DATABRICKS-CERTIFIED-ASSOCIAT

Q&As

Databricks Certified Associate Developer for Apache Spark 3.0

Pass Databricks DATABRICKS-CERTIFIED-ASSOCIATE-DEVELOPER-FOR-APACHE-SPARK Exam with 100% Guarantee

Free Download Real Questions & Answers **PDF** and **VCE** file from:

https://www.leads4pass.com/databricks-certified-associate-developer-for-apache-spark.html

100% Passing Guarantee
100% Money Back Assurance

Following Questions and Answers are all new published by Databricks Official Exam Center

QUESTION 1

Which of the following statements about reducing out-of-memory errors is incorrect?

A. Concatenating multiple string columns into a single column may guard against out-of- memory errors.

B. Reducing partition size can help against out-of-memory errors.

C. Limiting the amount of data being automatically broadcast in joins can help against out- of-memory errors.

D. Setting a limit on the maximum size of serialized data returned to the driver may help prevent out-ofmemory errors.

E. Decreasing the number of cores available to each executor can help against out-of- memory errors.

Correct Answer: A

Concatenating multiple string columns into a single column may guard against out-of- memory errors. Exactly, this is an incorrect answer! Concatenating any string columns does not reduce the size of the data, it just structures it a different way. This does little to how Spark processes the data and definitely does not reduce out-of-memory errors. Reducing partition size can help against out-of-memory errors. No, this is not incorrect. Reducing partition size is a viable way to aid against out-of- memory errors, since executors need to load partitions into memory before processing them. If the executor does not have enough memory available to do that, it will throw an out-of-memory error. Decreasing partition size can therefore be very helpful for preventing that. Decreasing the number of cores available to each executor can help against out-of- memory errors. No, this is not incorrect. To process a partition, this partition needs to be loaded into the memory of an executor. If you imagine that every core in every executor processes a partition, potentially in parallel with other executors, you can imagine that memory on the machine hosting the executors fills up quite quickly. So, memory usage of executors is a concern, especially when multiple partitions are processed at the same time. To strike a balance between performance and memory usage, decreasing the number of cores may help against out-of-memory errors. Setting a limit on the maximum size of serialized data returned to the driver may help prevent out-of-memory errors. No, this is not incorrect. When using commands like collect() that trigger the transmission of potentially large amounts of data from the cluster to the driver, the driver may experience out-of-memory errors. One strategy to avoid this is to be careful about using commands like collect() that send back large amounts of data to the driver. Another strategy is setting the parameter spark.driver.maxResultSize. If data to be transmitted to the driver exceeds the threshold specified by the parameter, Spark will abort the job and therefore prevent an out-of- memory error. Limiting the amount of data being automatically broadcast in joins can help against out-of- memory errors. Wrong, this is not incorrect. As part of Spark\\'s internal optimization, Spark may choose to speed up operations by broadcasting (usually relatively small) tables to executors. This broadcast is happening from the driver, so all the broadcast tables are loaded into the driver first. If these tables are relatively big, or multiple mid-size tables are being broadcast, this may lead to an out-ofmemory error. The maximum table size for which Spark will consider broadcasting is set by the spark.sql.autoBroadcastJoinThreshold parameter. More info: Configuration - Spark 3.1.2 Documentation and Spark OOM Error -- Closeup. Does the following look familiar when... | by Amit Singh Rathore | The Startup | Medium

QUESTION 2

The code block displayed below contains multiple errors. The code block should remove column transactionDate from DataFrame transactionsDf and add a column transactionTimestamp in which

dates that are expressed as strings in column transactionDate of DataFrame transactionsDf are converted into unix timestamps. Find the errors.

Sample of DataFrame transactionsDf:

1.+-------------+---------+-----+-------+---------+----+----------------+

2.|transactionId|predError|value|storeId|productId| f| transactionDate|
3.+-------------+---------+-----+-------+---------+----+----------------+

4.| 1| 3| 4| 25| 1|null|2020-04-26 15:35|

5.| 2| 6| 7| 2| 2|null|2020-04-13 22:01|

6.| 3| 3| null| 25| 3|null|2020-04-02 10:53|

7.+-------------+---------+-----+-------+---------+----+----------------+

Code block:

1.transactionsDf = transactionsDf.drop("transactionDate")

2.transactionsDf["transactionTimestamp"] = unix_timestamp("transactionDate", "yyyy-MM- dd")

A. Column transactionDate should be dropped after transactionTimestamp has been written. The string indicating the date format should be adjusted. The withColumn operator should be used instead of the existing column assignment. Operator to_unixtime() should be used instead of unix_timestamp().

B. Column transactionDate should be dropped after transactionTimestamp has been written. The withColumn operator should be used instead of the existing column assignment. Column transactionDate should be wrapped in a col() operator.

C. Column transactionDate should be wrapped in a col() operator.

D. The string indicating the date format should be adjusted. The withColumnReplaced operator should be used instead of the drop and assign pattern in the code block to replace column transactionDate with the new column transactionTimestamp.

E. Column transactionDate should be dropped after transactionTimestamp has been written. The string indicating the date format should be adjusted. The withColumn operator should be used instead of the existing column assignment.

