# 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**

SATISFACTION GUARANTEED
100%
SATISFACTION GUARANTEED

**QUESTION 1**

Which of the following code blocks stores a part of the data in DataFrame itemsDf on executors?

A. itemsDf.cache().count()

B. itemsDf.cache(eager=True)

C. cache(itemsDf)

D. itemsDf.cache().filter()

E. itemsDf.rdd.storeCopy()

Correct Answer: A

---

**QUESTION 2**

Which of the following code blocks saves DataFrame transactionsDf in location /FileStore/transactions.csv as a CSV file and throws an error if a file already exists in the location?

A. transactionsDf.write.save("/FileStore/transactions.csv")

B. transactionsDf.write.format("csv").mode("error").path("/FileStore/transactions.csv")

C. transactionsDf.write.format("csv").mode("ignore").path("/FileStore/transactions.csv")

D. transactionsDf.write("csv").mode("error").save("/FileStore/transactions.csv")

E. transactionsDf.write.format("csv").mode("error").save("/FileStore/transactions.csv")

Correct Answer: E

Static notebook | Dynamic notebook: See test 1, 28 (Databricks import instructions) (https://flrs.github.io/spark_practice_tests_code/#1/28.html , https://bit.ly/sparkpracticeexams_import_instructions)

---

**QUESTION 3**

Which of the following code blocks writes DataFrame itemsDf to disk at storage location filePath, making sure to substitute any existing data at that location?

A. itemsDf.write.mode("overwrite").parquet(filePath)

B. itemsDf.write.option("parquet").mode("overwrite").path(filePath)

C. itemsDf.write(filePath, mode="overwrite")

D. itemsDf.write.mode("overwrite").path(filePath)

E. itemsDf.write().parquet(filePath, mode="overwrite")

Correct Answer: A

**QUESTION 4**

The code block shown below should read all files with the file ending .png in directory path into Spark. Choose the answer that correctly fills the blanks in the code block to accomplish this.

spark.__1__.__2__(__3__).option(__4__, "*.png").__5__(path)

A. 1. read()

2.

format

3.

"binaryFile"

4.

"recursiveFileLookup"

5.

load

B. 1. read

2.

format

3.

"binaryFile"

4.

"pathGlobFilter"

5.

load

C. 1. read

2.

format

3.

binaryFile

4.

pathGlobFilter

5.

load

D. 1. open

2.

format

3.

"image"

4.

"fileType"

5.

open

E. 1. open

2.

as

3.

"binaryFile"

4.

"pathGlobFilter"

5.

load

Correct Answer: B

Correct code block: spark.read.format("binaryFile").option("recursiveFileLookup", "*.png").load(path) Spark can deal with binary files, like images. Using the binaryFile format specification in the SparkSession\\'s read API is the way to read in those files. Remember that, to access the read API, you need to start the command with spark.read. The pathGlobFilter option is a great way to filter files by name (and ending). Finally, the path can be specified using the load operator ?the open operator shown in one of the answers does not exist.

**QUESTION 5**

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