



DEV 3200 – Apache HBase Applications Design and Build Lab Guide

Winter 2017 – Version 5.1.0

For use with the following courses:

- DEV 3200 – Apache HBase Applications Design and Build
- DEV 320 – Apache HBase Data Model and Architecture
- DEV 325 – Apache HBase Schema Design
- DEV 330 – Developing Apache HBase Applications: Basics
- DEV 335 – Developing Apache HBase Applications: Advanced
- DEV 340 – Bulk Loading, Security, and Performance

This Guide is protected under U.S. and international copyright laws, and is the exclusive property of MapR Technologies, Inc.

© 2017, MapR Technologies, Inc. All rights reserved. All other trademarks cited here are the property of their respective owners.



Using This Guide

Overview of Labs

The table below lists the lab exercises included in this guide. Lab exercises are numbered to correspond to the learning goals in the course. Some learning goals may not have associated lab exercises, which will result in "skipped" numbers for lab exercises.

Lessons and Labs	Duration
Lesson 1: Introduction to Apache HBase <ul style="list-style-type: none">No labs	
Lesson 2: Apache HBase Data Model <ul style="list-style-type: none">Lab 2.4a – Perform CRUD operations with HBase shellLab 2.4b – Create a MapR-DB table using MCS	20 min 20 min
Lesson 3: Apache HBase Architecture <ul style="list-style-type: none">No labs	
Lesson 4: Basic Schema Design <ul style="list-style-type: none">Lab 4.2 – Import data with different row key designsLab 4.3 – Populate and examine trades tall and flat tables	40 min 15 min
Lesson 5: Design Schemas for Complex Data <ul style="list-style-type: none">Lab 5.4a – Model person-relatives schemaLab 5.4b – Model movie rental online store schemaLab 5.4c – Model customer click event or action	20 min 20 min 20 min
Lesson 6: Query HBase with Hive <ul style="list-style-type: none">Lab 6.1a – Use Hive with the airlines HBase tableLab 6.1b – Use Hive to query the trades table	15 min 10 min
Lesson 7: Java Client API Part 1 <ul style="list-style-type: none">Lab 7.1 – Import, build, and run “lab-exercises-shopping” projectLab 7.2 – Insert and get data in the ShoppingCartDAO classLab 7.3 – Delete data in the ShoppingCartDao class	20 min 20 min 20 min
Lesson 8: Java Client API Part 2 <ul style="list-style-type: none">Lab 8.2 – Work with shoppingcart application put, list, and batchLab 8.5 – Work with shoppingcart application checkout	30 min 20 min
Lesson 9: Java Client API for Administrative Features <ul style="list-style-type: none">Lab 9.3 – Working with LabAdminAPI in lab-exercises project	60 min

Lessons and Labs	Duration
Lesson 10: Advanced HBase Java API	
• Lab 10.1 – Java applications using filters	30 min
• Lab 10.2 – Java applications using increment	50 min
• Lab 10.3a – Examine two different schema design options	20 min
• Lab 10.3b – Finish code and run unit test	30 min
Lesson 11: Working with MapReduce on HBase	
• Lab 11.3a – Developing MapReduce applications for HBase (flat-wide)	50 min
• Lab 11.3b – Developing MapReduce applications for HBase (tall-narrow)	50 min
Lesson 12: Bulk Loading of Data	
• Lab 12.2 – Use ImportTsv and CopyTable	30 min
• Lab 12.3 – Use a custom MapReduce program to bulk load data	30 min
Lesson 13: Performance	
• Lab 13 – Install and use YCSB	30 min
Lesson 14: Security	
• Lab 14.2 – Create MapR-DB tables and set permissions	45 min

Course Sandbox

For instructor-led training, clusters are provided to students through the MapR Academy lab environment. Students taking the on-demand version of the course must download one of the MapR Sandboxes listed below to complete the lab exercises. See the *Connection Guide* provided with your student materials for details on how to use the sandboxes.

- VMware Course Sandbox: <http://package.mapr.com/releases/v5.1.0/sandbox/MapR-Sandbox-For-Hadoop-5.1.0-vmware.ova>
- VirtualBox Course Sandbox: <http://package.mapr.com/releases/v5.1.0/sandbox/MapR-Sandbox-For-Hadoop-5.1.0.ova>



CAUTION: Exercises for this course have been tested and validated **ONLY** with the Sandboxes listed above. *Do not* use the most current Sandbox from the MapR website for these labs.

Icons Used in This Guide

This lab guide uses the following icons to draw attention to different types of information:



Note: Additional information that will clarify something, provides details, or helps you avoid mistakes.





CAUTION: Details you **must** read to avoid potentially serious problems.



Q&A: A question posed to the learner during a lab exercise.



Try This! Exercises you can complete after class (or during class if you finish a lab early) to strengthen learning.

Command Syntax

When command syntax is presented, any arguments that are enclosed in chevrons, `<like this>`, should be substituted with an appropriate value. For example this:

```
# cp <source file> <destination file>
```

might be entered by the user as this:

```
# cp /etc/passwd /etc/passwd.bak
```



Note: Sample commands provide guidance, but do not always reflect exactly what you will see on the screen. For example, if there is output associated with a command, it may not be shown.



Caution: Code samples in this lab guide may not work correctly when cut and pasted. For best results, type commands in rather than cutting and pasting.





DEV 320 – Apache HBase Data Model and Architecture

Part of the DEV 3200 curriculum

Lesson 2: Apache HBase Data Model

Lab Overview

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

Background Information on HBase Shell

HBase is the Hadoop database, which provides random, real-time 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 interacting with HBase.

You can get help with the shell commands here: <https://learnhbase.wordpress.com/2013/03/02/hbase-shell-commands/>

To start the shell at the command line type:

```
$ hbase shell
```

Once connected, you can get help on commands:

```
hbase> help
```

Lab 2.4a: Perform CRUD Operations with HBase Shell

Estimated time to complete: 20 minutes

In this exercise, you will use HBase shell commands to create the `customer` table with the data for the column families `address` and `order`, and the columns `city` and `state`, as shown below.

Table: `customer`

row-key userid	addr		order	
	city	state	date	numb
jsmith	nashville	TN	01/01/2015	12345
bjones	miami	FL	02/02/2015	56565

1. Log into your node as the user `user01` with the password `mapr`.

2. Start the hbase shell:

```
$ hbase shell
```

3. Create a table in your home directory with two column families.

```
> create '/user/user01/customer', 'addr', 'order'
```

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

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

5. Execute the following statements to insert and get some records.

a. Put some data into the table:

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

b. Retrieve the data for `jsmith`:

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

c. 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'
```

d. Retrieve the data for `jsmith`:

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

e. Note that this gets all the data for the row. Limit this to the `addr` column family:

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

f. Limit this to the `numb` column:

```
> get '/user/user01/customer', 'jsmith', 'order:numb'
```

6. Alter the table to store more versions in the `order` column family:

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

7. Use `describe` to get the description of the table and notice the updated version number:

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

8. 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'
```

9. Get order number column cells:

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



10. 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}
```

11. Put more data for different rowkey users:

```
> 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'
```

12. Use `scan` to retrieve rows of data for the table:

a. Retrieve all rows, all columns:

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

b. Retrieve all rows for `addr` column family:

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

c. Retrieve all rows for `order` number column, 5 versions:

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

d. Retrieve rows with rowkey from `njo` onwards for `addr` column family:

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

e. Retrieve rows with rowkey starting with `j`, stopping before `t`:

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

f. Retrieve rows with rowkey starting from `a` onward:

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

g. Retrieve rows with rowkey starting from `t` onward:

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

13. Use `count` to retrieve the number of rows in the table:

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



14. Delete data from the table. First let's see the data in the row with key `njones`:

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

15. Delete a column and check deletion:

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

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

16. Delete a column family and check deletion:

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

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

17. Delete a row and check deletion:

```
> deleteall '/user/user01/customer', 'jsmith'
```

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

Lab 2.4b: Create a MapR-DB Table using MapR Control System (MCS)

Estimated time to complete: 20 minutes

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.

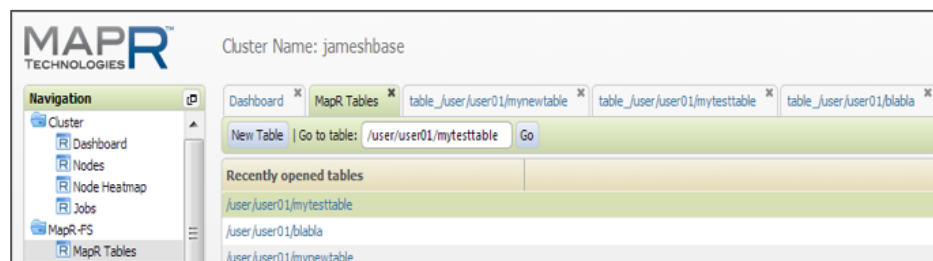
Connect to MapR Control System and See Table Properties

1. Connect to the MapR Control System (MCS) from a browser, using the information in the connection guide or information from your instructor for host information. Login as user `mapr` with a password of `mapr`.

