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 describes slots?

A. Slots are dynamically created and destroyed in accordance with an executor\\'s workload.

B. To optimize I/O performance, Spark stores data on disk in multiple slots.

C. A Java Virtual Machine (JVM) working as an executor can be considered as a pool of slots for task execution.

D. A slot is always limited to a single core. Slots are the communication interface for executors and are used for receiving commands and sending results to the driver.

Correct Answer: C

Slots are the communication interface for executors and are used for receiving commands and sending results to the driver. Wrong, executors communicate with the driver directly. Slots are dynamically created and destroyed in accordance with an executor\\'s workload. No, Spark does not actively create and destroy slots in accordance with the workload. Per executor, slots are made available in accordance with how many cores per executor (property spark.executor.cores) and how many CPUs per task (property spark.task.cpus) the Spark configuration calls for. A slot is always limited to a single core. No, a slot can span multiple cores. If a task would require multiple cores, it would have to be executed through a slot that spans multiple cores. In Spark documentation, "core" is often used interchangeably with "thread", although "thread" is the more accurate word. A single physical core may be able to make multiple threads available. So, it is better to say that a slot can span multiple threads. To optimize I/O performance, Spark stores data on disk in multiple slots. No ?Spark stores data on disk in multiple partitions, not slots. More info: Spark Architecture | Distributed Systems Architecture

QUESTION 2

The code block shown below should convert up to 5 rows in DataFrame transactionsDf that have the value 25 in column storeId into a Python list. Choose the answer that correctly fills the blanks in the code block to accomplish this.

Code block:

transactionsDf.__1__(__2__).__3__(__4__)

A. 1. filter

2.

"storeId"==25

3.

collect

4.

5

B. 1. filter

2.

col("storeId")==25

3.

toLocalIterator

4.

5

C. 1. select

2.

storeId==25

3.

head

4.

5

D. 1. filter

2.

col("storeId")==25

3.

take

4.

5

E. 1. filter

2.

col("storeId")==25

3.

collect

4.

5

Correct Answer: D

QUESTION 3

The code block displayed below contains an error. The code block should produce a DataFrame with color

as the only column and three rows with color values of red, blue, and green, respectively.

Find the error.

Code block:

1.spark.createDataFrame([("red",), ("blue",), ("green",)], "color")

2.Instead of calling spark.createDataFrame, just DataFrame should be called.

A. The commas in the tuples with the colors should be eliminated.

B. The colors red, blue, and green should be expressed as a simple Python list, and not a list of tuples.

C. Instead of color, a data type should be specified.

D. The "color" expression needs to be wrapped in brackets, so it reads ["color"].

Correct Answer: D

QUESTION 4

Which of the following code blocks returns a new DataFrame with only columns predError and values of

every second row of DataFrame transactionsDf?

Entire DataFrame transactionsDf:

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

2.|transactionId|predError|value|storeId|productId| f|

3.+-------------+---------+-----+-------+---------+----+

4.| 1| 3| 4| 25| 1|null|

5.| 2| 6| 7| 2| 2|null|

6.| 3| 3| null| 25| 3|null|

7.| 4| null| null| 3| 2|null|

8.| 5| null| null| null| 2|null|

9.| 6| 3| 2| 25| 2|null|

10.+-------------+---------+-----+-------+---------+----+
```

A. transactionsDf.filter(col("transactionId").isin([3,4,6])).select([predError, value])

B. transactionsDf.select(col("transactionId").isin([3,4,6]), "predError", "value")

C. transactionsDf.filter("transactionId" % 2 == 0).select("predError", "value")

D. transactionsDf.filter(col("transactionId") % 2 == 0).select("predError", "value") (Correct)

E. 1.transactionsDf.createOrReplaceTempView("transactionsDf") 2.spark.sql("FROM transactionsDf SELECT predError, value WHERE transactionId % 2 = 2")

F. transactionsDf.filter(col(transactionId).isin([3,4,6]))

Correct Answer: D

QUESTION 5

The code block shown below should return a copy of DataFrame transactionsDf with an added column

cos. This column should have the values in column value converted to degrees and having the cosine of

those converted values taken, rounded to two decimals. Choose the answer that correctly fills the blanks in

the code block to accomplish this.

Code block:

transactionsDf.__1__(__2__, round(__3__(__4__(__5__)),2))

A. 1. withColumn

2.

col("cos")

3.

cos

4.

degrees

5.

transactionsDf.value

B. 1. withColumnRenamed

2.

"cos"

3.

cos

4.

degrees

5.

"transactionsDf.value"

C. 1. withColumn

2.

"cos"

3.

cos

4.

degrees

5.

transactionsDf.value

D. 1. withColumn

2.

col("cos")

3.

cos

4.

degrees

5.

col("value")

E. 1. withColumn

2.

"cos"

3.

degrees

4.

cos

5.

col("value")

Correct Answer: C

QUESTION 6

Which of the following code blocks reads in the two-partition parquet file stored at filePath, making sure all columns are included exactly once even though each partition has a different schema?

Schema of first partition:

1.root

2.

|-- transactionId: integer (nullable = true)

3.

|-- predError: integer (nullable = true)

4.

|-- value: integer (nullable = true)

5.

|-- storeId: integer (nullable = true)

6.

|-- productId: integer (nullable = true)

7.

|-- f: integer (nullable = true)

Schema of second partition:

1.root

2.

|-- transactionId: integer (nullable = true)

3.

|-- predError: integer (nullable = true)

4.

|-- value: integer (nullable = true)

5.

|-- storeId: integer (nullable = true)

6.

|-- rollId: integer (nullable = true)

7.

|-- f: integer (nullable = true)

8.

|-- tax_id: integer (nullable = false)

A. spark.read.parquet(filePath, mergeSchema=\\'y\\')

B. spark.read.option("mergeSchema", "true").parquet(filePath)

C. spark.read.parquet(filePath)

D. 1.nx = 0 2.for file in dbutils.fs.ls(filePath):

3.

if not file.name.endswith(".parquet"):

4.

continue

5.

df_temp = spark.read.parquet(file.path)

6.

if nx == 0:

7.

df = df_temp

8.

else:

9.

df = df.union(df_temp)

10.

nx = nx+1

11.df

E. 1.nx = 0 2.for file in dbutils.fs.ls(filePath):

3.

if not file.name.endswith(".parquet"):

4.

continue

5.

df_temp = spark.read.parquet(file.path)

6.

if nx == 0:

7.

df = df_temp

8.

else:

9.

df = df.join(df_temp, how="outer")

10.

nx = nx+1

11.df

Correct Answer: B

QUESTION 7

Which of the following statements about DAGs is correct?

A. DAGs help direct how Spark executors process tasks, but are a limitation to the proper execution of a query when an executor fails.

B. DAG stands for "Directing Acyclic Graph".

C. Spark strategically hides DAGs from developers, since the high degree of automation in Spark means that developers never need to consider DAG layouts.

D. In contrast to transformations, DAGs are never lazily executed.

E. DAGs can be decomposed into tasks that are executed in parallel.

Correct Answer: E

QUESTION 8

Which of the following statements about garbage collection in Spark is incorrect?

A. Garbage collection information can be accessed in the Spark UI\\'s stage detail view.

B. Optimizing garbage collection performance in Spark may limit caching ability.

C. Manually persisting RDDs in Spark prevents them from being garbage collected.

D. In Spark, using the G1 garbage collector is an alternative to using the default Parallel garbage collector.

E. Serialized caching is a strategy to increase the performance of garbage collection.

Correct Answer: C

QUESTION 9

Which of the following describes the role of the cluster manager?

A. The cluster manager schedules tasks on the cluster in client mode.

B. The cluster manager schedules tasks on the cluster in local mode.

C. The cluster manager allocates resources to Spark applications and maintains the executor processes in client mode.

D. The cluster manager allocates resources to Spark applications and maintains the executor processes in remote mode.

E. The cluster manager allocates resources to the DataFrame manager.

