Optimizing MySQL Joins and Subqueries

http://bit.ly/2012optmysql

Sheeri Cabral

Senior DB Admin/Architect, Mozilla

@sheeri www.sheeri.com

Northeast PHP 2012

EXPLAIN

SQL extension

SELECT only

Can modify other statements:

```
UPDATE tbl SET fld1=“foo” WHERE fld2=“bar”;
```

can be changed to:

```
EXPLAIN SELECT fld1 FROM tbl WHERE fld2=”bar”;
```

What EXPLAIN Shows

How many tables

How tables are joined

How data is looked up

If there are subqueries, unions, sorts

What EXPLAIN Shows

If WHERE, DISTINCT are used

Possible and actual indexes used

Length of index used

Approx # of records examined

Metadata

Optimizer uses metadata: cardinality, # rows, etc.

InnoDB - approx stats

InnoDB - one method of doing dives into the data

MyISAM has better/more accurate metadata

EXPLAIN Output

`EXPLAIN` returns 10 fields:

```
mysql> EXPLAIN SELECT return_date
    -> FROM rental WHERE rental_id = 13534\G
******************* 1. row *******************
           id: 1
  select_type: SIMPLE
        table: rental
         type: const
possible_keys: PRIMARY
          key: PRIMARY
      key_len: 4
          ref: const
         rows: 1
        Extra:
1 row in set (0.00 sec)
```

Id

```
mysql> EXPLAIN SELECT return_date

    -> FROM rental WHERE rental_id = 13534\G
******************* 1. row *******************
           id: 1
```

Id = sequential identifier

One per table, subquery, derived table

No row returned for a view

- Because it is virtual

- Underlying tables are represented

select_type

```
mysql> EXPLAIN SELECT return_date
    -> FROM rental WHERE rental_id = 13534\G
******************* 1. row *******************
           id: 1
  select_type: SIMPLE
```

SIMPLE – one table, or JOINs

PRIMARY

- First SELECT in a UNION
- Outer query of a subquery

UNION, UNION RESULT

Other select_type output

Used in subqueries

- More on subqueries later

```
DEPENDENT UNION

DEPENDENT SUBQUERY

DERIVED

UNCACHEABLE SUBQUERY
```

table

```
mysql> EXPLAIN SELECT return_date
    -> FROM rental WHERE rental_id = 13534\G
******************* 1. row *******************
           id: 1
  select_type: SIMPLE
        table: rental
```

- One per table/alias
- NULL

NULL table

```
EXPLAIN SELECT 1+2\G

EXPLAIN SELECT return_date FROM rental WHERE
rental_id=0\G
```

type

```
mysql> EXPLAIN SELECT return_date
    -> FROM rental WHERE rental_id = 13534\G
******************* 1. row *******************
           id: 1
  select_type: SIMPLE
        table: rental
         type: const
```

“Data access method”

Get this as good as possible

type

ALL = full table scan

- Everything else uses an index

index = full index scan

- Scanning the entire data set?
- full index scan > full table scan (covering index)

range = partial index scan

- <, <=, >, >=
- IS NULL, BETWEEN, IN

type

index_subquery

- using a non-unique index of one table

unique subquery

- using a PRIMARY/UNIQUE KEY of one table

More about subqueries later

type

index_merge

- Use more than one index
- Extra field shows more information
  - sort_union, intersection, union

ref_or_null

Joining/looking up non-unique index values

JOIN uses a non-unique index or key prefix

Indexed fields compared with = != <=>

Extra pass for possible NULL values

ref

Joining/looking up non-unique index values

JOIN uses a non-unique index or key prefix

Indexed fields compared with = != <=>

No NULL value possibilities

Best data access strategy for non-unique values

eq_ref

Joining/looking up unique index values

JOIN uses a unique index or key prefix

Indexed fields compared with =

Fastest Data Access

Joining/looking up unique index values

```
SELECT return_date
FROM rental WHERE rental_id=13534;
```

System – system table, one value

EXPLAIN Plan indexes

possible_keys

key

key_len – longer keys = longer look up/ compare

ref – shows what is compared, field or “const”

Look closely if an index is not considered

Approx # rows examined

```
mysql> EXPLAIN SELECT return_date
    -> FROM rental WHERE rental_id = 13534\G
******************* 1. row *******************
           id: 1
  select_type: SIMPLE
        table: rental
         type: const
possible_keys: PRIMARY
          key: PRIMARY
      key_len: 4
          ref: const
         rows: 1
        Extra:
1 row in set (0.00 sec)
```

Approx # rows examined

```
mysql> EXPLAIN SELECT first_name,last_name FROM
customer LIMIT 10\G

*************** 1. row *****************
           id: 1
  select_type: SIMPLE
        table: customer
         type: ALL
possible_keys: NULL
          key: NULL
      key_len: NULL
          ref: NULL
         rows: 541
        Extra:
1 row in set (0.00 sec)
```

LIMIT does not change rows, even though it affects # rows examined.

Extra

Can be good, bad, neutral

- Sometimes you cannot avoid the bad

Distinct – stops after first row match

Full scan on NULL key – subquery, no index (bad)

Impossible WHERE noticed after reading const tables

Extra

Not exists – stops after first row match for each row set from previous tables

Select tables optimized away – Aggregate functions resolved by index or metadata (good)

Range checked for each record (index map: N)

- No good index; may be one after values from previous tables are known

Extra: Using (...)

Extra: Using filesort – does an extra pass to sort the data.

- Worse than using an index for sort order.

Index – uses index only, no table read

- Covering index

Index for group-by

- GROUP BY/DISTINCT resolved by index/metadata

Temporary

- Intermediate temporary table used

More EXPLAIN Information

MySQL Manual

http://www.pythian.com/news/wp-content/uploads/explain-diagram.pdf

Pages 590 – 614 of the MySQL Administrator's Bible

Sakila sample database: http://dev.mysql.com/doc/index-other.html

Sample Subquery EXPLAIN

```
mysql> EXPLAIN SELECT first_name,last_name,email
    -> FROM customer AS customer_outer
    -> WHERE customer_outer.customer_id
    -> IN (SELECT customer_id FROM rental AS rental_subquery
WHERE return_date IS NULL)\G

 ********* 1. row ********    ********** 2. row **********
          id: 1                          id: 2
   select_type: PRIMARY          select_type: DEPENDENT SUBQUERY
         table: customer_outer         table: rental_subquery
          type: ALL                     type: index_subquery
 possible_keys: NULL           possible_keys: idx_fk_customer_id
           key: NULL                     key: idx_fk_customer_id
       key_len: NULL                 key_len: 2
           ref: NULL                     ref: func
          rows: 541                     rows: 13
         Extra:                        Extra: Using where; Full
                               scan on NULL key
                               2 rows in set (0.00 sec)
```

MySQL and Subqueries

Avoid **unoptimized** subqueries

- Not all subqueries...any more

Derived tables → views or intermediate temp tbls

Subqueries → joins in some cases

Getting better all the time

- Optimized in MariaDB 5.3

MySQL Does Not Have

Materialized views

Materialized derived tables

Functional indexes (e.g. `WHERE date(ts)=2012_05_30`)

Convert a Subquery to a JOIN

```
SELECT first_name,last_name,email
IN (SELECT customer_id FROM rental AS rental_subquery WHERE
return_date IS NULL)
FROM customer AS customer_outer\G
```

Convert a Subquery to a JOIN

```
SELECT first_name,last_name,email
IN (SELECT customer_id FROM rental AS rental_subquery WHERE
return_date IS NULL)
FROM customer AS customer_outer\G
```

Think in data sets

Convert a Subquery to a JOIN

```
SELECT first_name,last_name,email
IN (SELECT customer_id FROM rental AS rental_subquery WHERE
return_date IS NULL)
FROM customer AS customer_outer\G
```

Think in data sets

```
SELECT first_name,last_name, email
FROM rental INNER JOIN customer
ON (customer.id=rental.customer_id)
WHERE return_date IS NULL
```

Convert a Subquery to a JOIN

```
SELECT first_name,last_name,email
IN (SELECT customer_id FROM rental AS rental_subquery WHERE
return_date IS NULL)
FROM customer AS customer_outer\G
```

Think in data sets

```
SELECT first_name,last_name, email
FROM rental INNER JOIN customer
ON (customer.id=rental.customer_id)
WHERE return_date IS NULL
```

Note the ANSI-style JOIN clause

Explicit declaration of JOIN conditions

Do not use theta-style implicit JOIN conditions in WHERE

ANSI vs. Theta JOINs

```
SELECT first_name,last_name, email
FROM rental INNER JOIN customer
ON (customer.id=rental.customer_id)
WHERE return_date IS NULL
```

```
SELECT first_name,last_name, email
FROM rental INNER JOIN customer
WHERE return_date IS NULL
AND customer.id=rental.customer_id
```

INNER JOIN, CROSS JOIN, JOIN are the same

Don't use a comma join (FROM rental,customer)

A Correlated Subquery

Show the last payment info for each customer:

For each customer, find the max payment date, then get that info

```
SELECT pay_outer.* FROM payment pay_outer
WHERE pay_outer.payment_date =
(SELECT MAX(payment_date)
FROM payment pay_inner
WHERE pay_inner.customer_id=pay_outer.customer_id)
```

EXPLAIN

```
SELECT pay_outer.* FROM payment pay_outer
WHERE pay_outer.payment_date =
(SELECT MAX(payment_date)
FROM payment pay_inner
WHERE pay_inner.customer_id=pay_outer.customer_id)
```

```
********************* 1. row *********************
           id: 1
  select_type: PRIMARY
        table: pay_outer
         type: ALL
possible_keys: NULL
          key: NULL
      key_len: NULL
          ref: NULL
         rows: 16374
        Extra: Using where
```

```
******************* 2. row *******************
           id: 2
  select_type: DEPENDENT SUBQUERY
        table: pay_inner
         type: ref
possible_keys: idx_fk_customer_id
          key: idx_fk_customer_id
      key_len: 2
          ref: sakila.pay_outer.customer_id
         rows: 14
        Extra:
2 rows in set (0.00 sec)
```

Think in Terms of Sets

Show the last payment info for each customer:

Set of last payment dates, set of all payment info, join the sets

```
SELECT payment.* FROM
(SELECT customer_id, MAX(payment_date) as last_order
FROM payment
GROUP BY customer_id) AS last_orders
INNER JOIN payment
ON payment.customer_id = last_orders.customer_id
AND payment.payment_date = last_orders.last_order\G
```

EXPLAIN

```
EXPLAIN SELECT payment.* FROM
(SELECT customer_id, MAX(payment_date) as last_order
FROM payment GROUP BY customer_id) AS last_orders
INNER JOIN payment
ON payment.customer_id = last_orders.customer_id
AND payment.payment_date = last_orders.last_order\G
```

```
********** 1. row **********
         id: 1
  select_type: PRIMARY
      table: <derived2>
       type: ALL
possible_keys: NULL
        key: NULL
     key_len: NULL
        ref: NULL
       rows: 599
      Extra:
```

```
************** 2. row **************
            id: 1
   select_type: PRIMARY
         table: payment
          type: ref
possible_keys:
idx_fk_customer_id,customer_id_pay
           key: customer_id_pay
        key_len: 10
           ref: last_orders.customer_id,
last_orders.last_order
          rows: 1
         Extra:
```

```
************ 3. row ************
            id: 2
  select_type: DERIVED
        table: payment
         type: range
possible_keys: NULL
           key: customer_id
      key_len: 2
           ref: NULL
         rows: 1301
        Extra: Using index for
group-by
3 rows in set (0.01 sec)
```

```
************ 1. row ************
           id: 1
  select_type: PRIMARY
        table: pay_outer
         type: ALL
possible_keys: NULL
          key: NULL
      key_len: NULL
          ref: NULL
         rows: 16374
        Extra: Using where
******************** 2. row ********************
           id: 2
  select_type: DEPENDENT SUBQUERY
        table: pay_inner
         type: ref
possible_keys: idx_fk_customer_id
          key: idx_fk_customer_id
      key_len: 2
          ref: sakila.pay_outer.customer_id
         rows: 14
        Extra:
2 rows in set (0.00 sec)
```

```
********** 1. row **********
           id: 1
  select_type: PRIMARY
        table: <derived2>
         type: ALL
possible_keys: NULL
          key: NULL
      key_len: NULL
          ref: NULL
         rows: 599
        Extra:
************** 2. row **************
           id: 1
  select_type: PRIMARY
        table: payment
         type: ref
possible_keys:
idx_fk_customer_id,customer_id_pay
          key: customer_id_pay
      key_len: 10
          ref: last_orders.customer_id,
last_orders.last_order
         rows: 1
        Extra:
************ 3. row ************
           id: 2
  select_type: DERIVED
        table: payment
         type: range
possible_keys: NULL
          key: customer_id
      key_len: 2
          ref: NULL
         rows: 1301
        Extra: Using index for
group-by
3 rows in set (0.01 sec)
```

Join-fu

http://joinfu.com/presentations/joinfu/joinfu_part_one.pdf

- p 22, mapping tables

http://joinfu.com/presentations/joinfu/joinfu_part_two.pdf

- heirarchies/graphs/nested sets

- GIS calculations

- reporting/aggregates/ranks

With thanks to Jay Pipes!

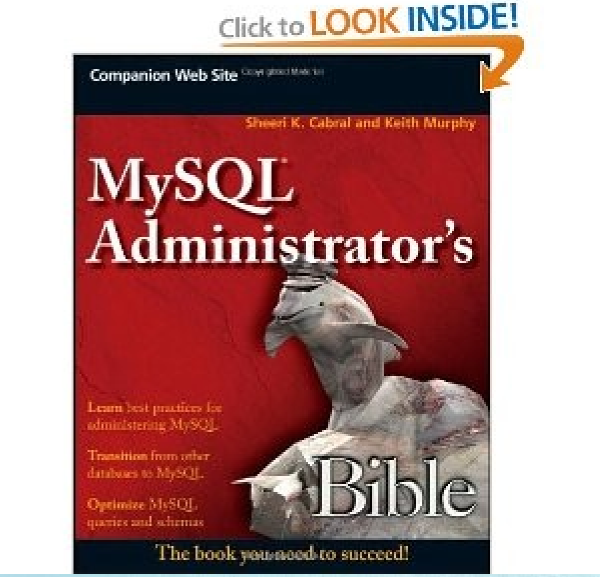
Questions? Comments?

