Advanced MySQL Query Tuning

Alexander Rubin
Principal Architect, Percona
April 18, 2015

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
- Helping customers improve MySQL performance
  - performance tuning
  - full text search
  - high availability

Agenda

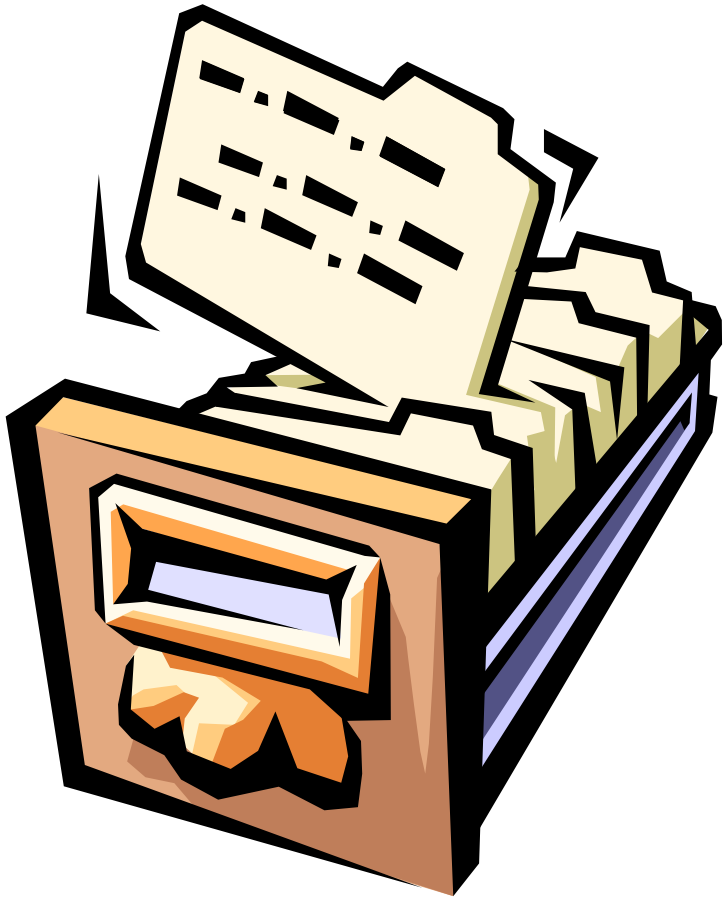
- Queries
  - Temporary Tables and Filesort GROUP BY Optimizations
  - ORDER BY Optimizations
  - DISTINCT Queries
  - “Calculated” fields – MySQL 5.7

New this year

- Removed basic staff to focus on advanced
- MySQL 5.6 by default
  - Will *not* talk about subqueries issues
- Improvements in MySQL 5.7

