Big Data Analytics - 2020

<u>Project: HBase architecture and role in Big Data along with extensive hands on exercise with use case</u>

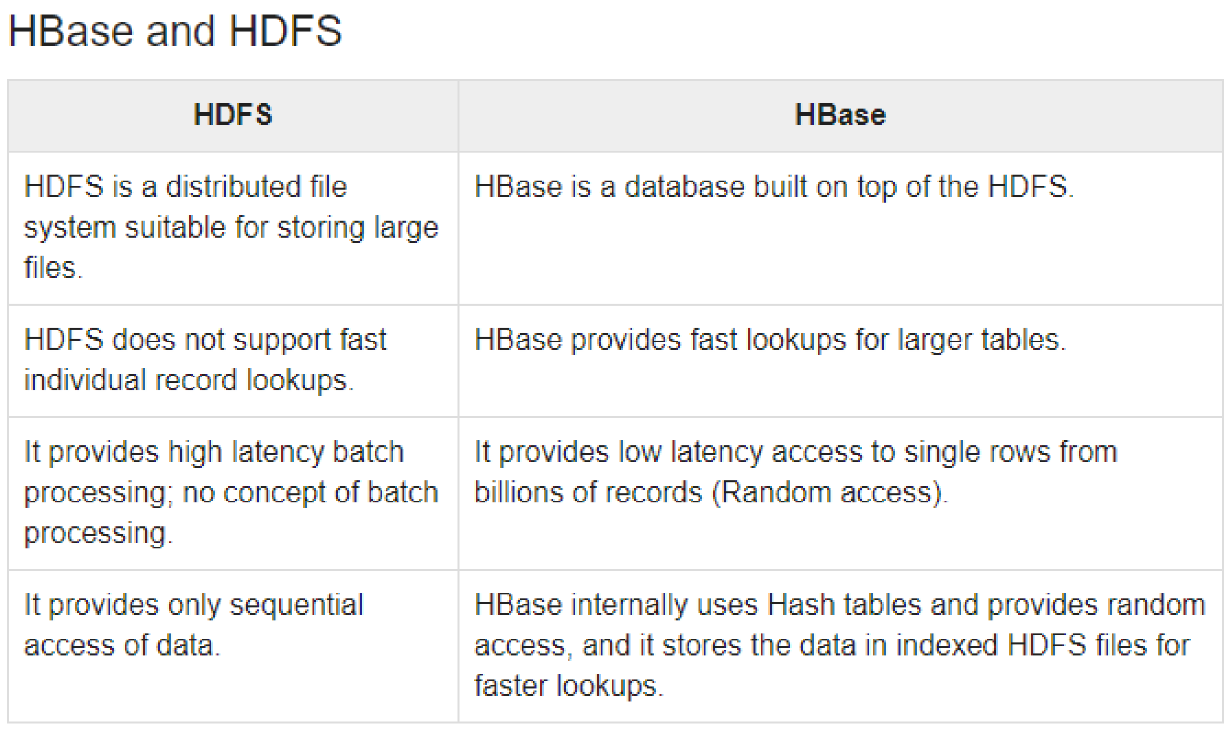
Project members: Rabiya Owais 05261 & Um E Rabna Bajwa 23374

10th January 2021

Table of Contents:

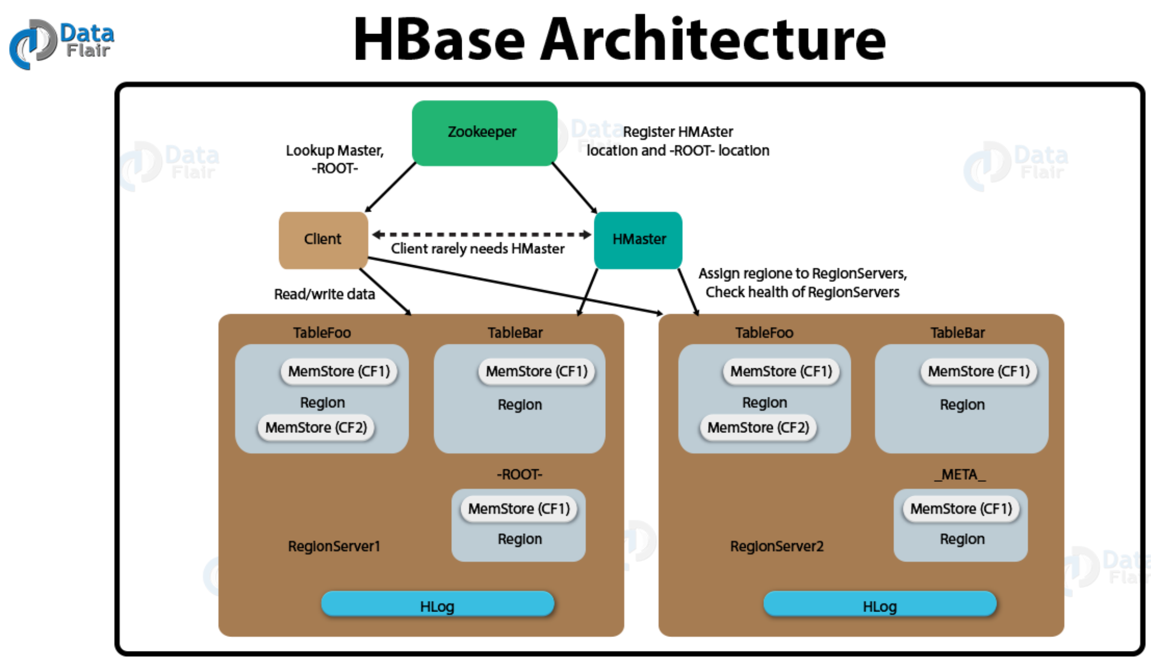
About HBase:

HBase is a column-oriented database designed for Big Data solutions; which means it stores data tables by columns rather than rows. This makes data retrieval faster. Tables in it are however, sorted row wise and table schema is defined only through column families which are in other words the attributes. The column families work as key-value pairs.



HBase and HDFS

HDFS	HBase
HDFS is a distributed file system suitable for storing large files.	HBase is a database built on top of the HDFS.
HDFS does not support fast individual record lookups.	HBase provides fast lookups for larger tables.
It provides high latency batch processing; no concept of batch processing.	It provides low latency access to single rows from billions of records (Random access).
It provides only sequential access of data.	HBase internally uses Hash tables and provides random access, and it stores the data in indexed HDFS files for faster lookups.



Benefits of using HBase:

Apache HBase is linearly scalable.

It provides automatic failure support.

It also offers consistent read and writes.

We can integrate it with Hadoop, both as a source as well as the destination.

Also, it has easy java API for the client.

HBase also offers data replication across clusters.

Most common use cases of HBase

Use cases of Apache HBase are:

- Random, real-time read/write access to **Big Data**
- It is possible to host very large tables on top of clusters of commodity hardware with Apache HBase.

- After Google’s Bigtable, HBase is a non-relational database modeled. Basically, as Bigtable acts up on Google File System, in same way HBase works on top of Hadoop and HDFS.

*Note: HBase permits **Java** API in communicating with HBase because HBase is written in Java.*

General Commands Lab:

Starting HBase Shell:

Command: **`hbase shell`**

```
HBase Shell; enter 'help<RETURN>' for list of supported commands.
Type "exit<RETURN>" to leave the HBase Shell
Version 0.94.23, rf42302b28aceaab773b15f234aa8718fff7eea3c, Wed Aug 27
00:54:09 UTC 2014

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

Exiting HBase Shell:

Command: **`exit`**

View existing tables in HBase:

Command: **`list`**

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

View existing servers in HBase:

Command: **`status`**

```
hbase(main):009:0> status
3 servers, 0 dead, 1.3333 average load
```

View version of HBase in use:

Command: **`version`**

```
hbase(main):009:0> version
0.98.8-hadoop2, r6cfc8d064754251365e070a10a82eb169956d5fe, Fri Nov 14
```

To open guide for tables in HBase:

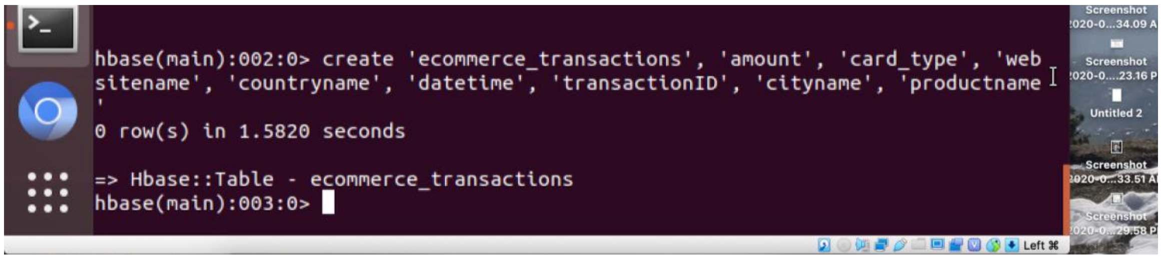
Command: **table_help**

```
hbase(main):002:0> table_help
Help for table-reference commands.
You can either create a table via 'create' and then manipulate the table
via commands like 'put', 'get', etc.
See the standard help information for how to use each of these commands.
However, as of 0.96, you can also get a reference to a table, on which
you can invoke commands.
For instance, you can get create a table and keep around a reference to
it via:
 hbase> t = create 't', 'cf'.....
```

Create table in HBase:

Command:

