NOSQL UNIT-2

Dr. SELVA KUMAR S

B.M.S COLLEGE OF ENGINEERING

AGENDA

- NOSQL Storage Architecture
- Performing CRUD operations
- Querying NoSQL stores
- Modifying and Managing NOSQL Data stores
- Indexing and ordering datasets

UNDERSTANDING THE STORAGE ARCHITECTURE

- Column-oriented databases are among the most popular types of non-relational databases.

Publication from Google on big table established beyond a doubt that a cluster of inexpensive hardware can be leveraged to hold huge amounts data, way more than a single machine can hold, and be processed effectively and efficiently within a reasonable timeframe.

Three key themes emerged:

1. Data needs to be stored in a networked filesystem that can expand to multiple machines.

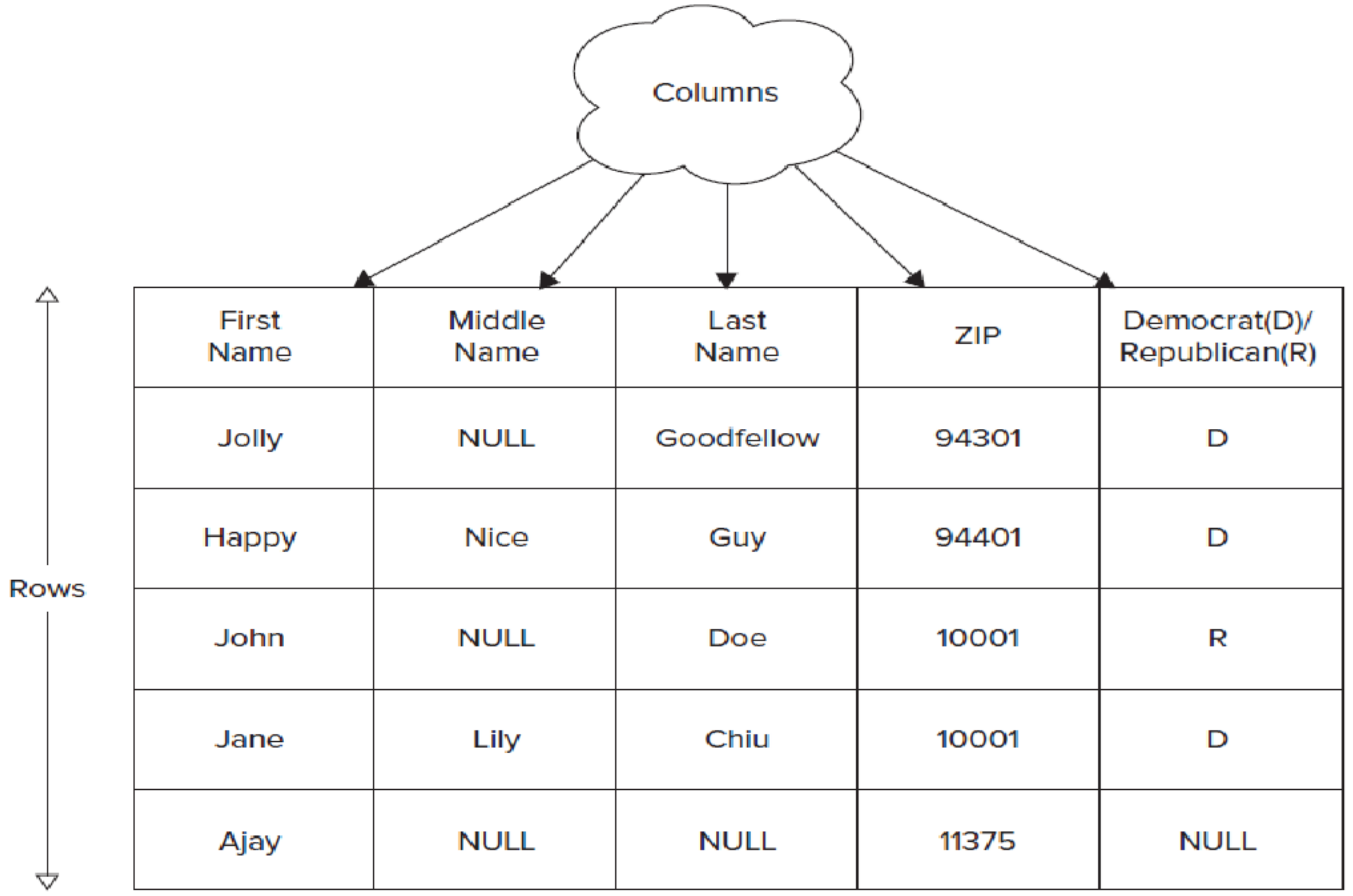
2. Data needs to be stored in a structure that provides more flexibility than the traditional normalized relational database structures.

The storage scheme needs to allow for effective storage of huge amounts of sparse data sets. It needs to accommodate for changing schemas without the necessity of altering the underlying tables.

3. Data needs to be processed in a way that computations on it can be performed in isolated subsets of the data and then combined to generate the desired output.

WORKING WITH COLUMN-ORIENTED DATABASES

- **Using Tables and Columns in Relational Databases**



First Name	Middle Name	Last Name	ZIP	Democrat(D)/ Republican(R)
Jolly	NULL	Goodfellow	94301	D
Happy	Nice	Guy	94401	D
John	NULL	Doe	10001	R
Jane	Lily	Chiu	10001	D
Ajay	NULL	NULL	11375	NULL

FIGURE 4-1

- 1)RDBMS table has a few columns, sometimes tens of them-> Millions of rows could potentially be held in a relational table may bring the data access to a halt, unless special considerations like denormalization are applied.

- 2) As you begin to use your table may need to alter it to hold a few additional attributes

As newer records are stored may have null values for these attributes the existing records.

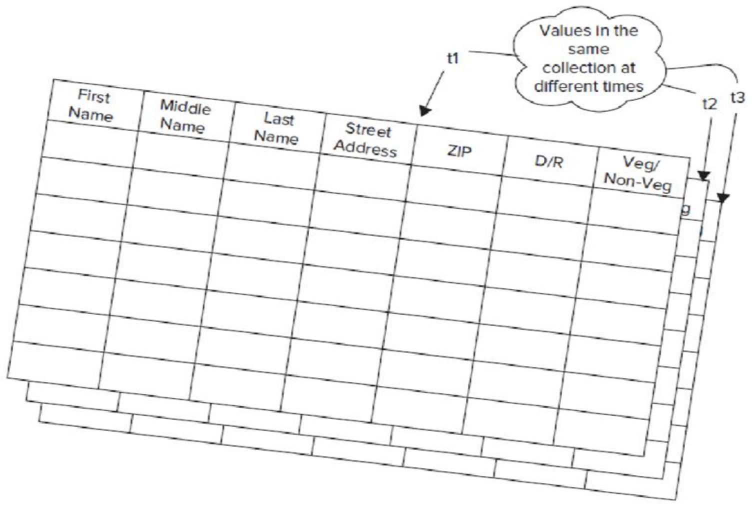
- 3)Keeping greater variety of attributes the likelihood of sparse data sets sets with null in many cells becomes increasingly real.

Consider that this data is evolving and you have to store each version of the cell value as it evolves.

Think of it like a three-dimensional Excel spreadsheet, where the third dimension is time.

Then the values as they evolve through time could be thought of as cell values in multiple spreadsheets put one behind the other in chronological order

Therefore
Altering the table as data evolves,
storing a lot of sparse cells,
and
working through value versions can get complex



CONTRASTING COLUMN DATABASES WITH RDBMS

- First and foremost, a column-oriented database imposes minimal need for upfront schema definition and can easily accommodate newer columns as the data evolves.
- In a typical column-oriented store, you predefine a column-family and not a column.
- A column-family is a set of columns grouped together into a bundle.
- In a column database, a column-family is analogous to a column in an RDBMS.
- Both are typically defined before data is stored in tables and are fairly static in nature.
- Columns in RDBMS define the type of data they can store.
- Column-families have no such limitation; they can contain any number of columns, which can store any type of data, as far as they can be persisted as an array of bytes.

- Each row of a column-oriented database table stores data values in only those columns for which it has valid values.

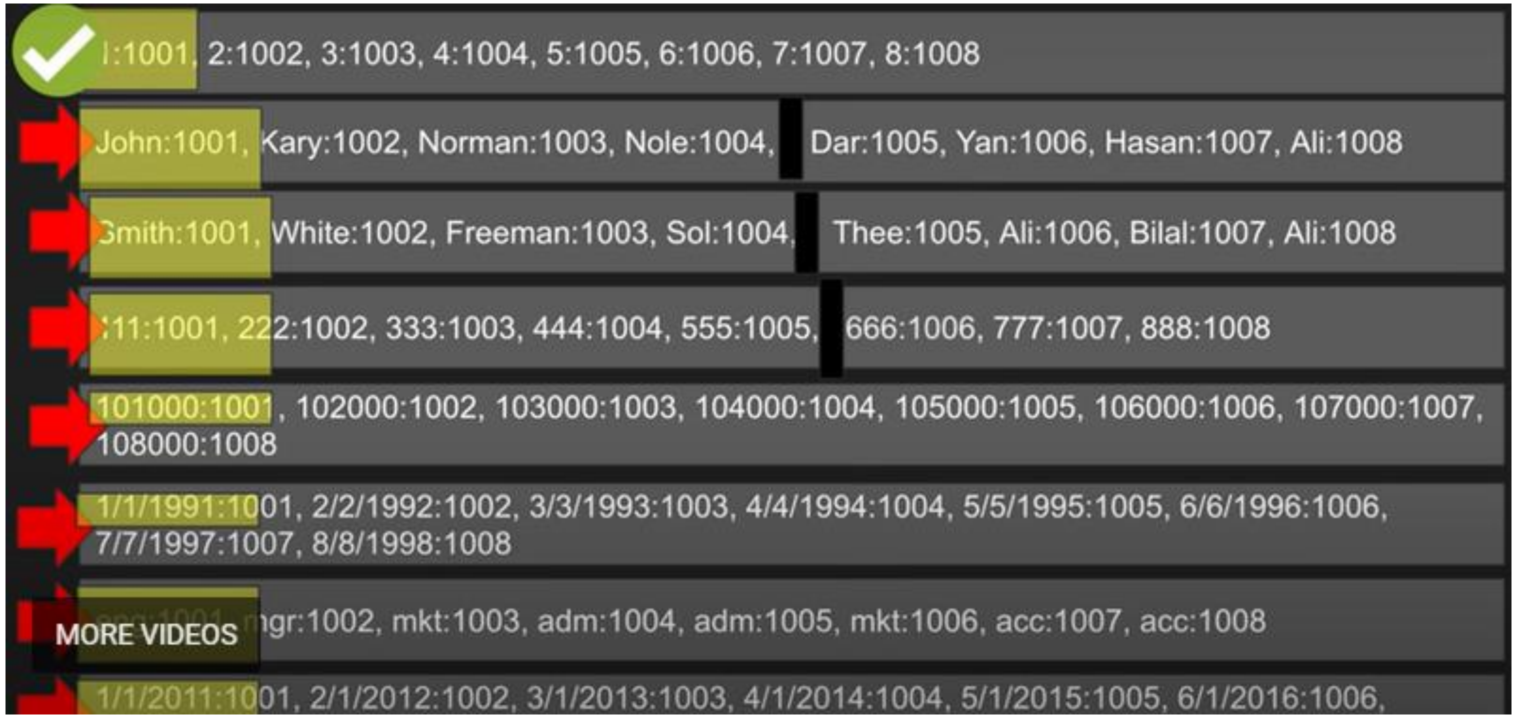
	name	location	preferences
	first name=>"...", last name=>"..."	zip=>"..."	d/r=>"...", veg/non-veg=>"..."

- Continuously evolving data would get stored in a column database as shown in Figure

	time	name	location	preferences
	t9	first name=>"...", last name=>"..."	zip=>"..."	d/r=>"...", veg/non-veg=>"..."
	t8			
	t7			
	t5			

EXAMPLE

- Select * from emp where id=1



1:1001, 2:1002, 3:1003, 4:1004, 5:1005, 6:1006, 7:1007, 8:1008

John:1001, Kary:1002, Norman:1003, Nole:1004, Dar:1005, Yan:1006, Hasan:1007, Ali:1008

Smith:1001, White:1002, Freeman:1003, Sol:1004, Thee:1005, Ali:1006, Bilal:1007, Ali:1008

111:1001, 222:1002, 333:1003, 444:1004, 555:1005, 666:1006, 777:1007, 888:1008

101000:1001, 102000:1002, 103000:1003, 104000:1004, 105000:1005, 106000:1006, 107000:1007, 108000:1008

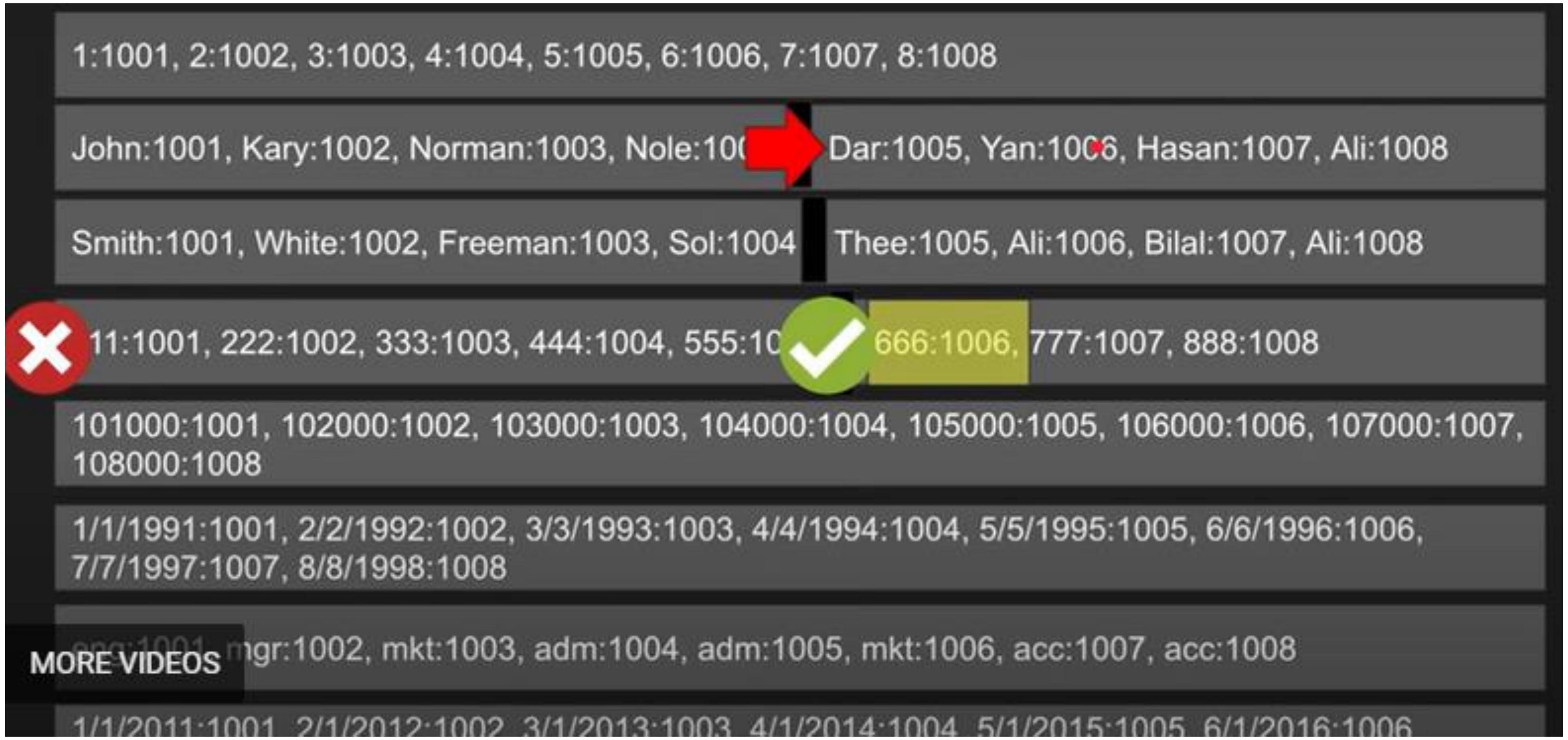
1/1/1991:1001, 2/2/1992:1002, 3/3/1993:1003, 4/4/1994:1004, 5/5/1995:1005, 6/6/1996:1006, 7/7/1997:1007, 8/8/1998:1008

[illegible] mgr:1002, mkt:1003, adm:1004, adm:1005, mkt:1006, acc:1007, acc:1008

1/1/2011:1001, 2/1/2012:1002, 3/1/2013:1003, 4/1/2014:1004, 5/1/2015:1005, 6/1/2016:1006,

