Database Tuning

As the person writing queries, you have control over:

- how you write the query
- providing hints to the optimizer

As a DB admin, you have control over:

- what indexes are available
  - specified as part of `CREATE TABLE`, or use `CREATE INDEX`
- whether the optimizer is using the most current key distribution information
  - can configure how often stats are recalculated automatically
  - update manually after substantial changes with `ANALYZE TABLE` *`tablename`*

Getting Info

`EXPLAIN SELECT …;`

`EXPLAIN` is also available for `DELETE`, `INSERT`, `UPDATE`

- returns information about how the query is carried out
- uses
  - can see where adding indexes would help
  - can see if optimizer uses the best join order for the tables


```
EXPLAIN SELECT *
FROM SAILOR S NATURAL JOIN RESERVATION R NATURAL JOIN BOAT B
WHERE B.Color='red'
```

produces the following result:

```
+-------+--------+----------------+-----------------+-----------------+------+----------+-------------+
| table | type   | possible_keys  | key             | ref             | rows | filtered | Extra       |
+-------+--------+----------------+-----------------+-----------------+------+----------+-------------+
| B     | ALL    | PRIMARY        | NULL            | NULL            |    4 |    25.00 | Using where |
| R     | ref    | PRIMARY,       | fk_RESERVATION_1 | ex_sailors.B.Bid |    2 |   100.00 | NULL        |
|       |        | fk_RESERVATION_1,|                 |                 |      |          |             |
|       |        | fk_RESERVATION_2 |                 |                 |      |          |             |
| S     | eq_ref | PRIMARY        | PRIMARY         | ex_sailors.R.Sid |    1 |   100.00 | NULL        |
+-------+--------+----------------+-----------------+-----------------+------+----------+-------------+
```

What order are the tables read in when processing the joins?

- S, then R, then B
- R, then B, then S
- B, then R, then S
- can't tell
- none of the above (can tell, but it's not one of the choices listed)


```
EXPLAIN SELECT *
FROM SAILOR S NATURAL JOIN RESERVATION R NATURAL JOIN BOAT B
WHERE B.Color='red'
```

produces the following result:

```
+-------+--------+----------------+-----------------+-----------------+------+----------+-------------+
| table | type   | possible_keys  | key             | ref             | rows | filtered | Extra       |
+-------+--------+----------------+-----------------+-----------------+------+----------+-------------+
| B     | ALL    | PRIMARY        | NULL            | NULL            |    4 |    25.00 | Using where |
| R     | ref    | PRIMARY,       | fk_RESERVATION_1 | ex_sailors.B.Bid |    2 |   100.00 | NULL        |
|       |        | fk_RESERVATION_1,|                 |                 |      |          |             |
|       |        | fk_RESERVATION_2 |                 |                 |      |          |             |
| S     | eq_ref | PRIMARY        | PRIMARY         | ex_sailors.R.Sid |    1 |   100.00 | NULL        |
+-------+--------+----------------+-----------------+-----------------+------+----------+-------------+
```


`id`	sequential number of the SELECT within the query
`select_type`	how the SELECT fits into the query
`type`	join type
`possible_keys`	which indexes mysql can choose from to find rows in this table
`key`	which index mysql used
`ref`	which attributes were compared to t
`rows`	number of rows mysql thinks it must execute the query (may be an estim
`filtered`	estimated selectivity (for rows examined)
`extra`	other information

Which tables have indexes that could be used? Choose all that apply.

- B
- R
- S
- none of the above
- can't tell

```
EXPLAIN SELECT *
FROM SAILOR S NATURAL JOIN RESERVATION R NATURAL JOIN BOAT B
WHERE B.Color='red'
```

produces the following result:

```
+-------+--------+-----------------+-----------------+-----------------+------+----------+-------------+
| table | type   | possible_keys   | key             | ref             | rows | filtered | Extra       |
+-------+--------+-----------------+-----------------+-----------------+------+----------+-------------+
| B     | ALL    | PRIMARY         | NULL            | NULL            |    4 |    25.00 | Using where |
| R     | ref    | PRIMARY,        | fk_RESERVATION_1| ex_sailors.B.Bid|    2 |   100.00 | NULL        |
|       |        | fk_RESERVATION_1,|                 |                 |      |          |             |
|       |        | fk_RESERVATION_2 |                 |                 |      |          |             |
| S     | eq_ref | PRIMARY         | PRIMARY         | ex_sailors.R.Sid|    1 |   100.00 | NULL        |
+-------+--------+-----------------+-----------------+-----------------+------+----------+-------------+
```

`id`	sequential number of the SELECT within the query
`select_type`	how the SELECT fits into the query
`type`	join type
`possible_keys`	which indexes mysql can choose from to find rows in this table
`key`	which index mysql used
`ref`	which attributes were compared to the i
`rows`	number of rows mysql thinks it must ex execute the query (may be an estimate
`filtered`	estimated selectivity (for rows examined)
`extra`	other information

For which tables are indexes actually used? Choose all that apply.

