Advanced MySQL Query and Schema Tuning

Alexander Rubin

April 25, 2013

PERCONA LIVE

About Me

My name is Alexander Rubin

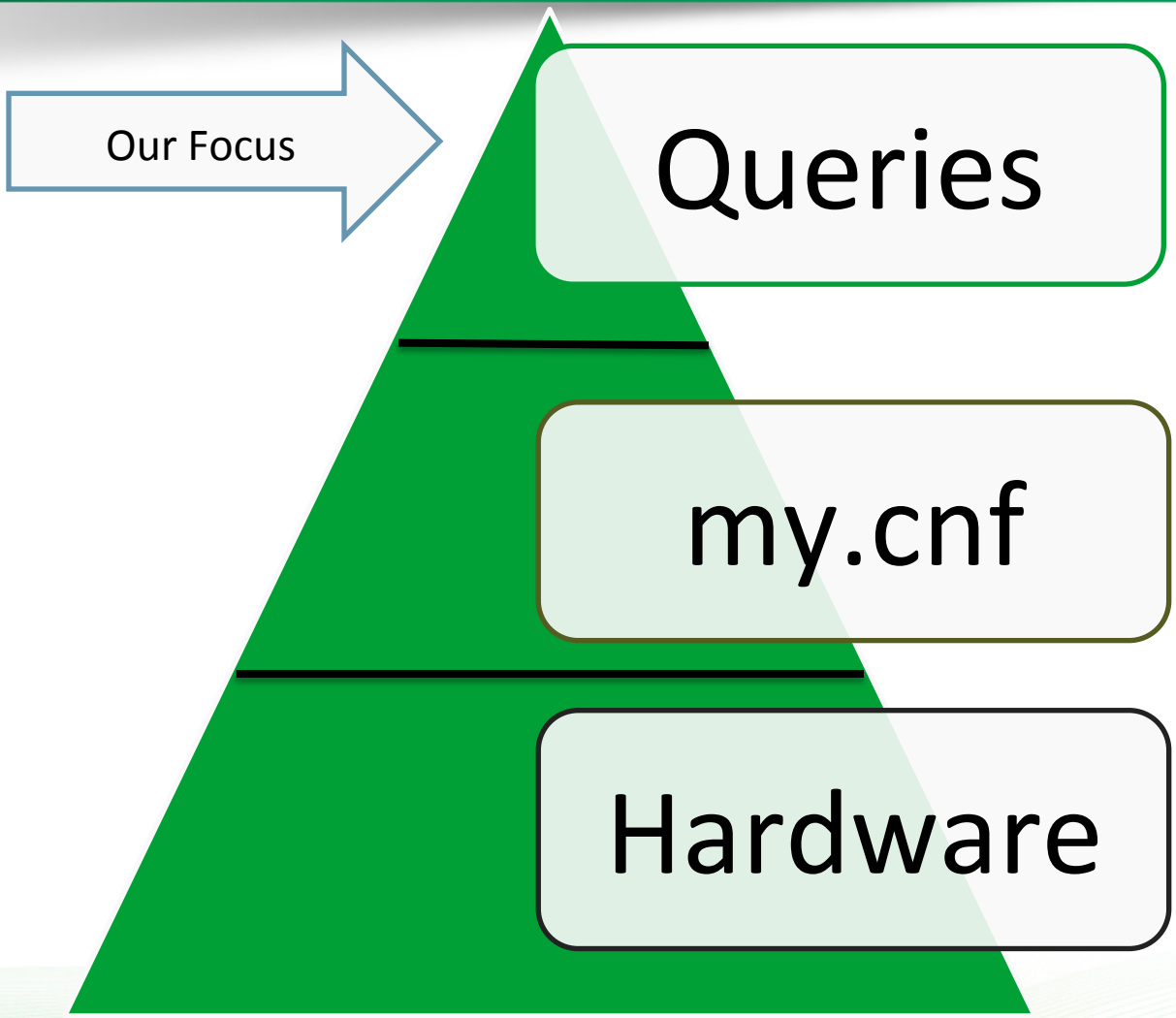
- Working with MySQL for over 10 years
  - Started at MySQL AB, then Sun Microsystems, then Oracle (MySQL Consulting)
  - Joined Percona recently
- Helping customers improve MySQL performance
  - performance tuning
  - full text search
  - high availability
  - Reporting, database infrastructure scale-outs
  - Big data

My Blog:
http://www.arubin.org

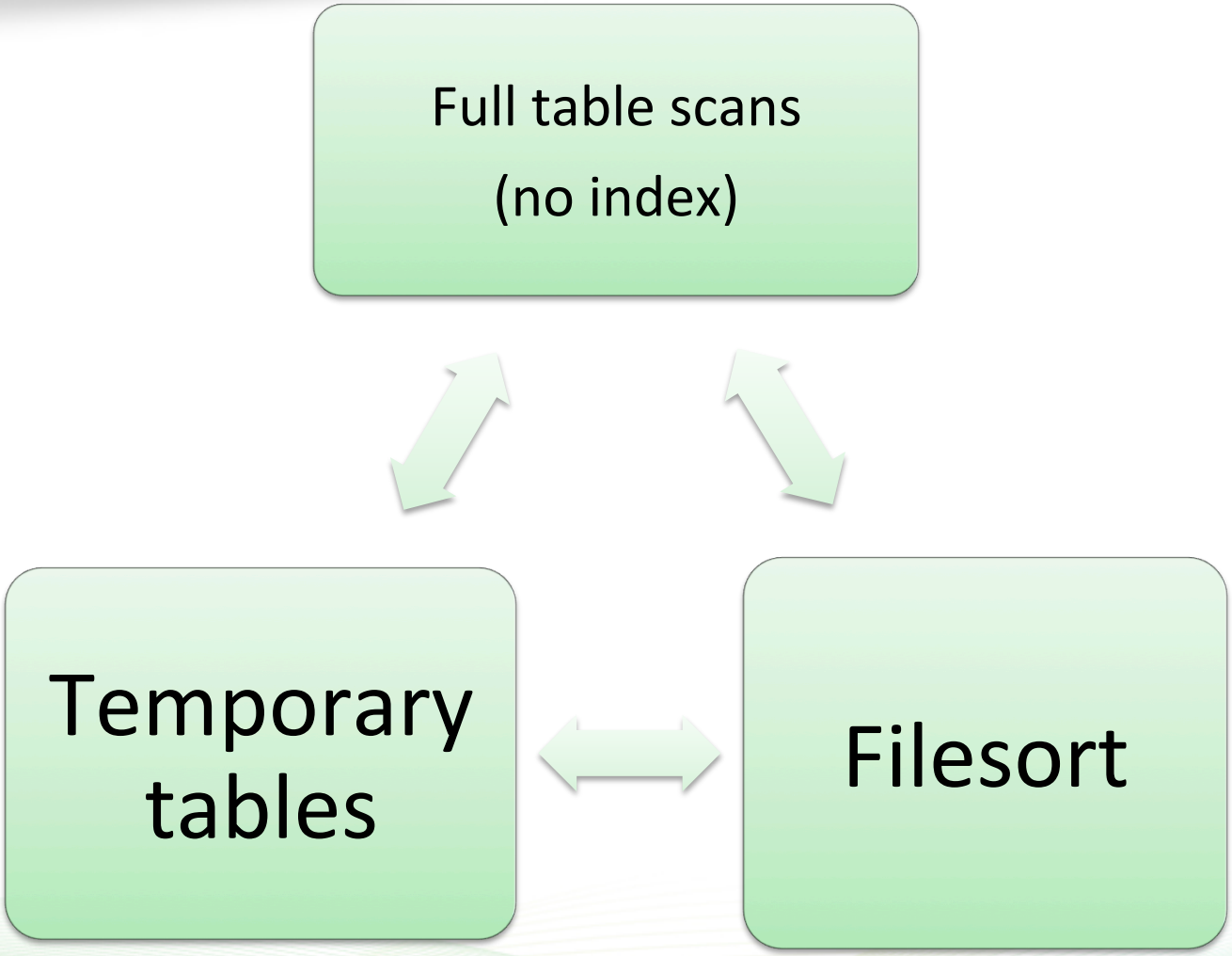
Agenda

- Indexes
  - How B-tree works
  - Range scans
- Queries
  - Temporary Tables and Filesort in MySQL
  - GROUP BY / ORDER BY Optimizations
  - Subqueries
  - Reporting Queries

