Assginment-10

Task-1:

1. **What is NoSQL data base?**

Ans:- NoSQL encompasses the wide variety of different database technologies that were developed in response to the demands presented modern applications. Developers are working with applications that create huge volumes of data with rapidly changing datatypes with structured, semi-structured, unstructured and polymorphic data.

Now-a-days all are using agile sprints which will push the code every week or two, sometimes multiple times a day.

Relational databases were not designed to cope with the scale and agility challenges that face modern applications, nor were they built to take advantage of the commodity storage and processing power available today.

Examples of NoSQL databases are HBase, MongoDB, Cassandra etc.

2. **How does data get stored in NoSQl database?**

Ans:- There are various NoSQL Databases. Each one uses a different method to store data. Some might use column store, some document, some graph, etc., Each database has its own unique characteristics.

You can have a look at the various categories of NoSQL databases here: NOSQL Databases.

I have personally played around with MongoDB for a while now and it has always impressed me. MongoDB is an document store, where data is stored as Key: Value pairs in JSON format.

For Example-:

```
{

 name: "sid",

 phone: 1234567890,

 address:

 {

street: "1234 Some_XYZ Pkwy" ,
```

Apt: 1001,

City: "Richardson",

State: "Texas"

}

}

3. What is a column family in HBase?

Ans:- 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. If you have data that is not often queried, assign that data to a separate column family.

The column family and column qualifier names are repeated for each row. Therefore, keep the names as short as possible to reduce the amount of data that HBase stores and reads. For example, use f:q instead of mycolumnfamily:mycolumnqualifier.

Because column families are stored in separate HFiles, keep the number of column families as small as possible. You also want to reduce the number of column families to reduce the frequency of MemStore flushes, and the frequency of compactions. And, by using the smallest number of column families possible, you can improve the LOAD time and reduce disk consumption.

The following example illustrates a best practice for defining column families:

```
CREATE HBASE TABLE ORDERS
(O_ORDERKEY BIGINT, O_CUSTKEY INTEGER, O_ORDERSTATUS CHAR(1),
 O_TOTALPRICE FLOAT, O_ORDERDATE string, O_ORDERPRIORITY CHAR(15),
O_CLERK CHAR(15), O_SHIPPRIORITY INTEGER, O_COMMENT VARCHAR(79)
)
COLUMN MAPPING (
KEY MAPPED BY
(O_ORDERDATE,O_CUSTKEY,O_ORDERKEY),
f:d MAPPED BY
(O_ORDERSTATUS,O_TOTALPRICE,O_ORDERPRIORITY,
```

O_SHIPPRIORITY,O_COMMENT,O_CLERK)

)...;

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

Ans:- HBase currently does not do well with anything above two or three column families so we should keep the number of column families in your schema low. Currently, flushing and compactions are done on a per Region basis so if one column family is carrying the bulk of the data bringing on flushes, the adjacent families will also be flushed though the amount of data they carry is small. We can introduce a second and third column family in the case where data access is usually column scoped; i.e. you query one column family or the other but usually not both at the one time.

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

Ans:- Column families must be declared up front at schema definition time whereas columns
do not need to be defined at schema time because it can be conjured on the fly while the table is up on running.

Column families are part of the schema of the table. You can add them at runtime with an online schema change. But you wouldn't add them dynamically the way that you can dynamically create new "columns" in an HBase table, if that's what you had in mind.

The reason column families are part of the schema and would require a schema change is that they profoundly impact the way the data is stored, both on disk and in memory. Each column family has its own set of HFiles, and its own set of data structures in memory of the RegionServer. It would be pretty expensive to dynamically create or start using new column families.

Column families are only needed when you need to configure differently various parts of a table (for instance you want some columns to have a TTL and others to not expire), or when you want to control the locality of accesses (things accessed together should better be in the same column family if you want good performance, as the cost of operations grows linearly with the number of column families). So, again, because of those specialized reasons, it doesn't make sense to dynamically add new column families at runtime the way you would add regular "columns" within a family.