```
https://<ipaddress>:8443
```

2. Use the MCS to connect to the table `/user/user01/customer` and see how many rows and regions there are for this table. Also note the properties for the column families.

- a. In the **Navigation** panel on the left hand side, navigate to **MapR-FS > MapR Tables**.
- b. In the **Go To Table** text box enter `/user/user01/customer` as shown in the image. Then click **Go**.

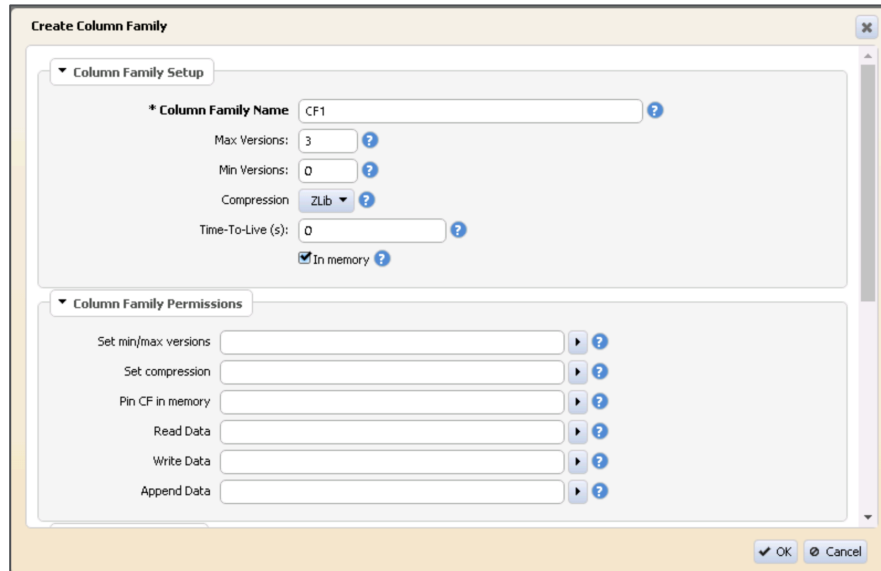


- c. Review the information displayed. You should see the size of the regions, the two column families, and their properties.



Create a Table Using MCS

1. Navigate to the **MapR Tables** tab in the MCS.
2. Click **New Table** to create a table called `user/user01/MCSNewTable` and click **OK**.
3. MCS will open a new tab called `table_/user/user01/MCSNewTable`. Click **New Column Family** and create a column family with your choice of name (for example, `CF1`). Change the **Max Versions**, **Compression**, and **Time-To-Live**. MCS lets you edit and delete tables without disabling them.



4. Click **OK**.

Delete a Table with the Shell

1. Before you delete a table via the shell, you must disable it. From the HBase shell on your node, disable the table:

```
> disable '/user/user01/customer'
```

2. Drop the table to delete it:

```
> drop '/user/user01/customer'
```

Bonus Activity

1. Create a table and pre-split it into four regions and then import data. Any data put into this table in the future will be automatically assigned to an existing region. For example, row keys starting with G will be placed in the second region, and row keys starting with Q will be placed in another.

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



2. See the description of the table and notice the Column Family created:

```
> describe '/user/user01/mynewtable'
```
3. Go back to the MCS and navigate to the **MapR Tables** page.
 - a. In the **Go to Table** search bar at the top of the page, type `/user/user01/mynewtable` and press **Go**.
 - b. Notice the new tab showing details of the table. In the **Regions** information section, notice the four existing regions made when the table was created.





DEV 325 – Apache HBase Schema Design

Part of the DEV 3200 curriculum

Lesson 4: Basic Schema Design

Lab 4.2: Import Data with Different Row Key Designs

Estimated time to complete: 40 minutes

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

Exercise 4.2a: Import Data with Date as Row Key

1. Download lab files to the cluster. Use `wget` to download the material on to the cluster:

```
$ wget http://course-files.mapr.com/DEV3200/DEV325-v5.2-Lab4.zip
```

2. After the zip file has been downloaded, use the `unzip` command to unzip it.

```
$ unzip DEV325-v5.2-Lab4.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 Lab_4_schema_import_lab/ontime.csv
```

You should see rows of output. Each row will appear similar to the following:

```
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 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 Lab_4_schema_import_lab/*.sh
$ mkdir data
$ mkdir tables
$ mv Lab_4_schema_import_lab/ontime.csv data/.
```

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

```
$ cd Lab_4_schema_import_lab/
$ more import1.sh
```

The script defines the variable for the user ID, `ME=user01`. Change this if you are not `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,
$CF:aircrafterdelay,$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
```

Run the script to import data:

```
$ sh 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: 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.



Q: Do you see a problem with this row key?

A: Yes – it could lead to hotspotting.

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

```
$ hbase shell
hbase(main):001:0> count '/user/user01/tables/airline'
31 row(s) in 0.0560 seconds
```



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

A: 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 row key 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.



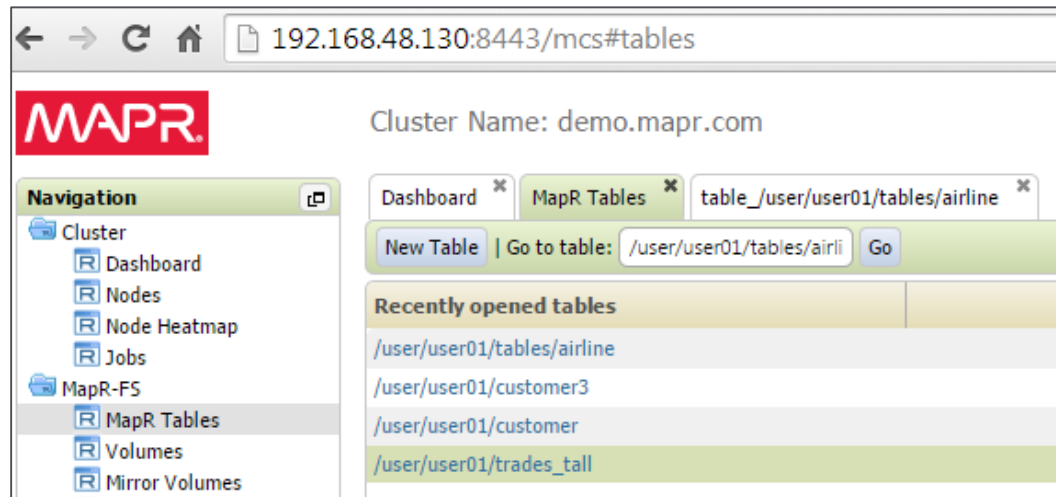
Note: With the standard `ImportTsv` tool as it stands today, you cannot easily create composite keys because it only allows you to select one field for the row key.



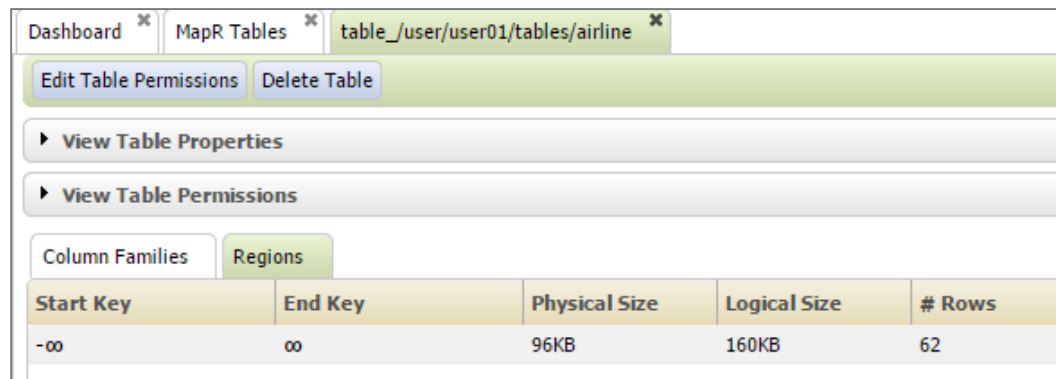
Using MapR Control Server (MCS) to View Tables

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 **Go to table** text box. Click **Go**.



2. Look in the **Regions** tab. It is normal if the MCS doesn't give an exact calculation for row count as it is an estimate and can include deleted versions.



Start Key	End Key	Physical Size	Logical Size	# Rows
-∞	∞	96KB	160KB	62

Queries using HBase Shell

Try out some queries on the table using the HBase shell (you can find HBase shell query info at <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 4.2b: 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. Make sure you are in the `Lab_4_schema_import_lab` directory, then execute the following commands:



```
$ 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

```
$ 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

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

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



Q: Using this row key, what can you retrieve?

A: 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.



Q: Would you query by carrier or destination?

A: 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 4.2c: Import Data with a Better Composite Row Key

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

- 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 row key:

```
-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)

- Using the HBase shell, scan one row of data.

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



Note: 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.

- 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	<pre>scan '/user/user01/tables/airline', {FILTER => "ValueFilter(=, 'binary:239.00')", LIMIT => 10}</pre>
--	---

Scan to get all row data where cancellation codes = 1	<pre>scan '/user/user01/tables/airline', {FILTER => "SingleColumnValueFilter('cf1','cnc1', =, 'binary:1.00')", LIMIT => 10}</pre>
---	---

Scan to get any column name with a prefix of cnc	<pre>scan '/user/user01/tables/airline', {FILTER => "ColumnPrefixFilter('cnc')", LIMIT => 10}</pre>
---	---

Scan with a start and stop row and a prefix filter on the column name	<pre>scan '/user/user01/tables/airline', {STARTROW => 'AA-1-2014-01-01-JFK-LAX', STOPROW => 'AA- 10', FILTER => "ColumnPrefixFilter('cnc')", LIMIT => 10}</pre>
---	---

Perform an HBase scan to find flights with a delay > 200.0	<pre>scan '/user/user01/tables/airline', {COLUMNS => ['cf1:carrierdelay'], FILTER => "(SingleColumnValueFilter('cf1','carrierdelay', =, 'regexstring:^(2-9){3}')", LIMIT => 10}</pre>
--	--



Q: What is the advantage of this row key compared to the previous two row key designs?

A: This will not hotspot, and the grouping by carrier is useful for scans.