Query Tuning: basics

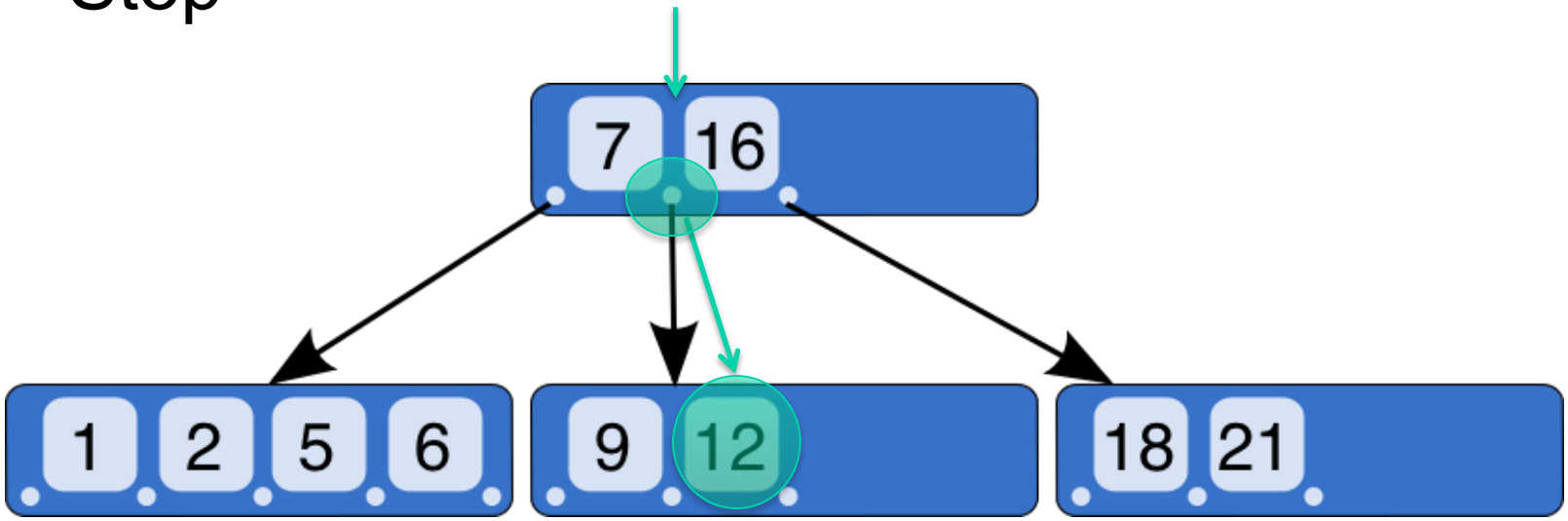
Indexes and Explain examples



MySQL Index: B-Tree

Equality search: select * from table where id = 12

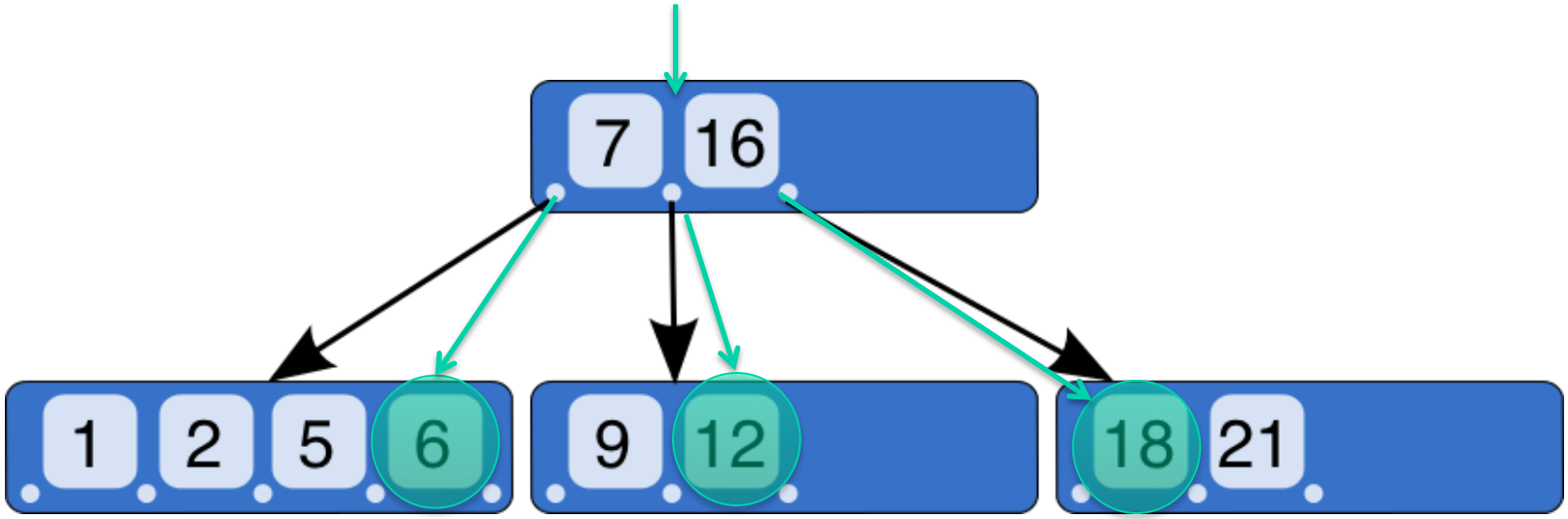
- Scan thru the tree and go directly to 1 leaf
- Stop



MySQL Index: B-Tree

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

- Scan thru the tree and visit many leafs/nodes



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

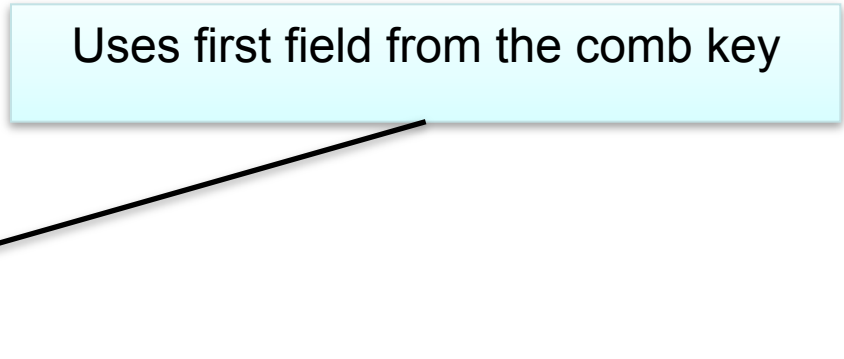
- Leftmost part of combined index

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

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
```



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

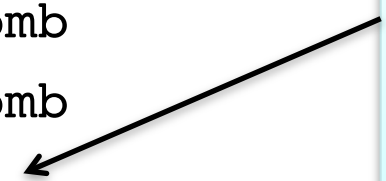
```
mysql> explain select * from City
where CountryCode = 'USA' and District = 'California'\G
*************************** 1. row ***************************
           table: City
            type: ref
possible_keys: comb
             key: comb
         key_len: 23
             ref: const,const
    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

Uses ***2*** first fields from the comb key
CountryCode = 3 chars
District = 20 chars
Total = 23

Combined Indexes: 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

```
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:

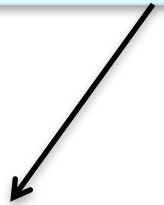
1. Where part
2. Group By/Order (not used now)
3. Select part (here: **name**)

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)
```

Year() = “calculated expression”
MySQL can’t use index

Rewritten: flights in 2013, group by airline

```
mysql> EXPLAIN SELECT carrier, count(*) FROM ontime_sm
       WHERE  FlightDate between '2013-01-01' and '2013-12-31'
       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)
```

