1.What is NoSQL data base?

NoSQL is a class of database management systems (DBMS) that do not follow all of the rules of a relational DBMS and cannot use traditional SQL to query data. The term is somewhat misleading when interpreted as "No SQL," and most translate it as "Not Only SQL," as this type of database is not generally a replacement but, rather, a complementary addition to RDBMSs and SQL.
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.

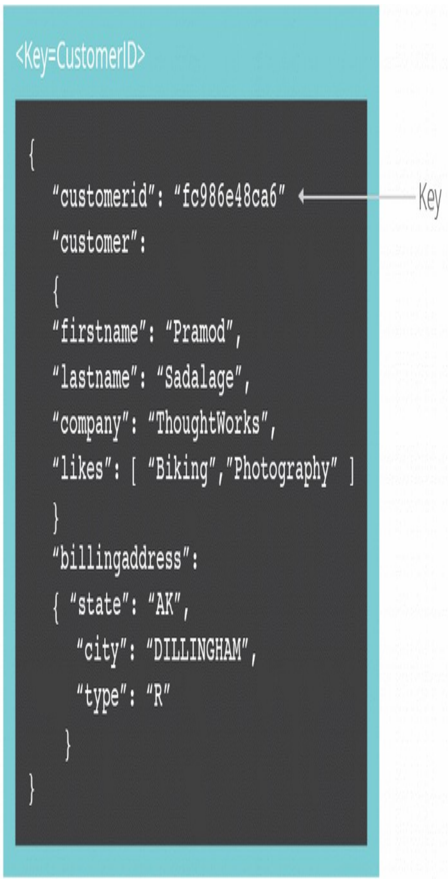
2.How does data get stored in NoSQL database?

NoSQL databases can broadly be categorized in four types.

Key-Value databases

Key-value stores are the simplest NoSQL data stores to use from an API perspective. The client can either get the value for the key, put 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.
Some of the popular key-value databases are Riak, Redis (often referred to as Data Structure server), Memcached and its flavors, Berkeley DB, upscaledb (especially suited for embedded use), Amazon DynamoDB (not open-source), Project Voldemort and Couchbase.
All key-value databases are not the same, there are major differences between these products, for example: Memcached data is not persistent while in Riak it is, these features are important when implementing certain solutions. Lets consider we need to implement caching of user preferences, implementing them in memcached means when the node goes down all the data is lost and needs to be refreshed from source system, if we store the same data in Riak we may not need to worry about losing data but we must also consider how to update stale data. Its important to not only choose a key-value database based on your requirements, it's also important to choose which key-value database.

