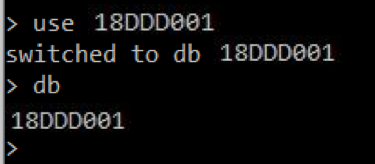
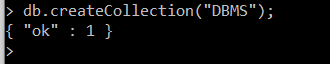
**PRACTICAL – 18**

**To create, insert values in MongoDB**

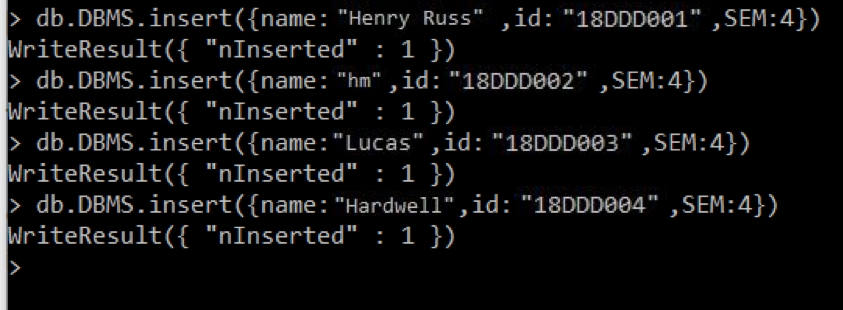
**(i) Create DB in MongoDB:**



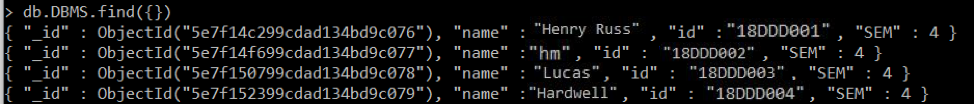
**(ii) Create collection in MongoDB:**



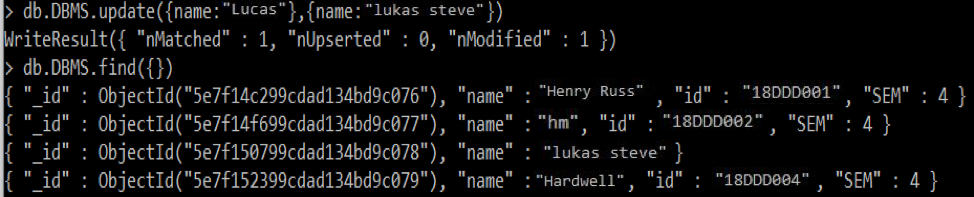
**(iii) Insert in MongoDB:**



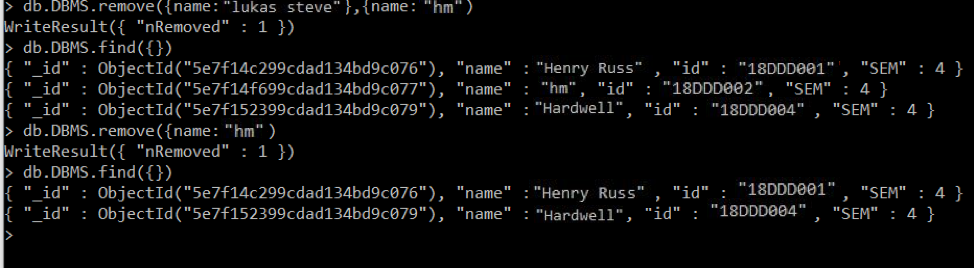
**(iv)** **Select in MongoDB :**



**(v) Update in MongoDB :**



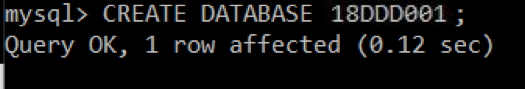