OurSQL Podcast

www.oursql.com

MySQL Administrator's Bible

- tinyurl.com/mysqlbible


Click to LOOK INSIDE!
Companion Web Site
Sheeri K. Cabral and Keith Murphy
MySQL Administrator's Bible
Learn best practices for administering MySQL.
Transition from other databases to MySQL.
Optimize MySQL queries and schemas
The book you need to succeed!

kimtag.com/mysql

planet.mysql.com

Optimizing MySQL Joins and Subqueries

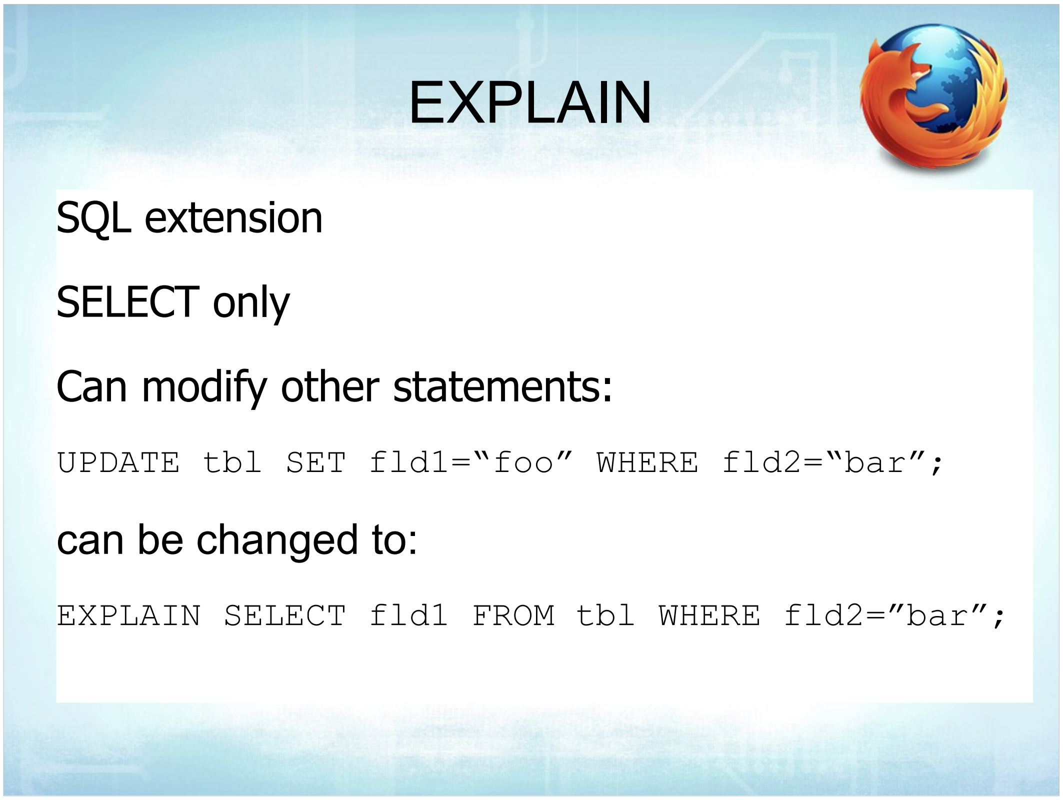
http://bit.ly/2012optmysql

Sheeri Cabral

Senior DB Admin/Architect, Mozilla

@sheeri www.sheeri.com

Northeast PHP 2012



EXPLAIN

SQL extension

SELECT only

Can modify other statements:

```
UPDATE tbl SET fld1=“foo” WHERE fld2=“bar”;
```

can be changed to:

```
EXPLAIN SELECT fld1 FROM tbl WHERE fld2=”bar”;
```

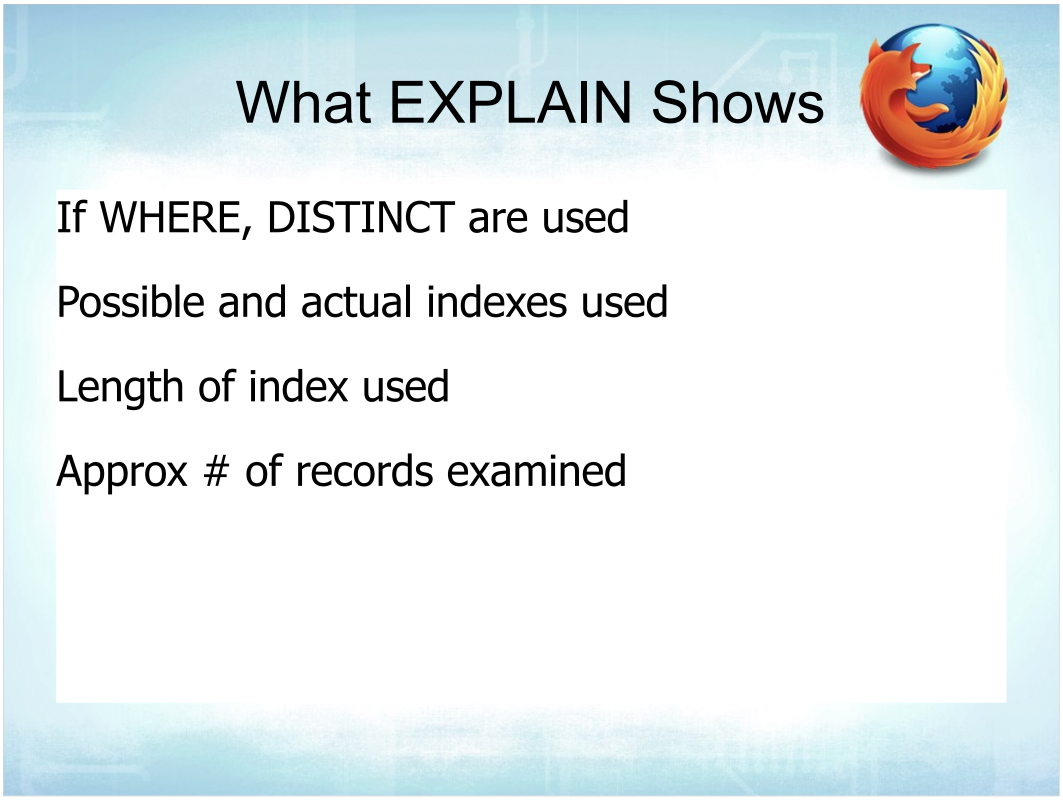
What EXPLAIN Shows

How many tables

How tables are joined

How data is looked up

If there are subqueries, unions, sorts



What EXPLAIN Shows

If WHERE, DISTINCT are used

Possible and actual indexes used

Length of index used

Approx # of records examined

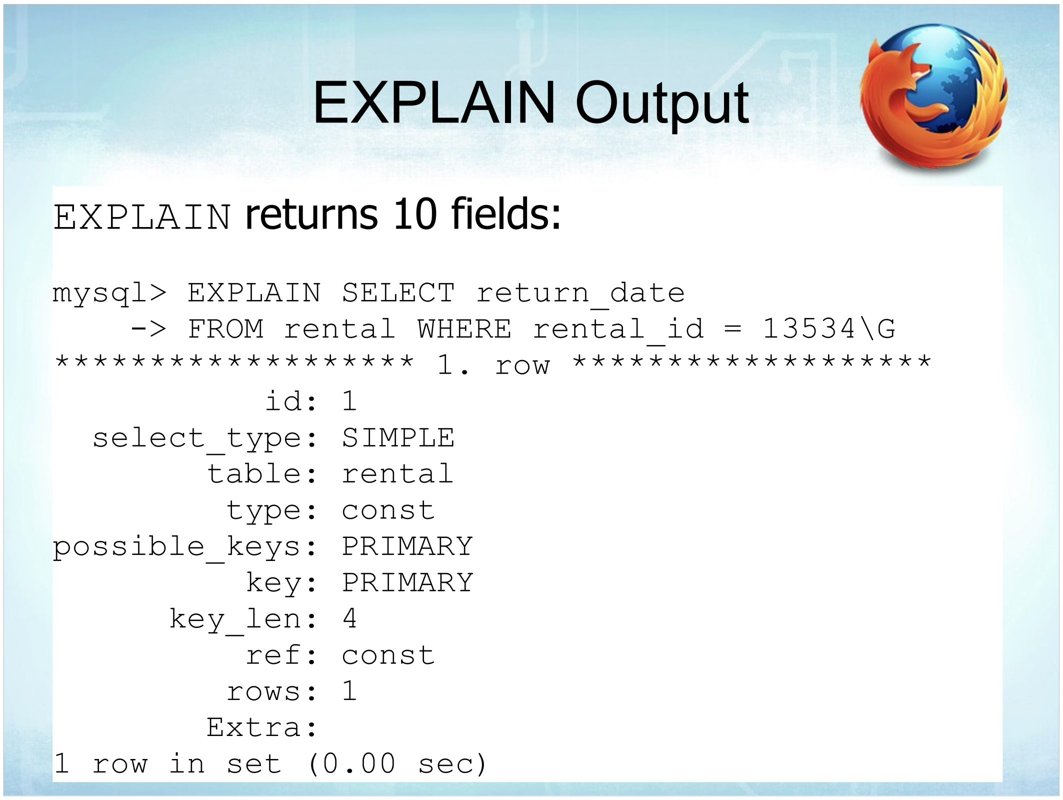
Metadata

Optimizer uses metadata: cardinality, # rows, etc.

InnoDB - approx stats

InnoDB - one method of doing dives into the data

MyISAM has better/more accurate metadata

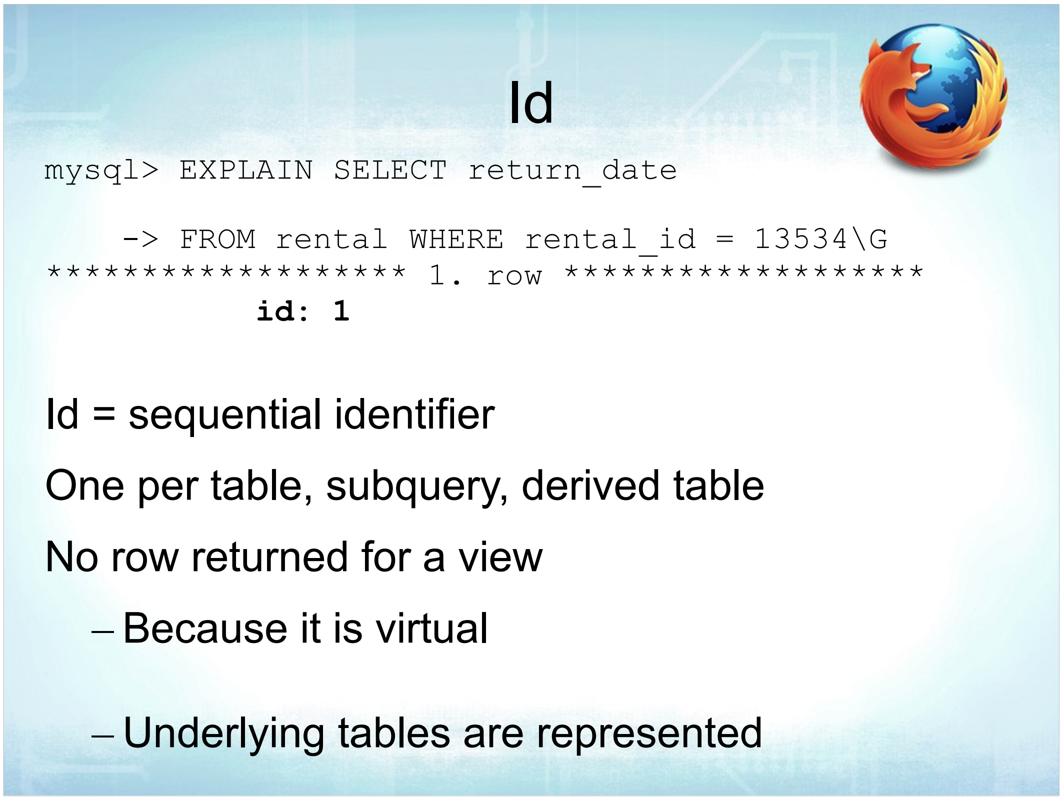


EXPLAIN Output

`EXPLAIN` returns 10 fields:

```
mysql> EXPLAIN SELECT return_date
    -> FROM rental WHERE rental_id = 13534\G
******************* 1. row *******************
           id: 1
  select_type: SIMPLE
        table: rental
         type: const
possible_keys: PRIMARY
          key: PRIMARY
      key_len: 4
          ref: const
         rows: 1
        Extra:
1 row in set (0.00 sec)
```

One row per table.



Id

```
mysql> EXPLAIN SELECT return_date

    -> FROM rental WHERE rental_id = 13534\G
******************* 1. row *******************
           id: 1
```

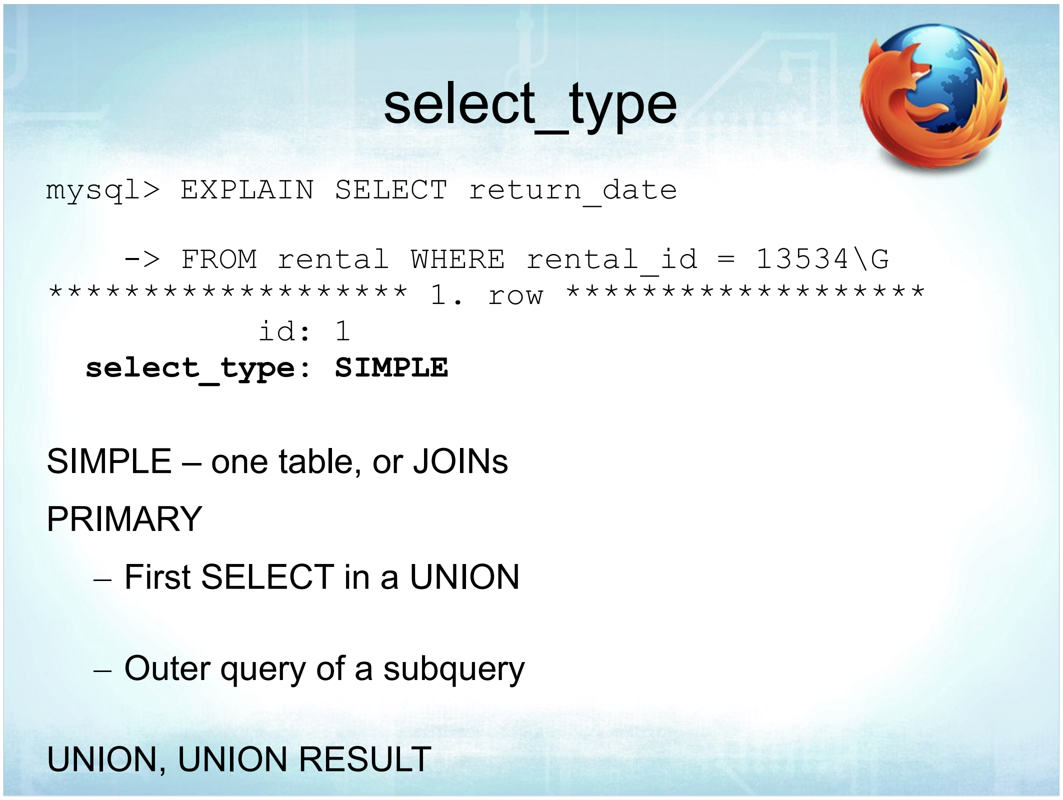
Id = sequential identifier

One per table, subquery, derived table

No row returned for a view

- Because it is virtual
- Underlying tables are represented

One row per table.



select_type

```
mysql> EXPLAIN SELECT return_date

    -> FROM rental WHERE rental_id = 13534\G
******************* 1. row *******************
           id: 1
  select_type: SIMPLE
```