```
create'ecommerce_transactions,'amount','card_type',
'websitename', 'countryname', 'datetime', 'transactionID',
'cityname', 'productname'
```

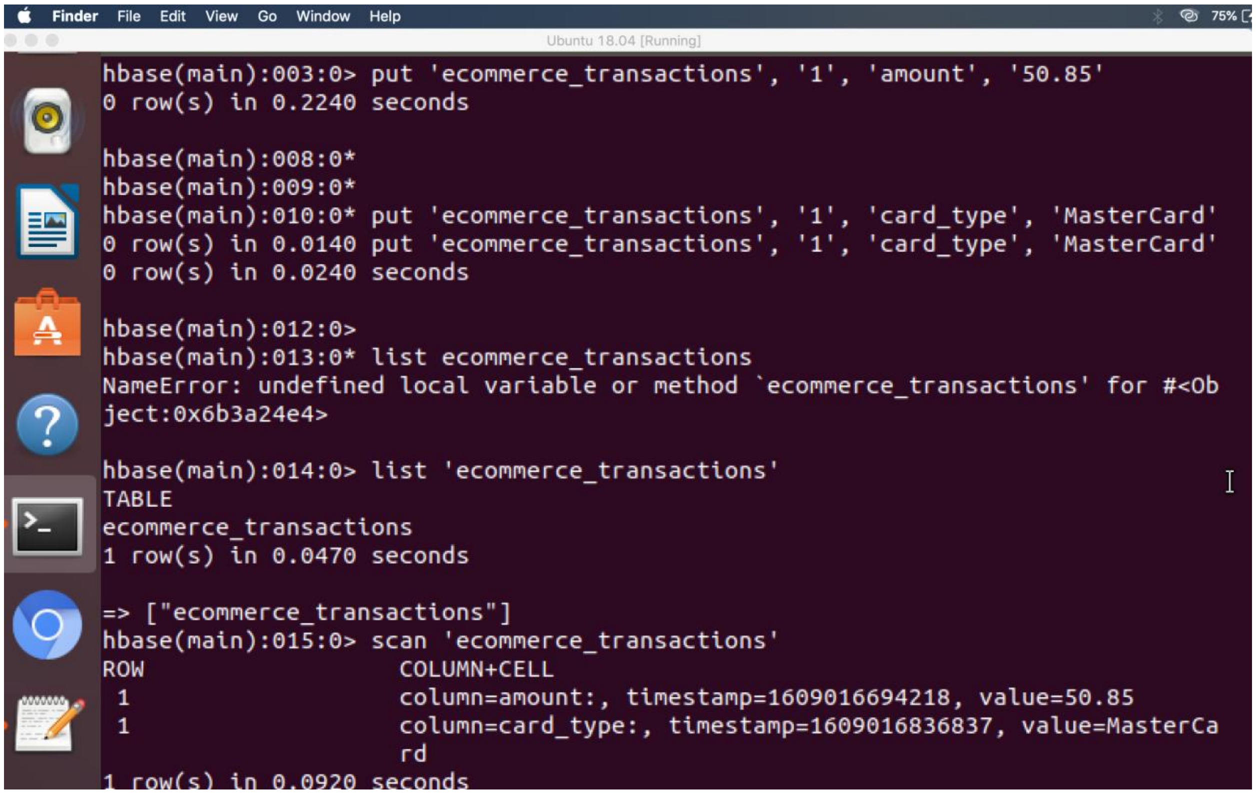


```
hbase(main):002:0> create 'ecommerce_transactions', 'amount', 'card_type', 'web
sitename', 'countryname', 'datetime', 'transactionID', 'cityname', 'productname
'
0 row(s) in 1.5820 seconds

=> Hbase::Table - ecommerce_transactions
hbase(main):003:0> 
```

Insert rows for tables in HBase:

Command:

```
put put 'ecommerce_transactions', '2', 'card_type', 'MasterCard'

put 'ecommerce_transactions', '3', 'card_type', 'Visa'

put 'ecommerce_transactions', '4', 'card_type', 'MasterCard'

put 'ecommerce_transactions', '5', 'card_type', 'Maestro'


put 'ecommerce_transactions', '1', 'amount', '50.87'

put 'ecommerce_transactions', '2', 'amount', '1023.2'
put 'ecommerce_transactions', '3', 'amount', '3321.1'

put 'ecommerce_transactions', '4', 'amount', '234.11'

put 'ecommerce_transactions', '5', 'amount', '321.11'
```

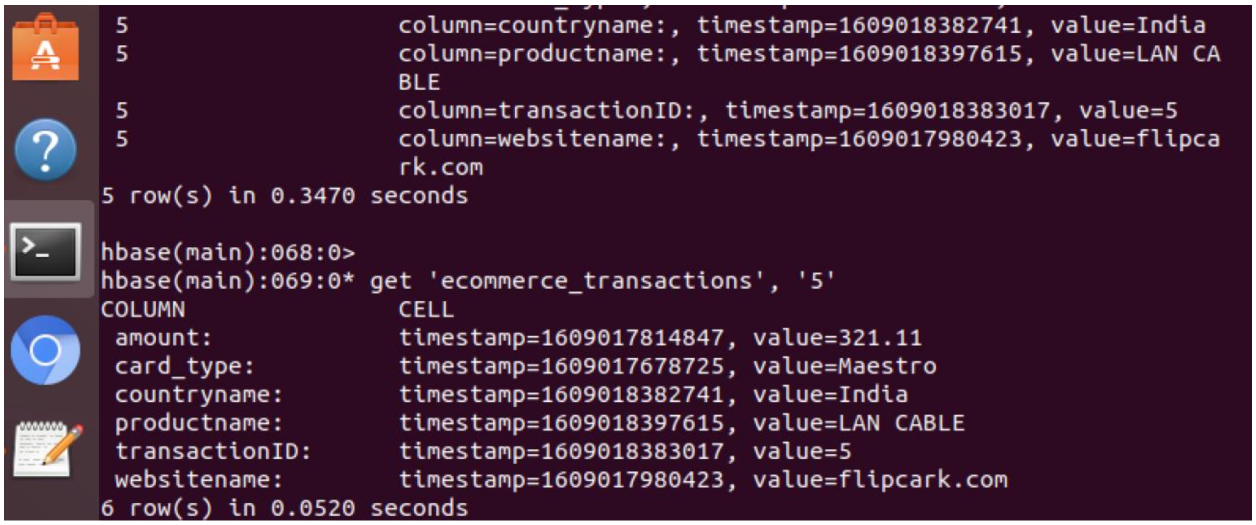


```
hbase(main):003:0> put 'ecommerce_transactions', '1', 'amount', '50.85'
0 row(s) in 0.2240 seconds

hbase(main):008:0*
hbase(main):009:0*
hbase(main):010:0* put 'ecommerce_transactions', '1', 'card_type', 'MasterCard'
0 row(s) in 0.0140 put 'ecommerce_transactions', '1', 'card_type', 'MasterCard'
0 row(s) in 0.0240 seconds

hbase(main):012:0>
hbase(main):013:0* list ecommerce_transactions
NameError: undefined local variable or method `ecommerce_transactions' for #<Ob
ject:0x6b3a24e4>

hbase(main):014:0> list 'ecommerce_transactions'
TABLE
ecommerce_transactions
1 row(s) in 0.0470 seconds

=> ["ecommerce_transactions"]
hbase(main):015:0> scan 'ecommerce_transactions'
ROW                   COLUMN+CELL
 1                    column=amount:, timestamp=1609016694218, value=50.85
 1                    column=card_type:, timestamp=1609016836837, value=MasterCa
                      rd
1 row(s) in 0.0920 seconds
```

Get transactions for only row value 2 and 3 in HBase:

Command:

```
get 'ecommerce_transactions', '2'

get 'ecommerce_transactions', '5'
```



```
 5                    column=countryname:, timestamp=1609018382741, value=India
 5                    column=productname:, timestamp=1609018397615, value=LAN CA
                      BLE
 5                    column=transactionID:, timestamp=1609018383017, value=5
 5                    column=websitename:, timestamp=1609017980423, value=flipca
                      rk.com
5 row(s) in 0.3470 seconds

hbase(main):068:0>
hbase(main):069:0* get 'ecommerce_transactions', '5'
COLUMN                CELL
 amount:              timestamp=1609017814847, value=321.11
 card_type:           timestamp=1609017678725, value=Maestro
 countryname:         timestamp=1609018382741, value=India
 productname:         timestamp=1609018397615, value=LAN CABLE
 transactionID:       timestamp=1609018383017, value=5
 websitename:         timestamp=1609017980423, value=flipcark.com
6 row(s) in 0.0520 seconds
```

Get description of the table in HBase:
Command:

```
describe 'ecommerce_transactions'
```

Alter is the command used to make changes to an existing table.

This command can help change the maximum number of cells of a column family, set and delete table scope operators, and delete a column family from a table:

Command:

```
alter 'ecommerce_transactions', NAME => ‘card_type’, VERSIONS =>
5
```

Deleting a cell in a table in HBase:
Command:

```
delete 'ecommerce_transactions', ‘4’, ‘card_type’
```

Deleting a row in a table in HBase:
Command:

```
deleteall 'ecommerce_transactions', ‘4’
```

Count number of rows in a table in HBase:
Command:

```
count 'ecommerce_transactions'
```

Disable, drop and recreate a table in HBase:

Command:

```
truncate 'ecommerce_transactions'
```

Check if a table is present in HBase:

Command:

```
exists 'ecommerce_transactions'
```

View all rows in a table of HBase:

Command:

