# *HBASE*

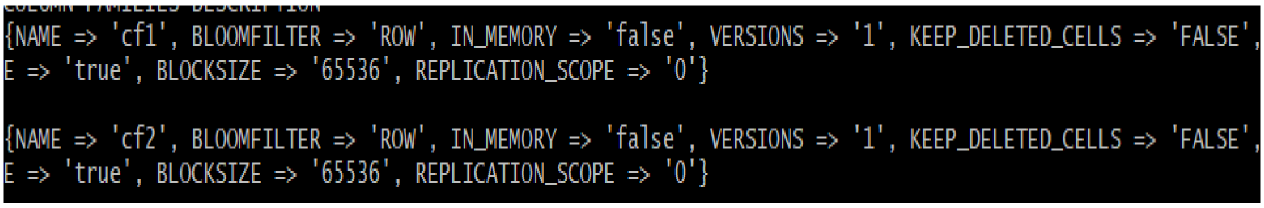
## Exercise 1:

## COMMAND:

**create 'csp554Tbl', 'cf1', 'cf2'**

**describe 'csp554Tbl'**

## Output:



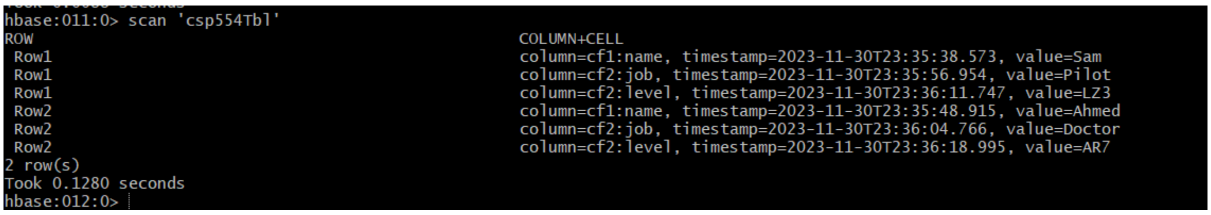
```
COLUMN FAMILIES DESCRIPTION
{NAME => 'cf1', BLOOMFILTER => 'ROW', IN_MEMORY => 'false', VERSIONS => '1', KEEP_DELETED_CELLS => 'FALSE',
E => 'true', BLOCKSIZE => '65536', REPLICATION_SCOPE => '0'}

{NAME => 'cf2', BLOOMFILTER => 'ROW', IN_MEMORY => 'false', VERSIONS => '1', KEEP_DELETED_CELLS => 'FALSE',
E => 'true', BLOCKSIZE => '65536', REPLICATION_SCOPE => '0'}
```

## Exercise 2:

## COMMAND:

**scan 'csp554Tbl'**

## OUTPUT:



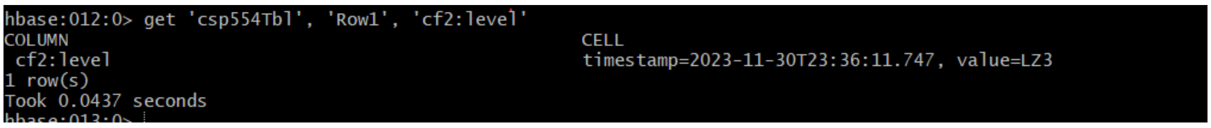
```
hbase:011:0> scan 'csp554Tbl'
ROW                                                   COLUMN+CELL
 Row1                                                 column=cf1:name, timestamp=2023-11-30T23:35:38.573, value=Sam
 Row1                                                 column=cf2:job, timestamp=2023-11-30T23:35:56.954, value=Pilot
 Row1                                                 column=cf2:level, timestamp=2023-11-30T23:36:11.747, value=LZ3
 Row2                                                 column=cf1:name, timestamp=2023-11-30T23:35:48.915, value=Ahmed
 Row2                                                 column=cf2:job, timestamp=2023-11-30T23:36:04.766, value=Doctor
 Row2                                                 column=cf2:level, timestamp=2023-11-30T23:36:18.995, value=AR7
2 row(s)
Took 0.1280 seconds
hbase:012:0>
```

