Databricks
Developer-for-Apache-Spark-Scala
Certified Associate Developer for Apache Spark - Scala

Latest Version: 6.0

Question: 1

Which of the following statements accurately describes the role of the Spark Driver in a Spark application?

A. The Driver is responsible for distributing tasks to executors and managing data partitioning.
B. The Driver is responsible for storing the final results of the computation.
C. The Driver is the main process that coordinates the execution of the Spark application, including submitting tasks to executors, tracking their progress, and aggregating results.
D. The Driver is responsible for reading and writing data to external data sources.
E. The Driver is responsible for managing the memory allocation for executors.

Answer: C

Explanation:
The Spark Driver is the main process that coordinates the execution of the Spark application. It is responsible for submitting tasks to executors, tracking their progress, and aggregating results. The Driver is not responsible for data partitioning, storing final results, reading/writing to external sources, or managing executor memory.

Question: 2

In the context of Spark, what is the primary function of a Spark Executor?

A. Executors are responsible for converting data from one format to another, like CSV to Parquet.
B. Executors are responsible for caching data in memory for faster retrieval.
C. Executors are responsible for running tasks assigned by the Driver and processing data on the worker nodes.
D. Executors are responsible for scheduling tasks across different worker nodes.
E. Executors are responsible for managing the communication between the Driver and the worker nodes.

Answer: C

Explanation:
Spark Executors are responsible for executing tasks assigned by the Driver. They are deployed on the worker nodes and process data locally. Data conversion, caching, task scheduling, and communication management are not the primary functions of executors.

Question: 3

What is the key advantage of using Spark's Adaptive Query Execution (AQE)?

A. AQE automatically optimizes the physical execution plan based on runtime data characteristics, potentially leading to faster query execution.
B. AQE allows for dynamic allocation of resources to executors based on workload demands.
C. AQE enables the use of different data sources without requiring specific configuration changes.
D. AQE ensures data integrity by automatically validating data transformations during execution.
E. AQE eliminates the need for manual optimization of Spark queries.

Answer: A

Explanation:
Adaptive Query Execution (AQE) in Spark dynamically adjusts the physical execution plan based on runtime data characteristics. This can lead to faster query execution by making optimal decisions regarding data shuffling, partitioning, and other aspects of the query plan. While AQE can enhance performance, it does not directly address resource allocation, data source compatibility, data validation, or eliminate the need for manual optimization.

Question: 4

Which of the following scenarios would benefit the most from using Adaptive Query Execution (AQE) in Spark?

A. A query that processes a fixed dataset with a well-defined schema and predictable data distribution.
B. A query that performs simple aggregations on a small dataset, where performance is not critical.
C. A query that involves complex joins, aggregations, and data transformations on a large and potentially skewed dataset.
D. A query that reads data from a real-time data stream, where data characteristics are highly dynamic.
E. A query that performs batch processing of data with static characteristics.

Answer: C,D

Explanation:
AQE is most beneficial when dealing with complex queries on large and potentially skewed datasets, as well as queries on dynamic data streams. In these scenarios, AQE can effectively adapt to data characteristics and optimize the execution plan for improved performance. AQE is less impactful on simple queries with fixed datasets, where pre-optimization might already be sufficient.

Question: 5

How does Spark's AQE dynamically adjust the query execution plan?

A. By analyzing the data distribution and skewness during execution, AQE dynamically adjusts the number of partitions and the shuffling strategy to optimize data movement.
B. AQE dynamically adjusts the memory allocation for executors based on the data volume being processed.
C. AQE dynamically adjusts the number of worker nodes used for execution based on the workload demands.
D. AQE dynamically adjusts the data serialization format used for data exchange between executors.
E. AQE dynamically adjusts the Spark configuration settings based on the query performance metrics.

Answer: A

Explanation:
A major aspect of AQE's dynamic adjustments involves analyzing data distribution and skewness during execution. This information is used to dynamically adjust the number of partitions and the shuffling strategy, effectively optimizing data movement and potentially reducing query execution time. While AQE can influence resource allocation (like memory), it does not directly manage the number of worker nodes, serialization format, or Spark configuration settings.

Question: 6

What is the role of the Spark Shuffle Manager in Adaptive Query Execution (AQE)?

A. The Shuffle Manager is responsible for managing the data partitioning and shuffling process, allowing AQE to dynamically adjust these operations based on runtime data characteristics.
B. The Shuffle Manager is responsible for caching frequently accessed data in memory, which can improve AQE's performance.
C. The Shuffle Manager is responsible for validating data transformations during execution, ensuring data integrity for AQE.
D. The Shuffle Manager is responsible for scheduling tasks across different worker nodes, which can be dynamically adjusted by AQE.
E. The Shuffle Manager is responsible for managing the communication between the Driver and the worker nodes, which can be optimized by AQE.