6. **How does data get managed in HBase?**

Ans:- HBase is not a relational database and requires a different approach to managing data. HBase actually defines a four-dimensional data model to manage data:
Row Key: Each row has a unique row key; the row key does not have a data type and is treated internally as a byte array.
Column Family: Data inside a row is organized into column families; each row has the same set of column families, but across rows, the same column families do not need the same column qualifiers. Under-the-hood, HBase stores column families in their own data files, so they need to be defined upfront, and changes to column families are difficult to make.
Column Qualifier: Column families define actual columns, which are called column qualifiers. You can think of column qualifiers as the columns themselves.A unique combination of row key, column family and column qualifier forms a cell. Data contained in a cell is referred to as cell value. There is no concept of data type when referring to cell values and they are stored as bytearrays. Indexing and sorting only happens on the row key.
Version: Each column can have a configurable number of versions, and you can access the data for a specific version of a column qualifier.

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

Ans:- The write path is how an HBase completes put or delete operations. This path begins at a client, moves to a region server, and ends when data eventually is written to an HBase data file called an HFile. Included in the design of the write path are features that HBase uses to prevent data loss in the event of a region server failure. Therefore understanding the write path can provide insight into HBase's native data loss prevention mechanism.

Each HBase table is hosted and managed by sets of servers which fall into three categories:

- ✓ One active master server
- ✓ One or more backup master servers
- ✓ Many region servers

Region servers contribute to handling the HBase tables. Because HBase tables can be large, they are broken up into partitions called regions. Each region server handles one or more of these regions. Note that because region servers are the only servers that serve HBase table data, a master server crash cannot cause data loss.

HBase data is organized similarly to a sorted map, with the sorted key space partitioned into different shards or regions. An HBase client updates a table by invoking put or delete commands. When a client requests a change, that request is routed to a region server right away by default. However, programmatically, a client can cache the changes in the client side, and flush these changes to region servers in a batch, by turning the autoflush off. If autoflush is turned off, the changes are cached until flush-commits is invoked, or the buffer is full depending on the buffer size set programmatically or configured with parameter “hbase.client.write.buffer”.

Since the row key is sorted, it is easy to determine which region server manages which key. A change request is for a specific row. Each row key belongs to a specific region which is served by a region server. So based on the put or delete’s key, an HBase client can locate a proper region server. At first, it locates the address of the region server hosting the -ROOT- region from the ZooKeeper quorum. From the root region server, the client finds out the location of the region server hosting the -META- region. From the meta region server, then we finally locate the actual region server which serves the requested region. This is a three-step process, so the region location is cached to avoid this expensive series of operations. If the cached location is invalid (for example, we get some unknown region exception), it’s time to re-locate the region and update the cache.

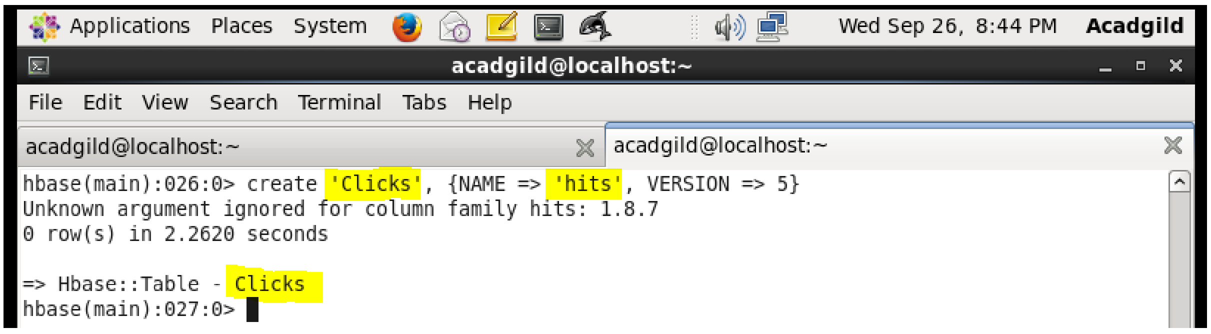
After the request is received by the right region server, the change cannot be written to a HFile immediately because the data in a HFile must be sorted by the row key. This allows searching for random rows efficiently when reading the data. Data cannot be randomly inserted into the HFile. Instead, the change must be written to a new file. If each update were written to a file, many small files would be created. Such a solution would not be scalable nor efficient to merge or read at a later time. Therefore, changes are not immediately written to a new HFile.