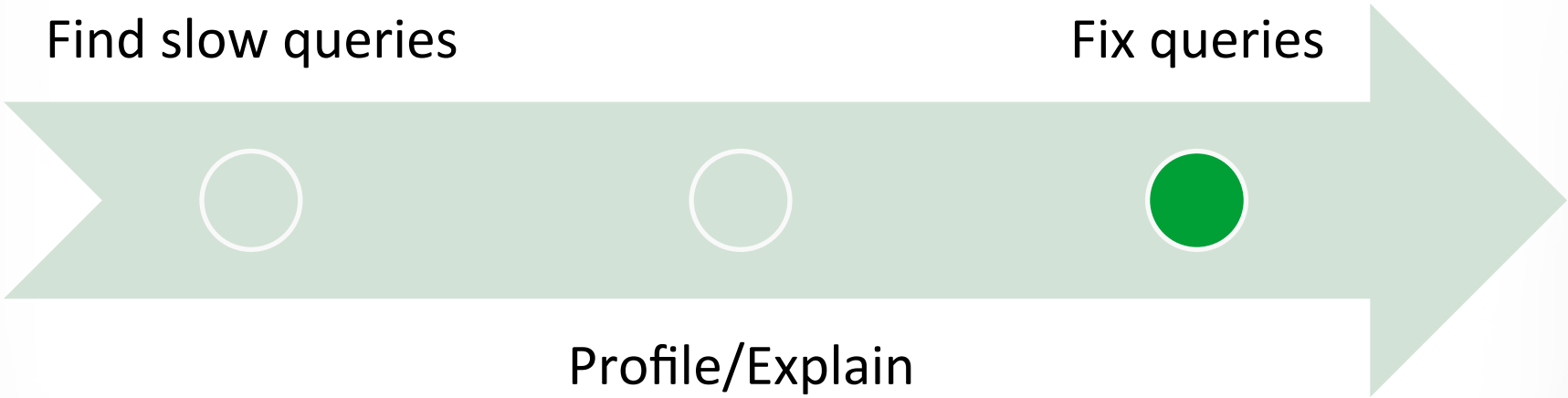
MySQL Tuning



Main Query Performance Problems



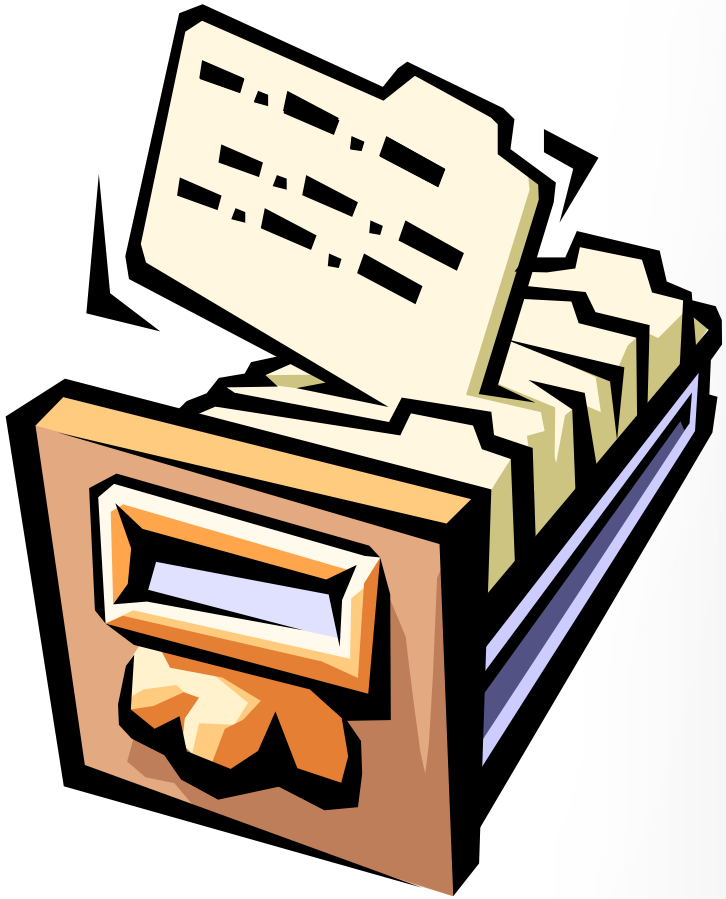
How to Deal with Slow Performance



- = Better performance!

How to Deal with Slow Performance

Indexes

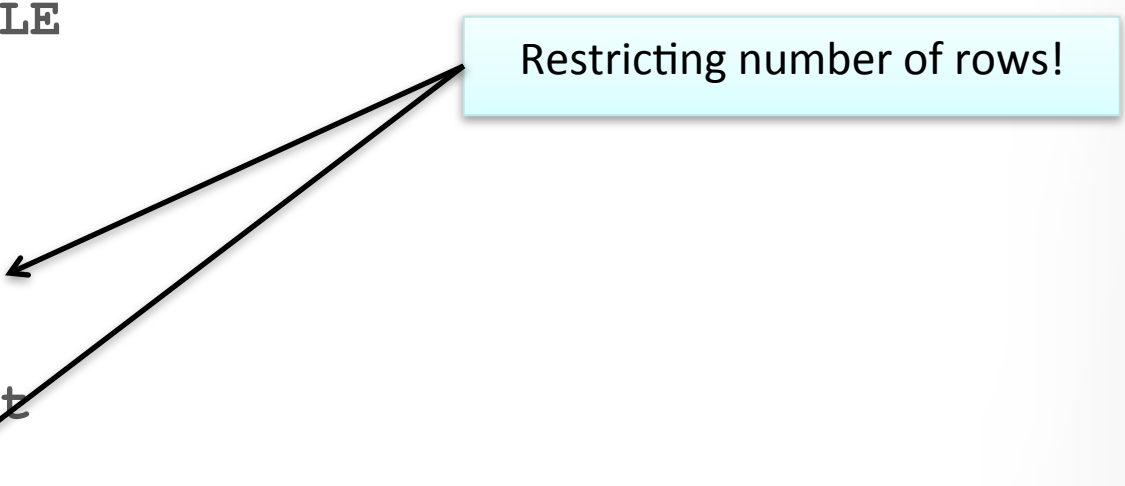


Using Explain: Simple Query Example

```
mysql> EXPLAIN select * from City where Name = 'London'\G
*************************** 1. row

           id: 1
  select_type: SIMPLE
        table: City
         type: ALL
possible_keys: NULL
          key: NULL
      key_len: NULL
          ref: NULL
         rows: 4079
        Extra: Using where
```