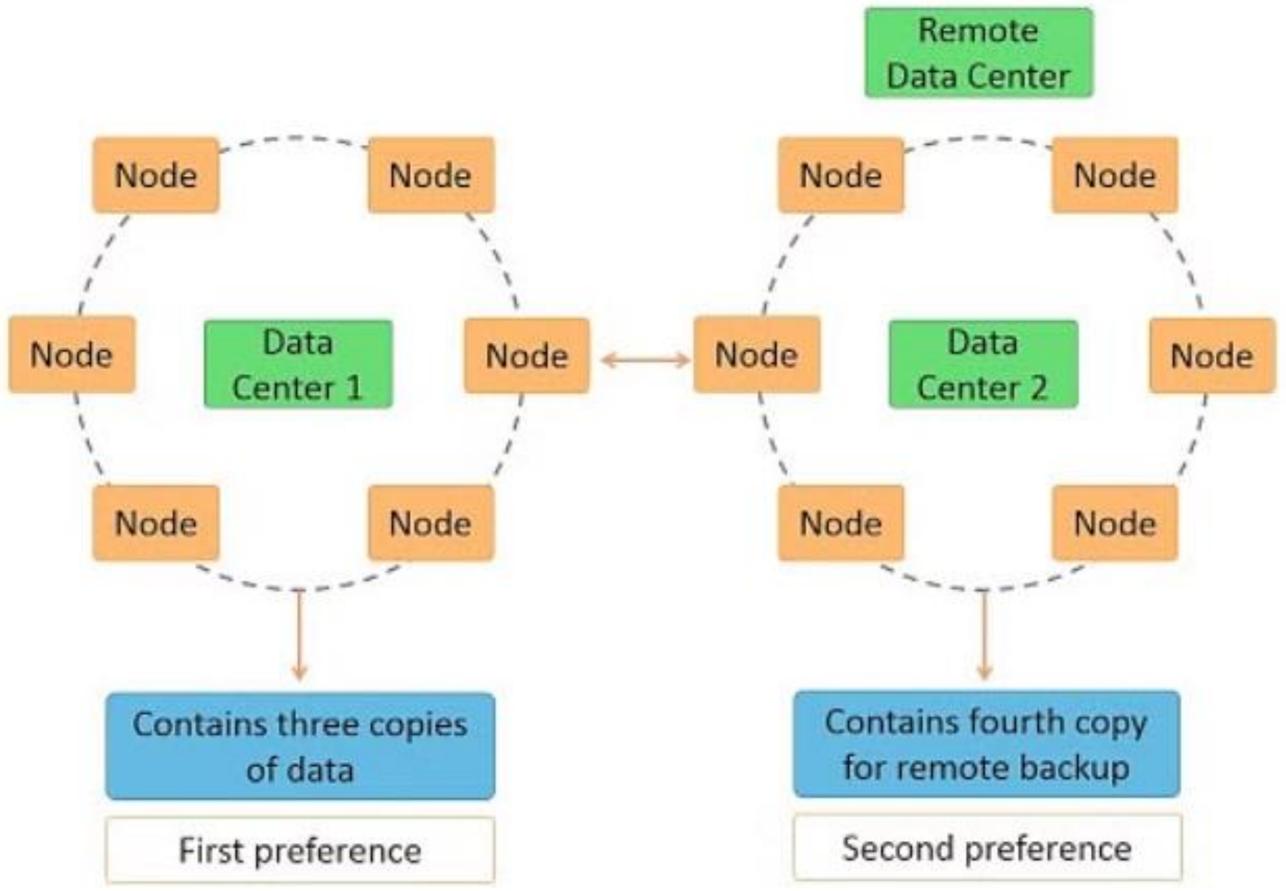
EXAMPLE

- Select first_name from emp where ssn=666



```
1:1001, 2:1002, 3:1003, 4:1004, 5:1005, 6:1006, 7:1007, 8:1008
John:1001, Kary:1002, Norman:1003, Nole:10  Dar:1005, Yan:1006, Hasan:1007, Ali:1008
Smith:1001, White:1002, Freeman:1003, Sol:1004  Thee:1005, Ali:1006, Bilal:1007, Ali:1008
11:1001, 222:1002, 333:1003, 444:1004, 555:10  666:1006, 777:1007, 888:1008
101000:1001, 102000:1002, 103000:1003, 104000:1004, 105000:1005, 106000:1006, 107000:1007,
108000:1008
1/1/1991:1001, 2/2/1992:1002, 3/3/1993:1003, 4/4/1994:1004, 5/5/1995:1005, 6/6/1996:1006,
7/7/1997:1007, 8/8/1998:1008
eng:1001, mgr:1002, mkt:1003, adm:1004, adm:1005, mkt:1006, acc:1007, acc:1008
1/1/2011:1001, 2/1/2012:1002, 3/1/2013:1003, 4/1/2014:1004, 5/1/2015:1005, 6/1/2016:1006
```

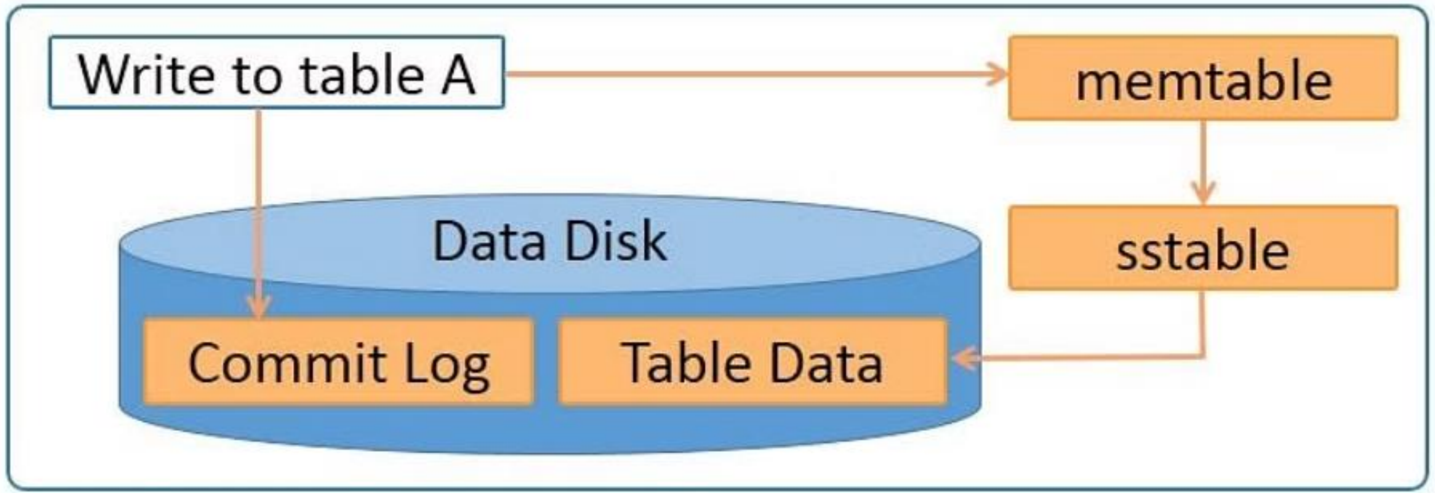
CASSANDRA ARCHITECTURE



CASSANDRA ARCHITECTURE

- Cassandra is designed such that it has no master or slave nodes.
- It has a ring-type architecture, that is, its nodes are logically distributed like a ring.
- Data is automatically distributed across all the nodes.
- Data is replicated across the nodes for redundancy.
- Data is kept in memory and lazily written to the disk.
- Hash values of the keys are used to distribute the data among nodes in the cluster.

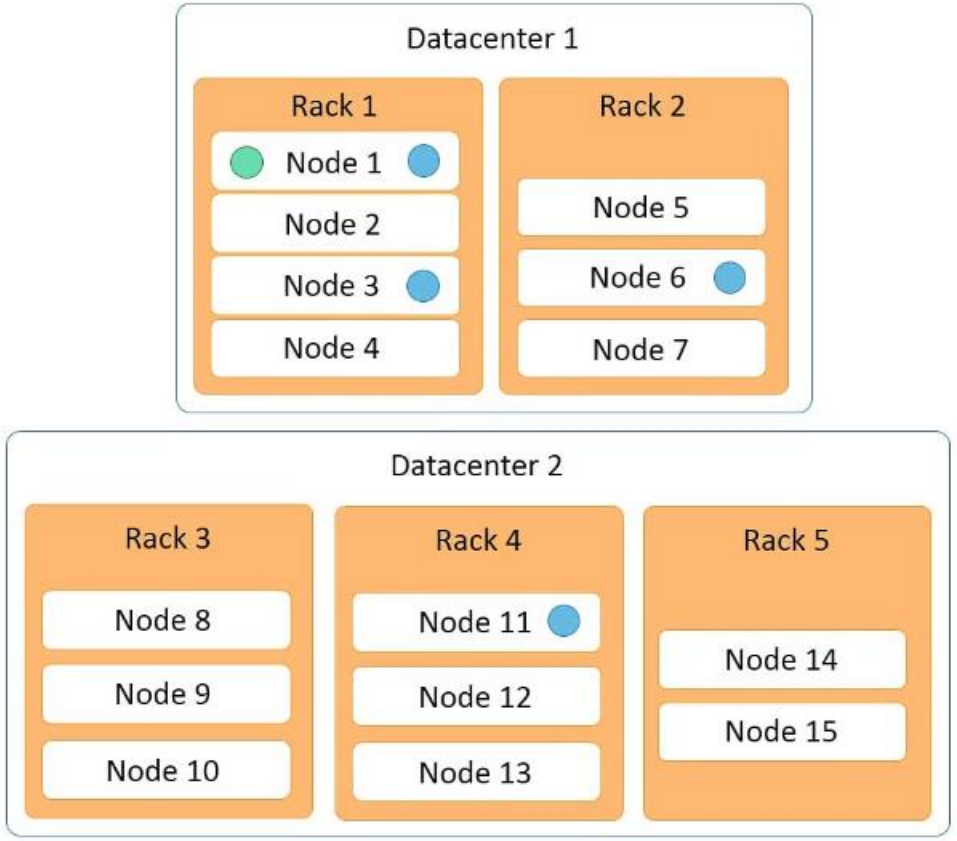
CASSANDRA WRITE PROCESS



CASSANDRA WRITE PROCESS

1. Data is written to a commitlog on disk.
2. The data is sent to a responsible node based on the hash value.
3. Nodes write data to an in-memory table called memtable.
4. From the memtable, data is written to an sstable in memory. Sstable stands for Sorted String table. This has a consolidated data of all the updates to the table.
5. From the sstable, data is updated to the actual table.
6. If the responsible node is down, data will be written to another node identified as tempnode. The tempnode will hold the data temporarily till the responsible node comes alive.

CASSANDRA READ PROCESS



CASSANDRA READ PROCESS

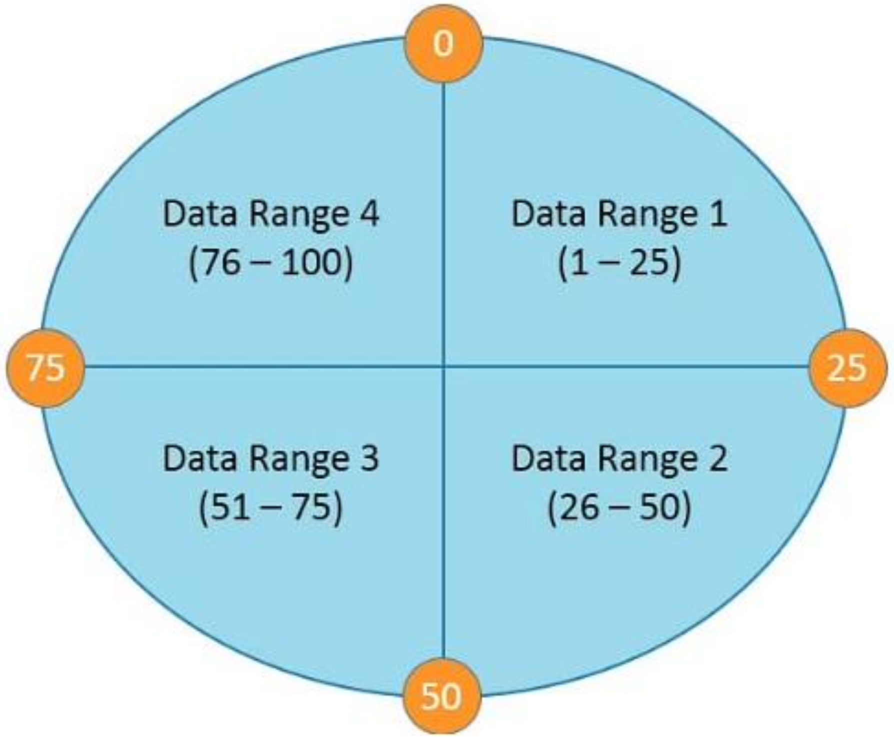
- Data on the same node is given first preference and is considered data local.
- Data on the same rack is given second preference and is considered rack local.
- Data on the same data center is given third preference and is considered data center local.
- Data in a different data center is given the least preference.
- Data in the memtable and sstable is checked first so that the data can be retrieved faster if it is already in memory.

DATA PARTITIONS

- Cassandra performs transparent distribution of data by horizontally partitioning the data in the following manner:
  - A hash value is calculated based on the primary key of the data.
  - The hash value of the key is mapped to a node in the cluster
  - The first copy of the data is stored on that node.
  - The distribution is transparent as you can both calculate the hash value and determine where a particular row will be stored.

DATA PARTITIONS

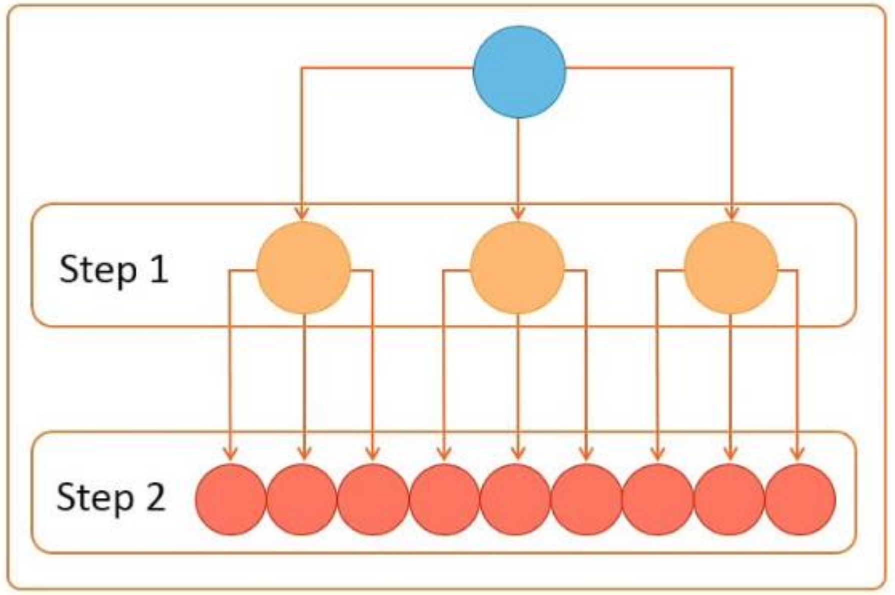
- The following diagram depicts a four node cluster with token values of 0, 25, 50 and 75.



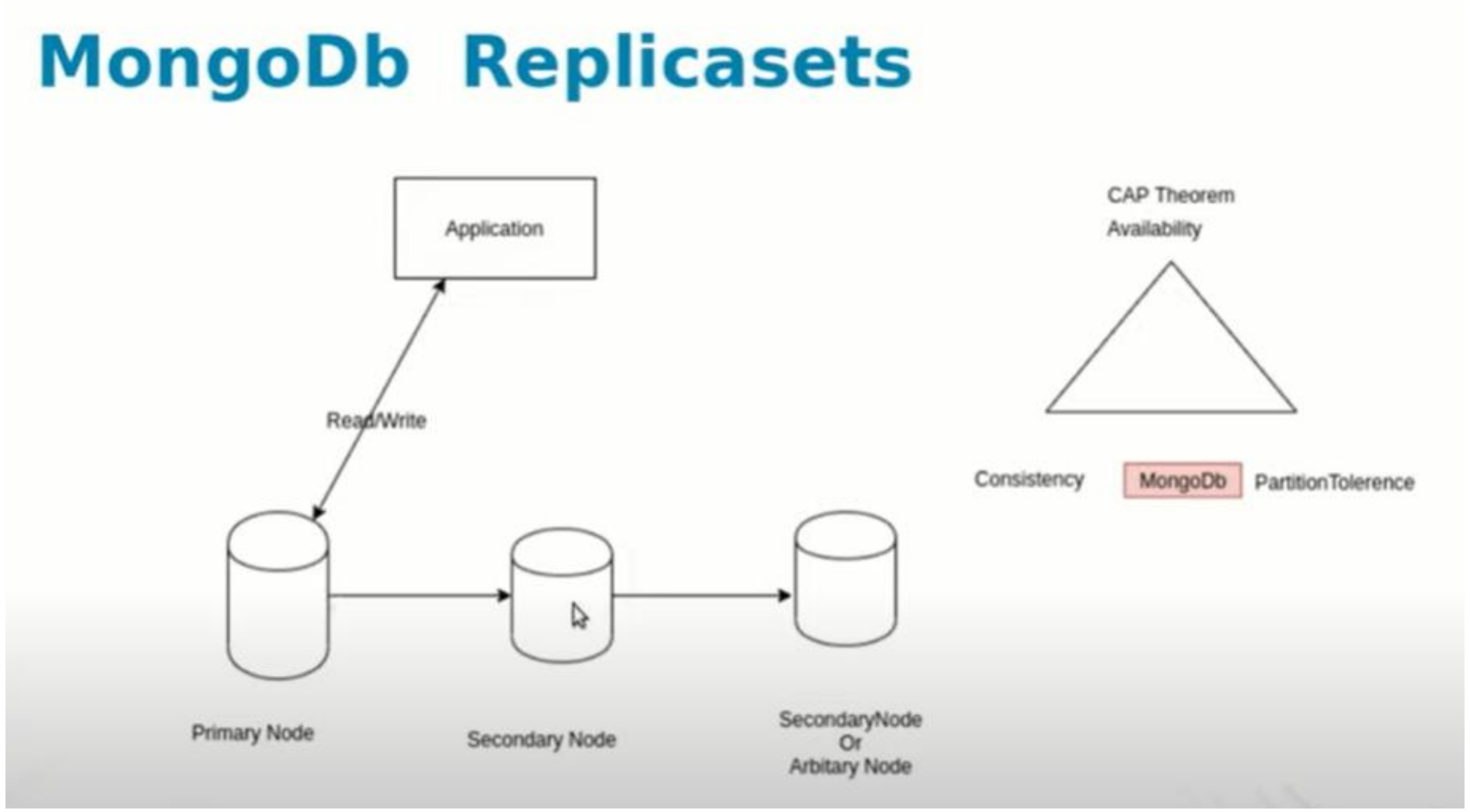
GOSSIP PROTOCOL

- Cassandra uses a gossip protocol to communicate with nodes in a cluster.
- It is an inter-node communication mechanism similar to the heartbeat protocol in Hadoop.
- The gossip process runs periodically on each node and exchanges state information with three other nodes in the cluster.
- Eventually, information is propagated

  to all cluster nodes.



