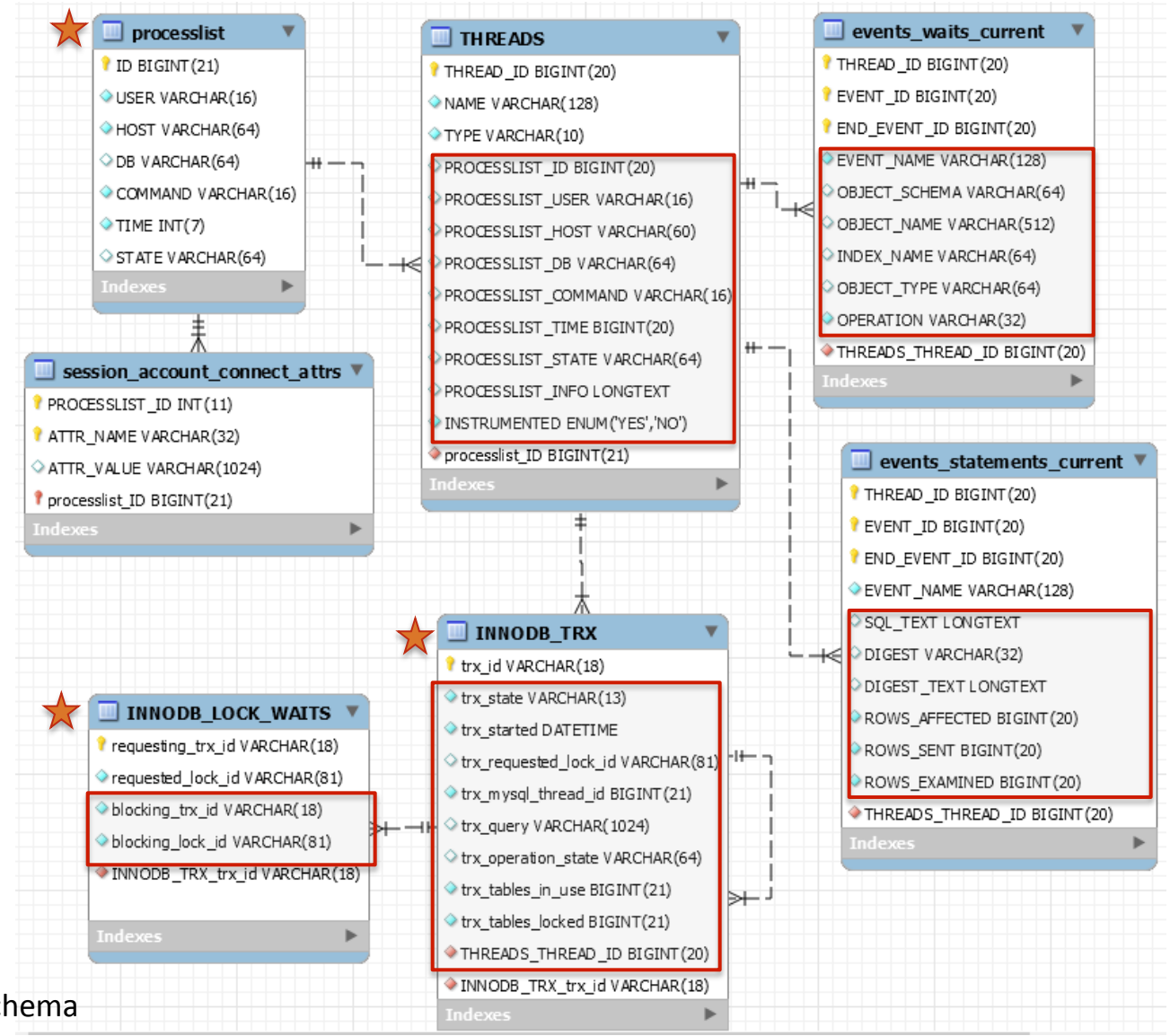
Identify Wait-Time at every step and rank bottlenecks by user impact.

- Understand the total time a Query spends in Database
- Measure time while Query executes
- MySQL helps by providing Instrumented Waits or Thread States

MySQL 5.6+

- Greatly improved Performance_Schema
  - More information / more instrumentation
- Current/Historical Events at all levels
  - Statements
  - Stages
  - Waits
- Performance_Schema consumers now ON by Default
- Storage Engine now defaults to INNODB
- 5.7 Performance_Schema
  - 35 new tables – 5.7 has 87, 5.6 had 52, 5.5 had 17
  - Reduced overhead & footprint
  - 1005 Instruments (5.7.10)
    - New ones for transactions, metadata locks, memory usage
    - and more…

MYSQL – Response Time Data



Information_Schema

PERCONA

Statements, Stages & Waits

SELECT visibility FROM linkdb . linktable WHERE id1 = ? AND id2 = ? AND link_type = ? FOR UPDATE

EVENT TYPE	EV ID	END ID	EVENT NAME	Event Time	Stmt Time	Stage Time	Wait Time
Statement	699	726	statement/sql/select	365	365	354	110