Exercise 4.2d: 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

```
$ more import4.sh
```

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:aircrafterdelay=32.00,
```

2. Execute the `import4.sh` script:

```
$ ./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}
```



Q: What is the advantage of separating columns into different column families?

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

Lab 4.3: Populate and Examine Trades Tall and Flat Tables

Estimated time to complete: 15 minutes

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 SYM_TIME	CF1	
	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_20131 020	@ts2: 12.34 @ts1: 12.00						@ts2: 1000 @ts1: 2000				



CSCO_2013 1023					@ts3: 1.23					@ts3: 3000
GOOG_2013 0817			@ts6: 2.34					@ts6: 1000		

Exercise 4.3: Populate Trades Tall Table

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

1. Navigate back to the user's home directory and into the other file, which was downloaded earlier.

```
$ cd /user/user01
$ ls
```

Notice the two files unzipped earlier: Lab_4_schema_import_lab and Lab_4_trades_table_lab.

```
$ cd Lab_4_trades_table_lab
$ ls
```

2. Run the CreateTable program as follows to create the tall table:

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

3. Run the CreateTable program as follows to create the flat table:

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

4. Try out some queries in HBase shell. Scan one row of each table using the HBase shell:

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



Lesson 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.

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:

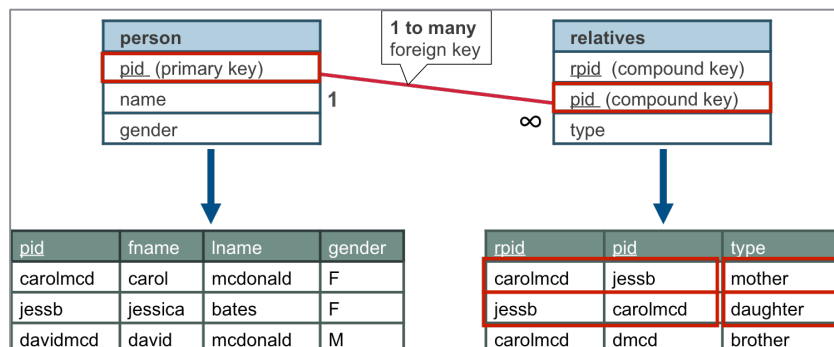
1. **Identify Entities:** What are the entities in each scenario? What information do we want?
2. **Identify Queries:** Identify the queries in each use case.
3. **Identifying Attributes:** Attributes that can be used to identify unique instances of the entity.
4. **Non-identifying Attributes:** Other 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 5.4a: Model Person-Relatives Schema

Estimated time to complete: 20 minutes

Review the use case and answer the questions. Look in the Answer Key for further information.

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



1. Identify the entities.
2. What type of relationship exists between the entities?
3. What information do you want to retrieve? i.e. Identify the queries?
4. What are the identifying attributes?
5. What are the non-identifying attributes?
6. How are you going to model the entity relationship?
7. Do you need to use a lookup table? If so, what would it be?

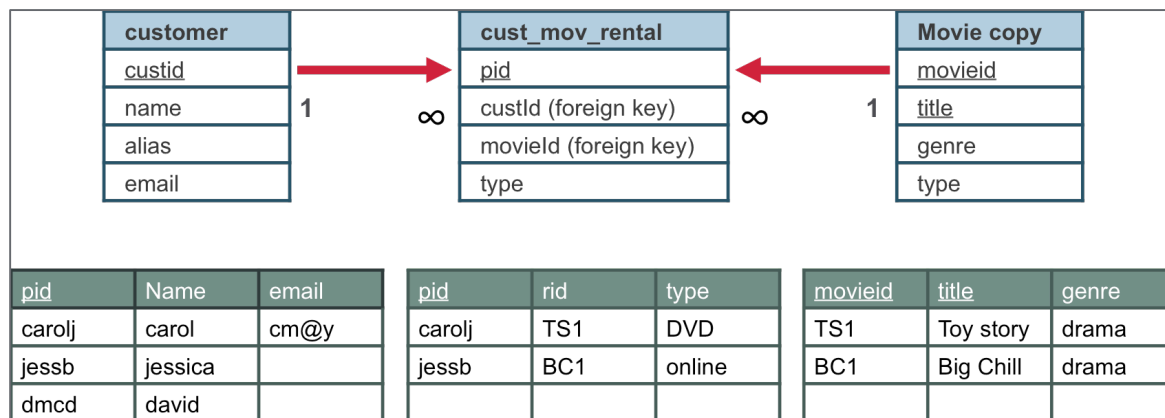
Based on your answers to the above:

8. How would you design the row key?
9. What would you define as your column family(ies)?
10. What would your columns be?

Lab 5.4b: Model Movie Rental Online Store Schema

Estimated time to complete: 20 minutes

Figure 2: Use Case 2 – Movie Rental Relational Model



1. Identify the entities.
2. What type of relationship exists between the entities?
3. What information do you want to retrieve? i.e. Identify the queries?
4. What are the identifying attributes?
5. What are the non-identifying attributes?
6. How are you going to model the entity relationship?
7. Do you need to use a lookup table? If so, what would it be?



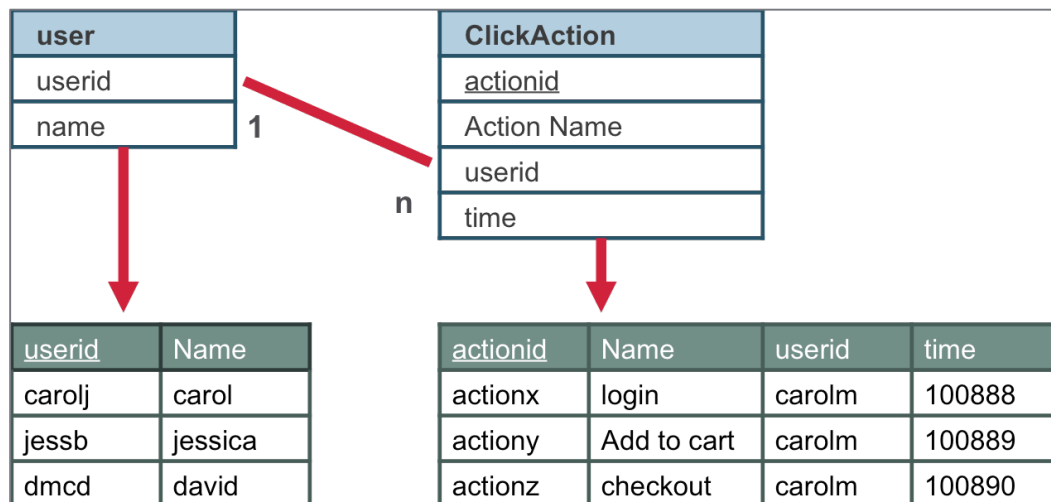
Based on your answers to the above:

8. How would you design the row key?
9. What would you define as your column family(ies)?
10. What would your columns be?

Lab 5.4c: Model Customer Click Event or Action

Estimated time to complete: 20 minutes

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



We 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
2. What type of relationship exists between the entities?
3. What information do you want to retrieve? i.e. Identify the queries?
4. What are the identifying attributes?
5. What are the non-identifying attributes?
6. How are you going to model the entity relationship?
7. Do you need to use a lookup table? If so, what would it be?

Based on your answers to the above:

8. How would you design the row key?
9. What would you define as your column family(ies)?
10. What would your columns be?



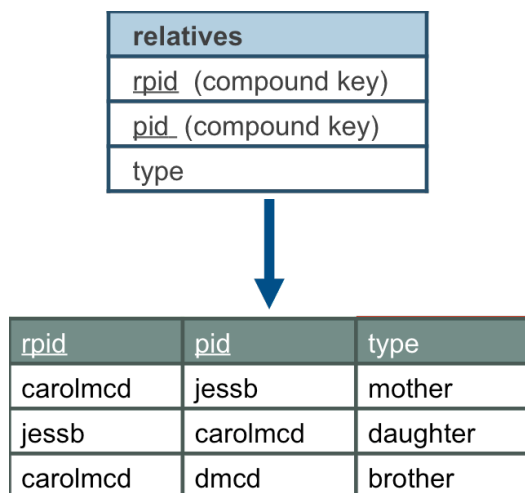
Lesson 5 Answer Key

Lab 5.4a – Model Person-Relatives Schema

See below for the answer to the questions posed in this lab.

1. *Identify the entities.* The entity will be **Person**.
2. *What types of relationship exists between the entities?* It is a one-to-many relationship, since a person can have more than one relative.
3. *What information do you want to retrieve?* You 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 all relations, all brothers, all daughters, and so on.
4. *What are the identifying attributes?*
 - Option A: pid
 - Option B: pid, first name, last name
5. *What are the non-identifying attributes?*
 - Option A: gender, first name, last name
 - Option B: gender
6. *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 becomes the row key and the child identifying attributes are column qualifiers.

The child identifying attributes in the case are rpid and type:



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

5

Option A
Child identifying attribute = type

Info column family relation column family

Person table

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

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



Q: Can you think of any problems you might encounter with this model?

A: Since there can be 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 = pid

5

Option B
Child identifying attribute = rpId

Info column family relation column family Dynamic column name

Person table

rowkey	info: fname	info: lname	info: gender	...	relation: jessicab	relation: sarahb	relation: caroljmc	relation: davidmcd	...
pid									
caroljmc	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.

7. Do you need to use a lookup table? If so, what would it be? You can also use a lookup table to model the relationship. Look at the figure below.



6 Secondary index (foreign keys)
– Put identifying data in lookup table

Person table

rowkey	info: fname	Info: lname	Info: gender
caroljmc	carol	mcdonald	F
jessicab	jessica	bates	F
...			

Relatives lookup table

rowkey	Info:ts
caroljmc_mother_jessicab	125666
caroljmc_mother_sarahb	125675
caroljmc_brother_davidmcd	
jessicab_daughter_caroljmc	
jessicab_sister_sarahb	

Requires querying 2 tables
Good solution if relationships (relatives) need frequent updating

Here we have two tables – the Person tables 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.

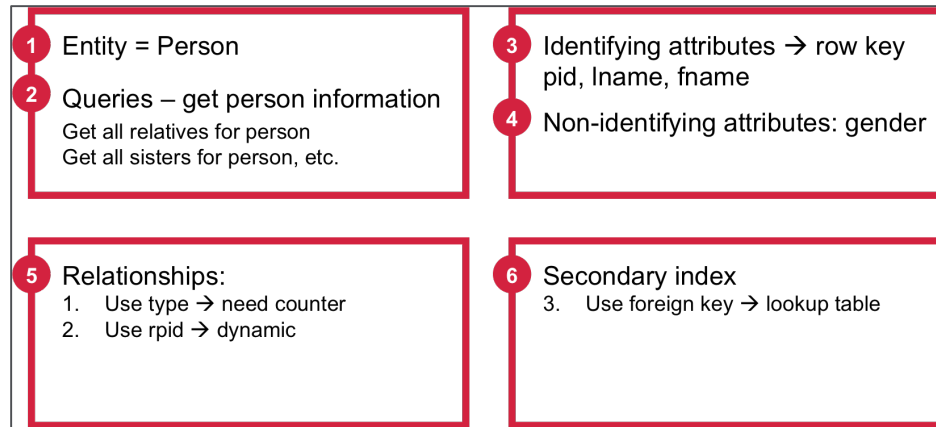
A disadvantage of this solution is if you want to get the relatives for a user and the user information, it would require two queries. An advantage of this solution is if the relationships between the users change frequently, it is easier to add and delete rows from the relative lookup table than to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

8. *How would you design the row key?* There are different ways to design the row key. Based on our steps we saw a few choices:
- Option A: pid
 - Option B: pid + lname + fname

With option B, you can scan by first name and last name, too.

9. *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 6 and 7, 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 5.4b – Model Move Rental Online Store Schema

See below for the answer to the questions posed in this lab.

1. *Identify the entities.* The entities here are **customer** and **movies**.
2. *What types of relationship exists between the entities?* This is a many-to-many relationship. A customer can rent many movies, and a movie title can be rented by many customers.
3. *What information do you want to retrieve?*
 - 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
4. *What are the identifying attributes?*
 - Customer table: custid
 - Movie table: movieid
5. *What are the non-identifying attributes?*
 - Customer table: name, gender, email, etc.
 - Movie table: title, description, genre, etc.
6. *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 moves (entity2) as columns. Similarly, for the Movie table, the Info column family has the non-identifying attributes for the Move entity as columns, and the rental column family has the identifying attribute for the customer entity as columns.



Customer table						
rowkey	info: name	Info: gender	...	rental: movieid1	rental: movieid2	...
custid1	dvd					
custid2	online					
Movie table						
rowkey	info: title	Info: description	...	rental: custid1	rental: custid2	...
movieid1	dvd					
movieid2	online					

- 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.
- How would you design the row key? See the figure above.
- What would you define as your column family(ies)? See the figure above.
- What would your columns be? See the figure above.

Lab 5.4c – Model Customer Click Event or Action

See below for the answer to the questions posed in this lab.

- Identify the entities. The entity will be **user**.
- What types of relationship exists between the entities? It is a one-to-many relationship.
- What information do you want to retrieve? Get users's name; get the latest 10 actions performed by a particular user; get the details of actions performed in a group of 10 actions at a time.
- What are the identifying attributes? The identifying attribute is userid.
- What are the non-identifying attributes? Non-identifying attributes are first and last name, etc.
- 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.
- 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.



Q: What purpose does the reverse timestamp in the row key serve?

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



rowkey	info: fname	Info: lname
userid	carol	mcdonald

rowkey	info:name
userid+reversetimestamp+actionid	Action name
carolm+9999950+checkid	Check out
carolm+9999867+selid	Select item
carolm+9999888+logid	login

8. *How would you design the row key?* See the figure above.
9. *What would you define as your column family(ies)?* See the figure above.
10. *What would your columns be?* See the figure above.



Lesson 6: Query HBase with Hive

Lab 6.1a: Use Hive with the airlines HBase table

Estimated time to complete: 15 minutes

If you have not already created and populated the airlines HBase table, refer to Lab 4.2d to do that first. The following diagram shows the airlines table:

Row key carrier-flightnumber-date-origin-destination-	delay			info			stats		timing	
	Air Craft delay	Arr delay	Carrier delay	cncl	cncl code	tail num	distance	elapsed time	arrival time	Departure 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.

