**\*HBASE\***

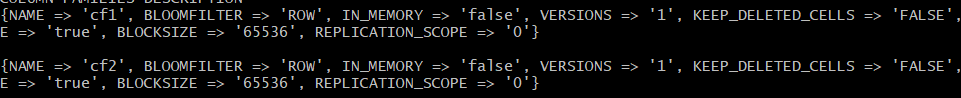
**Exercise 1:**

**COMMAND:**

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

**describe 'csp554Tbl'**

**Output:**

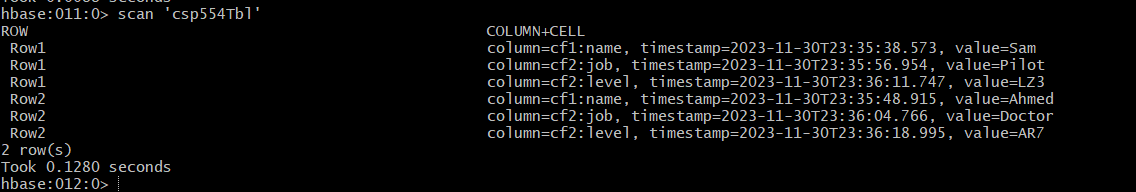


**Exercise 2:**

**COMMAND:**

**scan 'csp554Tbl'**

**OUTPUT:**

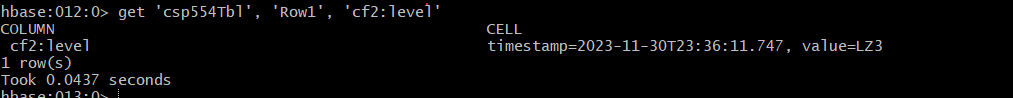


**Exercise 3:**

**COMMAND:**

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

**OUTPUT:**

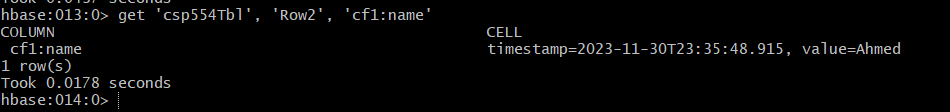


**Exercise 4:**

**COMMAND:**

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

**OUTPUT:**

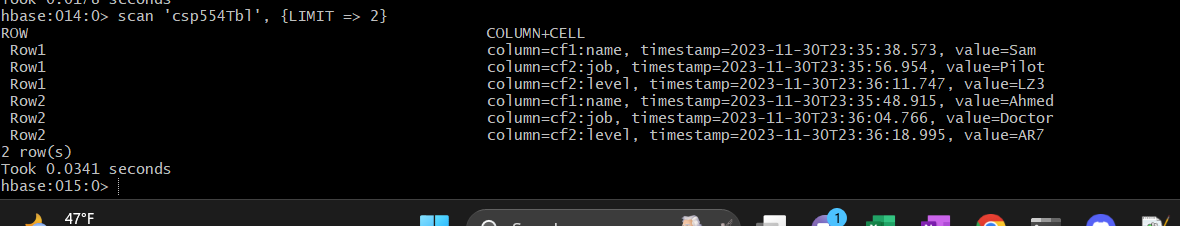


**Exercise 5:**

**COMMAND:**

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

**OUTPUT:**



**\*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:**

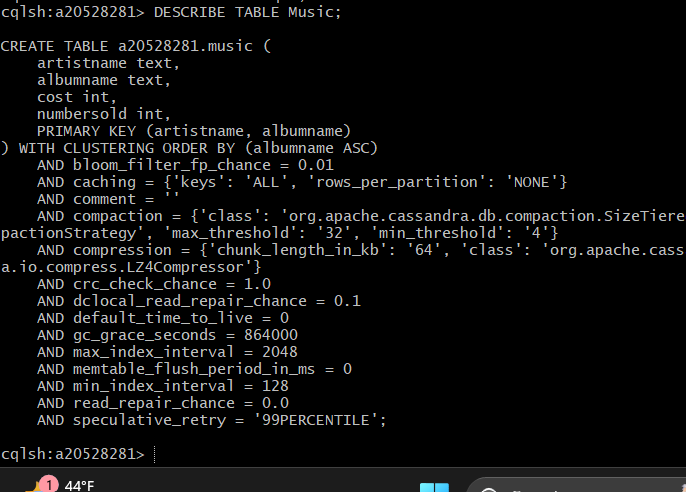
This methodology seems like a rigorous, systematic approach tailored to leverage Cassandra's architecture. Mapping application access patterns directly into the logical schema design is smart. The conceptual diagrams provide useful visualization. Automation through the KDM tool would be extremely helpful 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:**