Adding Index to Fix a Query

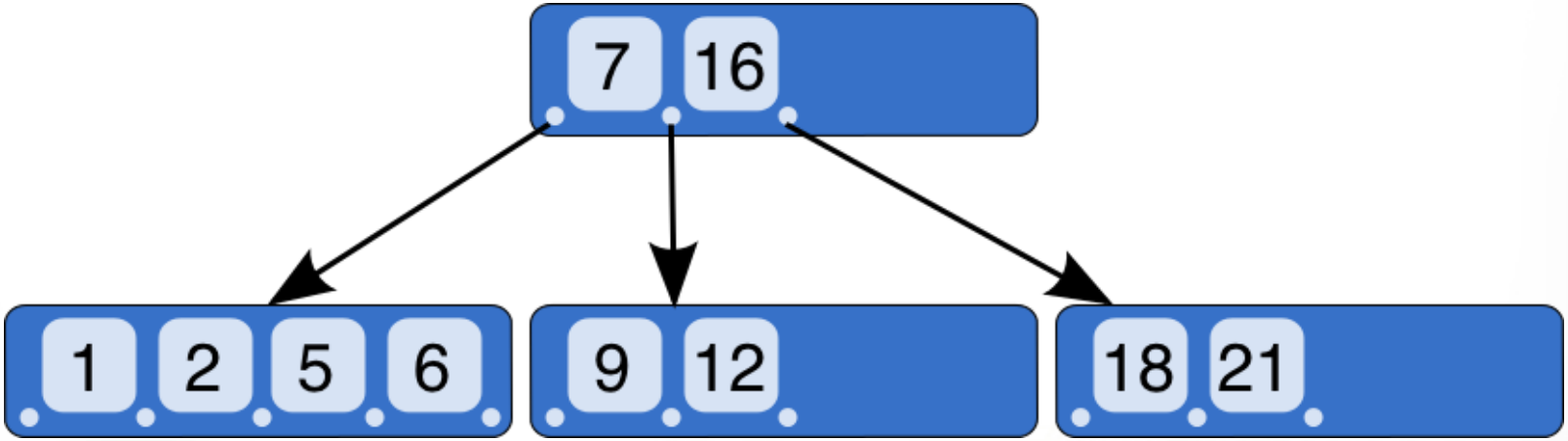
```
mysql> alter table City add key (Name);
Query OK, 4079 rows affected (0.02 sec)
Records: 4079  Duplicates: 0  Warnings: 0

mysql> explain select * from City where Name = 'London'\G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: City
         type: ref
possible_keys: Name
          key: Name
      key_len: 35
          ref: const
         rows: 1
        Extra: Using where
```