```
scan 'ecommerce_transactions'
```

Disable a table of HBase:

Command:

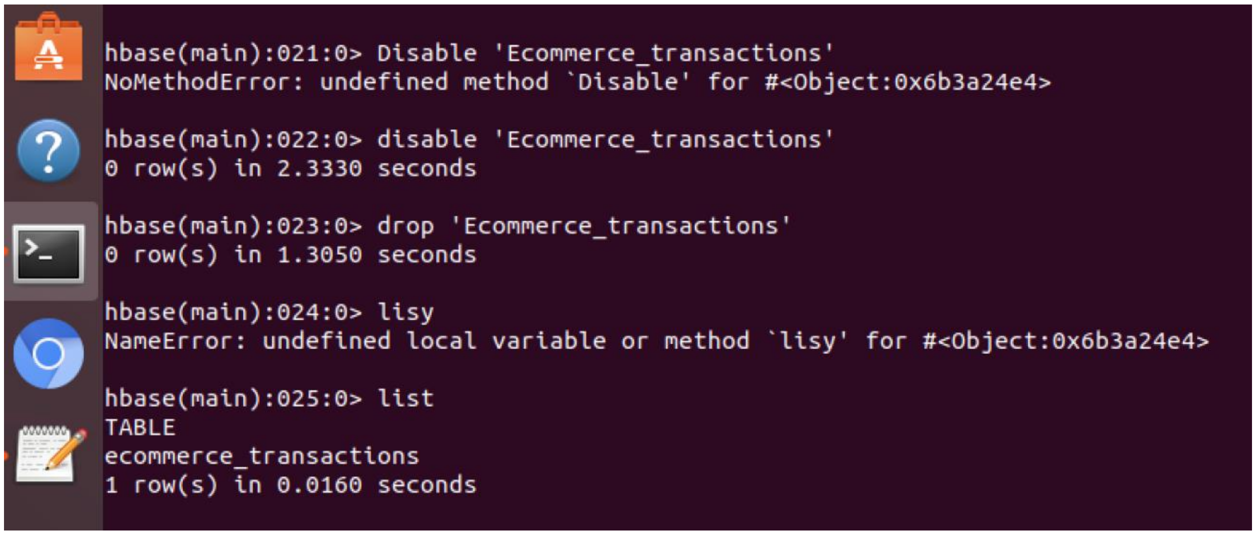
```
disable 'ecommerce_transactions'
```

Drop a table of HBase:

Command:

```
drop 'ecommerce_transactions'
```

Note: drop won't work till table has been disabled



```
hbase(main):021:0> Disable 'Ecommerce_transactions'
NoMethodError: undefined method `Disable' for #<Object:0x6b3a24e4>

hbase(main):022:0> disable 'Ecommerce_transactions'
0 row(s) in 2.3330 seconds

hbase(main):023:0> drop 'Ecommerce_transactions'
0 row(s) in 1.3050 seconds

hbase(main):024:0> lisy
NameError: undefined local variable or method `lisy' for #<Object:0x6b3a24e4>

hbase(main):025:0> list
TABLE
ecommerce_transactions
1 row(s) in 0.0160 seconds
```

Scan Filter: Concept and Lab

Using the data inserted in 'ecommerce_transactions' table, enhancing reads of data present in HBase using the scan filtering techniques. There are 6 most commonly used filters, but these can also be combined together for more specific searches.

Check for available filters in HBase to read data more specifically:

Command: `show_filters` (enter in HBase shell)

Using the data set we put earlier perform the following hands-on exercise:

Table:

'ecommerce_transactions'

Columns:

'amount',

'card_type',

'websitename',

 'countryname',

'datetime',

'transactionID',

'cityname',

'productname'

Commands to perform:

Using KeyOnlyFilter get the names of columns and column family qualifiers without seeing their values (typically used for a dataset with very large values and we just want to see the attributes):

Command:
```
scan 'ecommerce_transactions', {FILTER =>
"KeyOnlyFilter()"}
```

Using the PrefixFilter to find any pattern associated with our row keys:

Check which row key represents which website for transactions?

Command: `scan 'ecommerce_transactions', {FILTER => "PrefixFilter('1')"}`

Combining filters:

Suppose now we want to see which column families are on row key 2? Then we combine the two filters such that:

Command: `scan 'ecommerce_transactions',{FILTER => "PrefixFilter('2') AND KeyOnlyFilter()"}`

Using the ColumnPrefixFilter, find the column families which start with 'c'

Command: `scan 'ecommerce_transactions', {FILTER => "ColumnPrefixFilter('c')"}`

Similarly, specify two arguments in this filter and find all column families which start with 'c' or 'p'

Command: `scan 'ecommerce_transactions', {FILTER => "MultipleColumnPrefixFilter('c','p')"}`

Using the PageFilter, read first two rows of the HBase table 'ecommerce_transactions':

Command: `scan 'ecommerce_transactions', {FILTER => "PageFilter(2)"}`

Combining this with another filter, get 3 rows with the column family starting with the letter 'c'

Command: `scan 'ecommerce_transactions', {FILTER => "PageFilter(3) AND ColumnPrefixFilter('c')"}`

Using the InclusiveStopFilter, scan from start till the specified row 4 (including 4 in it)

Command:
```
scan 'ecommerce_transactions', {FILTER =>
"InclusiveStopFilter('4')"}
```

Now use the ValueFilter to do some basic search on values of the data. This works similar to the 'like' command of SQL.

Find the card type starting with 'M'

Command:
```
scan 'ecommerce_transactions', {COLUMNS => 'card_type',
FILTER => "ValueFilter(=,'binary prefix:M')"}
```

Return all countries starting with letters > 'I':

Command:
```
scan 'ecommerce_transactions', {COLUMNS =>
'countryname', FILTER => "ValueFilter(>,'binary prefix:I')"}
```

Using the STARTROW, scan only rows from row key 3 onwards:

Command:
```
scan 'ecommerce_transactions', {STARTROW => '3'}
```

Use Case and Lab:
“To implement HBase for big data storage in NoSQL form and run that with integration to Hive HQL language in order to reuse the existing SQL queries.”

Suppose an organization has a retail business with shops spread across multiple locations in a country. Currently the business operates with local SQL databases for their data storage and analysis purposes and has no online shopping mechanism.

Given a situation such as Covid19 pandemic, the organization decides to move towards an e-commerce based solution to keep their business continuity. Hence all their stores and products are moved online.

In order to address this quickly, the organization needs to implement a big data solution to cater to their ever-increasing customer transactional data and also continue using the same queries they have already designed.

A quick big data solution for this scenario is:

To implement HBase for big data storage in NoSQL form and run that with integration to Hive HQL language in order to reuse the existing SQL queries.

Benefits:

- HBase is open source and scalable hence it will be ideal to implement it for big data storage purposes.
- HBase provides user with random read/write access over HDFS making operations efficient.
- Using this solution also allows flexible schema and doesn’t require any pre-provisioned meta data.
- It can be integrated with Java and Rest APIs to programmatically access, alter and store tables.

Demo:

To demo this scenario, we will be using a dataset CSV format which has the following fields. Data set has been picked from Kaggle:

- Transaction ID
- Card Type
- Website name

- Product
- Transaction Amount
- Customer ID
- Country of origin

Pre-requisites for this demo:

1. Ensure HBase for Linux has been installed
2. Ensure Hive for Linux has been installed
3. Ensure Hadoop for Linux has been installed

Incase these three aren't individually installed on computer, then

- Docker container for cloudera can be used to connect to the Hadoop ecosystem environment.

Starting Cloudera container:

Command:

```
rabiyaowais@rabiyaowais-VirtualBox:~$ sudo docker run --
hostname=quickstart.cloudera --privileged=true -t -v
/home/rabiyaowais/Desktop/Dataset:/user/cloudera/shared -i -p
8889:8888 -p 7180:7181 cloudera/quickstart /usr/bin/docker-
quickstart
```

Starting HBase shell:

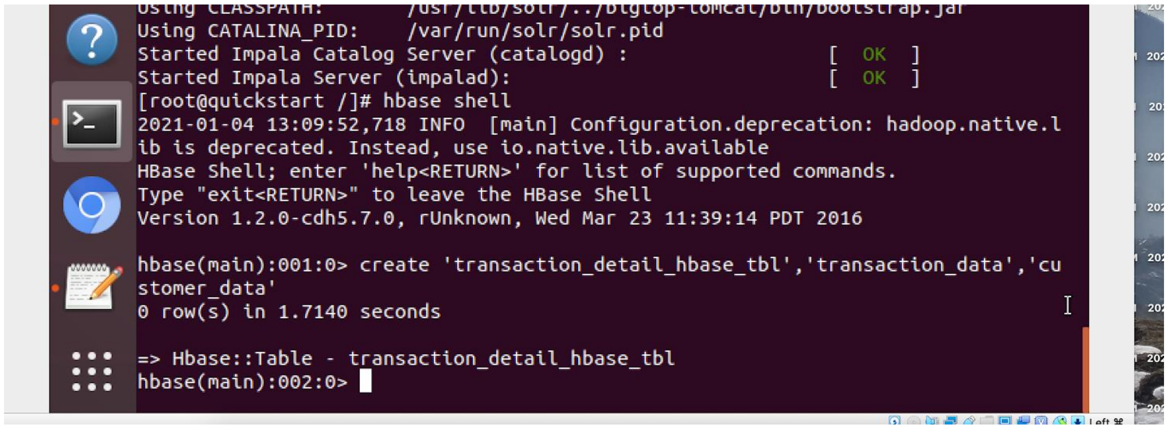
Command: `HBase shell`

Creating table in HBase for e-commerce transactions data:

Command:

```
create'transaction_detail_HBase_tbl','transaction_data','custome
r_data'
```

Note: the column families are declared in the create table command in order to create it successfully in HBase shell.



