Example 8:
Creating a table using the HBase Java API

HBase provides a Java API

All HBase shell operations, have a corresponding Java object / method

Let's create a table for notifications

We want to do the equivalent of

```
create 'notifications','attributes','metrics'
```

in Java

```
create 'notifications','attributes','metrics'
```

To create a table you need to

1. Specify the table name and column families — `HTableDescriptor`
2. Connect to HBase — `Connection`
3. Create the table — `HBaseAdmin`

```
create 'notifications','attributes','metrics'
```

```
public class createTable   {

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

        Configuration conf = HBaseConfiguration.create();
        Connection connection = ConnectionFactory.createConnection(conf);

        Admin admin = connection.getAdmin();

        HTableDescriptor tableName = new HTableDescriptor("notifications");

        tableName.addFamily(new HColumnDescriptor("attributes"));
        tableName.addFamily(new HColumnDescriptor("metrics"));


        if (!admin.tableExists(tableName.getTableName())) {
            System.out.print("Creating table. ");
            admin.createTable(tableName);
            System.out.println(" Done.");
        }
    }
}
```

```
public class createTable   {

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

        Configuration conf = HBaseConfiguration.create();
        Connection connection = ConnectionFactory.createConnection(conf);

        Admin admin = connection.getAdmin();

HTableDescriptor tableName = new HTableDescriptor("notifications");

tableName.addFamily(new HColumnDescriptor("attributes"));
tableName.addFamily(new HColumnDescriptor("metrics"));

        if (!admin.tableExists(tableName.getTableName())) {
            System.out.print("Creating table. ");
            admin.createTable(tableName);
            System.out.println(" Done.");
        }
    }
}
```

```
create 'notifications','attributes','metrics'
```

HBase provides a HTableDescriptor class to specify table properties

```
create 'notifications','attributes','metrics'
```

```
public class createTable   {

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

        Configuration conf = HBaseConfiguration.create();
        Connection connection = ConnectionFactory.createConnection(conf);

        Admin admin = connection.getAdmin();

HTableDescriptor tableName = new HTableDescriptor("notifications");

tableName.addFamily(new HColumnDescriptor("attributes"));
tableName.addFamily(new HColumnDescriptor("metrics"));

        if (!admin.tableExists(tableName.getTableName())) {
            System.out.print("Creating table. ");
            admin.createTable(tableName);
            System.out.println(" Done.");
        }
    }
}
```

HTableDescriptor is used to specify

1. The table name
2. Column families
3. Some other properties related to performance tuning

```
public class createTable   {

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

        Configuration conf = HBaseConfiguration.create();
        Connection connection = ConnectionFactory.createConnection(conf);

        Admin admin = connection.getAdmin();

HTableDescriptor tableName = new HTableDescriptor("notifications");

tableName.addFamily(new HColumnDescriptor("attributes"));
tableName.addFamily(new HColumnDescriptor("metrics"));

        if (!admin.tableExist
            System.out.print
            admin.createTable
            System.out.println(" Done.");
        }
    }
}
```

```
create 'notifications','attributes','metrics'
```

This specifies the table name as notifications

```
create 'notifications','attributes','metrics'
```

```
public class createTable   {

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

        Configuration conf = HBaseConfiguration.create();
        Connection connection = ConnectionFactory.createConnection(conf);

        Admin admin = connection.getAdmin();

HTableDescriptor tableName = new HTableDescriptor("notifications");

tableName.addFamily(new HColumnDescriptor("attributes"));
tableName.addFamily(new HColumnDescriptor("metrics"));

        if (!admin.tableExists(tableName.getTableName())) {
            System.out.print("Creating table. ");
            admin.createTable(tableName);
            System.out.println("Done.");
        }
    }
}
```

Specify the column families using HColumnDescriptor

```
create 'notifications','attributes','metrics'
```

```
public class createTable   {

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

        Configuration conf = HBaseConfiguration.create();
        Connection connection = ConnectionFactory.createConnection(conf);

        Admin admin = connection.getAdmin();

HTableDescriptor tableName = new HTableDescriptor("notifications");

tableName.addFamily(new HColumnDescriptor("attributes"));
tableName.addFamily(new HColumnDescriptor("metrics"));

        if (!admin.tableExists(tableName.getTableName())) {
            System.out.print("Creating table. ");
            admin.createTable(tableName);
            System.out.println(" Done.");
        }
    }
}
```

addFamily will add the column family to the table properties

```
create 'notifications','attributes','metrics'
```

```
public class createTable   {

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

```
Configuration conf = HBaseConfiguration.create();
Connection connection = ConnectionFactory.createConnection(conf);
```

```
        Admin admin = connection.getAdmin();

        HTableDescriptor tableName = new HTableDescriptor("notifications");

        tableName.addFamily(new HColumnDescriptor("attributes"));
        tableName.addFamily(new HColumnDescriptor("metrics"));


        if (!admin.tableExists(tableName.getTableName())) {
            System.out.print("Creating table. ");
            admin.createTable(tableName);
            System.out.println(" Done.");
        }
    }
}
```

Once you have set up the table properties, you need some boilerplate to connect to HBase and create the table

```
create 'notifications','attributes','metrics'
```

```
public class createTable   {

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

Configuration conf = HBaseConfiguration.create();
Connection connection = ConnectionFactory.createConnection(conf);
        Admin admin = connection.getAdmin();

        HTableDescriptor tableName = new HTableDescriptor("notifications");

        tableName.addFamily(new HColumnDescriptor("attributes"));
        tableName.addFamily(new HColumnDescriptor("metrics"));


        if (!admin.tableExists(tableName.getTableName())) {
            System.out.print("Creating table. ");
            admin.createTable(tableName);
            System.out.println(" Done.");
        }
    }
}
```

This represents our connection to HBase

```
create 'notifications','attributes','metrics'
```

```
Configuration conf = HBaseConfiguration.create();
Connection connection = ConnectionFactory.createConnection(conf);
```

The connection is created using the default settings in the HBase configuration (set up during install)

```
create 'notifications','attributes','metrics'
```

```
public class createTable   {

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

        Configuration conf = HBaseConfiguration.create();
        Connection connection = ConnectionFactory.createConnection(conf);

        Admin admin = connection.getAdmin();

        HTableDescriptor tableName = new HTableDescriptor("notifications");

        tableName.addFamily(new HColumnDescriptor("attributes"));
        tableName.addFamily(new HColumnDescriptor("metrics"));

        if (!admin.tableExists(tableName.getTableName())) {
            System.out.print("Creating table.");
            admin.createTable(tableName);
            System.out.println(" Done.");
        }
    }
}
```

The Connection object can provide an instance of HBaseAdmin

```
create 'notifications','attributes','metrics'
```

We use the HBaseAdmin to

```
Admin admin = connection.getAdmin();
```

1. create tables
2. delete tables
3. check if a table exists etc

```
create 'notifications','attributes','metrics'
```

First check if the notifications table already exists

```
public class createTable   {

    public static void main(String[] args) throws IOException {
        Configuration config = HBaseConfiguration.create();
        Connection connection = ConnectionFactory.createConnection(config);

        Admin admin = connection.getAdmin();

        HTableDescriptor tableName = new HTableDescriptor("notifications");

        tableName.addFamily(new HColumnDescriptor("attributes"));
        tableName.addFamily(new HColumnDescriptor("metrics"));

        if (!admin.tableExists(tableName.getTableName())) {
            System.out.print("Creating table.");
            admin.createTable(tableName);
            System.out.println(" Done.");
        }
    }
}
```

```
create 'notifications','attributes','metrics'
```

If it doesn't, create the table

```
public class createTable   {

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

        Configuration conf = HBaseConfiguration.create();
        Connection connection = ConnectionFactory.createConnection(conf);

        Admin admin = connection.getAdmin();

        HTableDescriptor tableName = new HTableDescriptor("notifications");

        tableName.addFamily(new HColumnDescriptor("attributes"));
        tableName.addFamily(new HColumnDescriptor("metrics"));


        if (!admin.tableExists(tableName.getTableName())) {
                System.out.print("Creating table.");
                admin.createTable(tableName);
                System.out.println(" Done.");
            }
    }
}
```

```
create 'notifications','attributes','metrics'
```

createTable requires an instance of HTableDescriptor with the table properties specified

```
public class createTable   {

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

        Configuration conf = HBaseConfiguration.create();
        Connection connection = ConnectionFactory.createConnection(conf);
        Admin admin = connection.getAdmin();

        HTableDescriptor tableName = new HTableDescriptor(TableName.valueOf("notifications"));

        tableName.addFamily(new HColumnDescriptor("attributes"));
        tableName.addFamily(new HColumnDescriptor("metrics"));


        if (!admin.tableExists(tableName.getTableName())) {
                System.out.print("Creating table.");
                admin.createTable(tableName);
                System.out.println(" Done.");
            }
    }
}
```

Example 9:

Inserting/Updating columns for a single row id

Put

Put

Let's insert/update a row id in the notifications table

We want to do the equivalent of

```
put 'notifications',2, 'attributes:for_user','Chaz'
```

in Java

Put

```
put 'notifications',2, 'attributes:for_user','Chaz'
```

1. Specify the row id, column and value

Put

2. Connect to HBase

Connection

3. Get a Table object to represent our table

HTable

4. Insert/update the row

Put

These steps are common to all table operations ie. put, get, delete, scan etc

2. Connect to HBase — `Connection`

3. Get a Table object to represent our table — `HTable`

Put

```
put 'notifications',2, 'attributes:for_user','Chaz'
```

```
public class singlePut{

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

        Configuration conf = HBaseConfiguration.create();

        Connection connection = ConnectionFactory.createConnection(conf);

        Table table = connection.getTable("notifications");
        //HTable table = new HTable(conf, "notifications");

        Put put =new Put(Bytes.toBytes("2"));

        put.addColumn(Bytes.toBytes("attributes"), Bytes.toBytes("type"), Bytes.toBytes("Comment"));
        put.addColumn(Bytes.toBytes("attributes"),Bytes.toBytes("for_user"),Bytes.toBytes("Chaz"));
        put.addColumn(Bytes.toBytes("metrics"),Bytes.toBytes("open"),Bytes.toBytes("0"));

        table.put(put);

    }
}
```

Put

```
put 'notifications',2, 'attributes:for_user','Chaz'
```

```
public class singlePut{

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

        Configuration conf = HBaseConfiguration.create();

        Connection connection = ConnectionFactory.createConnection(conf);

        Table table = connection.getTable("notifications");
        //HTable table = new HTable(conf, "notifications");

        Put put =new Put(Bytes.toBytes("2"));

        put.addColumn(Bytes.toBytes("attributes"), Bytes.toBytes("type"), Bytes.toBytes("Comment"));
        put.addColumn(Bytes.toBytes("attributes"),Bytes.toBytes("for_user"),Bytes.toBytes("Chaz"));
        put.addColumn(Bytes.toBytes("metrics"),Bytes.toBytes("open"),Bytes.toBytes("0"));

        table.put(put);

    }
}
```

Use the Put class to specify the data that will be inserted/updated

Put

```
put 'notifications',2, 'attributes:for_user','Chaz'
```

```
public class singlePut{

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

        Configuration conf = HBaseConfiguration.create();

        Connection connection = ConnectionFactory.createConnection(conf);

        Table table = connection.getTable("notifications");
        //HTable table = new HTable(conf, "notifications");

         Put put =new Put(Bytes.toBytes("2"));

        put.addColumn(Bytes.toBytes("attributes"), Bytes.toBytes("type"), Bytes.toBytes("Comment"));
        put.addColumn(Bytes.toBytes("attributes"),Bytes.toBytes("for_user"),Bytes.toBytes("Chaz"));
        put.addColumn(Bytes.toBytes("metrics"),Bytes.toBytes("open"),Bytes.toBytes("0"));

        table.put(put);

    }
}
```

The Put is used to insert/ update a specific row id

Put

```
put 'notifications',2, 'attributes:for_user','Chaz'
```

All data that is passed to HBase must be in the form of byte arrays

```
public class singlePut{

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

        Configuration conf = HBaseConfiguration.create();

        Connection connection = ConnectionFactory.createConnection(conf);

        Table table = connection.getTable("notifications");
        //HTable table = new HTable(conf, "notifications");

         Put put =new Put(Bytes.toBytes("2"));

        put.addColumn(Bytes.toBytes("attributes"), Bytes.toBytes("type"), Bytes.toBytes("Comment"));
        put.addColumn(Bytes.toBytes("attributes"),Bytes.toBytes("for_user"),Bytes.toBytes("Chaz"));
        put.addColumn(Bytes.toBytes("metrics"),Bytes.toBytes("open"),Bytes.toBytes("0"));

        table.put(put);

    }
}
```

byte[]

Put

```
put 'notifications',2, 'attributes:for_user','Chaz'
```

byte[]

row ids, column family names, column names, values

```
Put put =new Put(Bytes.toBytes("2"));

put.addColumn(Bytes.toBytes("attributes"), Bytes.toBytes("type"), Bytes.toBytes("Comment"));
put.addColumn(Bytes.toBytes("attributes"),Bytes.toBytes("for_user"),Bytes.toBytes("Chaz"));
put.addColumn(Bytes.toBytes("metrics"),Bytes.toBytes("open"),Bytes.toBytes("0"));

table.put(put);

    }
}
```

Put

```
put 'notifications',2, 'attributes:for_user','Chaz'
```

The Bytes class is a helper class provided by HBase

```
Table table = connection.getTable("notifications");
//HTable table = new HTable(conf, "notifications");