## Exercise 3:

## COMMAND:

**get 'csp554Tbl', 'Row1', 'cf2:level'**

## OUTPUT:



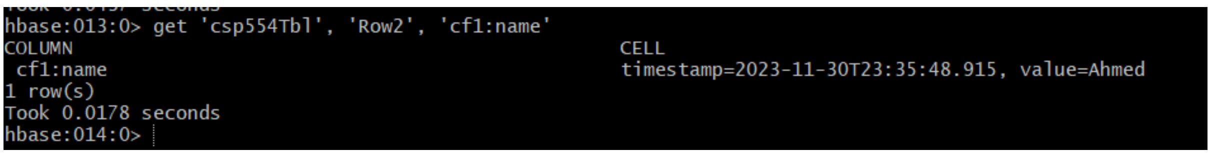
```
hbase:012:0> get 'csp554Tbl', 'Row1', 'cf2:level'
COLUMN                                                CELL
 cf2:level                                            timestamp=2023-11-30T23:36:11.747, value=LZ3
1 row(s)
Took 0.0437 seconds
hbase:013:0>
```

## Exercise 4:

## COMMAND:

**get 'csp554Tbl', 'Row2', 'cf1:name'**

## OUTPUT:



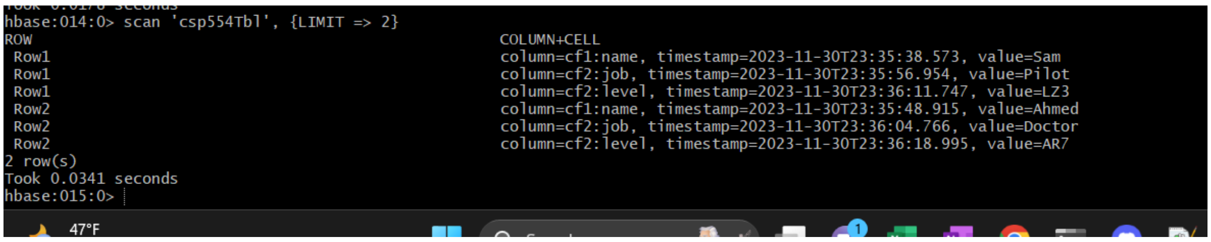
```
hbase:013:0> get 'csp554Tbl', 'Row2', 'cf1:name'
COLUMN                                                    CELL
 cf1:name                                                 timestamp=2023-11-30T23:35:48.915, value=Ahmed
1 row(s)
Took 0.0178 seconds
hbase:014:0>
```

## Exercise 5:

## COMMAND:

**scan 'csp554Tbl', {LIMIT => 2}**

## OUTPUT:



```
hbase:014:0> scan 'csp554Tbl', {LIMIT => 2}
ROW                                        COLUMN+CELL
 Row1                                      column=cf1:name, timestamp=2023-11-30T23:35:38.573, value=Sam
 Row1                                      column=cf2:job, timestamp=2023-11-30T23:35:56.954, value=Pilot
 Row1                                      column=cf2:level, timestamp=2023-11-30T23:36:11.747, value=LZ3
 Row2                                      column=cf1:name, timestamp=2023-11-30T23:35:48.915, value=Ahmed
 Row2                                      column=cf2:job, timestamp=2023-11-30T23:36:04.766, value=Doctor
 Row2                                      column=cf2:level, timestamp=2023-11-30T23:36:18.995, value=AR7
2 row(s)
Took 0.0341 seconds
hbase:015:0>
```

## *Cassandra*

## Exercise 1:

**ANS :** The article proposes a novel, query-driven data modeling methodology tailored for the Apache Cassandra NoSQL database. Traditional relational database modeling focuses primarily on conceptual data modeling and normalization. In contrast, this Cassandra methodology puts equal emphasis on understanding the application queries that will be run against the database. It has four main components:

- ✓ Conceptual data modeling to understand the data
- ✓ Application workflow modeling to understand the access patterns/queries
- ✓ Logical data modeling driven by the queries to design efficient Cassandra table schemas
- ✓ Physical data modeling to optimize data types, keys, partitions etc.

