Aims

This exercise aims to get you to:

- Install and configure HBase
- Manage data using HBase Shell
- Install and configure Hive
- Manage data using Hive

HBase Installation and Configuration

1. Download HBase 2.0.0

```
$ wget http://mirror.ventraip.net.au/apache/hbase/2.0.0/hbase-2.0.0-bin.tar.gz
```

Then unpack the package:

```
$ tar xvf hbase-2.0.0-bin.tar.gz
```

2. Define environment variables for HBase

We need to configure the working directory of HBase, i.e., `HBASE_HOME`.

Open the file `~/.bashrc` and add the following lines at the **end** of this file:

```
export HBASE_HOME = ~/hbase-2.0.0
export PATH = $HBASE_HOME/bin:$PATH
```

Save the file, and then run the following command to take these configurations into effect:

```
$ source ~/.bashrc
```

Open the HBase environment file, hbase-env.sh, using:

```
$ gedit $HBASE_HOME/conf/hbase-env.sh
```

Add the following lines at the **end** of this file (check your java version!):

```
export JAVA_HOME = /usr/lib/jvm/java-1.7.0-openjdk-amd64
export HBASE_MANAGES_ZK = true
```

3. Configure HBase as Pseudo-Distributed Mode

Open the HBase configuration file, hbase-site.xml, using:

```
$ gedit $HBASE_HOME/conf/hbase-site.xml
```

Add the following lines in between <configuration> and </configuration>:

```
<property>
  <name>hbase.rootdir</name>
  <value>hdfs://localhost:9000/hbase</value>
</property>
<property>
  <name>hbase.cluster.distributed</name>
  <value>true</value>
</property>
```

Now you have already done the basic configuration of HBase, and it is ready to use. Start HBase by the following command (start HDFS first!):

```
$ start-hbase.sh
```

You will see:

```
comp9313@comp9313-VirtualBox:~$ start-hbase.sh
localhost: starting zookeeper, logging to /home/comp9313/hbase/bin/../logs/hbase
-comp9313-zookeeper-comp9313-VirtualBox.out
starting master, logging to /home/comp9313/hbase/logs/hbase-comp9313-master-comp
9313-VirtualBox.out
starting regionserver, logging to /home/comp9313/hbase/logs/hbase-comp9313-1-reg
ionserver-comp9313-VirtualBox.out
```

Type “jps” in the terminal, you can see that more daemons are started.

```
comp9313@comp9313-VirtualBox:~$ jps
4129 HRegionServer
2854 DataNode
4212 Jps
3371 NodeManager
3067 SecondaryNameNode
3221 ResourceManager
2680 NameNode
4008 HMaster
3942 HQuorumPeer
```

Practice HBase Shell Commands

In this part, you will practice on how to manage data using HBase shell commands. As such, after completing this lab, you’ll know how to

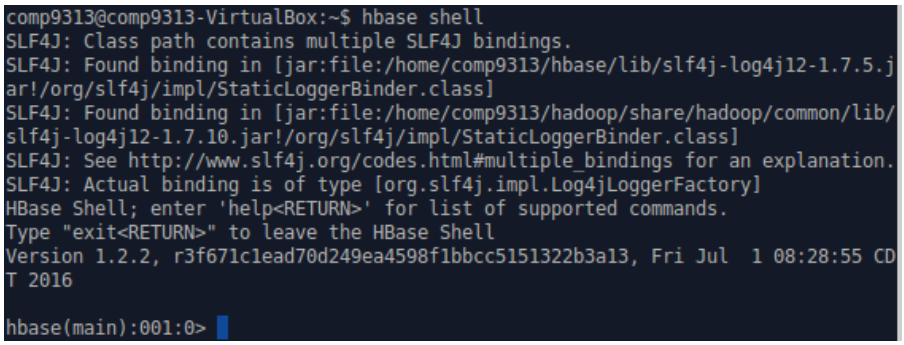
- Launch the HBase shell
- Create an HBase table
- Inspect the characteristics of a table
- Alter properties associated with a table
- Populate a table with data
- Retrieve data from a table