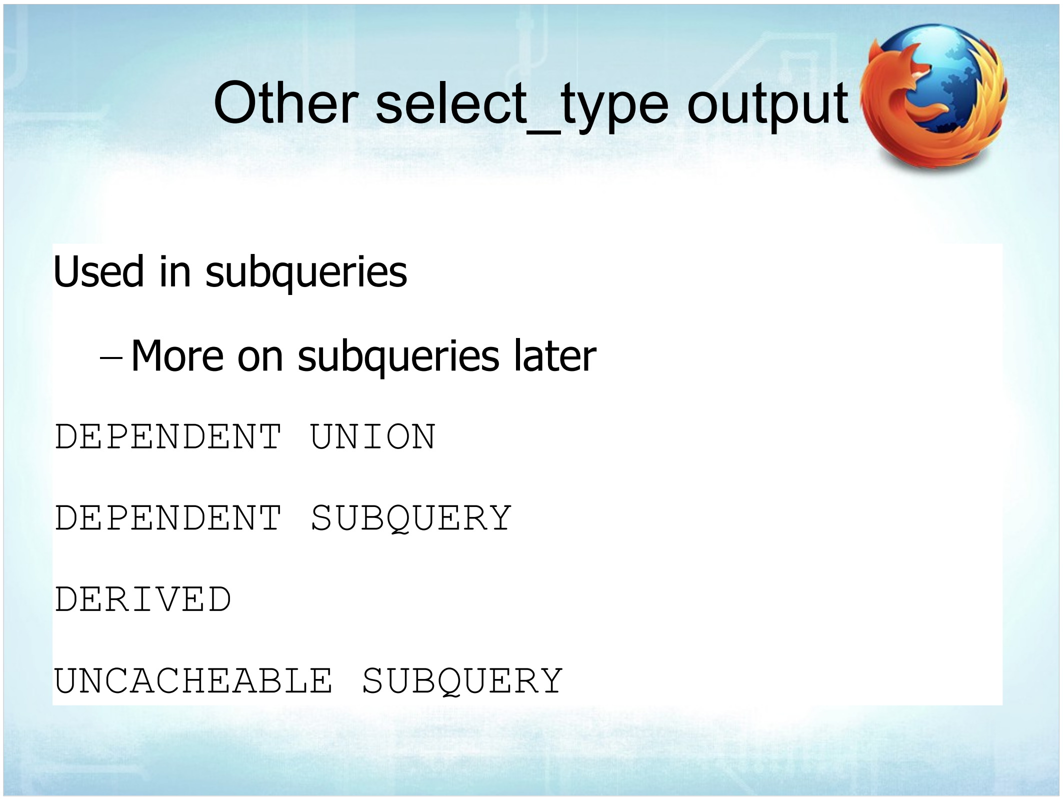
SIMPLE – one table, or JOINs

PRIMARY

- First SELECT in a UNION
- Outer query of a subquery

UNION, UNION RESULT

One row per table.



Other select_type output

Used in subqueries

- More on subqueries later

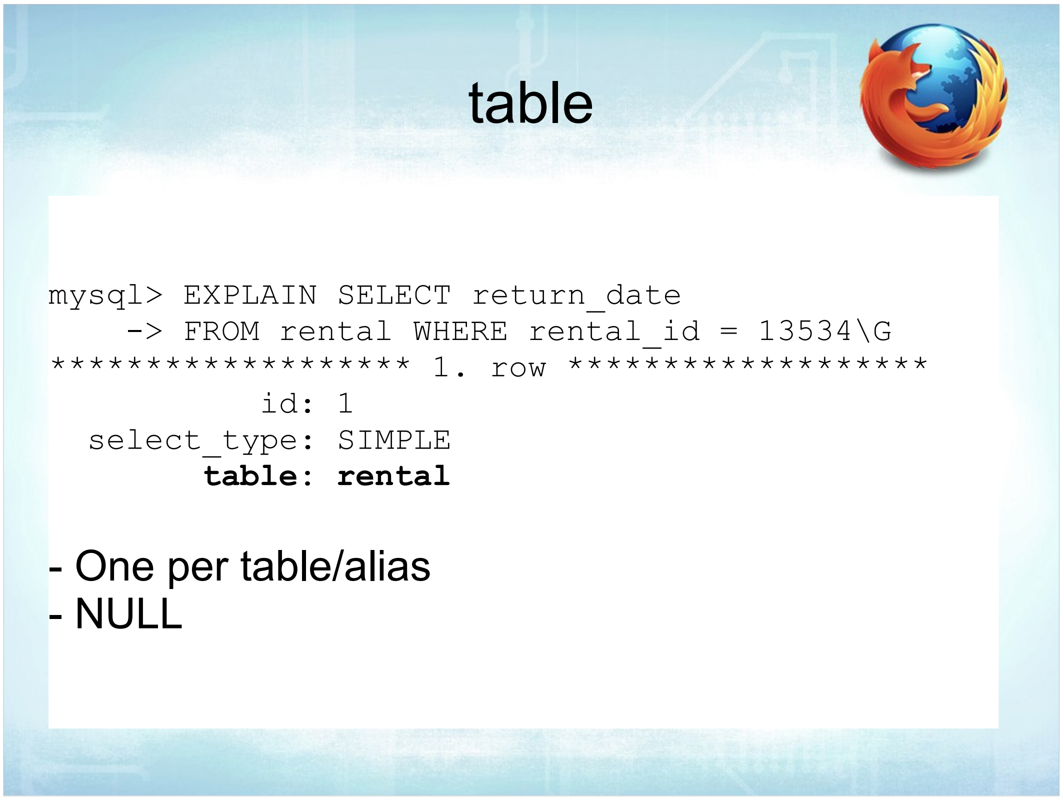
```
DEPENDENT UNION

DEPENDENT SUBQUERY

DERIVED

UNCACHEABLE SUBQUERY
```

One row per table.

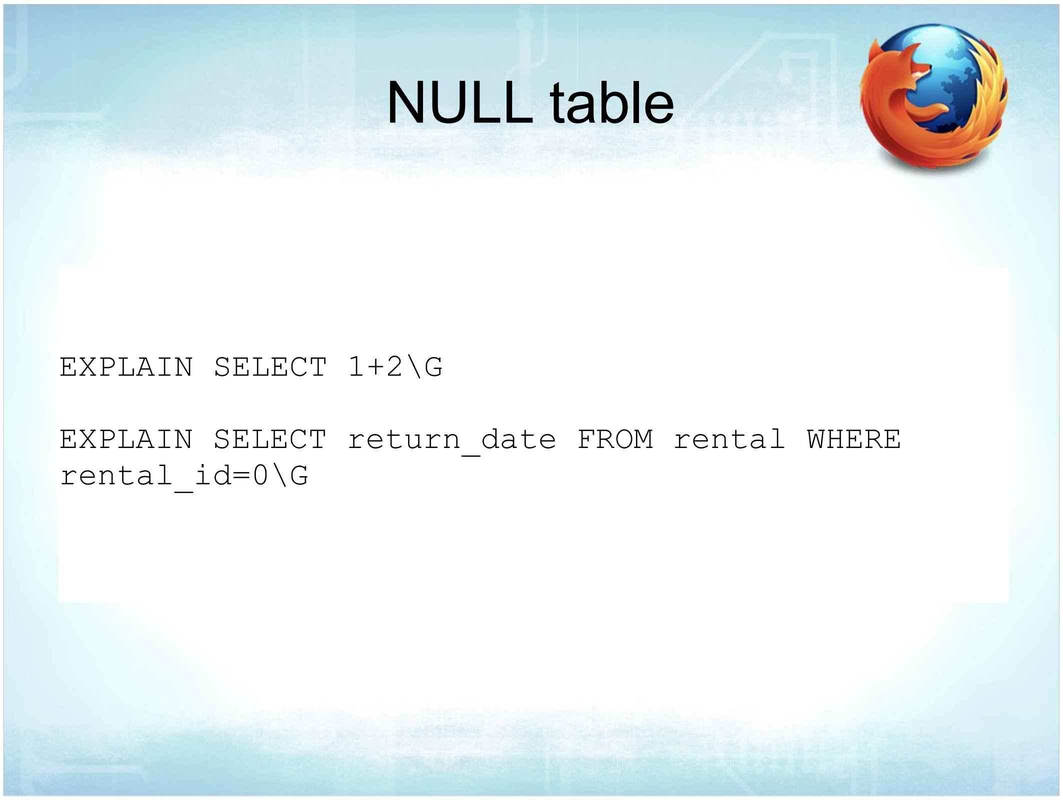


table

```
mysql> EXPLAIN SELECT return_date
    -> FROM rental WHERE rental_id = 13534\G
******************* 1. row *******************
           id: 1
  select_type: SIMPLE
        table: rental
```

- One per table/alias
- NULL

One row per table.

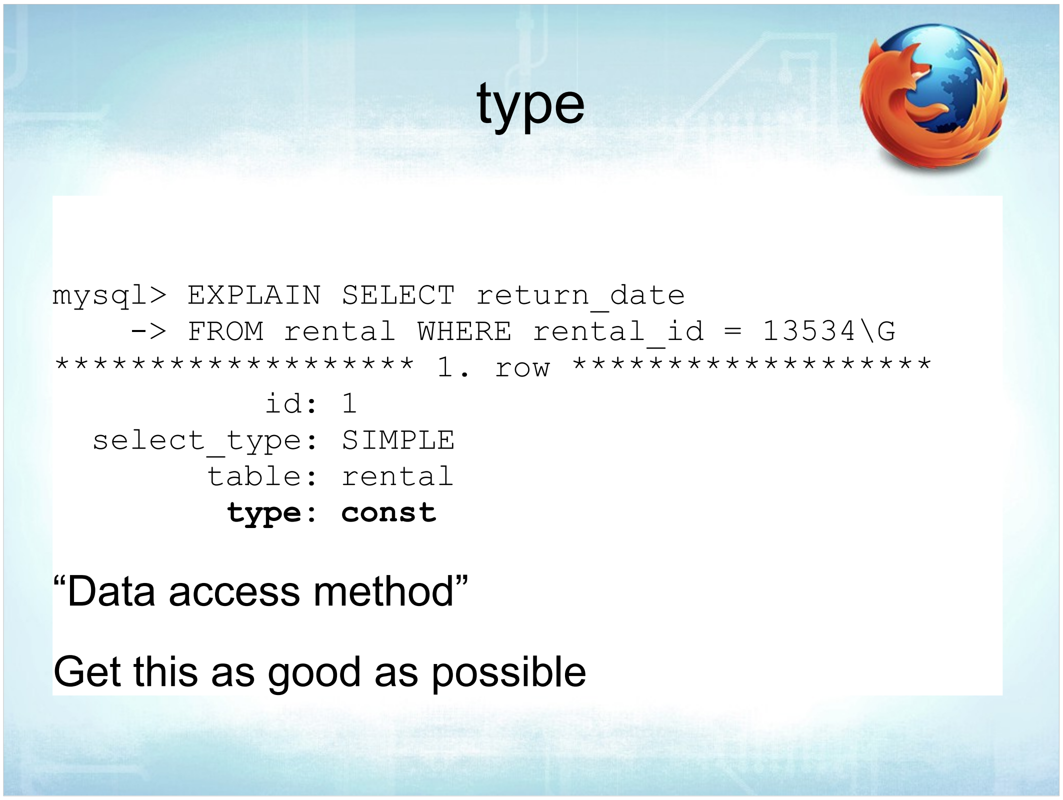


NULL table

```
EXPLAIN SELECT 1+2\G

EXPLAIN SELECT return_date FROM rental WHERE
rental_id=0\G
```

One row per table.



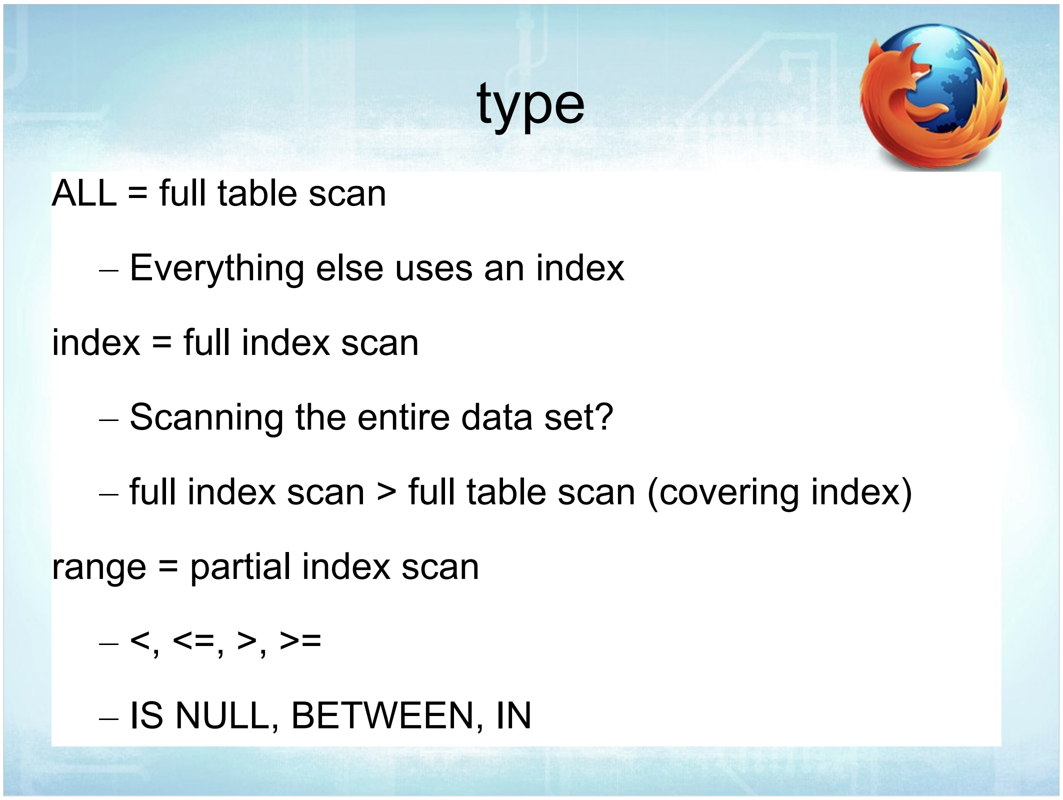
type

```
mysql> EXPLAIN SELECT return_date
    -> FROM rental WHERE rental_id = 13534\G
******************* 1. row *******************
           id: 1
  select_type: SIMPLE
        table: rental
         type: const
```

“Data access method”