The methodology relies heavily on principles like data nesting within table partitions and intentional data duplication across tables to optimize for fast writes and reads in Cassandra. The authors introduce new modeling principles, mapping rules, patterns and visualization diagrams to guide Cassandra schema design. They also present a tool, KDM, that automates major parts of the methodology.

## Comments:

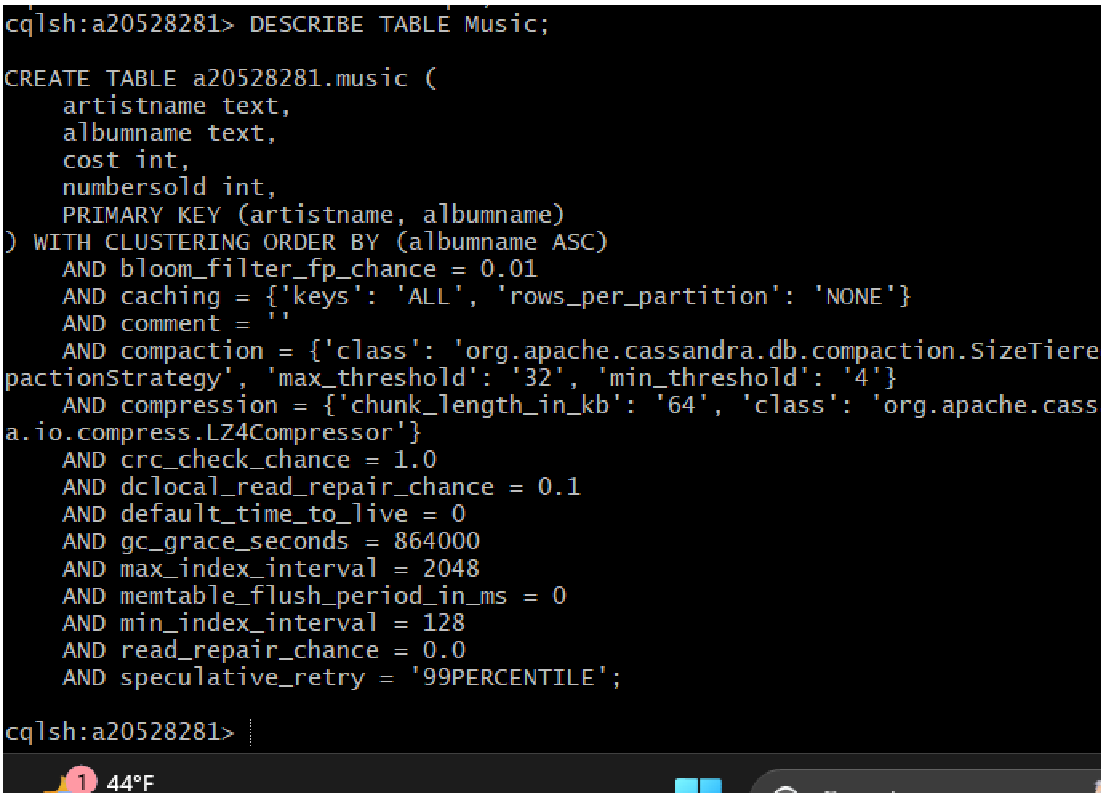
This methodology seems like a rigorous, systematic approach tailored to leverage Cassandra's architecture. Mapping application access patterns directly into the logical schema design is smart. The conceptual diagrams provide useful visualization. Automation through the KDM tool would be extremely helpful for developers. My main question would be how adaptable this is to schema changes over time as application queries evolve. But overall this appears to be a solid modeling foundation for Cassandra databases.