Year() = “calculated expression”
MySQL can’t use index

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

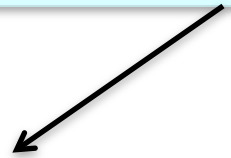
```
CREATE DEFINER = CURRENT_USER
TRIGGER ontime_insert
BEFORE INSERT ON ontime_sm_triggers
FOR EACH ROW
SET
NEW.Flight_dayofweek = dayofweek(NEW.FlightDate);
```

May be slower on insert

Virtual Columns in MySQL 5.7.7 (labs release)

```
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,
  `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,
  ...
`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/

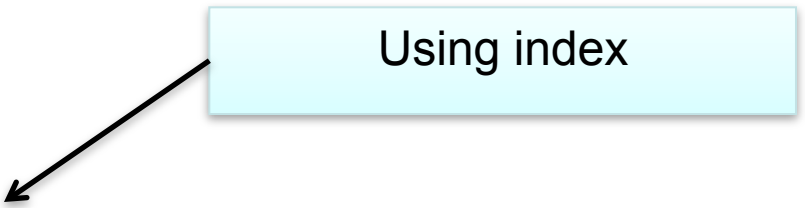
Virtual Columns in MySQL 5.7.7 (labs release)

```
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 5.7.7 (labs release)

```
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)
```

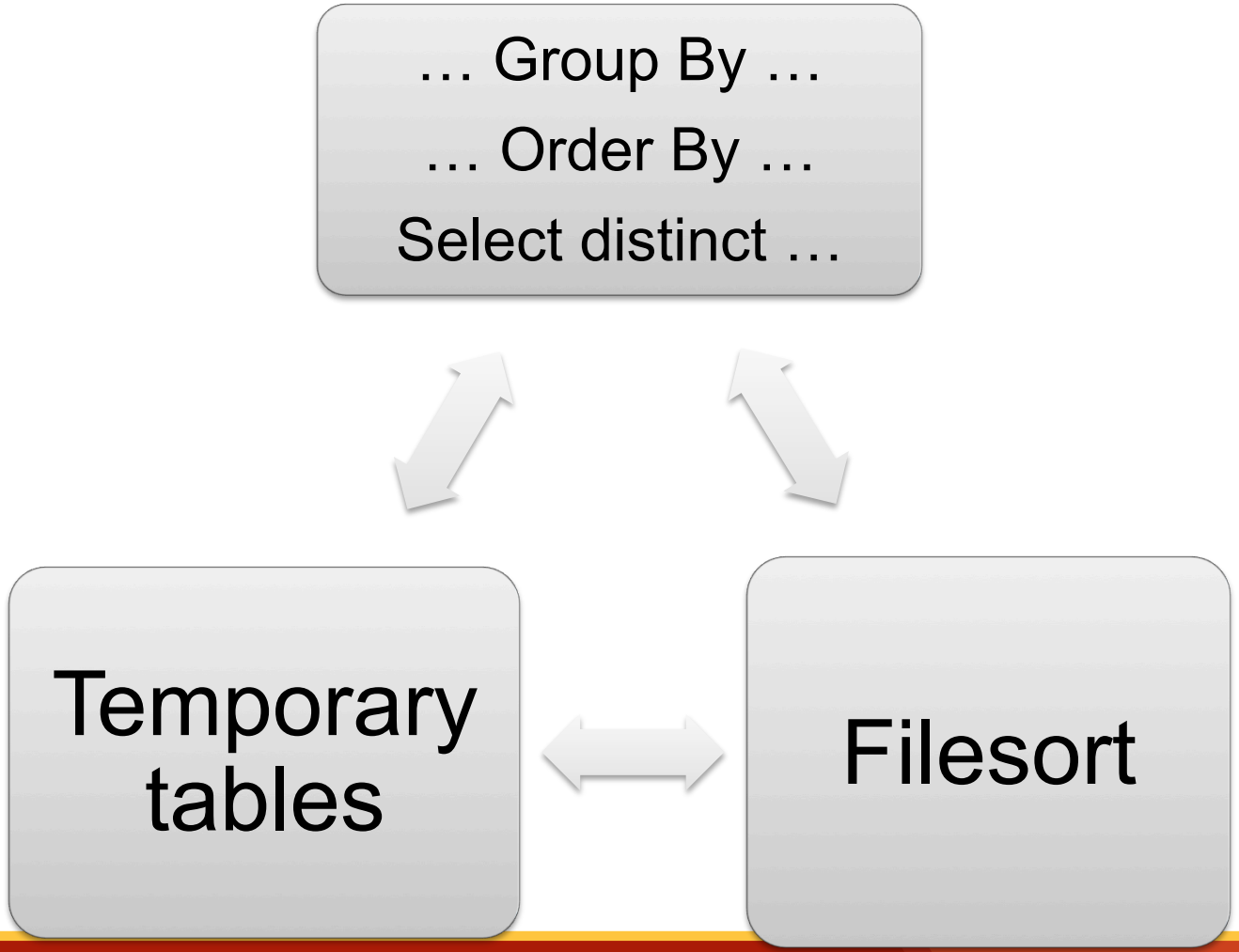


Virtual Columns in MySQL 5.7.7 Limitations

```
mysql> alter table ontime_sm_virtual
       add key comb(Flight_dayofweek, carrier);

ERROR 3105 (HY000): 'Virtual generated column combines with
other columns to be indexed together' is not supported for
generated columns.
```

