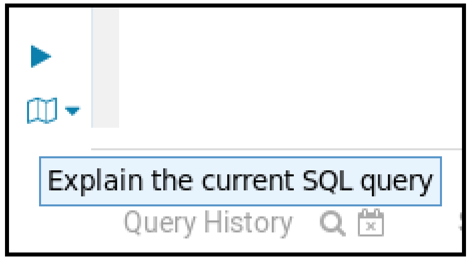
Understanding Execution Plans

To get a true grasp on what causes a query to take a long time, you need to understand what operations Hive or Impala will perform when it executes a query. To find this out, you can view the *execution plan* for a query. The execution plan is a description of the tasks required for a query, the order in which they'll be executed, and some details about each task.

To see an execution plan for a query, you can do either of these:

- Prefix the query with the keyword EXPLAIN, then run it.
- In Hue, simply click the Explain button, which has an icon of a folding map:



Execution plans can be long and complex. Fully understanding them requires a deep knowledge of MapReduce, which is beyond the scope of this course. However, the basics covered here provide a useful introduction that can help you identify trouble areas in your queries' execution plans.

The execution plans provided by Hive and by Impala look slightly different, but at a basic level, they provide more or less the same information. (Hive's execution plans provide much more detail, and understanding it all is beyond the scope of this course.)

The next several sections show these parts of the query plan for this example query:

```
CREATE TABLE flights_by_carrier AS
```

```
SELECT carrier, COUNT(flight) AS num

    FROM flights GROUP BY carrier;
```

This query is a CTAS statement that creates a new table named flights_by_carrier and populates it with the result of a SELECT query. The SELECT query groups the rows of the flights table by carrier and returns each carrier and the number of flights for that carrier.

Hive Execution

Hive's output of the EXPLAIN statement for the example is shown here, with some details removed:

```
+-----------------------------------------------------+--+
|                      Explain                        |
+-----------------------------------------------------+--+
| STAGE DEPENDENCIES:                                 |
|   Stage-1 is a root stage                           |
|   Stage-0 depends on stages: Stage-1                |
|   Stage-3 depends on stages: Stage-0                |
|   Stage-2 depends on stages: Stage-3                |
|                                                     |
| STAGE PLANS:                                        |
|   Stage: Stage-1                                    |
|     Map Reduce                                      |
|       Map Operator Tree:                            |
|           TableScan                                 |
|             alias: flights                          |
|             …                                       |
|             Select Operator                         |
|               …                                     |
|               outputColumnNames: carrier, flight    |
|               …                                     |
|               Group By Operator                     |
|                 aggregations: count(flight)         |
|                 keys: carrier (type: string)        |
|                 …                                   |
|       Reduce Operator Tree:                         |
|         Group By Operator                           |
|           aggregations: count(VALUE._col0)          |
|           keys: KEY._col0 (type: string)            |
|           …                                         |
|           outputColumnNames: _col0, _col1           |
|           …                                         |
|                                                     |
|   Stage: Stage-0                                    |
|     Move Operator                                   |
|       files:                                        |
|           hdfs directory: true                      |
|           destination: hdfs://…                     |
|                                                     |
|   Stage: Stage-3                                    |
```

Stage Dependencies

The example query will execute in four stages, Stage-0 to Stage-3. Each stage could be a MapReduce job, an HDFS action, a metastore action, or some other action performed by the Hive server.

The numbering does not imply an order of execution or dependency. The dependencies between stages determine the order in which they must execute, and Hive specifies these dependencies explicitly at the start of the EXPLAIN results.

A root stage, like Stage-1 in this example, has no dependencies and is free to run first. Non-root stages cannot run until the stages upon which they depend have completed.

Stage Plans

The stage plans part of the output shows descriptions of the stages. For Hive, read them by starting at the top and then going down.

Stage-1 is identified as a MapReduce job. The query plan shows that this job includes both a map phase (described by the Map Operator Tree) and a reduce phase (described by the Reduce Operator Tree). In the map phase, the map tasks read the flights table and select the carrier and flights columns. This data is passed to the reduce phase, in which the reduce tasks group the data by carrier and aggregate it by counting flights.

Following Stage-1 is Stage-0, which is an HDFS action (Move). In this stage, Hive moves the output of the previous stage to a new subdirectory in the warehouse directory in HDFS. This is the storage directory for the new table that will be named flights_by_carrier. (The actual HDFS path is too long to show here.)

Following Stage-0 is Stage-3, which is a metastore action: Create Table. In this stage, Hive creates a new table named flights_by_carrier in the fly database. The table has two columns: a STRING column named carrier and a BIGINT column named num.

The final stage, Stage-2, collects statistics. The details of this final stage are not important, but it gathers information such as the number of rows in the table, the number of files that store the table data in HDFS, and the number of unique values in each column in the table. These statistics can be used to optimize Hive queries, but further discussion of that is beyond the scope of this course.

Impala Execution

Impala's output of the EXPLAIN statement for the example is shown here:

```
+----------------------------------------------------------+
| Explain String                                           |
+----------------------------------------------------------+
| Max Per-Host Resource Reservation: Memory=3.94MB         |
| Per-Host Resource Estimates: Memory=108.00MB             |
|                                                          |
| WRITE TO HDFS [fly.flights_by_carrier, OVERWRITE=false]  |
| |  partitions=1                                          |
| |                                                        |
| 03:AGGREGATE [FINALIZE]                                  |
| |  output: count:merge(flight)                           |
| |  group by: carrier                                     |
| |                                                        |
| 02:EXCHANGE [HASH(carrier)]                              |
| |                                                        |
| 01:AGGREGATE [STREAMING]                                 |
| |  output: count(flight)                                 |
| |  group by: carrier                                     |
| |                                                        |
| 00:SCAN HDFS [fly.flights]                               |
|    partitions=1/1 files=6 size=917.61MB                  |
+----------------------------------------------------------+
```

For Impala's results, the stages are executed from the bottom up (so the first stage is 00 and the final stage is WRITE TO HDFS. There are four stages labeled 00 to 03, and a final unnumbered stage. The numbering normally does not imply execution order, although in this case, the stages will execute in order from 00 to 03. (Although this example does not show it, additional root stages can be shown within one of the other stages that depends on it, by indenting the line for the root stage.)

The stages are a bit different for Impala; it does not use the map-reduce phases that Hive does with the MapReduce engine. Impala's output is also a bit easier to read.

In Stage 00, Impala reads in the data for the flights table in the fly database.

In Stage 01, each daemon working on this task groups its data by carrier and counts the flight column. At this stage, it's probable that a particular carrier's data will be distributed across daemons.

Stage 02 is similar to the shuffle and sort in a MapReduce job. Data is exchanged, using carrier to determine which data goes to which daemon.

In Stage 03, the daemons again group the data by carrier and merge the individual counts for each carrier.

The final stage writes the results to the new table, flights_by_carrier in the fly database.

Try It!

1. Try running these EXPLAIN statements on the example query, in both Hive and Impala. (You do not need to run the query itself.)
2. See what you can understand from a few other examples. You can try different examples, from something simple (like SELECT * FROM fun.games;) to something a bit more complicated, like

```
SELECT COUNT(f.flight) FROM flights f
```

```
JOIN planes p ON (f.tailnum = p.tailnum)

WHERE p.year < 1968;
```