Get this as good as possible

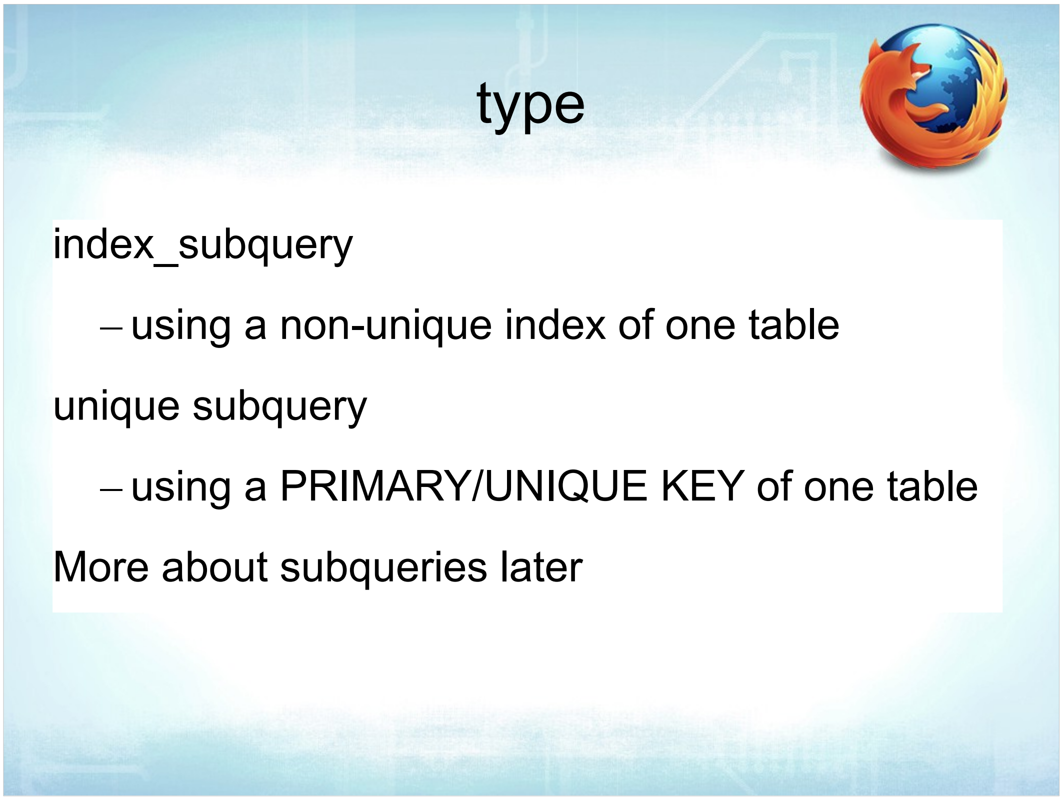
One row per table.



type

- ALL = full table scan
  - Everything else uses an index
- index = full index scan
  - Scanning the entire data set?
  - full index scan > full table scan (covering index)
- range = partial index scan
  - <, <=, >, >=
  - IS NULL, BETWEEN, IN

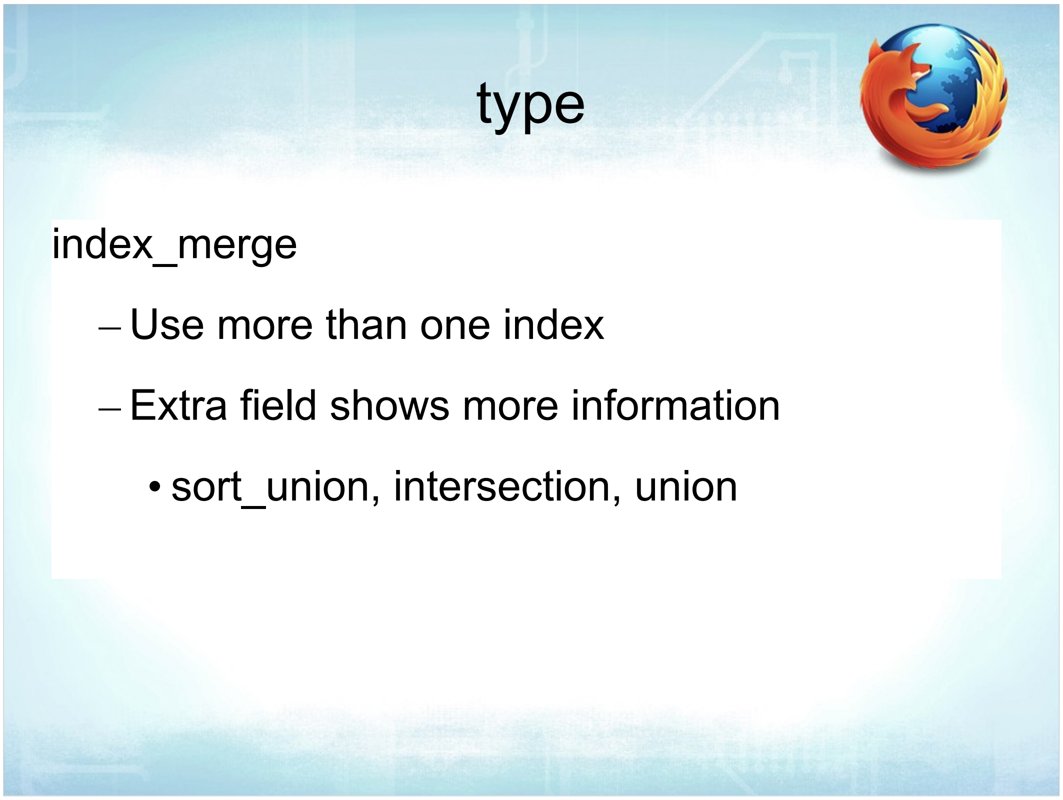
One row per table.



type

- index_subquery
  - using a non-unique index of one table
- unique subquery
  - using a PRIMARY/UNIQUE KEY of one table
- More about subqueries later

One row per table.

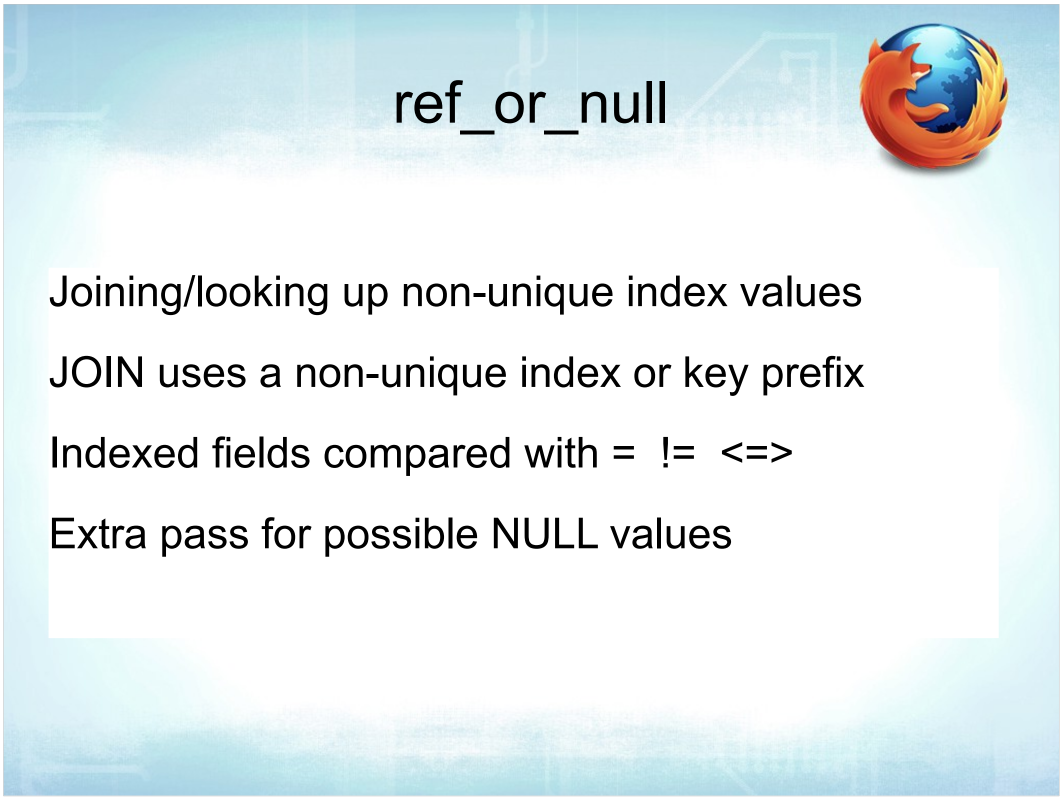


type

index_merge

- Use more than one index
- Extra field shows more information
  - sort_union, intersection, union

One row per table.



ref_or_null

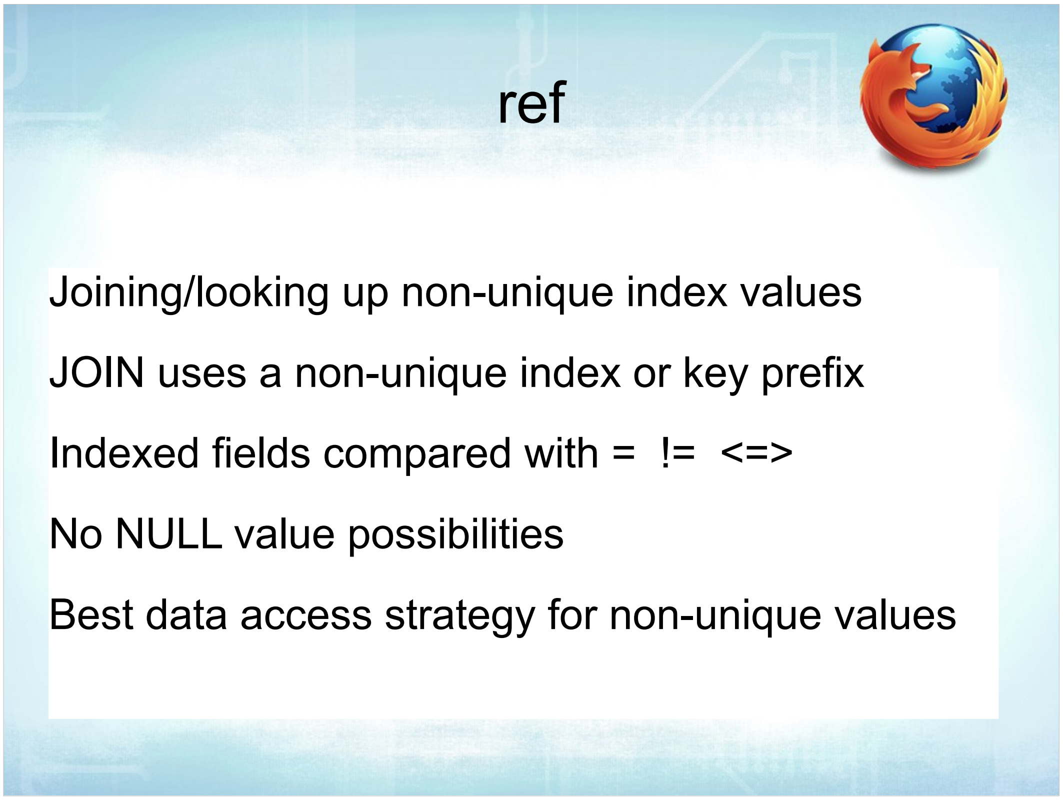
Joining/looking up non-unique index values

JOIN uses a non-unique index or key prefix

Indexed fields compared with = != <=>

Extra pass for possible NULL values

One row per table.



ref

Joining/looking up non-unique index values

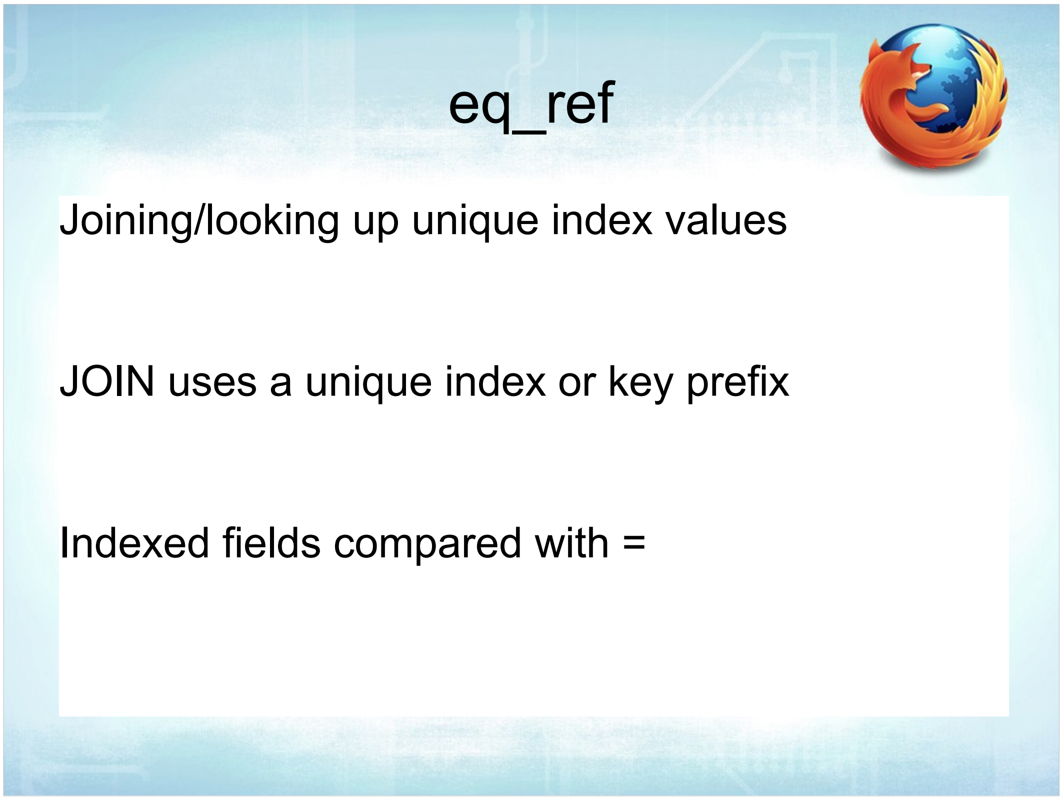
JOIN uses a non-unique index or key prefix

Indexed fields compared with = != <=>

No NULL value possibilities

Best data access strategy for non-unique values

One row per table.



