Session 10: HBASE BASICS Assignment 1

- By Prachi Mohite

Task 1

1.1 What is NoSQL data base?

Answer:

NoSQL is an approach to database design that can accommodate 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.

Motivations for this approach include:

- simplicity of design
- simpler horizontal scaling to clusters of machines
- finer control over availability.

The data structures used by NoSQL databases e.g. key-value, wide column, graph, or document are different from those used by default in relational databases, make some operations faster in NoSQL. Sometimes the data structures used by NoSQL databases are also viewed as "more flexible" than relational database tables.
Many NoSQL stores compromise consistency in favor of availability, partition tolerance, and speed.
Barriers to the greater adoption of NoSQL stores include:
1. the use of low-level query languages
2. lack of standardized interfaces,
3. huge previous investments in existing relational databases Most NoSQL databases offer a concept of "eventual consistency" in which database changes are propagated to all nodes, so queries for data might not return updated data immediately or might result in reading data that is not accurate, this problem is known as stale reads.

Additionally, some NoSQL systems may exhibit lost writes and other forms of data loss. Some NoSQL systems provide concepts such as write-ahead logging to avoid data loss.

1.2 How does data get stored in NoSQl database?

Answer:

In the in-memory databases like Redis/CouchBase/Tarantool/Aerospike everything is stored in RAM in balanced trees like RB-Tree or in hash tables. All the writes are applied on both RAM and disk, but on disk it goes in an append-only way. A file append can be done as fast as

100Mbytes per second on a normal magnetic disk. If a record size is, say, 1K, then the data will be written at 100krps.

In the on-disk NoSQL databases and db-engines like Cassandra/HBase/RocksDB/LevelDB/Sophia the main idea is that you have a snapshot file and a write ahead log (WAL) file. Snapshot contains already prepared data in a form of B-Tree with upper levels of that tree being permanently in RAM, that can be accesses for reading by doing only one disk seek. A WAL contains all the new changes on top of a current snapshot. A snapshot file is being totally rebuilt on a regular basis using current snapshot and a WAL. All the writes are done nearly as fast as with in-memory databases. "Nearly" because disk is partially busy by doing regular snapshot converting that was described earlier. Reads are significantly slower than that are in in-memory databases, because they take at least one disk seek, but good news is that they can be cached in optimized in-memory structures like RB-Trees/hash tables.

NoSQL Database Types

- **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.
- **Graph stores** are used to store information about networks of data, such as social connections.
- **Key-value stores** are the simplest NoSQL databases. Every single item in the database is stored as an attribute name (or 'key'), together with its value.
- **Wide-column** stores such as HBase are optimized for queries over large datasets, and store columns of data together, instead of rows.

1.3 What is a column family in HBase?
Answer :

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.

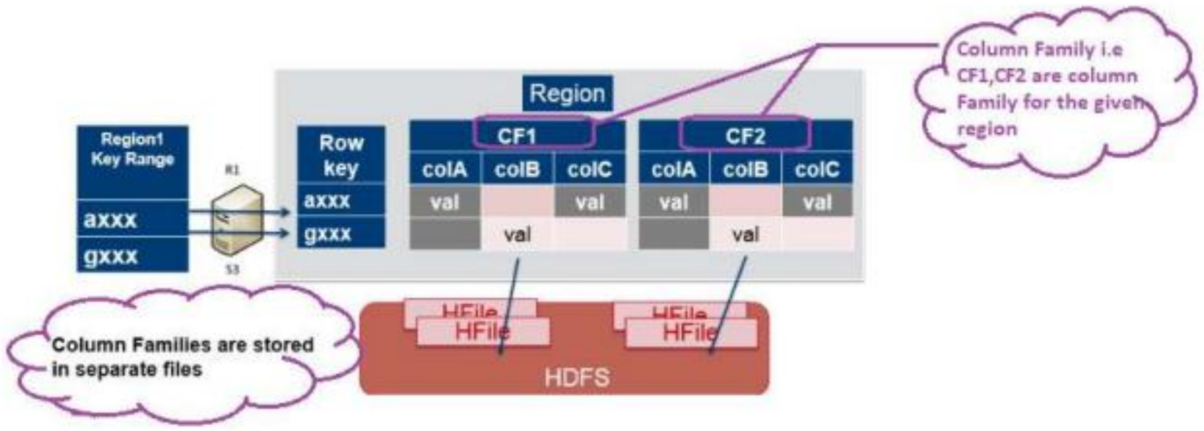
- Row is a collection of column families.
- Column family is a collection of columns.
- Column is a collection of key value pairs.

