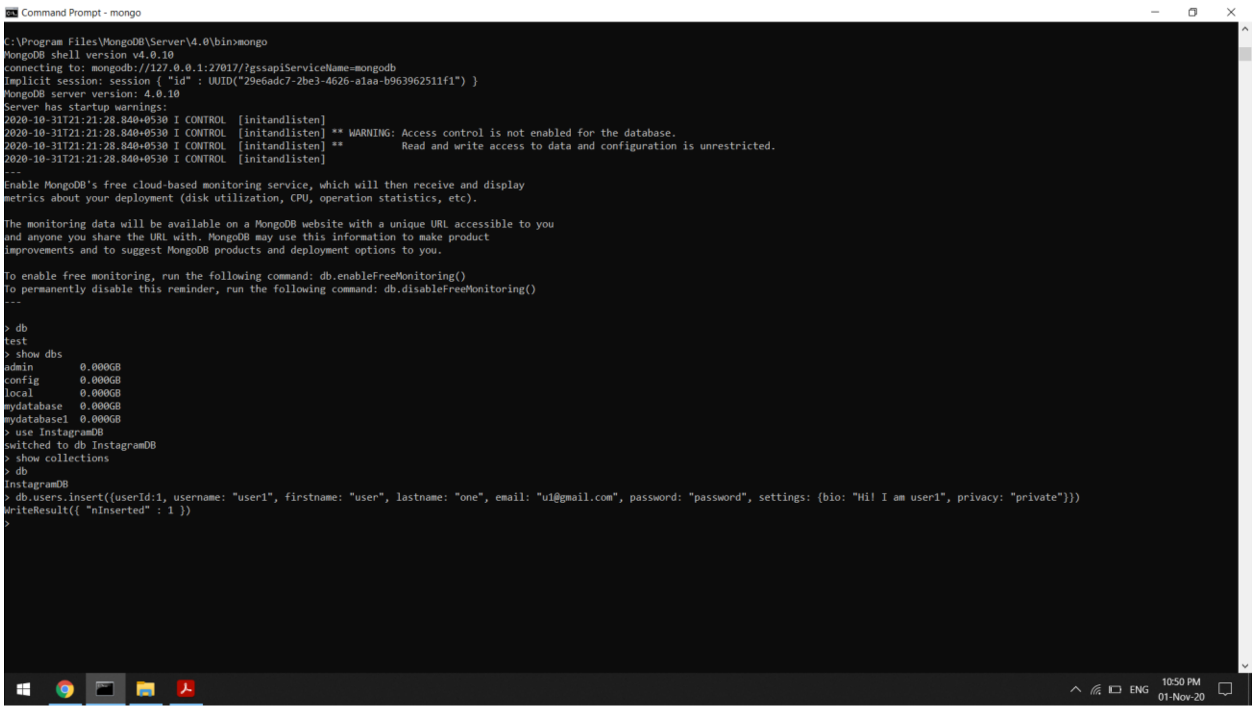
MongoDB:

Create a database and a collection-

We create the database with the *use* command. The database has no collections initially, hence, the *show collections* command does not print anything. Then we insert into users which causes a collection called users, to be created.

db.users.insert({userId:1, username: "user1", firstname: "user", lastname: "one", email: "u1@gmail.com", password: "password", settings: {bio: "Hi! I am user1", privacy: "private"}})

So for this user, we enter the fields as userId, username, firstname, lastname, email, password and settings.



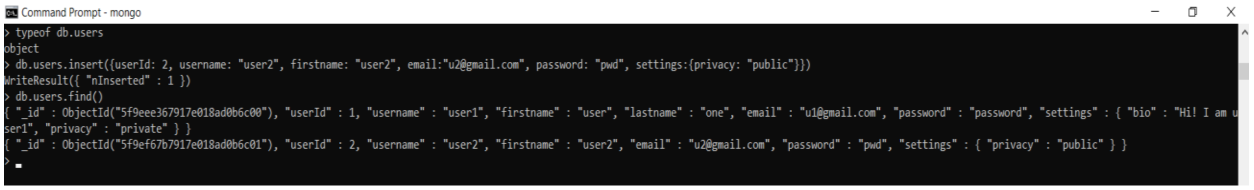
```
C:\Program Files\MongoDB\Server\4.0\bin>mongo
MongoDB shell version v4.0.10
connecting to: mongodb://127.0.0.1:27017/?gssapiServiceName=mongodb
Implicit session: session { "id" : UUID("29e6adc7-2be3-4626-a1aa-b963962511f1") }
MongoDB server version: 4.0.10
Server has startup warnings:
2020-10-31T21:21:28.840+0530 I CONTROL  [initandlisten]
2020-10-31T21:21:28.840+0530 I CONTROL  [initandlisten] ** WARNING: Access control is not enabled for the database.
2020-10-31T21:21:28.840+0530 I CONTROL  [initandlisten] **          Read and write access to data and configuration is unrestricted.
2020-10-31T21:21:28.840+0530 I CONTROL  [initandlisten]
---
Enable MongoDB's free cloud-based monitoring service, which will then receive and display
metrics about your deployment (disk utilization, CPU, operation statistics, etc).

The monitoring data will be available on a MongoDB website with a unique URL accessible to you
and anyone you share the URL with. MongoDB may use this information to make product
improvements and to suggest MongoDB products and deployment options to you.

To enable free monitoring, run the following command: db.enableFreeMonitoring()
To permanently disable this reminder, run the following command: db.disableFreeMonitoring()
---

> db
test
> show dbs
admin        0.000GB
config       0.000GB
local        0.000GB
mydatabase   0.000GB
mydatabase1  0.000GB
> use InstagramDB
switched to db InstagramDB
> show collections
> db
InstagramDB
> db.users.insert({userId:1, username: "user1", firstname: "user", lastname: "one", email: "u1@gmail.com", password: "password", settings: {bio: "Hi! I am user1", privacy: "private"}})
WriteResult({ "nInserted" : 1 })
>
```

Insertion-

- We insert a new user to the users collection by using the insert function. We can see the contents of the collection using the find function.

  db.users.insert({userId: 2, username: "user2", firstname: "user2", email:"u2@gmail.com", password: "pwd", settings:{privacy: "public"}})

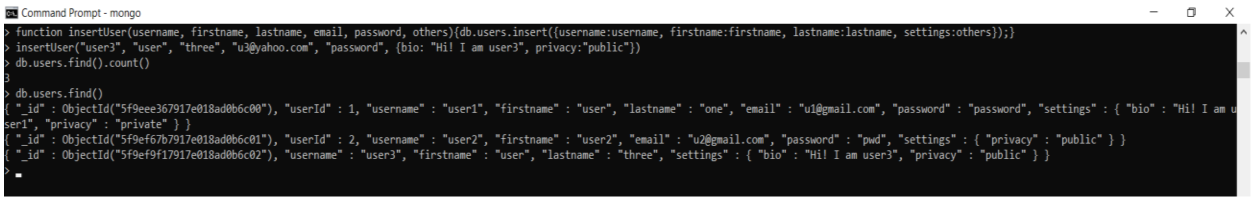
  Here, the fields maintained for the second user are slightly different from that of the first user, because there is no field for lastname, and no field for bio within settings.



```
Command Prompt - mongo
> typeof db.users
object
> db.users.insert({userId: 2, username: "user2", firstname: "user2", email:"u2@gmail.com", password: "pwd", settings:{privacy: "public"}})
WriteResult({ "nInserted" : 1 })
> db.users.find()
{ "_id" : ObjectId("5f9eee367917e018ad0b6c00"), "userId" : 1, "username" : "user1", "firstname" : "user", "lastname" : "one", "email" : "u1@gmail.com", "password" : "password", "settings" : { "bio" : "Hi! I am u
ser1", "privacy" : "private" } }
{ "_id" : ObjectId("5f9ef67b7917e018ad0b6c01"), "userId" : 2, "username" : "user2", "firstname" : "user2", "email" : "u2@gmail.com", "password" : "pwd", "settings" : { "privacy" : "public" } }
>
```

- We can also perform insertion by defining a function for the same, and calling the function. Count function along with find() helps to determine the number of entries in the collection.

  ```
  function insertUser(username, firstname, lastname, email, password, others){
  db.users.insert({username:username, firstname:firstname, lastname:lastname,
  settings:others});}

  insertUser("user3", "user", "three", "u3@yahoo.com", "password", {bio: "Hi! I am user3",
  privacy:"public"})
  ```