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

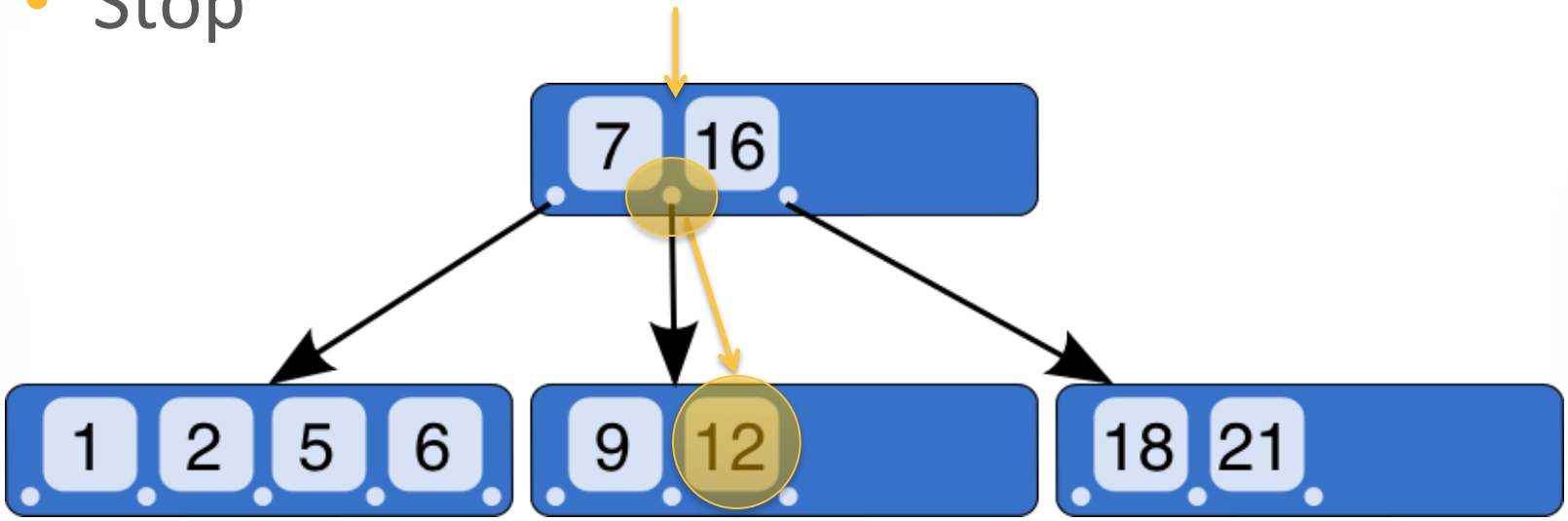


http://en.wikipedia.org/wiki/B-tree

MySQL Index Types: B-Tree

Equality search: select * from table where id = 12

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

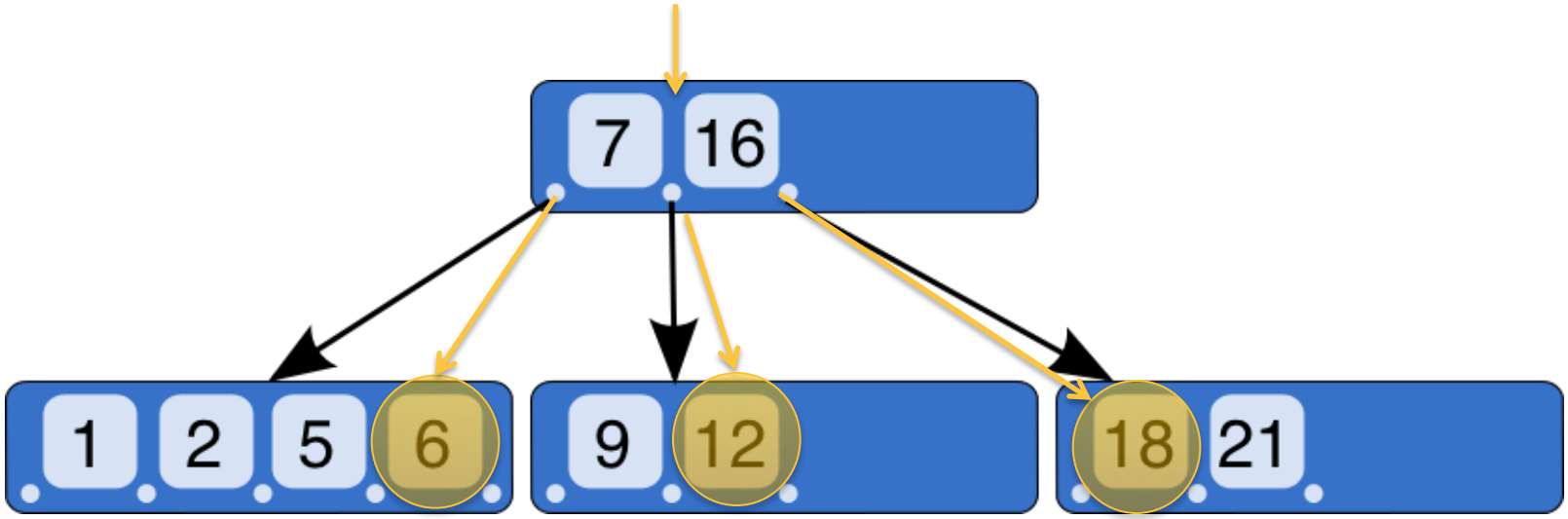


http://en.wikipedia.org/wiki/B-tree

MySQL Index Types: B-Tree

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

- Scan thru the tree and visit many leafs/nodes



http://en.wikipedia.org/wiki/B-tree

Indexes: Theory

- MySQL choose 1 (best) index per table
  - With some exceptions...
- Supports combined indexes
- Order of fields inside combined index matters
- MySQL can use leftmost part of any index
- MySQL can use index to satisfy ORDER BY/ GROUP BY
  - With some limitations

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
```

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)`

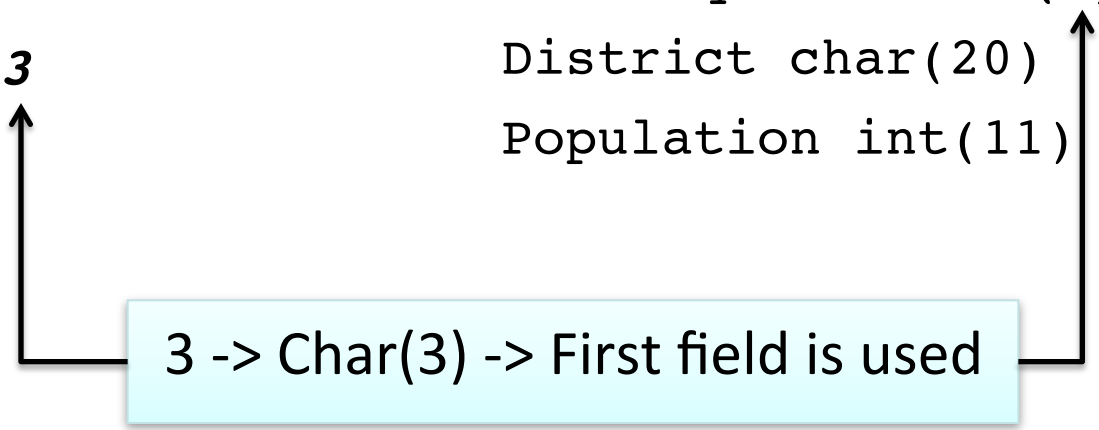
Explain:

key: comb

key_len: 3

Fields:

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



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

Index Cardinality

Cardinality = number of unique values

- Check the number of unique values
  - mysql> show keys from table;
- Higher cardinality = better!
- Primary key = best!
- Fields with 2 unique values *may* not be good candidates for index
  - “status”, “gender”, etc
  - Unless you do select count(*) where status = 1

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

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

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);

explain select name from City  where
district = 'California' and population > 3000\G
*********************** 1. row ***************************
           table: City
            type: ALL
possible_keys: comb2
             key: NULL
         key_len: NULL
             ref: NULL
            rows: 4162
           Extra: Using where
```

BAD! MySQL decided not to use index at all

Why?

