RDBMS and Performance

MariaDB is a drop-in replacement for MySQL, the most used open source database for online applications. MariaDB falls into the category of the NewSQL products,
i.e. a product that provides unique NoSQL features together with the typical features available in relational databases.
Therefore, aspects like transaction management, durability and consistency are available together with schema or schema-less modeling,
full text storage and analysis and integration with other NoSQL technologies.

MariaDB can be part of the database infrastructure for Big Data. It is not meant to be a replacement for Hadoop, but it can be a technology used in conjunction with it. Hadoop is used in batch, ad-hoc analysis. In projects that require the processing of Terabytes or Petabytes of data,
Hadoop is definitely a good fit. The results can be queried and reported via a standard MySQL/MariaDB interface, which is compatible with virtually
all the BI tools and development frameworks available today.

Let’s take a detour to check out the differences.

RDBMS

So now lets take a detour and look at a RDBMS.

<img src="https://user-images.githubusercontent.com/558905/40613898-7a6c70d6-624e-11e8-9178-7bde851ac7bd.png" align=“left” width=“50” height=“50” title=“ToDo Logo” />
<h4>1. Log in to MySQL DB</h4>

Since it's loaded we'll use MySQL (may be MariaDB also):

```
[centos@ip-10-0-0-54 ~]$ mysql -u root -p
Enter password:
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MariaDB connection id is 207
Server version: 5.5.56-MariaDB MariaDB Server

Copyright (c) 2000, 2017, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.
MariaDB [(none)]>
```

> 💡 *password should be just [return]*

<img src="https://user-images.githubusercontent.com/558905/40613898-7a6c70d6-624e-11e8-9178-7bde851ac7bd.png" align=“left” width=“50” height=“50” title=“ToDo Logo” />

<h4>2. Now look at the databases</h4>

```
MariaDB [(none)]> show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| hive               |
| mysql              |
| performance_schema |
| test               |
+--------------------+
5 rows in set (0.00 sec)
```

<img src="https://user-images.githubusercontent.com/558905/40613898-7a6c70d6-624e-11e8-9178-7bde851ac7bd.png" align=“left” width=“50” height=“50” title=“ToDo Logo” />

<h4>2. And create some database, and use it</h4>

```
MariaDB [(none)]> create database class_test;
Query OK, 1 row affected (0.00 sec)
```

```
MariaDB [(none)]> use class_test;
Database changed
MariaDB [class_test]>
```

<img src="https://user-images.githubusercontent.com/558905/40613898-7a6c70d6-624e-11e8-9178-7bde851ac7bd.png" align=“left” width=“50” height=“50” title=“ToDo Logo” />
<h4>3. Let’s create a table</h4>

```
MariaDB [class_test]> create table users(id int, name char(20), PRIMARY KEY(id));
Query OK, 0 rows affected (0.00 sec)
```

And another table - this one will be related to the `users` table by the FOREIGN KEY `user_id`:

```
MariaDB [class_test]> create table orders(id int, user_id int, order_info text, PRIMARY KEY(id), FOREIGN KEY (user_id) REFERENCES users(id));
Query OK, 0 rows affected (0.00 sec)
```

<img src="https://user-images.githubusercontent.com/558905/40613898-7a6c70d6-624e-11e8-9178-7bde851ac7bd.png" align=“left” width=“50” height=“50” title=“ToDo Logo” />

<h4>4. Now insert a record</h4>

```
MariaDB [class_test]> insert into users (id, name) VALUES
(1, 'bill');
Query OK, 1 row affected (0.00 sec)
```

Select the table:

```
MariaDB [class_test]> select * from users;
+----+------+
| id | name |
+----+------+
|  1 | bill |
+----+------+
1 row in set (0.00 sec)
```

And we see the where clause is supported and so forth:

```
MariaDB [class_test]> select * from users where id < 2;
+----+------+
| id | name |
+----+------+
|  1 | bill |
+----+------+
1 row in set (0.00 sec)
```

<img src="https://user-images.githubusercontent.com/558905/40613898-7a6c70d6-624e-11e8-9178-7bde851ac7bd.png" align=“left” width=“50” height=“50” title=“ToDo Logo” />

<h4>5. Create a row in the related table</h4>

```
MariaDB [class_test]> insert into orders (id, user_id, ord
er_info) VALUES (1,1,"something");
Query OK, 1 row affected (0.00 sec)
```

And check it:

```
MariaDB [class_test]> select * from orders;
+----+---------+------------+
| id | user_id | order_info |
+----+---------+------------+
|  1 |       1 | something  |
+----+---------+------------+
1 row in set (0.00 sec)
```

```
MariaDB [class_test]> insert into orders (id, user_id, ord
er_info) VALUES (2,1,"something");
Query OK, 1 row affected (0.00 sec)
```

So you see the way `2` becomes another order number, no problem:

```
MariaDB [class_test]> select * from orders;
+----+---------+------------+
| id | user_id | order_info |
+----+---------+------------+
|  1 |       1 | something  |
|  2 |       1 | something  |
+----+---------+------------+
2 rows in set (0.00 sec)
```

But what if we do something like this:

```
MariaDB [class_test]> insert into orders (id, user_id, ord
er_info) VALUES (1,2,"something");
```

What has happened?