- Use HBase Web interfaces to explore information about your environment

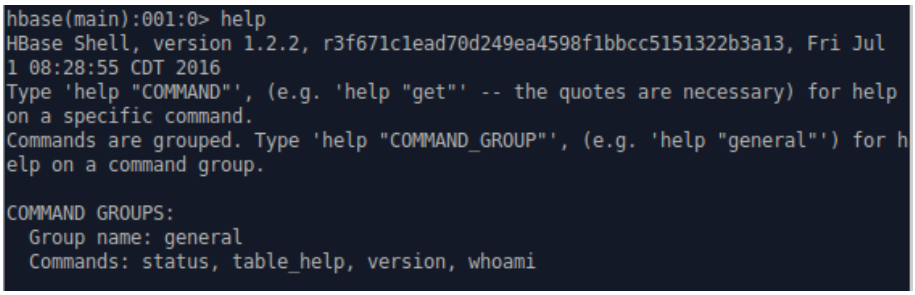
Launch the HBase shell

1. After HBase is started, use the following command to launch the shell:

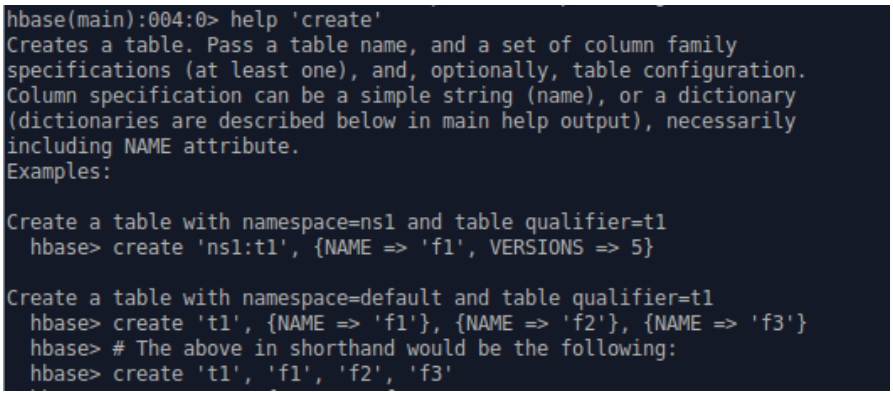
```
$ hbase shell
```



2. Once started, you can type in help, and then press Return, to get the help text (shown abbreviated):



You can request help for a specific command by adding the command when invoking help, or print out the help of all commands for a specific group when using the group name with the help command. The optional command or group name has to be enclosed in quotes. For example, type "help 'create'" in the shell, and you will see the usage of this command:



Creating and altering a table

1. Create an HBase table named reviews with 3 column families: summary, reviewer, and details.

```
$ create 'reviews', 'summary', 'reviewer', 'details'
```

2. Inspect the default properties associated with your new table:

```
$ describe 'reviews'
```

```
hbase(main):012:0> describe 'reviews'
Table reviews is ENABLED
reviews
COLUMN FAMILIES DESCRIPTION
{NAME => 'details', DATA_BLOCK_ENCODING => 'NONE', BLOOMFILTER => 'ROW', REPLICA
TION_SCOPE => '0', VERSIONS => '1', COMPRESSION => 'NONE', MIN_VERSIONS => '0',
TTL => 'FOREVER', KEEP_DELETED_CELLS => 'FALSE', BLOCKSIZE => '65536', IN_MEMORY
 => 'false', BLOCKCACHE => 'true'}
{NAME => 'reviewer', DATA_BLOCK_ENCODING => 'NONE', BLOOMFILTER => 'ROW', REPLIC
ATION_SCOPE => '0', VERSIONS => '1', COMPRESSION => 'NONE', MIN_VERSIONS => '0',
 TTL => 'FOREVER', KEEP_DELETED_CELLS => 'FALSE', BLOCKSIZE => '65536', IN_MEMOR
Y => 'false', BLOCKCACHE => 'true'}
{NAME => 'summary', DATA_BLOCK_ENCODING => 'NONE', BLOOMFILTER => 'ROW', REPLICA
TION_SCOPE => '0', VERSIONS => '1', COMPRESSION => 'NONE', MIN_VERSIONS => '0',
TTL => 'FOREVER', KEEP_DELETED_CELLS => 'FALSE', BLOCKSIZE => '65536', IN_MEMORY
 => 'false', BLOCKCACHE => 'true'}
3 row(s) in 0.0210 seconds
```