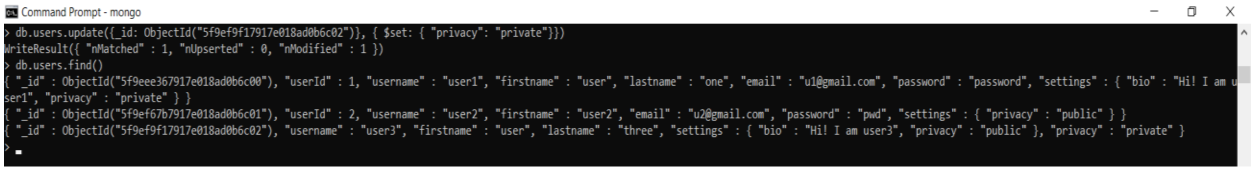


```
Command Prompt - mongo
> function insertUser(username, firstname, lastname, email, password, others){db.users.insert({username:username, firstname:firstname, lastname:lastname, settings:others});}
> insertUser("user3", "user", "three", "u3@yahoo.com", "password", {bio: "Hi! I am user3", privacy:"public"})
> db.users.find().count()
3
> db.users.find()
{ "_id" : ObjectId("5f9eee367917e018ad0b6c00"), "userId" : 1, "username" : "user1", "firstname" : "user", "lastname" : "one", "email" : "u1@gmail.com", "password" : "password", "settings" : { "bio" : "Hi! I am u
ser1", "privacy" : "private" } }
{ "_id" : ObjectId("5f9ef67b7917e018ad0b6c01"), "userId" : 2, "username" : "user2", "firstname" : "user2", "email" : "u2@gmail.com", "password" : "pwd", "settings" : { "privacy" : "public" } }
{ "_id" : ObjectId("5f9ef9f17917e018ad0b6c02"), "username" : "user3", "firstname" : "user", "lastname" : "three", "settings" : { "bio" : "Hi! I am user3", "privacy" : "public" } }
>
```

Update-

- Updating the privacy settings of a particular user from public to private using update and set.

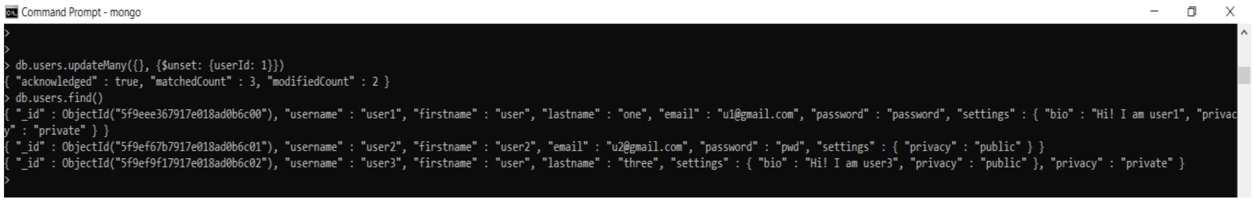
  ```
  db.users.update({_id: ObjectId("5f9ef9f17917e018ad0b6c02")}, { $set: { "privacy": "private"}})
  ```



```
Command Prompt - mongo
> db.users.update({_id: ObjectId("5f9ef9f17917e018ad0b6c02")}, { $set: { "privacy": "private"}})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
> db.users.find()
{ "_id" : ObjectId("5f9eee367917e018ad0b6c00"), "userId" : 1, "username" : "user1", "firstname" : "user", "lastname" : "one", "email" : "u1@gmail.com", "password" : "password", "settings" : { "bio" : "Hi! I am u
ser1", "privacy" : "private" } }
{ "_id" : ObjectId("5f9ef67b7917e018ad0b6c01"), "userId" : 2, "username" : "user2", "firstname" : "user2", "email" : "u2@gmail.com", "password" : "pwd", "settings" : { "privacy" : "public" } }
{ "_id" : ObjectId("5f9ef9f17917e018ad0b6c02"), "username" : "user3", "firstname" : "user", "lastname" : "three", "settings" : { "bio" : "Hi! I am user3", "privacy" : "public" }, "privacy" : "private" }
>
```

- We had initially maintained a userid for the first and second users, but the userid field is unnecessarily redundant because _id can be used for unique identification of users. So now, we update the entries to remove userid field from all the entries that have it, using the updateMany command with unset option.

  ```
  db.users.updateMany({}, {$unset: {userId: 1}})
  ```



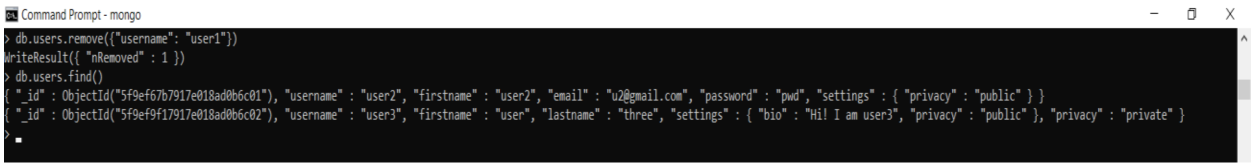
```
Command Prompt - mongo
>
>
> db.users.updateMany({}, {$unset: {userId: 1}})
{ "acknowledged" : true, "matchedCount" : 3, "modifiedCount" : 2 }
> db.users.find()
{ "_id" : ObjectId("5f9eee367917e018ad0b6c00"), "username" : "user1", "firstname" : "user", "lastname" : "one", "email" : "u1@gmail.com", "password" : "password", "settings" : { "bio" : "Hi! I am user1", "privac
y" : "private" } }
{ "_id" : ObjectId("5f9ef67b7917e018ad0b6c01"), "username" : "user2", "firstname" : "user2", "email" : "u2@gmail.com", "password" : "pwd", "settings" : { "privacy" : "public" } }
{ "_id" : ObjectId("5f9ef9f17917e018ad0b6c02"), "username" : "user3", "firstname" : "user", "lastname" : "three", "settings" : { "bio" : "Hi! I am user3", "privacy" : "public" }, "privacy" : "private" }
>
```

Delete-

The remove function removes all the documents that match the query expression.

```
db.users.remove({"username": "user1"})
```

We remove user1 from the collection.



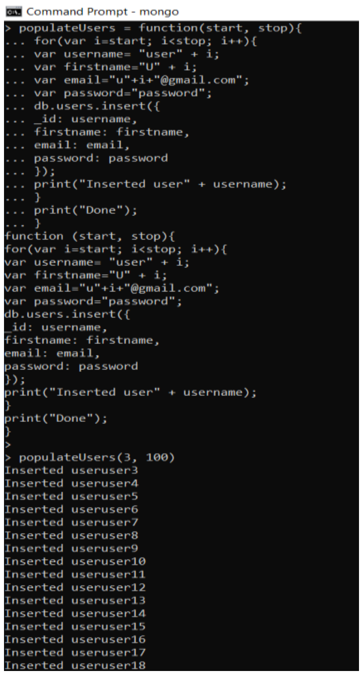
```
> db.users.remove({"username": "user1"})
WriteResult({ "nRemoved" : 1 })
> db.users.find()
{ "_id" : ObjectId("5f9ef67b7917e018ad0b6c01"), "username" : "user2", "firstname" : "user2", "email" : "u2@gmail.com", "password" : "pwd", "settings" : { "privacy" : "public" } }
{ "_id" : ObjectId("5f9ef9f17917e018ad0b6c02"), "username" : "user3", "firstname" : "user", "lastname" : "three", "settings" : { "bio" : "Hi! I am user3", "privacy" : "public" }, "privacy" : "private" }
> ▪
```

Aggregation-

Before we perform aggregation, we populate the users collection with some more user details by writing a function for the same.