EVENT TYPE	EV ID	END ID	EVENT NAME	Event Time	Stmt Time	Stage Time	Wait Time	SQL / Stage-State / Wait C
Stage	700	703	stage/sql/init	86	23.6%			init
Wait	701	701	wait/io/socket/sql/client_connection	8	2.1%			recv
Wait	702	702	wait/synch/mutex/sql/THD::LOCK_thd_data	5	1.4%			lock
Wait	703	703	wait/synch/mutex/sql/THD::LOCK_thd_data	4	1.1%			lock
Stage	704	704	stage/sql/checking permissions	8	2.1%			checking permissions
Stage	705	706	stage/sql/Opening tables	34	9.4%			Opening tables
Wait	706	706	wait/synch/mutex/sql/THD::LOCK_thd_data	3	0.8%			lock
Stage	707	707	stage/sql/init	35	9.6%			init
Stage	708	710	stage/sql/System lock	20	5.6%			System lock
Wait	709	709	wait/lock/table/sql/handler	4	1.2%			write external
Wait	710	710	wait/lock/table/sql/handler	3	0.8%			write allow write
Stage	711	711	stage/sql/optimizing	11	3.1%			optimizing
Stage	712	715	stage/sql/statistics	63	17.2%			statistics
Wait	713	715	wait/io/table/sql/handler	41	11.3%			fetch
Stage	716	716	stage/sql/preparing	7	2.0%			preparing
Stage	717	717	stage/sql/executing	3	0.8%			executing
Stage	718	718	stage/sql/Sending data	10	2.7%			Sending data
Stage	719	719	stage/sql/end	4	1.0%			end
Stage	720	720	stage/sql/query end	4	1.0%			query end
Stage	721	722	stage/sql/closing tables	11	3.1%			closing tables
Wait	722	722	wait/synch/mutex/sql/THD::LOCK_thd_data	2	0.7%			lock
Stage	723	724	stage/sql/freeing items	50	13.6%			freeing items
Wait	724	724	wait/io/socket/sql/client_connection	37	10.3%			send
Stage	725	726	stage/sql/cleaning up	8	2.2%			cleaning up
Wait	726	726	wait/synch/mutex/sql/THD::LOCK_thd_data	3	0.7%			lock

Thread State Example

Sending data

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

Resolved by

DBAs, storage admins, network admins

Solutions

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

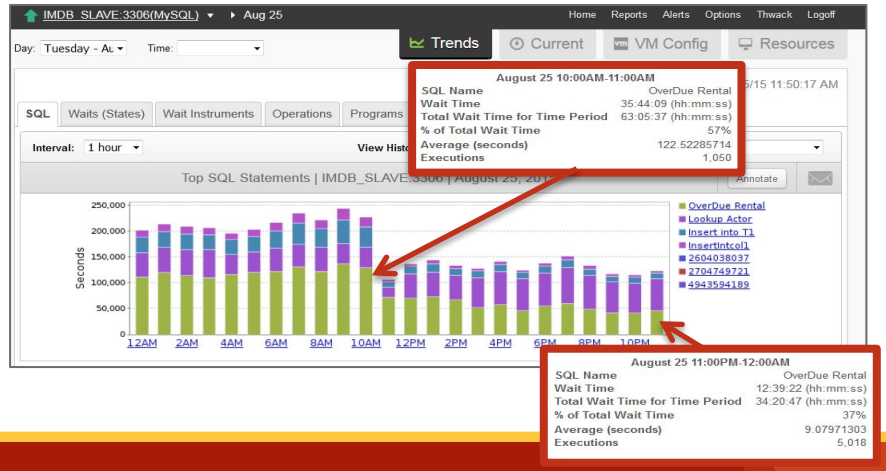
- Optimize SQL to reduce Sending Data wait time.
    1. On the Trends page for the instance, click the **Waits** tab.
    2. In the **Top Waits** graph, drill down to a day and time that have significant amounts of time spent in the **sending data** wait state.
    3. Click **sending data** on the graph to see the SQL statements most affected by this wait state.
    4. Click each SQL and compare **Rows Examined** to **Rows Affected or Sent** statistics in the **SQL Data** tab.
    5. If **Rows Examined** is more than ten times larger than **Rows Affected or Sent**, consider the following:
        - Use summary tables where possible to limit the number of rows processed.
        - Rewrite complicated queries to assist in processing fewer rows.
        - Evaluate WHERE clauses to ensure you process only rows that are required.
- If the number of rows retrieved hasn't changed but the time spent in Sending Data has increased, this could indicate slow I/O performance.
- This can indicate slow network speeds between the MySQL server and the application. This is prevalent in cases where the SQL returns a lot of rows back to the application.

5.7.10 Sys Schema

- SYS schema now provided by default
  - Built on performance_schema & information_schema
  - Contains many formatting procedures & functions
  - Easier to use than querying performance & information schemas
- 100 views
  - X$ views contain raw data
    - Other views convert from picoseconds to milliseconds, etc…
  - User / Host Summary Views
    - IO usage, Stages, Statement details
  - IO / Innodb / Memory Summary
    - By file, thread / globally / host / user
  - Schema Analysis – table / index statistics
  - Wait Analysis by user / host / globally
  - Statement Level Views shows sorting / full tables scans / errors

DPA – Quick 5 Minute Demo

- Quickly identify root cause that impact end-user response time
  - See historical trends over days, months, and years
  - Understand impact of VMware® performance
  - Agentless architecture, installs in minutes
  - Try Database Performance Analyzer FREE for 14 days