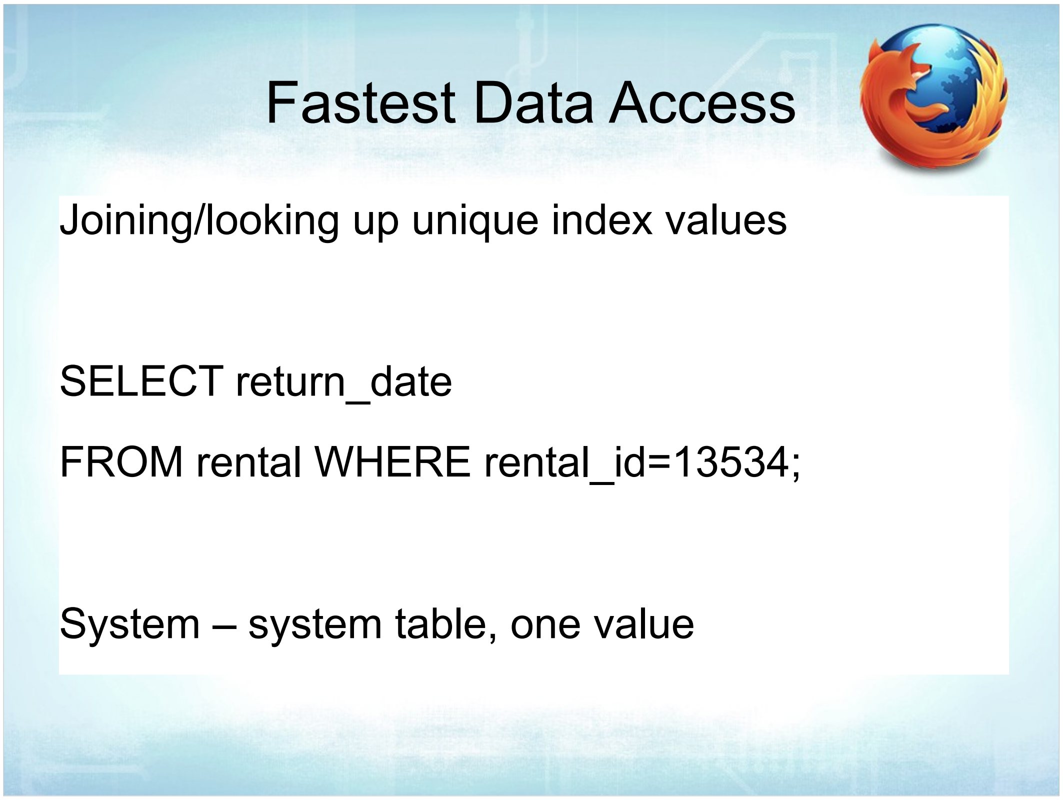
eq_ref

Joining/looking up unique index values

JOIN uses a unique index or key prefix

Indexed fields compared with =

One row per table.



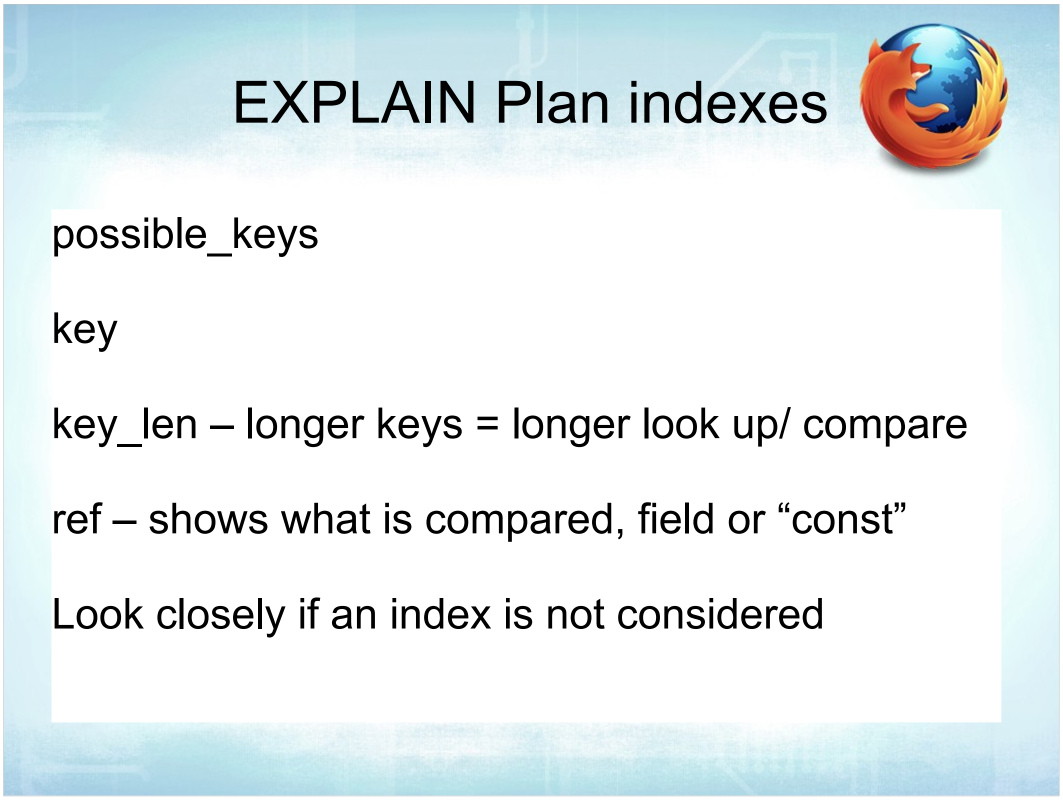
Fastest Data Access

Joining/looking up unique index values

```
SELECT return_date
FROM rental WHERE rental_id=13534;
```

System – system table, one value

One row per table.



EXPLAIN Plan indexes

possible_keys

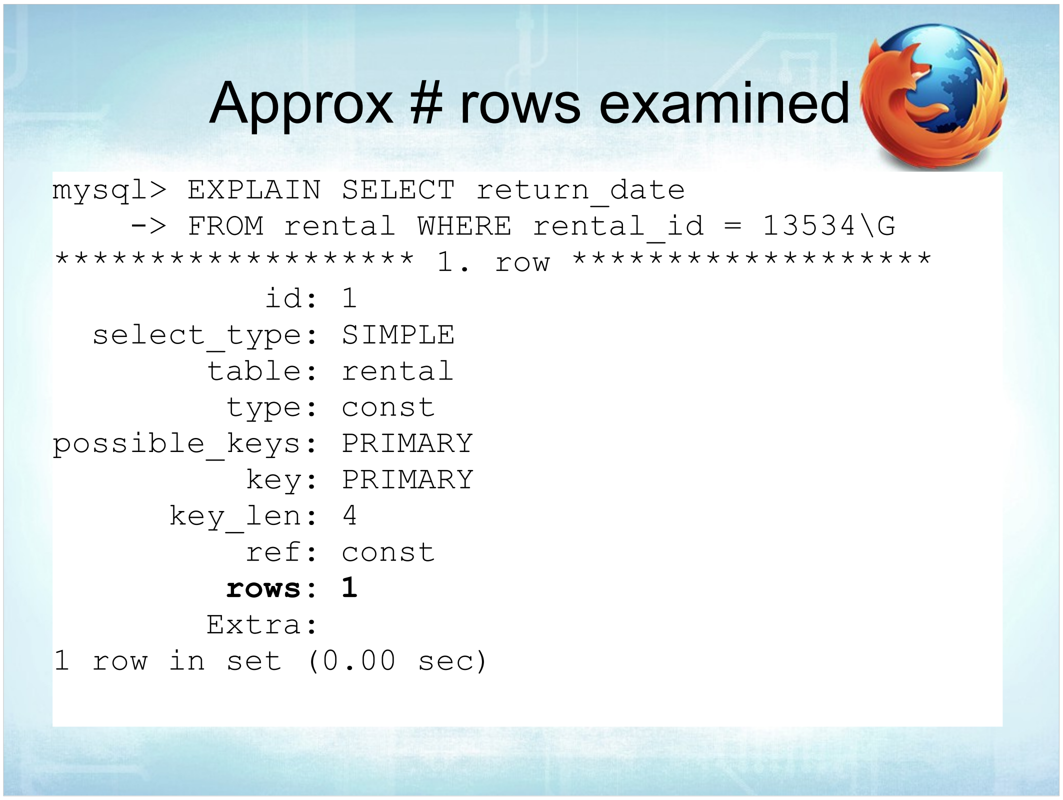
key

key_len – longer keys = longer look up/ compare

ref – shows what is compared, field or “const”

Look closely if an index is not considered

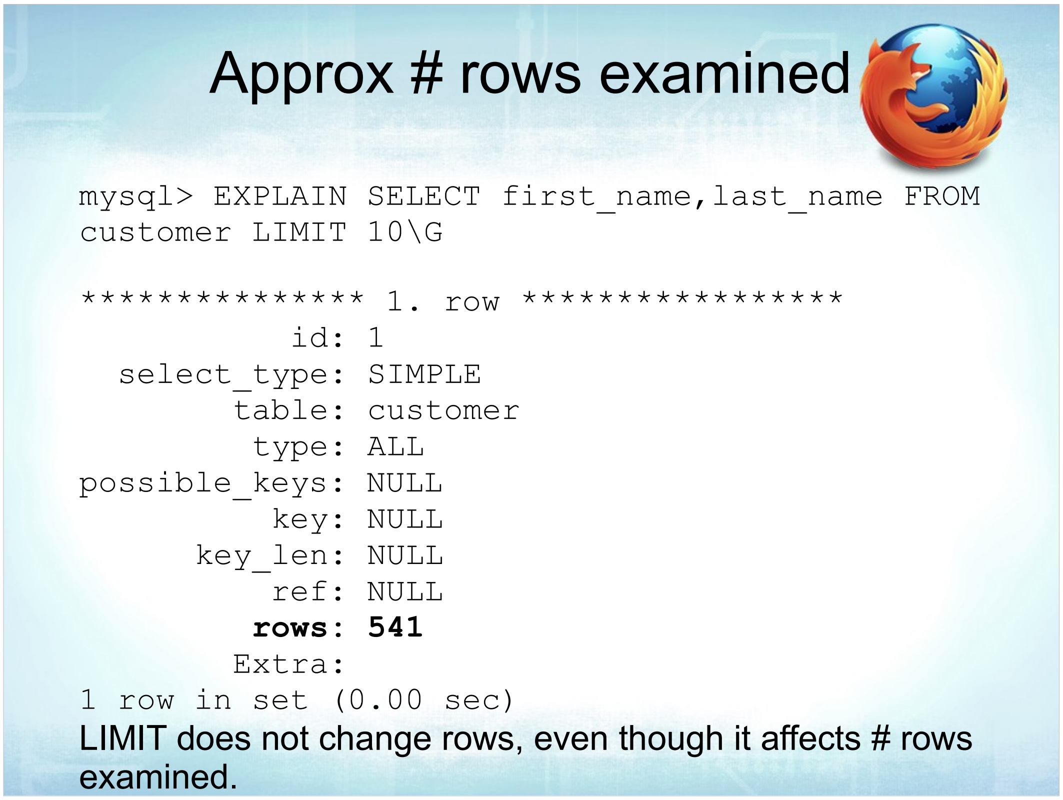
One row per table.



Approx # rows examined

```
mysql> EXPLAIN SELECT return_date
    -> FROM rental WHERE rental_id = 13534\G
******************* 1. row *******************
           id: 1
  select_type: SIMPLE
        table: rental
         type: const
possible_keys: PRIMARY
          key: PRIMARY
      key_len: 4
          ref: const
         rows: 1
        Extra:
1 row in set (0.00 sec)
```

One row per table.



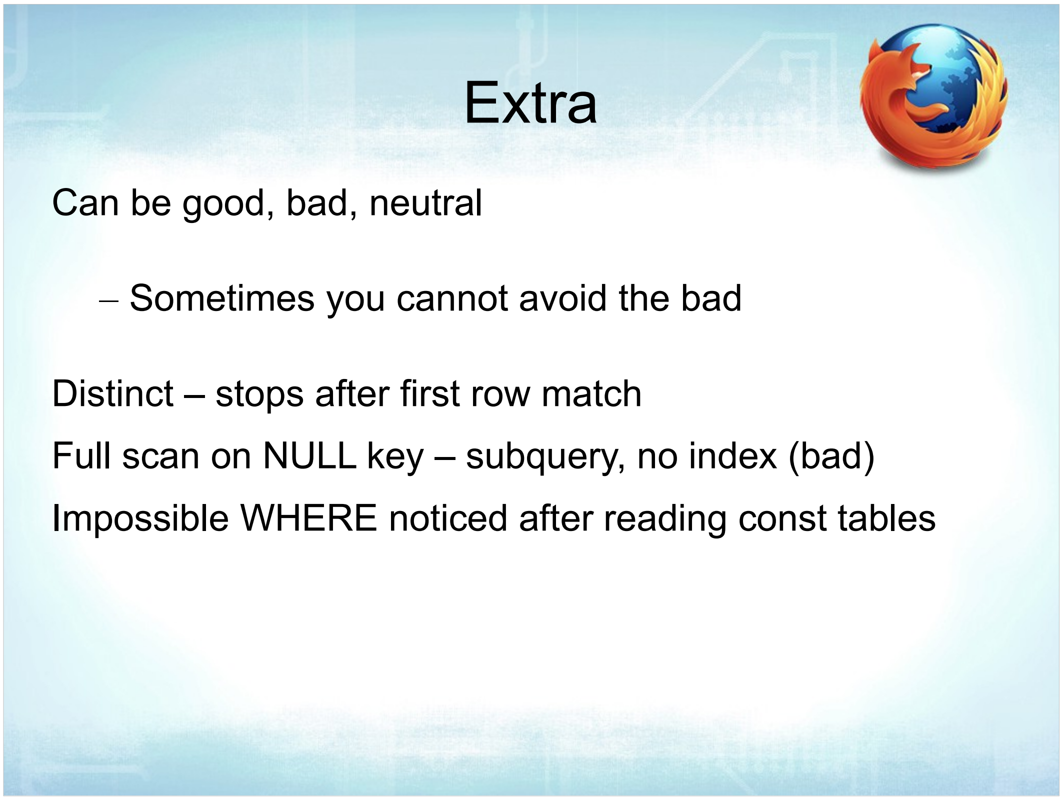
Approx # rows examined

```
mysql> EXPLAIN SELECT first_name,last_name FROM
customer LIMIT 10\G

*************** 1. row *****************
           id: 1
  select_type: SIMPLE
        table: customer
         type: ALL
possible_keys: NULL
          key: NULL
      key_len: NULL
          ref: NULL
         rows: 541
        Extra:
1 row in set (0.00 sec)
```

LIMIT does not change rows, even though it affects # rows examined.

One row per table.



Extra

Can be good, bad, neutral

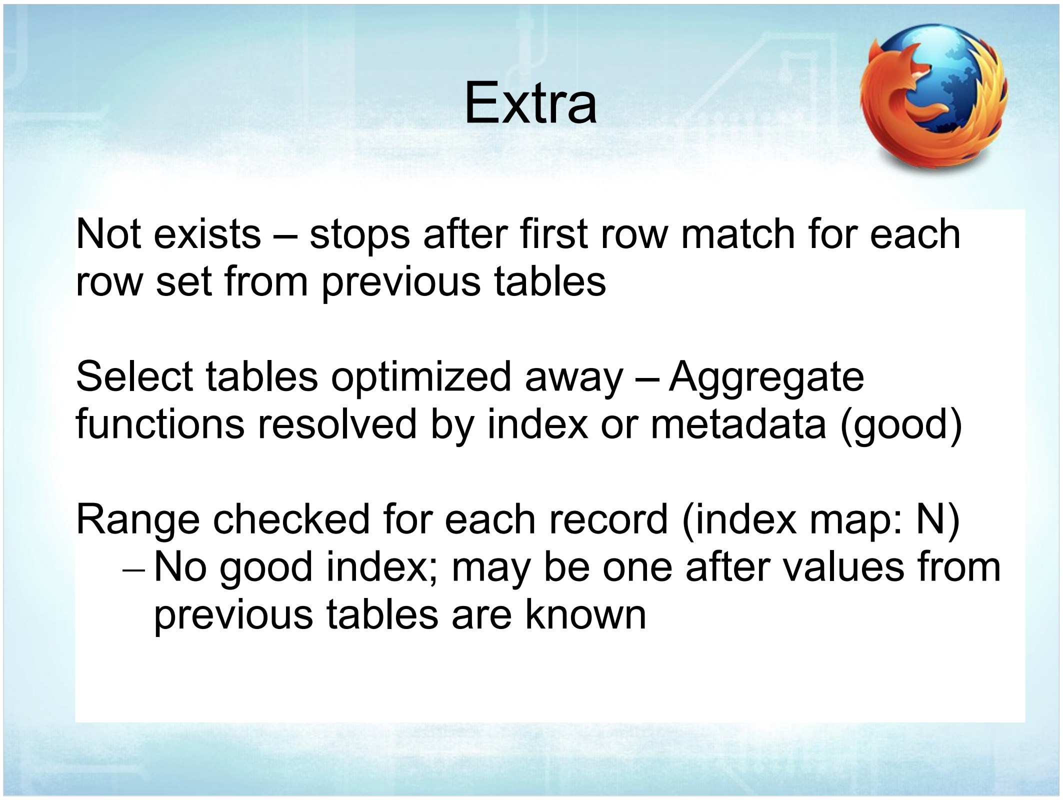
- Sometimes you cannot avoid the bad

Distinct – stops after first row match

Full scan on NULL key – subquery, no index (bad)

Impossible WHERE noticed after reading const tables

One row per table.



