MAPR Academy

DA 440 Apache Hive Essentials Lab Guide

Fall 2015

Using This Guide

Icons Used in the Guide

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

Note: Additional information that will clarify something, provide details, or help 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.

Lab Files

Here is a description of the materials (data files, licenses etc.) that are supplied to support the labs. These materials should be downloaded to your system before beginning the labs.

DA440_LabFiles.zip	Scripts, data, and other files needed to complete this lab
DA440 LAB GUIDE.pdf	Exercises, answer keys, and information about this lab
CONNECT TO MAPR SANDBOX.pdf	Description of how to connect to your Sandbox or AWS cluster and copy necessary files to your node or cluster

Get Started

Lab Overview

This short exercise introduces you to the Hive command line interfaces and helps you set up the environment for the rest of the day's labs. We'll begin by connecting to the MapR Cluster (MapR Sandbox or AWS) and transferring the DA440_LabFiles files to the Cluster. These instructions assume you have basic familiarity with working in a command line interface such as a Unix shell.

Part 1: Prepare the MapR Cluster (Sandbox or AWS) for Lab

Part 2: Load the Training Files

Lab Procedure

Part 1: Connect to the MapR Cluster (Sandbox or AWS) for Lab

Refer to the CONNECT TO MAPR SANDBOX.pdf file for instructions on how to connect to the MapR Sandbox or AWS Cluster for this lab.

Part 2: Load the training files

The training materials include a file called DA440_LabFiles.zip, which contains data, source code, reference materials, and utilities that you will use to complete the lab exercises. Refer to the CONNECT TO MAPR SANDBOX.pdf file for instructions on how to copy DA440_LabFiles.zip from your computer to the cluster you connected to in Part 1.

Verify the files were copied to your MapR Sandbox or AWS Cluster correctly using `ls`, then unpack them using `unzip`.

```
[user01@maprdemo ~]$ ls
[user01@maprdemo ~]$ unzip DA440_LabFiles.zip
```

Here you will not need to replace the username or IP address, as these should appear at the prompt in the terminal if you are connected to the Sandbox or AWS cluster properly. Change the working directory to the DA440_LabFiles folder using cd DA440_LabFiles, and then list the files using `ls`.

```
[user01@maprdemo ~]$ cd DA440_LabFiles
[user01@maprdemo ~]$ ls
```

Lesson 1: Hive in the Hadoop Ecosystem

Lab Overview

This short exercise introduces you to the Hive command line interfaces and helps you set up the environment for the rest of the day's labs. You will also briefly explore loading and typecasting data. These instructions assume you have basic familiarity with working in a command line interface such a Unix shell, and some basic familiarity with SQL.

Lab 1.1 Procedure

1. Open a shell. Log in to your MapR Sandbox or AWS Cluster using the procedure outlined above.

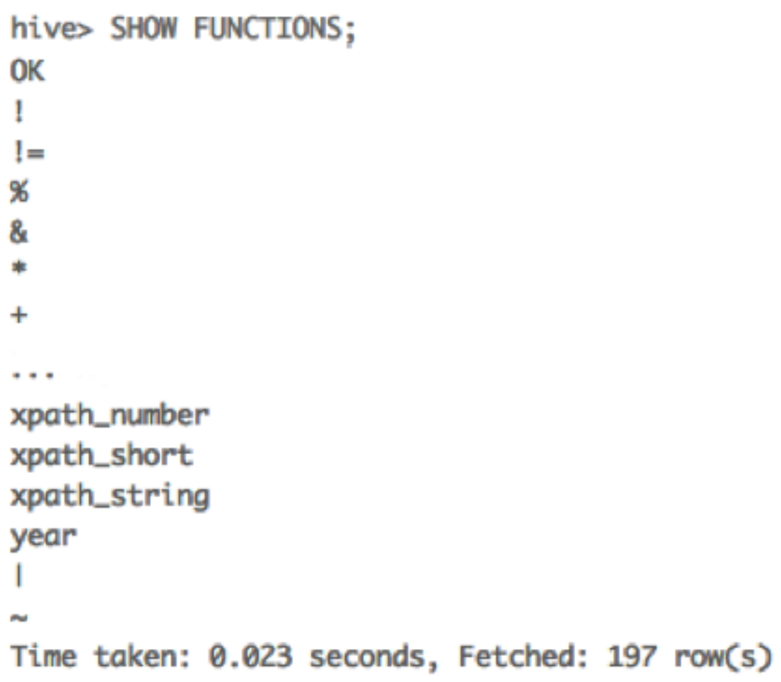
2. Connect using the Hive shell, and run some basic commands. Run the Hive shell program by typing `hive`. This may take a few moments.

```
[user01@maprdemo ~]$ hive
hive>
```