MySQL can only use “population” part

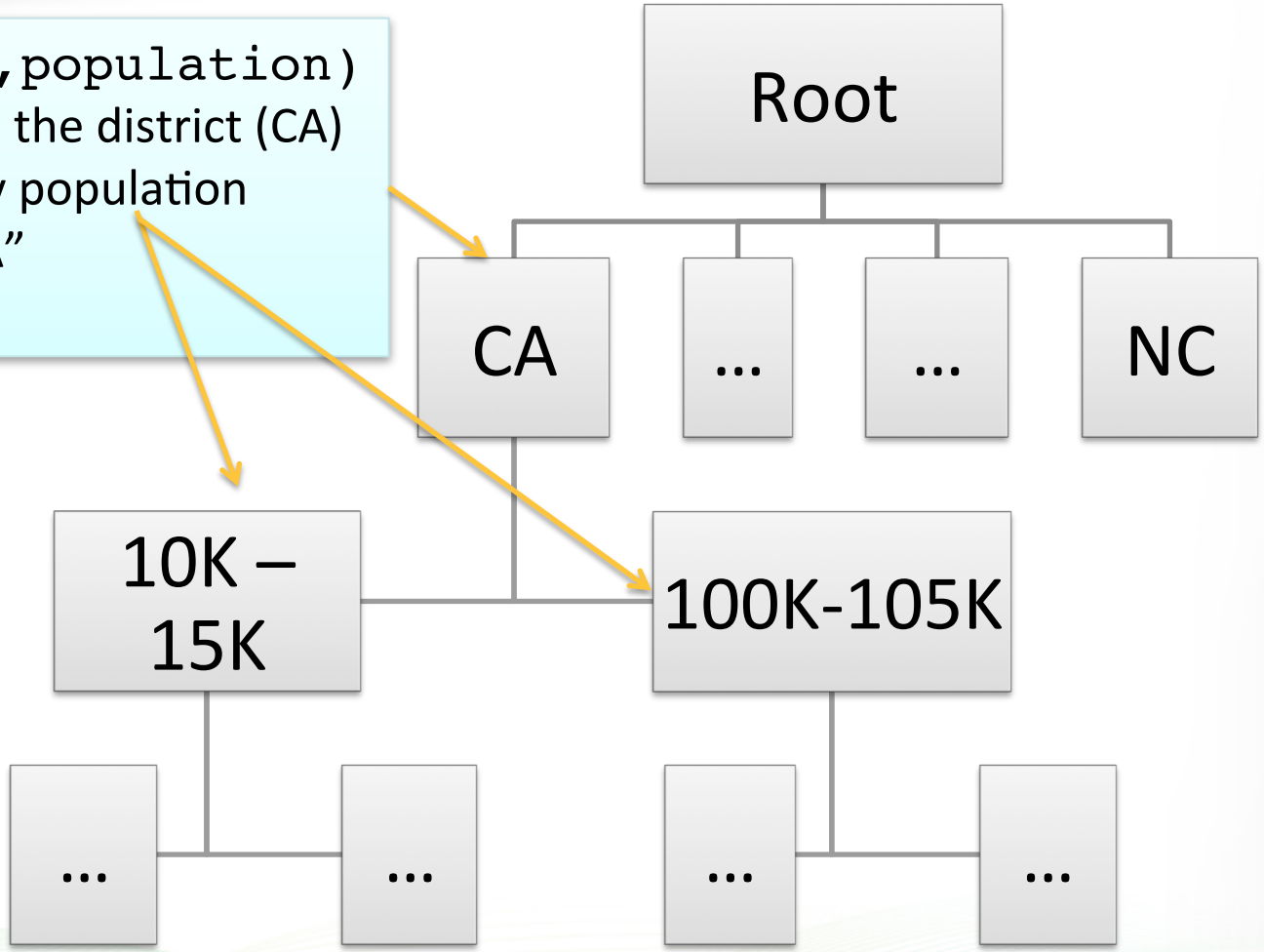
Too many cities with population > 3000

Simplified BTree Scan Example I

`Comb1(`***district***`,population)`

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

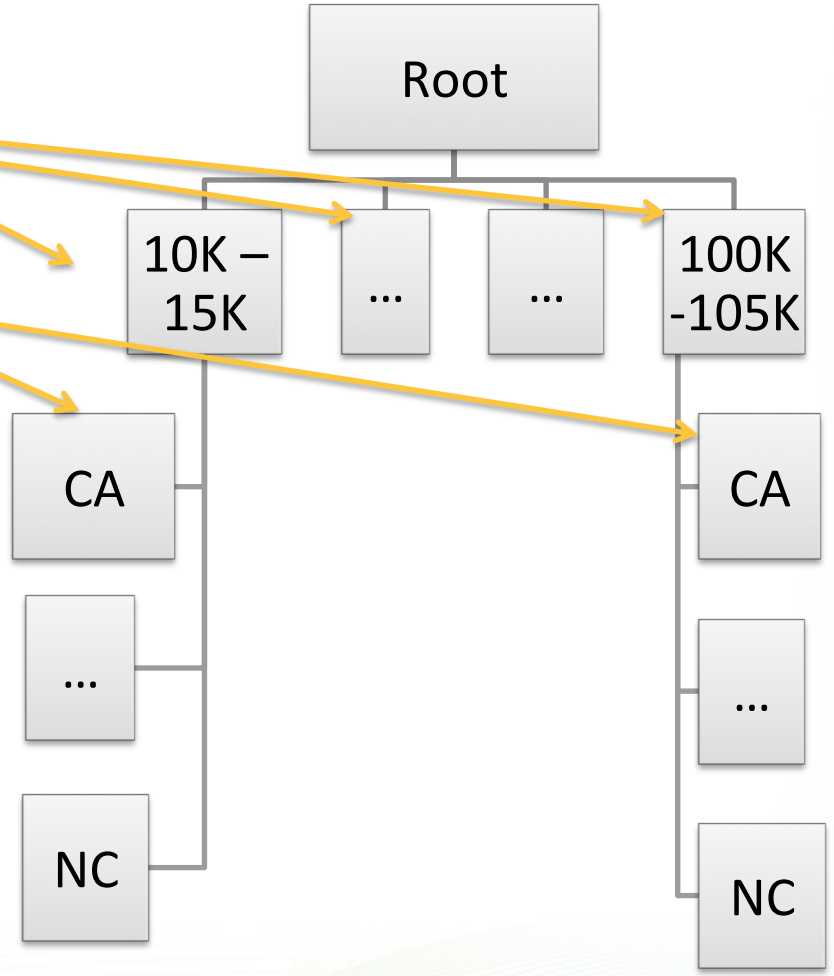
*via index scan



Simplified BTree Scan Example II

`Comb1(`***`population`***`,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



Index Cardinality: Example

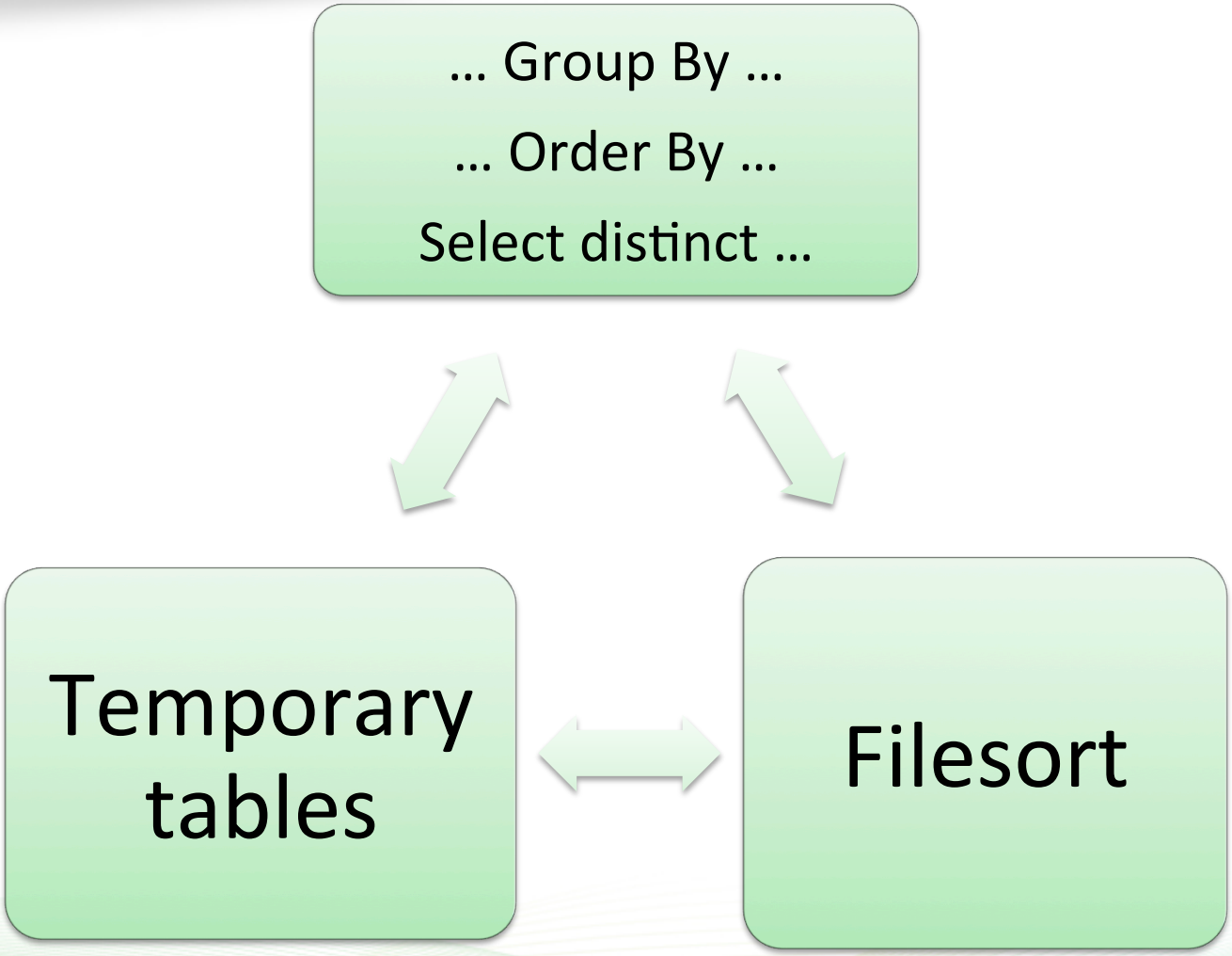
```
mysql> alter table City
add key comb2(population, District);

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



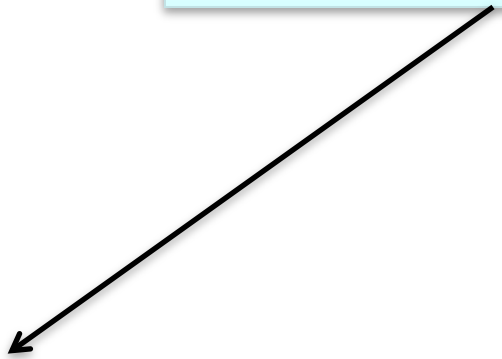
GROUP BY Queries

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

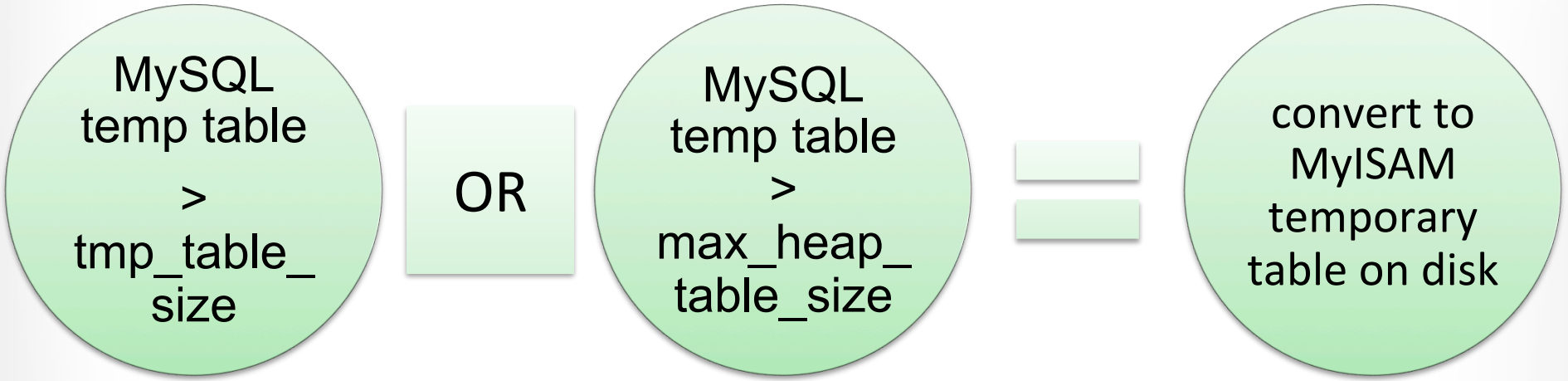
- MySQL can create temporary tables when query uses:
  - GROUP BY
  - Range + ORDER BY
  - Some other expressions

- 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

- ***Percona server uses the new MEMORY engine with BLOB/TEXT Support***
- ***BUT: it is not used for the temp tables***

Temporary Tables: Profiling

- Watch those status variables:
  - **Created_tmp_tables –** number of temporary table MySQL created in both *RAM and DISK*
  - **Created_tmp_disk_tables -** number of temporary table MySQL created on *DISK*

Temporary Tables: Profiling