- ☐ B
- ☐ R
- ☐ S
- ☐ none of the above
- ☐ can't tell


```
EXPLAIN SELECT *
FROM SAILOR S NATURAL JOIN RESERVATION R NATURAL JOIN BOAT B
WHERE B.Color='red'
```

produces the following result:

```
+-------+--------+-----------------+-----------------+-----------------+------+----------+-------------+
| table | type   | possible_keys   | key             | ref             | rows | filtered | Extra       |
+-------+--------+-----------------+-----------------+-----------------+------+----------+-------------+
| B     | ALL    | PRIMARY         | NULL            | NULL            |    4 |    25.00 | Using where |
| R     | ref    | PRIMARY,        | fk_RESERVATION_1| ex_sailors.B.Bid|    2 |   100.00 | NULL        |
|       |        | fk_RESERVATION_1,|                 |                 |      |          |             |
|       |        | fk_RESERVATION_2 |                 |                 |      |          |             |
| S     | eq_ref | PRIMARY         | PRIMARY         | ex_sailors.R.Sid|    1 |   100.00 | NULL        |
+-------+--------+-----------------+-----------------+-----------------+------+----------+-------------+
```


type	meaning
`system`	table only has one row
`const`	table has at most one matching row
`eq_ref`	one row is read from this table for each combination of rows from previous tables
`ref`	all rows with matching values in this table are read for each combination of rows in previous table
`range`	use index to retrieve rows according to a range condition
`index`	full index scan is done for each combination of rows from previous tables
`ALL`	full table scan for each combination of rows from previous tables

listed fastest (cheapest) to slowest (most expensive)

Identify the join type for each table.

B [Choose]

R [Choose]

S [Choose]


```
+-------+--------+-----------------+-----+-----+------+----------+-----------------------------------------------+
| table | type   | possible_keys   | key | ref | rows | filtered | Extra                                         |
+-------+--------+-----------------+-----+-----+------+----------+-----------------------------------------------+
| S     | ALL    | PRIMARY         | NULL| NULL|   10 |   100.00 | NULL                                          |
| R     | ALL    | PRIMARY,        | NULL| NULL|   10 |    25.00 | Using where;                                  |
|       |        | fk_RESERVATION_1,|    |     |      |          | Using join buffer (Block Nested Loop)         |
|       |        | fk_RESERVATION_2 |    |     |      |          |                                               |
| B     | ALL    | PRIMARY         | NULL| NULL|    4 |    25.00 | Using where;                                  |
|       |        |                 |     |     |      |          | Using join buffer (Block Nested Loop)         |
+-------+--------+-----------------+-----+-----+------+----------+-----------------------------------------------+
```

which of the EXPLAIN outputs indicates the most efficient processing of the query?

```
+-------+--------+-----------------+-----------------+-----------------+------+----------+-------------+
| table | type   | possible_keys   | key             | ref             | rows | filtered | Extra       |
+-------+--------+-----------------+-----------------+-----------------+------+----------+-------------+
| B     | ALL    | PRIMARY         | NULL            | NULL            |    4 |    25.00 | Using where |
| R     | ref    | PRIMARY,        | fk_RESERVATION_1| ex_sailors.B.Bid|    2 |   100.00 | NULL        |
|       |        | fk_RESERVATION_1,|                 |                 |      |          |             |
|       |        | fk_RESERVATION_2 |                 |                 |      |          |             |
| S     | eq_ref | PRIMARY         | PRIMARY         | ex_sailors.R.Sid|    1 |   100.00 | NULL        |
+-------+--------+-----------------+-----------------+-----------------+------+----------+-------------+
```