Note: You are now in the Hive Command Line Interface. For the remainder of this lab, you may assume that **>** is prompting a Hive command in either the Hive Command Line Interface or Beeline, while **$** is prompting a bash command in your Sandbox or AWS cluster.

You may find it useful to type commands in a text editor, such as NotePad++, rather than typing commands directly into the command line. Copy and pasting commands from a text editor will allow you to edit and save them, as well as allowing you to write longer queries.

3. Use the `SHOW FUNCTIONS` command to list available Hive Query Language functions.



```
hive> SHOW FUNCTIONS;
OK
!
!=
%
&
*
+
...
xpath_number
xpath_short
xpath_string
year
|
~
Time taken: 0.023 seconds, Fetched: 197 row(s)
```

4. Type `QUIT` to exit. Don't forget the semicolon!

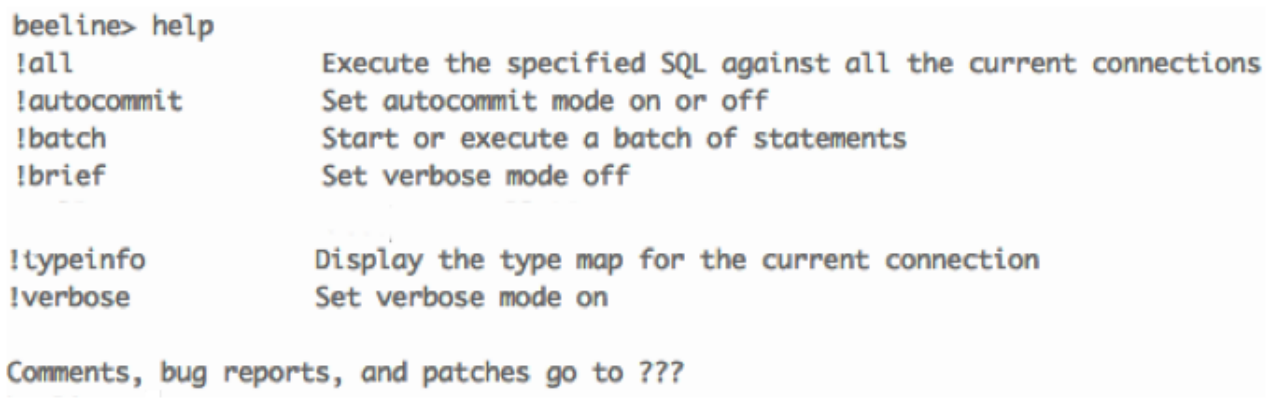
```
hive> QUIT;
[user01@maprdemo ~]$
```

You are now back in the bash shell associated with your MapR environment.

5. Next, try connecting to Hive using the Beeline client. Connect to the HiveServer2 by typing `hive --service beeline` in a bash shell.

```
[user01@maprdemo ~]$ hive --service beeline
Beeline version 1.0.0-mapr by Apache Hive
beeline>
```

6. Type `help` to see a list of beeline shell commands. You do not need a semicolon for Beeline commands, but you will for Hive Query Language commands.



```
beeline> help
!all                    Execute the specified SQL against all the current connections
!autocommit             Set autocommit mode on or off
!batch                  Start or execute a batch of statements
!brief                  Set verbose mode off

!typeinfo               Display the type map for the current connection
!verbose                Set verbose mode on

Comments, bug reports, and patches go to ???
```

`!quit` exits the beeline shell.

```
beeline> !quit
[user01@maprdemo ~]$
```

Note: The Hive Command Line Interface (CLI) uses the Hive Driver, while Beeline uses HiveServer2. Many of the same commands can be used in both the Hive CLI and Beeline, but you may encounter some version compatibility issues using one or the other. Consult http://doc.mapr.com/display/MapR/Ecosystem+Support+Matrix or your Hadoop System Administrator for more information to help you decide whether you should use Hive CLI or Beeline for your projects. In general, the Beeline shell is recommended when available.

