HBase Tutorial 2

CRUD operations

CRUD operations

- Create, Read, Update, and Delete can be done from the shell and using Java client API
- In Java client API
    - supported operations are close, exist, get, put, getTableName, ….

Inserting Data - Shell

- Data can be inserted into HBase table using **Put** command
- Syntax: `put ’<table name>’,’row1’,’<colfamily:colname>’,’<value>’`

```
> put 'users','1','personal data:name','Ali'
0 row(s) in 20.2040 seconds

1.8.7-p357 :002 > scan 'users'
ROW                                    COLUMN+CELL
 1
column=personal data:name, timestamp=1543060137052,
value=Ali
1 row(s) in 0.0200 seconds
```

Insert More Data

```
> put 'users','1','personal data:city','Ramallah'
```

```
> scan 'users'
ROW                                    COLUMN+CELL
 1                                column=personal data:city, timestamp=1543066300550, value=Ramallah
 1                                column=personal data:name, timestamp=1543060137052, value=Ali
1 row(s) in 0.0150 seconds
```

Add to another column family

```
> put 'users','1','professional data:job', 'Big Data Engineer'
> put 'users','1','professional data:salary', '5000'
```

```
> scan 'users'
ROW                                    COLUMN+CELL
 1                                column=personal data:city, timestamp=1543066300550, value=Ramallah
 1                                column=personal data:name, timestamp=1543060137052, value=Ali
 1                                column=professional data:job, timestamp=1543066799451, value=Big Data
Engineer
 1                                column=professional data:salary, timestamp=1543066783685, value=5000
1 row(s) in 0.0070 seconds
```

Insert data with Explicit TimeStamp

- after the value field in the put statement, specify the timestamp (optional) if not provides it will be now

```
put ’<table name>’,’row1’,’<colfamily:colname>’,’<value>’, ts1
```

- This query will add value = Sami to the row key 1, under column personal:name using 2009 as timestamp

```
put 'emp', '1','personal:name','Sami',2009
```

Inserting Data - Java API

- steps

1. create configuration object

```
Configuration conf = HbaseConfiguration.create();
```

2. Instantiate HTable class

```
HTable hTable = new HTable(conf, tableName);
```

3. Instantiate Put class

```
Put p = new Put(Bytes.toBytes("row1"));
```

4. insert

```
p.add(Bytes.toBytes("coloumn family "), Bytes.toBytes(“column name”),
Bytes.toBytes("value"));
```

5. save

```
hTable.put(p);
```

6. close

```
hTable.close();
```

Complete Java Code

```
public class InsertData{

   public static void main(String[] args) throws IOException {

      // Instantiating Configuration class
      Configuration config = HBaseConfiguration.create();

      // Instantiating HTable class
      HTable hTable = new HTable(config, "users");

      // Instantiating Put class
      // accepts a row name.
      Put p = new Put(Bytes.toBytes("row1"));

      // adding values using add() method
      // accepts column family name, qualifier/row name ,value
      p.add(Bytes.toBytes(“personal data"),
      Bytes.toBytes("name"),Bytes.toBytes("Ali"));

      p.add(Bytes.toBytes(“personal data"),
      Bytes.toBytes("city"),Bytes.toBytes("Ramallah"));

      p.add(Bytes.toBytes("professional"),Bytes.toBytes("job"),
      Bytes.toBytes(“Big Data Engineer”));

      p.add(Bytes.toBytes("professional"),Bytes.toBytes("salary"),
      Bytes.toBytes("50000"));

      // Saving the put Instance to the HTable.
      hTable.put(p);
      System.out.println("data inserted");

      // closing HTable
      hTable.close();
   }
}
```

Updating Data - Shell

- the same way as insert using put
  - only give it the new value

```
> put 'users','1','professional data:job', 'Senior Big Data Engineer'

> put 'users','1','personal data:name', 'Ali 2'

> put 'users','1','personal data:name', 'Ali'
```

  - scan will show the latest values

```
> scan 'users',{VERSIONS =>3}
```

  - To view all versions

```
ROW                COLUMN+CELL
 1               column=personal data:city, timestamp=1543066300550, value=Ramallah
 1               column=personal data:name, timestamp=1543070083285, value=Ali
 1               column=personal data:name, timestamp=1543069941522, value=Ali 2
 1               column=personal data:name, timestamp=1543060137052, value=Ali
 1               column=professional data:job, timestamp=1543069526059, value=Senior Big Data Engineer
 1               column=professional data:salary, timestamp=1543066783685, value=5000
1 row(s) in 0.0090 seconds
```

Reading Data -Shell

- Using get command

```
get '<table name>','row1'
```

- Syntax to get Single row:

```
>   get 'users','1'
COLUMN                           CELL
 personal data:city               timestamp=1543066300550, value=Ramallah
 personal data:name                  timestamp=1543070083285, value=Ali
 professional data:job              timestamp=1543069526059, value=Senior Big Data Engineer
 professional data:salary            timestamp=1543066783685, value=5000
4 row(s) in 0.0190 seconds
```

Reading Data -Shell

- Reading all data for a given row key
- Syntax:

```
> get 'table name', ‘rowid’
```

from users table, get everything related to row 1

```
> get 'users','1'
COLUMN                             CELL
 personal data:city                timestamp=1543066300550, value=Ramallah
 personal data:name                timestamp=1543070083285, value=Ali
 professional data:job             timestamp=1543069526059, value=Senior Big Data Engineer
 professional data:salary          timestamp=1543066783685, value=5000
```