```
Using CLASSPATH:        /usr/lib/solr/../bigtop-tomcat/bin/bootstrap.jar
Using CATALINA_PID:     /var/run/solr/solr.pid
Started Impala Catalog Server (catalogd) :                 [  OK  ]
Started Impala Server (impalad):                           [  OK  ]
[root@quickstart /]# hbase shell
2021-01-04 13:09:52,718 INFO  [main] Configuration.deprecation: hadoop.native.l
ib is deprecated. Instead, use io.native.lib.available
HBase Shell; enter 'help<RETURN>' for list of supported commands.
Type "exit<RETURN>" to leave the HBase Shell
Version 1.2.0-cdh5.7.0, rUnknown, Wed Mar 23 11:39:14 PDT 2016

hbase(main):001:0> create 'transaction_detail_hbase_tbl','transaction_data','cu
stomer_data'
0 row(s) in 1.7140 seconds

=> Hbase::Table - transaction_detail_hbase_tbl
hbase(main):002:0>
```

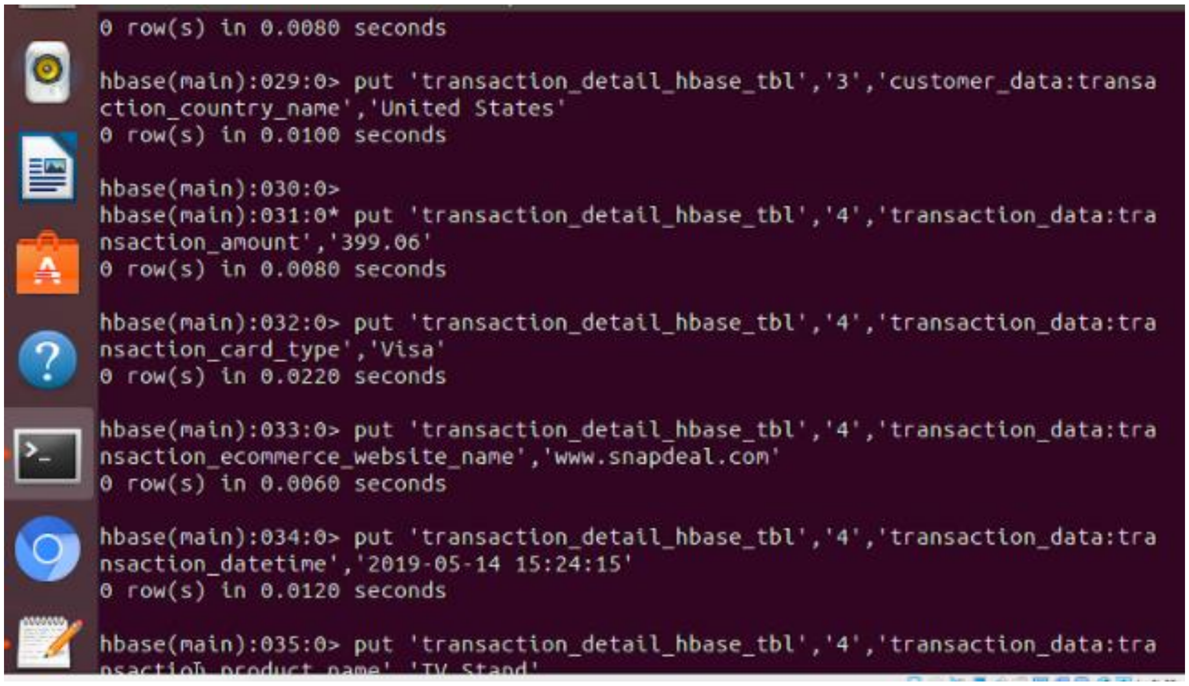
Entering some initial values to populate table:

Command:

```
put
'transaction_detail_HBase_tbl','1','transaction_data:transaction
_amount','50.85'
put
'transaction_detail_HBase_tbl','1','transaction_data:transaction
_card_type','MasterCard'
put
'transaction_detail_HBase_tbl','1','transaction_data:transaction
_ecommerce_website_name','www.ebay.com'
put
'transaction_detail_HBase_tbl','1','transaction_data:transaction
_datetime','2019-05-14 15:24:12'
put
'transaction_detail_HBase_tbl','1','transaction_data:transaction
_product_name','Laptop'
put
'transaction_detail_HBase_tbl','1','customer_data:transaction_ci
ty_name','Mumbai'
put
'transaction_detail_HBase_tbl','1','customer_data:transaction_co
untry_name','India'

put
'transaction_detail_HBase_tbl','2','transaction_data:transaction
_amount','259.12'
put
'transaction_detail_HBase_tbl','2','transaction_data:transaction
_card_type','MasterCard'
```

```
put
'transaction_detail_HBase_tbl','2','transaction_data:transaction
_ecommerce_website_name','www.amazon.com'
put
'transaction_detail_HBase_tbl','2','transaction_data:transaction
_datetime','2019-05-14 15:24:13'
put
'transaction_detail_HBase_tbl','2','transaction_data:transaction
_product_name','Wrist Band'
put
'transaction_detail_HBase_tbl','2','customer_data:transaction_ci
ty_name','Pune'
put
'transaction_detail_HBase_tbl','2','customer_data:transaction_co
untry_name','India'

put
'transaction_detail_HBase_tbl','3','transaction_data:transaction
_amount','328.16'
put
'transaction_detail_HBase_tbl','3','transaction_data:transaction
_card_type','MasterCard'
put
'transaction_detail_HBase_tbl','3','transaction_data:transaction
_ecommerce_website_name','www.flipkart.com'
put
'transaction_detail_HBase_tbl','3','transaction_data:transaction
_datetime','2019-05-14 15:24:14'
put
'transaction_detail_HBase_tbl','3','transaction_data:transaction
_product_name','TV Stand'
put
'transaction_detail_HBase_tbl','3','customer_data:transaction_ci
ty_name','New York City'
put
'transaction_detail_HBase_tbl','3','customer_data:transaction_co
untry_name','United States'

put
'transaction_detail_HBase_tbl','4','transaction_data:transaction
_amount','399.06'
put
'transaction_detail_HBase_tbl','4','transaction_data:transaction
_card_type','Visa'
put
'transaction_detail_HBase_tbl','4','transaction_data:transaction
_ecommerce_website_name','www.snapdeal.com'
```

```
put
'transaction_detail_HBase_tbl','4','transaction_data:transaction
_datetime','2019-05-14 15:24:15'
put
'transaction_detail_HBase_tbl','4','transaction_data:transaction
_product_name','TV Stand'
put
'transaction_detail_HBase_tbl','4','customer_data:transaction_ci
ty_name','New Delhi'
put
'transaction_detail_HBase_tbl','4','customer_data:transaction_co
untry_name','India'

put
'transaction_detail_HBase_tbl','5','transaction_data:transaction
_amount','194.52'
put
'transaction_detail_HBase_tbl','5','transaction_data:transaction
_card_type','Visa'
put
'transaction_detail_HBase_tbl','5','transaction_data:transaction
_ecommerce_website_name','www.ebay.com'
put
'transaction_detail_HBase_tbl','5','transaction_data:transaction
_datetime','2019-05-14 15:24:16'
put
'transaction_detail_HBase_tbl','5','transaction_data:transaction
_product_name','External Hard Drive'
put
'transaction_detail_HBase_tbl','5','customer_data:transaction_ci
ty_name','Rome'
put
'transaction_detail_HBase_tbl','5','customer_data:transaction_co
untry_name','Italy'
```

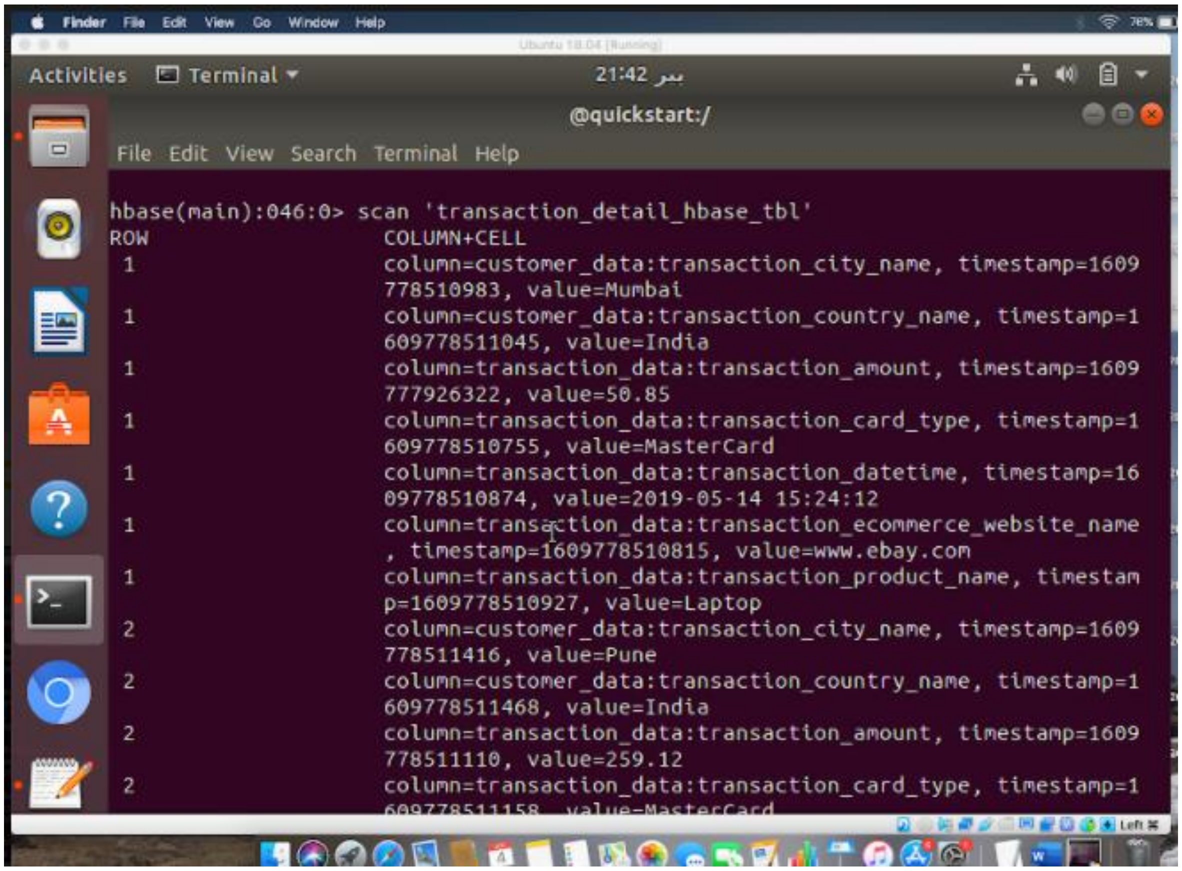


