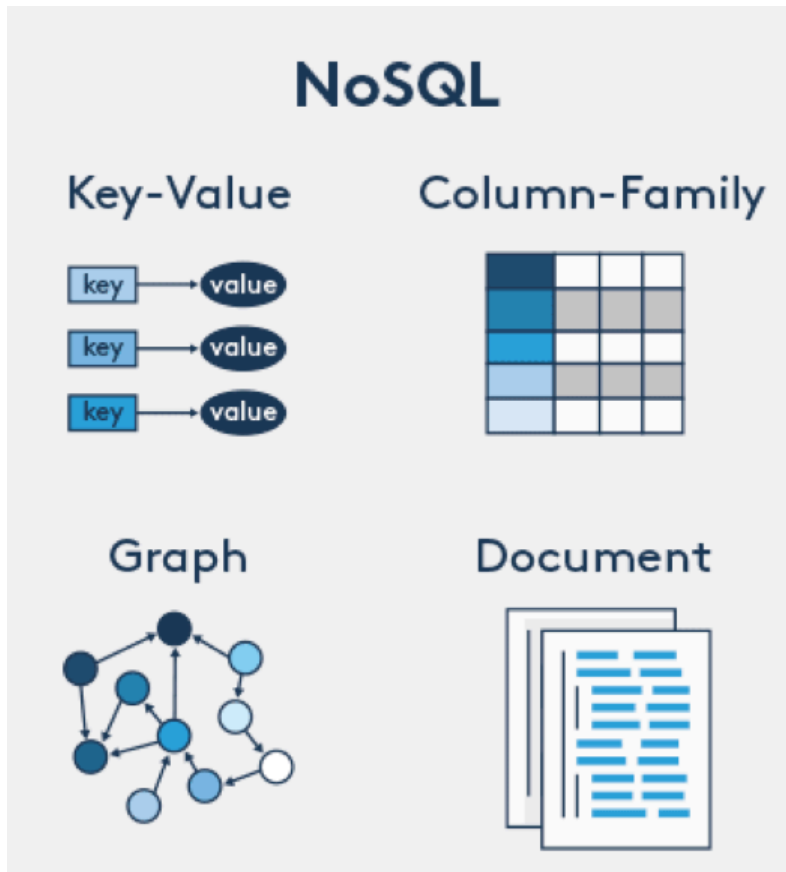


Clase 9 -Tipos de Motores NoSQL

- Bases de datos de valores clave. (“Key-Value Pair Databases”)
- Base de datos de documentos. (“Document Databases”)
- Bases de datos de columnas. (“Column-Family Databases”)
- Bases de datos de grafos. (“Graph Databases”)



-Ejemplos:



amazon
DynamoDB

Key Value

A P A C H E
HBASE

Column Family



neo4j

Graph



mongoDB

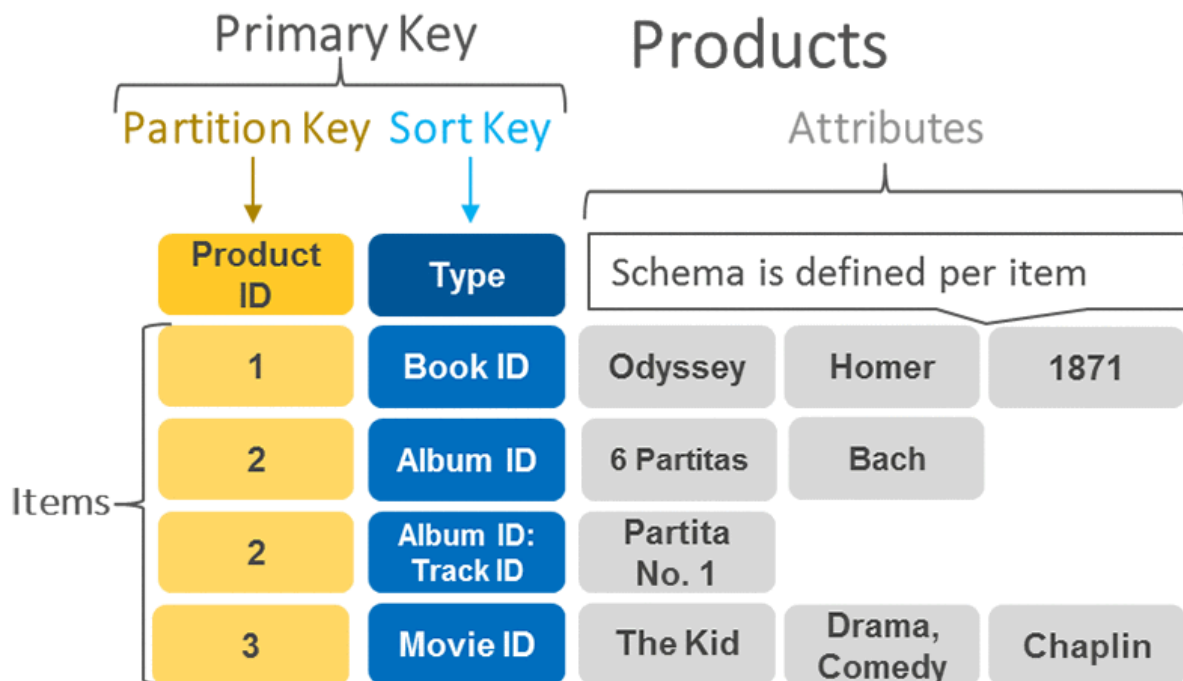
Document

-DynamoDB

DynamoDB es una base de datos NoSQL (Not only SQL) de tipo clave-valor disponible como servicio en AWS.

En DynamoDB las tablas son las colecciones de elementos, y los elementos son colecciones de atributos o pares clave-valor. La clave primaria de una tabla está compuesta de una clave de partición y de una clave de clasificación (sort key).

También, se puede usar un índice secundario global o GSI, que permite realizar consultas filtrando por columnas que no sean la clave de partición o de clasificación.



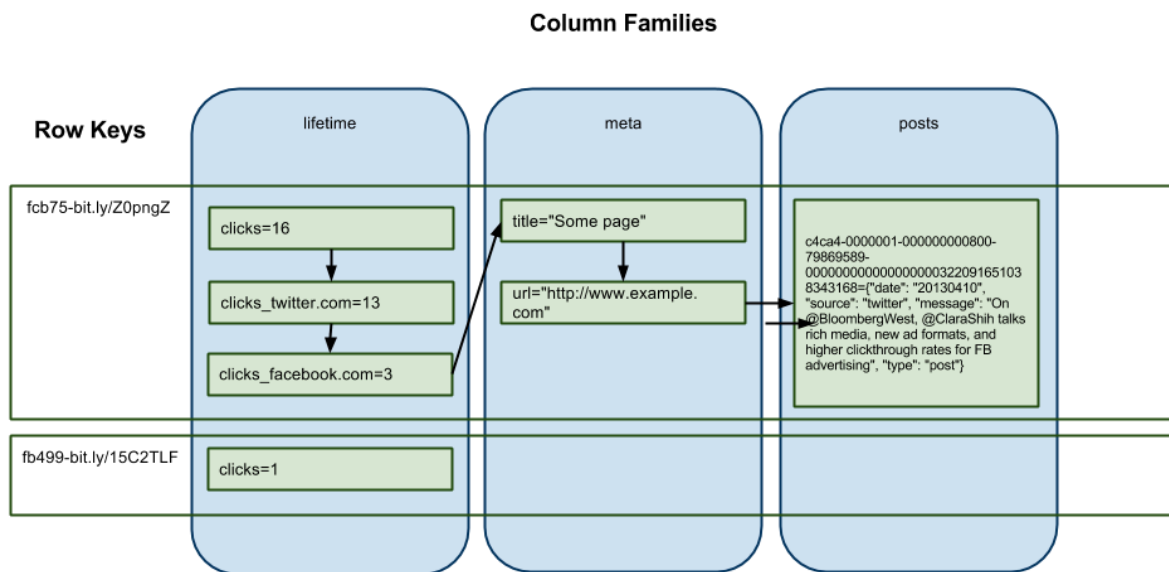
-HBase

Está compuesto por una serie de tablas que contienen filas y columnas, en forma similar a una base de datos tradicional.

Cada tabla consta de una Clave Primaria (“Primary Key”), todo acceso a las tablas es realizado usando la Clave Primaria.

Una columna en Hbase representa un atributo de un objeto.