Lab 1.2 Procedure

8. For this portion of the lab, we will work with a modified subset of some data from eBay auctions, which includes the opening bid price, the final bid price, the item type, and the number of days the auction was open.

   We will discuss creating tables and loading data in greater detail in Lesson 2. For now, try typing the following SQL data definition in the Hive shell:

   ```
   CREATE DATABASE ebay;
   CREATE TABLE ebay.auction (openingBid MONEY,
      finalBid MONEY, itemType STRING, days NUMERIC);
   ```

   What happens? Based on what you learned about data types in Lesson 1, how would you rewrite this data definition using HiveQL? If you get stuck, refer to the Answer Key at the end of this document.

Try this! If you were able to successfully create the **auction** table, try to load the eBay auction data with the following command:

```
LOAD DATA LOCAL INPATH
'/user/user01/DA440_LabFiles/auctiondata.csv'
INTO TABLE ebay.auction;
```

Try querying this data with SQL commands you know. Do they work as you expect them to?

Lesson 2: Create and Load Tables

Lab Overview

The purpose of this lab is to explore the Hive Data Definition Language. You will create databases and tables of various types and load them with data. You will explore creating and manipulating database objects using the Hive shell in interactive and non-interactive ways. This lab will use real weather data from a select number of research stations throughout the continent of Antarctica. The original files can be downloaded at https://legacy.bas.ac.uk/met/READER/data.html.

Lab Task Summary

- Create a database
- Create a simple table
- Create an external table
- Create a partitioned table
- Load data using `LOAD DATA`
- Examine database objects

Lab 2.1 Procedure

1. **Log in to your Sandbox or AWS Cluster**

   Use the procedures from Part 1 of Lesson 1 to connect to your Sandbox or AWS Cluster.

2. **Connect to Hive using the Hive shell**

   Use the procedures from Part 3 of Lesson 1 to connect to Hive using the Hive Command Line Interface or the Beeline shell. Make sure you see `hive>` or `beeline>` before your prompt.

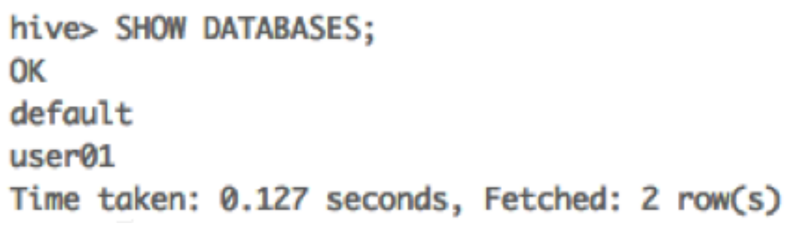
3. **Create a database**

   Use Hive data definition language (DDL) commands to create a database where you will store all of the database objects you'll create throughout the remainder of the lab exercises.

   You will create the database in your home directory, and name it the same as your user id.

   ```
   > CREATE DATABASE user01 LOCATION '/user/user01/hive/user01.db';
   ```

   You can use the command SHOW DATABASES to list all of the databases available in this Hive instance. Don't forget the semicolon after all hive commands!



```
hive> SHOW DATABASES;
OK
default
user01
Time taken: 0.127 seconds, Fetched: 2 row(s)
```

You should see your new user01 database now.

Note: Hive Query Language (HQL) commands are shown in upper case. This is a convention, not a requirement. HQL commands are case-insensitive, and may be written in either upper or lowercase. All HQL statements must end with a semicolon.