About Me

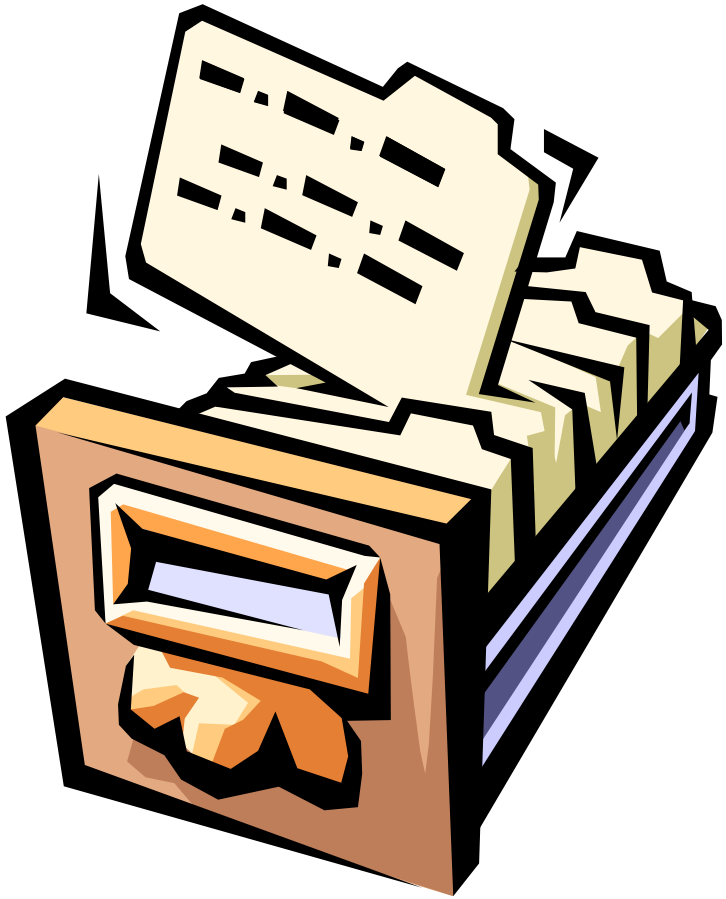
- My name is Alexander Rubin
- Working with MySQL for over 10 years
  - Started at MySQL AB, then Sun Microsystems,
  - Then Oracle (MySQL Consulting)
  - Joined Percona 2 years ago
  - https://www.linkedin.com/in/alexanderrubin

Agenda

- Queries
  - Temporary Tables and Filesort
  - GROUP BY Optimizations
  - ORDER BY Optimizations
  - “Calculated” fields – MySQL 5.7

Query Tuning: basics

Indexes and Explain examples



Example Table

```
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

```
mysql> alter table City add key
comb(CountryCode, District, Population);
```

Combined Indexes: Example

- Leftmost part of combined index

```
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

```
mysql> explain select * from City
where CountryCode = 'USA' and District = 'California'\G
*************************** 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: JSON

```
mysql> explain format=JSON select * from City  where CountryCode = 'USA' and District =
   'California'\G
*************************** 1. row ***************************
EXPLAIN: {
  "query_block": {
    "select_id": 1,
    "table": {
      "table_name": "City",
      "access_type": "ref",
      "possible_keys": [
        "comb"
      ],
        "key": "comb",
         "used_key_parts": [
            "CountryCode",
            "District"
         ],
      "key_length": "23",
      "ref": [
        "const",
        "const"
      ],
      "rows": 84,
    ...
```

Combined Indexes: Example

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

```
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:

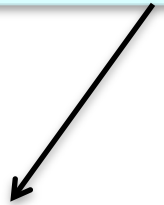
1. Where part
2. Group By/Order (not used now)
3. Select part (here: **name**)