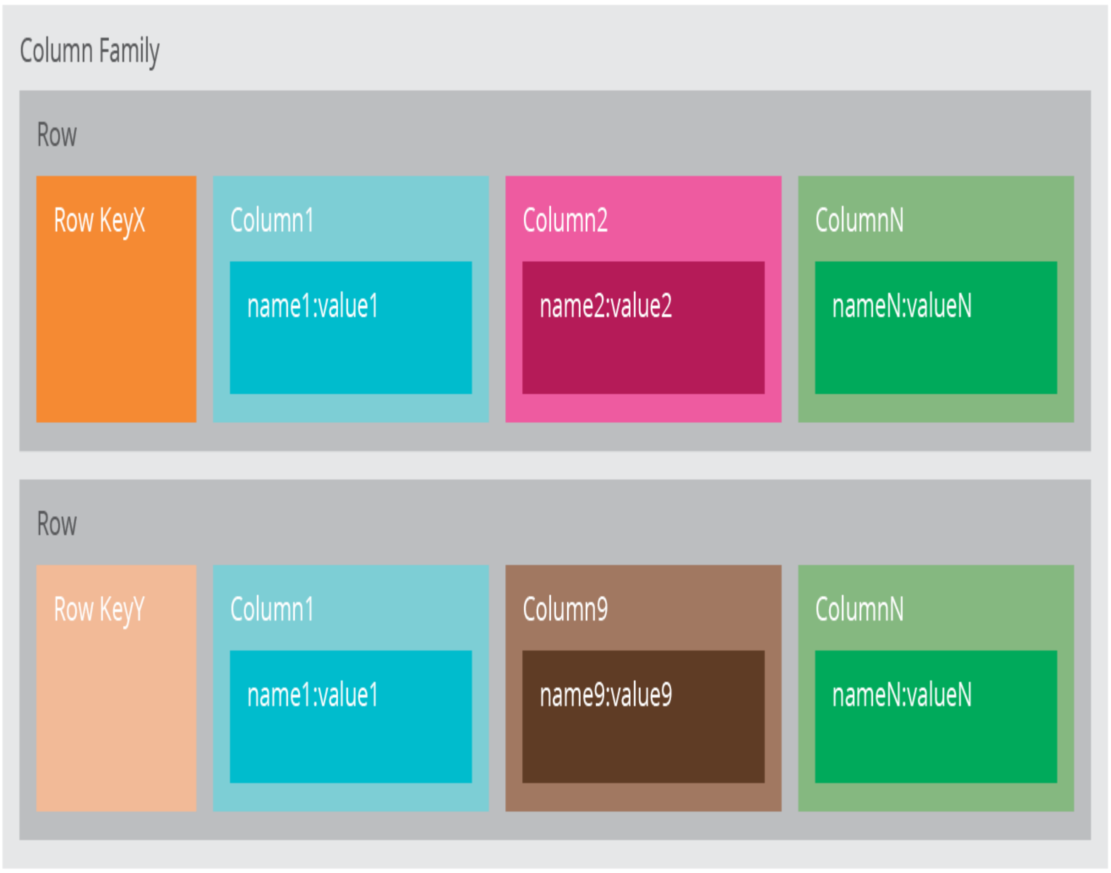
Document databases



```
{
  "customerid": "fc986e48ca6"
  "customer":
  {
  "firstname": "Pramod",
  "lastname": "Sadalage",
  "company": "ThoughtWorks",
  "likes": [ "Biking","Photography" ]
  }
  "billingaddress":
  { "state": "AK",
    "city": "DILLINGHAM",
    "type": "R"
   }
}
```

Documents are the main concept in document databases. The database stores and retrieves documents, which can be XML, JSON, BSON, and so on. These documents are self-describing, hierarchical tree data structures which can consist of maps, collections, and scalar values. The documents stored are similar to each other but do not have to be exactly the same. Document databases store documents in the value part of the key-value store; think about document databases as key-value stores where the value is examinable. Document databases such as MongoDB provide a rich query language and constructs such as database, indexes etc allowing for easier transition from relational databases.
Some of the popular document databases we have seen are MongoDB, CouchDB , Terrastore, OrientDB, RavenDB, and of course the well-known and often reviled Lotus Notes that uses document storage.

Column family stores



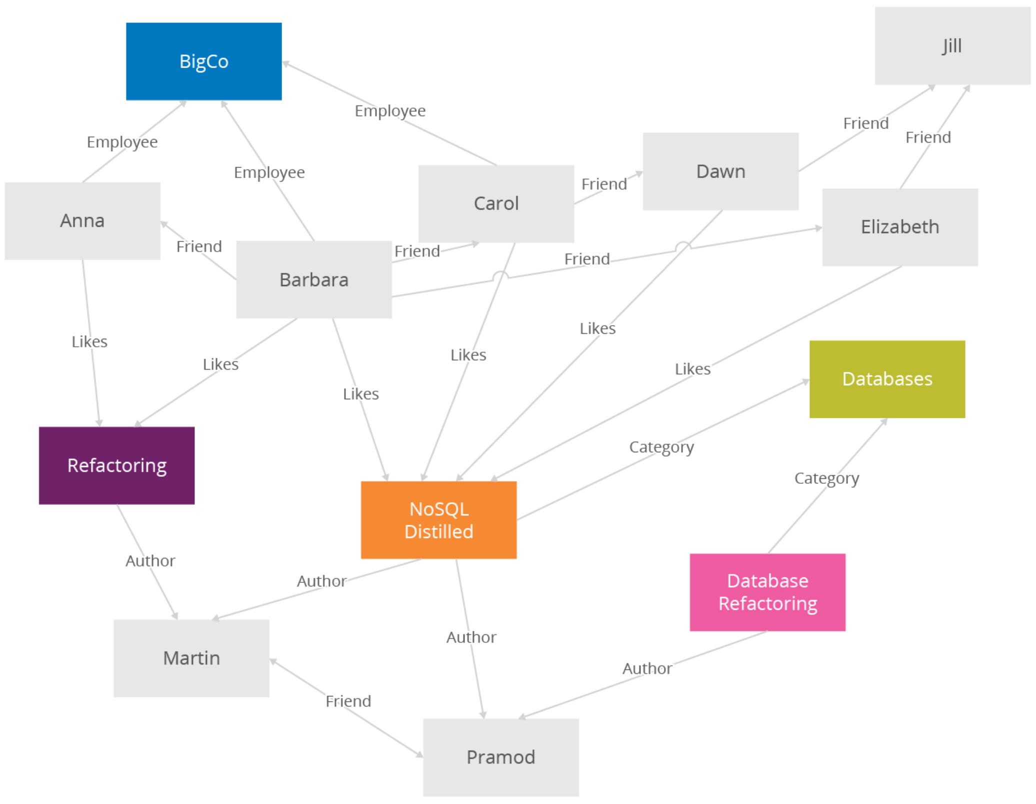
Column-family databases store data in column families as rows that have many columns associated with a row key (Figure 10.1). Column families are groups of related data that is often accessed together. For a Customer, we would often access their Profile information at the same time, but not their Orders.