Correct Answer: E

This requires a lot of thinking to get right. For solving it, you may take advantage of the digital notepad that is provided to you during the test. You have probably seen that the code block includes multiple errors. In the test, you are usually confronted with a code block that only contains a single error. However, since you are practicing here, this challenging multi-error will make it easier for you to deal with single-error questions in the real exam. You can clearly see that column transactionDate should be dropped only after transactionTimestamp has been written. This is because to generate column transactionTimestamp, Spark needs to read the values from column transactionDate. Values in column transactionDate in the original transactionsDf DataFrame look like 2020- 04-26 15:35. So, to convert those correctly, you would have to pass yyyy-MM-dd HH:mm. In other words: The string indicating the date format should be adjusted. While you might be tempted to change unix_timestamp() to to_unixtime() (in line with the from_unixtime() operator), this function does not exist in Spark. unix_timestamp() is the correct operator to use here. Also, there is no DataFrame.withColumnReplaced() operator. A similar operator that exists is DataFrame.withColumnRenamed(). Whether you use col() or not is irrelevant with unix_timestamp() - the command is fine with both. Finally, you cannot assign a column like transactionsDf["columnName"] = ... in Spark. This is Pandas syntax (Pandas is a popular Python package for data analysis), but it is not supported in Spark. So, you need to use Spark\\'s DataFrame.withColumn() syntax instead. More info: pyspark.sql.functions.unix_timestamp -- PySpark 3.1.2 documentation Static notebook | Dynamic notebook: See test 3, 28 (Databricks import instructions)

QUESTION 3

Which of the following describes Spark\\'s way of managing memory?

A. Spark uses a subset of the reserved system memory.

B. Storage memory is used for caching partitions derived from DataFrames.

C. As a general rule for garbage collection, Spark performs better on many small objects than few big objects.

D. Disabling serialization potentially greatly reduces the memory footprint of a Spark application.

E. Spark\\'s memory usage can be divided into three categories: Execution, transaction, and storage.

Correct Answer: B

Spark\\'s memory usage can be divided into three categories: Execution, transaction, and storage.

No, it is either execution or storage.

As a general rule for garbage collection, Spark performs better on many small objects than few big objects.

No, Spark\\'s garbage collection runs faster on fewer big objects than many small objects. Disabling

serialization potentially greatly reduces the memory footprint of a Spark application.

The opposite is true ?serialization reduces the memory footprint, but may impact performance in a

negative way.

Spark uses a subset of the reserved system memory. No, the reserved system memory is separate from

Spark memory. Reserved memory stores Spark\\'s internal objects.

More info: Tuning - Spark 3.1.2 Documentation, Spark Memory Management | Distributed Systems

Architecture, Learning Spark, 2nd Edition, Chapter 7

QUESTION 4

Which of the following statements about storage levels is incorrect?

A. The cache operator on DataFrames is evaluated like a transformation.

B. In client mode, DataFrames cached with the MEMORY_ONLY_2 level will not be stored in the edge node\\'s memory.

C. Caching can be undone using the DataFrame.unpersist() operator.

D. MEMORY_AND_DISK replicates cached DataFrames both on memory and disk.

E. DISK_ONLY will not use the worker node\\'s memory.

Correct Answer: D

MEMORY_AND_DISK replicates cached DataFrames both on memory and disk. Correct, this statement is

wrong. Spark prioritizes storage in memory, and will only store data on disk that does not fit into memory.

DISK_ONLY will not use the worker node\\'s memory.

Wrong, this statement is correct. DISK_ONLY keeps data only on the worker node\\'s disk, but not in

memory.

In client mode, DataFrames cached with the MEMORY_ONLY_2 level will not be stored in the edge node\\'s

memory.

Wrong, this statement is correct. In fact, Spark does not have a provision to cache DataFrames in the

driver (which sits on the edge node in client mode). Spark caches DataFrames in the executors\\' memory.

Caching can be undone using the DataFrame.unpersist() operator. Wrong, this statement is correct. Caching, as achieved via the DataFrame.cache() or DataFrame.persist() operators can be undone using the DataFrame.unpersist() operator. This operator will remove all of its parts from the executors\\' memory and disk. The cache operator on DataFrames is evaluated like a transformation. Wrong, this statement is correct. DataFrame.cache() is evaluated like a transformation: Through lazy evaluation. This means that after calling DataFrame.cache() the command will not have any effect until you call a subsequent action, like DataFrame.cache().count(). More info: pyspark.sql.DataFrame.unpersist -- PySpark 3.1.2 documentation

---

QUESTION 5

The code block displayed below contains one or more errors. The code block should load parquet files at location filePath into a DataFrame, only loading those files that have been modified before 2029-03-20

05:44:46. Spark should enforce a schema according to the schema shown below. Find the error.

Schema:

1.root

2.

|-- itemId: integer (nullable = true)

3.

|-- attributes: array (nullable = true)

4.

| |-- element: string (containsNull = true)

5.

|-- supplier: string (nullable = true)

Code block:

1.schema = StructType([

2.

StructType("itemId", IntegerType(), True),

3.

StructType("attributes", ArrayType(StringType(), True), True),

4.

StructType("supplier", StringType(), True)

5.])

6.

7.spark.read.options("modifiedBefore", "2029-03-20T05:44:46").schema(schema).load(filePath)

A. The attributes array is specified incorrectly, Spark cannot identify the file format, and the syntax of the call to Spark\\'s DataFrameReader is incorrect.

B. Columns in the schema definition use the wrong object type and the syntax of the call to Spark\\'s DataFrameReader is incorrect.

C. The data type of the schema is incompatible with the schema() operator and the modification date threshold is specified incorrectly.

D. Columns in the schema definition use the wrong object type, the modification date threshold is specified incorrectly, and Spark cannot identify the file format.

E. Columns in the schema are unable to handle empty values and the modification date threshold is specified incorrectly.

Correct Answer: D