Free Questions for Databricks-Certified-Associate-Developer-for-Apache-Spark-3.0 by certsinside

Shared by Hyde on 20-10-2022

For More Free Questions and Preparation Resources

Check the Links on Last Page

Question 1

Question Type: MultipleChoice

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)

Options:

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.

Answer:

D

Explanation:

Correct code block:

schema = StructType([

StructField('itemId', IntegerType(), True),

StructField('attributes', ArrayType(StringType(), True), True),

StructField('supplier', StringType(), True)

])

spark.read.options(modifiedBefore='2029-03-20T05:44:46').schema(schema).parquet(filePath)

This Question: is more difficult than what you would encounter in the exam. In the exam, for this Question: type, only one error needs to be identified and not 'one or multiple' as in the

question.

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

Correct! Columns in the schema definition should use the StructField type. Building a schema from pyspark.sql.types, as here using classes like StructType and StructField, is one of multiple ways

of expressing a schema in Spark. A StructType always contains a list of StructFields (see documentation linked below). So, nesting StructType and StructType as shown in the Question: is

wrong.

The modification date threshold should be specified by a keyword argument like options(modifiedBefore='2029-03-20T05:44:46') and not two consecutive non-keyword arguments as in the original

code block (see documentation linked below).

Spark cannot identify the file format correctly, because either it has to be specified by using the DataFrameReader.format(), as an argument to DataFrameReader.load(), or directly by calling, for

example, DataFrameReader.parquet().

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

No. If StructField would be used for the columns instead of StructType (see above), the third argument specified whether the column is nullable. The original schema shows that columns should be

nullable and this is specified correctly by the third argument being True in the schema in the code block.

It is correct, however, that the modification date threshold is specified incorrectly (see above).

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

Wrong. The attributes array is specified correctly, following the syntax for ArrayType (see linked documentation below). That Spark cannot identify the file format is correct, see correct answer

above. In addition, the DataFrameReader is called correctly through the SparkSession spark.

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

Incorrect, the object types in the schema definition are correct and syntax of the call to Spark's DataFrameReader is correct.

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

False. The data type of the schema is StructType and an accepted data type for the DataFrameReader.schema() method. It is correct however that the modification date threshold is specified

incorrectly (see correct answer above).

Question 2

Question Type: **MultipleChoice**

The code block shown below should return a DataFrame with only columns from DataFrame transactionsDf for which there is a corresponding transactionId in DataFrame itemsDf. DataFrame

itemsDf is very small and much smaller than DataFrame transactionsDf. The query should be executed in an optimized way. Choose the answer that correctly fills the blanks in the code block to

accomplish this.

__1__.__2__(__3__, __4__, __5__)

Options:

A- 1. transactionsDf
2. join
3. broadcast(itemsDf)
4. transactionsDf.transactionId==itemsDf.transactionId
5. 'outer'

B- 1. transactionsDf
2. join
3. itemsDf
4. transactionsDf.transactionId==itemsDf.transactionId
5. 'anti'

C- 1. transactionsDf
2. join
3. broadcast(itemsDf)
4. 'transactionId'
5. 'left_semi'

D- 1. itemsDf
2. broadcast
3. transactionsDf
4. 'transactionId'
5. 'left_semi'

E- 1. itemsDf
2. join

3. broadcast(transactionsDf)
4. 'transactionId'
5. 'left_semi'

Answer:

C

Explanation:

Correct code block:

transactionsDf.join(broadcast(itemsDf), 'transactionId', 'left_semi')

This Question: is extremely difficult and exceeds the difficulty of questions in the exam by far.

A first indication of what is asked from you here is the remark that 'the query should be executed in an optimized way'. You also have qualitative information about the size of itemsDf and

transactionsDf. Given that itemsDf is 'very small' and that the execution should be optimized, you should consider instructing Spark to perform a broadcast join, broadcasting the 'very small'

DataFrame itemsDf to all executors. You can explicitly suggest this to Spark via wrapping itemsDf into a broadcast() operator. One answer option does not include this operator, so you can disregard

it. Another answer option wraps the broadcast() operator around transactionsDf - the bigger of the two DataFrames. This answer option does not make sense in the optimization context and can

likewise be disregarded.