```
> hbase shell
```

```
> 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:elapsedtime, 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, 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
```



Note: Since the queries we will use with the Airlines table are more complex, all of the queries are in the file `Lab6.2-AirlinesQueries.txt`. You can download this to the desktop through your browser, or to the cluster with `wget`, as the file:

```
http://course-files.mapr.com/DEV3200/DEV325-v5.2-Lab6Ref.zip
```

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:de
pdelay,delay:weatherdelay,info:cncl,info:cnclcode,info:tailnum,stats
:airtime,stats:distance,stats:elaptime,timing:arrtime,timing:deptime
```



```
,timing:dom,timing:dow,timing:month")
TBLPROPERTIES("hbase.table.name" = "/user/user01/tables/airline");
```

Exploring the Flight Table Using Hive Queries

1. Select a few rows from the `flighttable`:

```
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 MapReduce 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.



Q: How many MapReduce jobs will be run?

A: A simple way to find this out is to insert the word `EXPLAIN` at the beginning of the select statement.

2. Type this query:

```
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 two MapReduce 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

```

- 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
737.77777777777777

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 6.1b: Stretch Lab – Use Hive to Query the Trades Table

Estimated time to complete: 10 minutes

If you have not already created and populated the `trades_tall` HBase table, refer to Lab 4.3 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:

```
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 the data.

```
hive> DESCRIBE trades;

hive> SELECT * FROM trades LIMIT 10;

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>





DEV 330 – Developing Apache HBase Applications: Basics

Part of the DEV 3200 curriculum

Lesson 7: Java Client API Part 1

Lab Overview

This lab shows how to create HBase Tables and to use the HBase Java API to insert data into tables, and then perform get, scan, and delete operations.

Lab 7.1: Import, Build, and Run “lab-exercises-shopping” Project

Estimated time to complete: 20 minutes

In the first lab exercise, we'll import the exercise project “lab-exercises-shopping” into NetBeans, review the code, build the project using Maven and run unit tests.

Exercise 7.1a: Import project and build

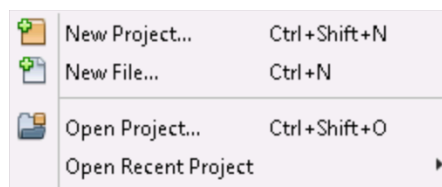
1. Open the browser and navigate to the following links to download the lab exercises, lab solutions, and a command reference that can be used as needed:

`http://course-files.mapr.com/DEV3200/DEV330-LabExercises.zip`

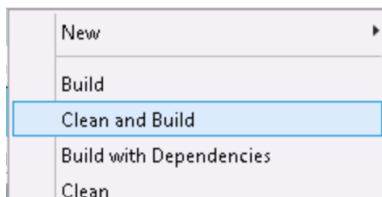
`http://course-files.mapr.com/DEV3200/DEV330-LabSolutions.zip`

`http://course-files.mapr.com/DEV3200/DEV330-CommandRef.zip`

2. When the zip files have downloaded, open Windows Explorer and **Extract all**. Copy the directories and files into **Documents**.
3. Open NetBeans, click **File** in the top left hand of the screen, then **Open Project**. Navigate to where you saved the exercises and open the lab-exercises-shopping project:



4. To build labs from NetBeans, right click the project and choose **Clean and Build** as shown in the figure below.



When finished, the console should output **BUILD SUCCESS** in green text, with a total time and a final memory count.

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.java` – a model object for inventory data
- `ShoppingCart.java` – a model object for shopping cart data

Exercise 7.1b: Run the Inventory Unit Test

1. Look in the **Test Packages** directory at `shopping.TestInventorySetup`.
2. Run the `TestInventorySetup` JUnit Test. Right click the class, and select **Test File**.
3. When finished, the console should output **BUILD SUCCESS** in green text, with a total time and a final memory count.
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]
```

Lab 7.2: Insert and Get Data in the ShoppingCartDAO Class

Estimated time to complete: 20 minutes

Exercise 7.2a: Put and Get Data in Shoppingcart Table

1. Look in the **Test Packages/shopping/** at `shopping.TestShoppingCartSetup`.
2. Look for `TODO 2a`. Uncomment the first unit test `@Test` line by removing the `//` before `@Test`.



- Look in the **Source Packages** directory at **shopping.dao.ShoppingCartDAO.java**. Look for `TODO 2a` 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 click on the class and select 'Test File'. 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			

Exercise 7.2b: Complete code for “TODO 2b”

Now you will complete the code with comments `//TODO 2b` to scan rows in the **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 **Test Packages** directory at **shopping.TestShoppingCartSetup**. Look for `TODO 2b`. Uncomment the second unit test `@Test` line.

This test calls the `ShoppingCartDAO addShoppingCart(String cartId, long pens, long notepads, long erasers)` method three times to put three 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 **Source Packages** directory at `shopping.dao.ShoppingCartDAO.java`. Look for `TODO 2b` in the `ShoppingCartDAO`. You need to finish the code following the `TODO` comments.

