# PES UNIVERSITY

# ELECTIVE 1: DATABASE TECHNOLOGIES (UE18CS315)

# ASSIGNMENT 4

**NAME:** SHAAZIN SHEIKH SHUKOOR

**SRN:** PES1201801754

**SEMESTER:** 5

**SECTION:** J

## Create a database in MongoDB and execute basic operations – insert, update, delete, and aggregation

**Starting Mongodb:**

```
mongo
```

```
user1@user1-Lenovo-G580:~$ sudo service mongodb start
user1@user1-Lenovo-G580:~$ sudo service mongodb status
● mongodb.service - An object/document-oriented database
   Loaded: loaded (/lib/systemd/system/mongodb.service; enabled; vendor preset: enabled)
   Active: active (running) since Fri 2020-10-30 21:20:51 IST; 3min 9s ago
     Docs: man:mongod(1)
 Main PID: 8229 (mongod)
    Tasks: 23 (limit: 4434)
   CGroup: /system.slice/mongodb.service
           └─8229 /usr/bin/mongod --unixSocketPrefix=/run/mongodb --config /etc/mongodb.conf

Oct 30 21:20:51 user1-Lenovo-G580 systemd[1]: Started An object/document-oriented database.
user1@user1-Lenovo-G580:~$ mongo
MongoDB shell version v3.6.3
connecting to: mongodb://127.0.0.1:27017
MongoDB server version: 3.6.3
Welcome to the MongoDB shell.
For interactive help, type "help".
For more comprehensive documentation, see
        http://docs.mongodb.org/
Questions? Try the support group
        http://groups.google.com/group/mongodb-user
Server has startup warnings:
2020-10-30T21:20:51.478+0530 I STORAGE  [initandlisten]
2020-10-30T21:20:51.478+0530 I STORAGE  [initandlisten] ** WARNING: Using the XFS filesystem is strongly recommended with the WiredTiger stora
ge engine
2020-10-30T21:20:51.478+0530 I STORAGE  [initandlisten] **          See http://dochub.mongodb.org/core/prodnotes-filesystem
2020-10-30T21:20:53.333+0530 I CONTROL  [initandlisten]
2020-10-30T21:20:53.333+0530 I CONTROL  [initandlisten] ** WARNING: Access control is not enabled for the database.
2020-10-30T21:20:53.333+0530 I CONTROL  [initandlisten] **          Read and write access to data and configuration is unrestricted.
2020-10-30T21:20:53.333+0530 I CONTROL  [initandlisten]
```

**Creating a database:**

```
use dbt
```

```
> show dbs;
admin   0.000GB
config  0.000GB
local   0.000GB
> use dbt;
switched to db dbt
> db
dbt
```

**Inserting records into the database:**

```
db.states.insert({
name:"Karnataka",
population:22200000,
lastCensus:ISODate("2019-06-19"),
famousFor:["dosa","coffee","beaches"],
mayor:{
name:"Tom",
party:"D"
}
})
```

```
> db.states.insert({
... name:"Karnataka",
... population:22200000,
... lastCensus:ISODate("2019-06-19"),
... famousFor:["dosa","coffee","beaches"],
... mayor:{
... name:"Tom",
... party:"D"
... }
... })
WriteResult({ "nInserted" : 1 })
> show collections
states
> db.states.find()
{ "_id" : ObjectId("5f9d6a6027705e6cf3157926"), "name" : "Karnataka", "population" : 22200000, "lastCensus" : ISODate("2019-06-19T00:00:00Z"),
 "famousFor" : [ "dosa", "coffee", "beaches" ], "mayor" : { "name" : "Tom", "party" : "D" } }
> db.states.find().count()
1
```

```
db.states.find()                    /*display all records*/
db.states.find().count()            /*count of the records*/
```

```
> show dbs;
admin   0.000GB
config  0.000GB
dbt     0.000GB
local   0.000GB
> db
dbt
> function insertPlace(
... name,population,lastCensus,
... famousFor,mayorInfo
... ){
... db.states.insert({
... name:name,
... poulation:population,
... lastCensus:ISODate(lastCensus),
... famousFor:famousFor,
... mayor:mayorInfo
... });
... }
> insertPlace("Kerala",6200,"2019-01-31",["Coconut"],{name:"Jerry"})
> insertPlace("Tamil Nadu",58000,"2019-09-20",["food","honey","forest"],{name:"Harry",party:"D"})
> db.states.find.count()
2020-10-31T19:30:42.124+0530 E QUERY    [thread1] TypeError: db.states.find.count is not a function :
@(shell):1:1
```

```
> db.states.find().count()
3
> db.states.find()
{ "_id" : ObjectId("5f9d6a6027705e6cf3157926"), "name" : "Karnataka", "population" : 22200000, "lastCensus" : ISODate("2019-06-19T00:00:00Z"),
 "famousFor" : [ "dosa", "coffee", "beaches" ], "mayor" : { "name" : "Tom", "party" : "D" } }
{ "_id" : ObjectId("5f9d6d86ba98701c046982b4"), "name" : "Kerala", "poulation" : 6200, "lastCensus" : ISODate("2019-01-31T00:00:00Z"), "famous
For" : [ "Coconut" ], "mayor" : { "name" : "Jerry" } }
{ "_id" : ObjectId("5f9d6df9ba98701c046982b5"), "name" : "Tamil Nadu", "poulation" : 58000, "lastCensus" : ISODate("2019-09-20T00:00:00Z"), "f
amousFor" : [ "food", "honey", "forest" ], "mayor" : { "name" : "Harry", "party" : "D" } }
```

Each record has its own unique object id

A record can be retrieved using its object id

```
db.states.find({"_id":ObjectId("5f9d6a6027705e6cf3157926")})
```

```
> db.states.find()
{ "_id" : ObjectId("5f9d6a6027705e6cf3157926"), "name" : "Karnataka", "population" : 22200000, "lastCensus" : ISODate("2019-06-19T00:
 "famousFor" : [ "dosa", "coffee", "beaches" ], "mayor" : { "name" : "Tom", "party" : "D" } }
{ "_id" : ObjectId("5f9d6d86ba98701c046982b4"), "name" : "Kerala", "poulation" : 6200, "lastCensus" : ISODate("2019-01-31T00:00:00Z")
For" : [ "Coconut" ], "mayor" : { "name" : "Jerry" } }
{ "_id" : ObjectId("5f9d6df9ba98701c046982b5"), "name" : "Tamil Nadu", "poulation" : 58000, "lastCensus" : ISODate("2019-09-20T00:00:
amousFor" : [ "food", "honey", "forest" ], "mayor" : { "name" : "Harry", "party" : "D" } }
> db.states.find({"_id":ObjectId("5f9d6a6027705e6cf3157926")})
{ "_id" : ObjectId("5f9d6a6027705e6cf3157926"), "name" : "Karnataka", "population" : 22200000, "lastCensus" : ISODate("2019-06-19T00:
 "famousFor" : [ "dosa", "coffee", "beaches" ], "mayor" : { "name" : "Tom", "party" : "D" } }
> db.states.find({"_id":ObjectId("5f9d6a6027705e6cf3157926")},{name:1})
{ "_id" : ObjectId("5f9d6a6027705e6cf3157926"), "name" : "Karnataka" }
> db.states.find({"_id":ObjectId("5f9d6a6027705e6cf3157926")},{name:0})
{ "_id" : ObjectId("5f9d6a6027705e6cf3157926"), "population" : 22200000, "lastCensus" : ISODate("2019-06-19T00:00:00Z"), "famousFor"
", "coffee", "beaches" ], "mayor" : { "name" : "Tom", "party" : "D" } }
```

**Updating records of the database:**

The attribute ‘population’ is misspelled. Correcting it using the update command:

```
db.states.update( { _id: ObjectId("5f9d6d86ba98701c046982b4") }, {
$rename: { 'poulation': 'population' } } )
```

```
> db.states.update( { _id: ObjectId("5f9d6d86ba98701c046982b4") }, { $rename: { 'poulation': 'population' } } )
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
```

Updating few more records:

```
> db.states.update( { _id: ObjectId("5f9d6a6027705e6cf3157926") }, { $set: { 'capital': 'Bengaluru' } } )
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
> db.states.find()
{ "_id" : ObjectId("5f9d6a6027705e6cf3157926"), "name" : "Karnataka", "population" : 22200000, "lastCensus" : ISODate("2019-06-19T00:00:00Z"),
 "famousFor" : [ "dosa", "coffee", "beaches" ], "mayor" : { "name" : "Tom", "party" : "D" }, "capital" : "Bengaluru" }
{ "_id" : ObjectId("5f9d6d86ba98701c046982b4"), "name" : "Kerala", "lastCensus" : ISODate("2019-01-31T00:00:00Z"), "famousFor" : [ "Coconut" ]
, "mayor" : { "name" : "Jerry" }, "population" : 6200 }
{ "_id" : ObjectId("5f9d6df9ba98701c046982b5"), "name" : "Tamil Nadu", "poulation" : 58000, "lastCensus" : ISODate("2019-09-20T00:00:00Z"), "f
amousFor" : [ "food", "honey", "forest" ], "mayor" : { "name" : "Harry", "party" : "D" } }
> db.states.update( { _id: ObjectId("5f9d6a6027705e6cf3157926") }, { $inc: { 'population':10 } } )
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
> db.states.find()
{ "_id" : ObjectId("5f9d6a6027705e6cf3157926"), "name" : "Karnataka", "population" : 22200010, "lastCensus" : ISODate("2019-06-19T00:00:00Z"),
 "famousFor" : [ "dosa", "coffee", "beaches" ], "mayor" : { "name" : "Tom", "party" : "D" }, "capital" : "Bengaluru" }
{ "_id" : ObjectId("5f9d6d86ba98701c046982b4"), "name" : "Kerala", "lastCensus" : ISODate("2019-01-31T00:00:00Z"), "famousFor" : [ "Coconut" ]
, "mayor" : { "name" : "Jerry" }, "population" : 6200 }
{ "_id" : ObjectId("5f9d6df9ba98701c046982b5"), "name" : "Tamil Nadu", "poulation" : 58000, "lastCensus" : ISODate("2019-09-20T00:00:00Z"), "f
amousFor" : [ "food", "honey", "forest" ], "mayor" : { "name" : "Harry", "party" : "D" } }
>
```

Usage of a function:

```
> var population_range={
... $lt:1000000000000,
... $gt:1000
... }
> db.states.find(
... ... {name:/^P/,population:population_range},
... ... {name:1}
... )
2020-10-31T20:10:01.911+0530 E QUERY    [thread1] SyntaxError: missing ) after argument list @(shell):4:0
> db.states.find( {name:/^P/,population:population_range}, {name:1} )
> db.states.find( {name:/^K/,population:population_range}, {name:1} )
{ "_id" : ObjectId("5f9d6a6027705e6cf3157926"), "name" : "Karnataka" }
{ "_id" : ObjectId("5f9d6d86ba98701c046982b4"), "name" : "Kerala" }
>
```

```
> db.help()
DB methods:
        db.adminCommand(nameOrDocument) - switches to 'admin' db, and runs command [just calls db.runCommand(...)]
        db.aggregate([pipeline], {options}) - performs a collectionless aggregation on this database; returns a cursor
        db.auth(username, password)
        db.cloneDatabase(fromhost)
        db.commandHelp(name) returns the help for the command
        db.copyDatabase(fromdb, todb, fromhost)
        db.createCollection(name, {size: ..., capped: ..., max: ...})
        db.createView(name, viewOn, [{$operator: {...}}, ...], {viewOptions})
```

```
> db.states.help()
DBCollection help
        db.states.find().help() - show DBCursor help
        db.states.bulkWrite( operations, <optional params> ) - bulk execute write operations, optional parameters are: w,
        db.states.count( query = {}, <optional params> ) - count the number of documents that matches the query, optional
t, skip, hint, maxTimeMS
        db.states.copyTo(newColl) - duplicates collection by copying all documents to newColl; no indexes are copied.
        db.states.convertToCapped(maxBytes) - calls {convertToCapped:'states', size:maxBytes}} command
        db.states.createIndex(keypattern[,options])
        db.states.createIndexes([keypatterns], <options>)
        db.states.dataSize()
```

**Searching for particular records and projecting certain columns:**

```
db.states.find( {name:/^K/,population:population_range}, {name:1} )
db.states.find( {famousFor:'food'}, {_id:0,name:1,famousFor:1})
db.states.find({famousFor:{$nin:['food','honey','forest']}},{_id:0,nam
e:1,famousFor:1})
db.states.find({'mayorparty':{$exists:false}},{_id:0,name:1,mayor:1})
db.states.find({'mayor.party':{$exists:false}},{_id:0,name:1,mayor:1})
{ "name" : "Kerala", "mayor" : { "name" : "Jerry" } }
```

```
> db.states.find( {famousFor:'food'}, {_id:0,name:1,famousFor:1})
{ "name" : "Tamil Nadu", "famousFor" : [ "food", "honey", "forest" ] }
> db.states.find({famousFor:[$all:['food','honey','forest']]})
2020-10-31T20:16:45.521+0530 E QUERY    [thread1] SyntaxError: missing ] after element list @(shell):1:31
> db.states.find({famousFor:[$all:['food','honey','forest']]},{_id:0,name:1,famousFor:1})
2020-10-31T20:17:16.912+0530 E QUERY    [thread1] SyntaxError: missing ] after element list @(shell):1:31
> db.states.find({famousFor:{$all:['food','honey','forest']]},{_id:0,name:1,famousFor:1})
{ "name" : "Tamil Nadu", "famousFor" : [ "food", "honey", "forest" ] }
> db.states.find({famousFor:{$nin:['food','honey','forest']]},{_id:0,name:1,famousFor:1})
{ "name" : "Karnataka", "famousFor" : [ "dosa", "coffee", "beaches" ] }
{ "name" : "Kerala", "famousFor" : [ "Coconut" ] }
> db.states.find({'mayorparty':{$exists:false}},{_id:0,name:1,mayor:1})
{ "name" : "Karnataka", "mayor" : { "name" : "Tom", "party" : "D" } }
{ "name" : "Kerala", "mayor" : { "name" : "Jerry" } }
{ "name" : "Tamil Nadu", "mayor" : { "name" : "Harry", "party" : "D" } }
> db.states.find({'mayor.party':{$exists:false}},{_id:0,name:1,mayor:1})
{ "name" : "Kerala", "mayor" : { "name" : "Jerry" } }
```