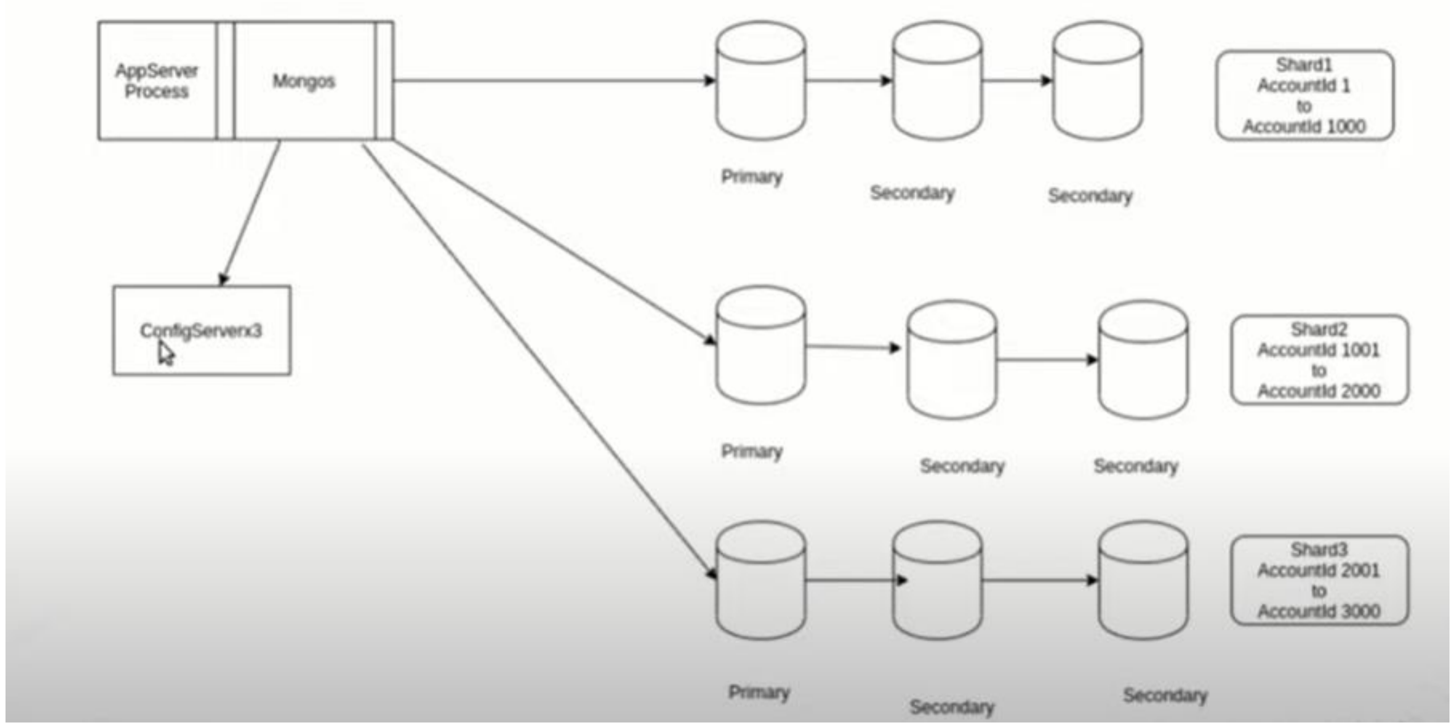
MONGODB ARCHITECTURE



REPLICATION IN MONGODB

- MongoDB achieves replication using the concept replica sets.
- A replica set is a group of mongod instances that host the same data set.
- One of the nodes is selected as the primary or main node.
- The primary node receives all the operations from the user and the secondaries are updated from the primary one by using the same operation to maintain consistency.
- If the primary node goes down, one of the secondary nodes is selected as the primary node and the operations are carried forward.
- When the fallen node recovers, it joins the cluster as the secondary nodes.
- We can control our cluster of mongo instances using Mongo Atlas.

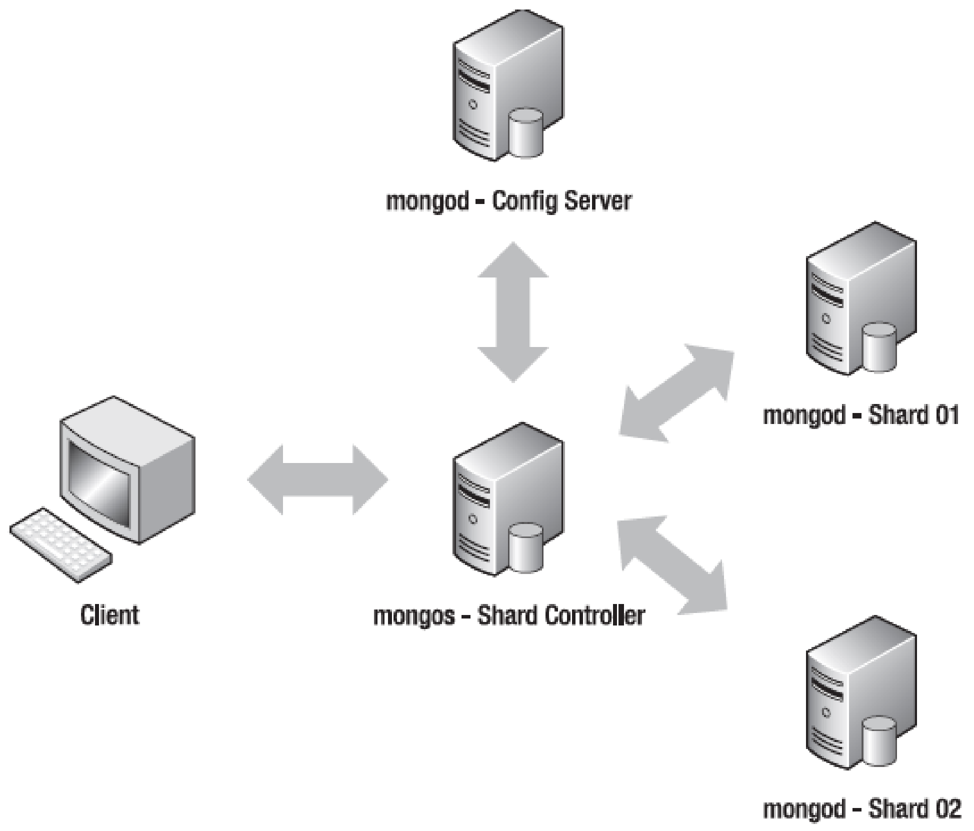
MongoDb Sharding


AppServer Process
Mongos
ConfigServerx3
Primary
Secondary
Secondary
Shard1 AccountId 1 to AccountId 1000
Primary
Secondary
Secondary
Shard2 AccountId 1001 to AccountId 2000
Primary
Secondary
Secondary
Shard3 AccountId 2001 to AccountId 3000

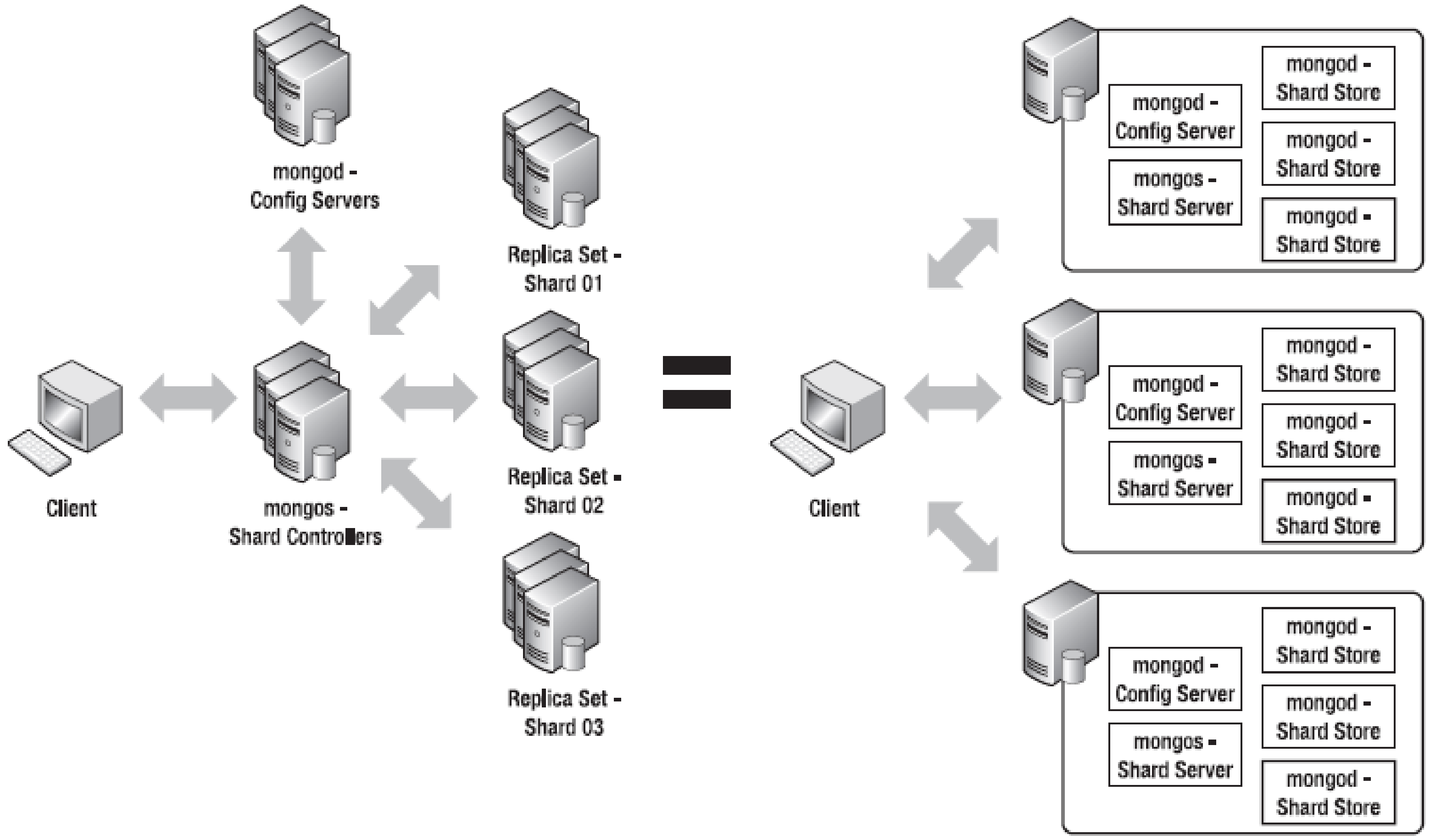
SHARDING IN MONGODB

- Sharding is used by MongoDB to store data across multiple machines.
- It uses horizontal scaling to add more machines to distribute data and operation with respect to the growth of load and demand.
- Sharding arrangement in MongoDB has mainly three components:
- 1. Shards or replica sets
- 2. Configuration Servers
- 3. Query Router