Run the 2b JUnit test

Run the 2b JUnit test in shopping, **TestShoppingCartSetup**. Right click on the **TestShoppingCartSetup** class and select **Test File**. 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
```

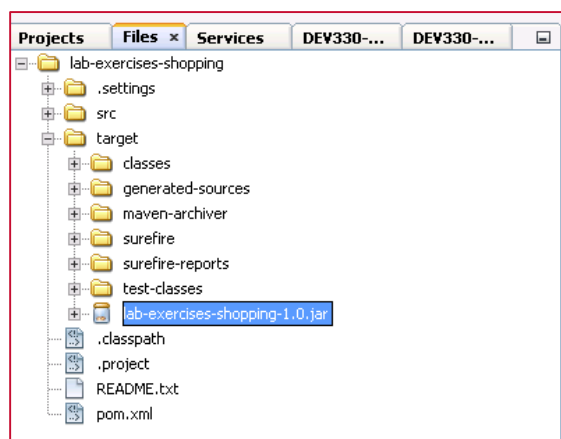
Run the 2c JUnit test

Uncomment the 2c unit test @Test line in TestShoppingCartSetup.java. Then run the JUnit test. In **TestShoppingCartSetup**, right click on the class and select **Test File**.

This test calls the `ShoppingCartApp.saveShoppingCartData(dao)` and `ShoppingCartApp.printShoppingcartTable(dao)` methods, which are the same methods that get called when you run `java -cp `hbase classpath` :./lab-exercises-shopping-1.0.shopping.ShoppingCartApp initshopping` on the cluster. The test should print out debug information.

Run the code on the cluster

1. Save any modified files.
2. To build labs from NetBeans, select the project **lab-exercises-shopping** and click **Clean and Build**. This will create a jar file in the target folder as shown here:



3. Copy the jar file "lab-exercises-shopping-1.0.jar" from the target folder to the cluster. Once you have built the project, upload the jar file to the cluster with WinScp to your user account.
4. 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 class path, 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 `init`. `java -cp `hbase classpath`` shown in the command below uses the `hbase classpath` utility to set the HBase dependencies in the Java class path.

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



Note: If you see the following warning messages, you can just ignore it, or 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
```

5. See what tables are created, data saved, and what data you have. The code prints debug information:

```
$ java -cp `hbase classpath`:./lab-exercises-shopping-solution-
1.0.jar shopping.ShoppingCartApp init

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

-----

Inserting rows in Inventory Table:

mkPut [Inventory [key=pens, quantity=9]]
mkPut [Inventory [key=notepads, quantity=21]]
mkPut [Inventory [key=erasers, quantity=10]]
...
*****
print Inventorys from Table ...
Inventory [key=erasers, quantity=10]
Inventory [key=notepads, quantity=21]
Inventory [key=pens, quantity=9]

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]]

...

*****

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]

```

Lab 7.3: Delete Data in the ShoppingCartDAO Class

Estimated time to complete: 20 minutes

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 **Test Packages** at **shopping.TestShoppingCartDelete**. Look for TODO 3a. 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 **Source Packages** directory at shopping.dao.ShoppingCartDAO.java. Look for TODO 3a in the ShoppingCartDAO. Finish the code following the TODO 3a comments.

Run the 3a unit test

Run the JUnit test: right click on the test class **shopping.TestShoppingCartDelete** and select **Test File**. 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 Mike`. The test should print out debug information.

Run the Shopping App on the cluster

1. Save any modified files.
2. Build the project by right clicking on the class and selecting **Clean and Build**.
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
```



Lesson 8: Java Client API Part 2

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.

Implement checkout functionality with the Shoppingcart application

As part of this application, we have two tables as shown below:

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:

1. There is one `Shoppingcart` per user (customer) at any given time.
2. When the user places an order or checkout, do the following:
 - a. Get the items for the user from the `shoppingcart`.

- b. For each item for the user, get inventory and make sure we have enough quantity for the user.
- c. 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.
- d. Remove the `shoppingcart` entry for the user.

Let's say Mike wants to check out the items. The Inventory table will look like the following after the checkout operation:

Table: Inventory

	cf:	
	stock	
	quantity	Mike
pens	9 8	1
notepads	21 19	2
erasers	10 7	3

Lab 8.2: Work with Shoppingcart Application put, list, and batch

Estimated time to complete: 30 minutes

Open Project and Build

If you have not already done so, use the following steps to open the project `lab-exercises-shopping2` into NetBeans and build the project using Maven.

1. Open the `lab-exercises-shopping2` project into NetBeans. Select **File > Open project**, then navigate to folder where projects are saved and choose **lab-exercises-shopping2**.
2. Build the project using Maven: Right-click project **lab-exercises-shopping2** and select **Clean and Build**. See section on Opening Projects into NetBeans.

Put data in the Inventory table

1. Look in the **TestPackages** 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.



4. Look for `TODO 1a` in `TestInventoryList` and `InventoryDAO`. Finish the code following the comments.
5. Run the 1a JUnit test
 - a. Right click on the class and select **Test File**.
 - b. 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 **Test Packages** directory for **shopping.TestInventoryList**.
 - a. Look for `TODO 1b`. Uncomment the `1b test @Test` line. This test creates a list of puts then calls `InventoryDAO.putInventoryBatch()` to put this list of puts in the table.
2. Look for `TODO 1b` in `InventoryDAO`. Finish the code following the comments.

Run the 1b JUnit test

1. Right click on the class and select **Test File**.
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 JUnit test.

1. Uncomment the `1c unit test @Test` line.
2. Run the JUnit test.
3. Right click on the class and select run **Test File**.

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 the cluster

Use the following steps to run the Shopping App on the cluster.

1. Save any modified files.
2. Build the project using Maven: Select project **lab-exercises-shopping2 > Clean and Build**.
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.

Lab 8.5: Work with Shoppingcart Application checkout

Estimated time to complete: 20 minutes

Use the following steps for this exercise:

Finish check out code

1. Look in the **Test Packages** directory for **shopping.TestCheckout**.
2. Look for `TODO 2a`. Uncomment the `2a test @Test` line.
3. Look for `TODO 2a` in the `Test ShoppingCartApp` and `InventoryDAO`.
4. Finish `ShoppingCartApp checkout()` and `InventoryDAO checkout()`.

Run the 2a JUnit test

1. Right click on the **TestCheckout.java** class and select **Test File**.
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 quanti 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 quanti 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 quantity 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 the 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]

*****

```

4. 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 quantity 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 quantity 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 quantity 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



Lesson 9: Java Client API for Administrative Features

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 one exercise with several steps.

Lab Procedure

We will identify some default values set for `MaxVersions`, `MinVersions`, `TimeToLive (TTL)`, etc. We will change some of Table properties. 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 and 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 include:

- `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, whether the table is a catalog table, `-ROOT-` or `.META.`, 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	

Lab 9.3: Working with LabAdminAPI in lab-exercises Project

Estimated time to complete: 60 minutes

If you have not already done so, use the following steps to open the project “lab-exercises” into NetBeans and build the project using Maven:

1. Open the **lab-exercises** project into NetBeans:
File --> Open project --> navigate to folder where projects are saved and choose **lab-exercises**
2. Build the project: Right click project lab-exercises and select **Clean and Build**

Review and complete the code for TODO 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);

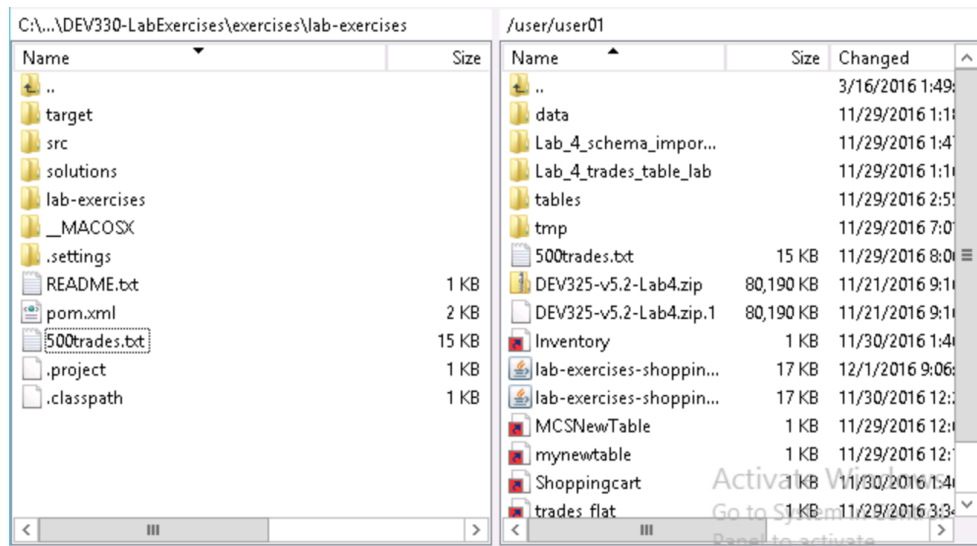
```

Build the Project, Copy the JAR, Run the Application

1. Build the project: Right click on the project name -> **Clean and Build**

Copy the jar file `lab-exercises-1.0.jar` from the target folder to the cluster using WinScp.
Also copy the data file `500trades.txt` from the project directory.





2. Login to the cluster and run the application with argument `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 08:26:22

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 must 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: Right-click on the project **lab-exercises** > **Clean and Build**

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
```