3. To alter (or drop) a table, you must first disable it:

```
$ disable 'reviews'
```

4. Alter the table to set the IN_MEMORY property of the summary column family to true.

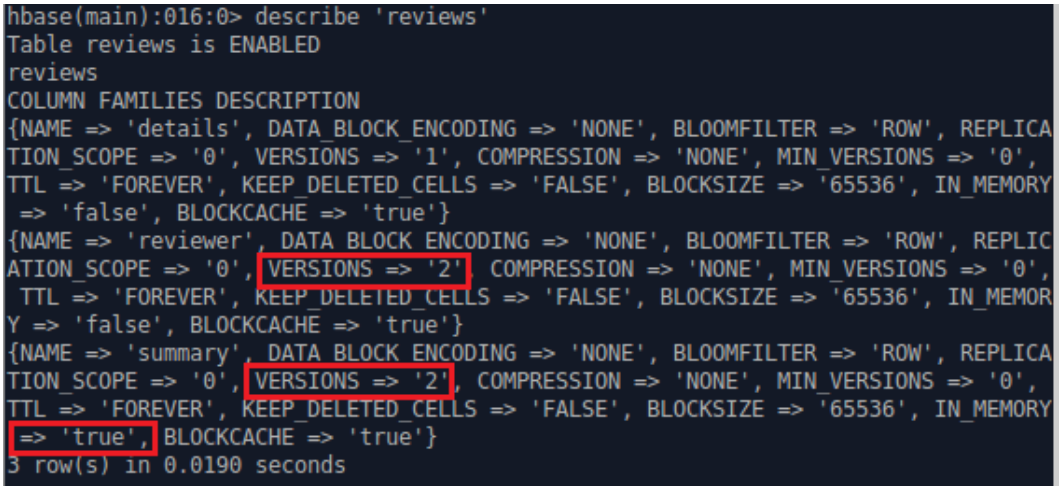
```
$ alter 'reviews', {NAME => 'summary', IN_MEMORY => 'true'}
```

5. Set the number of versions for the summary and reviewer column families to 2. HBase can store multiple versions of data for each column family. By default it is set to 1.

```
$ alter 'reviews', {NAME => 'summary', VERSIONS => 2}, {NAME =>
'reviewer', VERSIONS => 2}
```

Verify that your property changes were captured correctly:

```
$ describe 'reviews'
```



6. Enable (or activate) the table so that it's ready for use

```
$ enable 'reviews'
```

Now you can populate your table with data and query it.

Inserting and retrieving data

1. Insert some data into your HBase table. The PUT command enables you to write data into a single cell of an HBase table. This cell may reside in an existing row or may belong to a new row.

```
$ put 'reviews', '101', 'summary:product', 'hat'
```

What happened after executing this command

Executing this command caused HBase to add a row with a row key of 101 to the reviews table and to write the value of hat into the product column of the summary column family. Note that this command dynamically created the summary:product column and that no data type was specified for this column.

What if you have more data for this row? You need to issue additional PUT commands – one for each cell (i.e., each column family:column) in the target row. You'll do that shortly. But before you do, consider what HBase just did behind the scenes . . . .

