Practice Exam

Databricks Certified Associate Developer for Apache Spark 3.0 - Python

Overview

This is a practice exam for the Databricks Certified Associate Developer for Apache Spark 3.0 - Python exam. The questions here are retired questions from the actual exam that are representative of the questions one will receive while taking the actual exam. After taking this practice exam, one should know what to expect while taking the actual Associate Developer for Apache Spark 3.0 - Python exam.

Just like the actual exam, it contains 60 multiple-choice questions. Each of these questions has one correct answer. The correct answer for each question is listed at the bottom in the **Correct Answers** section.

There are a few more things to be aware of:

1. This practice exam is for the Python version of the actual exam, but it's incredibly similar to the Scala version of the actual exam, as well. There is a practice exam for the Scala version, too.
2. There is a two-hour time limit to take the actual exam.
3. In order to pass the actual exam, testers will need to correctly answer at least 42 of the 60 questions.
4. During the actual exam, testers will be able to reference a PDF version of the Apache Spark documentation. Please use this version of the documentation while taking this practice exam.
5. During the actual exam, testers will not be able to test code in a Spark session. Please do not use a Spark session when taking this practice exam.

6. These questions are representative of questions that are on the actual exam, but they are no longer on the actual exam.

If you have more questions, please review the Databricks Academy Certification FAQ.

Once you've completed the practice exam, evaluate your score using the correct answers at the bottom of this document. If you're ready to take the exam, head to Databricks Academy to register.

Exam Questions

Question 1

Which of the following statements about the Spark driver is incorrect?

A. The Spark driver is the node in which the Spark application's main method runs to coordinate the Spark application.
B. The Spark driver is horizontally scaled to increase overall processing throughput.
C. The Spark driver contains the SparkContext object.
D. The Spark driver is responsible for scheduling the execution of data by various worker nodes in cluster mode.
E. The Spark driver should be as close as possible to worker nodes for optimal performance.

Question 2

Which of the following describes nodes in cluster-mode Spark?

A. Nodes are the most granular level of execution in the Spark execution hierarchy.
B. There is only one node and it hosts both the driver and executors.
C. Nodes are another term for executors, so they are processing engine instances for performing computations.
D. There are driver nodes and worker nodes, both of which can scale horizontally.
E. Worker nodes are machines that host the executors responsible for the execution of tasks.

Question 3

Which of the following statements about slots is true?

A. There must be more slots than executors.
B. There must be more tasks than slots.
C. Slots are the most granular level of execution in the Spark execution hierarchy.
D. Slots are not used in cluster mode.
E. Slots are resources for parallelization within a Spark application.

Question 4

Which of the following is a combination of a block of data and a set of transformers that will run on a single executor?

A. Executor
B. Node
C. Job
D. Task
E. Slot

Question 5

Which of the following is a group of tasks that can be executed in parallel to compute the same set of operations on potentially multiple machines?

A. Job
B. Slot
C. Executor
D. Task
E. Stage

Question 6

Which of the following describes a shuffle?

A. A shuffle is the process by which data is compared across partitions.
B. A shuffle is the process by which data is compared across executors.
C. A shuffle is the process by which partitions are allocated to tasks.
D. A shuffle is the process by which partitions are ordered for write.
E. A shuffle is the process by which tasks are ordered for execution.

Question 7

DataFrame df is very large with a large number of partitions, more than there are executors in the cluster. Based on this situation, which of the following is incorrect? Assume there is one core per executor.

A. Performance will be suboptimal because not all executors will be utilized at the same time.
B. Performance will be suboptimal because not all data can be processed at the same time.
C. There will be a large number of shuffle connections performed on DataFrame df when operations inducing a shuffle are called.

D. There will be a lot of overhead associated with managing resources for data processing within each task.
E. There might be risk of out-of-memory errors depending on the size of the executors in the cluster.

Question 8

Which of the following operations will trigger evaluation?

A. `DataFrame.filter()`
B. `DataFrame.distinct()`
C. `DataFrame.intersect()`
D. `DataFrame.join()`
E. `DataFrame.count()`

Question 9

Which of the following describes the difference between transformations and actions?

