HBase Data Model and Architecture Lab Guide

Spring 2015

Get Started

Welcome to *HBase Data Model and Architecture*

This course is targeted towards data analysts, data architects and application developers. It serves as an introductory course for designing HBase schemas and also developing HBase applications. The focus of this course is to provide a deeper understanding of the HBase data model and architecture. It serves as a pre-requisite for HBase Schema Design, and also for Developing HBase Applications. Topics covered are:

- **Introduction to HBase**
  - Differentiate between RDBMS and HBase
  - Identify typical HBase Use Cases
- **HBase Data Model**
  - Describe the HBase Data model
  - List Data model components
  - Describe how logical data model maps physical storage on disk
  - Use data model operations
  - Create an HBase table
- **HBase Architecture**
  - Identify the components of an HBase cluster
  - Describe how the HBase components work together
  - Describe how regions work and their benefits
  - Define the function of minor and major compactions
  - Describe RegionServer splits
  - Describe how HBase handles fault tolerance
  - Differentiate MapRDB from HBase

Prerequisites

- A laptop (Linux, PC or Mac) to install the MapR Sandbox or connect to a Hadoop cluster via SSH and web browser

How to Use this Guide

Typographical Conventions

`Courier`	Things you type as shown
`Courier`	Things as seen on screen
italics	Things you replace with a value
Durations	Suggested time it takes to complete an exercise. Some are required, other are optional

Lab Exercise Timing

Each Lesson's Labs are listed with estimated time to completion.

Exercise	Required	Duration
0.1: Install needed Software	Yes	Before class
0.2. Connect to the MapR Virtual Machine (sandbox) or Cluster	Yes	15 min

Tools Needed to Complete Labs

In order to complete the labs you should have already installed these components:

1. **Install SSH and SCP clients**
2. **VMware player / VirtualBox (prerequisite for MapR Sandbox)**
3. **MapR Sandbox (https://www.mapr.com/products/mapr-sandbox-hadoop/download)**

For detailed instructions, visit:

http://doc.mapr.com/display/MapR/MapR+Sandbox+for+Hadoop

Connect to a MapR Cluster (Sandbox or AWS)

The purpose of this lab is for you to establish connectivity to your MapR Sandbox or a cluster environment that you will use for the hands-on exercises in this course.

Logging into the Sandbox

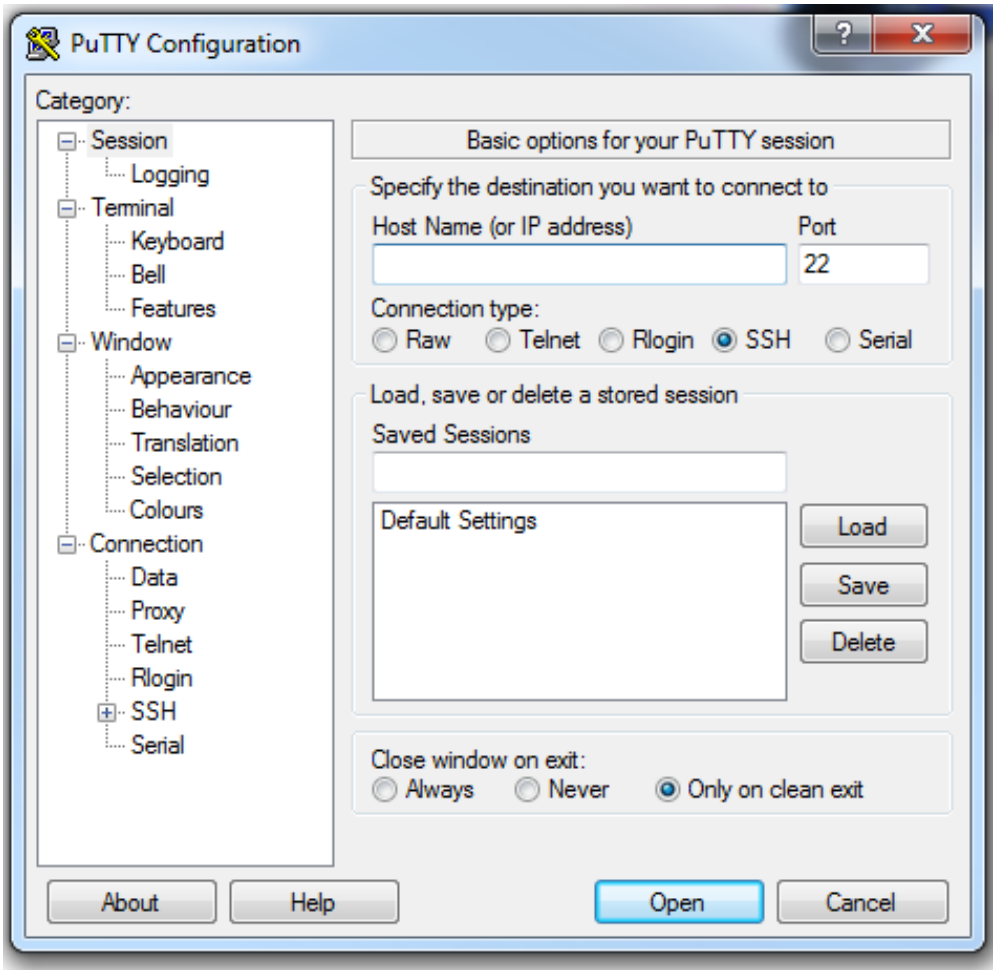
This procedure assumes that you have downloaded and installed the MapR Sandbox as part of the class prerequisites. If not, please do that before continuing here. Make a note of the IP address assigned to the Sandbox, as you will need it to log in remotely.

The Sandbox is preconfigured with several user accounts. You will be logging in as user01 to complete most of the lab exercises. The password for user01 on the sandbox is mapr, on the cluster the password is mapr.

You will need to use a Secure Shell (ssh) client to log into the Sandbox. PuTTY, Cygwin, or a bash shell on a Unix-like OS with OpenSSH installed are all acceptable clients. If you do not have an ssh client, you can log in to the Sandbox using the virtual machine console mode. Connect to your Sandbox as user01.

OPTION 1: Log in using PuTTY

1. Start your PuTTY Client.



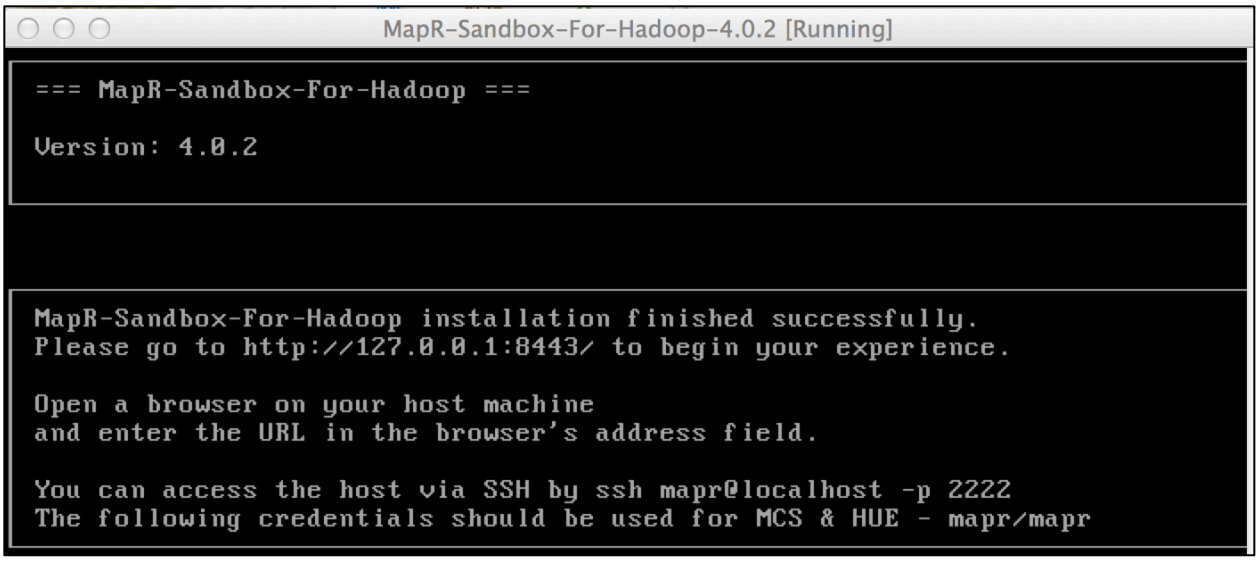
2. In the Host Name (or IP address) box, enter the host name or IP address for your Sandbox virtual machine.
3. Click the Open button to open the connection to your node.
4. When the terminal window appears, log into the terminal as user01.

OPTION 2: Log in using terminal SSH client

1. Start your terminal client. Use the ssh program to connect to the hostname or IP address for your Sandbox. Connect as user01.

   ```
   $ ssh user01@IP address
   ```

(In the example, below, you can connect using `ssh user01@localhost –p 2222`).



2. Copying files to the Sandbox or AWS Cluster
3. Use scp to copy files from your laptop to a Sandbox or AWS cluster. The syntax for the scp command is:

   scp [options] username1@source_host:directory1/filename1 username2@destination_host:directory2/filename2

4. On windows you can use

   http://winscp.net/eng/download.php

Lab 2: Perform CRUD operations using HBase shell

Lab Overview

The objective of this lab is to get you started with the HBase shell and perform CRUD operations to create a HBase Table, put data into the table, retrieve data from the table and delete data from the table. This lab also gives a brief introduction into MapR Control System (MCS) and we'll see how to create a HBase (MapR-DB) Table and add ColumnFamilies using MCS.

Exercise	Required	Duration
2.1: Perform CRUD operations with HBase shell	Yes	20 min
2.2: Create a MapR-DB Table using MapR Control System (MCS)	Yes	20 min

Background information on HBase Shell

HBase is the Hadoop database, which provides random, realtime read/write access to very large data. See the references on HBase for more information.

The HBase Shell is a ruby script that helps in interacting with the HBase system using a command line interface. This shell supports creating, deleting and altering tables and also performing other operations like inserting, listing, deleting data and to interact with HBase. You can get help with the shell commands here:

https://learnhbase.wordpress.com/2013/03/02/hbase-shell-commands/

Here are some reference notes on using the HBase shell:

To start the shell at the linux command line type:

$ hbase shell

```
// to get help on commands

hbase> help

// use create to create a new table

hbase> create '/path/tablename', {NAME =>'cfname'}, {NAME =>'cfname'}

//Use 'put' to insert data into the table

hbase> put '/path/tablename', 'rowkey', 'cfname:colname', 'value'

//To get row data with rowkey

hbase> get '/path/tablename', 'rowkey'

hbase> scan '/path/tablename'

hbase> describe '/path/mytable'
```


Exercise 2.1: Perform CRUD operations with HBase shell

The goal of this exercise is to create a table 'customer' with the data for Column Families address, order and columns city, state for as shown below using HBase shell commands.

Table: customer

Row-key userid	address		order	
	city	state	date	number
jsmith	nashville	tn	01/01/2015	12345
bjones	miami	fl	02/02/2015	56565

Use HBase shell

```
$ hbase shell
```

Once connected to the cluster and having started the HBase shell, explore the commands that you can perform: like 'help', 'help "put"' etc…

Note: if you are using a cluster, change user01 to your user, a text file with all of the commands is provided for your convenience in the LAB FILES directory.

Create an HBase table

1. Create a table in your home directory

```
create '/user/user01/customer', {NAME=>'addr'}, {NAME=>'order'}
```

2. Use 'describe' to get the description of the table.

```
describe    '/user/user01/customer'
```

Insert data in the table

1. Put some data into the table

```
put  '/user/user01/customer',  'jsmith',  'addr:city', 'nashville'
```


2. Put more data into the table

```
put   '/user/user01/customer',  'jsmith',  'addr:state', 'TN'

put   '/user/user01/customer',  'jsmith',  'order:numb', '1234'

put   '/user/user01/customer',  'jsmith',  'order:date', '10-18-2014'
```

Retrieve data from table

3. Use 'get' to retrieve the data for 'jsmith'

```
get   '/user/user01/customer', 'jsmith'
```

Note that this gets all the data for the row. How can we limit this to only one column family ?

```
get   '/user/user01/customer', 'jsmith', {COLUMNS=>['addr']}
```

How can we limit this to a specific column?

```
get   '/user/user01/customer', 'jsmith', {COLUMNS=>['order:numb']}
```

Alter table to store more versions in the order column family

```
alter   '/user/user01/customer' , NAME => 'order', VERSIONS => 5
```

Now you are going to execute some more commands to describe the table, add more data to the table, retrieve data form the table.

4. Use 'describe' to get the description of the table.

```
describe   '/user/user01/customer'
```

5. Put more order numbers

```
put   '/user/user01/customer',  'jsmith',  'order:numb', '1235'

put   '/user/user01/customer',  'jsmith',  'order:numb', '1236'

put   '/user/user01/customer',  'jsmith',  'order:numb', '1237'

put   '/user/user01/customer',  'jsmith',  'order:numb', '1238'
```

6. Get order number column cells

```
get   '/user/user01/customer', 'jsmith', {COLUMNS=>['order:numb']}
```

Note that you are getting the data for only one version per cell. How can you get more versions?

```
get  '/user/user01/customer', 'jsmith', {COLUMNS=>['order:numb'], VERSIONS =>
5}
```

7. Put more data for different rowkey userids

```
put   '/user/user01/customer',  'njones',  'addr:city', 'miami'

put   '/user/user01/customer',  'njones',  'addr:state', 'FL'

put   '/user/user01/customer',  'njones',  'order:numb', '5555'

put   '/user/user01/customer',  'tsimmons',  'addr:city', 'dallas'

put   '/user/user01/customer',  'tsimmons',  'addr:state', 'TX'

put   '/user/user01/customer',  'jsmith',  'addr:city', 'denver'

put   '/user/user01/customer',  'jsmith',  'addr:state', 'CO'

put   '/user/user01/customer',  'jsmith',  'order:numb', '6666'

put   '/user/user01/customer',  'njones',  'addr:state', 'TX'

put   '/user/user01/customer',  'amiller', 'addr:state', 'TX'
```

Use 'scan' to retrieve rows of data for the table

8. Retrieve all rows , all columns

```
scan   '/user/user01/customer'
```

9. Retrieve all rows , addr column family

```
scan   '/user/user01/customer',  {COLUMNS=>['addr']}
```

10. Retrieve all rows for order number column, 5 versions

```
scan   '/user/user01/customer',  {COLUMNS=>['order:numb'], VERSIONS => 5}
```

11. Retrieve rows with rowkey starting with 'njo', addr column family

```
scan   '/user/user01/customer',  {STARTROW => 'njo', COLUMNS=>['addr'] }
```

12. Retrieve rows with rowkey starting with 'j', stop before 't'

```
scan   '/user/user01/customer',  { STARTROW => 'j', STOPROW => 't'}
```

13. Retrieve rows with rowkey starting with 'a'

```
scan   '/user/user01/customer',  { STARTROW => 'a'}
```


14. Retrieve rows with rowkey starting with 'a'

```
scan   '/user/user01/customer',  { STARTROW => 't'}
```

Use 'count' to retrieve the number of rows in the table.

```
count   '/user/user01/customer'
```

Delete data from the table.

First retrieve data to see what is there. We will then delete data.

```
get    '/user/user01/customer',  'njones'
```

15. Delete a column

```
delete    '/user/user01/customer',  'njones',  'addr:city'
```

16. Delete a column family

```
delete    '/user/user01/customer',  'jsmith',   'addr:'

get    '/user/user01/customer',  'jsmith'
```

17. Delete a row

```
delete    '/user/user01/customer',  'jsmith'
```

Bonus Activity

1. Create a table and pre-split into 4 regions and then import data.

   create '/user/user01/mynewtable', 'CF', {SPLITS => ['F', 'L', 'S']}

2. See the description of the table: describe '/user/user01/mynewtable'

Exercise 2.2: Create a MapR-DB Table using MapR Control System (MCS)

The goal of this exercise is to get familiar with MCS and use it to perform Table creation and changing the properties for the column families.

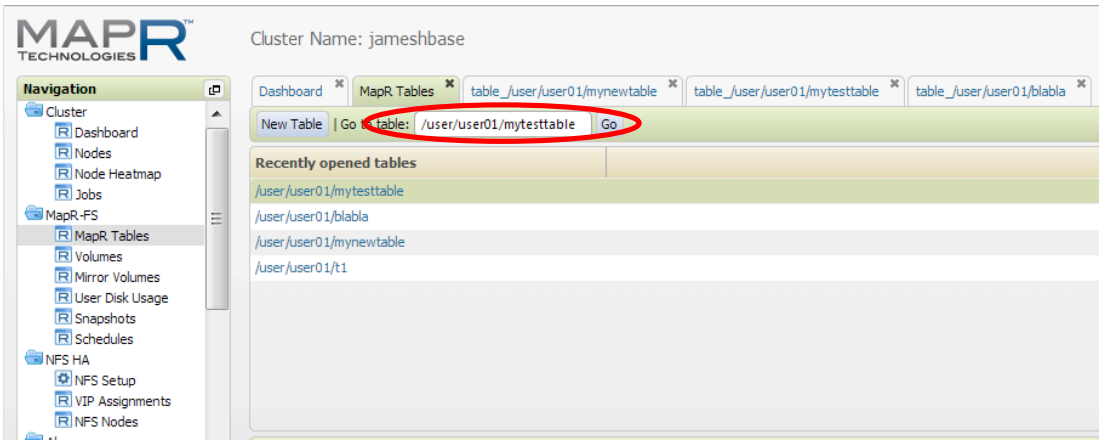
Connect to MapR Control System and see Table Properties

1. Connect to MapR cluster using MCS from a bowser using the notes from the instructor for host information. Login with your account username and password given to you in the class.

   `https://ipaddress:8443`

2. Connect to the table '/user/user01/customer' and see how many rows, regions are there for this table, Also note the properties for the column families.

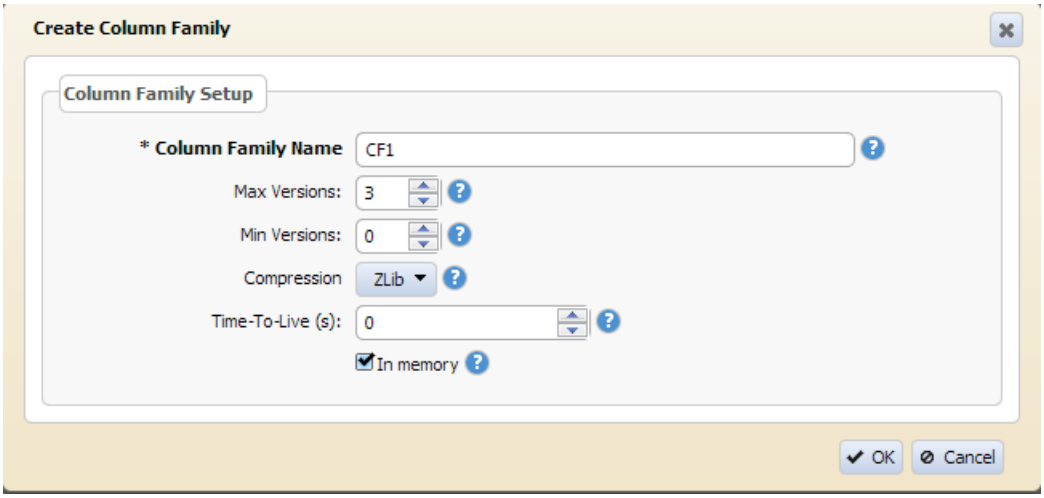
   a. Click on MapR-Fs, MapR Tables on the left, the in the Go To Table text box enter '/user/user01/customer' as shown in the image



Create a table using MCS

1. Create a table called 'MCSNewTable'.

2. Create column families similar to 'mytesttable'. Change the Max versions, Compression, TTL.



3. List the new table you created with pre-split and note the number of regions created.

4. You can edit , delete tables without disabling them with MCS

Delete a Table with the shell

To Delete a table with the hbase shell

To delete a table, first disable it and then delete it.

```
disable    '/user/user01/customer'
drop       '/user/user01/customer'
```


HBase Schema Design Lab Guide

Spring 2015

Lab 4: Basic Schema Design

Lab Overview

The objective of this lab is to load data in to HBase tables with different row keys and schema and observe the results. You will use a bulk import tool to load the data into HBase tables.

Exercise	Required	Duration
4.1: import data with different row key designs	Yes	40 min
4.2: Create, load, query Stock Trades Tall and Wide tables	Yes	15 min

Review of useful tools for lab

```
Accessing MCS:
https:// hostipaddress:8443
user: user01 ...
password:    mapr
```

Use putty on Windows

```
Login into remote host via putty  [change userxx to
user01 ...]:
userxx@hostipaddress
password: mapr
```

Use ssh on Mac

```
ssh –i user01@hostipaddress
```

Use scp to copy files from your laptop to a sandbox or aws cluster

```
scp [options] username1@source_host:directory1/filename1
username2@destination_host:directory2/filename2
```

On windows you can use

```
http://winscp.net/eng/download.php
```

Use HBase shell

```
$ hbase shell
```

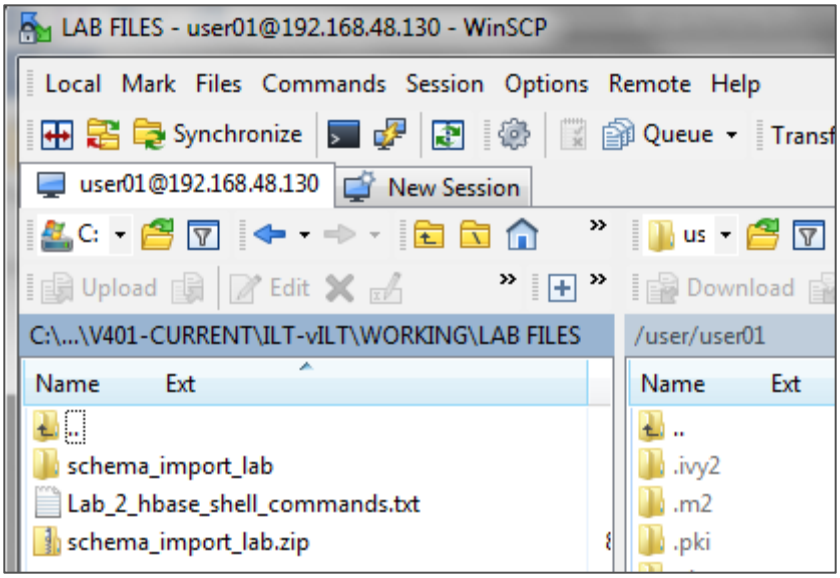
Lab 4.1: Import data with different row key designs

The goal of this exercise is to import data into an HBase table with different schemas and observe the results.

Exercise 4.1a: Import data with date as row key

1. Copy lab files to your sandbox or AWS cluster

Use scp to copy the **schema_import_lab.zip** from the local directory where you put your lab files to your sandbox or AWS cluster home folder.



2. After you have copied the zip file , use the unzip command at the linux command line to unzip it.

```
$ unzip schema_import_lab.zip
```

We are going to use a modified version of the HBase utility, ImportTsv, to import a sample airline data set CSV (comma separated) file into a table.

3. Look at the sample data file

Use tail at the linux command line to take a look at the data in the ontime.csv file :

```
$ tail ontime.csv

2014,1,1,31,5,2014-01-31,WN,N7704B,228,TUS,LAS,
```

```
1946,46.00,1958,43.00,0.00,,75.00,60.00,365.00,11.00,0.00,0.00,0.
00,32.00,
```

The dataset was gathered from here https://explore.data.gov/download/ar4r-an9z/DATAEXTRACTION and contains airline ontime information for the month of January 2014 .

4. In your home directory, make all shell scripts executable. Create two sub-directories - **data** and **tables**. Move the **ontime.csv** file to the **data** subdirectory:

```
$ chmod +x *.sh
$ mkdir data
$ mkdir tables
$ mv ontime.csv data/.
```

5. Now use more, or cat at the command line to take a look at the first script:

```
$ more import1.sh
```

The script defines the variable for the userid, ME=user01, change this if you are not using user01

```
ME="user01"
```

The script uses the mapr command line to create a MapR-DB table named /user/user01/tables/airline with one column family cf1

```
TABLE="/user/$ME/tables/airline"
FILE="ontime.csv"
CF="cf1"
```

The script uses the importtsv tool to import the data.

- The import tool takes the following parameters : -Dimporttsv.columns=a,b,c <tablename> <inputfile>
- The column names for the CSV data are specified using the -Dimporttsv.columns option. This option takes the form of comma-separated column names, where each column name is a columnfamily:qualifier.
- The special column name HBASE_ROW_KEY is used to designate that this column should be used as the row key for each imported record. With the **standard** importtsv tool you can only specify one column to be the row key.

Here is how the comma separated data maps to columns and a row key in import1.sh.
-Dimporttsv.columns=
$CF:year,$CF:qtr,$CF:month,$CF:dom,$CF:dow,**HBASE_ROW_KEY**,$CF:carrier,$CF:tailnum,
$CF:flightnumber,$CF:origin,$CF:dest,$CF:deptime,$CF:depdelay,$CF:arrtime,$CF:arrdelay,
$CF:cncl,$CF:cnclcode,$CF:elaptime,$CF:airtime,$CF:distance,$CF:carrierdelay,$CF:weatherdelay,$CF:nasdelay,$CF:securitydelay,$CF:aircraftdelay,$CF:dummy

Here is a row of data:

2014,1,1,31,5,**2014-01-31**,WN,N7704B,
228,TUS,LAS,1946,46.00,1958,43.00,
0.00,,75.00,60.00,365.00,11.00,0.00,
0.00,0.00,32.00,

Note that the row key maps to the flight date: HBASE_ROW_KEY = **2014-01-31** = flight date

6. Run the script to import data:

```
$ ./import1.sh
```

The importtsv tool runs a mapreduce job which reads from the csv file and inserts rows into HBase (or MapR-DB). After running this script you have the following table schema with data (not all columns and rows are shown)

Table: airline

Row-key date	cf1										
	year	Qtr	month	dom	dow	carrier	tailnum	Flight number	origin	dest	…
2014-01-31	2014	1	1	31	5	WN	N7704B	228	TUS	LAX	…

Note that the row key has been chosen to be the field in the airline data csv file which has the date. This file is for flight schedule data for the month of January 2014.

Do you see a problem with this row key?

Yes – it could lead to hotspotting.

7. Launch the HBase shell and use the count command to count the number of rows in the table :

```
$hbase shell
hbase(main):003:0> count '/user/user01/tables/airline'
31 row(s) in 0.0560 seconds
```

You see that the table only has 31 rows, even though the file had 471,949 records of flight information. Why?

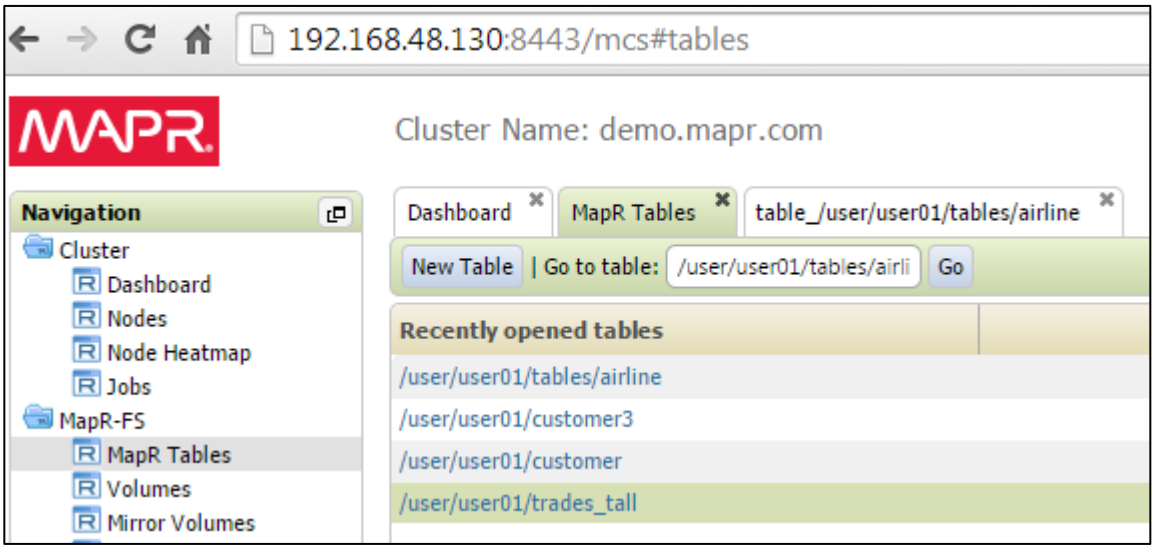
Based on how we set up our import, we set our row key to be the date. There are only 31 dates in this file for the month of January. Column values with the same date were inserted as a new

version because the rowkey is not unique. In the following exercises we will use our modified importtsv to select more than one field from the dataset to create composite row keys.

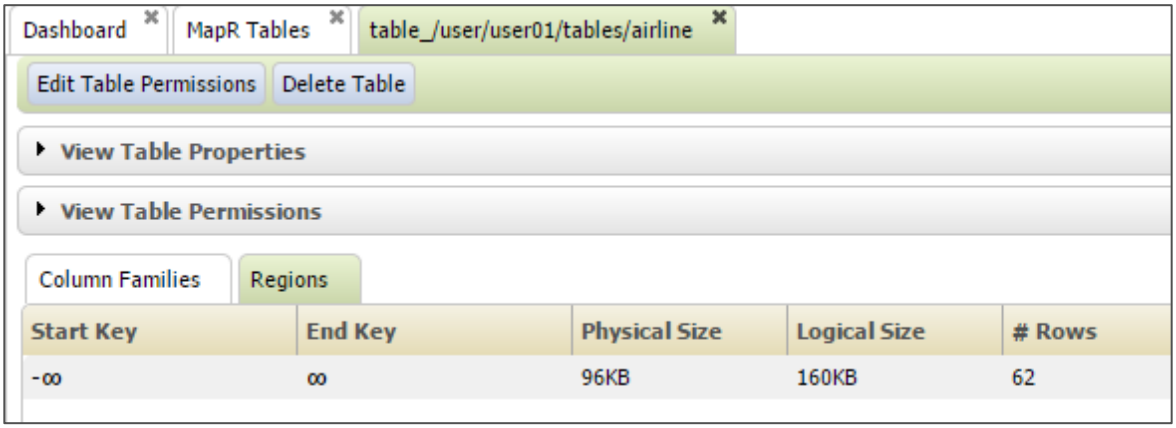
Note: This is the problem with the standard ImportTsv tool as it stands today. You cannot easily create composite keys because it only allows you to select one field for the row key.

Using MapR Control Server (MCS) to view tables

1. In MCS, you can pull up the table /user/user01/tables/airline you just loaded. Under **Navigation**, expand **MapR-FS**. Click **MapR Tables** and enter the full table name in the G**o to table** text box. Click **Go**.



2. Click the table hyperlink . You will see that it has 62 rows. This is because the row count in MCS is an *estimate* of the row count and includes deleted versions.



Queries using HBase Shell

Try out some queries on the table using the HBase shell (you can find hbase shell query info here https://learnhbase.wordpress.com/2013/03/02/hbase-shell-commands/)

1. Scan one row using the HBase shell

```
> scan '/user/user01/tables/airline', {LIMIT => 1}
```

Note that the output is in Key Value format with the complete cell coordinates for each value

```
ROW                COLUMN+CELL
2014-01-01         column=cf1:aircraftdelay, timestamp=1424960635661, value=0.00
2014-01-01         column=cf1:airtime, timestamp=1424960635661, value=201.00
2014-01-01         column=cf1:arrdelay, timestamp=1424960635661, value=26.00
2014-01-01         column=cf1:arrtime, timestamp=1424960635661, value=1311
2014-01-01         column=cf1:carrier, timestamp=1424960635661, value=WN
2014-01-01         column=cf1:carrierdelay, timestamp=1424960635661, value=0.00
2014-01-01         column=cf1:cncl, timestamp=1424960635661, value=0.00
2014-01-01         column=cf1:cnclcode, timestamp=1424960635661, value=
2014-01-01         column=cf1:depdelay, timestamp=1424960635661, value=1.00
2014-01-01         column=cf1:deptime, timestamp=1424960635661, value=0931
2014-01-01         column=cf1:dest, timestamp=1424960635661, value=MCO
2014-01-01         column=cf1:distance, timestamp=1424960635661, value=1142.00
2014-01-01         column=cf1:dom, timestamp=1424960635661, value=1
2014-01-01         column=cf1:dow, timestamp=1424960635661, value=3
2014-01-01         column=cf1:dummy, timestamp=1424960635661, value=
2014-01-01         column=cf1:elaptime, timestamp=1424960635661, value=195.00
2014-01-01         column=cf1:flightnumber, timestamp=1424960635661, value=1147
2014-01-01         column=cf1:month, timestamp=1424960635661, value=1
2014-01-01         column=cf1:nasdelay, timestamp=1424960635661, value=26.00
2014-01-01         column=cf1:origin, timestamp=1424960635661, value=MHT
2014-01-01         column=cf1:qtr, timestamp=1424960635661, value=1
2014-01-01         column=cf1:securitydelay, timestamp=1424960635661, value=0.00
2014-01-01         column=cf1:tailnum, timestamp=1424960635661, value=N264LV
2014-01-01         column=cf1:weatherdelay, timestamp=1424960635661, value=0.00
2014-01-01         column=cf1:year, timestamp=1424960635661, value=2014
1 row(s) in 0.1930 seconds
```

2. Try some other Scans using the HBase shell

```
scan '/user/user01/tables/airline'

scan  '/user/user01/tables/airline',  { STARTROW => '2014'}

scan  '/user/user01/tables/airline',  { STARTROW => '2014-01-20',
STOPROW => '2014-01-21'}
```

Exercise 4.1b: Import data with composite row key

With the **standard** importtsv tool you can only specify one column to be the row key, we are using a modified importtsv tool, which supports specifying multiple columns from the tsv file for as the row key. Now we will import the same data with a composite row key.