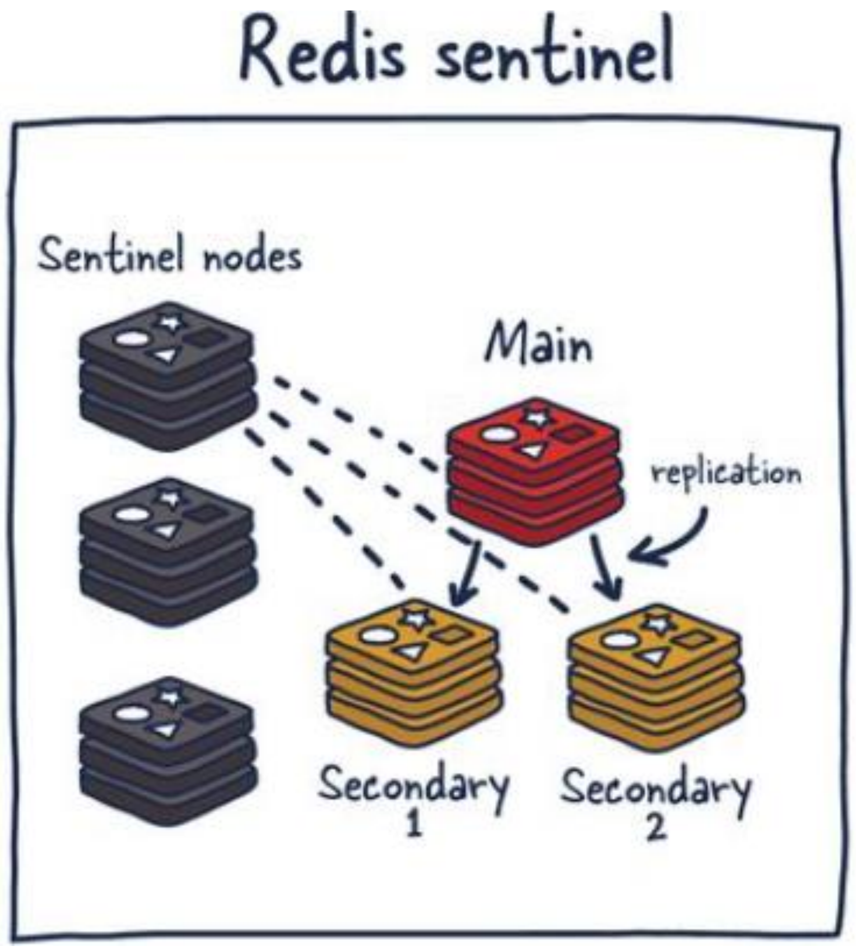
SIMPLE SHARDING SETUP



A REDUNDANT SHARDING CONFIGURATION

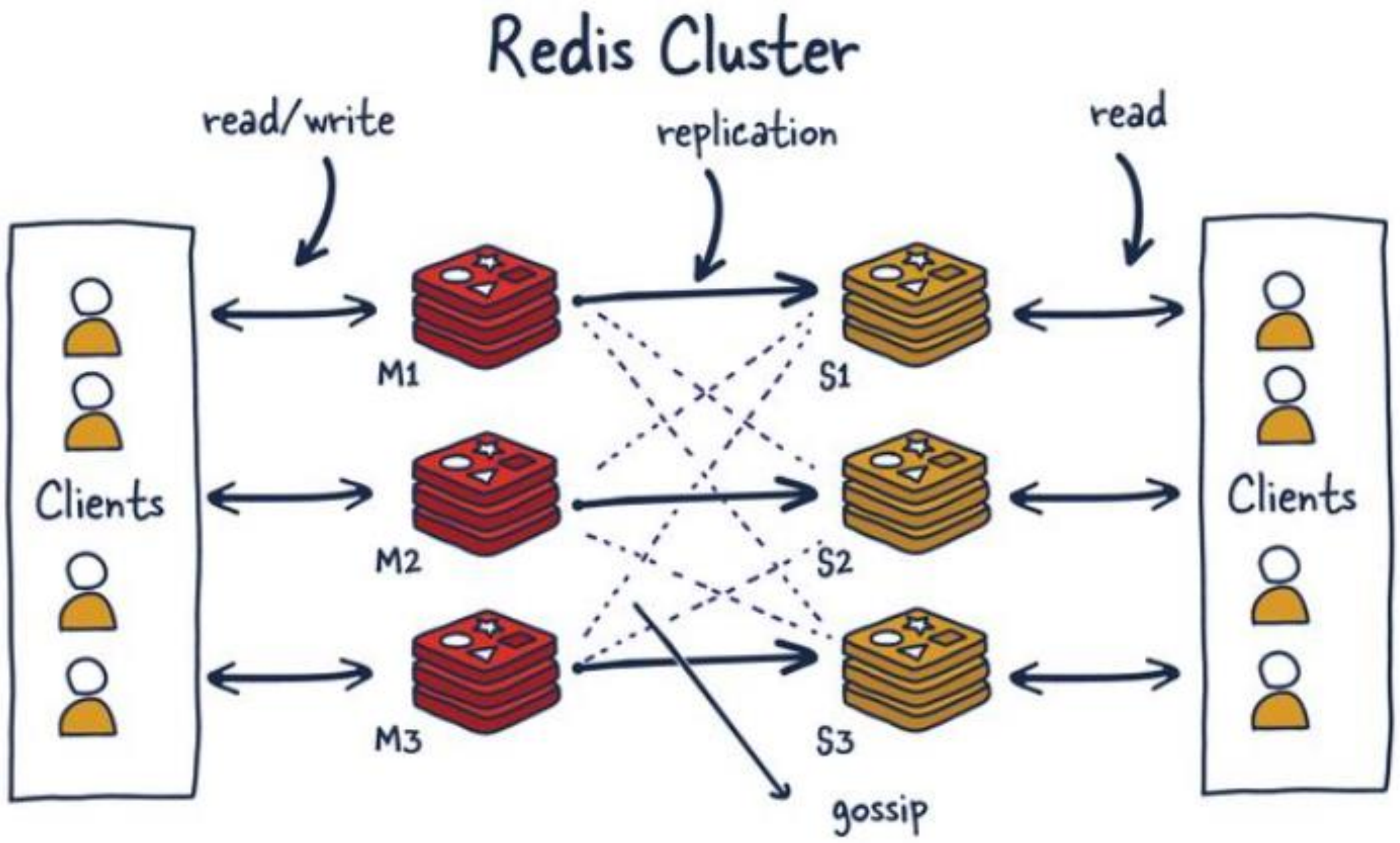


REDIS ARCHITECTURE

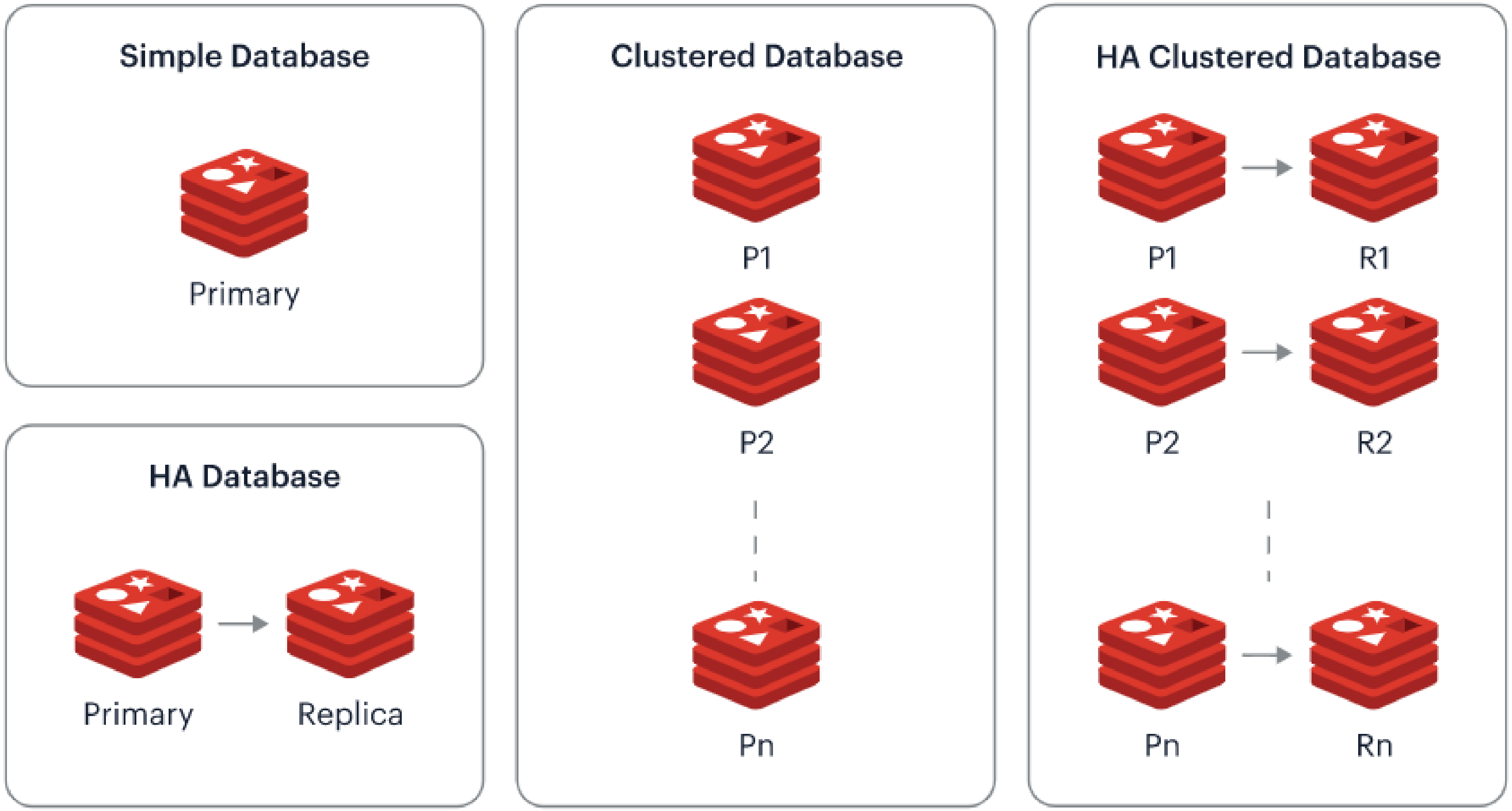


SENTINELS RESPONSIBILITIES

1. Monitoring — ensuring main and secondary instances are working as expected.
2. Notification — notify system admins about occurrences in the Redis instances.
3. Failover management — Sentinel nodes can start a failover process if the primary instance isn't available and enough (quorum of) nodes agree that is true.
4. Configuration management — Sentinel nodes also serve as a point of discovery of the current main Redis instance.

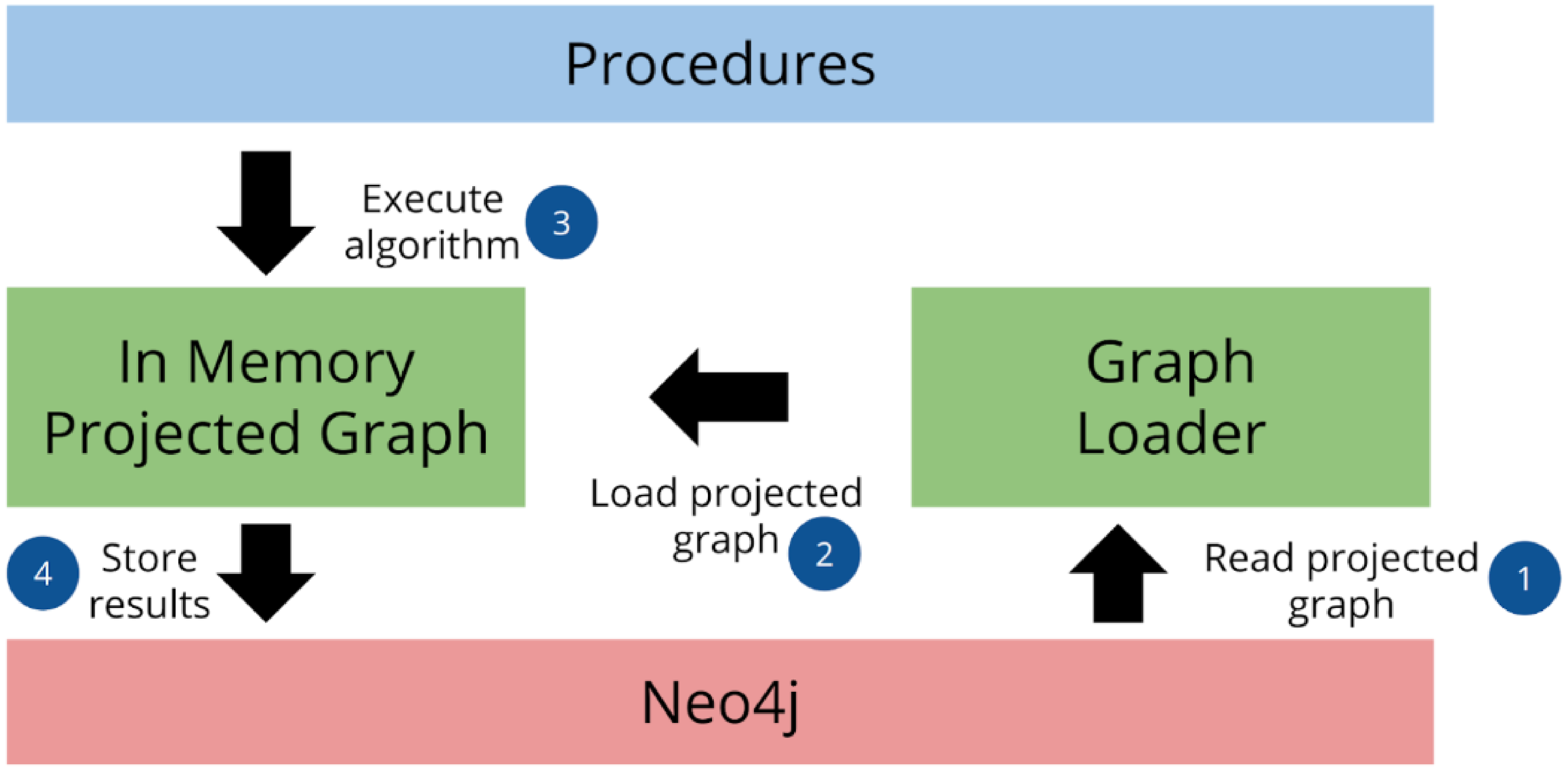

Redis Cluster
read/write
replication
read
Clients
M1
M2
M3
S1
S2
S3
Clients
gossip

REDIS ENTERPRISE CLUSTER NODE



- At any given time, a Redis Enterprise cluster node can include between zero and a few hundred Redis databases in one of the following types:
  - A simple database, i.e. a single primary shard
  - A highly available (HA) database, i.e. a pair of primary and replica shards
  - A clustered database, which contains multiple primary shards, each managing a subset of the dataset (or in Redis terms, a different range of "hash-slots")
  - An HA clustered database, i.e. multiple pairs of primary/replica shards

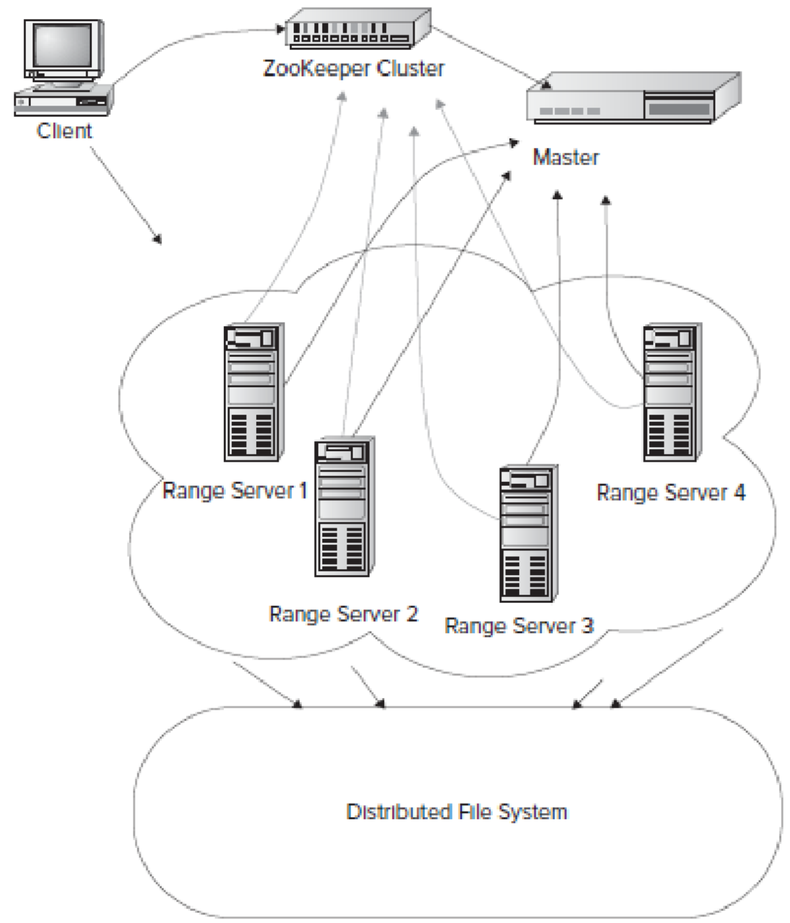
NEO4J ARCHITECTURE

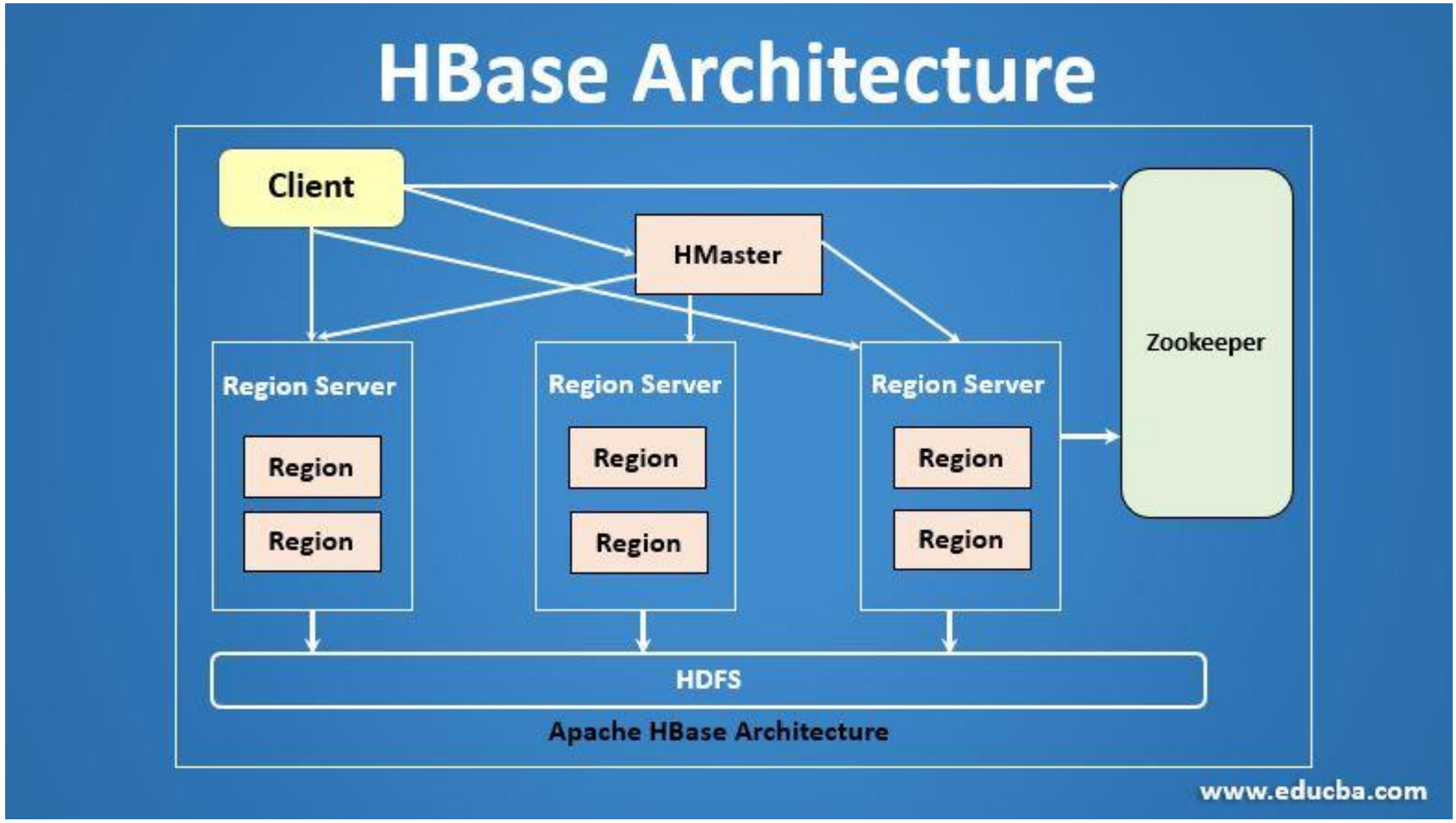


GRAPH DATA SCIENCE

1. Reading the graph data from Neo4j Database
2. Loading (projecting) the data into an in-memory graph
3. Running an algorithm on a projected graph
4. Writing the results back to Neo4j Database (if the algorithm runs in write mode)

HBASE DISTRIBUTED STORAGE ARCHITECTURE





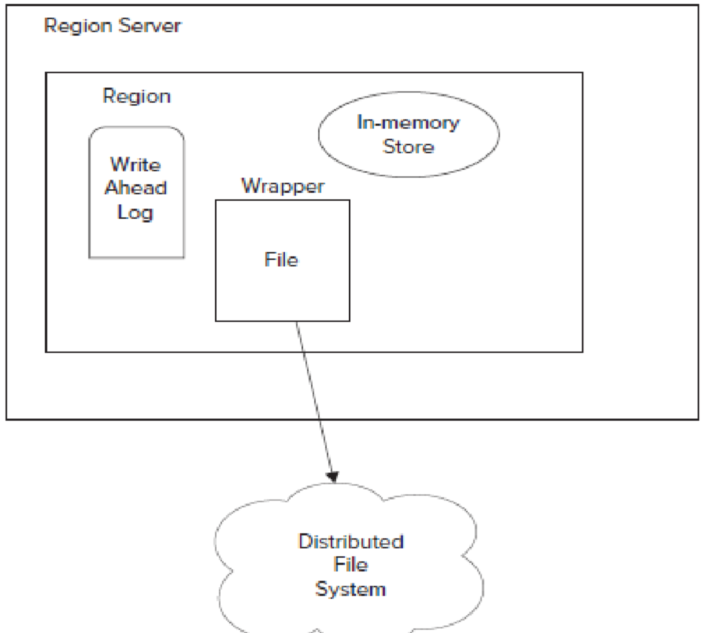
HBASE?