Instead, each change is stored in a place in memory called the memstore, which cheaply and efficiently supports random writes. 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. Completing one large write task is efficient and takes advantage to HDFS’ strengths.

Task -2:

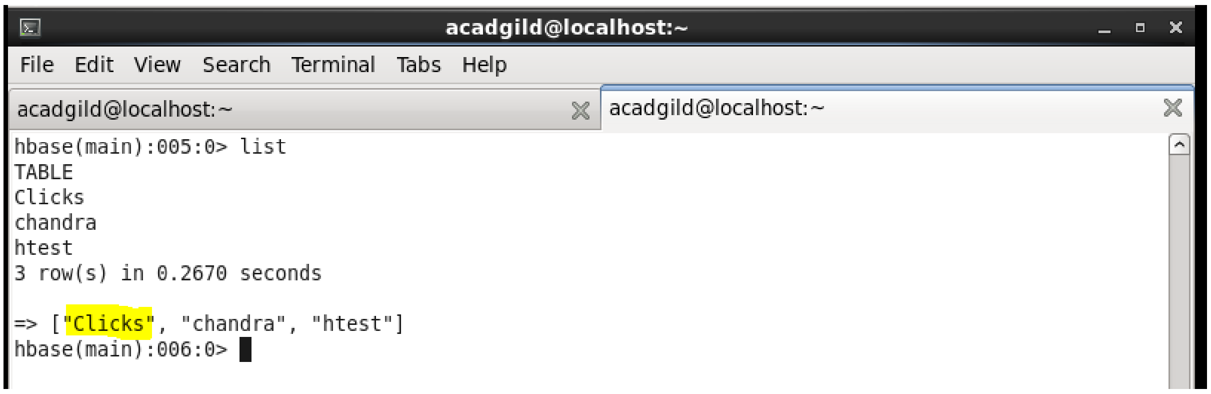
1. **Create an HBase table named 'clicks' with a column family 'hits' such that it should be able to store last 5 values of qualifiers inside 'hits' column family.**

Created table **'Clicks'** with column family **'hits'**.


Applications Places System
Wed Sep 26, 8:44 PM Acadgild
acadgild@localhost:~
File Edit View Search Terminal Tabs Help
acadgild@localhost:~
acadgild@localhost:~

```
hbase(main):026:0> create 'Clicks', {NAME => 'hits', VERSION => 5}
Unknown argument ignored for column family hits: 1.8.7
0 row(s) in 2.2620 seconds

=> Hbase::Table - Clicks
hbase(main):027:0>
```