1. View and then execute the import2.sh script

```
$  more import2.sh
$ ./import2.sh
```

Note that in this import script, we specify five different fields as our composite row key:

-Dimporttsv.columns=$CF:year,$CF:qtr,$CF:month,$CF:dom,$CF:dow,
HBASE_ROW_KEY_1,HBASE_ROW_KEY_2,$CF:tailnum,**HBASE_ROW_KEY_3,**
HBASE_ROW_KEY_4,HBASE_ROW_KEY_5,$CF:deptime,$CF:depdelay,$CF:arrtime,$CF:arrdelay,$CF:cncl
,$CF:cnclcode,$CF:elaptime,$CF:airtime,$CF:distance,$CF:carrierdelay,$CF:weatherdelay,$CF:nasdelay,$
CF:securitydelay,$CF:aircraftdely

HBASE_ROW_KEY_x
1 = flight date
2 = carrier
3 = flight number
4 = origin
5 = destination

Table: airline

Row-key Date-carrier-Flightnumber-Origin-destination	**cf1**									
	year	qtr	month	dom	dow	tailnum	Deptime	depdelay	arrtime	…
2014-01-01-AA-1003-MIA-PHL	2014	1	1	31	5	N7704B	0914	14.00	1238	…

2. Scan the table using the HBase shell

```
> scan '/user/user01/tables/airline', {LIMIT => 1}
```

Note that the output is in Key Value format with the complete cell coordinates for each value

ROW	**COLUMN+CELL**
2014-01-01-AA-1-JFK-LAX	column=cf1:aircraftdelay, timestamp=1424962418798, value=
2014-01-01-AA-1-JFK-LAX	column=cf1:airtime, timestamp=1424962418798, value=359.00
2014-01-01-AA-1-JFK-LAX	column=cf1:arrdelay, timestamp=1424962418798, value=13.00
2014-01-01-AA-1-JFK-LAX	column=cf1:arrtime, timestamp=1424962418798, value=1238
2014-01-01-AA-1-JFK-LAX	column=cf1:carrierdelay, timestamp=1424962418798, value=
2014-01-01-AA-1-JFK-LAX	column=cf1:cncl, timestamp=1424962418798, value=0.00

```
2014-01-01-AA-1-JFK-LAX    column=cf1:cnclcode, timestamp=1424962418798, value=
2014-01-01-AA-1-JFK-LAX    column=cf1:depdelay, timestamp=1424962418798, value=14.00
2014-01-01-AA-1-JFK-LAX    column=cf1:deptime, timestamp=1424962418798, value=0914
2014-01-01-AA-1-JFK-LAX    column=cf1:distance, timestamp=1424962418798, value=2475.00
2014-01-01-AA-1-JFK-LAX    column=cf1:dom, timestamp=1424962418798, value=1
2014-01-01-AA-1-JFK-LAX    column=cf1:dow, timestamp=1424962418798, value=3
2014-01-01-AA-1-JFK-LAX    column=cf1:dummy, timestamp=1424962418798, value=
2014-01-01-AA-1-JFK-LAX    column=cf1:elaptime, timestamp=1424962418798, value=385.00
2014-01-01-AA-1-JFK-LAX    column=cf1:month, timestamp=1424962418798, value=1
2014-01-01-AA-1-JFK-LAX    column=cf1:nasdelay, timestamp=1424962418798, value=
2014-01-01-AA-1-JFK-LAX    column=cf1:qtr, timestamp=1424962418798, value=1
2014-01-01-AA-1-JFK-LAX    column=cf1:securitydelay, timestamp=1424962418798, value=
2014-01-01-AA-1-JFK-LAX    column=cf1:tailnum, timestamp=1424962418798, value=N338AA
2014-01-01-AA-1-JFK-LAX    column=cf1:weatherdelay, timestamp=1424962418798, value=
2014-01-01-AA-1-JFK-LAX    column=cf1:year, timestamp=1424962418798, value=2014
1 row(s) in 0.1400 seconds
```

3. Try out some queries on the table using the HBase shell

```
count '/user/user01/tables/airline'

scan  '/user/user01/tables/airline',  { STARTROW => '2014'}

scan  '/user/user01/tables/airline',  { STARTROW => '2014-01-20',
STOPROW => '2014-01-21'}

scan  '/user/user01/tables/airline',  { STARTROW => '2014-01-16-
AA',  STOPROW => '2014-01-16-AB'}
```

Using this row key what data can you retrieve?

You can use get to retrieve a specific row i.e. a particular flight for a particular carrier with on a specific date with the origin and destination information.

You can do scans with partial row keys – for example, all flights for a particular carrier on a particular date; flights with the same origin on a particular date, etc.

Think about the queries that would be useful.

Would you query by carrier or destination?

Remember for scanning you should group the data that you want to retrieve together with the most scanned information on the left of the key.

Exercise 4.1c: Import data with a better composite row key

Now we will import the same data with a different composite row key.

1. View and then execute the import3.sh script

```
$  more import3.sh
$ ./import3.sh
```

Note that in this import script, we specify five different fields as our composite rowkey:

-Dimporttsv.columns=$CF:year,$CF:qtr,$CF:month,$CF:dom,$CF:dow,**HBASE_ROW_KEY_3, HBASE_ROW_KEY_1**,$CF:tailnum**,HBASE_ROW_KEY_2,HBASE_ROW_KEY_4,HBASE_ROW_KEY_5,$**CF:deptime,$CF:depdelay,$CF:arrtime,$CF:arrdelay,$CF:cncl,$CF:cnclcode,$CF:elaptime,$CF:airtime,$CF:distance,$CF:carrierdelay,$CF:weatherdelay,$CF:nasdelay,$CF:securitydelay,$CF:aircraftdelay,$CF:dummy

HBASE_ROW_KEY_x
1 = carrier
2 = flight number
3 = flight date
4 = origin
5 = destination
AA-1-2014-01-01-JFK-LAX

Table: airline

Row-key carrier flightnumber date origin destination	**cf1**									
	year	qtr	month	dom	dow	tailnum	Deptime	depdelay	arrtime	…
AA-1-2014-01-01-JFK-LAX	2014	1	1	31	5	N7704B	0914	14.00	1238	…

ROW	**COLUMN+CELL**
AA-1-2014-01-01-JFK-LAX	column=cf1:aircraftdelay, timestamp=1424930493341, value=
AA-1-2014-01-01-JFK-LAX	column=cf1:airtime, timestamp=1424930493341, value=359.00
AA-1-2014-01-01-JFK-LAX	column=cf1:arrdelay, timestamp=1424930493341, value=13.00
AA-1-2014-01-01-JFK-LAX	column=cf1:arrtime, timestamp=1424930493341, value=1238
AA-1-2014-01-01-JFK-LAX	column=cf1:carrierdelay, timestamp=1424930493341, value=
AA-1-2014-01-01-JFK-LAX	column=cf1:cncl, timestamp=1424930493341, value=0.00
AA-1-2014-01-01-JFK-LAX	column=cf1:cnclcode, timestamp=1424930493341, value=
AA-1-2014-01-01-JFK-LAX	column=cf1:depdelay, timestamp=1424930493341, value=14.00
AA-1-2014-01-01-JFK-LAX	column=cf1:deptime, timestamp=1424930493341, value=0914
AA-1-2014-01-01-JFK-LAX	column=cf1:distance, timestamp=1424930493341, value=2475.00
AA-1-2014-01-01-JFK-LAX	column=cf1:dom, timestamp=1424930493341, value=1
AA-1-2014-01-01-JFK-LAX	column=cf1:dow, timestamp=1424930493341, value=3
AA-1-2014-01-01-JFK-LAX	column=cf1:dummy, timestamp=1424930493341, value=
AA-1-2014-01-01-JFK-LAX	column=cf1:elaptime, timestamp=1424930493341, value=385.00

```
AA-1-2014-01-01-JFK-LAX     column=cf1:month, timestamp=1424930493341, value=1
AA-1-2014-01-01-JFK-LAX     column=cf1:nasdelay, timestamp=1424930493341, value=
AA-1-2014-01-01-JFK-LAX     column=cf1:qtr, timestamp=1424930493341, value=1
AA-1-2014-01-01-JFK-LAX     column=cf1:securitydelay, timestamp=1424930493341, value=
AA-1-2014-01-01-JFK-LAX     column=cf1:tailnum, timestamp=1424930493341, value=N338AA
AA-1-2014-01-01-JFK-LAX     column=cf1:weatherdelay, timestamp=1424930493341, value=
AA-1-2014-01-01-JFK-LAX     column=cf1:year, timestamp=1424930493341, value=2014
  1 row(s)
```

2. Using the HBase shell scan one row of data .

```
scan '/user/user01/tables/airline', {LIMIT => 1}
```

Note that the output is in Key Value format with the complete cell coordinates for each value.This composite key, with Carrier as the first field, will be better for queries that scan or filter by Carrier.

3. With the HBase shell you can also scan with filters. This allows you to narrow down your scans. Here are some examples of filters. Try them out and try changing them :

Scan to get any cell value = 239.00	`scan '/user/user01/tables/airline', FILTER=>"ValueFilter(=,'binary:239.00')"`
Scan to get any cancellation codes = 1	`scan '/user/user01/tables/airline', {FILTER => "SingleColumnValueFilter('cf1','cncl', =,'binary:1.00') "}`
Scan to get any column name with a prefix of cnc	`scan '/user/user01/tables/airline', FILTER=>"ColumnPrefixFilter('cnc')"`
Scan with a start and stop row and a prefix filter on the column name	`scan '/user/user01/tables/airline', { STARTROW => 'AA-1-2014-01-01-JFK-LAX', STOPROW => 'AA-10', FILTER=>"ColumnPrefixFilter('cnc')"}`
Perform an HBase scan to find flights with a delay > 200.0	`scan '/user/user01/tables/airline', {COLUMNS => ['cf1:carrierdelay'],FILTER =>" (SingleColumnValueFilter('cf1','carrierdelay',=,'regexstring:^[2-9]{3}') "}` You should see output with lines as shown: WN-999-2014-01-30-BNA-MCO column=cf1:carrierdelay, timestamp=1424930493341, value=9.00

4. You can also pass a script file to the hbase shell. Exit the shell and look at the file scan4, then Scan to find flights with AA and JFK in the row key

```
$ cat scan4
```

```
$ hbase shell < scan4
```

You should see output with lines like this:

AA-95-2014-01-31-JFK-SJU column=cf1:year, timestamp=1424930493341, value=2014

Do you see the advantage of this row key compared to the previous two row key designs?

Yes – this will not hotspot, and the grouping by carrier is useful for scans.

Exercise 4.1d: Import data with composite row key and multiple column families

We now want to explore how adding column families to your table helps with performance. Column families are stored in separate files on disk. If your query only needs to read from a certain group of columns, and those columns are in a column family that is isolated from others, then reading operations will only be performed on those files, saving disk I/O.

1. View the import4.sh script.

The row key for this script is the same as the last one. However, notice the script creates a table with four column families. If a query doesn't use any of the columns in a column family, it won't be read from disk.

For the following example import row from the csv file:

2014,1,1,31,5,2014-01-31,WN,N7704B,228,TUS,LAS,1946,46.00,1958,43.00,0.00,,75.00,60.00,365.00,11.00,0.00,0.00,0.00,32.00,

This is the mapping to the HBase Row Key and Columns:

timing:year=2014,
timing:qtr=1,
timing:month=1,
timing:dom=31,
timing:dow=5,
HBASE_ROW_KEY_3=2014-01-31, // date
HBASE_ROW_KEY_1=WN, // carrier
info:tailnum=N7704B,
HBASE_ROW_KEY_2=228, // flight number
HBASE_ROW_KEY_4=TUS, // orig
HBASE_ROW_KEY_5=LAS, // dest
timing:deptime=1946,
delay:depdelay=46.00,
timing:arrtime=1958,

delay:arrdelay=43.00,
info:cncl=0.00,
info:cnclcode="",
stats:elaptime=75.00,
stats:airtime=60.00,
stats:distance=365.00,
delay:carrierdelay=11.00,
delay:weatherdelay=0.00,
delay:nasdelay=0.00,
delay:securitydelay=0.00,
delay:aircraftdelay=32.00,

2. Execute the import4.sh script :

```
$ more import4.sh
$ ./import4.sh
```

This script creates a table like the following:

Table: airline

Row-key Carrier-Flightnumber-Date-Origin-destination	**delay**			**info**			**stats**		**timing**	
	Air Craft delay	Arr delay	Carrier delay	cncl	cnclcode	tailnum	distance	elaptime	arrtime	Dep time
AA-1-2014-01-01-JFK-LAX		13		0		N7704	2475	385.00	359	…

3. Using the HBase shell scan one row of data .

```
scan '/user/user01/tables/airline', {STARTROW => 'WN-228-2014-01-31-TUS-LAS', LIMIT => 1}
```

ROW	COLUMN+CELL
WN-228-2014-01-31-TUS-LAS	column=delay:aircraftdelay, timestamp=1425775096289, value=32.00
WN-228-2014-01-31-TUS-LAS	column=delay:arrdelay, timestamp=1425775096289, value=43.00
WN-228-2014-01-31-TUS-LAS	column=delay:carrierdelay, timestamp=1425775096289, value=11.00
WN-228-2014-01-31-TUS-LAS	column=delay:depdelay, timestamp=1425775096289, value=46.00
WN-228-2014-01-31-TUS-LAS	column=delay:nasdelay, timestamp=1425775096289, value=0.00
WN-228-2014-01-31-TUS-LAS	column=delay:securitydelay, timestamp=1425775096289, value=0.00
WN-228-2014-01-31-TUS-LAS	column=delay:weatherdelay, timestamp=1425775096289, value=0.00
WN-228-2014-01-31-TUS-LAS	column=info:cncl, timestamp=1425775096289, value=0.00
WN-228-2014-01-31-TUS-LAS	column=info:cnclcode, timestamp=1425775096289, value=
WN-228-2014-01-31-TUS-LAS	column=info:dummy, timestamp=1425775096289, value=
WN-228-2014-01-31-TUS-LAS	column=info:tailnum, timestamp=1425775096289, value=N7704B
WN-228-2014-01-31-TUS-LAS	column=stats:airtime, timestamp=1425775096289, value=60.00
WN-228-2014-01-31-TUS-LAS	column=stats:distance, timestamp=1425775096289, value=365.00
WN-228-2014-01-31-TUS-LAS	column=stats:elaptime, timestamp=1425775096289, value=75.00

```
WN-228-2014-01-31-TUS-LAS    column=timing:arrtime, timestamp=1425775096289, value=1958
WN-228-2014-01-31-TUS-LAS    column=timing:deptime, timestamp=1425775096289, value=1946
WN-228-2014-01-31-TUS-LAS    column=timing:dom, timestamp=1425775096289, value=31
WN-228-2014-01-31-TUS-LAS    column=timing:dow, timestamp=1425775096289, value=5
WN-228-2014-01-31-TUS-LAS    column=timing:month, timestamp=1425775096289, value=1
WN-228-2014-01-31-TUS-LAS    column=timing:qtr, timestamp=1425775096289, value=1
WN-228-2014-01-31-TUS-LAS    column=timing:year, timestamp=1425775096289, value=2014
```

4. Take a look at the results of the table load in the HBase shell

```
describe '/user/user01/tables/airline'

scan '/user/user01/tables/airline', {LIMIT => 5}

scan '/user/user01/tables/airline',{COLUMNS=>['stats'], STARTROW
=> 'AA', LIMIT => 5}

scan '/user/user01/tables/airline', {COLUMNS=>['delay'], STARTROW
=> 'AA', LIMIT => 5}
```

Do you see the advantage with separating columns into different column families?

This would be more obvious with more data and a high workload.

Lab 4.2: Populate and examine Trades Tall and Flat tables

This lab consists of the following steps:

1. Populate the trades tall and flat HBase table
2. Examine with HBase shell.

The following diagram shows the tall table schema, including an example of a few data points.

Row key	CF1	
SYM_TIME	**PRICE (long)**	**VOL (long)**
AMZN_98618600888	12.34	1000
CSCO_98618600666	1.23	3000
GOOG_9861860555	2.34	1000

The following diagram shows the Flat table schema, including an example of a few data points.

RowKey: SYM_DATE	Price						Vol				
	09	10	11	12	…	15	09	10	11	…	15
AMZN_20131020	@ts2: 12.34 @ts1: 12.00						@ts2: 1000 @ts1: 2000				
CSCO_20131023						@ts3: 1.23					@ts3: 3000
GOOG_20130817			@ts6: 2.34						@ts6: 1000		

Exercise 4.2a: Populate trades tall table

The objective of this lab is to populate the trades tall table and examine it

Populate the trades HBase tables

1. Use scp to copy the **trades_table_lab.zip** from the local directory where you put your lab files to your sandbox or aws cluster home folder.
2. After you have copied the zip file , use the unzip command at the linux command line to unzip it.

```
$ unzip trades_table_lab.zip
```

3. Run the CreateTable program as follows to create the tall table .

```
java -cp `hbase classpath`:./schemadesignsolution-1.0.jar
schemadesign.CreateTable tall ./500trades.txt
```

4. Run the CreateTable program as follows to create the flat table .

```
java -cp `hbase classpath`:./schemadesignsolution-1.0.jar
schemadesign.CreateTable flat ./500trades.txt
```

Examine the trades table

1. Try out some queries in HBase shell

Scan one row of each table using the HBase shell

```
scan '/user/user01/trades_flat', {LIMIT => 1}
scan '/user/user01/trades_tall', {LIMIT => 1}
```

Lab 5: Design Schemas for Complex Data Structures

Lab Overview

The objective of this lab is to design HBase schemas for each of the three use cases provided.

Exercise	Required	Duration
5.1: Design schema to model one-to-many relationship	Yes	20 min
5.2: Design a schema to model to many-to-may relationship	Yes	20 min
5.3: Design a schema to model one-to-many relationship	Yes	20 min

Review of HBase Modeling Concepts

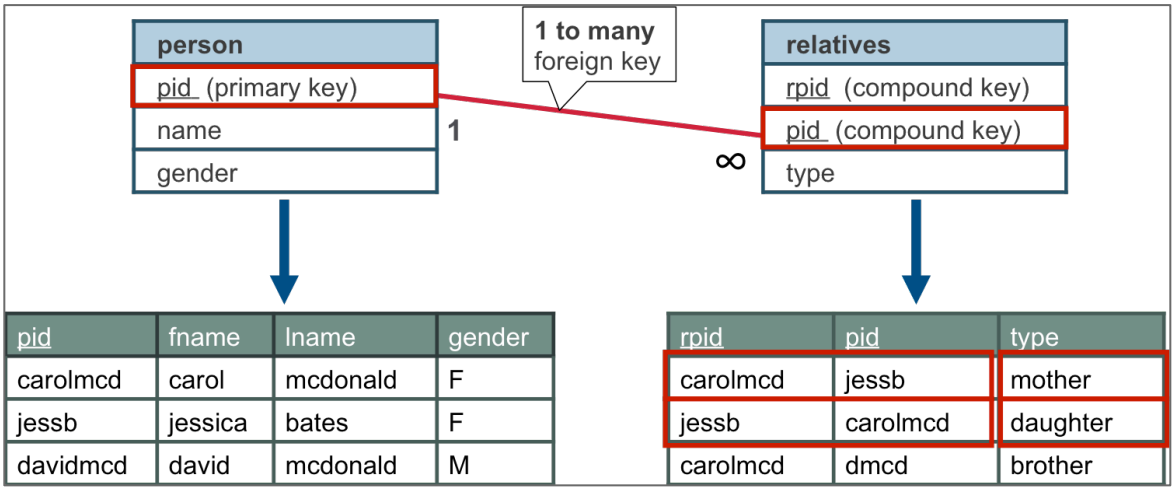
Start with listing all of the use cases your application needs to support. Think about the data you want to capture and the lookups your application needs to do. With HBase you need to design the row keys and table structure in terms of rows and column families to match the data access patterns of your application.

Recall the following steps:

1. Identify Entities: What are the entities in each scenario? What information do we want to retrieve?
2. Identify Queries: You have to know the questions to design your schema. Identify the queries in each use case.
3. Identifying Attributes: These are attributes that can be used to identify unique instances of the entity.
4. Non-identifying attributes: Other attributes become columns
5. Relationships: There are different ways to model relationships. You can use nested (embedded) entities and composite row keys
6. Secondary index (lookup tables) – See if you need to use lookup tables for secondary indexes.

Lab 5.1: Model Person-Relatives Schema

Figure 1:Use Case 1 - Person-Relatives Relational Model



pid	fname	lname	gender
carolmcd	carol	mcdonald	F
jessb	jessica	bates	F
davidmcd	david	mcdonald	M

rpid	pid	type
carolmcd	jessb	mother
jessb	carolmcd	daughter
carolmcd	dmcd	brother

1. Identify the entities

_____

What type of relationship exists between the entities?

_____

2. What information do you want to retrieve? i.e. Identify the queries?

_____

_____

_____

_____

_____

3. What are the identifying attributes?

_____

4. What are the non-identifying attributes?

_____

5. How are you going to model the entity relationship?

_____

_____

_____

6. Do you need to use a lookup table? If so, what would it be?

_____

_____

_____

Based on your answers to the above:

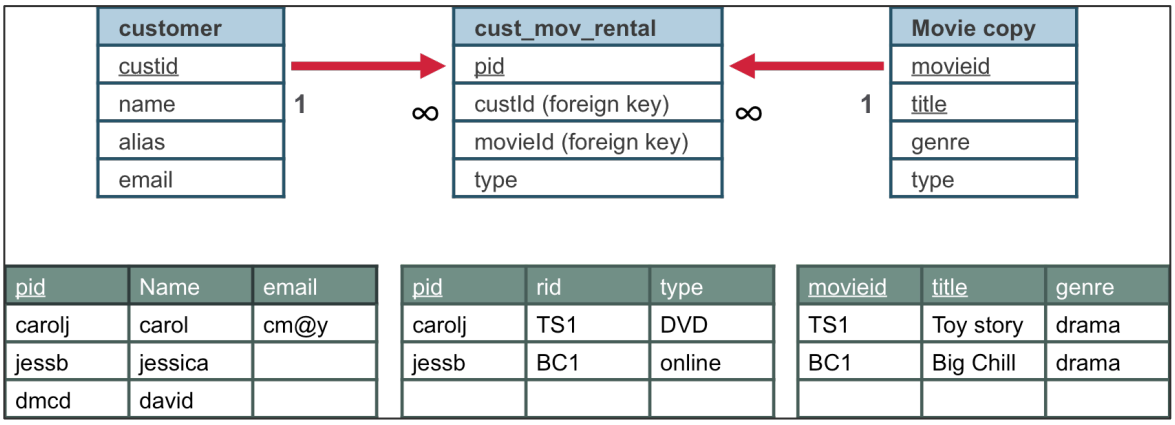
How would you design the row key?

What would you define as your column family (ies)?

What would your columns be?

Lab 5.2: Model Movie Rental Online Store Schema

Figure 2: Use Case 2 - Movie Rental Relational Model



pid	Name	email
carolj	carol	cm@y
jessb	jessica	
dmcd	david	

pid	rid	type
carolj	TS1	DVD
jessb	BC1	online

movieid	title	genre
TS1	Toy story	drama
BC1	Big Chill	drama

1. Identify the entities

_____

What type of relationship exists between the entities?

_____

2. What information do you want to retrieve? i.e. Identify the queries?

_____

_____

_____

_____

_____

3. What are the identifying attributes?

_____

4. What are the non-identifying attributes?

_____

5. How are you going to model the entity relationship?

_____

_____

_____

6. Do you need to use a lookup table? If so, what would it be?

_____

_____

_____

Based on your answers to the above:

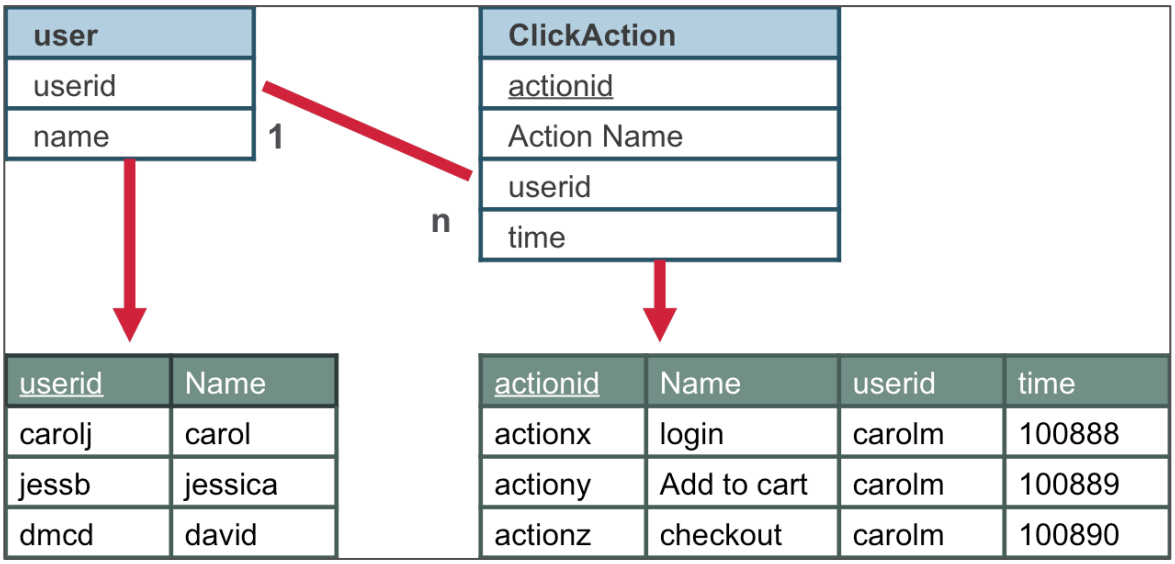
How would you design the row key?

What would you define as your column family (ies)?

What would your columns be?

Lab 5.3: Model Customer Click Event or Action

Figure 3: Use Case 3 - Customer Click Event or Action



userid	Name
carolj	carol
jessb	jessica
dmcd	david

actionid	Name	userid	time
actionx	login	carolm	100888
actiony	Add to cart	carolm	100889
actionz	checkout	carolm	100890

Want to get the latest 10 actions performed by a particular user; and details of action performed by user in groups of ten.

1. Identify the entities

_____

What type of relationship exists between the entities?

_____

2. What information do you want to retrieve? i.e. Identify the queries?

_____

_____

_____

_____

_____

3. What are the identifying attributes?

_____

4. What are the non-identifying attributes?

_____

5. How are you going to model the entity relationship?

_____

_____

_____

6. Do you need to use a lookup table? If so, what would it be?

_____

_____

_____

Based on your answers to the above:

How would you design the row key?

What would you define as your column family (ies)?

What would your columns be?

Lab 6: Use Hive to Query HBase Tables

Lab Overview

In this Lab you will run Hive queries on HBase tables.

Exercise	Required	Duration
6.1: Use Hive to query airlines tables	Yes	15 min
6.2: Use Hive to query trades table	No	

Exercise 6.1: Use Hive with the airlines HBase table

1. Copy lab files to your sandbox or AWS cluster

Use scp to copy the **Lab_6_hive_lab.zip file** from the local directory where you put your lab files to your sandbox or aws cluster home folder.

2. After you have copied the zip file , use the unzip command at the linux command line to unzip it.

```
$ unzip Lab_6_hive_lab.zip
```

If you have not already created and populated the airlines HBase table, refer to Lab 4.1d to do that first.

The following diagram shows the airlines table:

Table: airline

Row-key carrier flightnumber date origin destination	**delay**			**info**			**stats**		**timing**	
	Air Craft delay	Arr delay	Carrier delay	cncl	cnclcode	tailnum	distance	elaptime	arrtime	Dep time
AA-1-2014-01-01-JFK-LAX		13		0		N7704	2475	385.00	359	…

Here is an example row of data for this table.

```
scan '/user/user01/tables/airline', {STARTROW => 'WN-228-2014-01-
31-TUS-LAS', LIMIT => 1}
```

ROW	COLUMN+CELL
WN-228-2014-01-31-TUS-LAS	column=delay:aircraftdelay, timestamp=1425775096289, value=32.00
WN-228-2014-01-31-TUS-LAS	column=delay:arrdelay, timestamp=1425775096289, value=43.00
WN-228-2014-01-31-TUS-LAS	column=delay:carrierdelay, timestamp=1425775096289, value=11.00
WN-228-2014-01-31-TUS-LAS	column=delay:depdelay, timestamp=1425775096289, value=46.00
WN-228-2014-01-31-TUS-LAS	column=delay:nasdelay, timestamp=1425775096289, value=0.00
WN-228-2014-01-31-TUS-LAS	column=delay:securitydelay, timestamp=1425775096289, value=0.00
WN-228-2014-01-31-TUS-LAS	column=delay:weatherdelay, timestamp=1425775096289, value=0.00
WN-228-2014-01-31-TUS-LAS	column=info:cncl, timestamp=1425775096289, value=0.00
WN-228-2014-01-31-TUS-LAS	column=info:cnclcode, timestamp=1425775096289, value=
WN-228-2014-01-31-TUS-LAS	column=info:dummy, timestamp=1425775096289, value=
WN-228-2014-01-31-TUS-LAS	column=info:tailnum, timestamp=1425775096289, value=N7704B
WN-228-2014-01-31-TUS-LAS	column=stats:airtime, timestamp=1425775096289, value=60.00
WN-228-2014-01-31-TUS-LAS	column=stats:distance, timestamp=1425775096289, value=365.00
WN-228-2014-01-31-TUS-LAS	column=stats:elaptime, timestamp=1425775096289, value=75.00
WN-228-2014-01-31-TUS-LAS	column=timing:arrtime, timestamp=1425775096289, value=1958
WN-228-2014-01-31-TUS-LAS	column=timing:deptime, timestamp=1425775096289, value=1946
WN-228-2014-01-31-TUS-LAS	column=timing:dom, timestamp=1425775096289, value=31
WN-228-2014-01-31-TUS-LAS	column=timing:dow, timestamp=1425775096289, value=5
WN-228-2014-01-31-TUS-LAS	column=timing:month, timestamp=1425775096289, value=1
WN-228-2014-01-31-TUS-LAS	column=timing:qtr, timestamp=1425775096289, value=1
WN-228-2014-01-31-TUS-LAS	column=timing:year, timestamp=1425775096289, value=2014

Give Hive access to an existing HBase table

HIVE's external table functionality allows you to create a table that sources its data from an existing HBase table. An external table in hive with HBase is a metadata object that is defined over an HBase table. The metadata maps the table name, column names and types to the HBase table. Once that structure has been defined, you can query it using HiveQL. When you specify an HBase table as EXTERNAL, Hive will not create or drop the HBase table directly.

1. To start the hive shell, ssh into your sandbox or aws cluster (as described in getting started). At the linux command line type hive

```
$ hive

hive>
```

Note: If you see the following error, it is just a logging error, Hive will still work:

```
log4j:ERROR setFile(null,true) call failed.

java.io.FileNotFoundException: /opt/mapr/hive/hive-
0.13/logs/user01/hive.log (No such file or directory) at
java.io.FileOutputStream.open(Native Method)
```

You can also use Hive in non-interactive mode, by giving Hql commands with a file at the command line:

```
$ hive –f  filename.hql
```

Since the queries we will use the Airlines table are more complex, all of the queries are in the **queries.txt** file in the **Lab_6_hive_lab.zip** file, and some of the queries are also in separate .hql files

2. Enter the following command at the Hive prompt to give Hive access to the existing HBase airlines table:

```
hive> CREATE EXTERNAL TABLE flighttable(key string,aircraftdelay
FLOAT,arrdelay FLOAT,carrierdelay FLOAT,depdelay FLOAT,weatherdelay
float,cncl FLOAT,cnclcode string, tailnum string, airtime FLOAT,
distance FLOAT, elaptime FLOAT,arrtime int, deptime int, dom int, dow
int, month int) STORED BY
'org.apache.hadoop.hive.hbase.HBaseStorageHandler'  WITH
SERDEPROPERTIES ("hbase.columns.mapping" =
":key,delay:aircraftdelay,delay:arrdelay,delay:carrierdelay,delay:depd
elay,delay:weatherdelay,info:cncl,info:cnclcode,info:tailnum,stats:air
time,stats:distance,stats:elaptime,timing:arrtime,timing:deptime,timin
g:dom,timing:dow,timing:month") TBLPROPERTIES("hbase.table.name" =
"/user/user01/tables/airline");
```

Exploring the Flight table using Hive queries

```
1. hive> select * from flighttable LIMIT 2;
```

```
OK
AA-1-2014-01-01-JFK-LAX NULL 13.0   NULL 14.0   NULL  0.0 N338AA  359.0  2475.0 385.0  1238   914   1    3    1
AA-1-2014-01-02-JFK-LAX NULL  1.0   NULL -3.0   NULL  0.0 N338AA  340.0  2475.0 385.0  1226   857   2    4    1
Time taken: 0.204 seconds, Fetched: 2 row(s)Time taken: 0.204 seconds, Fetched: 2 row(s)
```

Think now for a moment what sorts of questions you would like to ask of this data. You can then transform those questions into queries which can be executed against the data, using HIVE which will run map reduce jobs. Some example questions:

- Worst delays? (by # of late flights; minutes late, etc.)
- Most common cancelation reasons?

Before you execute any queries in HIVE, it's important to understand what the query will do when it runs.

How many MapReduce jobs will be run?

A simple way to do this is to insert the word explain at the beginning of the select statement.

2. **`hive> explain select count(*) as cancellations, cnclcode from flighttable where cncl=1 group by cnclcode order by cancellations asc limit 100;`**

The output will be long and verbose, showing you details about each stage, column, and variable used to execute the query. One item that is important to look for is the number of times you see "Map Reduce" mentioned - which in this case is twice. This effectively means that 2 MR jobs will run if this query were to actually execute.

```
OK
STAGE DEPENDENCIES:
  Stage-1 is a root stage
  Stage-2 depends on stages: Stage-1
  Stage-0 is a root stage
STAGE PLANS:
  Stage: Stage-1
    Map Reduce
      Map Operator Tree:
          TableScan
            Filter Operator
              Select Operator
                Group By Operator
                  aggregations: count()
                  Reduce Output Operator
      Reduce Operator Tree:
        Group By Operator
          aggregations: count(VALUE._col0)
          Select Operator
            File Output Operator
  Stage: Stage-2
    Map Reduce
      Map Operator Tree:
          TableScan
            Reduce Output Operator
      Reduce Operator Tree:
        Extract
          Statistics: Num rows: 0 Data size: 0 Basic stats: NONE Column stats: NONE
          Limit
            File Output Operator
  Stage: Stage-0
    Fetch Operator
      limit: 100
```

3. Now remove the 'EXPLAIN' command from the beginning of the query and run it.

```
hive> select count(*) as cancellations, cnclcode  from
flighttable where cncl=1  group by cnclcode order by
cancellations asc limit 100;
```

Here is the (shortened) output:

```
Total jobs = 2
MapReduce Jobs Launched:
Job 0: Map: 1  Reduce: 1   Cumulative CPU: 13.3 sec   MAPRFS Read: 0 MAPRFS Write: 0 SUCCESS
Job 1: Map: 1  Reduce: 1   Cumulative CPU: 1.52 sec   MAPRFS Read: 0 MAPRFS Write: 0 SUCCESS
Total MapReduce CPU Time Spent: 14 seconds 820 msec
OK
4598    C
7146    A
```

19108 B

4. Now run a query to find the **count of cancellations for American Airlines**

```
hive> select count(*) as cancellations from flighttable where
cncl=1 and key like "AA%";
```

OK
1574

5. Now run a query to find the **longest air delays**:

```
hive> select arrdelay,key from flighttable where arrdelay > 1000
order by arrdelay desc limit 10;
```

OK
1530.0 AA-385-2014-01-18-BNA-DFW
1504.0 AA-1202-2014-01-15-ONT-DFW
1473.0 AA-1265-2014-01-05-CMH-LAX
1448.0 AA-1243-2014-01-21-IAD-DFW
1390.0 AA-1198-2014-01-11-PSP-DFW
1335.0 AA-1680-2014-01-21-SLC-DFW
1296.0 AA-1277-2014-01-21-BWI-DFW
1294.0 MQ-2894-2014-01-02-CVG-DFW
1201.0 MQ-3756-2014-01-01-CLT-MIA
1184.0 DL-2478-2014-01-10-BOS-ATL

6. Now run a different query for **longest air delays**:

```
hive> select key, a.arrdelay, a.elaptime, a.airtime, a.distance
from flighttable a order by a.arrdelay desc limit 20;
```

7. Now run a query **for fastest airspeed**:

```
hive> select key, a.arrdelay, a.elaptime, a.airtime, a.distance,
((a.distance / a.airtime) * 60) as airspeed from flighttable a
order by airspeed desc limit 4;
```

OK
EV-5449-2014-01-08-ELM-DTW 13.0 77.0 27.0 332.0 737.7777777777777
WN-1087-2014-01-17-ECP-HOU -3.0 120.0 50.0 571.0 685.2
EV-5471-2014-01-31-MSP-RDU 39.0 152.0 89.0 980.0 660.6741573033707
AA-2371-2014-01-28-STL-DFW -36.0 115.0 50.0 550.0 660.0

8. Now run a query for **maximum air delay** for united:

```
hive> select max(arrdelay) from flighttable  where key like
"UA%";
```

OK
1043

9. Now run a different query for **average carrier delay for Delta Airlines**:

```
hive> select avg(a.carrierdelay) as CarrierDelayMinutes from
flighttable a where  key like "DL%";
```

Lab 6.2 – Stretch Lab: Use Hive to query the Trades table

If you have not already created and populated the trades tall HBase table, refer to lab 4.2 to do that first.