A. Transformations work on DataFrames/Datasets while actions are reserved for native language objects.
B. There is no difference between actions and transformations.
C. Actions are business logic operations that do not induce execution while transformations are execution triggers focused on returning results.
D. Actions work on DataFrames/Datasets while transformations are reserved for native language objects.
E. Transformations are business logic operations that do not induce execution while actions are execution triggers focused on returning results.

Question 10

Which of the following DataFrame operations is always classified as a narrow transformation?

A. `DataFrame.sort()`
B. `DataFrame.distinct()`
C. `DataFrame.repartition()`
D. `DataFrame.select()`
E. `DataFrame.join()`

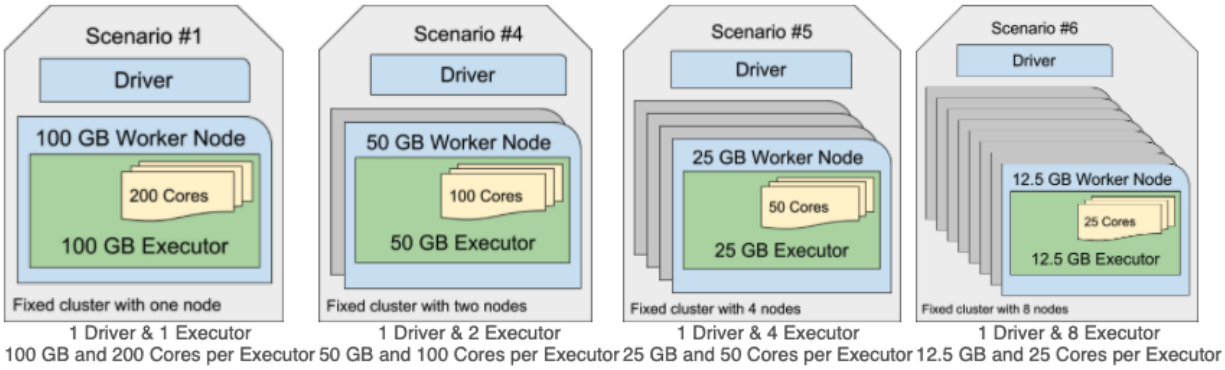
Question 11

Spark has a few different execution/deployment modes: cluster, client, and local. Which of the following describes Spark's execution/deployment mode?

A. Spark's execution/deployment mode determines where the driver and executors are physically located when a Spark application is run
B. Spark's execution/deployment mode determines which tasks are allocated to which executors in a cluster
C. Spark's execution/deployment mode determines which node in a cluster of nodes is responsible for running the driver program
D. Spark's execution/deployment mode determines exactly how many nodes the driver will connect to when a Spark application is run
E. Spark's execution/deployment mode determines whether results are run interactively in a notebook environment or in batch

Question 12

Which of the following cluster configurations will ensure the completion of a Spark application in light of a worker node failure?



Note: each configuration has roughly the same compute power using 100GB of RAM and 200 cores.

A. Scenario #1
B. They should all ensure completion because worker nodes are fault-tolerant.
C. Scenario #4
D. Scenario #5
E. Scenario #6

Question 13

Which of the following describes out-of-memory errors in Spark?

A. An out-of-memory error occurs when either the driver or an executor does not have enough memory to collect or process the data allocated to it.

B. An out-of-memory error occurs when Spark's storage level is too lenient and allows data objects to be cached to both memory and disk.
C. An out-of-memory error occurs when there are more tasks than are executors regardless of the number of worker nodes.
D. An out-of-memory error occurs when the Spark application calls too many transformations in a row without calling an action regardless of the size of the data object on which the transformations are operating.
E. An out-of-memory error occurs when too much data is allocated to the driver for computational purposes.

Question 14

Which of the following is the default storage level for `persist()` for a non-streaming DataFrame/Dataset?

A. `MEMORY_AND_DISK`
B. `MEMORY_AND_DISK_SER`
C. `DISK_ONLY`
D. `MEMORY_ONLY_SER`
E. `MEMORY_ONLY`

Question 15

Which of the following describes a broadcast variable?

A. A broadcast variable is a Spark object that needs to be partitioned onto multiple worker nodes because it's too large to fit on a single worker node.
B. A broadcast variable can only be created by an explicit call to the broadcast() operation.
C. A broadcast variable is entirely cached on the driver node so it doesn't need to be present on any worker nodes.
D. A broadcast variable is entirely cached on each worker node so it doesn't need to be shipped or shuffled between nodes with each stage.
E. A broadcast variable is saved to the disk of each worker node to be easily read into memory when needed.