Try this! You can also see the databases from your bash shell. If you are already in Hive, use `QUIT;` to exit, then try this command:

```
    $ hadoop fs -ls /user/user01/hive
```

This should also list the **`default`** and **`user01`** databases. To get back into Hive, use **`beeline`** or **`hive`**.

Lab 2.2 Procedure

4. **Create a simple table**

   For our first table, we'll create the **`location`** table. The table should have the following characteristics.

   - A station column, of type string
   - A latitude column, of type integer
   - A longitude column, of type integer
   - A row format of delimited
   - Fields terminated by comma
   - Lines terminated by the line feed character
   - Stored as a text file

   If you are familiar with SQL, try creating this table on your own. If you need a hint, refer to the Answer Key at the end of this document. We'll load this table with data in a later step.

Caution! Be sure to create this table inside the database you created in the previous step. There are two ways to do this. One is the **USE** statement, and the other is dot notation.

Dot notation uses the database name followed by the table name, as below:

```
> CREATE TABLE user01.location (station STRING, latitude
INT, longitude INT);
```

The **USE** statement is called before creating the table.

```
> USE user01;
> CREATE TABLE location (station STRING, latitude INT,
longitude INT);
```

Make sure to use the database associated with your username.

Q: How do I know if my tables have been created properly? What should I do if I accidentally create a table I do not need? How can I rename a column?

A: You can use `SHOW TABLES IN user01;` to list all the tables in your database. You can use `DESCRIBE user01.location;` to verify that this table has all the correct characteristics.

If you have accidentally created a table, or created a table incorrectly, you can use `DROP TABLE user01.location;` to delete this table.

If you misnamed a column, such as misspelling "longitude" as "long", you could use `ALTER TABLE user01.location CHANGE COLUMN long longitude INT;`.

Lab 2.3 Procedure

5. **Create a partitioned table**

Creating partitioned tables is important in Hive, since partitioning the data can speed up queries and optimize results. For this step, we'll create the **windspeed** table. The **windspeed** table should have the following characteristics:

- A year column, of type integer
- An month column, of type string
- A knots column, of type float

- A partition using station as the column, of type string
- Delimited row format
- Fields terminated by comma
- Lines terminated by linefeed
- Stored as a text file

As before, try to create this table on your own. You may want to write your query in a text editor first before trying it in the Hive CLI. If you need a hint, refer to the Answer Key at the end of this document.

6. **Create an external table**

External tables are read-only tables. This is useful for data you do not want to accidentally delete or change. External tables are typically linked to an external file, which contains the data. When creating an external table, you will describe the rows and metadata using Hive's DDL.

For this step, we'll create an external table that uses a text file stored on the distributed file system as its data store. We'll use one of the data files in your DA440_LabFiles folder. This will be the source of the data for the countries table, which we'll create in a later exercise.

Enter the HQL commands to create an external table in your database. The table should have the following characteristics:

- A station column, of type string
- A year column, of type integer
- A month column, of type string
- A celsius column, of type float
- Delimited row format
- Fields terminated by comma
- Lines terminated by linefeed
- Stored as a text file
- A location pointing to the **`temperature`** folder in your DA440_LabFiles folder. This uses the **`LOCATION`** clause, which follows the **`STORED AS`** clause.

Try to create the data description (DD) for this external table on your own in a text editor. If you have difficulty, refer to the Answer Key at the end of this document. Make sure that the file path in your **`LOCATION`** clause matches your own. Use bash commands like **`ls`** and **`pwd`** to verify the correct file path. Once the DD is created, load it by using Hive. You can either do this by copy and pasting the DD into the Hive CLI or by using Hive in non-interactive mode.

Since this is an external table linked to a CSV file, there should be some data in it! Try this:

```
> SELECT * FROM user01.temperature LIMIT 10;
```

If you are familiar with SQL, go ahead and explore this table if you'd like.