The following diagram shows the tall table schema, including an example of a few data points.

Row key	**CF1**	
RowKey: SYM_TIME	**PRICE (long)**	**VOL (long)**
AMZN_98618600888	12.34	1000
CSCO_98618600666	1.23	3000
GOOG_9861860555	2.34	1000

Give Hive access to an existing HBase table

Enter the following command at the Hive prompt to give Hive access to the existing HBase trades_tall table:

NOTE: all of the queries are in the queries.txt file in the Lab_6_hive_lab.zip file, you can edit them and copy paste them from here.

```
hive> CREATE EXTERNAL TABLE trades(key string, price bigint, vol
bigint) STORED BY
'org.apache.hadoop.hive.hbase.HBaseStorageHandler' WITH
SERDEPROPERTIES ("hbase.columns.mapping" =
"CF1:price#b,CF1:vol#b")  TBLPROPERTIES("hbase.table.name" =
"/user/user01/trades_tall");
```

For further information reference the following URL.
https://cwiki.apache.org/confluence/display/Hive/HBaseIntegration

Hive Queries

Enter some commands on the hive cli to explore

```
hive> describe trades ;
hive> select * from trades;
```

```
hive> select * from trades where key like "GOOG%";
```

```
hive> select avg(price) from trades where key like "GOOG%"
```

Check out the documentation here at the following URL.https://cwiki.apache.org/confluence/display/Hive/LanguageManual+Cli

Lab 5: Design Schemas for Complex Data Structures - Solution

Lab Overview

The objective of this lab is to design HBase schemas for each of the three use cases provided.

Exercise	Required	Duration
5.1: Design schema to model one-to-many relationship	Yes	20 min
5.2: Design a schema to model to many-to-may relationship	Yes	20 min
5.3: Design a schema to model one-to-many relationship	Yes	20 min

Review of HBase Modeling Concepts

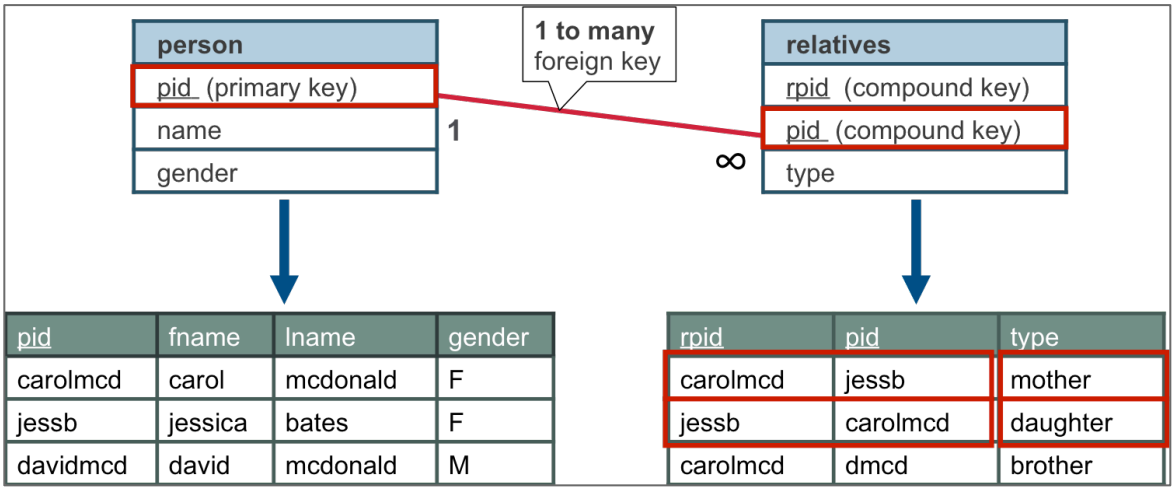
Start with listing all of the use cases your application needs to support. Think about the data you want to capture and the lookups your application needs to do. With HBase you need to design the row keys and table structure in terms of rows and column families to match the data access patterns of your application.

Recall the following steps:

1. Identify Entities: What are the entities in each scenario? What information do we want to retrieve?
2. Identify Queries: You have to know the questions to design your schema. Identify the queries in each use case.
3. Identifying Attributes: These are attributes that can be used to identify unique instances of the entity.
4. Non-identifying attributes: Other attributes become columns
5. Relationships: There are different ways to model relationships. You can use nested (embedded) entities and composite row keys
6. Secondary index (lookup tables) – See if you need to use lookup tables for secondary indexes.

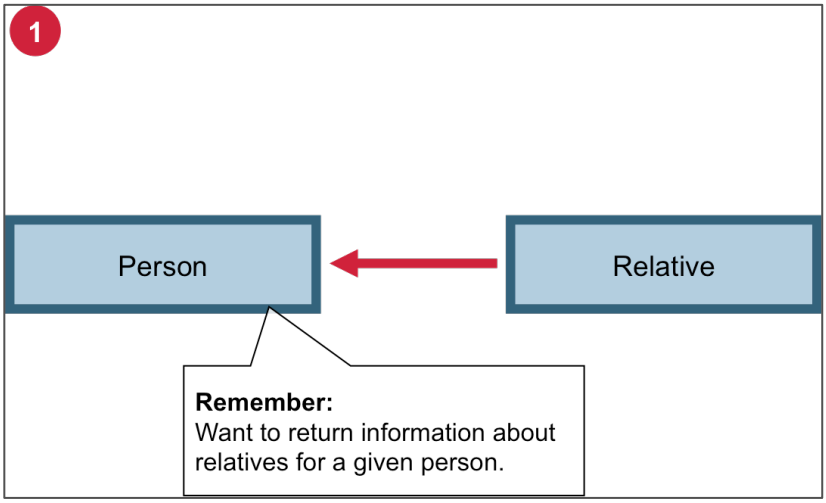
Lab 5.1: Model Person-Relatives Schema

Figure 1:Use Case 1 - Person-Relatives Relational Model



1. Identify the entities

Figure 2: Identify Entities



Here we pick Person as the entity.

What type of relationship exists between the entities?

It is a one-to-many relationship. A person can have more than one relative.

2. What information do you want to retrieve? i.e. Identify the queries?

We might want to retrieve information regarding the person. Look at the relational table here and note that we have the first name, last name and gender.

We would also like to retrieve information about the person's relatives such as retrieve all relations; get all brothers; get all daughters; get the person's mother, and so on.

3. What are the identifying attributes?

Option A: Identifying attribute = pid

Option B: Identifying attributes = pid, first name, last name

Recall that identifying attributes are included in the row key.

4. What are the non-identifying attributes?

Option A: Non-identifying attributes = gender, first name, last name

Option B: Non-identifying attributes = gender

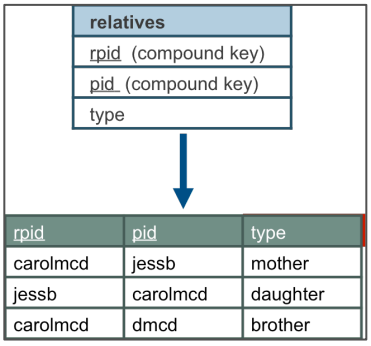
The non-identifying attributes map to columns.

5. How are you going to model the entity relationship?

It is a one-to-many relationship. One-to-many relationships can be modeled as nested or embedded entities. The parent identifying attribute(s) becomes the row key and the child identifying attributes are column qualifiers.

The child identifying attributes in this case are rpid and type.

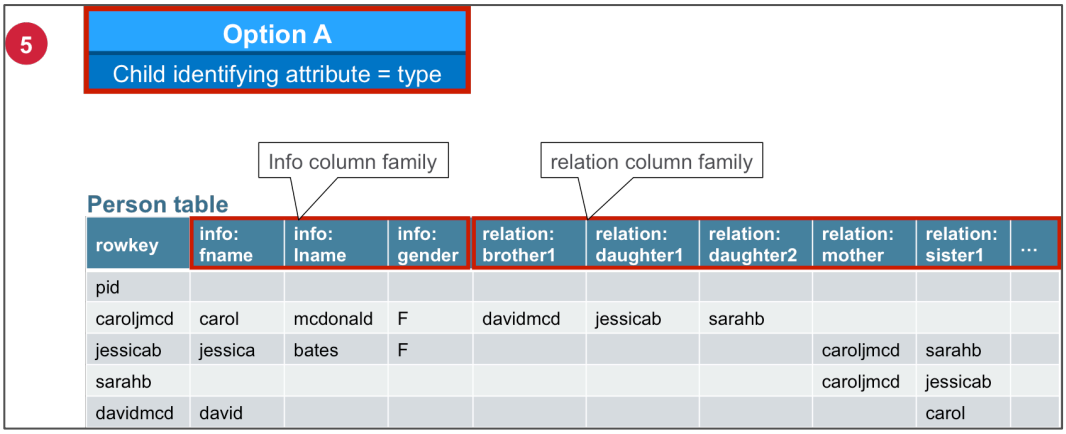
Figure 3: Relative table



Below you see both options for modeling the relationship – one with child identifying attribute being type and the other, with rpid.

Option A: Child Identifying attribute = type

Figure 4: Child identifying attribute = type



rowkey	info: fname	info: lname	info: gender	relation: brother1	relation: daughter1	relation: daughter2	relation: mother	relation: sister1	…
pid									
caroljmcd	carol	mcdonald	F	davidmcd	jessicab	sarahb			
jessicab	jessica	bates	F				caroljmcd	sarahb	
sarahb							caroljmcd	jessicab	
davidmcd	david							carol	

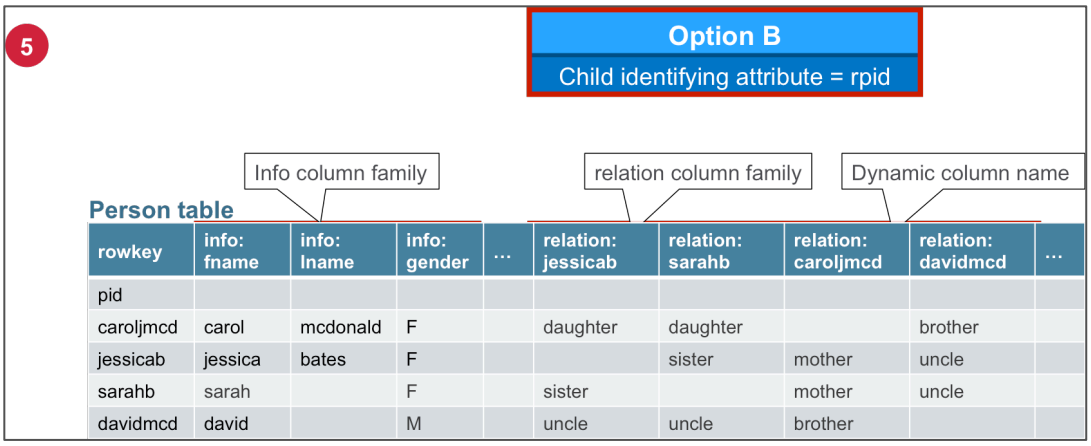
Here the "relation" column family has type (or relative) as columns such as brother, mother, daughter and sister.

Can you think of any problems you may encounter with this model?

Since, there can more than one daughter, sister, brother, etc., you would require a number in the name. This in turn would require a counter to keep track of the number complicating the column name.

Option B: Child Identifying attribute = rpid

Figure 5: Child identifying attribute = rpid



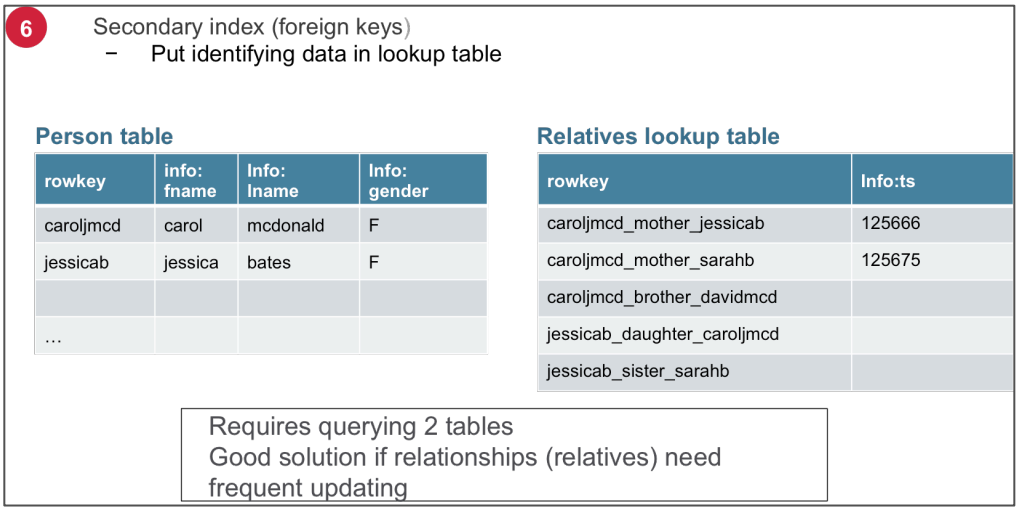
rowkey	info: fname	info: lname	info: gender	…	relation: jessicab	relation: sarahb	relation: caroljmcd	relation: davidmcd	…
pid									
caroljmcd	carol	mcdonald	F		daughter	daughter		brother	
jessicab	jessica	bates	F			sister	mother	uncle	
sarahb	sarah		F		sister		mother	uncle	
davidmcd	david		M		uncle	uncle	brother		

In this solution, in the "relation" column family, we use the rpid as the columns which is dynamic.

6. Do you need to use a lookup table? If so, what would it be?

We can also use a lookup table to model the relationship. Look at the Figure 6 below.

Figure 6: Using a lookup table



Here we have two tables - the Person table and the Relatives lookup table. We have put the identifying attributes for the relative foreign key as a relative lookup table. This means we can scan the relative lookup table to find all of the relatives and relationship types by pid.

A disadvantage of this solution is if you want to get the relatives for a user and the user information it would require two queries. An advantage of this solution is if the relationships between the users change frequently, it is easier to add and delete rows from the relative lookup table than to add and delete columns from the column family solution.

Based on your answers to the above:

How would you design the row key?

There are different ways to design your row key. Based on steps 3 and 4, we saw a few choices:

Option A: Row key = pid

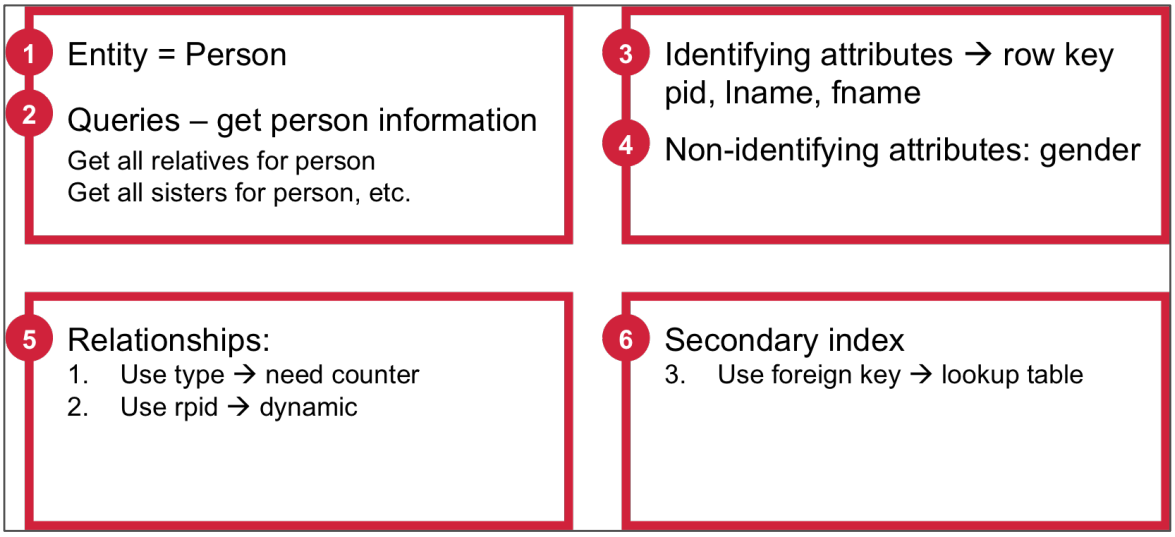
In this case, you would get and scan by pid.

Option B: Row key = pid + lname + fname

With this row key design, you can scan by first name and last name too.

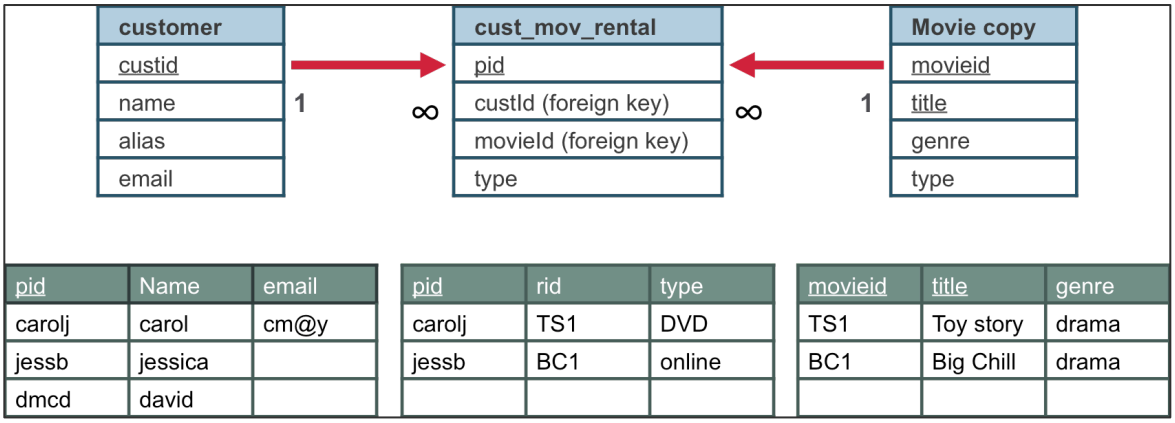
What would you define as your column family (ies)? What would your columns be?

Clearly, row key design is not the only factor. You also have to consider how your data is going to be accessed, what you are interested in retrieving and what type of entity relationship you are trying to model. In step 5 and 6, you see that the row key is defined as the pid, and you have a few choices for designing your column families and columns.



Lab 5.2: Model Movie Rental Online Store Schema

Figure 7: Use Case 2 - Movie Rental Relational Model



pid	Name	email
carolj	carol	cm@y
jessb	jessica	
dmcd	david	

pid	rid	type
carolj	TS1	DVD
jessb	BC1	online

movieid	title	genre
TS1	Toy story	drama
BC1	Big Chill	drama

In this scenario, we have a Movie rental online store. The basic premise here is that customers rent movies and we have information about each customer and each movie.

1. **Identify the entities**

The entities here are: customer; movies.

What type of relationship exists between the entities?

There is a many-to-many relationship here. A customer can rent many movies. More than one customer can rent a movie (movie title).

2. **What information do you want to retrieve? i.e. Identify the queries?**

- Get customer's email by custid

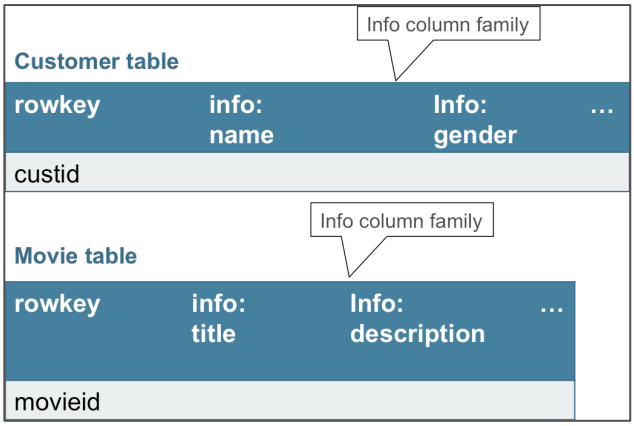
- Get Movie title, genre by movieid
- Get all movie rentals for a customer
- Get all customer rentals for a movie

3. **What are the identifying attributes?**
4. **What are the non-identifying attributes?**

Customer table: Identifying attribute = custid; Non-identifying attributes = name, gender, email etc.

Movie table: Identifying attribute = movieid; Non-identifying attributes = title, description, genre, etc.

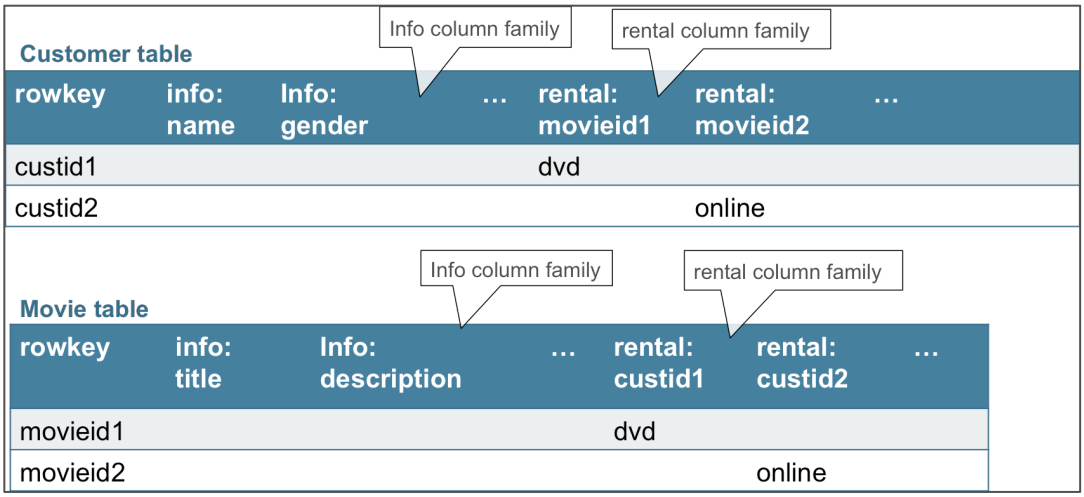
Figure 8: Identifying and Non-identifying attributes for Customer and Movies



5. **How are you going to model the entity relationship?**

This is a many-to-many relationship. We use two tables to model this type of relationship in HBase – entity1_by_entity2 and entity2_by_entity1. In this solution, we have two tables, Customer and Movie. The Customer table has two column families. The Info column family contains the non-identifying attributes for customer (entity1) as the columns. The rental column family uses the identifying attribute for movies (entity2) as columns. Similarly, for the Movie table, the Info column family has the non-identifying attributes for the Movie entity as columns and the rental column family has the identifying attribute for the customer entity as columns.

Figure 9: Many-to-many relationships - Using two tables



6. **Do you need to use a lookup table? If so, what would it be?**

Yes, you could use a lookup table for this. However, the solution proposed above gives faster reads.

Based on your answers to the above:

How would you design the row key?

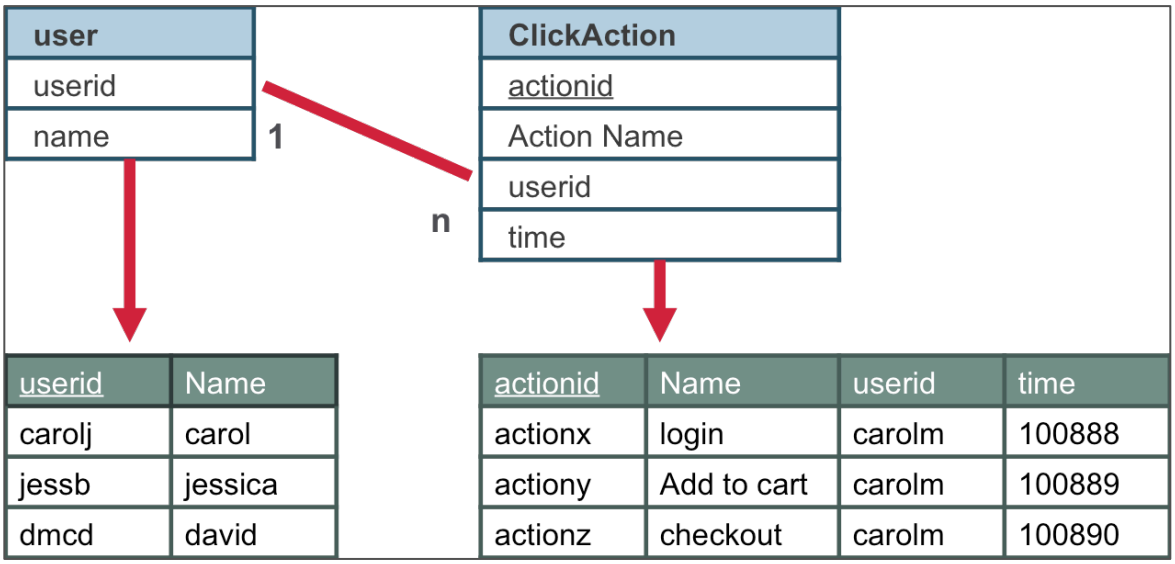
What would you define as your column family (ies)?

What would your columns be?

Look at Figure 9.

Lab 5.3: Model Customer Click Event or Action

Figure 10: Use Case 3 - Customer Click Event or Action



Want to get the latest 10 actions performed by a particular user; and details of action performed by user in groups of ten.

1. **Identify the entities**

Entity = user

What type of relationship exists between the entities?

This is a one-to-many relationship with

2. **What information do you want to retrieve? i.e. Identify the queries?**
   - Get user's name
   - Get the latest 10 actions performed by a particular user
   - Get the details of actions performed by user in a group of 10 actions at a time
3. **What are the identifying attributes?**
4. **What are the non-identifying attributes?**

Identifying attributes: userid

Non-identifying attributes: first name, last name, etc.

5. **How are you going to model the entity relationship?**

This is a one-to-many relationship and can be modeled using nested entities. In this solution, we use a lookup table.

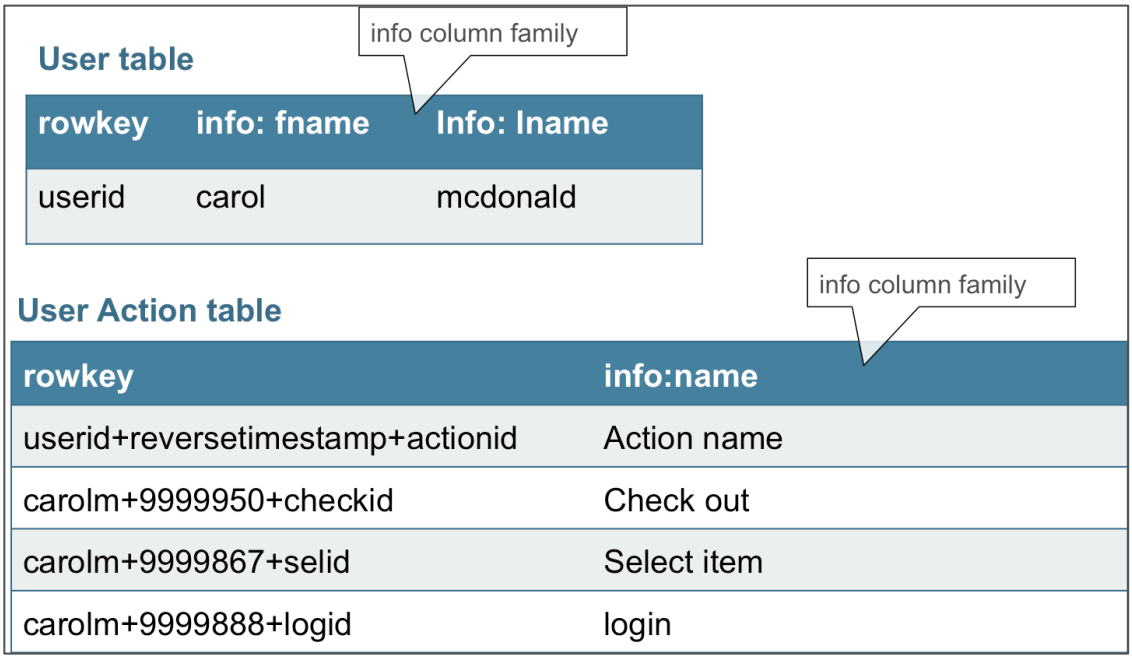
6. Do you need to use a lookup table? If so, what would it be?

We have the user table with the userid and row key. We have the User action table that is the lookup table with a composite row key. The row key is composed of the userid, reverse timestamp and the actionid.

What purpose the reverse timestamp in the row key serve?

We want to retrieve the latest 10 actions. The reverse timestamp in the row key will add the most recent to the top.

Figure 11: Using lookup table for one-to-many relationship



Based on your answers to the above:

How would you design the row key?

What would you define as your column family (ies)?

What would your columns be?

Refer to Figure 11 above.

Developing HBase Applications Lab Guide

Late 2014

Get Started

Welcome to *Developing HBase Applications*

This course teaches developers, via lectures and hands-on lab exercises, how to write HBase Applications using Java. This course prepares you to successfully pass the *MapR Certified Hadoop Developer*, provided that you complete all lab exercises. Here is the course outline

01 -- Java client API Part 1

- Lab on Java Client API Get, Put, Delete, Scan
  - Use these APIs in a Shopping application

02 - Java API Part 2

- Lab on Java Client API HTable Batch, checkAndPut
  - Use HTable Batch APIs in an application
  - Use HTable checkAndPut APIs for row transactions in a Shopping application

03 - Java Client API for Administrative Features

- Lab create tables and define properties
  - Use the HBaseAdmin Java Interface to create tables for a Time Series Application

04 - Advanced HBase Java API

- Lab
  - Use Filters in an Time Series Application
  - Use Counter Increment for row transactions in a Shopping application

05- Time Series application with Flat Wide and Tall Narrow Implementations

- Lab
  - Programming a Time Series application

06- MapReduce on HBase

- Lab
  - Reading from HBase and Writing back Daily Statistics for a Time Series Application

07- Social Application

- Lab (optional)
  - Programming a Social Application

08 - Bulk Loading of Data

- Lab

- Use importtsv and Copy_table tools for bulkloading
- Use MapReduce to load from a file into HBase

09 Performance

- Lab
  - Use YCSB Benchmark tool

10 Securing MAPR-DB tables

- Lab
  - Use ACEs for MAPR-DB Tables Authorization

Prerequisites

- Beginner-to-intermediate fluency with Java or object-oriented programming.
- A laptop (Linux, PC or Mac) to connect to a Hadoop cluster via SSH and web browser
- RECOMMENDED: Complete the *HBase Schema Design and Data Modeling* course first,

How to Use this Guide

Typographical Conventions

`Courier`	Things you type as shown
`Courier`	Things as seen on screen
italics	Things you replace with a value
Durations	Suggested time it takes to complete an exercise. Some are required, other are optional

Lab Exercise Timing

Each Lesson's Labs are listed with estimated time to completion.

Exercise	Required	Duration
0.1: Install needed Software	Yes	Before class
0.2. Connect to the MapR Sandbox or Cluster	Yes	15 min

Tools Needed to Complete Labs

In order to complete the labs you should have already installed these components:

See detailed instructions to install, configure & verify these components in the set up document.

1. **Install JDK 7**
2. **Install Eclipse (for Java developers)**
3. **Install SSH and SCP clients**
4. **MVMware player / VirtualBox (prerequisite for MapR Sandbox)**
5. **MapR Sandbox**
6. **MapR Client (optional – covered in a separate document)**

Connect to a MapR Cluster (Sandbox or AWS)

The purpose of this lab is for you to establish connectivity to your MapR Sandbox or a cluster environment that you will use for the hands-on exercises in this course.

Logging into the Sandbox or AWS cluster

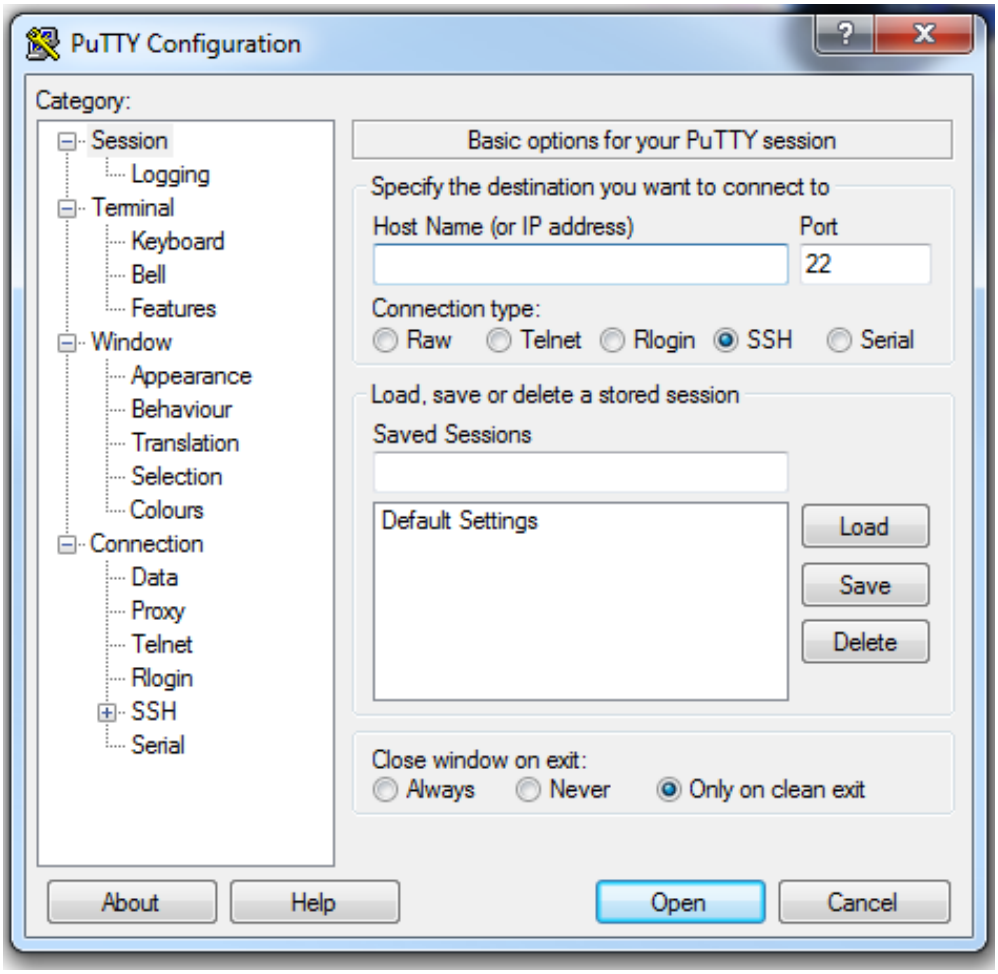
This procedure assumes that you have downloaded and installed the MapR Sandbox as part of the class prerequisites. If not, please do that before continuing here. Make a note of the IP address assigned to the Sandbox, as you will need it to log in remotely.

The Sandbox is preconfigured with several user accounts. You will be logging in as user01 to complete most of the lab exercises. The password for user01 on the sandbox is mapr. If you are using the AWS cluster in a class , the instructor will give you the ip address and userid.

You will need to use a Secure Shell (ssh) client to log in. PuTTY, Cygwin, or a bash shell on a Unix-like OS with OpenSSH installed are all acceptable clients. If you do not have an ssh client, you can log in to the Sandbox using the virtual machine console mode. Connect to your Sandbox as user01.

