Spark DataFrame SQL

Jim Harner

9/30/2020

Load `sparklyr` and establish the Spark connection.

```
library(dplyr, warn.conflicts = FALSE)
library(sparklyr)

# start the spark session
master <- "local"
# master <- "spark://master:7077"
sc <- spark_connect(master, spark_home = Sys.getenv("SPARK_HOME"),
                    method = c("shell"), app_name = "sparklyr")
Sys.getenv("SPARK_HOME")
```

```
## [1] "/opt/spark"
```

Use bash to determine the Spark version:

```
# echo $SPARK_HOME
# The following command displays the Spark version installed:
spark-submit --version
```

```
## SLF4J: Class path contains multiple SLF4J bindings.
## SLF4J: Found binding in [jar:file:/opt/spark/jars/slf4j-log4j12-1.7.30.jar!/org/slf4j/impl/StaticLog
## SLF4J: Found binding in [jar:file:/opt/hadoop-2.10.0/share/hadoop/common/lib/slf4j-log4j12-1.7.25.ja
## SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
## SLF4J: Actual binding is of type [org.slf4j.impl.Log4jLoggerFactory]
## Welcome to
##       ____              __
##      / __/__  ___ _____/ /__
##     _\ \/ _ \/ _ `/ __/  '_/
##    /___/ .__/\_,_/_/ /_/\_\   version 3.0.0
##       /_/
##
## Using Scala version 2.12.10, OpenJDK 64-Bit Server VM, 1.8.0_265
## Branch HEAD
## Compiled by user ubuntu on 2020-06-06T11:32:25Z
## Revision 3fdfce3120f307147244e5eaf46d61419a723d50
## Url https://gitbox.apache.org/repos/asf/spark.git
## Type --help for more information.
```

5.3 Spark DataFrame SQL

`sparklyr` can import a wide range of data directly into Spark from an external data source, e.g., json. In addition, it is possible to query Spark DataFrames directly.

We will be using the `nycflights13` data again. The `flights` and `airlines` R data frames are copied into Spark.

```
library(nycflights13)
flights_sdf <- copy_to(sc, flights, "flights",  overwrite = TRUE)
airlines_sdf <- copy_to(sc, airlines, "airlines", overwrite = TRUE)
```

5.3.1 Joining Spark Data Tables

In Section 5.2.1 the `dplyr` verbs were used to manipulate a Spark DataFrame. However, we often have multiple related Spark SQL tables which we need to combine prior to performing data manipulations.

A workflow was developed in Section 5.2.1 to find the flights with a departure delay greater than 1000 minutes. However, we did not have the carrier names since they were in a different table. Providing this information can be done with a `left_join`.

```
flights_sdf %>%
  left_join(airlines_sdf, by = "carrier") %>%
  select(carrier, name, flight, year:day, arr_delay, dep_delay) %>%
  filter(dep_delay > 1000) %>%
  arrange(desc(dep_delay))
```

```
## # Source:     spark<?> [?? x 8]
## # Ordered by: desc(dep_delay)
##   carrier name                   flight  year month   day arr_delay dep_delay
##   <chr>   <chr>                   <int> <int> <int> <int>     <dbl>     <dbl>
## 1 HA      Hawaiian Airlines Inc.     51  2013     1     9      1272      1301
## 2 MQ      Envoy Air                3535  2013     6    15      1127      1137
## 3 MQ      Envoy Air                3695  2013     1    10      1109      1126
## 4 AA      American Airlines Inc.    177  2013     9    20      1007      1014
## 5 MQ      Envoy Air                3075  2013     7    22       989      1005
```

Notice that three of the top five largest delays were associated with Envoy Air, which was not obvious based on the two-letter abbreviation.

`dplyr` has various verbs that combine two tables. If this is not adequate, then the joins, or other operations, must be done in the database prior to importing the data into Spark

5.3.2 Querying a Spark DataFrame

It is also possible to use Spark DataFrames as tables in a "database" using the Spark SQL interface, which forms the basis of Spark DataFrames.

The `spark_connect` object implements a DBI interface for Spark, which allows you to use `dbGetQuery` to execute SQL commands. The returned result is an R data frame.

We now show that the above workflow can be done in R except that R data frames are used.

```
library(DBI)
flights_df <- dbGetQuery(sc, "SELECT * FROM flights")
airlines_df <- dbGetQuery(sc, "SELECT * FROM airlines")
flights_df %>%
  left_join(airlines_df, by = "carrier") %>%
  select(carrier, name, flight, year:day, arr_delay, dep_delay) %>%
  filter(dep_delay > 1000) %>%
  arrange(desc(dep_delay))
```

```
##   carrier                   name flight year month day arr_delay dep_delay
## 1      HA Hawaiian Airlines Inc.     51 2013     1   9      1272      1301
```

```
## 2      MQ              Envoy Air   3535 2013     6  15      1127      1137
## 3      MQ              Envoy Air   3695 2013     1  10      1109      1126
## 4      AA American Airlines Inc.    177 2013     9  20      1007      1014
## 5      MQ              Envoy Air   3075 2013     7  22       989      1005
```

Of course, this assumes the Spark DataFrames can be imported into R, i.e., they must fit into local memory.

The `by` argument in the `left_join` is not needed if there is a single variable common to both tables. Alternately, we could use `by = c("carrier", "carrier")`, where the names could be different if they represent the same variable.