Put put =new Put(Bytes.toBytes("2"));

put.addColumn(Bytes.toBytes("attributes"), Bytes.toBytes("type"), Bytes.toBytes("Comment"));
put.addColumn(Bytes.toBytes("attributes"),Bytes.toBytes("for_user"),Bytes.toBytes("Chaz"));
put.addColumn(Bytes.toBytes("metrics"),Bytes.toBytes("open"),Bytes.toBytes("0"));

table.put(put);
```

to help convert any Java primitives to byte arrays

Put

```
put 'notifications',2, 'attributes:for_user','Chaz'
```

```
public class singlePut{

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

        Configuration conf = HBaseConfiguration.create();

        Connection connection = ConnectionFactory.createConnection(conf);

        Table table = connection.getTable("notifications");
        //HTable table = new HTable(conf, "notifications");

        Put put =new Put(Bytes.toBytes("1"));

          put.addColumn(Bytes.toBytes("attributes"),
                        Bytes.toBytes("for_user"),
                        Bytes.toBytes("Chaz"))

        put.addColumn(Bytes.toBytes("attributes"), Bytes.toBytes("type"), Bytes.toBytes("Comment"));
        put.addColumn(Bytes.toBytes("metrics"),Bytes.toBytes("open"), Bytes.toBytes("0"));

        table.put(put);

    }
}
```

Use addColumn to specify the column family, column and value

Put

```
put 'notifications',2, 'attributes for_user','Chaz'
```

```
public class singlePut{

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

        Configuration conf = HBaseConfiguration.create();

        Connection connection = ConnectionFactory.createConnection(conf);

        Table table = connection.getTable("notifications");
        //HTable table = new HTable(conf, "notifications");

        Put put =new Put(Bytes.toBytes("1"));

            put.addColumn(Bytes.toBytes("attributes"),
                          Bytes.toBytes("for_user"),
                          Bytes.toBytes("Chaz"))

        put.addColumn(Bytes.toBytes("attributes"), Bytes.toBytes("type"), Bytes.toBytes("Comment"));
        put.addColumn(Bytes.toBytes("metrics"),Bytes.toBytes("open"),Bytes.toBytes("0"));

        table.put(put);

            }
        }
```

Column Family

Put

```
put 'notifications',2, 'attributes:for_user','Chaz'
```

```
public class singlePut{

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

        Configuration conf = HBaseConfiguration.create();

        Connection connection = ConnectionFactory.createConnection(conf);

        Table table = connection.getTable("notifications");
        //HTable table = new HTable(conf, "notifications");

        Put put =new Put(Bytes.toBytes("1"));

            put.addColumn(Bytes.toBytes("attributes"),
                          Bytes.toBytes("for_user"),
                          Bytes.toBytes("Chaz"))

        put.addColumn(Bytes.toBytes("attributes"), Bytes.toBytes("type"), Bytes.toBytes("Comment"));
        put.addColumn(Bytes.toBytes("metrics"),Bytes.toBytes("open"),Bytes.toBytes("0"));

        table.put(put);

            }
        }
```

Column

Put

```
put 'notifications',2, 'attributes:for_user','Chaz'
```

```
public class singlePut{

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

        Configuration conf = HBaseConfiguration.create();

        Connection connection = ConnectionFactory.createConnection(conf);

        Table table = connection.getTable("notifications");
        //HTable table = new HTable(conf, "notifications");

        Put put =new Put(Bytes.toBytes("1"));

          put.addColumn(Bytes.toBytes("attributes"),
                        Bytes.toBytes("for_user"),
                        Bytes.toBytes("Chaz"))

        put.addColumn(Bytes.toBytes("attributes"), Bytes.toBytes("type"), Bytes.toBytes("Comment"));
        put.addColumn(Bytes.toBytes("metrics"),Bytes.toBytes("open"),Bytes.toBytes("0"));

        table.put(put);

            }
        }
```

Value

Put

```
put 'notifications',2, 'attributes:for_user','Chaz'
```

id	type	for user	from user	timestamp
1	Friend request status	Ryan	Jessica	146710201
2	Comment	Chaz	Daniel	146711200
3	Comment	Rick	Brendan	1467112205
4	Like	Rick	Brendan	1467112213

row	column	value
2	for user	Chaz

In the shell, you can only insert 1 cell with 1 put

Put

```
put 'notifications',2, 'attributes:for_user','Chaz'
```

```
put.addColumn(Bytes.toBytes("attributes"),Bytes.toBytes("for_user"),Bytes.toBytes("Chaz"));
put.addColumn(Bytes.toBytes("attributes"), Bytes.toBytes("type"), Bytes.toBytes("Comment"));
put.addColumn(Bytes.toBytes("metrics"),Bytes.toBytes("open"),Bytes.toBytes("0"));
```

id	type	for user	from user	timestamp
1	Friend request status	Ryan	Jessica	146710201
2	Comment	Chaz	Daniel	146711200
3	Comment	Rick	Brendan	1467112205
4	Like	Rick	Brendan	1467112213

In Java, you can insert multiple cells for 1 row id at one go

Put

```
put 'notifications',2, 'attributes:for_user','Chaz'
```

```
put.addColumn(Bytes.toBytes("attributes"),Bytes.toBytes("for_user"),Bytes.toBytes("Chaz"));
put.addColumn(Bytes.toBytes("attributes"),Bytes.toBytes("type"),Bytes.toBytes("Comment"));
put.addColumn(Bytes.toBytes("metrics"),Bytes.toBytes("open"),Bytes.toBytes("0"));
```

row	column	value
2	for user	Chaz

Put

```
put 'notifications',2, 'attributes:for_user','Chaz'
```

```
put.addColumn(Bytes.toBytes("attributes"),Bytes.toBytes("for_user"),Bytes.toBytes("Chaz"));
put.addColumn(Bytes.toBytes("attributes"), Bytes.toBytes("type"), Bytes.toBytes("Comment"));
put.addColumn(Bytes.toBytes("metrics"),Bytes.toBytes("open"),Bytes.toBytes("0"));
```

row	column	value
2	attributes : for user	Chaz
2	attributes : type	Comment

Put

```
put 'notifications',2, 'attributes:for_user','Chaz'
```

```
Put put =new Put(Bytes.toBytes("1"));

put.addColumn(Bytes.toBytes("attributes"),Bytes.toBytes("for_user"),Bytes.toBytes("Chaz"));
put.addColumn(Bytes.toBytes("attributes"), Bytes.toBytes("type"), Bytes.toBytes("Comment"));
put.addColumn(Bytes.toBytes("metrics"),Bytes.toBytes("open"),Bytes.toBytes("0"));

table.put(put);
```

row	column	value
2	attributes : for user	Chaz
2	attributes : type	Comment
2	metrics:open	0

All these are inserted/ updated for the same row id

Put

```
put 'notifications',2, 'attributes:for_user','Chaz'
```

```
public class singlePut{

    public static void main(String[] args) throws IOException{
Configuration conf = HBaseConfiguration.create();
Connection connection = ConnectionFactory.createConnection(conf);

        Table table = connection.getTable("notifications");
        //HTable table = new HTable(conf, "notifications");

        Put put =new Put(Bytes.toBytes("1"));

        put.addColumn(Bytes.toBytes("attributes"), Bytes.toBytes("type"), Bytes.toBytes("comment"));
        put.addColumn(Bytes.toBytes("attributes"), Bytes.toBytes("for_user"), Bytes.toBytes("Chaz"));
        put.addColumn(Bytes.toBytes("devices"), Bytes.toBytes("Chaz"), Bytes.toBytes("Nexus"));

        table.put(put);

    }
}
```

Once you have set up the Put, use the usual boilerplate to connect to HBase

Put

```
put 'notifications',2, 'attributes:for_user','Chaz'
```

```
public class singlePut{

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

        Configuration conf = HBaseConfiguration.create();

        Connection connection = ConnectionFactory.createConnection(conf);

        Table table = connection.getTable("notifications");
        //HTable table = new HTable(conf, "notifications");

        Put put =new Put(Bytes.toBytes("1"));

        put.addColumn(Bytes.toBytes("attributes"), Bytes.toBytes("type"), Bytes.toBytes("comment"));
        put.addColumn(Bytes.toBytes("attributes"),Bytes.toBytes("for_user"),Bytes.toBytes("Chaz"));
        put.addColumn(Bytes.toBytes("metrics"),Bytes.toBytes("open"),Bytes.toBytes("0"));

        table.put(put);

    }
}
```

The Connection object gives us an instance of HTable

Put

```
put 'notifications',2, 'attributes:for_user','Chaz'
```

```
public class singlePut{

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

        Configuration conf = HBaseConfiguration.create();

        Connection connection = ConnectionFactory.createConnection(conf);

        Table table = connection.getTable("notifications");
        //HTable table = new HTable(conf, "notifications");

        Put put =new Put(Bytes.toBytes("1"));

        put.addColumn(Bytes.toBytes("attributes"),Bytes.toBytes("type"),Bytes.toBytes("update"));
        put.addColumn(Bytes.toBytes("attributes"),Bytes.toBytes("for_user"),Bytes.toBytes("Chaz"));
        put.addColumn(Bytes.toBytes("metrics"),Bytes.toBytes("open"),Bytes.toBytes("0"));

        table.put(put);

    }
}
```

The HTable object represents our table in HBase

Put

```
put 'notifications',2, 'attributes:for_user','Chaz'
```

```
public class singlePut{

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

        Configuration conf = HBaseConfiguration.create();

        Connection connection = ConnectionFactory.createConnection(conf);

        Table table = connection.getTable("notifications");
        //HTable table = new HTable(conf, "notifications");

        Put put =new Put(Bytes.toBytes("1"));

        put.addColumn(Bytes.toBytes("attributes"),Bytes.toBytes("Company"),Bytes.toBytes("Cloudera"));
        put.addColumn(Bytes.toBytes("attributes"),Bytes.toBytes("for_user"),Bytes.toBytes("Chaz"));
        put.addColumn(Bytes.toBytes("metrics"),Bytes.toBytes("open"),Bytes.toBytes("0"));

        table.put(put);

    }
}
```

Manual set up of HTable objects was available in HBase

Put

```
put 'notifications',2, 'attributes:for_user','Chaz'
```

```
public class singlePut{

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

        Configuration conf = HBaseConfiguration.create();

        Connection connection = ConnectionFactory.createConnection(conf);

        Table table = connection.getTable("notifications");
        //HTable table = new HTable(conf, "notifications");

        Put put =new Put(Bytes.toBytes("1"));

        put.addColumn(Bytes.toBytes("attributes"), Bytes.toBytes("type"), Bytes.toBytes("Comment"));
        put.addColumn(Bytes.toBytes("attributes"), Bytes.toBytes("for_user"), Bytes.toBytes("Chaz"));
        put.addColumn(Bytes.toBytes("metrics"),B

        table.put(put);

    }
}
```

This is deprecated now

Put

```
put 'notifications',2, 'attributes:for_user','Chaz'
```

```
public class singlePut{

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

        Configuration conf = HBaseConfiguration.create();

        Connection connection = ConnectionFactory.createConnection(conf);

        Table table = connection.getTable("notifications");
        //HTable table = new HTable(conf, "notifications");

        Put put =new Put(Bytes.toBytes("1"));

        put.addColumn(Bytes.toBytes("attributes"),Bytes.toBytes("for_user"),Bytes.toBytes("Chaz"));
        put.addColumn(Bytes.toBytes("attributes"),Bytes.toBytes("for_user"),Bytes.toBytes("Chaz"));
        put.addColumn(Bytes.toBytes("metrics"),Bytes.toBytes("open"),Bytes.toBytes("0"));

        table.put(put);

    }
}
```

Use the connection object to get the HTable instance

Put

```
put 'notifications',2, 'attributes:for_user','Chaz'
```

```
public class singlePut{

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

        Configuration conf = HBaseConfiguration.create();

        Connection connection = ConnectionFactory.createConnection(conf);

        Table table = connection.getTable("notifications");
        //HTable table = new HTable(conf, "notifications");

        Put put =new Put(Bytes.toBytes("1"));

        put.addColumn(Bytes.toBytes("attributes"),Bytes.toBytes("type"),Bytes.toBytes("comment"));
        put.addColumn(Bytes.toBytes("attributes"),Bytes.toBytes("for_user"),Bytes.toBytes("Chaz"));
        put.addColumn(Bytes.toBytes("metrics"),Bytes.toBytes("open"),Bytes.toBytes("0"));

        table.put(put);

    }
}
```

Use the put method to insert/update the row id

Example 10:
Inserting multiple rows

List <Put>

List <Put>

Put objects are specific to 1 row id

```
Put put =new Put(Bytes.toBytes("2"));
```

To insert values for multiple row ids, use a list of Puts

List <Put>

Let's insert data for 2 row ids

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

        Configuration conf = HBaseConfiguration.create();
        Connection connection = ConnectionFactory.createConnection(conf);
        Table table = connection.getTable("notifications");


        Put put1 =new Put(Bytes.toBytes("2"));

        put1.addColumn(Bytes.toBytes("attributes"), Bytes.toBytes("type"), Bytes.toBytes("Friend Request"));
        put1.addColumn(Bytes.toBytes("attributes"),Bytes.toBytes("for_user"),Bytes.toBytes("Daniel"));
        put1.addColumn(Bytes.toBytes("attributes"), Bytes.toBytes("from_user"), Bytes.toBytes("Ryan"));


        Put put2 =new Put(Bytes.toBytes("3"));
        put2.addColumn(Bytes.toBytes("attributes"), Bytes.toBytes("type"), Bytes.toBytes("Like"));
        put2.addColumn(Bytes.toBytes("attributes"),Bytes.toBytes("for_user"),Bytes.toBytes("Brendan"));
        put2.addColumn(Bytes.toBytes("attributes"),Bytes.toBytes("from_user"),Bytes.toBytes("Rick"));
        put2.addColumn(Bytes.toBytes("attributes"),Bytes.toBytes("for_thing"),Bytes.toBytes("link"));
        put2.addColumn(Bytes.toBytes("attributes"), Bytes.toBytes("link"), Bytes.toBytes("link"));


        List<Put> puts = new ArrayList<Put>();
        puts.add(put1);
        puts.add(put2);

        table.put(puts);

    }
}
```

row	column	value
2	attributes : type	Friend request status
2	attributes : for user	Daniel
2	attributes : from user	Ryan
3	attributes : type	Like
3	attributes : for user	Brendan
3	attributes : from user	Rick
3	attributes : for_thing	link
3	attributes : link	“link”

