Associate-Developer-Apache-Spark Dumps Updated Jun 27, 2022 Practice Test and 179 unique questions [Q35-Q53

Associate-Developer-Apache-Spark Dumps Updated Jun 27, 2022 Practice Test and 179 unique questions
2022 Latest 100% Exam Passing Ratio - Associate-Developer-Apache-Spark Dumps PDF

NEW QUESTION 35

Which of the following describes the conversion of a computational query into an execution plan in Spark?
* Spark uses the catalog to resolve the optimized logical plan.
* The catalog assigns specific resources to the optimized memory plan.
* The executed physical plan depends on a cost optimization from a previous stage.
* Depending on whether DataFrame API or SQL API are used, the physical plan may differ.
* The catalog assigns specific resources to the physical plan.

Explanation

The executed physical plan depends on a cost optimization from a previous stage.

Correct! Spark considers multiple physical plans on which it performs a cost analysis and selects the final physical plan in accordance with the lowest-cost outcome of that analysis. That final physical plan is then executed by Spark.

Spark uses the catalog to resolve the optimized logical plan.

No. Spark uses the catalog to resolve the unresolved logical plan, but not the optimized logical plan. Once the unresolved logical plan is resolved, it is then optimized using the Catalyst Optimizer.

The optimized logical plan is the input for physical planning.

The catalog assigns specific resources to the physical plan.

No. The catalog stores metadata, such as a list of names of columns, data types, functions, and databases.

Spark consults the catalog for resolving the references in a logical plan at the beginning of the conversion of the query into an execution plan. The result is then an optimized logical plan.

Depending on whether DataFrame API or SQL API are used, the physical plan may differ.

Wrong &#8211; the physical plan is independent of which API was used. And this is one of the great strengths of Spark!

The catalog assigns specific resources to the optimized memory plan.

There is no specific &#8220;memory plan&#8221; on the journey of a Spark computation.

More info: Spark&#8217;s Logical and Physical plans &#8230; When, Why, How and Beyond. | by Laurent Leturgez | datalex | Medium

NEW QUESTION 36

The code block shown below should return a DataFrame with all columns of DataFrame transactionsDf, but only maximum 2 rows in which column productId has at least the value 2. Choose the answer that correctly fills the blanks in the code block to accomplish this.

transactionsDf.__1__(__2__).__3__
* 1. where

2. &#8220;productId&#8221; > 2

3. max(2)
* 1. where

2. transactionsDf[productId] >= 2

3. limit(2)
* 1. filter

2. productId > 2

3. max(2)
* 1. filter

2. col(&#8220;productId&#8221;) >= 2

3. limit(2)
* 1. where

2. productId >= 2

3. limit(2)
Explanation

Correct code block:

transactionsDf.filter(col(&#8220;productId&#8221;) >= 2).limit(2)

The filter and where operators in gap 1 are just aliases of one another, so you cannot use them to pick the right answer.

The column definition in gap 2 is more helpful. The DataFrame.filter() method takes an argument of type Column or str. From all possible answers, only the one including col(&#8220;productId&#8221;) >= 2 fits this profile, since it returns a Column type.

The answer option using &#8220;productId&#8221; > 2 is invalid, since Spark does not understand that &#8220;productId&#8221; refers to column productId. The answer option using transactionsDf[productId] >= 2 is wrong because you cannot refer to a column using square bracket notation in Spark (if you are coming from Python using Pandas, this is something to watch out for). In all other options, productId is being referred to as a Python variable, so they are relatively easy to eliminate.

Also note that the question asks for the value in column productId being at least 2. This translates to a

&#8220;greater or equal&#8221; sign (>= 2), but not a &#8220;greater&#8221; sign (> 2).

Another thing worth noting is that there is no DataFrame.max() method. If you picked any option including this, you may be confusing it with the pyspark.sql.functions.max method. The correct method to limit the amount of rows is the DataFrame.limit() method.

More info:

&#8211; pyspark.sql.DataFrame.filter &#8211; PySpark 3.1.2 documentation

&#8211; pyspark.sql.DataFrame.limit &#8211; PySpark 3.1.2 documentation

Static notebook | Dynamic notebook: See test 3

NEW QUESTION 37

Which of the following statements about the differences between actions and transformations is correct?
* Actions are evaluated lazily, while transformations are not evaluated lazily.
* Actions generate RDDs, while transformations do not.
* Actions do not send results to the driver, while transformations do.
* Actions can be queued for delayed execution, while transformations can only be processed immediately.
* Actions can trigger Adaptive Query Execution, while transformation cannot.
Explanation

Actions can trigger Adaptive Query Execution, while transformation cannot.

Correct. Adaptive Query Execution optimizes queries at runtime. Since transformations are evaluated lazily, Spark does not have any runtime information to optimize the query until an action is called. If Adaptive Query Execution is enabled, Spark will then try to optimize the query based on the feedback it gathers while it is evaluating the query.

Actions can be queued for delayed execution, while transformations can only be processed immediately.

No, there is no such concept as &#8220;delayed execution&#8221; in Spark. Actions cannot be evaluated lazily, meaning that they are executed immediately.