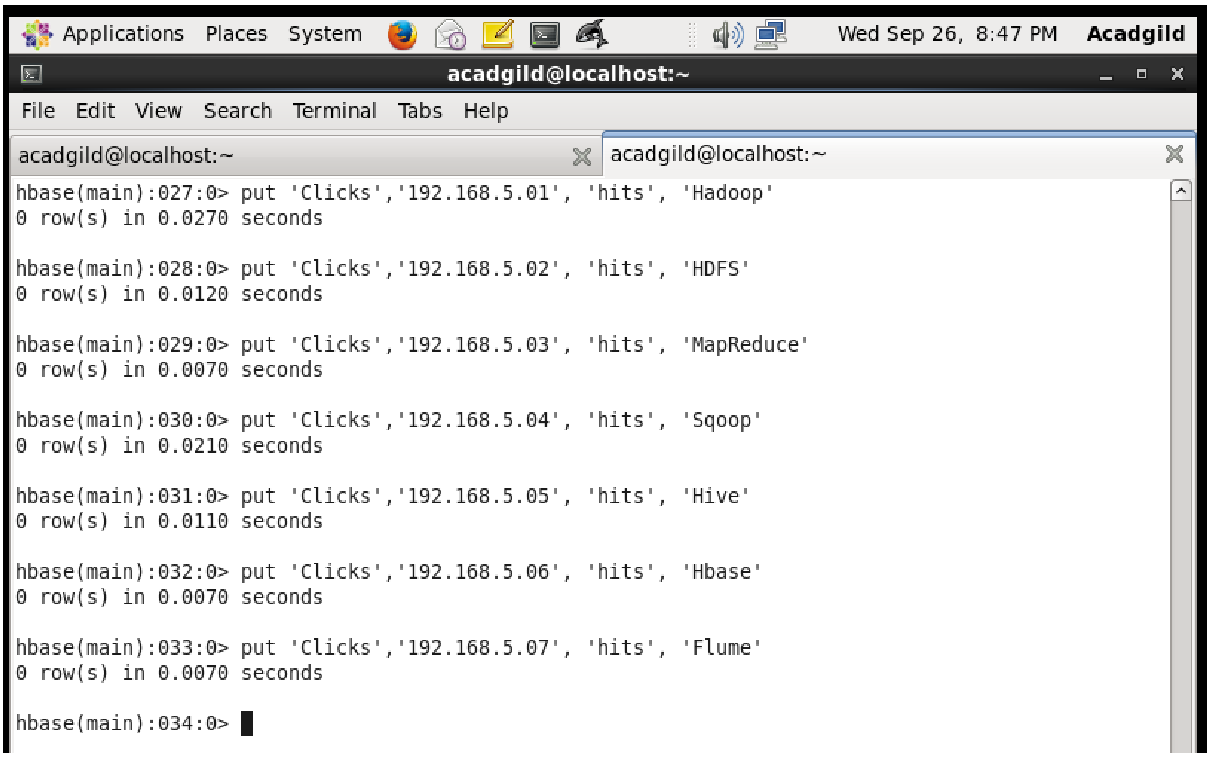
Used command **'list'** to check the tables present.



```
hbase(main):005:0> list
TABLE
Clicks
chandra
htest
3 row(s) in 0.2670 seconds

=> ["Clicks", "chandra", "htest"]
hbase(main):006:0>
```

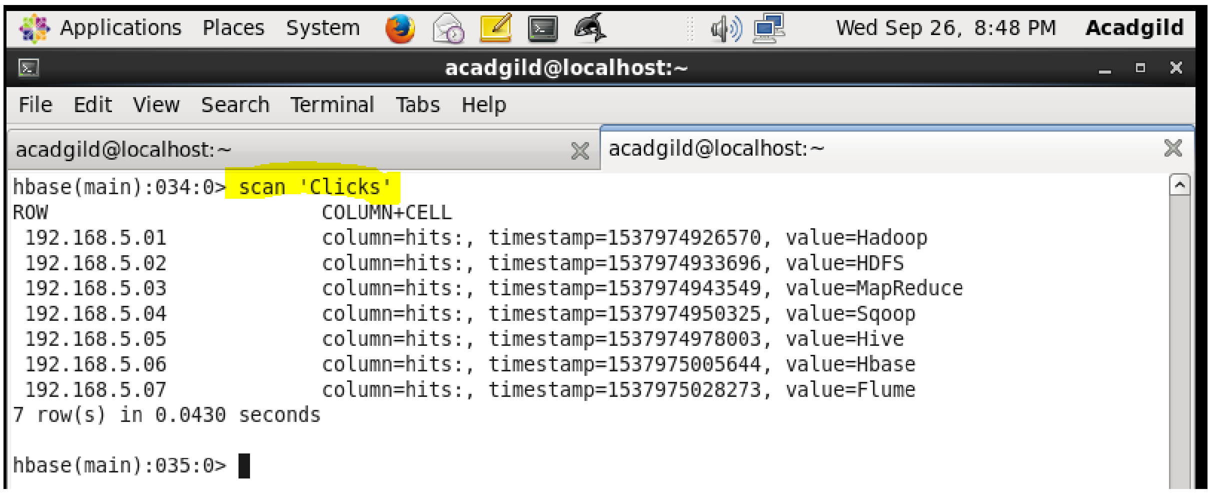
2. **Add few records in the table and update some of them. Use IP Address as row-key. Scan the table to view if all the previous versions are getting displayed.**