5.3.3 Sampling

We can sample random rows of a Spark DataFrame using:

- `sample_n` for a fixed number;
- `sample_frac` for a fixed fraction.

```
sample_n(flights_sdf, 10)
```

```
## # Source: spark<?> [?? x 19]
##     year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
##    <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>
##  1  2013     9    15      821            825        -4     1010           1029
##  2  2013     4    18     1802           1800         2     2015           2035
##  3  2013     4    24     1226           1235        -9     1526           1520
##  4  2013    12    16     1238           1155        43     1437           1355
##  5  2013     1    11     1646           1645         1     1952           2005
##  6  2013     3    17     1952           1929        23     2324           2247
##  7  2013    11    17      744            750        -6     1030           1040
##  8  2013     5    11      743            710        33      946            925
##  9  2013    12     3      636            645        -9      851            857
## 10  2013     5    29     1333           1327         6     1548           1545
## # ... with 11 more variables: arr_delay <dbl>, carrier <chr>, flight <int>,
## #   tailnum <chr>, origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>,
## #   hour <dbl>, minute <dbl>, time_hour <dttm>
```

```
sample_frac(flights_sdf, 0.01)
```

```
## # Source: spark<?> [?? x 19]
##     year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
##    <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>
##  1  2013     7     8     1738           1725        13     2104           2045
##  2  2013     4     4     1713           1715        -2     1954           2037
##  3  2013     9    27     2250           2255        -5     2344           2358
##  4  2013     3    24     1858           1900        -2     2025           2016
##  5  2013    10    22      814            815        -1     1042           1055
##  6  2013     6    11      833            815        18     1039           1033
##  7  2013     5    12     2010           1931        39     2309           2305
##  8  2013     5    29     1926           1859        27     2122           2134
##  9  2013    10    19      702            710        -8     1024           1000
## 10  2013    12    21     2113           2000        73     2358           2254
## # ... with more rows, and 11 more variables: arr_delay <dbl>, carrier <chr>,
## #   flight <int>, tailnum <chr>, origin <chr>, dest <chr>, air_time <dbl>,
## #   distance <dbl>, hour <dbl>, minute <dbl>, time_hour <dttm>
```

Sampling is often done during the development and testing cycle to limit the size of the data.

5.3.4 Writing Data to HDFS

We can save the results of our analysis or the tables that you have generated in Spark into HDFS persistent storage. Parquet is a commonly used persistent store for various data processing systems in the Hadoop ecosystem. It has a columnar storage format which Spark SQL supports for both reading and writing, including the schema of the original data.

As an example, we can write the `airlines_sdf` Spark DataFrame out to a Parquet file using the `spark_write_parquet` function.

```
library(rhdfs)
```

```
## Loading required package: rJava
```

```
##
## HADOOP_CMD=/opt/hadoop/bin/hadoop
```

```
##
## Be sure to run hdfs.init()
```

```
hdfs.init()
spark_write_parquet(airlines_sdf,
                    path = "hdfs://hadoop:9000/user/rstudio/airlines_parquet",
                    mode = "overwrite")
hdfs.ls("/user/rstudio")
```

```
##   permission   owner   group size          modtime
## 1 drwxr-xr-x rstudio rstudio    0 2020-09-30 23:54
##                              file
## 1 /user/rstudio/airlines_parquet
```

This writes the Spark DataFrame to the given HDFS path and names the Parquet file `airlines_parquet`.

You can use the `spark_read_parquet` function to read the same table back into a subsequent Spark session:

```
spark_read_parquet(sc, "airlines2_sdf",
                   "hdfs://hadoop:9000/user/rstudio/airlines_parquet")
```

```
## # Source: spark<airlines2_sdf> [?? x 2]
##    carrier name
##    <chr>   <chr>
##  1 9E      Endeavor Air Inc.
##  2 AA      American Airlines Inc.
##  3 AS      Alaska Airlines Inc.
##  4 B6      JetBlue Airways
##  5 DL      Delta Air Lines Inc.
##  6 EV      ExpressJet Airlines Inc.
##  7 F9      Frontier Airlines Inc.
##  8 FL      AirTran Airways Corporation
##  9 HA      Hawaiian Airlines Inc.
## 10 MQ      Envoy Air
## # ... with more rows
```

Note that `airlines2_sdf` is a Spark DataFrame. Use the `spark_write_csv` and `spark_write_json` functions among others to write data to HDFS as `csv` or `json` files, respectively.

5.3.5 Hive Functions

Many of Hive's built-in functions (UDF) and built-in aggregate functions (UDAF) can be called by `dplyr`'s `mutate` and `summarize` funcitons.

`datediff` and `current_date` are Hive UDFs to figure the difference between the `flight_date` and the current system date:

```
flights_sdf %>%
  mutate(flight_date = paste(year, month, day, sep="-"),
         days_since = datediff(current_date(), flight_date)) %>%
  group_by(flight_date, days_since) %>%
  tally() %>%
  arrange(days_since)
```

```
## # Source:     spark<?> [?? x 3]
## # Groups:     flight_date
## # Ordered by: days_since
##    flight_date days_since     n
##    <chr>            <int> <dbl>
##  1 2013-12-31        2465   776
##  2 2013-12-30        2466   968
##  3 2013-12-29        2467   888
##  4 2013-12-28        2468   814
##  5 2013-12-27        2469   963
##  6 2013-12-26        2470   936
##  7 2013-12-25        2471   719
##  8 2013-12-24        2472   761
##  9 2013-12-23        2473   985
## 10 2013-12-22        2474   895
## # ... with more rows
```

```
spark_disconnect(sc)
```