Each column family can be compared to a container of rows in an RDBMS table where the key identifies the row and the row consists of multiple columns. The difference is that various rows do not have to have the same columns, and columns can be added to any row at any time without having to add it to other rows.

When a column consists of a map of columns, then we have a super column. A super column consists of a name and a value which is a map of columns. Think of a super column as a container of columns.

Cassandra is one of the popular column-family databases; there are others, such as HBase, Hypertable, and Amazon DynamoDB. Cassandra can be described as fast and easily scalable with write operations spread across the cluster. The cluster does not have a master node, so any read and write can be handled by any node in the cluster.

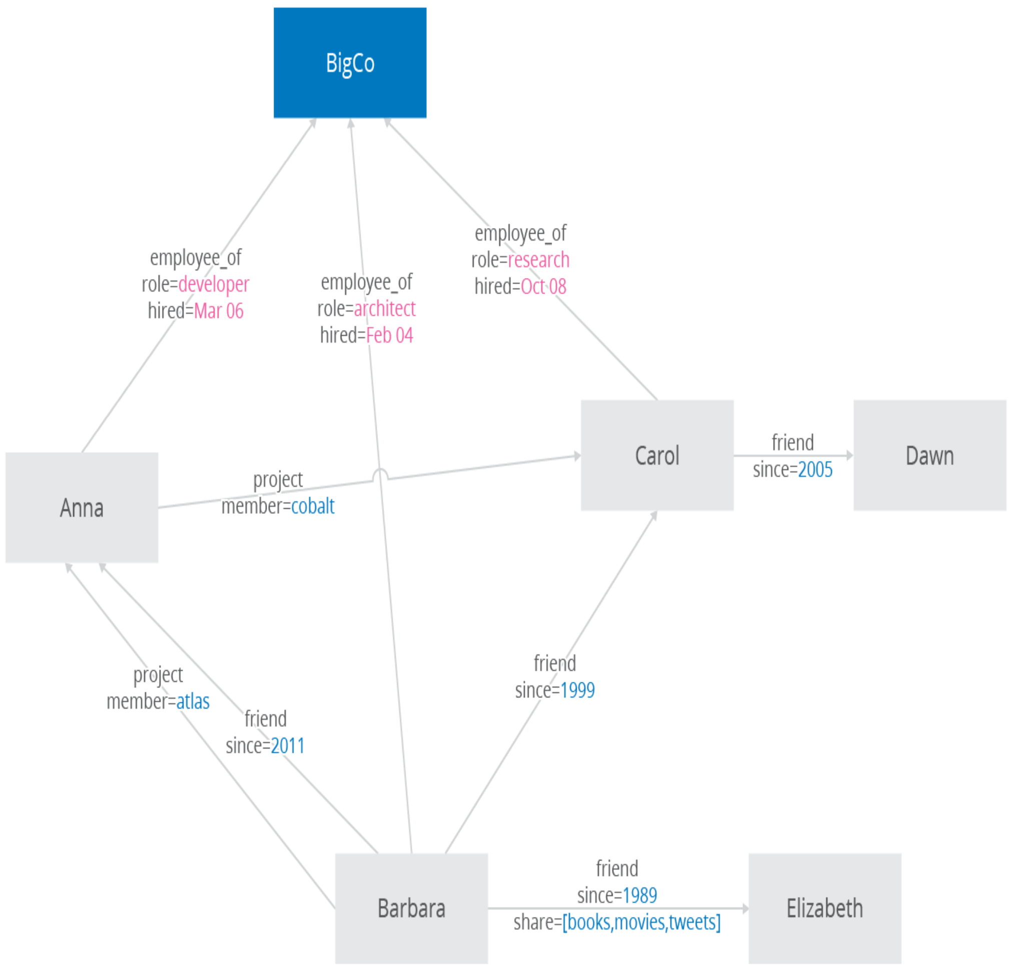
Graph Databases



Graph databases allow you to store entities and relationships between these entities. Entities are also known as nodes, which have properties. Think of a node as an instance of an object in the application. Relations are known as edges that can have properties. Edges have directional significance; nodes are organized by relationships which allow you to find interesting patterns between the nodes. The organization of the graph lets the data to be stored once and then interpreted in different ways based on relationships.

Usually, when we store a graph-like structure in RDBMS, it's for a single type of relationship ("who is my manager" is a common example). Adding another relationship to the mix usually means a lot of schema changes and data movement, which is not the case when we are using graph databases. Similarly, in relational databases we model the graph beforehand based on the Traversal we want; if the Traversal changes, the data will have to change.

In graph databases, traversing the joins or relationships is very fast. The relationship between nodes is not calculated at query time but is actually persisted as a relationship. Traversing persisted relationships is faster than calculating them for every query.



Nodes can have different types of relationships between them, allowing you to both represent relationships between the domain entities and to have secondary relationships for things like category, path, time-trees, quad-trees for spatial indexing, or linked lists for sorted access. Since there is no limit to the number and kind of relationships a node can have, they all can be represented in the same graph database.

Relationships are first-class citizens in graph databases; most of the value of graph databases is derived from the relationships. Relationships don't only have a type, a start node, and an end node, but can have properties of their own. Using these properties on the relationships, we can add intelligence to the relationship—for example, since when did they become friends, what is the distance between the nodes, or what aspects are shared between the nodes. These properties on the relationships can be used to query the graph.