List <Put>

```
public class listPuts {

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

        Configuration conf = HBaseConfiguration.create();
        Connection connection = ConnectionFactory.createConnection(conf);
        Table table = connection.getTable("notifications");
```

```
Put put1 =new Put(Bytes.toBytes("2"));

put1.addColumn(Bytes.toBytes("attributes"), Bytes.toBytes("type"), Bytes.toBytes("Friend Request"));
put1.addColumn(Bytes.toBytes("attributes"),Bytes.toBytes("for_user"),Bytes.toBytes("Daniel"));
put1.addColumn(Bytes.toBytes("attributes"), Bytes.toBytes("from_user"), Bytes.toBytes("Ryan"));
```

```
        Put put2 =new Put(Bytes.toBytes("3"));
        put2.addColumn(Bytes.toBytes("attributes"), Bytes.toBytes("type"), Bytes.toBytes("Like"));
        put2.addColumn(Bytes.toBytes("attributes"),Bytes.toBytes("for_user"),Bytes.toBytes("Brendan"));
        put2.addColumn(Bytes.toBytes("attributes"),Bytes.toBytes("from_user"),Bytes.toBytes("Rick"));
        put2.addColumn(Bytes.toBytes("attributes"),Bytes.toBytes("for_thing"),Bytes.toBytes("link"));
        put2.addColumn(Bytes.toBytes("attributes"), Bytes.toBytes("link"), Bytes.toBytes("link"));

        List<Put> puts = new ArrayList<Put>();
        puts.add(put1);
        puts.add(put2);
        table.put(puts);
    }
}
```

Create a Put for row id 2

List <Put>

```
public class listPuts {

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

        Configuration conf = HBaseConfiguration.create();
        Connection connection = ConnectionFactory.createConnection(conf);
        Table table = connection.getTable("notifications");
```

```
Put put1 =new Put(Bytes.toBytes("2"));

put1.addColumn(Bytes.toBytes("attributes"), Bytes.toBytes("type"), Bytes.toBytes("Friend Request"));
put1.addColumn(Bytes.toBytes("attributes"),Bytes.toBytes("for_user"),Bytes.toBytes("Daniel"));
put1.addColumn(Bytes.toBytes("attributes"), Bytes.toBytes("from_user"), Bytes.toBytes("Ryan"));
```

```
        Put put2 =new Put(Bytes.toBytes("3"));
        put2.addColumn(Bytes.toBytes("attributes"), Bytes.toBytes("type"), Bytes.toBytes("Like"));
        put2.addColumn(Bytes.toBytes("attributes"),Bytes.toBytes("for_user"),Bytes.toBytes("Brendan"));
        put2.addColumn(Bytes.toBytes("attributes"),Bytes.toBytes("from_user"),Bytes.toBytes("Rick"));
        put2.addColumn(Bytes.toBytes("attributes"),Bytes.toBytes("for_thing"),Bytes.toBytes("link"));
        put2.addColumn(Bytes.toBytes("attributes"), Bytes.toBytes("link"), Bytes.toBytes("link"));

        List<Put> puts = new A<Put>();
        puts.add(put1);
        puts.add(put2);

        table.put(puts);

    }
}
```

Specify the columns and values

row id	column	value
2	attributes : type	Friend request status
2	attributes : for user	Daniel
2	attributes : from user	Ryan

```
public class listPuts {

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

        Configuration conf = HBaseConfiguration.create();
        Connection connection = ConnectionFactory.createConnection(conf);
        Table table = connection.getTable("notifications");


        Put put1 =new Put(Bytes.toBytes("2"));

        put1.addColumn(Bytes.toBytes("attributes"), Bytes.toBytes("type"), Bytes.toBytes("Friend Request"));
        put1.addColumn(Bytes.toBytes("attributes"),Bytes.toBytes("for_user"),Bytes.toBytes("Daniel"));
        put1.addColumn(Bytes.toBytes("attributes"), Bytes.toBytes("from_user"), Bytes.toBytes("Ryan"));

Put put2 =new Put(Bytes.toBytes("3"));
put2.addColumn(Bytes.toBytes("attributes"), Bytes.toBytes("type"), Bytes.toBytes("Like"));
put2.addColumn(Bytes.toBytes("attributes"),Bytes.toBytes("for_user"),Bytes.toBytes("Brendan"));
put2.addColumn(Bytes.toBytes("attributes"),Bytes.toBytes("from_user"),Bytes.toBytes("Rick"));
put2.addColumn(Bytes.toBytes("attributes"),Bytes.toBytes("for_thing"),Bytes.toBytes("link"));
put2.addColumn(Bytes.toBytes("attributes"), Bytes.toBytes("link"), Bytes.toBytes("link"));

        List<Put> puts = new ArrayList<Put>();
        puts.add(put1);
        puts.add(put2);

        table.put(puts);
    }
}
```

Create another Put for row id 3

```
public class listPuts {

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

        Configuration conf = HBaseConfiguration.create();
        Connection connection = ConnectionFactory.createConnection(conf);
        Table table = connection.getTable("notifications");


        Put put1 =new Put(Bytes.toBytes("2"));

        put1.addColumn(Bytes.toBytes("attributes"), Bytes.toBytes("type"), Bytes.toBytes("Friend Request"));
        put1.addColumn(Bytes.toBytes("attributes"),Bytes.toBytes("for_user"),Bytes.toBytes("Daniel"));
        put1.addColumn(Bytes.toBytes("attributes"), Bytes.toBytes("from_user"), Bytes.toBytes("Ryan"));

Put put2 =new Put(Bytes.toBytes("3"));
put2.addColumn(Bytes.toBytes("attributes"), Bytes.toBytes("type"), Bytes.toBytes("Like"));
put2.addColumn(Bytes.toBytes("attributes"),Bytes.toBytes("for_user"),Bytes.toBytes("Brendan"));
put2.addColumn(Bytes.toBytes("attributes"),Bytes.toBytes("from_user"),Bytes.toBytes("Rick"));
put2.addColumn(Bytes.toBytes("attributes"),Bytes.toBytes("for_thing"),Bytes.toBytes("link"));
put2.addColumn(Bytes.toBytes("attributes"), Bytes.toBytes("link"), Bytes.toBytes("link"));

        List<Put> puts = new ArrayList<Put>();
        puts.add(put1);
        puts.add(put2);

        table.put(puts);

    }
}
```

row	column	value
3	attributes : type	Like
3	attributes : for user	Brendan
3	attributes : from user	Rick
3	attributes : for_thing	link
3	attributes : link	“link”

Specify the columns and values

List <Put>

Create a List and add the Put objects to it

```
public class listPuts {

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

        Configuration conf = HBaseConfiguration.create();
        Connection connection = ConnectionFactory.createConnection(conf);
        Table table = connection.getTable("notifications");


        Put put1 =new Put(Bytes.toBytes("2"));

        put1.addColumn(Bytes.toBytes("attributes"), Bytes.toBytes("type"), Bytes.toBytes("Friend Request"));
        put1.addColumn(Bytes.toBytes("attributes"),Bytes.toBytes("for_user"),Bytes.toBytes("Daniel"));
        put1.addColumn(Bytes.toBytes("attributes"), Bytes.toBytes("from_user"), Bytes.toBytes("Ryan"));


        Put put2 =new Put(Bytes.toBytes("3"));
        put2.addColumn(Bytes.toBytes("attributes"), Bytes.toBytes("type"), Bytes.toBytes("Like"));
        put2.addColumn(Bytes.toBytes("attributes"),Bytes.toBytes("for_user"),Bytes.toBytes("Brendan"));
        put2.addColumn(Bytes.toBytes("attributes"),Bytes.toBytes("from_user"),Bytes.toBytes("Rick"));
        put2.addColumn(Bytes.toBytes("attributes"),Bytes.toBytes("for_thing"),Bytes.toBytes("link"));
        put2.addColumn(Bytes.toBytes("attributes"), Bytes.toBytes("link"), Bytes.toBytes("link"));

        List<Put> puts = new ArrayList<Put>();
        puts.add(put1);
        puts.add(put2);

        table.put(puts);

    }
}
```

List <Put>

```
public class listPuts {

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

        Configuration conf = HBaseConfiguration.create();
        Connection connection = ConnectionFactory.createConnection(conf);
        Table table = connection.getTable("notifications");

        Put put1 =new Put(Bytes.toBytes("2"));

        put1.addColumn(Bytes.toBytes("attributes"), Bytes.toBytes("type"), Bytes.toBytes("Friend Request"));
        put1.addColumn(Bytes.toBytes("attributes"),Bytes.toBytes("for_user"),Bytes.toBytes("Daniel"));
        put1.addColumn(Bytes.toBytes("attributes"), Bytes.toBytes("from_user"), Bytes.toBytes("Ryan"));


        Put put2 =new Put(Bytes.toBytes("3"));
        put2.addColumn(Bytes.toBytes("attributes"), Bytes.toBytes("type"), Bytes.toBytes("Like"));
        put2.addColumn(Bytes.toBytes("attributes"),Bytes.toBytes("for_user"),Bytes.toBytes("Brendan"
        put2.addColumn(Bytes.toBytes("attributes"),Bytes.toBytes("from_user"),Bytes.toBytes("Rick"))
        put2.addColumn(Bytes.toBytes("attributes"),Bytes.toBytes("for_thing"),Bytes.toBytes("link"))
        put2.addColumn(Bytes.toBytes("attributes"), Bytes.toBytes("link"), Bytes.toBytes("link"));


        List<Put> puts = new ArrayList<Put>();
        puts.add(put1);
        puts.add(put2);

            table.put(puts);


    }
}
```

Do the usual boilerplate to get a connection and a HTable

List <Put>

```
public class listPuts {

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

        Configuration conf = HBaseConfiguration.creat (
        Connection connection = ConnectionFactory.cre (
        Table table = connection.getTable("notifications");

            Put put1 =new Put(Bytes.toBytes("2"));

            put1.addColumn(Bytes.toBytes("attributes"), Bytes.toBytes("type"), Bytes.toBytes("Friend Request"));
            put1.addColumn(Bytes.toBytes("attributes"),Bytes.toBytes("for_user"),Bytes.toBytes("Daniel"));
            put1.addColumn(Bytes.toBytes("attributes"), Bytes.toBytes("from_user"), Bytes.toBytes("Ryan"));


            Put put2 =new Put(Bytes.toBytes("3"));
            put2.addColumn(Bytes.toBytes("attributes"), Bytes.toBytes("type"), Bytes.toBytes("Like"));
            put2.addColumn(Bytes.toBytes("attributes"),Bytes.toBytes("for_user"),Bytes.toBytes("Brendan"));
            put2.addColumn(Bytes.toBytes("attributes"),Bytes.toBytes("from_user"),Bytes.toBytes("Rick"));
            put2.addColumn(Bytes.toBytes("attributes"),Bytes.toBytes("for_thing"),Bytes.toBytes("link"));
            put2.addColumn(Bytes.toBytes("attributes"), Bytes.toBytes("link"), Bytes.toBytes("link"));


            List<Put> puts = new ArrayList<Put>();
            puts.add(put1);
            puts.add(put2);

            table.put(puts);


    }
}
```

Use the put method with the List of Puts

Example 11:

Retrieving data from a single row

Get

Get

```
get 'notifications',2
```

Data is retrieved from HBase tables using the get operation

Let's see the equivalent in Java

Get

```
get 'notifications',2,'metrics:open'
```

1. Specify the row id, column

Get

2. Connect to HBase and get a Table object

Connection + HTable

3. Fetch the result

Result

Get

```
get 'notifications',2,'metrics:open'
```

```
public class singleGet {

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

        Configuration conf = HBaseConfiguration.create();
        Connection connection = ConnectionFactory.createConnection(conf);
        Table table = connection.getTable("notifications");

        Get get =new Get(Bytes.toBytes("2"));

        get.addColumn(Bytes.toBytes("attributes"), Bytes.toBytes("type"));
        get.addColumn(Bytes.toBytes("metrics"), Bytes.toBytes("open"));

        Result result = table.get(get);

        byte[] val= result.getValue(Bytes.toBytes("attributes"),Bytes.toBytes("type"));
        System.out.println("Value:"+Bytes.toString(val));

    }
}
```

Get

```
get 'notifications',2,'metrics:open'
```

```
public class singleGet {

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

        Configuration conf = HBaseConfiguration.create();
        Connection connection = ConnectionFactory.createConnection(conf);
        Table table = connection.getTable("notifications");

        Get get =new Get(Bytes.toBytes("2"));

        get.addColumn(Bytes.toBytes("attributes"), Bytes.toBytes("type"));
        get.addColumn(Bytes.toBytes("metrics"), Bytes.toBytes("open"));
```