Columns in Apache HBase are grouped into column families. All column members of a column family have the same prefix. For example, the columns courses:history and courses:math are both members of the courses column family. The colon character (:) delimits the column family from the . The column family prefix must be composed of printable characters. The qualifying tail, the column family qualifier, can be made of any arbitrary bytes. Column families must be declared up front at schema definition time whereas columns do not need to be defined at schema time but can be conjured on the fly while the table is up an running.

Physically, all column family members are stored together on the filesystem. Because tunings and storage specifications are done at the column family level, it is advised that all column family members have the same general access pattern and size characteristics.

e.g.

Row Key	**Column Family: {Column Qualifier:Version:Value}**
00001	CustomerName: {‘FN’: 1383859182496:‘John’, ‘LN’: 1383859182858:‘Smith’, ‘MN’: 1383859183001:’Timothy’, ‘MN’: 1383859182915:’T’} ContactInfo: {‘EA’: 1383859183030:‘John.Smith@xyz.com’, ’SA’: 1383859183073:’1 Hadoop Lane, NY 11111’}
00002	CustomerName: {‘FN’: 1383859183103:‘Jane’, ‘LN’: 1383859183163:‘Doe’, ContactInfo: { ’SA’: 1383859185577:’7 HBase Ave, CA 22222’}


Column Family i.e CF1,CF2 are column Family for the given region
Region
Region1 Key Range
axxx
gxxx
Row key
axxx
gxxx
CF1
CF2
colA
colB
colC
val
HFile
HDFS
Column Families are stored in separate files

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

Answer :

There is no limit on number of column families in HBase, in theory. In reality, there are several factors which can limit useable number of column families in HBase:

1. HBase Admin web UI usability. It will be very hard to show even 100s of column families in a table configuration page.
2. HDFS practical limit of maximum number of files. Say, 100m. If your table has N regions, M column families you will need NxM directories to support this configuration. Every region/column family, in turn, can contain up to K store files (depends on write load and many other configuration options). With very modest N = 100 and K = 10 we can say practical limit of maximum number of column families is less than 100K. Usually, much less than 100K.

Each column family has its own directory in HDFS and set of store files and, from performance point of view, the fewer directories (column families) you have the better performance for scan operations you get.

1.5 Why columns are not defined at the time of table creation in HBase?

Answer:

Column qualifiers need not be specified in advance, because Column qualifiers need not be consistent between rows. Like row keys, column qualifiers do not have a data type and are always treated as a byte[ ]. Column qualifiers are dynamic and can be defined at write time. They are stored as byte[ ] so you can even put data in them

1.6 How does data get managed in HBase?
Answer:

HBase is a column-oriented database and data is stored in tables. The tables are sorted by RowId. As shown below, HBase has RowId, which is the collection of several column families that are present in the table.

he column families that are present in the schema are key-value pairs. If we observe in detail each column family having a multiple numbers of columns. The column values stored in to disk memory. Each cell of the table has its own Meta data like time stamp and other information.

Coming to HBase the following are the key terms representing table schema

- **Table**: Collection of rows present.
- **Row**: Collection of column families.
- **Column Family:** Collection of columns.
- **Column**: Collection of key-value pairs.
- **Namespace**: Logical grouping of tables.
- **Cell**: A {row, column, version} tuple exactly specifies a cell definition in HBase.