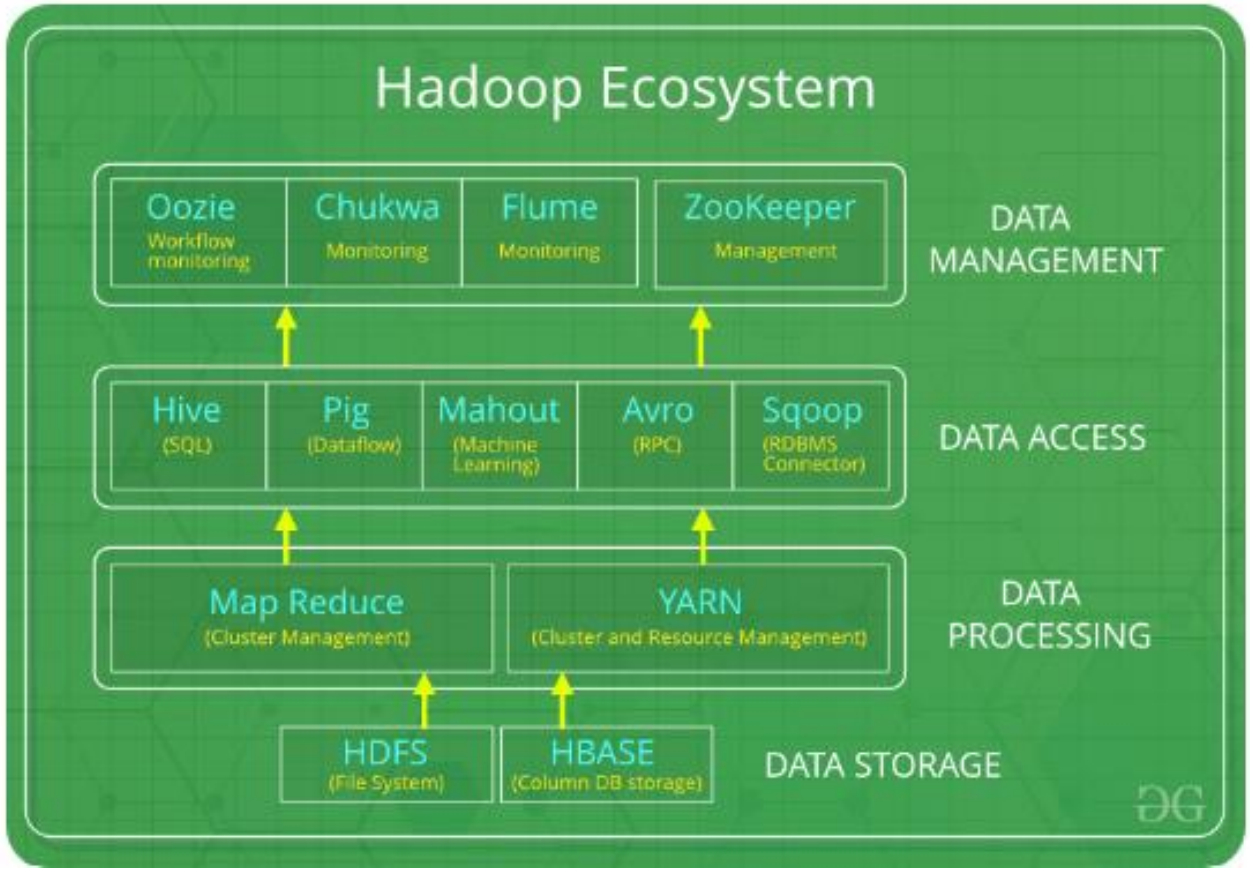
- HBase is a distributed column-oriented database built on top of the Hadoop file system.
- It is an open-source project and is horizontally scalable.
- HBase is a data model that is similar to Google’s big table designed to provide quick random access to huge amounts of structured data.
- It leverages the fault tolerance provided by the Hadoop File System (HDFS).
- It is a part of the Hadoop ecosystem that provides random real-time read/write access to data in the Hadoop File System.

- HBase deployment adheres to **a master-worker pattern**.
- Therefore, there is usually a master and a **set of workers**, commonly known as **region servers.**
- When HBase starts, the **master allocates a set of regions to a region server.**
- Each **region stores an ordered set of rows**, where each row is identified by a unique **row-key.**
- As the number of **rows stored in a region grows** in size beyond a configured threshold, the region is split into two and rows are divided between the two new ranges.

- HBase stores **columns in a column-family together**.
- Therefore, each **region maintains a separate store for each column-family**
- Each **store in turn maps to a physical file** that is stored in the underlying distributed filesystem.
- For each store, HBase **abstracts access to the underlying filesystem with the help of a thin wrapper that acts as the intermediary between the store and the underlying physical file.**

- Each region has an in-memory store, or cache, and a write-ahead-log (WAL)
- When data is written to a region, it's first written to the write-ahead-log.
- Soon afterwards, it's written to the region's in-memory store.
- If thein-memory store is full, data is flushed to disk and persisted in the underlying distributed storage.

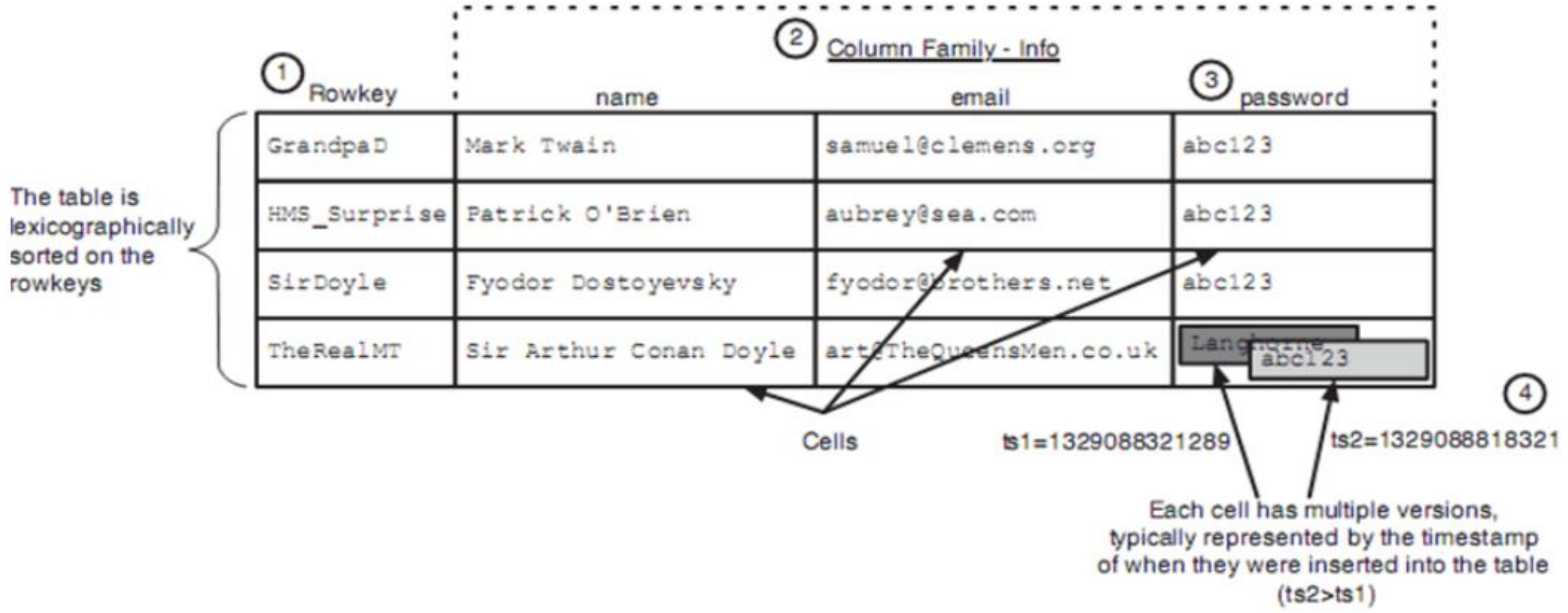



Hadoop Ecosystem
Oozie
Workflow monitoring
Chukwa
Monitoring
Flume
Monitoring
ZooKeeper
Management
DATA MANAGEMENT
Hive
(SQL)
Pig
(Dataflow)
Mahout
(Machine Learning)
Avro
(RPC)
Sqoop
(RDBMS Connector)
DATA ACCESS
Map Reduce
(Cluster Management)
YARN
(Cluster and Resource Management)
DATA PROCESSING
HDFS
(File System)
HBASE
(Column DB storage)
DATA STORAGE

HADOOP FRAMEWORK

- Hadoop is a framework that enables processing of large data sets which reside in the form of clusters.
- Hadoop is made up of several modules that are supported by a large ecosystem of technologies.
- *Hadoop Ecosystem* is a platform or a suite which provides various services to solve the big data problems.
- There are *four major elements of Hadoop* i.e. **HDFS, MapReduce, YARN, and Hadoop Common.**

HBASE DATA MODEL



② Column Family - Info
① Rowkey
③
④
The table is lexicographically sorted on the rowkeys
Cells
ts1=1329088321289
ts2=1329088818321
Each cell has multiple versions, typically represented by the timestamp of when they were inserted into the table (ts2>ts1)

Rowkey	name	email	password
GrandpaD	Mark Twain	samuel@clemens.org	abc123
HMS_Surprise	Patrick O'Brien	aubrey@sea.com	abc123
SirDoyle	Fyodor Dostoyevsky	fyodor@brothers.net	abc123
TheRealMT	Sir Arthur Conan Doyle	art@TheQueensMen.co.uk	Langhorne / abc123

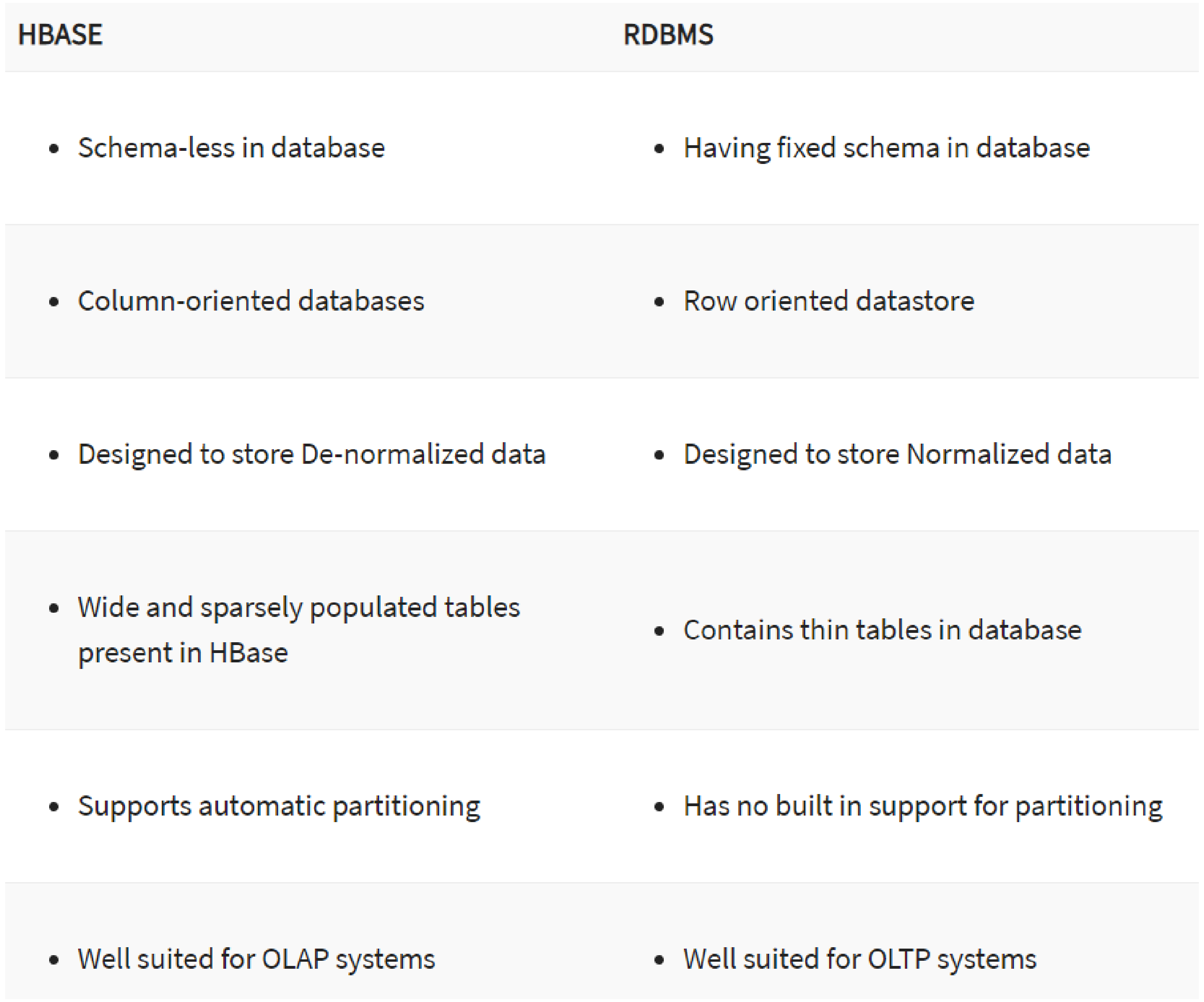
HOW IS HBASE DIFFERENT FROM OTHER NOSQL MODELS

- HBase stores data in the form of key/value pairs in a columnar model.
- In this model, all the columns are grouped together as Column families.
- HBase provides a flexible data model and low latency access to small amounts of data stored in large data sets.
- HBase on top of Hadoop will increase the throughput and performance of distributed cluster set up.
- In turn, it provides faster random reads and writes operations.

USE CASES

DataBase Type Based on Feature	Example of Database	Use case (When to Use)
Key/ Value	Redis, MemcacheDB	Caching, Queue-ing, Distributing information
Column-Oriented	Cassandra, HBase	Scaling, Keeping Unstructured, non-volatile
Document-Oriented	MongoDB, Couchbase	Nested Information, JavaScript friendly
Graph-Based	OrientDB, Neo4J	Handling Complex relational information. Modeling and Handling classification.

HBASE VS. RDBMS



HBASE	RDBMS
• Schema-less in database	• Having fixed schema in database
• Column-oriented databases	• Row oriented datastore
• Designed to store De-normalized data	• Designed to store Normalized data
• Wide and sparsely populated tables present in HBase	• Contains thin tables in database
• Supports automatic partitioning	• Has no built in support for partitioning
• Well suited for OLAP systems	• Well suited for OLTP systems

HBASE DDL COMMANDS

- **create** - Creates a table.
- **list** - Lists all the tables in HBase.
- **disable** - Disables a table.
- **is_disabled** - Verifies whether a table is disabled.
- **enable** - Enables a table.
- **is_enabled** - Verifies whether a table is enabled.
- **describe** - Provides the description of a table.
- **alter** - Alters a table.
- **exists** - Verifies whether a table exists.
- **drop** - Drops a table from HBase.
- **drop_all** - Drops the tables matching the ‘regex’ given in the command.

HBASE DML COMMANDS

- **put** - Puts a cell value at a specified column in a specified row in a particular table.
- **get** - Fetches the contents of row or a cell.
- **delete** - Deletes a cell value in a table.
- **deleteall** - Deletes all the cells in a given row.
- **scan** - Scans and returns the table data.
- **count** - Counts and returns the number of rows in a table.
- **truncate** - Disables, drops, and recreates a specified table.

EXAMPLE

- hbase>create 'emp', 'personal data', 'professional data’
- hbase>list
-   Table
-    emp
- hbase>Scan ‘emp’
-      1 column = personal data:city, timestamp = 1417516501, value = hyderabad
-      1 column = personal data:name, timestamp = 1417525058, value = ramu
- hbase>alter 'emp', NAME ⇒ 'personal data', VERSIONS ⇒ 5
- hbase> alter ‘ table name ’, ‘delete’ ⇒ ‘ column family ’   // Deleting a column family

EXAMPLE CONTD..