Question 16

Which of the following operations is most likely to induce a skew in the size of your data's partitions?

A. `DataFrame.collect()`
B. `DataFrame.cache()`
C. `DataFrame.repartition(n)`

D. `DataFrame.coalesce(n)`
E. `DataFrame.persist()`

Question 17

Which of the following data structures are Spark DataFrames built on top of?

A. Arrays
B. Strings
C. RDDs
D. Vectors
E. SQL Tables

Question 18

Which of the following code blocks returns a DataFrame containing only column **`storeId`** and column **`division`** from DataFrame **`storesDF`**?

A. `storesDF.select("storeId").select("division")`
B. `storesDF.select(storeId, division)`
C. `storesDF.select("storeId", "division")`
D. `storesDF.select(col("storeId", "division"))`
E. `storesDF.select(storeId).select(division)`

Question 19

Which of the following code blocks returns a DataFrame containing all columns from DataFrame **`storesDF`** except for column sqft and column **`customerSatisfaction`**?

A sample of DataFrame **`storesDF`** is below:

storeId	open	openDate	division	sqft	numberOfEmployees	customerSatisfaction
0	true	1100746394	Utah	43161	61	71.1
1	true	944572255	West Virginia	18132	96	43.46
2	false	925495628	West Virginia	79520	45	36.93
3	true	1397353092	Texas	47751	78	47.19
4	true	986505057	Delaware	81483	95	25.24
...	...	...	...	...	...	...

A. `storesDF.drop("sqft", "customerSatisfaction")`

B. `storesDF.select("storeId", "open", "openDate", "division")`
C. `storesDF.select(-col(sqft), -col(customerSatisfaction))`
D. `storesDF.drop(sqft, customerSatisfaction)`
E. `storesDF.drop(col(sqft), col(customerSatisfaction))`

Question 20

The below code shown block contains an error. The code block is intended to return a DataFrame containing only the rows from DataFrame **`storesDF`** where the value in DataFrame **`storesDF`**'s "sqft" column is less than or equal to 25,000. Assume DataFrame **`storesDF`** is the only defined language variable. Identify the error.

Code block:

```
storesDF.filter(sqft <= 25000)
```

A. The column name **`sqft`** needs to be quoted like **`storesDF.filter("sqft" <= 25000)`**.
B. The column name **`sqft`** needs to be quoted and wrapped in the **`col()`** function like **`storesDF.filter(col("sqft") <= 25000)`**.
C. The sign in the logical condition inside **`filter()`** needs to be changed from **`<=`** to **`>`**.
D. The sign in the logical condition inside **`filter()`** needs to be changed from **`<=`** to **`>=`**.
E. The column name sqft needs to be wrapped in the **`col()`** function like **`storesDF.filter(col(sqft) <= 25000)`**.

Question 21

The code block shown below should return a DataFrame containing only the rows from DataFrame **`storesDF`** where the value in column **`sqft`** is less than or equal to 25,000 OR the value in column **`customerSatisfaction`** is greater than or equal to 30. Choose the response that correctly fills in the numbered blanks within the code block to complete this task.

Code block:

```
storesDF.__1__(__2__ __3__ __4__)
```

A.

```
1. filter
2. (col("sqft") <= 25000)
3. |
4. (col("customerSatisfaction") >= 30)
```

B.

```
1. drop
2. (col(sqft) <= 25000)
3. |
4. (col(customerSatisfaction) >= 30)
```

C.

```
1. filter
2. col("sqft") <= 25000
3. |
4. col("customerSatisfaction") >= 30
```

D.

```
1. filter
2. col("sqft") <= 25000
3. or
4. col("customerSatisfaction") >= 30
```

E.

```
1. filter
2. (col("sqft") <= 25000)
3. or
4. (col("customerSatisfaction") >= 30)
```

Question 22

Which of the following operations can be used to convert a DataFrame column from one type to another type?