Covered Index: Example

- Explain

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



Query tuning example

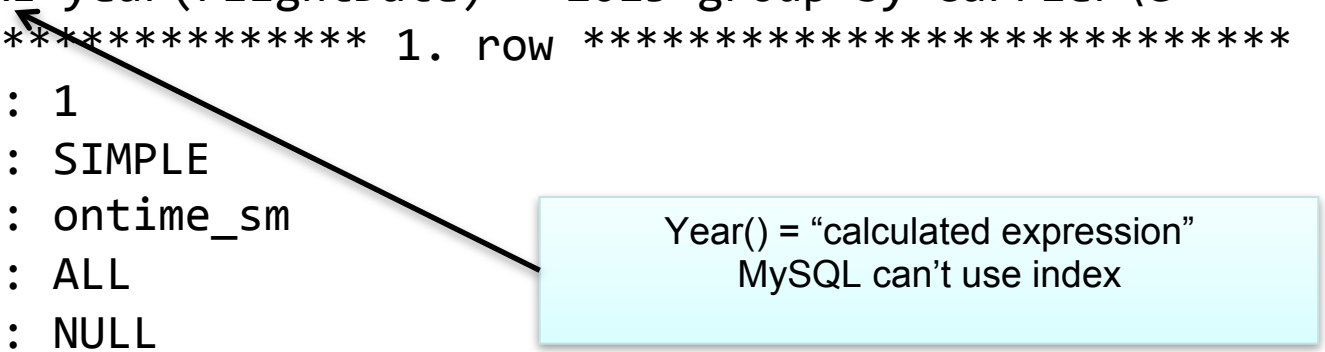
Calculated Expressions In MySQL Queries

Air Traffic Statistics table

```
CREATE TABLE `ontime` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `YearD` year(4) NOT NULL,
  `FlightDate` date DEFAULT NULL,
  `Carrier` char(2) DEFAULT NULL,
  `OriginAirportID` int(11) DEFAULT NULL,
  `OriginCityName` varchar(100) DEFAULT NULL,
  `OriginState` char(2) DEFAULT NULL,
  `DestAirportID` int(11) DEFAULT NULL,
  `DestCityName` varchar(100) DEFAULT NULL,
  `DestState` char(2) DEFAULT NULL,
  `DepDelayMinutes` int(11) DEFAULT NULL,
  `ArrDelayMinutes` int(11) DEFAULT NULL,
  `Cancelled` tinyint(4) DEFAULT NULL,
  `CancellationCode` char(1) DEFAULT NULL,
  `Diverted` tinyint(4) DEFAULT NULL,
  PRIMARY KEY (`id`),
  KEY `FlightDate` (`FlightDate`)
) ENGINE=InnoDB
```

All flights in 2013, group by airline

```
mysql> EXPLAIN SELECT carrier, count(*) FROM ontime_sm
          WHERE year(FlightDate) = 2013 group by carrier\G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: ontime_sm
         type: ALL
possible_keys: NULL
          key: NULL
      key_len: NULL
          ref: NULL
         rows: 151253427
        Extra: Using where; Using temporary; Using filesort

Results:
16 rows in set (1 min 49.48 sec)
```



Rewritten: flights in 2013, group by airline

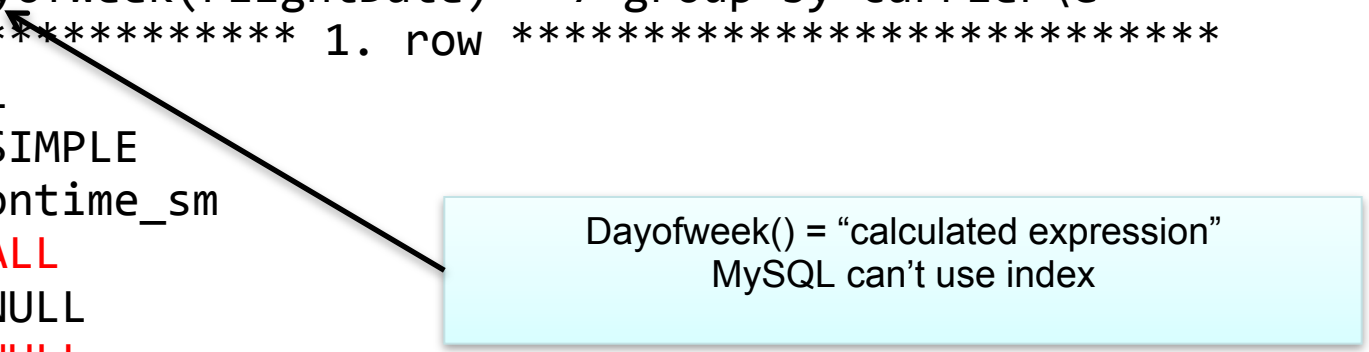
```
mysql> EXPLAIN SELECT carrier, count(*) FROM ontime_sm
       WHERE  FlightDate between '2013-01-01' and '2014-01-01'
       GROUP BY carrier\G

*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: ontime_sm
         type: range
possible_keys: FlightDate
          key: FlightDate
      key_len: 4
          ref: NULL
         rows: 10434762
        Extra: Using index condition; Using temporary; Using filesort
Results:
16 rows in set (11.98 sec)
```

RANGE condition
MySQL can use index

Almost 10x faster

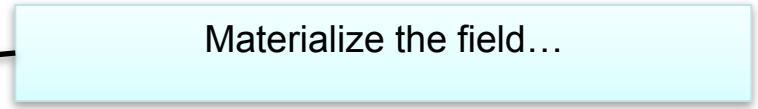
All flights on Sunday example

```
mysql> EXPLAIN SELECT carrier, count(*) FROM ontime_sm
       WHERE dayofweek(FlightDate) = 7 group by carrier\G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: ontime_sm
         type: ALL
possible_keys: NULL
          key: NULL
      key_len: NULL
          ref: NULL
         rows: 151253427
        Extra: Using where; Using temporary; Using filesort

Results:
32 rows in set (1 min 57.93 sec)
```



Storing additional field

```
CREATE TABLE `ontime_sm` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `YearD` year(4) NOT NULL,
  `FlightDate` date DEFAULT NULL,
  `Carrier` char(2) DEFAULT NULL,
  `OriginAirportID` int(11) DEFAULT NULL,
  `OriginCityName` varchar(100) DEFAULT NULL,
  `OriginState` char(2) DEFAULT NULL,
  `DestAirportID` int(11) DEFAULT NULL,
  `DestCityName` varchar(100) DEFAULT NULL,
  `DestState` char(2) DEFAULT NULL, DEFAULT
  `DepDelayMinutes` int(11) NULL,
  `ArrDelayMinutes` int(11) DEFAULT NULL,
  `Cancelled` tinyint(4) DEFAULT NULL,
  …
  Flight_dayofweek tinyint NOT NULL,
  PRIMARY KEY (`id`),
  KEY `Flight_dayofweek` (`Flight_dayofweek`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
```



