DA 4000 – Data Analysis with Apache Drill
Lab Guide

Revision 1.1

For use with the following courses:

DA 4000 – Data Analysis with Apache Drill
ESS 400 – Apache Drill Essentials
DA 400 – Introduction to SQL Analytics with Apache Drill
DA 401 – Apache Drill Performance and Debugging

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 results in "skipped" numbers for lab exercises.

Lessons and Labs	Duration
Lesson 1: Introduction to Apache Drill	
• Lab 1.2a – Explore the Drill SQL Interfaces	10 min
• Lab 1.2b – Perform Drill SQL Queries	5 min
Lesson 2: SQL Queries with Apache Drill	
• Lab 2.1 – Describe Schemas	15 min
• Lab 2.2 – Query Marketing Data	10 min
• Lab 2.3 – Query JSON Data	15 min
• Lab 2.4 – Combine Data Types	10 min
Lesson 3: Apache Drill Operations and Functions	
• Lab 3.1 – Create and Drop Tables and Views	20 min
• Lab 3.2a – Perform Nested Data Functions	5 min
• Lab 3.2b – Perform Aggregate Window Functions	10 min
• Lab 3.4 – Perform an End-to-End Drill Data Analysis	25 min
Lesson 4: Apache Drill Architecture	
• Lab 4.1 – Order Query Process Steps	10 min
• Lab 4.2 – Sketch Drillbit Architecture	10 min
Lesson 5: Apache Drill Query Plans and Optimization	
• Lab 5.2 – Examine Physical Query Plans	15 min
• Lab 5.3 – Create a Partitioned Table	15 min
Lesson 6: Apache Drill Logging and Debugging	
• Lab 6.1 – Examine Drill Log Files	15 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/v5.2.0/sandbox/MapR-Sandbox-For-Apache-Drill-1.8.0-5.2.0-vmware.ova
- VirtualBox Course Sandbox: http://package.mapr.com/releases/v5.2.0/sandbox/MapR-Sandbox-For-Apache-Drill-1.8.0-5.2.0.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.

Important Notes on Command Syntax

Note the following that apply to UNIX command presented in lab exercises:

- 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
```

- Longer commands may not fit on a single line in the lab guide. For these commands, the backslash character (\ ) – or escape character – will be put at the end of a line to signify that the line feed should be ignored when typing in the command. For example, in this command:

```
$ hadoop jar /opt/mapr/hadoop/hadoop-<version>/share/hadoop/\
mapreduce/hadoop-mapreduce-examples-2.7.0-mapr-16-7.jar terasort \
/terasort-input
```

  the entire command should be entered one a single line.

  At the end of the first line, there is **no space** after `/hadoop/`, since there is no space before the escape character.

  At the end of the second line, there is a space after `terasort`, since there is a space before the escape character.

- Commands in lab guides are appropriate for CentOS, which the lab environments use. For other distributions, such as Ubuntu, substitute the appropriate equivalent commands where necessary.
- 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.

ESS 400 – Apache Drill Essentials

Part of the DA 4000 curriculum

Lesson 1: Introduction to Apache Drill

Lab 1.2a: Explore the Drill SQL Interfaces

Estimated time to complete: 10 minutes

Overview

Apache Drill client interfaces include a graphical user interface (GUI) called the Drill Web UI, another GUI called Drill Explorer, and a command-line interface (CLI) called SQLLine.

You can use either interface for executing your queries. The first lab will show you how to use all three. After that, however, instructions in the lab guide will generally be given using SQLLine.

Similar to the SQL shell of many popular RDBMS systems, SQLLine can be run interactively with commands typed at the prompt. SQLLine connects to Drillbits using JDBC and runs SQL commands. An alias has been created for this training so you only have to enter `sqlline` to invoke SQLLine when using the MapR Sandbox with Apache Drill.

Run a Simple Query

1. Log in to your MapR environment as `user01` with the password `mapr`. See the *Connection Guide* for details.
2. Start the SQLLine interface:

   ```
   $ sqlline
   ```

3. Enter the following query at the SQLLine prompt:

   ```
   > USE hive;
   ```

   You will see a confirmation similar to the following:

   ```
   0: jdbc:drill:> USE hive;
   +-------+-----------------------------------+
   |  ok   |              summary              |
   +-------+-----------------------------------+
   | true  | Default schema changed to [hive]  |
   +-------+-----------------------------------+
   1 row selected (0.102 seconds)
   ```

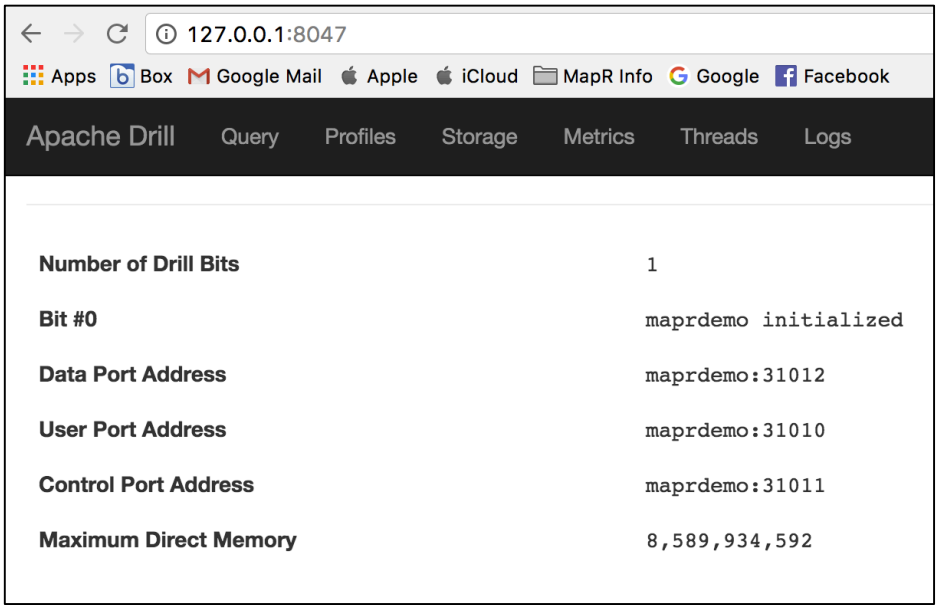
4. Run this `SELECT` query on the Hive database:

   ```
   > SELECT * FROM orders LIMIT 10;
   ```

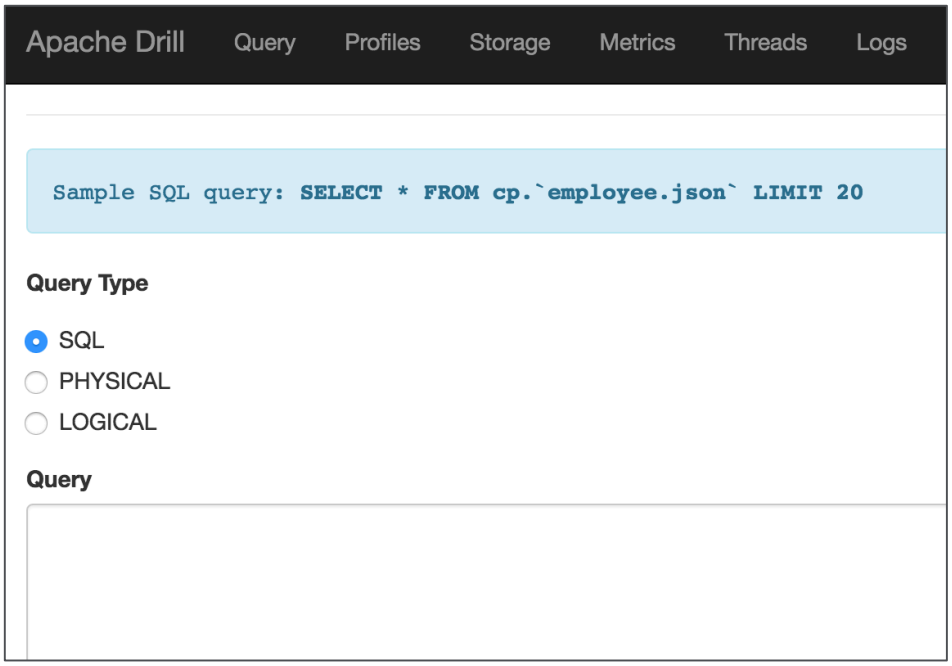
You will see 10 rows from the orders table, similar to the following:

```
0: jdbc:drill:> SELECT * FROM orders LIMIT 10;
+-----------+-------+----------+-------+---------+--------------+
| order_id  | month | cust_id  | state | prod_id | order_total  |
+-----------+-------+----------+-------+---------+--------------+
| 67212     | June  | 10001    | ca    | 909     | 13           |
| 70302     | June  | 10004    | ga    | 420     | 11           |
| 69090     | June  | 10011    | fl    | 44      | 76           |
| 68834     | June  | 10012    | ar    | 0       | 81           |
| 71220     | June  | 10018    | az    | 411     | 24           |
| 61287     | June  | 1001     | nj    | 104     | 134          |
| 68553     | June  | 10021    | ca    | 117     | 67           |
| 68109     | June  | 10022    | tx    | 337     | 10           |
| 68526     | June  | 10025    | mi    | 11      | 63           |
| 69362     | June  | 10028    | tx    | 430     | 65           |
+-----------+-------+----------+-------+---------+--------------+
10 rows selected (0.384 seconds)
```

5. Let's run this same query in the Web UI. Open a browser window and connect to the Drill UI at port 8047 (`<IP address>:8047`):



6. Click **Query** in the top menu. You will see the SQL query page, similar to the following:

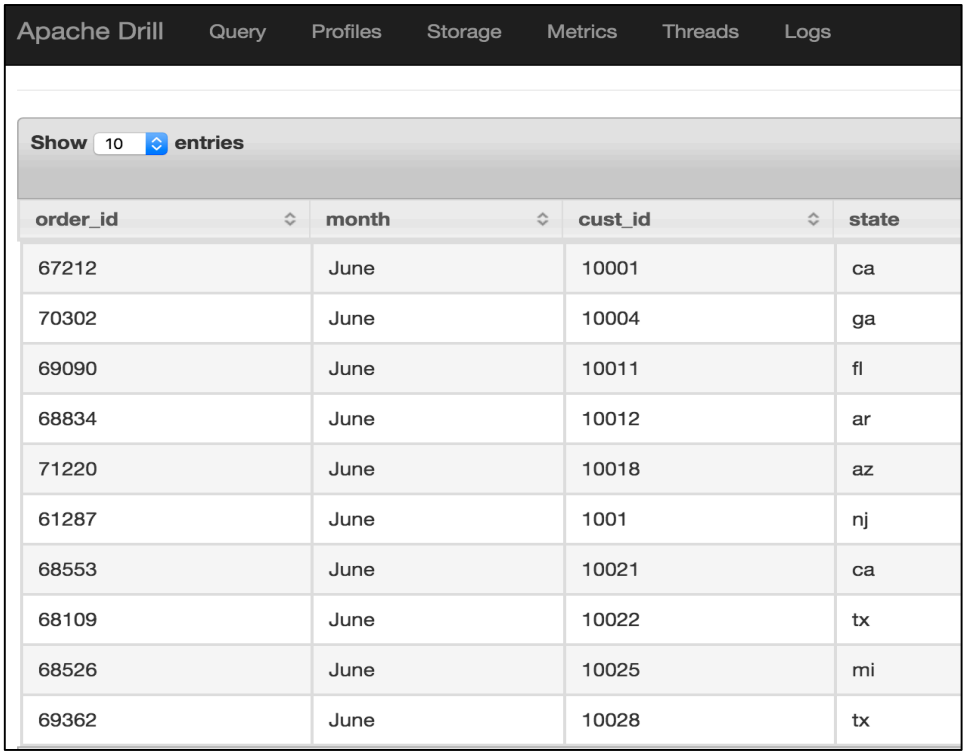


7. Make sure **SQL** is selected. Then, enter the `SELECT * FROM hive.orders LIMIT 10;` query in the box:

8. Click **Submit**. You will see the first ten rows of the `orders` table, similar to the following:

Note: If you are using the web interface, you must put the schema name at the beginning of the table name for every command. You can also omit the semicolon if you wish:

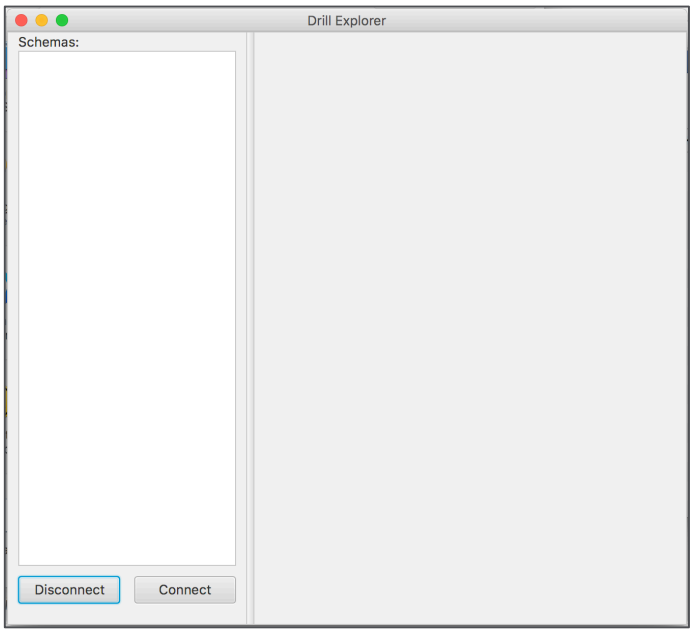
```
SELECT * FROM hive.orders LIMIT 10
```



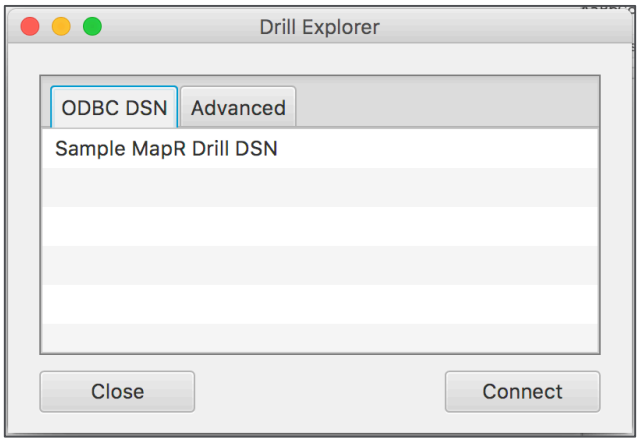
order_id	month	cust_id	state
67212	June	10001	ca
70302	June	10004	ga
69090	June	10011	fl
68834	June	10012	ar
71220	June	10018	az
61287	June	1001	nj
68553	June	10021	ca
68109	June	10022	tx
68526	June	10025	mi
69362	June	10028	tx

9. Drill also optionally includes a GUI called Drill Explorer. Instructor-led students will find a shortcut to Drill Explorer on their desktop. On-demand students will need to install Drill Explorer First.

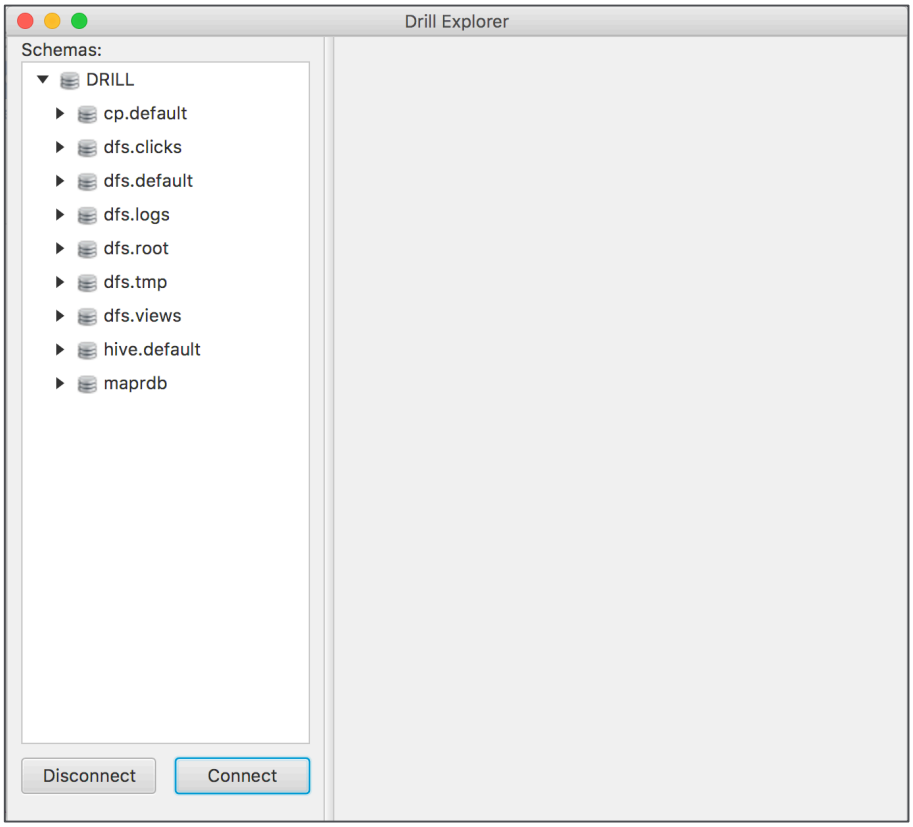
   Launch **Drill Explorer**. You will see the main Drill Explorer screen similar to the following:



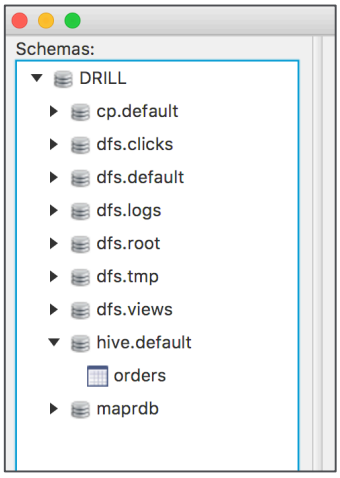
10. You will see a dialog box similar to the following. If you do not, click **Connect** in the main Drill Explorer screen.



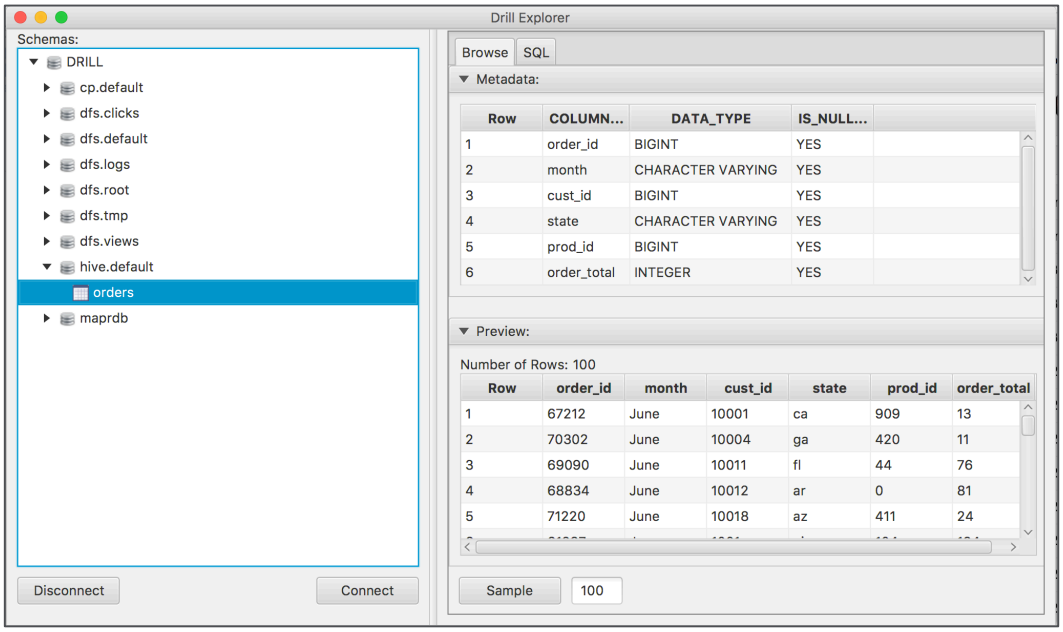
11. Select **Sample MapR Drill DSN** (or similar), then click **Connect**. When prompted, enter the IP address you used to connect to your MapR environment in Step 1.
12. Once you are connected, you will see all the schemas in the data sources this machine supports, similar to the following:



13. Click the arrow or plus sign by **hive.default**. It will expand, and you will see the table under this schema as follows:

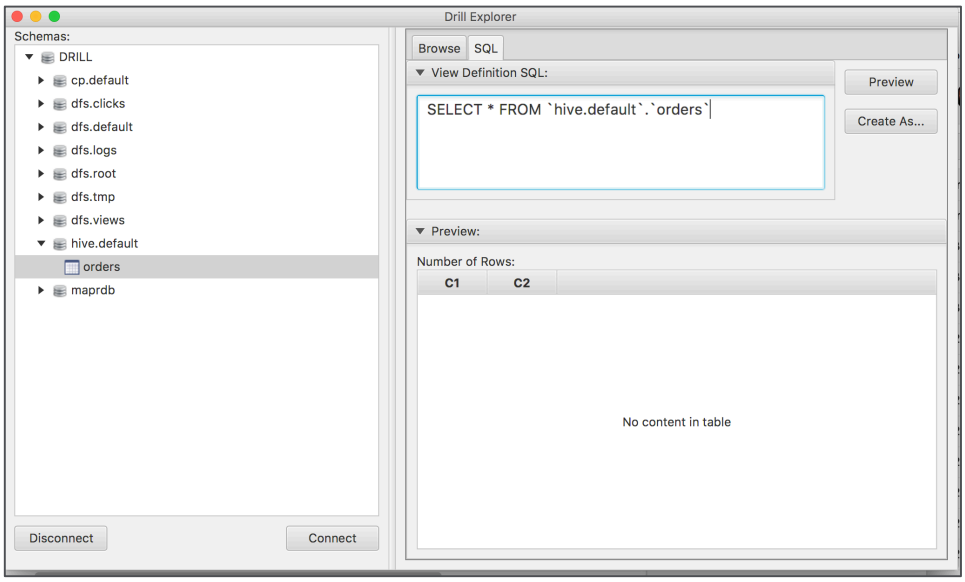


14. Click **orders**. You will see the following:



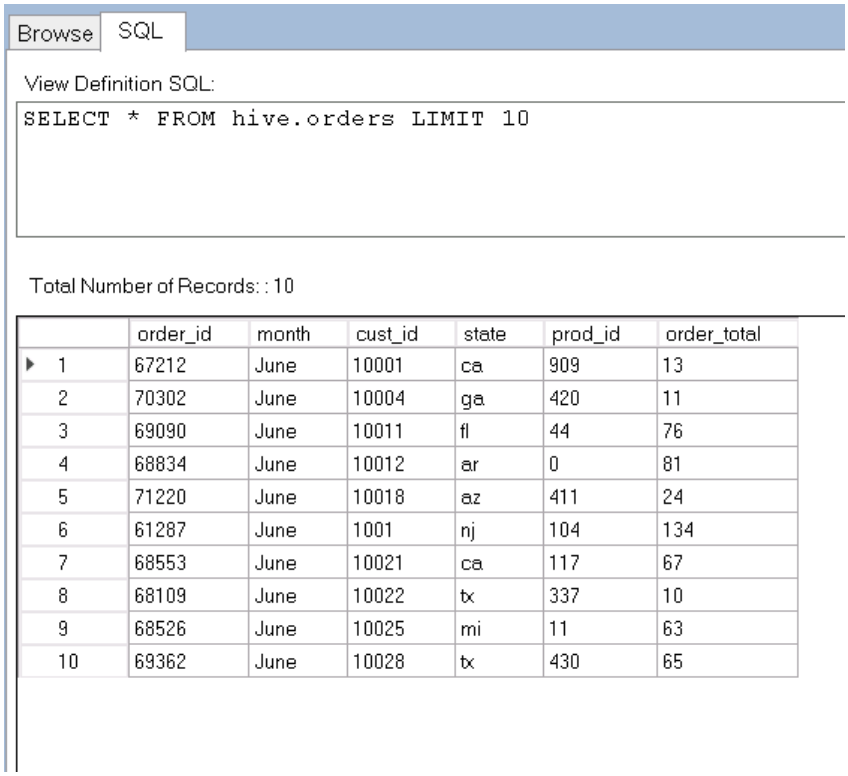
Note how you see a description of the columns in the top right pane, and the data in the table in the bottom right pane.

15. Click on the **SQL** tab at the top. You will see the SQL window, similar to the following:



This window is another place where you can execute SQL queries. In the labs, we will use the Drill WebUI or SQLLine to execute queries. But if you are more comfortable using the Drill Explorer, feel free to do so.

16. Enter `SELECT * FROM hive.orders LIMIT 10` and click **Preview. You will see results similar to the following:**



	order_id	month	cust_id	state	prod_id	order_total
1	67212	June	10001	ca	909	13
2	70302	June	10004	ga	420	11
3	69090	June	10011	fl	44	76
4	68834	June	10012	ar	0	81
5	71220	June	10018	az	411	24
6	61287	June	1001	nj	104	134
7	68553	June	10021	ca	117	67
8	68109	June	10022	tx	337	10
9	68526	June	10025	mi	11	63
10	69362	June	10028	tx	430	65

Note: In the Drill Explorer, you must put the schema name at the beginning of the table name for every command. You also must omit the semicolon for your queries to work.

```
SELECT * FROM hive.orders LIMIT 10
```

Lab 1.2b: Perform Drill SQL Queries

Estimated time to complete: 5 minutes

1. Go back to the SQLLine interface. Perform a SQL query on a Hive table:

   ```
   > SELECT * FROM hive.orders;
   ```

   You will see scrolling output similar to the following. In the example below, 122,000 rows are returned in 6.187 seconds. Notice that because the data is interactive, Drill displays column headings over each page.

```
| 63366     | May    | 9982     | md     | 14       | 82           |
| 64094     | May    | 9982     | ny     | 811      | 19           |
| 61533     | May    | 9988     | wa     | 11       | 28           |
| 62655     | May    | 9992     | ca     | 6        | 78           |
| 62451     | May    | 9993     | ny     | 805      | 21           |
| 61907     | May    | 9994     | me     | 795      | 60           |
+-----------+--------+----------+--------+----------+--------------+
| order_id  | month  | cust_id  | state  | prod_id  | order_total  |
+-----------+--------+----------+--------+----------+--------------+
| 63397     | May    | 9998     | il     | 766      | 13           |
+-----------+--------+----------+--------+----------+--------------+
122,000 rows selected (6.187 seconds)
0: jdbc:drill:>
```

2. Run another `SELECT` query on a JSON file:

```
> SELECT * FROM cp.`employee.json` LIMIT 10;
```

You will see scrolling output similar to the following:

```
Bachelors Degree  | S              | F       | Store Management   |
| 9           | Brenda Blumberg    | Brenda     | Blumberg    | 11          | Store Manager
   | 21        | 11             | 1979-06-23  | 1998-01-01 00:00:00.0  | 17000.0  | 5            |
Graduate Degree   | M              | F       | Store Management   |
| 10          | Darren Stanz       | Darren     | Stanz       | 5           | VP Finance
   | 0         | 5              | 1949-08-26  | 1994-12-01 00:00:00.0  | 50000.0  | 1            |
Partial College   | M              | M       | Senior Management  |
| 11          | Jonathan Murraiin  | Jonathan   | Murraiin    | 11          | Store Manager
   | 1         | 11             | 1967-06-20  | 1998-01-01 00:00:00.0  | 15000.0  | 5            |
Graduate Degree   | S              | M       | Store Management   |
+-------------+--------------------+----------+-------------+-------------+-----------------------
---+-----------+----------------+-------------+------------------------+----------+---------------+-
---------------+-------------+--------+-------------------+
10 rows selected (0.376 seconds)
0: jdbc:drill:>
```

Note: Querying JSON data with SQL is only possible with Drill.

Lessons Learned

- Drill contains both command-line and web interfaces

DA 400 – Introduction to SQL Analytics with Apache Drill

Part of the DA 4000 curriculum

Lesson 2: SQL Queries with Apache Drill

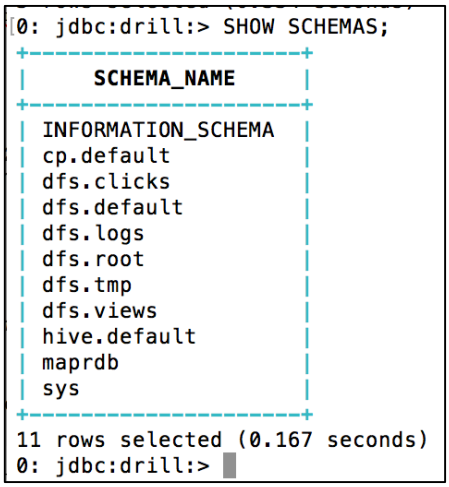
Lab 2.1: Describe Schemas

Estimated time to complete: 15 minutes

1. List the schemas in our database:

   ```
   > SHOW SCHEMAS;
   ```

   You will see a list of all the schemas in our data set, similar to the following:



2. Change the default schema to the Hive database. Enter the following query:

   ```
   > USE hive;
   ```

   Note: You cannot perform the `USE` query with Drill Explorer or the Drill Web UI. Instead, you must include the schema name with the table name on every query.

3. List the available tables in the hive schema. Enter the following query:

   ```
   > SHOW TABLES;
   ```

   You will see the names of the tables in the `hive.default` schema.

   Note: You cannot perform the `SHOW TABLES` query with Drill Explorer or the Drill Web UI.

4. The `hive.default` schema has one table, orders. Let's look at the structure of this table. Enter the following query:

```
> DESCRIBE orders;
```

You will see a description of the `orders` table, similar to the following:

```
0: jdbc:drill:> DESCRIBE orders;
+--------------+--------------------+--------------+
| COLUMN_NAME  |     DATA_TYPE      | IS_NULLABLE  |
+--------------+--------------------+--------------+
| order_id     | BIGINT             | YES          |
| month        | CHARACTER VARYING  | YES          |
| cust_id      | BIGINT             | YES          |
| state        | CHARACTER VARYING  | YES          |
| prod_id      | BIGINT             | YES          |
| order_total  | INTEGER            | YES          |
+--------------+--------------------+--------------+
6 rows selected (0.205 seconds)
```

Note: You cannot perform the `DESCRIBE` query with the Drill Web UI.

5. List the first 10 rows of the orders table:

```
> SELECT * FROM orders LIMIT 10;
```

You will see the first 10 rows of the orders table, similar to the following:

```
0: jdbc:drill:> SELECT * FROM orders LIMIT 10;
+-----------+--------+----------+--------+----------+--------------+
| order_id  | month  | cust_id  | state  | prod_id  | order_total  |
+-----------+--------+----------+--------+----------+--------------+
| 67212     | June   | 10001    | ca     | 909      | 13           |
| 70302     | June   | 10004    | ga     | 420      | 11           |
| 69090     | June   | 10011    | fl     | 44       | 76           |
| 68834     | June   | 10012    | ar     | 0        | 81           |
| 71220     | June   | 10018    | az     | 411      | 24           |
| 61287     | June   | 1001     | nj     | 104      | 134          |
| 68553     | June   | 10021    | ca     | 117      | 67           |
| 68109     | June   | 10022    | tx     | 337      | 10           |
| 68526     | June   | 10025    | mi     | 11       | 63           |
| 69362     | June   | 10028    | tx     | 430      | 65           |
+-----------+--------+----------+--------+----------+--------------+
10 rows selected (0.161 seconds)
```

Notice that we have the customer ID of the customer that placed the order, the state where the customer is located, and the product ID of the item ordered.

6. Now, let's look at the `maprdb` schema. Enter the following query:

```
> USE maprdb;
```

You will see a confirmation message, similar to the following:

```
0: jdbc:drill:> USE maprdb;
+-------+--------------------------------------+
|  ok   |               summary                |
+-------+--------------------------------------+
| true  | Default schema changed to [maprdb]   |
+-------+--------------------------------------+
1 row selected (0.1 seconds)
```

7. List the tables available in the `maprdb` schema. Enter the following query:

```
> SHOW TABLES;
```

You will see the names of the tables in the `maprdb` schema, similar to the following:

```
0: jdbc:drill:> SHOW TABLES;
+--------------+-----------------+
| TABLE_SCHEMA |   TABLE_NAME    |
+--------------+-----------------+
| maprdb       | customers       |
| maprdb       | embeddedclicks  |
| maprdb       | products        |
+--------------+-----------------+
3 rows selected (0.161 seconds)
0: jdbc:drill:>
```

Note that the orders table is a Hive table in the Hive schema. The `customers` and `products` tables are MapR-DB tables, and the `embeddedclicks` table is a JSON table.

8. Perform a SQL `SELECT` query to view the `customers` table. In order to view an MapR-DB table as readable text, we must use the `CONVERT_FROM` function. Enter the following query:

```
> SELECT CONVERT_FROM(customers.row_key,'UTF8')
AS cust_id, CONVERT_FROM(customers.address.state,'UTF8')
AS state FROM customers;
```

You will see scrolling output similar to the following:

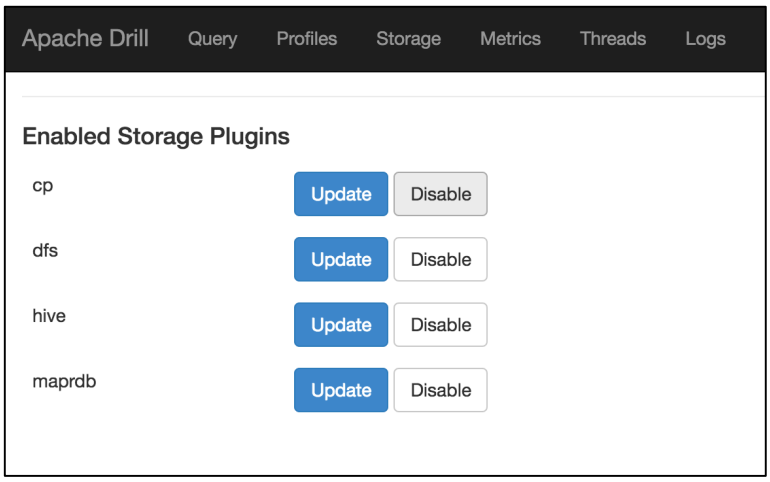
```
| 9974     | "ia"   |
| 9976     | "ny"   |
| 9977     | "ny"   |
| 9978     | "ny"   |
| 9979     | "fl"   |
| 9980     | "ca"   |
| 9983     | "ny"   |
+----------+--------+
12,792 rows selected (0.297 seconds)
```

Now you can query the customer ID (from the `row_key` column) and the state (from the `address` column) from the `customers` table. This will allow you to match these fields with the same fields from the `orders` table as you work through the rest of the labs.

Lab 2.2: Query Marketing Data

Estimated time to complete:10 minutes

1. Bring up the Drill Web UI and click the **Storage** tab:



   The Drill lab environment includes `cp`, `dfs`, `hive`, and `maprdb` storage plugin instances. The `cp` instance points to Drill's classpath. Drill's classpath is the location where various Java Archive (JAR) files are stored.

2. Click **Update** for the `hive` storage plugin. You will see a configuration file similar to the following:

```
{
  "type": "hive",
  "enabled": true,
  "configProps": {
    "hive.metastore.uris": "thrift://localhost:9083",
    "hive.metastore.sasl.enabled": "false"
  }
}
```

   This shows you all the parameters of the `hive` storage plugin. Here, we see that the `hive` plugin is enabled, which allows you to enter the `USE orders` command to get data from the `orders` table, which resides in the `hive` database. Now that we have verified that this plugin is enabled, we can start querying the data.

3. Our goal is to help the marketing department build a targeted advertising campaign. First, we will determine the top month of gross sales. Return to SQLLine, and enter the following query:

```
> SELECT `month`, SUM(order_total)
AS sales FROM hive.orders
GROUP BY `month` ORDER BY sales DESC;
```

You will see output similar to the following:

```
0: jdbc:drill:> SELECT `month`, SUM(order_total)
. . . . . . . > AS sales FROM hive.orders
. . . . . . . > GROUP BY `month` ORDER BY sales DESC;
+-----------+---------+
|   month   |  sales  |
+-----------+---------+
| June      | 950481  |
| May       | 947796  |
| March     | 836809  |
| April     | 807291  |
| July      | 757395  |
| October   | 676236  |
| August    | 572269  |
| February  | 532901  |
| September | 373100  |
| January   | 346536  |
+-----------+---------+
10 rows selected (0.781 seconds)
```

Note: We need to put backquotes around `month`, because `MONTH` is a reserved word in Drill SQL.

4. We receive a request from the marketing department to show the total of orders in each state for the top month of gross sales. Now that we know that the top month of gross sales is June, we can query for the sales total for that month in each state. Enter the following query at the Drill prompt:

```
> SELECT `month`, state, SUM(order_total)
AS sales FROM hive.orders WHERE `month`='June'
GROUP BY `month`, state ORDER BY sales DESC;
```

You will see scrolling output similar to the following:

```
0: jdbc:drill:> SELECT `month`, state, SUM(order_total)
. . . . . . . > AS sales FROM hive.orders WHERE `month`='June'
. . . . . . . > GROUP BY `month`, state ORDER BY sales DESC;
+--------+--------+---------+
| month  | state  |  sales  |
+--------+--------+---------+
| June   | ca     | 116322  |
| June   | tx     | 78363   |
| June   | fl     | 62199   |
| June   | ny     | 62052   |
| June   | il     | 38657   |
| June   | oh     | 37671   |
| June   | pa     | 36670   |
| June   | mi     | 29396   |
| June   | ga     | 28990   |
| June   | nc     | 27554   |
| June   | nj     | 24443   |
```

5. Our marketing department only wants to see the top three grossing months. How would you write this query to only display this data? Try to build this query on your own. If you need help with this or other queries, refer to the answer key at the end of this lesson.

6. Now, let's find the top ten products based on volume of sales. Enter this query at the Drill prompt:

```
> SELECT prod_id, SUM(order_total)
AS sales FROM hive.orders GROUP BY prod_id
ORDER BY sales DESC LIMIT 10;
```

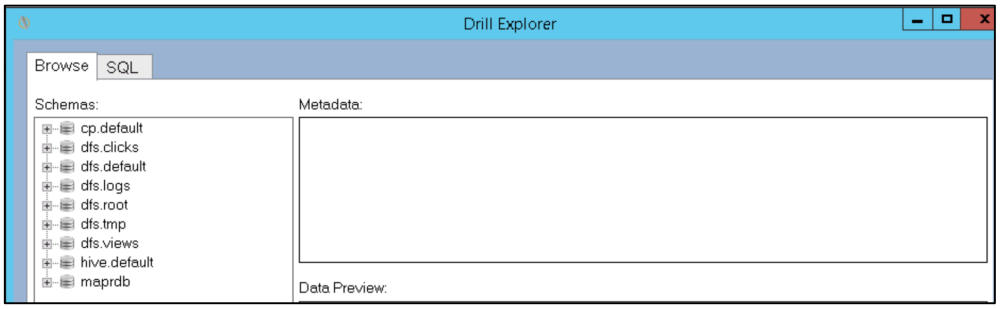
   You will see output similar to the following:

```
0: jdbc:drill:> SELECT prod_id, SUM(order_total)
. . . . . . . > AS sales FROM hive.orders GROUP BY prod_id
. . . . . . . > ORDER BY sales DESC LIMIT 10;
+----------+---------+
| prod_id  |  sales  |
+----------+---------+
| 0        | 204861  |
| 1        | 117177  |
| 2        | 92399   |
| 3        | 72571   |
| 4        | 64714   |
| 5        | 57969   |
| 6        | 53955   |
| 8        | 46547   |
| 7        | 44829   |
| 9        | 42482   |
+----------+---------+
10 rows selected (0.318 seconds)
```

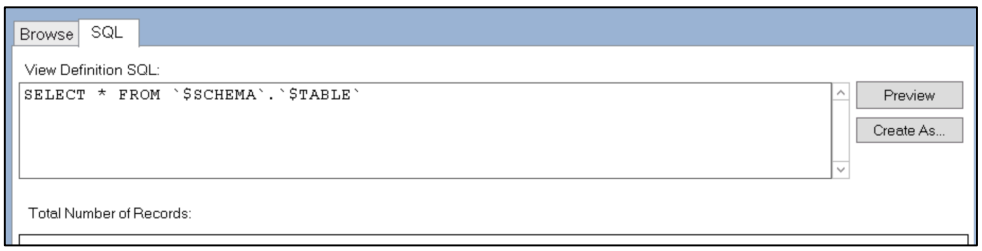
7. Our marketing department sends another request. They are only interested in states that generated more than $50,000 in sales in June. How would you write this query?

8. Now the marketing department wants to see the name of the product rather than the product ID. How would you write this query?

9. Let's see how these queries are performed in the Drill Explorer interface. Launch Drill Explorer.

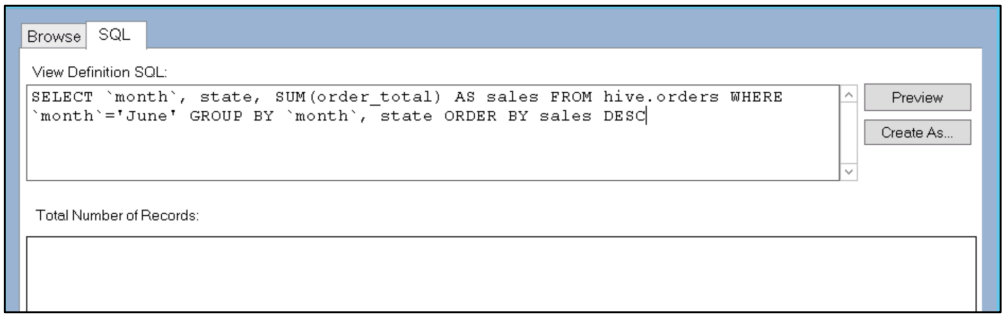


10. Click the **SQL** tab to switch to the SQL view:

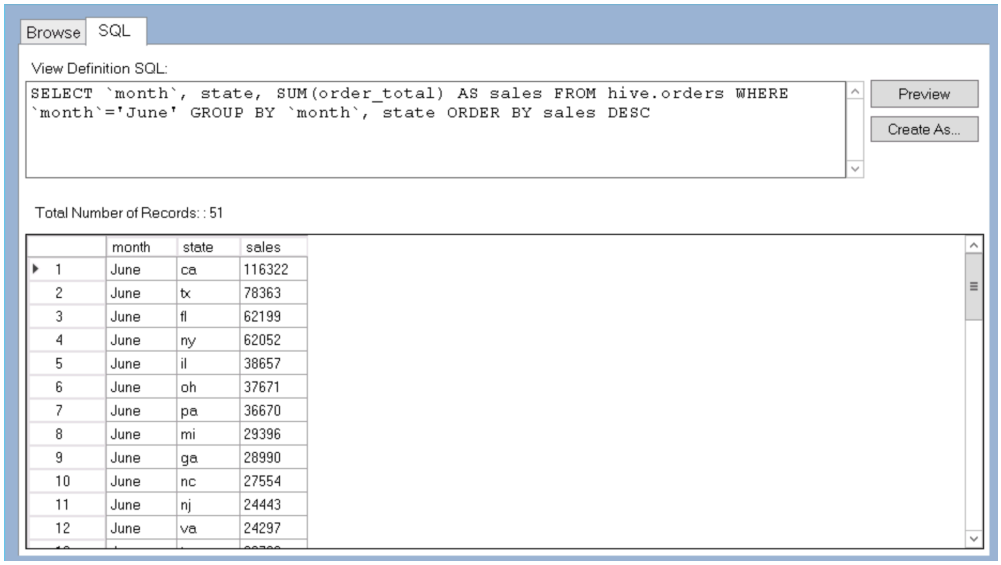


11. Let's try one of the queries we executed previously. In the **View Definition SQL:** text box, enter the following query:

```
SELECT `month`, state, SUM(order_total)
AS sales FROM hive.orders WHERE `month`='June'
GROUP BY `month`, state ORDER BY sales DESC
```



12. Click **Preview** to execute the query:



	month	state	sales
1	June	ca	116322
2	June	tx	78363
3	June	fl	62199
4	June	ny	62052
5	June	il	38657
6	June	oh	37671
7	June	pa	36670
8	June	mi	29396
9	June	ga	28990
10	June	nc	27554
11	June	nj	24443
12	June	va	24297

Lab 2.3: Query JSON Data

Estimated time to complete: 15 minutes

1. Go back to the SQLLine interface. Change the schema to `dfs.clicks`:

```
> USE dfs.clicks;
```

2. Find the transaction IDs of orders placed between April 5, 2014, and January 31, 2015. Enter the following query at the Drill prompt:

```
> SELECT c.trans_id, CAST(c.`date` AS DATE)
FROM `clicks/clicks.json` c WHERE c.`date`
BETWEEN '2014-04-05' AND '2015-01-31';
```

You will see scrolling output similar to the following:

```
| 29615    | 2014-04-30  |
| 21448    | 2014-04-30  |
| 29972    | 2014-04-30  |
| 25157    | 2014-04-30  |
| 27765    | 2014-04-30  |
| 29179    | 2014-04-30  |
| 28907    | 2014-04-30  |
| 29998    | 2014-04-30  |
| 22693    | 2014-04-30  |
+----------+-------------+
27,184 rows selected (0.812 seconds)
0: jdbc:drill:>
```

3. For any customer who visits the site, find the customer ID, the type of device they use to place their order, and the state they live in:

```
> SELECT t.user_info.cust_id AS custid,
t.user_info.device AS device,
t.user_info.state AS state
FROM `clicks/clicks.json` t LIMIT 5;
```

You will see output similar to the following:

custid	device	state
22526	IOS5	il
16368	AOS4.2	nc
21449	IOS6	oh
20323	IOS5	oh
15360	IOS5	ca

Note: The `LIMIT` clause is not needed, but has been added to this and several other queries to improve the readability of the output.

4. Now, find out whether customers purchased the products they viewed. Enter the following query at the Drill prompt:

```
> SELECT t.trans_info.prod_id AS prodid,
t.trans_info.purch_flag AS purchased
FROM `clicks/clicks.json` t LIMIT 5;
```

You will see output similar to the following:

```
0: jdbc:drill:> SELECT t.trans_info.prod_id AS prodid,
. . . . . . . > t.trans_info.purch_flag  AS purchased
. . . . . . . > FROM `clicks/clicks.json` t LIMIT 5;
+-----------------------------------------+-----------+
|                 prodid                  | purchased |
+-----------------------------------------+-----------+
| [174,2]                                 | false     |
| []                                      | false     |
| [582]                                   | false     |
| [710,47]                                | false     |
| [0,8,170,173,1,124,46,764,30,711,0,3,25] | true      |
+-----------------------------------------+-----------+
```

In this example, we see four customers did not purchase anything, but one customer did.

5. Which customers searched for at least 10 products? Enter the following query at the SQLLine prompt:

```
> SELECT t.user_info.cust_id AS cust_id,
t.trans_info.prod_id[9] AS prod_id
FROM `clicks/clicks.json` t
WHERE t.trans_info.prod_id[9] IS NOT NULL
ORDER BY trans_id LIMIT 5;
```

You will see output similar to the following:

```
0: jdbc:drill:> SELECT t.user_info.cust_id AS cust_id,
. . . . . . . > t.trans_info.prod_id[9] AS prod_id
. . . . . . . > FROM `clicks/clicks.json` t
. . . . . . . > WHERE t.trans_info.prod_id[9] IS NOT NULL
. . . . . . . > ORDER BY trans_id LIMIT 5;
+----------+----------+
| cust_id  | prod_id  |
+----------+----------+
| 2587     | 0        |
| 1772     | 5        |
| 1412     | 0        |
| 1429     | 3        |
| 4344     | 13       |
+----------+----------+
```

Arrays use the `[n]` notation, where `n` is the position of the value in an array, starting from position 0. So, `trans_info.prod_id[0]` refers to the first value in the nested `prod_id` column. By using `[9]`, this query returns customer IDs and product IDs for records that contain a non-null product ID at the 10th position in the array. In other words, it only returns data if there are at least 10 values in the array `prod_id` column.

6. Let's see which products have been purchased. Enter the following query:

```
> SELECT t.user_info.cust_id AS cust_id, t.trans_info.prod_id
AS prod_id, t.trans_info.purch_flag AS purchased
FROM `clicks/clicks.json` t
WHERE t.trans_info.purch_flag='true' LIMIT 5;
```

You will see output similar to the following:

```
0: jdbc:drill:> SELECT t.user_info.cust_id AS cust_id, t.trans_info.prod_id
. . . . . > AS prod_id, t.trans_info.purch_flag AS purchased
. . . . . > FROM `clicks/clicks.json` t
. . . . . > WHERE t.trans_info.purch_flag='true' LIMIT 5;
+----------+-------------------------------------------+------------+
| cust_id  |                  prod_id                  | purchased  |
+----------+-------------------------------------------+------------+
| 15360    | [0,8,170,173,1,124,46,764,30,711,0,3,25]  | true       |
| 16996    | [40,499,15]                               | true       |
| 20109    | []                                        | true       |
| 17557    | [4]                                       | true       |
| 16331    | [1]                                       | true       |
+----------+-------------------------------------------+------------+
```

This query returns customer IDs and product IDs where the `purchased` flag is `true`.

7. Enter the following query:

