Spark SQL and DataFrames

Objectives

- Describe what Spark SQL is, define the parts of a Spark SQL query, and explain benefits of using Spark SQL
- Describe what DataFrames are, define the parts of a DataFrame query and explain the benefits of using a DataFrame

Spark SQL

- Is a Spark module for structured data processing
- Used to query structured data inside Spark programs, using either SQL or a familiar DataFrame API
- Usable in Java, Scala, Python and R
- Runs SQL queries over imported data and existing RDDs independently of API or programming language

Spark SQL example

Spark SQL query using Python

```
results = spark.sql(
  "SELECT * FROM people")
names = results.map(lambda p:
p.name)
```

Spark SQL example

Spark SQL query using Python

```
results = spark.sql(
  "SELECT * FROM people")
names = results.map(lambda p:
p.name)
```

Spark SQL - Benefits

- Includes a cost-based optimizer, columnar storage, and code generation to make queries fast
- Scales to thousands of nodes and multi-hour queries using the Spark engine, which provides full mid-query fault tolerance
- Provides a programming abstraction called DataFrames and can also act as a distributed SQL query engine

DataFrames

- Distributed collection of data organized into named columns
- Conceptually equivalent to a table in a relational database or a data frame in R/Python, but with richer optimizations
- Built on top of the RDD API
- Uses RDDs
- Performs relational queries

DataFrame Benefits

- Ability to scale from kilobytes of data on a single laptop to petabytes on a large cluster
- Support for a wide array of data formats and storage systems
- State-of-the-art optimization and code generation through the Spark SQL Catalyst optimizer
- Seamless integration with all big data tooling and infrastructure via Spark

Create a DataFrame from reading a CSV/JSON/TXT

```
df_csv = spark.read.csv("people.csv", header=True, inferSchema=True)
```

```
df_json = spark.read.json("people.json", header=True, inferSchema=True)
```

```
df_txt = spark.read.txt("people.txt", header=True, inferSchema=True)
```

- Path to the file and two optional parameters
- Two optional parameters
  - `header=True` , `inferSchema=True`

DataFrame

Python code snippet to read from a JSON file and create a simple DataFrame.

```
df = spark.read.json("people.json")
df.show()
df.printSchema()


# Register the DataFrame as a SQL temporary view
df.createTempView("people")
```

DataFrame example

Input JSON file

```
{"name":"Michael"}
{"name":"Andy",
"age":30}
{"name":"Justin",
"age":19}
```

Created DataFrame

```
+----+-------+
| age| name  |
+----+-------+
|null|Michael|
| 30|  Andy  |
| 19| Justin |
+----+-------+
```

Create a DataFrame from RDD

```
iphones_RDD = sc.parallelize([
    ("XS", 2018, 5.65, 2.79, 6.24),
    ("XR", 2018, 5.94, 2.98, 6.84),
    ("X10", 2017, 5.65, 2.79, 6.13),
    ("8Plus", 2017, 6.23, 3.07, 7.12)
])
```

```
names = ['Model', 'Year', 'Height', 'Width', 'Weight']
```

```
iphones_df = spark.createDataFrame(iphones_RDD, schema=names)
type(iphones_df)
```

```
pyspark.sql.dataframe.DataFrame
```

Interacting with PySpark DataFrames

DataFrame operators in PySpark

- DataFrame operations: Transformations and Actions
- DataFrame Transformations:
  - select(), filter(), groupby(), orderby(), dropDuplicates() and withColumnRenamed()
- DataFrame Actions :
  - head(), show(), count(), columns and describe()

select() and show() operations

- `select()` transformation subsets the columns in the DataFrame

```
df_id_age = test.select('Age')
```

- `show()` action prints first 20 rows in the DataFrame

```
df_id_age.show(3)
```

```
+----+
|Age|
+----+
|  17|
|  17|
|  17|
+----+
only showing top 3 rows
```

filter() and show() operations

- `filter()` transformation filters out the rows based on a condition

```
new_df_age21 = new_df.filter(new_df.Age > 21)
new_df_age21.show(3)
```

```
+-------+------+---+
|User_ID|Gender|Age|
+-------+------+---+
|1000002|     M| 55|
|1000003|     M| 26|
|1000004|     M| 46|
+-------+------+---+
only showing top 3 rows
```

groupby() and count() operations

- `groupby()` operation can be used to group a variable

```
test_df_age_group = test_df.groupby('Age')
test_df_age_group.count().show(3)
```

```
+---+------+
|Age| count|
+---+------+
| 26|219587|
| 17|     4|
| 55| 21504|
+---+------+
only showing top 3 rows
```

orderby() Transformations

- `orderby()` operation sorts the DataFrame based on one or more columns

```
test_df_age_group.count().orderBy('Age').show(3)
```

```
+---+-----+
|Age|count|
+---+-----+
|  0|15098|
| 17|    4|
| 18|99660|
+---+-----+
only showing top 3 rows
```

dropDuplicates()