OPTION 1: Log in using PuTTY

1. Start your PuTTY Client.



2. In the Host Name (or IP address) box, enter the host name or IP address for your Sandbox virtual machine.
3. Click the Open button to open the connection to your node.
4. When the terminal window appears, log into the terminal as user01.

OPTION 2: Log in using terminal SSH client

1. Start your terminal client. Use the ssh program to connect to the hostname or IP address for your Sandbox. Connect as user01

   ```
   $ ssh user01@ IP address.
   ```

Copying Files to the Sandbox or AWS Cluster

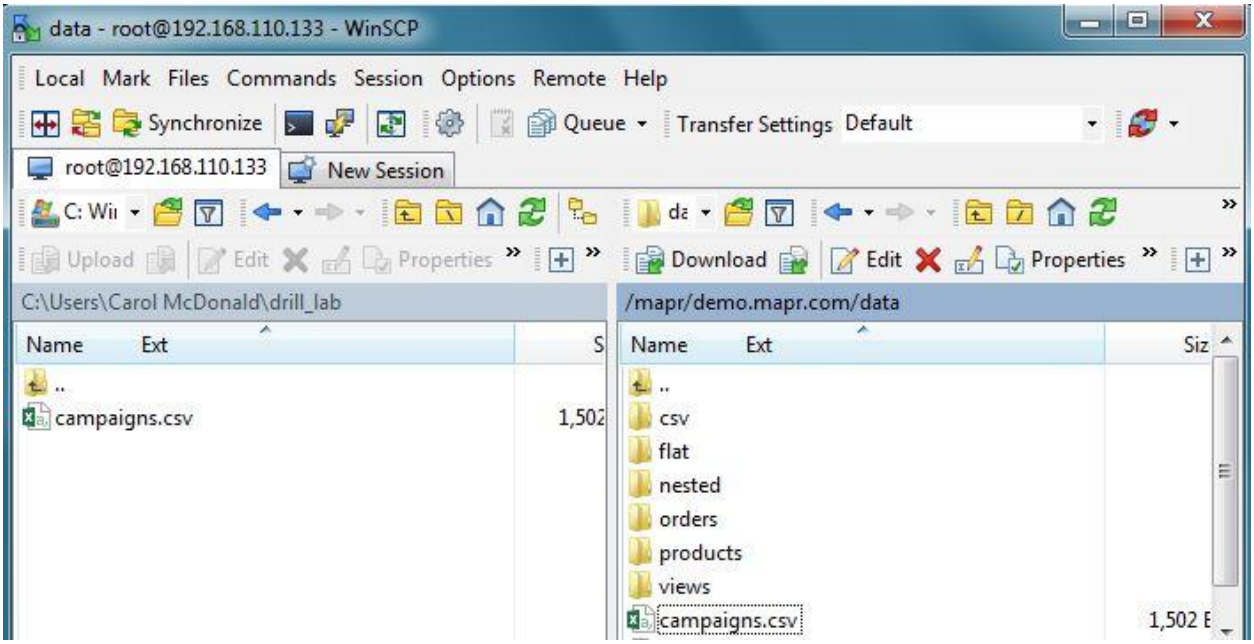
2. Copying files to the Sandbox or AWS Cluster
3. Use Scp to copy files from your laptop to a Sandbox or AWS cluster. The syntax for the scp command is:

```
scp [options] username1@source_host:directory1/filename1
username2@destination_host:directory2/filename2
```

4. On windows you can use winscp, download here

   http://winscp.net/eng/download.php **follow the instructions here**

   http://winscp.net/eng/docs/start **with userid user01, password mapr, cluster ip address and port 22.**



Connect to MCS on a MapR Cluster with a web browser

Use Firefox or Chrome to open the following URL: https:// ip address from instructor :8443/

Your browser will display a security warning that indicates the Web server is using a self-signed certificate. Different browsers show you this warning differently. Click the equivalent of "proceed despite this security warning" in your browser, and you will see the login page of the MapR Control System. Login with the userid and password given by the Instructor

Import Projects in Eclipse

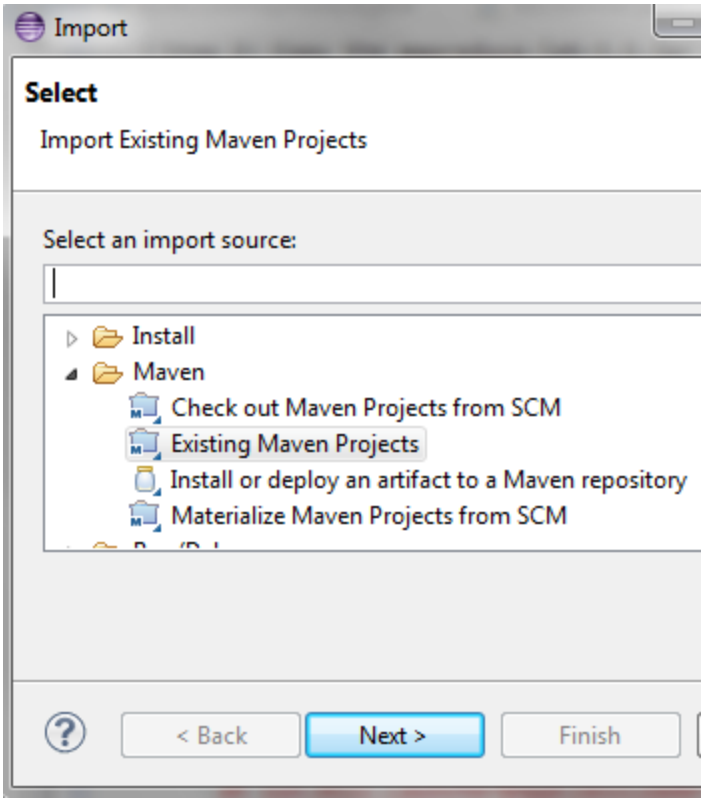
Lab Overview

The purpose of this lab is to import the lab project files into Eclipse. It should take approximately 5 minutes.

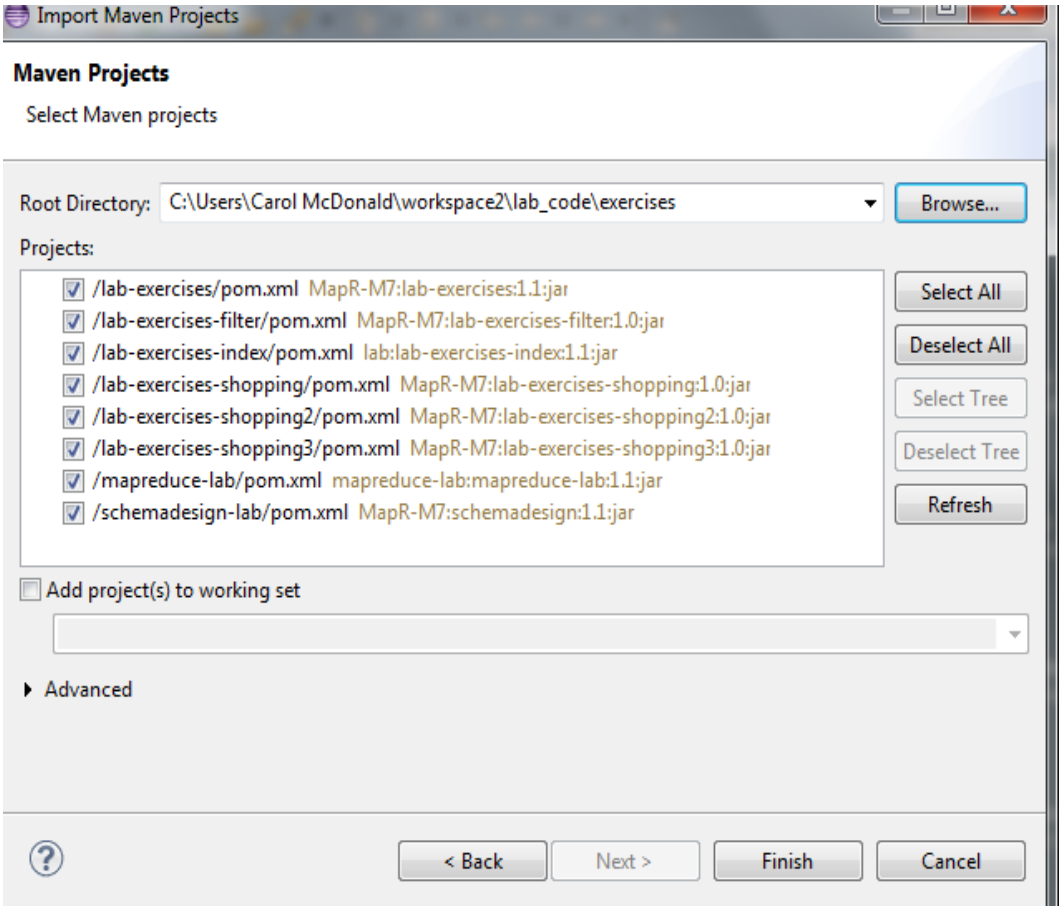
Exercise: Import Lab projects into Eclipse

1. Select Import from the File menu.

2. Expand the Maven Folder and select Existing Maven Projects.

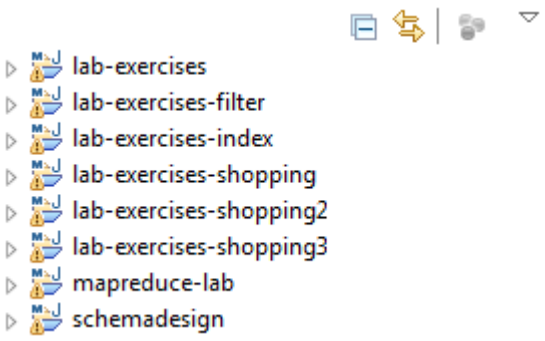


3. Browse to the lab exercises directory where you saved the lab code as shown in the following image.



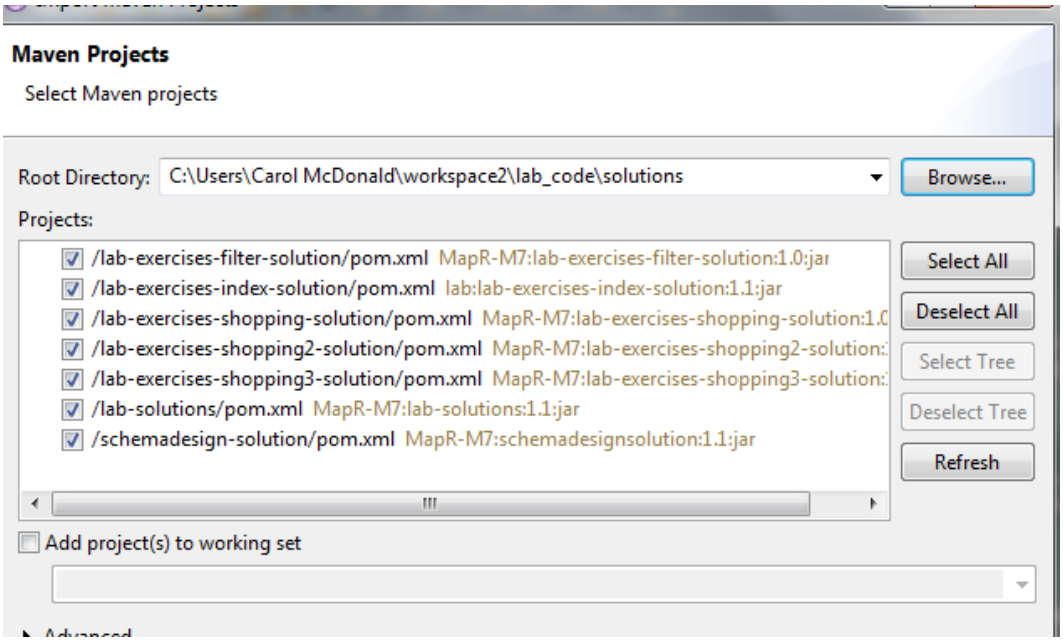
4. Select all of the pom.xml. Select Finish.

You should see the following:



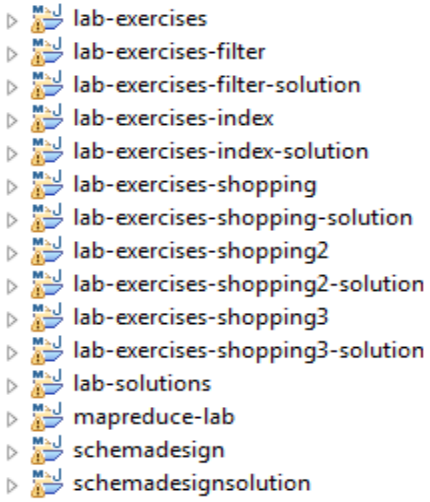
Import solutions

1. Browse to the lab solutions directory where you saved the lab code as shown in the following image.



2. Select all the pom.xml. Select Finish.

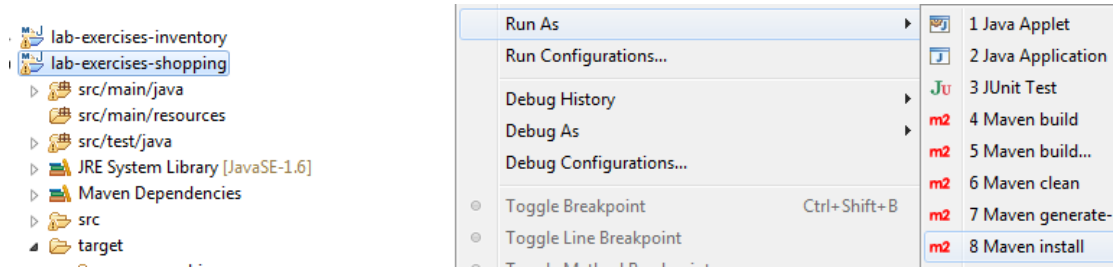
3. You should see the solutions as shown in the following image.



Build a Maven project to create the jar file

4. Select the project from Run menu.

5. Select Run As and Maven install as shown in the following image.

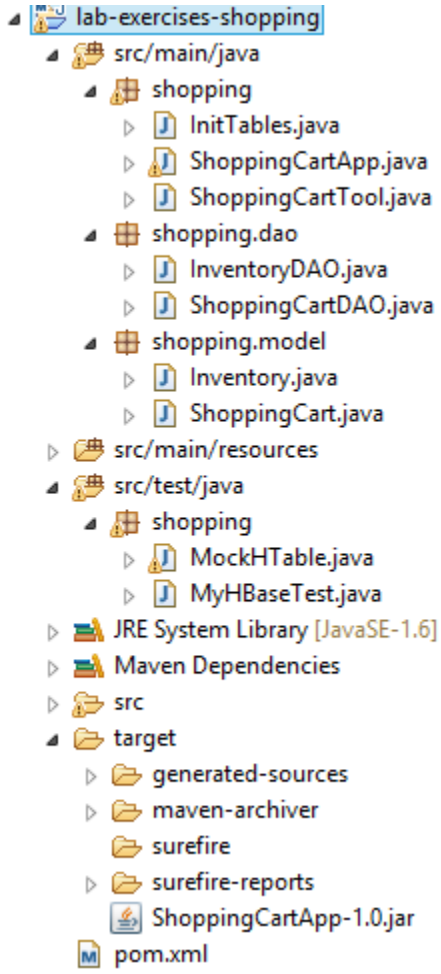


If you have problems building maven projects with eclipse, let the instructor know. You may need to delete your maven ${user.***home***}/.m2 ***directory***

http://maven.apache.org/guides/mini/guide-configuring-maven.html

Program Structure

A jar file will be created in the target folder as show in the following image.



Lab 1: Developing Java applications using HBase Java API

Lab Overview

This lab shows how to create HBase Tables and to use HBase Java API to insert data into table, and then perform get, scan & delete operations. As part of this training we'll be using MapR cluster to run the lab exercises

Exercise	Required	Duration
1.1 Import and build the "lab-exercises-shopping" project	Yes	20 min
1.2 Insert and get data in the ShoppingCartDAO class	Yes	20 min

Background information on HBase Java API

HBase is the Hadoop database, which provides random, realtime read/write access to very large data. See the references on HBase for more information. The HBase Java API is to allow Java developers to write applications to create HBase Tables and then perform all the CRUD operations on them. Some of the important classes that we'll use as part of this lab exercise are:

HTable: HTable manages connections to the HBase table.

```
Configuration conf = HBaseConfiguration.create();
HTable table = new HTable(conf, "mytable");
```

Put: "Put" adds new records into the HBase table.

```
HTable table = ... // instantiate HTable
Put put = new Put(Bytes.toBytes("key1"));
put.add(Bytes.toBytes("colfam"), Bytes.toBytes("qual"),
Bytes.toBytes("value"));
put.add(...); ...
table.put(put);
```

Get: "Get" fetches a single record given its key.

```
HTable table = ... // instantiate HTable
Get get = new Get(rowKey);
get.addColumn(Bytes.toBytes("colfam"), Bytes.toBytes("qual")); // to
fetch specific column get.addFamily("colfam2"); // to fetch the all
from column
Result result = table.get(get);
```

Delete: "Delete" deletes a single record given its key.

```
HTable table = ... // instantiate HTable
Delete toDelete = new Delete(rowKey);
table.delete(delete);
```

Scan: "Scan" searches through multiple rows iteratively for specified attributes.

```
HTable table = ... // instantiate HTable
Scan scan = new Scan();
scan.addColumn(Bytes.toBytes("cf"),Bytes.toBytes("qual"));
ResultScanner scanner = table.getScanner(scan)
try {
    for(Result result : scanner) {
            // process Result instance
    }
 } finally {
        scanner.close();
 }
```

References

- MapR document on Installing & getting started with HBase
- MapR M7 Tables
- MapR Contro Systems
- MapR Forum posts related to HBase
- HBase Javadocs for 0.94 release

Exercise 1.1: Import, build and run “lab-exercises-shopping” project

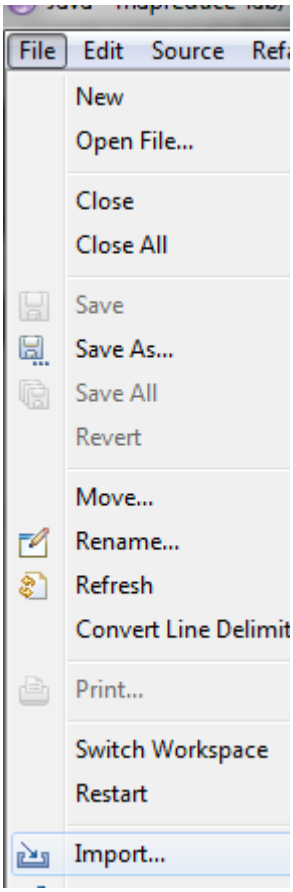
In the first lab exercise, we’ll import the exercise project “lab-exercises-shopping” into Eclipse, review the code, build the project using Maven, run unit tests, upload the jar file to the cluster and run.

If you have already imported the project skip to 1b

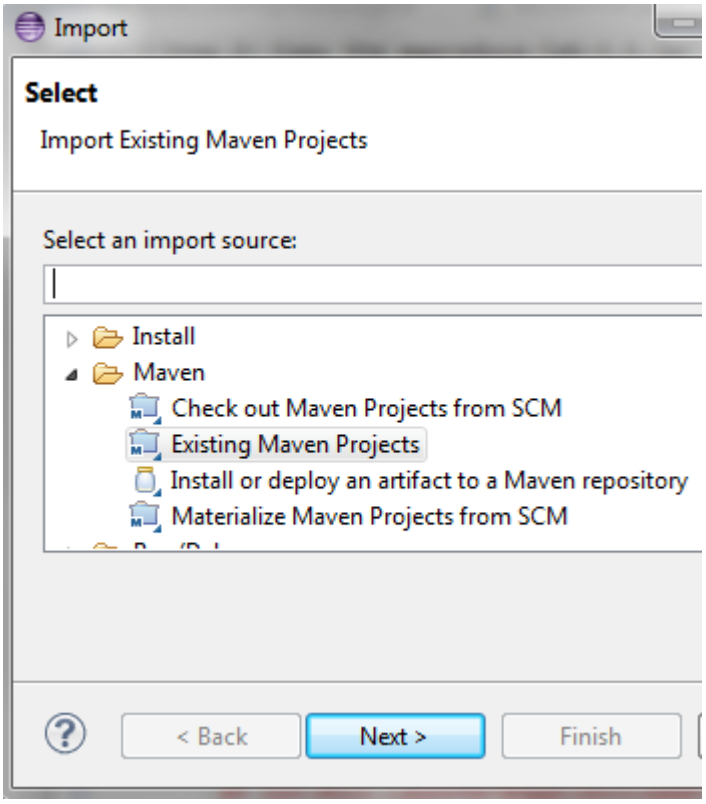
Import project and build

1. Import the **lab-exercises-shopping** project into Eclipse:

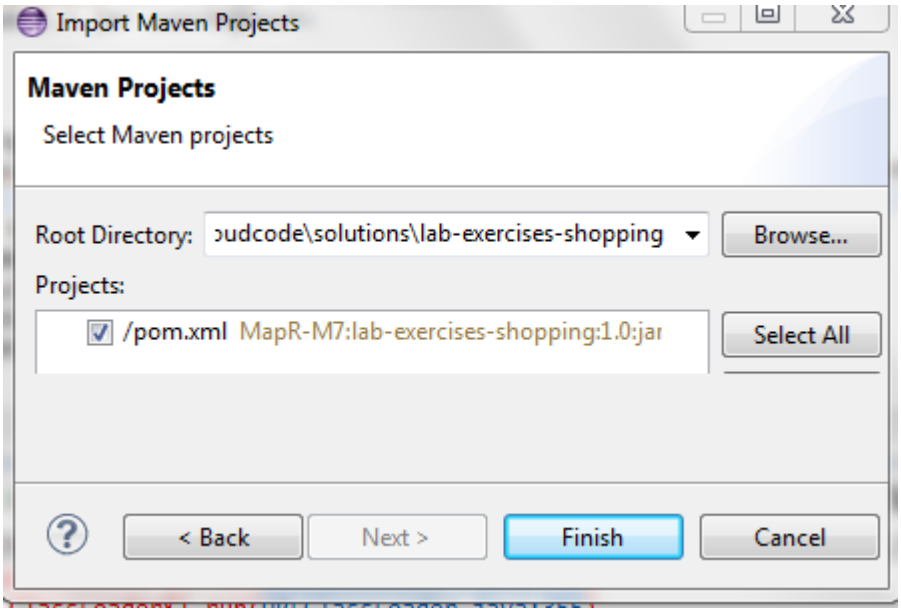
2. Under the ‘File’ menu select ‘Import…’



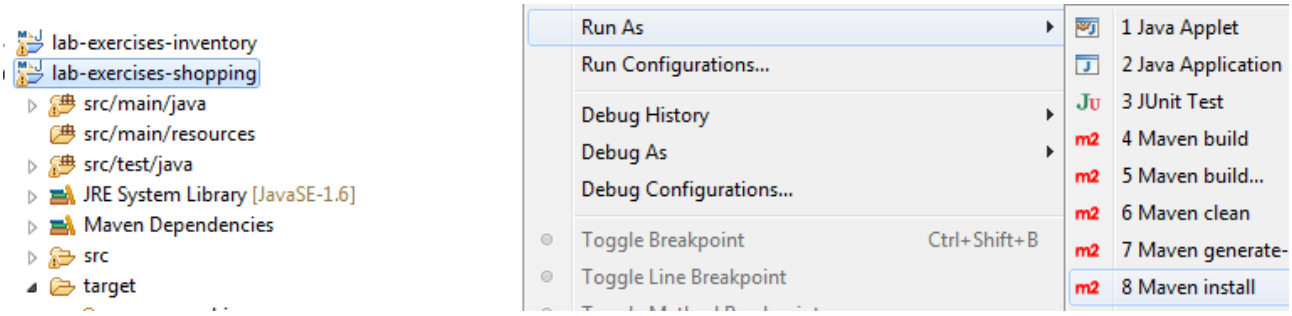
3. Expand the Maven Folder and select ‘Existing Maven Projects’



4. Then for Root Directory, browse to the directory where you saved the lab exercises as shown below, select the pom.xml, Then select 'Finish'.

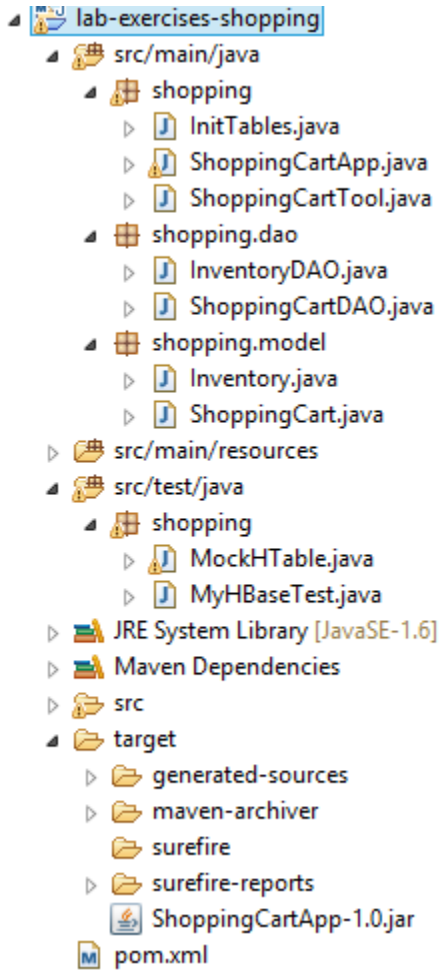


5. To build labs from Eclipse, Select the project and under the "Run" menu select "Run As" then select "Maven install" as shown in the figure below.



If you have problems building with eclipse , let the instructor know. You may need to delete your maven ${user.***home***}/.m2 ***directory***

Program Structure

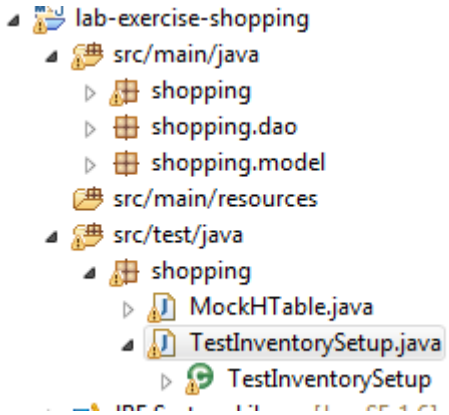


The project consists of the following Java classes:

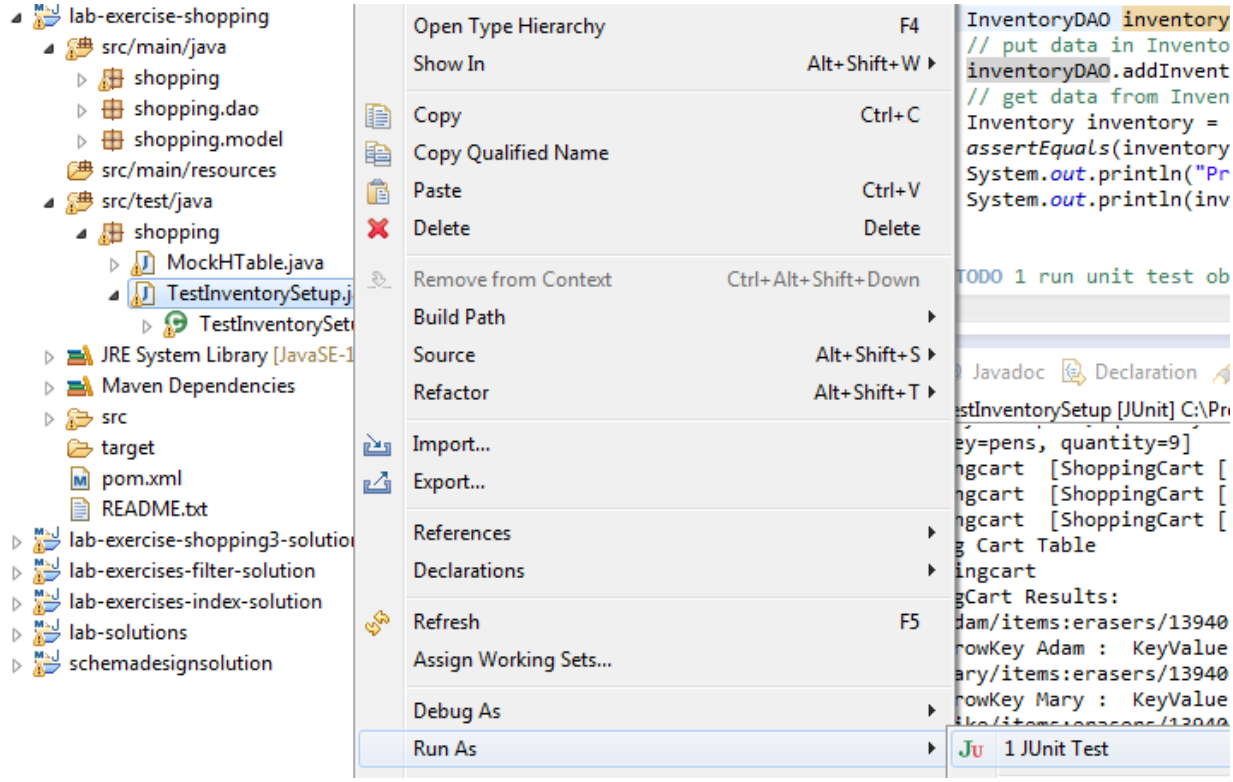
- **`ShoppingCartApp`** – A main driver class to create tables and populate tables with test data
- **`InventoryDAO`** – A Data Access Object (DAO) which contains the code to put, get, delete, scan Inventory objects
- **`ShoppingCartDAO`** – A Data Access Object (DAO) which contains the code to put, get, delete, scan ShoppingCart objects
- Inventory- a model object for inventory data
- ShoppingCart- a model object for shopping cart data

Steps to follow to run the Inventory Unit Test:

1. Look in the src/test/java directory at shopping.TestInventorySetup .
2. **Uncomment the first unit test @Test line, remove the // before @TEST**
3. Then run the TestInventorySetup junit test: right mouse click on the class, and select run as junit test.

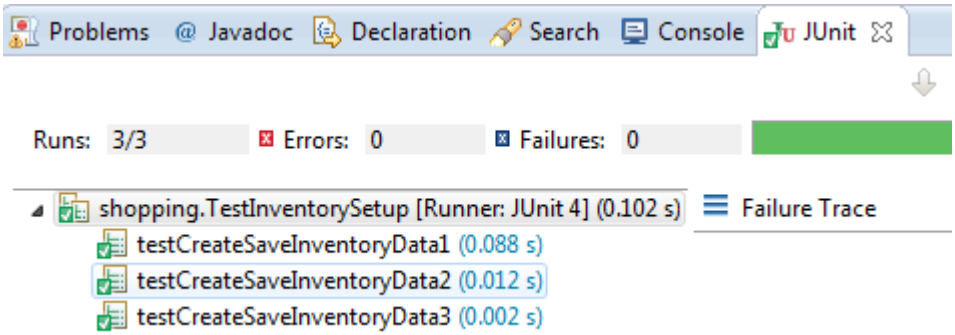


1. Right mouse click on the class, and select run as junit test.

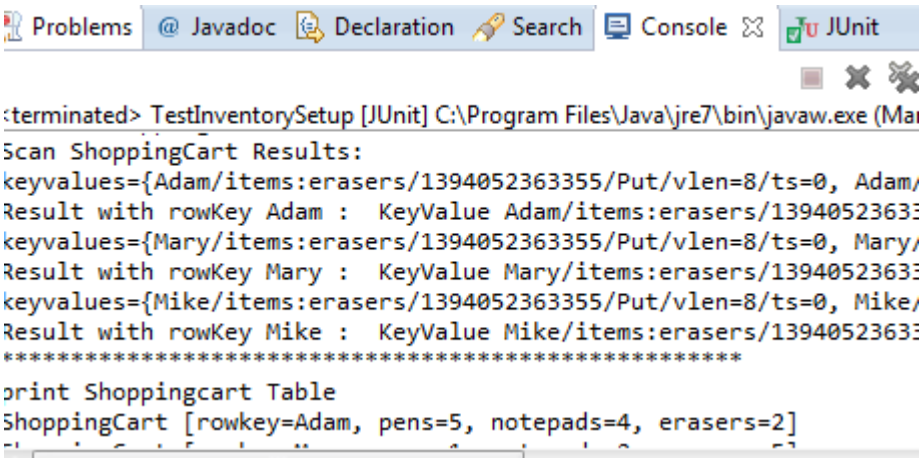


2. !f you do not see a run as junit, make sure you Uncommented the unit test @Test line, remove the // before @TEST

3. If the tests pass then the junit window should look like this



If the tests pass then the console window should look like this



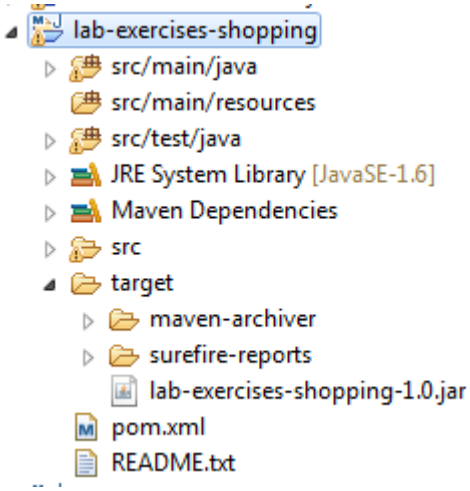
4. Observe what the unit test does, and prints out. The unit test uses a MockHBaseTable which is an apache open source in memory Table for unit testing purposes.

Sample output

```
Test 1
mkPut   [Inventory [key=pens, quantity=9]]
mkGet   [rowkey= pens]
Get Inventory Result :
Result with rowKey pens :  Family - stock : Qualifier - quantity : Value:
9
Print Inventory
Inventory [key=pens, quantity=9]
```

Run the Shopping App on your sandbox or aws cluster:

1. To build labs from Eclipse, Select the project and under the “Run” menu select “Run As” then select “Maven install” . This will create a jar file in the target folder as shown here:



2. Copy the jar file "lab-exercises-shopping-1.0.jar" from the target folder to your sandbox or a cluster node:

   Once you have built the project, upload the jar file to your sandbox or cluster node with scp to your user account. A typical scp command looks like the following : Replace userXX with your userid and *host* with your cluster host IP address.

   `scp lab-exercises-shopping-1.0.jar userXX@host:`

3. Login to the cluster, run the shopping cart application and see what table is created and what data you have:

   Invoke java specifying: your jar file in the classpath , HBase class path, and the name of the main class preceded by the package name. The example below shows you how to execute the **ShoppingCartApp** main class in the package called **shopping** passing the argument **setup**. **java -cp `hbase classpath`** shown in the command below uses the "hbase classpath" utility to set the HBase dependencies in the Java classpath.

   ```
   java -cp `hbase classpath`:./lab-exercises-shopping-1.0.jar
   shopping.ShoppingCartApp setup
   ```

   Sample Output from setup:

```
*************************************************
inventoryTable already exists so deleting it ...
1: Created table InventoryTable with family: stock
 shoppingcartTable already exists so deleting it ...
1: Created table ShoppingcartTableName with family:cartitems
*************************************
------------------------------
```

Inserting rows in Inventory Table:

```
Inserting rows in Inventory Table:
mkPut   [Inventory [key=pens, quantity=9]]
mkPut   [Inventory [key=notepads, quantity=21]]
mkPut   [Inventory [key=erasers, quantity=10]]

printInventoryTable ...
**************************************************
Scan Inventory Results :
Result with rowKey erasers :  Family - stock : Qualifier - quantity : Value: 10
Result with rowKey notepads :  Family - stock : Qualifier - quantity : Value: 21
Result with rowKey pens :  Family - stock : Qualifier - quantity : Value: 9
**************************************************
print Inventorys from Table ...
Inventory [key=erasers, quantity=10]
Inventory [key=notepads, quantity=21]
Inventory [key=pens, quantity=9]
```

Note that at the end of inserting data into Inventory table, here is what it looks like:

Table: **Inventory**

	cf: stock
	quantity
pens	9
notepads	21
eraser s	10

Note: If you see the following warning messages, then set LD_LIBRARY_PATH as shown here.

WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable

INFO security.JniBasedUnixGroupsMappingWithFallback: Falling back to shell based

```
export LD_LIBRARY_PATH=/opt/mapr/hadoop/hadoop-0.20.2/lib/native/Linux-amd64-64
```

Exercise 1.2: Insert and get data in the ShoppingCartDAO class

Complete code for "TODO 2a" to put and get data in Shoppingcart Table

1. Look in the src/test/java directory at shopping.TestShoppingCartSetup .
2. Look for "TODO 2"s. **Uncomment** the 1st unit test **@Test** line.
3. Look in the src/main/java directory at shopping.dao.ShoppingCartDAO.java. Look for "**TODO 2a**"s in the ShoppingCartDAO. Finish code following the TODO comments, to put and get the data in the 'Shoppingcart' table.

Run the 2a junit test

Run the first junit test in shopping.TestShoppingCartSetup , right mouse click on the class and select run as junit test. Observe what the unit test does, and prints out. The unit test uses a MockHBaseTable which is an apache open source in memory Table for unit testing purposes. If you completed the code correctly the test should run green, if it is red you need to correct the code.

The test prints debug information :

```
Running shopping.TestShoppingCartSetup Test 1
mkPut Shoppingcart  [ShoppingCart [rowkey=Mike, pens=1, notepads=2, erasers=3]]
mkGet Shoppingcart key  [Mike]
Get ShoppingCart Result :
Result with rowKey Mike :  Family - items : Qualifier - erasers : Value: 3  Family - items : Qualifier - notepads : Value: 2  Family - items : Qualifier - pens : Value: 1
Print Cart
ShoppingCart [rowkey=Mike, pens=1, notepads=2, erasers=3]
```

The test calls the ShoppingCartDAO to put and get the following data in the 'Shoppingcart' table.

[RowKey: Username]

	Cf: cartitems					
	pens	notepads	erasers			
Mike	1	2	3			

Complete code for "TODO 2b"

Now you will complete the code with comments **//TODO 2b** to scan rows in Shoppingcart Table.

The unit test code calls the ShoppingCartDAO to put the following data in the 'Shoppingcart' table. Then it calls the ShoppingCartDAO to scan and return the data in the table.

[RowKey: Username]

	Cf: cartitems					
	pens	notepads	erasers			
Mike	1	2	3			
Mary	1	2	5			
Adam	5	4	2			

1. Look in the src/test/java directory at shopping.TestShoppingCartSetup . Look for "**TODO 2b**"s. **Uncomment** the 2nd unit test **@Test** line.

   This test calls the ShoppingCartDAO addShoppingCart(String cartId, long pens, long notepads, long erasers) method 3 times to put 3 rows in the shopping cart table. Then it calls the ShoppingCartDAO getShoppingCarts() method to scan and return all of the rows from the table.

2. Look in the src/main/java directory at shopping.dao.ShoppingCartDAO.java. Look for "**TODO 2b**"s in the ShoppingCartDAO. You need to **finish** the code following the TODO comments.

Run the 2b junit test

1. Run the 2b junit test in shopping, TestShoppingCartSetup , right mouse click on the TestShoppingCartSetup class and select **run as junit test**. Observe what the unit test does, and prints out. If you completed the code correctly the test should run green, if it is red you need to correct the code.

   The test prints debug information :

   ```
   Test 2
   mkPut Shoppingcart [ShoppingCart [rowkey=Mike, pens=1, notepads=2, erasers=3]]
   mkPut Shoppingcart [ShoppingCart [rowkey=Mary, pens=1, notepads=2, erasers=5]]
   mkPut Shoppingcart [ShoppingCart [rowkey=Adam, pens=5, notepads=4, erasers=2]]
   mkScan Shoppingcart
   ```

Scan ShoppingCart Results:
Result with rowKey Adam : Family - items : Qualifier - erasers : Value: 2 Family - items : Qualifier - notepads : Value: 4 Family - items : Qualifier - pens : Value: 5
Result with rowKey Mary : Family - items : Qualifier - erasers : Value: 5 Family - items : Qualifier - notepads : Value: 2 Family - items : Qualifier - pens : Value: 1
Result with rowKey Mike : Family - items : Qualifier - erasers : Value: 3 Family - items : Qualifier - notepads : Value: 2 Family - items : Qualifier - pens : Value: 1
ShoppingCart [rowkey=Adam, pens=5, notepads=4, erasers=2]
ShoppingCart [rowkey=Mary, pens=1, notepads=2, erasers=5]
ShoppingCart [rowkey=Mike, pens=1, notepads=2, erasers=3] Family - cartitems : Qualifier - pens : Value: 3

Run the 2c junit test

1. **Uncomment** the 2c unit test **@Test** line. Then run the junit test. TestShoppingCartSetup , right mouse click on the class and select run as junit test. This test calls the ShoppingCartApp.saveShoppingCartData(dao) and ShoppingCartApp.printShoppingcartTable(dao) methods, the same methods that get called when you run **java -cp `hbase classpath`:./lab-exercises-shopping-1.0. shopping.ShoppingCartApp initshopping.** The test should print out debug information

Run the code on a sandbox or cluster

1. Save any modified files.
2. Build the project using Maven: Select *project lab-exercises-shopping -> Run As ->Maven Install*
3. Copy the jar file *lab-exercises-shopping-1.0.jar* from the target folder to the cluster
4. Login to the cluster, run the shopping cart application:
5. **`java -cp `hbase classpath`:./lab-exercises-shopping-1.0.jar shopping.ShoppingCartApp initshopping`**
6. See what tables are created, data saved, and what data you have. The code prints debug information :

mkPut Shoppingcart [ShoppingCart [rowkey=Mike, pens=1, notepads=2, erasers=3]]
mkPut Shoppingcart [ShoppingCart [rowkey=Mary, pens=1, notepads=2, erasers=5]]
mkPut Shoppingcart [ShoppingCart [rowkey=Adam, pens=5, notepads=4, erasers=2]]
Scan Shopping Cart Table
mkScan Shoppingcart
Scan ShoppingCart Results:
Result with rowKey Adam : Family - items : Qualifier - erasers : Value: 2 Family - items : Qualifier - notepads : Value: 4 Family - items : Qualifier - pens : Value: 5
Result with rowKey Mary : Family - items : Qualifier - erasers : Value: 5 Family - items : Qualifier - notepads : Value: 2 Family - items : Qualifier - pens : Value: 1

Result with rowKey Mike : Family - items : Qualifier - erasers : Value: 3 Family - items : Qualifier - notepads : Value: 2 Family - items : Qualifier - pens : Value: 1
****
print Shoppingcart Table
ShoppingCart [rowkey=Adam, pens=5, notepads=4, erasers=2]
ShoppingCart [rowkey=Mary, pens=1, notepads=2, erasers=5]
ShoppingCart [rowkey=Mike, pens=1, notepads=2, erasers=3]

Complete code for "TODO 3a"

Now you will Finish code in ShoppingCartDAO to Delete a user entry from the 'Shoppingcart' table by completing code marked "**TODO 3a**":

1. Look in the src/test/java directory at shopping.TestShoppingCartDelete . Look for "**TODO 3a**"s. **Uncomment** the 3a **@Test** line. This test calls the ShoppingCartDAO deleteShoppingCart(String cartId) method to delete a shopping cart row in the shopping cart table.

2. Look in the src/main/java directory at shopping.dao.ShoppingCartDAO.java. Look for "**TODO 3a**"s in the ShoppingCartDAO. Finish the code following the **TODO 3a** comments.

Run the 3a unit test

1. Uncomment the junit TestShoppingCartDelete test 3a **@Test** line.

2. Run the junit test, right mouse click on the class and select run as junit test. This test calls the ShoppingCartApp.deleteUserCart(dao, name) and ShoppingCartApp.printShoppingcartTable(dao) methods, the same methods that get called when you run *java -cp `hbase classpath`:./lab-exercises-shopping-1.0.jar shopping.ShoppingCartApp delete name.* The test should print out debug information.

Run the Shopping App on the cluster

1. Save any modified files.

2. Build the project using Maven: Select project lab-exercises-shopping -> Run As ->Maven Install

3. Copy the jar file "lab-exercises-shopping-1.0.jar" from the target folder to the cluster

4. Login to the cluster, run the shopping cart application with parameter **delete Mike**:

```
java -cp `hbase classpath`:./lab-exercises-shopping-1.0.jar
shopping.ShoppingCartApp  delete Mike
```

Lab 2: Building Shoppingcart applications using HBase API

Lab Overview

This lab will show how to use the HTable put(list) and HTable batch() to put multiple rows with one rpc call.

This lab will also simulate 'transaction' functionality where we need to process data in two different tables using HBase Java APIs.

Exercise	Required	Duration
2.1 Import and build the "lab-exercises-shopping2" project	Yes	20 min
2.2: Working with Shoppingcart Application put list and batch	Yes	20 min
2.3 Working with Shoppingcart Application checkout		

Working with the Shoppingcart application to implement checkout functionality

As part of this application, we have two tables as shown in the following image.

Table: **Inventory** [RowKey: items]

	cf: stock
	quantity
pens	10
notepads	21
erasers	10

Table: **Shoppingcart** [RowKey: Username]

	Cf: items					
	pens	notepads	erasers			
John	3	4	5			
Mike	1	2	3			
Mary	1	2	5			
Adam	5	4	0			

The next objective of this lab is to implement a 'checkout' operation, where when one of the users with data in the 'shoppingcart' table wants to checkout the items in the cart.

```
Assumptions made for this applications are:

There is one Shoppingcart per user(customer) at any given time.

When the user places an order / checkout, do the following:
     1) Get the items for the user from the shoppingcart.
     2) For each item for the user, get inventory and make sure we have enough
        quantity for the user.
     3) Once we have seen that there is enough quantity in Inventory table, we
        can use CheckAndPut to simulate a transaction, by manipulating the
        data in one call in one row. Here we add a new column for the user as
        part of this process and reserve the quantity under the user column.

     4) Remove the shoppingcart entry for the user.
```

Let's say Mike wants to checkout the items, the Inventory table will look like the following after the checkout operation.

Table: **Inventory**

	cf: stock	
	quantity	Mike
pens	~~10~~ 9	1
notepads	~~21~~ 19	2
erasers	~~10~~ 7	3

Exercise 2.1: Import and build "lab-exercises-shopping2" project

Import project and build

If you have not already done so, use the following steps to import the project "lab-exercises-shopping2" into Eclipse and build project using Maven:

1. Import the **lab-exercises-shopping2** project into Eclipse.
2. Build the project using Maven: Select project lab-exercises-shopping -> Run As ->Maven Install

   (See section on Importing Projects into Eclipse)

Exercise 2.2: Work with Shoppingcart Application put list and batch

Use a put List to put data in the Inventory table

1. Look in the src/test/java directory for **shopping.TestInventoryList** .
2. Look for "TODO 1a".

3. **Uncomment** the 1a unit test **@Test** line.

   This test creates a list of put then calls InventoryDAO putInventoryList() to put this list of puts in the table.

1. Look for "**TODO 1a**"s in **TestInventoryList** and InventoryDAO. **Finish** the code following the comments.

2. Run the 1a junit test

   - Right mouse click on the class and select run as junit test.
   - Observe what the unit test does, and prints out. If you completed the code correctly the test should run green, if it is red you need to correct the code.

Use batch to put data in the Inventory table

1. Look in the src/test/java directory for **shopping.TestInventoryList** .

   - Look for "TODO 1b"s. **Uncomment** the 1b test **@Test** line. This test creates a list of put then calls InventoryDAO putInventoryBatch() to put this list of puts in the table.

2. Look for "**TODO 1b**"s in InventoryDAO, **Finish** the code following the comments.

Run the 1b junit test.

1. Right mouse click on the class and select run as junit test.

2. Observe what the unit test does, and prints out. If you completed the code correctly the test should run green, if it is red you need to correct the code.

Run the 1c unit test.

1. **Uncomment** the 1c unit test **@Test** line.

2. Run the junit test

3. Right mouse click on the class and select run as junit test.

This test calls the ShoppingCartApp.initInventoryTableList(dao) and ShoppingCartApp.printShoppingcartTable(dao) methods, the same methods that get called when you run **java -cp `hbase classpath`:./lab-exercises-shopping2-1.0. shopping.ShoppingCartApp setuplist.** The test should print out debug information.

Run the code on your sandbox or a cluster

Use the following steps to run the Shopping App on your sandbox or cluster.

1. Save any modified files.
2. Build the project using Maven: Select project lab-exercises-shopping -> Run As ->Maven Install
3. Copy the jar file "lab-exercises-shopping2-1.0.jar" from the target folder to the cluster
4. Login to the cluster
5. Run the shopping cart application

```
java -cp `hbase classpath`:./lab-exercises-shopping2-1.0.jar
shopping.ShoppingCartApp setuplist
```

6. See what tables are created, data saved, and what data you have.

Exercise 2.3: Work with Shoppingcart Application checkout

Use the following steps for this exercise

Finish check out code

7. Look in the src/test/java directory for **shopping.TestCheckout**.
8. Look for "TODO 2a"s. **Uncomment** the 2a test **@Test** line.
9. Look for "**TODO 2a**"s in the Test ShoppingCartApp and InventoryDAO.
    a. **Finish** ShoppingCartApp checkout() and InventoryDAO checkout().

Run the 2a junit test

1. Right mouse click on the class and select run as junit test.

2. Observe what the unit test does, and prints out. If you completed the code correctly the test should run green, if it is red you need to correct the code.

```
Test Checkout
Print Inventory
Inventory [stockId=erasers, quantity=10]
Inventory [stockId=notepads, quantity=21]
Inventory [stockId=pens, quantity=9]
Checkout for CartId :  Mike
ShoppingCart [rowkey=Mike, pens=1, notepads=2, erasers=3]
--checkout Inventory stockId pens cartId Mike quantity 1
--checkout success: Changed pens quantity 9 to 8
--added column for cartId Mike quantiy 1
Pens Inventory  rowKey pens :  Family - stock : Qualifier - Mike : Value: 1
   Family - stock : Qualifier - quantity : Value: 8
--checkout Inventory stockId notepads cartId Mike quantity 2
--checkout success: Changed notepads quantity 21 to 19
--added column for cartId Mike quantiy 2
Notepads Inventory row Result with rowKey notepads :  Family - stock : Qualifier - Mike : Value: 2
  Family - stock : Qualifier - quantity : Value: 19
--checkout Inventory stockId erasers cartId Mike quantity 3
--checkout success: Changed erasers quantity 10 to 7
--added column for cartId Mike quantiy 3
erasers Inventory row Result with rowKey erasers :  Family - stock : Qualifier - Mike : Value: 3
  Family - stock : Qualifier - quantity : Value: 7
```

Table: **Inventory**

	cf: stock	
	quantity	Mike
Pens	~~10~~ 9	1
notepads	~~21~~ 19	3
erasers	~~10~~ 7	2

Run the Application on a sandbox or cluster

1. Copy the jar file "lab-exercises-shopping2-1.0.jar" from the target folder to the cluster

2. Login to the cluster, run with argument setup to put data in the inventory table

```
java -cp `hbase classpath`:./lab-exercises-shopping2-1.0.jar
shopping.ShoppingCartApp setup
```

3. Run with argument initshopping to put data in the shopping cart

```
java -cp `hbase classpath`:./lab-exercises-shopping2-1.0.jar
shopping.ShoppingCartApp initshopping
```

The output should look like the following when you run the **shoppingcart.ShoppingCartApp** program with **initshopping** argument.

```
------------------------------
 Inserting rows in Inventory Table:
Saved Inventory Table row with key [pens] with quantity = 9
Saved Inventory Table row with key [notepads] with quantity = 21
Saved Inventory Table row with key [erasers] with quantity = 10
printInventoryTable ...
************************************
Scan Inventory Results :
Result with rowKey erasers :  Family - stock : Qualifier - quantity : Value: 10
Result with rowKey notepads :  Family - stock : Qualifier - quantity : Value: 21
Result with rowKey pens :  Family - stock : Qualifier - quantity : Value: 9
------------------------------
 Inserting rows in shoppingcart Table:
Saved Shoppingcart Table row with key [Mike]
printShoppingcartTable
**********************************
Scan results for Shoppingcart Table:
ShoppingCart [rowkey=Mike, pens=1, notepads=2, erasers=3]
**************************************
```

Run with arguments checkout userid

```
java -cp `hbase classpath`:./lab-exercises-shopping2-1.0.jar
shopping.ShoppingCartApp checkout Mike
```

```
Test Checkout

Print Inventory

Inventory [stockId=erasers, quantity=10]

Inventory [stockId=notepads, quantity=21]

Inventory [stockId=pens, quantity=9]

Checkout for CartId :  Mike

ShoppingCart [rowkey=Mike, pens=1, notepads=2, erasers=3]

--checkout Inventory stockId pens cartId Mike quantity 1

--checkout success: Changed pens quantity 9 to 8

--added column for cartId Mike quantiy 1

Pens Inventory  rowKey pens :  Family - stock : Qualifier - Mike : Value: 1

   Family - stock : Qualifier - quantity : Value: 8

--checkout Inventory stockId notepads cartId Mike quantity 2

--checkout success: Changed notepads quantity 21 to 19

--added column for cartId Mike quantiy 2

Notepads Inventory row Result with rowKey notepads :  Family - stock : Qualifier - Mike : Value: 2

 Family - stock : Qualifier - quantity : Value: 19

--checkout Inventory stockId erasers cartId Mike quantity 3

--checkout success: Changed erasers quantity 10 to 7

--added column for cartId Mike quantiy 3

erasers Inventory row Result with rowKey erasers :  Family - stock : Qualifier - Mike : Value: 3

 Family - stock : Qualifier - quantity : Value: 7
```

Table: **Inventory**

	cf: stock	
	quantity	Mike
Pens	~~10~~ 9	1
notepads	~~21~~ 19	3
erasers	~~10~~ 7	2

Lab 3: Java applications using HBase Java Admin API

Lab Overview

The purpose of this lab is to show you how to use HBase Java Admin APIs to create HBase tables and change some default properties. This lab consists of the one exercise with several steps.

Exercise	Required	Duration
3.1 Working with LabAdminAPI in lab-exercises project	Yes	60 min

Lab Procedure

We will identify some default values set for MaxVersions, MinVersions, TimeToLive (TTL), etc. We will change some of Table properties for MaxVersions, TTL etc. We will presplit a table in this lab exercise.

Information on HBase Java Admin API

There are three main classes that we will use in this lab:

HBaseAdmin

This class provides an interface to manage HBase database table metadata + general administrative functions. Use HBaseAdmin to create, drop, list, enable, and disable tables. Use it to add and drop table column families.

Some of the key functionality this class provides are:

- **createTable**(HTableDescriptor desc) Creates a new table.
- **createTable**(HTableDescriptor desc, byte[][] splitKeys) Creates a new table with an initial set of empty regions defined by the specified split keys.
- **createTable**(HTableDescriptor desc, byte[] startKey, byte[] endKey, int numRegions) Creates a new table with the specified number of regions.
- **compact**(...)Compact a table or a column family within a table or an individual region.
- **disableTable**(byte[] tableName) Disable table and wait on completion.
- **deleteTable**(byte[] tableName) Deletes a table.

- **listTables**() List all the userspace tables.
- **modifyColumn**() Modify an existing column family on a table.
- **tableExists**(byte[] tableName) returns true if it exists.
- Other methods: split(), snapshot(), move(), modifyTable()

HTableDescriptor

This class contains the details about an HBase table such as the descriptors of all the column families, is the table a catalog table, -ROOT- or .META. , is the table is read only, the maximum size of the memstore, when the region split should occur, coprocessors associated with it etc.

HColumnDescriptor

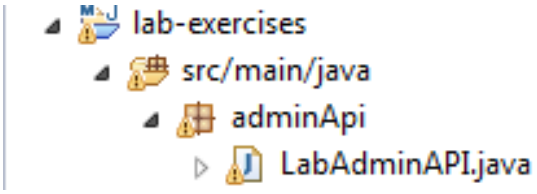
This class contains information about a column family such as the number of versions, compression settings, etc. It is used as input when creating a table or adding a column.

The diagram below shows the table **adminlabtrades** schema, with column families **trades** and **stats**.

RowKey	trades:price	trades:vol	stats: avgprice
AMZN	12.34	1000	
CSCO	1.23	3000	
GOOG	2.34	1000	

Exercise 3.1: Working with LabAdminAPI in lab-exercises project

We will import the exercise project "**lab-exercises**" into Eclipse, review the code, build the project using Maven, upload the jar file to the cluster and run in this exercise.



Use the following steps for this exercise.

Review and complete the code for TODO 1

1. Complete the implementation

In the lab-exercises project, adminApi.LabAdminAPI class , Complete code following all comments marked // TODO 1

```
78      public static void setupTables() throws IOException {
79
80          Configuration conf = HBaseConfiguration.create();
81          HBaseAdmin admin = new HBaseAdmin(conf);
82
83          // Create table adminTradesTable to store the stock trades
84          // with Column families trades & stats
85          if (admin.tableExists(adminTradesTableName)) {
86              System.out.println(" table already exists... so deleting it.");
87              admin.disableTable(adminTradesTableName);
88              admin.deleteTable(adminTradesTableName);
89          }
90
91          // create the tableDescriptor
92          HTableDescriptor tableDescriptor = new HTableDescriptor(
93                  TableName.valueOf (  adminTradesTableName));
94          // TODO 1 : Complete implementation
95          // TODO 1 add Column families trades & stats to tableDescriptor
96
```

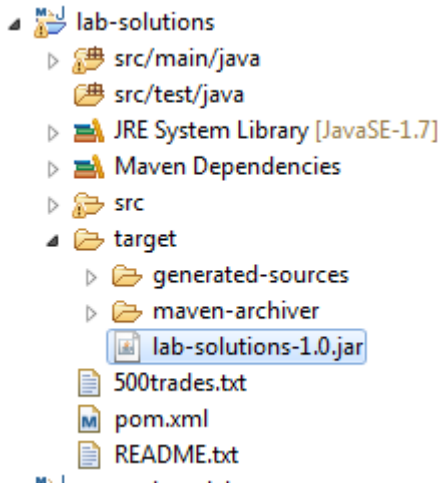
```
HTableDescriptor tableDescriptor = new

                                HTableDescriptor(adminTradesTableName);

tableDescriptor.addFamily(new HColumnDescriptor(tradesCF));

tableDescriptor.addFamily(new HColumnDescriptor(statsCF));

admin.createTable(tableDescriptor);



HTable adminTradesTable = new HTable(conf, adminTradesTableName);
```

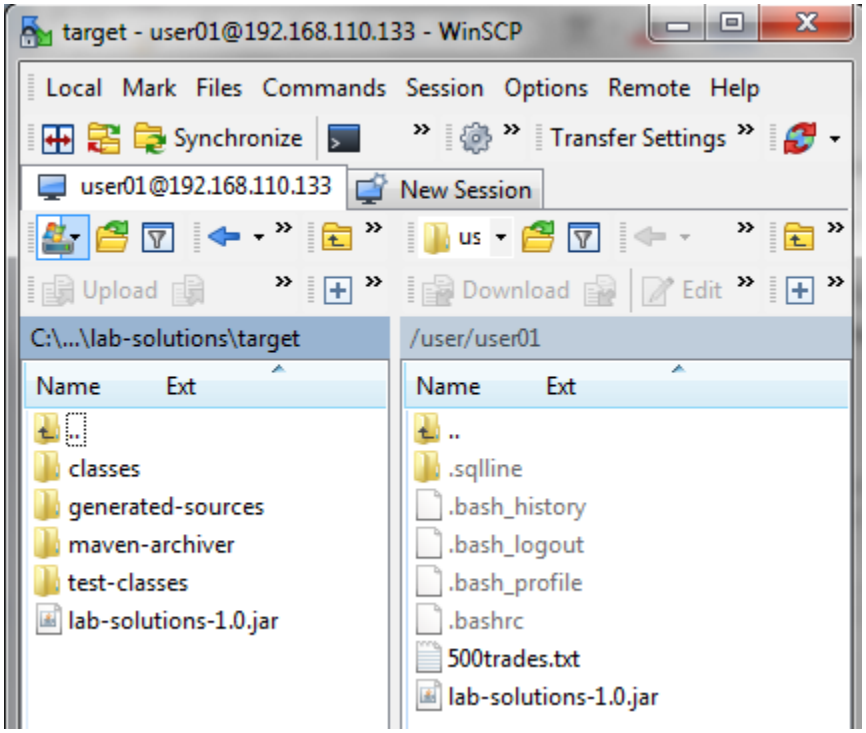
Build the Project, Copy the jar, Run the Application

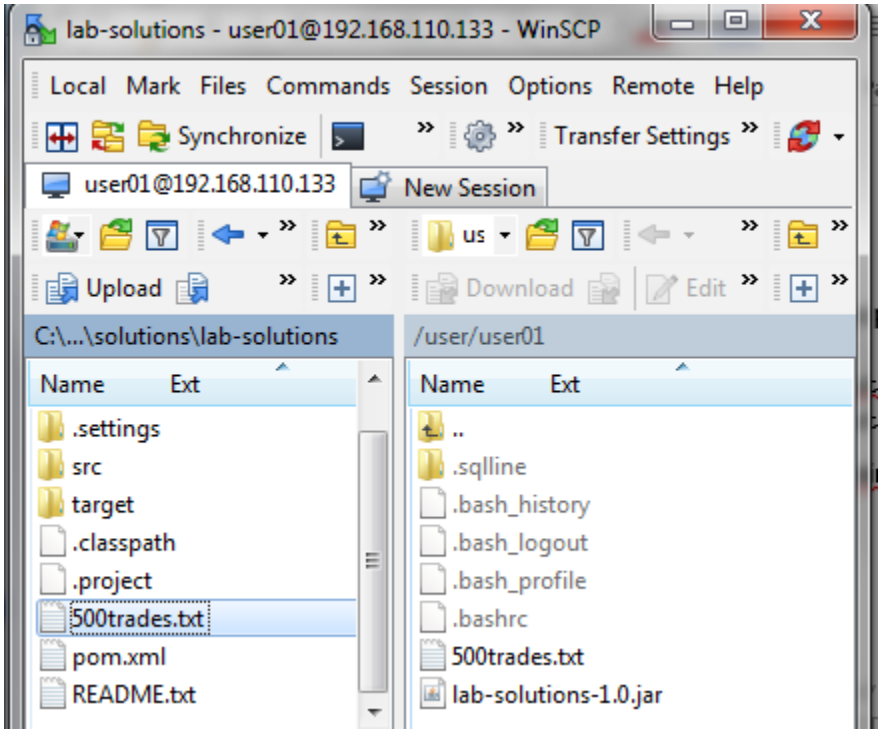
1. Build the project using Maven: Select project lab-exercises-shopping -> Run As ->Maven Install

   (See section on Importing Projects into Eclipse)

   Copy the jar file “lab-exercises-1.0.jar” from the target folder to the sandbox or cluster using scp. Also copy the data file 500trades.txt from the project directory. (See section in chapter 0 copying files to the sandbox or cluster with scp)







2. Login to the cluster and Run the application with arg setup

```
java -cp `hbase classpath`:./lab-exercises-1.0.jar
adminApi.LabAdminAPI setup
```

The output when you run the **adminApi.LabAdminAPI** program with **setup** argument should resemble the following.

```
Settingup adminlabtrades Table ...
**************************************************
1: Created table adminTradesTable...
Initializing adminTradesTable in initTradesTable() ...
Putting trade: AMZN: 1000 shares at $304.66 at 2013.10.10 02:12:43
Putting trade: AMZN: 1600 shares at $303.91 at 2013.10.10 02:29:24
Putting trade: AMZN: 1900 shares at $304.82 at 2013.10.10 02:46:05
Putting trade: GOOG: 7660 shares at $866.73 at 2013.10.10 07:44:37
Putting trade: GOOG: 7993 shares at $866.22 at 2013.10.10 07:52:58
```

```
Putting trade: GOOG: 8326 shares at $865.71 at 2013.10.10 08:01:19
Putting trade: GOOG: 8659 shares at $865.20 at 2013.10.10 08:09:40
Putting trade: GOOG: 8992 shares at $864.69 at 2013.10.10 08:18:01
Putting trade: GOOG: 9325 shares at $864.18 at 2013.10.10 08:26:22
Putting trade: ORCL: 9325 shares at $32.18 at 2013.10.10 08:26:22
Putting trade: ZNGA: 9000 shares at $4.18 at 2013.10.10 08:26:2

In printTradesTable ...
**********************************
Scan results for Table:
Result with rowKey AMZN
   Family - trades : Qualifier - price : Value(long): 304.82
   Family - trades : Qualifier - price : Value(long): 303.91
   Family - trades : Qualifier - price : Value(long): 304.66
   Family - trades : Qualifier - vol : Value(long): 1900
   Family - trades : Qualifier - vol : Value(long): 1600
   Family - trades : Qualifier - vol : Value(long): 1000
Result with rowKey CSCO
   Family - trades : Qualifier - price : Value(long): 22.99
   Family - trades : Qualifier - price : Value(long): 22.98
   Family - trades : Qualifier - price : Value(long): 22.96
Result with rowKey ZNGA
   Family - trades : Qualifier - price : Value(long): 4.18
   Family - trades : Qualifier - vol : Value(long): 9000
```

Review and complete the code for TODO 2

1. Complete the implementation to store and retrieve max versions:

In the lab-exercises project, adminApi.LabAdminAPI class , Complete code following all comments marked // TODO 2 (also look in the printTradesTable method)

Notice that you have to specify how many versions to store when you create a table, and how many versions to read when you perform a get or scan on a table.

2. Build the project using Maven: Select project lab-exercises-shopping -> Run As ->Maven Install

   Copy the jar file "lab-exercises-1.0.jar" from the target folder to the sandbox or cluster using scp.

Run the program with arg setupmaxversions

1. Use arg 'setupmaxversions'

```
java -cp `hbase classpath`:./lab-exercises-1.0.jar
adminApi.LabAdminAPI setupmaxversions
```

2. Observe the output.

{NAME => 'stats', DATA_BLOCK_ENCODING => 'NONE', BLOOMFILTER => 'NONE', REPLICATION_SCOPE => '0', VERSIONS => ' **2147483647**', TTL => '2147483647', MIN_VERSIONS => '0', KEEP_DELETED_CELLS => 'false', BLOCKSIZE => '65536', ENCODE_ON_DISK => 'true', IN_MEMORY => 'false', BLOCKCACHE => 'true'}

Max versions = 2147483647

Min versions = 0

TTL = 2147483647

Result with rowKey AMZN : Family - trades : Qualifier - price : Value(long): 304.82 : Value(long): 303.91 : Value(long): 304.66 : Qualifier - vol : Value(long): 1900 : Value(long): 1600 : Value(long): 1000

Result with rowKey CSCO : Family - trades : Qualifier - price : Value(long): 22.99 : Value(long): 22.98 : Value(long): 22.96 : Value(long): 22.94 : Value(long): 22.92 : Value(long): 22.9 : Value(long): 22.86 : Value(long): 22.84 : Value(long): 22.82 : Value(long): 22.8 : Value(long): 22.78 : Value(long): 22.76 : Qualifier - vol : Value(long): 6328 : Value(long): 5995 : Value(long): 5662 : Value(long): 5329 : Value(long): 100 : Value(long): 500000 : Value(long): 500 : Value(long): 600 : Value(long): 300 : Value(long): 1000 : Value(long): 250 : Value(long): 2300

Result with rowKey GOOG : Family - trades : Qualifier - price : Value(long): 864.18 : Value(long): 864.69 : Value(long): 865.2 : Value(long): 866.22 : Value(long): 866.73 : Value(long): 867.24 : Qualifier - vol : Value(long): 9325 : Value(long): 8992 : Value(long): 8659 : Value(long): 7993 : Value(long): 7660 : Value(long): 7327

Result with rowKey ORCL : Family - trades : Qualifier - price : Value(long): 32.18 : Qualifier - vol : Value(long): 9325

Result with rowKey ZNGA : Family - trades : Qualifier - price : Value(long): 4.18 : Qualifier - vol : Value(long): 9000

Use the HBase shell to get and scan the table

```
$ hbase shell
```

```
hbase(main):011:0> get '/user/user01/adminlabtrades', 'AMZN', {COLUMNS=>['trades:price']}
COLUMN                CELL
 trades:price         timestamp=1432658951600, value=C\x98h\xF6
1 row(s) in 0.0030 seconds


hbase(main):010:0> get '/user/user01/adminlabtrades', 'AMZN', {COLUMNS=>['trades:price'], VERSIONS => 3}
COLUMN                CELL
 trades:price         timestamp=1432658951600, value=C\x98h\xF6
 trades:price         timestamp=1432658951599, value=C\x97\xF4{
 trades:price         timestamp=1432658951597, value=C\x98T{
3 row(s) in 0.0350 seconds
```

Notice that you have to specify how many versions to store when you create a table, and how many versions to read when you perform a get or scan on a table.

Presplit the table

1. PreSplit the table at 'I' & 'P':

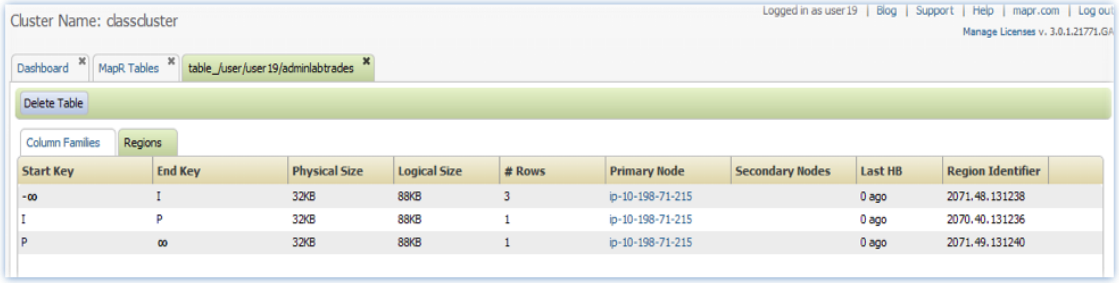
   In the lab-exercises project, adminApi.LabAdminAPI class , Complete the code following all comments marked // TODO 3. Build the project, copy the jar to the cluster using scp.

2. Run the program with arg '**presplit**'

```
java -cp `hbase classpath`:./lab-exercises-1.0.jar
adminApi.LabAdminAPI presplit
```

3. Observe the output.

You should observe the following in MapR Control System after you complete the implementation.



Optional Activity

- Complete the implementation in 'listTables()' to list all the tables in your home directory.

  In the lab-exercises project, adminApi.LabAdminAPI class , Complete the code following all comments marked // TODO 4. Build the project, copy the jar to the cluster using scp.

```
HTableDescriptor[] allTables = TODO

System.out.println("Printing all tables...");

for (HTableDescriptor tabledesc : allTables) {

      System.out.println(tabledesc.getNameAsString());
}
```

- Run the program with arg '**listtables**'

```
java -cp `hbase classpath`:./lab-exercises-1.0.jar
adminApi.LabAdminAPI listtables
```

The output should resemble the following.

```
Printing all tables...

/user/user01/adminlabtrades

…
```

Optional Activity

Explore the other functions for working with 'HBaseAdmin' like compact, balancer, deleteColumn etc…

Lab 4: Java applications using Advanced HBase Java API

Lab Overview

The first objective of this lab is to use advanced HBase Java APIs. We will apply the necessary filters on the server and then send only the required data to the client.

