HBase

HBase is an open source, multidimensional, distributed, scalable and a NoSQL database written in Java. HBase runs on top of HDFS (Hadoop Distributed File System) and provides BigTable like capabilities to Hadoop. It is designed to provide a fault tolerant way of storing large collection of sparse data sets.
It is a schema-less database which contains keys and values. Each key, points to a value which is an array of bytes, can be a string, BLOB, XML, etc.

Features of HBase

- HBase is linearly scalable.
- It has automatic failure support.
- It provides consistent read and writes.
- It integrates with Hadoop, both as a source and a destination.
- It has easy java API for client.
- It provides data replication across clusters.

Properties of HBase:

1. Columnar Storage
2. Denormalization Storage
3. Only CRUD operations
4. ACID properties follow at only row level

Where to Use HBase

- Apache HBase is used to have random, real-time read/write access to Big Data.
- It hosts very large tables on top of clusters of commodity hardware.
- Apache HBase is a non-relational database modeled after Google's Bigtable. Bigtable acts up on Google File System, likewise Apache HBase works on top of Hadoop and HDFS.

HBase Architecture and its Important Components

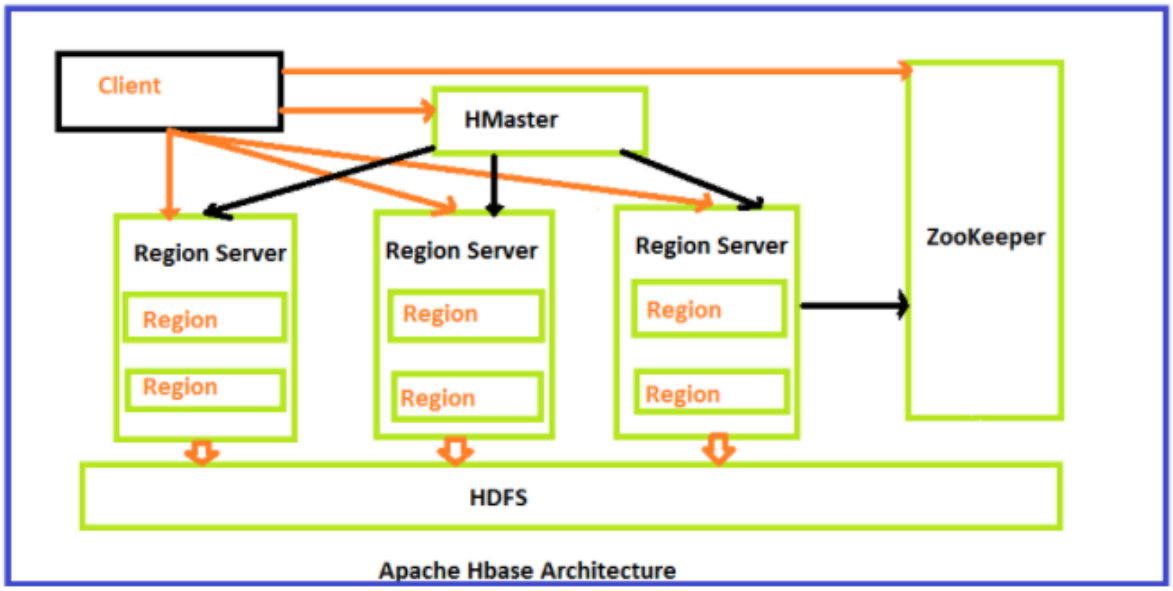

Fig: HBase Architecture

HBase architecture consists mainly of four components

- HMaster
- HRegionserver
- HRegions
- Zookeeper
- HDFS

HMaster:

Role of HMaster in HBase:

- Plays a vital role in terms of performance and maintaining nodes in the cluster.
- HMaster provides admin performance and distributes services to different region servers.
- HMaster assigns regions to region servers.
- HMaster has the features like controlling load balancing and failover to handle the load over nodes present in the cluster.
- When a client wants to change any schema and to change any Metadata operations, HMaster takes responsibility for these operations.