Lab 2.4 Procedure

7. **Load data into the simple table**

We'll use **LOAD DATA** to load the **location** table. Remember to replace **user01** with your own in both the file path and the database notation.

```
> LOAD DATA LOCAL INPATH
'/user/user01/DA440_LabFiles/location.csv'
INTO TABLE user01.location;
```

8. **Load data into the partitioned table**

We'll use **LOAD DATA** again to load the **windspeed** table. Since this table is partitioned, we'll have to add the **PARTITION** specification to our **LOAD DATA** commands.

```
> LOAD DATA LOCAL INPATH
'/user/user01/DA440_LabFiles/wind_Adelaide.csv'
INTO TABLE user01.windspeed PARTITION(station = 'Adelaide');
```

Make sure to substitute the file path, database, and username associated with your cluster!

Verify that the data loaded successfully by exploring the table a little. Once you have one partition loaded, load the other partitions – Clean Air, Faraday, Grytviken, Halley, Neumayer, Rothera, and Signy – and verify that each of them has loaded properly.

Try this! You can also explore the warehouse directory using Hadoop fs commands to see how the partitioned table is laid out. Exit Hive using **QUIT;**, then enter:

```
$ hadoop fs -ls /user/user01/hive/user01.db/windspeed
```

You can get back into Hive by using **beeline** or **hive** from the bash command line.

Lab 2.5 Procedure

9. **Examine database objects**

Use Hive commands to examine the objects you've created. Some commands to try:

```
> SHOW DATABASES;
> USE user01;
> SHOW TABLES;
> DESCRIBE location;
> DESCRIBE EXTENDED windspeed;
```

Take a moment to verify that you have successfully created all the tables above. Using **`SHOW TABLES IN user01;`** you should see a list of three tables. Try **`DESCRIBE`** on each of these tables to make sure they all have the correct row names and data types. If not, use **`DROP TABLE`** to delete the table, then recreate it properly using **`CREATE`**.

10. **Examine loaded tables**

Right now, our **`location`**, **`windspeed`** and **`temperature`** tables should have some data in them! If you are familiar with SQL, try running some basic queries on these tables. Since you probably don't want to print all the data, you can use **`LIMIT`**. Here are some queries to try:

```
> USE user01;
> SELECT * FROM location;
> SELECT count(*) FROM windspeed;
> SELECT * FROM windspeed LIMIT 20;
> SELECT * FROM temperature WHERE year = 2000;
```

Lesson 3: Data Manipulation

Lab Overview

The purpose of this lab is to gain experience with data manipulation in Hive. First we will explore interactive querying using HiveQL in the Hive Shell. We will look at the use of User-Defined functions in Hive queries. Finally, we will combine what we have learned to aggregate our data and create more advanced queries.

- Simple Queries
- User-Defined Functions
- **`JOIN`** and **`UNION`** tables
- Use **`CREATE TABLE AS SELECT`**

Lab 3.1 Procedure

1. **Log in to Cluster**

   Log in to your Sandbox or AWS Cluster using the procedure from Lesson 1.

2. **Simple Queries**

   Let's look at the data we've loaded. Log in to the Hive CLI or Beeline, whichever you prefer.

   Note: It is important to remember that certain Hive queries translate your SQL-like code into map-reduce code. This allows people who are familiar with SQL to easily write map-reduce jobs and query huge amounts of data, without needing to learn Java or other low-level languages. However, for those who are familiar with SQL and traditional relational databases, Hive can seem slow. You may have to wait several minutes for the map-reduce process to finish before seeing your results! Be patient as you try this next part of the lab.

   First, let's look at the `temperature` table. This table holds the average monthly temperatures, in degrees Celsius, from eight different weather stations in Antarctica from several decades.