RDBMS and Transactions

<img src="https://user-images.githubusercontent.com/558905/40613898-7a6c70d6-624e-11e8-9178-7bde851ac7bd.png" align=“left” width=“50” height=“50” title=“ToDo Logo” />

<h4>6. Do the same with an active Transaction</h4>

```
MariaDB [class_test]> start transaction;
Query OK, 0 rows affected (0.00 sec)
```

Now, delete an order:

```
MariaDB [class_test]> delete from orders where id=2;
Query OK, 1 row affected (0.01 sec)
```

And the order is now gone:

```
MariaDB [class_test]> select * from orders;
+----+---------+------------+
| id | user_id | order_info |
+----+---------+------------+
|  1 |       1 | something  |
+----+---------+------------+
1 row in set (0.00 sec)
```

But we made a mistake, so roll back the trransaction:

```
MariaDB [class_test]> rollback;
Query OK, 0 rows affected (0.00 sec)
```

```
MariaDB [class_test]> select * from orders;
+----+---------+------------+
| id | user_id | order_info |
+----+---------+------------+
|  1 |       1 | something  |
|  2 |       1 | something  |
+----+---------+-----------+
2 rows in set (0.00 sec)
```

NOSql Performance Indicators

So now let’s do some performance testing. We have a `.sql` file called `users.sql` in the directory above. :

<img src="https://user-images.githubusercontent.com/558905/40613898-7a6c70d6-624e-11e8-9178-7bde851ac7bd.png" align=“left” width=“50” height=“50” title=“ToDo Logo” />
<h4>7. Import into MySQL</h4>

```
[centos@ip-10-0-0-54 data]$ mysql -u root -p < users.sql
```

Now go into MySQL and see the table:

```
[centos@ip-10-0-0-54 data]$ mysql -u root -p
Enter password:

MariaDB [(none)]> show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| class_test         |
| hive               |
| mysql              |
| performance_schema |
| test               |
| user_data          |
+--------------------+
7 rows in set (0.00 sec)

MariaDB [(none)]> use user_data;
```

And count the table:

```
MariaDB [user_data]> select count(*) from users;
+----------+
| count(*) |
+----------+
|  1000000 |
+----------+
1 row in set (0.14 sec)
```

Rather large table, isn’t it?

<img src="https://user-images.githubusercontent.com/558905/40613898-7a6c70d6-624e-11e8-9178-7bde851ac7bd.png" align=“left” width=“50” height=“50” title=“ToDo Logo” />

<h4>8. Look at the Data</h4>

```
MariaDB [user_data]> select * from users limit 1 \G
*************************** 1. row ***********************
****
     id: 1
   name: Alexandre Sporer
address: 24594 Emmitt Locks, Greenfelderview, MT 48128
    dob: 2007-05-02
  phone: 07683017318
  state: MT
1 row in set (0.00 sec)
MariaDB [user_data]>
```

> *the '|G' just prints the data lengthwise*

You see that we have name, address, date of birth, phone and state in this table for 1,000,000 users.

Now see if there are any indexes on the table:

```
MariaDB [user_data]> show indexes from users \G;
*************************** 1. row ***********************
****
        Table: users
   Non_unique: 0
     Key_name: PRIMARY
 Seq_in_index: 1
```

```
  Column_name: id
    Collation: A
  Cardinality: 1002809
     Sub_part: NULL
       Packed: NULL
         Null:
   Index_type: BTREE
      Comment:
Index_comment:
1 row in set (0.00 sec)


ERROR: No query specified
```

> *if you see more than 1 index then drop the others:*

```
MariaDB [user_data]> drop index state on users;
Query OK, 0 rows affected (0.03 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

Now lets get a count by an un-indexed column:

```
MariaDB [user_data]> select count(*), state from users group by state \G
```

What were your results? We got:

```
...
*************************** 61. row ***********************
****
count(*): 15354
   state: WV
*************************** 62. row ***********************
****
count(*): 15277
   state: WY
62 rows in set (0.57 sec)
```

So, `0.57` seconds. Now let's run an `explain` on the query:

```
MariaDB [user_data]> explain
    -> select count(*), state from users group by state \G
*************************** 1. row ************************
****
           id: 1
  select_type: SIMPLE
        table: users
         type: ALL
possible_keys: NULL
          key: NULL
      key_len: NULL
          ref: NULL
         rows: 999129
        Extra: Using temporary; Using filesort
```

```
1 row in set (0.00 sec)
```

So the size of the table means that `Using temporary; Using filesort` is used by MySQL to run the query.

If you add some criteria on a column in the query, this is the `EXPLAIN`:

```
MariaDB [user_data]> explain select count(*), state from u
sers where state = 'CA' group by state \G
*************************** 1. row ***********************
****
           id: 1
  select_type: SIMPLE
        table: users
         type: ALL
possible_keys: NULL
          key: NULL
      key_len: NULL
          ref: NULL
         rows: 999129
        Extra: Using where
1 row in set (0.00 sec)
```

`Using temporary; Using filesort` is gone. Now if you run the query:

```
MariaDB [user_data]> select count(*), state from users whe
re state = 'CA' group by state \G
```

```
*************************** 1. row ***********************
****
count(*): 15195
   state: CA
1 row in set (0.24 sec)
```

Runtime is approximately `0.24` seconds.

<img src="https://user-images.githubusercontent.com/558905/40613898-7a6c70d6-624e-11e8-9178-7bde851ac7bd.png" align=“left” width=“50” height=“50” title=“ToDo Logo” />
<h4>9. Index the Column</h4>

Now, what if we index that column:

```
MariaDB [user_data]> alter table users add index(state);
Query OK, 0 rows affected (2.30 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

So now a b-tree structure is built to get only the rows matching the states we want.

Re-run the query:

```
MariaDB [user_data]> select count(*), state from users whe
re state = 'CA' group by state \G
```

```
*************************** 1. row ***************************
count(*): 15195
   state: CA
```

What are the results now?

Results

So we have seen where SQL databases (like MariaDB/MySQL) need to index a row by setting up an in-memory structure to make them perform better.

<button type=“button”><a href="https://virtuant.github.io/hadoop-overview-spark-hwx/">Go Back</a></button>
<br>
<br>