HBase Regions:

Role of HBase Regions in HBase:

- It contains distribution of tables and are comprised of Column families.
- It contains multiple stores, one for each column family.
- It consists of mainly two components, which are Memstore and Hfile.

HBase Regions Servers:

Role of HBase Regions Servers in HBase:

- Hosting and managing regions
- Splitting regions automatically
- Handling read and writes requests
- Communicating with the client directly

ZooKeeper:

Role of ZooKeeper in HBase:

- Maintains Configuration information
- Provides distributed synchronization
- Client Communication establishment with region servers
- Provides ephemeral nodes for which represent different region servers
- Master servers usability of ephemeral nodes for discovering available servers in the cluster
- To track server failure and network partitions

HDFS:

Role of HDFS in HBase:

- It provides a distributed environment for the storage.
- It is a file system designed in a way to run on commodity hardware.
- It stores each file in multiple blocks and to maintain fault tolerance, the blocks are replicated across a Hadoop cluster.

CRUD operations using Hbase

Creating a Table

We can create a table using the create command.
Syntax: create ‘<table name>’,’<column family>’

Example:

```
2.7.1 :003 > create 'census', 'personal', 'professional'
Created table census
Took 1.3996 seconds
 => Hbase::Table - census
2.7.1 :004 > list
TABLE
census
1 row(s)
Took 0.0281 seconds
 => ["census"]
```

Inserting Data

We can insert data in a table using the put command.
Syntax: put ’<table name>’,’row1’,’<colfamily:colname>’,’<value>’

Example:

```
2.7.1 :018 > put 'census', 2, 'professional:education', 'Post Graduate'
Took 0.0122 seconds
2.7.1 :019 > scan 'census'
ROW                    COLUMN+CELL
 1                     column=personal:gender, timestamp=2021-05-15T01:36:
                       29.295, value=Male
 1                     column=personal:marital_status, timestamp=2021-05-1
                       5T01:36:07.713, value=Unmarried
 1                     column=personal:name, timestamp=2021-05-15T01:35:21
                       .895, value=Shivam
 1                     column=professional:education, timestamp=2021-05-15
                       T01:40:58.776, value=High School
 1                     column=professional:employed, timestamp=2021-05-15T
                       01:40:15.205, value=Yes
 2                     column=personal:gender, timestamp=2021-05-15T01:43:
                       22.452, value=Female
 2                     column=personal:marital_status, timestamp=2021-05-1
                       5T01:42:59.151, value=Married
 2                     column=personal:name, timestamp=2021-05-15T01:42:36
                       .123, value=Tanvi
 2                     column=professional:education, timestamp=2021-05-15
                       T01:45:01.538, value=Post Graduate
 2                     column=professional:employed, timestamp=2021-05-15T
                       01:43:36.967, value=Yes
```

Reading Data

We can read data from a table using the get command.
Syntax: get ’<table name>’,’row1’

Example:

```
2.7.1 :022 > get 'census', 2
COLUMN                                CELL
 personal:gender                      timestamp=2021-05-15T01:43:22.452, value=Female
 personal:marital_status              timestamp=2021-05-15T01:42:59.151, value=Married
 personal:name                        timestamp=2021-05-15T01:42:36.123, value=Tanvi
 professional:education               timestamp=2021-05-15T01:45:01.538, value=Post Graduate
 professional:employed                timestamp=2021-05-15T01:43:36.967, value=Yes
1 row(s)
```

Deleting Data From Table

We can read data from a table using the command.
Syntax: delete ‘<table name>’, ‘<row>’, ‘<column name >’