Extra

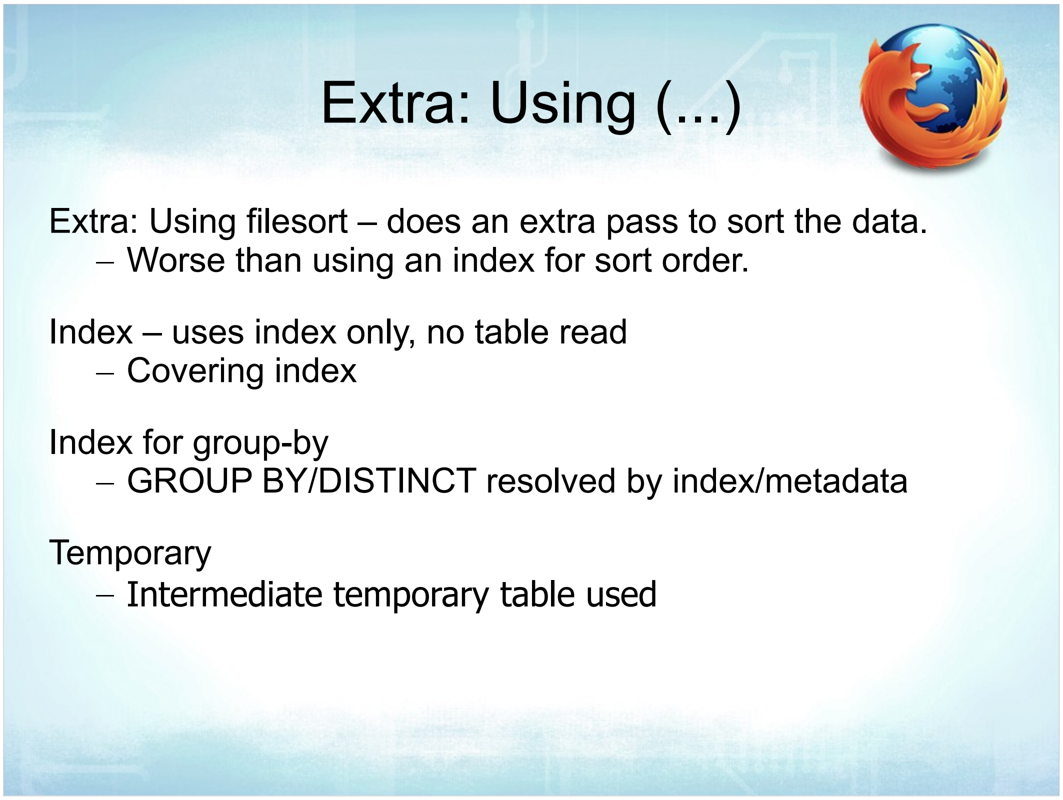
Not exists – stops after first row match for each row set from previous tables

Select tables optimized away – Aggregate functions resolved by index or metadata (good)

Range checked for each record (index map: N)

- No good index; may be one after values from previous tables are known

One row per table.



Extra: Using (...)

Extra: Using filesort – does an extra pass to sort the data.

- Worse than using an index for sort order.

Index – uses index only, no table read

- Covering index

Index for group-by

- GROUP BY/DISTINCT resolved by index/metadata

Temporary

- Intermediate temporary table used

One row per table.

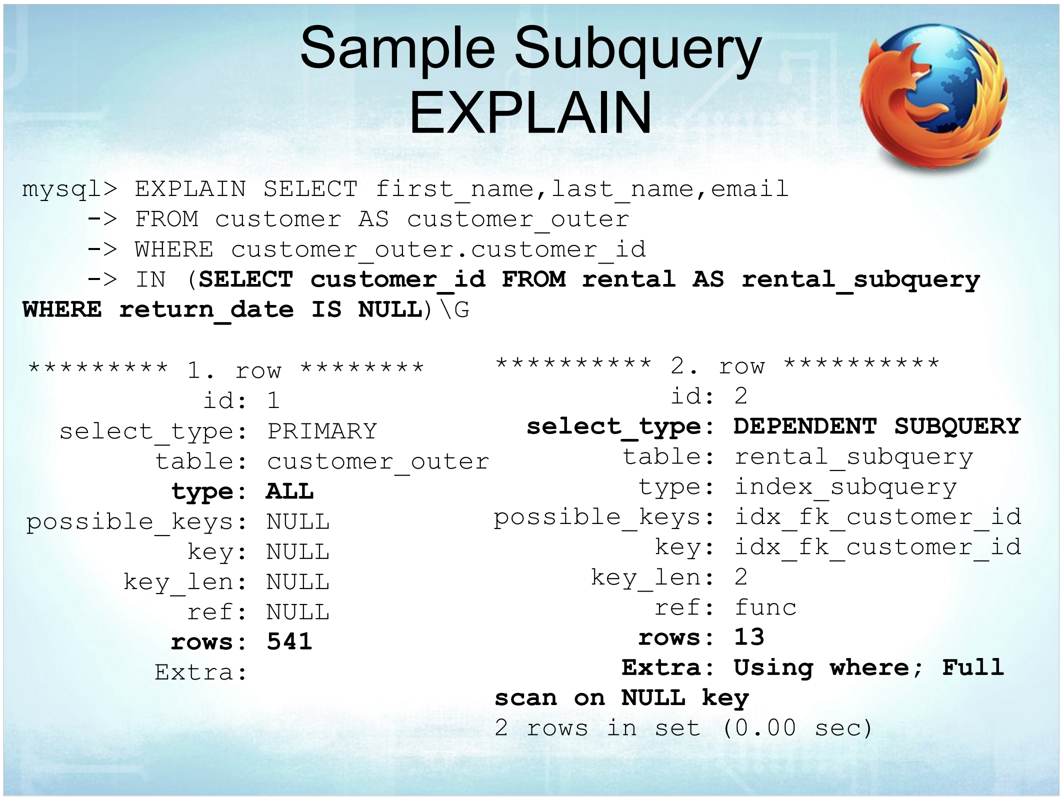
More EXPLAIN Information

MySQL Manual

http://www.pythian.com/news/wp-content/uploads/explain-diagram.pdf

Pages 590 – 614 of the MySQL Administrator's Bible

Sakila sample database: http://dev.mysql.com/doc/index-other.html

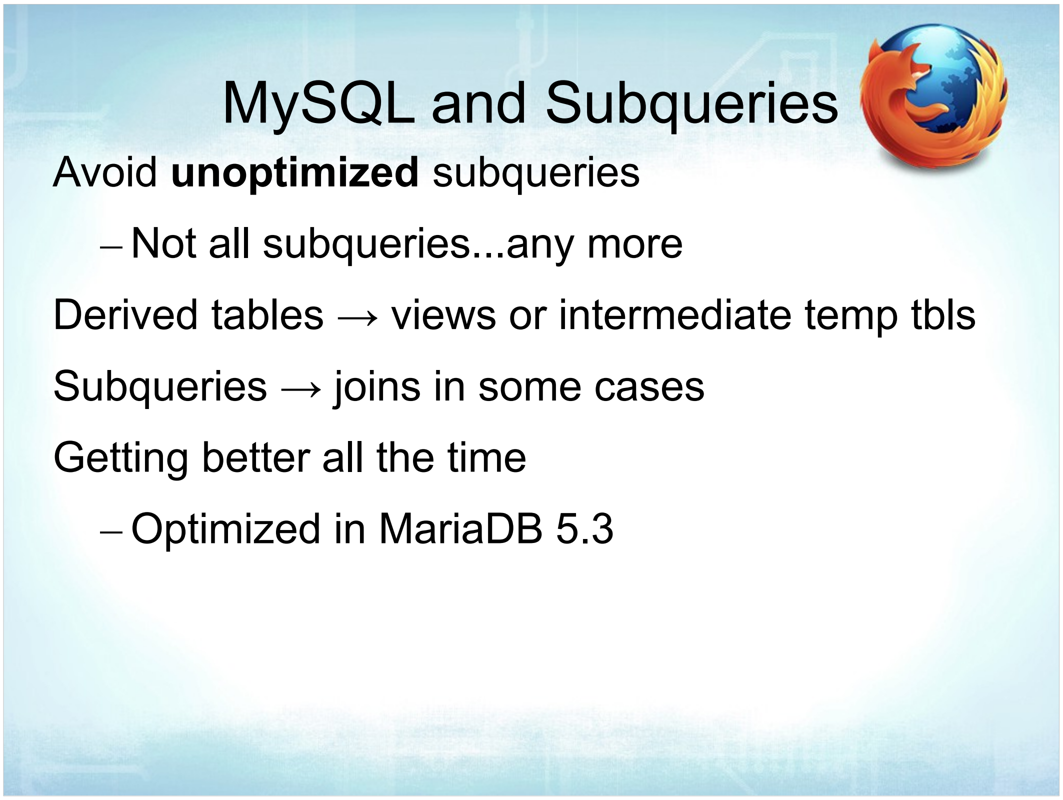


Sample Subquery EXPLAIN

```
mysql> EXPLAIN SELECT first_name,last_name,email
    -> FROM customer AS customer_outer
    -> WHERE customer_outer.customer_id
    -> IN (SELECT customer_id FROM rental AS rental_subquery
WHERE return_date IS NULL)\G

********* 1. row ********     ********** 2. row **********
           id: 1                         id: 2
  select_type: PRIMARY          select_type: DEPENDENT SUBQUERY
        table: customer_outer         table: rental_subquery
         type: ALL                     type: index_subquery
possible_keys: NULL           possible_keys: idx_fk_customer_id
          key: NULL                     key: idx_fk_customer_id
      key_len: NULL                 key_len: 2
          ref: NULL                     ref: func
         rows: 541                     rows: 13
        Extra:                        Extra: Using where; Full
                              scan on NULL key
                              2 rows in set (0.00 sec)
```

One row per table.



MySQL and Subqueries

- Avoid **unoptimized** subqueries
  - Not all subqueries...any more
- Derived tables → views or intermediate temp tbls
- Subqueries → joins in some cases
- Getting better all the time
  - Optimized in MariaDB 5.3

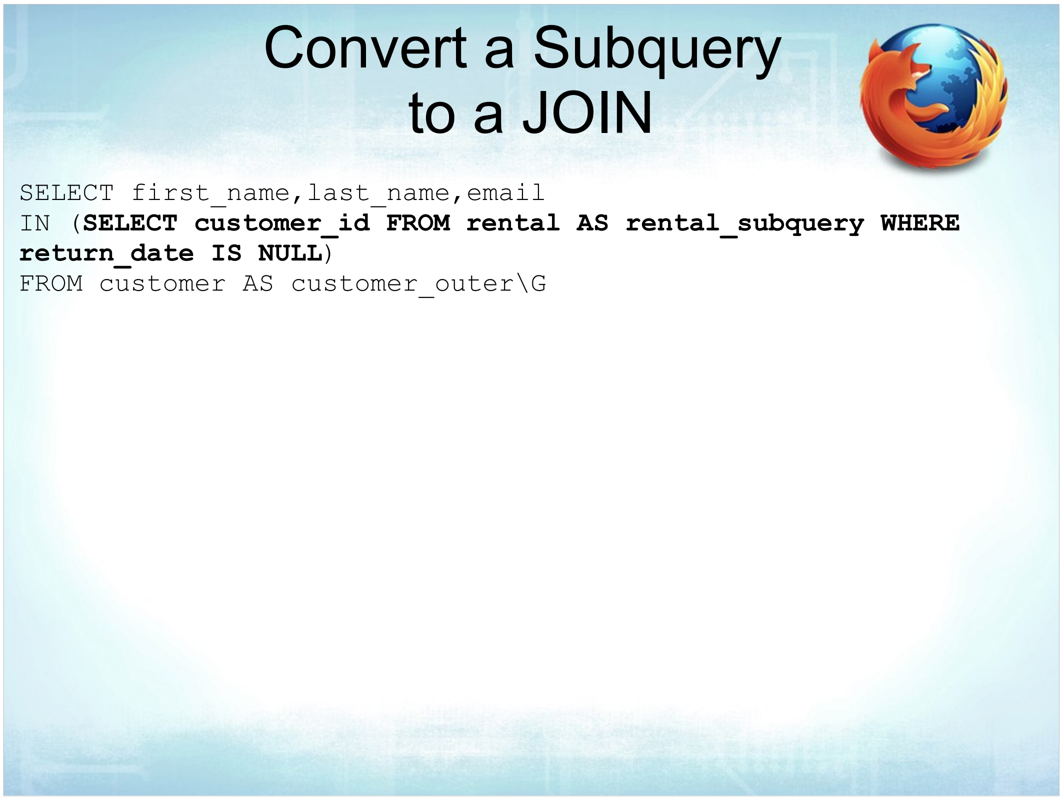
One row per table.

