CH 22

```
[1]  import os
     # Find the latest version of spark 3.x  from http://www.apache.org/dist/spark
     # For example:
     # spark_version = 'spark-3.5.1'
     spark_version = 'spark-3.5.1'
     os.environ['SPARK_VERSION']=spark_version

     # Install Spark and Java
     !apt-get update
     !apt-get install openjdk-11-jdk-headless -qq > /dev/null
     !wget -q http://www.apache.org/dist/spark/$SPARK_VERSION/$SPARK_VERSION-bin-h
     !tar xf $SPARK_VERSION-bin-hadoop3.tgz
     !pip install -q findspark

     # Set Environment Variables
     os.environ["JAVA_HOME"] = "/usr/lib/jvm/java-11-openjdk-amd64"
     os.environ["SPARK_HOME"] = f"/content/{spark_version}-bin-hadoop3"

     # Start a SparkSession
     import findspark
     findspark.init()
```

```
[2]  # Import packages
     from pyspark.sql import SparkSession
     import time

     # Create a SparkSession
     spark = SparkSession.builder.appName("SparkSQL").getOrCreate()
```

```
[5]  # 1. Read in the AWS S3 bucket into a DataFrame.
     from pyspark import SparkFiles
     url = "https://2u-data-curriculum-team.s3.amazonaws.com/dataviz-classroom/v1.

     spark.sparkContext.addFile(url)
     df = spark.read.csv(SparkFiles.get("home_sales_revised.csv"), sep=",", header

     # Show the home sales data.
     df.show()
```

```
+--------------------+----------+----------+------+--------+---------+-------
|                  id|      date|date_built| price|bedrooms|bathrooms|sqft_li
+--------------------+----------+----------+------+--------+---------+-------
|f8a53099-ba1c-47d...|2022-04-08|      2016|936923|       4|        3|
|7530a2d8-1ae3-451...|2021-06-13|      2013|379628|       2|        2|
|43de979c-0bf0-4c9...|2019-04-12|      2014|417866|       2|        2|
|b672c137-b88c-48b...|2019-10-16|      2016|239895|       2|        2|
|e0726d4d-d595-407...|2022-01-08|      2017|424418|       3|        2|
|5aa00529-0533-46b...|2019-01-30|      2017|218712|       2|        3|
|131492a1-72e2-4a8...|2020-02-08|      2017|419199|       2|        3|
|8d54a71b-c520-44e...|2019-07-21|      2010|323956|       2|        3|
|e81aacfe-17fe-46b...|2020-06-16|      2016|181925|       3|        3|
|2ed8d509-7372-46d...|2021-08-06|      2015|258710|       3|        3|
|f876d86f-3c9f-42b...|2019-02-27|      2011|167864|       3|        3|
|0a2bd445-8508-4d8...|2021-12-30|      2014|337527|       2|        3|
|941bad30-eb49-4a7...|2020-05-09|      2015|229896|       3|        3|
|dd61eb34-6589-4c0...|2021-07-25|      2016|210247|       3|        2|
|f1e4cef7-d151-439...|2019-02-01|      2011|398667|       2|        3|
|ea620c7b-c2f7-4c6...|2021-05-31|      2011|437958|       3|        3|
|f233cb41-6f33-4b0...|2021-07-18|      2016|437375|       4|        3|
|c797ca12-52cd-4b1...|2019-06-08|      2015|288650|       2|        3|
|0cfe57f3-28c2-472...|2019-10-04|      2015|308313|       3|        3|
|4566cd2a-ac6e-435...|2019-07-15|      2016|177541|       3|        3|
+--------------------+----------+----------+------+--------+---------+-------
only showing top 20 rows
```

```
     # 2. Create a temporary view of the DataFrame.
     df.createOrReplaceTempView('sold_homes')
```

[7]
```
# 3. What is the average price for a four bedroom house sold per year, rounde
query = """
SELECT
  extract(year from date) as year,
  round(avg(price), 2) as avg_price
FROM
  sold_homes
WHERE
  bedrooms = 4
GROUP BY
   extract(year from date)
order by
  avg(price) desc;
"""

spark.sql(query).show()
```

```
+----+---------+
|year|avg_price|
+----+---------+
|2021|301819.44|
|2019| 300263.7|
|2020|298353.78|
```