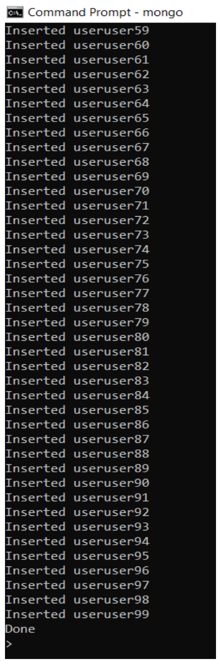
```
populateUsers = function(start, stop){
for(var i=start; i<stop; i++){
var username= "user" + i;
var firstname="U" + i;
var email="u"+i+"@gmail.com";
var password="password";
db.users.insert({
_id: username,
firstname: firstname,
email: email,
password: password
});
print("Inserted user" + username);
}
print("Done");
}
```

populateUsers(3, 100)



Command Prompt - mongo

```
> populateUsers = function(start, stop){
... for(var i=start; i<stop; i++){
... var username= "user" + i;
... var firstname="U" + i;
... var email="u"+i+"@gmail.com";
... var password="password";
... db.users.insert({
... _id: username,
... firstname: firstname,
... email: email,
... password: password
... });
... print("Inserted user" + username);
... }
... print("Done");
... }
function (start, stop){
for(var i=start; i<stop; i++){
var username= "user" + i;
var firstname="U" + i;
var email="u"+i+"@gmail.com";
var password="password";
db.users.insert({
_id: username,
firstname: firstname,
email: email,
password: password
});
print("Inserted user" + username);
}
print("Done");
}
>
> populateUsers(3, 100)
Inserted useruser3
Inserted useruser4
Inserted useruser5
Inserted useruser6
Inserted useruser7
Inserted useruser8
Inserted useruser9
Inserted useruser10
Inserted useruser11
Inserted useruser12
Inserted useruser13
Inserted useruser14
Inserted useruser15
Inserted useruser16
Inserted useruser17
Inserted useruser18
```

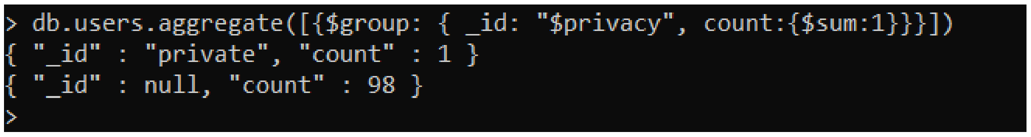


Command Prompt - mongo

```
Inserted useruser59
Inserted useruser60
Inserted useruser61
Inserted useruser62
Inserted useruser63
Inserted useruser64
Inserted useruser65
Inserted useruser66
Inserted useruser67
Inserted useruser68
Inserted useruser69
Inserted useruser70
Inserted useruser71
Inserted useruser72
Inserted useruser73
Inserted useruser74
Inserted useruser75
Inserted useruser76
Inserted useruser77
Inserted useruser78
Inserted useruser79
Inserted useruser80
Inserted useruser81
Inserted useruser82
Inserted useruser83
Inserted useruser84
Inserted useruser85
Inserted useruser86
Inserted useruser87
Inserted useruser88
Inserted useruser89
Inserted useruser90
Inserted useruser91
Inserted useruser92
Inserted useruser93
Inserted useruser94
Inserted useruser95
Inserted useruser96
Inserted useruser97
Inserted useruser98
Inserted useruser99
Done
>
```

Now we group the users based on their privacy settings.

db.users.aggregate([{$group: { _id: "$privacy", count:{$sum:1}}}])

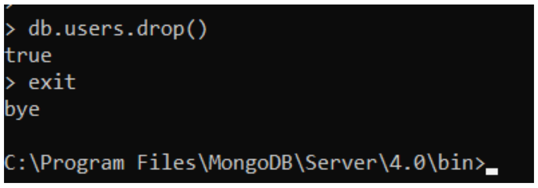


```
> db.users.aggregate([{$group: { _id: "$privacy", count:{$sum:1}}}])
{ "_id" : "private", "count" : 1 }
{ "_id" : null, "count" : 98 }
>
```

Drop-

Remove the collection from the database using the drop function.

db.users.drop()



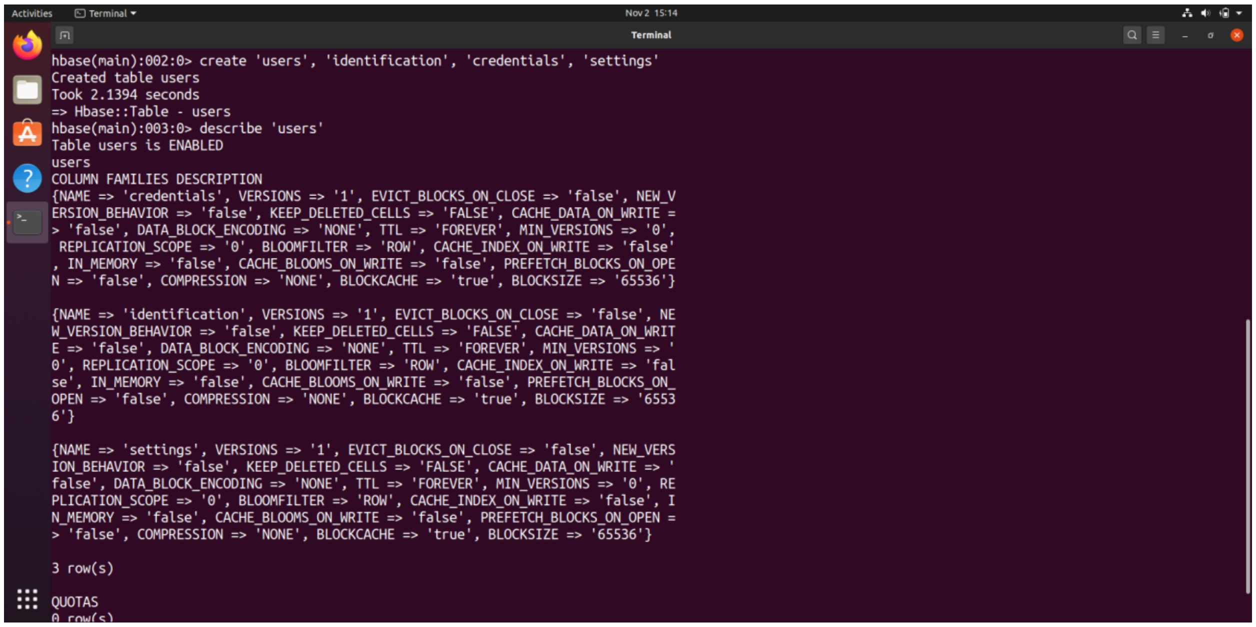
```
> db.users.drop()
true
> exit
bye

C:\Program Files\MongoDB\Server\4.0\bin>
```

Hbase:

Creating a table called ‘users’-

The create command is used to create the table; we’ve added three column families- identification, credentials and settings. The describe command gives us information about the table, including versioning information.

create 'users', 'identification', 'credentials', 'settings'



```
hbase(main):002:0> create 'users', 'identification', 'credentials', 'settings'
Created table users
Took 2.1394 seconds
=> Hbase::Table - users
hbase(main):003:0> describe 'users'
Table users is ENABLED
users
COLUMN FAMILIES DESCRIPTION
{NAME => 'credentials', VERSIONS => '1', EVICT_BLOCKS_ON_CLOSE => 'false', NEW_V
ERSION_BEHAVIOR => 'false', KEEP_DELETED_CELLS => 'FALSE', CACHE_DATA_ON_WRITE =
> 'false', DATA_BLOCK_ENCODING => 'NONE', TTL => 'FOREVER', MIN_VERSIONS => '0',
 REPLICATION_SCOPE => '0', BLOOMFILTER => 'ROW', CACHE_INDEX_ON_WRITE => 'false'
, IN_MEMORY => 'false', CACHE_BLOOMS_ON_WRITE => 'false', PREFETCH_BLOCKS_ON_OPE
N => 'false', COMPRESSION => 'NONE', BLOCKCACHE => 'true', BLOCKSIZE => '65536'}

{NAME => 'identification', VERSIONS => '1', EVICT_BLOCKS_ON_CLOSE => 'false', NE
W_VERSION_BEHAVIOR => 'false', KEEP_DELETED_CELLS => 'FALSE', CACHE_DATA_ON_WRIT
E => 'false', DATA_BLOCK_ENCODING => 'NONE', TTL => 'FOREVER', MIN_VERSIONS => '
0', REPLICATION_SCOPE => '0', BLOOMFILTER => 'ROW', CACHE_INDEX_ON_WRITE => 'fal
se', IN_MEMORY => 'false', CACHE_BLOOMS_ON_WRITE => 'false', PREFETCH_BLOCKS_ON_
OPEN => 'false', COMPRESSION => 'NONE', BLOCKCACHE => 'true', BLOCKSIZE => '6553
6'}

{NAME => 'settings', VERSIONS => '1', EVICT_BLOCKS_ON_CLOSE => 'false', NEW_VERS
ION_BEHAVIOR => 'false', KEEP_DELETED_CELLS => 'FALSE', CACHE_DATA_ON_WRITE => '
false', DATA_BLOCK_ENCODING => 'NONE', TTL => 'FOREVER', MIN_VERSIONS => '0', RE
PLICATION_SCOPE => '0', BLOOMFILTER => 'ROW', CACHE_INDEX_ON_WRITE => 'false', I
N_MEMORY => 'false', CACHE_BLOOMS_ON_WRITE => 'false', PREFETCH_BLOCKS_ON_OPEN =
> 'false', COMPRESSION => 'NONE', BLOCKCACHE => 'true', BLOCKSIZE => '65536'}

3 row(s)

QUOTAS
0 row(s)
```

Insertion and Scan-

This operation can be performed using the put command. With every insertion there is a timestamp associated.