Use the Get class to specify the row id, columns whose values will be retrieved

Get

```
get 'notifications',2,'metrics:open'
```

```
public class singleGet {

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

        Configuration conf = HBaseConfiguration.create();
        Connection connection = ConnectionFactory.createConnection(conf);
        Table table = connection.getTable("notifications");

        Get get =new Get(Bytes.toBytes("2"));

        get.addColumn(Bytes.toBytes("attributes"), Bytes.toBytes("type"));
        get.addColumn(Bytes.toBytes("metrics"), Bytes.toBytes("open"));
```

The row id is specified using a byte array

Get

```
get 'notifications',2,'metrics:open'
```

```
public class singleGet {

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

        Configuration conf = HBaseConfiguration.create();
        Connection connection = ConnectionFactory.createConnection(conf);
        Table table = connection.getTable("notifications");

        Get get =new Get(Bytes.toBytes("2"));

        get.addColumn(Bytes.toBytes("attributes"), Bytes.toBytes("type"));
        get.addColumn(Bytes.toBytes("metrics"), Bytes.toBytes("open"));
```

The Get is specific to a row id

Get

```
get 'notifications',2,'metrics:open'
```

```
public class singleGet {

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

        Configuration conf = HBaseConfiguration.create();
        Connection connection = ConnectionFactory.createConnection(conf);
        Table table = connection.getTable("notifications");

         Get get =new Get(Bytes.toBytes("2"));

get.addColumn(Bytes.toBytes("attributes"), Bytes.toBytes("type"));
get.addColumn(Bytes.toBytes("metrics"), Bytes.toBytes("open"));

        Result result = table.get(get);

        byte[] val=
        System.out.

    }
}
```

Use addColumn to specify the columns whose values should be retrieved

Get

```
get 'notifications',2,'metrics:open','attributes:type'
```

```
public class singleGet {

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

        Configuration conf = HBaseConfiguration.create();
        Connection connection = ConnectionFactory.createConnection(conf);
        Table table = connection.getTable("notifications");

         Get get =new Get(Bytes.toBytes("2"));

        get.addColumn(Bytes.toBytes("attributes"), Bytes.toBytes("type"));
        get.addColumn(Bytes.toBytes("metrics"), Bytes.toBytes("open"));

        Result result = table.get(get);

        byte[] val= result.getValue(Bytes.toBytes("attributes"),Bytes.toBytes("type"));
        System.out.println("Value:"+Bytes.toString(val));

    }
}
```

You can specify any number of columns for that specific row id

Get

```
get 'notifications',2,'metrics:open'
```

```
public class singleGet {

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

        Configuration conf = HBaseConfiguration.create();
        Connection connection = ConnectionFactory.createConnection(conf);
        Table table = connection.getTable("notifications");

        Get get =new Get(Bytes.toBytes("2"));

        get.addColumn(Bytes.toBytes("attributes"), Bytes.toBytes("type"));
        get.addColumn(Bytes.toBytes("metrics"), Bytes.toBytes("open"));

        Result result = table.get(get);

        byte[] val= result.getValue(Bytes.toBytes("attributes"),Bytes.toBytes("type"));
        System.out.println("value:"+Bytes.toString(val));
    }
}
```

Do the usual boilerplate to get an HTable instance

Get

```
get 'notifications',2,'metrics:open'
```

```
public class singleGet {

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

        Configuration config = HBaseConfiguration.create();
        Connection connection = ConnectionFactory.createConnection(config);
        Table table = connection.getTable(TableName.valueOf("notifications"));

        Get get =new Get(Bytes.toBytes("2"));

        get.addColumn(Bytes.toBytes("attributes"), Bytes.toBytes("type"));
        get.addColumn(Bytes.toBytes("metrics"), Bytes.toBytes("open"));

        Result result = table.get(get);

        byte[] val= result.getValue(Bytes.toBytes("attributes"),Bytes.toBytes("type"));
        System.out.println("Value:"+Bytes.toString(val));

    }
}
```

Use the get method to fetch the result

Get

```
get 'notifications',2,'metrics:open'
```

```
public class singleGet {

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

        Configuration conf = HBaseConfiguration.create();
        Connection connection = ConnectionFactory.createConnection(conf);
        Table table = connection.getTable("notifications");

        Get get =new Get(Bytes.toBytes("2"));

        get.addColumn(Bytes.toBytes("attributes"), Bytes.toBytes("type"));
        get.addColumn(Bytes.toBytes("metrics"), Bytes.toBytes("open"));

        Result result = table.get(get);

        byte[] val= result.getValue(Bytes.toBytes("attributes"),Bytes.toBytes("type"));
        System.out.println("Value:"+Bytes.toString(val));

    }
}
```

The get method returns a Result object

Get

```
get 'notifications',2,'metrics:open'
```

Keys

```
attributes:type
metrics:open
```

Values

```
value=Comment
value=1
```

```
byte[] val=result.getValue(Bytes.toBytes("attributes"),Bytes.toBytes("type"));
```

The Result object is like a map

Get

```
get 'notifications',2,'metrics:open'
```

Keys

```
attributes:type
metrics:open
```

Values

```
value=Comment
value=1
```

```
public class singleGet {

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

        Configuration conf = HBaseConfiguration.create();
                                                    n(conf);
                                                    ("
        get.addColumn(Bytes.toBytes("attributes"), Bytes.toBytes("type"));
        get.addColumn(Bytes.toBytes("metrics"), Bytes.toBytes("open"));

        Result result = table.get(get);

byte[] val= result.getValue(Bytes.toBytes("attributes"),Bytes.toBytes("type"));
        System.out.println("Value:"+Bytes.toString(val));

    }
}
```

You can lookup the value
for a column family, column

Get

```
get 'notifications',2,'metrics:open'
```

Keys

```
attributes:type
metrics:open
```

Values

```
value=Comment
value=1
```

```
byte[] val= result.getValue(Bytes.toBytes("attributes"),Bytes.toBytes("type"));
```

The data in the result object is in the form of a byte array

Get

```
get 'notifications',2,'metrics:open'
```

```
public class singleGet {

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

        Configuration conf = HBaseConfiguration.create();
        Connection connection = ConnectionFactory.createConnection(conf);
        Table table = connection.getTable("notifications");

        Get get = new Get(Bytes.toBytes("2"));

        get.addColumn(Bytes.toBytes("attributes"),Bytes.toBytes("type"));
        get.addColumn(Bytes.toBytes("metrics"),Bytes.toBytes("open"));

        Result result = table.get(get);

byte[] val= result.getValue(Bytes.toBytes("attributes"),Bytes.toBytes("type"));
System.out.println("Value:"+Bytes.toString(val));

    }
}
```

The Bytes helper class has a method to convert the byte array to String

Get

```
get 'notifications',2,'metrics:open'
```

```
public class singleGet {

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

        Configuration conf = HBaseConfiguration.create();
        Connection connection = ConnectionFactory.createConnection(conf);
        Table table = connection.getTable(TableName.valueOf("notifications"));

        Get get = new Get(Bytes.toBytes("2"));

        get.addColumn(Bytes.toBytes("attributes"), Bytes.toBytes("type"));
        get.addColumn(Bytes.toBytes("metrics"), Bytes.toBytes("open"));

        Result result = table.get(get);

byte[] val= result.getValue(Bytes.toBytes("attributes"),Bytes.toBytes("type"));
System.out.println("value:"+Bytes.toString(val));

    }
}
```

The getValue method will only give you 1 value at a time

Get

```
get 'notifications',2,'metrics:open'
```

```
public class singleGet {

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

        Get get = new Get(Bytes.toBytes("2"));

        get.addColumn(Bytes.toBytes("attributes"), Bytes.toBytes("type"));
        get.addColumn(Bytes.toBytes("metrics"), Bytes.toBytes("open"));

        Result result = table.get(get);

byte[] val= result.getValue(Bytes.toBytes("attributes"),Bytes.toBytes("type"));
System.out.println("value:"+Bytes.toString(val));

    }
}
```

The Result object is not iterable, so we need to do some work to get all the values from it

Get

```
get 'notifications',2,'metrics:open'
```

```
public class singleGet {

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

        Configuration conf = HBaseConfiguration.create();
        Connection connection = ConnectionFactory.createConnection(conf);
        Table table = connection.getTable("notifications");

        Get get = new Get(Bytes.toBytes("2"));

        get.addColumn(Bytes.toBytes("attributes"), Bytes.toBytes("type"));
        get.addColumn(Bytes.toBytes("metrics"), Bytes.toBytes("open"));

        Result result = table.get(get);

byte[] val= result.getValue(Bytes.toBytes("attributes"),Bytes.toBytes("type"));
System.out.println("Value:"+Bytes.toString(val));

printAllValues(result);

    }
}
```

Let's write a function to print all the values from the Result object

Get

printAllValues(result);

```
private static void printAllValues(Result result){

    NavigableMap<byte[],NavigableMap<byte[],NavigableMap<Long,byte[]>>> resultMap =
result.getMap();

    for(byte[] columnFamily:resultMap.keySet()){
        String cf = Bytes.toString(columnFamily);
        NavigableMap<byte[],NavigableMap<Long,byte[]>> columnMap = resultMap.get(columnFamily);

        for(byte[] column:columnMap.keySet()){
            String col = Bytes.toString(column);
            NavigableMap<Long,byte[]> timestampMap = columnMap.get(column);

            for(Long timestamp:timestampMap.keySet()) {
                String ts = timestamp.toString();
                String value = Bytes.toString(timestampMap.get(timestamp) );
                System.out.println("Column Family: " + cf
                        + " Column: "+col+" Value: "+value );
            }
        }
    }
```

Get

printAllValues(result);

```
NavigableMap<byte[],
        NavigableMap<byte[],
                NavigableMap<Long,byte[]>>> resultMap = result.getMap();
```

The Result object's **getMap method** returns a Java Map object

Get

printAllValues(result);

```
NavigableMap<byte[],
        NavigableMap<byte[],
                NavigableMap<Long,byte[]>>> resultMap = result.getMap();
```

getMap returns the values as a nested Map

Get

printAllValues(result);

```
NavigableMap<byte[],
        NavigableMap<byte[],
                NavigableMap<Long,byte[]>>> resultMap = result.getMap();
```

<ColumnFamily,
<Column,
<Timestamp,Value>>>

Get

printAllValues(result);

```
private static void printAllValues(Result result){
    NavigableMap<byte[],
            NavigableMap<byte[],
                    NavigableMap<Long,byte[]>>> resultMap = result.getMap();
```

<ColumnFamily,
<Column,
<Timestamp,Value>>>

```
attributes:type    timestamp=1467181276487    value=Comment
metrics:open       timestamp=1467184097569    value=1
```

Get

printAllValues(result);

```
private static void printAllValues(Result result){

 NavigableMap<byte[]
        NavigableMap<byte[]
                NavigableMap<Long,byte[]>>> resultMap = result.getMap();
```

<ColumnFamily,

<Column,

<Timestamp,Value>>>

```
attributes:type  timestamp=1467181276487  value=Comment
metrics:open     timestamp=1467184097569  value=1
```

Get

printAllValues(result);

```
private static void printAllValues(Result result){

 NavigableMap<byte[],
         NavigableMap<byte[],
                 NavigableMap<Long,byte[]>>> resultMap = result.getMap();
```

<ColumnFamily,
<Column,
<Timestamp,Value>>>

```
attributes:type  timestamp=1467181276487  value=Comment
metrics:open     timestamp=1467184097569  value=1
```

Get

printAllValues(result);

```
private static void printAllValues(Result result){

 NavigableMap<byte[],
         NavigableMap<byte[],
                 NavigableMap<Long, byte[]>>> resultMap = result.getMap();
```

<ColumnFamily,
<Column,
<Timestamp, Value>>>

```
attributes:type  timestamp=146718127648 7  value=Comment
metrics:open     timestamp=146718409756 9  value=1
```

Get

printAllValues(result);

For each ColumnFamily

```
for(byte[] columnFamily:resultMap.keySet()){
```

For each Column

```
for(byte[] column:columnMap.keySet()){
```

For each Timestamp

```
for(Long timestamp:timestampMap.keySet()) {
```

Iterate through each level of the nested map

Get

printAllValues(result);

For each ColumnFamily

```
for(byte[] columnFamily:resultMap.keySet()){
```

For each Column

```
    for(byte[] column:columnMap.keySet()){
```

For each Timestamp

```
        for(Long timestamp:timestampMap.keySet()) {
```

Print the Value

Get

printAllValues(result);

For each ColumnFamily

```
private static void printAllValues(Result result){

    NavigableMap<byte[],NavigableMap<byte[],NavigableMap<Long,byte[]>>> resultMap = result.getMap();

    for(byte[] columnFamily:resultMap.keySet()) {
        String cf = Bytes.toString(columnFamily);
        NavigableMap<byte[],NavigableMap<Long,byte[]>> columnMap = resultMap.get(columnFamily);

        for(byte[] column:columnMap.keySet()){
            String col = Bytes.toString(column);
            NavigableMap<Long,byte[]> timestampMap = columnMap.get(column);

            for(Long timestamp:timestampMap.keySet()) {
                String ts = timestamp.toString();
                String value = Bytes.toString(timestampMap.get(timestamp) );
                System.out.println("Column Family: " + cf
                        + " Column: "+col+" Value: "+value );
            }
        }
    }
```

First we iterate through the column families

Get

printAllValues(result);

For each ColumnFamily

```
private static void printAllValues(Result result){

    NavigableMap<byte[],NavigableMap<byte[],NavigableMap<Long,byte[]>>> resultMap = result.getMap();

    for(byte[] columnFamily:resultMap.keySet()){
        String cf = Bytes.toString(columnFamily);
        NavigableMap<byte[],NavigableMap<Long,byte[]>> columnMap = resultMap.get(columnFamily);

        for(byte[] column:columnMap.keySet()){
            String col = Bytes.toString(column);
            NavigableMap<Long,byte[]> timestampMap = columnMap.get(column);

            for(Long timestamp:timestampMap.keySet()) {
                String ts = timestamp.toString();
                String value = Bytes.toString(timestampMap.get(timestamp));
                System.out.println("Column Family: "+cf
                        + " Column: "+col+" Value: "+value);
            }
        }
    }
```