   ```
   > USE user01;
   > SELECT * FROM temperature LIMIT 20;
   ```

   As you can see, there is the station name, the year, the month, and the temperature. Let's look at all the temperatures from January, 1970, which is the time when Unix time began:

   ```
   > SELECT * FROM temperature WHERE month = 'jan' AND year = 1970;
   ```

It looks like the temperatures aren't so bad; in fact some of the numbers are positive! But remember, it is summer in January in Antarctica. Let's try July:

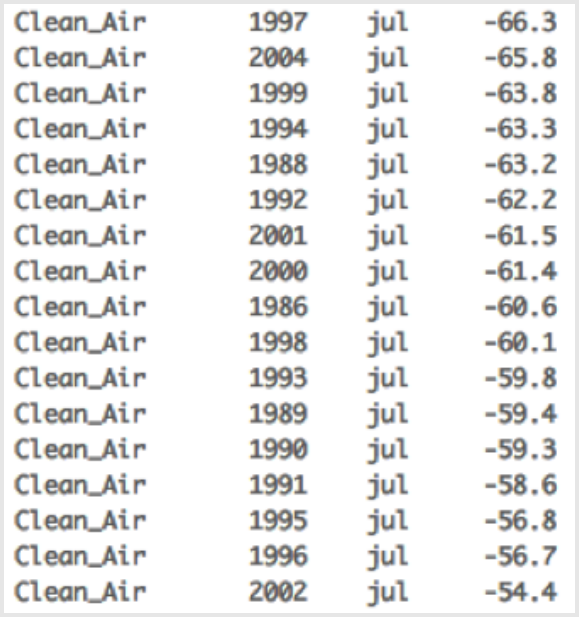
```
> SELECT * FROM temperature WHERE month = 'jul' AND year = 1970;
```

July is a much colder month than January! The name of the weather station at the South Pole is called Clean Air, because very little man-made pollution can be found there. Let's find out what temperatures were recorded in July at the South Pole:

```
> SELECT * FROM temperature
WHERE station = 'Clean_Air'
AND month = 'jul';
```

Q: Is this the result you expected? How can we make this query more useful or interesting?

A: The temperatures need to be in order, and we need to get rid of the **`NULL`** values. Try to write a query on your own that lists the temperatures in order, from coldest to warmest, and which omits all the **`NULL`** values. If you need a hint, refer to the Answer Key at the end of this document. Your final results should look something like this:



```
Clean_Air       1997    jul     -66.3
Clean_Air       2004    jul     -65.8
Clean_Air       1999    jul     -63.8
Clean_Air       1994    jul     -63.3
Clean_Air       1988    jul     -63.2
Clean_Air       1992    jul     -62.2
Clean_Air       2001    jul     -61.5
Clean_Air       2000    jul     -61.4
Clean_Air       1986    jul     -60.6
Clean_Air       1998    jul     -60.1
Clean_Air       1993    jul     -59.8
Clean_Air       1989    jul     -59.4
Clean_Air       1990    jul     -59.3
Clean_Air       1991    jul     -58.6
Clean_Air       1995    jul     -56.8
Clean_Air       1996    jul     -56.7
Clean_Air       2002    jul     -54.4
```

Try some other queries, such as finding out what the temperature was during your birth month or finding all the temperatures at your favorite weather station since the year 2000. Once you're satisfied with exploring this dataset and the **`windspeed`** table, move on to the next step.

Lab 3.2 Procedure

3. **User-Defined Functions**

As we talked about in the lecture, much of Hive's functionality is implemented in User-Defined Functions (UDF). The **`count()`** function is a User-Defined Aggregate Function (UDAF). Statistical analysis of data in tables is easily done with UDFs and UDAFs, such as **`min()`**, **`max()`**, and **`avg()`**.

Let's review the use of **`count()`**. The following query will count how many rows there are in the **`temperature`** table which do not have **`NULL`** values for the **`celsius`** column:

```
> SELECT count(*) FROM temperature WHERE celsius IS NOT NULL;
```

Let's find out what the average temperature in Antarctica was in 1970:

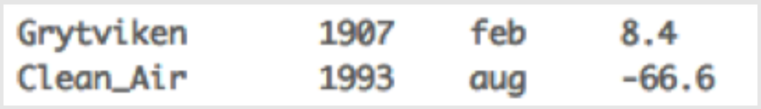
```
> SELECT avg(celsius) FROM temperature WHERE year=1970;
```

Now let's find out what the hottest and coldest temperatures were recorded in Antarctica:

```
> SELECT min(celsius), max(celsius) FROM temperature;
```

Q: Is this the result you expected? How can we make this query more useful or interesting?