When thinking about the broadcast() operator, you may also remember that it is a method of pyspark.sql.functions. One answer option, however, resolves to itemsDf.broadcast([...]). The DataFrame

class has no broadcast() method, so this answer option can be eliminated as well.

All two remaining answer options resolve to transactionsDf.join([...]) in the first 2 gaps, so you will have to figure out the details of the join now. You can pick between an outer and a left semi join. An

outer join would include columns from both DataFrames, where a left semi join only includes columns from the 'left' table, here transactionsDf, just as asked for by the question. So, the correct

answer is the one that uses the left_semi join.

Question 3

Question Type: MultipleChoice

The code block shown below should show information about the data type that column storeId of DataFrame transactionsDf contains. Choose the answer that correctly fills the blanks in the code

block to accomplish this.

Code block:

transactionsDf.__1__(__2__).__3__

Options:

A- 1. select
2. 'storeId'
3. print_schema()

B- 1. limit
2. 1
3. columns

C- 1. select
2. 'storeId'
3. printSchema()

D- 1. limit
2. 'storeId'
3. printSchema()

E- 1. select
2. storeId
3. dtypes

Answer:

B

Explanation:

Correct code block:

transactionsDf.select('storeId').printSchema()

The difficulty of this Question: is that it is hard to solve with the stepwise first-to-last-gap approach that has worked well for similar questions, since the answer options are so different from

one

another. Instead, you might want to eliminate answers by looking for patterns of frequently wrong answers.

A first pattern that you may recognize by now is that column names are not expressed in quotes. For this reason, the answer that includes storeId should be eliminated.

By now, you may have understood that the DataFrame.limit() is useful for returning a specified amount of rows. It has nothing to do with specific columns. For this reason, the answer that resolves to

limit('storeId') can be eliminated.

Given that we are interested in information about the data type, you should Question: whether the answer that resolves to limit(1).columns provides you with this information. While

DataFrame.columns is a valid call, it will only report back column names, but not column types. So, you can eliminate this option.

The two remaining options either use the printSchema() or print_schema() command. You may remember that DataFrame.printSchema() is the only valid command of the two. The select('storeId')

part just returns the storeId column of transactionsDf - this works here, since we are only interested in that column's type anyways.

More info: pyspark.sql.DataFrame.printSchema --- PySpark 3.1.2 documentation

Static notebook | Dynamic notebook: See test 3, Question: 57 (Databricks import instructions)

Question 4

Question Type: **MultipleChoice**

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?

Options:

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')

Answer:

A

Explanation:

itemsDf.write.mode('overwrite').parquet(filePath)

Correct! itemsDf.write returns a pyspark.sql.DataFrameWriter instance whose overwriting behavior can be modified via the mode setting or by passing mode='overwrite' to the parquet() command.

Although the parquet format is not prescribed for solving this question, parquet() is a valid operator to initiate Spark to write the data to disk.

itemsDf.write.mode('overwrite').path(filePath)

No. A pyspark.sql.DataFrameWriter instance does not have a path() method.

itemsDf.write.option('parquet').mode('overwrite').path(filePath)

Incorrect, see above. In addition, a file format cannot be passed via the option() method.

itemsDf.write(filePath, mode='overwrite')

Wrong. Unfortunately, this is too simple. You need to obtain access to a DataFrameWriter for the DataFrame through calling itemsDf.write upon which you can apply further methods to control how

Spark data should be written to disk. You cannot, however, pass arguments to itemsDf.write directly.

itemsDf.write().parquet(filePath, mode='overwrite')

False. See above.

More info: pyspark.sql.DataFrameWriter.parquet --- PySpark 3.1.2 documentation

Static notebook | Dynamic notebook: See test 3, Question: 56 (Databricks import instructions)

Question 5

Question Type: **MultipleChoice**

Which of the following code blocks displays various aggregated statistics of all columns in DataFrame transactionsDf, including the standard deviation and minimum of values in each column?