In the figure below we first use the put command with row key as r1, column family as identification, and a column within that column family as username, and the value given is user1. Following this command, with every column family details given in the following put commands we have explicitly specified the timestamp as that of the first put command so that all of them get the same timestamp and it is as if we have inserted all the details for row key r1 at once.

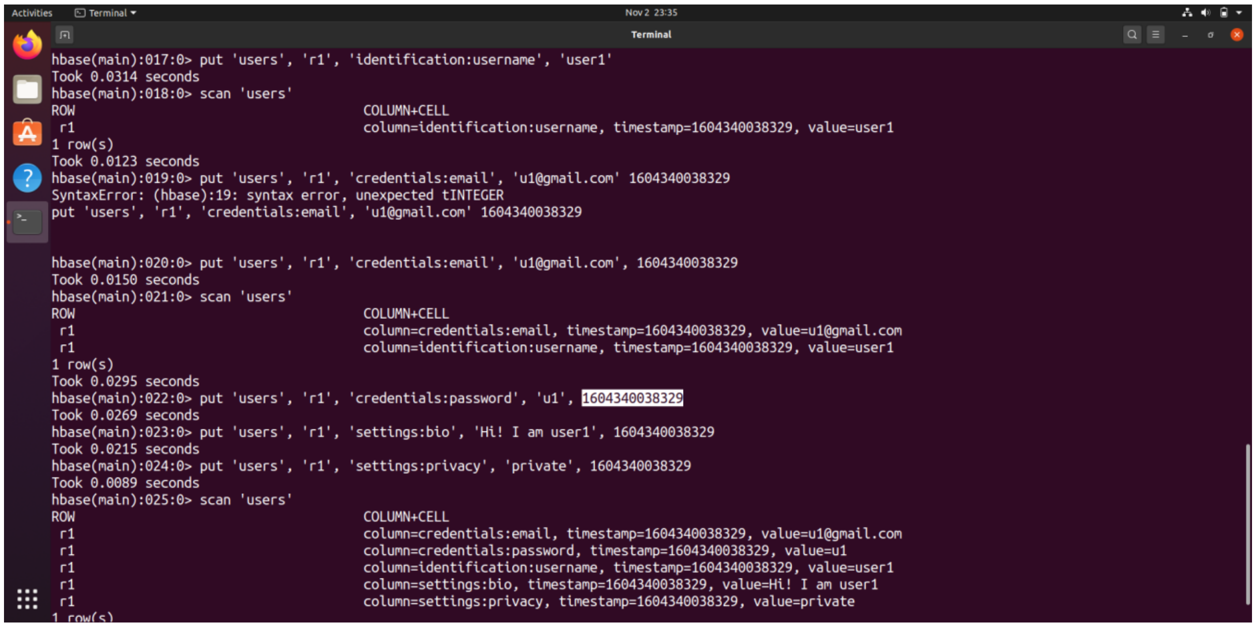
put 'users', 'r1', 'identification:username', 'user1'

put 'users', 'r1', 'credentials:email', 'u1@gmail.com', 1604340038329

put 'users', 'r1', 'credentials:password', 'u1', 1604340038329

put 'users', 'r1', 'settings:bio', 'Hi! I am user1', 1604340038329

put 'users', 'r1', 'settings:privacy', 'private', 1604340038329 scan 'users'



```
hbase(main):017:0> put 'users', 'r1', 'identification:username', 'user1'
Took 0.0314 seconds
hbase(main):018:0> scan 'users'
ROW                                        COLUMN+CELL
 r1                                        column=identification:username, timestamp=1604340038329, value=user1
1 row(s)
Took 0.0123 seconds
hbase(main):019:0> put 'users', 'r1', 'credentials:email', 'u1@gmail.com' 1604340038329
SyntaxError: (hbase):19: syntax error, unexpected tINTEGER
put 'users', 'r1', 'credentials:email', 'u1@gmail.com' 1604340038329


hbase(main):020:0> put 'users', 'r1', 'credentials:email', 'u1@gmail.com', 1604340038329
Took 0.0150 seconds
hbase(main):021:0> scan 'users'
ROW                                        COLUMN+CELL
 r1                                        column=credentials:email, timestamp=1604340038329, value=u1@gmail.com
 r1                                        column=identification:username, timestamp=1604340038329, value=user1
1 row(s)
Took 0.0295 seconds
hbase(main):022:0> put 'users', 'r1', 'credentials:password', 'u1', 1604340038329
Took 0.0269 seconds
hbase(main):023:0> put 'users', 'r1', 'settings:bio', 'Hi! I am user1', 1604340038329
Took 0.0215 seconds
hbase(main):024:0> put 'users', 'r1', 'settings:privacy', 'private', 1604340038329
Took 0.0089 seconds
hbase(main):025:0> scan 'users'
ROW                                        COLUMN+CELL
 r1                                        column=credentials:email, timestamp=1604340038329, value=u1@gmail.com
 r1                                        column=credentials:password, timestamp=1604340038329, value=u1
 r1                                        column=identification:username, timestamp=1604340038329, value=user1
 r1                                        column=settings:bio, timestamp=1604340038329, value=Hi! I am user1
 r1                                        column=settings:privacy, timestamp=1604340038329, value=private
1 row(s)
```

In the figure below the details of the second user are added with row key r2. But this time the timestamps are not mentioned in the put commands, so with each put command there is a unique timestamp. The scan command shows us the contents of the table.

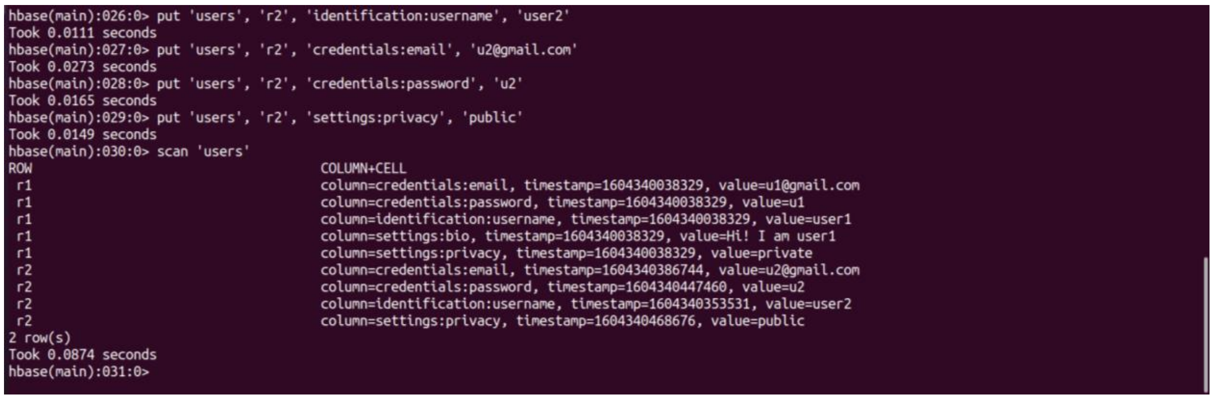
put 'users', 'r2', 'identification:username', 'user2'

put 'users', 'r2', 'credentials:email', 'u2@gmail.com'

put 'users', 'r2', 'credentials:password', 'u2'

put 'users', 'r2', 'settings:privacy', 'public'

scan 'users'