The list of distinct column families is the list of keys from the result map

Get

printAllValues(result);

For each ColumnFamily

```
private static void printAllValues(Result result){

    NavigableMap<byte[],NavigableMap<byte[],NavigableMap<Long,byte[]>>> resultMap = result.getMap();

for(byte[] columnFamily:resultMap.keySet()){
String cf = Bytes.toString(columnFamily);
NavigableMap<byte[],NavigableMap<Long,byte[]>> columnMap = resultMap.get(columnFamily);

        for(byte[] column:columnMap.keySet()){
            String col = Bytes.toString(column);
            NavigableMap<Long,byte[]> timestampMap = columnMap.get(column);

            for(Long timestamp:timestampMap.keySet()) {
                String ts = timestamp.toString();
                String value = Bytes.toString(timestampMap.get(timestamp));
                System.out.println("Column Family: " +cf
                        + " Column: "+col+" Value: "+value);
            }
        }
    }
```

Convert the byte array representing the Column Family to a String

Get

printAllValues(result);

For each ColumnFamily

```
for(byte[] columnFamily:resultMap.keySet()){
String cf = Bytes.toString(columnFamily);
NavigableMap<byte[],NavigableMap<Long,byte[]>> columnMap = resultMap.get(columnFamily);
```

Get another map representing the columns within that column family

Get

printAllValues(result);

```
private static void printAllValues(Result result){

    NavigableMap<byte[],NavigableMap<byte[],NavigableMap<Long,byte[]>>> resultMap = result.getMap();

    for(byte[] columnFamily:resultMap.keySet()){
        String cf = Bytes.toString(columnFamily);
        NavigableMap<byte[],NavigableMap<Long,byte[]>> columnMap = resultMap.get(columnFamily);
        for(byte[] column:columnMap.keySet()){
            String col = Bytes.toString(column);
            NavigableMap<Long,byte[]> timestampMap = columnMap.get(column);

            for(Long timestamp:timestampMap.keySet()) {
                String ts = timestamp.toString();
                String value = Bytes.toString(timestampMap.get(timestamp) );
                System.out.println("Column Family: " + cf
                        + " Column: "+col+" Value: "+value );
            }
        }
    }
```

For each ColumnFamily

For each Column

From the column map, get the list of columns in the column family

Get

printAllValues(result);

```
private static void printAllValues(Result result){

    NavigableMap<byte[],NavigableMap<byte[],NavigableMap<Long,byte[]>>> resultMap = result.getMap();

    for(byte[] columnFamily:resultMap.keySet()){
        String cf = Bytes.toString(columnFamily);
        NavigableMap<byte[],NavigableMap<Long,byte[]>> columnMap = resultMap.get(columnFamily);
for(byte[] column:columnMap.keySet()){
String col = Bytes.toString(column);
NavigableMap<Long,byte[]> timestampMap = columnMap.get(column);

            for(Long timestamp:timestampMap.keySet()) {
                String ts = timestamp.toString();
                String value = Bytes.toString(timestampMap.get(timestamp) );
                System.out.println("Column Family: " + cf
                        + " Column: "+col+" Value: "+value );
            }
        }
    }
```

For each ColumnFamily

For each Column

Convert the byte array representing the Column to a String

Get

printAllValues(result);

For each ColumnFamily

For each Column

```
for(byte[] column:columnMap.keySet()){
String col = Bytes.toString(column);
NavigableMap<Long,byte[]> timestampMap = columnMap.get(column);
```

Get another map representing the timestamps (versions) within that column

Get

printAllValues(result);

```
public static void printAllValues(Result result){
    NavigableMap<byte[],NavigableMap<byte[],NavigableMap<Long,byte[]>>> resultMap = result.getMap();

    for(byte[] columnFamily:resultMap.keySet()){
        String cf = Bytes.toString(columnFamily);
        NavigableMap<byte[],NavigableMap<Long,byte[]>> columnMap = resultMap.get(columnFamily);

        for(byte[] column:columnMap.keySet()){
            String col = Bytes.toString(column);
            NavigableMap<Long,byte[]> timestampMap = columnMap.get(column);

            for(Long timestamp:timestampMap.keySet()) {
                String ts = timestamp.toString();
                String value = Bytes.toString(timestampMap.get(timestamp));
                System.out.println("Column Family: " + cf
                        + " Column: "+col+" Value: "+value );
            }
        }
    }
```

For each ColumnFamily

For each Column

For each Timestamp

From the timestamp map, get the list of versions for that column

Get

```
printAllValues(result);
```

```
public static void printAllValues(Result result){
    NavigableMap<byte[],NavigableMap<byte[],NavigableMap<Long,byte[]>>> resultMap = result.getMap();

    for(byte[] columnFamily:resultMap.keySet()){
        String cf = Bytes.toString(columnFamily);
        NavigableMap<byte[],NavigableMap<Long,byte[]>> columnMap = resultMap.get(columnFamily);

        for(byte[] column:columnMap.keySet()){
            String col = Bytes.toString(column);
            NavigableMap<Long,byte[]> timestampMap = columnMap.get(column);

            for(Long timestamp:timestampMap.keySet()) {
                String ts = timestamp.toString();
                String value = Bytes.toString(timestampMap.get(timestamp));
                System.out.println("Column Family: " + cf
                        + " Column: "+col+" Value: "+value );
            }
        }
    }
```

For each ColumnFamily

For each Column

For each Timestamp

Convert the Long representing the version to a String

Get

printAllValues(result);

```
public static void printAllValues(Result result){
    NavigableMap<byte[],NavigableMap<byte[],NavigableMap<Long,byte[]>>> resultMap = result.getMap();

    for(byte[] columnFamily:resultMap.keySet()){
        String cf = Bytes.toString(columnFamily);
        NavigableMap<byte[],NavigableMap<Long,byte[]>> columnMap = resultMap.get(columnFamily);

        for(byte[] column:columnMap.keySet()){
            String col = Bytes.toString(column);
            NavigableMap<Long,byte[]> timestampMap = columnMap.get(column);

            for(Long timestamp:timestampMap.keySet()) {
                String ts = timestamp.toString();
                String value = Bytes.toString(timestampMap.get(timestamp));
                System.out.println("Column Family: " + cf
                        + " Column: "+col+" Value: "+value );
            }
        }
    }
```

For each ColumnFamily

For each Column

For each Timestamp

Lookup the value for this version and convert it to a String

Get

printAllValues(result);

```
public static void printAllValues(Result result){
    NavigableMap<byte[],NavigableMap<byte[],NavigableMap<Long,byte[]>>> resultMap = result.getMap();

    for(byte[] columnFamily:resultMap.keySet()){
        String cf = Bytes.toString(columnFamily);
        NavigableMap<byte[],NavigableMap<Long,byte[]>> columnMap = resultMap.get(columnFamily);

        for(byte[] column:columnMap.keySet()){
            String col = Bytes.toString(column);
            NavigableMap<Long,byte[]> timestampMap = columnMap.get(column);

            for(Long timestamp:timestampMap.keySet()) {
                String ts = timestamp.toString();
                String value = Bytes.toString(timestampMap.get(timestamp));
                System.out.println("Column Family: " + cf
                        + " Column: "+col+" Value: "+value );
            }
        }
    }
}
```

For each ColumnFamily

For each Column

For each Timestamp

Print out the values

Get

printAllValues(result);

```
private static void printAllValues(Result result){

    NavigableMap<byte[],NavigableMap<byte[],NavigableMap<Long,byte[]>>> resultMap = result.getMap();

    for(byte[] columnFamily:resultMap.keySet()){
        String cf = Bytes.toString(columnFamily);
        NavigableMap<byte[],NavigableMap<Long,byte[]>> columnMap = resultMap.get(columnFamily);

        for(byte[] column:columnMap.keySet()){
            String col = Bytes.toString(column);
            NavigableMap<Long,byte[]> timestampMap = columnMap.get(column);

            for(Long timestamp:timestampMap.keySet()) {
                String
                String
                Syste

            }
        }
    }
```

```
Column Family: attributes Column: type Value: Comment
Column Family: metrics Column: open Value: 1
```

Example 12:
Retrieving data from multiple rows

List<Get>

List <Get>

Like Put objects, get objects are specific to 1 row id

```
Get get =new Get(Bytes.toBytes("2"));
```

To retrieve values from multiple rows, use a list of Gets

List <Get>

Let's get data from 2 row ids

```
public class listGets {

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

        Configuration conf = HBaseConfiguration.create();
        Connection connection = ConnectionFactory.createConnection(conf);
        Table table = connection.getTable("notifications");

        List<Get> gets = new ArrayList<Get>();


        Get get1 =new Get(Bytes.toBytes("2"));
        get1.addColumn(Bytes.toBytes("attributes"), Bytes.toBytes("type"));
        get1.addColumn(Bytes.toBytes("attributes"), Bytes.toBytes("for_user"));

        Get get2 =new Get(Bytes.toBytes("3"));
        get2.addColumn(Bytes.toBytes("attributes"), Bytes.toBytes("type"));
        get2.addColumn(Bytes.toBytes("attributes"), Bytes.toBytes("from_user"));

        gets.add(get1);
        gets.add(get2);


        Result[] results = table.get(gets);

        for(Result result:results) {
           printAllValues(result);
        }
    }
}
```

row	column
2	attributes : type
2	attributes : for user
3	attributes : type
3	attributes : for user

List <Get>

```
public class listGets {

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

        Configuration conf = HBaseConfiguration.create();
        Connection connection = ConnectionFactory.createConnection(conf);
        Table table = connection.getTable("notifications");

        List<Get> gets = new ArrayList<Get>();
```

```
Get get1 =new Get(Bytes.toBytes("2"));
  get1.addColumn(Bytes.toBytes("attributes"), Bytes.toBytes("type"));
  get1.addColumn(Bytes.toBytes("attributes"), Bytes.toBytes("for_user"));
```

```
Get get2 =new Get(Bytes.toBytes("3"));
  get2.addColumn(Bytes.toBytes("attributes"), Bytes.toBytes("type"));
  get2.addColumn(Bytes.toBytes("attributes"), Bytes.toBytes("from_user"));
```

```
        ets.add(get1);
        ers.add(get2);
        [] r
        for(Result result:results) {
            printAllValues(result);
        }
    }
}
```

Create a Get for row id 2

row id	column
2	attributes : type
2	attributes : for user

List <Get>

```
public class listGets {

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

        Configuration conf = HBaseConfiguration.create();
        Connection connection = ConnectionFactory.createConnection(conf);
        Table table = connection.getTable("notifications");

        List<Get> gets = new ArrayList<Get>();
```

```
Get get1 =new Get(Bytes.toBytes("2"));
  get1.addColumn(Bytes.toBytes("attributes"), Bytes.toBytes("type"));
  get1.addColumn(Bytes.toBytes("attributes"), Bytes.toBytes("for_user"));
```

```
Get get2 =new Get(Bytes.toBytes("3"));
  get2.addColumn(Bytes.toBytes("attributes"), Bytes.toBytes("type"));
  get2.addColumn(Bytes.toBytes("attributes"), Bytes.toBytes("from_user"));
```

```
        gets.add(get1);
        gets.add(get2);
    }
}
```

Create another Get for row id 3

row id	column
3	attributes : type
3	attributes : for user

List <Get>

```
public class listGets {

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

        Configuration conf = HBaseConfiguration.create();
        Connection connection = ConnectionFactory.createConnection(conf);
        Table table = connection.getTable("notifications");



        Get get1 =new Get(Bytes.toBytes("2"));
        get1.addColumn(Bytes.toBytes("attributes"), Bytes.toBytes("type"));
        get1.addColumn(Bytes.toBytes("attributes"), Bytes.toBytes("for_user"));

        Get get2 =new Get(Bytes.toBytes("3"));
        get2.addColumn(Bytes.toBytes("attributes"), Bytes.toBytes("type"));
        get2.addColumn(Bytes.toBytes("attributes"), Bytes.toBytes("from_user"));

        List<Get> gets = new ArrayList<Get>();
        gets.add(get1);
        gets.add(get2);

        Result[] results = table.get(gets);

        for(Result result:results) {
            printAllValues(result);
        }
    }
}
```

Create a List and add the Get objects to it

List <Get>

Do the usual boilerplate to get a connection and a HTable

```
public class listGets {

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

Configuration conf = HBaseConfiguration.create();
Connection connection = ConnectionFactory.createConnection(conf);
Table table = connection.getTable("notifications");

        List<Get> gets = new ArrayList<Get>();

        Get get1 =new Get(Bytes.toBytes("2"));
        get1.addColumn(Bytes.toBytes("attributes"), Bytes.toBytes("type"));
        get1.addColumn(Bytes.toBytes("attributes"), Bytes.toBytes("for_user"));

        Get get2 =new Get(Bytes.toBytes("3"));
        get2.addColumn(Bytes.toBytes("attributes"), Bytes.toBytes("type"));
        get2.addColumn(Bytes.toBytes("attributes"), Bytes.toBytes("from_user"));

        gets.add(get1);
        gets.add(get2);

Result[] results = table.get(gets)

        for(Result result:results) {
            printAllValues(result);
        }
    }
}
```