Answer: A

Explanation:
The Spark Shuffle Manager plays a crucial role in AQE by managing the data partitioning and shuffling process. This allows AQE to dynamically adjust these operations based on runtime data characteristics. While the Shuffle Manager is important for data management, its role in AQE is primarily focused on data partitioning and shuffling, not caching, data validation, task scheduling, or communication management.

Question: 7

You have a Spark DataFrame with columns 'customer_id', 'order_date', and 'amount'. You need to calculate the total amount spent by each customer for orders placed in the last 30 days. Which Spark DataFrame API functions would you use to achieve this?

A. filter, groupBy, agg
B. select, where, sum
C. join, distinct, count
D. withColumn, orderBy, limit
E. map, reduce, collect

Answer: A

Explanation:
The correct approach involves filtering the DataFrame for orders within the last 30 days, grouping by 'customer_id', and then aggregating the 'amount' column using the 'sum' function. This aligns with options A's functions: filter, groupBy, and agg. The other options are either incorrect or not relevant to the specific task.

Question: 8

You have a DataFrame 'df' with columns 'product_id', 'product_name', and 'product_price'. You want to calculate the average 'product_price' for each unique 'product_name' and create a new DataFrame with columns 'product_name' and 'average price'. Which of the following Python code snippets correctly achieves this?

○
```
df.groupBy("product_name").agg(avg("product_price").alias("average_price"))
```

○
```
df.select("product_name", avg("product_price").alias("average_price")).groupBy("product_name").agg(avg("product_price"))
```

○
```
df.groupBy("product_name").mean("product_price")
```

○
```
df.groupBy("product_name").agg(avg("product_price"))
```

○
```
df.groupBy("product_name").agg(mean("product_price"))
```

A. Option A
B. Option B
C. Option C
D. Option D
E. Option E

Answer: A

Explanation:

Option A correctly calculates the average price for each product name using the ‘groupBy’ and ‘agg’ functions. The ‘avg’ function calculates the average, and ‘alias' is used to rename the aggregated column. Option B is incorrect because it performs ‘select' before ‘groupBy’ , which would result in an error. Option C is incorrect because the ‘mean' function is not used for aggregation in PySpark. Option D is incorrect because it does not rename the aggregated column. Option E is incorrect because it uses the ‘mean' function, which is not used for aggregation in PySpark.

Question: 9

You have a DataFrame named 'transactions' with columns 'transaction_id', 'customer_id', 'amount', and 'timestamp'. You need to filter the DataFrame to keep only transactions with amounts greater than $100 and then sort the remaining transactions in descending order based on the timestamp. Which Python code snippet achieves this?

○
```
transactions.filter("amount > 100").sort("timestamp", ascending=False)
```

○
```
transactions.filter(transactions.amount > 100).sort(transactions.timestamp.desc)
```

○
```
transactions.filter("amount > 100").orderBy(transactions.timestamp.desc)
```

○
```
transactions.where("amount > 100").orderBy(transactions.timestamp, ascending=False)
```

○
```
transactions.filter(col("amount") > 100).sort(col("timestamp"), ascending=False)
```

A. Option A
B. Option B
C. Option C
D. Option D
E. Option E

Answer: E

Explanation:
The correct answer is E. Using the 'col' function to reference the column names within the filter and sort methods is the recommended approach in Spark. Option A uses string literals instead of column objects which is not recommended. Option B tries to access a nonexistent attribute 'desc’ on the column. Option C uses ‘orderBy’ instead of ‘sort' , which is functionally equivalent but uses a different syntax. Option D uses ‘where' instead of 'filter' , which again is equivalent but the preferred method is ‘filter for clarity.

Question: 10

You have a DataFrame named 'users' with columns 'user_id', 'name', 'city', and 'country'. You need to calculate the average age of users from each city. Which Python code snippet achieves this?

☐

```
users.groupBy("city").avg("age")
```

☐

```
users.groupBy("city").agg(avg("age"))
```

☐

```
users.groupBy("city").mean("age")
```

☐

```
users.select("city", avg("age")).groupBy("city").show()
```

☐

```
users.groupBy("city").agg(mean("age"))
```

A. Option A
B. Option B
C. Option C
D. Option D
E. Option E

Answer: B,E

Explanation:
The correct answers are B and E. Both options use the ‘groupBy' function to group users by their city and then apply the ‘avg’ or mean' aggregation function to calculate the average age. Option A does not correctly use the aggregation function, while Option C does not provide a method for calculating the mean. Option D attempts to select and group in separate steps, which is less efficient than the single step aggregation provided by ‘agg .

60 DAY MONEY BACK GUARANTEE
100%
SATISFACTION GUARANTEED