'Put' command is used to insert data into the table **'Clicks'**


Applications Places System
Wed Sep 26, 8:47 PM Acadgild
acadgild@localhost:~
File Edit View Search Terminal Tabs Help
acadgild@localhost:~
acadgild@localhost:~

```
hbase(main):027:0> put 'Clicks','192.168.5.01', 'hits', 'Hadoop'
0 row(s) in 0.0270 seconds

hbase(main):028:0> put 'Clicks','192.168.5.02', 'hits', 'HDFS'
0 row(s) in 0.0120 seconds

hbase(main):029:0> put 'Clicks','192.168.5.03', 'hits', 'MapReduce'
0 row(s) in 0.0070 seconds

hbase(main):030:0> put 'Clicks','192.168.5.04', 'hits', 'Sqoop'
0 row(s) in 0.0210 seconds

hbase(main):031:0> put 'Clicks','192.168.5.05', 'hits', 'Hive'
0 row(s) in 0.0110 seconds

hbase(main):032:0> put 'Clicks','192.168.5.06', 'hits', 'Hbase'
0 row(s) in 0.0070 seconds

hbase(main):033:0> put 'Clicks','192.168.5.07', 'hits', 'Flume'
0 row(s) in 0.0070 seconds

hbase(main):034:0>
```

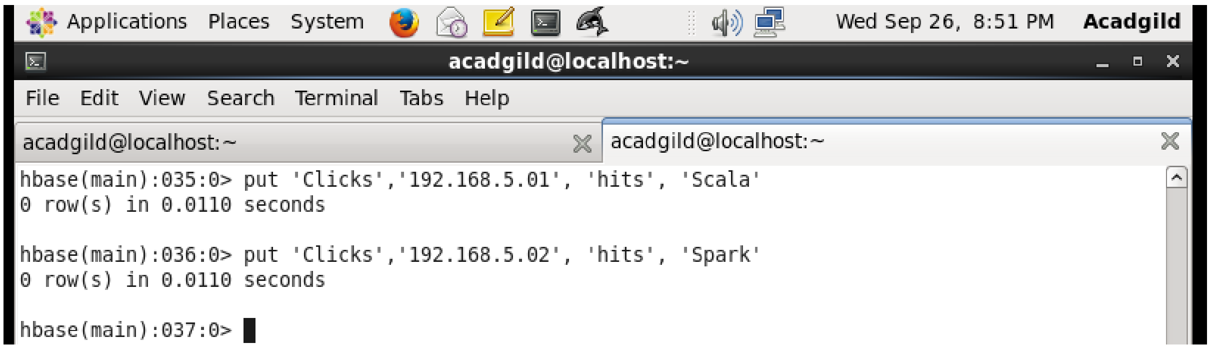
'scan' command is used to see the data present in the table.


Applications Places System
Wed Sep 26, 8:48 PM Acadgild
acadgild@localhost:~
File Edit View Search Terminal Tabs Help
acadgild@localhost:~
acadgild@localhost:~

```
hbase(main):034:0> scan 'Clicks'
ROW                         COLUMN+CELL
 192.168.5.01               column=hits:, timestamp=1537974926570, value=Hadoop
 192.168.5.02               column=hits:, timestamp=1537974933696, value=HDFS
 192.168.5.03               column=hits:, timestamp=1537974943549, value=MapReduce
 192.168.5.04               column=hits:, timestamp=1537974950325, value=Sqoop
 192.168.5.05               column=hits:, timestamp=1537974978003, value=Hive
 192.168.5.06               column=hits:, timestamp=1537975005644, value=Hbase
 192.168.5.07               column=hits:, timestamp=1537975028273, value=Flume
7 row(s) in 0.0430 seconds

hbase(main):035:0>
```