Can’t create a combined index
on virtual and “normal” columns

Complex Slow Queries



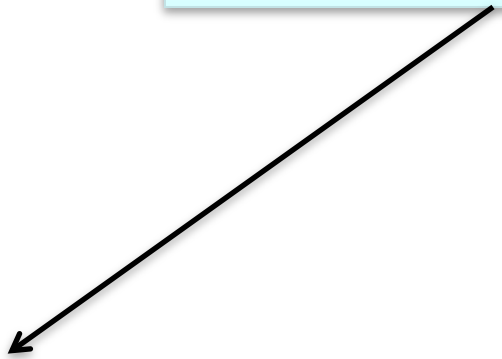
GROUP BY Queries

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: Theory

Temporary Tables, I

Main performance issues

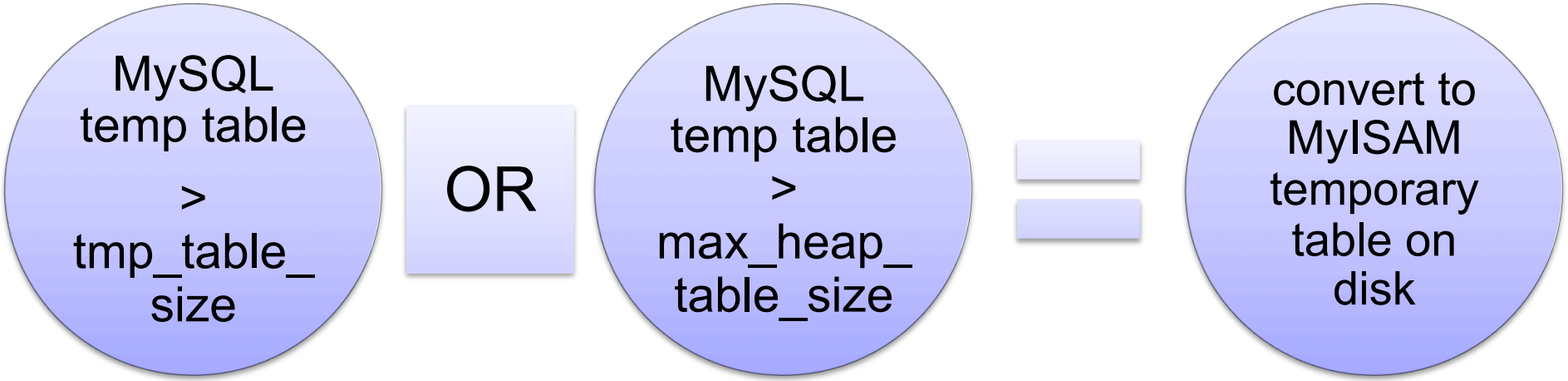
- MySQL can create temporary tables when query uses:
  - GROUP BY
  - Range + ORDER BY
  - Some other expressions

- 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



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

Temporary Tables: Practice

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

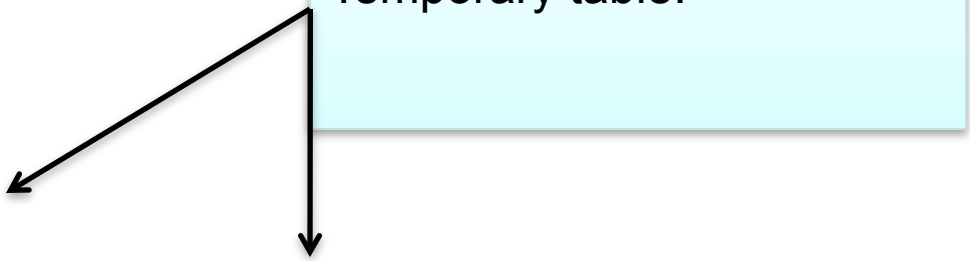
- Find maximum delay for flights on Sunday
- Group by airline

```
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

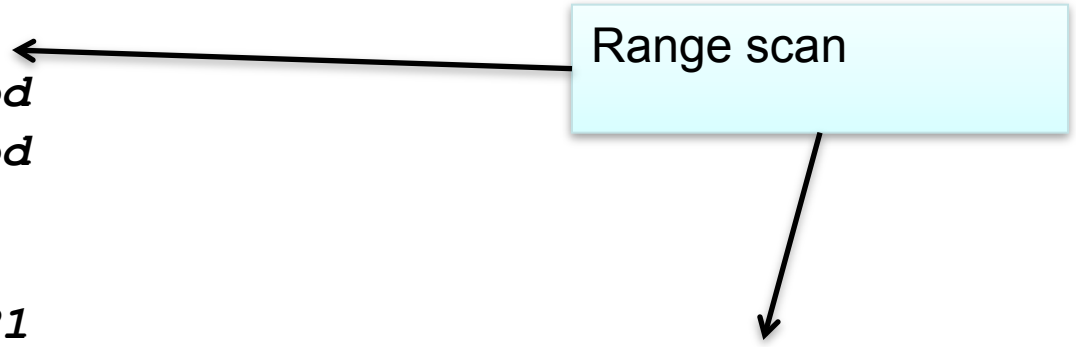

```
mysql> alter table ontime_2012
add key covered(dayofweek, Carrier, DepDelayMinutes);
explain select max(DepDelayMinutes), Carrier, dayofweek from
ontime_2012 where dayofweek =7 group by Carrier\G
...
possible_keys: DayOfWeek,covered
          key: covered
      key_len: 2
          ref: const
         rows: 905138
        Extra: Using where; Using index
```

No temporary table!
MySQL will only use index

- 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
...

           type: range
possible_keys: covered
            key: covered
        key_len: 2
            ref: NULL
           rows: 2441781
          Extra: Using where; Using index; Using temporary;
Using filesort
```