```
mysql> show session status like 'created%';
+-------------------------+-------+
| Variable_name           | Value |
+-------------------------+-------+
| Created_tmp_disk_tables | 1     |
...
| Created_tmp_tables      | 10    |
+-------------------------+-------+
3 rows in set (0.00 sec)
```

Temporary Tables: Practice

Air Traffic Statistics Table for Testing

5M rows, ~2G in size

```
CREATE TABLE `ontime_2010` (
  `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

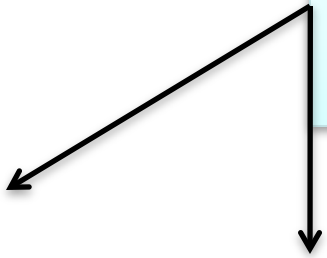
- Find maximum delay for flights on Sunday
- Group by airline

```
SELECT max(DepDelayMinutes),
carrier, dayofweek
FROM ontime_2010
WHERE dayofweek = 7
GROUP BY Carrier
```

GROUP BY Query Example

```
select max(DepDelayMinutes),  carrier, dayofweek
from ontime_2010
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
```



Full table scan!
Temporary table!

Adding Index: Fixing Full Table Scan


```
mysql> alter table ontime_2010 add key (dayofweek);
```

```
explain select max(DepDelayMinutes), Carrier,
dayofweek from ontime_2010 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!

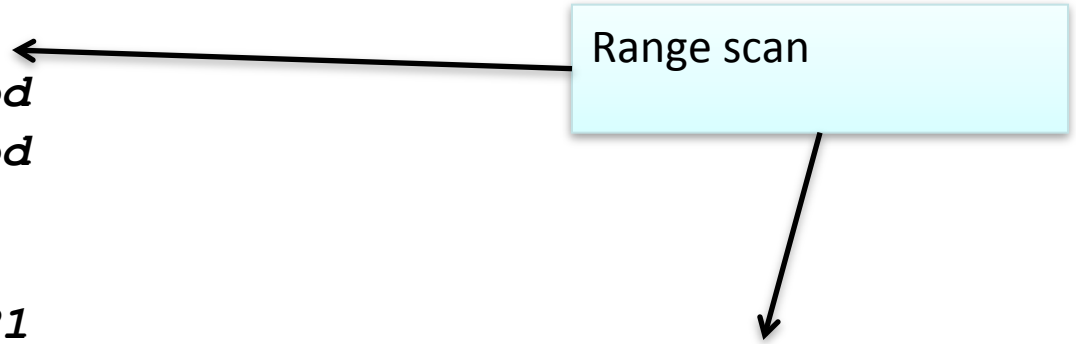
GROUP BY: Adding Covered Index


```
mysql> alter table ontime_2010 add key covered(dayofweek,
Carrier, DepDelayMinutes);
explain select max(DepDelayMinutes), Carrier, dayofweek from
ontime_2010 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

When Covered Indexes Aren't Good …

```
mysql> explain select max(DepDelayMinutes), Carrier,
dayofweek from ontime_2010
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
```



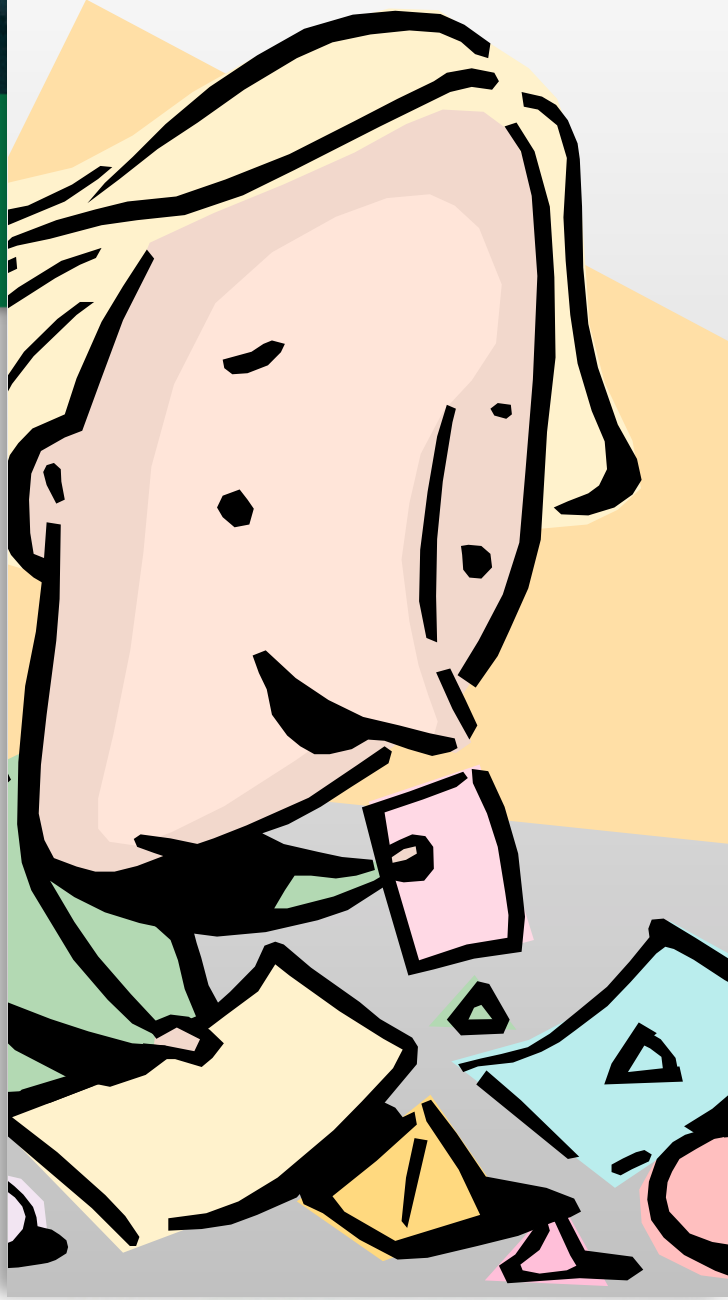
Converting query to Union


```
(select max(DepDelayMinutes), Carrier, dayofweek
from ontime_2010
where dayofweek = 3
group by Carrier, dayofweek)
union
(select max(DepDelayMinutes), Carrier, dayofweek
from ontime_2010
where dayofweek = 4
group by Carrier, dayofweek)
```