Adding another collection ‘countries’:

```
> db.countries.insert({ _id:"in", name:"india", exports:{ foods:[ {name:"bacon",tasty:true}, {name:"burgers"} ]}})
WriteResult({ "nInserted" : 1 })
> show collections
countries
states
```

Usage of ‘this’ to go through every record:

```
> db.states.find("this.population>100")
{ "_id" : ObjectId("5f9d6a6027705e6cf3157926"), "name" : "Karnataka", "population" : 22200010, "lastCensus" : ISODate("2019-06-19T00:00:00Z"),
 "famousFor" : [ "dosa", "coffee", "beaches" ], "mayor" : { "name" : "Tom", "party" : "D" }, "capital" : "Bengaluru" }
{ "_id" : ObjectId("5f9d6d86ba98701c046982b4"), "name" : "Kerala", "lastCensus" : ISODate("2019-01-31T00:00:00Z"), "famousFor" : [ "Coconut" ]
, "mayor" : { "name" : "Jerry" }, "population" : 6200 }
>
```

Creating a reference and indirectly accessing a record using the reference:

```
var
karnataka=db.states.findOne({_id:ObjectId("5f9d6a6027705e6cf3157926")}
)
var karnatakaCountryRef=karnataka.country.$ref;
db.countries.findOne({_id:karnataka.country.$id})
```

```
> db.states.update({_id:ObjectId("5f9d6a6027705e6cf3157926")},{$set:{country:{$ref:"countries",$id:"in"}}})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
> db.states.find()
{ "_id" : ObjectId("5f9d6a6027705e6cf3157926"), "name" : "Karnataka", "population" : 22200010, "lastCensus" : ISODate("2019-06-19T00:00:00Z"),
 "famousFor" : [ "dosa", "coffee", "beaches" ], "mayor" : { "name" : "Tom", "party" : "D" }, "capital" : "Bengaluru", "country" : DBRef("count
ries", "in") }
{ "_id" : ObjectId("5f9d6d86ba98701c046982b4"), "name" : "Kerala", "lastCensus" : ISODate("2019-01-31T00:00:00Z"), "famousFor" : [ "Coconut" ]
, "mayor" : { "name" : "Jerry" }, "population" : 6200 }
{ "_id" : ObjectId("5f9d6df9ba98701c046982b5"), "name" : "Tamil Nadu", "poulation" : 58000, "lastCensus" : ISODate("2019-09-20T00:00:00Z"), "f
amousFor" : [ "food", "honey", "forest" ], "mayor" : { "name" : "Harry", "party" : "D" } }
>
```

```
> var karnataka=db.states.findOne({_id:ObjectId("5f9d6a6027705e6cf3157926")})
> var karnatakaCountryRef=karnataka.country.$ref;
> db.countries.findOne({_id:karnataka.country.$id})
{
        "_id" : "in",
        "name" : "india",
        "exports" : {
                "foods" : [
                        {
                                "name" : "bacon",
                                "tasty" : true
                        },
                        {
                                "name" : "burgers"
                        }
                ]
        }
}
>
```

**Deleting records:**

Deleting a record from collection:

```
db.countries.remove({"name":"United States of America"})
```

```
> db.countries.insert({ _id:"usa", name:"United States of America"})
WriteResult({ "nInserted" : 1 })
> db.countries.remove({"name":"United States of America"})
WriteResult({ "nRemoved" : 1 })
> db.countries.find()
{ "_id" : "in", "name" : "india", "exports" : { "foods" : [ { "name" : "bacon", "tasty" : true }, { "name" : "burgers" } ] } }
>
```

Deleting the whole collection:

```
db.countries.drop()
```

```
> db.countries.drop()
true
> show collections
states
>
```

**Aggregate functions:**

Count the number of records with population greater than 7000:

```
db.states.count({'population':{$gt:7000}})
```

```
> db.states.count({'population':{$gt:7000}})
2
> db.states.find()
{ "_id" : ObjectId("5f9d6a6027705e6cf3157926"), "name" : "Karnataka", "population" : 22200010, "lastCensus" : ISODate("2019-06-19T00:00:00Z"),
 "famousFor" : [ "dosa", "coffee", "beaches" ], "mayor" : { "name" : "Tom", "party" : "D" }, "capital" : "Bengaluru", "country" : DBRef("count
ries", "in") }
{ "_id" : ObjectId("5f9d6d86ba98701c046982b4"), "name" : "Kerala", "lastCensus" : ISODate("2019-01-31T00:00:00Z"), "famousFor" : [ "Coconut" ]
, "mayor" : { "name" : "Jerry" }, "population" : 6200 }
{ "_id" : ObjectId("5f9d6df9ba98701c046982b5"), "name" : "Tamil Nadu", "lastCensus" : ISODate("2019-09-20T00:00:00Z"), "famousFor" : [ "food",
 "honey", "forest" ], "mayor" : { "name" : "Harry", "party" : "D" }, "population" : 58000 }
>
```

Using distinct to find the unique values:

```
db.states.distinct('name')
```

```
> insertPlace("Kerala",6200,"2019-01-31",["Coconut"],{name:"larry"})
> db.states.find()
{ "_id" : ObjectId("5f9d6a6027705e6cf3157926"), "name" : "Karnataka", "population" : 22200010, "lastCensus" : ISODate("2019-06-19T00:00:00Z"),
 "famousFor" : [ "dosa", "coffee", "beaches" ], "mayor" : { "name" : "Tom", "party" : "D" }, "capital" : "Bengaluru", "country" : DBRef("count
ries", "in") }
{ "_id" : ObjectId("5f9d6d86ba98701c046982b4"), "name" : "Kerala", "lastCensus" : ISODate("2019-01-31T00:00:00Z"), "famousFor" : [ "Coconut" ]
, "mayor" : { "name" : "Jerry" }, "population" : 6200 }
{ "_id" : ObjectId("5f9d6df9ba98701c046982b5"), "name" : "Tamil Nadu", "lastCensus" : ISODate("2019-09-20T00:00:00Z"), "famousFor" : [ "food",
 "honey", "forest" ], "mayor" : { "name" : "Harry", "party" : "D" }, "population" : 58000 }
{ "_id" : ObjectId("5f9d8b42ba98701c046982b6"), "name" : "Kerala", "poulation" : 6200, "lastCensus" : ISODate("2019-01-31T00:00:00Z"), "famous
For" : [ "Coconut" ], "mayor" : { "name" : "larry" } }
> db.states.distinct('name')
[ "Karnataka", "Kerala", "Tamil Nadu" ]
>
```

Using aggregate, group, sum and count to find the total population of each states and also its count, grouping by state name:

```
db.states.aggregate([{$group:{_id:{name:"$name"},totalpopulation:{$sum:"$population"},count:{$sum:1}}}])
```

## Create a table in HBase or any columnar database and execute basic operations – create, insert, update, delete specific revisions

### Starting HBase:

```
./hbase shell
```

```
shaaz@PES1201801754:~/hbase/bin$./start-hbase.sh
localhost: running zookeeper, logging to /home/shaaz/hbase/bin/../logs/hbase-shaaz-zookeeper-shaaz.out
running master, logging to /home/shaaz/hbase/logs/hbase-shaaz-master-shaaz.out
: running regionserver, logging to /home/shaaz/hbase/logs/hbase-shaaz-regionserver-shaaz.out
: OpenJDK 64-Bit Server VM warning: ignoring option PermSize=128m; support was removed in 8.0
: OpenJDK 64-Bit Server VM warning: ignoring option MaxPermSize=128m; support was removed in 8.0
shaaz@PES1201801754:~/hbase/bin$jps
3905 HMaster
3844 HQuorumPeer
3156 NodeManager
3976 HRegionServer
3002 ResourceManager
2813 SecondaryNameNode
2446 NameNode
2606 DataNode
4015 Jps
shaaz@PES1201801754:~/hbase/bin$./hbase shell
```

### Create a table:

Create a table ‘test’ with column family ‘cf’:

```
create 'test', 'cf'
```

```
shaaz@PES1201801754:~/hbase/bin$./hbase shell
2020-11-06 21:22:48,196 WARN  [main] util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java class
es where applicable
HBase Shell
Use "help" to get list of supported commands.
Use "exit" to quit this interactive shell.
Version 1.4.13, r38bf65a22b7e9320f07aeb27677e4533b9a77ef4, Sun Feb 23 02:06:36 PST 2020

hbase(main):001:0> create 'test', 'cf'
0 row(s) in 20.2640 seconds

=> Hbase::Table - test
hbase(main):002:0> list
TABLE
test
1 row(s) in 0.2050 seconds

=> ["test"]
hbase(main):003:0> describe 'test'
Table test is ENABLED
test
COLUMN FAMILIES DESCRIPTION
{NAME => 'cf', BLOOMFILTER => 'ROW', VERSIONS => '1', IN_MEMORY => 'false', KEEP_DELETED_CELLS => 'FALSE', DATA_BLOCK_ENCODING => 'NONE', TTL
=> 'FOREVER', COMPRESSION => 'NONE', MIN_VERSIONS => '0', BLOCKCACHE => 'true', BLOCKSIZE => '65536', REPLICATION_SCOPE => '0'}
1 row(s) in 0.5100 seconds
```

### Inserting records into the table:

```
put 'test', 'row1', 'cf:a', 'value1'
```

```
hbase(main):004:0> put 'test', 'row1', 'cf:a', 'value1'
0 row(s) in 0.3380 seconds

hbase(main):005:0> put 'test', 'row2', 'cf:b', 'value2'
0 row(s) in 0.0070 seconds
```

Get the number of records using ‘count’ and retrieve all records using ‘scan’, Specific record is retrieved using ‘get’:

```
count 'test'
scan 'test'
get 'test', 'row2'
```

```
hbase(main):006:0> put 'test', 'row3', 'cf:c', 'value3'
0 row(s) in 0.0720 seconds

hbase(main):007:0> count 'test'
3 row(s) in 0.1380 seconds

=> 3
hbase(main):008:0> scan 'test'
ROW                                   COLUMN+CELL
 row1                                 column=cf:a, timestamp=1604678231433, value=value1
 row2                                 column=cf:b, timestamp=1604678255597, value=value2
 row3                                 column=cf:c, timestamp=1604678379015, value=value3
3 row(s) in 0.0770 seconds

hbase(main):009:0> get 'test', 'row2'
COLUMN                                CELL
 cf:b                                 timestamp=1604678255597, value=value2
1 row(s) in 0.6580 seconds
```

**Update table:**

```
alter 'test', {NAME => 'cf', VERSIONS => 3}
```

```
hbase(main):010:0> disable 'test'
0 row(s) in 4.7350 seconds

hbase(main):011:0> alter 'test', {NAME => 'cf', VERSIONS => 3}
Updating all regions with the new schema...
1/1 regions updated.
Done.
0 row(s) in 2.3920 seconds

hbase(main):012:0> enable 'test'
0 row(s) in 1.5880 seconds
```

```
put 'test', 'row3', 'cf:c', 'value300'
```

```
hbase(main):013:0> put 'test', 'row3', 'cf:c', 'value300'
0 row(s) in 0.0470 seconds

hbase(main):014:0> scan 'test'
ROW                                   COLUMN+CELL
 row1                                 column=cf:a, timestamp=1604678231433, value=value1
 row2                                 column=cf:b, timestamp=1604678255597, value=value2
 row3                                 column=cf:c, timestamp=1604678717504, value=value300
3 row(s) in 0.0710 seconds

hbase(main):015:0> get 'test', 'row3', {COLUMN => 'cf:c', VERSIONS => 3}
COLUMN                                CELL
 cf:c                                 timestamp=1604678717504, value=value300
 cf:c                                 timestamp=1604678379015, value=value3
1 row(s) in 0.0970 seconds
```

Inserting and updating more records:

A row can have more than one column family

```
=> ["test"]
hbase(main):032:0> put 'test', 'row1', 'cf:a', 'value1'
0 row(s) in 0.0510 seconds

hbase(main):033:0> put 'test', 'row2', 'cf:b', 'value2'
0 row(s) in 0.0370 seconds

hbase(main):034:0> put 'test', 'row3', 'cf:c', 'value3'
0 row(s) in 0.0330 seconds

hbase(main):035:0> put 'test', 'row2', 'cf:b', 'value4'
0 row(s) in 0.0150 seconds

hbase(main):036:0> scan 'test'
ROW                                   COLUMN+CELL
 row1                                 column=cf:a, timestamp=1604679686620, value=value1
 row2                                 column=cf:b, timestamp=1604679728680, value=value4
 row3                                 column=cf:c, timestamp=1604679704261, value=value3
3 row(s) in 0.0150 seconds

hbase(main):037:0> put 'test', 'row2', 'cf:d', 'value2'
0 row(s) in 0.0370 seconds

hbase(main):038:0> scan 'test'
ROW                                   COLUMN+CELL
 row1                                 column=cf:a, timestamp=1604679686620, value=value1
 row2                                 column=cf:b, timestamp=1604679728680, value=value4
 row2                                 column=cf:d, timestamp=1604679753658, value=value2
 row3                                 column=cf:c, timestamp=1604679704261, value=value3
3 row(s) in 0.0390 seconds
```

```
hbase(main):042:0> get 'test', 'row2'
COLUMN                                CELL
 cf:b                                 timestamp=1604679728680, value=value4
 cf:d                                 timestamp=1604679753658, value=value2
1 row(s) in 0.0480 seconds

hbase(main):043:0> put 'test', 'row2', 'cf:a', 'value2'
0 row(s) in 0.0520 seconds

hbase(main):044:0> scan 'test'
ROW                                   COLUMN+CELL
 row1                                 column=cf:a, timestamp=1604679686620, value=value1
 row2                                 column=cf:a, timestamp=1604680046583, value=value2
 row2                                 column=cf:b, timestamp=1604679728680, value=value4
 row2                                 column=cf:d, timestamp=1604679753658, value=value2
 row3                                 column=cf:c, timestamp=1604679704261, value=value3
3 row(s) in 0.0680 seconds
```

```
hbase(main):044:0> scan 'test'
ROW                                   COLUMN+CELL
 row1                                 column=cf:a, timestamp=1604679686620, value=value1
 row2                                 column=cf:a, timestamp=1604680046583, value=value2
 row2                                 column=cf:b, timestamp=1604679728680, value=value4
 row2                                 column=cf:d, timestamp=1604679753658, value=value2
 row3                                 column=cf:c, timestamp=1604679704261, value=value3
3 row(s) in 0.0680 seconds

hbase(main):045:0> deleteall 'test', 'row3'
0 row(s) in 0.0130 seconds

hbase(main):046:0> delete 'test', 'row2', 'cf:d'
0 row(s) in 0.0250 seconds

hbase(main):047:0> delete 'test', 'row2', 'cf:a'
0 row(s) in 0.0330 seconds

hbase(main):048:0> scan 'test'
ROW                                   COLUMN+CELL
 row1                                 column=cf:a, timestamp=1604679686620, value=value1
 row2                                 column=cf:b, timestamp=1604679728680, value=value4
2 row(s) in 0.0210 seconds

hbase(main):049:0> put 'test', 'row2', 'cf:b', 'value2'
0 row(s) in 0.0110 seconds

hbase(main):050:0> scan 'test'
ROW                                   COLUMN+CELL
 row1                                 column=cf:a, timestamp=1604679686620, value=value1
 row2                                 column=cf:b, timestamp=1604680388867, value=value2
2 row(s) in 0.0160 seconds
```

**Deleting records and table:**

Deleting specific record of a particular timestamp:

```
delete 'test', 'row3', 'cf:c', 1604678717504
```

```
hbase(main):013:0> put 'test', 'row3', 'cf:c', 'value300'
0 row(s) in 0.0470 seconds

hbase(main):014:0> scan 'test'
ROW                                   COLUMN+CELL
 row1                                 column=cf:a, timestamp=1604678231433, value=value1
 row2                                 column=cf:b, timestamp=1604678255597, value=value2
 row3                                 column=cf:c, timestamp=1604678717504, value=value300
3 row(s) in 0.0710 seconds

hbase(main):015:0> get 'test', 'row3', {COLUMN => 'cf:c', VERSIONS => 3}
COLUMN                                CELL
 cf:c                                 timestamp=1604678717504, value=value300
 cf:c                                 timestamp=1604678379015, value=value3
1 row(s) in 0.0970 seconds

hbase(main):016:0> delete 'test', 'row3', 'cf:c', 1604678717504
0 row(s) in 0.1880 seconds

hbase(main):017:0> scan 'test'
ROW                                   COLUMN+CELL
 row1                                 column=cf:a, timestamp=1604678231433, value=value1
 row2                                 column=cf:b, timestamp=1604678255597, value=value2
 row3                                 column=cf:c, timestamp=1604678379015, value=value3
3 row(s) in 0.5910 seconds
```

Deleting all versions of particular record:

```
deleteall 'test', 'row3'
```

```
hbase(main):023:0> deleteall 'test', 'row3'
0 row(s) in 0.0090 seconds

hbase(main):024:0> scan 'test'
ROW                                   COLUMN+CELL
 row1                                 column=cf:a, timestamp=1604678231433, value=value1
 row2                                 column=cf:b, timestamp=1604678255597, value=value2
2 row(s) in 0.0420 seconds
```

Deleting all records:

```
truncate 'test'
```

```
hbase(main):025:0> truncate 'test'
Truncating 'test' table (it may take a while):
 - Disabling table...
 - Truncating table...
0 row(s) in 6.2210 seconds

hbase(main):026:0> scan 'test'
ROW                                   COLUMN+CELL
0 row(s) in 0.4990 seconds
```

Deleting the table:

```
drop 'test'
```

```
hbase(main):027:0> disable 'test'
0 row(s) in 2.3150 seconds

hbase(main):028:0> drop 'test'
0 row(s) in 1.4630 seconds

hbase(main):029:0> list
TABLE
0 row(s) in 0.0120 seconds

=> []
hbase(main):030:0>
```

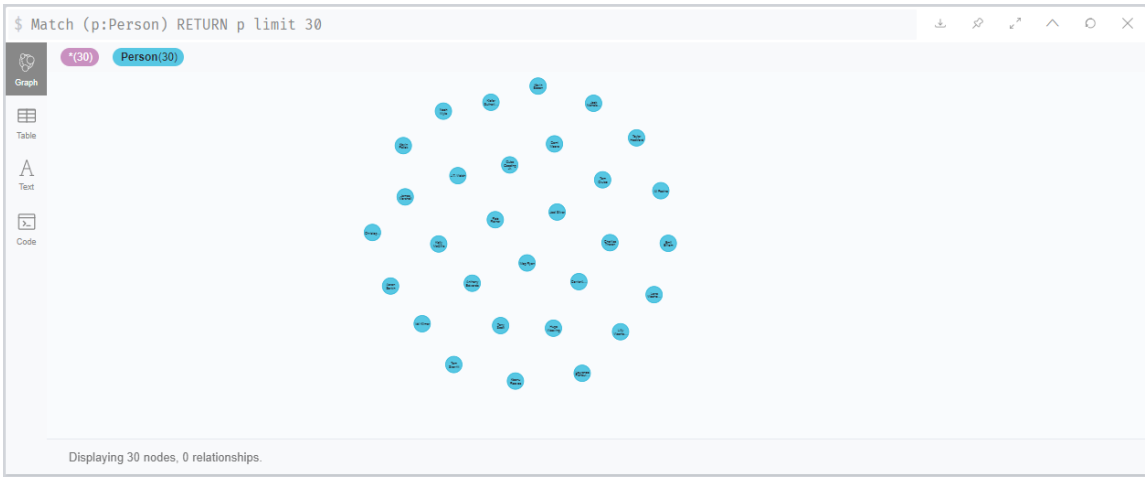
## Use movies database in Neo4J sandbox or any graph database and execute operations to insert a node, establish a relationship and retrieve related attributes

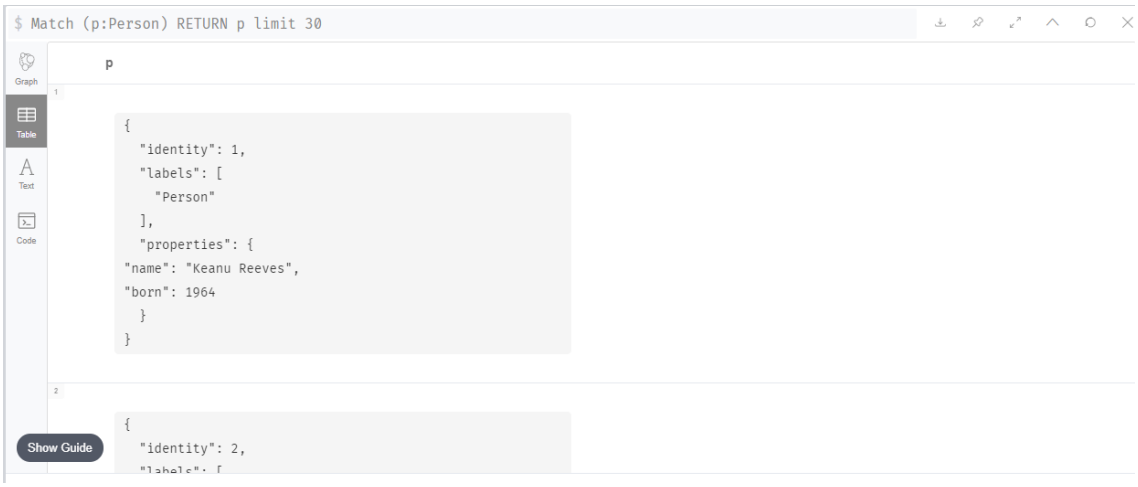
**Retrieving records:**

Retrieve is done using ‘MATCH’
Retrieve those movies that were released after year 1000 and limit it to 3:

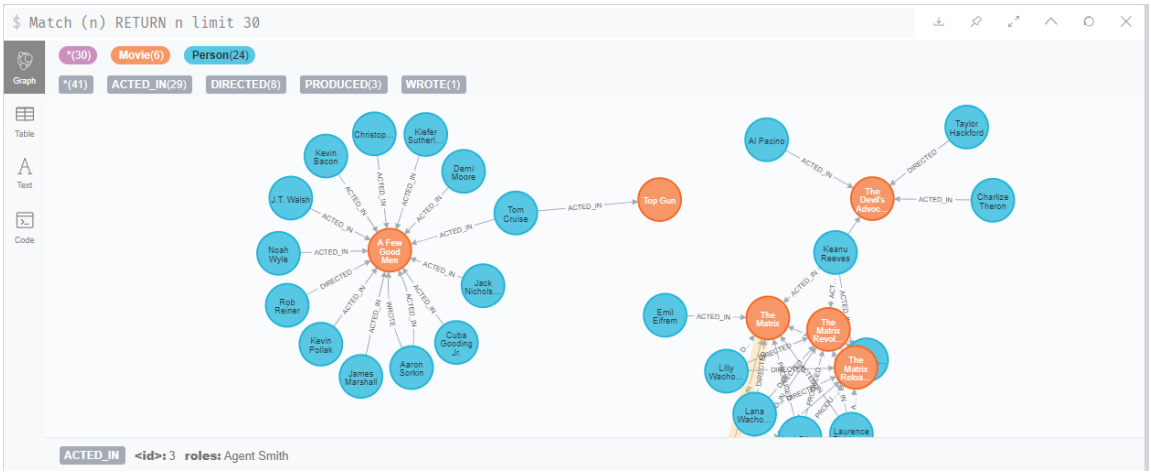
```
Match (m:Movie) where m.released > 1000 RETURN m limit 3
```

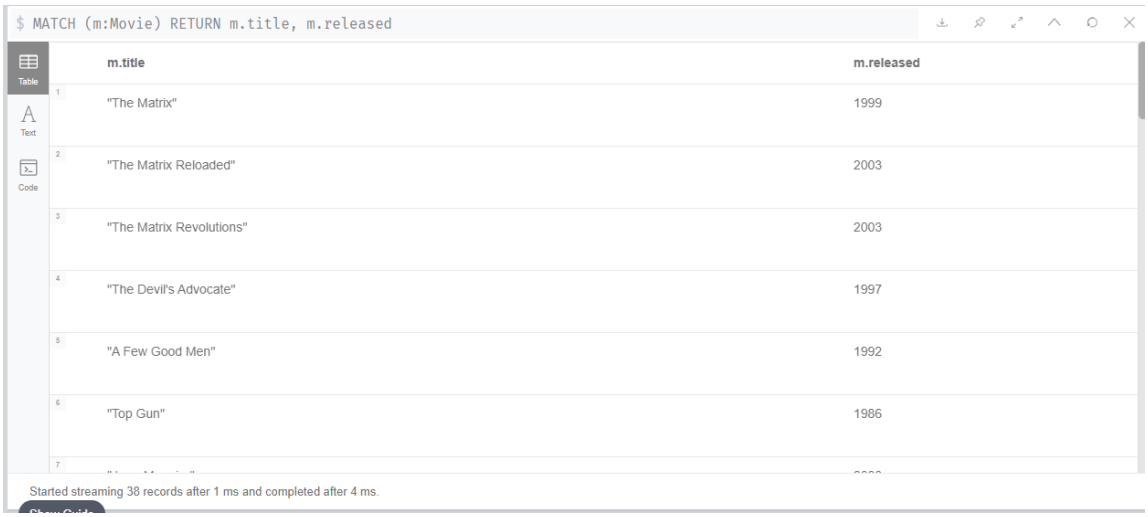

Some more retrieval queries:




$ Match (p:Person) RETURN p limit 30
Graph
Table
Text
Code
p
1
{
"identity": 1,
"labels": [
"Person"
],
"properties": {
"name": "Keanu Reeves",
"born": 1964
}
}
2
{
"identity": 2,
Show Guide

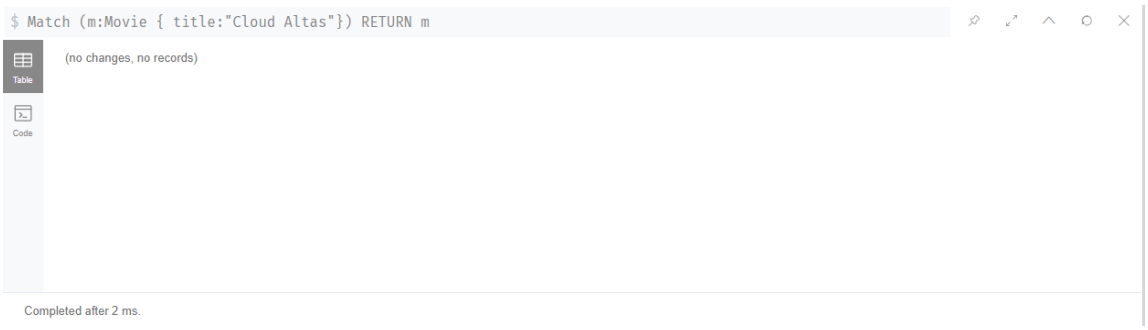

$ Match (p:Person) RETURN p limit 30
Graph
Table
Text
Code
"p"
{"name":"Keanu Reeves","born":1964}
{"name":"Carrie-Anne Moss","born":1967}
{"name":"Laurence Fishburne","born":1961}
{"name":"Hugo Weaving","born":1960}
{"name":"Lilly Wachowski","born":1967}
{"name":"Lana Wachowski","born":1965}
{"name":"Joel Silver","born":1952}
{"name":"Emil Eifrem","born":1978}
{"name":"Charlize Theron","born":1975}
{"name":"Al Pacino","born":1940}
{"name":"Taylor Hackford","born":1944}
MAX COLUMN WIDTH:


$ Match (n) RETURN n limit 30
Graph
Table
Text
Code
*(30)
Movie(6)
Person(24)
*(41)
ACTED_IN(29)
DIRECTED(8)
PRODUCED(3)
WROTE(1)
Kevin Bacon
Christop...
Kiefer Sutherl...
Demi Moore
J.T. Walsh
Tom Cruise
Top Gun
Noah Wyle
A Few Good Men
Jack Nichols...
Rob Reiner
Kevin Pollak
James Marshall
Aaron Sorkin
Cuba Gooding Jr.
Al Pacino
Taylor Hackford
The Devil's Advoc...
Charlize Theron
Keanu Reeves
Emil Eifrem
The Matrix
The Matrix Revol...
The Matrix Reloa...
Lilly Wacho...
Lana Wacho...
Laurence
ACTED_IN
DIRECTED
WROTE
ACTED_IN <id>: 3 roles: Agent Smith




Below two examples show that the record can be retrieved in two ways:


If a record is not found:




Specifying more than one condition using ‘and’:

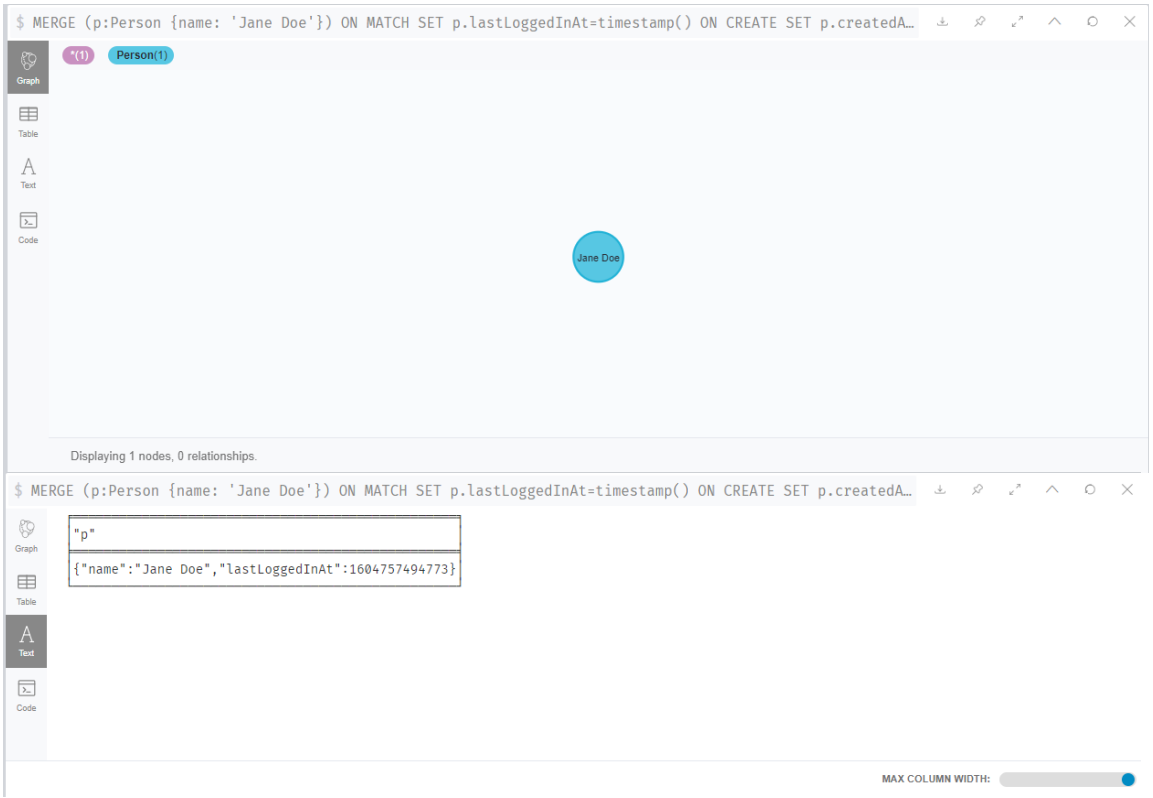
```
Match (m:Movie) where m.released > 2010 and m.released <2015 RETURN m
```


**Creating new instance of a node:**

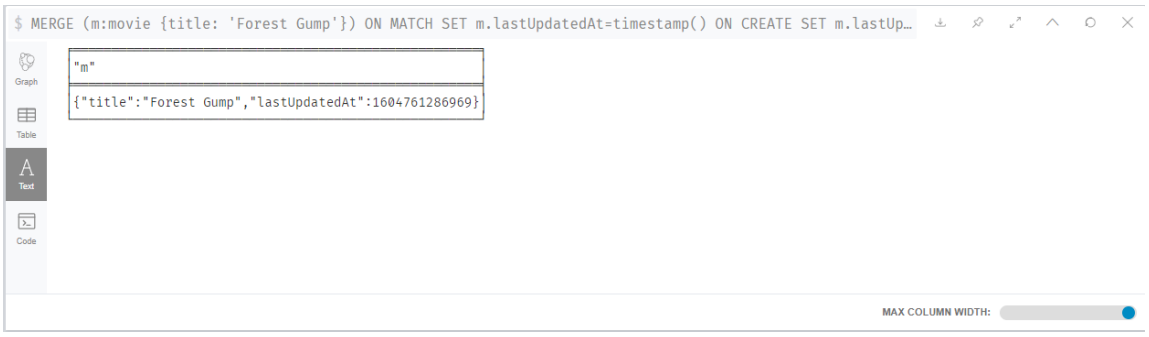
```
Create (p:Person {name: 'Jane Doe'}) RETURN p
```


**Updating metadata to the newly created node:**

```
MERGE (p:Person {name: 'Jane Doe'}) ON MATCH SET
p.lastLoggedInAt=timestamp() ON CREATE SET p.createdAt=timestamp()
Return p
```



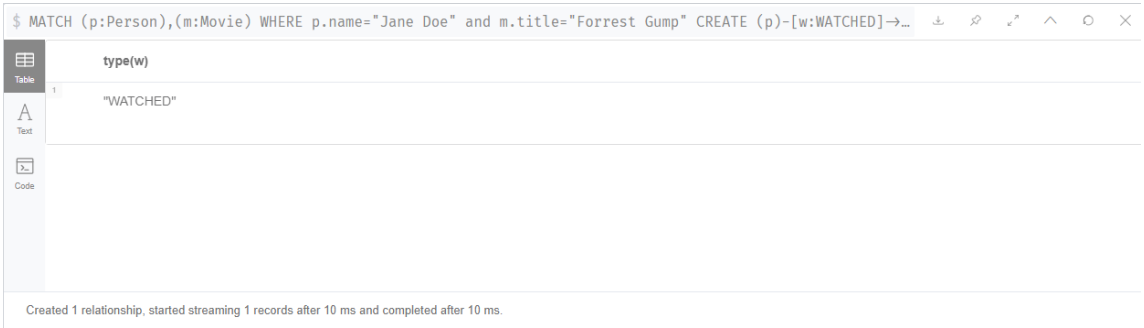
Create a node ‘Forrest Gump’ and update values:


**Establishing and creating a new relation:**

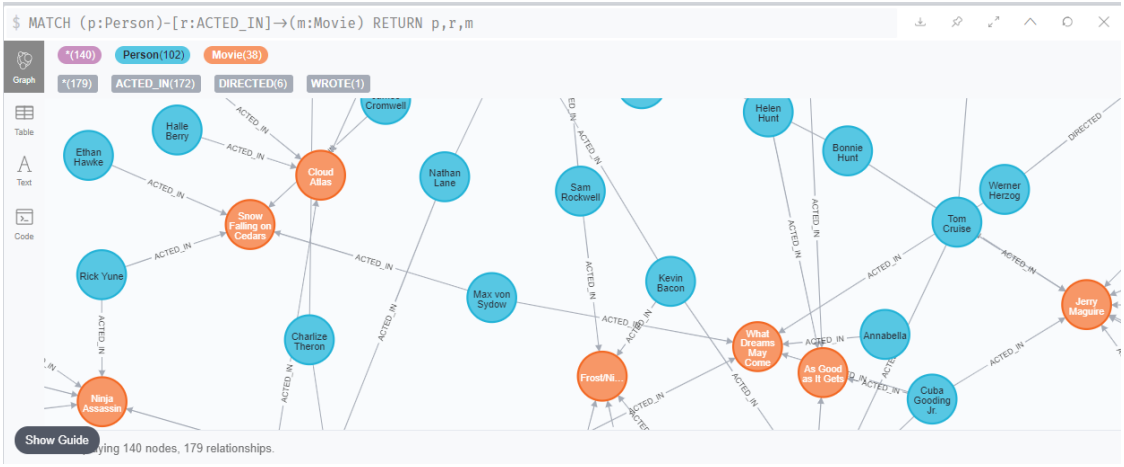
Establishing a relation ‘WATCHED’ between jane doe and forrest gump:

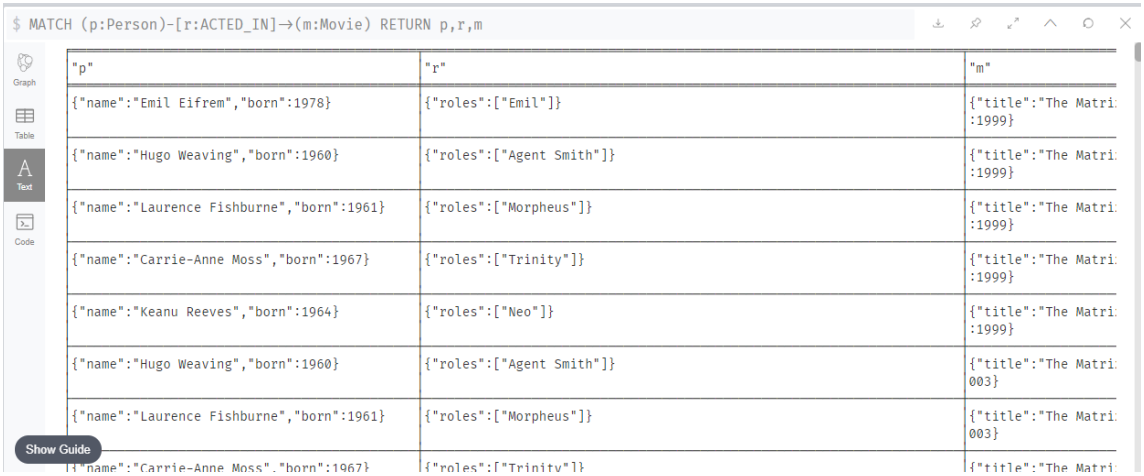
```
MATCH (p:Person),(m:Movie) WHERE p.name=\"Jane Doe\" and
m.title=\"Forrest Gump\" CREATE (p)-[w:WATCHED]->(m) RETURN type(w)
```



**Retrieving records between two nodes with the specified relation:**

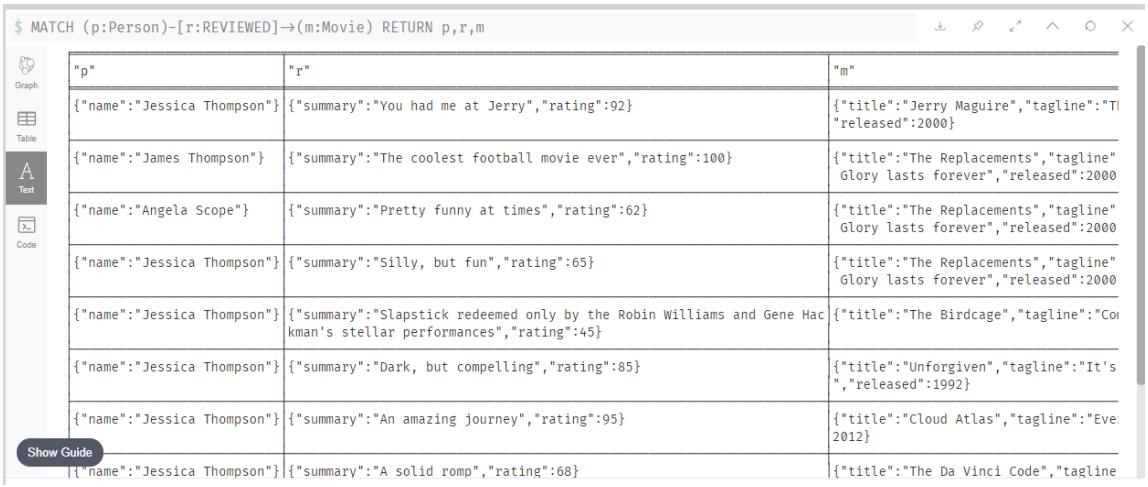
```
MATCH (p:Person)-[r:ACTED_IN]->(m:Movie) RETURN p,r,m
```