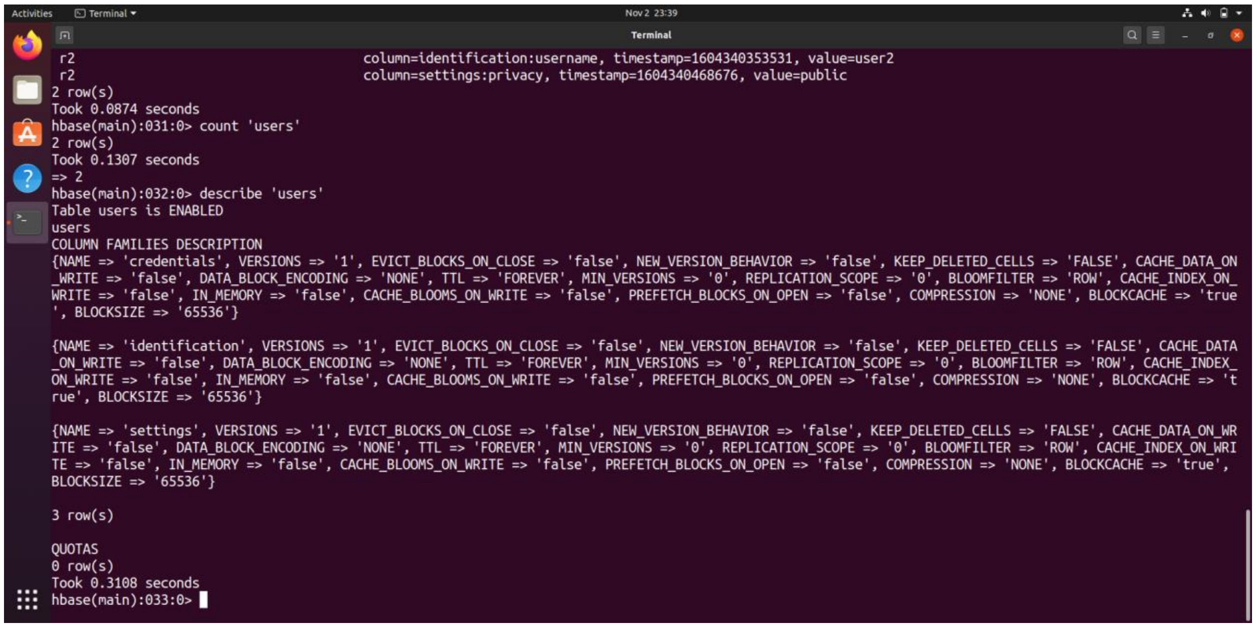


```
hbase(main):026:0> put 'users', 'r2', 'identification:username', 'user2'
Took 0.0111 seconds
hbase(main):027:0> put 'users', 'r2', 'credentials:email', 'u2@gmail.com'
Took 0.0273 seconds
hbase(main):028:0> put 'users', 'r2', 'credentials:password', 'u2'
Took 0.0165 seconds
hbase(main):029:0> put 'users', 'r2', 'settings:privacy', 'public'
Took 0.0149 seconds
hbase(main):030:0> scan 'users'
ROW                                        COLUMN+CELL
 r1                                        column=credentials:email, timestamp=1604340038329, value=u1@gmail.com
 r1                                        column=credentials:password, timestamp=1604340038329, value=u1
 r1                                        column=identification:username, timestamp=1604340038329, value=user1
 r1                                        column=settings:bio, timestamp=1604340038329, value=Hi! I am user1
 r1                                        column=settings:privacy, timestamp=1604340038329, value=private
 r2                                        column=credentials:email, timestamp=1604340386744, value=u2@gmail.com
 r2                                        column=credentials:password, timestamp=1604340447460, value=u2
 r2                                        column=identification:username, timestamp=1604340353531, value=user2
 r2                                        column=settings:privacy, timestamp=1604340468676, value=public
2 row(s)
Took 0.0874 seconds
hbase(main):031:0>
```

Now we can view the number of users/rows in the table is 2.

count 'users'

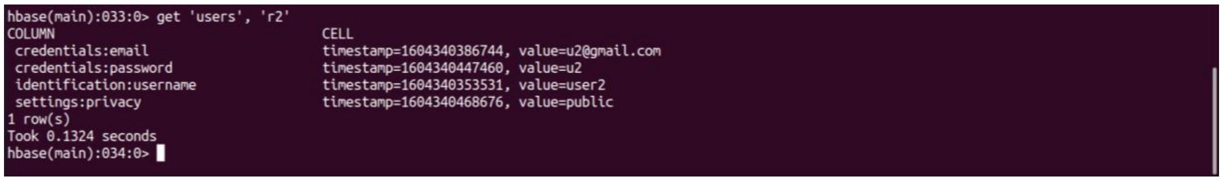
describe 'users'



```
 r2                              column=identification:username, timestamp=1604340353531, value=user2
 r2                              column=settings:privacy, timestamp=1604340468676, value=public
2 row(s)
Took 0.0874 seconds
hbase(main):031:0> count 'users'
2 row(s)
Took 0.1307 seconds
=> 2
hbase(main):032:0> describe 'users'
Table users is ENABLED
users
COLUMN FAMILIES DESCRIPTION
{NAME => 'credentials', VERSIONS => '1', EVICT_BLOCKS_ON_CLOSE => 'false', NEW_VERSION_BEHAVIOR => 'false', KEEP_DELETED_CELLS => 'FALSE', CACHE_DATA_ON
_WRITE => 'false', DATA_BLOCK_ENCODING => 'NONE', TTL => 'FOREVER', MIN_VERSIONS => '0', REPLICATION_SCOPE => '0', BLOOMFILTER => 'ROW', CACHE_INDEX_ON_
WRITE => 'false', IN_MEMORY => 'false', CACHE_BLOOMS_ON_WRITE => 'false', PREFETCH_BLOCKS_ON_OPEN => 'false', COMPRESSION => 'NONE', BLOCKCACHE => 'true
', BLOCKSIZE => '65536'}

{NAME => 'identification', VERSIONS => '1', EVICT_BLOCKS_ON_CLOSE => 'false', NEW_VERSION_BEHAVIOR => 'false', KEEP_DELETED_CELLS => 'FALSE', CACHE_DATA
_ON_WRITE => 'false', DATA_BLOCK_ENCODING => 'NONE', TTL => 'FOREVER', MIN_VERSIONS => '0', REPLICATION_SCOPE => '0', BLOOMFILTER => 'ROW', CACHE_INDEX_
ON_WRITE => 'false', IN_MEMORY => 'false', CACHE_BLOOMS_ON_WRITE => 'false', PREFETCH_BLOCKS_ON_OPEN => 'false', COMPRESSION => 'NONE', BLOCKCACHE => 't
rue', BLOCKSIZE => '65536'}

{NAME => 'settings', VERSIONS => '1', EVICT_BLOCKS_ON_CLOSE => 'false', NEW_VERSION_BEHAVIOR => 'false', KEEP_DELETED_CELLS => 'FALSE', CACHE_DATA_ON_WR
ITE => 'false', DATA_BLOCK_ENCODING => 'NONE', TTL => 'FOREVER', MIN_VERSIONS => '0', REPLICATION_SCOPE => '0', BLOOMFILTER => 'ROW', CACHE_INDEX_ON_WRI
TE => 'false', IN_MEMORY => 'false', CACHE_BLOOMS_ON_WRITE => 'false', PREFETCH_BLOCKS_ON_OPEN => 'false', COMPRESSION => 'NONE', BLOCKCACHE => 'true',
BLOCKSIZE => '65536'}

3 row(s)

QUOTAS
0 row(s)
Took 0.3108 seconds
hbase(main):033:0>
```

The details of particular user can be obtained with the get command.

get 'users', 'r2'



```
hbase(main):033:0> get 'users', 'r2'
COLUMN                                   CELL
 credentials:email                       timestamp=1604340386744, value=u2@gmail.com
 credentials:password                    timestamp=1604340447460, value=u2
 identification:username                 timestamp=1604340353531, value=user2
 settings:privacy                        timestamp=1604340468676, value=public
1 row(s)
Took 0.1324 seconds
hbase(main):034:0>
```

Updating-

We can perform updation of an existing row's details using the put command itself. Here, the aim to to store three versions of changes for any field. So we first disable the table, use an alter command to update the number of versions from 1 to 3 and then enable the table. Following that we can make changes using put.