A. `col().cast()`
B. `convert()`
C. `castAs()`
D. `col().coerce()`
E. `col()`

Question 23

Which of the following code blocks returns a new DataFrame with a new column **`sqft100`** that is 1/100th of column **`sqft`** in DataFrame **`storesDF`**? Note that column **`sqft100`** is not in the original DataFrame **`storesDF`**.

A. `storesDF.withColumn("sqft100", col("sqft") * 100)`
B. `storesDF.withColumn("sqft100", sqft / 100)`
C. `storesDF.withColumn(col("sqft100"), col("sqft") / 100)`
D. `storesDF.withColumn("sqft100", col("sqft") / 100)`
E. `storesDF.newColumn("sqft100", sqft / 100)`

Question 24

Which of the following code blocks returns a new DataFrame from DataFrame **`storesDF`** where column **`numberOfManagers`** is the constant integer **`1`**?

A. **`storesDF.withColumn("numberOfManagers", col(1))`**
B. **`storesDF.withColumn("numberOfManagers", 1)`**
C. **`storesDF.withColumn("numberOfManagers", lit(1))`**
D. **`storesDF.withColumn("numberOfManagers", lit("1"))`**
E. **`storesDF.withColumn("numberOfManagers", IntegerType(1))`**

Question 25

The code block shown below contains an error. The code block intends to return a new DataFrame where column **`storeCategory`** from DataFrame **`storesDF`** is split at the underscore character into column **`storeValueCategory`** and column **`storeSizeCategory`**. Identify the error.

A sample of DataFrame **`storesDF`** is displayed below:

storeId	open	openDate	storeCategory
0	true	1100746394	VALUE_MEDIUM
1	true	944572255	MAINSTREAM_SMALL
2	false	925495628	PREMIUM_LARGE
3	true	1397353092	VALUE_MEDIUM
4	true	986505057	VALUE_LARGE
5	true	955988614	PREMIUM_LARGE
...	...	...	...

Code block:

```
(storesDF.withColumn(
        "storeValueCategory", col("storeCategory").split("_")[0]
    ).withColumn(
        "storeSizeCategory", col("storeCategory").split("_")[1]
    )
)
```

A. The **`split()`** operation comes from the imported functions object. It accepts a string column name and split character as arguments. It is not a method of a Column object.

B. The **`split()`** operation comes from the imported functions object. It accepts a Column object and split character as arguments. It is not a method of a Column object.
C. The index values of 0 and 1 should be provided as second arguments to the **`split()`** operation rather than indexing the result.
D. The index values of 0 and 1 are not correct — they should be 1 and 2, respectively.
E. The **`withColumn()`** operation cannot be called twice in a row.

Question 26

Which of the following operations can be used to split an array column into an individual DataFrame row for each element in the array?

A. **`extract()`**
B. **`split()`**
C. **`explode()`**
D. **`arrays_zip()`**
E. **`unpack()`**

Question 27

Which of the following code blocks returns a new DataFrame where column **`storeCategory`** is an all-lowercase version of column **`storeCategory`** in DataFrame **`storesDF`**? Assume DataFrame **`storesDF`** is the only defined language variable.

A.
```
storesDF.withColumn("storeCategory",
lower(col("storeCategory")))
```
B.
```
storesDF.withColumn("storeCategory",
col("storeCategory").lower())
```
C.
```
storesDF.withColumn("storeCategory",
tolower(col("storeCategory")))
```
D. `storesDF.withColumn("storeCategory", lower("storeCategory"))`
E. `storesDF.withColumn("storeCategory", lower(storeCategory))`

Question 28

The code block shown below contains an error. The code block is intended to return a new DataFrame where column **`division`** from DataFrame **`storesDF`** has been renamed to column **`state`** and column **`managerName`** from DataFrame **`storesDF`** has been renamed to column **`managerFullName`**. Identify the error.

Code block:

```
(storesDF.withColumnRenamed("state", "division")
         .withColumnRenamed("managerFullName", "managerName"))
```

A. Both arguments to operation **`withColumnRenamed()`** should be wrapped in the **`col()`** operation.
B. The operations **`withColumnRenamed()`** should not be called twice, and the first argument should be **`["state", "division"]`** and the second argument should be **`["managerFullName", "managerName"]`**.
C. The old columns need to be explicitly dropped.
D. The first argument to operation **`withColumnRenamed()`** should be the old column name and the second argument should be the new column name.
E. The operation **`withColumnRenamed()`** should be replaced with **`withColumn()`**.