HBase wrote your data to a Write-Ahead Log (WAL) in your distributed file system to allow for recovery from a server failure. In addition, it cached your data (in a MemStore) of a specific region managed by a specific Region Server. At some point, when the MemStore becomes full, your data will be flushed to disk and stored in files (HFiles) in your distributed file system. Each HFile contains data related to a specific column family.

2. Retrieve the row. To do so, provide the table name and row key value to the GET command:

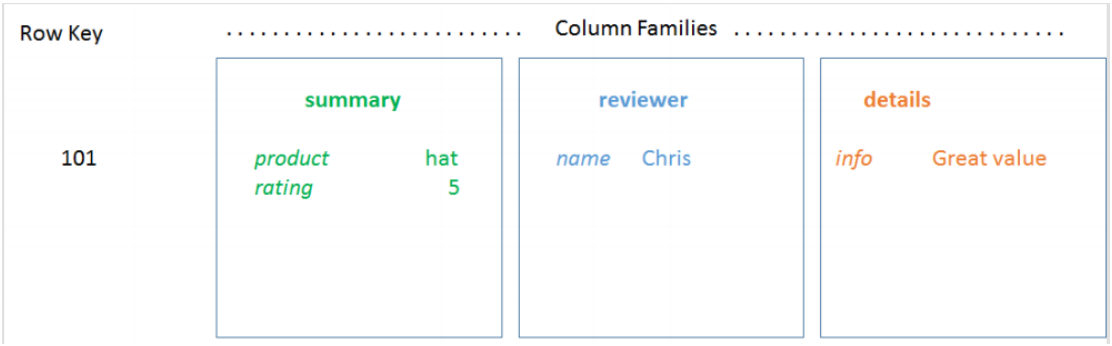
```
$ get 'reviews', '101'
```

```
hbase(main):019:0* get 'reviews', '101'
COLUMN                CELL
 summary:product      timestamp=1472405345388, value=hat
1 row(s) in 0.0350 seconds
```

3. Add more cells (columns and data values) to this row:

```
$ put 'reviews', '101', 'summary:rating', '5'
$ put 'reviews', '101', 'reviewer:name', 'Chris'
$ put 'reviews', '101', 'details:comment', 'Great value'
```

Conceptually, your table looks something like this:



Retrieve the row again:

```
hbase(main):023:0> get 'reviews', '101'
COLUMN                CELL
 details:comment      timestamp=1472405491873, value=Great value
 reviewer:name        timestamp=1472405487241, value=Chris
 summary:product      timestamp=1472405345388, value=hat
 summary:rating       timestamp=1472405481127, value=5
4 row(s) in 0.0230 seconds
```

This output can be a little confusing at first, because it's showing that 4 rows are returned. This row count refers to the number of lines (rows) displayed on the screen. Since information about each cell is displayed on a separate line and there are 4 cells in row 101, the GET command reports 4 rows.

4. Count the number of rows in the entire table and verify that there is only 1 row:

```
$ count 'reviews'
```

5. Add 2 more rows to your table using these commands:

```
$ put 'reviews', '112', 'summary:product', 'vest'
$ put 'reviews', '112', 'summary:rating', '5'
$ put 'reviews', '112', 'reviewer:name', 'Tina'
$ put 'reviews', '133', 'summary:product', 'vest'
$ put 'reviews', '133', 'summary:rating', '4'
$ put 'reviews', '133', 'reviewer:name', 'Helen'
$ put 'reviews', '133', 'reviewer:location', 'USA'
$ put 'reviews', '133', 'details:tip', 'Sizes run small. Order 1 size
up.'
```

Note that review 112 lacks any detailed information (e.g., a comment), while review 133 contains a tip in its details. Note also that review 133 includes the reviewer's location, which is not present in the other rows.