Example:

```
2.7.1 :036 > delete 'census', 1, 'personal:marital_status'
Took 0.0228 seconds
2.7.1 :037 > scan 'census'
ROW                                   COLUMN+CELL
 1                                    column=personal:gender, timestamp=2021-05-15T01:36:29.295, value=Male
 1                                    column=personal:name, timestamp=2021-05-15T01:35:21.895, value=Shivam
 1                                    column=professional:education, timestamp=2021-05-15T01:40:58.776, value=High School
 1                                    column=professional:employed, timestamp=2021-05-15T01:40:15.205, value=Yes
 2                                    column=personal:gender, timestamp=2021-05-15T01:43:22.452, value=Female
 2                                    column=personal:marital_status, timestamp=2021-05-15T01:42:59.151, value=Married
 2                                    column=personal:name, timestamp=2021-05-15T01:42:36.123, value=Tanvi
 2                                    column=professional:education, timestamp=2021-05-15T01:45:01.538, value=Post Graduate
 2                                    column=professional:employed, timestamp=2021-05-15T01:43:36.967, value=Yes
```

Deleting Table

We can delete a table using the drop command before deleting the table we have to disable it first.
Syntax: drop '<Table_Name>'

```
2.7.1 :038 > disable 'census'
Took 1.3278 seconds
2.7.1 :039 > drop 'census'
Took 1.3901 seconds
2.7.1 :040 > list
TABLE
0 row(s)
Took 0.0231 seconds
 => []
```

CRUD operations using Hbase Java API

Creating a Table

Create Table Code:

hbase/pom.xml | *GettingStarted.java

```java
package hbase;

import java.io.*;

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

        Configuration conf = HBaseConfiguration.create();
        Connection connection = ConnectionFactory.createConnection(conf);
        try {
            Admin admin = connection.getAdmin();
            HTableDescriptor tableName = new HTableDescriptor(TableName.valueOf("census"));
            tableName.addFamily(new HColumnDescriptor("personal"));
            tableName.addFamily(new HColumnDescriptor("professional"));

            if (!admin.tableExists(tableName.getTableName())) {
                System.out.println("Creating the Census Table");
                admin.createTable(tableName);
                System.out.println("Done");
            }
            else {
                System.out.println("Table Already Exists");
            }

        }
        finally {
            connection.close();
        }
    }
}
```

Output:

```
2.7.1 :002 > list
TABLE
census
1 row(s)
Took 0.0451 seconds
 => ["census"]
```

Inserting Data

Insert Data Code:

hbase/pom.xml | *GettingStarted.java

```java
package hbase;

import java.io.*;

public class GettingStarted {
    private static byte[] PERSONAL_CF = Bytes.toBytes("personal");
    private static byte[] PROFESSIONAL_CF = Bytes.toBytes("professional");
    private static byte[] NAME_COLUMN = Bytes.toBytes("name");
    private static byte[] GENDER_COLUMN = Bytes.toBytes("gender");
    private static byte[] MARITAL_STATUS_COLUMN = Bytes.toBytes("marital_status");
    private static byte[] EMPLOYED_COLUMN = Bytes.toBytes("employed");
    private static byte[] FIELD_COLUMN = Bytes.toBytes("field");

    public static void main(String[] args) throws IOException {
        Configuration conf = HBaseConfiguration.create();
        Connection connection = ConnectionFactory.createConnection(conf);
        Table table = null;
        try {
            table = connection.getTable(TableName.valueOf("census"));
            Put put1 = new Put(Bytes.toBytes("1"));
            put1.addColumn(PERSONAL_CF, NAME_COLUMN, Bytes.toBytes("Mike Jon"));
            put1.addColumn(PERSONAL_CF, GENDER_COLUMN, Bytes.toBytes("Male"));
            put1.addColumn(PERSONAL_CF, MARITAL_STATUS_COLUMN, Bytes.toBytes("Married"));
            put1.addColumn(PROFESSIONAL_CF, EMPLOYED_COLUMN, Bytes.toBytes("Yes"));
            put1.addColumn(PROFESSIONAL_CF, FIELD_COLUMN, Bytes.toBytes("Construction"));
            table.put(put1);
            Put put2 = new Put(Bytes.toBytes("2"));
            put2.addColumn(PERSONAL_CF, NAME_COLUMN, Bytes.toBytes("Dolcy"));
            put2.addColumn(PERSONAL_CF, GENDER_COLUMN, Bytes.toBytes("Female"));
            put2.addColumn(PROFESSIONAL_CF, EMPLOYED_COLUMN, Bytes.toBytes("Yes"));
            Put put3 = new Put(Bytes.toBytes("3"));
            put3.addColumn(PERSONAL_CF, NAME_COLUMN, Bytes.toBytes("David"));
            put3.addColumn(PERSONAL_CF, GENDER_COLUMN, Bytes.toBytes("Male"));
            put3.addColumn(PERSONAL_CF, MARITAL_STATUS_COLUMN, Bytes.toBytes("Unmarried"));
            List<Put> putList = new ArrayList<Put>();
            putList.add(put1);
            putList.add(put2);
            putList.add(put3);
            table.put(putList);
            System.out.println("Inserted row for Mike Jon, Dolcy, David");

        }
        finally {
            connection.close();
            if (table != null) {
                table.close();
            }
```