**Exercise 3:**

**CODE :**

INSERT INTO Music (artistName, albumName, numberSold, cost)

VALUES

('Mozart', 'Greatest Hits', 100000, 10),

('Taylor Swift', 'Fearless', 2300000, 15),

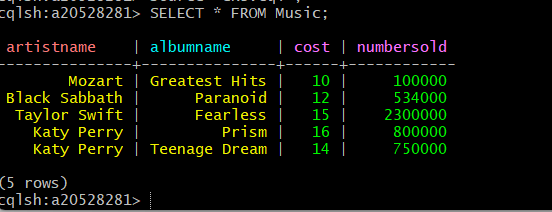
('Black Sabbath', 'Paranoid', 534000, 12),

('Katy Perry', 'Prism', 800000, 16),

('Katy Perry', 'Teenage Dream', 750000, 14);

**COMMAND:  
 SELECT \* FROM Music;**

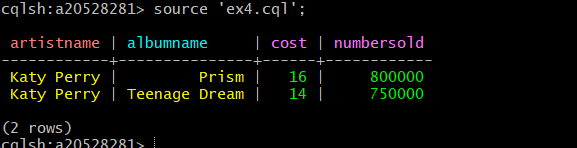
**OUTPUT:**



**Exercise 4 :**

**COMMAND:  
 source ‘ex4.cql’;**

**Output:**

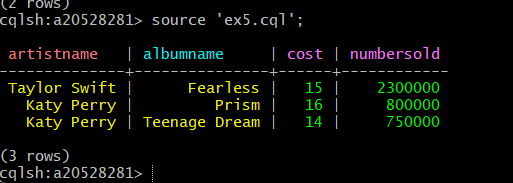


**Exercise 5:**

**COMMAND:**

**Source ‘ex5.cql’;**

**Output:**



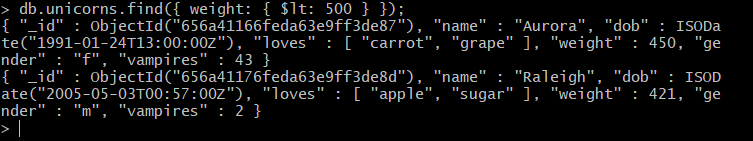
**MONGO DB**

**Exercise 1:**

**COMMAND:**

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

**Output:**

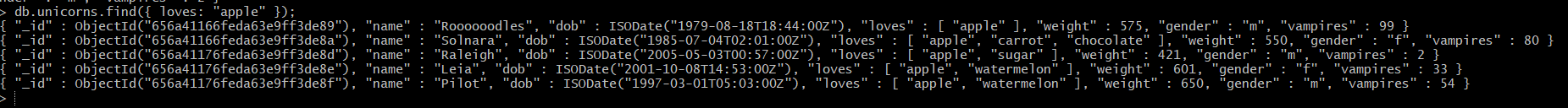


**Exercise 2:**

**Command:**

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

**Output:**



**Exercise 3:**

**Command:**

**db.unicorns.insert({**

**name: "Malini",**

**dob: new Date("2008-03-11T00:00:00Z"),**

**loves: ["pears", "grapes", "carrots"],**

**weight: 470,**

**gender: "F",**

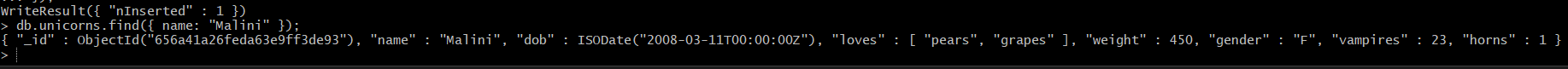
**vampires: 24,**

**horns: 1**

**});**

**db.unicorns.find({ name: "Malini" });**

**Output:**



**Exercise 4:**

**Command:**

**db.unicorns.update(**

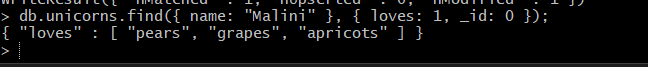
**{ name: "Malini" },**

**{ $push: { loves: "grapes" } }**

**);**

**db.unicorns.find({ name: "Malini" }, { loves: 1, \_id: 0 });**

**Output:**



**Exercise 5:**

**Command:**

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

**Output:**