Notice that the latest three versions are maintained, even if the number of changes made to a field are more than 3.

disable 'users'

alter 'users', {NAME => 'settings', VERSIONS => 3}

enable 'users'

put 'users', 'r1', 'settings:privacy', 'public'

get 'users', 'r1', {COLUMN => 'settings', VERSIONS => 3}

put 'users', 'r1', 'settings:privacy', 'private'

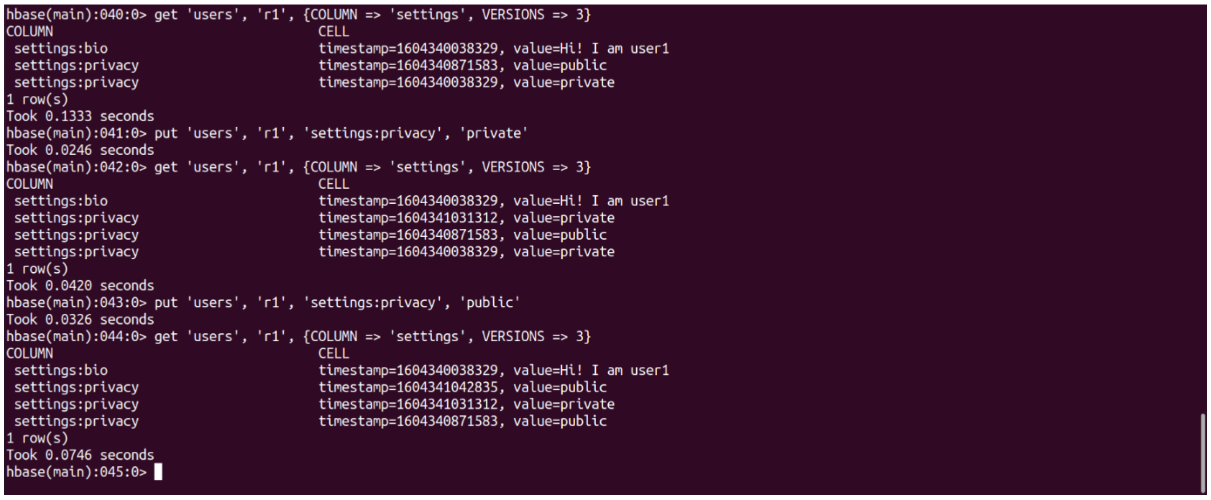
get 'users', 'r1', {COLUMN => 'settings', VERSIONS => 3}

put 'users', 'r1', 'settings:privacy', 'public'

get 'users', 'r1', {COLUMN => 'settings', VERSIONS => 3}

put 'users', 'r1', 'settings:privacy', 'private'

get 'users', 'r1', {COLUMN => 'settings', VERSIONS => 3



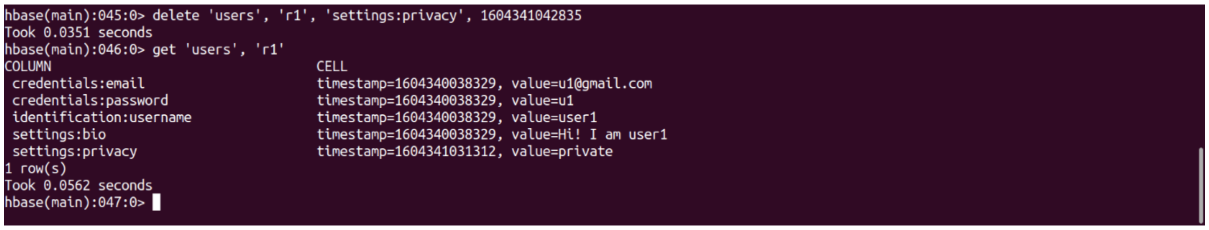
```
hbase(main):040:0> get 'users', 'r1', {COLUMN => 'settings', VERSIONS => 3}
COLUMN                                CELL
 settings:bio                         timestamp=1604340038329, value=Hi! I am user1
 settings:privacy                     timestamp=1604340871583, value=public
 settings:privacy                     timestamp=1604340038329, value=private
1 row(s)
Took 0.1333 seconds
hbase(main):041:0> put 'users', 'r1', 'settings:privacy', 'private'
Took 0.0246 seconds
hbase(main):042:0> get 'users', 'r1', {COLUMN => 'settings', VERSIONS => 3}
COLUMN                                CELL
 settings:bio                         timestamp=1604340038329, value=Hi! I am user1
 settings:privacy                     timestamp=1604341031312, value=private
 settings:privacy                     timestamp=1604340871583, value=public
 settings:privacy                     timestamp=1604340038329, value=private
1 row(s)
Took 0.0420 seconds
hbase(main):043:0> put 'users', 'r1', 'settings:privacy', 'public'
Took 0.0326 seconds
hbase(main):044:0> get 'users', 'r1', {COLUMN => 'settings', VERSIONS => 3}
COLUMN                                CELL
 settings:bio                         timestamp=1604340038329, value=Hi! I am user1
 settings:privacy                     timestamp=1604341042835, value=public
 settings:privacy                     timestamp=1604341031312, value=private
 settings:privacy                     timestamp=1604340871583, value=public
1 row(s)
Took 0.0746 seconds
hbase(main):045:0> 
```

Deletion-

Deletion of details can be performed based on timestamp, in case a specific version needs to be deleted.

delete 'users', 'r1', 'settings:privacy', 1604341042835

get 'users', 'r1'

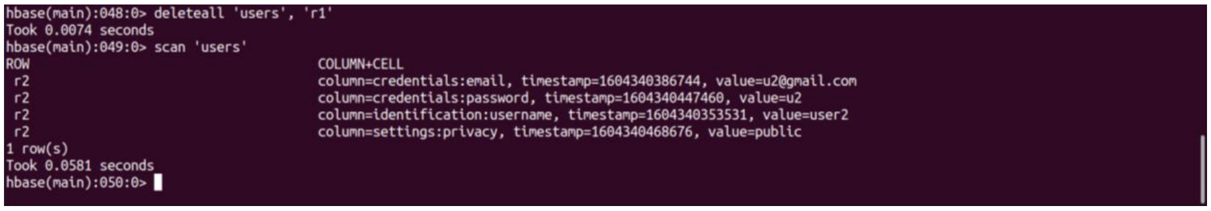


```
hbase(main):045:0> delete 'users', 'r1', 'settings:privacy', 1604341042835
Took 0.0351 seconds
hbase(main):046:0> get 'users', 'r1'
COLUMN                                CELL
 credentials:email                    timestamp=1604340038329, value=u1@gmail.com
 credentials:password                 timestamp=1604340038329, value=u1
 identification:username              timestamp=1604340038329, value=user1
 settings:bio                         timestamp=1604340038329, value=Hi! I am user1
 settings:privacy                     timestamp=1604341031312, value=private
1 row(s)
Took 0.0562 seconds
hbase(main):047:0> 
```

Or a row key can be entirely deleted with all the versions using the deleteall command.

deleteall 'users', 'r1'

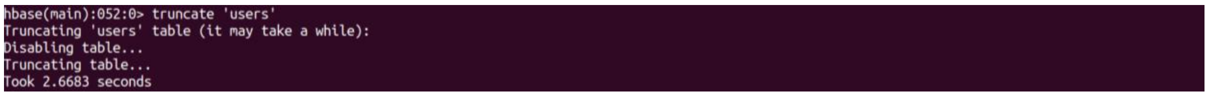
scan 'users'



```
hbase(main):048:0> deleteall 'users', 'r1'
Took 0.0074 seconds
hbase(main):049:0> scan 'users'
ROW                                     COLUMN+CELL
 r2                                     column=credentials:email, timestamp=1604340386744, value=u2@gmail.com
 r2                                     column=credentials:password, timestamp=1604340447460, value=u2
 r2                                     column=identification:username, timestamp=1604340353531, value=user2
 r2                                     column=settings:privacy, timestamp=1604340468676, value=public
1 row(s)
Took 0.0581 seconds
hbase(main):050:0>
```

A table can be emptied of all rows using the truncate command. It automatically diables the table before performing the operation.

truncate 'users'

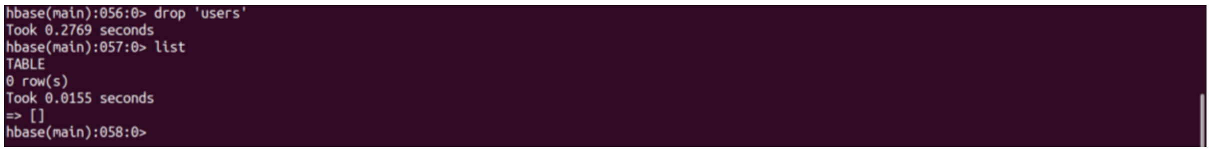


```
hbase(main):052:0> truncate 'users'
Truncating 'users' table (it may take a while):
Disabling table...
Truncating table...
Took 2.6683 seconds
```

A table can be dropped using the drop command.

disable 'users'

drop 'users'