3. 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}
```

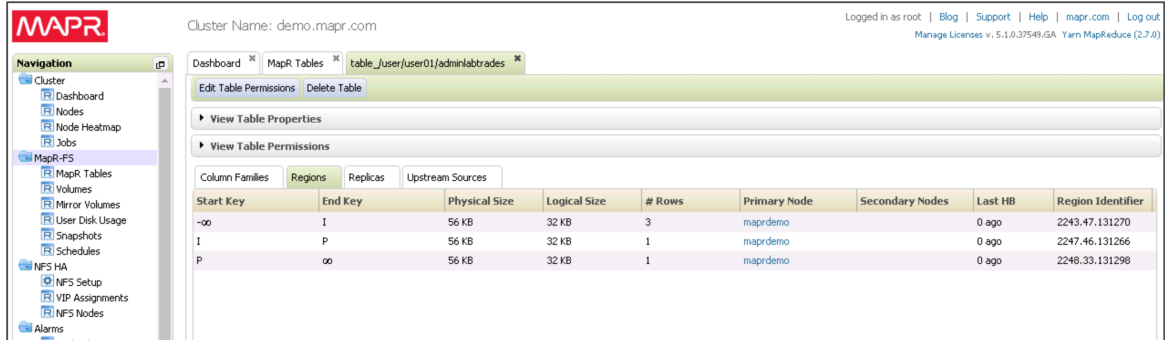
COLUMN	CELL
trades:price value=C\x98h\xF6	timestamp=1432658951600,
trades:price value=C\x97\xF4{	timestamp=1432658951599,
trades:price value=C\x98T{	timestamp=1432658951597,

3 row(s) in 0.0350 seconds

Pre-split the table

1. Pre-split the table at 'I' and '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 argument `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. Look at the `/user/user01/adminlabtrades` table.



Cluster Name: demo.mapr.com

Logged in as root | [Blog](#) | [Support](#) | [Help](#) | [mapr.com](#) | [Log out](#)

Manage Licenses v. 5.1.0.37549-GA Yarn MapReduce (2.7.0)

Navigation: Cluster, Dashboard, Nodes, Node Heatmap, Jobs, MapR-FS, MapR Tables, Volumes, Mirror Volumes, User Disk Usage, Snapshots, Schedules, NFS HA, NFS Setup, VIP Assignments, NFS Nodes, Alarms, Node Alarms

Dashboard | MapR Tables | table_user/user01/adminlabtrades

Edit Table Permissions | Delete Table

View Table Properties

View Table Permissions

Column Families | Regions | Replicas | Upstream Sources

Start Key	End Key	Physical Size	Logical Size	# Rows	Primary Node	Secondary Nodes	Last HB	Region Identifier
-∞	I	56 KB	32 KB	3	mapdemo		0 ago	2243.47.131270
I	P	56 KB	32 KB	1	mapdemo		0 ago	2247.46.131266
P	∞	56 KB	32 KB	1	mapdemo		0 ago	2248.33.131298

Optional Activity

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

1. In the **lab-exercises** project `adminApi.LabAdminAPI` class, complete the code following all comments marked `// TODO` 4. Build the project, then 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());  
}
```

2. Run the program with the argument `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  
...
```





DEV 335 – Developing Apache HBase Applications: Advanced

Part of the DEV 3200 curriculum

Lesson 10: 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.

Information on HBase Java API for Filters

The 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

Lab 10.1: Java Applications Using Filters

Estimated time to complete: 30 minutes

We will open the exercise project **lab-exercises-filter** into NetBeans, 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 two columns to store `Price` and `Volume` values. The composite row key is formed by combining the stock symbol and a reversed timestamp, `(Long.MAX_VALUE - timestamp)`. For example: `AMZN_98618600888`.

Because the row key 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: <code>SYM_TIME</code>	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 : scanner) {
    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. Open the exercise project **lab-exercises-filter** into NetBeans
2. Build the project using Maven: Select project **lab-exercises-filter** > **Clean and Build**.

Test 1

1. Navigate to **Test Packages > filter.TestFilterTall**.
2. Uncomment the first test `@Test`. The first test scans without a filter and prints the results.
3. Run the `TestFilterTall` test by right clicking on the class and selecting **Test File**.
4. 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 **Test Packages > filter.TestFilterTall**.
2. Uncomment the second test `@Test`.
3. Locate `TODO 2` in the test in `FilterTradesTall` and `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 click on the class. Select **Test File**. 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 **Test Packages > filter.TestFilterTall**.
7. Uncomment the test 2b @Test.
8. Run the `TestFilterTall` test. Right click the class, then select **Test File**.
9. Observe the output.

Test 3

1. Navigate to **Test Packages > filter.TestFilterTall**.
2. Uncomment the third @Test.
3. Locate `TODO 3` in the test, in `FilterTradesTall`.
4. Finish the code as instructed.
5. Run the `TestFilterTall` test: Right click on the class, select **Test File**.
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 **Test Packages > filter.TestFilterTall**.
2. Uncomment the fourth @Test `testFilterList()`.
3. Locate `TODO 4` in the test, in `FilterTradesTall`.
4. Finish the code as instructed.
5. Run the `TestFilterTall` test: Right click the **TestFilterTall** class, then select **Test File**.
6. Observe the output.

Run the Code on the Cluster

Now, copy the JAR to the cluster. Populate a tall table with data as follows.



1. Build the project by selecting the project folder 'lab-exercises-filter' and choosing **Clean and Build**.
2. Use WinSCP to upload the target jar, lab-exercises-filter-1.0.jar, to your user directory on the cluster.
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
```

4. 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
```

5. Run with argument `prefix` for a prefix filter.

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

6. Run with argument `row` for a row filter

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



Lab 10.2: Java Applications Using Increment

Estimated time to complete: 50 minutes

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 and 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 the `Inventory` table should be as shown below:

Table: Inventory

	cf: stock	
	Quantity	Adam
pens	13 8	5
notepads	23 19	4
erasers	10	

For pens, change 13 to 8 and insert 5 under the new `Adam` qualifier in one operation.

For notepads, change it to 19 and insert 4 under the `Adam` qualifier.

Open and Build the `lab-exercises-shopping3` Project

1. Open the `lab-exercises-shopping3` project into NetBeans. See the notes in the **Working With Lab Projects In NetBeans** document.
2. Build the project: Select **project lab-exercises-shopping3 > Clean and Build**.



Working with Shoppingcart Application checkout using Increment

Finish check out code

1. Look in the **Test Packages** directory at `shopping.TestCheckout`. Look for `TODO 1a`. 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 click on the class and select **Test File**. 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 quanti y 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 quanti y 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 quanti 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	9 8	1
notepads	21 19	3
erasers	10 7	2

Run on the Cluster to Initialize the Data for the Inventory and Shoppingcart tables

1. Build the JAR file for lab-exercises-shopping3. Click on the project and select **Clean and Build**.
2. Copy the JAR file lab-exercises-shopping3-1.0.jar from the target folder to the cluster using WinSCP.
3. 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
```

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 quanti y 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 quanti y 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 quanti y 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	9 8	1
notepads	21 19	3
erasers	10 7	2

Optional Labs

The following two can be performed during class if you complete previous labs ahead of time, or on your own time if you would like more practice.

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.

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 row key is a composite of the company symbol and the date, formatted `YYYYMMDD`. For example: `AMZN_20131020` would be Amazon stock traded on October 20, 2013.

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	@ts 2: 12.3 4						@ts 2: 100 0							
	@ts 1: 12.0 0						@ts 1: 200 0							
CSCO_20131023						@ts3 : 1.23					@ts 3: 300 0			
GOOG_20130817			@ts 6: 2.34						@ts 6: 100 0					

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 row key is formed by combining the stock symbol and a reversed timestamp, (`Long.MAX_VALUE - timestamp`). For example: `AMZN_98618600888`. Because the row key 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.



Lab 10.3a: Examine two different schema design options

Estimated time to complete: 20 minutes

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?

- a. Tall schema, CF1 column family: _____
- b. Flat schema, Price column family: _____
- c. Flat schema, Volume column family: _____
- d. 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 row key 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 row key 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 row keys 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:
 - a. Tall schema, CF1 column family: 1
 - b. Flat schema, Price column family: 3,600,000
(Each hour has 3,600,000 milliseconds, which is the max theoretical versions to store.)
 - c. Flat schema, Volume column family: 3,600,000
 - d. Flat schema, Stats column family: 1

Lab 10.3b: Finish code and Run Unit test

Estimated time to complete: 30 minutes

Perform the following tasks:

1. If you have not already done so, open the project `schemadesign` into NetBeans.
2. Build the project by right-clicking on the project and choosing **Clean and Build**.
3. Expand `Test Packages`. In `schemadesign.MyHBaseTestFlat` and `schemadesign.TradeDAOFlat`, finish the code following the `TODO` comments. Then, run the `MyHBaseTestFlat` JUnit test, by right-clicking on the class and selecting **Test File**. If it does not run green, fix the errors.

Populate a flat table with data, and read values back

1. Build the project by right-clicking the `schemadesign` project and choosing **Clean and Build**.
2. Use WinSCP to upload the target JAR, `schemadesign-1.0.jar`, to your user directory on your cluster.
3. There is a test data file, `500trades.txt`, at the top level of the project directory. If you have not already uploaded this file to your user directory with WinSCP, upload it now.
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 three 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 NetBeans, modify the `CreateTableUtils.createTable()` method to increase the `MaxVersions` for all column families. For now, it is okay 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 user ID.

```
disable '/user/user01/trades_flat'
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` comments.
5. Implement `formRowkey()` for `TradeDAOTall`. This method takes a (string) stock symbol and a (long) timestamp, and returns the row key. Use a reverse timestamp in the row key by



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

6. Implement `getTradesByDate()` for `TradeDAOTall`.
7. Run the `MyHBaseTestTall` JUnit test
8. Right click on the class and select **Test File**. If it does not run green, fix the errors.
9. Build the project by selecting the project folder and choosing **Clean and Build**.
10. Upload the target JAR, `schemadesign-1.0.jar`, to your user directory on the cluster.
11. 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.
12. 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?



Lesson 11: Working with MapReduce on HBase

Lab 11.3a: Developing MapReduce Applications for HBase (flat-wide)

Estimated time to complete: 50 minutes

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 three types of quotes used in this lab. Use the correct quotes in your shell.

Single quote '

Back quote `

Double quote "

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 `schemadesignsolution-1.0.jar` file to your home directory on the cluster using WinSCP. This file can be found in `DEV3200-LabSolutions` under `schemadesign-solution` directory.
2. Run the `CreateTable` program as follows.

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

3. Verify your table is populated.