Options:

A- transactionsDf.summary()

B- transactionsDf.agg('count', 'mean', 'stddev', '25%', '50%', '75%', 'min')

C- transactionsDf.summary('count', 'mean', 'stddev', '25%', '50%', '75%', 'max').show()

D- transactionsDf.agg('count', 'mean', 'stddev', '25%', '50%', '75%', 'min').show()

E- transactionsDf.summary().show()

Answer:

E

Explanation:

The DataFrame.summary() command is very practical for quickly calculating statistics of a DataFrame. You need to call .show() to display the results of the calculation. By default, the command

calculates various statistics (see documentation linked below), including standard deviation and minimum. Note that the answer that lists many options in the summary() parentheses does not

include the minimum, which is asked for in the question.

Answer options that include agg() do not work here as shown, since DataFrame.agg() expects more complex, column-specific instructions on how to aggregate values.

More info:

- pyspark.sql.DataFrame.summary --- PySpark 3.1.2 documentation

- pyspark.sql.DataFrame.agg --- PySpark 3.1.2 documentation

Static notebook | Dynamic notebook: See test 3, Question: 46 (Databricks import instructions)

Question 6

Question Type: **MultipleChoice**

The code block displayed below contains an error. The code block should merge the rows of DataFrames transactionsDfMonday and transactionsDfTuesday into a new DataFrame, matching

column names and inserting null values where column names do not appear in both DataFrames. Find the error.

Sample of DataFrame transactionsDfMonday:

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

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

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

4. | 5| null| null| null| 2|null|

5. | 6| 3| 2| 25| 2|null|

6. +-------------+---------+-----+-------+---------+----+

Sample of DataFrame transactionsDfTuesday:

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

2. |storeId|transactionId|productId|value|

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

4. | 25| 1| 1| 4|

5. | 2| 2| 2| 7|

6. | 3| 4| 2| null|

7. | null| 5| 2| null|

8. +-------+-------------+---------+-----+

Code block:

```
sc.union([transactionsDfMonday, transactionsDfTuesday])
```

Options:

A- The DataFrames' RDDs need to be passed into the sc.union method instead of the DataFrame variable names.

B- Instead of union, the concat method should be used, making sure to not use its default arguments.

C- Instead of the Spark context, transactionDfMonday should be called with the join method instead of the union method, making sure to use its default arguments.

D- Instead of the Spark context, transactionDfMonday should be called with the union method.

E- Instead of the Spark context, transactionDfMonday should be called with the unionByName method instead of the union method, making sure to not use its default arguments.

Answer:

E

Explanation:

Correct code block:

transactionsDfMonday.unionByName(transactionsDfTuesday, True)

Output of correct code block:

+-------------+---------+-----+-------+---------+----+

|transactionId|predError|value|storeId|productId| f|

+-------------+---------+-----+-------+---------+----+

| 5| null| null| null| 2|null|

| 6| 3| 2| 25| 2|null|

| 1| null| 4| 25| 1|null|

| 2| null| 7| 2| 2|null|

| 4| null| null| 3| 2|null|

| 5| null| null| null| 2|null|

+-------------+---------+-----+-------+---------+----+

For solving this question, you should be aware of the difference between the DataFrame.union() and DataFrame.unionByName() methods. The first one matches columns independent of their

names, just by their order. The second one matches columns by their name (which is asked for in the question). It also has a useful optional argument, allowMissingColumns. This allows you to

merge DataFrames that have different columns - just like in this example.

sc stands for SparkContext and is automatically provided when executing code on Databricks. While sc.union() allows you to join RDDs, it is not the right choice for joining DataFrames. A hint away

from sc.union() is given where the Question: talks about joining 'into a new DataFrame'.

concat is a method in pyspark.sql.functions. It is great for consolidating values from different columns, but has no place when trying to join rows of multiple DataFrames.

Finally, the join method is a contender here. However, the default join defined for that method is an inner join which does not get us closer to the goal to match the two DataFrames as instructed,

especially given that with the default arguments we cannot define a join condition.