**(vi) Delete in MongoDB :**



**PRACTICAL – 19**

**To create, modify, delete, execute and recompile a stored procedure in SQL Server/ MySQL**

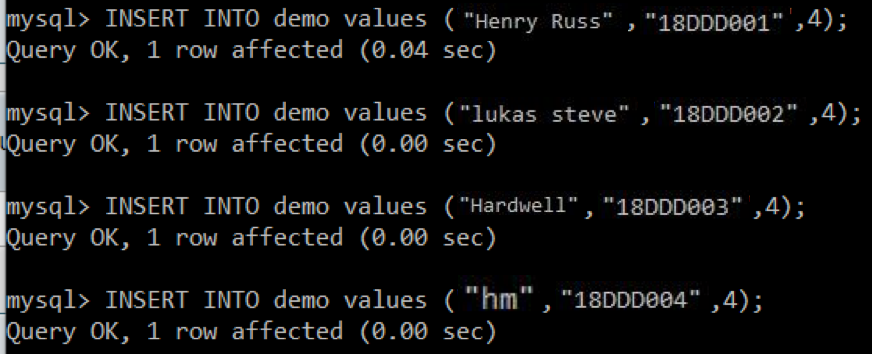
**(i) Create DB in MySQL:**



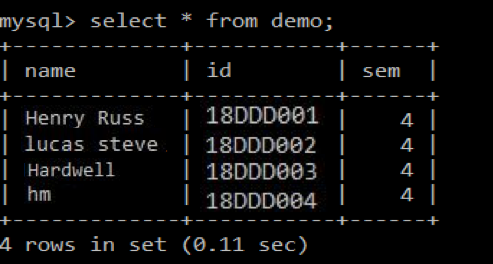
**(ii) Create table in MySQL:**

page98image42770144

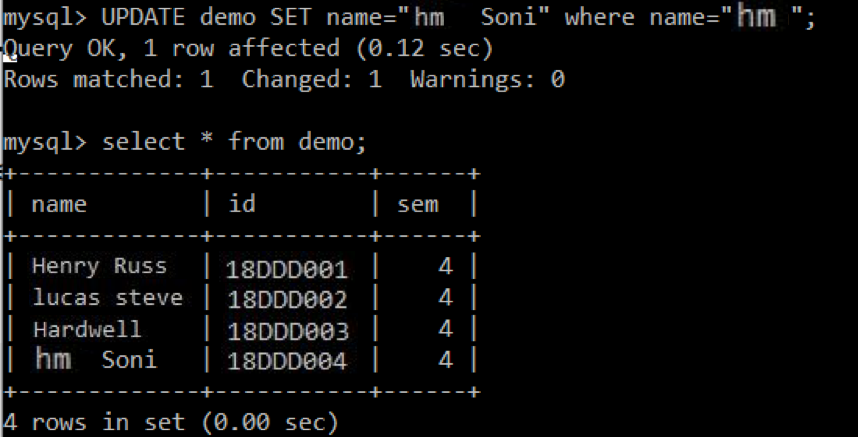
**(iii) Insert in MySQL:**



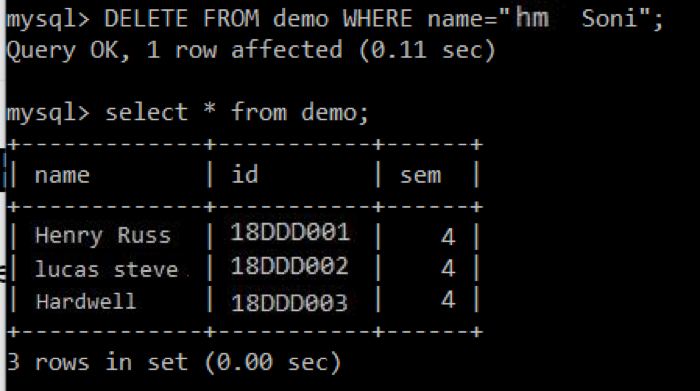
**(iv) Select in MySQL:**



**(v) Update in MySQL:**



**(vi) Delete in MySQL:**



**PRACTICAL – 20**

**Case study on Software used for NoSQL database.**

**Theory:**

What is NoSQL ?