Pass the list of gets to the get method

List <Get>

A Result array is returned

```
Result[] results = table.get(gets);
```

A Result for each Get in the list

List <Get>

Iterate through the Result array and print the results

```
public class listGets {

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

        Configuration conf = HBaseConfiguration.create();
        Connection connection = ConnectionFactory.createConnection(conf);
        Table table = connection.getTable("notifications");

        List<Get> gets = new ArrayList<Get>();


        Get get1 =new Get(Bytes.toBytes("2"));
        get1.addColumn(Bytes.toBytes("attributes"), Bytes.toBytes("type"));
        get1.addColumn(Bytes.toBytes("attributes"), Bytes.toBytes("from_user"));

        Get get2 =new Get(Bytes.toBytes("3"));
        get2.addColumn(Bytes.toBytes("attributes"), Bytes.toBytes("type"));
        get2.addColumn(Bytes.toBytes("attributes"), Bytes.toBytes("from_user"));

        gets.add(get1);
        gets.add(get2);


        Result[] results = table.get(gets);

for(Result result:results) {
    printAllValues(result);
        }


    }
}
```

Example 13:

Deleting from a single row

Delete

Delete

```
delete 'notifications',2, 'attributes:for_user'
```

Data is deleted from HBase tables using the delete operation

Let's see the equivalent in Java

Delete

```
delete 'notifications',2, 'attributes:for_user'
```

1. Specify the row id, column

`Delete`

2. Connect to HBase and get a Table object

`Connection + HTable`

3. Delete the value

Delete

```
delete 'notifications',2, 'attributes:for_user'
```

```
public class singleDelete {

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

        Configuration conf = HBaseConfiguration.create();
        Connection connection = ConnectionFactory.createConnection(conf);
        Table table = connection.getTable("notifications");

        Delete delete =new Delete(Bytes.toBytes("2"));

        delete.addColumn(Bytes.toBytes("attributes"), Bytes.toBytes("type"));
        delete.addColumn(Bytes.toBytes("attributes"), Bytes.toBytes("for_user"));

        table.delete(delete);

        table.close();

    }
}
```

Delete

```
delete 'notifications',2, 'attributes:for_user'
```

Use the Delete class to specify the row , column that will be deleted

```
public class singleDelete {

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

        Configuration conf = HBaseConfiguration.create();
        Connection connection = ConnectionFactory.createConnection(conf);
        Table table = connection.getTable("notifications");

        Delete delete =new Delete(Bytes.toBytes("2"));

        delete.addColumn(Bytes.toBytes("attributes"), Bytes.toBytes("type"));
        delete.addColumn(Bytes.toBytes("attributes"), Bytes.toBytes("for_user"));

        table.delete(delete);

        table.close();

    }
}
```

Delete

```
delete 'notifications',2, 'attributes:for_user'
```

```
public class singleDelete {

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

        Configuration conf = HBaseConfiguration.create();
        Connection connection = ConnectionFactory.createConnection(conf);
        Table table = connection.getTable("notifications");

        Delete delete =new Delete(Bytes.toBytes("2"));

        delete.addColumn(Bytes.toBytes("attributes"), Bytes.toBytes("type"));
        delete.addColumn(Bytes.toBytes("attributes"), Bytes.toBytes("for_user"));

        table.delete(delete);

        table.close();

    }
}
```

The row id is specified using a byte array

Delete

```
delete 'notifications',2, 'attributes:for_user'
```

```
public class singleDelete {

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

        Configuration conf = HBaseConfiguration.create();
        Connection connection = ConnectionFactory.createConnection(conf);
        Table table = connection.getTable("notifications");

        Delete delete =new Delete(Bytes.toBytes("2"));

        delete.addColumn(Bytes.toBytes("attributes"), Bytes.toBytes("type"));
        delete.addColumn(Bytes.toBytes("attributes"), Bytes.toBytes("for_user"));

        table.delete(delete);

        table.close();

    }
}
```

The Delete is specific to a row id

Delete

```
delete 'notifications',2, attributes:for_user
```

```
public class singleDelete {

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

        Configuration conf = HBaseConfiguration.create();
        Connection connection = ConnectionFactory.createConnection(conf);
        Table table = connection.getTable("notifications");

Delete delete =new Delete(Bytes.toBytes("2"));
delete.addColumn(Bytes.toBytes("attributes"), Bytes.toBytes("type"));
delete.addColumn(Bytes.toBytes("attributes"), Bytes.toBytes("for_user"));

        table.delete(delete);
        table.close();

    }
}
```

Use addColumn to specify the columns whose values should be deleted

Delete

```
delete 'notifications',2, 'attributes:for_user'
```

```
public class singleDelete {

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

        Configuration conf = HBaseConfiguration.create();
        Connection connection = ConnectionFactory.createConnection(conf);
        Table table = connection.getTable("notifications");

Delete delete =new Delete(Bytes.toBytes("2"));
delete.addColumn(Bytes.toBytes("attributes"), Bytes.toBytes("type"));
delete.addColumn(Bytes.toBytes("attributes"), Bytes.toBytes("for_user"));

        table.delete(delete);
        table.close();

    }
}
```

You can specify any number of columns for that specific row id

Delete

```
delete 'notifications',2, 'attributes:for_user'
```

```
public class singleDelete {

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

        Configuration conf = HBaseConfiguration.create();
        Connection connection = ConnectionFactory.createConnection(conf);
        Table table = connection.getTable("notifications");

        Delete delete = new Delete(Bytes.toBytes("2"));

        delete.addColumn(Bytes.toBytes("attributes"), Bytes.toBytes("type"));
        delete.addColumn(Bytes.toBytes("attributes"), Bytes.toBytes("for_user"));

        table.delete(delete);

        table.close();

    }
}
```

Do the usual boilerplate to get an HTable instance

Use the delete method to delete the value

Example 14:
Deleting from multiple rows

List<Delete>

List <Delete>

Like Put and get objects,Delete objects are specific to 1 row id

```
Delete delete =new Delete(Bytes.toBytes("2"));
```

To delete values from multiple rows, use a list of Deletes

List <Delete>

Let's delete data from 2 row ids

```
public class listDeletes {

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

        Configuration conf = HBaseConfiguration.create();
        Connection connection = ConnectionFactory.createConnection(conf);
        Table table = connection.getTable("notifications");

        List<Delete> deletes = new ArrayList<Delete>();

        Delete delete1 =new Delete(Bytes.toBytes("1"));

        delete1.addColumn(Bytes.toBytes("attributes"), Bytes.toBytes("type"));
        deletes.add(delete1);

        Delete delete2 =new Delete(Bytes.toBytes("3"));
        delete2.addColumn(Bytes.toBytes("attributes"), Bytes.toBytes("type"));
        deletes.add(delete2);

        table.delete(deletes);

    }
}
```

row id	column
1	attributes : type
3	attributes : type

List <Delete>

```
public class listDeletes {

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

        Configuration conf = HBaseConfiguration.create();
        Connection connection = ConnectionFactory.createConnection(conf);
        Table table = connection.getTable("notifications");

Delete delete1 =new Delete(Bytes.toBytes("1"));
delete1.addColumn(Bytes.toBytes("attributes"),
Bytes.toBytes("type"));

Delete delete2 =new Delete(Bytes.toBytes("3"));
delete2.addColumn(Bytes.toBytes("attributes"),
Bytes.toBytes("type"));

        List<Delete> deletes = new ArrayList<Delete>();
        deletes.add(delete1);
        deletes.add(delete2);
        table.delete(deletes);
    }}
```

Create a Delete for row id 1

row id	column
1	attributes : type

List <Delete>

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

    Configuration conf = HBaseConfiguration.create();
    Connection connection = ConnectionFactory.createConnection(conf);
    Table table = connection.getTable("notifications");

    Delete delete1 =new Delete(Bytes.toBytes("1"));
    delete1.addColumn(Bytes.toBytes("attributes"),
    Bytes.toBytes("type"));

    Delete delete2 =new Delete(Bytes.toBytes("3"));
    delete2.addColumn(Bytes.toBytes("attributes"),
    Bytes.toBytes("type"));

    List<Delete> deletes = new ArrayList<Delete>();
    deletes.add(delete1);
    deletes.add(delete2);

    table.delete(deletes);
  }}
```

Create another Delete for row id 3

row id	column
3	attributes : type

List <Delete>

```
public class listDeletes {

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

        Configuration conf = HBaseConfiguration.create();
        Connection connection = ConnectionFactory.createConnection(conf);
        Table table = connection.getTable("notifications");

        Delete delete1 =new Delete(Bytes.toBytes("1"));
        delete1.addColumn(Bytes.toBytes("attributes"), Bytes.toBytes("type"));

        Delete delete2 =new Delete(Bytes.toBytes("3"));
        delete2.addColumn(Bytes.toBytes("attributes"), Bytes.toBytes("type"));

        List<Delete> deletes = new ArrayList<Delete>();
        deletes.add(delete1);
        deletes.add(delete2);

        table.delete(deletes);
    }}
```

Create a List and add the Delete objects to it

List <Delete>

Do the usual boilerplate to get a connection and a HTable

```
public class listDeletes {

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

Configuration conf = HBaseConfiguration.create();
Connection connection = ConnectionFactory.createConnection(conf);
Table table = connection.getTable("notifications");

        Delete delete1 =new Delete(Bytes.toBytes("1"));
        delete1.addColumn(Bytes.toBytes("attributes"), Bytes.toBytes("type"));

        Delete delete2 =new Delete(Bytes.toBytes("2"));
        delete2.addColumn(Bytes.toBytes("attributes"), Bytes.toBytes("type"));

        List<Delete> deletes = new ArrayList<Delete>();
        deletes.add(delete1);
        deletes.add(delete2);

        table.delete(deletes);
 }}
```

Pass the list of Deletes to the delete method

Example 15:

Running a combination of puts, gets, deletes

batch

Using the batch method of HTable

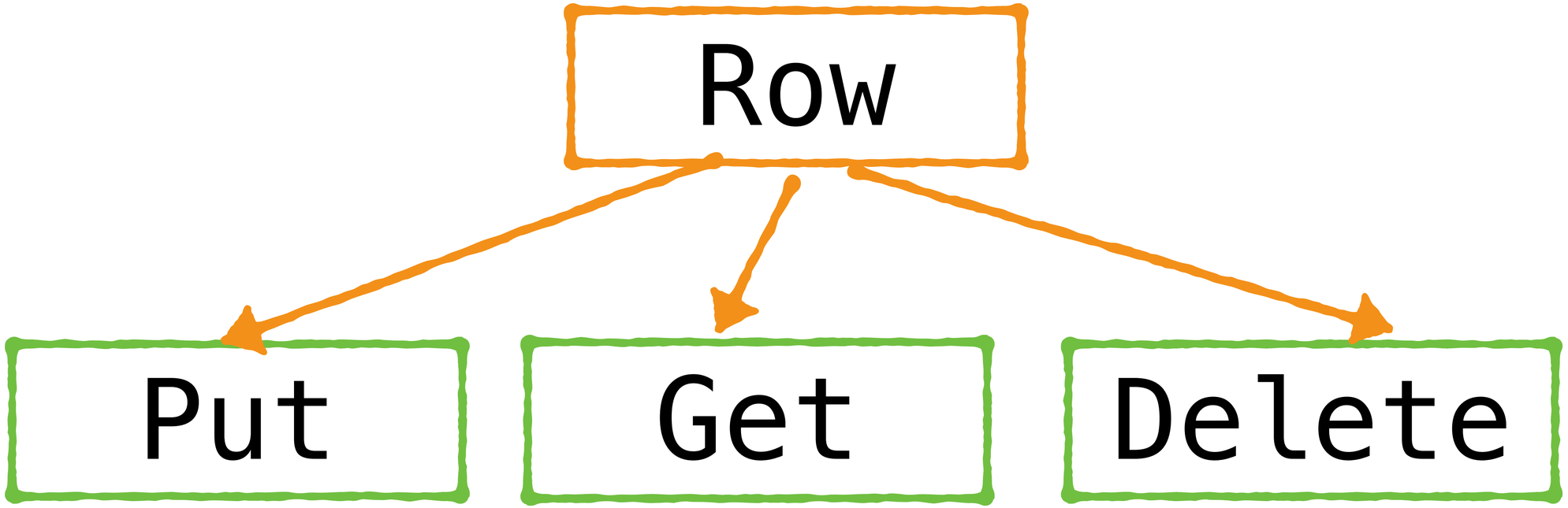
You can perform a combination of put, get, delete operations

The batch method

Takes a List of Row objects

each representing operations
like Put, Get, Delete

The batch method



The batch method

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

        Configuration conf = HBaseConfiguration.create();
        Connection connection = ConnectionFactory.createConnection(conf);
        Table table = connection.getTable("notifications");

        Put put =new Put(Bytes.toBytes("2"));
        put.addColumn(Bytes.toBytes("attributes"), Bytes.toBytes("type"), Bytes.toBytes("Comment"));
        put.addColumn(Bytes.toBytes("attributes"), Bytes.toBytes("for_user"), Bytes.toBytes("Swetha"));
        put.addColumn(Bytes.toBytes("metrics"), Bytes.toBytes("open"), Bytes.toBytes("0"));

        Get get =new Get(Bytes.toBytes("2"));
        get.addColumn(Bytes.toBytes("attributes"), Bytes.toBytes("type"));
        get.addColumn(Bytes.toBytes("metrics"), Bytes.toBytes("open"));

        Delete delete =new Delete(Bytes.toBytes("2"));
        delete.addColumn(Bytes.toBytes("attributes"), Bytes.toBytes("type"));
        delete.addColumn(Bytes.toBytes("attributes"), Bytes.toBytes("for_user"));

        List<Row> batch = new ArrayList<Row>();
        batch.add(put);
        batch.add(get);
        batch.add(delete);

        Object[] results = new Object[batch.size()];

        try{
            table.batch(batch,results);
        }catch (Exception e){
            System.err.println("Error: "+e);}

       for (int i=0; i<results.length; i++){
    Result res = (Result) results[i];
    if(!res.isEmpty()){
        printAllValues((Result) results[i]);
    }
}
   }
}
```