Question 29

Which of the following code blocks returns a DataFrame where rows in DataFrame **`storesDF`** containing missing values in every column have been dropped?

A. `storesDF.nadrop("all")`
B. `storesDF.na.drop("all", subset = "sqft")`
C. `storesDF.dropna()`
D. `storesDF.na.drop()`
E. `storesDF.na.drop("all")`

Question 30

Which of the following operations fails to return a DataFrame where every row is unique?

A. `DataFrame.distinct()`
B. `DataFrame.drop_duplicates(subset = None)`
C. `DataFrame.drop_duplicates()`
D. `DataFrame.dropDuplicates()`
E. `DataFrame.drop_duplicates(subset = "all")`

Question 31

Which of the following code blocks will not always return the exact number of distinct values in column division?

A. `storesDF.agg(approx_count_distinct(col("division")).alias("divisionDistinct"))`
B. `storesDF.agg(approx_count_distinct(col("division"), 0).alias("divisionDistinct"))`
C. `storesDF.agg(countDistinct(col("division")).alias("divisionDistinct"))`
D. `storesDF.select("division").dropDuplicates().count()`
E. `storesDF.select("division").distinct().count()`

Question 32

The code block shown below should return a new DataFrame with the mean of column **`sqft`** from DataFrame **`storesDF`** in column **`sqftMean`**. Choose the response that correctly fills in the numbered blanks within the code block to complete this task.

Code block:

```
storesDF.__1__(__2__(__3__).alias("sqftMean"))
```

A.
1. `agg`
2. `mean`
3. `col("sqft")`

B.
1. `mean`
2. `col`
3. `"sqft"`

C.
1. `withColumn`
2. `mean`
3. `col("sqft")`

D.
1. `agg`
2. `mean`
3. `"sqft"`

E.
1. `agg`
2. `average`
3. `col("sqft")`

Question 33

Which of the following code blocks returns the number of rows in DataFrame **`storesDF`**?

A. `storesDF.withColumn("numberOfRows", count())`
B. `storesDF.withColumn(count().alias("numberOfRows"))`
C. `storesDF.countDistinct()`
D. `storesDF.count()`
E. `storesDF.agg(count())`

Question 34

Which of the following code blocks returns the sum of the values in column **`sqft`** in DataFrame **`storesDF`** grouped by distinct value in column **`division`**?

A. `storesDF.groupBy.agg(sum(col("sqft")))`
B. `storesDF.groupBy("division").agg(sum())`
C. `storesDF.agg(groupBy("division").sum(col("sqft")))`
D. `storesDF.groupby.agg(sum(col("sqft")))`
E. `storesDF.groupBy("division").agg(sum(col("sqft")))`

Question 35

Which of the following code blocks returns a DataFrame containing summary statistics only for column **`sqft`** in DataFrame **`storesDF`**?

A. `storesDF.summary("mean")`
B. `storesDF.describe("sqft")`
C. `storesDF.summary(col("sqft"))`
D. `storesDF.describeColumn("sqft")`
E. `storesDF.summary()`

Question 36

Which of the following operations can be used to sort the rows of a DataFrame?

A. `sort()` and `orderBy()`
B. `orderby()`
C. `sort()` and `orderby()`
D. `orderBy()`
E. `sort()`

Question 37

The code block shown below contains an error. The code block is intended to return a 15 percent sample of rows from DataFrame **`storesDF`** without replacement. Identify the error.

Code block:

storesDF.sample(True, fraction = 0.15)

A. There is no argument specified to the **`seed`** parameter.

B. There is no argument specified to the **`withReplacement`** parameter.
C. The **`sample()`** operation does not sample without replacement — **`sampleby()`** should be used instead.
D. The **`sample()`** operation is not reproducible.
E. The first argument **`True`** sets the sampling to be with replacement.

Question 38

Which of the following operations can be used to return the top **`n`** rows from a DataFrame?

A. `DataFrame.n()`
B. `DataFrame.take(n)`
C. `DataFrame.head`
D. `DataFrame.show(n)`
E. `DataFrame.collect(n)`

Question 39

The code block shown below should extract the value for column **`sqft`** from the first row of DataFrame **`storesDF`**. Choose the response that correctly fills in the numbered blanks within the code block to complete this task.