Correct Answer: C

The cluster manager allocates resources to Spark applications and maintains the executor processes in client mode. Correct. In cluster mode, the cluster manager is located on a node other than the client machine. From there it starts and ends executor processes on the cluster nodes as required by the Spark application running on the Spark driver. The cluster manager allocates resources to Spark applications and maintains the executor processes in remote mode. Wrong, there is no "remote" execution mode in Spark. Available execution modes are local, client, and cluster. The cluster manager allocates resources to the DataFrame manager Wrong, there is no "DataFrame manager" in Spark. The cluster manager schedules tasks on the cluster in client mode. No, in client mode, the Spark driver schedules tasks on the cluster ?not the cluster manager. The cluster manager schedules tasks on the cluster in local mode. Wrong: In local mode, there is no "cluster". The Spark application is running on a single machine, not on a cluster of machines.

QUESTION 10

Which of the following code blocks reads the parquet file stored at filePath into DataFrame itemsDf, using a valid schema for the sample of itemsDf shown below?

Sample of itemsDf:

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

2.|itemId|attributes |supplier |

3.+------+-----------------------------+-------------------+

4.|1 |[blue, winter, cozy] |Sports Company Inc.|

5.|2 |[red, summer, fresh, cooling]|YetiX |

6.|3 |[green, summer, travel] |Sports Company Inc.|

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

A. 1.itemsDfSchema = StructType([

2.

StructField("itemId", IntegerType()),

3.

StructField("attributes", StringType()),

4.

StructField("supplier", StringType())])

5.

6.itemsDf = spark.read.schema(itemsDfSchema).parquet(filePath)

B. 1.itemsDfSchema = StructType([

2.

StructField("itemId", IntegerType),

3.

StructField("attributes", ArrayType(StringType)),

4.

StructField("supplier", StringType)])

5.

6.itemsDf = spark.read.schema(itemsDfSchema).parquet(filePath)

C. 1.itemsDf = spark.read.schema(\\'itemId integer, attributes , supplier string\\').parquet(filePath)

D. 1.itemsDfSchema = StructType([

2.

 StructField("itemId", IntegerType()),

3.

 StructField("attributes", ArrayType(StringType())),

4.

 StructField("supplier", StringType())])

5.

6.itemsDf = spark.read.schema(itemsDfSchema).parquet(filePath)

E. 1.itemsDfSchema = StructType([

2.

 StructField("itemId", IntegerType()),

3.

 StructField("attributes", ArrayType([StringType()])),

4.

 StructField("supplier", StringType())])

5.

6.itemsDf = spark.read(schema=itemsDfSchema).parquet(filePath)

Correct Answer: D

The challenge in this comes from there being an array variable in the schema. In addition, you should know how to pass a schema to the DataFrameReader that is invoked by spark.read. The correct way to define an array of strings in a schema is through ArrayType(StringType()). A schema can be passed to the DataFrameReader by simply appending schema(structType) to the read() operator. Alternatively, you can also define a schema as a string. For example, for the schema of itemsDf, the following string would make sense: itemId integer, attributes array, supplier string. A thing to keep in mind is that in schema definitions, you always need to instantiate the types, like so: StringType(). Just using StringType does not work in pySpark and will fail. Another concern with schemas is whether columns should be nullable, so allowed to have null values. In the case at hand, this is not a concern however, since the just asks for a "valid" schema. Both non-nullable and nullable column schemas would be valid here, since no null value appears in the DataFrame sample. More info: Learning Spark, 2nd Edition, Chapter 3 Static notebook | Dynamic notebook: See test 3, 19 (Databricks import instructions)

QUESTION 11

Which of the following code blocks returns a copy of DataFrame transactionsDf that only includes columns

transactionId, storeId, productId and f?

Sample of DataFrame transactionsDf:

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

2.|transactionId|predError|value|storeId|productId| f|

3.+-------------+---------+-----+-------+---------+----+

4.| 1| 3| 4| 25| 1|null|

5.| 2| 6| 7| 2| 2|null|

6.| 3| 3| null| 25| 3|null|

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

A. transactionsDf.drop(col("value"), col("predError"))

B. transactionsDf.drop("predError", "value")

C. transactionsDf.drop(value, predError)

D. transactionsDf.drop(["predError", "value"])

E. transactionsDf.drop([col("predError"), col("value")])

Correct Answer: B

QUESTION 12

Which of the following describes a valid concern about partitioning?

A. A shuffle operation returns 200 partitions if not explicitly set.

B. Decreasing the number of partitions reduces the overall runtime of narrow transformations if there are more executors available than partitions.