Converting query to Union


```
(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)
```

Converting query to Union


```
*************************** 1. row ***************************
           table: ontime_2012
             key: covered
...
           Extra: Using where; Using index
*************************** 2. row ***************************
           table: ontime_2012
             key: covered
...
           Extra: Using where; Using index
*************************** 3. row ***************************
              id: NULL
     select_type: UNION RESULT
           table: <union1,2>
            type: ALL
   possible_keys: NULL
             key: NULL
         key_len: NULL
             ref: NULL
            rows: NULL
           Extra: Using temporary
```

UNION ALL
Will not create temp table

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 columns should form a leftmost prefix of the index
  - The only aggregate functions = 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
    ...
         table: ontime_2012
          type: range
possible_keys: NULL
           key: loose_index_scan
       key_len: 5
           ref: NULL
          rows: 201
         Extra: Using where; Using index for group-by
```

Loose index scan
Very fast
“Range” works!

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
```

Loose index scan vs. tight index scan

Loose index scan

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

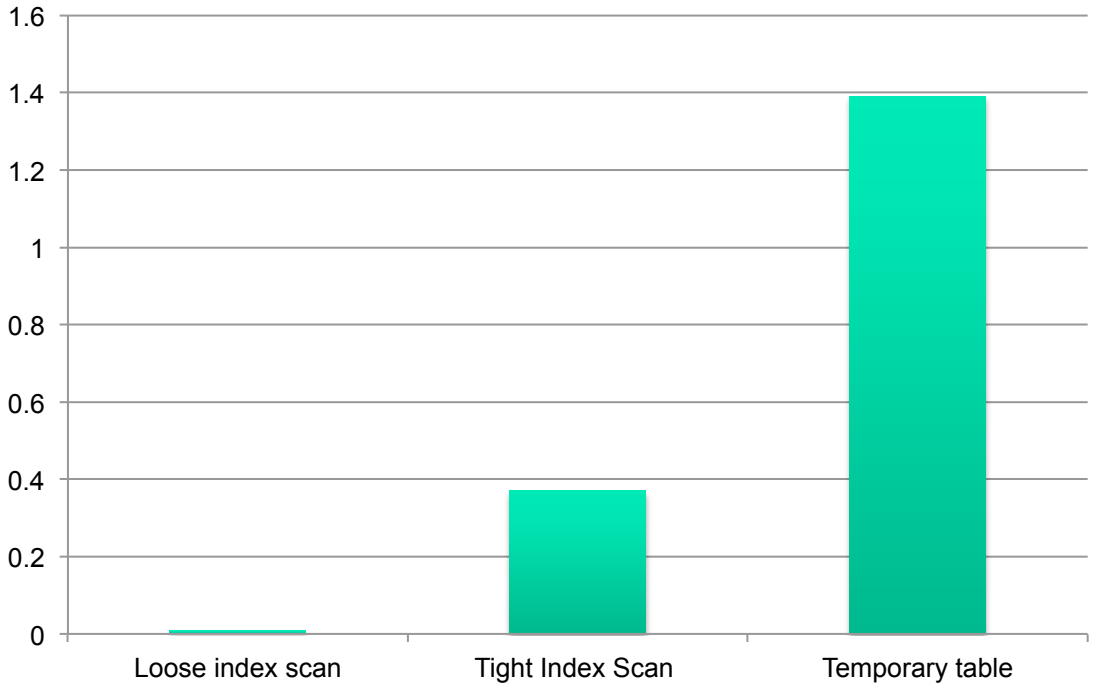
```
        table: ontime_2012
         type: range
possible_keys: covered
          key: loose_index_scan
      key_len: 5
          ref: NULL
         rows: 201
        Extra: Using where; Using index for group-by
```

Carrier, DayOfWeek, DepDelayMinutes

```
mysql> select ...
+------+---------+-----------+
| ddm  | Carrier | dayofweek |
+------+---------+-----------+
| 1606 | AA      |         7 |
..
30 rows in set (0.00 sec)
```