## Exercise 2:

## COMMAND:

**DESCRIBE TABLE MUSIC;**

## OUTPUT:



```
cqlsh:a20528281> DESCRIBE TABLE Music;

CREATE TABLE a20528281.music (
    artistname text,
    albumname text,
    cost int,
    numbersold int,
    PRIMARY KEY (artistname, albumname)
) WITH CLUSTERING ORDER BY (albumname ASC)
    AND bloom_filter_fp_chance = 0.01
    AND caching = {'keys': 'ALL', 'rows_per_partition': 'NONE'}
    AND comment = ''
    AND compaction = {'class': 'org.apache.cassandra.db.compaction.SizeTiere
pactionStrategy', 'max_threshold': '32', 'min_threshold': '4'}
    AND compression = {'chunk_length_in_kb': '64', 'class': 'org.apache.cass
a.io.compress.LZ4Compressor'}
    AND crc_check_chance = 1.0
    AND dclocal_read_repair_chance = 0.1
    AND default_time_to_live = 0
    AND gc_grace_seconds = 864000
    AND max_index_interval = 2048
    AND memtable_flush_period_in_ms = 0
    AND min_index_interval = 128
    AND read_repair_chance = 0.0
    AND speculative_retry = '99PERCENTILE';

cqlsh:a20528281>
```

## Exercise 3:

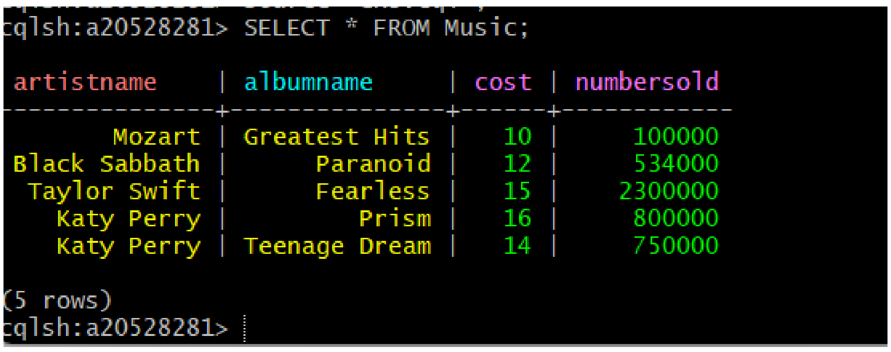
## CODE :

```
INSERT INTO Music (artistName, albumName, numberSold, cost)
VALUES
   ('Mozart', 'Greatest Hits', 100000, 10),
   ('Taylor Swift', 'Fearless', 2300000, 15),
   ('Black Sabbath', 'Paranoid', 534000, 12),
   ('Katy Perry', 'Prism', 800000, 16),
   ('Katy Perry', 'Teenage Dream', 750000, 14);
```

**COMMAND:**

**SELECT * FROM Music;**

**OUTPUT:**



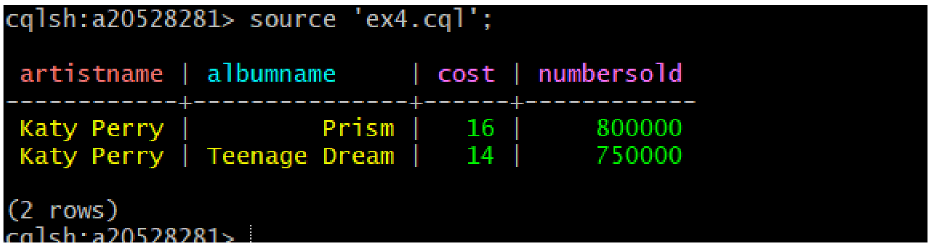
```
cqlsh:a20528281> SELECT * FROM Music;

 artistname    | albumname     | cost | numbersold
---------------+---------------+------+------------
        Mozart | Greatest Hits |   10 |     100000
 Black Sabbath |      Paranoid |   12 |     534000
  Taylor Swift |      Fearless |   15 |    2300000
    Katy Perry |         Prism |   16 |     800000
    Katy Perry | Teenage Dream |   14 |     750000

(5 rows)
cqlsh:a20528281>
```