6. Retrieve the entire contents of the table using this SCAN command:

```
$ scan 'reviews'
```

```
hbase(main):034:0> scan 'reviews'
ROW                     COLUMN+CELL
 101                    column=details:comment, timestamp=1472405491873, value=Great value
 101                    column=reviewer:name, timestamp=1472405487241, value=Chris
 101                    column=summary:product, timestamp=1472405345388, value=hat
 101                    column=summary:rating, timestamp=1472405481127, value=5
 112                    column=reviewer:name, timestamp=1472405576782, value=Tina
 112                    column=summary:product, timestamp=1472405565373, value=vest
 112                    column=summary:rating, timestamp=1472405570311, value=5
 133                    column=details:tip, timestamp=1472405611320, value=Sizes run small. Ord
                        er 1 size up.
 133                    column=reviewer:location, timestamp=1472405607022, value=USA
 133                    column=reviewer:name, timestamp=1472405601078, value=Helen
 133                    column=summary:product, timestamp=1472405591649, value=vest
 133                    column=summary:rating, timestamp=1472405596526, value=4
3 row(s) in 0.0180 seconds
```

Note that SCAN correctly reports that the table contains 3 rows. The display contains more than 3 lines, because each line includes information for a single cell in a row. Note also that each row in your table has a different schema and that missing information is simply omitted.

Furthermore, each displayed line includes not only the value of a particular cell in the table but also its associated row key (e.g., 101), column family name (e.g., details), column name (e.g., comment), and timestamp. As you learned earlier, HBase is a key-value store. Together, these four attributes (row key, column family name, column qualifier, and timestamp) form the key.

Consider the implications of storing this key information with each cell value. Having a large number of columns with values for all rows (in other words, dense data) means that a lot of key information is repeated. Also, large row key values and long column family / column names increase the table's storage requirements.

7. Finally, restrict the scan results to retrieve only the contents of the summary column family and the reviewer:name column for row keys starting at '120' and ending at '150'.

```
$ scan 'reviews', {COLUMNS => ['summary', 'reviewer:name'], STARTROW
=> '120', STOPROW => '150'}
```

Given your sample data, only row '133' qualifies. Note that the reviewer's location (reviewer:location) and all the review details (details:tip) were omitted from the results due to the scan parameters you specified.

Updating data

1. Update Tina's review (row key 112) to change the rating to '4':

```
$ put 'reviews', '112', 'summary:rating', '4'
```

2. Scan the table to inspect the change.

```
hbase(main):037:0> scan 'reviews'
ROW                     COLUMN+CELL
 101                    column=details:comment, timestamp=1472405491873, value=Great value
 101                    column=reviewer:name, timestamp=1472405487241, value=Chris
 101                    column=summary:product, timestamp=1472405345388, value=hat
 101                    column=summary:rating, timestamp=1472405481127, value=5
 112                    column=reviewer:name, timestamp=1472405576782, value=Tina
 112                    column=summary:product, timestamp=1472405565373, value=vest
 112                    column=summary:rating, timestamp=1472405950015, value=4
 133                    column=details:tip, timestamp=1472405611320, value=Sizes run small. Ord
                        er 1 size up.
 133                    column=reviewer:location, timestamp=1472405607022, value=USA
 133                    column=reviewer:name, timestamp=1472405601078, value=Helen
 133                    column=summary:product, timestamp=1472405591649, value=vest
 133                    column=summary:rating, timestamp=1472405596526, value=4
3 row(s) in 0.0470 seconds
```

By default, HBase returns the most recent version of data for each cell. Value 5 is not shown in the results.

3. To see multiple versions of your data, issue this command:

```
$ scan 'reviews', {VERSIONS => 2}
```

4. You can also GET the original rating value from row 112 by explicitly specifying the timestamp value. This value will differ on your system, so you will need to substitute the value appropriate for your environment for the timestamp shown below. Consult the output from the previous step to obtain this value.

```
$ get 'reviews', '112', {COLUMN => 'summary:rating', TIMESTAMP =>
1421878110712}
```

Deleting data

1. Delete Tina's name from her review (row 112)

```
$ delete 'reviews', '112', 'reviewer:name'
```

Scan the table to inspect the change.

2. Delete all cells associated with Tina's review (i.e., all data for row 112) and scan the table to inspect the change.

```
$ deleteall 'reviews', '112'
```

Scan the table again to see the results.