Hbase es diferente a las bases RDBMS en términos de cómo los datos son almacenados. Todos los datos de Hbase se almacenan en archivos HDFS.



-Cassandra

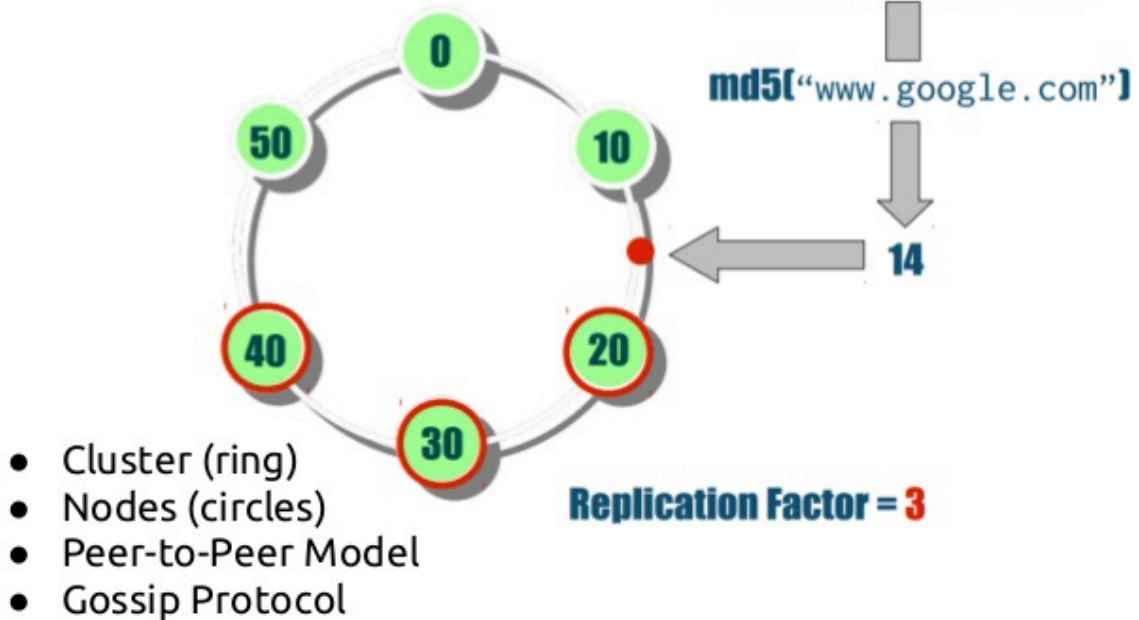
Apache Cassandra es un híbrido entre bases de datos de valores clave y de columnas. Provee una arquitectura escalable, disponibilidad continua, protección de datos, replicación “multi-data” sobre centros de datos (“data centers”), compresión de datos y un lenguaje “SQL Like”.

Su arquitectura es “master-less”, en el cual todos los nodos actúan por igual, comunicándose entre ellos por medio de un protocolo distribuido y escalable llamado Gossip.

El modelo de datos de Cassandra consiste en filas particionadas que se almacenan en tablas con un nivel de consistencia configurable, indexadas por medio de llaves (“keys”).