```
0 row(s) in 0.0080 seconds

hbase(main):029:0> put 'transaction_detail_hbase_tbl','3','customer_data:transa
ction_country_name','United States'
0 row(s) in 0.0100 seconds

hbase(main):030:0>
hbase(main):031:0* put 'transaction_detail_hbase_tbl','4','transaction_data:tra
nsaction_amount','399.06'
0 row(s) in 0.0080 seconds

hbase(main):032:0> put 'transaction_detail_hbase_tbl','4','transaction_data:tra
nsaction_card_type','Visa'
0 row(s) in 0.0220 seconds

hbase(main):033:0> put 'transaction_detail_hbase_tbl','4','transaction_data:tra
nsaction_ecommerce_website_name','www.snapdeal.com'
0 row(s) in 0.0060 seconds

hbase(main):034:0> put 'transaction_detail_hbase_tbl','4','transaction_data:tra
nsaction_datetime','2019-05-14 15:24:15'
0 row(s) in 0.0120 seconds

hbase(main):035:0> put 'transaction_detail_hbase_tbl','4','transaction_data:tra
nsaction_product_name','TV Stand'
```

View the populated table to see the rows entered in the previous step:

Command:

```
Scan 'transaction_detail_HBase_tbl'
```



```
hbase(main):046:0> scan 'transaction_detail_hbase_tbl'
ROW                    COLUMN+CELL
 1                     column=customer_data:transaction_city_name, timestamp=1609
                       778510983, value=Mumbai
 1                     column=customer_data:transaction_country_name, timestamp=1
                       609778511045, value=India
 1                     column=transaction_data:transaction_amount, timestamp=1609
                       777926322, value=50.85
 1                     column=transaction_data:transaction_card_type, timestamp=1
                       609778510755, value=MasterCard
 1                     column=transaction_data:transaction_datetime, timestamp=16
                       09778510874, value=2019-05-14 15:24:12
 1                     column=transaction_data:transaction_ecommerce_website_name
                       , timestamp=1609778510815, value=www.ebay.com
 1                     column=transaction_data:transaction_product_name, timestam
                       p=1609778510927, value=Laptop
 2                     column=customer_data:transaction_city_name, timestamp=1609
                       778511416, value=Pune
 2                     column=customer_data:transaction_country_name, timestamp=1
                       609778511468, value=India
 2                     column=transaction_data:transaction_amount, timestamp=1609
                       778511110, value=259.12
 2                     column=transaction_data:transaction_card_type, timestamp=1
                       609778511158, value=MasterCard
```

Once table has been populated, we exit the HBase shell and enter the Hive shell for integration purposes:

Exiting HBase Shell:

Command: `exit`

Next we enter Hive shell to make a connecting table to our HBase table.

Enter Hive shell:

Command:

```
hive shell
```

View existing databases in Hive shell:

Command:

```
show databases;
```

Create new database mydb300 in hive shell:

Command:

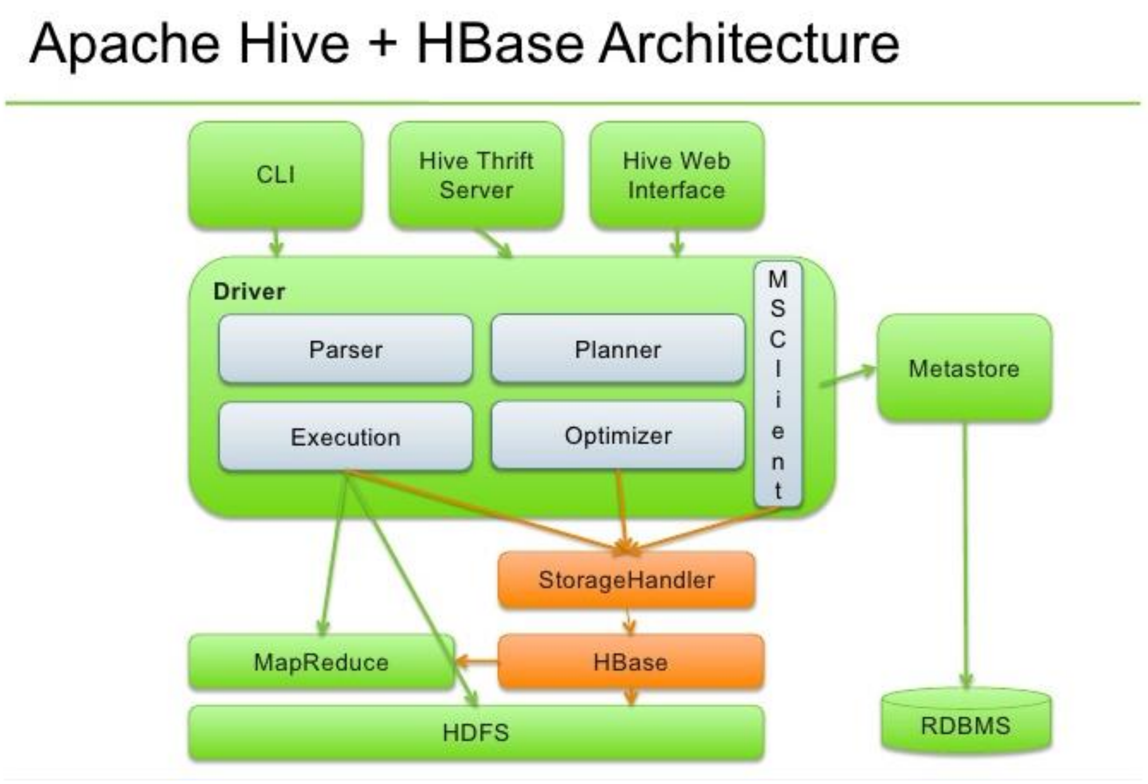
```
create database mydb300;
```

In order to activate database for usage, enter use database command in hive shell:

Command:

```
use mydb300;
```

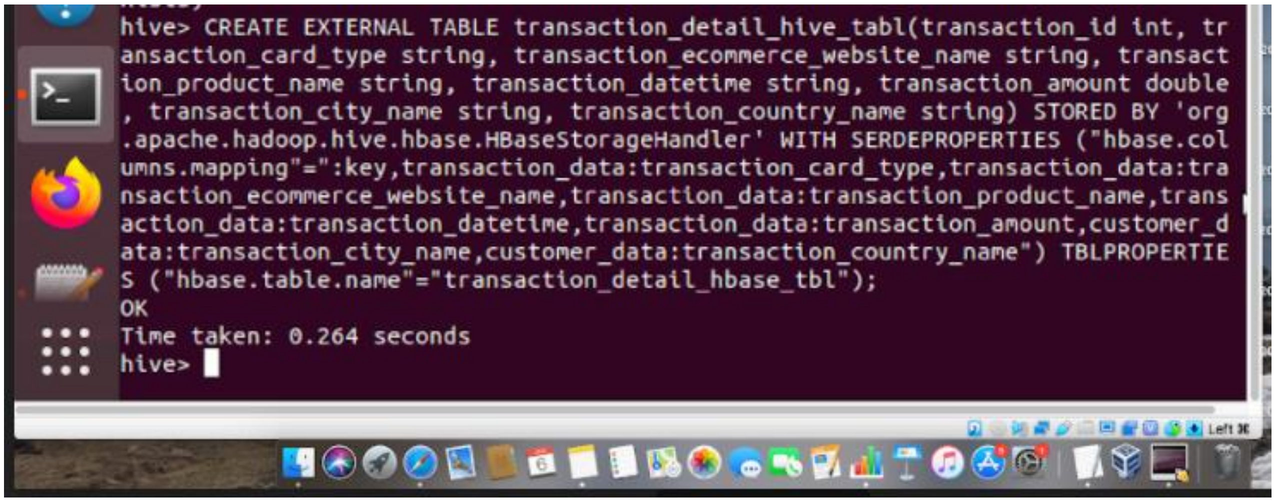
Now create an external table in hive which points to the table with data kept in HBase following an architecture such that:



Command:

```
CREATE EXTERNAL TABLE transaction_detail_hive_tbl(transaction_id
int, transaction_card_type string,
transaction_ecommerce_website_name string,
transaction_product_name string, transaction_datetime string,
transaction_amount double, transaction_city_name string,
transaction_country_name string)


STORED BY 'org.apache.hadoop.hive.HBase.HBaseStorageHandler'
```

```
WITH SERDEPROPERTIES
("HBase.columns.mapping"=":key,transaction_data:transaction_card
_type,transaction_data:transaction_ecommerce_website_name,transa
ction_data:transaction_product_name,transaction_data:transaction
_datetime,transaction_data:transaction_amount,customer_data:tran
saction_city_name,customer_data:transaction_country_name")


TBLPROPERTIES
("HBase.table.name"="transaction_detail_HBase_tbl");
```



```
hive> CREATE EXTERNAL TABLE transaction_detail_hive_tabl(transaction_id int, tr
ansaction_card_type string, transaction_ecommerce_website_name string, transact
ion_product_name string, transaction_datetime string, transaction_amount double
, transaction_city_name string, transaction_country_name string) STORED BY 'org
.apache.hadoop.hive.hbase.HBaseStorageHandler' WITH SERDEPROPERTIES ("hbase.col
umns.mapping"=":key,transaction_data:transaction_card_type,transaction_data:tra
nsaction_ecommerce_website_name,transaction_data:transaction_product_name,trans
action_data:transaction_datetime,transaction_data:transaction_amount,customer_d
ata:transaction_city_name,customer_data:transaction_country_name") TBLPROPERTIE
S ("hbase.table.name"="transaction_detail_hbase_tbl");
OK
Time taken: 0.264 seconds
hive>
```

How this integration works:

A new table is created in Hive database using the **create external table** command which has all attributes and datatypes declared in Hive in a similar way as SQL tables.

This table is linked with HBase using a **stored by** operation using the storage handler for HBase as depicted in the architecture diagram above.

Furthermore, this command also adds **serdeproperties** where we specify all the column families and column family qualifiers that have to be connected to in HBase listing them in the same sequence as listed in the attributes declared after the create external table command.

Lastly this command creates a link to the HBase table using **tblproperties**.

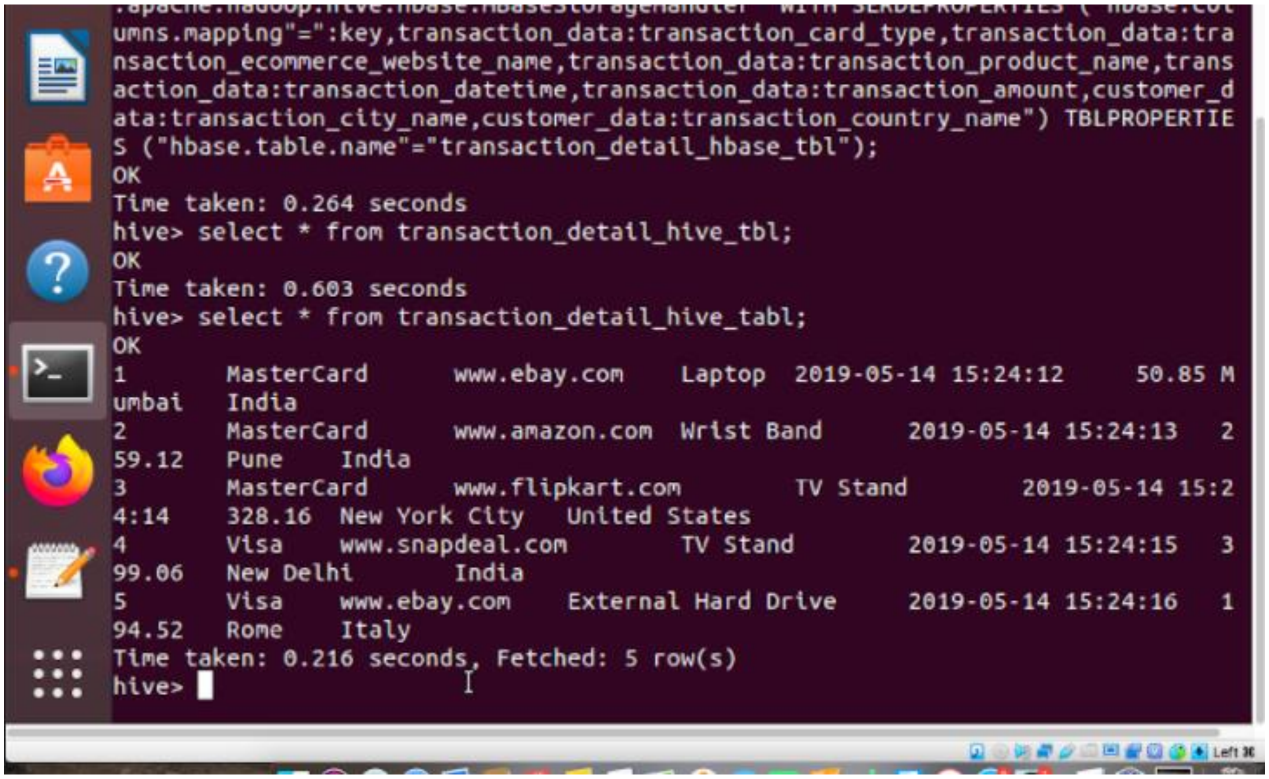
Note: HBase table may have many columns however, in Hive we do not have to fetch all columns, but just the columns we need for analysis.

Once this is executed, the data can be fetched in Hive using similar queries to SQL.

View the HBase connected table in Hive:

Command:

```
select * from transaction_detail_hive_tbl;
```



```
.apache.hadoop.hive.hbase.HBaseStorageHandler' WITH SERDEPROPERTIES ("hbase.col
umns.mapping"=":key,transaction_data:transaction_card_type,transaction_data:tra
nsaction_ecommerce_website_name,transaction_data:transaction_product_name,trans
action_data:transaction_datetime,transaction_data:transaction_amount,customer_d
ata:transaction_city_name,customer_data:transaction_country_name") TBLPROPERTIE
S ("hbase.table.name"="transaction_detail_hbase_tbl");
OK
Time taken: 0.264 seconds
hive> select * from transaction_detail_hive_tbl;
OK
Time taken: 0.603 seconds
hive> select * from transaction_detail_hive_tabl;
OK
1       MasterCard      www.ebay.com    Laptop  2019-05-14 15:24:12     50.85 M
umbai   India
2       MasterCard      www.amazon.com  Wrist Band      2019-05-14 15:24:13     2
59.12   Pune    India
3       MasterCard      www.flipkart.com        TV Stand        2019-05-14 15:2
4:14    328.16  New York City   United States
4       Visa    www.snapdeal.com        TV Stand        2019-05-14 15:24:15     3
99.06   New Delhi       India
5       Visa    www.ebay.com    External Hard Drive     2019-05-14 15:24:16     1
94.52   Rome    Italy
Time taken: 0.216 seconds, Fetched: 5 row(s)
hive> 
```

Note: all this data is stored in HBase and is directly coming from it.

Nothing is ever stored in Hive. Hive is only being used as interface to query the data for simplification purposes.

Inserting Big Data in HBase programmatically:

As seen in the above steps, a sample data was inserted using the ‘put’ command in the HBase table to demo the integration with Hive.

However, for big data, multiple put commands will make the data insertion process inefficient. A better approach would be to write a program for data insertion.

Example program code for inserting large data set in HBase ***(this code is in jRuby, but can be written in java or python as well)***

Create a JRuby file named multi.rb through which we will insert multiple rows in an HBase table:

Code:

```
import ‘org.apache.hadoop.HBase.client.HTable’
import ‘org.apache.hadoop.HBase.client.Put’

def jbytes(*args)
     args.map {|arg| arg.to_s.to_java_bytes}
end
```

Setting up packages and function to initiate HBase insertion program.

```
table = HTable.new(@HBase.configuration,
“transaction_detail_HBase_tbl”)
row = Put.new(*jbytes(“200”))
```

Creating HTable as an object called tables pointing to the transaction_detail_HBase_tbl
Creating an object called “row” with a row key specified

```
row.add
(*jbytes(“customer_data”, “card_type”, “master card”))
(*jbytes(“customer_data”, “card_type”, “visa card”))
(*jbytes(“customer_data”, “card_type”, “multi card”))
```

Using the add method to put data into the “row” object

```
table.put(row)
```

Executing from linux command prompt:

Command:

```
HBase shell multi.rb
```

Big data import in HBase - Importing data from a CSV file :

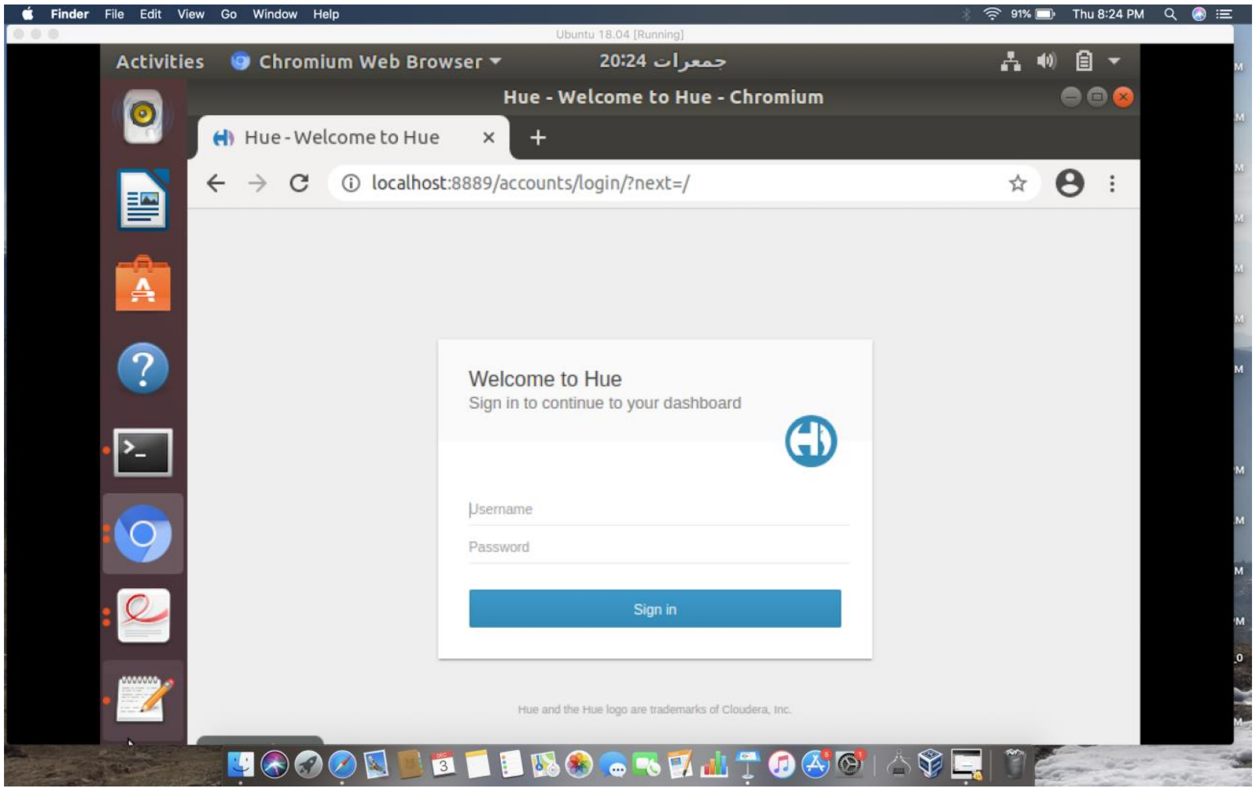
Once the table has been created in HBase, then locate the CSV file on your computer. (transaction_detail.csv)

Make sure to remove the header row as attributes are declared in HBase.

Connect to the HBase server - in this case hue since we are using docker

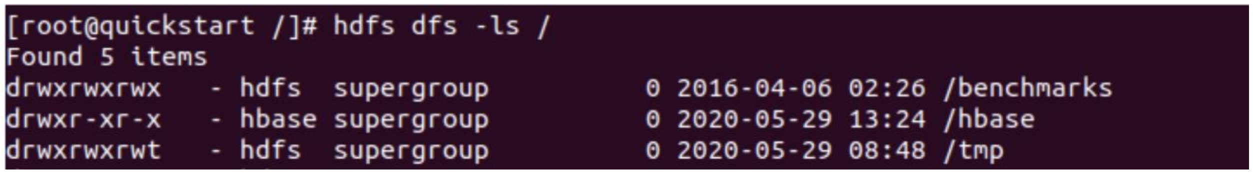
Open browser and enter:

localhost:8889/


Activities
Chromium Web Browser
Hue - Welcome to Hue - Chromium
Hue - Welcome to Hue
localhost:8889/accounts/login/?next=/
Welcome to Hue
Sign in to continue to your dashboard
Username
Password
Sign in
Hue and the Hue logo are trademarks of Cloudera, Inc.

Upload (transaction_detail.csv) to section of HBase so file can reside in the server.

Once file is in server, go back to hbase shell and find the file location



```
[root@quickstart /]# hdfs dfs -ls /
Found 5 items
drwxrwxrwx   - hdfs  supergroup          0 2016-04-06 02:26 /benchmarks
drwxr-xr-x   - hbase supergroup          0 2020-05-29 13:24 /hbase
drwxrwxrwt   - hdfs  supergroup          0 2020-05-29 08:48 /tmp
```

Use the following command to import transaction_detail.csv file in HBase table:

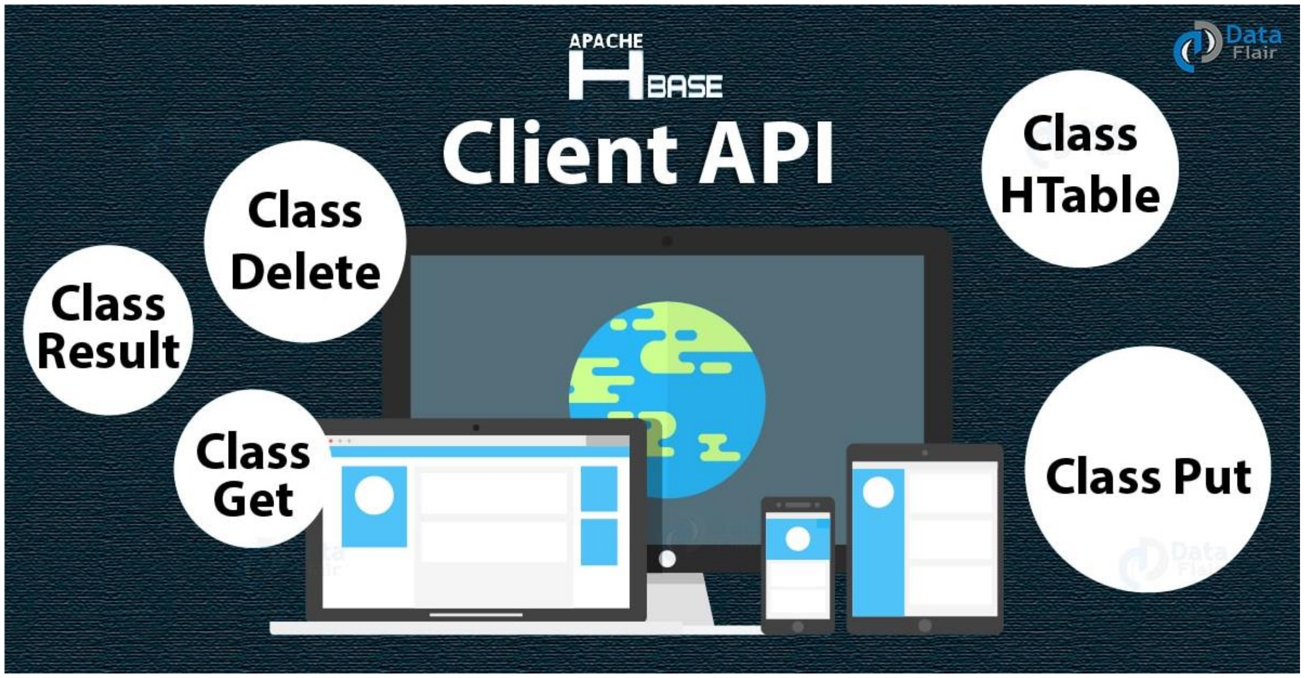
```
hbase org.apache.hadoop.hbase.mapreduce.ImportTsv -
Dimporttsv.separator=',' -
Dimporttsv.columns=HBASE_ROW_KEY,transaction_data:transaction_ca
rd_type,transaction_data:transaction_ecommerce_website_name,tran
saction_data:transaction_product_name,transaction_data:transacti
on_datetime,transaction_data:transaction_amount,customer_data:tr
ansaction_city_name,customer_data:transaction_country_name
transaction_detail_HBase_tbl /user/hdfs/transaction_detail.csv
```

All csv file data will be stored in HBase table `transaction_detail_HBase_tbl`

Using the hive and HBase integration concept explained with the demo above, the data from the csv file can be analyzed with SQL queries while residing in a noSQL form.

Client API: The Basics

In in this part of the HBase Lab we will learn the basics of client API, CRUD operations, KeyValue Class, application for inserting data into database, application for retrieving data from database, and row locks.



In order to perform CRUD operations which stands for create, read, update, and delete, on HBase table we use Java client API for HBase. Since HBase has a Java Native API and it is written in Java thus it offers programmatic access to DML (Data Manipulation Language).

- **Class HBase Configuration** : This class adds HBase configuration files to a Configuration. It belongs to the org.apache.hadoop.hbase package.
- **Method** : To create a Configuration with HBase resources, we use the following method: static org.apache.hadoop.conf.Configuration create()

Class HTable in HBase Client API:

An HBase internal class which represents an HBase table is HTable. Basically, to communicate with a single HBase table, we use this implementation of a table. It belongs to the org.apache.hadoop.hbase.client class.

- Constructors: We can create an object to access an HBase table by using the following constructors:
    1. HTable()
    2. HTable(TableName tableName, ClusterConnection connection, ExecutorService pool)

- Methods:
    1. **void close() :** To release all the resources of the HTable, we use this method.
    2. **void delete(Delete delete)** : The method "void delete(Delete delete)" helps to delete the specified cells/row.
    3. **Result get(Get get) :** This method retrieves certain cells from a given row.
    4. **org.apache.hadoop.conf.Configuration getConfiguration() :** It returns the Configuration object used by this instance.

5. **TableName getName() :** This method returns the table name instance of the table.
1. **HTableDescriptor getTableDescriptor() :** It returns the table descriptor for the table.
2. **byte[] getTableName() :** This method returns the name of the table.
3. **void put(Put put):** We can insert data into the table, by using this method.

Class Put in HBase Client API:

In order to perform put operations for a single row, we use the class that belongs to the **org.apache.hadoop.hbase.client** package.

- Constructors

1. **Put(byte[] row):** We can create a Put operation for the specified row, by using this constructor.
2. **Put(byte[] rowArray, int rowOffset, int rowLength):** To make a copy of the passed-in row key to keep local, we use it.
3. **Put(byte[] rowArray, int rowOffset, int rowLength, long ts):** We can make a copy of the passed-in row key to keep local, by using this constructor.
4. **Put(byte[] row, long ts):** To create a Put operation for the specified row, using a given timestamp, we use it.

- Methods

1. **Put add(byte[] family, byte[] qualifier, byte[] value):** The method "Put add(byte[] family, byte[] qualifier, byte[] value)" adds the specified column and value to this Put operation.
2. **Put add(byte[] family, byte[] qualifier, long ts, byte[] value):** With the specified timestamp, it adds the specified column and value, as its version to this Put operation.
3. **Put add(byte[] family, ByteBuffer qualifier, long ts, ByteBuffer value)**: This method adds the specified column and value, with the specified timestamp as its version to this Put operation.

4. **Put add(byte[] family, ByteBuffer qualifier, long ts, ByteBuffer value)**: With the specified timestamp, it adds the specified column and value, as its version to this Put operation.

- **The KeyValue class:** In our code we may have to deal with KeyValue instances directly. These instances contain the data as well as the coordinates of one specific cell. The coordinates are the row key, name of the column family, column qualifier, and timestamp. The class provides a plethora of constructors that allow you to combine all of these in many variations. The fully specified constructor looks like this: KeyValue(byte[] row, int roffset, int rlength, byte[] family, int foffset, int flength, byte[] qualifier, int qoffset, int qlength, long timestamp, Type type, byte[] value, int voffset, int vlength)

Class Get in HBase Client API:

To perform Get operations on a single row, we use the class that belongs to the org.apache.hadoop.hbase.client package.

- Constructor:

1. **Get(byte[] row):** It is possible to create a Get operation for the specified row, by using this constructor.

- Methods:

1. **Get addColumn(byte[] family, byte[] qualifier):** To retrieve the column from the specific family with the specified qualifier, this method helps.
2. **Get addFamily(byte[] family):** This one helps to retrieve all columns from the specified family.

Class Delete in HBase Client API

- Constructors

1. **Delete(byte[] row)**: To create a delete operation for the specified row, we use it.

2. **Delete(byte[] rowArray, int rowOffset, int rowLength):** This constructor creates a Delete operation for the specified row and timestamp.
3. **Delete(byte[] rowArray, int rowOffset, int rowLength, long ts):** This constructor performs Delete operation.
4. **Delete(byte[] row, long timestamp):** Again this constructor performs Delete operation.

- Methods

1. **Delete addColumn(byte[] family, byte[] qualifier):** This method helps to delete the latest version of the specified column.
2. **Delete addColumns(byte[] family, byte[] qualifier, long timestamp):** To delete all versions of the specified column we use this method, especially, with a timestamp less than or equal to the specified timestamp.
3. **Delete addFamily(byte[] family):** This method deletes all versions of all columns of the specified family.
4. **Delete addFamily(byte[] family, long timestamp):** Again, with a timestamp less than or equal to the specified timestamp, this method also deletes all columns of the specified family.

Class Result in HBase Client API

- Constructor
  1. **Result():** With no KeyValue payload, it is possible to create an empty Result; returns null if you call raw Cells(), by using this constructor.
- Methods
  1. **byte[] getValue(byte[] family, byte[] qualifier)**: In order to get the latest version of the specified column, we use this method.
     2. byte[] getRow(): Moreover, to retrieve the row key which corresponds to the row from which this Result was created, we use this method.

Row Locks

The region servers provide a row lock feature ensuring that only a client holding the matching lock can modify a row. In practice, though, most client applications do not provide an explicit lock, but rather rely on the mechanism in place that guards each operation separately. When you send, for example, a put() call to the server with an instance of Put, created with the following constructor: Put(byte[] row) , Which is not providing a RowLock instance parameter, the servers will create a lock on our behalf, just for the duration of the call.

Example Application:
1. Inserting data into HBase Table Using Java API