product of rows is estimate of number of rows examined to satisfy the query

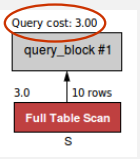
```
+-------+--------+-----------------+-----------------+-----------------+------+----------+---------------------+
| table | type   | possible_keys   | key             | ref             | rows | filtered | Extra               |
+-------+--------+-----------------+-----------------+-----------------+------+----------+---------------------+
| B     | ALL    | PRIMARY         | NULL            | NULL            |    4 |    25.00 | Using where         |
| S     | ALL    | PRIMARY         | NULL            | NULL            |   10 |   100.00 | Using join buffer   |
|       |        |                 |                 |                 |      |          | (Block Nested Loop) |
| R     | eq_ref | PRIMARY,        | PRIMARY         | ex_sailors.S.Sid,|    1 |   100.00 | NULL                |
|       |        | fk_RESERVATION_1,|                 | ex_sailors.B.Bid|      |          |                     |
|       |        | fk_RESERVATION_2 |                 |                 |      |          |                     |
+-------+--------+-----------------+-----------------+-----------------+------+----------+---------------------+
```


Comparing Execution Plans

```
SELECT *
FROM SAILOR S
WHERE S.Sname='Dustin'
```

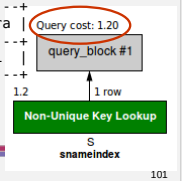
- with no index on `Sname` – must scan whole file

```
+-------------+-------+------+----+------+------+----------+-------------+
| select_type | table | type |key | ref  | rows | filtered | Extra       |
+-------------+-------+------+----+------+------+----------+-------------+
| SIMPLE      | S     | ALL  |    | NULL |   10 |    10.00 | Using where |
+-------------+-------+------+----+------+------+----------+-------------+
```



- with an index on `Sname` – can use index

```
+-------------+-------+------+-----------+------+------+----------+-------+
| select_type | table | type |key        | ref  | rows | filtered | Extra |
+-------------+-------+------+-----------+------+------+----------+-------+
| SIMPLE      | S     | ref  |snameindex | const|    1 |   100.00 | NULL  |
+-------------+-------+------+-----------+------+------+----------+-------+
```

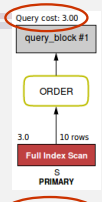


Comparing Execution Plans

```
SELECT *
FROM SAILOR S
ORDER BY S.Sid
```

the file is already in order by Sid and so can just be scanned – no need for additional sort
(EXPLAIN treats this as an index scan because the primary index is combined with the data file in InnoDB)

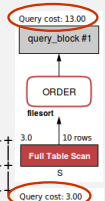
```
+----+-------------+-------+-------+---------+------+----------+-------+
| id | select_type | table | type  | key     | rows | filtered | Extra |
+----+-------------+-------+-------+---------+------+----------+-------+
|  1 | SIMPLE      | S     | index | PRIMARY |   10 |   100.00 | NULL  |
+----+-------------+-------+-------+---------+------+----------+-------+
```




```
SELECT *
FROM SAILOR S
ORDER BY S.Sname
```

the file is not in order by Sname, so the results must be sorted
scanning an index on Sname saves sorting, but accessing a data block for each record is much more expensive

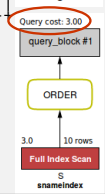
```
+----+-------------+-------+------+------+------+----------+----------------+
| id | select_type | table | type | key  | rows | filtered | Extra          |
+----+-------------+-------+------+------+------+----------+----------------+
|  1 | SIMPLE      | S     | ALL  | NULL |   10 |   100.00 | Using filesort |
+----+-------------+-------+------+------+------+----------+----------------+
```




```
SELECT Sname
FROM SAILOR S
ORDER BY S.Sname
```

with an index in order by Sname and only Sname desired in the results, the index can be scanned with no need to look at the file

```
+----+-------------+-------+-------+-------------+------+----------+-------------+
| id | select_type | table | type  | key         | rows | filtered | Extra       |
+----+-------------+-------+-------+-------------+------+----------+-------------+
|  1 | SIMPLE      | S     | index | snameindex  |   10 |   100.00 | Using index |
+----+-------------+-------+-------+-------------+------+----------+-------------+
```

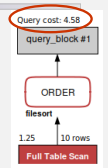


Comparing Execution Plans

```
SELECT *
FROM SAILOR S
WHERE S.Sname >= 'Dustin'
ORDER BY S.Sname
```

there is an index on Sname which could be used to satisfy the range search (and avoid sorting) but so many rows match the WHERE condition that it is better to just scan the file and sort the results

