DirectCertify

Unlock Your Potential, Certify Your Success

Databricks

Databricks-Certified-Associate-Developer-for-Apache-Spark-3.5 Exam

Certified Associate Developer for Apache Spark 3.5 - Python Exam

Exam Latest Version: 7.1

DEMO Version

Full Version Features:

• 90 Days Free Updates

• 30 Days Money Back Guarantee

• Instant Download Once Purchased

• 24 Hours Live Chat Support

Full version is available at link below with affordable price.

https://www.directcertify.com/databricks/databricks-certified-associate-developer-for-apache-spark-3.5

Question 1. (Single Select)

A data scientist of an e-commerce company is working with user data obtained from its subscriber database and has stored the data in a DataFrame df_user. Before further processing the data, the data scientist wants to create another DataFrame df_user_non_pii and store only the non-PII columns in this DataFrame. The PII columns in df_user are first_name, last_name, email, and birthdate.

Which code snippet can be used to meet this requirement?

A: df_user_non_pii = df_user.drop("first_name", "last_name", "email", "birthdate")
B: df_user_non_pii = df_user.drop("first_name", "last_name", "email", "birthdate")
C: df_user_non_pii = df_user.dropfields("first_name", "last_name", "email", "birthdate")
D: df_user_non_pii = df_user.dropfields("first_name, last_name, email, birthdate")

Correct Answer: A

Explanation:

To remove specific columns from a PySpark DataFrame, the drop() method is used. This method returns a new DataFrame without the specified columns. The correct syntax for dropping multiple columns is to pass each column name as a separate argument to the drop() method.

Correct Usage:

df_user_non_pii = df_user.drop("first_name", "last_name", "email", "birthdate")

This line of code will return a new DataFrame df_user_non_pii that excludes the specified PII columns.

Explanation of Options:

A . Correct. Uses the drop() method with multiple column names passed as separate arguments, which is the standard and correct usage in PySpark.

B . Although it appears similar to Option A, if the column names are not enclosed in quotes or if there's a syntax error (e.g., missing quotes or incorrect variable names), it would result in an error. However, as written, it's identical to Option A and thus also correct.

C . Incorrect. The dropfields() method is not a method of the DataFrame class in PySpark. It's used with StructType columns to drop fields from nested structures, not top-level DataFrame columns.

D . Incorrect. Passing a single string with comma-separated column names to dropfields() is not valid syntax in PySpark.

PySpark Documentation: DataFrame.drop

Stack Overflow Discussion: How to delete columns in PySpark DataFrame

Question 2. (Multi Select)

A data engineer is reviewing a Spark application that applies several transformations to a DataFrame but notices that the job does not start executing immediately.

Which two characteristics of Apache Spark's execution model explain this behavior?

Choose 2 answers:

A: The Spark engine requires manual intervention to start executing transformations.
B: Only actions trigger the execution of the transformation pipeline.
C: Transformations are executed immediately to build the lineage graph.
D: The Spark engine optimizes the execution plan during the transformations, causing delays.
E: Transformations are evaluated lazily.

Correct Answer: B, E

Explanation:

Apache Spark employs a lazy evaluation model for transformations. This means that when transformations (e.g., map(), filter()) are applied to a DataFrame, Spark does not execute them immediately. Instead, it builds a logical plan (lineage) of transformations to be applied.

Execution is deferred until an action (e.g., collect(), count(), save()) is called. At that point, Spark's Catalyst optimizer analyzes the logical plan, optimizes it, and then executes the physical plan to produce the result.

This lazy evaluation strategy allows Spark to optimize the execution plan, minimize data shuffling, and improve overall performance by reducing unnecessary computations.

Question 3. (Single Select)

A data scientist is analyzing a large dataset and has written a PySpark script that includes several transformations and actions on a DataFrame. The script ends with a collect() action to retrieve the results.

How does Apache Spark™'s execution hierarchy process the operations when the data scientist runs this script?