Actions are evaluated lazily, while transformations are not evaluated lazily.

Incorrect, it is the other way around: Transformations are evaluated lazily and actions trigger their evaluation.

Actions generate RDDs, while transformations do not.

No. Transformations change the data and, since RDDs are immutable, generate new RDDs along the way.

Actions produce outputs in Python and data types (integers, lists, text files,&#8230;) based on the RDDs, but they do not generate them.

Here is a great tip on how to differentiate actions from transformations: If an operation returns a DataFrame, Dataset, or an RDD, it is a transformation. Otherwise, it is an action.

Actions do not send results to the driver, while transformations do.

No. Actions send results to the driver. Think about running DataFrame.count(). The result of this command will return a number to the driver. Transformations, however, do not send results back to the driver. They produce RDDs that remain on the worker nodes.

More info: What is the difference between a transformation and an action in Apache Spark? | Bartosz Mikulski, How to Speed up SQL Queries with Adaptive Query Execution

NEW QUESTION 38

Which of the following code blocks reads in the parquet file stored at location filePath, given that all columns in the parquet file contain only whole numbers and are stored in the most appropriate format for this kind of data?
* 1.spark.read.schema(

2. StructType(

3. StructField(&#8220;transactionId&#8221;, IntegerType(), True),

4. StructField(&#8220;predError&#8221;, IntegerType(), True)

5.)).load(filePath)
* 1.spark.read.schema([

2. StructField(&#8220;transactionId&#8221;, NumberType(), True),

3. StructField(&#8220;predError&#8221;, IntegerType(), True)

4.]).load(filePath)
* 1.spark.read.schema(

2. StructType([

3. StructField(&#8220;transactionId&#8221;, StringType(), True),

4. StructField(&#8220;predError&#8221;, IntegerType(), True)]

5.)).parquet(filePath)
* 1.spark.read.schema(

2. StructType([

3. StructField(&#8220;transactionId&#8221;, IntegerType(), True),

4. StructField(&#8220;predError&#8221;, IntegerType(), True)]

5.)).format(&#8220;parquet&#8221;).load(filePath)
* 1.spark.read.schema([

2. StructField(&#8220;transactionId&#8221;, IntegerType(), True),

3. StructField(&#8220;predError&#8221;, IntegerType(), True)

4.]).load(filePath, format=&#8221;parquet&#8221;)
Explanation

The schema passed into schema should be of type StructType or a string, so all entries in which a list is passed are incorrect.

In addition, since all numbers are whole numbers, the IntegerType() data type is the correct option here.

NumberType() is not a valid data type and StringType() would fail, since the parquet file is stored in the &#8220;most appropriate format for this kind of data&#8221;, meaning that it is most likely an IntegerType, and Spark does not convert data types if a schema is provided.

Also note that StructType accepts only a single argument (a list of StructFields). So, passing multiple arguments is invalid.

Finally, Spark needs to know which format the file is in. However, all of the options listed are valid here, since Spark assumes parquet as a default when no file format is specifically passed.

More info: pyspark.sql.DataFrameReader.schema &#8211; PySpark 3.1.2 documentation and StructType &#8211; PySpark 3.1.2 documentation

NEW QUESTION 39

Which of the following code blocks uses a schema fileSchema to read a parquet file at location filePath into a DataFrame?
* spark.read.schema(fileSchema).format(&#8220;parquet&#8221;).load(filePath)

* spark.read.schema(&#8220;fileSchema&#8221;).format(&#8220;parquet&#8221;).load(filePath)
* spark.read().schema(fileSchema).parquet(filePath)
* spark.read().schema(fileSchema).format(parquet).load(filePath)
* spark.read.schema(fileSchema).open(filePath)

Explanation

Pay attention here to which variables are quoted. fileSchema is a variable and thus should not be in quotes.

parquet is not a variable and therefore should be in quotes.

SparkSession.read (here referenced as spark.read) returns a DataFrameReader which all subsequent calls reference &#8211; the DataFrameReader is not callable, so you should not use parentheses here.

Finally, there is no open method in PySpark. The method name is load.

Static notebook | Dynamic notebook: See test 1

NEW QUESTION 40

Which of the following code blocks immediately removes the previously cached DataFrame transactionsDf from memory and disk?
* array_remove(transactionsDf, &#8220;*&#8221;)
* transactionsDf.unpersist()

(Correct)
* del transactionsDf
* transactionsDf.clearCache()
* transactionsDf.persist()

Explanation

transactionsDf.unpersist()

Correct. The DataFrame.unpersist() command does exactly what the question asks for &#8211; 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, &#8220;*&#8221;)

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&#8217;s DataFrame does not have a clearCache() method.

More info: pyspark.sql.DataFrame.unpersist &#8211; PySpark 3.1.2 documentation, python &#8211; How to delete an RDD in PySpark for the purpose of releasing resources? &#8211; Stack Overflow Static notebook | Dynamic notebook: See test 3

NEW QUESTION 41

Which of the following code blocks creates a new DataFrame with 3 columns, productId, highest, and lowest, that shows the biggest and smallest values of column value per value in column productId from DataFrame transactionsDf?