Since most of the power from the graph databases comes from the relationships and their properties, a lot of thought and design work is needed to model the relationships in the domain that we are trying to work with. Adding new relationship types is easy; changing existing nodes and their relationships is similar to data migration, because these changes will have to be done on each node and each relationship in the existing data.

3.What is a column family in HBase?

HBase is a **column-oriented database** and the tables in it are sorted by row. The table schema defines only column families, which are the key value pairs. A table have multiple column families and each column family can have any number of columns. Subsequent column values are stored contiguously on the disk. Each cell value of the table has a timestamp.
In short, in an Hbase:

- •Table is a collection of rows.
- •Row is a collection of column families.
- •Column family is a collection of columns.
- •Column is a collection of key value pairs.

4. How many maximum number of columns can be added to HBase table?

The table schema defines only column families, which are the key value pairs. A table have multiple column families and each column family can have any number of columns.

5.Why columns are not defined at the time of table creation in Hbase?

HBase data model consists of tables containing rows. Data is organized into column families grouping columns in each row**.** So We can not define columns at time table creation, we have to define column-families.
An HBase table is made of column families which are the logical and physical grouping of columns. The columns in one family are stored separately from the columns in another family.
A single column family contains one or more columns, Column families must be defined at table creation time but columns can be added dynamically after table creation (if an insert statement states a column that does not exist for a column family it will create it).

6.How does data get managed in Hbase?

HBase is a **column-oriented database** and the tables in it are sorted by row. The table schema defines only column families, which are the key value pairs. A table have multiple column families and each column family can have any number of columns. Subsequent column values are stored contiguously on the disk. Each cell value of the table has a timestamp. In short, in an HBase:

- •Table is a collection of rows.
- •Row is a collection of column families.
- •Column family is a collection of columns.
- •Column is a collection of key value pairs.

7.What happens internally when new data gets inserted into HBase table?