More info:

- pyspark.sql.DataFrame.unionByName --- PySpark 3.1.2 documentation

- pyspark.SparkContext.union --- PySpark 3.1.2 documentation

- pyspark.sql.functions.concat --- PySpark 3.1.2 documentation

Static notebook | Dynamic notebook: See test 3, Question: 45 (Databricks import instructions)

Question 7

Question Type: **MultipleChoice**

The code block displayed below contains an error. The code block should combine data from DataFrames itemsDf and transactionsDf, showing all rows of DataFrame itemsDf that have a matching

value in column itemId with a value in column transactionsId of DataFrame transactionsDf. Find the error.

Code block:

itemsDf.join(itemsDf.itemId==transactionsDf.transactionId)

Options:

A- The join statement is incomplete.

B- The union method should be used instead of join.

C- The join method is inappropriate.

D- The merge method should be used instead of join.

E- The join expression is malformed.

Answer:

A

Explanation:

Correct code block:

itemsDf.join(transactionsDf, itemsDf.itemId==transactionsDf.transactionId)

The join statement is incomplete.

Correct! If you look at the documentation of DataFrame.join() (linked below), you see that the very first argument of join should be the DataFrame that should be joined with. This first argument is

missing in the code block.

The join method is inappropriate.

No. By default, DataFrame.join() uses an inner join. This method is appropriate for the scenario described in the question.

The join expression is malformed.

Incorrect. The join expression itemsDf.itemId==transactionsDf.transactionId is correct syntax.

The merge method should be used instead of join.

False. There is no DataFrame.merge() method in PySpark.

The union method should be used instead of join.

Wrong. DataFrame.union() merges rows, but not columns as requested in the question.

More info: pyspark.sql.DataFrame.join --- PySpark 3.1.2 documentation, pyspark.sql.DataFrame.union --- PySpark 3.1.2 documentation

Static notebook | Dynamic notebook: See test 3, Question: 44 (Databricks import instructions)

Question 8

Question Type: MultipleChoice

In which order should the code blocks shown below be run in order to create a DataFrame that shows the mean of column predError of DataFrame transactionsDf per column storeId and productId,

where productId should be either 2 or 3 and the returned DataFrame should be sorted in ascending order by column storeId, leaving out any nulls in that column?

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. +-------------+---------+-----+-------+---------+----+

1. .mean("predError")

2. .groupBy("storeId")

3. .orderBy("storeId")

4. transactionsDf.filter(transactionsDf.storeId.isNotNull())

5. .pivot("productId", [2, 3])

Options:

A- 4, 5, 2, 3, 1

B- 4, 2, 1

C- 4, 1, 5, 2, 3

D- 4, 2, 5, 1, 3

E- 4, 3, 2, 5, 1

Answer:

D

Explanation:

Correct code block:

transactionsDf.filter(transactionsDf.storeId.isNotNull()).groupBy('storeId').pivot('productId', [2, 3]).mean('predError').orderBy('storeId')

Output of correct code block:

+-------+----+----+

|storeId| 2| 3|

+-------+----+----+

| 2| 6.0|null|

| 3|null|null|

| 25| 3.0| 3.0|

+-------+----+----+

This Question: is quite convoluted and requires you to think hard about the correct order of operations. The pivot method also makes an appearance - a method that you may not know all

that much

about (yet).

At the first position in all answers is code block 4, so the Question: is essentially just about the ordering of the remaining 4 code blocks.

The Question: states that the returned DataFrame should be sorted by column storeId. So, it should make sense to have code block 3 which includes the orderBy operator at the very end of

the code

block. This leaves you with only two answer options.

Now, it is useful to know more about the context of pivot in PySpark. A common pattern is groupBy, pivot, and then another aggregating function, like mean. In the documentation linked below you

can see that pivot is a method of pyspark.sql.GroupedData - meaning that before pivoting, you have to use groupBy. The only answer option matching this requirement is the one in which code

block 2 (which includes groupBy) is stated before code block 5 (which includes pivot).