```
$ 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	@ts 2: 12.3 4						@ts 2: 100 0							
	@ts 1: 12.0 0						@ts 1: 200 0							
CSCO_20131023						@ts3 : 1.23					@ts 3: 300 0			
GOOG_20130817			@ts 6: 2.34						@ts 6: 100 0					

The reduce job will populate the statistics column family with `count`, `max`, `min`, and `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. Open the `mapreduce-lab` project in NetBeans.
2. Modify the code TODO's.
 - a. Navigate to **Source Packages > mapredcelab.flatwide > StockMapper**.
 - b. Finish the code in `StockMapper` following the TODO 2.
3. Run the `StockMapperTest` JUnit test.
 - a. Navigate to **Test Packages > mapredcelab.flatwide > StockMapperTest**.
 - b. Uncomment `@Test` in `StockMapperTest`.
 - c. Right click on the class and select **Test File**.
 - d. If it does not run green, fix the errors.
4. Build the JAR by right clicking on the project and choosing **Clean and Build**.
5. Copy the JAR file to the cluster using WinSCP.
6. Login to the cluster and run the code.

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

7. Examine the output.

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

Run a MapReduce Program on an HBase Table

The format of the flat-wide reduce input key, prices:

```
GOOG_20131022    5005, 6155, 9996,
```

The format of the 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. Calculate the `min`, `max` and `mean` by modifying the TODO 3.
2. Navigate to **Source Packages > mapredcelab.flatwide > StockReducer**.
3. Finish the code in `StockReducer` following the TODO Exercise 3 comments.
4. Run the `StockReducerTest` JUnit test.
5. Navigate to **Test Packages > mapredcelab.flatwide > StockReducerTest**.
6. Uncomment the `@Test` in `StockReducerTest`.
7. Right click on the class and select **Test File**. If it does not run green, fix the errors.



8. Modify the code for `TODO 3` in `StockDriver.java`.
9. Navigate to **Source Packages > mapreducelab.flatwide > StockDriver.java**.
10. Build the JAR by right clicking on the **mapreduce-lab project** and choosing **Clean and Build**.
11. Copy the JAR file to the cluster using WinSCP.
12. Login to the cluster and run the code:

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

13. Examine the newly created table columns of `min` and `count`:

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

You should now see values in the statistics column family

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

14. Examine the output. Why does it not exist?

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

Lab 11.3b: Developing MapReduce Applications for HBase (tall-narrow)

Estimated time to complete: 50 minutes

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 MapReduce program on an HBase table.
4. Calculate other statistics.

Populate the `trades_tall` HBase table (if necessary)

1. If you did not perform the schema design lab in which the `~/trades_tall` table is created, then perform the tasks in this exercise. Copy the `schemadesignsolution-1.0.jar` to your home



directory on the cluster using WinSCP. This file can be found in **DEV3200-LabSolutions** under the **schemadesign-solution** directory.

2. Run the `CreateTable` program as follows:

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

3. Verify your table is populated.

```
$ 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. Go to the `mapreduce-lab` project in NetBeans.
2. Modify the code TODO's for exercise 2 and run the unit test
 - a. Navigate to Source Packages -> `mapreducelab.tallnarrow` -> `StockMapper`
 - b. Finish the code in `StockMapper` following the TODO 2's.



3. Run the StockMapperTest junit test
4. Navigate to **Test Packages > mapreduce-lab.tallnarrow > StockMapperTest**.
5. Uncomment the `@Test` line.
6. Right click on the class and select **Test File**. If it does not run green, fix the errors.
7. Build the JAR by right clicking on the **mapreduce-lab** project and choosing **Clean and Build**.
8. Copy the JAR file to the cluster using WinSCP.
9. Log in to the cluster and run the code:

```
$ java -cp `hbase classpath` :./mapreduce-lab-1.0.jar
mapreduce-lab.tallnarrow.StockDriver ~/OUT4
```

10. Examine the output:

```
$ cat ~/OUT4/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. Calculate the `min`, `max` and `mean` by modifying the TODO's for Exercise 3:
 - a. Navigate to **Source Packages > mapreduce-lab.tallnarrow > StockReducer**.
2. Finish the code in `StockReducer` following the TODO 3 comments.
 - a. Run the `StockReducerTest` JUnit test
 - b. Navigate to Test Packages -> mapreduce-lab.tallnarrow -> StockReducerTest
 - c. Uncomment '`@Test`' line
 - d. Right click on the class and select **Test File**. If it does not run green, fix the errors.
3. Modify the code TODO's for Exercise 3.
 - a. Uncomment the line following `TODO3` in **Source Packages > mapreduce-lab.tallnarrow > StockDriver.java**.
4. Build the JAR by right clicking on **mapreduce-lab** project and choosing **Clean and Build**.
5. Copy the JAR file to the cluster using WinSCP.



6. Login to the cluster and run the code:

```
$ java -cp `hbase classpath`:/mapreduce-lab-1.0.jar  
mapreducelab.tallnarrow.StockDriver ~/OUT5
```

7. Examine the newly created table columns of `min` and `count`:

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





DEV 340 – Bulk Loading, Security, and Performance

Part of the DEV 3200 curriculum

Lesson 12: Bulk Loading of 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://maprdocs.mapr.com/51/MapR-DB/BulkLoadingandMapR-DBTabl_29657142-d3e63.html

Lab 12.2: Use ImportTsv and CopyTable

Estimated time to complete: 30 minutes

Create a Table for Bulk loading

1. Create a table with the HBase Shell:

```
$ create '/user/user01/voter_data_table', 'cf1', 'cf2', 'cf3',  
BULKLOAD => 'true'
```

2. Observe the new table you have created in the UI and at the CLI.

- a. In MCS: highlight MapR tables> Go To Table /user/userxx/voter_data_table

3. Ensure you are in the directory '/user/user01'. Download and unzip the sample data folder.

```
$ wget http://course-files.mapr.com/DEV3200/DEV340-v5.2-Lab12.zip  
$ unzip DEV340-v5.2-Lab12.zip
```

4. Copy voter data to user01's directory:

```
$ cp 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`:

```
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:

```
# maprccli table edit -path /user/user01/voter_data_table -bulkload
false

hbase shell> alter '/user/user01/voter_data_table', BULKLOAD =>
'false'
```

6. While the import job is processing, look at the MCS to view changes to the table and puts being processed on the node.
 - a. Click **Nodes** under **Cluster** in the **Navigation panel** on the left hand side.
 - b. Click the **Overview** drop-down, change the value to **Performance**, then click the cluster name.
 - c. If necessary, scroll to the right so you can see the **Gets**, **Puts** and **Scans** columns.
 - d. 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.
 - e. Click **MapR Tables** under **MapR-FS**.
 - f. Click the name of the table you used for the import under **Recently opened tables**.
 - g. In an HBase shell, examine the data that has been imported.

```
$ hbase shell

hbase(main):> scan '/user/user01/voter_data_table', LIMIT => 5
```

See the sample output presented on the following page.



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



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', 'cf1', 'cf2', 'cf3',
BULKLOAD => 'true'
```

In this table we will import a larger file of similar data and create a new `HBASE_ROW_KEY` from field position 2 – assuming that the user names are all unique in the database.

2. Unzip and copy the larger voter data to your home directory:

```
$ gunzip data/voter10M.gz
$ cp data/voter10M /user/user01/
```

3. Import data to the new table using `/user/user01/voter10M`.

```
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 the HBase shell, take the table out of `bulkload` mode and examine the data that has been imported:

```
$ hbase shell
> alter '/user/user01/voter_data_table_10m', 'cf1', 'cf2', '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. Use the HBase shell to create another table called `/user/user01/voter_data_table_cp`:

```
> create '/user/user01/voter_data_table_cp', 'cf1', 'cf2', 'cf3',
BULKLOAD => 'true'
```

2. At the system shell type in the following:

```
$ hbase com.mapr.fs.hbase.tools.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
```

Lab 12.3: Use a Custom MapReduce Program to Bulk Load Data

Estimated time to complete: 30 minutes

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 in the folder downloaded earlier in this lab. 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` using the HBase shell command:

```
> create '/user/user01/hly_temp', 'readings', BULKLOAD => 'true'
```
2. In the system shell, create an input directory in your home directory

```
$ mkdir /user/user01/input
```
3. Copy the data file `hly-temp-10pctl.txt` to `/user/user01/input`

```
$ cp data/hly-temp-10pctl.txt 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. Note:
 - 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:

```
# maprccli table edit -path /user/user01/hly_temp -bulkload false  
hbase shell> alter '/user/user01/hly_temp', 'readings', BULKLOAD =>  
'false'
```



Lesson 13: Performance

Lab Overview

This lab will show how to use YCSB to learn about benchmarking a cluster.

The performance characteristics of a MapR-DB cluster 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.

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. (eg: 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>

Lab 13: Install and use YCSB

Estimated time to complete: 30 minutes

Install YCSB and create a table

1. Download and unpack the YCSB benchmarking tool, and position yourself in the working directory:

```
$ wget http://course-files.mapr.com/DEV3200/DEV340-v5.2-Lab13.tar.gz
$ tar zxvf DEV340-v5.2-Lab13.tar.gz
$ cd ycsb-0.1.4/
```

The YCSB tests in the following sections are intended to be run in the `ycsb-0.1.4` directory.



2. Create a table using MCS called '/user/user01/usertable' with the following schema:

- a. Column family = family

```
hbase shell
> create '/user/user01/usertable', 'family'
> list '/user/user01'
> exit
```

2. Observe the new table you have created in the UI or at the CLI (your choice). If you use the MCS, navigate to **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 two YCSB processes, each with three 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 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.



