databricks spark_review_for_databricks_cert.py

Databricks Developer Certification (python)

https://academy.databricks.com/exam/crt020-python (https://academy.databricks.com/exam/crt020-python)

Expectation of Knowledge

Spark Architecture Components

Candidates are expected to be familiar with the following architectural components and their relationship to each other:

- Driver
- Executor
- Cores/Slots/Threads
- Partitions

Spark Execution

Candidates are expected to be familiar with Spark's execution model and the breakdown between the different elements:

- Jobs
- Stages
- Tasks

Spark Concepts

Candidates are expected to be familiar with the following concepts:

- Caching
- Shuffling
- Partitioning
- Wide vs. Narrow Transformations
- DataFrame Transformations vs. Actions vs. Operations

- High-level Cluster Configuration

DataFrames API

- SparkContext
  - Candidates are expected to know how to use the SparkContext to control basic configuration settings such as spark.sql.shuffle.partitions.

- SparkSession
  - Candidates are expected to know how to:
    - Create a DataFrame/Dataset from a collection (e.g. list or set)
    - Create a DataFrame for a range of numbers
    - Access the DataFrameReaders
    - Register User Defined Functions (UDFs)

- DataFrameReader
  - Candidates are expected to know how to:
    - Read data for the "core" data formats (CSV, JSON, JDBC, ORC, Parquet, text and tables)
    - How to configure options for specific formats
    - How to read data from non-core formats using format() and load()
    - How to specify a DDL-formatted schema
    - How to construct and specify a schema using the StructType classes

- DataFrameWriter
  - Candidates are expected to know how to:
    - Write data to the "core" data formats (csv, json, jdbc, orc, parquet, text and tables)
    - Overwriting existing files
    - How to configure options for specific formats
    - How to write a data source to 1 single file or N separate files
    - How to write partitioned data
    - How to bucket data by a given set of columns

- DataFrame
  - Have a working understanding of every action such as take(), collect(), and foreach()

  - Have a working understanding of the various transformations and how they work such as producing a distinct set, filtering data, repartitioning and coalescing, performing joins and unions as well as producing aggregates
  - Know how to cache data, specifically to disk, memory or both
  - Know how to uncache previously cached data
  - Converting a DataFrame to a global or temp view.
  - Applying hints

- Row & Column
  - Candidates are expected to know how to work with row and columns to successfully extract data from a DataFrame

- Spark SQL Functions
  - When instructed what to do, candidates are expected to be able to employ the multitude of Spark SQL functions. Examples include, but are not limited to:
    - Aggregate functions: getting the first or last item from an array or computing the min and max values of a column.
    - Collection functions: testing if an array contains a value, exploding or flattening data.
    - Date time functions: parsing strings into timestamps or formatting timestamps into strings
    - Math functions: computing the cosign, floor or log of a number
    - Misc functions: converting a value to crc32, md5, sha1 or sha2
    - Non-aggregate functions: creating an array, testing if a column is null, not-null, nan, etc
    - Sorting functions: sorting data in descending order, ascending order, and sorting with proper null handling
    - String functions: applying a provided regular expression, trimming string and extracting substrings.
    - UDF functions: employing a UDF function.
    - Window functions: computing the rank or dense rank.

SparkContext

Candidates are expected to know how to use the SparkContext to control basic configuration settings such as spark.sql.shuffle.partitions.

```
spark.conf.set("spark.sql.shuffle.partitions", 6)
spark.conf.set("spark.executor.memory", "2g")
```

```
print(spark.conf.get("spark.sql.shuffle.partitions"), ",",
spark.conf.get("spark.executor.memory"))
```

```
%sql
SET spark.sql.shuffle.partitions = 8;
SET spark.executor.memory = 4g;
```

```
print(spark.conf.get("spark.sql.shuffle.partitions"), ",",
spark.conf.get("spark.executor.memory"))
```

SparkSession

Candidates are expected to know how to:

- Create a DataFrame/Dataset from a collection (e.g. list or set)

```
from pyspark.sql.types import IntegerType

list_df = spark.createDataFrame([1, 2, 3, 4], IntegerType())
display(my_list_df)
```

- Create a DataFrame for a range of numbers

```
ints_df = spark.range(1000).toDF("number")
display(ints_df)
```

- Access the DataFrameReaders

```
df = spark.read.csv('/FileStore/tables/input.csv', inferSchema=True)
# spark.read.parquet()
# spark.read.json()
# spark.read.format().open()
```

- Register User Defined Functions (UDFs)

```
from pyspark.sql.functions import udf
from pyspark.sql.types import IntegerType

def power3(value):
  return value ** 3

spark.udf.register("power3py", power3, IntegerType())
```

```
power3udf = udf(power3, IntegerType())
power3_ints_df = ints_df.select("number", power3udf("number").alias("power3"))
display(power3_ints_df)
```

```
spark.range(1, 20).registerTempTable("test")
```

```
%sql select id, power3py(id) as power3 from test
```

DataFrameReader

- Read data for the "core" data formats (CSV, JSON, JDBC, ORC, Parquet, text and tables)

```
data_file = "/FileStore/tables/sales.csv"

df = spark.read.csv(data_file)
display(df)
```

- How to configure options for specific formats

```
df = spark.read.csv(data_file, header=True, inferSchema=True)
display(df)
```

- How to read data from non-core formats using format() and load()

```
df =
spark.read.format("csv").option("inferSchema","true").option("header","true").l
oad(data_file)
```

- How to construct and specify a schema using the StructType classes

```
from pyspark.sql.types import StructField, StructType, StringType, LongType

myManualSchema = StructType([
  StructField("field1", StringType()),
  StructField("field2", StringType()),
  StructField("field3", StringType())
])

df3 =
spark.read.format("csv").schema(myManualSchema).option("header","true").load(da
ta_file)
df3.show()
```

- How to specify a DDL-formatted schema

DataFrameWriter

- Write data to the "core" data formats (csv, json, jdbc, orc, parquet, text and tables)

```
df.write.parquet('myparquetfile')
df.write.saveAsTable('mytable')
```

- Overwriting existing files

```
#df.write.mode("overwrite")
```

- How to configure options for specific formats

```
csvFile = (spark.read.format("csv")
  .option("header", "true")
  .option("inferSchema", "true")
  .load(data_file))

csvFile.write.format("csv").mode("overwrite").option("sep", "\t").save("my-tsv-
file.tsv")
```

- How to write a data source to 1 single file or N separate files

```
# df.coalesce(1)
# df.repartition(1)
```

- How to write partitioned data
- How to bucket data by a given set of columns

```
(df
    .write
    .partitionBy("ProductKey")
    .bucketBy(42, "OrderDateKey")
    .saveAsTable("orders_partitioned_bucketed"))
```

DataFrame

- Have a working understanding of every action such as take(), collect(), and foreach()

- Have a working understanding of the various transformations and how they work such as producing a distinct set, filtering data, repartitioning and coalescing, performing joins and unions as well as producing aggregates

- Know how to cache data, specifically to disk, memory or both

```
from pyspark.storagelevel import StorageLevel

df.persist(StorageLevel.MEMORY_AND_DISK)
```

- Know how to uncache previously cached data

```
df.unpersist()
```

- Converting a DataFrame to a global or temp view

```
# df.createOrReplaceTempView("<table-name>")
```

- Applying hints