Code block:

`__1__.__2__.__3__`

A.
1. `storesDF`
2. `first`
3. `col("sqft")`

B.
1. `storesDF`
2. `first`
3. `sqft`

C.
1. `storesDF`
2. `first`
3. `["sqft"]`

D.
1. `storesDF`
2. `first()`

3. `sqft`

E.

1. `storesDF`
2. `first()`
3. `col("sqft")`

Question 40

Which of the following lines of code prints the schema of a DataFrame?

A. `print(storesDF)`
B. `storesDF.schema`
C. `print(storesDF.schema())`
D. `DataFrame.printSchema()`
E. `DataFrame.schema()`

Question 41

In what order should the below lines of code be run in order to create and register a SQL UDF named **`"ASSESS_PERFORMANCE"`** using the Python function **`assessPerformance`** and apply it to column **`customerSatistfaction`** in table **`stores`**?

Lines of code:

1. `spark.udf.register("ASSESS_PERFORMANCE", assessPerformance)`
2. `spark.sql("SELECT customerSatisfaction, assessPerformance(customerSatisfaction) AS result FROM stores")`
3. `spark.udf.register(assessPerformance, "ASSESS_PERFORMANCE")`
4. `spark.sql("SELECT customerSatisfaction, ASSESS_PERFORMANCE(customerSatisfaction) AS result FROM stores")`

A. 3, 4
B. 1, 4
C. 3, 2
D. 2
E. 1, 2

Question 42

In what order should the below lines of code be run in order to create a Python UDF **`assessPerformanceUDF()`** using the integer-returning Python function **`assessPerformance`** and apply it to column **`customerSatisfaction`** in DataFrame **`storesDF`**?

Lines of code:

1. `assessPerformanceUDF = udf(assessPerformance, IntegerType)`
2. `assessPerformanceUDF = spark.register.udf("ASSESS_PERFORMANCE", assessPerformance)`
3. `assessPerformanceUDF = udf(assessPerformance, IntegerType())`
4. `storesDF.withColumn("result", assessPerformanceUDF(col("customerSatisfaction")))`
5. `storesDF.withColumn("result", assessPerformance(col("customerSatisfaction")))`
6. `storesDF.withColumn("result", ASSESS_PERFORMANCE(col("customerSatisfaction")))`

A. 3, 4
B. 2, 6
C. 3, 5
D. 1, 4
E. 2, 5

Question 43

Which of the following operations can execute a SQL query on a table?

A. `spark.query()`
B. `DataFrame.sql()`
C. `spark.sql()`
D. `DataFrame.createOrReplaceTempView()`
E. `DataFrame.createTempView()`

Question 44

Which of the following code blocks creates a single-column DataFrame from Python list years which is made up of integers?

A. `spark.createDataFrame([years], IntegerType())`
B. `spark.createDataFrame(years, IntegerType())`
C. `spark.DataFrame(years, IntegerType())`

D. `spark.createDataFrame(years)`
E. `spark.createDataFrame(years, IntegerType)`

Question 45

Which of the following operations can be used to cache a DataFrame only in Spark's memory assuming the default arguments can be updated?

A. `DataFrame.clearCache()`
B. `DataFrame.storageLevel`
C. `StorageLevel`
D. `DataFrame.persist()`
E. `DataFrame.cache()`

Question 46

The code block shown below contains an error. The code block is intended to return a new 4-partition DataFrame from the 8-partition DataFrame **`storesDF`** without inducing a shuffle. Identify the error.

Code block:

```
storesDF.repartition(4)
```

A. The **`repartition`** operation will only work if the DataFrame has been cached to memory.
B. The **`repartition`** operation requires a column on which to partition rather than a number of partitions.
C. The number of resulting partitions, 4, is not achievable for an 8-partition DataFrame.
D. The **`repartition`** operation induced a full shuffle. The **`coalesce`** operation should be used instead.
E. The **`repartition`** operation cannot guarantee the number of result partitions.

Question 47

Which of the following code blocks will always return a new 12-partition DataFrame from the 8-partition DataFrame **`storesDF`**?