MySQL Does Not Have

Materialized views

Materialized derived tables

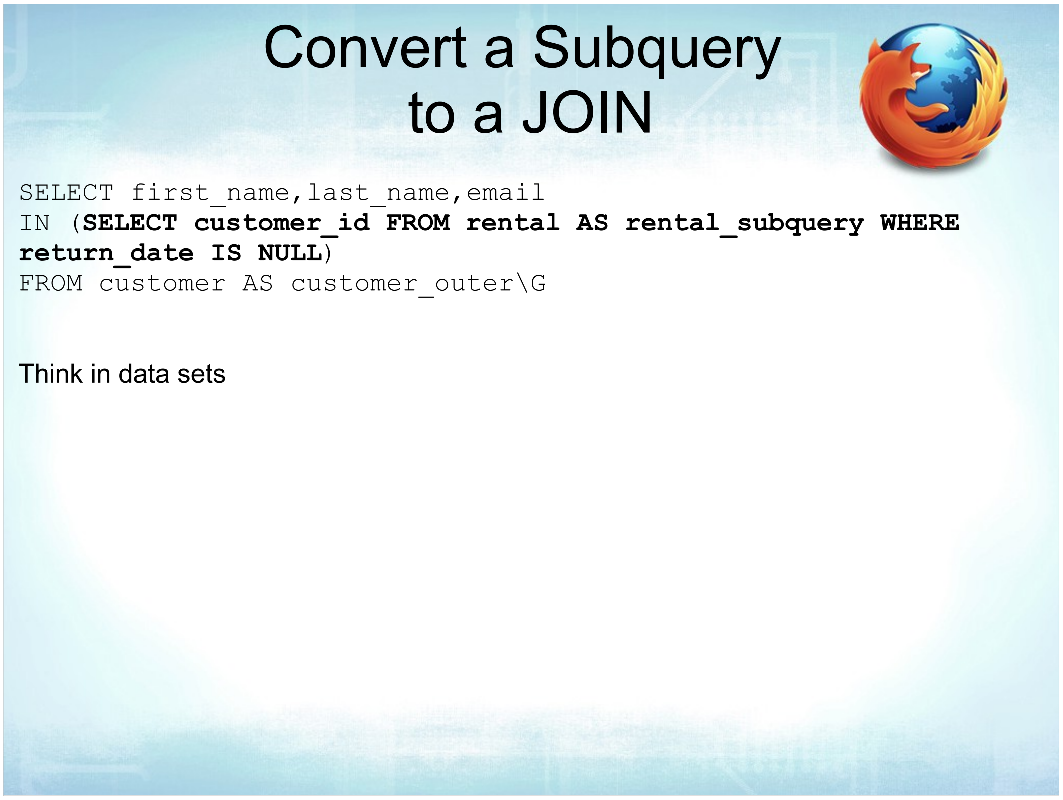
Functional indexes (e.g. `WHERE date(ts)=2012_05_30`)



Convert a Subquery to a JOIN

```
SELECT first_name,last_name,email
IN (SELECT customer_id FROM rental AS rental_subquery WHERE
return_date IS NULL)
FROM customer AS customer_outer\G
```

One row per table.

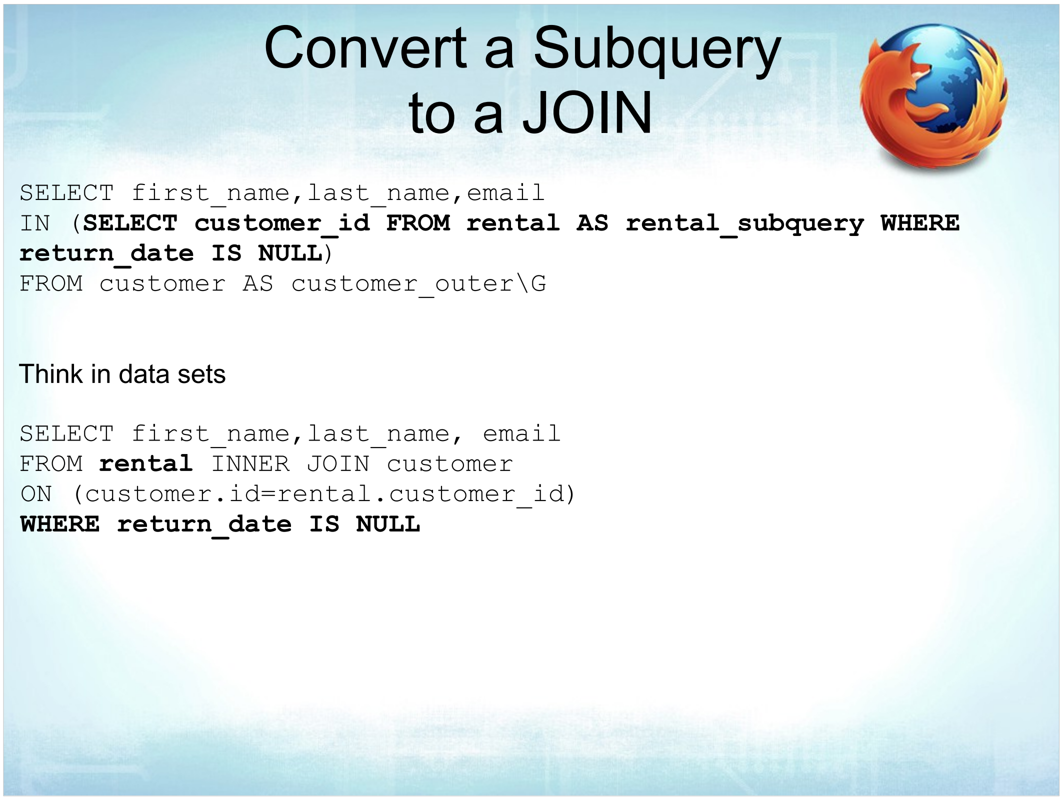


Convert a Subquery to a JOIN

```
SELECT first_name,last_name,email
IN (SELECT customer_id FROM rental AS rental_subquery WHERE
return_date IS NULL)
FROM customer AS customer_outer\G
```

Think in data sets

One row per table.



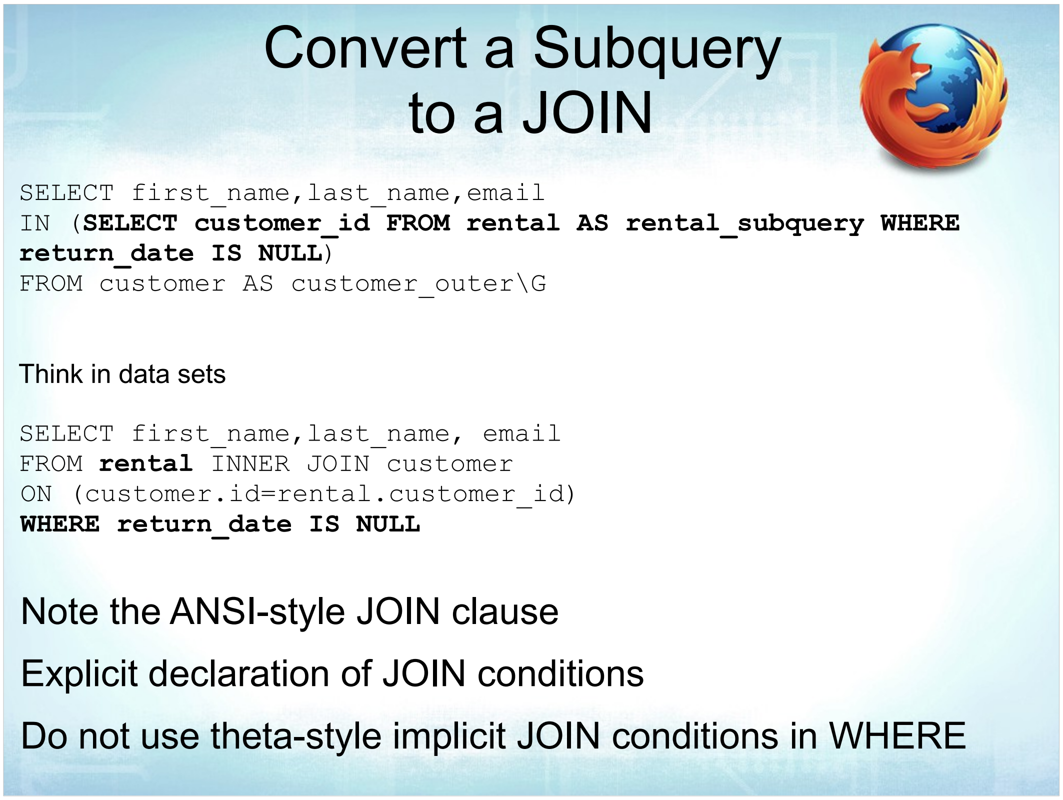
Convert a Subquery to a JOIN

```
SELECT first_name,last_name,email
IN (SELECT customer_id FROM rental AS rental_subquery WHERE
return_date IS NULL)
FROM customer AS customer_outer\G
```

Think in data sets

```
SELECT first_name,last_name, email
FROM rental INNER JOIN customer
ON (customer.id=rental.customer_id)
WHERE return_date IS NULL
```

One row per table.



Convert a Subquery to a JOIN

```
SELECT first_name,last_name,email
IN (SELECT customer_id FROM rental AS rental_subquery WHERE
return_date IS NULL)
FROM customer AS customer_outer\G
```

Think in data sets

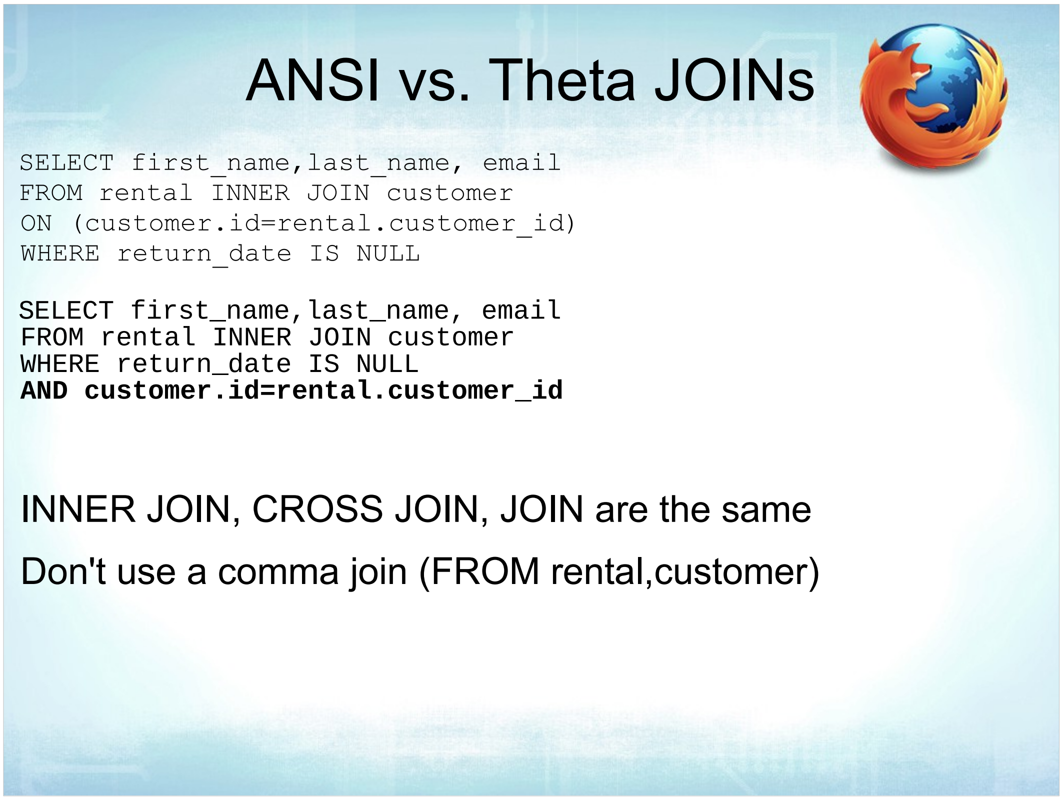
```
SELECT first_name,last_name, email
FROM rental INNER JOIN customer
ON (customer.id=rental.customer_id)
WHERE return_date IS NULL
```

Note the ANSI-style JOIN clause

Explicit declaration of JOIN conditions

Do not use theta-style implicit JOIN conditions in WHERE

One row per table.



ANSI vs. Theta JOINs

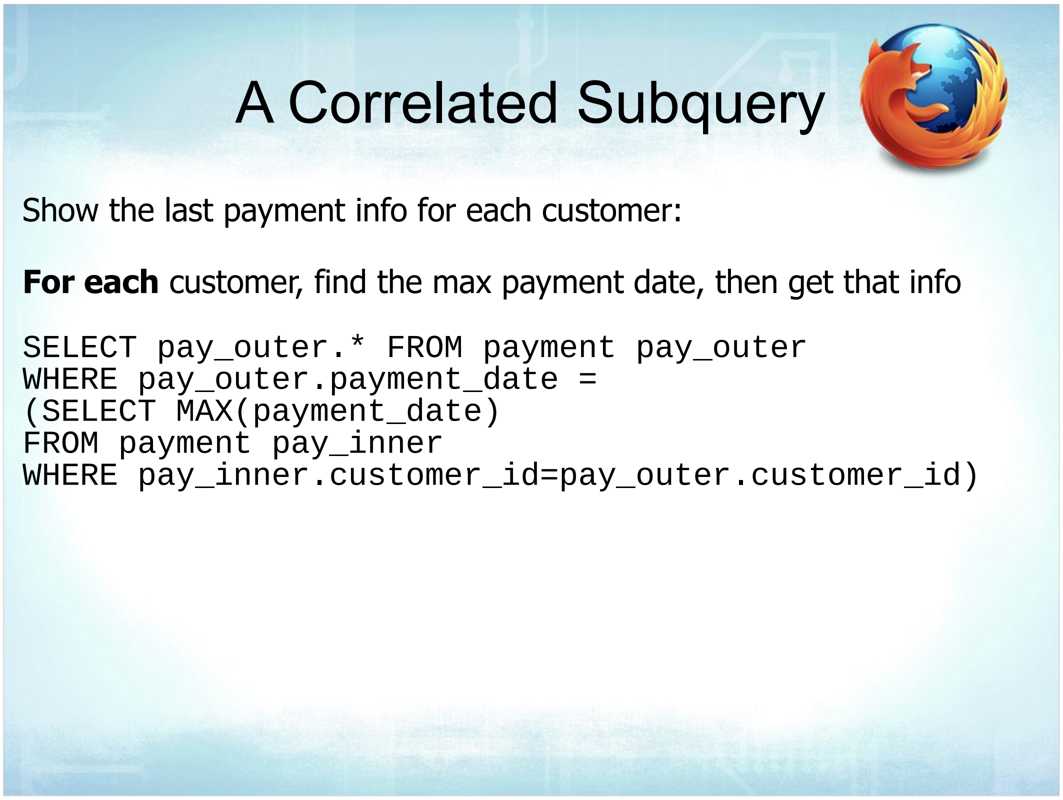
```
SELECT first_name,last_name, email
FROM rental INNER JOIN customer
ON (customer.id=rental.customer_id)
WHERE return_date IS NULL
```

```
SELECT first_name,last_name, email
FROM rental INNER JOIN customer
WHERE return_date IS NULL
AND customer.id=rental.customer_id
```

INNER JOIN, CROSS JOIN, JOIN are the same

Don't use a comma join (FROM rental,customer)

One row per table.