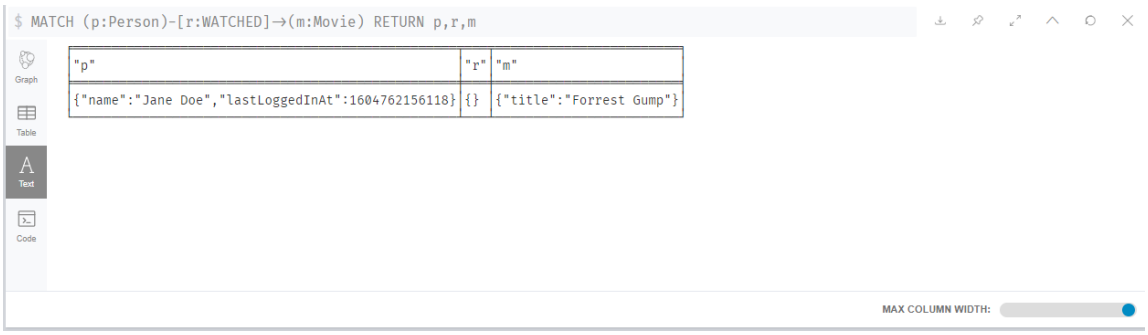


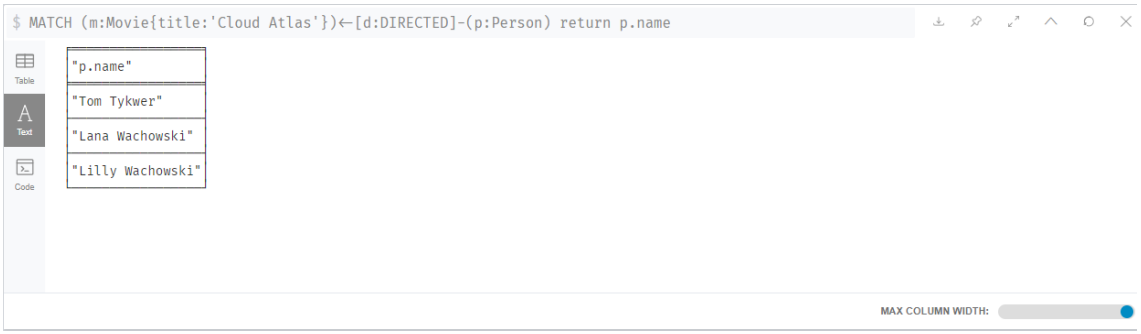

Retrieving some more records between two nodes:

People who have reviewed a particular movie:

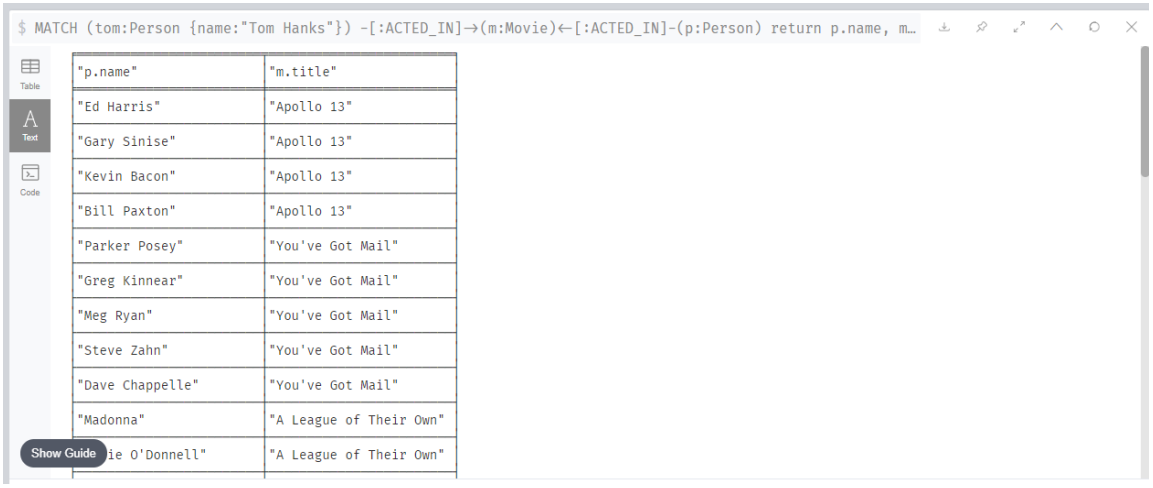




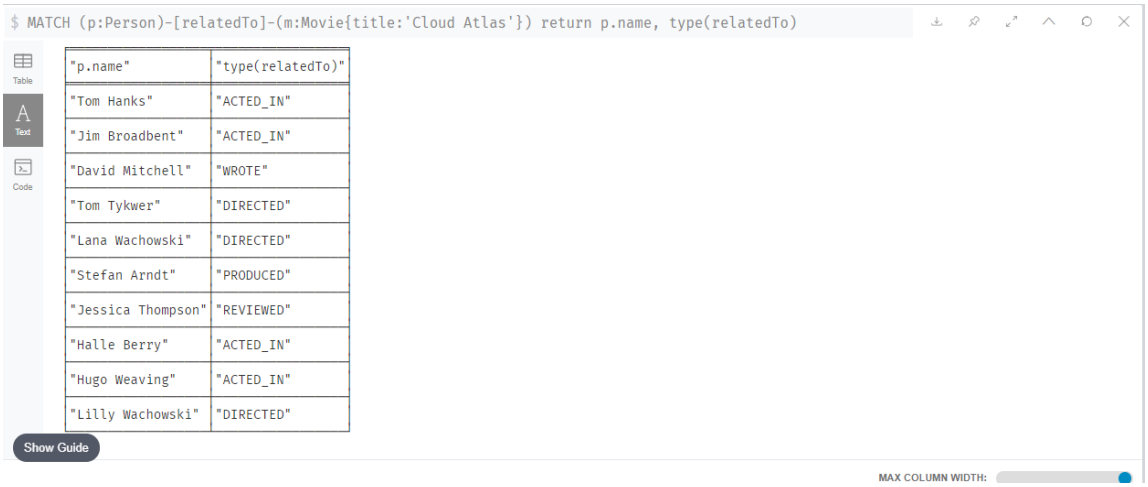

Retrieving the name of the person who directed ‘cloud atlas’:




Retrieving all movies where tom hanks has acted:



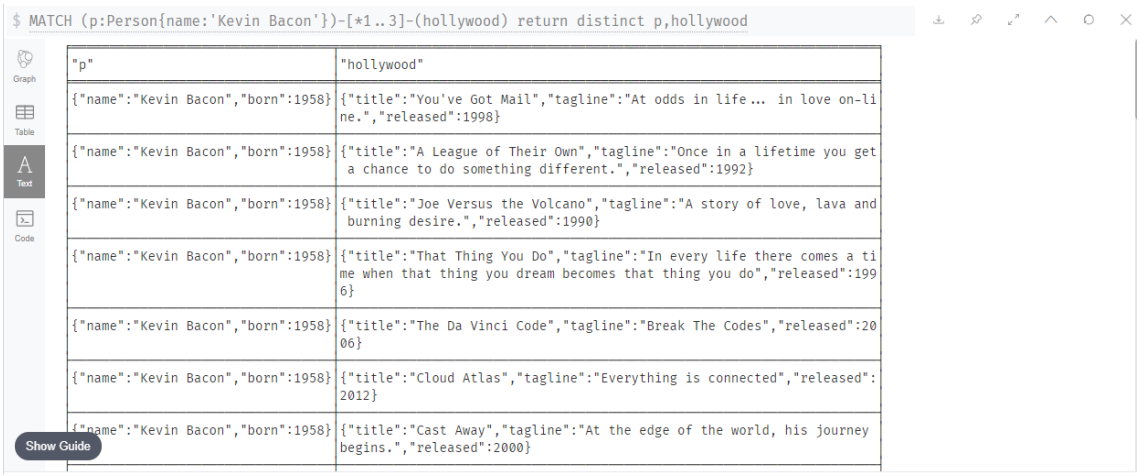
Retrieve all names associated with movie ‘cloud atlas’:




| "p.name" | "type(relatedTo)" |
|---|---|
| "Tom Hanks" | "ACTED_IN" |
| "Jim Broadbent" | "ACTED_IN" |
| "David Mitchell" | "WROTE" |
| "Tom Tykwer" | "DIRECTED" |
| "Lana Wachowski" | "DIRECTED" |
| "Stefan Arndt" | "PRODUCED" |
| "Jessica Thompson" | "REVIEWED" |
| "Halle Berry" | "ACTED_IN" |
| "Hugo Weaving" | "ACTED_IN" |
| "Lilly Wachowski" | "DIRECTED" |

Retrieve records of the people connected to ‘kevin bacon’ upto ‘3’ hops:

```
MATCH (p:Person{name:'Kevin Bacon'})-[*1..3]-(hollywood) return distinct p,Hollywood
```

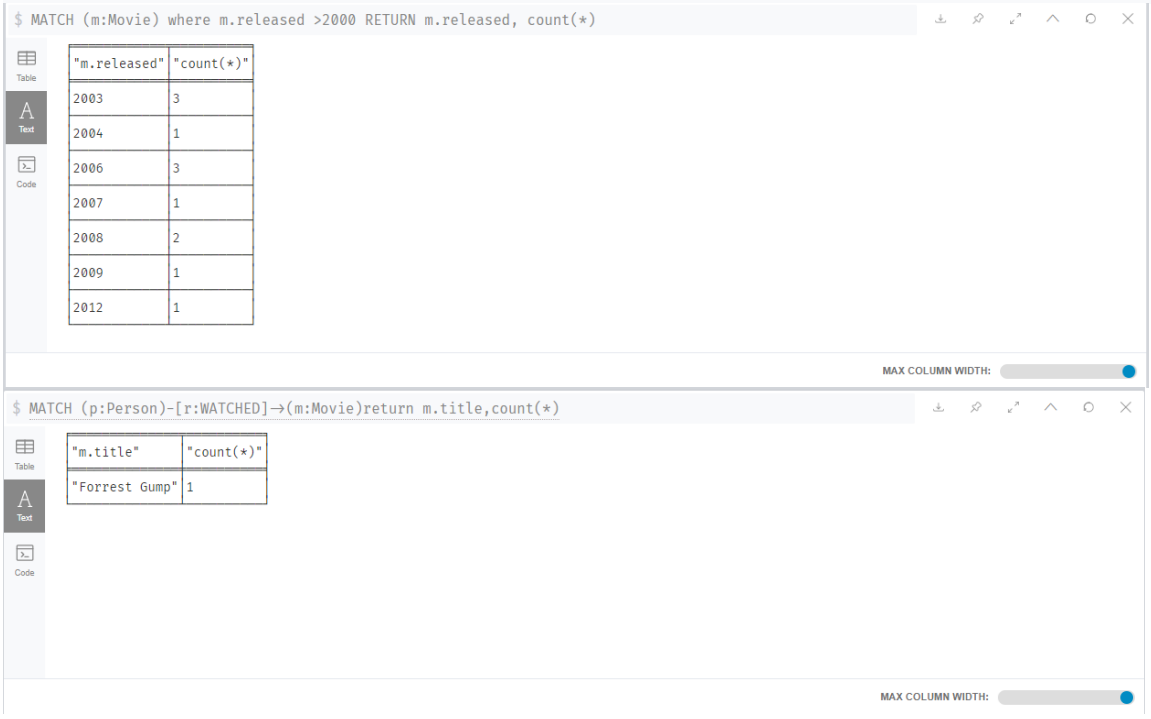



| "p" | "hollywood" |
|---|---|
| {"name":"Kevin Bacon","born":1958} | {"title":"You've Got Mail","tagline":"At odds in life... in love on-line.","released":1998} |
| {"name":"Kevin Bacon","born":1958} | {"title":"A League of Their Own","tagline":"Once in a lifetime you get a chance to do something different.","released":1992} |
| {"name":"Kevin Bacon","born":1958} | {"title":"Joe Versus the Volcano","tagline":"A story of love, lava and burning desire.","released":1990} |
| {"name":"Kevin Bacon","born":1958} | {"title":"That Thing You Do","tagline":"In every life there comes a time when that thing you dream becomes that thing you do","released":1996} |
| {"name":"Kevin Bacon","born":1958} | {"title":"The Da Vinci Code","tagline":"Break The Codes","released":2006} |
| {"name":"Kevin Bacon","born":1958} | {"title":"Cloud Atlas","tagline":"Everything is connected","released":2012} |
| {"name":"Kevin Bacon","born":1958} | {"title":"Cast Away","tagline":"At the edge of the world, his journey begins.","released":2000} |

**Aggregation:**

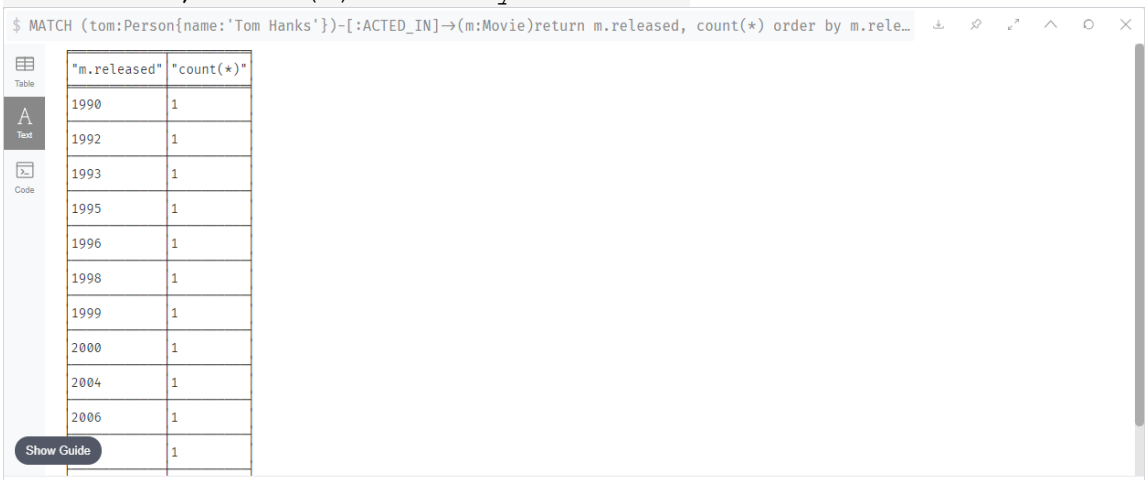
Count of movies released after 2000:

```
MATCH (m:Movie) where m.released >2000 RETURN m.released, count(*)
```