- `dropDuplicates()` removes the duplicate rows of a DataFrame

```
test_df_no_dup = test_df.select('User_ID','Gender', 'Age').dropDuplicates()
test_df_no_dup.count()
```

```
5892
```

withColumnRenamed Transformations

- `withColumnRenamed()` renames a column in the DataFrame

```
test_df_sex = test_df.withColumnRenamed('Gender', 'Sex')
test_df_sex.show(3)
```

```
+-------+---+---+
|User_ID|Sex|Age|
+-------+---+---+
|1000001|  F| 17|
|1000001|  F| 17|
|1000001|  F| 17|
+-------+---+---+
```

printSchema()

- `printSchema()` operation prints the types of columns in the DataFrame

```
test_df.printSchema()
```

```
 |-- User_ID: integer (nullable = true)
 |-- Product_ID: string (nullable = true)
 |-- Gender: string (nullable = true)
 |-- Age: string (nullable = true)
 |-- Occupation: integer (nullable = true)
 |-- Purchase: integer (nullable = true)
```

columns actions

- `columns` operator prints the columns of a DataFrame

```
test_df.columns
```

```
['User_ID', 'Gender', 'Age']
```

describe() actions

- `describe()` operation compute summary statistics of numerical columns in the DataFrame

```
test_df.describe().show()
```

```
+-------+------------------+------+------------------+
|summary|           User_ID|Gender|               Age|
+-------+------------------+------+------------------+
|  count|            550068|550068|            550068|
|   mean|1003028.8424013031|  null|30.382052764385495|
| stddev|1727.5915855307312|  null|11.866105189533554|
|    min|           1000001|     F|                 0|
|    max|           1006040|     M|                55|
+-------+------------------+------+------------------+
```

Interacting with DataFrames using PySpark SQL

DataFrame API vs SQL queries

- In PySpark You can interact with SparkSQL through DataFrame API and SQL queries
- The DataFrame API provides a programmatic domain-specific language (DSL) for data
- DataFrame transformations and actions are easier to construct programmatically
- SQL queries can be concise and easier to understand and portable
- The operations on DataFrames can also be done using SQL queries

Executing SQL Queries

- The SparkSession `sql()` method executes SQL query
- `sql()` method takes a SQL statement as an argument and returns the result as DataFrame

```
df.createOrReplaceTempView("table1")
```

```
df2 = spark.sql("SELECT field1, field2 FROM table1")
df2.collect()
```

```
[Row(f1=1, f2='row1'), Row(f1=2, f2='row2'), Row(f1=3, f2='row3')]
```

SQL query to extract data

```
test_df.createOrReplaceTempView("test_table")
```

```
query = '''SELECT Product_ID FROM test_table'''
```

```
test_product_df = spark.sql(query)
test_product_df.show(5)
```

```
+-------------------+
|Product_ID|
+-------------------+
|  P00069042|
|  P00248942|
|  P00087842|
|  P00085442|
|  P00285442|
+-------------------+
```

Summarizing and grouping data using SQL queries

```
test_df.createOrReplaceTempView("test_table")
```

```
query = '''SELECT Age, max(Purchase) FROM test_table GROUP BY Age'''
```

```
spark.sql(query).show(5)
```

```
+-----+-------------+
|  Age|max(Purchase)|
+-----+-------------+
|18-25|        23958|
|26-35|        23961|
| 0-17|        23955|
|46-50|        23960|
|51-55|        23960|
+-----+-------------+
only showing top 5 rows
```

Filtering columns using SQL queries

```
test_df.createOrReplaceTempView("test_table")
```

```
query = '''SELECT Age, Purchase, Gender FROM test_table WHERE Purchase > 20000 AND Gender == "F"'''
```

```
spark.sql(query).show(5)
```

```
+-----+--------+------+
|  Age|Purchase|Gender|
+-----+--------+------+
|36-45|   23792|     F|
|26-35|   21002|     F|
|26-35|   23595|     F|
|26-35|   23341|     F|
|46-50|   20771|     F|
+-----+--------+------+
only showing top 5 rows
```

DataFrame + Spark SQL

SQL Query

```
spark.sql("SELECT name FROM people").show()
```

DataFrame Python API

```
df.select("name").show()
df.select(df["name"]).show()
```

Result

```
+-------+
| name  |
+-------+
|Michael|
| Andy  |
| Justin|
+-------+
```

DataFrame + Spark SQL

SQL Query

```
spark.sql("SELECT name FROM people").show()
```

DataFrame Python API

```
df.select("name").show()
df.select(df["name"]).show()
```

Result

```
+-------+
| name  |
+-------+
|Michael|
| Andy  |
| Justin|
+-------+
```

DataFrame + Spark SQL

SQL Query

```
spark.sql("SELECT name FROM people").show()
```

DataFrame Python API

```
df.select("name").show()
df.select(df["name"]).show()
```

Result

```
+-------+
| name  |
+-------+
|Michael|
| Andy  |
| Justin|
+-------+
```

DataFrame + Spark SQL

SQL Query

```
spark.sql("SELECT age, name
FROM people WHERE age >
21").show()
```

DataFrame Python API

```
df.filter(df["age"]>21).show()
```

Result

```
+---+----+
|age|name|
+---+----+
| 30|Andy|
+---+----+
```

Summary

In this video, you learned that:

- Spark SQL is a Spark module for structured data processing
- Spark SQL provides a programming abstraction called DataFrames and can also act as a distributed SQL query engine
- DataFrames are conceptually equivalent to a table in a relational database or a data frame in R/Python, but with richer optimizations