About DELETE

DELETE doesn't remove data from the table immediately. Instead, it marks the data for deletion, which prevents the data from being included in any subsequent data retrieval operations. Because the underlying files that form an HBase table (HFiles) are immutable, storage for deleted data will not be recovered until an administrator initiates a major compaction operation. This operation consolidates data and reconciles deletions by removing both the deleted data and the delete indicator.

Browse the Web UI of HBase

You can explore some of the meta data available to you about your table as well as your overall HBase environment using the HBase Web UI. The HBase Master Service Web interface port is 16010. Open the URL http://localhost:16010 in a browser. The port information can be configured in the hbase-site.xml file within the installation directory of HBase, by setting the hbase.master.info.port property.

Dropping a table

Disable the table first, and then drop the table.

```
$ disable 'reviews'
$ drop 'reviews'
```

Try more commands.

You can find more commands at https://hbase.apache.org/book.html#shell. Try them using the 'reviews' table.

Hive Installation and Configuration

1. Download Hive 2.3.3

```
$ wget http://apache.mirror.digitalpacific.com.au/hive/hive-
2.3.3/apache-hive-2.3.3-bin.tar.gz
```

Then unpack the package:

```
$ tar xvf apache-hive-2.3.3-bin.tar.gz
```

2. Define environment variables for Hive

We need to configure the working directory of Hive, i.e., `HIVE_HOME`.

Open the file `~/.bashrc` and add the following lines at the **end** of this file:

```
export HIVE_HOME = ~/apache-hive-2.3.3-bin
export PATH = $HIVE_HOME/bin:$PATH
```

Save the file, and then run the following command to take these configurations into effect:

```
$ source ~/.bashrc
```

3. Create /tmp and /user/hive/warehouse and set them chmod g+w for more than one user usage

```
$ hdfs dfs -mkdir /tmp
$ hdfs dfs -mkdir -p /user/hive/warehouse
$ hdfs dfs -chmod g+w /tmp
$ hdfs dfs -chmod g+w /user/hive/warehouse
```

4. Run the schematool command to initialize Hive

```
$ schematool -dbType derby -initSchema
```

Now you have already done the basic configuration of Hive, and it is ready to use. Start Hive shell by the following command (start HDFS first!):

```
$ hive
```

```
comp9313@comp9313-VirtualBox:~$ hive
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/home/comp9313/hive/lib/log4j-slf4j-impl-2.4.1
.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/home/comp9313/hadoop/share/hadoop/common/lib/
slf4j-log4j12-1.7.10.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.apache.logging.slf4j.Log4jLoggerFactory]

Logging initialized using configuration in jar:file:/home/comp9313/hive/lib/hive
-common-2.1.0.jar!/hive-log4j2.properties Async: true
Hive-on-MR is deprecated in Hive 2 and may not be available in the future versio
ns. Consider using a different execution engine (i.e. tez, spark) or using Hive
1.X releases.
hive>
```

Manage Data Using Hive

1. Download the test file “employees.txt” from the course webpage. The file contains only 7 records. Put the file at the home folder.

2. Create a database

```
$ hive> create database employee_data;
$ hive> use employee_data;
```

3. All databases are created under /user/hive/warehouse directory.

```
$ hdfs dfs –ls /user/hive/warehouse
```

```
comp9313@comp9313-VirtualBox:~$ hdfs dfs -ls /user/hive/warehouse
Found 2 items
drwxr-xr-x   - comp9313 supergroup          0 2016-09-05 08:08 /user/hive/wareho
use/employee_data.db
```

4. Create the employee table

```
$ hive> CREATE TABLE employees (
  name         STRING,
  salary       FLOAT,
  subordinates ARRAY<STRING>,
  deductions   MAP<STRING, FLOAT>,
  address      STRUCT<street:STRING, city:STRING, state:STRING,
zip:INT>
)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY '\001'
COLLECTION ITEMS TERMINATED BY '\002'
MAP KEYS TERMINATED BY '\003'
LINES TERMINATED BY '\n'
STORED AS TEXTFILE;
```

Because '\001', '\002', '\003', and '\n' are by default, and thus you can ignore “ROW FORMAT DELIMITED”. “STORED AS TEXTFILE” is also by default, and can be ignored as well.