Option	Meaning	Value	Notes
-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/user01/usertable	
-load	Perform the "load" portion of the test		This value should remain unchanged.
-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
```

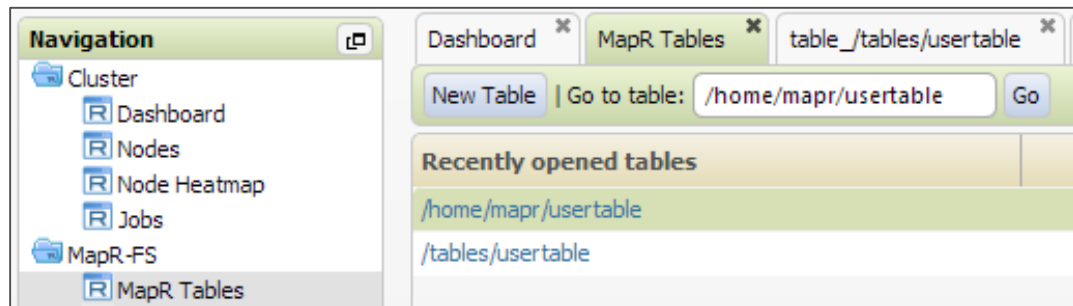
Record the results of this activity below:

Test	Result
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 or at the CLI (your choice).
 - a. In MCS: highlight **MapR tables** > **Go To Table** > **/user/user01/usertable**
 - b. Click on the primary **Node**.
 - c. Look at the DB puts.
3. Click on **MapR Tables**, enter the table name, click **Go**.



4. Click the **Regions** tab.

Node Heatmap	Column Families	Regions
Jobs	Start Key	End Key
MapR-FS	-∞	∞
MapR Tables	Physical Size	Logical Size
Volumes	432KB	384KB
	# Rows	298

5. Click the primary **Node**.

Node Heatmap	Column Families	Regions
Jobs	Logical Size	# Rows
MapR-FS	744KB	600
MapR Tables	Primary Node	maprdemo.local
Volumes		

6. Look at the DB Puts, Memory, disk for the primary node.

Cluster

Dashboard

Nodes

Node Heatmap

Jobs

MapR-FS

MapR Tables

Volumes

Mirror Volumes

User Disk Usage

Snapshots

Schedules

NFS HA

NFS Setup

VIP Assignments

Forget Node

Change Topology

Memory Used (**71% of 5.7GB**

Disk Used (GB) **0% of 14GB in use for 2 disk(s)**

CPU:

CPU

2

% CPU used

11

Network I/O:

In (per sec)

4.4KB

Out (per sec)

4.4KB

RPC I/O:

Count (per sec)

-

In (per sec)

184B

Out (per sec)

394B

Reads (per

Writes (per

TaskTracker Slots

	Used	Total
Map Slots	-	2
Reduce Slots	-	1

DB Gets, Puts, Scans

	Gets	Puts	Scans
10s	-	-	-
1m	-	-	-
5m	-	-	-
15m	-	198	-



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:

Test	Result
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 $ycsb_threads -db com.yahoo.ycsb.db.HBaseClient \
    -P workloads/workloada -p columnfamily=family -p \
    table=/user/user01/usertable -t -s \
    -p insertstart=${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/workloada	The range scan test is included in workloada. This value should remain unchanged.



Option	Meaning	Value	Notes
-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.
-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.

Record the results of this activity below:

Test	Result
YCSB scan test throughput	
YCSB scan test 99th percentile latency	

Take a look at the YCSB HBaseClient code

1. Unzip the ycsb-master.zip file. Import the maven project.
2. 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){
        ..
    }
}
```



Lesson 14: Security

Lab 14.2: Create MapR-DB tables and set permissions

Estimated time to complete: 45 minutes

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	v12	v13	v14

1. Create a table in your home directory `/user/user01/`:

```
hbase> create '/user/user01/t_user01', {NAME=>'cf1'}, {NAME=>'cf2'}
```

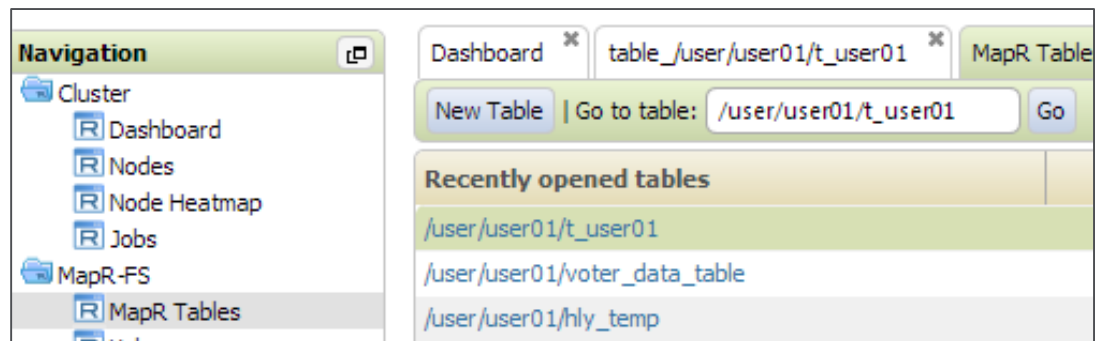
2. List the tables in your home directory:

```
hbase> list '/user/user01/'
```

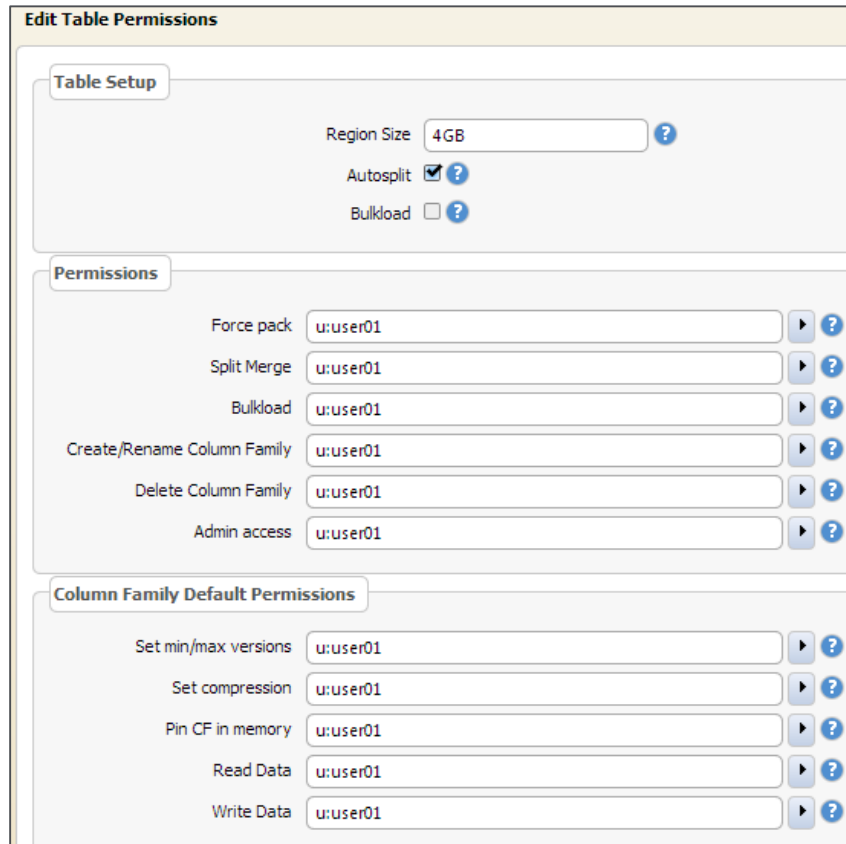
3. Execute the following put statements to insert the records into `t_user01` using the HBase shell:

```
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'
```

4. Go to the table in MCS:



5. Examine the table permissions:



6. Use `scan` with `user01`:

```
hbase> scan '/user/user01/t_user01'
```

Shows one row, four 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` (the password is `mapr`), and try to scan the table.

```
$ su user02
$ echo "scan '/user/user01/t_user01'" | hbase shell
```



Q: The output shows zero rows. Why?

A: The user `user02` does not have the required permissions.



8. Switch back to `user01`. Give read permissions to `user02` on column family `cf1`, as well as execute permission on the directories.

```
$ maprcli table cf edit -path /user/user01/t_user01 -cfname cf1
-readperm "u:user01 | u:user02"
$ chmod +x /mapr/<cluster>/user/user01
$ chmod +x /mapr/<cluster>/user/user01/t_user01
```

9. Switch back to `user02` and try the scan again.

```
$ echo "scan '/user/user01/t_user01'" | hbase shell
```

Shows one 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`:

Column Family Permissions	
Set min/max versions	u:user01
Set compression	u:user01
Pin CF in memory	u:user01
Read Data	u:user01 u:user02
Write Data	u:user01
Append Data	u:user01

10. As `user02`, try to scan `cf2`.

```
$ echo "scan '/user/user01/t_user01', {COLUMNS => ['cf2']}" | hbase
shell
```

Zero rows are returned.

11. Log back in as `user01`, and give write permissions to `user02` on column family `cf1`.

```
$ maprcli table cf edit -path /user/user01/t_user01 -cfname cv1
-writeperm "u:user01 | u:user02"
```

12. Log in as `user02` and insert a new row. Read the new row as `user01`.

```
$ 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 two rows, three values.



13. Column Level Permissions: Allow only user02 read permissions on c1 in cf1.

```
$ maprccli table cf edit -path /user/user01/t_user01 -cfname cf1  
-readperm "u:root | u:user02"  
  
$ maprccli 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
```

Results should show two rows, three values.

Access is given by “ANDing” the permissions on the Column Family → Column.