C. No data is exchanged between executors when coalesce() is run.

D. Short partition processing times are indicative of low skew.

E. The coalesce() method should be used to increase the number of partitions.

Correct Answer: A

QUESTION 13

The code block displayed below contains an error. The code block is intended to return all columns of DataFrame transactionsDf except for columns predError, productId, and value.

Find the error.

Excerpt of DataFrame transactionsDf:

transactionsDf.select(~col("predError"), ~col("productId"), ~col("value"))

A. The select operator should be replaced by the drop operator and the arguments to the drop operator should be column names predError, productId and value wrapped in the col operator so they should be expressed like drop(col(predError), col(productId), col(value)).

B. The select operator should be replaced with the deselect operator.

C. The column names in the select operator should not be strings and wrapped in the col operator, so they should be expressed like select(~col(predError), ~col(productId), ~col(value)).

D. The select operator should be replaced by the drop operator.

E. The select operator should be replaced by the drop operator and the arguments to the drop operator should be column names predError, productId and value as strings.

Correct Answer: E

Correct code block:

transactionsDf.drop("predError", "productId", "value") Static notebook | Dynamic notebook: See test 1, 37

(Databricks import instructions)

QUESTION 14

Which of the following code blocks immediately removes the previously cached DataFrame transactionsDf from memory and disk?

A. array_remove(transactionsDf, "*")

B. transactionsDf.unpersist()

C. del transactionsDf

D. transactionsDf.clearCache()

E. transactionsDf.persist()

Correct Answer: B

transactionsDf.unpersist()

Correct. The DataFrame.unpersist() command does exactly what the asks for - it removes all cached

parts of the DataFrame from memory and disk.

del transactionsDf

False. While this option can help remove the DataFrame from memory and disk, it does not do so

immediately. The reason is that this command just notifies the Python garbage collector that the

transactionsDf now may be deleted from memory. However, the garbage collector does not do so

immediately and, if you wanted it to run immediately, would need to be specifically triggered to do

so. Find more information linked below.

array_remove(transactionsDf, "*")

Incorrect. The array_remove method from pyspark.sql.functions is used for removing elements from arrays

in columns that match a specific condition. Also, the first argument would be a column, and

not a DataFrame as shown in the code block.

transactionsDf.persist()

No. This code block does exactly the opposite of what is asked for: It caches (writes) DataFrame

transactionsDf to memory and disk. Note that even though you do not pass in a specific storage

level here, Spark will use the default storage level (MEMORY_AND_DISK).

transactionsDf.clearCache()

Wrong. Spark\\'s DataFrame does not have a clearCache() method.

More info: pyspark.sql.DataFrame.unpersist -- PySpark 3.1.2 documentation, python - How to delete an

RDD in PySpark for the purpose of releasing resources? - Stack Overflow

Static notebook | Dynamic notebook: See test 3, 40 (Databricks import instructions)

QUESTION 15

Which of the following describes Spark\\'s standalone deployment mode?

A. Standalone mode uses a single JVM to run Spark driver and executor processes.

B. Standalone mode means that the cluster does not contain the driver.

C. Standalone mode is how Spark runs on YARN and Mesos clusters.

D. Standalone mode uses only a single executor per worker per application.

E. Standalone mode is a viable solution for clusters that run multiple frameworks, not only Spark.

Correct Answer: D

Standalone mode uses only a single executor per worker per application. This is correct and a limitation of Spark\\'s standalone mode. Standalone mode is a viable solution for clusters that run multiple frameworks. Incorrect. A limitation of standalone mode is that Apache Spark must be the only framework running on the cluster. If you would want to run multiple frameworks on the same cluster in parallel, for example Apache Spark and Apache Flink, you would consider the YARN deployment mode. Standalone mode uses a single JVM to run Spark driver and executor processes. No, this is what local mode does. Standalone mode is how Spark runs on YARN and Mesos clusters. No. YARN and Mesos modes are two deployment modes that are different from standalone mode. These modes allow Spark to run alongside other frameworks on a cluster. When Spark is run in standalone mode, only the Spark framework can run on the cluster. Standalone mode means that the cluster does not contain the driver. Incorrect, the cluster does not contain the driver in client mode, but in standalone mode the driver runs on a node in the cluster. More info: Learning Spark, 2nd Edition, Chapter 1