Reading Data -Shell

- Reading specific column family
    - all columns will be read

```
> get 'table name', ‘rowid’, {COLUMN ⇒ ‘column family’}
```

- Syntax:

```
> get 'users','1',{COLUMN => 'personal data'}
COLUMN                          CELL
 personal data:city              timestamp=1543066300550, value=Ramallah
 personal data:name                timestamp=1543070083285, value=Ali
```

personal data has 2 columns; city and name

Reading Data -Shell

- Reading specific column

```
> get 'table name', ‘rowid’, {COLUMN ⇒ ‘column family:column name ’}
```

- Syntax:

- Example get city column from personal data column family:

```
> get 'users','1',{COLUMN => 'personal data:city'}
COLUMN                              CELL
 personal data:city                  timestamp=1543066300550, value=Ramallah
```

Reading Data -Shell

- Reading data for given timestamp
  - Using TIMESTAMP for single point of time
- The time filter works with entire row, entire column family, or single column

timestamp with entire row

```
> get 'emp','Sami',{TIMESTAMP=>2010}
```

timestamp with entire column family

```
> get 'emp','Sami',{COLUMN=>'personal',TIMESTAMP=>2010}
```

timestamp with single column

```
> get 'emp','Sami',{COLUMN=>'personal:city',TIMESTAMP=>2010}
```

Reading Data -Shell

- Reading data for given time range
    - using TIMERANGE for period of time
- The time filter works with entire row, entire column family, or single column

timestamp with entire row

```
> get 'emp','Sami',{TIMERANGE=>[2009,2011]}
```

timestamp with entire column family

```
>get 'emp','Sami',{COLUMN=>'personal',TIMERANGE=>[2010,2011]}
```

timestamp with single column

```
> get 'emp','Sami',{COLUMN=>'personal:name',TIMERANGE=>[2010,2011]}
```

different columns

```
> get 'emp','Sami',{COLUMN=>['personal:name','professional:job'],TIMERANGE=>[2010,2011]}
```

Reading Data -Java API

1. Instantiate the configuration class   `Configuration conf = HbaseConfiguration.create();`

2. Instantiate the HTable class   `HTable table = new HTable(conf, tableName);`

3. Instantiate the Get class   `Get g = new Get(toBytes("row1"));`

4. read data from the entire column family or specific column

   g.addFamily(column family)

   or

   g.addFamily(column family, column name)

6. Get the results   `Result result = table.get(g);`

7. Read the values from result

```
byte [] value = result.getValue(Bytes.toBytes(“column family”),
                                Bytes.toBytes("column"));
```

Table Scan - Shell

- View table content using **scan** command

```
> scan 'users'
ROW                          COLUMN+CELL
 1                          column=personal data:city, timestamp=1543066300550, value=Ramallah
 1                          column=personal data:name, timestamp=1543070083285, value=Ali
 1                          column=professional data:job, timestamp=1543069526059, value=Senior Big Data Engineer
 1                          column=professional data:salary, timestamp=1543066783685, value=5000
```

- work with TIMESTAMP and TIMERANGE

- scan the entire table, or specify column family, or particular columns

- syntax: scan ‘table name’

Table Scan - Java API

```
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.hbase.HBaseConfiguration;
import org.apache.hadoop.hbase.util.Bytes;
import org.apache.hadoop.hbase.client.HTable;
import org.apache.hadoop.hbase.client.Result;
import org.apache.hadoop.hbase.client.ResultScanner;
import org.apache.hadoop.hbase.client.Scan;

public class ScanTable{

   public static void main(String args[]) throws IOException{

 1    // Instantiating Configuration class
      Configuration config = HBaseConfiguration.create();

 2    // Instantiating HTable class
      HTable table = new HTable(config, “users");

      // Instantiating the Scan class
      Scan scan = new Scan();

      // Scanning the required columns
 3    scan.addColumn(Bytes.toBytes(“personal data"), Bytes.toBytes("name"));
      scan.addColumn(Bytes.toBytes(“personal data”), Bytes.toBytes("city"));

 4    // Getting the scan result
      ResultScanner scanner = table.getScanner(scan);

      // Reading values from scan result
 5    for (Result result = scanner.next(); result != null; result = scanner.next()){

      System.out.println("Found row : " + result);}
      //closing the scanner
      scanner.close();
   }
}
```

Delete - Shell

- using the delete command
- delete specific cell: specify the row, column family , column, and the timestamp

```
> delete '<table name>', '<row>', '<column name >', '<time stamp>'
```

- delete all cell in a given row

```
> deleteall '<table name>', '<row>'
```

Delete - Java API

1. instantiate the configuration class `Configuration conf = HbaseConfiguration.create();`
2. instantiate the HTable class `HTable hTable = new HTable(conf, tableName);`
3. instantiate the delete class `Delete delete = new Delete(toBytes("row1"));`
4. specify the data to be deleted; whole column family or specific column, or specific cell

single column `delete.deleteColumn(Bytes.toBytes("personal"), Bytes.toBytes("name"));`

whole column family `delete.deleteFamily(Bytes.toBytes("professional"));`

6. perform the delete `table.delete(delete);`
7. close the connection `table.close();`

Shutdown HBase

- exit the shell by using exit command

```
> exit
```

- stop HBase by stopping the hbase service

```
./bin/stop-hbase.sh
```