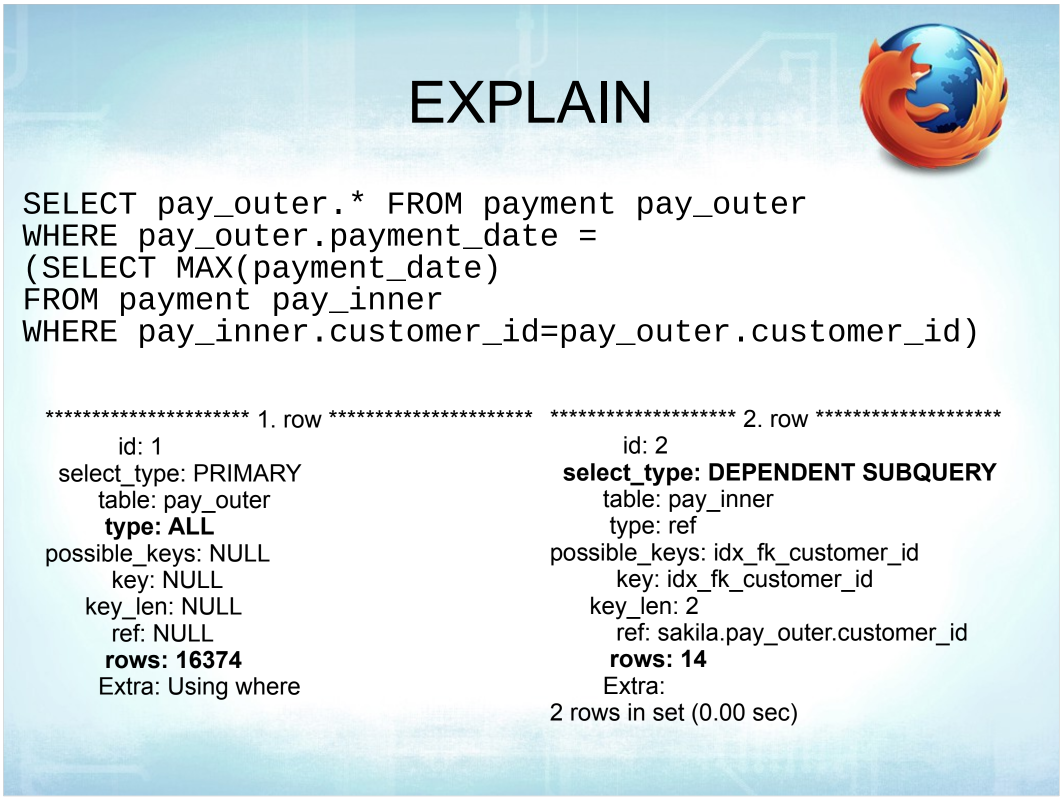


A Correlated Subquery

Show the last payment info for each customer:

For each customer, find the max payment date, then get that info

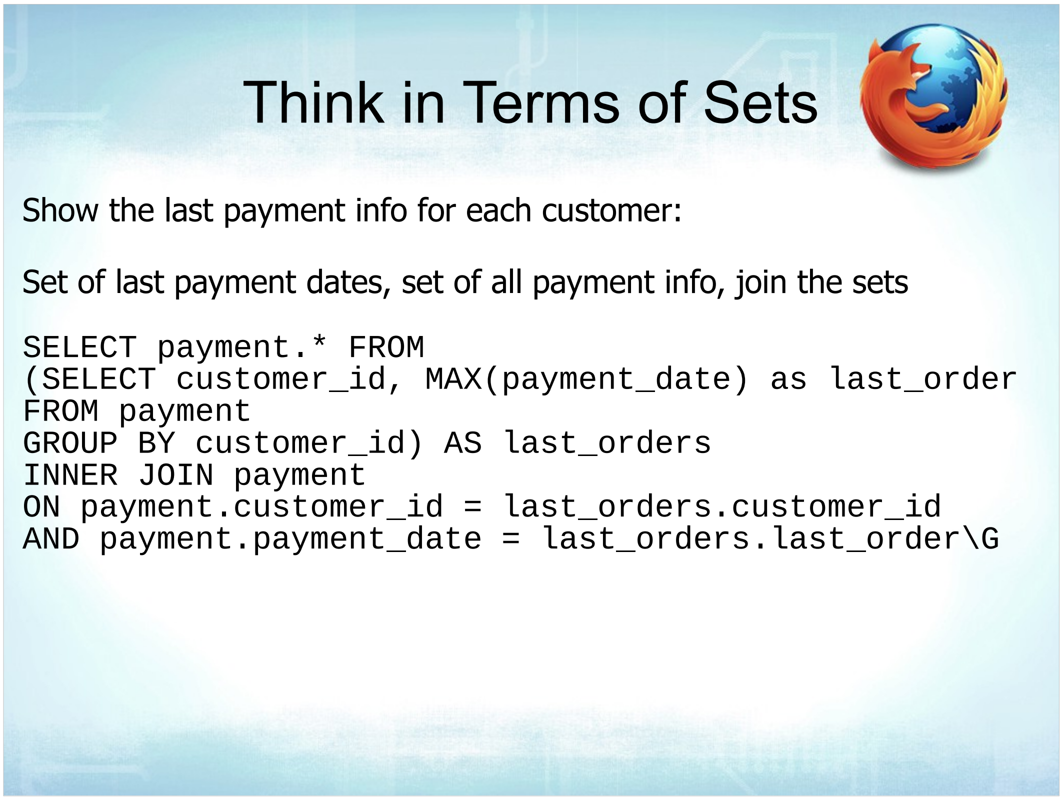
```
SELECT pay_outer.* FROM payment pay_outer
WHERE pay_outer.payment_date =
(SELECT MAX(payment_date)
FROM payment pay_inner
WHERE pay_inner.customer_id=pay_outer.customer_id)
```



EXPLAIN

```
SELECT pay_outer.* FROM payment pay_outer
WHERE pay_outer.payment_date =
(SELECT MAX(payment_date)
FROM payment pay_inner
WHERE pay_inner.customer_id=pay_outer.customer_id)
```

```
*********************** 1. row ***********************
           id: 1
  select_type: PRIMARY
        table: pay_outer
         type: ALL
possible_keys: NULL
          key: NULL
      key_len: NULL
          ref: NULL
         rows: 16374
        Extra: Using where
********************* 2. row *********************
           id: 2
  select_type: DEPENDENT SUBQUERY
        table: pay_inner
         type: ref
possible_keys: idx_fk_customer_id
          key: idx_fk_customer_id
      key_len: 2
          ref: sakila.pay_outer.customer_id
         rows: 14
        Extra:
2 rows in set (0.00 sec)
```

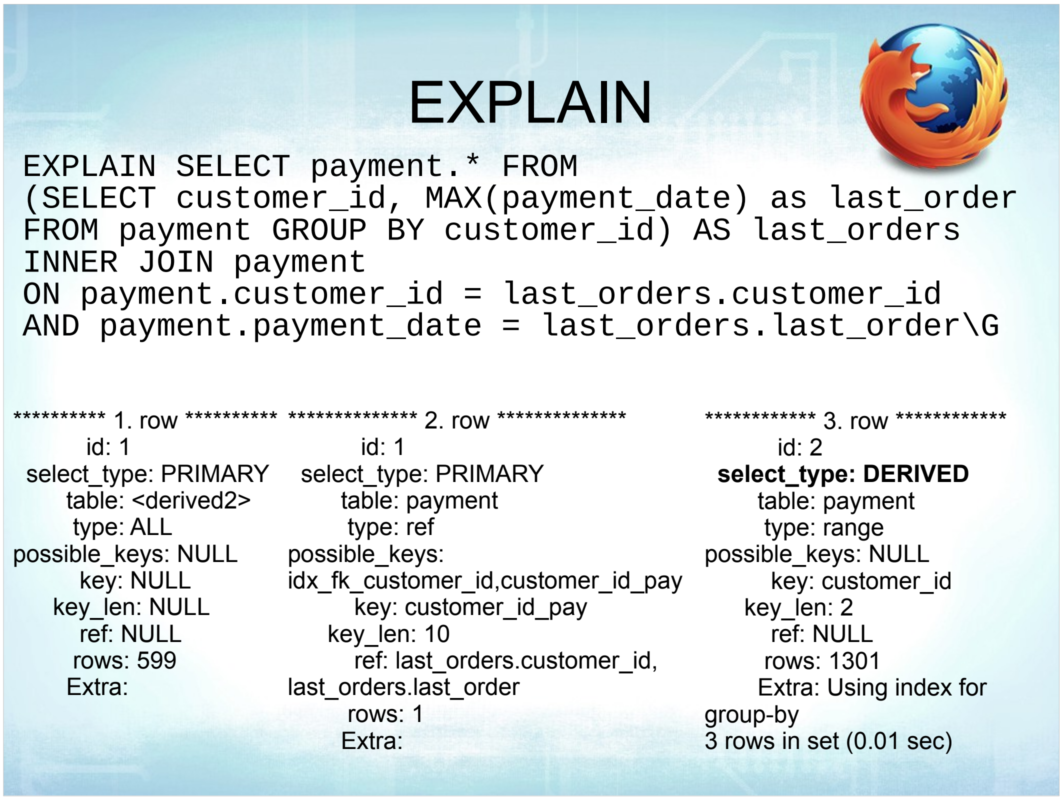


Think in Terms of Sets

Show the last payment info for each customer:

Set of last payment dates, set of all payment info, join the sets

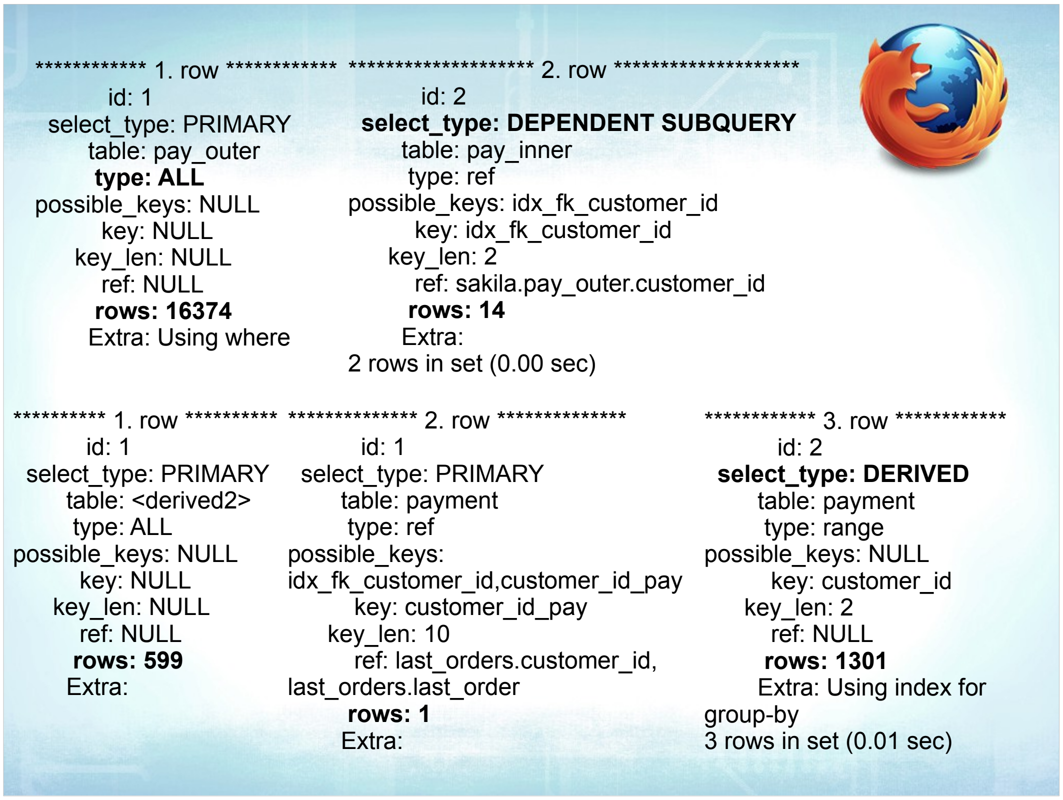
```
SELECT payment.* FROM
(SELECT customer_id, MAX(payment_date) as last_order
FROM payment
GROUP BY customer_id) AS last_orders
INNER JOIN payment
ON payment.customer_id = last_orders.customer_id
AND payment.payment_date = last_orders.last_order\G
```



EXPLAIN

```
EXPLAIN SELECT payment.* FROM
(SELECT customer_id, MAX(payment_date) as last_order
FROM payment GROUP BY customer_id) AS last_orders
INNER JOIN payment
ON payment.customer_id = last_orders.customer_id
AND payment.payment_date = last_orders.last_order\G
```

```
********** 1. row **********
          id: 1
 select_type: PRIMARY
       table: <derived2>
        type: ALL
possible_keys: NULL
         key: NULL
     key_len: NULL
         ref: NULL
        rows: 599
       Extra:
************** 2. row **************
          id: 1
 select_type: PRIMARY
       table: payment
        type: ref
possible_keys:
idx_fk_customer_id,customer_id_pay
         key: customer_id_pay
     key_len: 10
         ref: last_orders.customer_id,
last_orders.last_order
        rows: 1
       Extra:
************ 3. row ************
          id: 2
 select_type: DERIVED
       table: payment
        type: range
possible_keys: NULL
         key: customer_id
     key_len: 2
         ref: NULL
        rows: 1301
       Extra: Using index for
group-by
3 rows in set (0.01 sec)
```



```
************ 1. row ************
           id: 1
  select_type: PRIMARY
        table: pay_outer
         type: ALL
possible_keys: NULL
          key: NULL
      key_len: NULL
          ref: NULL
         rows: 16374
        Extra: Using where
```

```
******************** 2. row ********************
           id: 2
  select_type: DEPENDENT SUBQUERY
        table: pay_inner
         type: ref
possible_keys: idx_fk_customer_id
          key: idx_fk_customer_id
      key_len: 2
          ref: sakila.pay_outer.customer_id
         rows: 14
        Extra:
2 rows in set (0.00 sec)
```

```
********** 1. row **********
           id: 1
  select_type: PRIMARY
        table: <derived2>
         type: ALL
possible_keys: NULL
          key: NULL
      key_len: NULL
          ref: NULL
         rows: 599
        Extra:
```

```
************** 2. row **************
           id: 1
  select_type: PRIMARY
        table: payment
         type: ref
possible_keys:
idx_fk_customer_id,customer_id_pay
          key: customer_id_pay
      key_len: 10
          ref: last_orders.customer_id,
last_orders.last_order
         rows: 1
        Extra:
```

```
************ 3. row ************
           id: 2
  select_type: DERIVED
        table: payment
         type: range
possible_keys: NULL
          key: customer_id
      key_len: 2
          ref: NULL
         rows: 1301
        Extra: Using index for
group-by
3 rows in set (0.01 sec)
```

Join-fu

http://joinfu.com/presentations/joinfu/joinfu_part_one.pdf

- p 22, mapping tables

http://joinfu.com/presentations/joinfu/joinfu_part_two.pdf

- heirarchies/graphs/nested sets

- GIS calculations

- reporting/aggregates/ranks

With thanks to Jay Pipes!

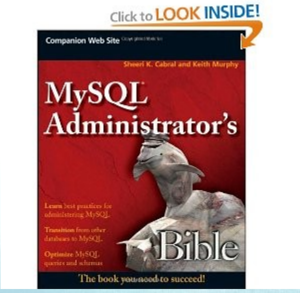
Questions? Comments?

OurSQL Podcast

www.oursql.com

MySQL Administrator's Bible

- tinyurl.com/mysqlbible


Click to LOOK INSIDE!
Companion Web Site
Sheeri K. Cabral and Keith Murphy
MySQL Administrator's Bible
Learn best practices for administering MySQL
Transition from other databases to MySQL
Optimize MySQL queries and schemas
The book you need to succeed!

kimtag.com/mysql

planet.mysql.com