Basic HBase Shell Queries

As we know, HBase is a non-relational database, so it does not support SQL. Instead, it uses its query engine to perform operations like fetch, update, and delete.

Before applying any operations, we need to first enter code inside the cluster to execute the shell commands:

```
Desktop> hbase shell
```

If we get the following in our terminal/command line, we are inside the HBase Shell and we are good to execute shell commands:

```
hbase(main):001:0>
```

Majorly used HBase Shell commands are as follows:

1. CREATE

The 'create' command is used to create a table within HBase Shell. We need to specify column families while creating the table.

```
hbase(main):001:0> create 'table_name', 'column_family1', 'column_family2'....
```

Let's create a table product with two column-families, "shoe" and "tshirt" —

```
hbase(main):001:0> create 'product', 'shoe', 'tshirt'
```

0 row(s) in 2.8130 seconds

2. LIST

The 'list' command is used to list all tables within the shell.

```
hbase(main):001:0> list
TABLE
product
2 row(s) in 0.0310 seconds
```

3. PUT

At a time, we can add or update only one column of one rowkey with a PUT query.

```
# Add or update color column of rowkey = 1
put 'TABLE_NAME','ROW_KEY','COLUMN_FAMILY:COLUMN','VALUE'
```

Let’s first insert some data and try to see the update

```
# Insertion Query
put 'product', '1', 'shoe:title', 'Adidas Shoe'
put 'product', '1', 'shoe:description', 'Running Shoes For Men'
put 'product', '1', 'shoe:color', 'black'
put 'product', '1', 'shoe:price', '40'
put 'product', '1', 'shoe:currency', 'USD'
put 'product', '1', 'shoe:size', '28'
put 'product', '2', 'tshirt:title', 'Puma Tshirt'
put 'product', '2', 'tshirt:description', 'Graphic Print Men Round Neck Pink T-Shirt'
put 'product', '2', 'tshirt:color', 'pink'
put 'product', '2', 'tshirt:price', '20'
put 'product', '2', 'tshirt:currency', 'USD'
put 'product', '2', 'tshirt:size', 'M'
```

```
hbase(main):003:0> put 'product', '1', 'shoe:title', 'Adidas Shoe'
0 row(s) in 0.3310 seconds

hbase(main):004:0> put 'product', '1', 'shoe:description', 'Running Shoes For Men'
0 row(s) in 0.0310 seconds

hbase(main):005:0> put 'product', '1', 'shoe:color', 'black'
0 row(s) in 0.0230 seconds

hbase(main):006:0> put 'product', '1', 'shoe:price', '40'
0 row(s) in 0.0170 seconds

hbase(main):007:0> put 'product', '1', 'shoe:currency', 'USD'
0 row(s) in 0.0170 seconds

hbase(main):008:0> put 'product', '1', 'shoe:size', '28'
0 row(s) in 0.0140 seconds

hbase(main):009:0> put 'product', '2', 'tshirt:title', 'Puma Tshirt'
0 row(s) in 0.0140 seconds

hbase(main):010:0> put 'product', '2', 'tshirt:description', 'Graphic Print Men Round Neck Pink T-Shirt'
0 row(s) in 0.0160 seconds

hbase(main):011:0> put 'product', '2', 'tshirt:color', 'pink'
0 row(s) in 0.0170 seconds

hbase(main):012:0> put 'product', '2', 'tshirt:price', '20'
0 row(s) in 0.0150 seconds

hbase(main):013:0> put 'product', '2', 'tshirt:currency', 'USD'
0 row(s) in 0.0130 seconds

hbase(main):014:0> put 'product', '2', 'tshirt:size', 'M'
0 row(s) in 0.0130 seconds
```

Now, let’s update product table with rowkey = 1, column_family = shoe, column = color.

```
put 'product','1','shoe:color','white'
```

```
hbase(main):016:0> put 'product','1','shoe:color','white'
0 row(s) in 0.0180 seconds

hbase(main):017:0> get 'product', '1'
COLUMN                                          CELL
 shoe:color                                     timestamp=1646794810972, value=white
 shoe:currency                                  timestamp=1646794268302, value=USD
 shoe:description                               timestamp=1646794237772, value=Running Shoes For Men
 shoe:price                                     timestamp=1646794250502, value=40
 shoe:size                                      timestamp=1646794274106, value=28
 shoe:title                                     timestamp=1646794230283, value=Adidas Shoe
6 row(s) in 0.0200 seconds
```