Sample of DataFrame transactionsDf:

1.+&#8212;&#8212;&#8212;&#8212;-+&#8212;&#8212;&#8212;+&#8212;&#8211;+&#8212;&#8212;-+&#8212;&#8212;&#8212;+&#8212;-+

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

3.+&#8212;&#8212;&#8212;&#8212;-+&#8212;&#8212;&#8212;+&#8212;&#8211;+&#8212;&#8212;-+&#8212;&#8212;&#8212;+&#8212;-+

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.+&#8212;&#8212;&#8212;&#8212;-+&#8212;&#8212;&#8212;+&#8212;&#8211;+&#8212;&#8212;-+&#8212;&#8212;&#8212;+&#8212;-+
* transactionsDf.max(&#8216;value&#8217;).min(&#8216;value&#8217;)
* transactionsDf.agg(max(&#8216;value&#8217;).alias(&#8216;highest&#8217;), min(&#8216;value&#8217;).alias(&#8216;lowest&#8217;))
* transactionsDf.groupby(col(productId)).agg(max(col(value)).alias(&#8220;highest&#8221;), min(col(value)).alias(&#8220;lowest&#8221;))
* transactionsDf.groupby(&#8216;productId&#8217;).agg(max(&#8216;value&#8217;).alias(&#8216;highest&#8217;), min(&#8216;value&#8217;).alias(&#8216;lowest&#8217;))
* transactionsDf.groupby(&#8220;productId&#8221;).agg({&#8220;highest&#8221;: max(&#8220;value&#8221;), &#8220;lowest&#8221;: min(&#8220;value&#8221;)})

Explanation

transactionsDf.groupby(&#8216;productId&#8217;).agg(max(&#8216;value&#8217;).alias(&#8216;highest&#8217;), min(&#8216;value&#8217;).alias(&#8216;lowest&#8217;)) Correct. groupby and aggregate is a common pattern to investigate aggregated values of groups.

transactionsDf.groupby(&#8220;productId&#8221;).agg({&#8220;highest&#8221;: max(&#8220;value&#8221;), &#8220;lowest&#8221;: min(&#8220;value&#8221;)}) Wrong. While DataFrame.agg() accepts dictionaries, the syntax of the dictionary in this code block is wrong.

If you use a dictionary, the syntax should be like {&#8220;value&#8221;: &#8220;max&#8221;}, so using the column name as the key and the aggregating function as value.

transactionsDf.agg(max(&#8216;value&#8217;).alias(&#8216;highest&#8217;), min(&#8216;value&#8217;).alias(&#8216;lowest&#8217;)) Incorrect. While this is valid Spark syntax, it does not achieve what the question asks for. The question specifically asks for values to be aggregated per value in column productId &#8211; this column is not considered here. Instead, the max() and min() values are calculated as if the entire DataFrame was a group.

transactionsDf.max(&#8216;value&#8217;).min(&#8216;value&#8217;)

Wrong. There is no DataFrame.max() method in Spark, so this command will fail.

transactionsDf.groupby(col(productId)).agg(max(col(value)).alias(&#8220;highest&#8221;), min(col(value)).alias(&#8220;lowest&#8221;)) No. While this may work if the column names are expressed as strings, this will not work as is. Python will interpret the column names as variables and, as a result, pySpark will not understand which columns you want to aggregate.

More info: pyspark.sql.DataFrame.agg &#8211; PySpark 3.1.2 documentation

Static notebook | Dynamic notebook: See test 3

NEW QUESTION 42

Which of the following code blocks returns a copy of DataFrame itemsDf where the column supplier has been renamed to manufacturer?
* itemsDf.withColumn([&#8220;supplier&#8221;, &#8220;manufacturer&#8221;])
* itemsDf.withColumn(&#8220;supplier&#8221;).alias(&#8220;manufacturer&#8221;)
* itemsDf.withColumnRenamed(&#8220;supplier&#8221;, &#8220;manufacturer&#8221;)
* itemsDf.withColumnRenamed(col(&#8220;manufacturer&#8221;), col(&#8220;supplier&#8221;))
* itemsDf.withColumnsRenamed(&#8220;supplier&#8221;, &#8220;manufacturer&#8221;)
Explanation

itemsDf.withColumnRenamed(&#8220;supplier&#8221;, &#8220;manufacturer&#8221;)

Correct! This uses the relatively trivial DataFrame method withColumnRenamed for renaming column supplier to column manufacturer.