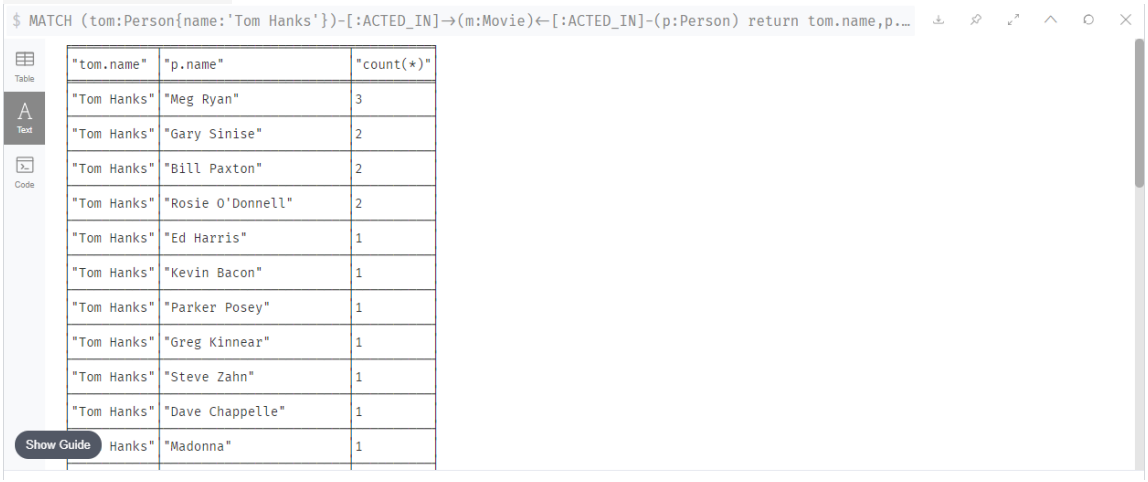

Count of movies Tom Hank acted in and order by count in a particular year:

```
MATCH (tom:Person{name:'Tom Hanks'})-[:ACTED_IN]->(m:Movie)return
m.released, count(*) order by m.release
```



Ordering the records in descending order:

```
MATCH (tom:Person{name:'Tom Hanks'})-[:ACTED_IN]->(m:Movie)<-
[:ACTED_IN]-(p:Person) return tom.name,p.name,count(*) order by
count(*) desc
```



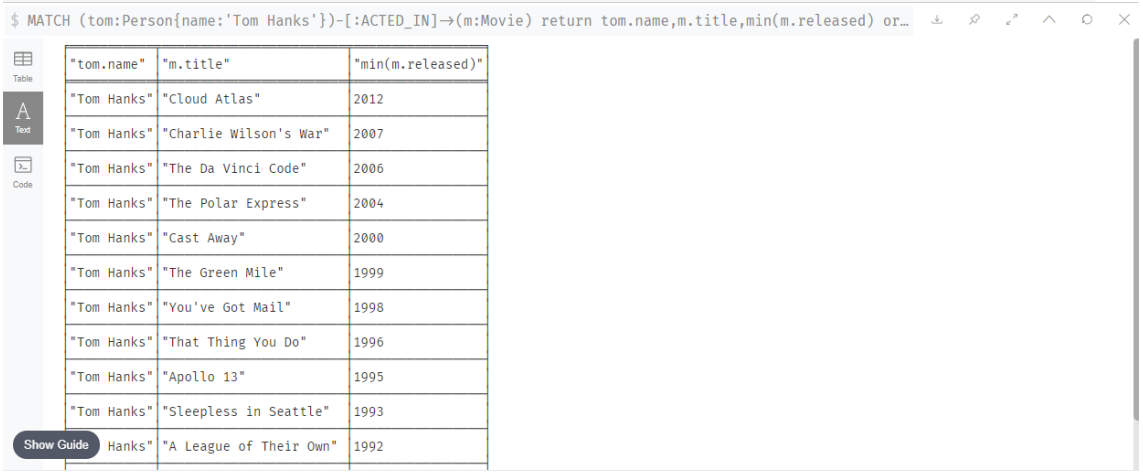
```
$ MATCH (tom:Person{name:'Tom Hanks'})-[:ACTED_IN]→(m:Movie)←[:ACTED_IN]-(p:Person) return tom.name,p.…
```

| "tom.name" | "p.name" | "count(*)" |
|---|---|---|
| "Tom Hanks" | "Meg Ryan" | 3 |
| "Tom Hanks" | "Gary Sinise" | 2 |
| "Tom Hanks" | "Bill Paxton" | 2 |
| "Tom Hanks" | "Rosie O'Donnell" | 2 |
| "Tom Hanks" | "Ed Harris" | 1 |
| "Tom Hanks" | "Kevin Bacon" | 1 |
| "Tom Hanks" | "Parker Posey" | 1 |
| "Tom Hanks" | "Greg Kinnear" | 1 |
| "Tom Hanks" | "Steve Zahn" | 1 |
| "Tom Hanks" | "Dave Chappelle" | 1 |
| Hanks" | "Madonna" | 1 |

Table Text Code Show Guide

Using the aggregation function min:

```
MATCH (tom:Person{name:'Tom Hanks'})-[:ACTED_IN]->(m:Movie) return
tom.name,m.title,min(m.released) order by min(m.released) desc
```



```
$ MATCH (tom:Person{name:'Tom Hanks'})-[:ACTED_IN]→(m:Movie) return tom.name,m.title,min(m.released) or…
```

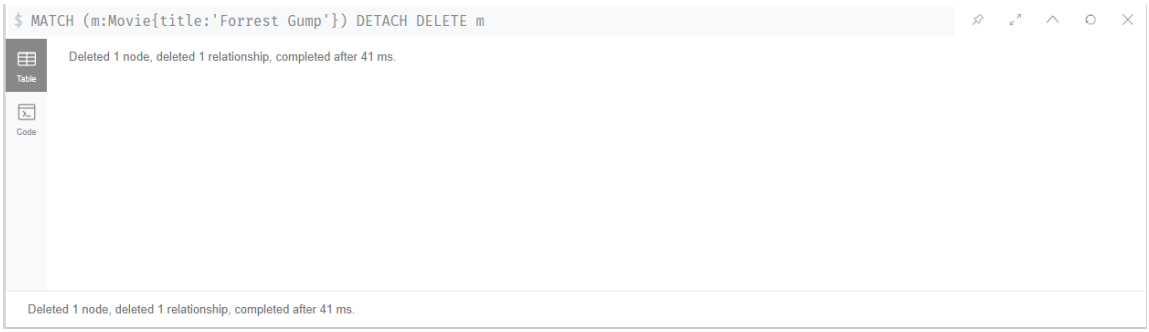
| "tom.name" | "m.title" | "min(m.released)" |
|---|---|---|
| "Tom Hanks" | "Cloud Atlas" | 2012 |
| "Tom Hanks" | "Charlie Wilson's War" | 2007 |
| "Tom Hanks" | "The Da Vinci Code" | 2006 |
| "Tom Hanks" | "The Polar Express" | 2004 |
| "Tom Hanks" | "Cast Away" | 2000 |
| "Tom Hanks" | "The Green Mile" | 1999 |
| "Tom Hanks" | "You've Got Mail" | 1998 |
| "Tom Hanks" | "That Thing You Do" | 1996 |
| "Tom Hanks" | "Apollo 13" | 1995 |
| "Tom Hanks" | "Sleepless in Seattle" | 1993 |
| Hanks" | "A League of Their Own" | 1992 |

Table Text Code Show Guide

**Deleting a node:**

Deleting a node and its relations:

```
MATCH (m:Movie{title:'Forrest Gump'}) DETACH DELETE m
```