Output:

```
2.7.1 :007 > scan 'census'
ROW                                    COLUMN+CELL
 1                                     column=personal:gender, timestamp=2021-05-15T19:04:42.214, value=Male
 1                                     column=personal:marital_status, timestamp=2021-05-15T19:04:42.214, value=Married
 1                                     column=personal:name, timestamp=2021-05-15T19:04:42.214, value=Mike Jon
 1                                     column=professional:employed, timestamp=2021-05-15T19:04:42.214, value=Yes
 1                                     column=professional:field, timestamp=2021-05-15T19:04:42.214, value=Construction
 2                                     column=personal:gender, timestamp=2021-05-15T19:04:42.214, value=Female
 2                                     column=personal:name, timestamp=2021-05-15T19:04:42.214, value=Dolcy
 2                                     column=professional:employed, timestamp=2021-05-15T19:04:42.214, value=Yes
 3                                     column=personal:gender, timestamp=2021-05-15T19:04:42.214, value=Male
 3                                     column=personal:marital_status, timestamp=2021-05-15T19:04:42.214, value=Unmarried
 3                                     column=personal:name, timestamp=2021-05-15T19:04:42.214, value=David
3 row(s)
```

Reading Data

Read Data from Table Code:

hbase/pom.xml | GettingStarted.java

```java
package hbase;

import java.io.*;

public class GettingStarted {
    private static byte[] PERSONAL_CF = Bytes.toBytes("personal");
    private static byte[] PROFESSIONAL_CF = Bytes.toBytes("professional");
    private static byte[] NAME_COLUMN = Bytes.toBytes("name");
    private static byte[] FIELD_COLUMN = Bytes.toBytes("field");

    public static void main(String[] args) throws IOException {
        Configuration conf = HBaseConfiguration.create();
        Connection connection = ConnectionFactory.createConnection(conf);
        Table table = null;
        try {
            table = connection.getTable(TableName.valueOf("census"));
            Get get = new Get(Bytes.toBytes("1"));
            get.addColumn(PERSONAL_CF, NAME_COLUMN);
            get.addColumn(PROFESSIONAL_CF, FIELD_COLUMN);
            Result result = table.get(get);
            byte[] nameValue = result.getValue(PERSONAL_CF, NAME_COLUMN);
            System.out.println("Name: "+ Bytes.toString(nameValue));
            byte[] fieldValue = result.getValue(PROFESSIONAL_CF, FIELD_COLUMN);
            System.out.println("Field: "+ Bytes.toString(fieldValue));
            List<Get> getList = new ArrayList<Get>();
            Get get1 = new Get(Bytes.toBytes("2"));
            get1.addColumn(PERSONAL_CF, NAME_COLUMN);
            Get get2 = new Get(Bytes.toBytes("3"));
            get1.addColumn(PERSONAL_CF, NAME_COLUMN);
            getList.add(get1);
            getList.add(get2);
            Result[] results = table.get(getList);
            for (Result res : results) {
                nameValue = res.getValue(PERSONAL_CF, NAME_COLUMN);
                System.out.println("Name: "+Bytes.toString(nameValue));
            }
        }
        finally {
            connection.close();
            if (table != null) {
                table.close();
            }
        }
    }
```

Output:

Problems | Javadoc | Declaration | Console

```
<terminated> GettingStarted [Java Application] /usr/lib/jvm/java-11-openjdk-amd64/bin/java (15-May-2
WARNING: An illegal reflective access operation has occurred
WARNING: Illegal reflective access by org.apache.hadoop.hbase.util.UnsafeAvailChecker (file:/home/piyushpp/.m2/repo
WARNING: Please consider reporting this to the maintainers of org.apache.hadoop.hbase.util.UnsafeAvailChecker
WARNING: Use --illegal-access=warn to enable warnings of further illegal reflective access operations
WARNING: All illegal access operations will be denied in a future release
log4j:WARN No appenders could be found for logger (org.apache.hadoop.util.Shell).
log4j:WARN Please initialize the log4j system properly.
log4j:WARN See http://logging.apache.org/log4j/1.2/faq.html#noconfig for more info.
Name: Mike Jon
Field: Construction
Name: Dolcy
Name: David
```

Deleting Data from Table

Code to Delete Data from Table:

hbase/pom.xml | *GettingStarted.java

```java
package hbase;

import java.io.*;

public class GettingStarted {

    private static byte[] PERSONAL_CF = Bytes.toBytes("personal");
    private static byte[] PROFESSIONAL_CF = Bytes.toBytes("professional");
    private static byte[] GENDER_COLUMN = Bytes.toBytes("gender");
    private static byte[] FIELD_COLUMN = Bytes.toBytes("field");

    public static void main(String[] args) throws IOException {
        Configuration conf = HBaseConfiguration.create();
        Connection connection = ConnectionFactory.createConnection(conf);
        Table table = null;
        try {
            table = connection.getTable(TableName.valueOf("census"));

            Delete delete = new Delete(Bytes.toBytes("1"));

            delete.addColumn(PERSONAL_CF, GENDER_COLUMN);
            delete.addColumn(PROFESSIONAL_CF, FIELD_COLUMN);

            table.delete(delete);
            System.out.println("Deletion Complete!");
        }
        finally {
            connection.close();
            if (table != null) {
                table.close();
            }
        }
    }
}
```

Output:

```
2.7.1 :008 > scan 'census'
ROW                                    COLUMN+CELL
 1                                     column=personal:gender, timestamp=2021-05-15T19:04:42.214, value=Male
 1                                     column=personal:marital_status, timestamp=2021-05-15T19:04:42.214, value=Married
 1                                     column=personal:name, timestamp=2021-05-15T19:04:42.214, value=Mike Jon
 1                                     column=professional:employed, timestamp=2021-05-15T19:04:42.214, value=Yes
 1                                     column=professional:field, timestamp=2021-05-15T19:04:42.214, value=Construction
 2                                     column=personal:gender, timestamp=2021-05-15T19:04:42.214, value=Female
 2                                     column=personal:name, timestamp=2021-05-15T19:04:42.214, value=Dolcy
 Terminal                              column=professional:employed, timestamp=2021-05-15T19:04:42.214, value=Yes
 3                                     column=personal:gender, timestamp=2021-05-15T19:04:42.214, value=Male
 3                                     column=personal:marital_status, timestamp=2021-05-15T19:04:42.214, value=Unmarried
 3                                     column=personal:name, timestamp=2021-05-15T19:04:42.214, value=David
3 row(s)
Took 0.0316 seconds
2.7.1 :009 > scan 'census'
ROW                                    COLUMN+CELL
 1                                     column=personal:marital_status, timestamp=2021-05-15T19:04:42.214, value=Married
 1                                     column=personal:name, timestamp=2021-05-15T19:04:42.214, value=Mike Jon
 1                                     column=professional:employed, timestamp=2021-05-15T19:04:42.214, value=Yes
 2                                     column=personal:gender, timestamp=2021-05-15T19:04:42.214, value=Female
 2                                     column=personal:name, timestamp=2021-05-15T19:04:42.214, value=Dolcy
 2                                     column=professional:employed, timestamp=2021-05-15T19:04:42.214, value=Yes
 3                                     column=personal:gender, timestamp=2021-05-15T19:04:42.214, value=Male
 3                                     column=personal:marital_status, timestamp=2021-05-15T19:04:42.214, value=Unmarried
 3                                     column=personal:name, timestamp=2021-05-15T19:04:42.214, value=David
3 row(s)
```