Note that the question asks for &#8220;a copy of DataFrame itemsDf&#8221;. This may be confusing if you are not familiar with Spark yet. RDDs (Resilient Distributed Datasets) are the foundation of Spark DataFrames and are immutable. As such, DataFrames are immutable, too. Any command that changes anything in the DataFrame therefore necessarily returns a copy, or a new version, of it that has the changes applied.

itemsDf.withColumnsRenamed(&#8220;supplier&#8221;, &#8220;manufacturer&#8221;)

Incorrect. Spark&#8217;s DataFrame API does not have a withColumnsRenamed() method.

itemsDf.withColumnRenamed(col(&#8220;manufacturer&#8221;), col(&#8220;supplier&#8221;))

No. Watch out &#8211; although the col() method works for many methods of the DataFrame API, withColumnRenamed is not one of them. As outlined in the documentation linked below, withColumnRenamed expects strings.

itemsDf.withColumn([&#8220;supplier&#8221;, &#8220;manufacturer&#8221;])

Wrong. While DataFrame.withColumn() exists in Spark, it has a different purpose than renaming columns.

withColumn is typically used to add columns to DataFrames, taking the name of the new column as a first, and a Column as a second argument. Learn more via the documentation that is linked below.

itemsDf.withColumn(&#8220;supplier&#8221;).alias(&#8220;manufacturer&#8221;)

No. While DataFrame.withColumn() exists, it requires 2 arguments. Furthermore, the alias() method on DataFrames would not help the cause of renaming a column much. DataFrame.alias() can be useful in addressing the input of join statements. However, this is far outside of the scope of this question. If you are curious nevertheless, check out the link below.

More info: pyspark.sql.DataFrame.withColumnRenamed &#8211; PySpark 3.1.1 documentation, pyspark.sql.DataFrame.withColumn &#8211; PySpark 3.1.1 documentation, and pyspark.sql.DataFrame.alias &#8211; PySpark 3.1.2 documentation (https://bit.ly/3aSB5tm , https://bit.ly/2Tv4rbE , https://bit.ly/2RbhBd2) Static notebook | Dynamic notebook: See test 1 (https://flrs.github.io/spark_practice_tests_code/#1/31.html ,

https://bit.ly/sparkpracticeexams_import_instructions)

NEW QUESTION 43

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([(&#8220;red&#8221;,), (&#8220;blue&#8221;,), (&#8220;green&#8221;,)], &#8220;color&#8221;)

Instead of calling spark.createDataFrame, just DataFrame should be called.
* The commas in the tuples with the colors should be eliminated.
* The colors red, blue, and green should be expressed as a simple Python list, and not a list of tuples.
* Instead of color, a data type should be specified.
* The &#8220;color&#8221; expression needs to be wrapped in brackets, so it reads [&#8220;color&#8221;].

Explanation

Correct code block:

spark.createDataFrame([(&#8220;red&#8221;,), (&#8220;blue&#8221;,), (&#8220;green&#8221;,)], [&#8220;color&#8221;])

The createDataFrame syntax is not exactly straightforward, but luckily the documentation (linked below) provides several examples on how to use it. It also shows an example very similar to the code block presented here which should help you answer this question correctly.

More info: pyspark.sql.SparkSession.createDataFrame &#8211; PySpark 3.1.2 documentation Static notebook | Dynamic notebook: See test 2

NEW QUESTION 44

The code block shown below should return a new 2-column DataFrame that shows one attribute from column attributes per row next to the associated itemName, for all suppliers in column supplier whose name includes Sports. Choose the answer that correctly fills the blanks in the code block to accomplish this.

Sample of DataFrame itemsDf:

1.+&#8212;&#8212;+&#8212;&#8212;&#8212;&#8212;&#8212;&#8212;&#8212;&#8212;&#8212;&#8212;&#8212;-+&#8212;&#8212;&#8212;&#8212;&#8212;&#8212;&#8212;&#8212;&#8212;&#8212;&#8211;+&#8212;&#8212;&#8212;&#8212;&#8212;&#8212;-+

2.|itemId|itemName |attributes |supplier |

3.+&#8212;&#8212;+&#8212;&#8212;&#8212;&#8212;&#8212;&#8212;&#8212;&#8212;&#8212;&#8212;&#8212;-+&#8212;&#8212;&#8212;&#8212;&#8212;&#8212;&#8212;&#8212;&#8212;&#8212;&#8211;+&#8212;&#8212;&#8212;&#8212;&#8212;&#8212;-+

4.|1 |Thick Coat for Walking in the Snow|[blue, winter, cozy] |Sports Company Inc.|

5.|2 |Elegant Outdoors Summer Dress |[red, summer, fresh, cooling]|YetiX |

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

7.+&#8212;&#8212;+&#8212;&#8212;&#8212;&#8212;&#8212;&#8212;&#8212;&#8212;&#8212;&#8212;&#8212;-+&#8212;&#8212;&#8212;&#8212;&#8212;&#8212;&#8212;&#8212;&#8212;&#8212;&#8211;+&#8212;&#8212;&#8212;&#8212;&#8212;&#8212;-+ Code block:

itemsDf.__1__(__2__).select(__3__, __4__)
* 1. filter

2. col(&#8220;supplier&#8221;).isin(&#8220;Sports&#8221;)

3. &#8220;itemName&#8221;

4. explode(col(&#8220;attributes&#8221;))
* 1. where

2. col(&#8220;supplier&#8221;).contains(&#8220;Sports&#8221;)

3. &#8220;itemName&#8221;

4. &#8220;attributes&#8221;

* 1. where

2. col(supplier).contains(&#8220;Sports&#8221;)

3. explode(attributes)