A: While it is interesting that the hottest average monthly temperature in this dataset was 8.4C, and the coldest was -66.6C, it would be more interesting to find out where and when this happened. Use the OR operator to write a single query that returns the location and date of these two extremes. Your results should look something like this:



```
Grytviken      1907    feb     8.4
Clean_Air      1993    aug     -66.6
```

Hint: Negative numbers must be in quotes! Try to write this query on your own. Refer to the Answer Key at the end of this document if you need more help.

Note: You may have tried to use a query like this:

```
> SELECT * FROM temperature
WHERE celsius=min(celsius)
OR celsius=max(celsius);
```

If you did, depending on the version of Hive you are using, you may have encountered an error. This is because most versions of Hive do not allow UDFs to appear in WHERE clauses.

4. **Using arithmetic operators**

So far, all the data we have examined has been in degrees Celsius in the **temperature** table. In the **windspeed** table, we can see data in knots. For example, let's look at the wind speed data from January 1970:

```
> SELECT * FROM windspeed WHERE month = 'jan' AND year = 1970;
```

However, most people outside the meteorology and navigation industries aren't familiar with knots. Let's convert it to a more familiar unit of speed, such as kilometers per hour (KPH). To convert knots to KPH, multiply by 1.852. Since we specify the month and year, let's only select the research station, the wind speed in knots, and our converted KPH wind speed:

```
> SELECT station, knots, knots*1.852 AS kph
FROM windspeed WHERE month = 'jan' AND year = 1970;
```

Q: Is this the result you expected? How can we make this query more useful or interesting?

A: While accuracy is important, most weather data is only stored to a few significant figures. Having long decimals is difficult to read. Let's round to the hundredths place, and also get rid of any null values:

```
SELECT station, knots, round(knots*1.852, 2) AS kph
FROM windspeed WHERE month = 'jan' AND year = 1970
AND knots IS NOT NULL;
```

Now, try to write a query that gives us the station, temperature in degrees Celsius, and temperature in degrees Fahrenheit for January 1970. Remember, the formula for degrees Fahrenheit is C * 1.8 + 32. Try to round your answer, and get rid of any null values as well. Refer to the Answer Key at the end of this document if you need help.

Lab 3.3 Procedure

5. **Joining tables**

What if I want to know how wind speed relates to latitude or longitude? We know the coordinate information is in the **location** table, so we can **JOIN** this with the **windspeed** table. For the join, we know the station names will be the same, so let's **JOIN ON station**. We'll also need to give **windspeed** and **location** some aliases: **w** and **l**. Let's further limit our query to data more recent than 2012, wind speeds greater than 15 knots, and omit nulls:

```
> SELECT w.station, w.year, w.month, w.knots,
l.latitude, l.longitude FROM windspeed w
JOIN location l ON w.station = l.station
WHERE w.year > 2012 AND w.knots > 15 AND w.knots IS NOT NULL;
```

It seems that most of the fastest recent winds occur between 67 and 75 degrees South latitude! Now try writing another `JOIN` which adds the data from the temperature table as well. You can limit it by whatever parameters you like, such as omitting nulls, specifying a year, or specifying a certain temperature, wind speed, or latitude. Make sure the month and years are joined correctly in your data as well! If you need a hint, refer to the Answer Key at the end of this document.

6. Creating tables from queries

Let's combine everything we've learned so far into another join. This time, the table will include temperature in Celsius and Fahrenheit, wind speed in knots and KPH, latitude, and longitude. We'll just look at the first 10 rows of this table to begin with:

```
> SELECT t.station, t.year, t.month, t.celsius,
round(t.celsius*1.8+32, 2) as fahrenheit, w.knots,
round(w.knots*1.852,2) as kph, l.latitude, l.longitude
FROM windspeed w JOIN location l ON w.station = l.station
JOIN temperature t ON w.station = t.station
AND w.month = t.month AND w.year = t.year LIMIT 10;
```