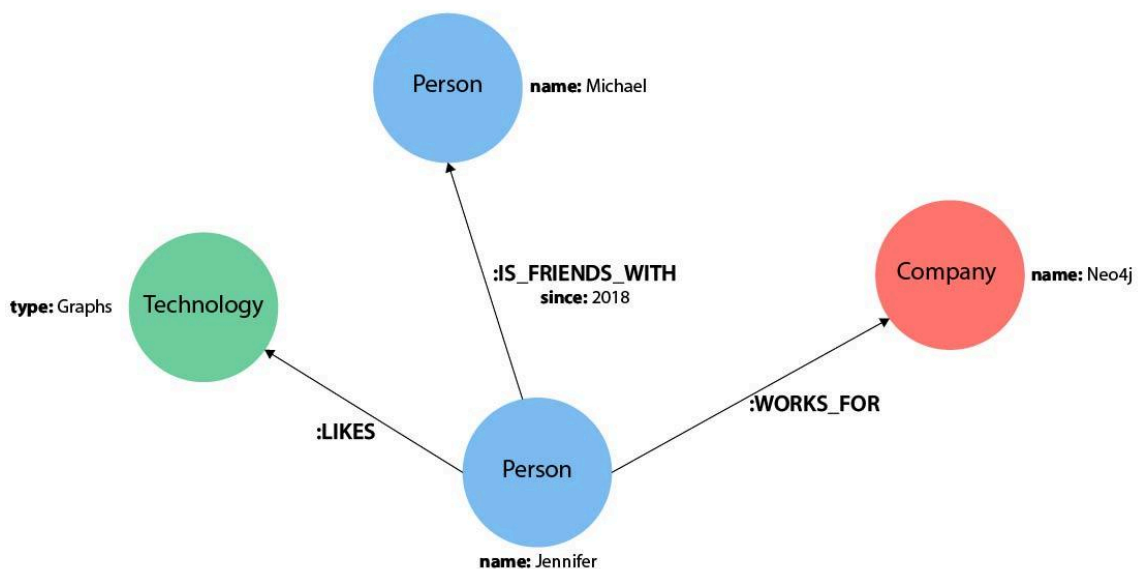
Modelo Peer-To- Peer, permite que los nodos se relacionen entre sí.

Architecture



-Neo4J

Neo4j es un software libre de Base de datos orientada a grafos, implementado en Java. Es un motor de persistencia embebido, basado en disco, implementado en Java, completamente transaccional, que almacena datos estructurados en grafos en lugar de en tablas.