Loose index scan vs. tight index scan

Results



Where loose index scan is not supported

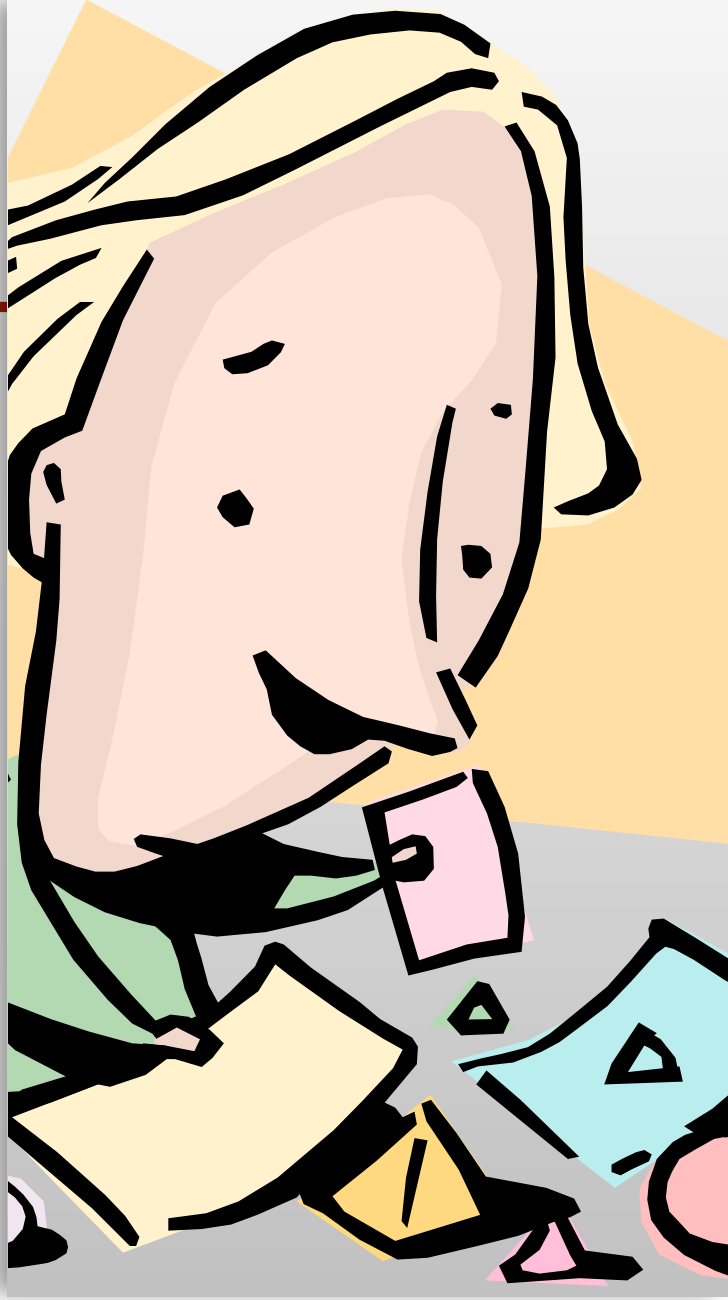
- AVG() + Group By – loose index scan is not supported

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

          table: ontime_2012
           type: range
            key: covered
        key_len: 2
            ref: NULL
           rows: 2961617
          Extra: Using where; Using index; Using temporary; Using filesort
```

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

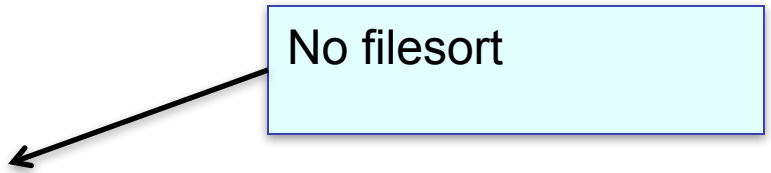
Find 10 cities in the US with the largest population

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

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

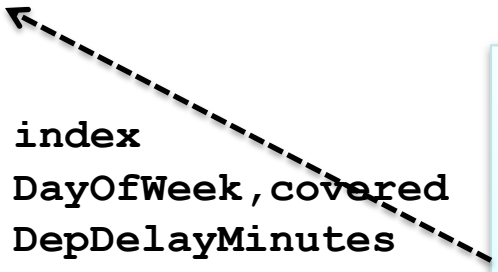
Fixing Filesort: Adding Index

```
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
```



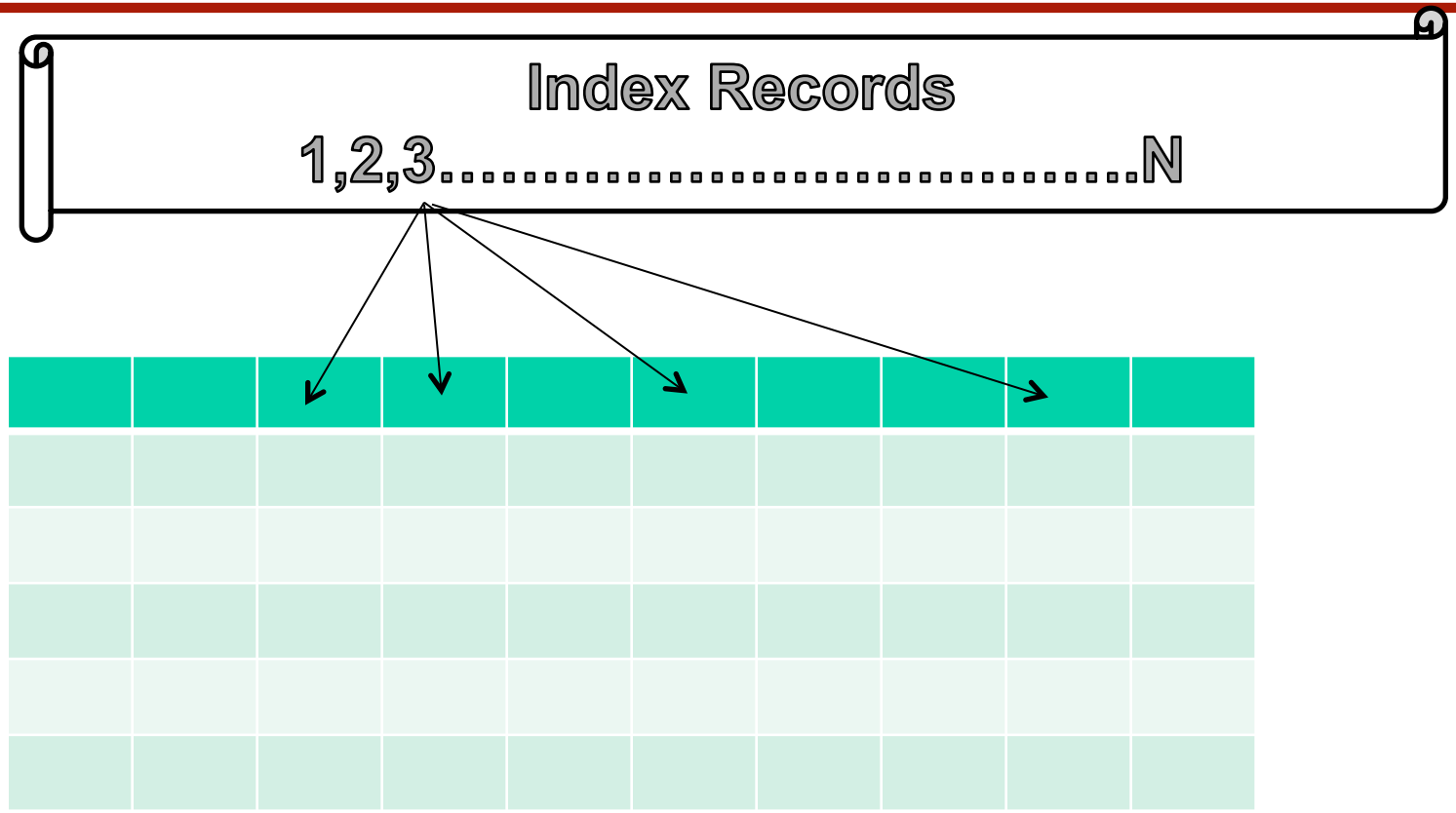
Sorting and Limit