Storing additional field … and populating it with trigger

```
CREATE DEFINER = CURRENT_USER
TRIGGER ontime_insert
BEFORE INSERT ON ontime_sm_triggers
FOR EACH ROW
SET
NEW.Flight_dayofweek = dayofweek(NEW.FlightDate);
```

May be slower on insert

Virtual Columns in MySQL 5.7

```
CREATE TABLE `ontime_sm_virtual` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `YearD` year(4) NOT NULL,
  `FlightDate` date DEFAULT NULL,
  `Carrier` char(2) DEFAULT NULL,
...
`Flight_dayofweek` tinyint(4)
GENERATED ALWAYS AS (dayofweek(FlightDate)) VIRTUAL,
PRIMARY KEY (`id`),
KEY `Flight_dayofweek` (`Flight_dayofweek `)
) ENGINE=InnoDB;
```

Does not store the column
But INDEX it

http://mysqlserverteam.com/generated-columns-in-mysql-5-7-5/
https://dev.mysql.com/worklog/task/?id=8114
http://labs.mysql.com/

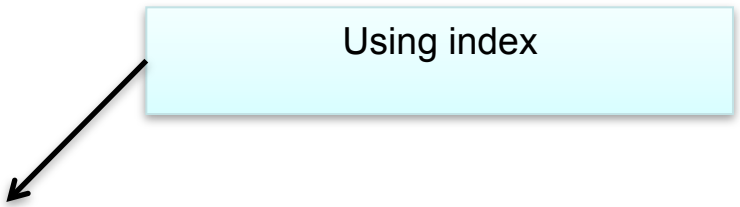
Virtual Columns in MySQL 5.7

```
mysql> insert into ontime_sm_triggers (id, YearD, FlightDate,
Carrier, OriginAirportID, OriginCityName,  OriginState,
DestAirportID, DestCityName, DestState, DepDelayMinutes,
ArrDelayMinutes, Cancelled, CancellationCode,Diverted,
CRSElapsedTime, ActualElapsedTime, AirTime, Flights, Distance)
select * from ontime_sm;

Query OK, 999999 rows affected (27.86 sec)
Records: 999999  Duplicates: 0  Warnings: 0

mysql> insert into ontime_sm_virtual (id, YearD, FlightDate,
Carrier, OriginAirportID, OriginCityName,  OriginState,
DestAirportID, DestCityName, DestState, DepDelayMinutes,
ArrDelayMinutes, Cancelled, CancellationCode,Diverted,
CRSElapsedTime, ActualElapsedTime, AirTime, Flights, Distance)
select * from ontime_sm;

Query OK, 999999 rows affected (16.29 sec)
Records: 999999  Duplicates: 0  Warnings: 0
```

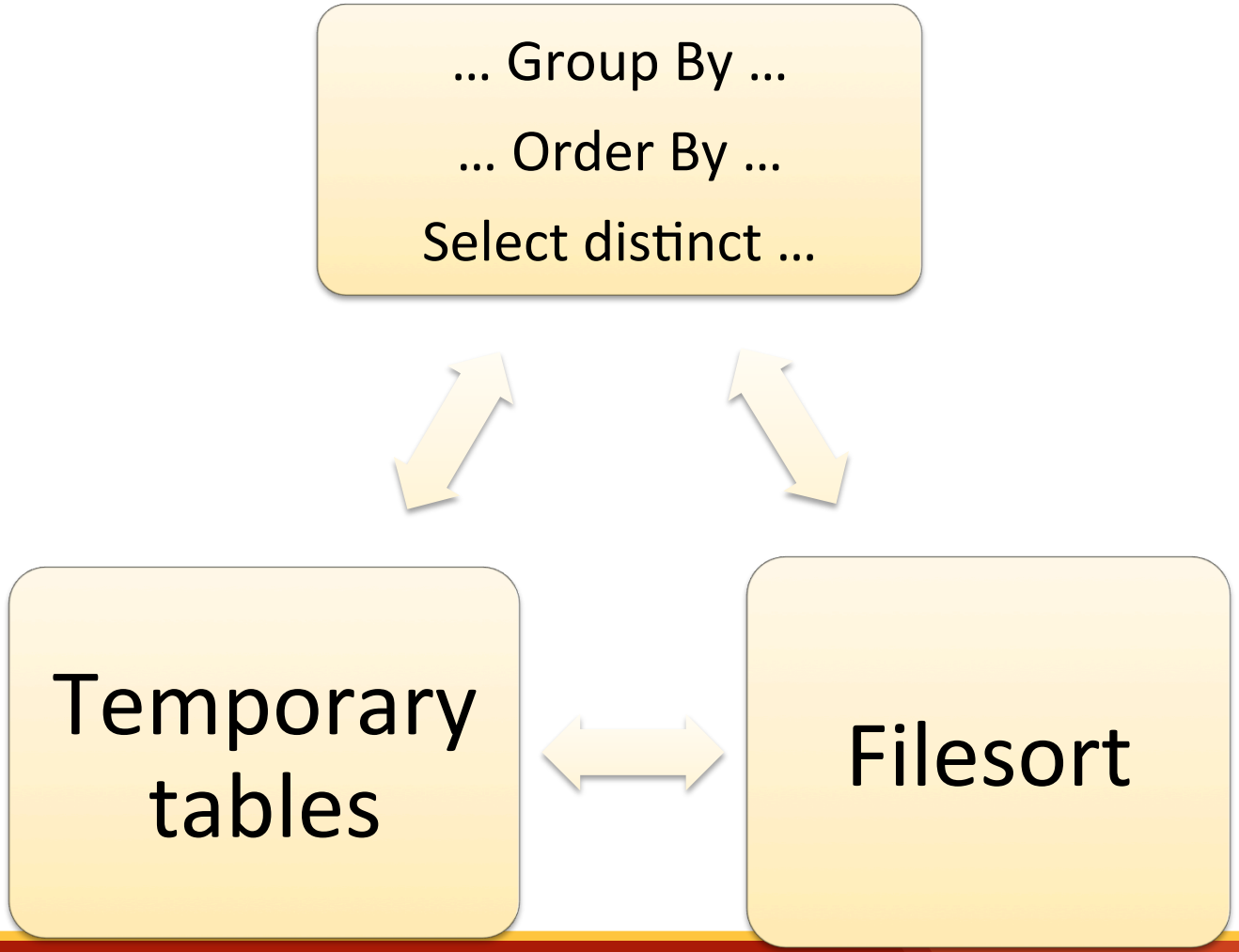
Much faster on load
Compared to triggers