4. itemName
* 1. where

2. &#8220;Sports&#8221;.isin(col(&#8220;Supplier&#8221;))

3. &#8220;itemName&#8221;

4. array_explode(&#8220;attributes&#8221;)
* 1. filter

2. col(&#8220;supplier&#8221;).contains(&#8220;Sports&#8221;)

3. &#8220;itemName&#8221;

4. explode(&#8220;attributes&#8221;)

Explanation

Output of correct code block:

```
+&#8212;&#8212;&#8212;&#8212;&#8212;&#8212;&#8212;&#8212;&#8212;&#8212;&#8212;-+&#8212;&#8212;+

|itemName |col |

+&#8212;&#8212;&#8212;&#8212;&#8212;&#8212;&#8212;&#8212;&#8212;&#8212;&#8212;-+&#8212;&#8212;+

|Thick Coat for Walking in the Snow|blue |

|Thick Coat for Walking in the Snow|winter|

|Thick Coat for Walking in the Snow|cozy |

|Outdoors Backpack |green |

|Outdoors Backpack |summer|

|Outdoors Backpack |travel|

+&#8212;&#8212;&#8212;&#8212;&#8212;&#8212;&#8212;&#8212;&#8212;&#8212;&#8212;-+&#8212;&#8212;+
```

The key to solving this question is knowing about Spark&#8217;s explode operator. Using this operator, you can extract values from arrays into single rows. The following guidance steps through the answers systematically from the first to the last gap. Note that there are many ways to solving the gap questions and filtering out wrong answers, you do not always have to start filtering out from the first gap, but can also exclude some answers based on obvious problems you see with them.

The answers to the first gap present you with two options: filter and where. These two are actually synonyms in PySpark, so using either of those is fine. The answer options to this gap therefore do not help us in selecting the right answer.

The second gap is more interesting. One answer option includes &#8220;Sports&#8221;.isin(col(&#8220;Supplier&#8221;)). This construct does not work, since Python&#8217;s string does not have an isin method. Another option contains col(supplier). Here, Python will try to interpret supplier as a variable. We have not set this variable, so this is not a viable answer. Then, you are left with answers options that include col (&#8220;supplier&#8221;).contains(&#8220;Sports&#8221;) and col(&#8220;supplier&#8221;).isin(&#8220;Sports&#8221;). The question states that we are looking for suppliers whose name includes Sports, so we have to go for the contains operator here.

We would use the isin operator if we wanted to filter out for supplier names that match any entries in a list of supplier names.

Finally, we are left with two answers that fill the third gap both with &#8220;itemName&#8221; and the fourth gap either with explode(&#8220;attributes&#8221;) or &#8220;attributes&#8221;. While both are correct Spark syntax, only explode (&#8220;attributes&#8221;) will help us achieve our goal. Specifically, the question asks for one attribute from column attributes per row &#8211; this is what the explode() operator does.

One answer option also includes array_explode() which is not a valid operator in PySpark.

More info: pyspark.sql.functions.explode &#8211; PySpark 3.1.2 documentation

Static notebook | Dynamic notebook: See test 3

NEW QUESTION 45

Which of the following statements about Spark&#8217;s DataFrames is incorrect?
* Spark&#8217;s DataFrames are immutable.
* Spark&#8217;s DataFrames are equal to Python&#8217;s DataFrames.
* Data in DataFrames is organized into named columns.
* RDDs are at the core of DataFrames.
* The data in DataFrames may be split into multiple chunks.
Explanation

Spark&#8217;s DataFrames are equal to Python&#8217;s or R&#8217;s DataFrames.

No, they are not equal. They are only similar. A major difference between Spark and Python is that Spark&#8217;s DataFrames are distributed, whereby Python&#8217;s are not.

NEW QUESTION 46

The code block shown below should return all rows of DataFrame itemsDf that have at least 3 items in column itemNameElements. Choose the answer that correctly fills the blanks in the code block to accomplish this.

Example of DataFrame itemsDf:

1.+&#8212;&#8212;+&#8212;&#8212;&#8212;&#8212;&#8212;&#8212;&#8212;&#8212;&#8212;&#8212;&#8212;-+&#8212;&#8212;&#8212;&#8212;&#8212;&#8212;-+&#8212;&#8212;&#8212;&#8212;&#8212;&#8212;&#8212;&#8212;&#8212;&#8212;&#8212;&#8212;&#8212;&#8212;&#8212;+

2.|itemId|itemName |supplier |itemNameElements |

3.+&#8212;&#8212;+&#8212;&#8212;&#8212;&#8212;&#8212;&#8212;&#8212;&#8212;&#8212;&#8212;&#8212;-+&#8212;&#8212;&#8212;&#8212;&#8212;&#8212;-+&#8212;&#8212;&#8212;&#8212;&#8212;&#8212;&#8212;&#8212;&#8212;&#8212;&#8212;&#8212;&#8212;&#8212;+

4.|1 |Thick Coat for Walking in the Snow|Sports Company Inc.|[Thick, Coat, for, Walking, in, the, Snow]|