- hbase> put 'selva', 'r1', 'c1', 'value, 10
- hbase> put 'selva', 'r1', 'c1', 'value, 15
- hbase> put 'selva', 'r1', 'c1', 'value, 20

- hbase> get 'selva', 'r1', 'c1’
- hbase> get 'selva', 'r1’
- hbase>get 'selva', 'r1', {TIMERANGE => [ts1, ts2]}
- hbase>get 'selva', 'r1', {COLUMN => ['c1', 'c2', 'c3’]}

- hbase> delete 'selva', 'r1', 'c1'
- hbase> deleteall 'selva', 'c1'

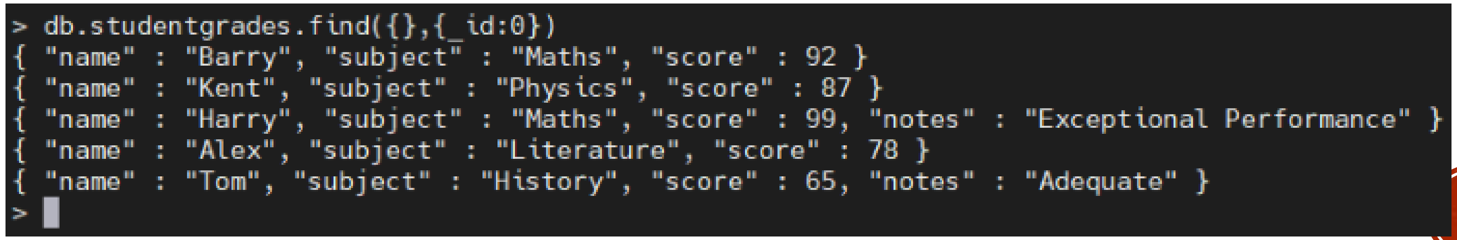
INDEXING IN MONGODB

- MongoDB uses indexing in order to make the query processing more efficient.
- If there is no indexing, then the MongoDB must scan every document in the collection and retrieve only those documents that match the query.
- Indexes are special data structures that store a small part of the Collection’s data in a way that can be queried easily.
- The indexes are order by the value of the field specified in the index.
- In MongoDB, querying without indexes is called a collection scan. A collection scan will:

- Result in various performance bottlenecks
- Significantly slow down your application

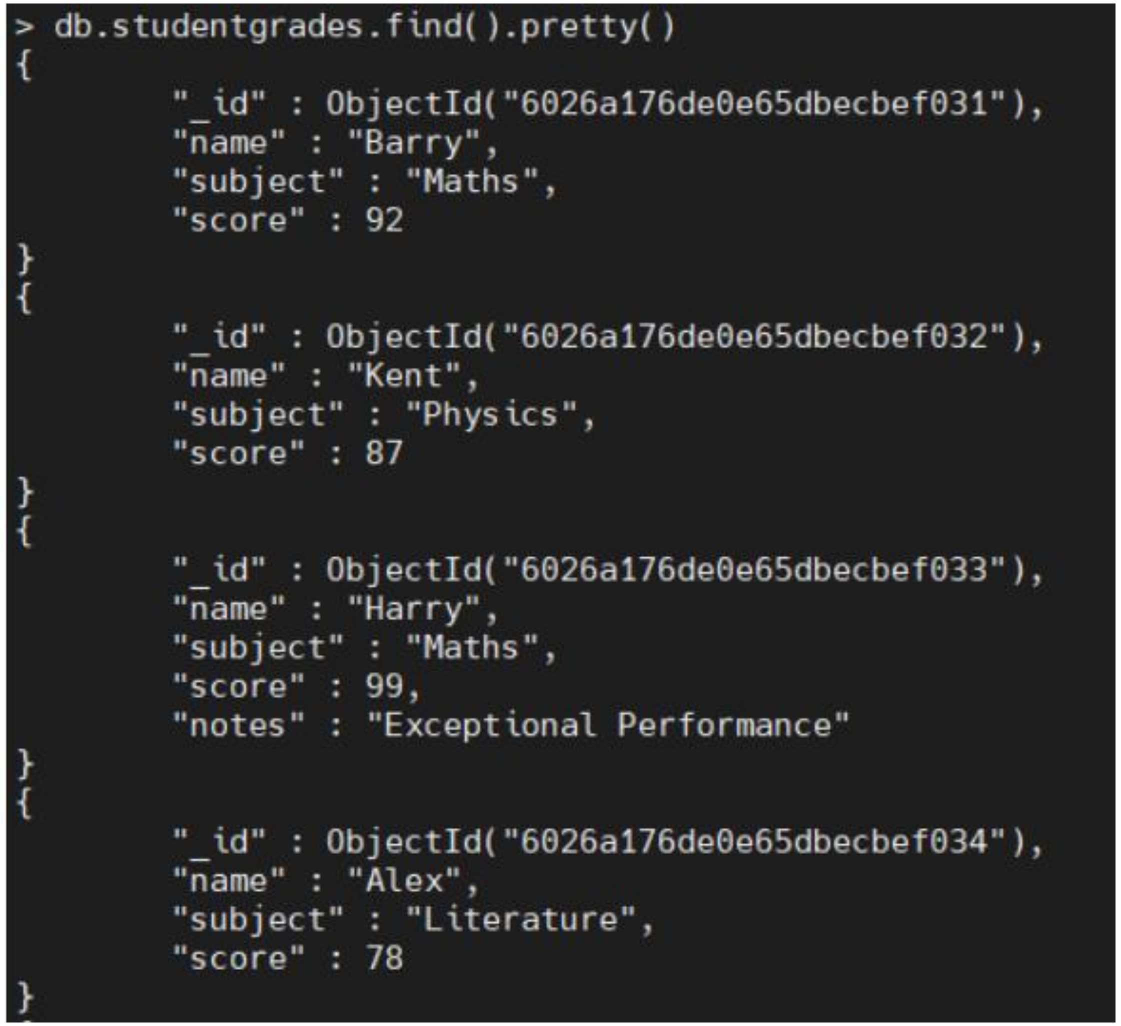
EXAMPLE

- use students
- db.createCollection("studentgrades")
- db.studentgrades.insertMany(
-     [
-         {name: "Barry", subject: "Maths", score: 92},
-         {name: "Kent", subject: "Physics", score: 87},
-         {name: "Harry", subject: "Maths", score: 99, notes: "Exceptional Performance"},
-         {name: "Alex", subject: "Literature", score: 78},
-         {name: "Tom", subject: "History", score: 65, notes: "Adequate"}
-     ]
-)
- db.studentgrades.find({},{_id:0})



```
> db.studentgrades.find({},{_id:0})
{ "name" : "Barry", "subject" : "Maths", "score" : 92 }
{ "name" : "Kent", "subject" : "Physics", "score" : 87 }
{ "name" : "Harry", "subject" : "Maths", "score" : 99, "notes" : "Exceptional Performance" }
{ "name" : "Alex", "subject" : "Literature", "score" : 78 }
{ "name" : "Tom", "subject" : "History", "score" : 65, "notes" : "Adequate" }
> 
```

DEFAULT _ID INDEX



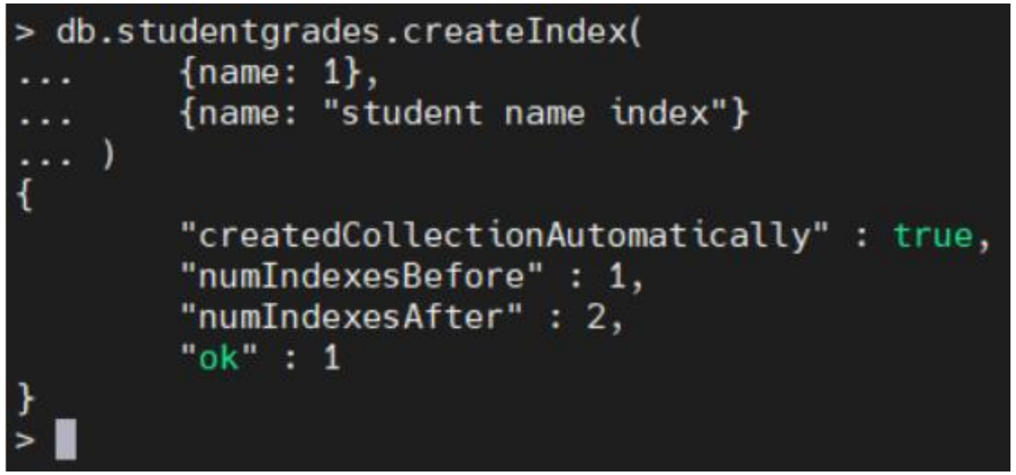
```
> db.studentgrades.find().pretty()
{
        "_id" : ObjectId("6026a176de0e65dbecbef031"),
        "name" : "Barry",
        "subject" : "Maths",
        "score" : 92
}
{
        "_id" : ObjectId("6026a176de0e65dbecbef032"),
        "name" : "Kent",
        "subject" : "Physics",
        "score" : 87
}
{
        "_id" : ObjectId("6026a176de0e65dbecbef033"),
        "name" : "Harry",
        "subject" : "Maths",
        "score" : 99,
        "notes" : "Exceptional Performance"
}
{
        "_id" : ObjectId("6026a176de0e65dbecbef034"),
        "name" : "Alex",
        "subject" : "Literature",
        "score" : 78
}
```

CREATE INDEX METHOD

- db.<collection>.createIndex(<Key and Index Type>, <Options>)

- When creating an index, you need to define the field to be indexed
- The direction of the key (1 or -1) to indicate ascending or descending order.

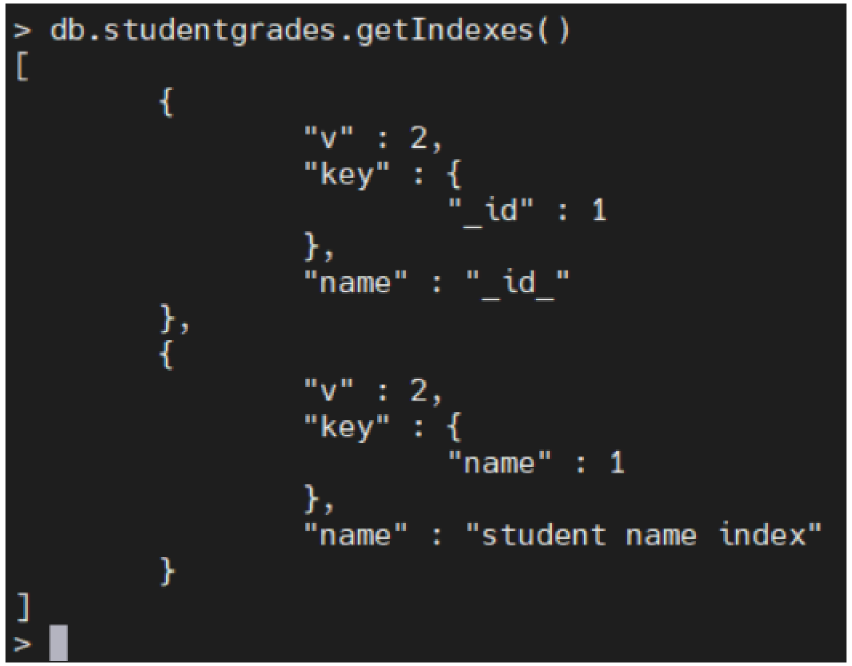
- db.studentgrades.createIndex(
- {name: 1},
- {name: "student name index"}
-)



```
> db.studentgrades.createIndex(
...         {name: 1},
...         {name: "student name index"}
... )
{
        "createdCollectionAutomatically" : true,
        "numIndexesBefore" : 1,
        "numIndexesAfter" : 2,
        "ok" : 1
}
> 
```

GET INDEXES METHOD

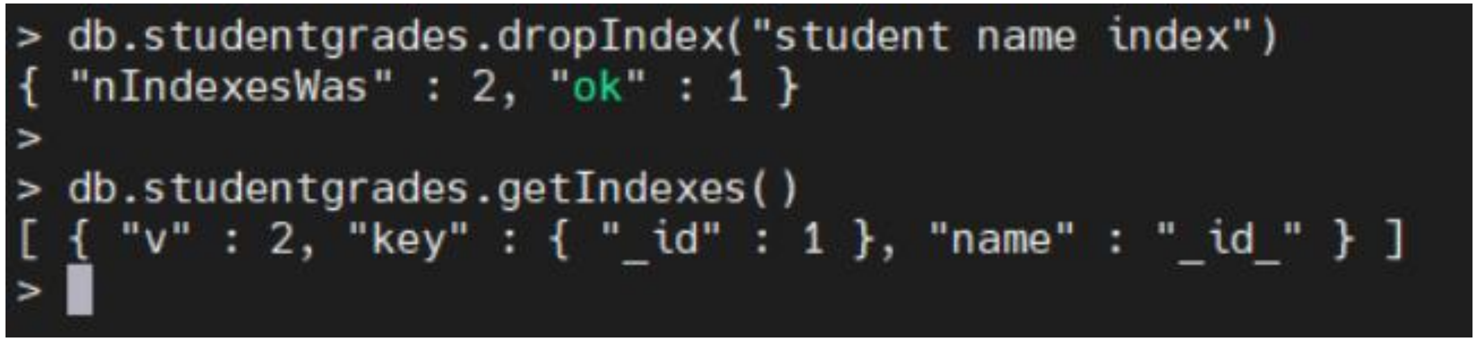
- db.<collection>.getIndexes()
- db.studentgrades.getIndexes()



```
> db.studentgrades.getIndexes()
[
        {
                "v" : 2,
                "key" : {
                        "_id" : 1
                },
                "name" : "_id_"
        },
        {
                "v" : 2,
                "key" : {
                        "name" : 1
                },
                "name" : "student name index"
        }
]
> 
```

DROP INDEX METHOD

- db.<collection>.dropIndex(<Index Name / Field Name>)
- db.studentgrades.dropIndex("student name index")



```
> db.studentgrades.dropIndex("student name index")
{ "nIndexesWas" : 2, "ok" : 1 }
>
> db.studentgrades.getIndexes()
[ { "v" : 2, "key" : { "_id" : 1 }, "name" : "_id_" } ]
>
```

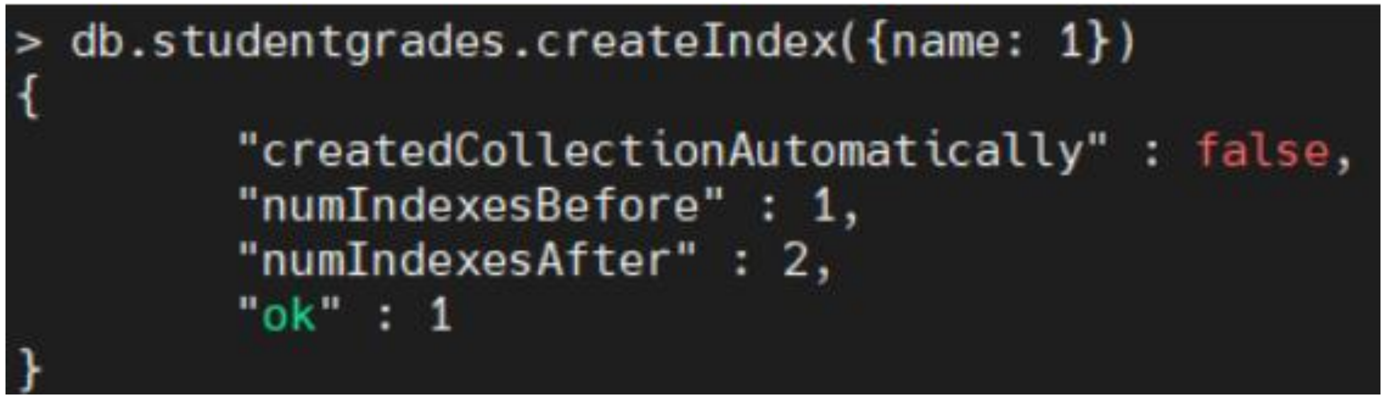
- db.studentgrades.dropIndexes()

MONGODB INDEX TYPES

- Single field index
- Compound index
- Multikey index

SINGLE FIELD INDEX

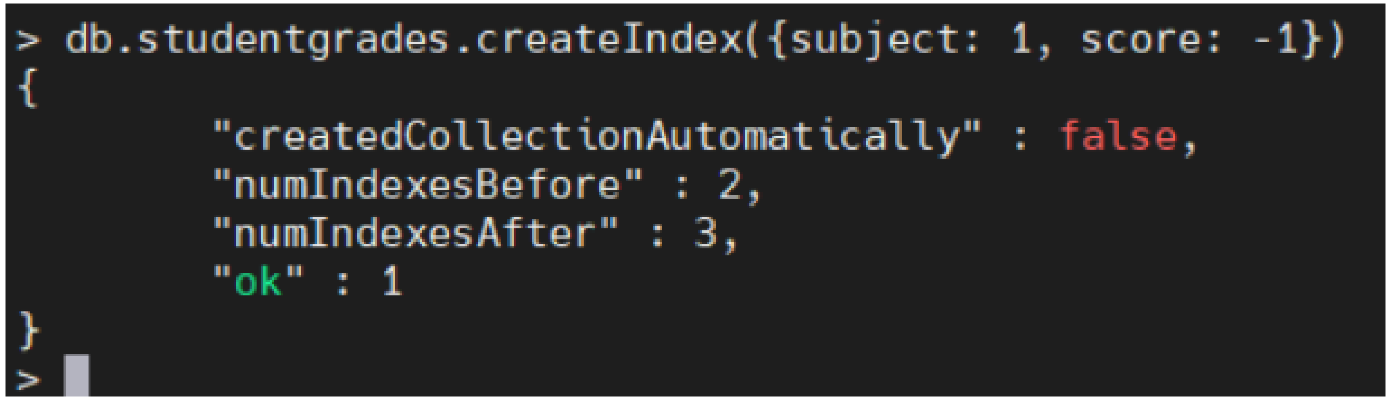
- db.studentgrades.createIndex({name: 1})



```
> db.studentgrades.createIndex({name: 1})
{
        "createdCollectionAutomatically" : false,
        "numIndexesBefore" : 1,
        "numIndexesAfter" : 2,
        "ok" : 1
}
```

COMPOUND INDEX

- db.studentgrades.createIndex({subject: 1, score: -1})



```
> db.studentgrades.createIndex({subject: 1, score: -1})
{
        "createdCollectionAutomatically" : false,
        "numIndexesBefore" : 2,
        "numIndexesAfter" : 3,
        "ok" : 1
}
>
```

MULTIKEY INDEX

- MongoDB supports indexing array fields.
- These multikey indexes enable users to query documents using the elements within the array.
- MongoDB will automatically create a multikey index when encountered with an array field without requiring the user to explicitly define the multikey type.
- db.createCollection("studentperformance")
- db.studentperformance.insertMany(
- [
- {name: "Barry", school: "ABC Academy", grades: [85, 75, 90, 99] },
- {name: "Kent", school: "FX High School", grades: [74, 66, 45, 67]},
- {name: "Alex", school: "XYZ High", grades: [80, 78, 71, 89]},
-]
-)
- db.studentperformance.find({},{_id:0}).pretty()

INDEXING IN APACHE CASSANDRA

- Built-in indexes are the best option on a table which having many rows and that rows contain the indexed value.
- In a particular column which column having more unique values in that case we can used indexing.
- Table which more overhead due to several reason like column having more entries then in that case we can used indexing.

CREATE INDEX

- Command ‘Create index’ creates an index on the column specified by the user.
- After creating an index, Cassandra indexes new data automatically when data is inserted.
- The index cannot be created on primary key as a primary key is already indexed.
- Indexes on collections are not supported in Cassandra.
- Without indexing on the column, Cassandra can’t filter that column unless it is a primary key.