This is a very interesting table! It contains all the data from all our files, as well as some data we created with some arithmetic and UDFs! Let's save it as a new table, called `weather`. This time, we don't want to limit the data, since we want our weather table to contain everything.

```
> CREATE TABLE weather AS SELECT t.station, t.year, t.month,
t.celsius, round(t.celsius*1.8+32, 2) as fahrenheit, w.knots,
round(w.knots*1.852,2) as kph, l.latitude, l.longitude
FROM windspeed w JOIN location l ON w.station = l.station
JOIN temperature t ON w.station = t.station
AND w.month = t.month AND w.year = t.year;
```

Now we can query the weather table without having to `SELECT` and `JOIN` all this data again.

```
> SELECT * FROM weather WHERE latitude > 70 AND fahrenheit > 35;
```

Finally, let's export this new table. Exit the Hive CLI with `QUIT;` and enter this command:

```
$ hive -e 'SELECT * FROM user01.weather' >
/user/user01/weather.csv
```

Now you can use `vi` to see your new `weather.csv` file. If you'd like, you can also use `scp` to copy this file from your Sandbox or AWS cluster to your local machine.

Try this! With the remainder of the lab time, try querying the `weather` table. Be creative, and explore the data any way you like! Some ideas about to get you started:

1. What is the average wind speed at 90 degrees latitude (the South Pole)?
2. When were the coldest and warmest temperatures recorded for each station?
3. How would you create another new table, `weather2`, which has no null values?

Appendix: Answer Key

Lesson 1, Step 8: Cast data in Hive

```
CREATE TABLE default.auction (
      openingBid FLOAT,
      finalBid FLOAT,
      itemType STRING,
      days INT);
```

Lesson 2, Step 4: Create a simple table

```
CREATE TABLE user01.location (
      station STRING COMMENT 'Station Name',
      latitude INT COMMENT 'Degrees South',
      longitude INT COMMENT 'Degrees West')
      ROW FORMAT DELIMITED
      FIELDS TERMINATED BY ','
      LINES TERMINATED BY '\n'
      STORED AS TEXTFILE;
```

Lesson 2, Step 5: Create a partitioned table

```
CREATE TABLE user01.windspeed (
      year INT COMMENT 'Year',
      month STRING COMMENT 'Month',
      knots FLOAT COMMENT 'wind speed')
      PARTITIONED BY (station STRING COMMENT 'Station Name')
      ROW FORMAT DELIMITED
      FIELDS TERMINATED BY ','
      LINES TERMINATED BY '\n'
      STORED AS TEXTFILE;
```

Lesson 2, Step 6: Create an external table

```
CREATE EXTERNAL TABLE user01.temperature (
      station STRING COMMENT 'Station Name',
      year INT COMMENT 'Year',
      month STRING COMMENT 'Month',
      celsius FLOAT COMMENT 'degrees celsius')
      ROW FORMAT DELIMITED
      FIELDS TERMINATED BY ','
      LINES TERMINATED BY '\n'
      STORED AS TEXTFILE
      LOCATION '/user/user01/DA440_LabFiles/temperature';
```

Lesson 3, Step 1: Simple Queries

```
SELECT * FROM temperature
     WHERE station = 'Clean_Air'
     AND month = 'jul'
     AND celsius IS NOT NULL
     ORDER BY celsius;
```

Lesson 3, Step 3: User-Defined Functions

```
SELECT * FROM temperature WHERE celsius = '-66.6' OR celsius =
8.4;
```

Lesson 3, Step 4: Using arithmetic operators

```
SELECT station, celsius, round(celsius*1.8+32, 2) AS fahrenheit
     FROM temperature WHERE month = 'jan' AND year = 1970
     AND celsius IS NOT NULL;
```

Lesson 3, Step 5: Joining tables

```
SELECT w.station, w.year, w.month, w.knots,
     t.celsius, l.latitude, l.longitude FROM windspeed w
     JOIN location l ON w.station = l.station
     JOIN temperature t ON w.station = t.station
     AND w.month = t.month
     AND w.year = t.year
     WHERE w.year = 1970
     AND t.celsius IS NOT NULL
     AND w.knots IS NOT NULL;
```