A. `storesDF.coalesce(12)`
B. `storesDF.repartition()`
C. `storesDF.repartition(12)`
D. `storesDF.coalesce()`
E. `storesDF.coalesce(12, "storeId")`

Question 48

Which of the following Spark config properties represents the number of partitions used in wide transformations like **`join()`**?

A. `spark.sql.shuffle.partitions`
B. `spark.shuffle.partitions`
C. `spark.shuffle.io.maxRetries`
D. `spark.shuffle.file.buffer`
E. `spark.default.parallelism`

Question 49

In what order should the below lines of code be run in order to return a DataFrame containing a column **`openDateString`**, a string representation of Java's SimpleDateFormat?

Note that column **`openDate`** is of type **`integer`** and represents a date in the UNIX epoch format — the number of seconds since midnight on January 1st, 1970.

An example of Java's SimpleDateFormat is **`"Sunday, Dec 4, 2008 1:05 PM"`**.

A sample of **`storesDF`** is displayed below:

storeId	openDate
0	1100746394
1	1474410343
2	1116610009
3	1180035265
4	1408024997
...	...

Lines of code:

1. `storesDF.withColumn("openDateString", from_unixtime(col("openDate"), simpleDateFormat))`
2. `simpleDateFormat = "EEEE, MMM d, yyyy h:mm a"`
3. `storesDF.withColumn("openDateString", from_unixtime(col("openDate"), SimpleDateFormat()))`
4. `storesDF.withColumn("openDateString", date_format(col("openDate"), simpleDateFormat))`

5. `storesDF.withColumn("openDateString", date_format(col("openDate"), SimpleDateFormat()))`
6. `simpleDateFormat = "wd, MMM d, yyyy h:mm a"`

A. 2, 3
B. 2, 1
C. 6, 5
D. 2, 4
E. 6, 1

Question 50

Which of the following code blocks returns a DataFrame containing a column **`month`**, an integer representation of the month from column **`openDate`** from DataFrame **`storesDF`**?

Note that column **`openDate`** is of type integer and represents a date in the UNIX epoch format — the number of seconds since midnight on January 1st, 1970.

A sample of **`storesDF`** is displayed below:

storeId	openDate
0	1100746394
1	1474410343
2	1116610009
3	1180035265
4	1408024997
...	...

A. `storesDF.withColumn("month", getMonth(col("openDate")))`
B. `storesDF.withColumn("openTimestamp", col("openDate").cast("Timestamp")).withColumn("month", month(col("openTimestamp")))`
C. `storesDF.withColumn("openDateFormat", col("openDate").cast("Date")).withColumn("month", month(col("openDateFormat")))`
D. `storesDF.withColumn("month", substr(col("openDate"), 4, 2))`
E. `storesDF.withColumn("month", month(col("openDate")))`

Question 51

Which of the following operations performs an inner join on two DataFrames?

A. `DataFrame.innerJoin()`
B. `DataFrame.join()`
C. Standalone `join()` function
D. `DataFrame.merge()`
E. `DataFrame.crossJoin()`

Question 52

Which of the following code blocks returns a new DataFrame that is the result of an outer join between DataFrame **storesDF** and DataFrame **employeesDF** on column **storeId**?

A. `storesDF.join(employeesDF, "storeId", "outer")`
B. `storesDF.join(employeesDF, "storeId")`
C. `storesDF.join(employeesDF, "outer", col("storeId"))`
D. `storesDF.join(employeesDF, "outer", storesDF.storeId == employeesDF.storeId)`
E. `storesDF.merge(employeesDF, "outer", col("storeId"))`

Question 53

The below code block contains an error. The code block is intended to return a new DataFrame that is the result of an inner join between DataFrame **`storesDF`** and DataFrame **`employeesDF`** on column **`storeId`** and column **`employeeId`** which are in both DataFrames. Identify the error.

Code block:

```
storesDF.join(employeesDF, [col("storeId"), col("employeeId")])
```

A. The **`join()`** operation is a standalone function rather than a method of DataFrame — the **`join()`** operation should be called where its first two arguments are **`storesDF`** and **`employeesDF`**.
B. There must be a third argument to **`join()`** because the default to the how parameter is not **`"inner"`**.
C. The **`col("storeId")`** and **`col("employeeId")`** arguments should not be separate elements of a list — they should be tested to see if they're equal to one another like **`col("storeId") == col("employeeId")`**.
D. There is no **`DataFrame.join()`** operation — **`DataFrame.merge()`** should be used instead.