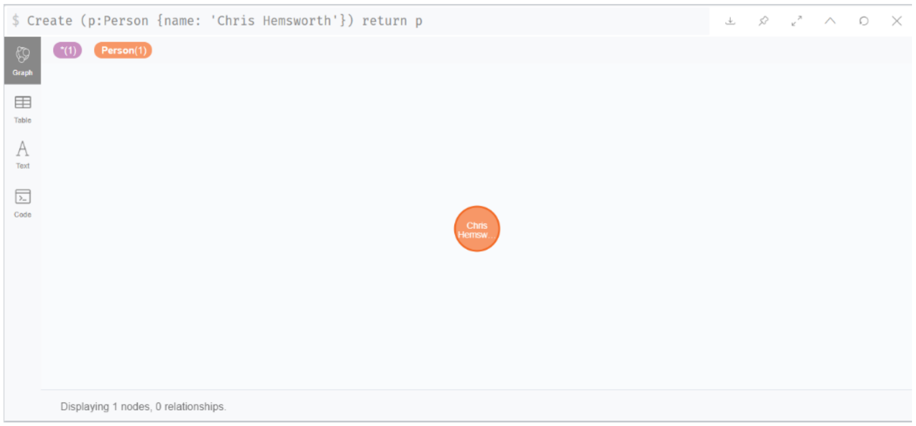
```
hbase(main):056:0> drop 'users'
Took 0.2769 seconds
hbase(main):057:0> list
TABLE
0 row(s)
Took 0.0155 seconds
=> []
hbase(main):058:0>
```

Neo4J:

Creating nodes-

We create a new node under Person in the Movies database.

Create (p:Person {name: 'Chris Hemsworth'}) return p

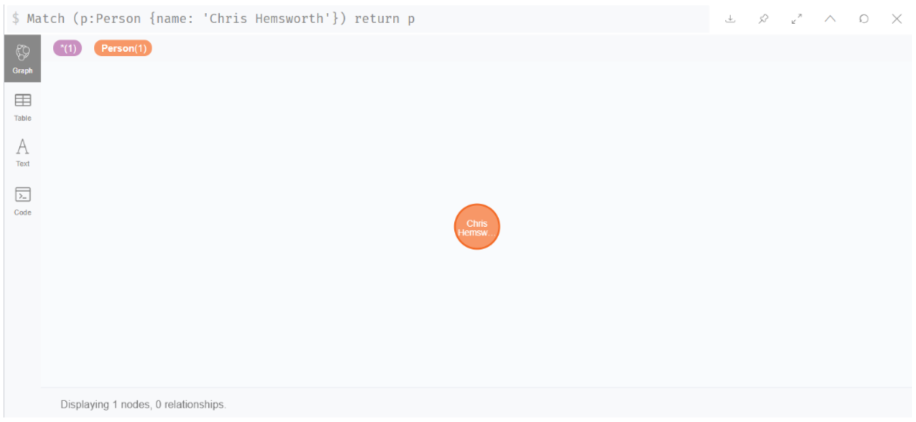


```
$ Create (p:Person {name: 'Chris Hemsworth'}) return p
```

Displaying 1 nodes, 0 relationships.

The node created can be displayed using the MATCH command.

MATCH (p:Person {name: 'Chris Hemsworth'}) return p

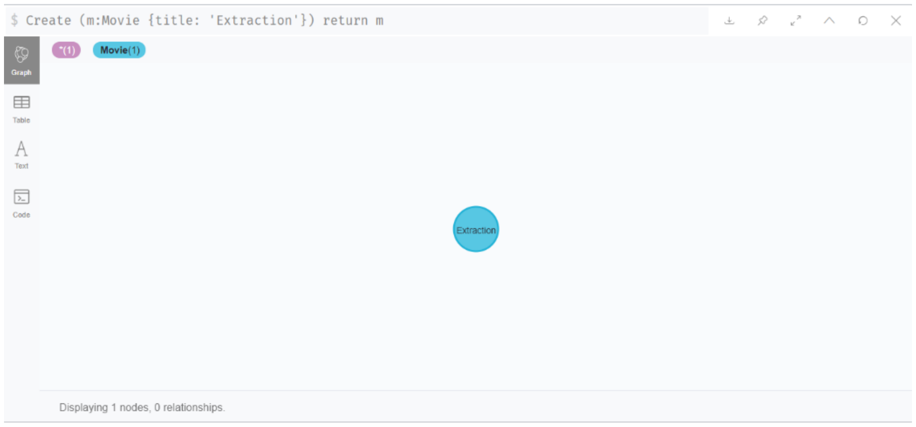


```
$ Match (p:Person {name: 'Chris Hemsworth'}) return p
```

Displaying 1 nodes, 0 relationships.

We also add a new Movie called ‘Extraction’ to which we can map the person we added above with the acted_in relationship.

Create (m:Movie {title: 'Extraction'}) return m



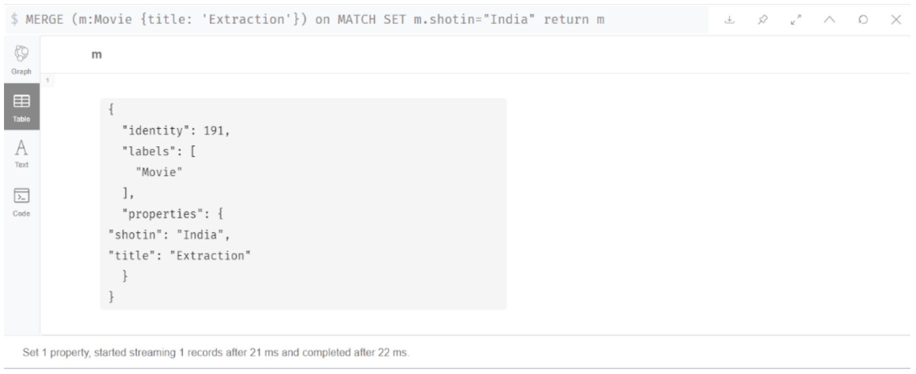
```
$ Create (m:Movie {title: 'Extraction'}) return m
```

Displaying 1 nodes, 0 relationships.

Adding attributes to the new nodes-

We add the shotin attribute to the movie titled ‘Extraction’ which denotes the place where the movie was shot.

MERGE (m:Movie {title: 'Extraction'}) on MATCH SET m.shotin="India" return m



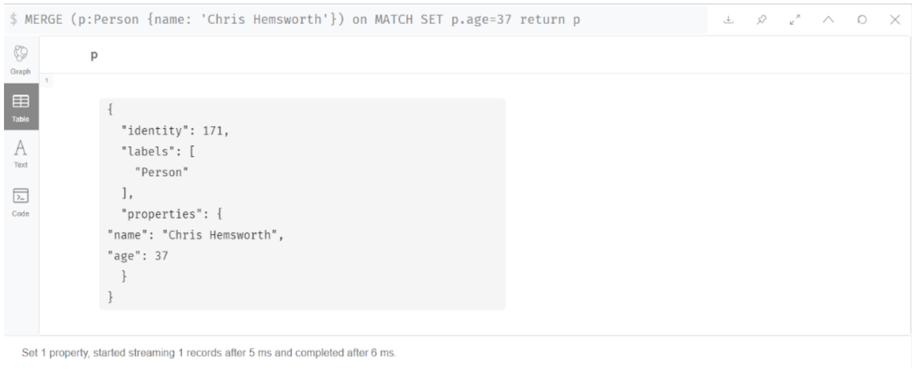
```
$ MERGE (m:Movie {title: 'Extraction'}) on MATCH SET m.shotin="India" return m
```

```
{
  "identity": 191,
  "labels": [
    "Movie"
  ],
  "properties": {
"shotin": "India",
"title": "Extraction"
  }
}
```

Set 1 property, started streaming 1 records after 21 ms and completed after 22 ms.

Similarly, for the person just added, the age attribute is added.