5.|2 |Elegant Outdoors Summer Dress |YetiX |[Elegant, Outdoors, Summer, Dress] |

6.|3 |Outdoors Backpack |Sports Company Inc.|[Outdoors, Backpack] |

7.+&#8212;&#8212;+&#8212;&#8212;&#8212;&#8212;&#8212;&#8212;&#8212;&#8212;&#8212;&#8212;&#8212;-+&#8212;&#8212;&#8212;&#8212;&#8212;&#8212;-+&#8212;&#8212;&#8212;&#8212;&#8212;&#8212;&#8212;&#8212;&#8212;&#8212;&#8212;&#8212;&#8212;&#8212;+ Code block:

itemsDf.__1__(__2__(__3__)__4__)
* 1. select

2. count

3. col(&#8220;itemNameElements&#8221;)

4. >3
* 1. filter

2. count

3. itemNameElements

4. >=3
* 1. select

2. count

3. &#8220;itemNameElements&#8221;

4. >3
* 1. filter

2. size

3. &#8220;itemNameElements&#8221;

4. >=3

(Correct)
* 1. select

2. size

3. &#8220;itemNameElements&#8221;

4. >3
Explanation

Correct code block:

itemsDf.filter(size(&#8220;itemNameElements&#8221;)>3)

Output of code block:

+&#8212;&#8212;+&#8212;&#8212;&#8212;&#8212;&#8212;&#8212;&#8212;&#8212;&#8212;&#8212;&#8212;-+&#8212;&#8212;&#8212;&#8212;&#8212;&#8212;-+&#8212;&#8212;&#8212;&#8212;&#8212;&#8212;&#8212;&#8212;&#8212;&#8212;&#8212;&#8212;&#8212;&#8212;+

|itemId|itemName |supplier |itemNameElements |

+&#8212;&#8212;+&#8212;&#8212;&#8212;&#8212;&#8212;&#8212;&#8212;&#8212;&#8212;&#8212;&#8212;-+&#8212;&#8212;&#8212;&#8212;&#8212;&#8212;-+&#8212;&#8212;&#8212;&#8212;&#8212;&#8212;&#8212;&#8212;&#8212;&#8212;&#8212;&#8212;&#8212;&#8212;+

|1 |Thick Coat for Walking in the Snow|Sports Company Inc.|[Thick, Coat, for, Walking, in, the, Snow]|

|2 |Elegant Outdoors Summer Dress |YetiX |[Elegant, Outdoors, Summer, Dress] |

+&#8212;&#8212;+&#8212;&#8212;&#8212;&#8212;&#8212;&#8212;&#8212;&#8212;&#8212;&#8212;&#8212;-+&#8212;&#8212;&#8212;&#8212;&#8212;&#8212;-+&#8212;&#8212;&#8212;&#8212;&#8212;&#8212;&#8212;&#8212;&#8212;&#8212;&#8212;&#8212;&#8212;&#8212;+ The big difficulty with this question is in knowing the difference between count and size (refer to documentation below). size is the correct function to choose here since it returns the number of elements in an array on a per-row basis.

The other consideration for solving this question is the difference between select and filter. Since we want to return the rows in the original DataFrame, filter is the right choice. If we would use select, we would simply get a single-column DataFrame showing which rows match the criteria, like so:

+&#8212;&#8212;&#8212;&#8212;&#8212;&#8212;&#8212;&#8212;&#8212;-+

|(size(itemNameElements) > 3)|

+&#8212;&#8212;&#8212;&#8212;&#8212;&#8212;&#8212;&#8212;&#8212;-+

|true |

|true |

|false |

+&#8212;&#8212;&#8212;&#8212;&#8212;&#8212;&#8212;&#8212;&#8212;-+

More info:

Count documentation: pyspark.sql.functions.count &#8211; PySpark 3.1.1 documentation Size documentation: pyspark.sql.functions.size &#8211; PySpark 3.1.1 documentation Static notebook | Dynamic notebook: See test 1

NEW QUESTION 47

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.+&#8212;&#8212;&#8212;&#8212;-+&#8212;&#8212;&#8212;+&#8212;&#8211;+&#8212;&#8212;-+&#8212;&#8212;&#8212;+&#8212;-+

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

3.+&#8212;&#8212;&#8212;&#8212;-+&#8212;&#8212;&#8212;+&#8212;&#8211;+&#8212;&#8212;-+&#8212;&#8212;&#8212;+&#8212;-+

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

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

6.+&#8212;&#8212;&#8212;&#8212;-+&#8212;&#8212;&#8212;+&#8212;&#8211;+&#8212;&#8212;-+&#8212;&#8212;&#8212;+&#8212;-+

Sample of DataFrame transactionsDfTuesday:

1.+&#8212;&#8212;-+&#8212;&#8212;&#8212;&#8212;-+&#8212;&#8212;&#8212;+&#8212;&#8211;+

2.|storeId|transactionId|productId|value|

3.+&#8212;&#8212;-+&#8212;&#8212;&#8212;&#8212;-+&#8212;&#8212;&#8212;+&#8212;&#8211;+

4.| 25| 1| 1| 4|

5.| 2| 2| 2| 7|

6.| 3| 4| 2| null|

7.| null| 5| 2| null|

8.+&#8212;&#8212;-+&#8212;&#8212;&#8212;&#8212;-+&#8212;&#8212;&#8212;+&#8212;&#8211;+

Code block:

sc.union([transactionsDfMonday, transactionsDfTuesday])
* The DataFrames&#8217; RDDs need to be passed into the sc.union method instead of the DataFrame variable names.
* Instead of union, the concat method should be used, making sure to not use its default arguments.