A: The script is first divided into multiple applications, then each application is split into jobs, stages, and finally tasks.
B: The entire script is treated as a single job, which is then divided into multiple stages, and each stage is further divided into tasks based on data partitions.
C: The collect() action triggers a job, which is divided into stages at shuffle boundaries, and each stage is split into tasks that operate on individual data partitions.
D: Spark creates a single task for each transformation and action in the script, and these tasks are grouped into stages and jobs based on their dependencies.

Correct Answer: C

Explanation:

In Apache Spark, the execution hierarchy is structured as follows:

Application: The highest-level unit, representing the user program built on Spark.

Job: Triggered by an action (e.g., collect(), count()). Each action corresponds to a job.

Stage: A job is divided into stages based on shuffle boundaries. Each stage contains tasks that can be executed in parallel.

Task: The smallest unit of work, representing a single operation applied to a partition of the data.

When the collect() action is invoked, Spark initiates a job. This job is then divided into stages at points where data shuffling is required (i.e., wide transformations). Each stage comprises tasks

that are distributed across the cluster's executors, operating on individual data partitions.

This hierarchical execution model allows Spark to efficiently process large-scale data by parallelizing tasks and optimizing resource utilization.

Question 4. (Single Select)

A developer is trying to join two tables, sales.purchases_fct and sales.customer_dim, using the following code:

```
import pyspark.sql.functions as F

purch_df = spark.table('sales.purchases_fct')
cust_df = spark.table('sales.customer_dim').dropDuplicates(['cust_id'])

fact_df = purch_df.join(cust_df, F.col('customer_id') == F.col('cust_id'))
```

fact_df = purch_df.join(cust_df, F.col('customer_id') == F.col('custid')) The developer has discovered that customers in the purchases_fct table that do not exist in the customer_dim table are being dropped from the joined table. Which change should be made to the code to stop these customer records from being dropped?

A: fact_df = purch_df.join(cust_df, F.col('customer_id') == F.col('custid'), 'left')
B: fact_df = cust_df.join(purch_df, F.col('customer_id') == F.col('custid'))
C: fact_df = purch_df.join(cust_df, F.col('cust_id') == F.col('customer_id'))
D: fact_df = purch_df.join(cust_df, F.col('customer_id') == F.col('custid'), 'right_outer')

Correct Answer: A

Explanation:

In Spark, the default join type is an inner join, which returns only the rows with matching keys in both DataFrames. To retain all records from the left DataFrame (purch_df) and include matching records from the right DataFrame (cust_df), a left outer join should be used.

By specifying the join type as 'left', the modified code ensures that all records from purch_df are preserved, and matching records from cust_df are included. Records in purch_df without a corresponding match in cust_df will have null values for the columns from cust_df.

This approach is consistent with standard SQL join operations and is supported in PySpark's DataFrame API.

Question 5. (Multi Select)

A data engineer is reviewing a Spark application that applies several transformations to a DataFrame but notices that the job does not start executing immediately.

Which two characteristics of Apache Spark's execution model explain this behavior?

Choose 2 answers:

A: The Spark engine requires manual intervention to start executing transformations.
B: Only actions trigger the execution of the transformation pipeline.
C: Transformations are executed immediately to build the lineage graph.
D: The Spark engine optimizes the execution plan during the transformations, causing delays.
E: Transformations are evaluated lazily.

Correct Answer: B, E

Explanation:

Apache Spark employs a lazy evaluation model for transformations. This means that when transformations (e.g., map(), filter()) are applied to a DataFrame, Spark does not execute them immediately. Instead, it builds a logical plan (lineage) of transformations to be applied.

Execution is deferred until an action (e.g., collect(), count(), save()) is called. At that point, Spark's Catalyst optimizer analyzes the logical plan, optimizes it, and then executes the physical plan to produce the result.

This lazy evaluation strategy allows Spark to optimize the execution plan, minimize data shuffling, and improve overall performance by reducing unnecessary computations.