Let us summarize the insert/write process:

•When the client gives a command to Write, Instruction is directed to **Write Ahead Log**.
•Once the log entry is done, the data to be written is forwarded to **MemStore** which is actually the RAM of the data node.
•Data in the memstore is sorted in the same manner as data in a HFile.
•When the memstore accumulates enough data, the entire sorted set is written to a new **HFile** in HDFS.
•Once writing data is completed, **ACK** (Acknowledgement) is sent to the client as a confirmation of task completed.

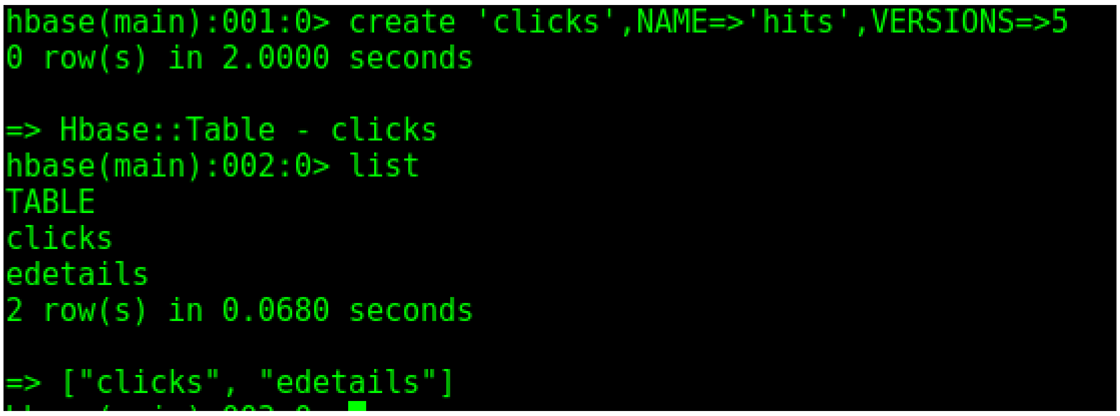
Task 2

1. Create an HBase table named 'clicks' with a column family 'hits' such that it should be able to store last 5 values of qualifiers inside 'hits' column family.

Command format:
Create <'table name'>,<'column family=>value'>,VERSIONS=>5

HBASE Shell command

create 'clicks',NAME=>'hits',VERSIONS=>5



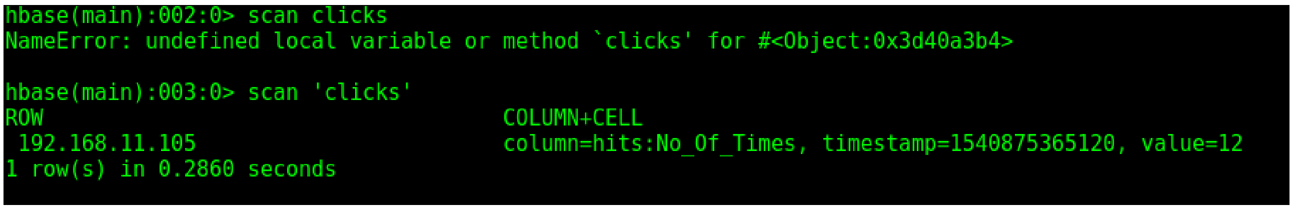
```
hbase(main):001:0> create 'clicks',NAME=>'hits',VERSIONS=>5
0 row(s) in 2.0000 seconds

=> Hbase::Table - clicks
hbase(main):002:0> list
TABLE
clicks
edetails
2 row(s) in 0.0680 seconds

=> ["clicks", "edetails"]
```

2. Add few records in the table and update some of them. Use IP Address as row-key. Scan the table to view if all the previous versions are getting displayed.

put 'clicks','192.168.11.105','hits:No_Of_Times','12'