HBase is a structured noSQL database that rides atop Hadoop. You can store Hbase data in the HDFS . And you can also store HBase data in Amazon S3, which has an entirely different architecture. HBase is structured because it has the row-and-column structure of an RDBMS, like Oracle. But it a column-oriented database and not a row-oriented one. What HBase does is provide random access to big data. Hadoop does not: it is a batch system that only writes files but does not update them. HBase stores data in a memory table and then flushes it to storage in massive writes to disk. So that gives it the high throughput needed for big data applications.

1.7 What happens internally when new data gets inserted into HBase table?
Answer:
To create data in an HBase table, the following commands and methods are used:

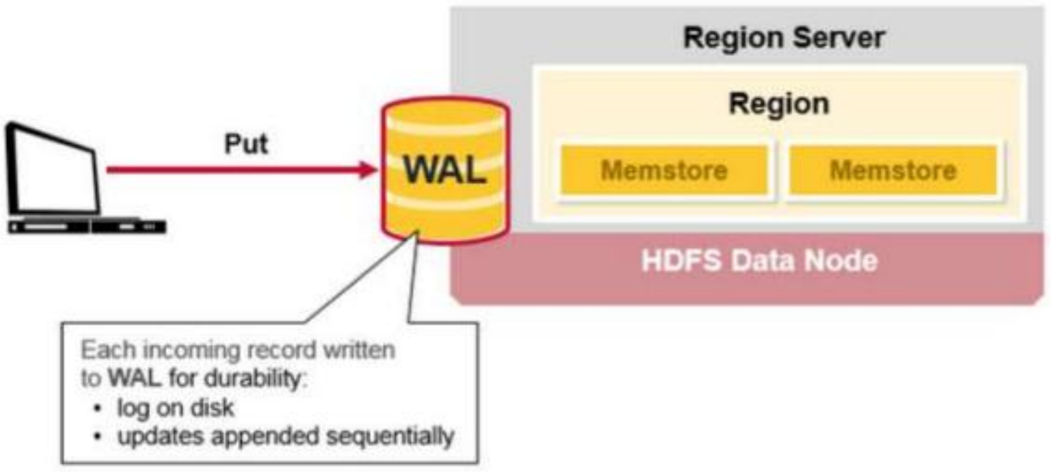
- put command,
- add() method of Put class, and
- put() method of HTable class.

When the client gives a command to Write, the following steps occur:

- Instruction is directed to Write Ahead Log and first, writes important logs to it. Although it is not the area where the data is stored, it is done for the fault tolerant purpose. So, later if any error occurs while writing data, HBase always has WAL to look into.
- Once the log entry is done, the data to be written is forwarded to MemStore which is actually the RAM of the data node. All the data is written in MemStore which is faster than RDBMS (Relational databases).
- Later, the data is dumped in HFile, where the actual data is stored in HDFS. If the MemCache is full, the data is stored in HFile directly.
- Once writing data is completed, ACK (Acknowledgement) is sent to the client as a confirmation of task completed.

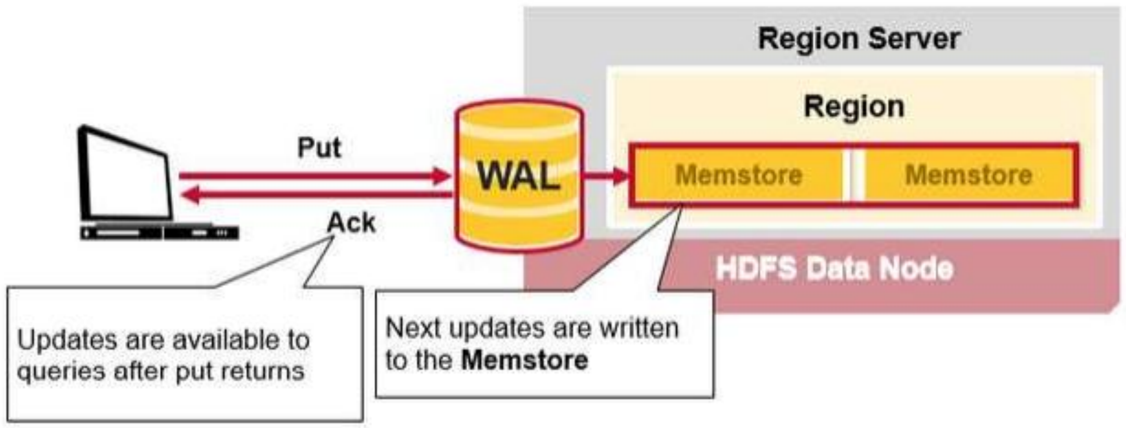
HBase Write Steps (1)