Virtual Columns in MySQL 5.7

```
mysql> EXPLAIN SELECT carrier, count(*) FROM ontime_sm_virtual
WHERE Flight_dayofweek = 7 group by carrier\G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: ontime_sm_virtual
   partitions: NULL
         type: ref
possible_keys: Flight_dayofweek
          key: Flight_dayofweek
      key_len: 2
          ref: const
         rows: 165409
     filtered: 100.00
        Extra: Using where; Using temporary; Using filesort
1 row in set, 1 warning (0.00 sec)
```



Complex Slow Queries



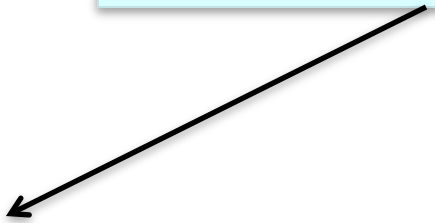
GROUP BY Queries

GROUP BY and Temporary Tables

- How many cities in each country?

```
mysql> explain select CountryCode, count(*) from City group by
    CountryCode\G

           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

- 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

Air Traffic Statistics Table for Testing

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

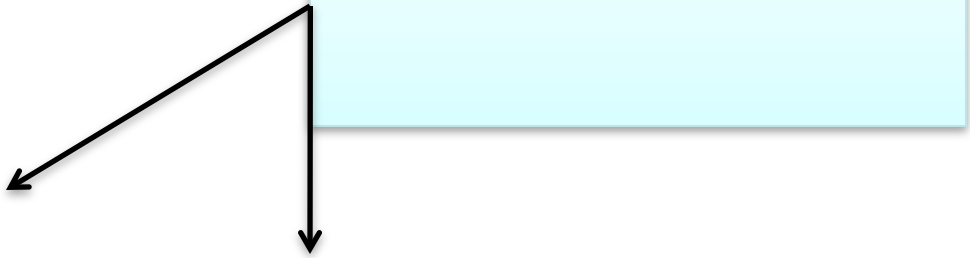
- Find maximum delay for flights on Sunday
- Group by airline

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

GROUP BY Query Example