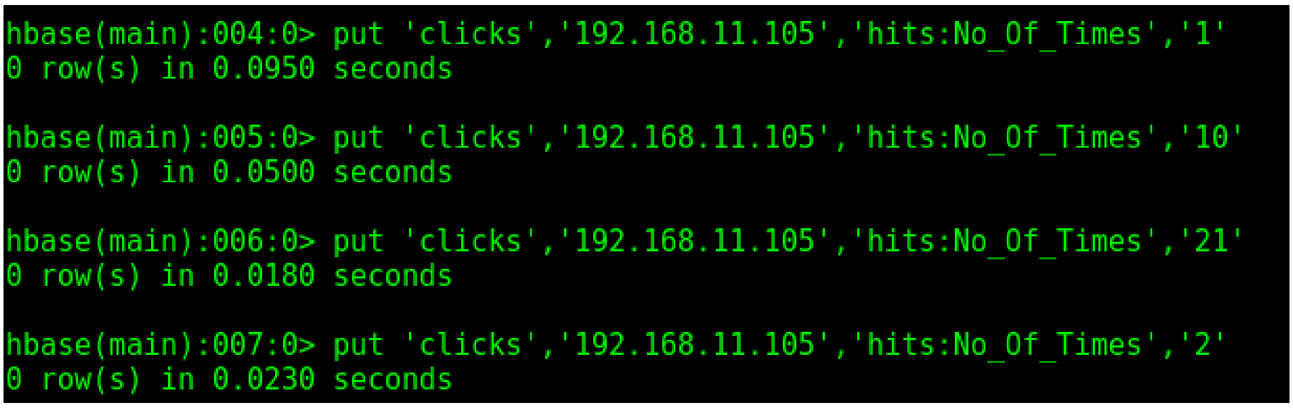


```
hbase(main):002:0> scan clicks
NameError: undefined local variable or method `clicks' for #<Object:0x3d40a3b4>

hbase(main):003:0> scan 'clicks'
ROW                                COLUMN+CELL
 192.168.11.105                    column=hits:No_Of_Times, timestamp=1540875365120, value=12
1 row(s) in 0.2860 seconds
```

Update the row by adding some values,

put 'clicks','192.168.11.105','hits:No_Of_Times','1'
put 'clicks','192.168.11.105','hits:No_Of_Times','10'
put 'clicks','192.168.11.105','hits:No_Of_Times','21'
put 'clicks','192.168.11.105','hits:No_Of_Times','2'



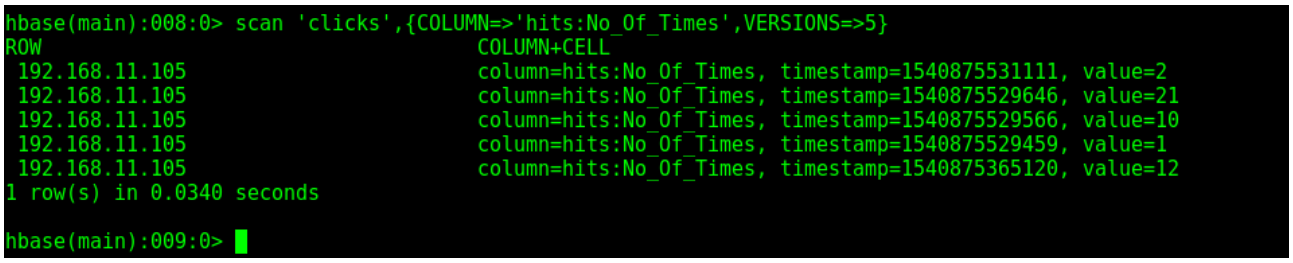
```
hbase(main):004:0> put 'clicks','192.168.11.105','hits:No_Of_Times','1'
0 row(s) in 0.0950 seconds

hbase(main):005:0> put 'clicks','192.168.11.105','hits:No_Of_Times','10'
0 row(s) in 0.0500 seconds

hbase(main):006:0> put 'clicks','192.168.11.105','hits:No_Of_Times','21'
0 row(s) in 0.0180 seconds

hbase(main):007:0> put 'clicks','192.168.11.105','hits:No_Of_Times','2'
0 row(s) in 0.0230 seconds
```

scan 'clicks',{COLUMN=>'hits:No_Of_Times',VERSIONS=>5}



```
hbase(main):008:0> scan 'clicks',{COLUMN=>'hits:No_Of_Times',VERSIONS=>5}
ROW                                      COLUMN+CELL
 192.168.11.105                          column=hits:No_Of_Times, timestamp=1540875531111, value=2
 192.168.11.105                          column=hits:No_Of_Times, timestamp=1540875529646, value=21
 192.168.11.105                          column=hits:No_Of_Times, timestamp=1540875529566, value=10
 192.168.11.105                          column=hits:No_Of_Times, timestamp=1540875529459, value=1
 192.168.11.105                          column=hits:No_Of_Times, timestamp=1540875365120, value=12
1 row(s) in 0.0340 seconds

hbase(main):009:0> 
```