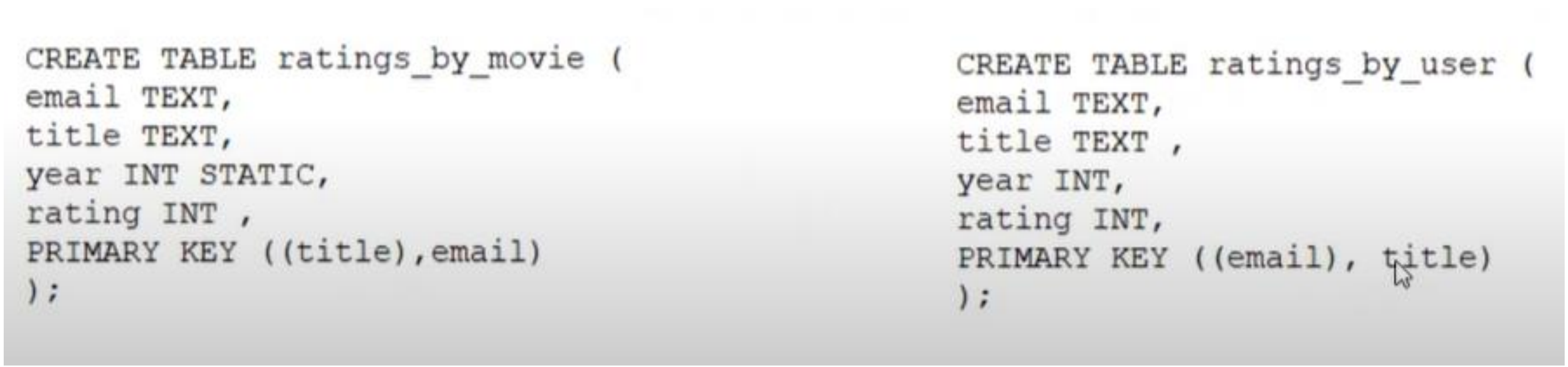
Query first Approach

-- Get all ratings left for a movie

Select * from ratings_by_movie where title='Avatar';

-- Get all rating left by a user

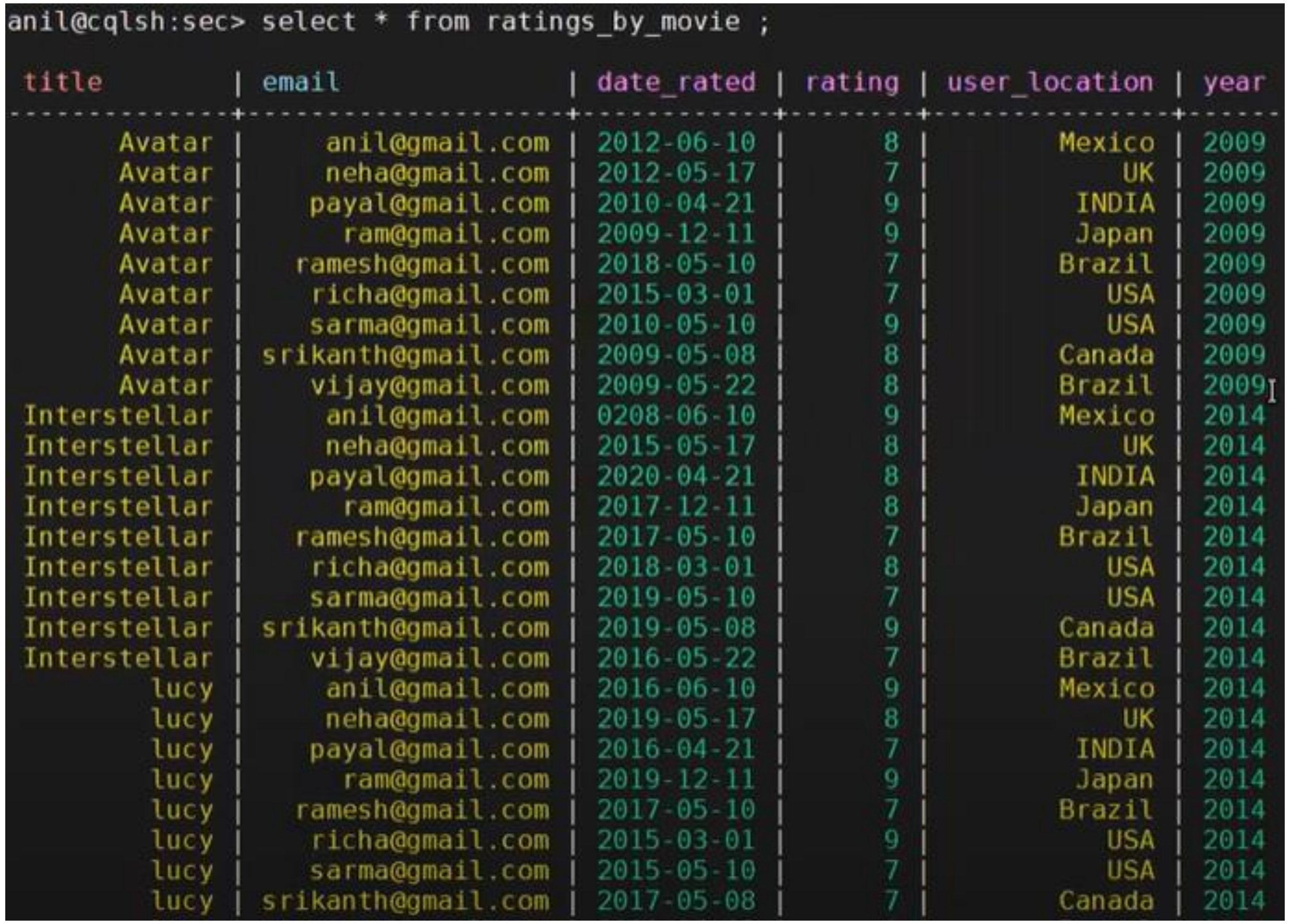
Select * from ratings_by_user where user='anil@gmail.com';



```
CREATE TABLE ratings_by_movie (
email TEXT,
title TEXT,
year INT STATIC,
rating INT ,
PRIMARY KEY ((title),email)
);
```

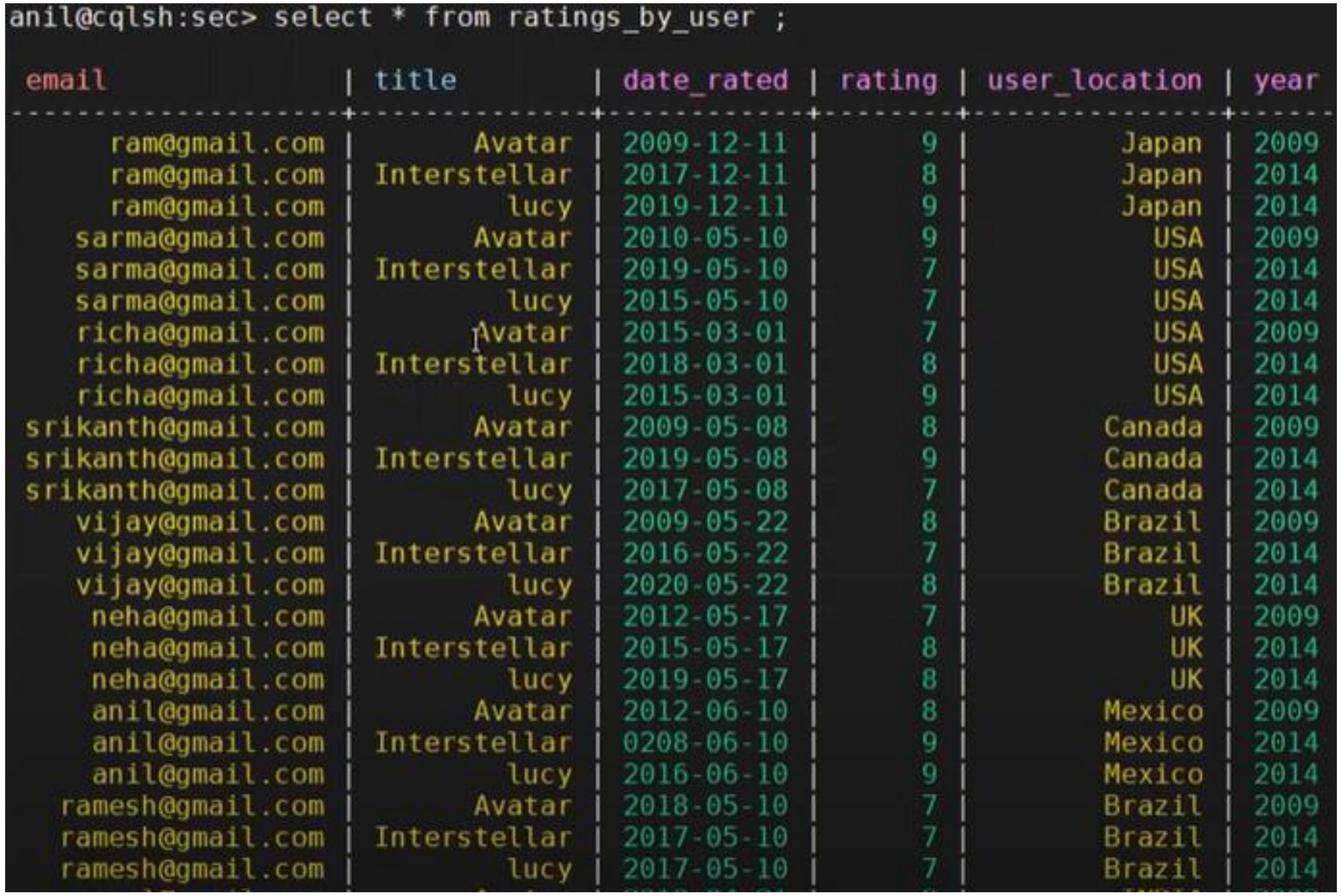
```
CREATE TABLE ratings_by_user (
email TEXT,
title TEXT ,
year INT,
rating INT,
PRIMARY KEY ((email), title)
);
```

QUERY-1



```
anil@cqlsh:sec> select * from ratings_by_movie ;

 title        | email              | date_rated | rating | user_location | year
--------------+--------------------+------------+--------+---------------+------
       Avatar |     anil@gmail.com | 2012-06-10 |      8 |        Mexico | 2009
       Avatar |     neha@gmail.com | 2012-05-17 |      7 |            UK | 2009
       Avatar |    payal@gmail.com | 2010-04-21 |      9 |         INDIA | 2009
       Avatar |      ram@gmail.com | 2009-12-11 |      9 |         Japan | 2009
       Avatar |   ramesh@gmail.com | 2018-05-10 |      7 |        Brazil | 2009
       Avatar |    richa@gmail.com | 2015-03-01 |      7 |           USA | 2009
       Avatar |    sarma@gmail.com | 2010-05-10 |      9 |           USA | 2009
       Avatar | srikanth@gmail.com | 2009-05-08 |      8 |        Canada | 2009
       Avatar |    vijay@gmail.com | 2009-05-22 |      8 |        Brazil | 2009
 Interstellar |     anil@gmail.com | 0208-06-10 |      9 |        Mexico | 2014
 Interstellar |     neha@gmail.com | 2015-05-17 |      8 |            UK | 2014
 Interstellar |    payal@gmail.com | 2020-04-21 |      8 |         INDIA | 2014
 Interstellar |      ram@gmail.com | 2017-12-11 |      8 |         Japan | 2014
 Interstellar |   ramesh@gmail.com | 2017-05-10 |      7 |        Brazil | 2014
 Interstellar |    richa@gmail.com | 2018-03-01 |      8 |           USA | 2014
 Interstellar |    sarma@gmail.com | 2019-05-10 |      7 |           USA | 2014
 Interstellar | srikanth@gmail.com | 2019-05-08 |      9 |        Canada | 2014
 Interstellar |    vijay@gmail.com | 2016-05-22 |      7 |        Brazil | 2014
         lucy |     anil@gmail.com | 2016-06-10 |      9 |        Mexico | 2014
         lucy |     neha@gmail.com | 2019-05-17 |      8 |            UK | 2014
         lucy |    payal@gmail.com | 2016-04-21 |      7 |         INDIA | 2014
         lucy |      ram@gmail.com | 2019-12-11 |      9 |         Japan | 2014
         lucy |   ramesh@gmail.com | 2017-05-10 |      7 |        Brazil | 2014
         lucy |    richa@gmail.com | 2015-03-01 |      9 |           USA | 2014
         lucy |    sarma@gmail.com | 2015-05-10 |      7 |           USA | 2014
         lucy | srikanth@gmail.com | 2017-05-08 |      7 |        Canada | 2014
```

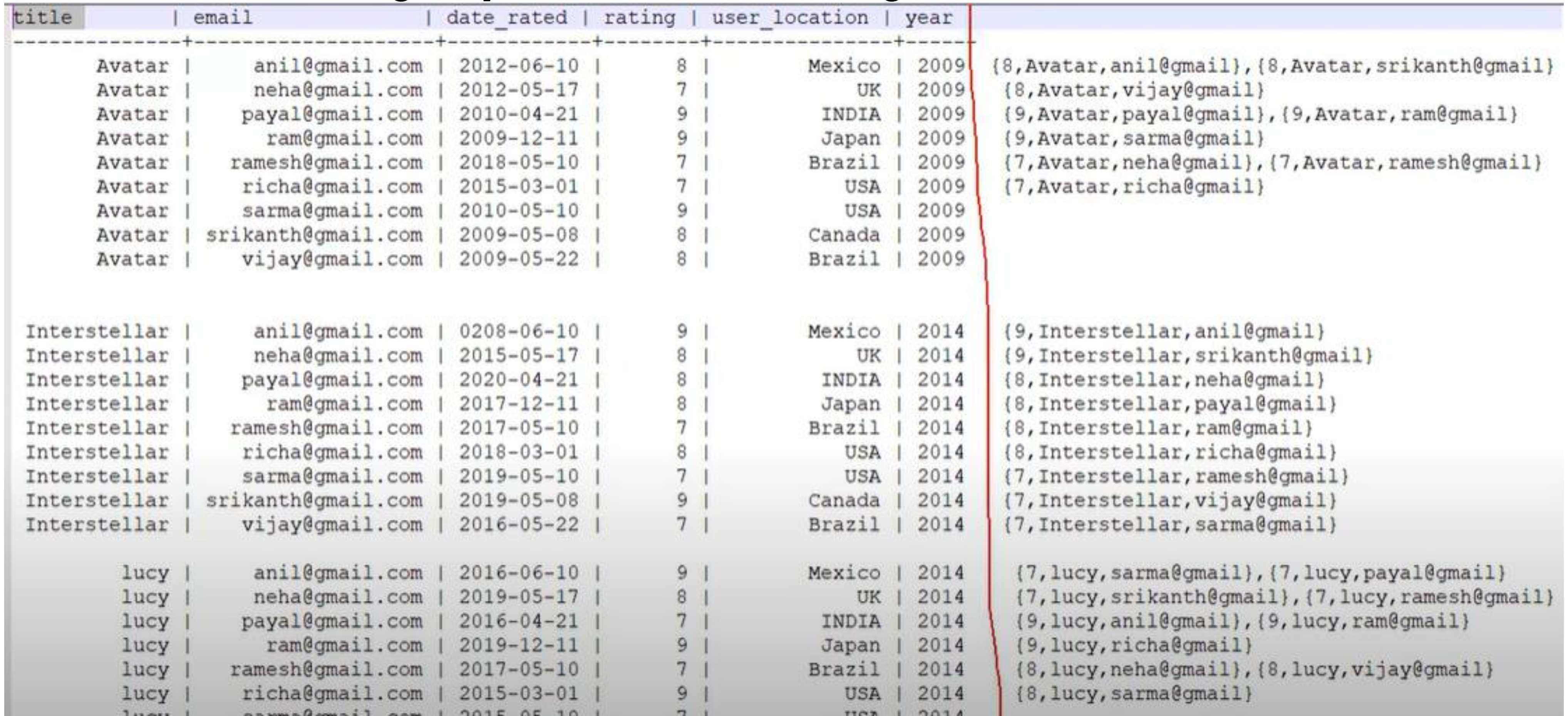
QUERY-2



```
anil@cqlsh:sec> select * from ratings_by_user ;

 email              | title        | date_rated | rating | user_location | year
--------------------+--------------+------------+--------+---------------+------
      ram@gmail.com |       Avatar | 2009-12-11 |      9 |         Japan | 2009
      ram@gmail.com | Interstellar | 2017-12-11 |      8 |         Japan | 2014
      ram@gmail.com |         lucy | 2019-12-11 |      9 |         Japan | 2014
    sarma@gmail.com |       Avatar | 2010-05-10 |      9 |           USA | 2009
    sarma@gmail.com | Interstellar | 2019-05-10 |      7 |           USA | 2014
    sarma@gmail.com |         lucy | 2015-05-10 |      7 |           USA | 2014
    richa@gmail.com |       Avatar | 2015-03-01 |      7 |           USA | 2009
    richa@gmail.com | Interstellar | 2018-03-01 |      8 |           USA | 2014
    richa@gmail.com |         lucy | 2015-03-01 |      9 |           USA | 2014
 srikanth@gmail.com |       Avatar | 2009-05-08 |      8 |        Canada | 2009
 srikanth@gmail.com | Interstellar | 2019-05-08 |      9 |        Canada | 2014
 srikanth@gmail.com |         lucy | 2017-05-08 |      7 |        Canada | 2014
    vijay@gmail.com |       Avatar | 2009-05-22 |      8 |        Brazil | 2009
    vijay@gmail.com | Interstellar | 2016-05-22 |      7 |        Brazil | 2014
    vijay@gmail.com |         lucy | 2020-05-22 |      8 |        Brazil | 2014
     neha@gmail.com |       Avatar | 2012-05-17 |      7 |            UK | 2009
     neha@gmail.com | Interstellar | 2015-05-17 |      8 |            UK | 2014
     neha@gmail.com |         lucy | 2019-05-17 |      8 |            UK | 2014
     anil@gmail.com |       Avatar | 2012-06-10 |      8 |        Mexico | 2009
     anil@gmail.com | Interstellar | 0208-06-10 |      9 |        Mexico | 2014
     anil@gmail.com |         lucy | 2016-06-10 |      9 |        Mexico | 2014
   ramesh@gmail.com |       Avatar | 2018-05-10 |      7 |        Brazil | 2009
   ramesh@gmail.com | Interstellar | 2017-05-10 |      7 |        Brazil | 2014
   ramesh@gmail.com |         lucy | 2017-05-10 |      7 |        Brazil | 2014
```