```
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)
```



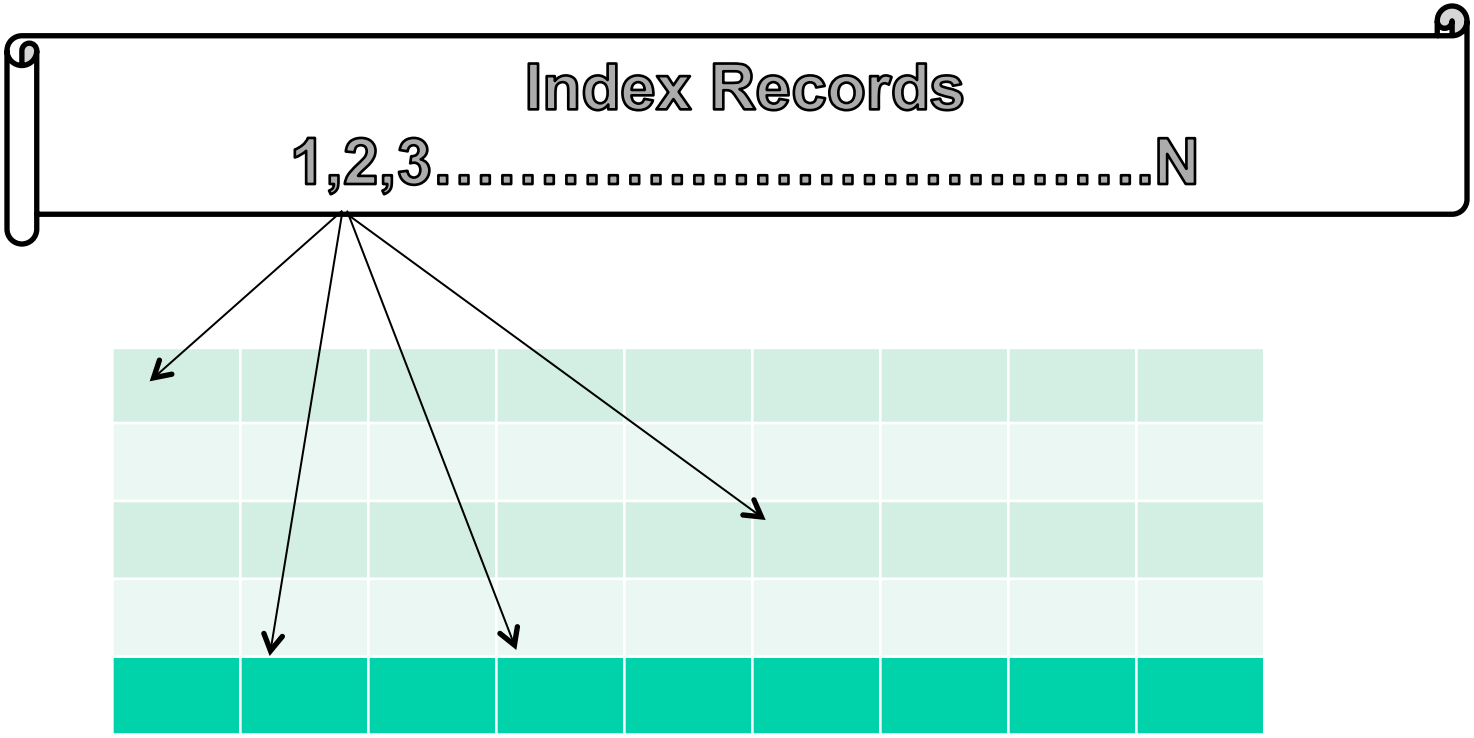
1. Index is sorted
2. Scan the **whole table** in the order of the index
3. Filter results
4. Stop after finding 10 rows matching the “where” condition

Sorting and Limit



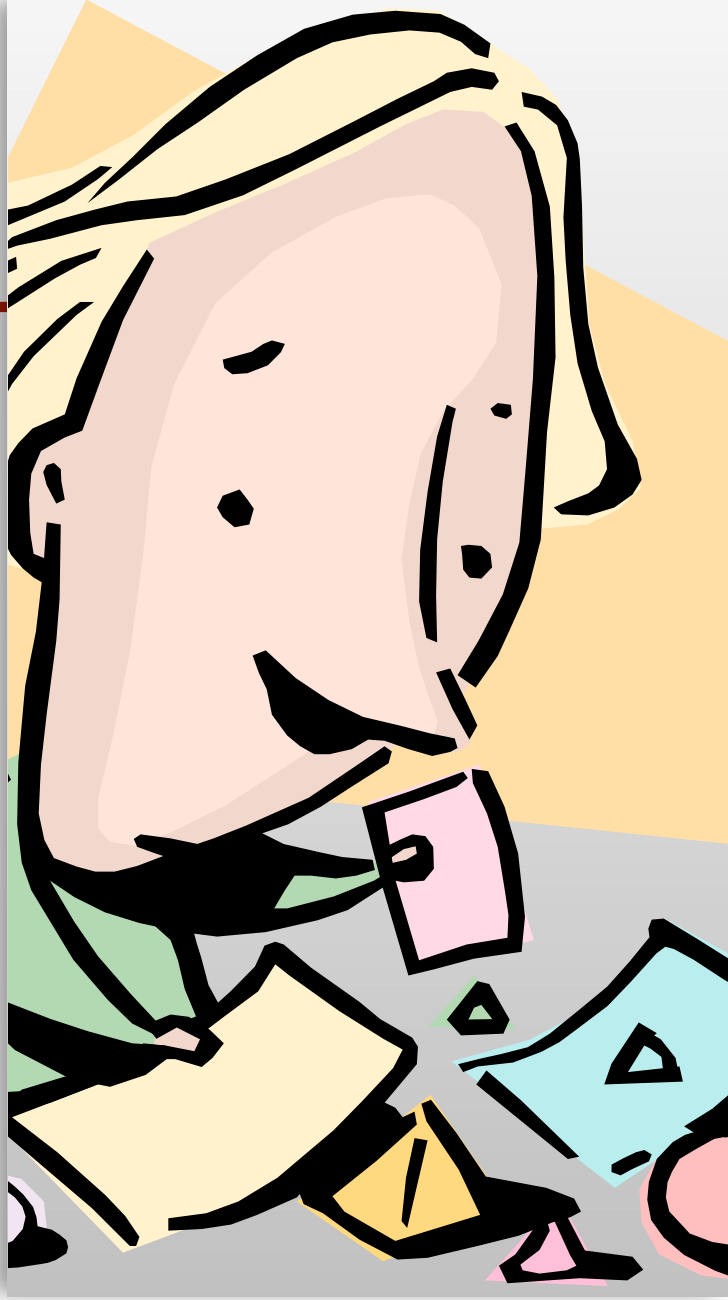
If Index points to the beginning of the table (physically) = fast
As it stops after 10 rows (LIMIT 10)

Sorting and Limit



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

SELECT DISTINCT



DISTINCT example