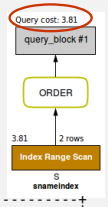
```
+----+-------------+-------+------+------+------+----------+----------------------------+
| id | select_type | table | type | key  | rows | filtered | Extra                      |
+----+-------------+-------+------+------+------+----------+----------------------------+
|  1 | SIMPLE      | S     | ALL  | NULL |   10 |    60.00 | Using where; Using filesort |
+----+-------------+-------+------+------+------+----------+----------------------------+
```




```
SELECT *
FROM SAILOR S
WHERE S.Sname >= 'Rusty'
ORDER BY S.Sname
```

this time the number of matched rows is smaller, so it is more efficient to scan the index (which is in sorted order!) and retrieve data blocks for just the matching rows

```
+----+-------------+-------+-------+-------------+------+----------+-------------------------+
| id | select_type | table | type  | key         | rows | filtered | Extra                   |
+----+-------------+-------+-------+-------------+------+----------+-------------------------+
|  1 | SIMPLE      | S     | range | snameindex  |    2 |   100.00 | Using index condition   |
+----+-------------+-------+-------+-------------+------+----------+-------------------------+
```



Optimizer Hints

- for overriding the optimizer's decisions

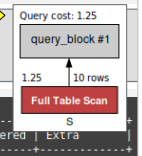
R STRAIGHT_JOIN *S*
  - R is guaranteed to be read first (optimizer won't swap the order of R and S)

FROM *table* [AS *alias]*
FORCE INDEX [FOR {JOIN | ORDER BY | GROUP BY} (*indexlist*)
  - assume that table scan (SL) is very expensive, and should be used only if no index is applicable

FROM *table* [AS *alias]*
USE INDEX [FOR {JOIN | ORDER BY | GROUP BY} (*indexlist*)
  - only use the indexes named

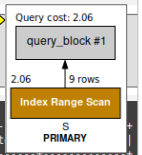
FROM *table* [AS *alias]*
IGNORE INDEX [FOR {JOIN | ORDER BY | GROUP BY} (*indexlist*)
  - don't use the indexes named


```
EXPLAIN SELECT *
FROM SAILOR S
WHERE S.Sname='Dustin'
```

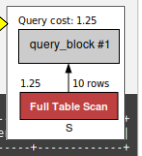



```
mysql> EXPLAIN SELECT * FROM SAILOR S WHERE S.Sname='Dustin';
+----+-------------+-------+------------+------+---------------+------+---------+------+------+----------+-------------+
| id | select_type | table | partitions | type | possible_keys | key  | key_len | ref  | rows | filtered | Extra       |
+----+-------------+-------+------------+------+---------------+------+---------+------+------+----------+-------------+
|  1 | SIMPLE      | S     | NULL       | ALL  | NULL          | NULL | NULL    | NULL |   10 |    10.00 | Using where |
+----+-------------+-------+------------+------+---------------+------+---------+------+------+----------+-------------+
```

```
EXPLAIN SELECT *
FROM SAILOR S
WHERE S.Sid>22
```




```
mysql> EXPLAIN SELECT * FROM SAILOR S WHERE S.Sid>22;
+----+-------------+-------+------------+-------+---------------+---------+---------+------+------+----------+-------------+
| id | select_type | table | partitions | type  | possible_keys | key     | key_len | ref  | rows | filtered | Extra       |
+----+-------------+-------+------------+-------+---------------+---------+---------+------+------+----------+-------------+
|  1 | SIMPLE      | S     | NULL       | range | PRIMARY       | PRIMARY | 2       | NULL |    9 |   100.00 | Using where |
+----+-------------+-------+------------+-------+---------------+---------+---------+------+------+----------+-------------+
```




```
mysql> EXPLAIN SELECT * FROM SAILOR S IGNORE INDEX FOR JOIN (PRIMARY) WHERE S.Sid>22;
+----+-------------+-------+------------+------+---------------+------+---------+------+------+----------+-------------+
| id | select_type | table | partitions | type | possible_keys | key  | key_len | ref  | rows | filtered | Extra       |
+----+-------------+-------+------------+------+---------------+------+---------+------+------+----------+-------------+
|  1 | SIMPLE      | S     | NULL       | ALL  | NULL          | NULL | NULL    | NULL |   10 |    33.33 | Using where |
+----+-------------+-------+------------+------+---------------+------+---------+------+------+----------+-------------+
```