Deleting Table

Code to delete table:

hbase/pom.xml | GettingStarted.java

```java
package hbase;

import java.io.*;

public class GettingStarted {

    public static void main(String[] args) throws IOException {
        Configuration conf = HBaseConfiguration.create();
        Connection connection = ConnectionFactory.createConnection(conf);
        try {
            TableName tableName = TableName.valueOf("census");
            Admin admin = connection.getAdmin();

            if(admin.tableExists(tableName)) {
                System.out.println("Deleting Table ...");
                admin.disableTable(tableName);
                admin.deleteTable(tableName);
                System.out.println("Done!");
            }
            else {
                System.out.println("Table does not exists.");
            }
        }
        finally {
            connection.close();
        }
    }
}
```

Output:

```
2.7.1 :010 > list
TABLE
census
1 row(s)
Took 0.0298 seconds
 => ["census"]
2.7.1 :011 > list
TABLE
0 row(s)
Took 0.0099 seconds
 => []
```

Filter rows based on condition

hbase/pom.xml *GettingStarted.java

```java
package hbase;

import java.io.*;

public class GettingStarted {

    private static void printResults(ResultScanner scanResult) {
        System.out.println();
        System.out.println("Results: ");

        for(Result res : scanResult) {
            for(Cell cell : res.listCells()) {
                String row = new String(CellUtil.cloneRow(cell));
                String family = new String(CellUtil.cloneFamily(cell));
                String column = new String(CellUtil.cloneQualifier(cell));
                String value = new String(CellUtil.cloneValue(cell));

                System.out.println(row + " " + family + " " + column + " " + value);
            }
        }
    }

    public static void main(String[] args) throws IOException {
        Configuration conf = HBaseConfiguration.create();
        Connection connection = ConnectionFactory.createConnection(conf);
        Table table = null;
        ResultScanner scanResult = null;
        try {
            table = connection.getTable(TableName.valueOf("census"));

            SingleColumnValueFilter filter = new SingleColumnValueFilter(
            Bytes.toBytes("personal"), Bytes.toBytes("gender"), CompareFilter.CompareOp.EQUAL, new BinaryComparator(Bytes.toBytes("
            filter.setFilterIfMissing(true);
            Scan userScan = new Scan();
            userScan.setFilter(filter);

            scanResult = table.getScanner(userScan);
            printResults(scanResult);
            filter = new SingleColumnValueFilter(Bytes.toBytes("personal"),
            Bytes.toBytes("name"), CompareFilter.CompareOp.EQUAL, new SubstringComparator("Jones"));
            userScan.setFilter(filter);
            scanResult = table.getScanner(userScan);
            printResults(scanResult);
        }

        finally {
            connection.close();
            if (table != null) {
                table.close();
            }
            if (scanResult != null) {
                scanResult.close();
            }
        }
    }
}
```

Output:

Problems Javadoc Declaration Console

```
<terminated> GettingStarted [Java Application] /usr/lib/jvm/java-11-openjdk-amd64/bin/java (16-May-2021, 1:57
WARNING: Please consider reporting this to the maintainers of org.apache.hadoop.security.authentication.util.KerberosUtil
WARNING: Use --illegal-access=warn to enable warnings of further illegal reflective access operations
WARNING: All illegal access operations will be denied in a future release

Results:
1 personal gender male
1 personal marital_status Unmarried
1 personal name Mike Jones
1 professional education_level high school
1 professional employed yes
1 professional field construction
2 personal gender male
2 personal marital_status divorced
2 personal name Ben

Results:
1 personal gender male
1 personal marital_status Unmarried
1 personal name Mike Jones
1 professional education_level high school
1 professional employed yes
1 professional field construction
```

MapReduce with Hbase

MapReduce is a process which is designed to solve the problem of processing in excess of terabytes of data in a scalable way. MapReduce follows a divide-and-conquer approach by splitting the data located on a distributed filesystem so that the servers available can access these chunks of data and process them as fast as they can.