```
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\G
```

```
           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);
```

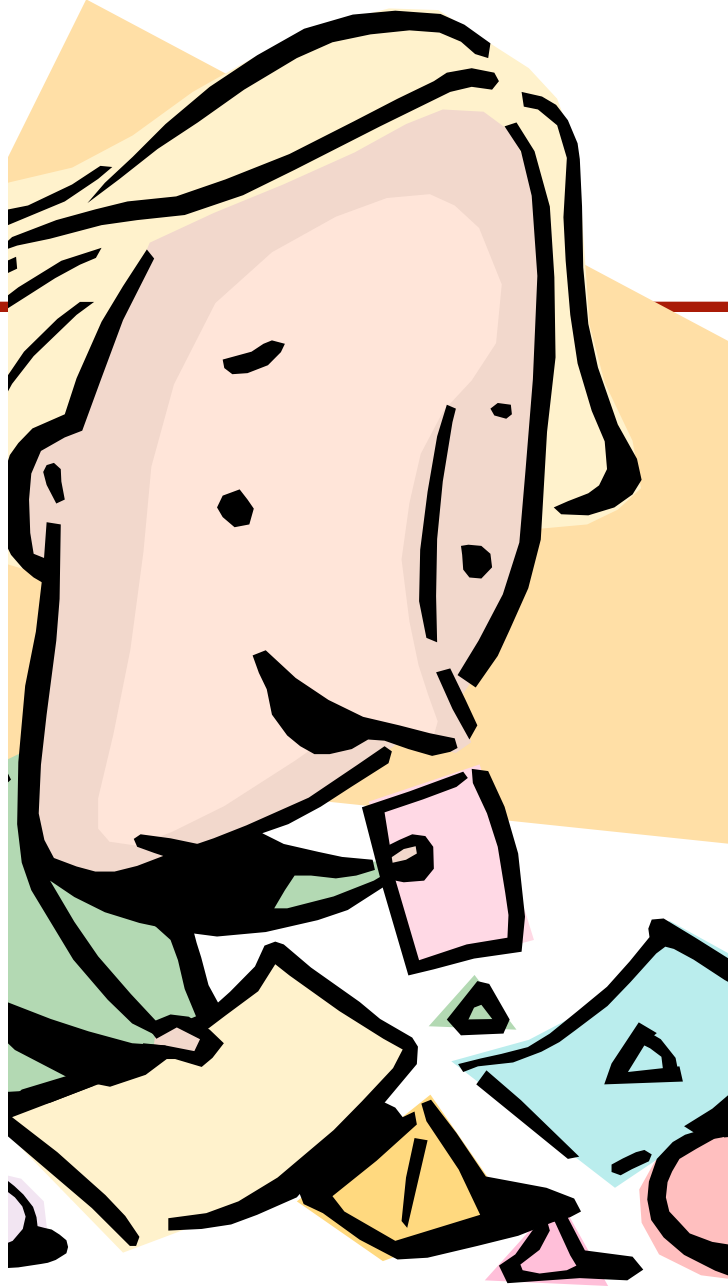
```
explain select max(DepDelayMinutes), Carrier, dayofweek from ontime_2012 where
dayofweek =7 group by Carrier\G
```

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

No temporary table!
MySQL will only use index

PERCONA

ORDER BY and filesort



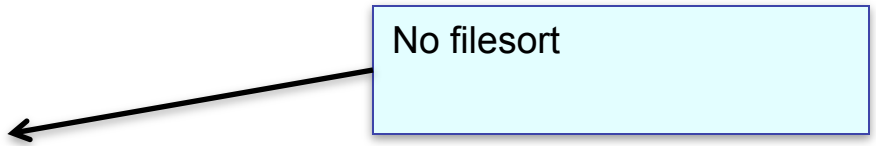
ORDER BY and filesort

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: City
            type: ALL
possible_keys: NULL
             key: NULL
         key_len: NULL
             ref: NULL
            rows: 4079
           Extra: Using where; Using filesort
```

Fixing Filesort: Adding Index

```
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
```