When the client issues a Put request, the first step is to write the data to the write-ahead log, the WAL: ¬ Edits are appended to the end of the WAL file that is stored on disk. ¬ The WAL is used to recover not-yet-persisted data in case a server crashes

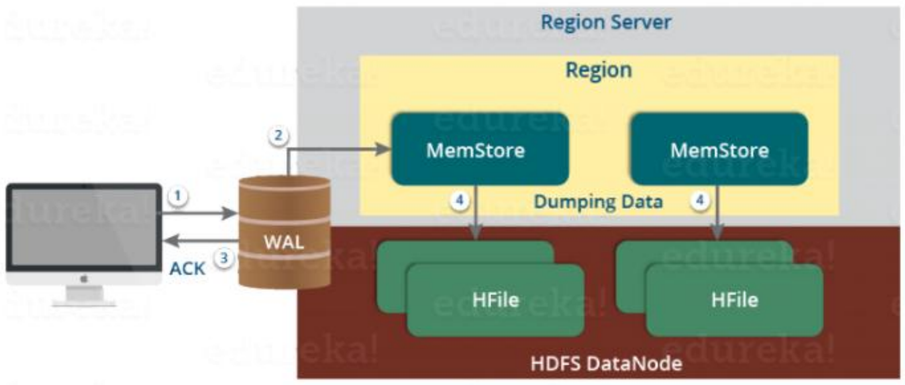

Region Server
Region
Put
WAL
Memstore
Memstore
HDFS Data Node
Each incoming record written to WAL for durability:
• log on disk
• updates appended sequentially

HBase Write Steps (2)

Once the data is written to the WAL, it is placed in the MemStore. Then, the put request acknowledgement returns to the client.


Region Server
Region
Put
WAL
Memstore
Memstore
Ack
HDFS Data Node
Updates are available to queries after put returns
Next updates are written to the **Memstore**

This below image explains the write mechanism in HBase.



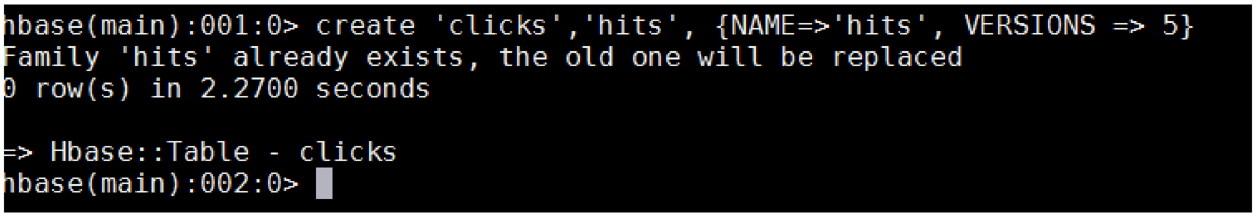
<u>Task 2</u>

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.

Solution Approach:

- Need to use the Create command
- As need to save the last 5 values of qualifiers inside hits column we need to maintain the history of that column which is nothing but versions of a row in HBase table.