MERGE (p:Person {name: 'Chris Hemsworth'}) on MATCH SET p.age=37 return p



```
$ MERGE (p:Person {name: 'Chris Hemsworth'}) on MATCH SET p.age=37 return p
```

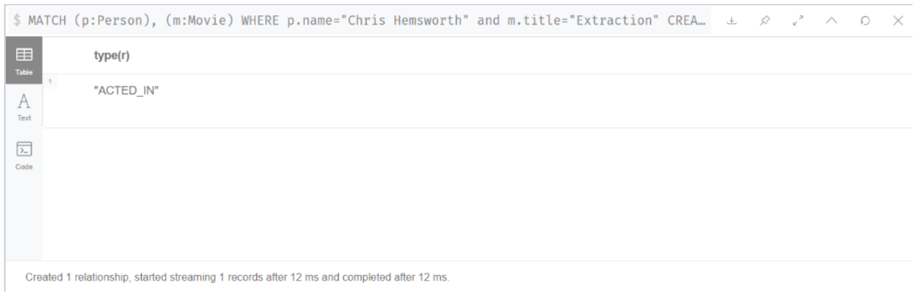
```
{
  "identity": 171,
  "labels": [
    "Person"
  ],
  "properties": {
"name": "Chris Hemsworth",
"age": 37
  }
}
```

Set 1 property, started streaming 1 records after 5 ms and completed after 6 ms.

Establishing relationship-

We relate the person node to the movie node newly created using the MATCH command based on the acted_in relationship.

```
MATCH (p:Person), (m:Movie)
WHERE p.name="Chris Hemsworth" and m.title="Extraction"
CREATE (p)-[r:ACTED_IN]->(m)
RETURN type(r)
```



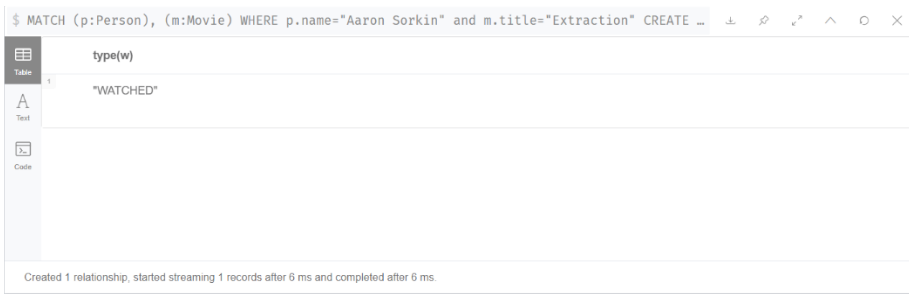
```
$ MATCH (p:Person), (m:Movie) WHERE p.name="Chris Hemsworth" and m.title="Extraction" CREA…
```

	type(r)
1	"ACTED_IN"

```
Created 1 relationship, started streaming 1 records after 12 ms and completed after 12 ms.
```

Another relationship we add is between the person 'Aaron Sorkin' and the movie 'Extraction', and the relationship is that this person has watched this movie.

```
MATCH (p:Person), (m:Movie)
WHERE p.name="Aaron Sorkin" and m.title="Extraction"
CREATE (p)-[w:WATCHED]->(m)
RETURN type(w)
```



```
$ MATCH (p:Person), (m:Movie) WHERE p.name="Aaron Sorkin" and m.title="Extraction" CREATE …
```

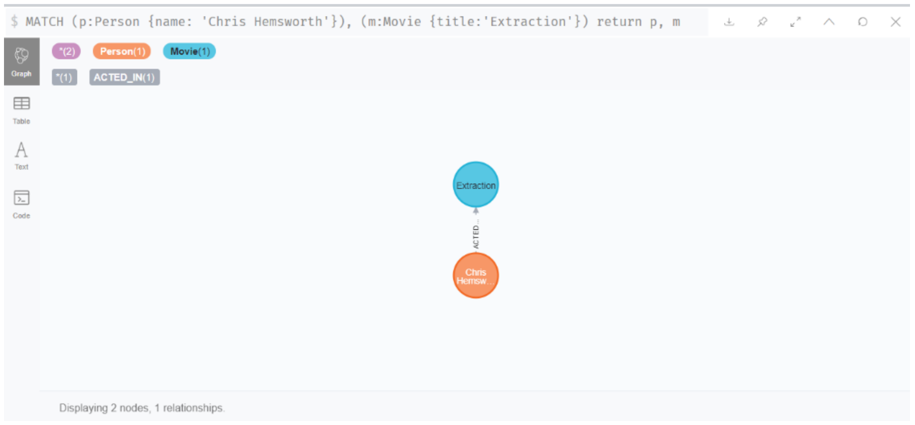
	type(w)
1	"WATCHED"

```
Created 1 relationship, started streaming 1 records after 6 ms and completed after 6 ms.
```

Retrieving the newly added attributes-

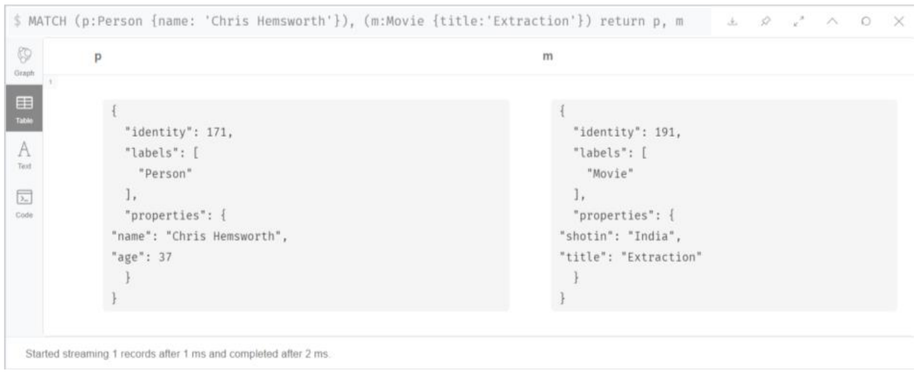
All the attributes for the newly added nodes can be displayed with the follwing command.

MATCH (p:Person {name: 'Chris Hemsworth'}), (m:Movie {title:'Extraction'}) return p, m



```
$ MATCH (p:Person {name: 'Chris Hemsworth'}), (m:Movie {title:'Extraction'}) return p, m
```

Displaying 2 nodes, 1 relationships.



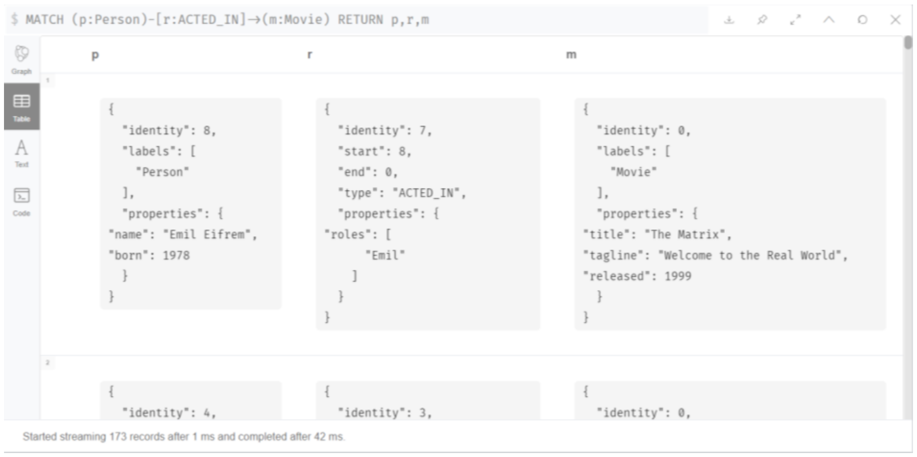
```
$ MATCH (p:Person {name: 'Chris Hemsworth'}), (m:Movie {title:'Extraction'}) return p, m
```

p	m
{ "identity": 171, "labels": ["Person"], "properties": { "name": "Chris Hemsworth", "age": 37 } }	{ "identity": 191, "labels": ["Movie"], "properties": { "shotin": "India", "title": "Extraction" } }

Started streaming 1 records after 1 ms and completed after 2 ms.

The following command is useful to perform a join of the tables Person and Movie based on the acted_in relationship.

MATCH (p:Person)-[r:ACTED_IN]->(m:Movie) RETURN p,r,m



```
$ MATCH (p:Person)-[r:ACTED_IN]→(m:Movie) RETURN p,r,m
```

p	r	m
{ "identity": 8, "labels": ["Person"], "properties": { "name": "Emil Eifrem", "born": 1978 } }	{ "identity": 7, "start": 8, "end": 0, "type": "ACTED_IN", "properties": { "roles": ["Emil"] } }	{ "identity": 0, "labels": ["Movie"], "properties": { "title": "The Matrix", "tagline": "Welcome to the Real World", "released": 1999 } }
{ "identity": 4,	{ "identity": 3,	{ "identity": 0,

Started streaming 173 records after 1 ms and completed after 42 ms.