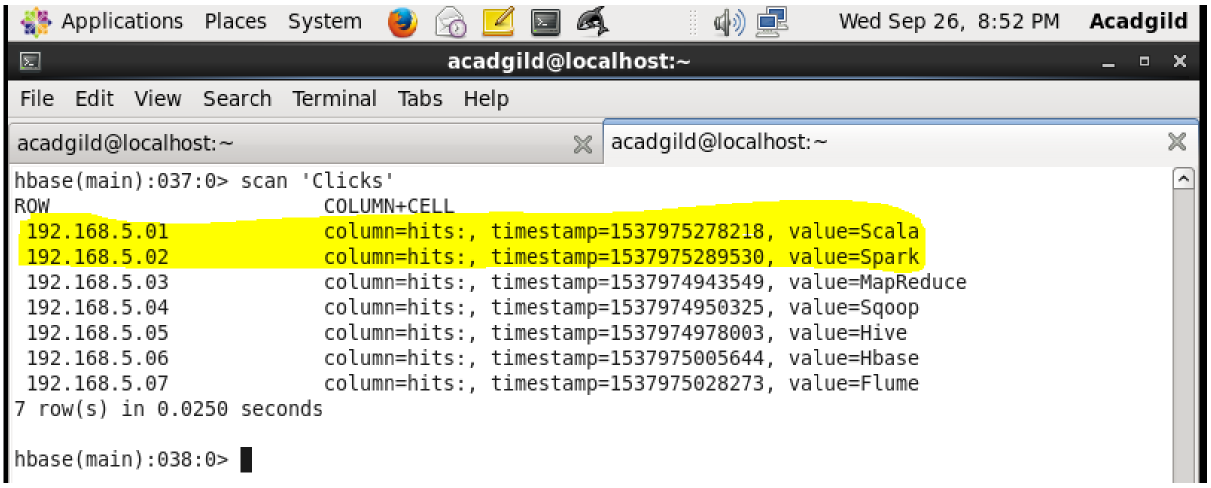
Updating the records:

Updated the value in row 1 using row key **'192.168.5.01'** with value as **'Scala'** and having column family **'hits'**


Applications Places System
Wed Sep 26, 8:51 PM Acadgild
acadgild@localhost:~
File Edit View Search Terminal Tabs Help
acadgild@localhost:~
acadgild@localhost:~

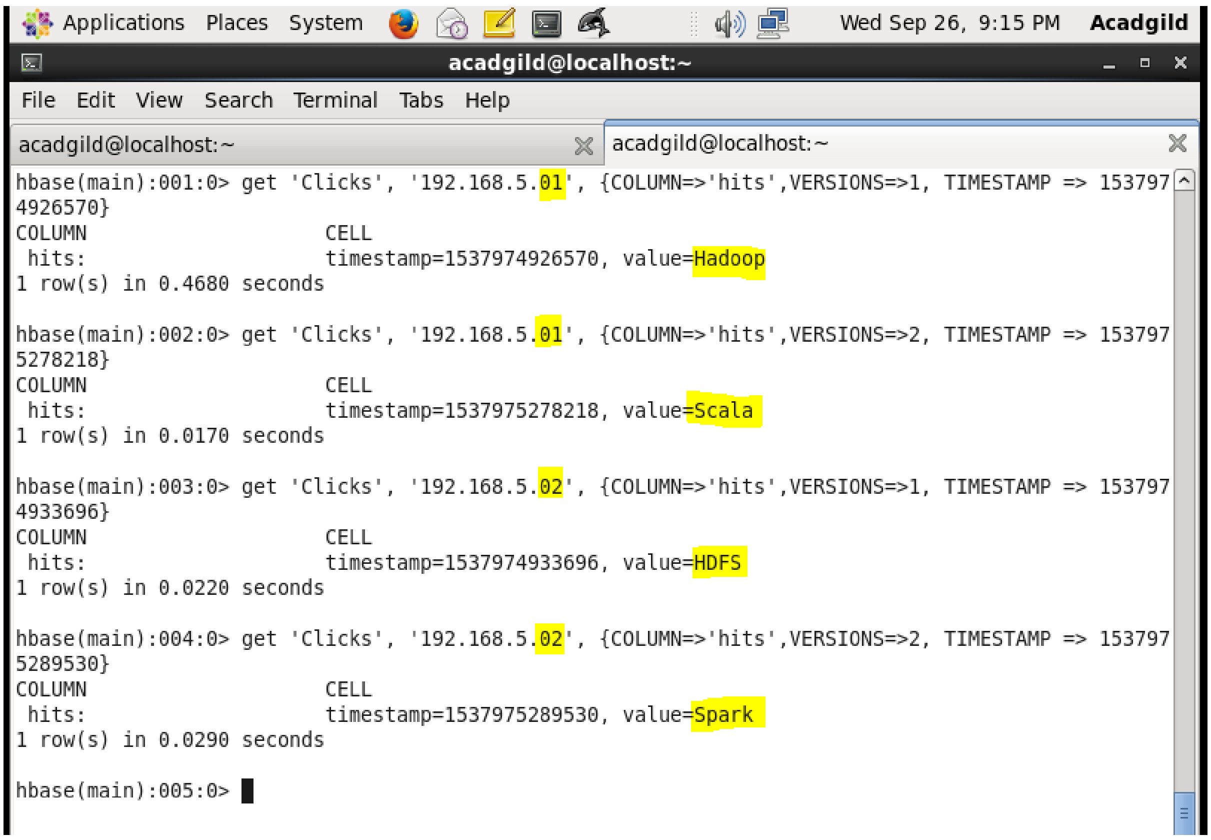
```
hbase(main):035:0> put 'Clicks','192.168.5.01', 'hits', 'Scala'
0 row(s) in 0.0110 seconds

hbase(main):036:0> put 'Clicks','192.168.5.02', 'hits', 'Spark'
0 row(s) in 0.0110 seconds

hbase(main):037:0>
```