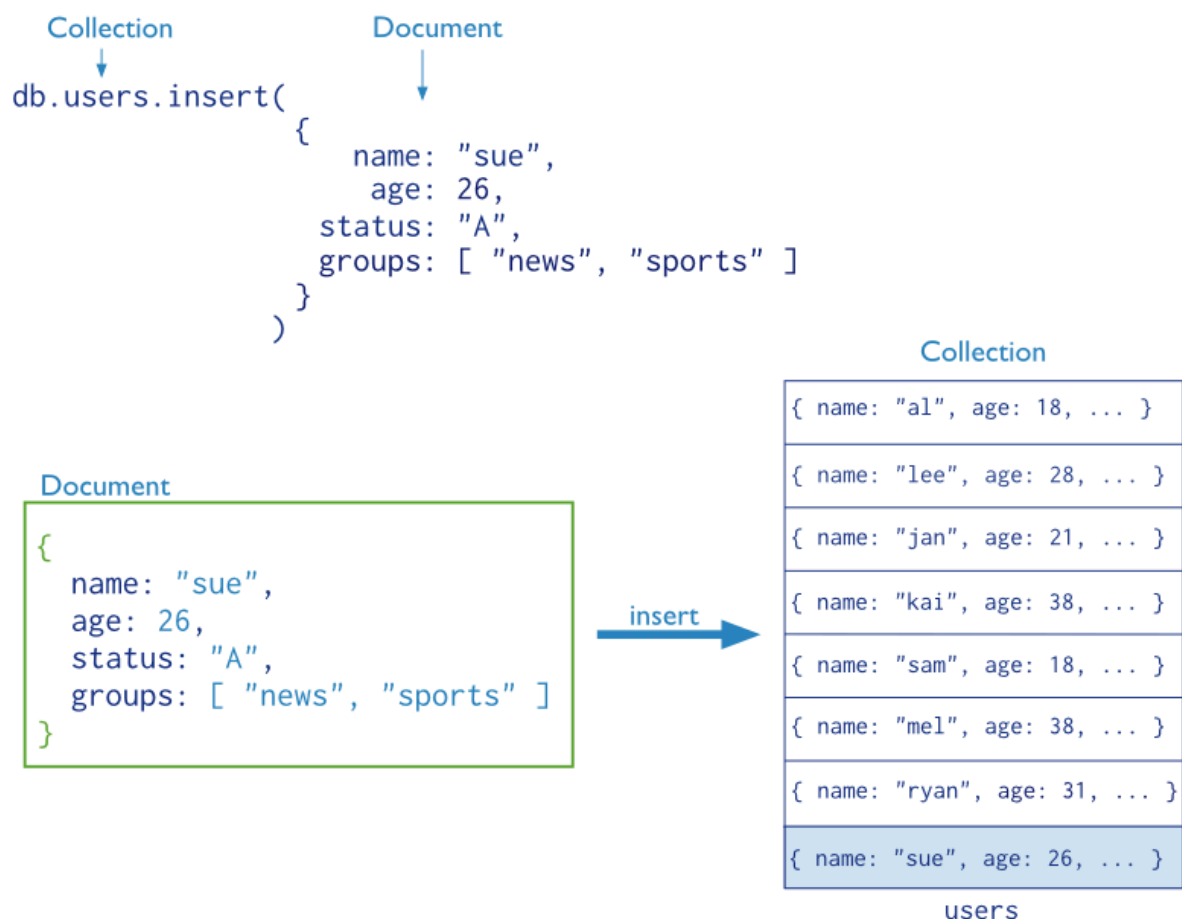


-MongoDB

MongoDB es una base de datos distribuida, basada en documentos y de uso general que ha sido diseñada para desarrolladores de aplicaciones modernas y para la era de la nube. Un registro en MongoDB es un documento, con una estructura de datos compuesta por campo ("field") y pares de valores ("value pairs"). Los documentos MongoDB son similares a objetos JSON.

Características:

- Lenguaje de consultas CRUD.
- Alta performance.
- Alta disponibilidad.
- Escalabilidad horizontal.
- Soporte para diferentes motores de almacenamiento.



-Enlaces de referencia:

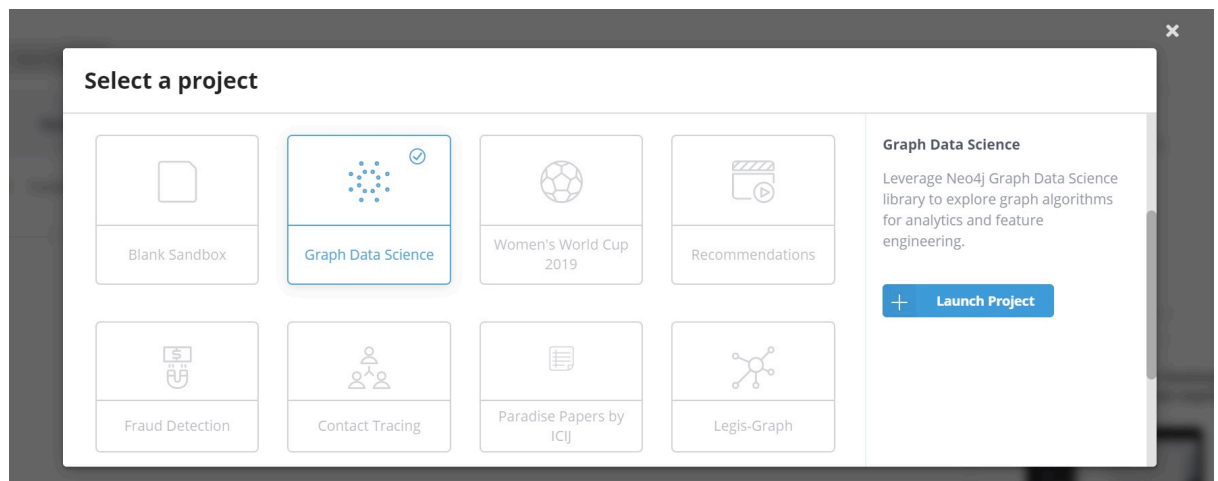
- HBase https://hbase.apache.org/book.html#_architecture
- Cassandra <https://docs.datastax.com/en/dse/6.8/cql/index.html>
- MongoDB <https://docs.mongodb.com/manual/tutorial/getting-started/>
- DynamoDB:
 - <https://aws.amazon.com/es/getting-started/hands-on/create-nosql-table/>
 - <https://www.allthingsdistributed.com/files/amazon-dynamo-sosp2007.pdf>
- Neo4J <https://neo4j.com/developer/get-started/>
- Drill <http://drill.apache.org/docs/drill-in-10-minutes/>
- <https://www.freecodecamp.org/news/nosql-databases-5f6639ed9574/>

Laboratorio:

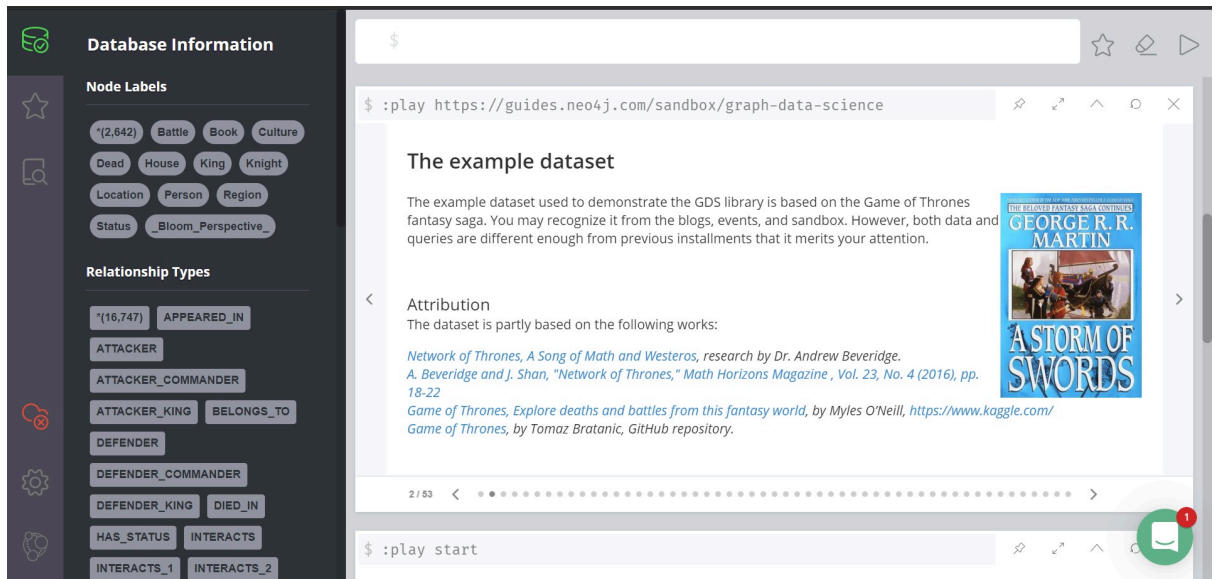
-Vamos a utilizar las plataformas online que ofrecen los vendedores de soluciones NoSQL.

- Neo4J
 - <https://sandbox.neo4j.com/login>

Paso 1



Paso 2



Paso 3

- Cassandra
 - <https://astra.datastax.com/register>

Paso 1

DATASTAX ASTRA

0 Databases

[← Astra Databases](#) [Create New Database](#)

Compute Size

Choose the compute power of your database. Higher tiers provide higher throughput at lower latency.

☒ Free tier: 10 GB of storage. Free forever, data is not deleted automatically.
☐ C10: starter configuration suitable for development and production workloads.
C10 databases start with 1CU (HA 3 replica set with 500gb of storage per replica) and can be easily scaled horizontally

Location

us-east1

Estimated Cost

☒ Per Hour ☐ Per Month

when running: \$0.00/hour
 when parked: \$0.00/hour

Database Details

Database name *
EducacionIT

Keyspace name *
training

Enter alphanumeric characters only. Use quotes to preserve case.

Database username *
training

Database user password *

Confirm Password *

Paso 2

```
Logging in with cqlsh
User: training
Password:
Connected to caas-cluster at caas-cluster-caas-dc-service:9042.
[cqlsh 6.8.0 | DSE 6.8.2.125 | CQL spec 3.4.5 | DSE protocol v2]
Use HELP for help.
training@cqlsh>
training@cqlsh>
training@cqlsh>
```

- MongoDB
 - <https://docs.mongodb.com/manual/tutorial/getting-started/>

Examples


```
MongoDB Web Shell

Click to connect
Connecting...
MongoDB shell version v4.2.0
connecting to: mongodb://127.0.0.1:27017/?
authSource=admin&compressors=disabled&gssapiServiceName=mongodb
Implicit session: session { "id" : UUID("12e89398-a6e1-465c-8893-0ffd7bd73fcc") }
MongoDB server version: 4.2.0
type "help" for help
>>>
```

Click inside the shell to connect. Once connected, you can run the examples in the shell above.