* NoSQL is an approach to database design that can accomodate a wide variety of data models, including key-value, document, columnar and graph formats. NoSQL, which stand for "not only SQL," is an alternative to traditional relational databases in which data is placed in tables and data schema is carefully designed before the database is built. NoSQL databases are especially useful for working with large sets of distributed data.
* NoSQL encompasses a wide variety of different database technologies that were developed in response to the demands presented in building modern applications:
* Developers are working with applications that create massive volumes of new, rapidly changing data types — structured, semi-structured, unstructured and polymorphic data.
* Long gone is the twelve-to-eighteen month waterfall development cycle. Now small teams work in agile sprints, iterating quickly and pushing code every week or two, some even multiple times every day.
* Applications that once served a finite audience are now delivered as services that must be always-on, accessible from many different devices and scaled globally to millions of users.
* Organizations are now turning to scale-out architectures using open source software, commodity servers and cloud computing instead of large monolithic servers and storage infrastructure.

How does NoSQL Database works ?

* Each of the systems labelled with the generic name works differently, but the basic idea is to offer better scalability and performance by using DB models that don't support all the functionality of a generic RDBMS, but still enough functionality to be useful. In a way it's like MySQL, which at one time lacked support for transactions but, exactly because of that, managed to outperform other DB systems. If you could write your app in a way that didn't require transactions, it was great.

* Types of NoSQL databases- NoSQL databases typically fall into one of four categories:
* Key-value stores are the simplest. Every item in the database is stored as an attribute name (or "key") together with its value. Riak, Voldemort, and Redis are the most wellknown in this category.

* Wide-column stores store data together as columns instead of rows and are optimized for queries over large datasets. The most popular are Cassandra and HBase.

* Document databases pair each key with a complex data structure known as a document. Documents can contain many different key-value pairs, or key-array pairs, or even nested documents. MongoDB is the most popular of these databases.

* Graph databases are used to store information about networks, such as social connections. Examples are Neo4J and Hypergraph DB.

* Key-value store:
  + From an API perspective, key-value stores are the simplest NoSQL data stores to use. The client can either get the value for the key, assign a value for a key or delete a key from the data store.
  + The value is a blob that the data store just stores, without caring or knowing what’s inside; it’s the responsibility of the application to understand what was stored.
  + Since key-value stores always use primary-key access, they generally have great performance and can be easily scaled. The key-value database uses a hash table to store unique keys and pointers (in some databases it’s also called the inverted index) with respect to each data value it stores.
  + There are no column type relations in the database; hence, its implementation is easy. Key-value databases give great performance and can be very easily scaled as per business needs.
* Document store:
  + - It is similar to key-value databases in that there’s a key and a value. Data is stored as a value. Its associated key is the unique identifier for that value.
    - The difference is that, in a document database, the value contains structured or semi-structured data. This structured/semi-structured value is referred to as a document and can be in XML, JSON or BSON format.

* Column store:

In column-oriented NoSQL databases, data is stored in cells grouped in columns of data rather than as rows of data. Columns are logically grouped into column families. Column families can contain a virtually unlimited number of columns that can be created at runtime or while defining the schema.

* + - Read and write is done using columns rather than rows. Column families are groups of similar data that is usually accessed together. As an example, we often access customers’ names and profile information at the same time, but not the information on their orders.
* Graph base:
  + - Graph databases are basically built upon the Entity – Attribute – Value model. Entities are also known as nodes, which have properties. It is a very flexible way to describe how data relates to other data.
    - Nodes store data about each entity in the database, relationships describe a relationship between nodes, and a property is simply the node on the opposite end of the relationship. Whereas a traditional database stores a description of each possible relationship in foreign key fields or junction tables, graph databases allow for virtually any relationship to be defined on-the-fly.

* Comparison between types of NoSQL database :

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Data model | Performance | Scalability | Flexibility | Complexity | Functionality |
| Key–  value store | high | high | high | none | variable (none) |
| Column-  oriented store | high | high | moderate | low | minimal |
| Document  -oriented store | high | variable (high) | high | low | variable (low) |
| Graph database | variable | variable | high | high | graph theory |