SECONDARY INDEX

- Select * from ratings_by_movies where rating=8;



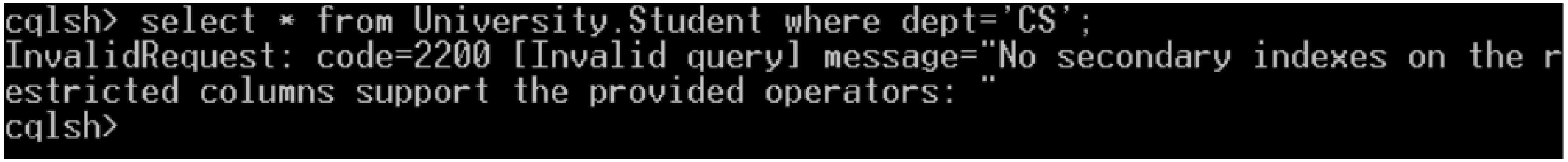
```
title         | email              | date_rated | rating | user_location | year
--------------+--------------------+------------+--------+---------------+-----
       Avatar |     anil@gmail.com | 2012-06-10 |      8 |        Mexico | 2009  {8,Avatar,anil@gmail},{8,Avatar,srikanth@gmail}
       Avatar |     neha@gmail.com | 2012-05-17 |      7 |            UK | 2009   {8,Avatar,vijay@gmail}
       Avatar |    payal@gmail.com | 2010-04-21 |      9 |         INDIA | 2009   {9,Avatar,payal@gmail},{9,Avatar,ram@gmail}
       Avatar |      ram@gmail.com | 2009-12-11 |      9 |         Japan | 2009   {9,Avatar,sarma@gmail}
       Avatar |   ramesh@gmail.com | 2018-05-10 |      7 |        Brazil | 2009   {7,Avatar,neha@gmail},{7,Avatar,ramesh@gmail}
       Avatar |    richa@gmail.com | 2015-03-01 |      7 |           USA | 2009   {7,Avatar,richa@gmail}
       Avatar |    sarma@gmail.com | 2010-05-10 |      9 |           USA | 2009
       Avatar | srikanth@gmail.com | 2009-05-08 |      8 |        Canada | 2009
       Avatar |    vijay@gmail.com | 2009-05-22 |      8 |        Brazil | 2009


 Interstellar |     anil@gmail.com | 0208-06-10 |      9 |        Mexico | 2014   {9,Interstellar,anil@gmail}
 Interstellar |     neha@gmail.com | 2015-05-17 |      8 |            UK | 2014   {9,Interstellar,srikanth@gmail}
 Interstellar |    payal@gmail.com | 2020-04-21 |      8 |         INDIA | 2014   {8,Interstellar,neha@gmail}
 Interstellar |      ram@gmail.com | 2017-12-11 |      8 |         Japan | 2014   {8,Interstellar,payal@gmail}
 Interstellar |   ramesh@gmail.com | 2017-05-10 |      7 |        Brazil | 2014   {8,Interstellar,ram@gmail}
 Interstellar |    richa@gmail.com | 2018-03-01 |      8 |           USA | 2014   {8,Interstellar,richa@gmail}
 Interstellar |    sarma@gmail.com | 2019-05-10 |      7 |           USA | 2014   {7,Interstellar,ramesh@gmail}
 Interstellar | srikanth@gmail.com | 2019-05-08 |      9 |        Canada | 2014   {7,Interstellar,vijay@gmail}
 Interstellar |    vijay@gmail.com | 2016-05-22 |      7 |        Brazil | 2014   {7,Interstellar,sarma@gmail}

         lucy |     anil@gmail.com | 2016-06-10 |      9 |        Mexico | 2014    {7,lucy,sarma@gmail},{7,lucy,payal@gmail}
         lucy |     neha@gmail.com | 2019-05-17 |      8 |            UK | 2014    {7,lucy,srikanth@gmail},{7,lucy,ramesh@gmail}
         lucy |    payal@gmail.com | 2016-04-21 |      7 |         INDIA | 2014    {9,lucy,anil@gmail},{9,lucy,ram@gmail}
         lucy |      ram@gmail.com | 2019-12-11 |      9 |         Japan | 2014    {9,lucy,richa@gmail}
         lucy |   ramesh@gmail.com | 2017-05-10 |      7 |        Brazil | 2014    {8,lucy,neha@gmail},{8,lucy,vijay@gmail}
         lucy |    richa@gmail.com | 2015-03-01 |      9 |           USA | 2014    {8,lucy,sarma@gmail}
```

SYNTAX

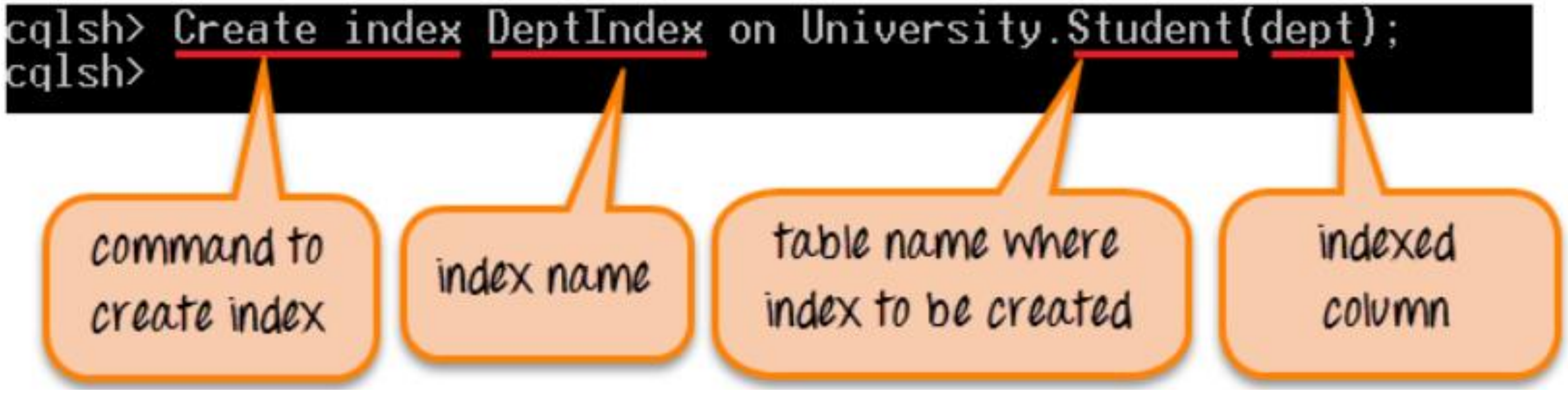
- Create index IndexName on KeyspaceName.TableName(ColumnName);

- CREATE INDEX [IF NOT EXISTS] index_name
- ON [keyspace_name.]table_name
- ([(KEYS | FULL)] column_name)
- (ENTRIES column_name);

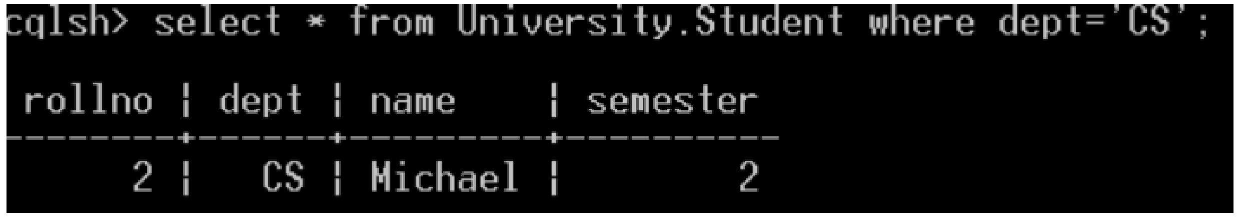
EXAMPLE



```
cqlsh> select * from University.Student where dept='CS';
InvalidRequest: code=2200 [Invalid query] message="No secondary indexes on the r
estricted columns support the provided operators: "
cqlsh>
```



```
cqlsh> Create index DeptIndex on University.Student(dept);
cqlsh>
```

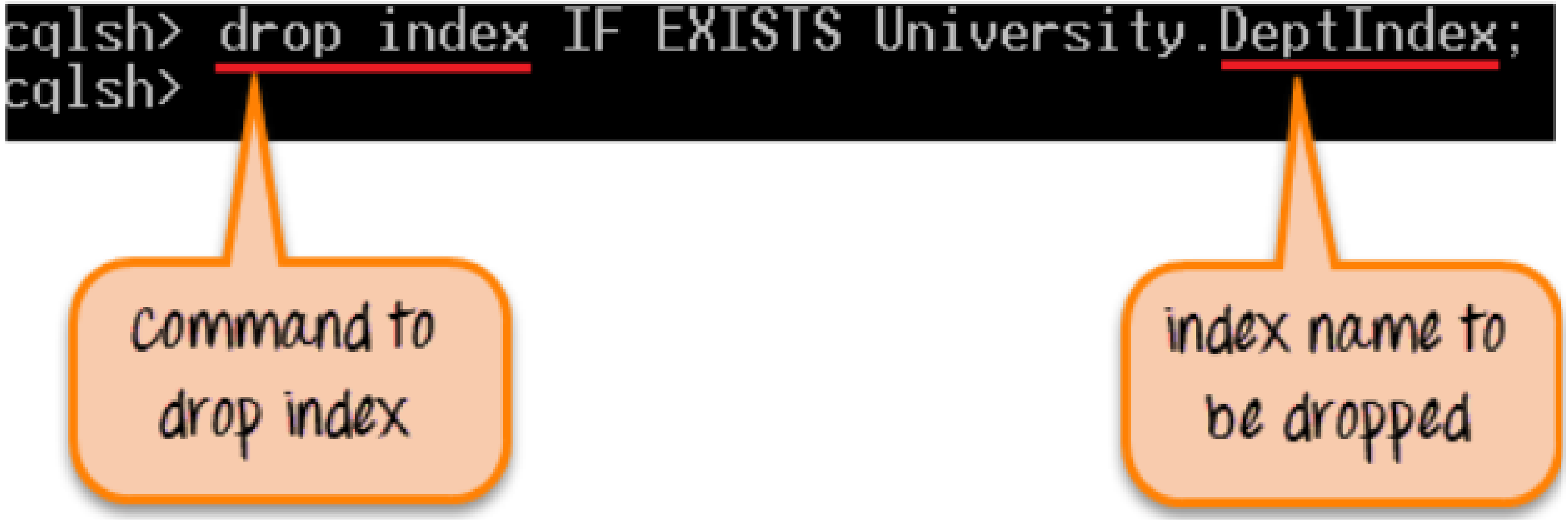


```
cqlsh> select * from University.Student where dept='CS';

 rollno | dept | name    | semester
--------+------+---------+----------
      2 |   CS | Michael |        2
```

DROP INDEX

- Command ‘Drop index’ drops the specified index.
- If the index does not exist, it will return an error unless IF EXISTS is used that will return no-op.
- During index creation, you have to specify keyspace name with the index name otherwise index will be dropped from the current keyspace.
- Syntax:
- Drop index IF EXISTS KeyspaceName.IndexName

EXAMPLE



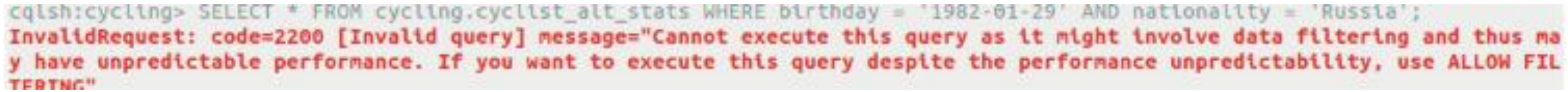
```
cqlsh> drop index IF EXISTS University.DeptIndex;
cqlsh>
```

- After successful execution of the command, DeptIndex will be dropped from the keyspace. Now data cannot be filtered by the column dept.

USING MULTIPLE INDEXES

- Indexes can be created on multiple columns and used in queries.
- cqlsh> CREATE TABLE cycling.cyclist_alt_stats (id UUID PRIMARY KEY, lastname text, birthday timestamp, nationality text, weight text, height text);
- cqlsh> CREATE INDEX birthday_idx ON cycling.cyclist_alt_stats (birthday);
- CREATE INDEX nationality_idx ON cycling.cyclist_alt_stats (nationality);

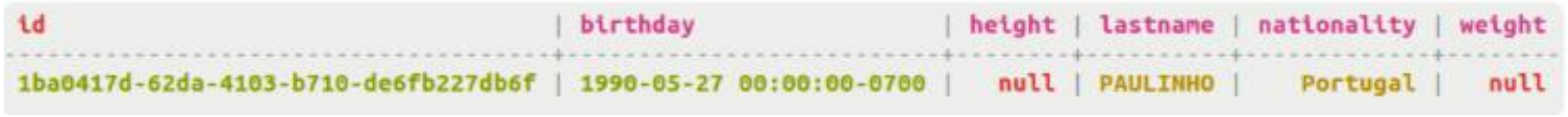
- cqlsh> SELECT * FROM cycling.cyclist_alt_stats WHERE birthday = '1982-01-29' AND nationality = 'Russia';



```
cqlsh:cycling> SELECT * FROM cycling.cyclist_alt_stats WHERE birthday = '1982-01-29' AND nationality = 'Russia';
InvalidRequest: code=2200 [Invalid query] message="Cannot execute this query as it might involve data filtering and thus ma
y have unpredictable performance. If you want to execute this query despite the performance unpredictability, use ALLOW FIL
TERING"
```

MULTIPLE INDEXES

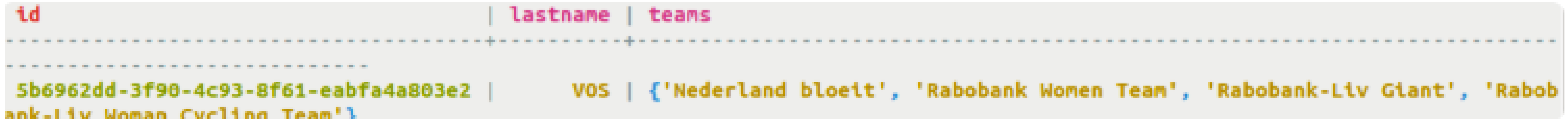
- The indexes have been created on appropriate low cardinality columns, but the query still fails.
- The error is not due to multiple indexes, but the lack of a partition key definition in the query.
- When you attempt a potentially expensive query, such as searching a range of rows, Cassandra requires the ALLOW FILTERING directive.
- cqlsh> SELECT * FROM cycling.cyclist_alt_stats WHERE birthday = '1990-05-27' AND nationality = 'Portugal' ALLOW FILTERING



```
 id                                   | birthday                 | height | lastname | nationality | weight
--------------------------------------+--------------------------+--------+----------+-------------+--------
 1ba0417d-62da-4103-b710-de6fb227db6f | 1990-05-27 00:00:00-0700 |   null | PAULINHO |    Portugal |   null
```

INDEXING A COLLECTION

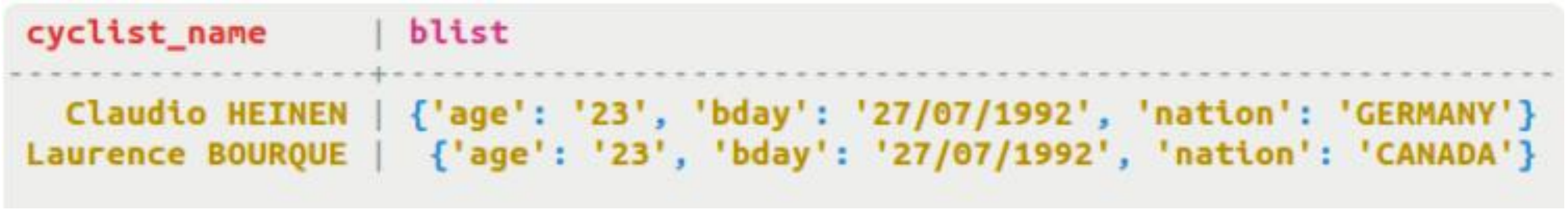
- Collections can be indexed and queried to find a collection containing a particular value.
- Set and List collections
- CREATE INDEX team_idx ON cycling.cyclist_career_teams (teams);
- SELECT * FROM cycling.cyclist_career_teams WHERE teams CONTAINS 'Nederland bloeit’;



```
 id                                   | lastname | teams
--------------------------------------+----------+-----------------------------------------------------------------------------------------
-------------------------------
 5b6962dd-3f90-4c93-8f61-eabfa4a803e2 |      VOS | {'Nederland bloeit', 'Rabobank Women Team', 'Rabobank-Liv Giant', 'Rabob
ank-Liv Woman Cycling Team'}
```

INDEXING A COLLECTION

- Maps
- CREATE TABLE cycling.birthday_list (cyclist_name text PRIMARY KEY, blist map<text,text>);
- CREATE INDEX blist_idx ON cycling.birthday_list (ENTRIES(blist));
- SELECT * FROM cycling.birthday_list WHERE blist['age'] = '23';



```
 cyclist_name      | blist
-------------------+-------------------------------------------------------------
   Claudio HEINEN | {'age': '23', 'bday': '27/07/1992', 'nation': 'GERMANY'}
 Laurence BOURQUE |  {'age': '23', 'bday': '27/07/1992', 'nation': 'CANADA'}
```

THANK YOU