Classes that are involved in the Map reduce:

- **InputFormat:** First it splits the input data, and then it returns a RecordReader instance that defines the classes of the key and value objects, and provides a next() method that is used to iterate over each input record.
- **Mapper:** In this step, each record read using the RecordReader is processed using the map() method.
- **Reducer:** The Reducer stage and class hierarchy is very similar to the Mapper stage. This time we get the output of a Mapper class and process it after the data has been shuffled and sorted.
- **OutputFormat**: Its job is to persist the data in various locations. There are specific implementations that allow output to files, or to HBase tables in the case of the TableOutputFormat class.
- **TableMapReduceUtil:** The MapReduce support comes with the TableMapReduceUtil class that helps in setting up MapReduce jobs over HBase.

Mapper Code

hbase/pom.xml | GettingStarted.java | MaritalStatusMapper.java | MaritalStatusReducer.java

```java
package mapreduce;

import java.io.IOException;

import org.apache.hadoop.hbase.client.Result;
import org.apache.hadoop.hbase.io.ImmutableBytesWritable;

import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.hbase.mapreduce.TableMapper;

public class MaritalStatusMapper extends TableMapper<ImmutableBytesWritable, IntWritable>{

    public static final byte[] CF = "personal".getBytes();
    public static final byte[] MARITAL_STATUS = "marital_status".getBytes();

    private final IntWritable ONE = new IntWritable(1);
    private ImmutableBytesWritable key = new ImmutableBytesWritable();

    public void map(ImmutableBytesWritable row, Result value, Context context) throws IOException, InterruptedException{
        key.set(value.getValue(CF,MARITAL_STATUS));
        context.write(key, ONE);
    }

}
```

Reducer Code

hbase/pom.xml | GettingStarted.java | MaritalStatusMapper.java | MaritalStatusReducer.java | Main.java

```java
package mapreduce;

import java.io.IOException;...

public class MaritalStatusReducer extends TableReducer<ImmutableBytesWritable, IntWritable, ImmutableBytesWritable>{

    public static final byte[] CF = "marital_status".getBytes();
    public static final byte[] COUNT = "count".getBytes();

    public void reduce(ImmutableBytesWritable key, Iterable<IntWritable> value, Context context) throws IOException, InterruptedException{
        Integer sum = 0;
        for(IntWritable val : value) {
            sum += val.get();
        }
        Put put = new Put(key.get());
        put.addColumn(CF, COUNT, Bytes.toBytes(sum.toString()));
        context.write(key, put);
    }

}
```

Driver Code

hbase/pom.xml | GettingStarted.java | MaritalStatusMapper.java | Marit

```java
package mapreduce;

import org.apache.hadoop.hbase.HBaseConfiguration;...

public class Main {
    public static void main(String[] args) throws Exception{
        Configuration conf = HBaseConfiguration.create();
        Job job = Job.getInstance(conf);
        String sourceTable = "census";
        String targetTable = "summary";

        Scan scan = new Scan();
        scan.setCaching(500);
        scan.setCacheBlocks(false);

        TableMapReduceUtil.addDependencyJars(job);
        TableMapReduceUtil.initTableMapperJob(sourceTable, scan,
        MaritalStatusMapper.class, ImmutableBytesWritable.class, IntWritable.class, job);
        TableMapReduceUtil.initTableReducerJob(targetTable, MaritalStatusReducer.class, job);
        job.setNumReduceTasks(1);
        job.waitForCompletion(true);
    }

}
```

Output:

```
2.7.1 :011 > scan 'summary'
ROW                                    COLUMN+CELL
 divorced                              column=marital_status:count, timestamp=2021-05-16T14:11:47.365, value=2
 married                               column=marital_status:count, timestamp=2021-05-16T14:12:20.344, value=1
 unmarried                             column=marital_status:count, timestamp=2021-05-16T14:12:42.705, value=1
3 row(s)
```