```
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.hbase.HBaseConfiguration;
import org.apache.hadoop.hbase.client.HTable;
import org.apache.hadoop.hbase.client.Put;
import org.apache.hadoop.hbase.util.Bytes;
import java.io.IOException;
public class PutExample {
public static void main(String[] args) throws IOException {
```

1. Create the required configuration.
   ```
   Configuration conf = HBaseConfiguration.create();
   ```
2. Instantiate a new client.
   ```
   HTable table = new HTable(conf, "testtable");
   ```
3. Create Put with specific row.
   ```
   Put put = new Put(Bytes.toBytes("row1"));
   ```
4. Add a column, whose name is “colfam1:qual1”, to the Put.
   ```
   put.add(Bytes.toBytes("colfam1"), Bytes.toBytes("qual1"),
   Bytes.toBytes("val1"));
   ```
5. Add another column, whose name is “colfam1:qual2”, to the Put.
   ```
   put.add(Bytes.toBytes("colfam1"), Bytes.toBytes("qual2"),
   Bytes.toBytes("val2"));
   ```
6. Store the row with the column into the HBase table.
   ```
   table.put(put);
   }
   }
   }
   ```

2. Retrieving data from HBase Table Using Java API

1. Create the configuration:

   ```
   Configuration conf = HBaseConfiguration.create();
   ```

2. Instantiate a new table reference:

   ```
   HTable table = new HTable(conf, "testtable");
   ```

3. Create a Get with a specific row:

   ```
   Get get = new Get(Bytes.toBytes("row1"));
   ```

4. Add a column to the Get:

   ```
   get.addColumn(Bytes.toBytes("colfam1"), Bytes.toBytes("qual1"));
   ```

5. Retrieve a row with selected columns from HBase:

   ```
   Result result = table.get(get);
   ```

6. Get a specific value for the given column:

   ```
   byte[] val = result.getValue(Bytes.toBytes("colfam1"),
   Bytes.toBytes("qual1"));
   ```

7. Print out the value while converting it back:

   ```
   System.out.println("Value: " + Bytes.toString(val));
   ```

3. Deleting data from HBase Table Using Java API

1. Create a Delete with a specific row: `Delete delete = new Delete(Bytes.toBytes("row1"));`
2. Set a timestamp for row deletes:

   `delete.setTimestamp(1);`

3. Delete a specific version in one column:

   `delete.deleteColumn(Bytes.toBytes("colfam1"), Bytes.toBytes("qual1"), 1);`

4. Delete all versions in one column:

   `delete.deleteColumns(Bytes.toBytes("colfam2"), Bytes.toBytes("qual1"));`

5. Delete the given and all older versions in one column:

   `delete.deleteColumns(Bytes.toBytes("colfam2"), Bytes.toBytes("qual3"), 15);`

6. Delete the entire family, all columns and versions:

   `delete.deleteFamily(Bytes.toBytes("colfam3"));`

7. Delete the given and all older versions in the entire column family, that is, from all columns therein:

   `delete.deleteFamily(Bytes.toBytes("colfam3"), 3);`

8. Delete the data from the HBase table:

   `table.delete(delete);`

```
table.close();
```

4. Scan The HBase Table Using Java API

- The getScanner() method of the HTable class can be used to scan the entire data of Hbase table. The getScanner() method expects an instance of the Scan class. To summarize, steps to scan the entire data of HBase table are:
    1. Instantiate the Configuration Class:
       ```
       Configuration conf = HBaseConfiguration.create();
       ```
    2. Instantiate the HTable Class:
       ```
       HTable table = new HTable(conf, "testtable");
       ```
    3. Instantiate the Scan Class:
       ```
       Scan scan = new Scan();
       // scan the columns
       scan.addColumn(Bytes.toBytes("colfam1"),
       Bytes.toBytes("qual1"));
       scan.addColumn(Bytes.toBytes("colfam1"),
       Bytes.toBytes("val1"));
       ```
    4. Get the ResultScanner instances by calling HTable's getScanner() method:
       ```
       ResultScanner scanner = table.getScanner(scan);
       for (Result result = scanner.next(); result != null;
       result=scanner.next())
               System.out.println("Found row : " + result);
       ```
    5. Read values from ResultScanner and close it:
       ```
         scanner.close();
       ```

Conclusion:

Hence, in this part of the lab, we have seen the whole concept of HBase Client API. Moreover, we saw Class Htable, Class put, Class Get, Class delete, and Class result in HBase client. Also, we looked at methods, constructors & a working example of HBase Client API.

Reference Links:

HBase Shell & Commands - Usage & Starting HBase Shell - DataFlair (data-flair.training)

HBase Shell Commands with Examples (guru99.com)

HBase Shell Commands Cheat Sheet — SparkByExamples

https://www.kaggle.com/regivm/retailtransactiondata?select=Retail_Data_Response.csv

https://medium.com/@ramprakash.ukd3/hbase-commands-aa8fe5dd8e41

https://docs.cloudera.com/runtime/7.2.2/managing-hbase/topics/hbase-perform-scans-using-hbase-shell.html

https://www.tutorialspoint.com/hbase/hbase_scan.htm

https://www.guru99.com/hbase-tutorials.html

https://www.youtube.com/watch?v=3ZJ-2H6L9Hg

https://www.syncfusion.com/ebooks/hive-succinctly/external-tables-over-hbase

https://www.corejavaguru.com/bigdata/hbase-tutorial/hbase-java-client-api-examples

https://www.tutorialspoint.com/hbase/hbase_client_api.htm