The second objective of this lab exercise is to use Increment instead of checkAndPut to manage a transaction of checkout inventory items. This simplifies in case of conflicts as the programmer does not have to get the value and check it before updating it.

Exercise	Required	Duration
4.1: Java applications using Advanced HBase Java API with Filters	Yes	30 min
4.2 Java applications using Advanced HBase Java API using Increment	Yes	50 min

Information on HBase Java API for Filters

Package org.apache.hadoop.hbase.filter provides row-level filters applied to HRegion scan results during calls to ResultScanner.next().

FamilyFilter: This filter is used to filter based on the column family. It takes an operator (equal, greater, not equal, etc) and a byte [] comparator for the column family portion of a key.

RowFilter: This filter is used to filter based on the key. It takes an operator (equal, greater, not equal, etc) and a byte [] comparator for the row, and column qualifier portions of a key.

QualifierFilter: This filter is used to filter based on the column qualifier. It takes an operator (equal, greater, not equal, etc) and a byte [] comparator for the column qualifier portion of a key.

ValueFilter: This filter is used to filter based on column value. It takes an operator (equal, greater, not equal, etc) and a byte [] comparator for the cell value. This filter can be wrapped with WhileMatchFilter and SkipFilter to add more control. To test the value of a single qualifier when scanning multiple qualifiers, use SingleColumnValueFilter.

FilterList: Implementation of Filter that represents an ordered List of Filters which will be evaluated with a specified boolean operator FilterList.Operator.MUST_PASS_ALL (!AND) or FilterList.Operator.MUST_PASS_ONE (!OR). Since you can use Filter Lists as children of Filter Lists, you can create a hierarchy of filters to be evaluated. Defaults to FilterList.Operator.MUST_PASS_ALL.

CompareFilter: This is a generic filter to be used to filter by comparison. It takes an operator (equal, greater, not equal, etc) and a byte [] comparator.

SingleColumnValueFilter: This filter is used to filter cells based on value. It takes a CompareFilter.CompareOp operator (equal, greater, not equal, etc), and either a byte [] value or a WritableByteArrayComparable.

If we have a byte [] value then we just do a lexicographic compare. For example, if passed value is 'b' and cell has 'a' and the compare operator is LESS, then we will filter out this cell (return true).

You must also specify a family and qualifier. Only the value of this column will be tested. When using this filter on a Scan with specified inputs, the column to be tested should also be added as input (otherwise the filter will regard the column as missing).

Program Structure

Start with the following Java classes:

- **`CreateTable`** – A driver class to create a table and populate it with test data
  - **`CreateTablesUtil`** – A helper class with methods used by CreateTable
- **`FilterTradesTall`** – A driver class to filter trades for the Tall table
- **`FilterTradesFlat`** – A driver class to filter trades for the Flat table
- **`Trade`** – A Java object that holds data for a single trade
- **`TradeDAO`** – A Data Access Object (DAO) interface that hides details of schema implementation
  - **`TradeDAOFlat`** – The flat-wide implementation of the TradeDAO. This code is provided for you.
  - **`TradeDAOTall`** – The tall-narrow implementation of the TradeDAO. You will write code for this class.

Use the following junit test classes.

- TestFilterTall: junit tests for filtering with the tall table
- TestFilterFlat:junit tests for filtering with the flat table

Exercise 4.1: Java applications using Advanced HBase Java API with Filters

We will import the exercise project "lab-exercises-filter" into Eclipse, review the code, build the project using Maven, finish code, run unit tests, upload the jar file to the cluster and run.

Filters for the Tall Narrow schema

Every row represents one trade in this tall schema. There is only one column family, with 2 columns to store Price and Volume values. The composite rowkey is formed by combining the stock symbol and a reversed timestamp. (`Long.MAX_VALUE - timestamp`). For example: AMZN_98618600888. Because the rowkey contains timestamp data, the trade time is not stored anywhere else in the table.

The following diagram shows the tall table schema, including an example of a few data points.

RowKey: SYM_TIME	PRICE (long)	VOL (long)
CFs:	CF1	
AMZN_98618600888	12.34	1000
AMZN_98618600777	12.00	2000
CSCO_98618600666	1.23	3000
exactlyGOOG_9861860555	2.34	1000

RowFilter: This filter is used to filter based on the key. It takes an operator (equal, greater, not equal, etc) and a byte[] comparator for the row key.

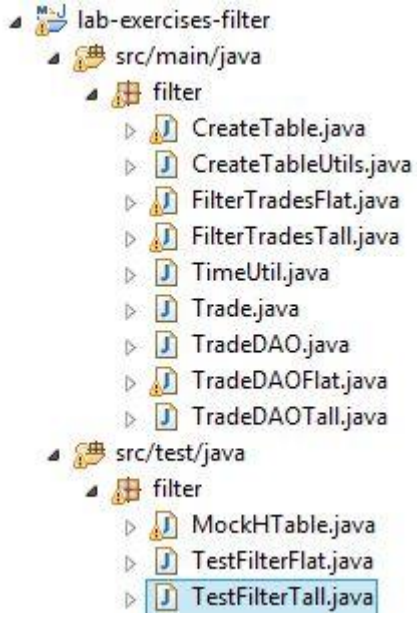
The following is an example of using the RowFilter to filter Row keys that contain "986186"

```
Scan scan = new Scan();

Filter filter = new RowFilter(CompareFilter.CompareOp.EQUAL,
      new SubStringComparator("986186")
    scan.setFilter(filter);
    ResultScanner  scanner = table.getScanner(scan);
    for (Result res : scannerr) {
      System.out.println(res);
    }
    scanner.close();
```

This code example returns all rows with row keys containing, "986186", which is all in this small example.

Finish code and run unit test

1. Import the exercise project "lab-exercises-filter" into Eclipse
2. Build the project using Maven: Select project lab-exercises -> Run As ->Maven Install



Perform the following tasks:

Test 1

1. Navigate to src/test/java filter.TestFilterTall .
2. Uncomment the first test @Test.

The first test scans without a filter and prints the results.

3. Run the TestFilterTall test
4. Right mouse click on the class
5. Select run as junit test.
6. Observe the output.

```
Storing the test data set...
Put trade: GOOG: 7327 shares at $867.24 at 2013.10.10 10:36:16
Put key: GOOG_9223370655438999807 family: CF1 columns: price 86724 vol 7327
Put trade: GOOG: 8327 shares at $767.24 at 2013.10.10 10:36:15
Put key: GOOG_9223370655439000807 family: CF1 columns: price 76724 vol 8327
Put trade: AMZN: 60071 shares at $600.71 at 2013.10.10 11:01:19
```

```
Put key: AMZN_9223370655437496807 family: CF1 columns: price 60071 vol 60071

Put trade: CSCO: 8326 shares at $500.71 at 2013.10.10 07:14:39

Put key: CSCO_9223370655451096807 family: CF1 columns: price 50071 vol 8326

 Test No Filter

 Scan No Filter

**********************************

Scan results for Table without any filters:

Scan Result

 RowKey AMZN_9223370655437496807

 Family : CF1 Column : price , Price : 600.71

 Family : CF1 Column : vol,  Volume : 60071

Scan Result

 RowKey CSCO_9223370655451096807

 Family : CF1 Column : price , Price : 500.71

 Family : CF1 Column : vol,  Volume : 8326

Scan Result

 RowKey GOOG_9223370655438999807

 Family : CF1 Column : price , Price : 867.24

 Family : CF1 Column : vol,  Volume : 7327

Scan Result

 RowKey GOOG_9223370655439000807

 Family : CF1 Column : price , Price : 767.24

 Family : CF1 Column : vol,  Volume : 8327

Printing 4 trades.

AMZN: 60071 shares at $600.71 at 2013.10.10 11:01:19
```

```
CSCO: 8326 shares at $500.71 at 2013.10.10 07:14:39

GOOG: 7327 shares at $867.24 at 2013.10.10 10:36:16

GOOG: 8327 shares at $767.24 at 2013.10.10 10:36:15
```

Test 2

1. Navigate to src/test/java filter TestFilterTall.
2. Uncomment the second test @Test.
3. Locate TODO 2 in the test in FilterTradesTalll in TradeDAOTall.
4. Finish the code as instructed.

The following is an example of using the RowFilter to filter Row keys that contain “GOOG”;

```
Scan scan = new Scan;
Filter filter = new RowFilter(CompareFilter.CompareOp.EQUAL,
      new SubStringComparator(“GOOG”)
scan.setFilter(filter);
ResultScanner  scanner = table.getScanner(scan);
for (Result res : scanner) {
      System.out.println(res);
}
scanner.close();
```

5. Run the TestFilterTall test: Right mouse click on the class. Select run as junit test.

If the test runs correctly you will observe the following output:

```
Storing the test data set...

Same as above

******filter row keys containing string GOOG

**********************************

Scan results for Table with filters:
```

```
Scan Result
 RowKey GOOG_9223370655438999807
 Family : CF1 Column : price , Price : 867.24
 Family : CF1 Column : vol,  Volume : 7327
Scan Result
 RowKey GOOG_9223370655439000807
 Family : CF1 Column : price , Price : 767.24
 Family : CF1 Column : vol,  Volume : 8327
```

6. Navigate to src/test/java filter.TestFilterTall.
7. Uncomment the test 2b @Test.
8. Run the TestFilterTall test : Right mouse click the class, Select run as junit test.
9. Observe the output

Test 3

1. Navigate to src/test/java filter.TestFilterTall.
2. Uncomment the 3rd @Test.
3. Locate TODO 3 in the test, in FilterTradesTall.
4. Finish the code as instructed.
5. Run the TestFilterTall test: Right mouse click on the class , Select run as junit test.
6. Observe the output

```
Test PrefixFilter
 Prefix filter GOOG
**********************************
Scan results for Table with filters:
Scan Result
```

```
RowKey GOOG_9223370655438999807 , price : 867.24, vol : 7327

Scan Result

RowKey GOOG_9223370655439000807 , price : 767.24, vol : 8327
```

Test 4

1. Navigate to src/test/java filter.TestFilterTall.
2. Uncomment the 4th @Test testFilterList().
3. Locate TODO 4 in the test, in FilterTradesTall.
4. Finish the code as instructed.
5. Run the TestFilterTall test: Right mouse click the class , Select run as junit test.
6. Observe the output

Run the code on the Cluster

Now Copy the jar to the cluster. Populate a tall table with data. Run Filters:

1. Build the project by selecting the project folder and choosing Run as > Maven Install.
2. Upload the target jar, lab-exercise-filter-1.0.jar, to your user directory on the cluster.

   ```
   scp ./lab-exercise-filter-1.0.jar
   mapr@<clusterhostid>:/user/mapr
   ```

3. Run the CreateTable driver class to generate a table for stock trade data.

   ```
   java -cp `hbase classpath`:./lab-exercises-filter-1.0.jar  \
   filter.CreateTable tall
   ```

   Run the FilterTrades driver class to run different filters on the data. (You can run with no arguments to view a usage statement.) For example:

   ```
   java -cp `hbase classpath`:./lab-exercises-filter-1.0.jar
   filter.FilterTradesTall list
   ```

   Run with argument prefix for a prefix filter

   ```
   java -cp `hbase classpath`:./lab-exercises-filter-1.0.jar
   filter.FilterTradesTall prefix
   ```

   Run with argument row for a row filter

```
java -cp `hbase classpath`:./lab-exercises-filter-1.0.jar
filter.FilterTradesTall row
```

Exercise 4.2: Java applications using Advanced HBase Java API using Increment

Overview

The objective of this lab exercise is to use Increment instead of checkAndPut to manage a transaction of checkout inventory items. This simplifies in case of conflicts as the programmer does not have to get the value and check it before updating it.

Information on HBase Java API for Increment

Here is example code for using table increment()

```
Increment increment1 = new Increment(Bytes.toBytes(rowkey));
increment1.addColumn(Bytes.toBytes(cf), Bytes.toBytes(col1), value);
increment1.addColumn(Bytes.toBytes(cf), Bytes.toBytes(col2), value);
Result result1 = table.increment(increment1);
```

Hands-on Lab Exercise Overview

You will use Increment instead of checkAndPut for the Shoppingcart application and see the differences between these two ways of implementing 'checkout' functionality.

You will Create a table called 'Inventory' with the following schema:

Table: **Inventory**

	cf: stock
	quantity
pens	13
notepads	23
erasers	10

Next you will Create & Insert the following data into the 'Shoppingcart' table:

[RowKey: Username]

	Cf: cartitems					
	pens	notepads	erasers			
John	2	1	0			
Mike	1	1	1			
Mary	1	2	5			
Adam	5	4	0			

Then you will complete the implementation for checkOutIncrement () in ShoppingCartApp:

Say you want to checkout Adam, the result of that operation for Inventory Table should be as shown below:

Table: **Inventory**

	cf: stock	
	Quantity	Adam
pens	~~13~~ 8	5
notepads	~~23~~ 19	4
eraser s	10	

Here for pens, change 13 to 8 and insert 5 under the new 'Adam' qualifier in one operation.

Also for notepads, change it to 19 and insert 4 under the Adam qualifier.

Import and build the "lab-exercises-shopping3" project

Import the exercise project "lab-exercises-shopping3" into Eclipse, build the project using Maven.

1. Import the **lab-exercises-shopping3** project into Eclipse.

   See the notes in "Working With Lab Projects In Eclipse" document.

2. Build the project using Maven: Select project lab-exercises-shopping3 -> Run As ->Maven Install

Working with Shoppingcart Application checkout using Increment

Finish check out code

1. Look in the src/test/java directory at **shopping.TestCheckout**. Look for "TODO 1a"s. **Uncomment** the test **@Test** line. This test checkouts using Increment. Look for "**TODO 1a**"s in the Test ShoppingCartApp, and InventoryDAO, **to finish code** in ShoppingCartApp checkout() and InventoryDAO checkoutWithIncrement() .

2. Then run the junit test, right mouse click on the class and select run as junit test. Observe what the unit test does, and prints out. If you completed the code correctly the test should run green, if it is red you need to correct the code.

```
Test Checkout

Print Inventory

Inventory [stockId=erasers, quantity=10]

Inventory [stockId=notepads, quantity=21]

Inventory [stockId=pens, quantity=9]



Checkout for CartId :  Mike

ShoppingCart [rowkey=Mike, pens=1, notepads=2, erasers=3]



--checkout Inventory stockId pens cartId Mike quantity 1



--checkout success: Changed pens quantity 9 to 8
```

--added column for cartId Mike quantiy 1

Pens Inventory rowKey pens : Family - stock : Qualifier - Mike : Value: 1

Family - stock : Qualifier - quantity : Value: 8

--checkout Inventory stockId notepads cartId Mike quantity 2

--checkout success: Changed notepads quantity 21 to 19

--added column for cartId Mike quantiy 2

Notepads Inventory row Result with rowKey notepads : Family - stock : Qualifier - Mike : Value: 2

Family - stock : Qualifier - quantity : Value: 19

--checkout Inventory stockId erasers cartId Mike quantity 3

--checkout success: Changed erasers quantity 10 to 7

--added column for cartId Mike quantiy 3

erasers Inventory row Result with rowKey erasers : Family - stock : Qualifier - Mike : Value: 3

Family - stock : Qualifier - quantity : Value: 7

Table: **Inventory**

	cf: stock	
	quantity	Mike
pens	~~10~~ 9	1
notepads	~~21~~ 19	3
erasers	~~10~~ 7	2

Run on the cluster to Initialize the data for the two tables Inventory & Shoppingcart:

1. Copy the jar file "lab-exercises-shopping3-1.0.jar" from the target folder to the cluster
2. Login to the cluster, run the application and see what is created and what data you have.

   Setup the shopping cart table :

   ```
   java -cp `hbase classpath`:./lab-exercises-shopping3-1.0.jar shopping.ShoppingCartApp setup
   ```

   put items in the shopping cart table :

   ```
   java -cp `hbase classpath`:./lab-exercises-shopping3-1.0.jar shopping.ShoppingCartApp initshopping
   ```

   Run checkout :

   ```
   java -cp `hbase classpath`:./lab-exercises-shopping3-1.0.jar shopping.ShoppingCartApp checkout Mike
   ```

```
Test Checkout

Print Inventory

Inventory [stockId=erasers, quantity=10]

Inventory [stockId=notepads, quantity=21]

Inventory [stockId=pens, quantity=9]



Checkout for CartId :  Mike

ShoppingCart [rowkey=Mike, pens=1, notepads=2, erasers=3]



--checkout Inventory stockId pens cartId Mike quantity 1



--checkout success: Changed pens quantity 9 to 8

--added column for cartId Mike quantiy 1

Pens Inventory  rowKey pens :  Family - stock : Qualifier - Mike : Value: 1

   Family - stock : Qualifier - quantity : Value: 8
```

--checkout Inventory stockId notepads cartId Mike quantity 2

--checkout success: Changed notepads quantity 21 to 19

--added column for cartId Mike quantiy 2

Notepads Inventory row Result with rowKey notepads : Family - stock : Qualifier - Mike : Value: 2

Family - stock : Qualifier - quantity : Value: 19

--checkout Inventory stockId erasers cartId Mike quantity 3

--checkout success: Changed erasers quantity 10 to 7

--added column for cartId Mike quantiy 3

erasers Inventory row Result with rowKey erasers : Family - stock : Qualifier - Mike : Value: 3

Family - stock : Qualifier - quantity : Value: 7

Table: **Inventory**

	cf: stock	
	quantity	Mike
pens	~~10~~ 9	1
notepads	~~21~~ 19	3
erasers	~~10~~ 7	2

Lab 5: Time Series Application

Lab Overview

The purpose of this lab is to explore and implement tall-narrow and flat-wide schemas for HBase tables, using stock-exchange data as an example of time-series data. We will be given the complete implementation of the flat-wide schema, and our challenge will be to implement a tall-narrow version.

Exercise	Required	Duration
5.1 Examine two different schema design options	Yes	20 min
5.2 Finish code and Run Unit test	Yes	30 min

Example Application: Store Stock Trade History

We will write a program designed to store stock trade history for our access patterns. A trade contains the following information, which form the elements of a Trade class.

- timestamp of trade event
- stock symbol
- price per share
- volume of trade

The following are the data access methods we will focus on for this lab. These constitute a simple interface to a DAO for the stock trade datastore.

- Store a trade
- Retrieve trades by date for a company

Imagine that trades will arrive in real-time and be written to the datastore for our access patterns. We want to store data such that the most recently-written data is easily retrieved first. We want to be able to efficiently compute daily statistics about each company, such as the day's highest price, lowest price, volume, etc. Finally, we'd like data to be structured to enable efficient access to a single hour of data.

Flat-Wide Schema

In this flat schema, all trades for each day are stored in a single row. Trades are grouped by the hour of the day, using a column for every hour. Price and Volume values are stored in separate column families. Every version of a cell represents one trade, and the cell version (a long) stores the timestamp of the trade in milliseconds. Because every version of a cell is significant, the Price and Volume column families

must be configured to keep all versions. The rowkey is a composite of the company symbol and the day-of-year, formatted YYYYMMDD. For example: AMZN_20131020.

The following diagram shows the flat table schema, including an example of a few data points.

RowKey: SYM_DATE	**09**	**10**	**11**	**12**	**…**	**15**	**09**	**10**	**11**	**…**	**15**	**DAY HI**	**DAY LO**	**DAY VOL**
CFs:	PRICE						VOL					STATS		
AMZN_20131020	@ts2: 12.34 @ts1: 12.00						@ts2: 1000 @ts1: 2000							
CSCO_20131023						@ts3: 1.23					@ts3: 3000			
GOOG_20130817			@ts6: 2.34						@ts6: 1000					

This schema also defines a column family to hold statistics for each company for the day, but we will not use this column family in this lab.

Tall-Narrow Schema

In this tall schema, every row represents one trade. There is only one column family, with 2 columns to store Price and Volume values. The composite rowkey is formed by combining the stock symbol and a reversed timestamp, (`Long.MAX_VALUE - timestamp`). For example: AMZN_98618600888. Because the rowkey contains timestamp data, the trade time is not stored anywhere else in the table.

The following diagram shows the tall table schema, including an example of a few data points.

RowKey: SYM_TIME	**PRICE (long)**	**VOL (long)**
CFs:	CF1	
AMZN_98618600888	12.34	1000

AMZN_98618600777	12.00	2000
CSCO_98618600666	1.23	3000
GOOG_9861860555	2.34	1000

Program Structure

Start with the following Java classes:

- **`CreateTable`** – A driver class to create a table and populate it with test data
  - **`CreateTablesUtil`** – A helper class with methods used by CreateTable
- **`LookupTrades`** – A driver class to lookup trades
- **`Trade`** – A Java object that holds data for a single trade
- **`TradeDAO`** – A Data Access Object (DAO) interface that hides details of schema implementation
  - **`TradeDAOFlat`** – The flat-wide implementation of the TradeDAO. You will finish code for this class.
  - **`TradeDAOTall`** – The tall-narrow implementation of the TradeDAO. You will finish code for this class.

Note the following about the test data used in this lab:

- Our test data uses randomly-generated timestamps within a date range, so some trade timestamps correspond to times that the stock market is not active. The schema diagrams above show only columns for hours 09 – 15, the hours that the stock market is open. However, in our test data the hours may range from 00 – 24, and we will treat all timestamps as valid trades.
- The test stock prices are also randomly generated, and do not accurately represent the price for any of the companies shown here.

Exercise 5.1:Examine two different schema design options

Examine the two schemas above and consider the following questions with respect to the proposed schemas. Answers are at the end of this exercise.

1. Given a realistic stream of stock trades, will either of these schemas exhibit hot-spotting? Why or why not?

2. What do these schemas assume about the nature of trade frequency? How could you improve the schema? (Hint: Timestamp granularity is to the millisecond.)

3. In order to calculate the daily statistics (Daily high, Daily low, Daily volume), for each schema, what data-access operation would you perform to gather the trades for the day?

4. The flat schema provides a column family to store daily statistics with each row. In the tall schema, where would you store daily statistics?

5. In the tall schema, what is the right number of Max Versions for the following cells?

- Tall schema, CF1 column family: ___
- Flat schema, Price column family: ___
- Flat schema, Volume column family: ___
- Flat schema, Stats column family: ___

ANSWERS to Exercise 1

Below are possible answers to the questions posed in Exercise 1.

1. Assuming that trades for companies arrive in no particular order, this rowkey will not exhibit hot-spotting.

2. Both schemas assume that one company cannot have more than one trade at the same millisecond. In a real-world stock market, this assumption might not hold true. For the tall-narrow schema, we could augment the rowkey to include a unique trade ID.

3. We can use a Get operation to return a single row for a particular company and date in the flat table. We need to scan a range of rows with rowkeys matching the millisecond boundaries for the date range in the tall table.

4. The tall-narrow schema does not lend itself to storing daily summary data in the same table as the trade data. We will have to store statistics data separately, possibly in another table.

5. The following values are acceptable for cell Max Versions

- Tall schema, CF1 column family: 1

- Flat schema, Price column family: 3,600,000
  (Each hour has 3,600,000 milliseconds, which is the max theoretical versions to store.)
- Flat schema, Volume column family: 3,600,000
- Flat schema, Stats column family: 1

Exercise 5.2: Finish code and Run Unit test

Perform the following tasks:

1. If you have not already done so, import the Maven project `schemadesign` into Eclipse, as described in chapter 0 Import Projects in Eclipse.

2. Build the project by selecting the project folder and choosing **Run as > Maven Install**.

3. Look in schemadesign.MyHBaseTestFlat and schemadesign.TradeDAOFlat, finish the code, following the TODO's. Then run the MyHBaseTestFlat junit test, right mouse click on the class and select run as junit test. If it does not run green, fix the errors.

Populate a flat table with data, and read values back

1. Build the project by selecting the project folder and choosing Run as > Maven Install.

2. Upload the target jar, `schemadesign-1.0.jar`, to your user directory on your sandbox or cluster as explained in chapter 0 "Copying Files to the Sandbox or AWS Cluster". For example:

   ```
   $ scp ./schemadesign-1.0.jar mapr@<cluster ip>:/user/mapr
   ```

3. There is a test data file, `500trades.txt`, at the top level of the project directory. Upload the test data to your user directory.

4. Run the `CreateTable` driver class to generate a table for stock trade data, based on the file 500trades.txt. The test data includes trades for the week of October 21 to October 25, 2013, for 3 companies: Amazon (AMZN), Cisco Systems (CSCO), and Google (GOOG). (You can run with no arguments to view a usage statement.)

For example:

```
$ java -cp `hbase classpath`:./schemadesign-1.0.jar \
    schemadesign.CreateTable flat ./500trades.txt
```

Run the `LookupTrades` driver class to read back the trades for a particular company. (You can run with no arguments to view a usage statement.)

For example:

```
java -cp `hbase classpath`:./schemadesign-1.0.jar \
   schemadesign.LookupTrades flat AMZN
```

```
No start and stop dates specified. Retrieving all trades for AMZN
Using DAO: class schemadesign.TradeDAOFlat
Printing 158 trades.
AMZN: 7736 shares at $95.36 at 2013.10.21 07:39:01
AMZN: 5612 shares at $50.34 at 2013.10.21 08:46:00
AMZN: 8277 shares at $78.53 at 2013.10.21 09:39:26
```

5. Run the `LookupTrades` driver class to read back the trades for all companies (AMZN, CSCO, GOOG), one by one, and add up the total number of trades. (the output says the total first: Using DAO: class schemadesign.TradeDAOFlat Printing 158 trades.)
   Why doesn't the total add up to 500, as we expect? Hint: Use the MCS to look up column family properties for the flat table.

Modify CreateTableUtils class

Perform the following tasks:

1. In Eclipse, modify the `CreateTableUtils.createTable()` method to increase the Max Versions for all column families. (For now, it is OK to increase MaxVersions for all column families, even though not all column families need it.)

2. Delete the table `trades_flat`, using the hbase shell (substitute your userid)

   a. disable '/user/user01/trades_flat'
   b. drop '/user/user01/trades_flat'

3. Re-create the table using the `CreateTable` class. Use the same test data input, `500trades.txt`.

4. Use the `LookupTrades` driver to count the number of trades stored for each company again. Does the total add up to 500?

Finish code in the data access object for a tall table schema

In the following tasks you will write code to implement portions of the DAO for the tall-narrow schema, `TradeDAOTall`. For clues on how to proceed, you can refer to the methods in `TradeDAOFlat`.

1. Look in schemadesign.MyHBaseTestTall

2. Uncomment the @Test.

3. Look in schemadesign.TradeDAOTall

4. Finish the code as described in the following steps, following the TODO's.

5. Implement `formRowkey()` for `TradeDAOTall`. This method takes a (String) stock symbol and a (long) timestamp, and returns the rowkey. Use a reverse timestamp in the rowkey by

subtracting `(Long.MAX_VALUE - time)`. An example rowkey looks like this: `GOOG_92233706544769`

6. Implement getTradesByDate() for TradeDAOTall.
7. Run the MyHBaseTestTall junit test
8. Right mouse click on the class and select run as junit test. If it does not run green , fix the errors.
9. Build the project by selecting the project folder and choosing Run as > Maven Install.
10. Upload the target jar, schemadesign-1.0.jar, to your user directory on the cluster. For example:

    $ scp ./schemadesign-1.0.jar mapr@<cluster hostid>:/user/mapr

There is a test data file, `500trades.txt`, at the top level of the project directory, make sure the file is uploaded to your user directory.

1. Run the `CreateTable` driver class to generate a table for stock trade data, this time using the tall-narrow implementation. For example:

```
$ java -cp `hbase classpath`:./schemadesign-1.0.jar \
    schemadesign.CreateTable tall  ./500trades.txt
```

Run the `LookupTrades` driver class to read back the trades for a particular company.

For example:

```
$ java -cp `hbase classpath`:./schemadesign-1.0.jar \
    schemadesign.LookupTrades tall AMZN 20131021 20131022

Retrieving trades for AMZN from 20131021 to 20131022
Using DAO: class schemadesign.TradeDAOTall
Printing 26 trades.
AMZN: 2106 shares at $78.47 at 2013.10.21 22:59:21
AMZN: 4712 shares at $80.00 at 2013.10.21 22:29:06
AMZN: 7606 shares at $96.98 at 2013.10.21 21:58:35
```

Reflect on potential improvements for the schema design

Reflect on your experience with the two table designs:

1. Which of these implementations do you think is the better design?

2. For the better design, name two ways you could you modify the schema or DAO implementations to improve upon this design? Why?

Lab 6: Developing MapReduce Applications for HBase (flat-wide and tall narrow)

Lab Overview

The purpose of this lab is to develop a MapReduce application in Java that calculates basic statistics for data in MapR-DB tables using both flat-wide and tall-narrow schemas. There are 2 lab exercises in this module.

Exercise	Required	Duration
6.1: Developing MapReduce Applications for HBase (flat-wide)	Yes	50 min
6.2: Developing MapReduce Applications for HBase (tall-narrow)	Yes	30 min

Exercise 6.1: Developing MapReduce Applications for HBase (flat-wide)

The objective of this lab is to develop a MapReduce application in Java that calculates basic statistics for data in MapR-DB tables using a flat-wide schema.

This lab consists of the following steps:

1. Populate the trades flat HBase table
2. Run a map-only program on an HBase table.
3. Run a map-reduce program on an HBase table.
4. Calculate other statistics.

NOTE: There are 3 types of quotes used in this lab – use the correct quotes in your shell**.**

- Single quote = ‘
- Back tick = `
- Double quote = “

Identify your sandbox or aws cluster node hostname or IP address and replace the host variable in the subsequent exercises with that value.

Populate the trades_flat HBase table (if necessary)

If you did not perform the schema design lab in which the ~/trades_flat table is created, then perform the tasks in this exercise.

1. Copy the **schemadesign-1.0.jar** file to your home directory.
2. Replace userXX with your userid and *host* with your cluster host IP address.

```
scp  schemadesign-1.0.jar userXX@host:
```

3. Run the CreateTable program as follows.

```
 java -cp `hbase classpath`:./schemadesignsolution-1.0.jar
schemadesign.CreateTable flat ./500trades.txt
```

4. Verify your table is populated (replace user01 with your userid)

```
echo "scan '/user/user01/trades_flat'" | hbase shell
```

The following diagram shows the flat table schema, including an example of a few data points.

RowKey: SYM_DATE	**09**	**10**	**11**	**12**	**…**	**15**	**09**	**10**	**11**	**…**	**15**	**DAY HI**	**DAY LO**	**DAY VOL**
CFs:	PRICE						VOL					STATS		
AMZN_20131020	@ts2: 12.34 @ts1: 12.00						@ts2: 1000 @ts1: 2000							
CSCO_20131023						@ts3: 1.23					@ts3: 3000			
GOOG_20130817			@ts6: 2.34						@ts6: 1000					

The reduce job will populate the statistics column family with count, max, min, mean, for each company for the day.

Run a **map-only** program on an HBase table

The format of the flat wide **map input** key, row is:

GOOG_20131024 column=price:15, timestamp=1382655321309, value=9996

GOOG_20131024 column=price:15, timestamp=1382662214757, value=7059

GOOG_20131024 column=price:18, timestamp=1382663397116, value=5005

The format of the flat wide **map output** key , price (long) :

GOOG_20131025 5135

GOOG_20131025 6155

GOOG_20131025 5095

1. Import the map-reduce-lab project into Eclipse.
2. Modify the code TODO's for exercise 2
   - Finish the code in mapreducelab.flatwide.StockMapperTest and mapreducelab.flatwide.StockMapper following the TODO's.
3. Run the StockMapperTest junit test
   - Right mouse click on the class
   - Select run as junit test.
   - If it does not run green, fix the errors.
4. Copy the jar file to the cluster (replace "userxx" with your user id and *host* with the IP address of your cluster node)

   ```
   scp mapreduce-lab-1.0.jar userXX@host:
   ```

5. Login to the cluster as your user and run the code

   ```
   java -cp `hbase classpath`:./mapreduce-lab-1.0.jar
   mapreducelab.flatwide.StockDriver ~/OUT2
   ```

6. Examine the output

   ```
   cat ~/OUT2/part-r-00000
   ```

Run a map-reduce program on an HBase table

The format of the flat wide **reduce input** key, prices

GOOG_20131022 5005 , 6155, 9996,

Flat wide **reduce output** key put

GOOG_20131022	column=stats:count, timestamp=1385137969400, value=37
GOOG_20131022	column=stats:max, timestamp=1385137969400, value=99.96
GOOG_20131022	column=stats:mean, timestamp=1385137969400, value=75.59
GOOG_20131022	column=stats:min, timestamp=1385137969400, value=50.05

1. Modify the code TODO's for **exercise 3** and run the unit test
   - Finish the code in mapreducelab.flatwide.StockReducerTest and mapreducelab.flatwide.StockReducer following the TODO's.
2. Run the StockReducerTest junit test
   - Right mouse click on the class and select run as junit test.
   - If it does not run green, fix the errors.
3. Modify the code TODO's for exercise 3 and run the job

   These are in the **StockDriver.java** file.
4. Copy the jar file to the cluster (replace "userXX" with your user id and host with your cluster node IP address)

   ```
   scp mapreduce-lab-1.0.jar userXX@host:
   ```
5. Login to the cluster as your user and run the code

   ```
   java -cp `hbase classpath`:./mapreduce-lab-1.0.jar
   mapreducelab.flatwide.StockDriver ~/OUT3
   ```
6. Examine the newly created table columns of min and count (replace "userXX" with your user id)

   ```
   echo "scan '/user/user01/trades_flat'" | hbase shell
   ```

   you should now see values in the statistics column family

GOOG_20131024	**column=stats:count, timestamp=1432677164563, value=27**
GOOG_20131024	**column=stats:max, timestamp=1432677164563, value=92.74**
GOOG_20131024	**column=stats:mean, timestamp=1432677164563, value=77.29**
GOOG_20131024	**column=stats:min, timestamp=1432677164563, value=51.7**

7. Examine the output (why does it not exist?)

```
cat ~/OUT3/part-r-00000
```

Exercise 6.2: Developing MapReduce Applications for HBase (tall-narrow)

The objective of this lab is to develop a MapReduce application in Java that calculates basic statistics for data in HBase tables using a tall-narrow schema. This lab consists of the following steps:

1. Populate the trades_tall HBase table
2. Run a map-only program on an HBase table.
3. Run a map-reduce program on an HBase table.
4. Calculate other statistics.

NOTE: There are 3 types of quotes used in this lab – use the correct quotes in your shell.

- Single quote = ‘
- Back tick = `
- Double quote = “

Identify your cluster node hostname or IP address and replace the *host* variable in the subsequent exercises with that value.

Populate the trades_tall HBase table (if necessary)

If you did not perform the schema design lab in which the ~/trades_tall table is created, then perform the tasks in this exercise.

8. Run the CreateTable program as follows. Replace userXX with your userid.

```
java -cp `hbase classpath`:./schemadesignsolution-1.0.jar
schemadesign.CreateTable tall
```

9. Verify your table is populated (replace userXX with your userid)

```
echo "scan '/user/user01/trades_tall'" | hbase shell
```

The following diagram shows the tall table schema, including an example of a few data points. The Reduce job will add the Mean, Max, and Min.

RowKey: SYM_TIME	PRICE (long)	VOL (long)	Mean	Max	Min
CFs:	CF1				
AMZN_98618600888	12.34	1000			
AMZN_98618600777	12.00	2000			
CSCO_98618600666	1.23	3000			
GOOG_9861860555	2.34	1000			
AMZN			12	12.34	12

Run a map-only program on an HBase table

tall narrow map input

GOOG_922337065438554 column=CF1:price, timestamp=1383943828225, value=9996

GOOG_922337065438700 column=CF1:price, timestamp=1383943828322, value=5005

tall narrow map output

GOOG 9996

GOOG 5005

1. Import the map-reduce-lab project into Eclipse.
2. Modify the code TODO's for exercise 2 and run the unit test

- Finish the code in mapreducelab.tallnarrow.StockMapperTest and mapreducelab.tallnarrow.StockMapper following the TODO's.