✓ 1s

[8]
```
# 4. What is the average price of a home for each year the home was built,
# that have 3 bedrooms and 3 bathrooms, rounded to two decimal places?

query = """
SELECT
  date_built,
  round(avg(price), 2) as avg_price
FROM
  sold_homes
WHERE
  bedrooms = 3
  and bathrooms = 3
GROUP BY
   date_built
order by
  date_built desc;
"""

spark.sql(query).show()
```

```
+----------+---------+
|date_built|avg_price|
+----------+---------+
|      2017|292676.79|
|      2016|290555.07|
|      2015| 288770.3|
|      2014|290852.27|
|      2013|295962.27|
```

```
# 5. What is the average price of a home for each year the home was built,
# that have 3 bedrooms, 3 bathrooms, with two floors,
# and are greater than or equal to 2,000 square feet, rounded to two decimal
query = """
SELECT
  date_built,
  round(avg(price), 2) as avg_price
FROM
  sold_homes
WHERE
  bedrooms = 3
  and bathrooms = 3
  and floors = 2
  and sqft_living >= 2000
GROUP BY
   date_built
order by
  date_built desc;
"""

spark.sql(query).show()
```

```
+----------+---------+
|date_built|avg_price|
+----------+---------+
|      2017|280317.58|
|      2016| 293965.1|
|      2015|297609.97|
|      2014|298264.72|
|      2013|303676.79|
```

```
# 6. What is the average price of a home per "view" rating, rounded to two de
# having an average home price greater than or equal to $350,000? Order by de
# Although this is a small dataset, determine the run time for this query.

start_time = time.time()

query = """
SELECT
 view,
 round(avg(price), 2) as avg_price
FROM
  sold_homes
GROUP BY
   view
HAVING
  avg(price) >= 350000
order by
  view desc;
"""

spark.sql(query).show()



print("--- %s seconds ---" % (time.time() - start_time))
```

```
+----+----------+
|view| avg_price|
+----+----------+
|  99|1061201.42|
|  98|1053739.33|
|  97|1129040.15|
|  96|1017815.92|
|  95| 1054325.6|
|  94| 1033536.2|
|  93|1026006.06|
|  92| 970402.55|
|  91|1137372.73|
|  90|1062654.16|
|  89|1107839.15|
|  88|1031719.35|
|  87| 1072285.2|
|  86|1070444.25|
|  85|1056336.74|
|  84|1117233.13|
|  83|1033965.93|
|  82| 1063498.0|
|  81|1053472.79|
|  80| 991767.38|
+----+----------+
only showing top 20 rows

--- 1.1263461112976074 seconds ---
```

✓ 2s

```
[15] # 7. Cache the the temporary table home_sales.
     spark.sql("cache table sold_homes")
```

```
DataFrame[]
```

✓ 0s

```
[16] # 8. Check if the table is cached.
     spark.catalog.isCached('sold_homes')
```

```
True
```

```
# 9. Using the cached data, run the last query above, that calculates
# the average price of a home per "view" rating, rounded to two decimal place
# having an average home price greater than or equal to $350,000.
# Determine the runtime and compare it to the uncached runtime.

start_time = time.time()

query = """
SELECT
 view,
 round(avg(price), 2) as avg_price
FROM
  sold_homes
GROUP BY
   view
HAVING
  avg(price) >= 350000
order by
  view desc;
"""

spark.sql(query).show()


print("--- %s seconds ---" % (time.time() - start_time))
```

[17]
```
+----+----------+
|view| avg_price|
+----+----------+
|  99|1061201.42|
|  98|1053739.33|
|  97|1129040.15|
|  96|1017815.92|
|  95| 1054325.6|
|  94| 1033536.2|
|  93|1026006.06|
|  92| 970402.55|
|  91|1137372.73|
|  90|1062654.16|
|  89|1107839.15|
|  88|1031719.35|
|  87| 1072285.2|
|  86|1070444.25|
|  85|1056336.74|
|  84|1117233.13|
|  83|1033965.93|
|  82| 1063498.0|
|  81|1053472.79|
|  80| 991767.38|
+----+----------+
only showing top 20 rows

--- 0.6154038906097412 seconds ---
```

[18] 5s
```
# 10. Partition by the "date_built" field on the formatted parquet home sales
df.write.partitionBy("date_built").mode("overwrite").parquet("home_sold_parti
```

[19] 0s
```
# 11. Read the parquet formatted data.
parquet_df = spark.read.parquet("home_sold_partitioned")
```

[20] 0s
```
# 12. Create a temporary table for the parquet data.
parquet_df.createOrReplaceTempView("partitioned_home_sold")
```

```
# 13. Using the parquet DataFrame, run the last query above, that calculates
# the average price of a home per "view" rating, rounded to two decimal places,
# having an average home price greater than or equal to $350,000.
# Determine the runtime and compare it to the cached runtime.

start_time = time.time()

query = """
SELECT
 view,
 round(avg(price), 2) as avg_price
FROM
  partitioned_home_sold
GROUP BY
   view
HAVING
  avg(price) >= 350000
order by
  view desc;
"""
spark.sql(query).show()
print("--- %s seconds ---" % (time.time() - start_time))
```

```
+----+----------+
|view| avg_price|
+----+----------+
|  99|1061201.42|
|  98|1053739.33|
|  97|1129040.15|
|  96|1017815.92|
|  95| 1054325.6|
|  94| 1033536.2|
|  93|1026006.06|
|  92| 970402.55|
|  91|1137372.73|
|  90|1062654.16|
|  89|1107839.15|
|  88|1031719.35|
|  87| 1072285.2|
|  86|1070444.25|
|  85|1056336.74|
|  84|1117233.13|
|  83|1033965.93|
|  82| 1063498.0|
|  81|1053472.79|
|  80| 991767.38|
+----+----------+
only showing top 20 rows

--- 1.4120416641235352 seconds ---
```

[24] 0s

```
# 14. Uncache the home_sales temporary table.
spark.sql("uncache table sold_homes")
```

```
DataFrame[]
```

```
# 15. Check if the home_sales is no longer cached

spark.catalog.isCached("sold_homes")
```

```
False
```