0. Switch Database	1. Populate a collection (Insert)	2. Select All Documents	3. Specify Equality Matches	4. Specify Fields to Return (Projection)
--------------------	-----------------------------------	-------------------------	-----------------------------	--

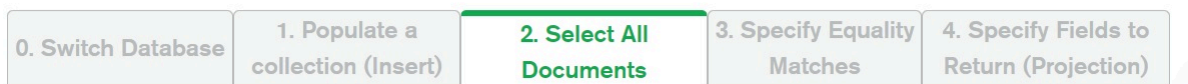
Examples



```
MongoDB Web Shell

>>> db.inventory.find({}).pretty()
{
  "_id" : ObjectId("5f0eb0d0315314662a2c1333"),
  "item" : "journal",
  "qty" : 25,
  "status" : "A",
  "size" : {
    "h" : 14,
    "w" : 21,
    "uom" : "cm"
  },
  "tags" : [
    "blank",
    "red"
  ]
}
```

Click inside the shell to connect. Once connected, you can run the examples in the shell above.



- DynamoDB (opcional, debemos contar con una cuenta en AWS)
 - <https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/GettingStartedDynamoDB.html>

Proyecto Integrador:

No-SQL

Se puede utilizar el entorno docker-compose-v3.yml

En esta etapa, se va a probar el uso de las herramienta HBase, MongoDB, Neo4J. Para lo cuál, es necesario seguir las instrucciones indicadas.

1) HBase:

```
1- sudo docker exec -it hbase-master hbase shell
create 'personal','personal_data'
list 'personal'
put 'personal',1,'personal_data:name','Juan'
put 'personal',1,'personal_data:city','Córdoba'
put 'personal',1,'personal_data:age','25'
put 'personal',2,'personal_data:name','Franco'
put 'personal',2,'personal_data:city','Lima'
put 'personal',2,'personal_data:age','32'
put 'personal',3,'personal_data:name','Ivan'
put 'personal',3,'personal_data:age','34'
put 'personal',4,'personal_data:name','Eliecer'
put 'personal',4,'personal_data:city','Caracas'
```

```
get 'personal','4'
```

2-En el namenode del cluster:

```
hdfs dfs -put personal.csv /hbase/data/personal.csv
```

```
3-sudo docker exec -it hbase-master bash
```

```
hbase org.apache.hadoop.hbase.mapreduce.ImportTsv -Dimporttsv.separator=''  
-Dimporttsv.columns=HBASE_ROW_KEY,personal_data:name,personal_data:city,personal_  
data:age personal hdfs://namenode:9000/hbase/data/personal.csv
```

```
hbase shell
```

```
scan 'personal'
```

```
create 'album','label','image'
```

```
put 'album','label1','label:size','10'
```

```
put 'album','label1','label:color','255:255:255'
```

```
put 'album','label1','label:text','Family album'
```

```
put 'album','label1','image:name','holiday'
```

```
put 'album','label1','image:source','/tmp/pic1.jpg'
```

```
get 'album','label1'
```

2) MongoDB

```
1) sudo docker cp iris.csv mongodb:/data/iris.csv
```

```
sudo docker cp iris.json mongodb:/data/iris.json
```

```
2) sudo docker exec -it mongodb bash
```

```
3) mongoimport /data/iris.csv --type csv --headerline -d dataprueba -c iris_csv
```

```
mongoimport --db dataprueba --collection iris_json --file /data/iris.json --jsonArray
```

```
4) mongosh
```

```
use dataprueba
```

```
show collections
```

```
db.iris_csv.find()
```

```
db.iris_json.find()
```

```
5) mongoexport --db dataprueba --collection iris_csv --fields  
sepal_length,sepal_width,petal_length,petal_width,species --type=csv --out  
/data/iris_export.csv
```

```
mongoexport --db dataprueba --collection iris_json --fields  
sepal_length,sepal_width,petal_length,petal_width,species --type=json --out  
/data/iris_export.json
```