```
> SELECT * FROM (SELECT t.user_info.cust_id
AS cust_id, t.trans_info.prod_id[0] AS prodid
FROM `clicks/clicks.json` t) sq WHERE sq.prodid
BETWEEN 700 AND 750 ORDER BY sq.prodid;
```

You will see output similar to the following:

```
| 7989     | 748     |
| 7956     | 748     |
| 17418    | 749     |
| 8907     | 749     |
| 16286    | 750     |
| 16818    | 750     |
| 17967    | 750     |
+----------+---------+
155 rows selected (0.449 seconds)
```

This returns customer IDs and product IDs where the first `prod_id` is between 700 and 750.

8. Enter the following query:

```
> SELECT o.cust_id, o.order_total, o.state,
t.trans_info.prod_id[0] AS prod_id FROM hive.orders
AS o, `clicks/clicks.json` t
WHERE o.cust_id=t.user_info.cust_id;
```

You will see scrolling output similar to the following:

```
| 6962     | 37          | nj     | 481      |
| 6962     | 13          | il     | 481      |
| 6962     | 75          | pa     | 481      |
| 5425     | 34          | tx     | null     |
| 5425     | 19          | ca     | null     |
| 5425     | 11          | ca     | null     |
+----------+-------------+--------+----------+
208,433 rows selected (7.831 seconds)
```

9. Create a table with the results from the previous query. Enter the following query at the Drill prompt:

```
> CREATE TABLE product_search AS
SELECT o.cust_id, o.order_total, o.state,
t.trans_info.prod_id[0] AS prod_id
FROM hive.orders AS o, `clicks/clicks.json` t
WHERE o.cust_id=t.user_info.cust_id;
```

You will see output similar to the following:

```
0: jdbc:drill:> CREATE TABLE product_search AS
. . . . . > SELECT o.cust_id, o.order_total, o.state,
. . . . . > t.trans_info.prod_id[0] AS prod_id
. . . . . > FROM hive.orders AS o, `clicks/clicks.json` t
. . . . . > WHERE o.cust_id=t.user_info.cust_id;
+----------+---------------------------+
| Fragment | Number of records written |
+----------+---------------------------+
| 0_0      | 208433                    |
+----------+---------------------------+
```

10. Query the new table for a list of products searched for by customers who have placed orders that are above average in total cost in their states. Enter the following query:

```
> SELECT p.cust_id, p.state, p.order_total, p.prod_id
FROM product_search p WHERE p.order_total >
(SELECT AVG(o.order_total) FROM product_search o
WHERE o.state = p.state);
```

You will see scrolling output similar to the following:

```
+---------+-------+-------------+---------+
| cust_id | state | order_total | prod_id |
+---------+-------+-------------+---------+
| 7790    | tx    | 72          | 24      |
| 7790    | ny    | 76          | 24      |
| 7413    | va    | 64          | null    |
| 6988    | md    | 75          | 782     |
| 6988    | ga    | 67          | 782     |
| 9258    | ca    | 78          | null    |
| 9258    | ga    | 70          | null    |
| 8896    | ca    | 92          | null    |
| 6962    | tx    | 94          | 481     |
| 6962    | pa    | 75          | 481     |
+---------+-------+-------------+---------+
```

11. Next, we'll look at the JSON embedded clicks table. This table has one Column Family blob, which contains one JSON column.

First, we need to set the maximum width of the JSON columns in the table. Enter the following command at the Drill prompt:

```
> !set maxwidth 10000
```

12. View the columns in the `embeddedclicks` table:

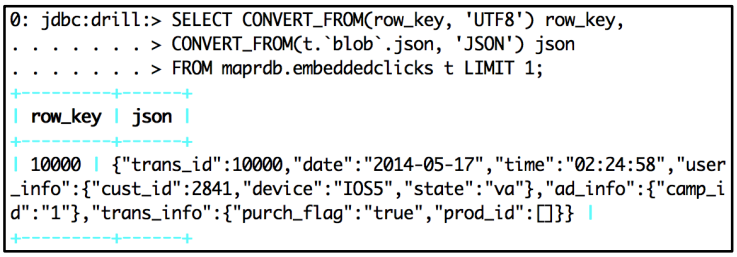
```
> SELECT * FROM maprdb.embeddedclicks t LIMIT 1;
```

You will see output similar to the following:

13. Now, use the `CONVERT_FROM` function to convert the bytes to a readable format. Enter the following query at the Drill prompt:

```
> SELECT CONVERT_FROM(row_key, 'UTF8') row_key,
CONVERT_FROM(t.`blob`.json, 'JSON') json
FROM maprdb.embeddedclicks t LIMIT 1;
```

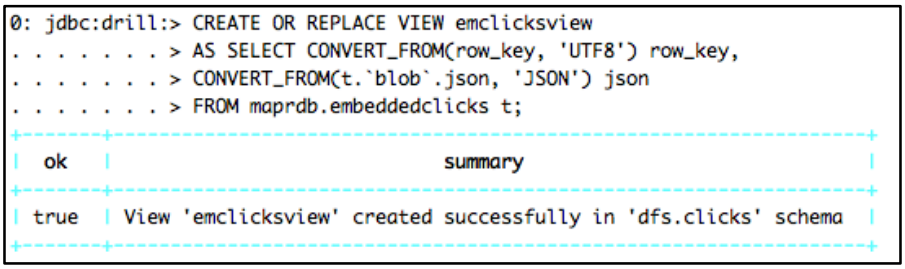
You will see output similar to the following:



14. Create a view from the previous query using `CREATE OR REPLACE VIEW`. Omit `LIMIT` to ensure the entire JSON file becomes a view. Enter the following:

```
> CREATE OR REPLACE VIEW emclicksview
AS SELECT CONVERT_FROM(row_key, 'UTF8') row_key,
CONVERT_FROM(t.`blob`.json, 'JSON') json
FROM maprdb.embeddedclicks t;
```

You will see output similar to the following:



15. Query the view you just created. You will see that, with the view, the blob is converted to JSON format. Enter the following query at the Drill prompt:

```
> SELECT * FROM emclicksview LIMIT 2;
```

You will see output similar to the following:

```
0: jdbc:drill:> select * from emclicksview limit 2;
+---------+-------+
| row_key | json  |
+---------+-------+
| 10000 | {"trans_id":10000,"date":"2014-05-17","time":"02:24:58","user_info":{"cust_id":2
841,"device":"IOS5","state":"va"},"ad_info":{"camp_id":"1"},"trans_info":{"purch_flag":"tr
ue","prod_id":[]}} |
| 10001 | {"trans_id":10001,"date":"2014-05-22","time":"17:51:14","user_info":{"cust_id":3
222,"device":"IOS5","state":"fl"},"ad_info":{"camp_id":"1"},"trans_info":{"purch_flag":"fa
lse","prod_id":[51,4,2,1,0,0,47,1,9,1,83]}} |
+---------+-------+
2 rows selected (0.267 seconds)
```

16. With this view, you can use the `table.column.column` notation to extract nested column data from the JSON blob column. Enter the following query at the Drill prompt:

```
> SELECT row_key, em.json.trans_id
AS transid, em.json.`date`
AS edate, em.json.user_info.cust_id
AS cust_id , em.json.trans_info.purch_flag
AS purch_flag, em.json.trans_info.prod_id
AS prod_id FROM emclicksview em LIMIT 2;
```

You will see output similar to the following:

```
0: jdbc:drill:> SELECT row_key, em.json.trans_id AS transid, em.json.`date` AS edate, em.json
.user_info.cust_id AS cust_id , em.json.trans_info.purch_flag AS purch_flag, em.json.trans_in
fo.prod_id AS prod_id FROM emclicksview em LIMIT 2;
+----------+----------+------------+----------+------------+-----------------------------+
| row_key  | transid  |    edate   | cust_id  | purch_flag |           prod_id           |
+----------+----------+------------+----------+------------+-----------------------------+
| 10000    | 10000    | 2014-05-17 | 2841     | true       | []                          |
| 10001    | 10001    | 2014-05-22 | 3222     | false      | [51,4,2,1,0,0,47,1,9,1,83]  |
+----------+----------+------------+----------+------------+-----------------------------+
2 rows selected (0.366 seconds)
```

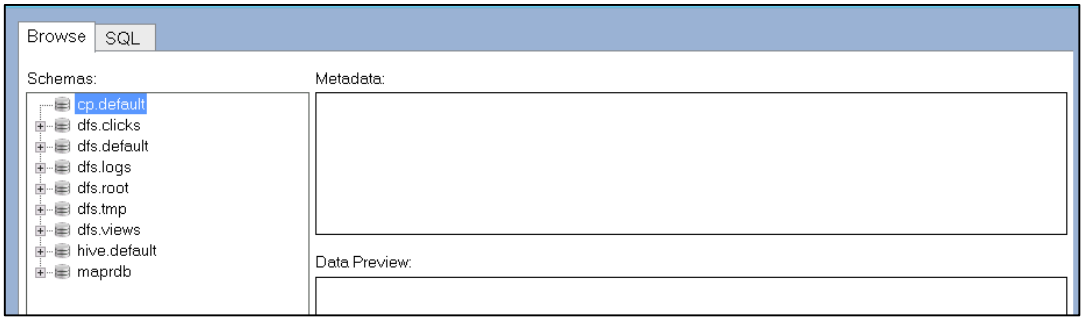
17. With this view you can also use the [n] notation on arrays in the JSON blob column. View fields from the `emclicksview`:

```
> SELECT row_key, em.json.trans_id
AS transid, em.json.`date`
AS edate, em.json.user_info.cust_id
AS cust_id, em.json.trans_info.purch_flag
AS purch_flag, em.json.trans_info.prod_id[0]
AS prod_id FROM emclicksview em LIMIT 2;
```

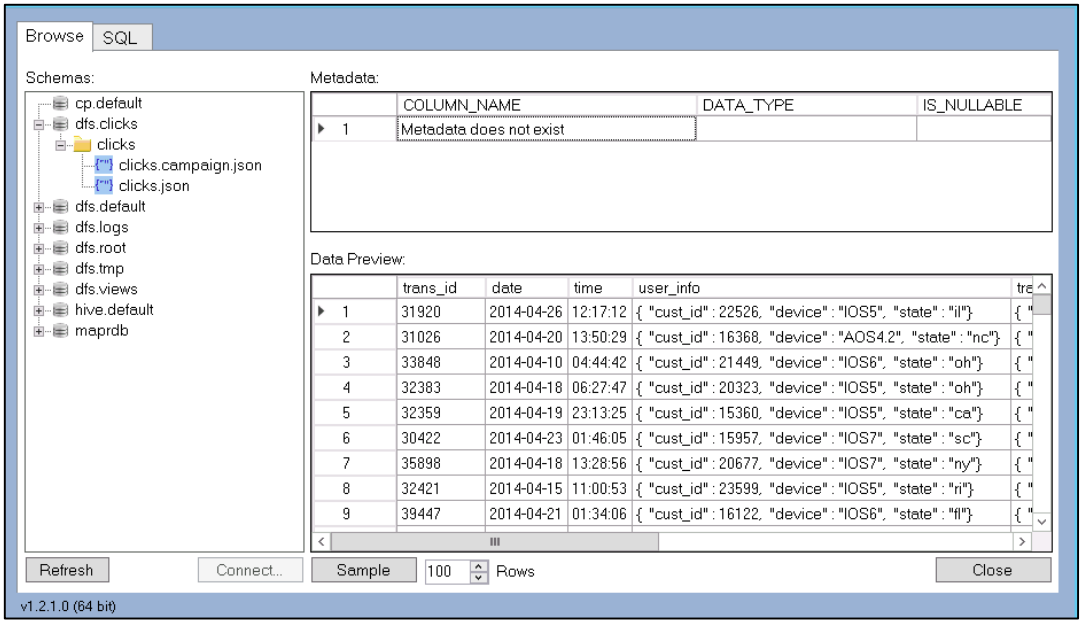
You will see output similar to the following:

```
0: jdbc:drill:> SELECT row_key, em.json.trans_id AS transid, em.json.`date` AS edate, em.jso
.user_info.cust_id AS cust_id , em.json.trans_info.purch_flag AS purch_flag, em.json.trans_i
fo.prod_id[0] AS prod_id FROM emclicksview em LIMIT 2;
+----------+----------+------------+----------+--------------+----------+
| row_key  | transid  |   edate    | cust_id  | purch_flag   | prod_id  |
+----------+----------+------------+----------+--------------+----------+
| 10000    | 10000    | 2014-05-17 | 2841     | true         | null     |
| 10001    | 10001    | 2014-05-22 | 3222     | false        | 51       |
+----------+----------+------------+----------+--------------+----------+
2 rows selected (0.269 seconds)
```

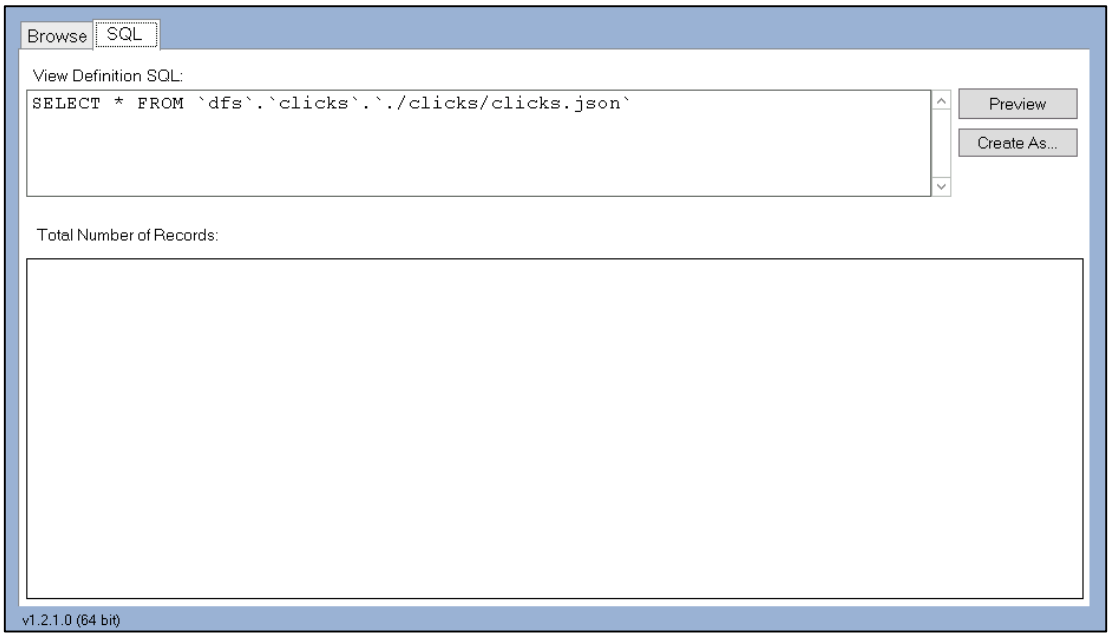
18. Drill Explorer makes it easy to see JSON files, without having to make queries or do conversions. Launch Drill Explorer:



19. Expand **`dfs.clicks > clicks > clicks.json`**. You will see the contents of the `clicks.json` file:

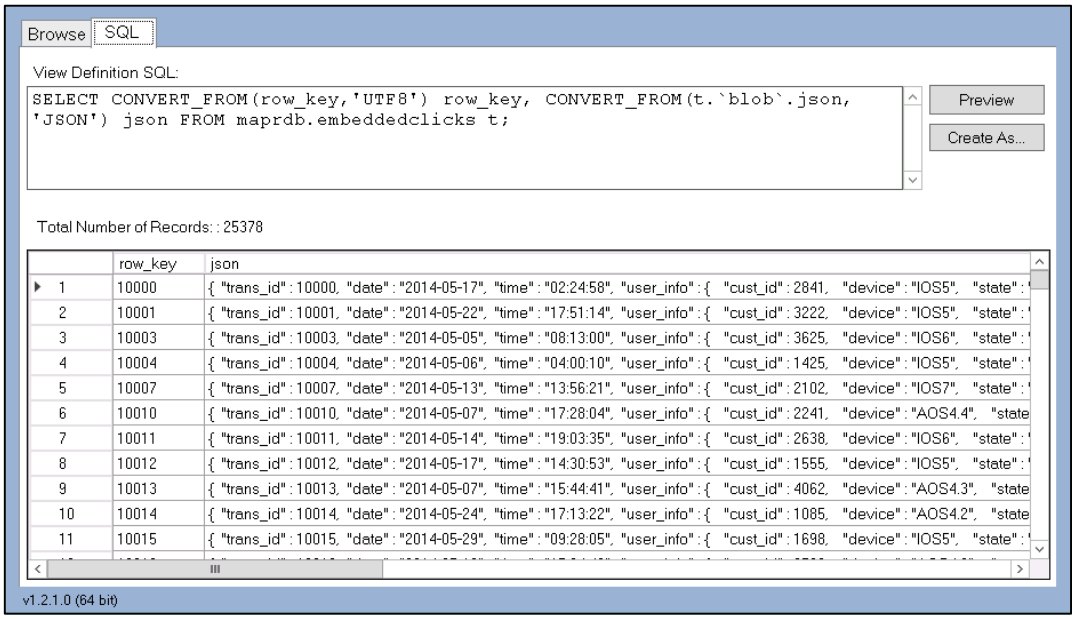


20. Let's execute the statement we used to create `emclicksview`. Click the **SQL** tab. You will see the query which generated the output in the **Browse** window:

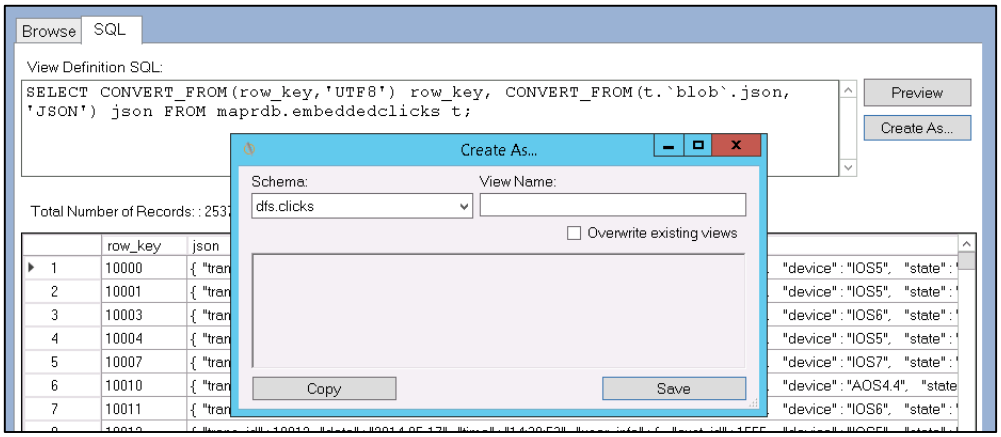


21. Enter the query used to generate `emclicksview` and click **Preview**:

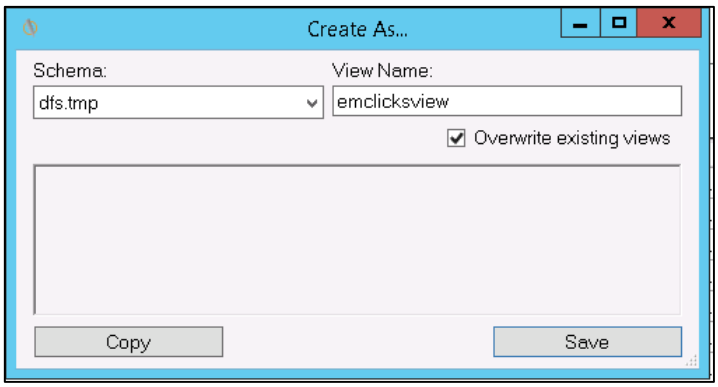
```
SELECT CONVERT_FROM(row_key, 'UTF8') row_key,
CONVERT_FROM(t.`blob`.json, 'JSON')
json FROM maprdb.embeddedclicks t
```



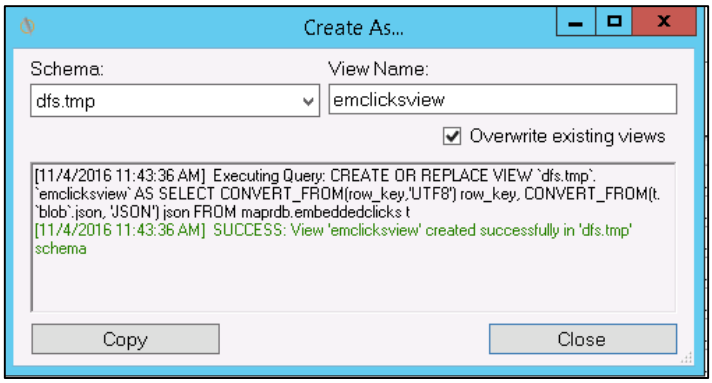
22. Click **Create As…** A dialog will open that prompts you for a **Schema** and a **View Name**:



23. Choose **`dfs.tmp`** for the Schema, and **`emclicksview`** for the View Name:



24. Click **Save**. You will see a confirmation message:



25. Click **Close** to close the dialog.

    In Drill Explorer, the view is not only created in the schema, but saved as a file named `emclicksview.drill` as well.

Lab 2.4: Combine Data Types

Estimated time to complete: 10 minutes

1. What products did each customer order? Enter the following query at the SQLLine prompt:

```
> SELECT o.cust_id, o.order_total, o.state,
t.trans_info.prod_id[0] AS prod_id
FROM hive.orders AS o, `clicks/clicks.json` t
WHERE o.cust_id=t.user_info.cust_id;
```

   You will see scrolling output similar to the following:

```
0: jdbc:drill:> SELECT o.cust_id, o.order_total, o.state, t.trans_info.prod_id[0
] AS prod_id FROM hive.orders AS o, `clicks/clicks.json` t WHERE o.cust_id=t.use
r_info.cust_id LIMIT 10;
+----------+--------------+--------+----------+
| cust_id  | order_total  | state  | prod_id  |
+----------+--------------+--------+----------+
| 22526    | 25           | mt     | 174      |
| 22526    | 17           | ky     | 174      |
| 22526    | 22           | ca     | 174      |
| 16368    | 25           | ny     | null     |
| 21449    | 18           | tx     | 582      |
| 21449    | 27           | mn     | 582      |
| 21449    | 22           | va     | 582      |
| 20323    | 16           | il     | 710      |
| 20323    | 41           | ma     | 710      |
| 20323    | 19           | pa     | 710      |
+----------+--------------+--------+----------+
10 rows selected (0.597 seconds)
```

2. Now, let's do the same query with the keyword `JOIN`:

```
> SELECT o.cust_id, o.order_total, o.state,
t.trans_info.prod_id[0] AS prod_id
FROM hive.orders AS o JOIN `clicks/clicks.json` t
ON o.cust_id=t.user_info.cust_id;
```

   You will see scrolling output similar to the following:

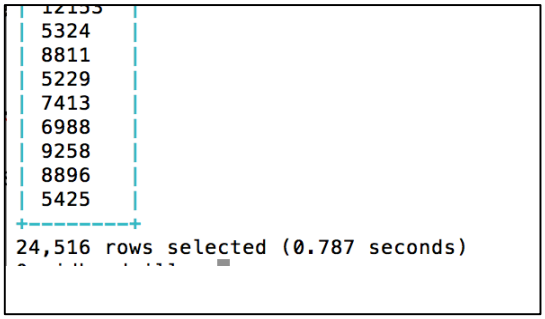
```
0: jdbc:drill:> SELECT o.cust_id, o.order_total, o.state, t.trans_info.prod_id[0
] AS prod_id FROM hive.orders AS o JOIN `clicks/clicks.json` t ON o.cust_id=t.us
er_info.cust_id LIMIT 10;
+----------+--------------+--------+----------+
| cust_id  | order_total  | state  | prod_id  |
+----------+--------------+--------+----------+
| 22526    | 25           | mt     | 174      |
| 22526    | 17           | ky     | 174      |
| 22526    | 22           | ca     | 174      |
| 16368    | 25           | ny     | null     |
| 21449    | 18           | tx     | 582      |
| 21449    | 27           | mn     | 582      |
| 21449    | 22           | va     | 582      |
| 20323    | 16           | il     | 710      |
| 20323    | 41           | ma     | 710      |
| 20323    | 19           | pa     | 710      |
+----------+--------------+--------+----------+
10 rows selected (0.401 seconds)
```

Notice that the output is exactly the same. You can use either query to combine the output of multiple tables. There is no difference in performance with these two queries – use whichever one you are most comfortable with.

3. Let's perform another query:

```
> SELECT c.user_info.cust_id
FROM `clicks/clicks.json` c
WHERE c.trans_info.purch_flag = 'false';
```

You will see output similar to the following:

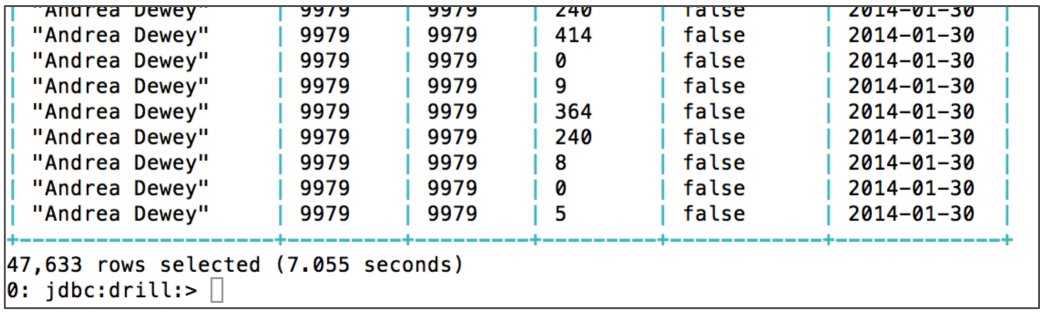


Now, we can use the clicks data to see which customers viewed, but did not purchase, particular products.

4. Combine the tables with a `WHERE` clause:

```
> SELECT * FROM (SELECT CAST(c1.personal.name AS VARCHAR(40))
AS cust_name, CAST(c1.row_key AS BIGINT)
AS row_key FROM maprdb.customers AS c1) AS cust,
(SELECT CAST(c.user_info.cust_id AS BIGINT)
AS cust_id, CAST(FLATTEN(c.trans_info.prod_id) AS BIGINT)
AS prod_id, c.trans_info.purch_flag AS purch_flag,
TO_DATE(c.`date`, 'yyyy-mm-dd') AS `date`
FROM `/clicks/clicks.json` AS c) AS clk
WHERE cust.row_key = clk.cust_id AND purch_flag = FALSE;
```

You will see scrolling output similar to the following:



You can now view output from the `customer` table and the `clicks.json` table in one query.

Lessons Learned

- Drill allows us to make ANSI SQL queries on different types of data, such as Hive, HBase, MapR-DB, and JSON
- Storage Plugins allow Drill to interface with a type of data set, such as MapR-DB, Hive, or JSON. Several storage plugins are defined by Drill. You can update these storage plugins. Storage Plugins can be enabled or disabled
- With Drill Explorer, we can view data from JSON files without having to make any queries or data conversions. Drill Explorer also automatically saves any views we create as files on disk
- Drill allows us to perform queries which combine information from tables of different types of data
- To join tables, you can either use the SQL keyword `JOIN` or a `WHERE` clause – there is no difference in performance

Lesson 2 Answer Key

Lab 2.2: Query Marketing Data

Step	Instruction	Solution
Step 5	Show only the top three grossing months of sales	`> SELECT;`month`, SUM(order_total) AS sales FROM hive.orders GROUP BY `month` ORDER BY sales DESC LIMIT 3;`
Step 7	Show only states with more than $50,000 in sales in June	`> SELECT * FROM (SELECT `month`, state, SUM(order_total) AS sales FROM hive.orders WHERE `month` = 'June' GROUP BY `month`, state) qry WHERE sales > 50000;`
Step 8	Display product name instead of product ID number	`> SELECT o.prod_id AS prod_id, CAST(p.details.name AS VARCHAR(40) AS name, SUM(o.order_total) AS sales FROM hive.orders o, maprdb.products p WHERE o.prod_id = CAST(p.row_key AS BIGINT) GROUP BY o.prod_id, p.details.name ORDER BY sales DESC LIMIT 10;`

Lesson 3: Apache Drill Operations and Functions

Lab 3.1 Create and Drop Tables and Views

Estimated time to complete: 20 minutes

Big Office Supply Company has a customer loyalty program. This program has three different levels – gold, silver, and platinum – based on the amount that a customer has purchased during the past year.

This information is stored in the `maprdb.customers` table. This view is in the `dfs.views` schema. In this lab, you will create a table to store this information. The table will have two columns: `cust_id` and `loyalty_level`. We are going to create this table in the `dfs.views` schema, because this is a schema that we have permissions to write to in our lab environment.

1. In SQLLine, change the schema to `dfs.views`:

   ```
   > USE dfs.views;
   ```

2. Create the loyalty table:

   ```
   > CREATE TABLE loyalty_table AS
   (SELECT CAST(c.row_key AS BIGINT),
   CAST(c.personal.name AS VARCHAR(40))AS cust_name,
   CAST(c.loyalty.membership AS VARCHAR(40))
   AS cust_loyalty FROM maprdb.customers c);
   ```

   You will see output similar to the following:

   ```
   0: jdbc:drill:> CREATE TABLE loyalty_table AS
   . . . . . . . > (SELECT CAST(c.row_key AS BIGINT),
   . . . . . . . > CAST(c.personal.name AS VARCHAR(40))AS cust_name,
   . . . . . . . > CAST(c.loyalty.membership AS VARCHAR(40))
   . . . . . . . > AS cust_loyalty FROM maprdb.customers c);
   +-----------+----------------------------+
   | Fragment  | Number of records written  |
   +-----------+----------------------------+
   | 0_0       | 12792                      |
   +-----------+----------------------------+
   1 row selected (0.581 seconds)
   ```

   Caution! If you see 0 under `Number of Records Written`, it means your `CREATE TABLE` query did not work. Make sure you are in the `dfs.views` schema, check your spelling, and try again.

3. Big Office Supply Company decided we no longer need this table, because we can always get the relevant information from a view. To save overhead and data storage, we delete the table:

   ```
   > DROP TABLE loyalty_table;
   ```

4. Create a view to organize customer data by state:

```
> CREATE VIEW state_products AS
(SELECT cust_id, state, prod_id, order_total
FROM hive.orders);
```

You will see output similar to the following:

```
0: jdbc:drill:> CREATE VIEW state_products AS
. . . . . > SELECT cust_id, state, prod_id, order_total
. . . . . > FROM hive.orders;
+-------+-----------------------------------------------------------+
|  ok   |                          summary                          |
+-------+-----------------------------------------------------------+
| true  | View 'state_products' created successfully in 'dfs.views' schema  |
+-------+-----------------------------------------------------------+
1 row selected (0.755 seconds)
```

5. Query the view to find the customers with the highest order total in any state:

```
> SELECT cust_id, state, order_total
FROM state_products ORDER BY order_total DESC;
```

You will see scrolling output similar to the following:

```
| 24013    | ny     | 5           |
| 24109    | va     | 5           |
| 22121    | pa     | 5           |
| 22132    | mn     | 5           |
| 22263    | la     | 5           |
| 22452    | ia     | 5           |
| 22483    | ga     | 5           |
+----------+--------+-------------+
| cust_id  | state  | order_total |
+----------+--------+-------------+
| 22490    | fl     | 5           |
+----------+--------+-------------+
122,000 rows selected (4.107 seconds)
```

6. Find the customer with the highest order total in Texas:

```
> SELECT cust_id, state, order_total
FROM state_products WHERE state = 'tx'
ORDER BY order_total DESC LIMIT 1;
```

You will see output similar to the following:

```
0: jdbc:drill:> SELECT cust_id, state, order_total
. . . . . > FROM state_products WHERE state = 'tx'
. . . . . > ORDER BY order_total DESC LIMIT 1;
+----------+--------+-------------+
| cust_id  | state  | order_total |
+----------+--------+-------------+
| 1428     | tx     | 157         |
+----------+--------+-------------+
1 row selected (1.206 seconds)
```

7. Now we want to see everyone in Texas, sorted by the order total. Rewrite the `SELECT` statement in the previous step to do this. Try to build this query on your own. If you need help, refer to the answer key at the end of this lesson.

Lab 3.2a: Perform Nested Data Functions

Estimated time to complete: 5 minutes

1. Set the default schema to `dfs.clicks` in preparation for our next query:

```
> USE dfs.clicks;
```

2. Using `REPEATED_COUNT()`, find the number of products in each clicks transaction that converted into a purchase, and rank them in descending order:

```
> SELECT t.user_info.cust_id
AS cust_id, t.user_info.device
AS device, REPEATED_COUNT(t.trans_info.prod_id)
AS prod_count, t.trans_info.purch_flag
AS purch_flag FROM `clicks/clicks.json` t
WHERE t.trans_info.purch_flag = 'true'
ORDER BY prod_count DESC;
```

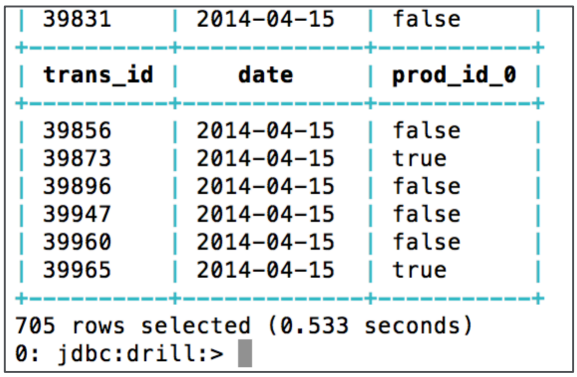
You will see output similar to the following:

```
| 10722    | IOS7     | 0          | true       |
| 14657    | IOS5     | 0          | true       |
| 10629    | IOS5     | 0          | true       |
| 5819     | AOS4.2   | 0          | true       |
| 6179     | IOS7     | 0          | true       |
| 9516     | AOS4.2   | 0          | true       |
| 6897     | IOS5     | 0          | true       |
| 13697    | AOS4.2   | 0          | true       |
| 7451     | AOS4.2   | 0          | true       |
| 11940    | AOS4.2   | 0          | true       |
| 12840    | IOS6     | 0          | true       |
| 8702     | IOS5     | 0          | true       |
| 7837     | IOS5     | 0          | true       |
```

3. Now let's try the `REPEATED_CONTAINS()` function to find whether transactions on April 15, 2014 have 0 in the `prod_id` array:

```
> SELECT trans_id, `date`,
REPEATED_CONTAINS(clk.trans_info.prod_id,0) AS prod_id_0
FROM `clicks/clicks.json` AS clk WHERE `date` = '2014-04-15'
ORDER BY trans_id;
```

You will see output similar to the following:



```
| 39831     | 2014-04-15   | false      |
+-----------+--------------+------------+
| trans_id  |     date     | prod_id_0  |
+-----------+--------------+------------+
| 39856     | 2014-04-15   | false      |
| 39873     | 2014-04-15   | true       |
| 39896     | 2014-04-15   | false      |
| 39947     | 2014-04-15   | false      |
| 39960     | 2014-04-15   | false      |
| 39965     | 2014-04-15   | true       |
+-----------+--------------+------------+
705 rows selected (0.533 seconds)
0: jdbc:drill:>
```

Lab 3.2b: Perform Aggregate Window Functions

Estimated time to complete: 10 minutes

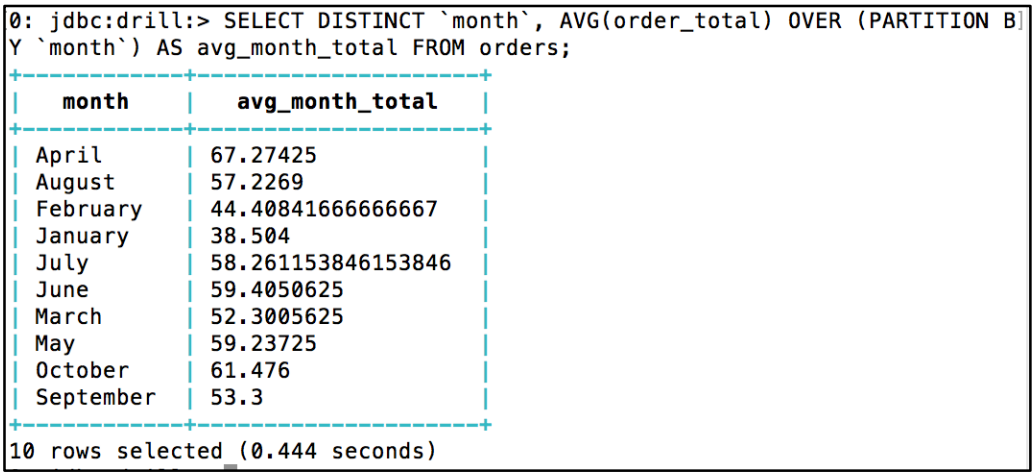
1. Make sure the Hive schema is your default by performing the following query:

```
> USE hive;
```

2. Display the average order total for all orders for each month:

```
> SELECT DISTINCT `month`, AVG(order_total)
OVER (PARTITION BY `month`) AS avg_month_total
FROM orders;
```

You will see output similar to the following:



```
0: jdbc:drill:> SELECT DISTINCT `month`, AVG(order_total) OVER (PARTITION B
Y `month`) AS avg_month_total FROM orders;
+------------+---------------------+
|   month    |   avg_month_total   |
+------------+---------------------+
| April      | 67.27425            |
| August     | 57.2269             |
| February   | 44.40841666666667   |
| January    | 38.504              |
| July       | 58.261153846153846  |
| June       | 59.4050625          |
| March      | 52.3005625          |
| May        | 59.23725            |
| October    | 61.476              |
| September  | 53.3                |
+------------+---------------------+
10 rows selected (0.444 seconds)
```

3. Display the sum of the order amount for each state. Try to write this query on your own.

4. Display the count of the order amount for each state. Try to write this query on your own.

Lab 3.4: Perform an End-to-End Drill Data Analysis

Estimated time to complete: 25 minutes

Let's look at a new scenario. The Big Oil Company is using Hadoop to store and process its data. It has received a cvs file from the field with the results of testing one of its oil wells.

The data analysts in the company have no idea what this data looks like. They do not know how many columns are in the data, or the data type of the columns. Do they have to open the file, build a schema for the data, and perform an ETL transform?

The answer is: no! As long as a storage plugin exists for it, you can open up a file in Drill, and Drill will automatically figure out the format of the data and build a schema for it.

In this exercise, you will perform an end-to-end Drill data analysis. This consists of the following:

1. Create a subdirectory for the lab files
2. Download the lab files into the subdirectory
3. Add the subdirectory to the dfs storage plugin
4. Query the file with SQL in Drill Explorer
5. Create and save a view

Download the lab files

1. Use `!quit` to exit SQLLine.
2. Use `su mapr` to log in to your node as `mapr` with the password `mapr`.
3. After you log in, make sure you are in the `/mapr/demo.mapr.com` directory:

   ```
   $ cd /mapr/demo.mapr.com
   ```

4. Create a subdirectory called `labdata`, and position yourself there:

   ```
   $ mkdir labdata
   $ cd labdata
   ```

5. Enter the following commands to download and unzip the lab files:

   ```
   $ wget http://course-files.mapr.com/DA4000-R1/Lab3.4zip
   $ unzip Lab3.4.zip
   ```

   You should see two new files appear in the `labdata` sub-directory:

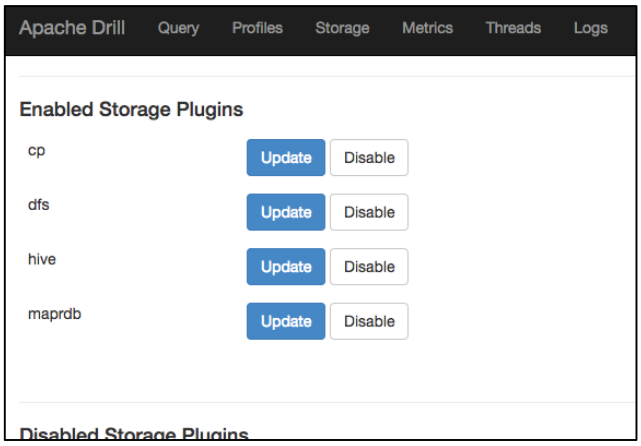
   ```
   $ ls
   BigOil.csv
   zips2000.csv
   ```

Add the `labfiles` Subdirectory to the `dfs` Plugin

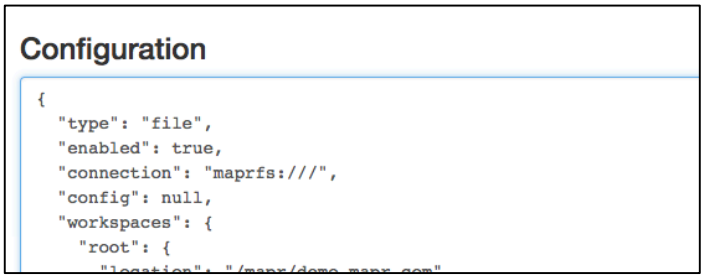
1. Open the Drill Web UI.

Remember: use a browser to access the Drill Web UI at port 8047.
`<IP address>:8047`

2. Click **Storage:**



3. Click **Update** next to **dfs** under **Enabled Storage Plugin**. You will see configuration information:

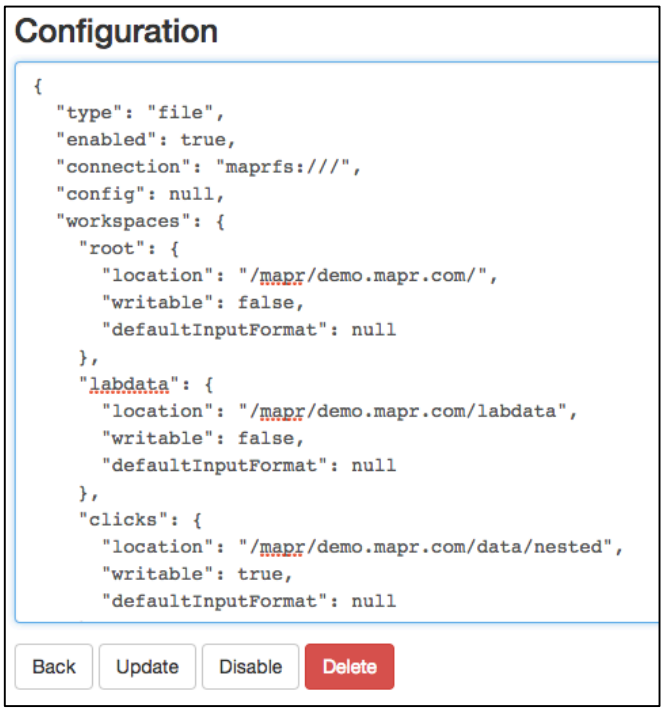


4. We need to create a new workspace for the file we want to open in Drill. Copy the **`clicks`** workspace information and paste it in right above the current **`clicks`** workspace definition.

5. Change the values of the pasted text as follows:

   a. Change the name to `"labdata"`.
   b. Change the location to `"/mapr/demo.mapr.com/labdata"`.
   c. Change the writable variable to `false`.

   When you are completed, your storage plugin should look like this:



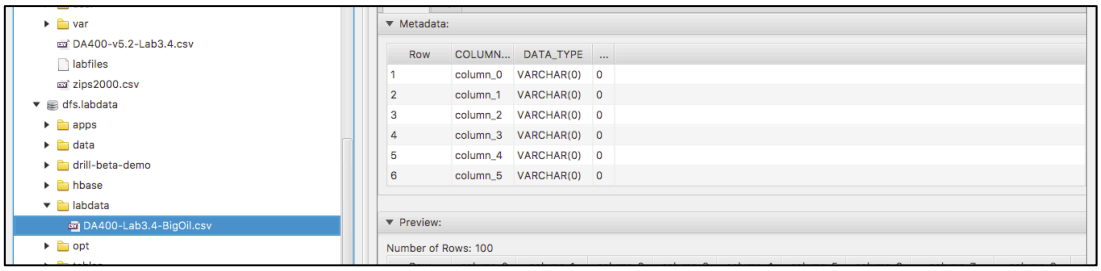
6. Click **Update**.

Question: What does this mean?

Answer: In Drill, we opened a file with unknown data. We did not have to provide Drill with a schema of the data – Drill automatically created a schema when it opened the file. All we had to do is add our `labdata` subdirectory to Drill in the `dfs` storage plugin, to tell Drill where to look for the file.

Query the file in Drill Explorer

1. Open Drill Explorer.
2. Expand the **dfs.labdata** folder in the left-hand pane.
3. Click the file **BigOil.csv**. You will see the contents of the file:



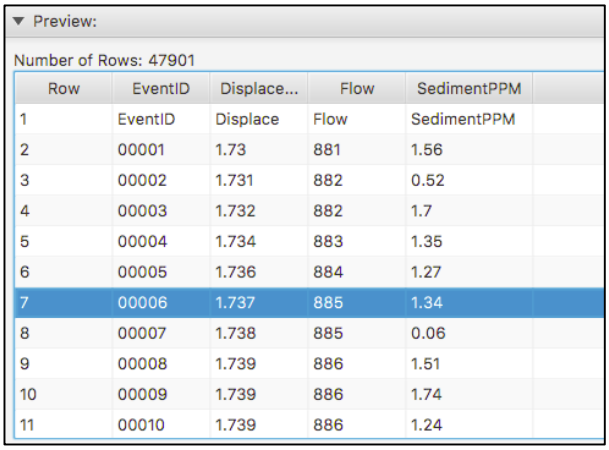
4. Click the **SQL** tab. You will see the SQL used to generate this output.

   We can now see that Drill has built a schema where the columns of the file are all stored in a `columns[]` array. We can use this array to refer to the columns in SQL queries.

5. The Big Oil Company would like to see the Displacement, Flow, and Sediment PPM for each EventID. Perform the following query in Drill Explorer:

```
SELECT columns[0] AS EventID, columns[5] AS Displacement,
columns[6] AS Flow, columns[7] AS SedimentPPM
FROM `dfs.labdata`.`BigOil.csv`
```

6. Click **Preview**. You will see the results of your query:

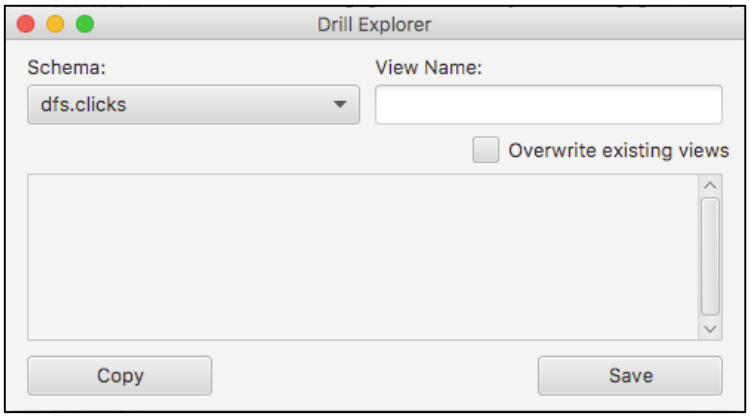


Row	EventID	Displace...	Flow	SedimentPPM
1	EventID	Displace	Flow	SedimentPPM
2	00001	1.73	881	1.56
3	00002	1.731	882	0.52
4	00003	1.732	882	1.7
5	00004	1.734	883	1.35
6	00005	1.736	884	1.27
7	00006	1.737	885	1.34
8	00007	1.738	885	0.06
9	00008	1.739	886	1.51
10	00009	1.739	886	1.74
11	00010	1.739	886	1.24

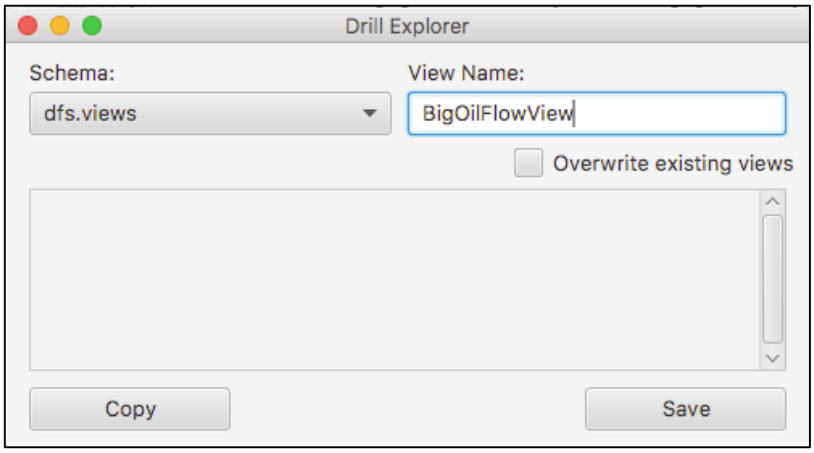
   You can make any other SQL query you wish on the file, as long as it is in the `labdata` subdirectory.

Create and Save a View

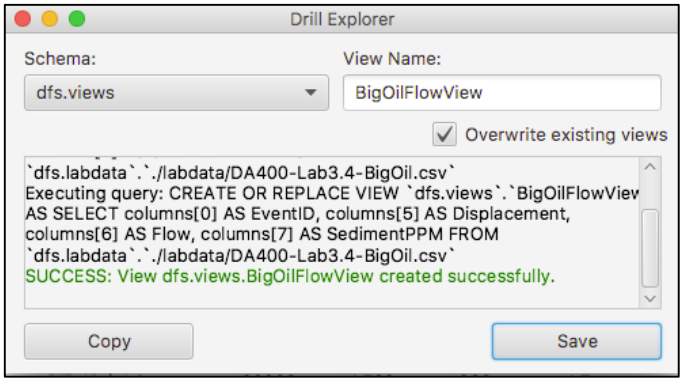
1. With your data from the last SQL query still up, click **Create As…** You will see the following dialog box:



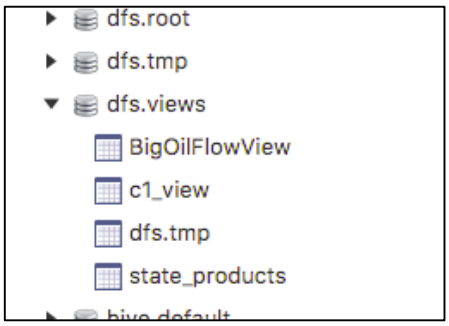
2. For **Schema**, choose **dfs.views**.
3. For **View Name**, enter **BigOilFlowView**:



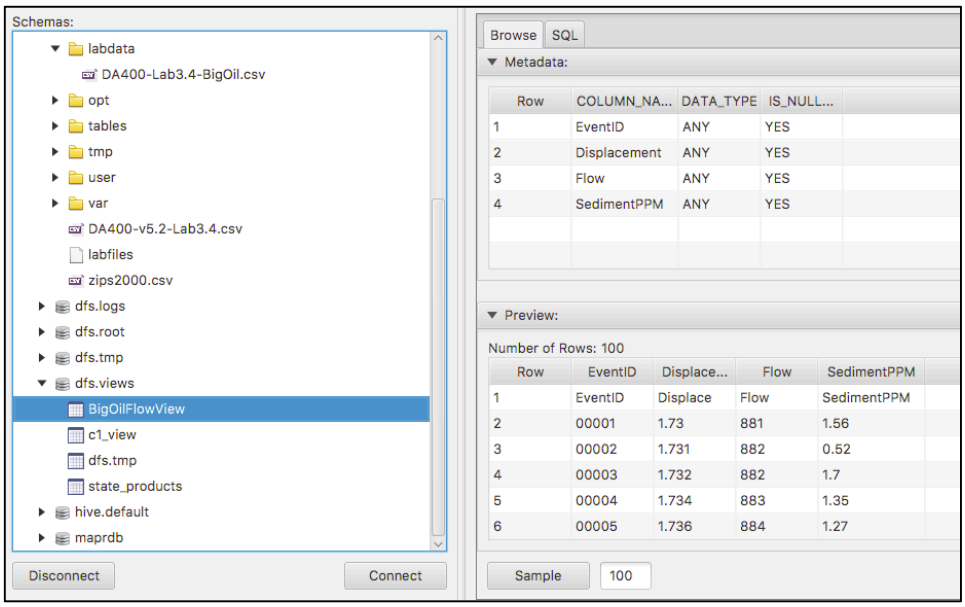
4. Click **Save**. You will see a confirmation message:



5. **Close** the confirmation message window.
6. Click the **Browse** tab.
7. In the left-hand pane, expand **dfs.views**. You should see **BigOilFlowView**:



8. Click **BigOilFlowView**. You will see the first 100 rows of the view:



In addition to creating the view in the Drill Explorer environment, Drill Explorer also saves the view to a file. Let's look at that file.

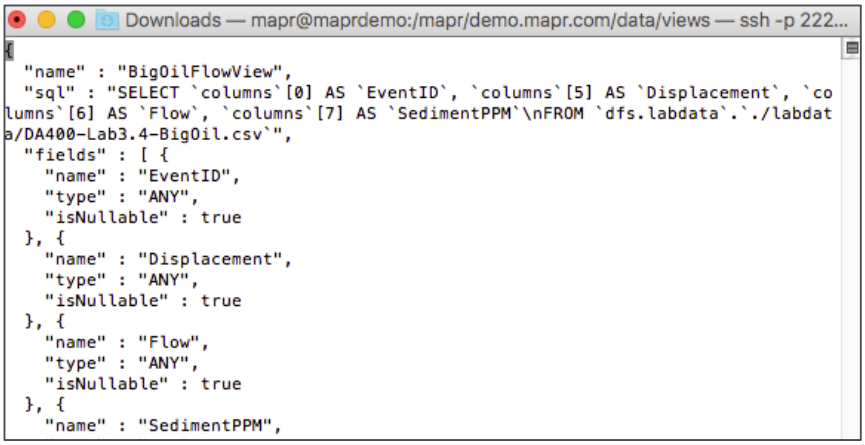
9. Go back to your Drill command-line interface. Do not login to SQLLine. Enter the following command:

```
$ cd /mapr/demo.mapr.com/data/views
```

10. View the file:

```
$ more BigOilFlowView.view.drill
```

You will see the contents of the file:



The data for the view is in JSON format. This is how Drill stores views in a file.

11. The file **`zips2000.csv`**, contains information pertaining to zip codes. Perform the steps in this lab again, with the following changes:

    a. Query for the zip code, latitude, and longitude; sort by zip code.
    b. Name the view you create `ZipCodeView`.

Note: You do not need to create a new storage plugin workspace, because you've already created one for the `labdata` subdirectory. This is where the `zips2000.csv` file is located.

Lessons Learned

- The `CREATE` command can be used to create a table or a view
- The `DROP` command can be used to drop a table or a view
- The `REPEATED_COUNT()` function finds the number of rows that match a certain equality criteria. `REPEATED_CONTAINS()` finds the number of rows that contain a value. Both these functions must be used on an array
- In a Drill window function, you specify the window of rows you want to include in the query by using the `PARTITION BY` clause
- Drill allows you to open files without a schema. The only requirement is that there is an entry to the location of the files in a storage plugin
- When you create a view in Drill Explorer, it automatically saves the view to a file

Lesson 3 Answer Key

Lab 3.1: Create Tables and Views

Step	Instruction	Solution
Step 7	Show all customers in Texas, sorted by order total	`> SELECT cust_id, state, order_total FROM state_products WHERE state = 'tx' ORDER BY order_total DESC;`

Lab 3.2b: Perform Aggregate Window Functions

Step	Instruction	Solution
Step 3	Display the sum of the order amount for each state	`> SELECT state, SUM(order_total) AS sum_order_total FROM hive.orders GROUP BY state ORDER BY state;`
Step 4	Display the count of the order amount for each state	`> SELECT state, COUNT(order_total) AS count_order_total FROM hive.orders GROUP BY state ORDER BY state;`

Lab 3.4: Perform an End-to-End Drill Data Analysis

Step	Instruction	Solution
Step 11a	Query for the zip code, latitude, and longitude; sort by zip code	`SELECT columns[0] AS zip, columns[1] AS latitude, columns[2] AS longitude FROM` `` `dfs.labdata`.`zips2000.csv` ORDER BY zip ``
Step 11b	Name the view `ZipCodeView`	1. Click **Create As** 2. In the **Schema** field, select `dfs.views` 3. In the **View Name** field, specify `ZipCodeView` 4. Click **Save**

DA 401 – Apache Drill Performance and Debugging

Part of the DA 4000 curriculum

Lesson 4: Apache Drill Architecture

Lab 4.1: Order Query Process Steps

Estimated time to complete: 10 minutes

Below are the steps for processing a query in Drill, in incorrect order. Put these steps in the correct order by entering the correct step number in the right-hand column. Refer to the Answer Key at the end of this document for the correct answers.

Details	Step
The Foreman gets a list of available Drillbit nodes from Zookeeper	
The Foreman streams results back to the client	
The client contacts ZooKeeper for a list of available nodes to process a query	
The client sends a query to an available Drillbit, which becomes the Foreman	
The individual Drillbits finish executing the query and return data to the Foreman	
The Foreman sends the query to the list of nodes it obtained from ZooKeeper	

Lab 4.2: Sketch Drillbit Architecture

Estimated time to complete: 10 minutes

1. Below is the diagram of how queries flow through Drillbits, from left to right, without the component names filled in. Enter the correct component name in each box, from the list below.

File System	Execution	HBase/MapR-DB	Hive
Storage Engine Interface	Physical Plan	Logical Plan	Optimizer
RPC Endpoint	SQL Parser	Others	

2. What is the function of the following components?
    a. Storage Engine Interface
    b. Optimizer
    c. Physical Plan

Lessons Learned

- The Drill Zookeeper keeps a list of available nodes to execute a query
- The first Drillbit that a client sends a query to becomes the Foreman for that query
- The optimizer takes the logical plan, divides the query into fragments, and generates a physical plan for query execution
- The physical plan specifies how Drill is going to execute the query among available Drillbit nodes

Lesson 4 Answer Key

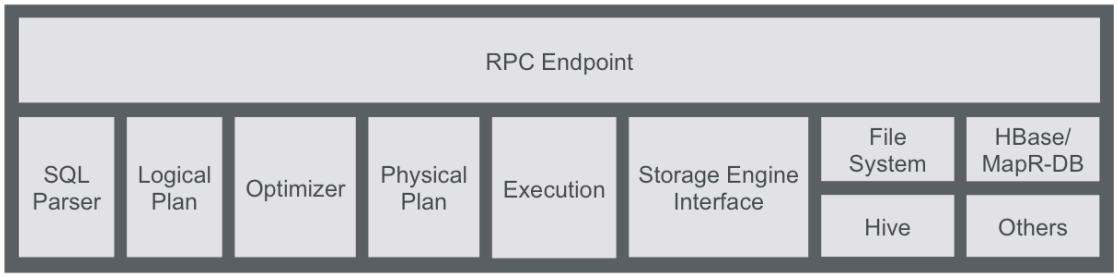
Lab 4.1: Order Query Process Steps

Details	Step
The Foreman gets a list of available Drillbit nodes from Zookeeper	3
The Foreman streams results back to the client	6
The client contacts ZooKeeper for a list of available nodes to process a query	1
The client sends a query to an available Drillbit, which becomes the Foreman	2
The individual Drillbits finish executing the query and return data to the Foreman	5
The Foreman sends the query to the list of nodes it obtained from ZooKeeper	4

1. The client contacts ZooKeeper for a list of available nodes to process a query.
2. The client sends a query to an available Drillbit, which becomes the Foreman.
3. The Foreman gets a list of available Drillbit nodes from ZooKeeper.
4. The Foreman sends the query to the list of nodes it obtained from ZooKeeper.
5. The individual Drillbits finish executing the query and return data to the Foreman.
6. The Foreman streams results back to the client.

Lab 4.2: Sketch Drillbit Architecture

1. Enter the correct component name in each box, form the list that appears below the diagram.



2. What is the function of the following components?

   a. **Storage Engine Interface**: During the execution process, the storage engine interface uses storage plugins to interact with the data sources. Storage plugins provide Drill with the metadata for the data source, the interfaces for Drill to read from and write to data sources, and the location of data. They also provide a set of optimization rules for efficient and faster execution of Drill queries on a specific data source.

   b. **Optimizer:** The optimizer takes the logical plan, divides the query into fragments, and generates a physical plan for query execution.

   c. **Physical Plan:** The physical plan specifies how Drill is going to execute the query among available Drillbit nodes.

Lesson 5: Apache Drill Query Plans and Optimization

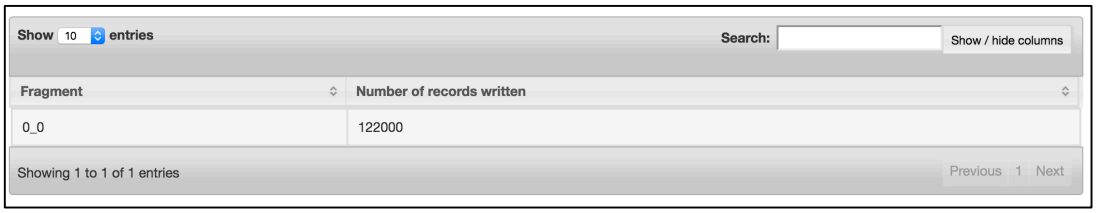
Lab 5.2: Examine Physical Query Plans

Estimated time to complete: 15 minutes

1. Open the Drill Web UI. Click the **Query** tab at the top of the screen and select **SQL**.
2. In the **Query** box, enter:

```
CREATE TABLE dfs.tmp.hiveorders
AS (SELECT * FROM hive.orders);
```

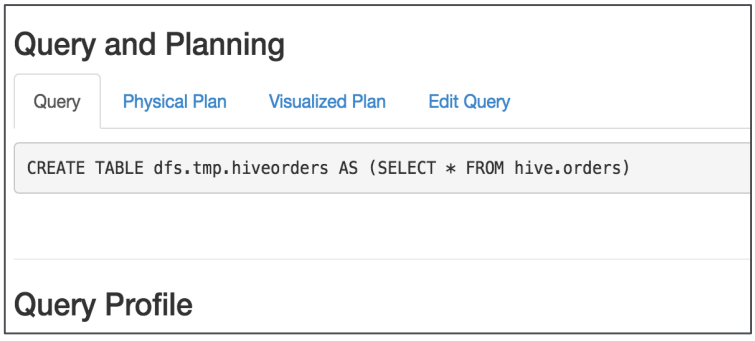
3. Click **Submit**. You will see the output similar to the following:



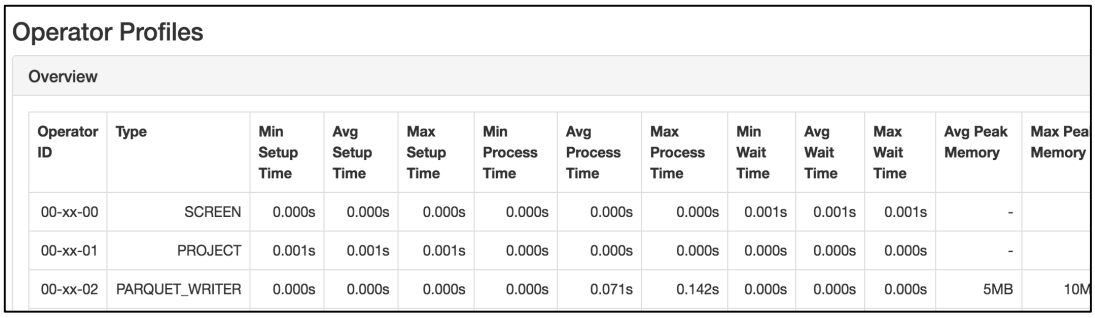
4. Click the **Profiles** tab at the top of your screen. You will see a window with all your executed queries, similar to the following:

Notice that on the right side, Drill shows that the query has been completed. It also shows that the Foreman for this query is named **maprdemo**.

5. Click the first query in the list, the `CREATE TABLE` query from Step 1. You will see the **Query and Planning** window, similar to the following:



6. Scroll down until you can view **Operator Profiles**, similar to the following:



Operator Profiles

Overview

Operator ID	Type	Min Setup Time	Avg Setup Time	Max Setup Time	Min Process Time	Avg Process Time	Max Process Time	Min Wait Time	Avg Wait Time	Max Wait Time	Avg Peak Memory	Max Pea Memory
00-xx-00	SCREEN	0.000s	0.000s	0.000s	0.000s	0.000s	0.000s	0.001s	0.001s	0.001s	-	
00-xx-01	PROJECT	0.001s	0.001s	0.001s	0.000s	0.000s	0.000s	0.000s	0.000s	0.000s	-	
00-xx-02	PARQUET_WRITER	0.000s	0.000s	0.000s	0.000s	0.071s	0.142s	0.000s	0.000s	0.000s	5MB	10M

7. Make a note of the following for **HIVE_SUB_SCAN**:
   - Min Process Time:
   - Avg Process Time:
   - Max Process Time:
   - Min Wait Time:
   - Avg Wait Time:
   - Max Wait Time:
8. Now we'll try another query, and see if there's any difference in performance. Click the **Query** tab at the top of the screen, and select **SQL**.
9. In the box, enter the query:

```
CREATE VIEW dfs.tmp.hiveordersview
AS (SELECT * FROM hive.orders);
```

10. Click the **Profiles** tab at the top of your screen.
11. Click the first query in the list, the `CREATE VIEW` query from Step 9. You will see the **Query and Planning** window. Scroll down until you can view **Operator Profiles**. Note that you don't see all the processes you did for the last query, when we created a table. We just see a process called **DIRECT_SUB_SCAN**. Write down the values for **DIRECT_SUB_SCAN**:
    - Min Process Time:
    - Avg Process Time:
    - Max Process Time:
    - Min Wait Time:
    - Avg Wait Time:
    - Max Wait Time:

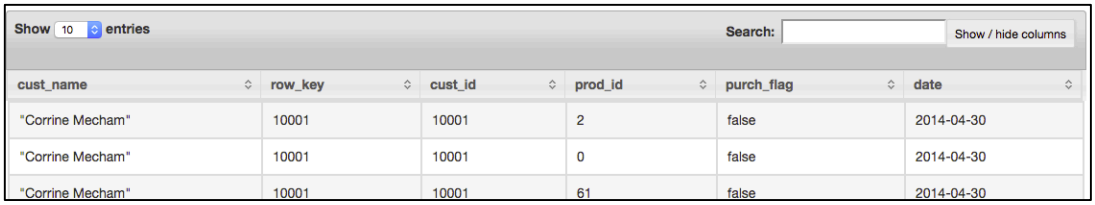
    Is there a difference in these times between the two queries? Why or why not? Consult the Answer Key at the end of this lesson for the answer.

12. Now let's look at the difference between a join query using `WHERE`, and a join query using the keyword `JOIN`. Click **Query** and select **SQL**.

13. Enter this query:

```
SELECT * FROM
(SELECT CAST(c1.personal.name AS VARCHAR(40))
AS cust_name, CAST(c1.row_key AS BIGINT)
AS row_key FROM maprdb.customers AS c1) AS cust,
(SELECT CAST(c.user_info.cust_id AS BIGINT) AS cust_id,
CAST(FLATTEN(c.trans_info.prod_id) AS BIGINT) AS prod_id,
c.trans_info.purch_flag AS purch_flag, c.`date` AS `date`
FROM dfs.clicks.`/clicks/clicks.json` AS c)
AS clk WHERE cust.row_key = clk.cust_id;
```

    You will see the output of the query in a new window, similar to the following:


Show 10 entries Search: Show / hide columns

cust_name	row_key	cust_id	prod_id	purch_flag	date
"Corrine Mecham"	10001	10001	2	false	2014-04-30
"Corrine Mecham"	10001	10001	0	false	2014-04-30
"Corrine Mecham"	10001	10001	61	false	2014-04-30

14. Click the **Profiles** tab at the top of your screen.

15. Click on the first query in the list, the `SELECT` query from Step 13. You will see the **Query and Planning** window. Scroll down until you can view **Operator Profiles**.

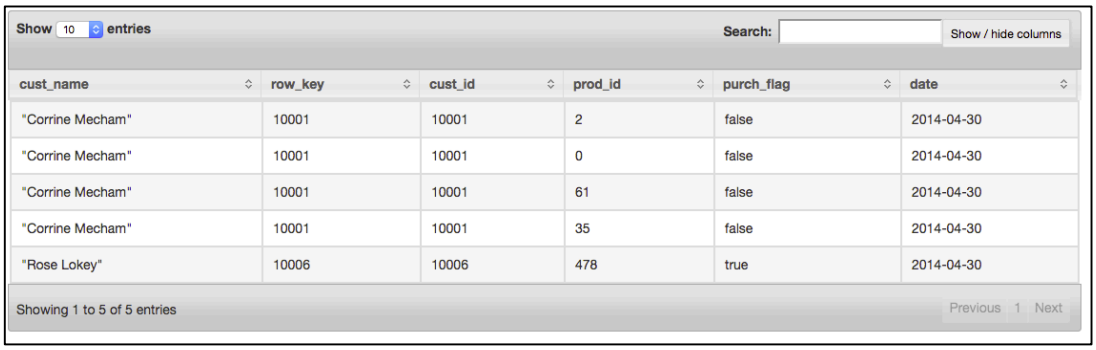
16. Scroll down until you can view **Operator Profiles**. The first entry should be **JSON_SUB_SCAN**. Make a note of the following column values:

    - Min Process Time:
    - Avg Process Time:
    - Max Process Time:
    - Min Wait Time:
    - Avg Wait Time:
    - Max Wait Time:

17. Click **Query** and make sure **SQL** is selected. In the box, enter the query:

```
SELECT * FROM (SELECT CAST(c1.personal.name AS VARCHAR(40)) AS
cust_name, CAST(c1.row_key AS BIGINT)
AS row_key FROM maprdb.customers AS c1) AS cust
JOIN (SELECT CAST(c.user_info.cust_id AS BIGINT)
AS cust_id, CAST(FLATTEN(c.trans_info.prod_id) AS BIGINT)
AS prod_id, c.trans_info.purch_flag AS purch_flag,
c.`date` AS `date` FROM dfs.clicks.`/clicks/clicks.json` AS c) AS
clk ON cust.row_key = clk.cust_id;
```

You will see the output of the query in a new window, similar to the following:



18. Click the **Profiles** tab at the top of your screen.
19. Click the first query in the list, the `SELECT` query. You will see the **Query and Planning** window.
20. Scroll down to the **Operator Profiles** table. The first entry should be **JSON_SUB_SCAN**. Make a note of the following column values:
    - Min Process Time:
    - Avg Process Time:
    - Max Process Time:
    - Min Wait Time:
    - Avg Wait Time:
    - Max Wait Time:

    Is there a difference in these times between the two queries? Why or why not? Consult the Answer Key at the end of this document for the answer.

Lab 5.3: Create a Partitioned Table

Estimated time to complete: 15 minutes

1. From the Drill Web UI, execute the following SQL query to create the `dfs.tmp.ca_month_table`:

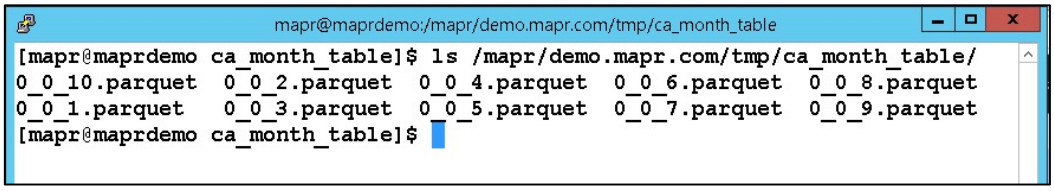
```
CREATE TABLE dfs.tmp.ca_month_table
(ca_month, ca_order_total, state)
PARTITION BY (ca_month) AS SELECT
`month`, SUM(order_total), state
FROM hive.orders WHERE state = 'ca'
GROUP BY `month`, state;
```

2. When the query completes, open Drill Explorer. In the **Browse** tab, expand the **dfs.tmp** folder.
3. Click the `ca_month_table` entry. In the **Data Preview** section, you will see 10 rows of data, for 10 months (there were no orders in our table for November and December). Because the table was partitioned by month, it has 10 partitions.

4. At the terminal window, list the contents of the location where the table was created:

```
$ ls /mapr/demo.mapr.com/tmp/ca_month_table
```

You will see one parquet file for each partition:



Lessons Learned

- You can access information about the physical plan of a query from the **Profiles** tab in the Drill Web UI
- There is no appreciable performance difference between using a `WHERE` clause and using the `JOIN` keyword to join tables in a SQL query
- Creating a partitioned table can help speed up searches if the data is to be stored in specific categories, such as month of the year

Lesson 5 Answer Key

Lab 5.2: Examine Physical Query Plans

Although the exact numbers will vary according to the machine you run the queries on, in general, the numbers for the `CREATE VIEW` process and wait times should be lower than the `CREATE TABLE` query. This is because the `CREATE TABLE` query actually creates a table on disk, and the disk read and write operations add time to the query processing. A view only creates metadata, not table data, so the disk operations are far fewer for the `CREATE VIEW` query.

In the case of `WHERE` and `JOIN`, there is no real difference between the performance numbers. That's because there is no performance difference between using these two options to join tables. You can use whichever SQL option you are comfortable with to join tables.

Lesson 6: Apache Drill Logging and Debugging

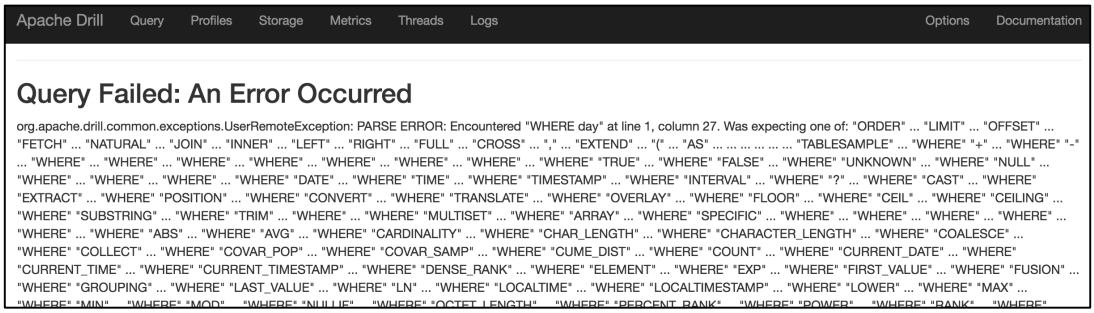
Lab 6.1: Examine Drill Log Files

Estimated time to complete: 15 minutes

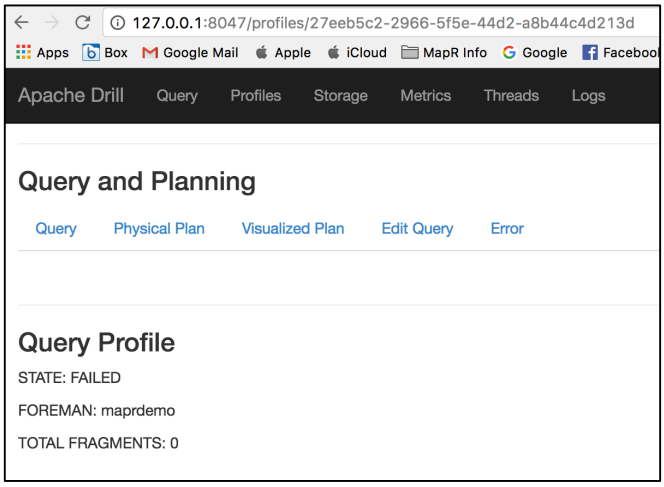
1. Log in to the Drill Web UI. Click **Query** at the top of the screen, and make sure **SQL** is selected.
2. Enter the following in the Query window, then click **Submit**:

   ```
   SELECT * FROM hive.orders WHERE day = 'Monday';
   ```

   You will see an error message:

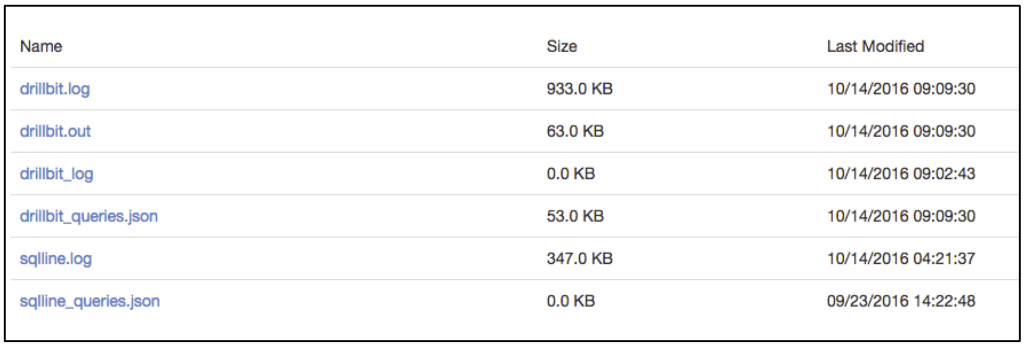
   

   To diagnose the error, first let's look at the profile for this query.
3. Click **Profiles**. You will see up to the last100 queries that have been executed.
4. Click the text of your failed query. You will see the query's profile page:

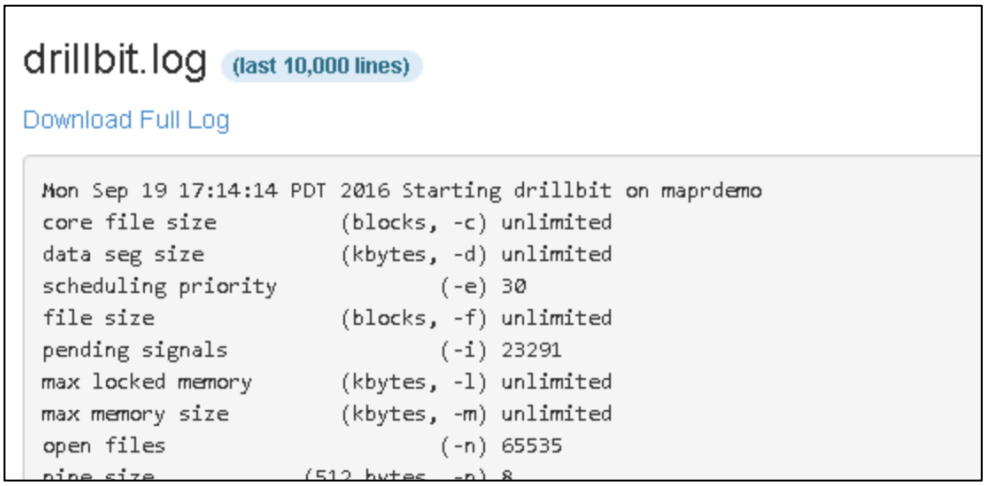
5. Write down the sequence of numbers and characters that form the last part of the URL for this page. In this example, this would be `27eeb5c2-2966-5f5e-44d2-a8b44c4d213d`.

6. Click the **Logs** tab at the top of the screen. You will see the **Logs** page, similar to the following:



Name	Size	Last Modified
drillbit.log	933.0 KB	10/14/2016 09:09:30
drillbit.out	63.0 KB	10/14/2016 09:09:30
drillbit_log	0.0 KB	10/14/2016 09:02:43
drillbit_queries.json	53.0 KB	10/14/2016 09:09:30
sqlline.log	347.0 KB	10/14/2016 04:21:37
sqlline_queries.json	0.0 KB	09/23/2016 14:22:48

7. Let's look at the `drillbit.log` file. Click on **drillbit.log**. You'll see the contents of the file, similar to the following:



8. Click **Download Full Log**. The entire log file will be downloaded to your computer.
9. Open the `drillbit.log` file with a text editor.
10. Search `drillbit.log` for the query ID you wrote down in Step 5.

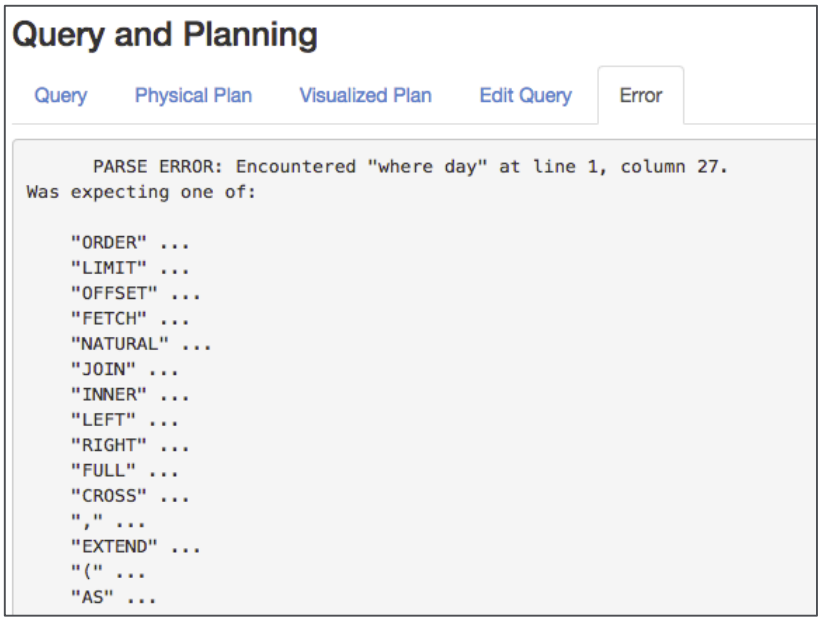
    When you find the query, scroll until you find the SQL error message. What is the problem with your query? How would you fix it?

11. Now we'll look at an easier way to find errors. Click **Profiles** at the top of the Drill WebUI.

Note: If you don't find your query on the **Profiles** tab, it will be stored in the `drillbit_queries.json` file. This file stores information about whether the query succeeded or failed, but not the cause of the failure. To determine the cause, you need to look at `drillbit.log`.

12. Click the text of the failed query you executed.

13. Click the **Error** tab. You will see the Error page, similar to the following:



At the top of this page, we see the error in the query. The parser is having trouble with "`where day`".

Lessons Learned

- The Drill log files contain useful information, but they are not always the best way to find the cause of errors in SQL queries. To find the cause of errors, the fastest and easiest way is often to look at the **Error** page from the **Profiles** page.
- The Drill log files you can check for Drill errors and behavior. These include:
  - drillbit.log
  - drillbit_queries.json
- You can view the contents of the Drill log files through the **Logs** tab in the Drill WebUI
- The Drill log files are stored in the directory `<drill-installation-dir>/conf`. You can change this by changing an option in the Drill configuration file `logback.xml`.