**Exercise 4 :**

**COMMAND:**

**source ‘ex4.cql’;**

**Output:**



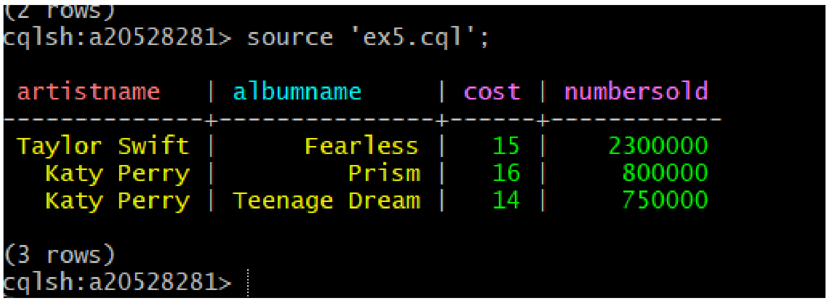
```
cqlsh:a20528281> source 'ex4.cql';

 artistname | albumname     | cost | numbersold
------------+---------------+------+------------
 Katy Perry |         Prism |   16 |     800000
 Katy Perry | Teenage Dream |   14 |     750000

(2 rows)
cqlsh:a20528281>
```

**Exercise 5:**

**COMMAND:**

**Source ‘ex5.cql’;**

**Output:**



```
(2 rows)
cqlsh:a20528281> source 'ex5.cql';

 artistname    | albumname      | cost | numbersold
---------------+----------------+------+------------
 Taylor Swift  |       Fearless |   15 |    2300000
    Katy Perry |          Prism |   16 |     800000
    Katy Perry | Teenage Dream  |   14 |     750000

(3 rows)
cqlsh:a20528281>
```

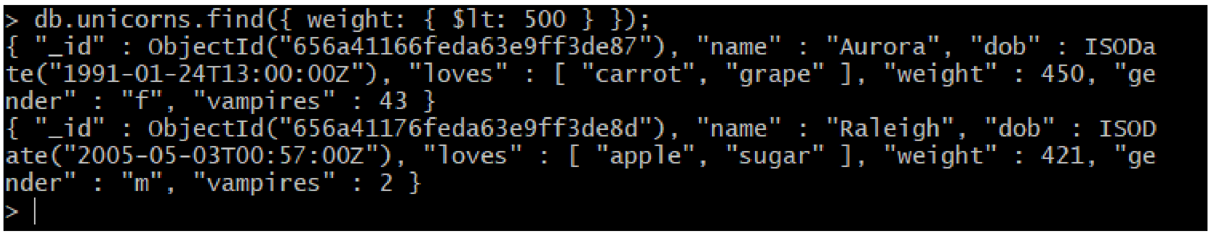
## MONGO DB

**Exercise 1:**

**COMMAND:**

**db.unicorns.find({ weight: { $lt: 500 } });**

**Output:**



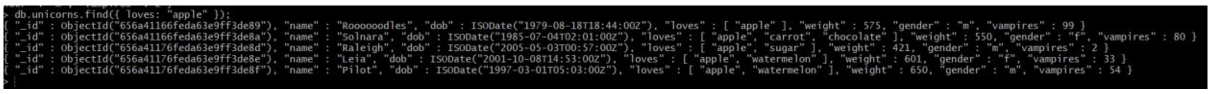
```
> db.unicorns.find({ weight: { $lt: 500 } });
{ "_id" : ObjectId("656a41166feda63e9ff3de87"), "name" : "Aurora", "dob" : ISODa
te("1991-01-24T13:00:00Z"), "loves" : [ "carrot", "grape" ], "weight" : 450, "ge
nder" : "f", "vampires" : 43 }
{ "_id" : ObjectId("656a41176feda63e9ff3de8d"), "name" : "Raleigh", "dob" : ISOD
ate("2005-05-03T00:57:00Z"), "loves" : [ "apple", "sugar" ], "weight" : 421, "ge
nder" : "m", "vampires" : 2 }
>
```