# MongoDb-Commands Executed

```
sudo apt-get install mongodb
sudo service mongodb status
sudo service mongodb start
show dbs;

use dbt;
db

db.states.insert({
name:"Karnataka",
population:22200000,
lastCensus:ISODate("2019-06-19"),
famousFor:["dosa","coffee","beaches"],
mayor:{
name:"Tom",
party:"D"
}
})

show collections
db.states.find()
db.states.find().count()

db.help()
db.states.help()
typeof db.states
typeof db.states.insert

function insertPlace(
name,population,lastCensus,
famousFor,mayorInfo
){
db.states.insert({
name:name,
poulation:population,
lastCensus:ISODate(lastCensus),
famousFor:famousFor,
mayor:mayorInfo
});
}
insertPlace("Kerala",6200,"2019-01-31",["Coconut"],{name:"Jerry"})
insertPlace("Tamil Nadu",58000,"2019-09-20",["food","honey","forest"],{name:"Harry",party:"D"})

> db.states.find().count()
3
```

```
> db.states.find()
{ "_id" : ObjectId("5f9d6a6027705e6cf3157926"), "name" : "Karnataka", "population" : 22200000,
"lastCensus" : ISODate("2019-06-19T00:00:00Z"), "famousFor" : [ "dosa", "coffee", "beaches" ], "mayor"
: { "name" : "Tom", "party" : "D" } }
{ "_id" : ObjectId("5f9d6d86ba98701c046982b4"), "name" : "Kerala", "poulation" : 6200, "lastCensus" :
ISODate("2019-01-31T00:00:00Z"), "famousFor" : [ "Coconut" ], "mayor" : { "name" : "Jerry" } }
{ "_id" : ObjectId("5f9d6df9ba98701c046982b5"), "name" : "Tamil Nadu", "poulation" : 58000,
"lastCensus" : ISODate("2019-09-20T00:00:00Z"), "famousFor" : [ "food", "honey", "forest" ], "mayor" : {
"name" : "Harry", "party" : "D" } }

> db.states.find({"_id":ObjectId("5f9d6a6027705e6cf3157926")})
{ "_id" : ObjectId("5f9d6a6027705e6cf3157926"), "name" : "Karnataka", "population" : 22200000,
"lastCensus" : ISODate("2019-06-19T00:00:00Z"), "famousFor" : [ "dosa", "coffee", "beaches" ], "mayor"
: { "name" : "Tom", "party" : "D" } }

> db.states.find({"_id":ObjectId("5f9d6a6027705e6cf3157926")},{name:1})
{ "_id" : ObjectId("5f9d6a6027705e6cf3157926"), "name" : "Karnataka" }

> db.states.find({"_id":ObjectId("5f9d6a6027705e6cf3157926")},{name:0})
{ "_id" : ObjectId("5f9d6a6027705e6cf3157926"), "population" : 22200000, "lastCensus" :
ISODate("2019-06-19T00:00:00Z"), "famousFor" : [ "dosa", "coffee", "beaches" ], "mayor" : { "name" :
"Tom", "party" : "D" } }

> db.states.find(
... {name:/^K/,population:{$lt:60000000}},
... {_id:0,name:1,population:1})
{ "name" : "Karnataka", "population" : 22200000 }

> db.states.find( {name:/^K/,population:{$gt:60000}}, {_id:0,name:1,population:1})
{ "name" : "Karnataka", "population" : 22200000 }

> db.states.update( { _id: ObjectId("5f9d6d86ba98701c046982b4") }, { $rename: { 'poulation':
'population' } } )
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })

> db.states.find( {name:/^K/,population:{$lt:60000000}}, {_id:0,name:1,population:1})
{ "name" : "Karnataka", "population" : 22200000 }
{ "name" : "Kerala", "population" : 6200 }

> var population_range={
... $lt:1000000000000,
... $gt:1000
... }

> db.states.find( {name:/^P/,population:population_range}, {name:1} )

> db.states.find( {name:/^K/,population:population_range}, {name:1} )
{ "_id" : ObjectId("5f9d6a6027705e6cf3157926"), "name" : "Karnataka" }
```

```
{ "_id" : ObjectId("5f9d6d86ba98701c046982b4"), "name" : "Kerala" }


> db.states.find( {famousFor:'food'}, {_id:0,name:1,famousFor:1})
{ "name" : "Tamil Nadu", "famousFor" : [ "food", "honey", "forest" ] }

> db.states.find({famousFor:{$all:['food','honey','forest']}},{_id:0,name:1,famousFor:1})
{ "name" : "Tamil Nadu", "famousFor" : [ "food", "honey", "forest" ] }

> db.states.find({famousFor:{$nin:['food','honey','forest']}},{_id:0,name:1,famousFor:1})
{ "name" : "Karnataka", "famousFor" : [ "dosa", "coffee", "beaches" ] }
{ "name" : "Kerala", "famousFor" : [ "Coconut" ] }

> db.states.find({'mayorparty':{$exists:false}},{_id:0,name:1,mayor:1})
{ "name" : "Karnataka", "mayor" : { "name" : "Tom", "party" : "D" } }
{ "name" : "Kerala", "mayor" : { "name" : "Jerry" } }
{ "name" : "Tamil Nadu", "mayor" : { "name" : "Harry", "party" : "D" } }
> db.states.find({'mayor.party':{$exists:false}},{_id:0,name:1,mayor:1})
{ "name" : "Kerala", "mayor" : { "name" : "Jerry" } }

> db.states.aggregate([{$group : {_id:ObjectId("5f9d6d86ba98701c046982b4") }}])
{ "_id" : ObjectId("5f9d6d86ba98701c046982b4") }

> db.countries.insert({ _id:"in", name:"india", exports:{ foods:[ {name:"bacon",tasty:true},
{name:"burgers"} ]}})
WriteResult({ "nInserted" : 1 })

> show collections
countries
states

> db.states.update( { _id: ObjectId("5f9d6a6027705e6cf3157926") }, { $set: { 'capital': 'Bengaluru' } } )
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })

> db.states.find()
{ "_id" : ObjectId("5f9d6a6027705e6cf3157926"), "name" : "Karnataka", "population" : 22200000,
"lastCensus" : ISODate("2019-06-19T00:00:00Z"), "famousFor" : [ "dosa", "coffee", "beaches" ], "mayor"
: { "name" : "Tom", "party" : "D" }, "capital" : "Bengaluru" }
{ "_id" : ObjectId("5f9d6d86ba98701c046982b4"), "name" : "Kerala", "lastCensus" : ISODate("2019-01-
31T00:00:00Z"), "famousFor" : [ "Coconut" ], "mayor" : { "name" : "Jerry" }, "population" : 6200 }
{ "_id" : ObjectId("5f9d6df9ba98701c046982b5"), "name" : "Tamil Nadu", "poulation" : 58000,
"lastCensus" : ISODate("2019-09-20T00:00:00Z"), "famousFor" : [ "food", "honey", "forest" ], "mayor" : {
"name" : "Harry", "party" : "D" } }

> db.states.update( { _id: ObjectId("5f9d6a6027705e6cf3157926") }, { $inc: { 'population':10 } } )
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })

> db.states.find()
```

```
{ "_id" : ObjectId("5f9d6a6027705e6cf3157926"), "name" : "Karnataka", "population" : 22200010,
"lastCensus" : ISODate("2019-06-19T00:00:00Z"), "famousFor" : [ "dosa", "coffee", "beaches" ], "mayor"
: { "name" : "Tom", "party" : "D" }, "capital" : "Bengaluru" }
{ "_id" : ObjectId("5f9d6d86ba98701c046982b4"), "name" : "Kerala", "lastCensus" : ISODate("2019-01-
31T00:00:00Z"), "famousFor" : [ "Coconut" ], "mayor" : { "name" : "Jerry" }, "population" : 6200 }
{ "_id" : ObjectId("5f9d6df9ba98701c046982b5"), "name" : "Tamil Nadu", "poulation" : 58000,
"lastCensus" : ISODate("2019-09-20T00:00:00Z"), "famousFor" : [ "food", "honey", "forest" ], "mayor" : {
"name" : "Harry", "party" : "D" } }

> db.countries.insert({ _id:"usa", name:"United States of America"})
WriteResult({ "nInserted" : 1 })
> db.countries.remove({"name":"United States of America"})
WriteResult({ "nRemoved" : 1 })

> db.countries.find()
{ "_id" : "in", "name" : "india", "exports" : { "foods" : [ { "name" : "bacon", "tasty" : true }, { "name" :
"burgers" } ] } }

> db.states.find("this.population>100")
{ "_id" : ObjectId("5f9d6a6027705e6cf3157926"), "name" : "Karnataka", "population" : 22200010,
"lastCensus" : ISODate("2019-06-19T00:00:00Z"), "famousFor" : [ "dosa", "coffee", "beaches" ], "mayor"
: { "name" : "Tom", "party" : "D" }, "capital" : "Bengaluru" }
{ "_id" : ObjectId("5f9d6d86ba98701c046982b4"), "name" : "Kerala", "lastCensus" : ISODate("2019-01-
31T00:00:00Z"), "famousFor" : [ "Coconut" ], "mayor" : { "name" : "Jerry" }, "population" : 6200 }

>
db.states.update({_id:ObjectId("5f9d6a6027705e6cf3157926")},{$set:{country:{$ref:"countries",$id:"in"
}}})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })

> db.states.find()
{ "_id" : ObjectId("5f9d6a6027705e6cf3157926"), "name" : "Karnataka", "population" : 22200010,
"lastCensus" : ISODate("2019-06-19T00:00:00Z"), "famousFor" : [ "dosa", "coffee", "beaches" ], "mayor"
: { "name" : "Tom", "party" : "D" }, "capital" : "Bengaluru", "country" : DBRef("countries", "in") }
{ "_id" : ObjectId("5f9d6d86ba98701c046982b4"), "name" : "Kerala", "lastCensus" : ISODate("2019-01-
31T00:00:00Z"), "famousFor" : [ "Coconut" ], "mayor" : { "name" : "Jerry" }, "population" : 6200 }
{ "_id" : ObjectId("5f9d6df9ba98701c046982b5"), "name" : "Tamil Nadu", "poulation" : 58000,
"lastCensus" : ISODate("2019-09-20T00:00:00Z"), "famousFor" : [ "food", "honey", "forest" ], "mayor" : {
"name" : "Harry", "party" : "D" } }

> var karnataka=db.states.findOne({_id:ObjectId("5f9d6a6027705e6cf3157926")})

> var karnatakaCountryRef=karnataka.country.$ref;

> db.countries.findOne({_id:karnataka.country.$id})
{
        "_id" : "in",
        "name" : "india",
```

```
        "exports" : {
                "foods" : [
                        {
                                "name" : "bacon",
                                "tasty" : true
                        },
                        {
                                "name" : "burgers"
                        }
                ]
        }
}

> db.countries.drop()
true

> show collections
states

> db.states.count({'population':{$gt:500}})
2

> db.states.find()
{ "_id" : ObjectId("5f9d6a6027705e6cf3157926"), "name" : "Karnataka", "population" : 22200010,
"lastCensus" : ISODate("2019-06-19T00:00:00Z"), "famousFor" : [ "dosa", "coffee", "beaches" ], "mayor"
: { "name" : "Tom", "party" : "D" }, "capital" : "Bengaluru", "country" : DBRef("countries", "in") }
{ "_id" : ObjectId("5f9d6d86ba98701c046982b4"), "name" : "Kerala", "lastCensus" : ISODate("2019-01-
31T00:00:00Z"), "famousFor" : [ "Coconut" ], "mayor" : { "name" : "Jerry" }, "population" : 6200 }
{ "_id" : ObjectId("5f9d6df9ba98701c046982b5"), "name" : "Tamil Nadu", "poulation" : 58000,
"lastCensus" : ISODate("2019-09-20T00:00:00Z"), "famousFor" : [ "food", "honey", "forest" ], "mayor" : {
"name" : "Harry", "party" : "D" } }

> db.states.update( { _id: ObjectId("5f9d6df9ba98701c046982b5") }, { $rename: { 'poulation':
'population' } } )
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })

> db.states.count({'population':{$gt:7000}})
2

> db.states.find()
{ "_id" : ObjectId("5f9d6a6027705e6cf3157926"), "name" : "Karnataka", "population" : 22200010,
"lastCensus" : ISODate("2019-06-19T00:00:00Z"), "famousFor" : [ "dosa", "coffee", "beaches" ], "mayor"
: { "name" : "Tom", "party" : "D" }, "capital" : "Bengaluru", "country" : DBRef("countries", "in") }
{ "_id" : ObjectId("5f9d6d86ba98701c046982b4"), "name" : "Kerala", "lastCensus" : ISODate("2019-01-
31T00:00:00Z"), "famousFor" : [ "Coconut" ], "mayor" : { "name" : "Jerry" }, "population" : 6200 }
{ "_id" : ObjectId("5f9d6df9ba98701c046982b5"), "name" : "Tamil Nadu", "lastCensus" : ISODate("2019-
09-20T00:00:00Z"), "famousFor" : [ "food", "honey", "forest" ], "mayor" : { "name" : "Harry", "party" :
"D" }, "population" : 58000 }
```

```
> function insertPlace(
... name,population,lastCensus,
... famousFor,mayorInfo
... ){
... db.states.insert({
... name:name,
... poulation:population,
... lastCensus:ISODate(lastCensus),
... famousFor:famousFor,
... mayor:mayorInfo
... });
... }

> insertPlace("Kerala",6200,"2019-01-31",["Coconut"],{name:"larry"})

> db.states.find()
{ "_id" : ObjectId("5f9d6a6027705e6cf3157926"), "name" : "Karnataka", "population" : 22200010,
"lastCensus" : ISODate("2019-06-19T00:00:00Z"), "famousFor" : [ "dosa", "coffee", "beaches" ], "mayor"
: { "name" : "Tom", "party" : "D" }, "capital" : "Bengaluru", "country" : DBRef("countries", "in") }
{ "_id" : ObjectId("5f9d6d86ba98701c046982b4"), "name" : "Kerala", "lastCensus" : ISODate("2019-01-
31T00:00:00Z"), "famousFor" : [ "Coconut" ], "mayor" : { "name" : "Jerry" }, "population" : 6200 }
{ "_id" : ObjectId("5f9d6df9ba98701c046982b5"), "name" : "Tamil Nadu", "lastCensus" : ISODate("2019-
09-20T00:00:00Z"), "famousFor" : [ "food", "honey", "forest" ], "mayor" : { "name" : "Harry", "party" :
"D" }, "population" : 58000 }
{ "_id" : ObjectId("5f9d8b42ba98701c046982b6"), "name" : "Kerala", "poulation" : 6200, "lastCensus" :
ISODate("2019-01-31T00:00:00Z"), "famousFor" : [ "Coconut" ], "mayor" : { "name" : "larry" } }

> db.states.distinct('name')
[ "Karnataka", "Kerala", "Tamil Nadu" ]

> insertPlace("Tamil Nadu",6000,"2019-01-31",["Coconut"],{name:"larry"})

> insertPlace("Tamil Nadu",8880,"2019-01-31",["Coconut"],{name:"larry"})

> db.states.find()
{ "_id" : ObjectId("5f9d6a6027705e6cf3157926"), "name" : "Karnataka", "population" : 22200010,
"lastCensus" : ISODate("2019-06-19T00:00:00Z"), "famousFor" : [ "dosa", "coffee", "beaches" ], "mayor"
: { "name" : "Tom", "party" : "D" }, "capital" : "Bengaluru", "country" : DBRef("countries", "in") }
{ "_id" : ObjectId("5f9d6d86ba98701c046982b4"), "name" : "Kerala", "lastCensus" : ISODate("2019-01-
31T00:00:00Z"), "famousFor" : [ "Coconut" ], "mayor" : { "name" : "Jerry" }, "population" : 6200 }
{ "_id" : ObjectId("5f9d6df9ba98701c046982b5"), "name" : "Tamil Nadu", "lastCensus" : ISODate("2019-
09-20T00:00:00Z"), "famousFor" : [ "food", "honey", "forest" ], "mayor" : { "name" : "Harry", "party" :
"D" }, "population" : 58000 }
{ "_id" : ObjectId("5f9d8b42ba98701c046982b6"), "name" : "Kerala", "poulation" : 6200, "lastCensus" :
ISODate("2019-01-31T00:00:00Z"), "famousFor" : [ "Coconut" ], "mayor" : { "name" : "larry" } }
```

```
{ "_id" : ObjectId("5f9d8be3ba98701c046982b7"), "name" : "Tamil Nadu", "poulation" : 6000,
"lastCensus" : ISODate("2019-01-31T00:00:00Z"), "famousFor" : [ "Coconut" ], "mayor" : { "name" :
"larry" } }
{ "_id" : ObjectId("5f9d8beaba98701c046982b8"), "name" : "Tamil Nadu", "poulation" : 8880,
"lastCensus" : ISODate("2019-01-31T00:00:00Z"), "famousFor" : [ "Coconut" ], "mayor" : { "name" :
"larry" } }

> db.states.find().count()
6

> db.states.update( { _id: ObjectId("5f9d8beaba98701c046982b8") }, { $rename: { 'poulation':
'population' } } )
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })

> db.states.update( { _id: ObjectId("5f9d8be3ba98701c046982b7") }, { $rename: { 'poulation':
'population' } } )
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })

> db.states.update( { _id: ObjectId("5f9d8b42ba98701c046982b6") }, { $rename: { 'poulation':
'population' } } )
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })

> db.states.find().count()
6

> db.states.find()
{ "_id" : ObjectId("5f9d6a6027705e6cf3157926"), "name" : "Karnataka", "population" : 22200010,
"lastCensus" : ISODate("2019-06-19T00:00:00Z"), "famousFor" : [ "dosa", "coffee", "beaches" ], "mayor"
: { "name" : "Tom", "party" : "D" }, "capital" : "Bengaluru", "country" : DBRef("countries", "in") }
{ "_id" : ObjectId("5f9d6d86ba98701c046982b4"), "name" : "Kerala", "lastCensus" : ISODate("2019-01-
31T00:00:00Z"), "famousFor" : [ "Coconut" ], "mayor" : { "name" : "Jerry" }, "population" : 6200 }
{ "_id" : ObjectId("5f9d6df9ba98701c046982b5"), "name" : "Tamil Nadu", "lastCensus" : ISODate("2019-
09-20T00:00:00Z"), "famousFor" : [ "food", "honey", "forest" ], "mayor" : { "name" : "Harry", "party" :
"D" }, "population" : 58000 }
{ "_id" : ObjectId("5f9d8b42ba98701c046982b6"), "name" : "Kerala", "lastCensus" : ISODate("2019-01-
31T00:00:00Z"), "famousFor" : [ "Coconut" ], "mayor" : { "name" : "larry" }, "population" : 6200 }
{ "_id" : ObjectId("5f9d8be3ba98701c046982b7"), "name" : "Tamil Nadu", "lastCensus" : ISODate("2019-
01-31T00:00:00Z"), "famousFor" : [ "Coconut" ], "mayor" : { "name" : "larry" }, "population" : 6000 }
{ "_id" : ObjectId("5f9d8beaba98701c046982b8"), "name" : "Tamil Nadu", "lastCensus" : ISODate("2019-
01-31T00:00:00Z"), "famousFor" : [ "Coconut" ], "mayor" : { "name" : "larry" }, "population" : 8880 }
>
db.states.aggregate([{$group:{_id:{name:"$name"},totalpopulation:{$sum:"$population"},count:{$sum:
1}}}])
{ "_id" : { "name" : "Tamil Nadu" }, "totalpopulation" : 72880, "count" : 3 }
{ "_id" : { "name" : "Kerala" }, "totalpopulation" : 12400, "count" : 2 }
{ "_id" : { "name" : "Karnataka" }, "totalpopulation" : 22200010, "count" : 1 }
> db.states.update({}, {$unset: {lastCensus:1}} , {multi: true});
WriteResult({ "nMatched" : 6, "nUpserted" : 0, "nModified" : 6 })
```

```
> db.states.find()
{ "_id" : ObjectId("5f9d6a6027705e6cf3157926"), "name" : "Karnataka", "population" : 22200010, "famousFor" : [ "dosa", "coffee", "beaches" ], "mayor" : { "name" : "Tom", "party" : "D" }, "capital" : "Bengaluru", "country" : DBRef("countries", "in") }
{ "_id" : ObjectId("5f9d6d86ba98701c046982b4"), "name" : "Kerala", "famousFor" : [ "Coconut" ], "mayor" : { "name" : "Jerry" }, "population" : 6200 }
{ "_id" : ObjectId("5f9d6df9ba98701c046982b5"), "name" : "Tamil Nadu", "famousFor" : [ "food", "honey", "forest" ], "mayor" : { "name" : "Harry", "party" : "D" }, "population" : 58000 }
{ "_id" : ObjectId("5f9d8b42ba98701c046982b6"), "name" : "Kerala", "famousFor" : [ "Coconut" ], "mayor" : { "name" : "larry" }, "population" : 6200 }
{ "_id" : ObjectId("5f9d8be3ba98701c046982b7"), "name" : "Tamil Nadu", "famousFor" : [ "Coconut" ], "mayor" : { "name" : "larry" }, "population" : 6000 }
{ "_id" : ObjectId("5f9d8beaba98701c046982b8"), "name" : "Tamil Nadu", "famousFor" : [ "Coconut" ], "mayor" : { "name" : "larry" }, "population" : 8880 }
```