It will update the color column from black to white. The equivalent SQL query for a relational database is as follows—

```
UPDATE table_name
SET column_name = value
WHERE id = primary_key;
```

4. SCAN

Unlike "get", "scan" applies to all the rowkey and in turn all columns within the specified table.

```
# Get all data of a table
scan 'TABLE_NAME'
```

Example

```
scan 'product'
```

```
hbase(main):019:0> scan 'product'
ROW                                     COLUMN+CELL
 1                                      column=shoe:color, timestamp=1646848939510, value=white
 1                                      column=shoe:currency, timestamp=1646848843703, value=USD
 1                                      column=shoe:description, timestamp=1646848819102, value=Running Shoes For Men
 1                                      column=shoe:price, timestamp=1646848836595, value=40
 1                                      column=shoe:size, timestamp=1646848850512, value=28
 1                                      column=shoe:title, timestamp=1646848812395, value=Adidas Shoe
 2                                      column=tshirt:color, timestamp=1646848872613, value=pink
 2                                      column=tshirt:currency, timestamp=1646848887778, value=USD
 2                                      column=tshirt:description, timestamp=1646848866744, value=Graphic Print Men Round Neck Pink T-Shirt
 2                                      column=tshirt:price, timestamp=1646848882064, value=20
 2                                      column=tshirt:size, timestamp=1646848893553, value=M
 2                                      column=tshirt:title, timestamp=1646848859925, value=Puma Tshirt
2 row(s) in 0.0750 seconds
```

The equivalent SQL query for a relational database is as follows—

```
SELECT * FROM table_name
```

We can also apply filters to SCAN queries.

5. GET

The "get" keyword is used to fetch data associated with a particular row key.

```
# Get all data associated with on row key
get 'TABLE_NAME', 'ROW_KEY'
```

Example:

```
# Get product data associated with rowKey=1
get 'product', '1'
```

```
hbase(main):020:0> get 'product', '1'
COLUMN                                  CELL
 shoe:color                             timestamp=1646848939510, value=white
 shoe:currency                          timestamp=1646848843703, value=USD
 shoe:description                       timestamp=1646848819102, value=Running Shoes For Men
 shoe:price                             timestamp=1646848836595, value=40
 shoe:size                              timestamp=1646848850512, value=28
 shoe:title                             timestamp=1646848812395, value=Adidas Shoe
6 row(s) in 0.0250 seconds

hbase(main):021:0>
```

The above query fetches all the columns associated with one rowkey. The equivalent SQL query for relational databases is:

SELECT * FROM table_name WHERE id = primary_key

If we want to query one particular column or aggregate of multiple columns, we need to **apply filters**.

6. LIMIT

If we want to get first n rows' data, LIMIT fits in perfectly. This is applicable only for SCAN.

```
# Fetch all columns for n rows
scan 'TABLE_NAME', {LIMIT => n}
```

where 'n' is any positive integer.

Example:

```
# Fetch all columns for first 1 row
scan 'product', {LIMIT => 1}
```

```
hbase(main):021:0> scan 'product', {LIMIT => 1}
ROW                                             COLUMN+CELL
 1                                              column=shoe:color, timestamp=1646848939510, value=white
 1                                              column=shoe:currency, timestamp=1646848843703, value=USD
 1                                              column=shoe:description, timestamp=1646848819102, value=Running Shoes For Men
 1                                              column=shoe:price, timestamp=1646848836595, value=40
 1                                              column=shoe:size, timestamp=1646848850512, value=28
 1                                              column=shoe:title, timestamp=1646848812395, value=Adidas Shoe
1 row(s) in 0.0410 seconds

hbase(main):022:0>
```

7. DELETE TABLE

We cannot delete the HBase table directly. First, we need to disable the table, then we can delete it safely.

```
disable 'table_name'
drop 'table_name'
```

Example:

```
disable 'product'
drop 'product'
```

```
hbase(main):022:0> disable 'product'
0 row(s) in 2.3750 seconds

hbase(main):023:0> drop 'product'
0 row(s) in 1.3250 seconds

hbase(main):024:0>
```