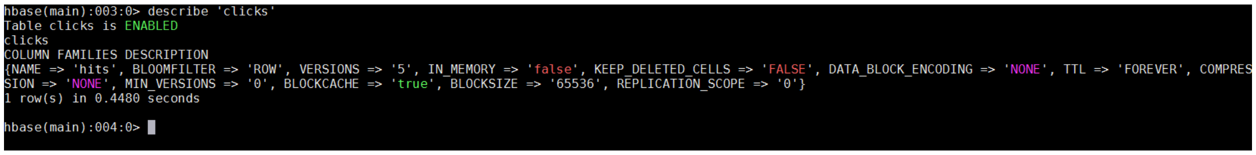
- Hbase allows you to have multiple versions of a row. This arises because data changes are not applied in place, instead a change results in a new version. To control how this happens you specify the number of versions



```
hbase(main):001:0> create 'clicks','hits', {NAME=>'hits', VERSIONS => 5}
Family 'hits' already exists, the old one will be replaced
0 row(s) in 2.2700 seconds

=> Hbase::Table - clicks
hbase(main):002:0>
```

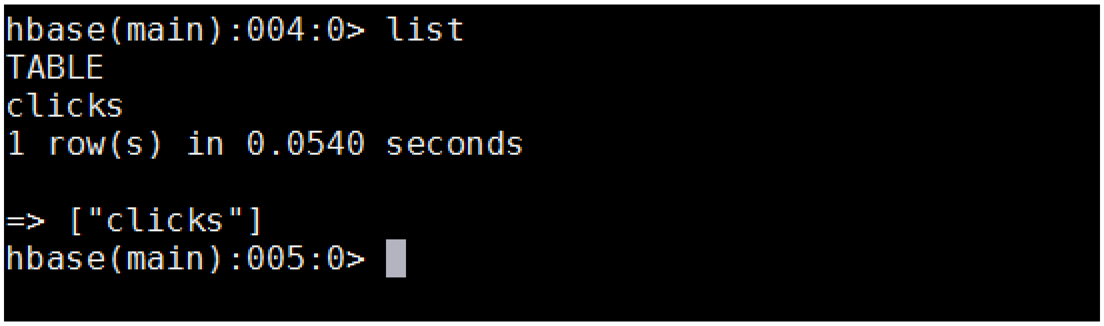
To verify if table is created we can use

- Describe <tableName>



```
hbase(main):003:0> describe 'clicks'
Table clicks is ENABLED
clicks
COLUMN FAMILIES DESCRIPTION
{NAME => 'hits', BLOOMFILTER => 'ROW', VERSIONS => '5', IN_MEMORY => 'false', KEEP_DELETED_CELLS => 'FALSE', DATA_BLOCK_ENCODING => 'NONE', TTL => 'FOREVER', COMPRES
SION => 'NONE', MIN_VERSIONS => '0', BLOCKCACHE => 'true', BLOCKSIZE => '65536', REPLICATION_SCOPE => '0'}
1 row(s) in 0.4480 seconds

hbase(main):004:0>
```

- List



```
hbase(main):004:0> list
TABLE
clicks
1 row(s) in 0.0540 seconds

=> ["clicks"]
hbase(main):005:0>
```

1.2 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.