# HBase-Commands Executed

```
shaaz@PES1201801754:~/hbase/bin$./start-hbase.sh
localhost: running zookeeper, logging to /home/shaaz/hbase/bin/../logs/hbase-shaaz-zookeeper-shaaz.out
running master, logging to /home/shaaz/hbase/bin/../logs/hbase-shaaz-master-shaaz.out
OpenJDK 64-Bit Server VM warning: ignoring option PermSize=128m; support was removed in 8.0
OpenJDK 64-Bit Server VM warning: ignoring option MaxPermSize=128m; support was removed in 8.0
: running regionserver, logging to /home/shaaz/hbase/bin/../logs/hbase-shaaz-regionserver-shaaz.out
: OpenJDK 64-Bit Server VM warning: ignoring option PermSize=128m; support was removed in 8.0
: OpenJDK 64-Bit Server VM warning: ignoring option MaxPermSize=128m; support was removed in 8.0
shaaz@PES1201801754:~/hbase/bin$jps
3905 HMaster
3844 HQuorumPeer
3156 NodeManager
3976 HRegionServer
3002 ResourceManager
2813 SecondaryNameNode
2446 NameNode
2606 DataNode
4015 Jps
shaaz@PES1201801754:~/hbase/bin$./hbase shell
2020-11-06 21:22:48,196 WARN  [main] util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
HBase Shell
Use "help" to get list of supported commands.
Use "exit" to quit this interactive shell.
Version 1.4.13, r38bf65a22b7e9320f07aeb27677e4533b9a77ef4, Sun Feb 23 02:06:36 PST 2020
hbase(main):001:0> create 'test', 'cf'
0 row(s) in 20.2640 seconds

=> Hbase::Table - test
hbase(main):002:0> list
TABLE
test
1 row(s) in 0.2050 seconds

=> ["test"]
hbase(main):003:0> describe 'test'
Table test is ENABLED
test
```

```
COLUMN FAMILIES DESCRIPTION
{NAME => 'cf', BLOOMFILTER => 'ROW', VERSIONS => '1', IN_MEMORY => 'false', KEEP_DELETED_CELLS
=> 'FALSE', DATA_BLOCK_ENCODING => 'NONE', TTL
=> 'FOREVER', COMPRESSION => 'NONE', MIN_VERSIONS => '0', BLOCKCACHE => 'true', BLOCKSIZE =>
'65536', REPLICATION_SCOPE => '0'}
1 row(s) in 0.5100 seconds

hbase(main):004:0> put 'test', 'row1', 'cf:a', 'value1'
0 row(s) in 0.3380 seconds

hbase(main):005:0> put 'test', 'row2', 'cf:b', 'value2'
0 row(s) in 0.0070 seconds

hbase(main):006:0> put 'test', 'row3', 'cf:c', 'value3'
0 row(s) in 0.0720 seconds

hbase(main):007:0> count 'test'
3 row(s) in 0.1380 seconds

=> 3
hbase(main):008:0> scan 'test'
ROW                          COLUMN+CELL
 row1                        column=cf:a, timestamp=1604678231433, value=value1
 row2                        column=cf:b, timestamp=1604678255597, value=value2
 row3                        column=cf:c, timestamp=1604678379015, value=value3
3 row(s) in 0.0770 seconds

hbase(main):009:0> get 'test', 'row2'
COLUMN                           CELL
 cf:b                        timestamp=1604678255597, value=value2
1 row(s) in 0.6580 seconds

hbase(main):010:0> disable 'test'
0 row(s) in 4.7350 seconds

hbase(main):011:0> alter 'test', {NAME => 'cf', VERSIONS => 3}
Updating all regions with the new schema...
1/1 regions updated.
Done.
0 row(s) in 2.3920 seconds

hbase(main):012:0> enable 'test'
```

```
0 row(s) in 1.5880 seconds

hbase(main):013:0> put 'test', 'row3', 'cf:c', 'value300'
0 row(s) in 0.0470 seconds

hbase(main):014:0> scan 'test'
ROW                    COLUMN+CELL
 row1                  column=cf:a, timestamp=1604678231433, value=value1
 row2                  column=cf:b, timestamp=1604678255597, value=value2
 row3                  column=cf:c, timestamp=1604678717504, value=value300
3 row(s) in 0.0710 seconds

hbase(main):015:0> get 'test', 'row3', {COLUMN => 'cf:c', VERSIONS => 3}
COLUMN                 CELL
 cf:c                  timestamp=1604678717504, value=value300
 cf:c                  timestamp=1604678379015, value=value3
1 row(s) in 0.0970 seconds

hbase(main):016:0> delete 'test', 'row3', 'cf:c', 1604678717504
0 row(s) in 0.1880 seconds

hbase(main):017:0> scan 'test'
ROW                    COLUMN+CELL
 row1                  column=cf:a, timestamp=1604678231433, value=value1
 row2                  column=cf:b, timestamp=1604678255597, value=value2
 row3                  column=cf:c, timestamp=1604678379015, value=value3
3 row(s) in 0.5910 seconds

hbase(main):018:0> deleteall 'test', 'row3'
0 row(s) in 0.0160 seconds

hbase(main):019:0> scan 'test'
ROW                    COLUMN+CELL
 row1                  column=cf:a, timestamp=1604678231433, value=value1
 row2                  column=cf:b, timestamp=1604678255597, value=value2
2 row(s) in 0.0570 seconds

hbase(main):020:0> put 'test', 'row3', 'cf:c', 'value3'
0 row(s) in 0.0070 seconds

hbase(main):021:0> put 'test', 'row3', 'cf:c', 'value300'
0 row(s) in 0.0120 seconds
```

```
hbase(main):022:0> scan 'test'
ROW                     COLUMN+CELL
 row1                   column=cf:a, timestamp=1604678231433, value=value1
 row2                   column=cf:b, timestamp=1604678255597, value=value2
 row3                   column=cf:c, timestamp=1604679163668, value=value300
3 row(s) in 0.0740 seconds

hbase(main):023:0> deleteall 'test', 'row3'
0 row(s) in 0.0090 seconds

hbase(main):024:0> scan 'test'
ROW                     COLUMN+CELL
 row1                   column=cf:a, timestamp=1604678231433, value=value1
 row2                   column=cf:b, timestamp=1604678255597, value=value2
2 row(s) in 0.0420 seconds

hbase(main):025:0> truncate 'test'
Truncating 'test' table (it may take a while):
 - Disabling table...
 - Truncating table...
0 row(s) in 6.2210 seconds

hbase(main):026:0> scan 'test'
ROW                     COLUMN+CELL
0 row(s) in 0.4990 seconds

hbase(main):027:0> disable 'test'
0 row(s) in 2.3150 seconds

hbase(main):028:0> drop 'test'
0 row(s) in 1.4630 seconds

hbase(main):029:0> list
TABLE
0 row(s) in 0.0120 seconds

=> []
hbase(main):030:0> create 'test', 'cf'
0 row(s) in 2.4080 seconds

=> Hbase::Table - test
```

```
hbase(main):031:0> list
TABLE
test
1 row(s) in 0.0150 seconds

=> ["test"]
hbase(main):032:0> put 'test', 'row1', 'cf:a', 'value1'
0 row(s) in 0.0510 seconds

hbase(main):033:0> put 'test', 'row2', 'cf:b', 'value2'
0 row(s) in 0.0370 seconds

hbase(main):034:0> put 'test', 'row3', 'cf:c', 'value3'
0 row(s) in 0.0330 seconds

hbase(main):035:0> put 'test', 'row2', 'cf:b', 'value4'
0 row(s) in 0.0150 seconds

hbase(main):036:0> scan 'test'
ROW                    COLUMN+CELL
 row1                  column=cf:a, timestamp=1604679686620, value=value1
 row2                  column=cf:b, timestamp=1604679728680, value=value4
 row3                  column=cf:c, timestamp=1604679704261, value=value3
3 row(s) in 0.0150 seconds

hbase(main):037:0> put 'test', 'row2', 'cf:d', 'value2'
0 row(s) in 0.0370 seconds

hbase(main):038:0> scan 'test'
ROW                    COLUMN+CELL
 row1                  column=cf:a, timestamp=1604679686620, value=value1
 row2                  column=cf:b, timestamp=1604679728680, value=value4
 row2                  column=cf:d, timestamp=1604679753658, value=value2
 row3                  column=cf:c, timestamp=1604679704261, value=value3
3 row(s) in 0.0390 seconds

hbase(main):039:0> count 'test'
3 row(s) in 0.0420 seconds

=> 3

hbase(main):042:0> get 'test', 'row2'
```

```
COLUMN                        CELL
 cf:b                         timestamp=1604679728680, value=value4
 cf:d                         timestamp=1604679753658, value=value2
1 row(s) in 0.0480 seconds

hbase(main):043:0> put 'test', 'row2', 'cf:a', 'value2'
0 row(s) in 0.0520 seconds

hbase(main):044:0> scan 'test'
ROW                           COLUMN+CELL
 row1                         column=cf:a, timestamp=1604679686620, value=value1
 row2                         column=cf:a, timestamp=1604680046583, value=value2
 row2                         column=cf:b, timestamp=1604679728680, value=value4
 row2                         column=cf:d, timestamp=1604679753658, value=value2
 row3                         column=cf:c, timestamp=1604679704261, value=value3
3 row(s) in 0.0680 seconds

hbase(main):045:0> deleteall 'test', 'row3'
0 row(s) in 0.0130 seconds

hbase(main):046:0> delete 'test', 'row2', 'cf:d'
0 row(s) in 0.0250 seconds

hbase(main):047:0> delete 'test', 'row2', 'cf:a'
0 row(s) in 0.0330 seconds

hbase(main):048:0> scan 'test'
ROW                           COLUMN+CELL
 row1                         column=cf:a, timestamp=1604679686620, value=value1
 row2                         column=cf:b, timestamp=1604679728680, value=value4
2 row(s) in 0.0210 seconds

hbase(main):049:0> put 'test', 'row2', 'cf:b', 'value2'
0 row(s) in 0.0110 seconds

hbase(main):050:0> scan 'test'
ROW                           COLUMN+CELL
 row1                         column=cf:a, timestamp=1604679686620, value=value1
 row2                         column=cf:b, timestamp=1604680388867, value=value2
2 row(s) in 0.0160 seconds
```

## Neo4j-Commands Executed

```
Match (m:Movie) where m.released > 1000 RETURN m limit 3

Match (p:Person) RETURN p limit 30

Match (n) RETURN n limit 30

MATCH (m:Movie) RETURN m.title, m.released

Create (p:Person {name: 'Jane Doe'}) RETURN p

Match (p:Person {name: 'Tom Hanks'}) RETURN p

Match (p:Person)where p.name='Tom Hanks' RETURN p

Match (m:Movie { title:"Cloud Altas"}) RETURN m

Match (m:Movie) where m.released > 2010 and m.released <2015 RETURN m

Create (m:Movie {title:'Forrest Gump'}) Return m

MERGE (p:Person {name: 'Jane Doe'}) ON MATCH SET
p.lastLoggedInAt=timestamp() ON CREATE SET p.createdAt=timestamp()
Return p

MATCH (p:PERSON),(m:Movie) WHERE p.name='Tom Hanks' and m.title=
'Forrest Gump' CREATE(p)-[r:ACTED_IN]->(m) Return type(r)

MATCH (p:Person),(m:Movie) WHERE p.name=\"Jane Doe\" and
m.title=\"Forrest Gump\" CREATE (p)-[w:WATCHED]->(m) RETURN type(w)

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

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

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

MATCH (m:Movie{title:'Cloud Atlas'})<-[d:DIRECTED]-(p:Person) return

p.name

MATCH (tom:Person {name:\"Tom Hanks\"}) -[:ACTED_IN]->(m:Movie)<-

[:ACTED_IN]-(p:Person) return p.name, m.title
```

```
MATCH (p:Person)-[relatedTo]-(m:Movie{title:'Cloud Atlas'}) return
p.name, type(relatedTo)
MATCH (p:Person{name:'Kevin Bacon'})-[*1..3]-(hollywood) return
distinct p,Hollywood
MATCH (m:Movie) where m.released >2000 RETURN m.released, count(*)
MATCH (p:Person)-[r:WATCHED]->(m:Movie)return m.title,count(*)
MATCH (tom:Person{name:'Tom Hanks'})-[:ACTED_IN]->(m:Movie)return
m.released, count(*) order by m.released
MATCH (tom:Person{name:'Tom Hanks'})-[:ACTED_IN]->(m:Movie)<-
[:ACTED_IN]-(p:Person) return tom.name,p.name,count(*) order by
count(*) desc
MATCH (tom:Person{name:'Tom Hanks'})-[:ACTED_IN]->(m:Movie) return
tom.name,m.title,min(m.released) order by min(m.released) desc
MATCH (m:Movie{title:"Forrest Gump"}) return m
MATCH (m:Movie{title:"Forrest Gump"}) SET m.released=1994 RETURN m
MATCH (tom:Person{name:'Tom Hanks'})-[:ACTED_IN]->(m:Movie)<-
[:ACTED_IN]-(p:Person{name:'Meg Ryan'}) return tom.name,p.name,m.title
MATCH (m:Movie{title:'Forrest Gump'}) DETACH DELETE m
CREATE (M:Movie{tile:"philadelphia",released:1993}) return M
```