Databricks

Databricks-Certified-Associate-Developer-for-Apache-Spark-3.0

Databricks Certified Associate Developer for Apache Spark 3.0

QUESTION & ANSWERS

Question: 1

PESCH Tests	Number of Questions	Total Question
Practice Test Scala #1	60	1 to 60
Practice Test Python #1	60	61 to 120
Practice Test Scats #2	60	121 to 180
Practice Test Python #2	62	181 to 242
Total		242

The code block down below intends to join df1 with df2 with inner join but it contains an error. Identify the error. d1.join(d2, “inner”, d1.col(“id”) === df2.col(“id"))

1. We cannot do inner join in spark 3.0, but it is in the roadmap.
2. d1.join(d2, d1.col(“id”) == df2.col(‘id'), “inner”)
3. There should be two == instead of ===. So the correct query is d1.join(d2, “inner”, d1.col(“id”) == df2.col(‘id'))
4. The join type is not in right order. The correct query should be d1.join(d2, d1.col(“id”) === df2.col(‘id'), “inner”)

Answer: D

Explanation/Reference:

Correct syntax is;
df1.join(df2, joinExpression, joinType)

Question: 2

Which of the following code blocks returns a DataFrame with a new column aSquared and all previously existing columns from DataFrame df given that df has a column named a ?

1. df.withColumn(“aSquared”, col(“a”) * col(“a”))
2. df.withColumn(“aSquared”, col(a) * col(a))
3. df.withColumn(aSquared, col(a) * col(a))
4. df.withColumn(aSquared, col(“a”) * col(“a”))
5. df.withColumn(col(“a”) * col(“a”), “aSquared”)

Answer: A

Explanation/Reference:

You will have such questions in the exam, be careful while reading the responses.

Question: 3

Which of the following 3 DataFrame operations are classified as a wide transformation ? Choose 3 answers:

1. orderBy()
2. drop()
3. repartition()
4. distinct()
5. filter()
6. cache()

Answer: A,C,D

Explanation/Reference:

Please get familiar with wide transformations, narrow transformations and actions. You will be tested on this topic in your exam.
Narrow transformation – In Narrow transformation, all the elements that are required to compute the records in single partition live in the single partition of parent
RDD. A limited subset of partition is used to calculate the result. Narrow transformations are the result of map(), filter().
Wide transformation – In wide transformation, all the elements that are required to compute the records in the single partition may live in many partitions of parent
RDD. The partition may live in many partitions of parent RDD. Wide transformations are the result of groupbyKey() and reducebyKey().
see https://spark.apache.org/docs/latest/rdd-programming-guide.html#transformations

Question: 4

Given that the number of partitions of dataframe df is 4 and we want to write a parquet file in a given path. Choose the correct number of files after a successful write operation.

1. 4
2. 1
3. 8

Answer: A

Explanation/Reference:

We control the parallelism of files that we write by controlling the partitions prior to writing and therefore the number of partitions before writing equals to number of files created after the write operation.

Question: 5

Which of the following statement is true for broadcast variables ?

1. Broadcast variables are shared, immutable variables that are cached on every machine in the cluster instead of serialized with every single task
2. It provides a mutable variable that a Spark cluster can safely update on a per-row basis
3. The canonical use case is to pass around a extermely large table that does not fit in memory on the executors.
4. It is a way of updating a value inside of a variety of transformations and propagating that value to the driver node in an efficient and fault-tolerant way.

Answer: A

Explanation/Reference:

Broadcast variables are a way you can share an immutable value efficiently around the cluster without encapsulating that variable in a function closure. The normal way to use a variable in your driver node inside your tasks is to simply reference it in your function closures (e.g., in a map operation), but this can be inefficient, especially for large variables such as a lookup table or a machine learning model. The reason for this is that when you use a variable in a closure, it must be deserialized on the worker nodes many times (one per task)

Question: 6

When joining two dataframes, if there is a need to evaluate the keys in both of the DataFrames or tables and include all rows from the left DataFrame as well as any rows in the right DataFrame that have a match in the left DataFrame also If there is no equivalent row in the right DataFrame, we want to instert null: which join type we should select ? df1.join(person, joinExpression, joinType)

1. val joinType = “left_outer”
2. val joinType = “left_semi”
3. val joinType = “leftOuter”
4. val joinType = “leftAnti”

Answer: A

Explanation/Reference:

Correct answer is joinType = "left_outer". For example df1.join(person, joinExpression, “left_outer”).show()

Question: 7

Given the following statements regarding caching:
1. Red: The default storage level for a DataFrame is StorageLevel.MEMORY_AND_DISK
2.
3. Green: The DataFrame class does not have an uncache() operation
4.
5. Blue: The persist() method immediately loads data from its source to materialize the DataFrame in cache
6.
7. White: Explicit caching can decrease application performance by interferring with the Catalyst optimizer's ability to optimize some queries.
Which of these statements are TRUE?