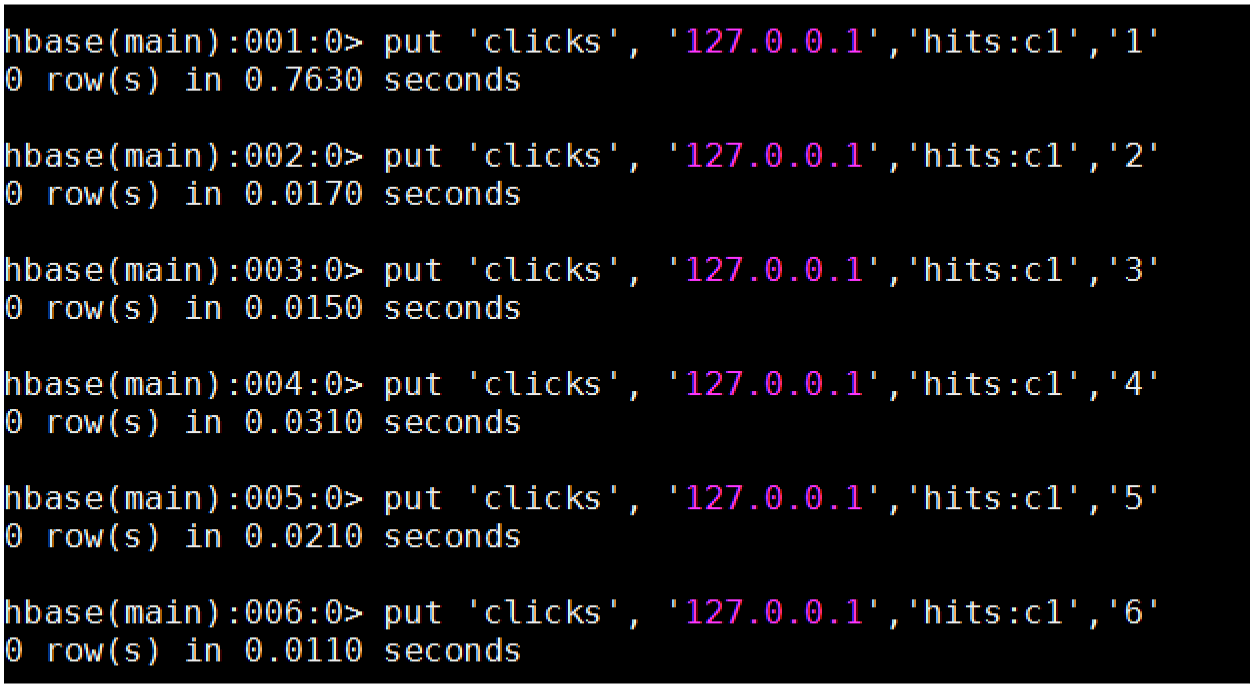
To create data in an HBase table, the following commands and methods are used:

- put command, - via hbase shell
- add() method of Put class – via JAVA API
- put() method of HTable class via JAVA API

Here we will be using put command through hbase shell

```
put ’<table name>’,’row1’,’<colfamily:colname>’,’<value>’
```

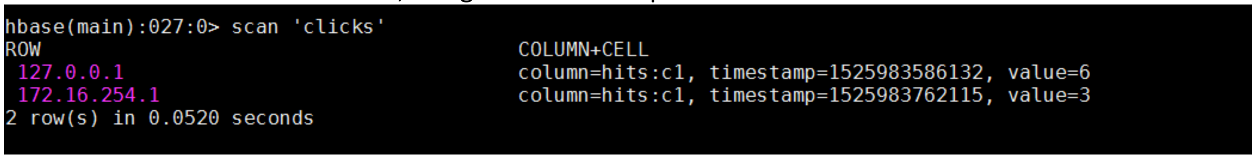
put 'clicks', '172.16.254.1', 'hits:c1', '1'



```
hbase(main):001:0> put 'clicks', '127.0.0.1','hits:c1','1'
0 row(s) in 0.7630 seconds

hbase(main):002:0> put 'clicks', '127.0.0.1','hits:c1','2'
0 row(s) in 0.0170 seconds

hbase(main):003:0> put 'clicks', '127.0.0.1','hits:c1','3'
0 row(s) in 0.0150 seconds

hbase(main):004:0> put 'clicks', '127.0.0.1','hits:c1','4'
0 row(s) in 0.0310 seconds

hbase(main):005:0> put 'clicks', '127.0.0.1','hits:c1','5'
0 row(s) in 0.0210 seconds

hbase(main):006:0> put 'clicks', '127.0.0.1','hits:c1','6'
0 row(s) in 0.0110 seconds
```