Converting query to Union


```
*************************** 1. row ***************************
        table: ontime_2010
          key: covered
...
        Extra: Using where; Using index
*************************** 2. row ***************************
        table: ontime_2010
          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
```

ORDER BY and filesort



ORDER BY and filesort

Find 10 cities in the US with the largest population

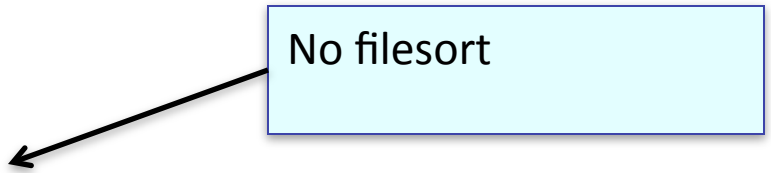
```
mysql> explain select district, name, population from City
where CountryCode = 'USA' order by population desc limit 10\G
```

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

MySQL 5.6:
Faster Sort with Limit

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
```

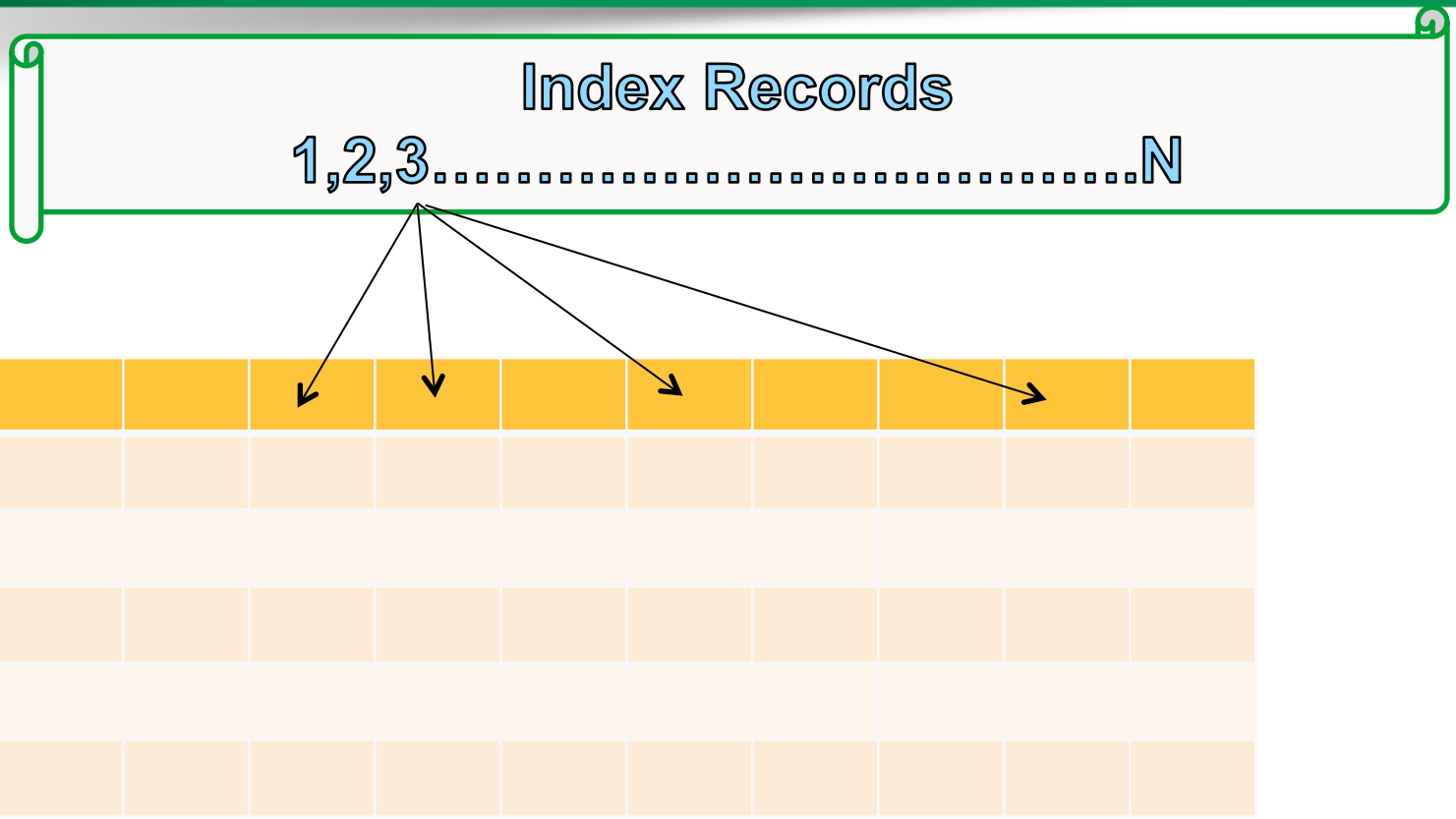


Sorting and Limit