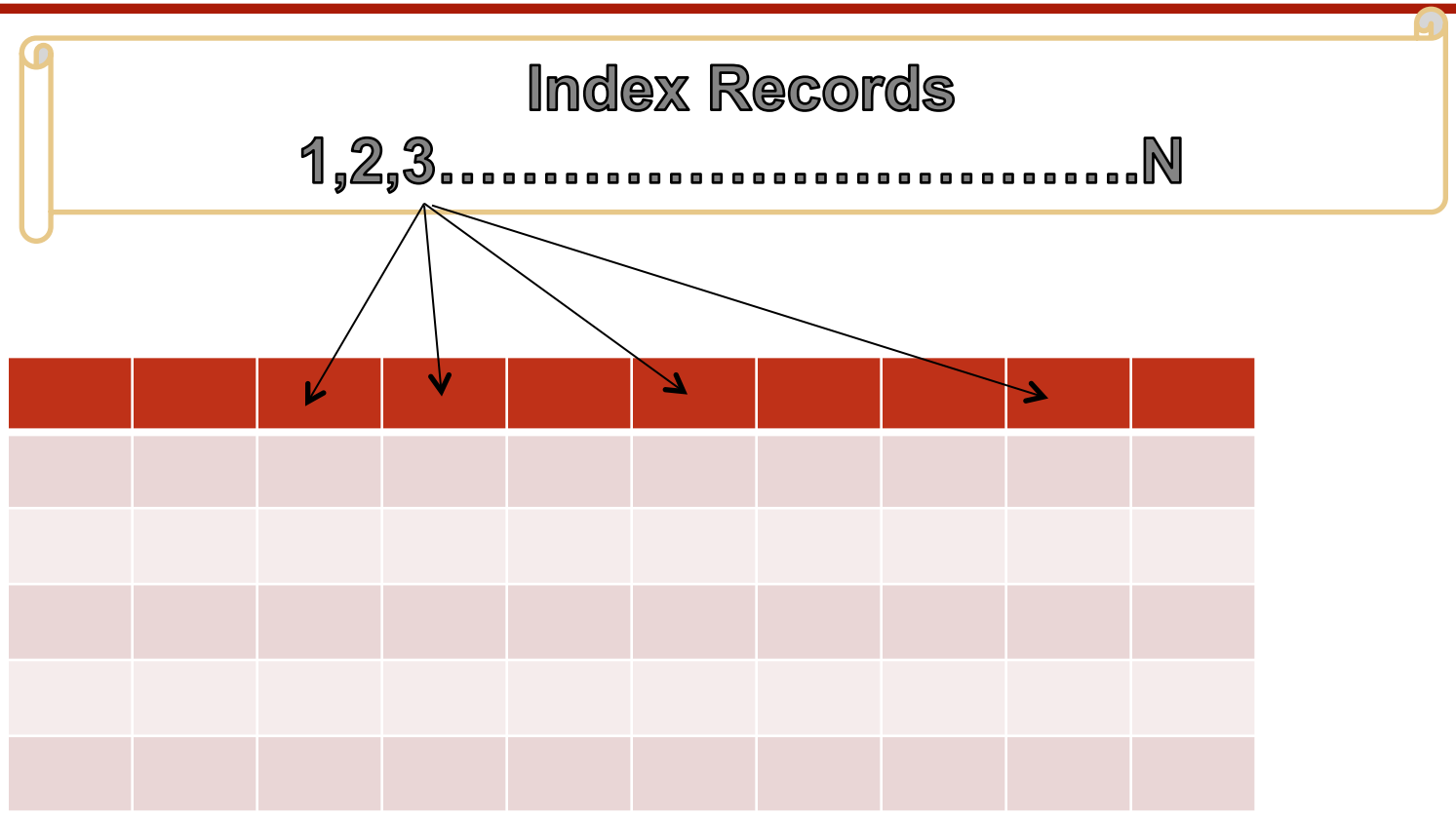


Sorting and Limit

```
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)
```

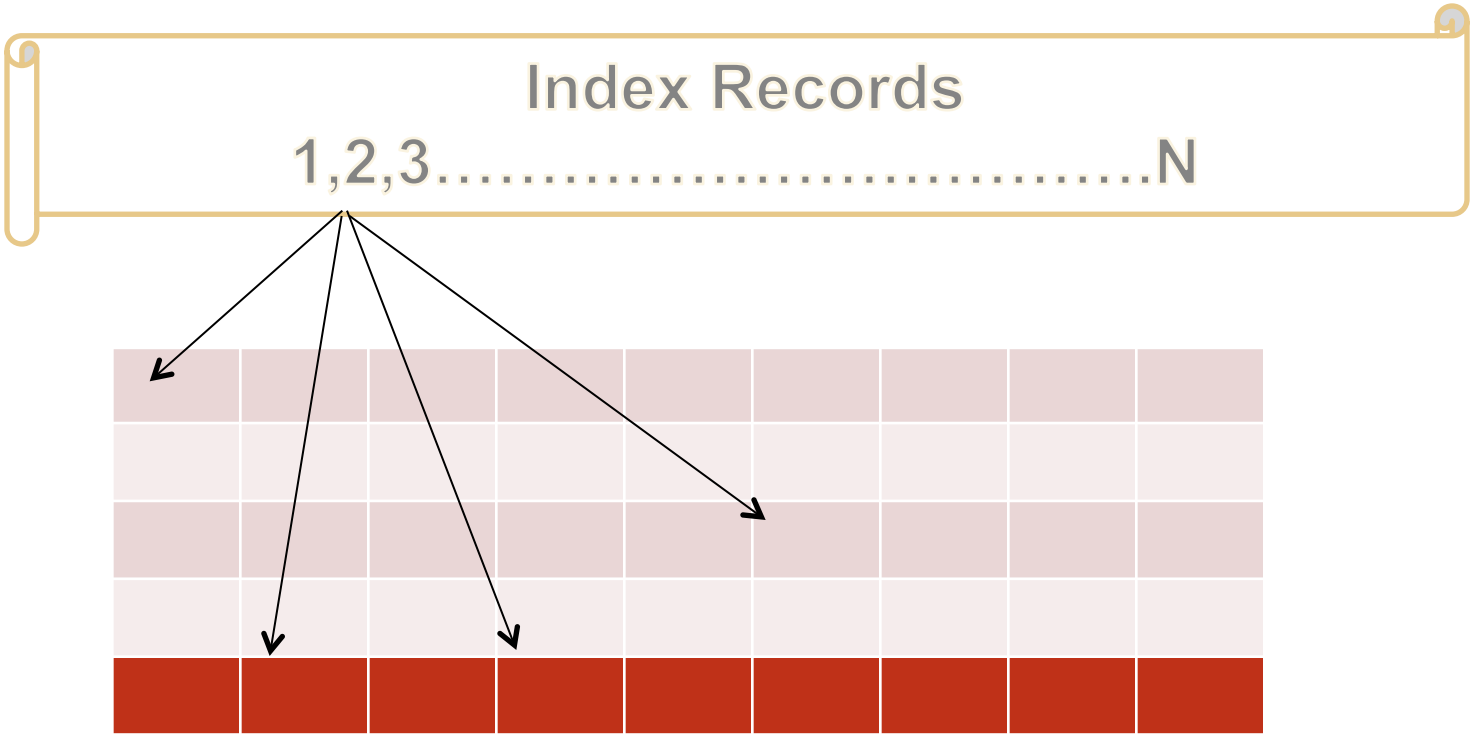
1. Index is sorted
2. Scan the **whole table** in the order of the index
3. Filter results
4. Stop after finding 10 rows matching the “where” condition

Sorting and Limit



If Index points to the beginning of the table (physically) = fast
As it stops after 10 rows (LIMIT 10)

Sorting and Limit



If Index points to the end of table (physically) or random = slower
Much more rows to scan (and skip)

Thank you!

Webinar Replay:
Advanced Query Tuning in MySQL 5.6 and 5.7
https://www.percona.com/resources/mysql-webinars/advanced-query-tuning-mysql-56-and-57