Data present in table after updating the records.


Applications Places System
Wed Sep 26, 8:52 PM Acadgild
acadgild@localhost:~
File Edit View Search Terminal Tabs Help
acadgild@localhost:~
acadgild@localhost:~

```
hbase(main):037:0> scan 'Clicks'
ROW                          COLUMN+CELL
 192.168.5.01                column=hits:, timestamp=1537975278218, value=Scala
 192.168.5.02                column=hits:, timestamp=1537975289530, value=Spark
 192.168.5.03                column=hits:, timestamp=1537974943549, value=MapReduce
 192.168.5.04                column=hits:, timestamp=1537974950325, value=Sqoop
 192.168.5.05                column=hits:, timestamp=1537974978003, value=Hive
 192.168.5.06                column=hits:, timestamp=1537975005644, value=Hbase
 192.168.5.07                column=hits:, timestamp=1537975028273, value=Flume
7 row(s) in 0.0250 seconds

hbase(main):038:0>
```

Get the records based on previous and new timestamp and using version as well, by default hbase maintains upto 3 versions.


Applications Places System
Wed Sep 26, 9:15 PM Acadgild
acadgild@localhost:~
File Edit View Search Terminal Tabs Help
acadgild@localhost:~
acadgild@localhost:~

```
hbase(main):001:0> get 'Clicks', '192.168.5.01', {COLUMN=>'hits',VERSIONS=>1, TIMESTAMP => 153797
4926570}
COLUMN                    CELL
 hits:                    timestamp=1537974926570, value=Hadoop
1 row(s) in 0.4680 seconds

hbase(main):002:0> get 'Clicks', '192.168.5.01', {COLUMN=>'hits',VERSIONS=>2, TIMESTAMP => 153797
5278218}
COLUMN                    CELL
 hits:                    timestamp=1537975278218, value=Scala
1 row(s) in 0.0170 seconds

hbase(main):003:0> get 'Clicks', '192.168.5.02', {COLUMN=>'hits',VERSIONS=>1, TIMESTAMP => 153797
4933696}
COLUMN                    CELL
 hits:                    timestamp=1537974933696, value=HDFS
1 row(s) in 0.0220 seconds

hbase(main):004:0> get 'Clicks', '192.168.5.02', {COLUMN=>'hits',VERSIONS=>2, TIMESTAMP => 153797
5289530}
COLUMN                    CELL
 hits:                    timestamp=1537975289530, value=Spark
1 row(s) in 0.0290 seconds

hbase(main):005:0> 
```