The batch method

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

        Configuration conf = HBaseConfiguration.create();
        Connection connection = ConnectionFactory.createConnection(conf);
        Table table = connection.getTable("notifications");
```

```
Put put =new Put(Bytes.toBytes("2"));
put.addColumn(Bytes.toBytes("attributes"), Bytes.toBytes("type"), Bytes.toBytes("Comment"));
put.addColumn(Bytes.toBytes("attributes"), Bytes.toBytes("for_user"), Bytes.toBytes("Swetha"));
put.addColumn(Bytes.toBytes("metrics"), Bytes.toBytes("open"), Bytes.toBytes("0"));

Get get =new Get(Bytes.toBytes("2"));
get.addColumn(Bytes.toBytes("attributes"), Bytes.toBytes("type"));
get.addColumn(Bytes.toBytes("metrics"), Bytes.toBytes("open"));

Delete delete =new Delete(Bytes.toBytes("2"));
delete.addColumn(Bytes.toBytes("attributes"), Bytes.toBytes("type"));
delete.addColumn(Bytes.toBytes("attributes"), Bytes.toBytes("for_user"));
```

```
        List<Row> batch = new ArrayList<Row>();
        batch.add(put);
        batch.add(get);
        batch.add(delete);

        Object[] results = new Object[batch.size()];

        try{
            table.batch(batch,results);
        }catch (Exception e){
            System.err.println("Error: "+e);}

        for (int i=0; i<results.length; i++){
    Result res = (Result) results[i];
    if(!res.isEmpty()){
        printAllValues((Result) results[i]);
    }
}   }
    }
}
```

Define 3 operations

The batch method

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

        Configuration conf = HBaseConfiguration.create();
        Connection connection = ConnectionFactory.createConnection(conf);
        Table table = connection.getTable("notifications");

        Put put =new Put(Bytes.toBytes("2"));
        put.addColumn(Bytes.toBytes("attributes"), Bytes.toBytes("type"), Bytes.toBytes("Comment"));
        put.addColumn(Bytes.toBytes("attributes"), Bytes.toBytes("for_user"), Bytes.toBytes("Swetha"));
        put.addColumn(Bytes.toBytes("metrics"), Bytes.toBytes("open"), Bytes.toBytes("0"));

        Get get =new Get(Bytes.toBytes("2"));
        get.addColumn(Bytes.toBytes("attributes"), Bytes.toBytes("type"));
        get.addColumn(Bytes.toBytes("metrics"), Bytes.toBytes("open"));

        Delete delete =new Delete(Bytes.toBytes("2"));
        delete.addColumn(Bytes.toBytes("attributes"), Bytes.toBytes("type"));
        delete.addColumn(Bytes.toBytes("attributes"), Bytes.toBytes("for_user"));

        List<Row> batch = new ArrayList<Row>();
        batch.add(put);
        batch.add(get);
        batch.add(delete);

        Object[] results = new Object[batch.size()];

        try{
            table.batch(batch,results);
        }catch (Exception e){
            System.err.println("Error: "+e);}

        for (int i=0; i<results.length; i++){
    Result res = (Result) results[i];
    if(!res.isEmpty()){
        printAllValues((Result) results[i]);
    }
}   }
    }
}
```

A Put, a Get and a Delete

The batch method

```
List<Row> batch = new ArrayList<Row>();
batch.add(put);
batch.add(get);
batch.add(delete);
```

Create a list of Row objects

The batch method

```
List<Row> batch = new ArrayList<Row>();
batch.add(put);
batch.add(get);
batch.add(delete);
```

Add all 3 operations to the List

The batch method

```
Object[] results = new Object[batch.size()];

        try{
            table.batch(batch,results);
        }catch (Exception e){
            System.err.println("Error: "+e);}
```

Create an array to hold the results from each of the operations

The batch method

```
Object[] results = new Object[batch.size()];

        try{
            table.batch(batch,results);
        }catch (Exception e){
            System.err.println("Error: "+e);}
```

Pass the List of Row objects and the results array to the batch method

The batch method

```
Object[] results = new Object[batch.size()];

        try{
            table.batch(batch, results);
        }catch (Exception e){
            System.err.println("Error: "+e);}
```

The results from the batch operations will be stored here

The batch method

```
Object[] results = new Object[batch.size()];

        try{
            table.batch(batch, results);
        }catch (Exception e){
            System.err.println("Error: "+e);}
```

Get operations return a Result object

The batch method

```
Object[] results = new Object[batch.size()];

        try{
            table.batch(batch, results);
        }catch (Exception e){
            System.err.println("Error: "+e);}
```

Put and Delete operations return an empty Result object

The batch method

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

        Configuration conf = HBaseConfiguration.create();
        Connection connection = ConnectionFactory.createConnection(conf);
        Table table = connection.getTable("notifications");

        Put put =new Put(Bytes.toBytes("2"));
        put.addColumn(Bytes.toBytes("attributes"), Bytes.toBytes("type"), Bytes.toBytes("Comment"));
        put.addColumn(Bytes.toBytes("attributes"), Bytes.toBytes("for_user"), Bytes.toBytes("Swetha"));
        put.addColumn(Bytes.toBytes("metrics"), Bytes.toBytes("open"), Bytes.toBytes("0"));

        Get get =new Get(Bytes.toBytes("2"));
        get.addColumn(Bytes.toBytes("attributes"), Bytes.toBytes("type"));
        get.addColumn(Bytes.toBytes("metrics"), Bytes.toBytes("open"));

        Delete delete =new Delete(Bytes.toBytes("2"));
        delete.addColumn(Bytes.toBytes("attributes"), Bytes.toBytes("type"));
        delete.addColumn(Bytes.toBytes("attributes"), Bytes.toBytes("for_user"));

        List<Row> batch = new ArrayList<Row>();
        batch.add(put);
        batch.add(get);
        batch.add(delete);

        Object[] results = new Object[batch.size()];

        try{
            table.batch(batch,results);
        }catch (Exception e){
            System.err.println("Error: "+e);}
```

```
for (int i=0; i<results.length; i++){
    Result res = (Result) results[i];
    if(!res.isEmpty()){
        printAllValues((Result) results[i]);
    }
}
```

Iterate through the returned Result objects

The batch method

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

        Configuration conf = HBaseConfiguration.create();
        Connection connection = ConnectionFactory.createConnection(conf);
        Table table = connection.getTable("notifications");

        Put put =new Put(Bytes.toBytes("2"));
        put.addColumn(Bytes.toBytes("attributes"), Bytes.toBytes("type"), Bytes.toBytes("Comment"));
        put.addColumn(Bytes.toBytes("attributes"), Bytes.toBytes("for_user"), Bytes.toBytes("Swetha"));
        put.addColumn(Bytes.toBytes("metrics"), Bytes.toBytes("open"), Bytes.toBytes("0"));

        Get get =new Get(Bytes.toBytes("2"));
        get.addColumn(Bytes.toBytes("attributes"), Bytes.toBytes("type"));
        get.addColumn(Bytes.toBytes("metrics"), Bytes.toBytes("open"));

        Delete delete =new Delete(Bytes.toBytes("2"));
        delete.addColumn(Bytes.toBytes("attributes"), Bytes.toBytes("type"));
        delete.addColumn(Bytes.toBytes("attributes"), Bytes.toBytes("for_user"));

        List<Row> batch = new ArrayList<Row>();
        batch.add(put);
        batch.add(get);
        batch.add(delete);

        Object[] results = new Object[batch.size()];

        try{
            table.batch(batch,results);
        }catch (Exception e){
            System.err.println("Error: "+e);}
```

```
for (int i=0; i<results.length; i++){
    Result res = (Result) results[i];
    if(!res.isEmpty()){
        printAllValues((Result) results[i]);
    }
}
```

If the Result is not empty

The batch method

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

        Configuration conf = HBaseConfiguration.create();
        Connection connection = ConnectionFactory.createConnection(conf);
        Table table = connection.getTable("notifications");

        Put put =new Put(Bytes.toBytes("2"));
        put.addColumn(Bytes.toBytes("attributes"), Bytes.toBytes("type"), Bytes.toBytes("Comment"));
        put.addColumn(Bytes.toBytes("attributes"), Bytes.toBytes("for_user"), Bytes.toBytes("Swetha"));
        put.addColumn(Bytes.toBytes("metrics"), Bytes.toBytes("open"), Bytes.toBytes("0"));

        Get get =new Get(Bytes.toBytes("2"));
        get.addColumn(Bytes.toBytes("attributes"), Bytes.toBytes("type"));
        get.addColumn(Bytes.toBytes("metrics"), Bytes.toBytes("open"));

        Delete delete =new Delete(Bytes.toBytes("2"));
        delete.addColumn(Bytes.toBytes("attributes"), Bytes.toBytes("type"));
        delete.addColumn(Bytes.toBytes("attributes"), Bytes.toBytes("for_user"));

        List<Row> batch = new ArrayList<Row>();
        batch.add(put);
        batch.add(get);
        batch.add(delete);

        Object[] results = new Object[batch.size()];

        try{
            table.batch(batch,results);
        }catch (Exception e){
            System.err.println("Error: "+e);}
```

```
for (int i=0; i<results.length; i++){
    Result res = (Result) results[i];
    if(!res.isEmpty()){
        printAllValues((Result) results[i]);
    }
}
```

Print the values in the result

Example 16:

Retrieving a range of rows

Scan

Scan

With get operations we need to specify each row id separately

Scan

HBase tables are sorted maps, ie. row ids are sorted

With the scan operation, you can retrieve row ids within a specified range

Scan

```
scan 'notifications'
```

1. Specify which rows should be scanned

`Scan`

2. Connect to HBase and get a Table object

`Connection + HTable`

3. Fetch the result

`ResultScanner`

Scan

```
scan 'notifications'
```

```
public class scanRows {


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

        Configuration conf = HBaseConfiguration.create();

        Connection connection = ConnectionFactory.createConnection(conf);
        Table table = connection.getTable("notifications");


        Scan fullScan = new Scan();
        ResultScanner fullScanResult = table.getScanner(fullScan);
        for (Result res:fullScanResult){
             printAllValues(res);
        }
        fullScanResult.close();

        Scan colScan = new Scan();
        colScan.addFamily(Bytes.toBytes("metrics"));
        ResultScanner colScanResult = table.getScanner(colScan);
        for (Result res:colScanResult){
             printAllValues(res);
        }
        colScanResult.close();

        Scan rangeScan = new Scan();
        rangeScan.addColumn(Bytes.toBytes("attributes"), Bytes.toBytes("type"))
                .setStartRow(Bytes.toBytes("2"))
                .setStopRow(Bytes.toBytes("2"));


        ResultScanner rangeScanResult = table.getScanner(rangeScan);
        for (Result res:rangeScanResult){
             printAllValues(res);
        }
        rangeScanResult.close();


    }
    }
```

Let's look at a couple of variations of scan

1. Return all rows, columns

2. Return specific columns for all rows

3. Return specific columns for rows in a specified range

Scan

```
scan 'notifications'
```

```
public class scanRows {

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

        Configuration conf = HBaseConfiguration.create();

        Connection connection = ConnectionFactory.createConnection(conf);
        Table table = connection.getTable("notifications");

        Scan fullScan = new Scan();
        ResultScanner fullScanResult = table.getScanner(fullScan);
        for (Result res:fullScanResult){
             printAllValues(res);
        }
        fullScanResult.close();

        Scan colScan = new Scan();
        colScan.addFamily(Bytes.toBytes("metrics"));
        ResultScanner colScanResult = table.getScanner(colScan);
        for (Result res:colScanResult){
             printAllValues(res);
        }
        colScanResult.close();

        Scan rangeScan = new Scan();
        rangeScan.addColumn(Bytes.toBytes("attributes"), Bytes.toBytes("type"))
                .setStartRow(Bytes.toBytes("2"))
                .setStopRow(Bytes.toBytes("2"));


        ResultScanner rangeScanResult = table.getScanner(rangeScan);
        for (Result res:rangeScanResult){
             printAllValues(res);
        }
        rangeScanResult.close();


    }
    }
```

1. Return all rows, columns

2. Return specific columns for all rows

3. Return specific columns for rows in a specified range

Scan

```
scan 'notifications'
```

1. Return all rows, columns

```
Scan fullScan = new Scan();
ResultScanner fullScanResult = table.getScanner(fullScan);
 for (Result res:fullScanResult){
            printAllValues(res);
   }
 fullScanResult.close();
```

Scan

```
scan 'notifications'
```

1. Return all rows, columns

```
Scan fullScan = new Scan();
```

Set up a new Scan

```
public class ScanRows {

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

        Configuration conf = HBaseConfiguration.create();

        Connection connection = ConnectionFactory.createConnection(conf);
        Table table = connection.getTable("notifications");

        ResultScanner fullScanResult = table.getScanner(fullScan);
        for (Result res:fullScanResult){
             printAllValues(res);
        }
        fullScanResult.close();

        Scan colScan = new Scan();
        colScan.addFamily(Bytes.toBytes("metrics"));
        ResultScanner colScanResult = table.getScanner(colScan);
        for (Result res:colScanResult){
             printAllValues(res);
        }
        colScanResult.close();

        Scan rangeScan = new Scan();
        rangeScan.addColumn(Bytes.toBytes("attribute
                .setStartRow(Bytes.toBytes("2"))
                .setStopRow(Bytes.toBytes("2"));