More info: pyspark.sql.GroupedData.pivot --- PySpark 3.1.2 documentation

Static notebook | Dynamic notebook: See test 3, Question: 43 (Databricks import instructions)

Question 9

Question Type: MultipleChoice

Which of the following code blocks returns a DataFrame that matches the multi-column DataFrame itemsDf, except that integer column itemId has been converted into a string column?

Options:

A- itemsDf.withColumn('itemId', convert('itemId', 'string'))

B- itemsDf.withColumn('itemId', col('itemId').cast('string'))
(Correct)

C- itemsDf.select(cast('itemId', 'string'))

D- itemsDf.withColumn('itemId', col('itemId').convert('string'))

E- spark.cast(itemsDf, 'itemId', 'string')

Answer:

B

Explanation:

itemsDf.withColumn('itemId', col('itemId').cast('string'))

Correct. You can convert the data type of a column using the cast method of the Column class. Also note that you will have to use the withColumn method on itemsDf for replacing the existing itemId

column with the new version that contains strings.

itemsDf.withColumn('itemId', col('itemId').convert('string'))

Incorrect. The Column object that col('itemId') returns does not have a convert method.

itemsDf.withColumn('itemId', convert('itemId', 'string'))

Wrong. Spark's spark.sql.functions module does not have a convert method. The Question: is trying to mislead you by using the word 'converted'. Type conversion is also called 'type

casting'. This

may help you remember to look for a cast method instead of a convert method (see correct answer).

itemsDf.select(astype('itemId', 'string'))

False. While astype is a method of Column (and an alias of Column.cast), it is not a method of pyspark.sql.functions (what the code block implies). In addition, the Question: asks to return a

full

DataFrame that matches the multi-column DataFrame itemsDf. Selecting just one column from itemsDf as in the code block would just return a single-column DataFrame.

spark.cast(itemsDf, 'itemId', 'string')

No, the Spark session (called by spark) does not have a cast method. You can find a list of all methods available for the Spark session linked in the documentation below.

More info:

- pyspark.sql.Column.cast --- PySpark 3.1.2 documentation

- pyspark.sql.Column.astype --- PySpark 3.1.2 documentation

- pyspark.sql.SparkSession --- PySpark 3.1.2 documentation

Static notebook | Dynamic notebook: See test 3, Question: 42 (Databricks import instructions)

Question 10

Question Type: MultipleChoice

Which of the following code blocks returns a 2-column DataFrame that shows the distinct values in column productId and the number of rows with that productId in DataFrame transactionsDf?

Options:

A- transactionsDf.count('productId').distinct()

B- transactionsDf.groupBy('productId').agg(col('value').count())

C- transactionsDf.count('productId')

D- transactionsDf.groupBy('productId').count()

E- transactionsDf.groupBy('productId').select(count('value'))

Answer:

D

Explanation:

transactionsDf.groupBy('productId').count()

Correct. This code block first groups DataFrame transactionsDf by column productId and then counts the rows in each group.

transactionsDf.groupBy('productId').select(count('value'))

Incorrect. You cannot call select on a GroupedData object (the output of a groupBy) statement.

transactionsDf.count('productId')

No. DataFrame.count() does not take any arguments.

transactionsDf.count('productId').distinct()

Wrong. Since DataFrame.count() does not take any arguments, this option cannot be right.

transactionsDf.groupBy('productId').agg(col('value').count())

False. A Column object, as returned by col('value'), does not have a count() method. You can see all available methods for Column object linked in the Spark documentation below.

More info: pyspark.sql.DataFrame.count --- PySpark 3.1.2 documentation, pyspark.sql.Column --- PySpark 3.1.2 documentation

Static notebook | Dynamic notebook: See test 3, Question: 41 (Databricks import instructions)

Question 11

Question Type: **MultipleChoice**

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

Options:

A- array_remove(transactionsDf, '*')

B- transactionsDf.unpersist()
(Correct)

C- del transactionsDf

D- transactionsDf.clearCache()

E- transactionsDf.persist()

Answer:

B

Explanation:

transactionsDf.unpersist()

Correct. The DataFrame.unpersist() command does exactly what the Question: 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, Question: 40 (Databricks import instructions)