E. The references to **`"storeId"`** and **`"employeeId"`** should not be inside the **`col()`** function — removing the **`col()`** function should result in a successful join.

Question 54

Which of the following Spark properties is used to configure the broadcasting of a DataFrame without the use of the **`broadcast()`** operation?

A. `spark.sql.autoBroadcastJoinThreshold`
B. `spark.sql.broadcastTimeout`
C. `spark.broadcast.blockSize`
D. `spark.broadcast.compress`
E. `spark.executor.memoryOverhead`

Question 55

The code block shown below should return a new DataFrame that is the result of a cross join between DataFrame **`storesDF`** and DataFrame **`employeesDF`**. Choose the response that correctly fills in the numbered blanks within the code block to complete this task.

Code block:

```
__1__.__2__(__3__)
```

A.
1. `storesDF`
2. `crossJoin`
3. `employeesDF, "storeId"`

B.
1. `storesDF`
2. `join`
3. `employeesDF, "cross"`

C.
1. `storesDF`
2. `crossJoin`
3. `employeesDF, "storeId"`

D.
1. `storesDF`
2. `join`
3. `employeesDF, "storeId", "cross"`

E.

1. `storesDF`
2. `crossJoin`
3. `employeesDF`

Question 56

Which of the following operations performs a position-wise union on two DataFrames?

A. The standalone **`concat()`** function
B. The standalone **`unionAll()`** function
C. The standalone **`union()`** function
D. **`DataFrame.unionByName()`**
E. **`DataFrame.union()`**

Question 57

Which of the following code blocks writes DataFrame **`storesDF`** to file path **`filePath`** as parquet?

A. `storesDF.write.option("parquet").path(filePath)`
B. `storesDF.write.path(filePath)`
C. `storesDF.write().parquet(filePath)`
D. `storesDF.write(filePath)`
E. `storesDF.write.parquet(filePath)`

Question 58

The code block shown below contains an error. The code block is intended to write DataFrame **`storesDF`** to file path **`filePath`** as parquet and partition by values in column **`division`**. Identify the error.

Code block:

```
storesDF.write.repartition("division").parquet(filePath)
```

A. The argument **`division`** to operation **`repartition()`** should be wrapped in the **`col()`** function to return a Column object.
B. There is no **`parquet()`** operation for DataFrameWriter — the **`save()`** operation should be used instead.
C. There is no **`repartition()`** operation for DataFrameWriter — the **`partitionBy()`** operation should be used instead.

D. **`DataFrame.write`** is an operation — it should be followed by parentheses to return a DataFrameWriter.
E. The **`mode()`** operation must be called to specify that this write should not overwrite existing files.

Question 59

Which of the following code blocks reads a parquet at the file path **`filePath`** into a DataFrame?

A. `spark.read().parquet(filePath)`
B. `spark.read().path(filePath, source = "parquet")`
C. `spark.read.path(filePath, source = "parquet")`
D. `spark.read.parquet(filePath)`
E. `spark.read().path(filePath)`

Question 60

Which of the following code blocks reads JSON at the file path **`filePath`** into a DataFrame with the specified schema **`schema`**?

A. `spark.read().schema(schema).format(json).load(filePath)`
B. `spark.read().schema(schema).format("json").load(filePath)`
C. `spark.read.schema("schema").format("json").load(filePath)`
D. `spark.read.schema("schema").format("json").load(filePath)`
E. `spark.read.schema(schema).format("json").load(filePath)`

Correct Answers

1. B
2. E
3. E
4. D
5. E
6. A
7. A
8. E
9. E
10. D
11. A
12. B
13. A

14. A
15. D
16. D
17. C
18. C
19. A
20. B
21. A
22. A
23. D
24. C
25. B
26. C
27. A
28. D
29. E
30. E
31. A
32. A
33. D
34. E
35. B
36. A
37. E
38. B
39. D
40. D
41. B
42. A
43. C
44. B
45. D
46. D
47. C
48. A
49. B
50. B
51. B
52. A
53. E
54. A
55. E

56. E
57. E
58. C
59. D
60. E