5. Show all tables in the current database

```
$ hive> show tables;
```

```
hive> show tables;
OK
employees
Time taken: 0.01 seconds, Fetched: 1 row(s)
```

6. Load data from local file system into table

```
$ hive> LOAD DATA LOCAL INPATH '/home/comp9313/employees.txt'
OVERWRITE INTO TABLE employees;
```

```
hive>  LOAD DATA LOCAL INPATH '/home/comp9313/employees.txt' OVERWRITE INTO TABL
E employees;
Loading data to table employee_data.employees
OK
Time taken: 0.564 seconds
```

After loading the data into the table, you can check in HDFS what happened:

```
$ hdfs dfs -ls /user/hive/warehouse/employee_data.db/employees
```

The file employees.txt is copied into this folder corresponding to the table.

7. Check the data in the table

```
$ select * from employees;
```

8. You can do various queries based on the employees table, just as in an RDBMS. For example:

Question 1: show the number of employees and their average salary

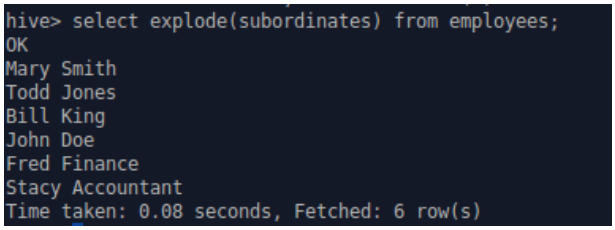
Hint: use count() and avg()

Question 2: find the employee who has the highest salary

Hint: use max(), IN clause, and subquery in where clause

9. Usage of explode(). Find all employees who are the subordinate of another person. explode() takes in an array (or a map) as an input and outputs the elements of the array (map) as separate rows.

```
$ hive> SELECT explode(subordinates) FROM employees;
```



10. Hive partitions. When defining employees, it is not partitioned, and thus you cannot add a partition to it. You can only add a new partition to a table has already been partitioned!

Create a table employees2, and load the same file into it.

```
$ hive> CREATE TABLE employees2 (
  name         STRING,
  salary       FLOAT,
  subordinates ARRAY<STRING>,
  deductions   MAP<STRING, FLOAT>,
  address      STRUCT<street:STRING, city:STRING, state:STRING,
zip:INT>
)PARTITIONED BY (join_year STRING);
$ hive> LOAD DATA LOCAL INPATH '/home/comp9313/employees.txt'
OVERWRITE INTO TABLE employees2 PARTITION (join_year="2015");
```

Now check HDFS again to see what happened:

```
$ hdfs dfs -ls /user/hive/warehouse/employ_data.db/employees2
```

You will see a folder "join_year=2015" created in this folder, corresponding to the partition join_year= "2015".

Add a new partition join_year="2016" to the table.

```
$ hive> ALTER TABLE employees2 ADD PARTITION (join_year='2016')
LOCATION
'/user/hive/warehouse/employee_data.db/employees2/join_year=2016';
```

Check in HDFS, and you will see a new folder created for this partition.

11. Insert a record to partition join_year="2016".

Because Hive does not support literals for complex types (array, map, struct, union), so it is not possible to use them in INSERT INTO...VALUES clauses. You need to create a file to store the new record, and then load it into the partition.

```
$ cp employees.txt employees2016.txt
```

Then use vim or gedit to edit employees2016.txt to add some records, and then load the file into the partition.

12. Query on a partition. Question: find all employees joined in the year 2016 whose salary is more than 60000.

13. (optional) Do word count in Hive, using the file employees.txt.