* 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.
* Instead of the Spark context, transactionDfMonday should be called with the union method.
* 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.
Explanation

Correct code block:

transactionsDfMonday.unionByName(transactionsDfTuesday, True)

Output of correct code block:

+&#8212;&#8212;&#8212;&#8212;-+&#8212;&#8212;&#8212;+&#8212;&#8211;+&#8212;&#8212;-+&#8212;&#8212;&#8212;+&#8212;-+

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

+&#8212;&#8212;&#8212;&#8212;-+&#8212;&#8212;&#8212;+&#8212;&#8211;+&#8212;&#8212;-+&#8212;&#8212;&#8212;+&#8212;-+

| 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|

+&#8212;&#8212;&#8212;&#8212;-+&#8212;&#8212;&#8212;+&#8212;&#8211;+&#8212;&#8212;-+&#8212;&#8212;&#8212;+&#8212;-+

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 &#8211; 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 &#8220;into a new DataFrame&#8221;.

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:

&#8211; pyspark.sql.DataFrame.unionByName &#8211; PySpark 3.1.2 documentation

&#8211; pyspark.SparkContext.union &#8211; PySpark 3.1.2 documentation

&#8211; pyspark.sql.functions.concat &#8211; PySpark 3.1.2 documentation

Static notebook | Dynamic notebook: See test 3

NEW QUESTION 48

Which of the following describes the role of tasks in the Spark execution hierarchy?
* Tasks are the smallest element in the execution hierarchy.
* Within one task, the slots are the unit of work done for each partition of the data.
* Tasks are the second-smallest element in the execution hierarchy.
* Stages with narrow dependencies can be grouped into one task.
* Tasks with wide dependencies can be grouped into one stage.

Explanation

Stages with narrow dependencies can be grouped into one task.

Wrong, tasks with narrow dependencies can be grouped into one stage.

Tasks with wide dependencies can be grouped into one stage.

Wrong, since a wide transformation causes a shuffle which always marks the boundary of a stage. So, you cannot bundle multiple tasks that have wide dependencies into a stage.

Tasks are the second-smallest element in the execution hierarchy.

No, they are the smallest element in the execution hierarchy.

Within one task, the slots are the unit of work done for each partition of the data.

No, tasks are the unit of work done per partition. Slots help Spark parallelize work. An executor can have multiple slots which enable it to process multiple tasks in parallel.

NEW QUESTION 49

The code block shown below should write DataFrame transactionsDf to disk at path csvPath as a single CSV file, using tabs (t characters) as separators between columns, expressing missing values as string n/a, and omitting a header row with column names. Choose the answer that correctly fills the blanks in the code block to accomplish this.

transactionsDf.__1__.write.__2__(__3__, &#8221; &#8220;).__4__.__5__(csvPath)
* 1. coalesce(1)

2. option

3. &#8220;sep&#8221;

4. option(&#8220;header&#8221;, True)

5. path
* 1. coalesce(1)

2. option

3. &#8220;colsep&#8221;

4. option(&#8220;nullValue&#8221;, &#8220;n/a&#8221;)

5. path
* 1. repartition(1)

2. option

3. &#8220;sep&#8221;

4. option(&#8220;nullValue&#8221;, &#8220;n/a&#8221;)

5. csv

(Correct)
* 1. csv

2. option

3. &#8220;sep&#8221;

4. option(&#8220;emptyValue&#8221;, &#8220;n/a&#8221;)

5. path

* 1. repartition(1)

2. mode

3. &#8220;sep&#8221;

4. mode(&#8220;nullValue&#8221;, &#8220;n/a&#8221;)

5. csv
Explanation

Correct code block:

transactionsDf.repartition(1).write.option(&#8220;sep&#8221;, &#8220;t&#8221;).option(&#8220;nullValue&#8221;,

&#8220;n/a&#8221;).csv(csvPath) It is important here to understand that the question specifically asks for writing the DataFrame as a single CSV file. This should trigger you to think about partitions. By default, every partition is written as a separate file, so you need to include repatition(1) into your call. coalesce(1) works here, too!

Secondly, the question is very much an invitation to search through the parameters in the Spark documentation that work with DataFrameWriter.csv (link below). You will also need to know that you need an option() statement to apply these parameters.

The final concern is about the general call structure. Once you have called accessed write of your DataFrame, options follow and then you write the DataFrame with csv. Instead of csv(csvPath), you could also use save(csvPath, format=&#8217;csv&#8217;) here.