1. Red, White and Green
2. Green and White
3. Red, Blue, and White
4. Green and Blue

Answer: A

Explanation/Reference:

To materialize the DataFrame in cache, you need to call an action (and also you need to be using all partitions with that action otherwise it will only cache some partitions)

Question: 8

Consider the following DataFrame:
1. import org.apache.spark.sql.functions._
2. val peopleDF =
3. Seq(
4. ("Ali", 0, Seq(100)),
5. ("Barbara", 1, Seq(300, 250, 100)),
6. ("Cesar", 1, Seq(350, 100)),
7. ("Dongmei", 1, Seq(400, 100)),
8. ("Eli", 2, Seq(250)),
9. ("Florita", 2, Seq(500, 300, 100)),
10. ("Gatimu", 3, Seq(300, 100))
11.).toDF("name", "department", "score")
Select the code fragment that produces the following result:
|department| name |highest|
0	Ali	100
1	Dongmei	400
2	Florita	500
3	Gatimu	300

1. 1. peopleDF
   2. .withColumn("score", explode(col("score")))
   3. .groupBy("department")
   4. .max("score")
   5. .withColumnRenamed("max(score)", "highest")
   6. .orderBy("department")
   7. .show()

2. 1. val maxByDept = peopleDF
   2. .withColumn("score", explode(col("score")))
   3. .groupBy("department")

4. .max("score")
5. .withColumnRenamed("max(score)", "highest")
6.
7. maxByDept.join(people, "department")
8. .select("department", "name", "highest")
9. .orderBy("department")
10. .dropDuplicates("department")
11. .show()

3. 1. peopleDF
   2. .withColumn("score", explode(col("score")))
   3. .orderBy("department", "score")
   4. .select(col("name"), col("department"), first(col("score")).as("highest"))
   5. .show()
4. 1. import org.apache.spark.sql.expressions.Window
   2. val windowSpec = Window.partitionBy("department").orderBy(col("score").desc)
   3.
   4. peopleDF
   5. .withColumn("score", explode(col("score")))
   6. .select(col("department"),col("name"),dense_rank().over(windowSpec).alias("rank"), max(col("score")).over(windowSpec).alias("highest"))
   7. .where(col("rank") === 1)
   8. .drop("rank")
   9. .orderBy("department")
   10. .show()

Answer: D

Question: 9

tableA is a DataFrame consisting of 20 fields and 40 billion rows of data with a surrogate key field. tableB is a DataFrame functioning as a lookup table for the surrogate
key consisting of 2 fields and 5,000 rows. If the in-memory size of tableB is 22MB, what occurs when the following code is executed:? val df = tableA.join(tableB,
"primary_key")

1. The contents of tableB will be partitioned so that each of the keys that need to be joined on in tableA partitions on each executor will match.
2. A non-broadcast join will be executed with a shuffle phase since the broadcast table is greater than the 10MB default threshold and the broadcast hint was not specified.
3. An exception will be thrown due to tableB being greater than the 10MB default threshold for a broadcast join.
4. The contents of tableB will be replicated and sent to each executor to eliminate the need for a shuffle stage during the join.

Answer: B

Explanation/Reference:

By default spark.sql.autoBroadcastJoinThreshold= 10MB and any value above this thershold will not force a broadcast join

Question: 10

If we want to store RDD as deserialized Java objects in the JVM and if the RDD does not fit in memory, store the partitions that don’t fit on disk, and read them from there when they’re needed also replicate each partition on two cluster nodes, which storage level we need to choose ?

1. MEMORY_AND_DISK_2_SER
2. MEMORY_AND_DISK_2
3. MEMORY_AND_DISK
4. MEMORY_ONLY_2