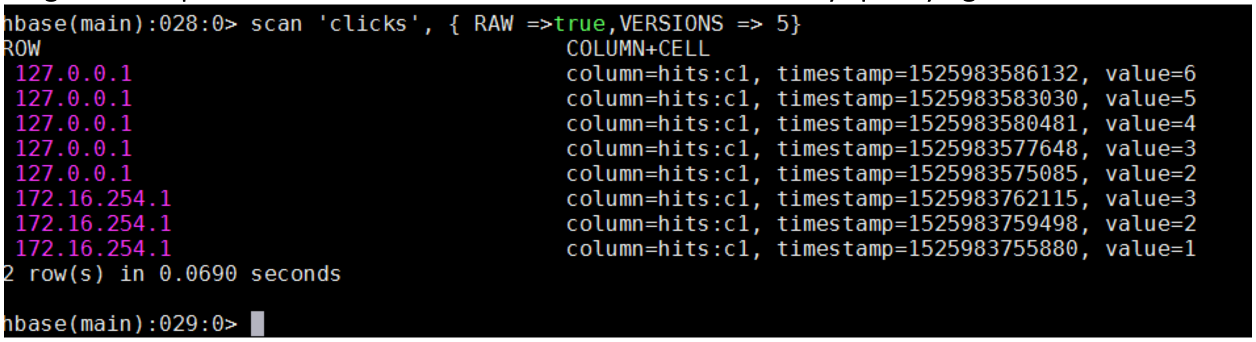
Scan the table

Scan command without versions , will give the latest update on the table

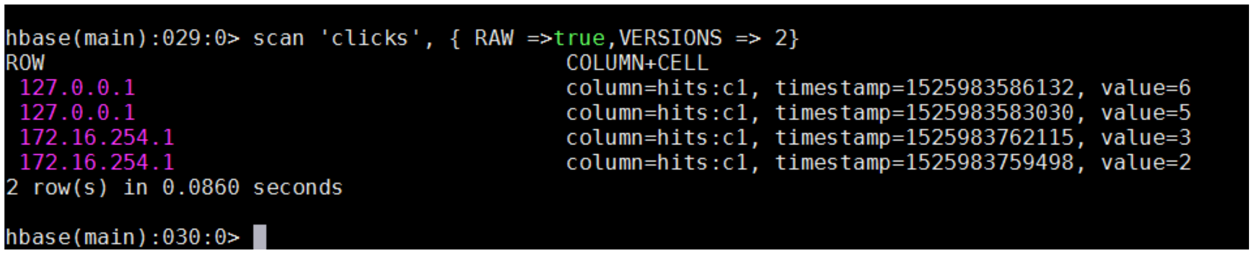


```
hbase(main):027:0> scan 'clicks'
ROW                                    COLUMN+CELL
 127.0.0.1                             column=hits:c1, timestamp=1525983586132, value=6
 172.16.254.1                          column=hits:c1, timestamp=1525983762115, value=3
2 row(s) in 0.0520 seconds
```

To get the all past available version need to ask for all version by specifying the versions count



```
hbase(main):028:0> scan 'clicks', { RAW =>true,VERSIONS => 5}
ROW                                        COLUMN+CELL
 127.0.0.1                                 column=hits:c1, timestamp=1525983586132, value=6
 127.0.0.1                                 column=hits:c1, timestamp=1525983583030, value=5
 127.0.0.1                                 column=hits:c1, timestamp=1525983580481, value=4
 127.0.0.1                                 column=hits:c1, timestamp=1525983577648, value=3
 127.0.0.1                                 column=hits:c1, timestamp=1525983575085, value=2
 172.16.254.1                              column=hits:c1, timestamp=1525983762115, value=3
 172.16.254.1                              column=hits:c1, timestamp=1525983759498, value=2
 172.16.254.1                              column=hits:c1, timestamp=1525983755880, value=1
2 row(s) in 0.0690 seconds

hbase(main):029:0>
```



```
hbase(main):029:0> scan 'clicks', { RAW =>true,VERSIONS => 2}
ROW                                     COLUMN+CELL
 127.0.0.1                              column=hits:c1, timestamp=1525983586132, value=6
 127.0.0.1                              column=hits:c1, timestamp=1525983583030, value=5
 172.16.254.1                           column=hits:c1, timestamp=1525983762115, value=3
 172.16.254.1                           column=hits:c1, timestamp=1525983759498, value=2
2 row(s) in 0.0860 seconds

hbase(main):030:0> 
```