3. Run the StockMapperTest junit test

- Right mouse click on the class
- Select run as junit test. If it does not run green , fix the errors.

4. Modify the code TODO's for exercise 2

These are in the StockDriver.java file.

5. Copy the jar file to the cluster (replace "user01" with your user id and *host* with the IP address of your cluster node)

```
scp mapreduce-lab-1.0.jar userXX@host:
```

6. Login to the cluster as your user and run the code

```
java -cp `hbase classpath`:./mapreduce-lab-1.0.jar
mapreducelab.tallnarrow.StockDriver ~/OUT2
```

7. Examine the output

```
cat ~/OUT2/part-r-00000
```

Run a map-reduce program on an HBase table

tall narrow reduce input

GOOG 9996

GOOG 5005

tall narrow reduce output

GOOG	column=CF1:count, timestamp=1385138594695, value=37
GOOG	column=CF1:max, timestamp=1385138594695, value=99.96
GOOG	column=CF1:mean, timestamp=1385138594695, value=75.59
GOOG	column=CF1:min, timestamp=1385138594695, value=50.05

1. Modify the code TODO's for exercise 3 and run the unit test

- Finish the code in mapreducelab.tallnarrow.StockReducerTest and mapreducelab.tallnarrow.StockReduce following the TODO's.

2. Run the StockReducerTest junit test

- Right mouse click on the class
- Select run as junit test. If it does not run green , fix the errors.

3. Modify the code TODO's for exercise 3

These are in the StockDriver.java file.

4. Copy the jar file to the cluster

```
scp mapreduce-lab-1.0.jar userXX@host:
```

5. Login to the cluster as your user and run the code

```
java -cp `hbase classpath`:./mapreduce-lab-1.0.jar
mapreducelab.tallnarrow.StockDriver ~/OUT3
```

6. Examine the newly created table columns of min and count (replace “userXX” with your user id)

```
echo "scan '/user/user01/trades_tall'" | hbase shell
```

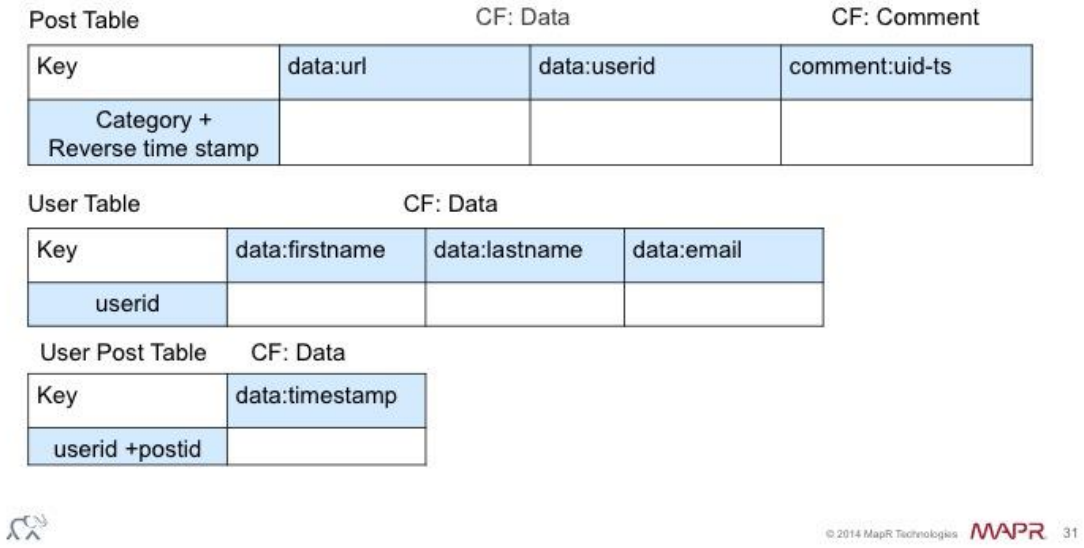
Lab 7: HBase Social Application

Lab Overview

The objective of this lab is to learn how we can implement Secondary indexes to do a faster search in HBase when there is a need to do a search with some data that is not in the row key. We use a "Social" application where any user can "post" a URL to an article and other users can "comment" on the original post. The tables shown below are created to complete the Hbase java "Social" application which de-normalizes (duplicates) data in a secondary index and a nested entity for faster read access.

Exercise	Required	Duration
7.1 Run unfinished example	Yes	20 min
7.2 Import the "lab-exercises-index-solution" project into Eclipse	Yes	20 min
7.3 Modify and complete lab-exercises-index to add the secondary index like the solution	Yes	20 min

Social Application



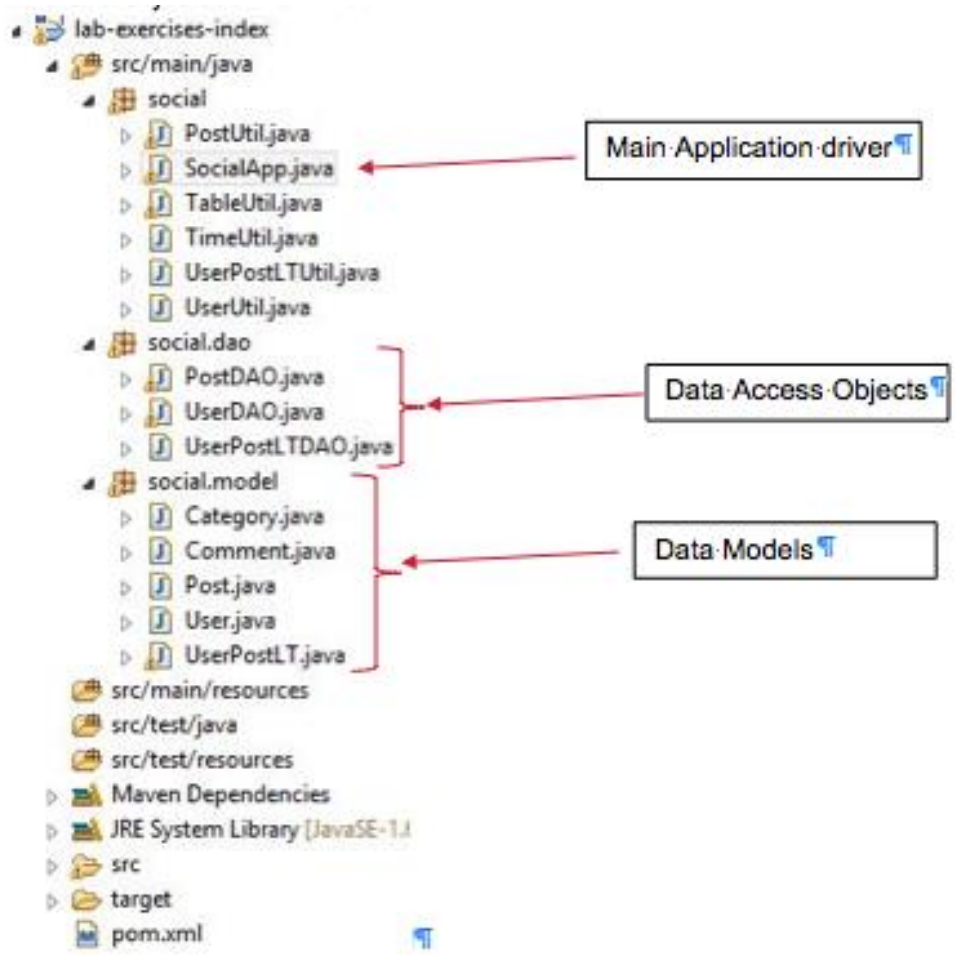
Exercise 7.1: Run unfinished example

Import

Import the Exercises project “lab-exercises-index” into Eclipse, build the project using Maven.

See the instructions in “Importing Projects into Eclipse” lab.

Examine the code



In the social.dao package there is a Data Access Object DAO for each table, these classes contain the code to read and write to their respective tables. In the social.model package there are model objects for User, Post, Comment, Category, and User Post id lookup. In the social package there are utility classes that use the DAOs to print and add objects to the tables. There is also the main SocialApp class. You can pass parameters to the main class:

- **Setup** : to create the tables , populate with data, and print
- **Delete** : to delete the tables
- **Print** : to print the content of the tables

Row keys for the tables

- **User** table key:
  - userId
- **Post** table key:
  - Category code + separator + reverse time stamp

- Allows to get the latest urls by category

- **User-Post** table key:
  - userId + separator + post key
  - Allows to scan for post keys by user , and user , category, date

- Nested Comment Column qualifier:

Userid + timestamp

Example code

Here is example code to add a comment as a nested entity to a post:

```
HTableInterface table = pool.getTable(TABLE_NAME);
Put p = new Put(Bytes.toBytes(postId));
//comment id = userId  | TIMESTAMP
p.add(COMMENT_CF, Bytes.toBytes(comment.id),
Bytes.toBytes(comment.text));
table.put(p);
```

Here is example code to scan for posts by userid in the user post lookup table:

```
//userpost key = userId  | postid
HTableInterface table = pool.getTable(USER_POST_TABLE_NAME);
byte[] startRow = Bytes.toBytes(userId);
byte[] stopRow = Bytes.toBytes(userId); //stop key set to userId
stopRow[stopRow.length - 1]++; //stop key set to userId + 1
Scan s = new Scan(startRow, stopRow);
s.addFamily(CF);
ResultScanner results = table.getScanner(scan);
```

Finish code and Run Unit test

Perform the following tasks:

1. Look in social.MyHBaseTestSocial.java
2. Uncomment the first test.
3. Finish the code in social.dao.UserDAO.java: Search for // TODO 1, and follow the instructions in comments to finish the code.

4. Run the MyHBaseTestSocial junit test

   - Right mouse click on the class
   - Select run as junit test. If it does not run green , fix the errors.

Upload the jar and run on the server

1. Build the project using Maven
2. Copy the "lab-exercises-index-1.0.jar" from the target folder to the cluster.
3. Login to the cluster
4. Run the Social application and see what table is created and what data you have.

```
java -cp `hbase classpath`:./lab-exercises-index-1.0.jar
social.SocialApp setup
```

Exercise 7.2: Import "lab-exercises-index-solution" project into Eclipse

This is optional, you can run the solution to see the behavior that you will finish in the unfinished project

Import

1. Import the Exercises project "lab-exercises-index-solution" into Eclipse

   - See the instructions in "Importing Projects into Eclipse" lab.

Build

2. Build the project using Maven,
3. Copy the "lab-exercises-index-solution-1.0.jar" from the target folder to the cluster.
4. Login to the cluster

Run the application

1. Run the Social application and see what tables are created and what data you have.

```
java -cp `hbase classpath`:./lab-exercises-index-solution-1.0.jar
social.SocialApp delete
```

```
java -cp `hbase classpath`:./lab-exercises-index-solution-1.0.jar
social.SocialApp setup
```

Exercise 7.3: Modify and complete lab-exercises-index to add the secondary index like the solution

Finish the Code

Follow the TODO's marked // TODO 3 as described below

1. Look in social.MyHBaseTestSocial.java. Uncomment the second test.

2. Take a look at the UserPostLTUtil.java addUserPostLT(userId, postId) method. This method creates a UserPostLT model object and then calls the UserPostLTDAO to put the object in the table.

3. Take a look at the UserPostLT.java constructor UserPostLT(String userId, String postId). This constructor calls makeId to make the key for the UserPostLT row.

4. Go to UserPostLT.java model class.

    - Search for // TODO 3, and follow the instructions in comments to finish the code.

5. Go to UserPostLTDAO.java class.

    - Search for // TODO 3, and follow the instructions in comments to finish the code.

Run the unit test

1. Run the MyHBaseTestSocial junit tests, right mouse click on the class and select run as junit test. If it does not run green , fix the errors.

2. Go to SocialApp.java class

    - Search for // TODO 3, and follow the instructions in comments to finish the code.

Build

3. Build the project using Maven,

1. Copy the “lab-exercises-index-1.0.jar” from the target folder to the cluster.
2. Login to the cluster
3. Run the Social application to delete and then setup to see what tables are created and what data you have. If it is working correctly you should now save and print user post lookup ids.

```
java -cp `hbase classpath`:./lab-exercises-index-1.0.jar
social.SocialApp delete
```

```
java -cp `hbase classpath`:./lab-exercises-index-1.0.jar
social.SocialApp setup
```

Exercise 7.4: Modify and complete lab-exercises-index to add the nested entity

Finish the code

Follow the TODO’s marked // TODO 4 as described below

1. Look in social.MyHBaseTestSocial .java
2. Uncomment the third test.
3. In the SocialApp.java class
   - Search for // TODO 4, and follow the instructions in comments.
4. Go to the PostDAO.java class
   - Search for // TODO 4, and follow the instructions in comments.
5. Go to the Comment.java class
   - Search for // TODO 4, and follow the instructions in comments.
6. Go to the PostUtil.java class,
   - Search for // TODO 4, and follow the instructions in comments.

Run the unit test

1. Run MyHBaseTestSocial junit test
2. Right mouse click on the class and select run as junit test. If it does not run green , fix the errors.

Build

1. Build the project using Maven,

1. Copy the “lab-exercises-index-1.0.jar” from the target folder to the cluster.

2. Login to the cluster

3. Run the Social application to delete and then setup to see what tables are created and what data you have.

```
java -cp `hbase classpath`:./lab-exercises-index-1.0.jar
social.SocialApp delete
```

```
java -cp `hbase classpath`:./lab-exercises-index-1.0.jar
social.SocialApp setup
```

Lab 8: Bulk Load data

Lab Overview

The objective of this lab is to use 'importtsv' and 'copytable' commands to bulk load data into MapR-DB tables.

For more information see http://doc.mapr.com/display/MapR/Bulk+Loading+and+MapR+Tables

Exercise	Required	Duration
8.1: Using importtsv and copytable to bulk load data	Yes	30 min
8.2 Using MapReduce to bulk Load data	Yes	30 min

Exercise 8.1: Using importtsv, copytable

Create a Table for Bulk loading

1. Create a Table with either **HBase Shell**:

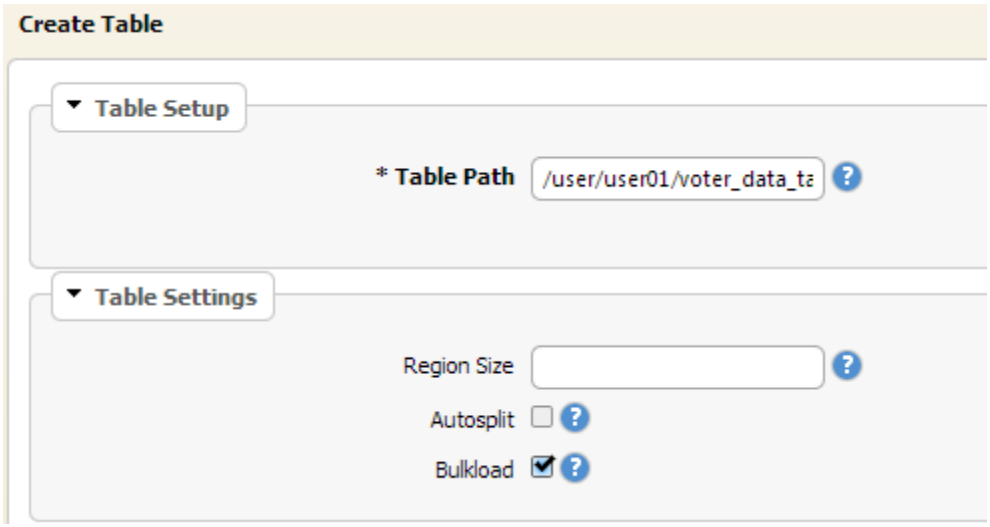
   Create a table using the HBase shell with bulk load true

   ```
   create '/user/user01/voter_data_table', {NAME=>'cf1'},
   {NAME=>'cf2'}, {NAME=>'cf3'},  BULKLOAD => 'true'
   ```

   Or with MCS:

   Create a table called '/user/user01/voter_data_table' (NOTE: make sure you are logged into MCS with your user)

   - Select the bulkload property



a. If using MCS, Add the following Column families:

- Column family = cf1
- Column family = cf2
- Column family = cf3

2. Observe the new table you have created in the UI and at the cli(your choice)

- In MCS: highlight MapR tables> Go To Table /user/userxx/voter_data_table

3. Locate the sample data file (voter data on the cluster in '/tmp/data' folder)

4. Copy voter data to your directory: cp /tmp/data/voter1M ~

The following Table shows sample data from this file:

1	david Davidson	49	socialist	369.78	5108
2	priscilla steinbeck	61	democrat	111.76	2987
3	ethan allen	40	democrat	961.51	13817
4	zach van buren	71	libertarian	421.63	8822
5	gabriella young	74	independent	409.68	11798
6	zach ichabod	52	libertarian	480.76	26113
7	tom thompson	36	republican	830.34	28480
8	mike robinson	38	independent	178.64	29148

9	luke johnson	56	socialist	200.81	4255
10	oscar xylophone	74	green	265.97	24970

5. Run the hbase command with importtsv change user01 to your user :

```
hbase org.apache.hadoop.hbase.mapreduce.ImportTsv -
Dimporttsv.columns=HBASE_ROW_KEY,cf1:name,\

cf2:age,cf2:party,cf3:contribution_amount,\

cf3:voter_number \

-Dimporttsv.bulk.output=/user/user01/dummy \

/user/user01/voter_data_table \

/user/user01/voter1M
```

Notice the first column is defined as HBASE_ROW_KEY, this will take the first field of data (namely the numerical index field) and make it the rowkey.

Important: also notice that command above identifies each column in the data file as well as the column family it belongs in. The column family used in the example below is cf1, cf2 and cf3. If the table you are importing into has a different column family name, then you will need to modify the command below to match the correct column family name

After completing a full bulk load operation, take the table out of bulk load mode to restore normal client operations. You can do this from the command line or the HBase shell with the following commands:

```
# maprcli table edit -path /user/user01/voter_data_table -bulkload
false
```

(command line)

or

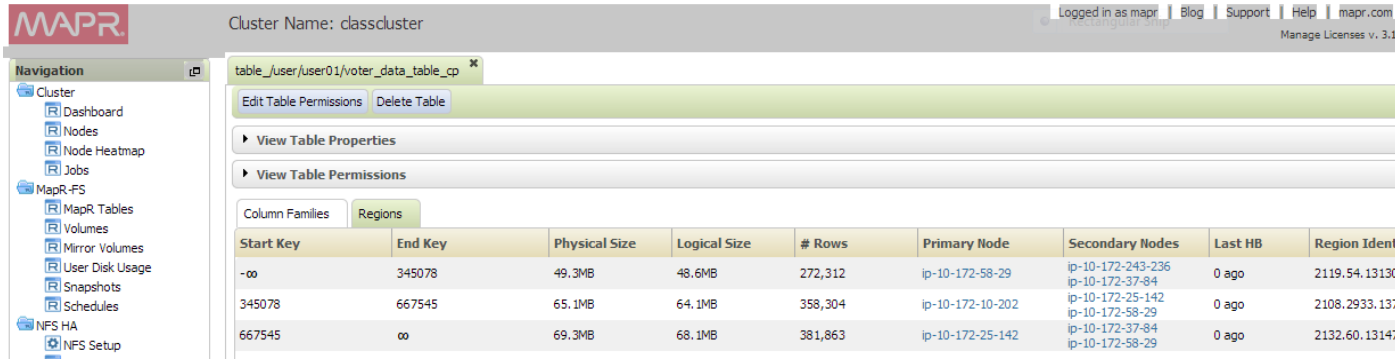
```
hbase shell> alter '/user/user01/voter_data_table', BULKLOAD => 'false'
```

(hbase shell)

While the import job is processing, look at the MCS to view changes to the table and puts being processed on the node

1. Click Nodes under Cluster
2. Click the Overview dropdown and change the value to Performance

3. If necessary, scroll to the right so you can see the Gets, Puts and Scans columns.
4. Normally You would see a large number of puts across several nodes while your import is processing, but since this is a bulk import it skips the puts
5. Click MapR Tables under MapR-FS
6. Click the name of the table you used for the import under Recently opened tables
7. Select the Regions tab
8. You should see that your table automatically split into a number of regions during the import (if the import data was big enough)
9. The following image shows an example of a table after splitting



In an hbase shell examine the data that has been imported.

```
hbase shell
```

```
hbase(main):> scan '/user/user01/voter_data_table', LIMIT => 5
```

Sample output

```
ROW                                    COLUMN+CELL
1       column=cf1:name, timestamp=1377028204508, value=david davidson
1       column=cf2:age, timestamp=1377028204508, value=49
1       column=cf2:party, timestamp=1377028204508, value=socialist
1       column=cf3:contribution_amount, timestamp=1377028204508, value=369.78
1       column=cf3:voter_number, timestamp=1377028204508, value=5108
10      column=cf1:name, timestamp=1377028204508, value=oscar xylophone
10      column=cf2:age, timestamp=1377028204508, value=74
```

```
 10     column=cf2:party, timestamp=1377028204508, value=green
 10     column=cf3:contribution_amount, timestamp=1377028204508, value=265.97
 10     column=cf3:voter_number, timestamp=1377028204508, value=24970
 100    column=cf1:name, timestamp=1377028204508, value=oscar carson
 100    column=cf2:age, timestamp=1377028204508, value=77
 100    column=cf2:party, timestamp=1377028204508, value=socialist
 100    column=cf3:contribution_amount, timestamp=1377028204508, value=123.56
 100    column=cf3:voter_number, timestamp=1377028204508, value=213
 1000   column=cf1:name, timestamp=1377028204508, value=yuri brown
 1000   column=cf2:age, timestamp=1377028204508, value=32
 1000   column=cf2:party, timestamp=1377028204508, value=socialist
 1000   column=cf3:contribution_amount, timestamp=1377028204508, value=847.50
 1000   column=cf3:voter_number, timestamp=1377028204508, value=23520
 10000  column=cf1:name, timestamp=1377028204508, value=ulysses zipper
 10000  column=cf2:age, timestamp=1377028204508, value=33
 10000  column=cf2:party, timestamp=1377028204508, value=libertarian
 10000  column=cf3:contribution_amount, timestamp=1377028204508, value=866.72
 10000  column=cf3:voter_number, timestamp=1377028204508, value=10729
5 row(s) in 0.0960 seconds
```

10. Exit the hbase shell

- `hbase(main):004:0>` **`exit`**

Create another table

1. In MCS or the hbase shell , Create another table called '/user/user01/voter_data_table_10m' with the following schema, and the bulkload property set:

- Column family = cf1
- Column family = cf2
- Column family = cf3

```
create '/user/user01/voter_data_table_10m', {NAME=>'cf1'},
{NAME=>'cf2'}, {NAME=>'cf3'}, BULKLOAD => 'true'
```

In this table we will import the same data and create a new HBASE_ROW_KEY from field position 2 – assuming that the User Names are all unique in the database.

2. Copy a bigger voter data to your directory: cp /tmp/data/voter10M ~

3. Import data to this table using the 10 Million data file /user/user01/voter10M . Change the user to your user.

```
hbase org.apache.hadoop.hbase.mapreduce.ImportTsv -
Dimporttsv.columns=cf1:number,HBASE_ROW_KEY,\
cf2:age,cf2:party,cf3:contribution_amount,\
cf3:voter_number \
-Dimporttsv.bulk.output=/user/user01/dummy2 \
/user/user01/voter_data_table_10m \
   /user/user01/voter10M
```

4. Check the job progress in the MCS as you did before in the previous step

5. In an hbase shell examine the data that has been imported

```
hbase shell
alter '/user/user01/voter_data_table_10m', {NAME=>'cf1'},
{NAME=>'cf2'}, {NAME=>'cf3'}, BULKLOAD => 'false'
scan '/user/user01/voter_data_table_10m', LIMIT => 5
```

Create a Table using MapR Control System (MCS) and copy data using copytable

1. In MCS, Create another table called '/user/user01/voter_data_table_cp' with the following schema:
   - Column family = cf1
   - Column family = cf2
   - Column family = cf3

Or use the hbase shell:

```
create '/user/user01/voter_data_table_cp', {NAME=>'cf1'},
{NAME=>'cf2'}, {NAME=>'cf3'}, BULKLOAD => 'true'
```

2. At the system shell type in the following: (change user01 to your user)

```
hbase com.mapr.fs.hbase.mapreduce.CopyTable -src
/user/user01/voter_data_table -dst /user/user01/voter_data_table_cp
```

3. Check the job progress in the MCS as you did before in the previous step

4. In an hbase shell examine the data that has been imported

   hbase shell

   ```
   alter '/user/user01/voter_data_table_cp', BULKLOAD => 'false'

   scan '/user/user01/voter_data_table_cp', LIMIT => 5
   ```

Exercise 8.2: Bulk Load data with MapReduce

In this exercise, you will implement a MapReduce job that reads from a text file and writes to a MapR M7 table.

Implement and test

Implement and test a job that reads in text data and outputs the data to a MapR M7 table. The data is provided for you as a text file. The data file is 'hly-temp-10pctl.txt' which is in TSV format.

For each station, temperatures are recorded on an hourly basis and data is written for each day on a single line for a given station ID.

1. Create the table 'hly_temp' suing the hbase shell command.

   ```
   create '/user/user01/hly_temp', 'readings',  BULKLOAD => 'true'
   ```

2. Create an input directory in your home directory

   ```
   mkdir /user/user01/input
   ```

3. Upload the data file hly-temp-10pctl.txt to your sandbox or cluster into /user/user01/input

4. Create the mapper class ImportMapper as an inner static class of the BulkLoadMapReduce class that is provided to you and override the map method.
   - The station ID is at the beginning of each line and is up to 11 characters long
   - The month is between characters 12 and 14
   - The day is between characters 15 and 17
   - For the row key concatenate the station ID with the month and day.
   - Compose the column name dynamically and start the column name with letter v for value and use the leftPad method to give each column a number corresponding to a temperature so that the column is called v001 for value 1, v002 for value 2 and so on for each temperature.
   - There are up to 24 values for the temperature (captured on an hourly basis) following the day. Each temperature may take up to 6 characters.

- Use the same timestamp for all the inserts.
- To save the data to an HTable invoke this job with the following arguments

```
java -cp `hbase classpath`:./lab-solutions-1.0.jar
bulkload.BulkLoadMapReduce /user/user01/hly_temp /user/user01/input/
/user/user01/output/
```

5. After completing a full bulk load operation, take the table out of bulk load mode to restore normal client operations. You can do this from the command line or the HBase shell with the following commands:

   (command line)

```
# maprcli table edit -path /user/user01/hly_temp -bulkload false
```

```
(hbase shell)
```

```
hbase shell> alter '/user/user01/hly_temp', 'readings', BULKLOAD =>
'false'
```

Lab 10: Using ycsb

Lab Overview

This lab will show how to use ycsb to learn about benchmarking a cluster.

The performance characteristics of a MapR-DBcluster that you could measure include at least the following:

- Overall throughput (operations per second)
- Average latency (average time per operation)
- 99th percentile latency (the time within which 99 percent of individual operations were completed)
- Minimum latency
- Maximum latency
- Distribution of operation latencies

Since we are sharing a small cluster or using a VM we will not use large workloads in our exercises, you can do that on your own cluster.

Exercise	Required	Duration
10.1 Installing and using YCSB	Yes	30 min

YCSB Overview

The goal of the YCSB project is to develop a framework and common set of workloads for evaluating the performance of different data stores. The project comprises two things:

- The YCSB Client, an extensible workload generator
- The Core workloads, a set of workload scenarios to be executed by the generator

The Client is extensible so that you can define new and different workloads. It is straightforward to extend the HBase client to benchmark your schema and workload.

YCSB comes with different pre-defined workloads, and workload property files in the workloads directory:

- Workload A: Update heavy workload
  - Application example: Session store recording recent actions
  - Read/update ratio: 50/50
- Workload B: Read mostly workload
  - Application example: photo tagging; add a tag is an update, but most operations are to read tags
  - Read/update ratio: 95/5
- Workload C: Read only
  - Application example: user profile cache, where profiles are constructed elsewhere (e.g., Hadoop)
  - Read/update ratio: 100/0
- Workload D: Read latest workload
  - Application example: user status updates; people want to read the latest
  - Read/update/insert ratio: 95/0/5
- Workload F: Read-modify-write workload
  - Application example: user database, where user records are read and modified by the user or to record user activity.
  - Read/read-modify-write ratio: 50/50
- Workload E: Short ranges
  - Application example: threaded conversations, where each scan is for the posts in a given thread (assumed to be clustered by thread id)
  - Scan/insert ratio: 95/5

The workloads insert into a **usertable** with 1 KB records (10 fields, 100 bytes each, plus key.

	Cf: family					
key	field0	field1	…	field7	field8	field9
User1	100b	100b		100b	100b	100b

More information is here: https://github.com/brianfrankcooper/YCSB/wiki/Running-a-Workload

Exercise 10.1: Installing and using YCSB

Install YCSB

1. Download and unpack the YCSB benchmarking tool:

```
cd
wget http://cloud.github.com/downloads/brianfrankcooper/\
      YCSB/ycsb-0.1.4.tar.gz
tar zxvf ycsb-0.1.4.tar.gz
cd ycsb-0.1.4/
```

2. The YCSB tests in the following sections are intended to be run in the ycsb-0.1.4 directory. Substitute your username for user01.

Create a Table '/user/user01/usertable' using MCS or the HBase shell

1. Substitute your username for user01.
2. Create a table using MCS called '/user/user01/usertable' with the following schema:
   - Column family = family

```
hbase shell
> create '/user/user01/usertable', 'family'
> list
> exit
```

3. Observe the new table you have created in the UI and at the cli (your choice)
4. Using MCS: highlight MapR tables> Go To Table /user/user01/usertable

Write-Only HBase workload

This test will load records into the YCSB test table.

```
export cp=core/lib/core-0.1.4.jar:\
      hbase-binding/lib/hbase-binding-0.1.4.jar
export ycsb_processes=2
export ycsb_threads=3
```

```
cd ~/ycsb-0.1.4
for i in $(seq 1 $ycsb_processes); do
      HBASE_CLASSPATH=$cp \
      hbase com.yahoo.ycsb.Client \
      -threads $ycsb_threads -db com.yahoo.ycsb.db.HBaseClient \
      -P workloads/workloada -p columnfamily=family -p
table=/user/user01/usertable -load -s \
      -p insertstart=${i}000 -p insertcount=1000 \
      > ~/ycsb_write_${i}.txt &
done
```

This command will spawn 2 YCSB processes, each with 3 threads, attempting to write 100 rows to the M7 table named 'usertable'. Here are the detailed command line options and their meanings:

Option	Meaning	Value	Notes
HBASE_CLASSPATH=$cp	This is the classpath that YCSB needs in order to load its supporting classes.	see above	This should remain unchanged.
$ycsb_processes	The number of YCSB processes to spawn on this machine.	6	Depending on the CPU and memory resources on the client machine, this value may be increased or decreased.
$ycsb_threads	The number of threads in each YCSB process.	4	Set this to half the number of cores in the client machine.
-P	The name of the YCSB workload to run.	workloads/ workloada	This value should remain unchanged.
-p columnfamily	The name of the column family to use in usertable.	family	This must match the name of a column family that exists in usertable.
-p table	The name of the m7 table	/user/user0 1/usertable	
-load	Perform the "load" portion of the test		This value should remain unchanged.

Option	Meaning	Value	Notes
-p insertstart	The row offset that this process should start inserting at.	${i}000	We set this to the value of the loop counter multiplied by 1,000 so that each process gets to write its own set of 1,000 rows.
-p insertcount	How many rows to insert.	1000	Each record written by YCSB is 1k bytes in size. Inserting 1,000 rows means inserting 1000kb total data in each YCSB process.
> ~/ycsb_write_${i}.txt	The output file to write.		Each YCSB process will write its output to a file in the current user's home directory named ycsb_write_*number*.txt. These files will contain the results of the YCSB tests.

Since in this lab we are sharing a small cluster or using a VM we will not use large workloads in our exercises, you can do that on your own cluster. **On your own cluster (not in lab)** for example increase:

- ycsb_processes=6
- ycsb_threads=4
- -p insertstart=${i}0000000 -p insertcount=1000000

This will spawn 6 YCSB processes, each with 4 threads, attempting to write 100 MB to the M7 table. Each record written by YCSB is 1k bytes in size. Inserting 1,000,000 rows means inserting 1GB total data in each YCSB process.

Understanding YCSB test results

The YCSB test outputs a variety of metrics. Examine the file 'ycsb_write_1.txt'. Here are the first few lines:

```
YCSB Client 0.1
Command line:
[OVERALL], RunTime(ms), 1176
[OVERALL], Throughput(ops/sec), 849
[UPDATE], AverageLatency(us), 1000
[UPDATE], MinLatency(us), 1000
[UPDATE], MaxLatency(us), 1000
```

```
[UPDATE], 99thPercentileLatency(us), 1000
```

The output indicates:

- The total execution time
- The average throughput was 98.9 operations/sec (across all threads)
- There were 491 update operations, with associated average, min, max, 95th and 99th percentile latencies
- All 491 update operations had a return code of zero (success in this case)
- 464 operations completed in less than 1 ms, while 27 completed between 1 and 2 ms.

Of particular interest are the **throughput**, meaning the **number of operations processed per second**, and the **99th percentile latency**, meaning the time within which 99 percent of individual operations were completed.