```
select t.*, c.name from title t
join movie_companies m on t.id = m.movie_id
join company_name c on m.company_id = c.id
join company_type ct on m.company_type_id = ct.id
where
production_year > 1960  and ct.kind <> 'distributors'
order by production_year desc limit 10\G
...
10 rows in set (0.00 sec)
```

JOINs are for filtering
Result set may contain duplicates
DISTINCT?

DISTINCT example

```
select DISTINCT t.*, c.name from title t
join movie_companies m on t.id = m.movie_id
join company_name c on m.company_id = c.id
join company_type ct on m.company_type_id = ct.id
where
production_year > 1960  and ct.kind <> 'distributors'
order by production_year desc limit 10\G
...
10 rows in set (37.27 sec)
```

DISTINCT example

```
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: t
         type: range
possible_keys: PRIMARY,production_year
          key: production_year
      key_len: 5
          ref: NULL
         rows: 1163888
        Extra: Using index condition; Using temporary
*************************** 2. row ***************************
           id: 1
  select_type: SIMPLE
        table: m
  ...
```

DISTINCT example

```
mysql> show status like 'created%';
+-------------------------+-------+
| Variable_name           | Value |
+-------------------------+-------+
| Created_tmp_disk_tables | 1     |
| Created_tmp_files       | 0     |
| Created_tmp_tables      | 1     |
+-------------------------+-------+
3 rows in set (0.00 sec)
```

DISTINCT example: Hack

```
select distinct * from
 (select t.*, c.name from title t
  join movie_companies m on t.id = m.movie_id
  join company_name c on m.company_id = c.id
  join company_type ct on m.company_type_id = ct.id
  where production_year > 1960
  and ct.kind <> 'distributors'
  order by production_year desc limit 1000)
as a limit 10;

...
10 rows in set (0.02 sec)
```

Questions?

Thank you!