More info: pyspark.sql.DataFrameWriter.csv &#8211; PySpark 3.1.1 documentation Static notebook | Dynamic notebook: See test 1

NEW QUESTION 50

Which of the following describes a shuffle?
* A shuffle is a process that is executed during a broadcast hash join.
* A shuffle is a process that compares data across executors.
* A shuffle is a process that compares data across partitions.
* A shuffle is a Spark operation that results from DataFrame.coalesce().
* A shuffle is a process that allocates partitions to executors.
Explanation

A shuffle is a Spark operation that results from DataFrame.coalesce().

No. DataFrame.coalesce() does not result in a shuffle.

A shuffle is a process that allocates partitions to executors.

This is incorrect.

A shuffle is a process that is executed during a broadcast hash join.

No, broadcast hash joins avoid shuffles and yield performance benefits if at least one of the two tables is small in size (<= 10 MB by default). Broadcast hash joins can avoid shuffles because instead of exchanging partitions between executors, they broadcast a small table to all executors that then perform the rest of the join operation locally.

A shuffle is a process that compares data across executors.

No, in a shuffle, data is compared across partitions, and not executors.

More info: Spark Repartition & Coalesce &#8211; Explained (https://bit.ly/32KF7zS)

NEW QUESTION 51

The code block displayed below contains an error. The code block is intended to join DataFrame itemsDf with the larger DataFrame transactionsDf on column itemId. Find the error.

Code block:

transactionsDf.join(itemsDf, &#8220;itemId&#8221;, how=&#8221;broadcast&#8221;)
* The syntax is wrong, how= should be removed from the code block.
* The join method should be replaced by the broadcast method.
* Spark will only perform the broadcast operation if this behavior has been enabled on the Spark cluster.
* The larger DataFrame transactionsDf is being broadcasted, rather than the smaller DataFrame itemsDf.
* broadcast is not a valid join type.
Explanation

broadcast is not a valid join type.

Correct! The code block should read transactionsDf.join(broadcast(itemsDf), &#8220;itemId&#8221;). This would imply an inner join (this is the default in DataFrame.join()), but since the join type is not given in the question, this would be a valid choice.

The larger DataFrame transactionsDf is being broadcasted, rather than the smaller DataFrame itemsDf.

This option does not apply here, since the syntax around broadcasting is incorrect.

Spark will only perform the broadcast operation if this behavior has been enabled on the Spark cluster.

No, it is enabled by default, since the spark.sql.autoBroadcastJoinThreshold property is set to 10 MB by default. If that property would be set to -1, then broadcast joining would be disabled.

More info: Performance Tuning &#8211; Spark 3.1.1 Documentation (https://bit.ly/3gCz34r) The join method should be replaced by the broadcast method.

No, DataFrame has no broadcast() method.

The syntax is wrong, how= should be removed from the code block.

No, having the keyword argument how= is totally acceptable.

NEW QUESTION 52

The code block displayed below contains an error. When the code block below has executed, it should have divided DataFrame transactionsDf into 14 parts, based on columns storeId and transactionDate (in this order). Find the error.

Code block:

transactionsDf.coalesce(14, (&#8220;storeId&#8221;, &#8220;transactionDate&#8221;))
* The parentheses around the column names need to be removed and .select() needs to be appended to the code block.
* Operator coalesce needs to be replaced by repartition, the parentheses around the column names need to be removed, and .count() needs to be appended to the code block.

(Correct)
* Operator coalesce needs to be replaced by repartition, the parentheses around the column names need to be removed, and .select() needs to be appended to the code block.
* Operator coalesce needs to be replaced by repartition and the parentheses around the column names need to be replaced by square brackets.
* Operator coalesce needs to be replaced by repartition.
Explanation

Correct code block:

transactionsDf.repartition(14, &#8220;storeId&#8221;, &#8220;transactionDate&#8221;).count()

Since we do not know how many partitions DataFrame transactionsDf has, we cannot safely use coalesce, since it would not make any change if the current number of partitions is smaller than 14.

So, we need to use repartition.

In the Spark documentation, the call structure for repartition is shown like this:

DataFrame.repartition(numPartitions, *cols). The * operator means that any argument after numPartitions will be interpreted as column. Therefore, the brackets need to be removed.

Finally, the question specifies that after the execution the DataFrame should be divided. So, indirectly this question is asking us to append an action to the code block. Since .select() is a transformation. the only possible choice here is .count().

More info: pyspark.sql.DataFrame.repartition &#8211; PySpark 3.1.1 documentation Static notebook | Dynamic notebook: See test 1

NEW QUESTION 53

Which of the following options describes the responsibility of the executors in Spark?
* The executors accept jobs from the driver, analyze those jobs, and return results to the driver.
* The executors accept tasks from the driver, execute those tasks, and return results to the cluster manager.
* The executors accept tasks from the driver, execute those tasks, and return results to the driver.
* The executors accept tasks from the cluster manager, execute those tasks, and return results to the driver.
* The executors accept jobs from the driver, plan those jobs, and return results to the cluster manager.
Explanation

More info: Running Spark: an overview of Spark&#8217;s runtime architecture &#8211; Manning (https://bit.ly/2RPmJn9)