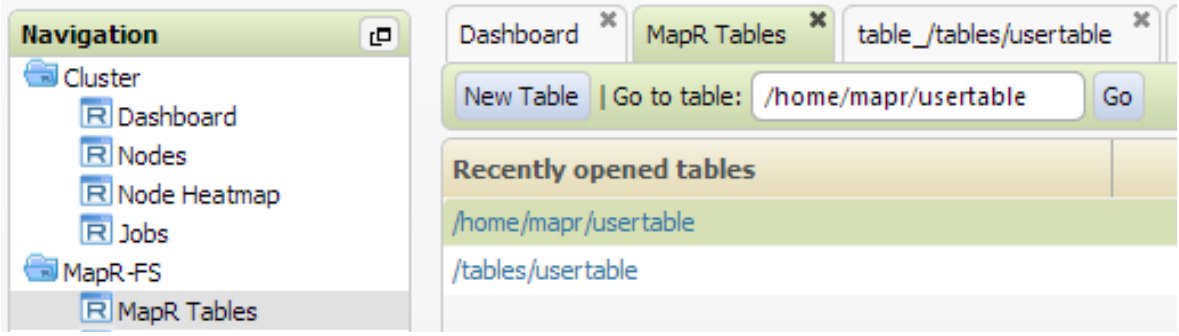
An easy way to collect these metric from the individual YCSB output files would be a command line like the following:

```
for i in ~/ycsb*.txt; do grep -H 'Throughput\|99thPercentile' $i; done

This would result in the following output after having run the write test with 2 YCSB processes:

ycsb_write_1.txt:[OVERALL], Throughput(ops/sec), 27905.825803034256

ycsb_write_1.txt:[INSERT], 99thPercentileLatency(ms), 5

ycsb_write_2.txt:[OVERALL], Throughput(ops/sec), 23117.2088053002776

ycsb_write_2.txt:[INSERT], 99thPercentileLatency(ms), 6
```

Record the results of this activity below:

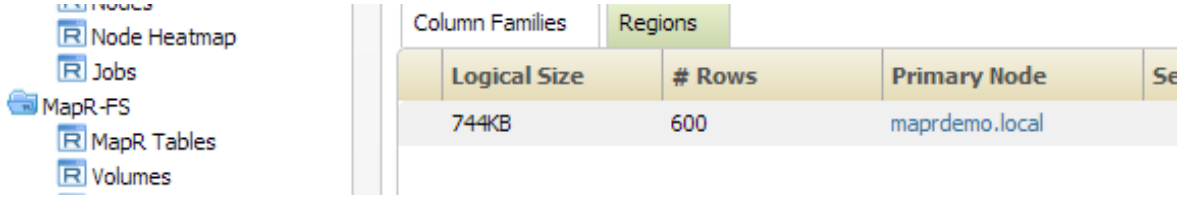
Test	
YCSB write-only throughput	
YCSB write-only 99th percentile latency	

1. Look at the Table '/user/user01/usertable' using MCS
2. Observe the table you have created in the UI and at the cli(your choice)
   - In MCS: highlight MapR tables> Go To Table /user/user01/usertable
   - Click on the primary Node
   - Look at the DB puts
3. Click on MapR Tables, enter the table name, click Go

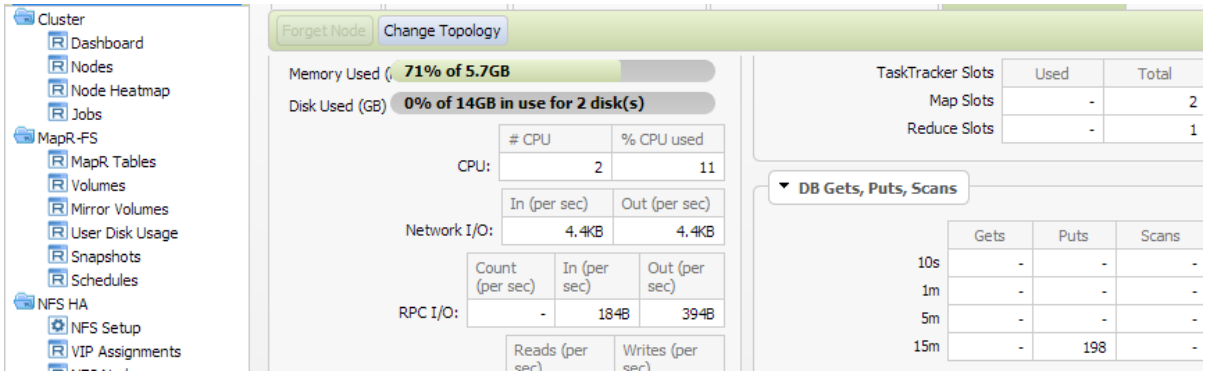


4. Click the Regions Tab

5. Click the primary Node



6. Look at the DB Puts, Memory, disk for the primary node



Read/Write HBase workload

This test will perform 50% reads and 50% writes on random records selected from the data that was created before:

```
export cp=core/lib/core-0.1.4.jar:\
      hbase-binding/lib/hbase-binding-0.1.4.jar
export ycsb_processes=2
export ycsb_threads=3
cd ~/ycsb-0.1.4
for i in $(seq 1 $ycsb_processes); do
      HBASE_CLASSPATH=$cp \
      hbase com.yahoo.ycsb.Client \
      -threads $ycsb_threads -db com.yahoo.ycsb.db.HBaseClient \
      -P workloads/workloada -p columnfamily=family -p
table=/user/user01/usertable -t -s \
      -p insertstart=${i}0000 -p insertcount=1000 \
      -p operationcount=600 \
      > ~/ycsb_rw_${i}.txt &
done
```

Here are the parameters that are different:

Option	Meaning	Value	Notes
-t	Just as -load selects the load portion of the test, -t selects the transaction mode, 50/50 read/write part of the test.		
-p operationcount=600	Number of operations to perform	600	The number of operations to execute. A number of 600 means that the YCSB test will randomly read or write 600 * 1000 bytes per YCSB thread. You can adjust this number to a time that takes a few minutes to run
$ycsb_threads	The number of threads in each YCSB process.	3	Set this to four times the number of cores in the client machine, as gets can wait for some time and we want to have significant concurrency.
> ~/ycsb_rw_${i}.txt	The output file to write.		Each YCSB process will write its output to a file in the current user's home directory named ycsb_rw_*number*.txt. These files will contain the results of the YCSB tests.

Record the results of this activity below:

YCSB read-write throughput	
YCSB read-write 99th percentile latency	

Scan HBase workload

The scan test will insert records at a 5% proportion and scan at 95% proportion. The scan will be short ranges of up to 50 entries as described in the command below:

```
export cp=core/lib/core-0.1.4.jar:\
      hbase-binding/lib/hbase-binding-0.1.4.jar
export ycsb_processes=2
export ycsb_threads=3

cd ~/ycsb-0.1.4
for i in $(seq 1 $ycsb_processes); do
      HBASE_CLASSPATH=$cp \
      hbase com.yahoo.ycsb.Client \
      -threads $ycsb_threads -db com.yahoo.ycsb.db.HBaseClient \
      -P workloads/workloade -p columnfamily=family -p
table=/user/user01/usertable -t -s \
      -p insertstart=${i}0000 -p insertcount=1000 \
      -p operationcount=600 -p maxscanlength=50 \
      > ~/ycsb_scan_${i}.txt &
done
```

Here are the parameters that are different:

Option	Meaning	Value	Notes
-P	The name of the YCSB workload to run.	workloads/ workloade	The range scan test is included in workloade. This value should remain unchanged.
-p maxscanlength	The maximum length range to scan. Range queries will be uniformly distributed between 1 and this number.	50	This value should remain unchanged.
-p operationcount	The number of operations.	600	For accuracy, adjust this number to create a test that takes at least a few minutes to run.

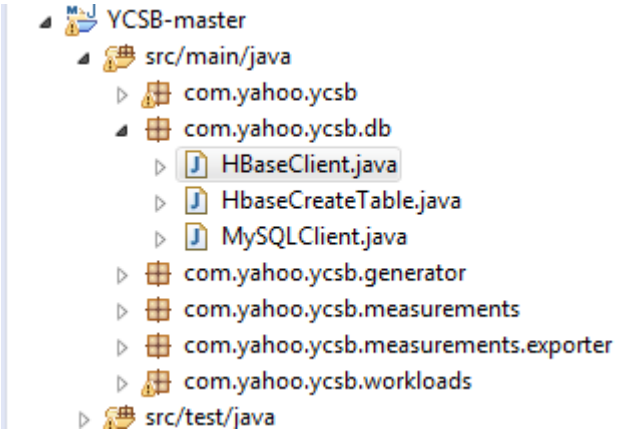
Option	Meaning	Value	Notes
-threads $ycsb_threads	The number of threads in each YCSB process.	3	Set this to four times the number of cores in the client machine, as scans can wait for some time and we want to have significant concurrency.
> ~/ycsb_scan_${i}.txt	The output file to write.		Each YCSB process will write its output to a file in the current user's home directory named ycsb_scan_*number*.txt. These files will contain the results of the YCSB tests.

1. Record the results of this activity below:

Test	
YCSB scan test throughput	
YCSB scan test 99th percentile latency	

2. Take a look at the YCSB HBaseClient code

3. Unzip the ycsb-master.zip file. Import the maven project into eclipse.



4. Take a look at the HBaseClient code. The HBaseClient extends the DB client and overrides the methods to read, write, and scan.

```
public class HBaseClient extends com.yahoo.ycsb.DB {
 @Override
  public int read(String table, String key, Iterable<String> fields,
Map<String, String> result){
   ..
}
```

Lab 11: Security ACE

Lab Overview

The purpose of this lab is to create and edit table permissions and then to test reading / writing to tables.

Exercise	Required	Duration
11.1: Create MapR-DB tables and set permissions	Yes	45 min

Exercise 11.1: Create MapR-DB tables and set permissions

NOTE: For Instructor led training you should do this exercise in pairs. Change user01, user02 to your user ids.

You will create a Table like this with two column families cf1 and cf2 using the hbase shell.

Column Family cf1 has columns c1 and c2, Column Family cf2 has columns c3 and c4

Table: t_user01

Row-key	cf1		cf2	
	c1	c2	c3	c4
r1	v11	v22	v13	v14

Connect to the MapR sandbox or cluster

Accessing MCS:

`https://hostipaddress:8443`

user: user01 ...

password: mapr

Use putty on Windows

Login into remote host via putty [change userxx to user01 ...]:

User01@ hostipaddress

password: mapr

Use ssh on Mac

```
ssh –i user01@hostipaddress
```

Start HBase shell once connected to the cluster

```
-bash-4.1$ hbase shell
```

Create a table in your home directory /user/user01/

```
hbase> create '/user/user01/t_user01', {NAME=>'cf1'}, {NAME=>'cf2'}
```

1. List the tables in your home directory

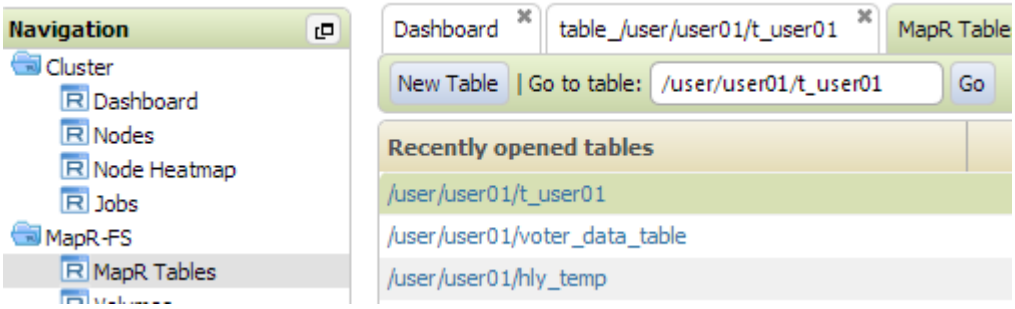
```
hbase> list '/user/user01/'
```

2. Execute the following **put** statements to insert the records into 'mytesttable' **using the hbase shell,**

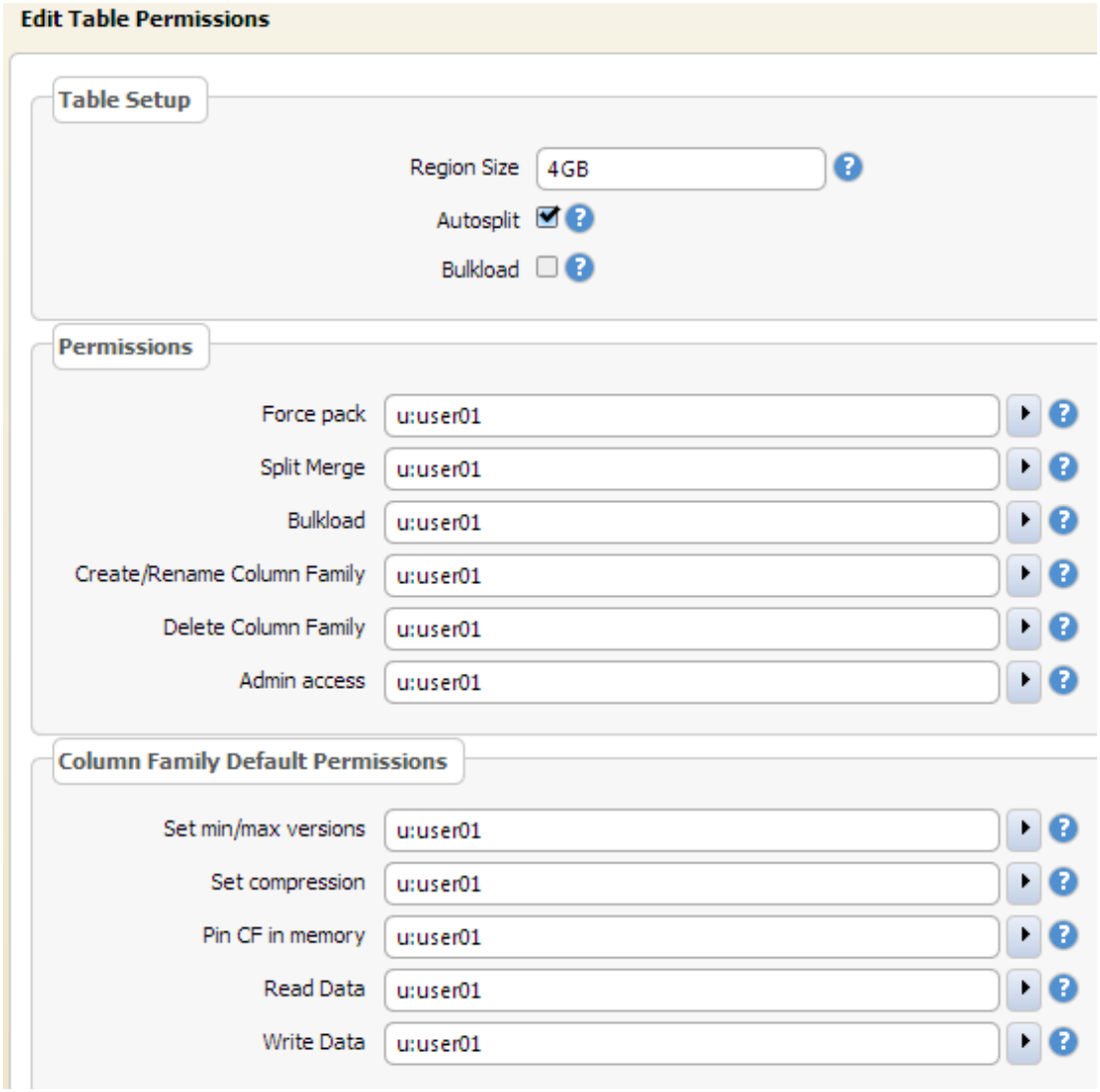
```
put   '/user/user01/t_user01',  'r1',  'cf1:c1', 'v11'
put   '/user/user01/t_user01',  'r1',  'cf1:c2', 'v12'
put   '/user/user01/t_user01',  'r1',  'cf2:c3', 'v13'
put   '/user/user01/t_user01',  'r1',  'cf2:c4', 'v14'
```

3. Complete the rest of the put commands to complete the table as shown on page1.

4. Go to the table in MCS



5. Examine the table permissions



6. Use scan with user01

```
Hbase> scan '/user/user01/t_user01'
```

shows 1 row , 4 values

```
hbase> scan '/user/user01/t_user01'
hbase> list_perm '/user/user01/t_user01'
```

Until now only the owner of the table, user01, has read and write permissions on the table.

7. Switch to user02 and try to scan the table.

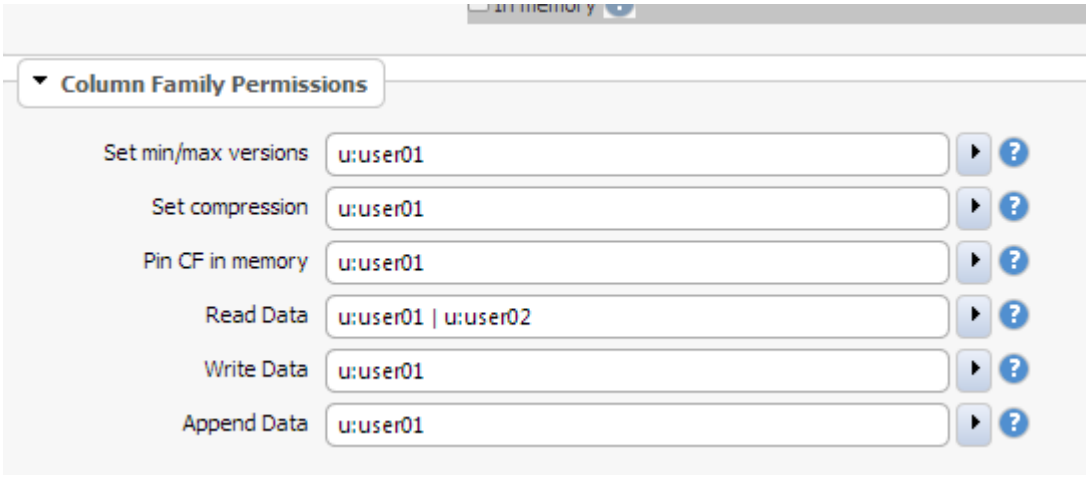
```
$ su user02
$ echo "scan '/user/user01/t_user01'" | hbase shell
```

Shows 0 rows, why ?

8. Give Read Permissions to user02 on column family cf1, then try the scan again.

```
$ maprcli table cf edit -path /user/user01/t_user01 -cfname cf1 -
readperm "u:user01 | u:user02"
$ echo "scan '/user/user01/t_user01'" | hbase shell
```

Shows 1 row: r1, column family cf1, column c1 and c2 values, but not column family cf2. Look at the column family permissions for the table, the image below is for cf1




```
$echo "scan '/user/user01/t_user01', {COLUMNS => ['cf2']}" | hbase shell
```

shows 0 rows.

9. Give Write Permissions to User02 on column family cf1. Insert a new row as user02. Read the new row as user01

```
$ maprcli table cf edit -path /user/user01/t_user01 -cfname cf1 -writeperm "u:user01 | u:user02"

$ su user02

$ echo "put '/user/user01/t_user01', 'r2', 'cf1:c1', 'v21'" | hbase shell

$ su user01

$ echo "scan '/user/user01/t_user01', {COLUMNS => ['cf1']}" | hbase shell
```

Results should show 2 rows 3 values.

10. Column Level Permissions: Allow only user02 read permissions on c1 in cf1

```
$ maprcli table cf edit -path /user/user01/t_user01 -cfname cf1 -readperm "u:root | u:user02"

$ maprcli table cf colperm set -path /user/user01/t_user01 -cfname cf1 -name c1 -readperm "u:mapr| u:user02"

$ su user02

$ echo "scan '/user/user01/t_user01', {COLUMNS => ['cf1']}" | hbase shell
```

11. Results should show 2 rows 3 values.

Access is given by “ANDing” the permissions on the Column Family → Column.

Install and Configure MapR Client

In order to run the Java applications from Eclipse and to operate on the MapR Sandbox cluster, you need to install MapR Client on your laptop and configure it. We recommend performing these steps as it will make the labs easier, but it is not a requirement.

Download and install the MapR client. Then run configure script

- http://doc.mapr.com/display/MapR/Setting+Up+the+Client [For latest Sandbox]

Install & configure MapR Client on Windows

1. Download the appropriate zip file for amd64 or x856

   Default installation directory on Windows is C:\opt\mapr. On a Windows machine, download :

   http://package.mapr.com/releases/

2. Unpack the zip file into the \opt\mapr directory.
3. Edit C:\opt\mapr\hadoop\hadoop-0.20.2\conf\core-site.xml

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl"href="configuration.xsl"?>
<!-- Put site-specific property overrides in this file. -->
<configuration>
  <property>
    <name>hadoop.spoofed.user.uid</name>
    <value>2001</value>
  </property>
  <property>
    <name>hadoop.spoofed.user.gid</name>
    <value>2000</value>
  </property>
  <property>
    <name>hadoop.spoofed.user.username</name>
    <value>user01</value>
  </property>
</configuration>
```

The uid and gid for "user01" has to match for this account on Sandbox. Confirm that the same user (user01) exists on the Sandbox.

You should see the following entry in /etc/password. If not add it in /etc/passw:

```
user01:x:2001:2000::/user/user01:/bin/bash
```

4. Edit the env file: C:\opt\mapr\conf\env.bat

```
@echo off
rem Please set all environment variable you want to be used during MapR cluster
rem namely MAPR_HOME, JAVA_HOME, MAPR_SUBNETS
set JAVA_HOME=C:\Program Files (x86)\Java\jdk1.7.0_51
set MAPR_HOME=C:\opt\mapr
```

5. Create C:\opt\mapr**runsetupvm.bat** [or runsetup.sh on Mac/Linux]. Edit the VM ip address before you run this bat file.

```
set JAVA_HOME=C:\Program Files (x86)\Java\jdk1.7.0_51
set MAPR_HOME=C:\opt\mapr
set PATH=%MAPR_HOME%\bin;%PATH%
set PATH=%MAPR_HOME%\hadoop\hadoop-0.20.2\bin;%PATH%

%MAPR_HOME%\server\configure.bat -c -C <192.168.1.93>:7222
```

6. Change directory to C:\opt\mapr and run the 'runsetupvm.bat' that you just created to setup your laptop as a client to the Sandbox running on the VM at the ip address.

```
cd c:\opt\mapr
runsetupvm.bat
```

Install and configure MapR Client on Mac

See the documentation at MapR site for latest instructions.

1. Download the archive:

   http://package.mapr.com/releases/<version>/mac/

2. Open the **Terminal** application.
3. Create the directory /opt:

   ```
   sudo mkdir -p /opt
   ```

4. Extract the tar file into the **/opt** directory. Example:

   ```
   sudo tar -C /opt -xvf mapr-client-<version>.tar.gz
   ```

Create a user on the sandbox

On the Mac since uid/gid information will flow properly from the operating system to MapR, create a user on the Sandbox that maps to the desktop user you are using on the Mac. This will come in handy should you choose to run jobs directly from Eclipse. You don't have to do this if you prefer to just ssh into the cluster and run tasks there.

1. Determine your Mac user's uid and primary gid using the 'id' command from the Mac shell.
2. Ssh onto the sandbox as root and create that user. For example "useradd –u <mac uid> -g <mac gid> <mac username>". It is unlikely that the group name on the Mac will map properly to the same group name on the sandbox (the match is based upon gid), but that doesn't really matter.
3. Determine the group name on the sandbox that corresponds to the gid from the Mac. Use "grep <gid> /etc/groups" on the Sandbox. If there is a valid group entry that will be just fine even if the name is different. Just remember the name. If there is no group with that gid, it will be easiest if you simply create a group with that name by editing /etc/groups on the Sandbox and create a group with that gid and an appropriate name on the sandbox.
4. On the sandbox as root create a home directory for yourself and ensure proper ownership. For example "hadoop fs –mkdir /user/<username>" and then "hadoop fs –chown <username>:<groupname> /user/<username>". Note that the group name is the group name on the sandbox that has the same gid as the user's group on the Mac.
5. Run configure.sh to configure the client, using the -C (uppercase) option to specify the CLDB nodes, and the -c (lowercase) option to specify a client configuration. Example:

   ```
   sudo /opt/mapr/server/configure.sh -N demo.mapr.com -c -C <ip address>:7222
   ```

6. Optionally update your path:

   ```
   PATH=$PATH:/opt/mapr/bin:/opt/mapr/hadoop/hadoop-<version>/bin
   ```

MapR Client install Verification

1. Verify the cluster configuration file.

On Windows:

```
more %MAPR_HOME%\conf\mapr-cluster.conf
```

On Mac:

```
cat /opt/mapr/conf/mapr-clusters.conf
```

You should see demo.mapr.com and the ip address, for example:

```
#cat /opt/mapr/conf/mapr-clusters.conf
demo.mapr.com secure=false 192.168.56.101:7222
```

2. Run a Hadoop command to verify that the connection is working.

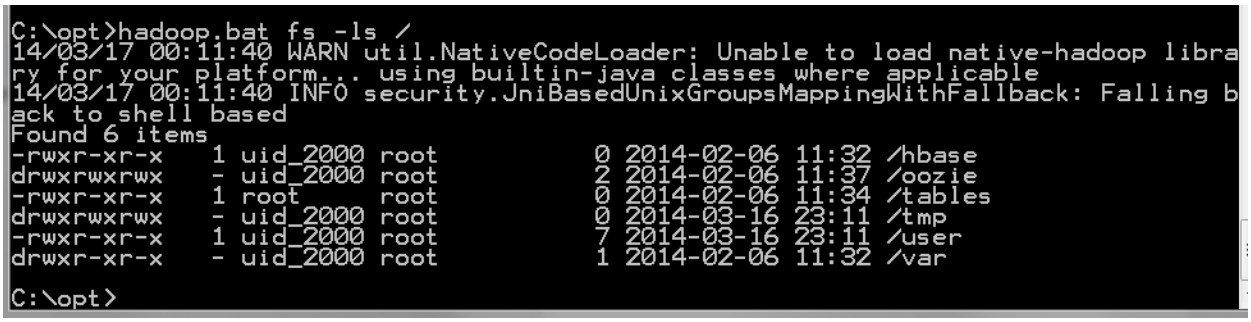
3. Run the following command from the laptop:

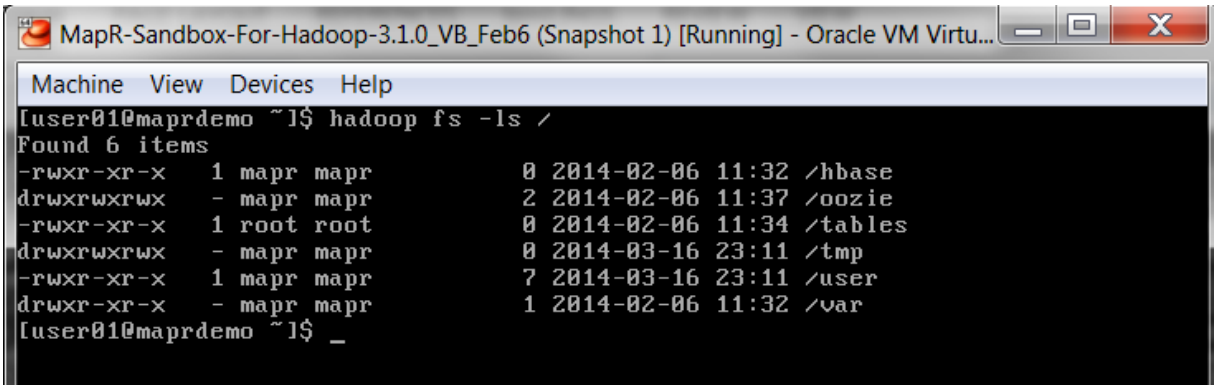
On Windows:

```
hadoop.bat fs -ls /
```

On Mac:

```
hadoop fs –ls /
```



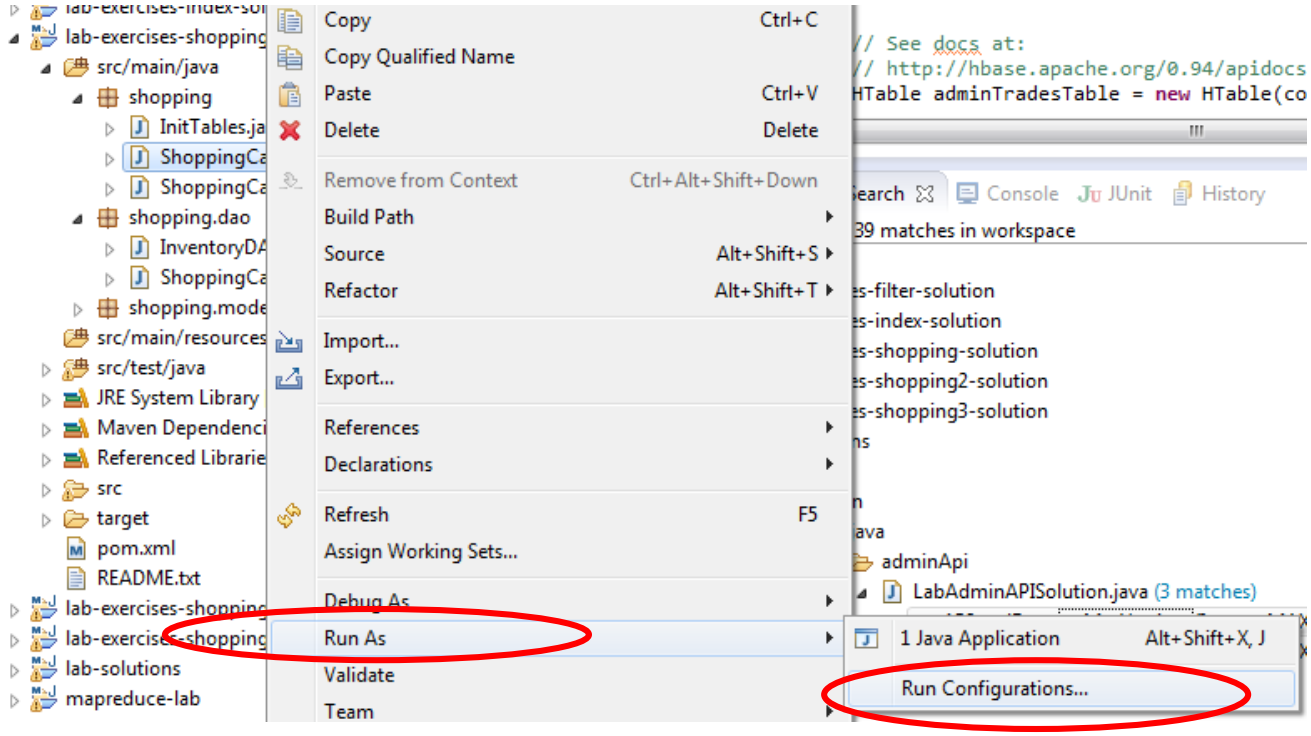


Run the Shopping App from Eclipse

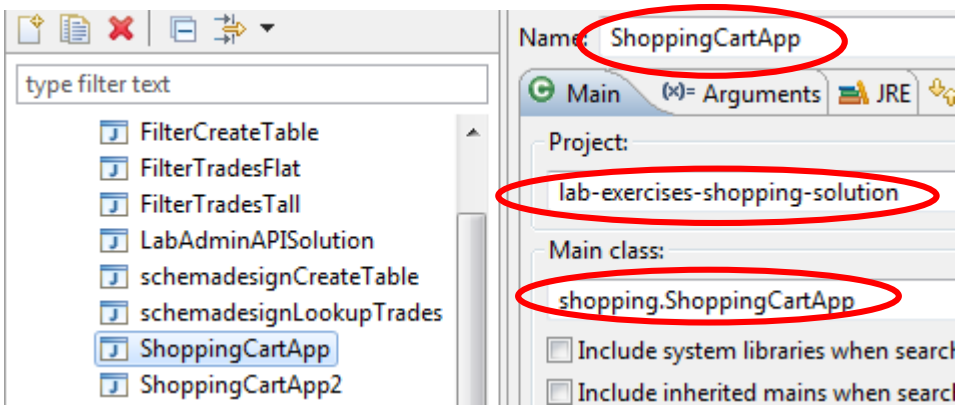
Must have the MapR client installed & configured!

To run the HBase Java application ShoppingCartApp using the MapR client and Sandbox, edit the "Run Configuration". At a conference during tutorial session, just run the Junit test and you can configure the MapR client later.

1. Select the **lab-exercises-shopping** project, package **shopping**, File **ShoppingCartApp**-> Run As -> Run configurations:



2. Click on new launch configuration on the top left.



3. Make sure you have the following for the “Main” tab:

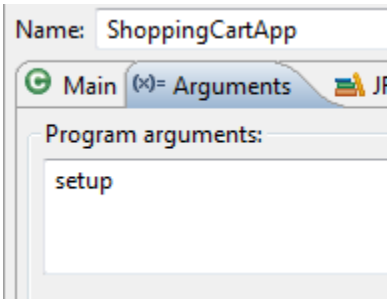
   Name: **ShoppingCartApp**

   Project: **lab-exercises-shopping**

   Main class: **shopping.ShoppingCartApp**

4. Next, Click on the **Arguments** tab and enter ***setup***:

5. And under VM arguments, enter: ***-Djava.library.path=C:\opt\mapr\lib***

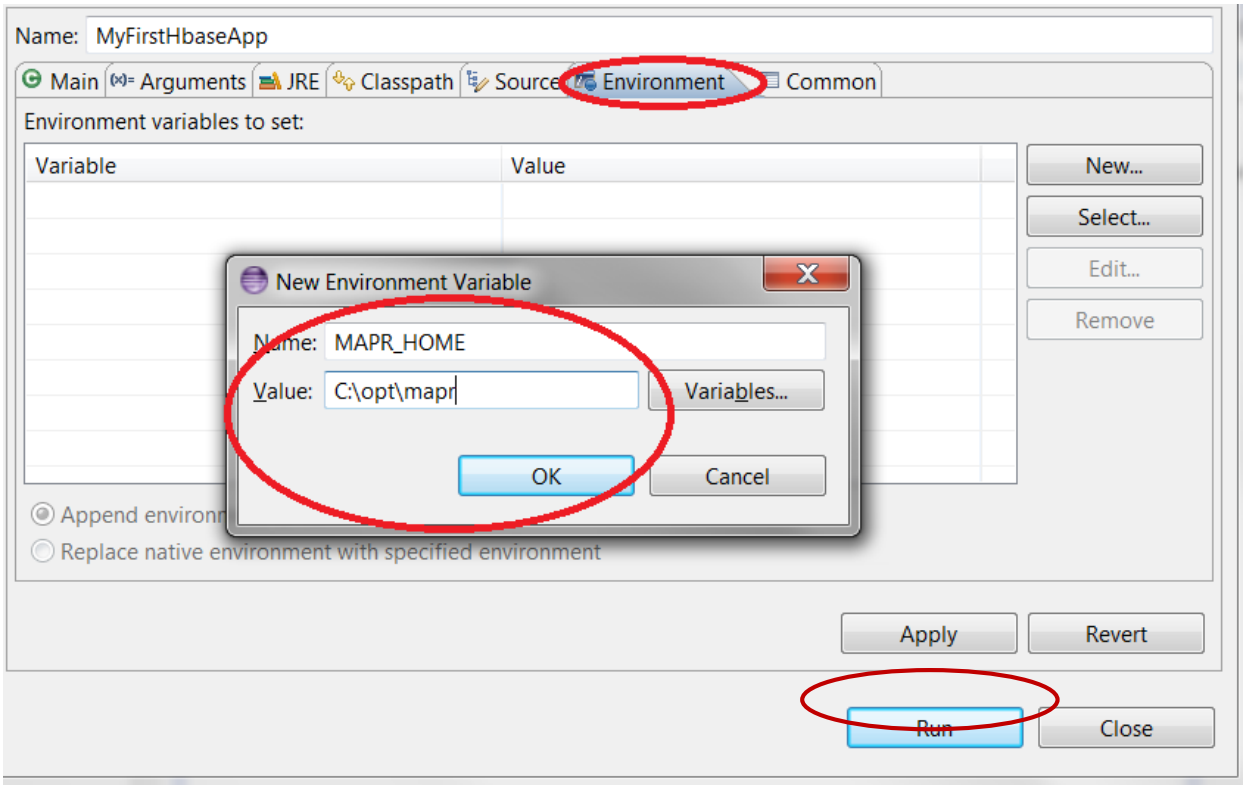


   MAPR_HOME is where you installed MapR Client in step #4 in installation process.

6. Next, Click on **Environment** tab and set ***MAPR_HOME*** to ***C:\opt\mapr***,

7. Click on ***Apply*** and ***Run***:

   MAPR_HOME is where you installed MapR Client in step #4 in installation process.

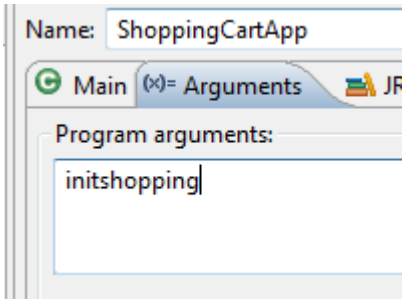


8. **Click on New …**
9. Enter **MAPR_HOME C:\opt\mapr** as shown above.
10. Click on **Apply then click on Run**

Run the Shopping App from Eclipse (if you have the MapR client installed)

Now change the argument for the shopping app

1. Select the lab-exercises-shopping File “ShoppingCartApp”-> Run As -> Run configurations
2. Next, Click on the **Arguments** tab and enter **initshopping**.



3. Click on “Environment” Tab and add variable MAPR_HOME value <where you installed MapR CLinet> C:\opt\mapr31
4. Click apply and run