```
mysql> alter table ontime_2010 add key (DepDelayMinutes);
Query OK, 0 rows affected (38.68 sec)

mysql> explain select * from ontime_2010
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)
```

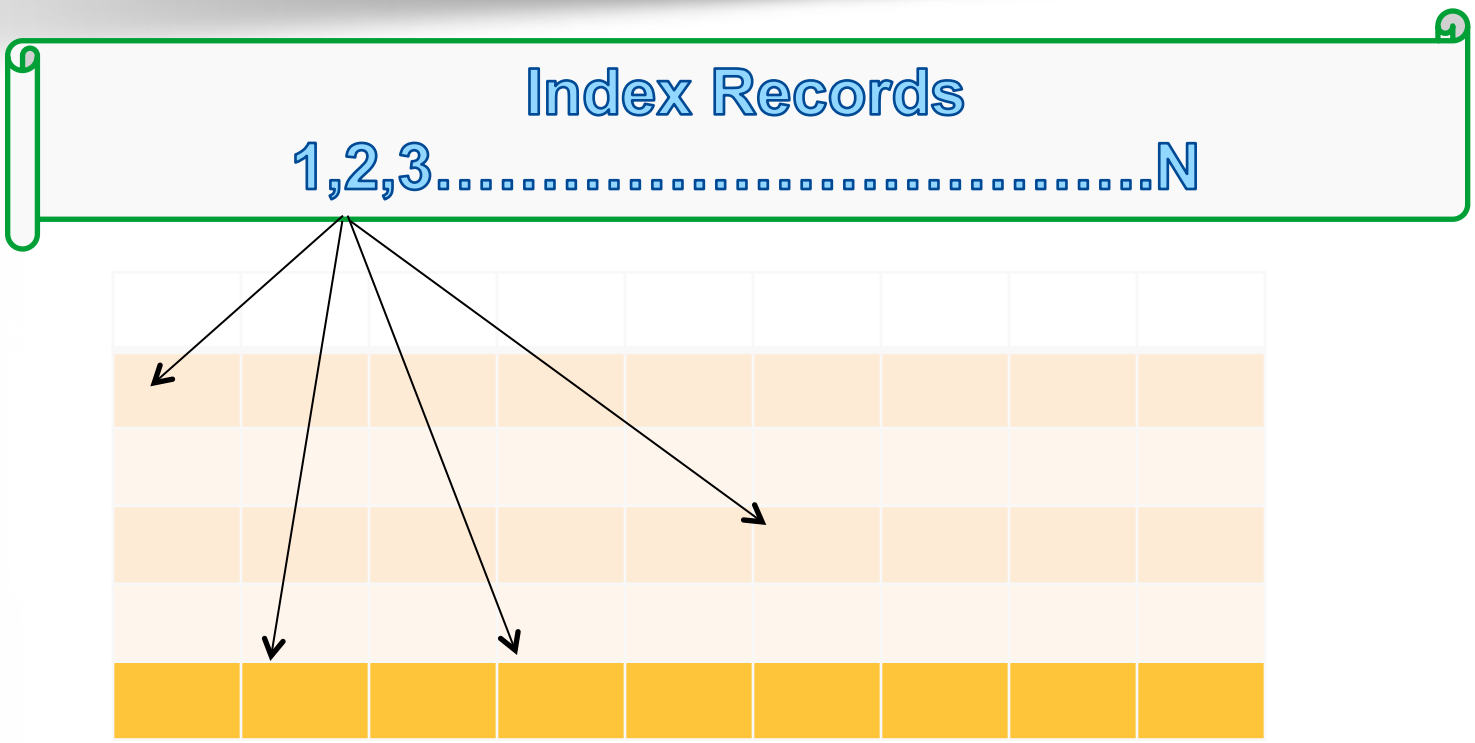
1. Index is sorted
2. Scan the **whole table** in the order of the index
3. Filter results
4. Stop after finding 10 rows matching the “where” condition

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)

Sub-queries Optimizations

Subqueries, I

- Subquery inside where

```
SELECT * FROM t1 WHERE
column1 in (SELECT column2 FROM t2);
```

- Subquery in FROM and joins

```
SELECT sb1,sb2,sb3 FROM (SELECT s1 AS sb1, s2
  AS sb2, s3*2 AS sb3 FROM t1) AS sb
WHERE sb1 > 1;
```

Subqueries, II

MySQL 5.6:
resolved,
semi-join subquery
optimization strategies

- Subquery inside where

```
SELECT * FROM t1 WHERE
column1 in
(SELECT column2 FROM t2 where ..);
```

- Will not use index on **column1**
  - http://bugs.mysql.com/bug.php?id=8139
  - http://bugs.mysql.com/bug.php?id=9021
- Can rewrite query as join
- http://dev.mysql.com/doc/refman/5.6/en/rewriting-subqueries.html

Subqueries, IV

- Subquery in FROM and joins

```
SELECT sb1,sb2,sb3 FROM (SELECT s1 AS sb1, s2
  AS sb2, s3*2 AS sb3 FROM t1) AS t2
JOIN t1.id = t2.id WHERE sb1 > 1;
```

- MySQL will create a temporary table for `(SELECT s1 AS sb1, s2 AS sb2, s3*2 AS sb3 FROM t1)` with no indexes
- (fixed in MySQL 5.6)
- Rewrite as join
  - http://dev.mysql.com/doc/refman/5.6/en/rewriting-subqueries.html

MySQL 5.6: Resolved
Adds an index on the derived table

Reporting Queries

Reporting Example: Table

```
mysql> CREATE TABLE `lineitem` (
  `l_shipdate` date NOT NULL,
  `l_orderkey` int(11) NOT NULL,
  `l_partkey` int(11) NOT NULL,
  `l_suppkey` int(11) NOT NULL,
  `l_linenumber` int(11) NOT NULL,
  `l_quantity` decimal(15,2) NOT NULL,
  `l_extendedprice` decimal(15,2) NOT NULL,
  `l_discount` decimal(15,2) NOT NULL,
  `l_tax` decimal(15,2) NOT NULL,
  `l_returnflag` char(1) NOT NULL,
  `l_linestatus` char(1) NOT NULL,
  KEY `lineitem_fk2` (`l_suppkey`),
  KEY `lineitem_fk3` (`l_partkey`,`l_suppkey`),
  KEY `li_shp_dt_idx` (`l_shipdate`),
  KEY `li_com_dt_idx` (`l_commitdate`),
  KEY `li_rcpt_dt_idx` (`l_receiptdate`)  ) ENGINE=InnoDB;
```

Reporting Example: Query

Group by year(date)

```
mysql> explain select sum(l_extendedprice), year(l_shipdate) as yr
from lineitem group by yr limit 10\G
*************************** 1. row
           id: 1
  select_type: SIMPLE
        table: lineitem
         type: ALL
possible_keys: NULL
          key: NULL
      key_len: NULL
          ref: NULL
         rows: 116771866
        Extra: Using temporary; Using filesort
```

year(field) = calculated
MySQL can't use index

Adding Year/Month

```
mysql> alter table lineitem add yr year, add key(yr);
mysql> update lineitem set yr = year(l_shipdate);
mysql> explain  select sum(l_extendedprice),  yr
from lineitem  group by yr desc limit 10\G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: lineitem
         type: index
possible_keys: NULL
          key: yr
      key_len: 2
          ref: NULL
         rows: 116771866
        Extra:
```

No temporary, no filesort
Add covered index for better performance

Summary Tables, I

```
mysql> create table lineitem_summary as
select year(l_shipdate) as yr,
month(l_shipdate) as mon,
sum(l_extendedprice) as revenue,
count(*) as num_orders
from lineitem group by yr, mon;
```

Data is already aggregated in summary table

Summary Tables, II

- Aggregate by Year, based on summary table

```
mysql> select yr,
sum(l_extendedprice) as revenue,
count(*) as num_orders
from lineitem_summary group by yr;
```

Data already aggregated
Small number of records
= Queries are much faster!

Summary Tables, III

Advantages

- Significantly faster for queries
- Smaller number of rows

Disadvantages

- Needs to be updated: cron or manually
- More data to store

Make sense for reporting

Conclusion, I

- Monitor your queries
- Explain/Profile queries

Conclusion, II

- Query tuning
  - Add indexes to speed-up queries
  - Avoid things, which are not optimal in MySQL
  - Use Summary tables

Questions?

Thank you!