        ResultScanner rangeScanResult = table.getScanner(rangeScan);
        for (Result res:rangeScanResult){
             printAllValues(res);
        }
        rangeScanResult.close();


    }
    }
```

Scan

```
scan 'notifications'
```

1. Return all rows, columns

```
Scan fullScan = new Scan();
```

This is where we would set up any properties such as specific columns , row range

Scan

```
scan 'notifications'
```

1. Return all rows, columns

```
Scan fullScan = new Scan();
```

By default, the Scan operation will return all the rows, columns

Scan

```
scan 'notifications'
```

1. Return all rows, columns

```
Scan fullScan = new Scan();
ResultScanner fullScanResult = table.getScanner(fullScan);
```

Use the getScanner method to get a ResultScanner

Scan

```
scan 'notifications'
```

1. Return all rows, columns

```
Scan fullScan = new Scan();
ResultScanner fullScanResult = table.getScanner(fullScan);
```

The ResultScanner is like a cursor for a database connection

Scan

```
scan 'notifications'
```

1. Return all rows, columns

```
for (Result res:fullScanResult){
        printAllValues(res);
    }
```

Iterate through the results returned by the ResultScanner and print the values

Scan

```
scan 'notifications'
```

1. Return all rows, columns

```
fullScanResult.close();
```

Once done close the ResultScanner just as you would a database cursor

Scan

```
scan 'notifications'
```

1. Return all rows, columns

```
Scan fullScan = new Scan();
ResultScanner fullScanResult = table.getScanner(fullScan);
 for (Result res:fullScanResult){
            printAllValues(res);
   }
 fullScanResult.close();
```

Scan

```
scan 'notifications'
```

1. Return all rows, columns

```
Scan colScan = new Scan();
colScan.addFamily(Bytes.toBytes("metrics"));
ResultScanner colScanResult = table.getScanner(colScan);
for (Result res:colScanResult){
    printAllValues(res);
     }
colScanResult.close();
```

2. Return specific columns for all rows

Scan

```
scan 'notifications'
```

2. Return specific columns for all rows

```
Scan colScan = new Scan();
colScan.addFamily(Bytes.toBytes("metrics"));
ResultScanner colScanResult = table.getScanner(colScan);
for (Result res:colScanResult){
        printAllValues(res);
    }
colScanResult.close();
```

This Scan is almost exactly the same

Scan

```
scan 'notifications'
```

2. Return specific columns for all rows

```
public class scanRows {

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

        Configuration conf = HBaseConfiguration.create();

        Connection connection = ConnectionFactory.createConnection(conf);
        Table table = connection.getTable("notifications");


        Scan fullScan = new Scan();
        ResultScanner fullScanResult = table.getScanner(fullScan);
        for (Result res:fullScanResult){
            printAllValues(res);
        }
        fullScanResult.close();

        Scan colScan = new Scan();
        colScan.addFamily(Bytes.toBytes("metrics"));
        ResultScanner colScanResult = table.getScanner(colScan);
        for (Result res:colScanResult){
            printAllValues(res);
        }
        colScanResult.close();

        Scan rangeScan = new Scan();
        rangeScan.addColumn(Bytes.toBytes("attributes"), Bytes.toBytes("type"))
                .setStartRow(Bytes.toBytes("2"))
                .setStopRow(Bytes.toBytes("2"));


        ResultScanner rangeScanResult = table.getScanner(rangeScan);
        for (Result res:rangeScanResult){
            printAllValues(res);
        }
        rangeScanResult.close();

    }
}
```

Just add the column family/ columns that you want to retrieve to the Scan

Scan

```
scan 'notifications'
```

2. Return specific columns for all rows

```
public class scanRows {

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

        Configuration conf = HBaseConfiguration.create();

        Connection connection = ConnectionFactory.createConnection(conf);
        Table table = connection.getTable("notifications");


        Scan fullScan = new Scan();
        ResultScanner fullScanResult = table.getScanner(fullScan);
        for (Result res:fullScanResult){
            printAllValues(res);
        }
        fullScanResult.close();

        Scan colScan = new Scan();
        colScan.addFamily(Bytes.toBytes("metrics"));
        ResultScanner colScanResult = table.getScanner(colScan);
        for (Result res:colScanResult){
                printAllValues(res);
            }
        colScanResult.close();

        Scan rangeScan = new Scan();
        rangeScan.addColumn(Bytes.toBytes("attributes"), Bytes.toBytes("type"))
                .setStartRow(Bytes.toBytes("2"))
                .setStopRow(Bytes.toBytes("2"));


        ResultScanner rangeScanResult = table.getScanner(rangeScan);
        for (Result res:rangeScanResult){
            printAllValues(res);
        }
        rangeScanResult.close();


    }
}
```

The rest is exactly as before

Scan

```
scan 'notifications'
```

1. Return all rows, columns

```
Scan colScan = new Scan();
colScan.addFamily(Bytes.toBytes("metrics"));
ResultScanner colScanResult = table.getScanner(colScan);
for (Result res:colScanResult){
    printAllValues(res);
     }
colScanResult.close();
```

2. Return specific columns for all rows

Scan

```
scan 'notifications'
```

3. Return specific columns for rows in a specified range

```
  Scan rangeScan = new Scan();
   rangeScan.addColumn(Bytes.toBytes("attributes"), Bytes.toBytes("type"))
                .setStartRow(Bytes.toBytes("2"))
                .setStopRow(Bytes.toBytes("3"));
ResultScanner rangeScanResult = table.getScanner(rangeScan);
for (Result res:rangeScanResult){
            printAllValues(res);
      }
rangeScanResult.close();
```

Scan

```
scan 'notifications'
```

```
Scan rangeScan = new Scan();
rangeScan.addColumn(Bytes.toBytes("attributes"), Bytes.toBytes("type"))
         .setStartRow(Bytes.toBytes("2"))
         .setStopRow(Bytes.toBytes("3"));
```

Set up a new Scan

Scan

```
scan 'notifications'
```

```
Scan rangeScan = new Scan();
 rangeScan.addColumn(Bytes.toBytes("attributes"), Bytes.toBytes("type"))
          .setStartRow(Bytes.toBytes("2"))
          .setStopRow(Bytes.toBytes("3"));
```

Add the column families and columns you want to retrieve data for

Scan

```
scan 'notifications'
```

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

    Configuration conf = HBaseConfiguration.create();

    Connection connection = ConnectionFactory.createConnection(conf);
    Table table = connection.getTable("notifications");


    Scan fullScan = new Scan();
    ResultScanner fullScanResult = table.getScanner(fullScan);
    for (Result res:fullScanResult){
        printAllValues(res);
    }
    fullScanResult.close();

    Scan colScan = new Scan();
    colScan.addFamily(Bytes.toBytes("metrics"));
    ResultScanner colScanResult = table.getScanner(colScan);
    for (Result res:colScanResult){
        printAllValues(res);
    }
    colScanResult.close();

    Scan rangeScan = new Scan();
     rangeScan.addColumn(Bytes.toBytes("attributes"), Bytes.toBytes("type"))
              .setStartRow(Bytes.toBytes("2"))
              .setStopRow(Bytes.toBytes("3"));

    ResultScanner rangeScanResult = table.getScanner(rangeScan);
    for (Result res:rangeScanResult){
        printAllValues(res);
    }
    rangeScanResult.close();


}
}
```

Set the start Row and stop Row

Scan

```
scan 'notifications'
```

Get the ResultScanner and print the results

```
Scan rangeScan = new Scan();
     rangeScan.addColumn(Bytes.toBytes("attributes"), Bytes.toBytes("type"))
              .setStartRow(Bytes.toBytes("2"))
              .setStopRow(Bytes.toBytes("3"));

     ResultScanner rangeScanResult = table.getScanner(rangeScan);
     for (Result res:rangeScanResult){
          printAllValues(res);
     }
     rangeScanResult.close();

}
```

Example 17:
Deleting a table

Let's delete the notifications table

We want to do the equivalent of

```
disable 'notifications'
```

```
drop 'notifications'
```

in Java

```
disable 'notifications'
```

```
drop 'notifications'
```

To delete a table you need to

1. Specify the table name

`HTableDescriptor`

2. Connect to HBase

`Connection`

3. Delete the table

`HBaseAdmin`

Delete a table

```
disable 'notifications'
```

```
drop 'notifications'
```

```
public class deleteTable {

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

        Configuration conf = HBaseConfiguration.create();

        Connection connection = ConnectionFactory.createConnection(conf);

        Admin admin = connection.getAdmin();

        HTableDescriptor tableName = new HTableDescriptor(TableName.valueOf("notifications"));

        if (admin.tableExists(tableName.getTableName())) {
            System.out.print("Table exists, Deleting.. ");
            admin.disableTable(tableName.getTableName());
            admin.deleteTable(tableName.getTableName());
            System.out.println(" Done.");
        }
    }
}
```

Delete a table

```
disable 'notifications'
```

```
drop 'notifications'
```

```
public class deleteTable {

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

        Configuration conf = HBaseConfiguration.create();
        Connection connection = ConnectionFactory.createConnection(conf);

        Admin admin = connection.getAdmin();

        HTableDescriptor tableName = new HTableDescriptor(TableName.valueOf("notifications"));

        if (admin.tableExists(tableName.getTableName())) {
            System.out.print("Table exists. Deleting...");
            admin.disableTable(tableName.getTableName());
            admin.deleteTable(tableName.getTableName());
            System.out.println(" Done.");
        }
    }
}
```

Get a connection to HBase

Delete a table

```
disable 'notifications'
```

```
drop 'notifications'
```

```
public class deleteTable {

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

          Configuration conf = HBaseConfiguration.create();

        Connection connection = ConnectionFactory.createConnection(conf);

        Admin admin = connection.getAdmin();

HTableDescriptor tableName = new
HTableDescriptor(TableName.valueOf("notifications"));

        if (admin.tableExists(tableName.getTableName())) {
            System.out.print("Table exists, Deleting.. ");
            admin.disableTable(tableName.getTableName());
            admin.deleteTable(tableName.getTableName());
            System.out.println(" Done.");
        }
    }
}
```

HTableDescriptor is used to specify the table name

Delete a table

```
disable 'notifications'
```

```
drop 'notifications'
```

```
public class deleteTable {

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

        Configuration conf = HBaseConfiguration.create();

        Connection connection = ConnectionFactory.createConnection(conf);

        Admin admin = connection.getAdmin();

        HTableDescriptor tableName = new HTableDescriptor(TableName.valueOf("notifications"));

        if (admin.tableExists(tableName.getTableName())) {
            System.out.print("Table exists, Deleting.. ");
            admin.disableTable(tableName.getTableName());
            admin.deleteTable(tableName.getTableName());
            System.out.println("Done!");
        }
    }
}
```

The Connection object can provide an instance of HBaseAdmin

Delete a table

```
disable 'notifications'
```

```
drop 'notifications'
```

We use the HBaseAdmin to

```
Admin admin = connection.getAdmin();
```

1. check if a table exists
2. disable the table
3. delete it

Delete a table

```
disable 'notifications'
```

```
drop 'notifications'
```

First check if the notifications table exists

```
public class deleteTable {

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

        Configuration conf = HBaseConfiguration.create();

        Connection connection = ConnectionFactory.createConnection(conf);

        Admin admin = connection.getAdmin();

        HTableDescriptor tableName = new HTableDescriptor(TableName.valueOf("notifications"));

        if (admin.tableExists(tableName.getTableName())) {
            System.out.print("Table exists, Deleting.. ");
            admin.disableTable(tableName.getTableName());
            admin.deleteTable(tableName.getTableName());
            System.out.println(" Done.");
        }
    }
}
```

Delete a table

```
disable 'notifications'
```

```
drop 'notifications'
```

If yes, disable and delete the table

```
public class deleteTable {
    public static void main(String[] args) throws IOException {
        Configuration conf = HBaseConfiguration.create();
        Connection connection = ConnectionFactory.createConnection(conf);
        Admin admin = connection.getAdmin();
        HTableDescriptor tableName = new HTableDescriptor(TableName.valueOf("notifications"));
        if (admin.tableExists(tableName.getTableName())) {
            System.out.print("Table exists, Deleting.. ");
            admin.disableTable(tableName.getTableName())
            admin.deleteTable(tableName.getTableName());
            System.out.println(" Done.");
        }
    }
}
```

Delete a table

```
disable 'notifications'
```

```
drop 'notifications'
```

Don't pass the HTableDescriptor directly

```
public class deleteTable {

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

        Configuration conf = HBaseConfiguration.create();

        Connection connection = ConnectionFactory.createConnection(conf);

        Admin admin = connection.getAdmin();

        HTableDescriptor tableName = new HTableDescriptor(TableName.valueOf("notifications"));

        if (admin.tableExists(tableName.getTableName())) {
            System.out.print("Table exists, Deleting.. ");
            admin.disableTable(tableName.getTableName());
            admin.deleteTable(tableName.getTableName());
            System.out.println(" Done.");
        }
    }
}
```

Delete a table

```
disable 'notifications'
```

```
drop 'notifications'
```

Pass the TableName object from HTableDescriptor

```
public class deleteTable {

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

        Configuration conf = HBaseConfiguration.create();

        Connection connection = ConnectionFactory.createConnection(conf);

        Admin admin = connection.getAdmin();

        HTableDescriptor tableName = new HTableDescriptor(TableName.valueOf("notifications"));

        if (admin.tableExists(tableName.getTableName())) {
                System.out.print("Table exists, Deleting.. ");
                admin.disableTable(tableName.getTableName());
                admin.deleteTable(tableName.getTableName());
                System.out.println(" Done.");
            }
    }
}
```