**Exercise 2:**

**Command:**

**db.unicorns.find({ loves: "apple" });**

**Output:**



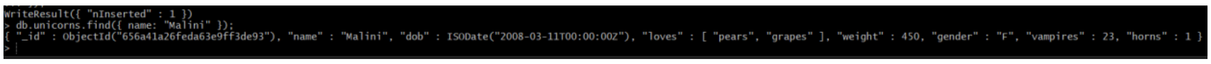
```
> db.unicorns.find({ loves: "apple" });
{ "_id" : ObjectId("656a41166feda63e9ff3de89"), "name" : "Roooooodles", "dob" : ISODate("1979-08-18T18:44:00Z"), "loves" : [ "apple" ], "weight" : 575, "gender" : "m", "vampires" : 99 }
{ "_id" : ObjectId("656a41166feda63e9ff3de8a"), "name" : "Solnara", "dob" : ISODate("1985-07-04T02:01:00Z"), "loves" : [ "apple", "carrot", "chocolate" ], "weight" : 550, "gender" : "f", "vampires" : 80 }
{ "_id" : ObjectId("656a41176feda63e9ff3de8d"), "name" : "Raleigh", "dob" : ISODate("2005-05-03T00:57:00Z"), "loves" : [ "apple", "sugar" ], "weight" : 421, "gender" : "m", "vampires" : 2 }
{ "_id" : ObjectId("656a41176feda63e9ff3de8e"), "name" : "Leia", "dob" : ISODate("2001-10-08T14:53:00Z"), "loves" : [ "apple", "watermelon" ], "weight" : 601, "gender" : "f", "vampires" : 33 }
{ "_id" : ObjectId("656a41176feda63e9ff3de8f"), "name" : "Pilot", "dob" : ISODate("1997-03-01T05:03:00Z"), "loves" : [ "apple", "watermelon" ], "weight" : 650, "gender" : "m", "vampires" : 54 }
>
```

**Exercise 3:**

**Command:**

```
        db.unicorns.insert({
  name: "Malini",
  dob: new Date("2008-03-11T00:00:00Z"),
  loves: ["pears", "grapes", "carrots"],
  weight: 470,
  gender: "F",
  vampires: 24,
  horns: 1
 });
                 db.unicorns.find({ name: "Malini" });
```

**Output:**



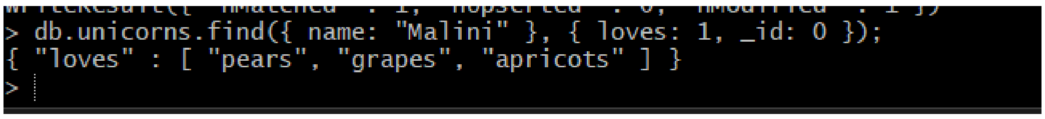
```
WriteResult({ "nInserted" : 1 })
> db.unicorns.find({ name: "Malini" });
{ "_id" : ObjectId("656a41a26feda63e9ff3de93"), "name" : "Malini", "dob" : ISODate("2008-03-11T00:00:00Z"), "loves" : [ "pears", "grapes" ], "weight" : 450, "gender" : "F", "vampires" : 23, "horns" : 1 }
>
```

**Exercise 4:**

**Command:**

```
        db.unicorns.update(
 { name: "Malini" },
 { $push: { loves: "grapes" } }
);
                db.unicorns.find({ name: "Malini" }, { loves: 1, _id: 0 });
```

**Output:**



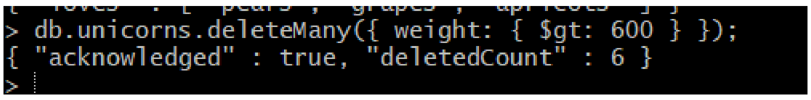
```
> db.unicorns.find({ name: "Malini" }, { loves: 1, _id: 0 });
{ "loves" : [ "pears", "grapes", "apricots" ] }
>
```

**Exercise 5:**

**Command:**

```
        db.unicorns.deleteMany({ weight: { $gt: 600 } });
```

**Output:**



```
> db.unicorns.deleteMany({ weight: { $gt: 600 } });
{ "acknowledged" : true, "deletedCount" : 6 }
>
```