6) Descargar desde <https://search.maven.org/search?q=g:org.mongodb:mongo-hadoop>
los jar:

```
https://search.maven.org/search?q=a:mongo-hadoop-hive
```

```
https://search.maven.org/search?q=a:mongo-hadoop-spark
```

```
sudo docker cp mongo-hadoop-hive-2.0.2.jar
```

```
hive-server:/opt/hive/lib/mongo-hadoop-hive-2.0.2.jar
```

```
sudo docker cp mongo-hadoop-core-2.0.2.jar
```

```
hive-server:/opt/hive/lib/mongo-hadoop-core-2.0.2.jar
```

```

sudo docker cp mongo-hadoop-spark-2.0.2.jar
hive-server:/opt/hive/lib/mongo-hadoop-spark-2.0.2.jar
sudo docker cp mongo-java-driver-3.12.11.jar
hive-server:/opt/hive/lib/mongo-java-driver-3.12.11.jar

```

```

7) sudo docker cp iris.hql hive-server:/opt/iris.hql
    sudo docker exec -it hive-server bash

```

```

8) hiveserver2
   chmod 777 iris.hql
   hive -f iris.hql

```

3) Neo4J

```

CREATE (a:Location {name: 'A'}),
       (b:Location {name: 'B'}),
       (c:Location {name: 'C'}),
       (d:Location {name: 'D'}),
       (e:Location {name: 'E'}),
       (f:Location {name: 'F'}),
       (a)-[:ROAD {cost: 50}]->(b),
       (b)-[:ROAD {cost: 50}]->(a),
       (a)-[:ROAD {cost: 50}]->(c),
       (c)-[:ROAD {cost: 50}]->(a),
       (a)-[:ROAD {cost: 100}]->(d),
       (d)-[:ROAD {cost: 100}]->(a),
       (b)-[:ROAD {cost: 40}]->(d),
       (d)-[:ROAD {cost: 40}]->(b),
       (c)-[:ROAD {cost: 40}]->(d),
       (d)-[:ROAD {cost: 40}]->(c),
       (c)-[:ROAD {cost: 80}]->(e),
       (e)-[:ROAD {cost: 80}]->(c),
       (d)-[:ROAD {cost: 30}]->(e),
       (e)-[:ROAD {cost: 30}]->(d),
       (d)-[:ROAD {cost: 80}]->(f),
       (f)-[:ROAD {cost: 80}]->(d),
       (e)-[:ROAD {cost: 40}]->(f),
       (f)-[:ROAD {cost: 40}]->(e);

```

```

CALL gds.graph.project(
    'miGrafo',
    'Location',
    'ROAD',
    {
        relationshipProperties: 'cost'
    }
)

```

```
MATCH (l:Location) RETURN l
```

```
MATCH (source:Location {name: 'A'}), (target:Location {name: 'E'})
  CALL gds.shortestPath.dijkstra.write.estimate('miGrafo', {
    sourceNode: source,
    targetNode: target,
    relationshipWeightProperty: 'cost',
    writeRelationshipType: 'PATH'
  })
YIELD nodeCount, relationshipCount, bytesMin, bytesMax, requiredMemory
RETURN nodeCount, relationshipCount, bytesMin, bytesMax,
requiredMemory
```

Ejemplo de logística:

<https://neo4j.com/docs/graph-data-science/current/alpha-algorithms/minimum-weight-spanning-tree/>

```
MATCH (n:Location {name: 'A'})
CALL gds.alpha.spanningTree.minimum.write('miGrafo', {
  startNodeId: id(n),
  relationshipWeightProperty: 'cost',
  writeProperty: 'MINST',
  weightWriteProperty: 'writeCost'
})
YIELD preProcessingMillis, computeMillis, writeMillis, effectiveNodeCount
RETURN preProcessingMillis, computeMillis, writeMillis, effectiveNodeCount;
```

Ejemplo de logística:

```
MATCH path = (n:Location {name: 'A'})-[:MINST*]-()
WITH relationships(path) AS rels
UNWIND rels AS rel
WITH DISTINCT rel AS rel
RETURN startNode(rel).name AS source, endNode(rel).name AS destination,
rel.writeCost AS cost
```

```
MATCH (n) DETACH DELETE n
```

```
sudo docker cp producto.csv neo4j:/var/lib/neo4j/import/producto.csv
sudo docker cp tipo_producto.csv
neo4j:/var/lib/neo4j/import/tipo_producto.csv
sudo docker cp cliente.csv neo4j:/var/lib/neo4j/import/cliente.csv
sudo docker cp venta.csv neo4j:/var/lib/neo4j/import/venta.csv
(Ver Archivo "ejemploNeo4J.txt")
```