databricks

Databricks Exam Guide

Databricks Certified Associate Developer for Apache Spark

Provide Exam Guide Feedback

Purpose of this Exam Guide

This exam guide gives you an overview of the Certified Associate Developer for Apache Spark exam and what it covers to help you determine your exam readiness. **Please check back two weeks before you take your exam to make sure you have the most current version. This version covers the currently live version as of Oct 30, 2025.**

Audience Description

The Databricks Certified Associate Developer for Apache Spark certification exam assesses the understanding of the Apache Spark Architecture and Components and the ability to apply the Spark DataFrame API to complete basic data manipulation tasks within a Spark session. These tasks include selecting, renaming, and manipulating columns; filtering, dropping, sorting, and aggregating rows; handling missing data; combining, reading, writing, and partitioning DataFrames with schemas; and working with UDFs and Spark SQL functions. In addition, the exam will assess the basics of the Spark architecture, like execution/deployment modes, the execution hierarchy, fault tolerance, garbage collection, lazy evaluation, Shuffling and usage of Actions and broadcasting, Structured Streaming, Spark Connect, and common troubleshooting and tuning techniques. Individuals who pass this certification exam can be expected to complete basic Spark DataFrame tasks using Python.

About the Exam

- Number of items: 45 scored multiple-choice questions
- Time Limit: 90 minutes
- Delivery method: Online Proctored and test center proctored
- Prerequisite: None is required; related course attendance and six months of hands-on experience in Apache Spark™ are highly recommended.
- Validity: 2 years
- Test Aides: No Test Aides provided, including API Documentation.
- Recertification: Recertification is required every two years to maintain your certified status. To recertify, you must take the full exam that is currently live.
- Unscored questions: Exams may include unscored questions to gather statistical information for future use. These questions are not identified on the form and do not

impact your score. Additional time is factored into the exams to account for these questions.

Recommended Training

- Instructor-led: Apache Spark™ Programming with Databricks
- Self-paced (available in Databricks Academy):
    - Introduction to Apache Spark™
    - Developing Applications with Apache Spark™
    - Stream Processing and Analysis with Apache Spark™
    - Monitoring and Optimizing Apache Spark™ Workloads on Databricks

Exam Outline

Section 1: Apache Spark Architecture and Components

- Identify the advantages and challenges of implementing Spark.
- Identify the role of core components of Apache Spark™'s Architecture, including cluster, driver node, worker nodes/executors, CPU cores, and memory.
- Describe the architecture of Apache Spark™, including DataFrame and Dataset concepts, SparkSession lifecycle, caching, storage levels, and garbage collection.
- Explain the Apache Spark™ Architecture execution hierarchy..
- Configure Spark partitioning in distributed data processing, including shuffles and partitions
- Describe the execution patterns of the Apache Spark™ engine, including actions, transformations, and lazy evaluation.
- Identify the features of the Apache Spark Modules, including Core, Spark SQL, DataFrames, Pandas API on Spark, Structured Streaming, and MLib.

Section 2: Using Spark SQL

- Utilize common data sources such as JDBC, files, etc., to efficiently read from and write to Spark DataFrames using Spark SQL, including overwriting and partitioning by column.
- Execute SQL queries directly on files, including ORC Files, JSON Files, CSV Files, Text Files, and Delta Files, and understand the different save modes for outputting data in Spark SQL.
- Save data to persistent tables while applying sorting and partitioning to optimize data retrieval.
- Register DataFrames as temporary views in Spark SQL, allowing them to be queried with SQL syntax.

Section 3: Developing Apache Spark™ DataFrame/DataSet API Applications

- Manipulate columns, rows, and table structures by adding, dropping, splitting, renaming column names, applying filters, and exploding arrays.
- Perform data deduplication and validation operations on DataFrames.
- Perform aggregate operations on DataFrames such as count, approximate count distinct, and mean, summary.

- Manipulate and utilize Date data type, such as Unix epoch to date string, and extract date component.
- Combine DataFrames with operations such as Inner join, left join, broadcast join, multiple keys, cross join, union, and union all.
- Manage input and output operations by writing, overwriting, and reading DataFrames with schemas.
- Perform operations on DataFrames such as sorting, iterating, printing schema, and conversion between DataFrame and sequence/list formats.
- Create and invoke user-defined functions with or without stateful operators, including StateStores.
- Describe different types of variables in Spark, including broadcast variables and accumulators.
- Describe the purpose and implementation of broadcast joins

Section 4: Troubleshooting and Tuning Apache Spark DataFrame API Applications.

- Implement performance tuning strategies & optimize cluster utilization, including partitioning, repartitioning, coalescing, identifying data skew, and reducing shuffling
- Describe Adaptive Query Execution (AQE) and its benefits.
- Perform logging and monitoring of Spark applications - publish, customize, and analyze Driver logs and Executor logs to diagnose out-of-memory errors, cluster underutilization, etc.

Section 5: Structured Streaming

- Explain the Structured Streaming engine in Spark, including its functions, programming model, micro-batch processing, exactly-once semantics, and fault tolerance mechanisms.
- Create and write Streaming DataFrames and Streaming Datasets, including the basic output modes and output sinks.
- Perform basic operations on Streaming DataFrames and Streaming Datasets, such as selection, projection, window and aggregation.
- Perform Streaming Deduplication in Structured Streaming, both with and without watermark usage.

Section 6: Using Spark Connect to deploy applications

- Describe the features of Spark Connect.
- Describe the different deployment mode types (Client, Cluster, Local) in the Apache Spark™ environment.

Section 7: Using Pandas API on Spark

- Explain the advantages of using Pandas API on Spark.
- Create and invoke Pandas UDF.

Sample Questions

These questions are similar to actual question items and give you a general sense of how questions are asked on this exam. They include exam objectives as they are stated in the exam guide and give you a sample question that aligns with the objective. The exam guide lists all of the objectives that could be covered on an exam. The best way to prepare for a certification exam is to review the exam outline in the exam guide.

Question 1

Objective - Utilize common data sources such as JDBC, files, etc. to efficiently read from and write to Spark DataFrames using SparkSQL, including overwrite and partitioning by column.

A data engineer needs to write a DataFrame df to a Parquet file, partitioned by the column country, and overwrite any existing data at the destination path.

Which code should the data engineer use to accomplish this task in Apache Spark?

A. `df.write.mode("append").partitionBy("country").parquet("/data/output")`
B. `df.write.partitionBy("country").parquet("/data/output")`
C. `df.write.mode("overwrite").parquet("/data/output")`
D.
`df.write.mode("overwrite").partitionBy("country").parquet("/data/output")`

Question 2

Objective - Perform logging and monitoring of Spark applications - publish, customize, and analyze Driver logs and Executor logs to diagnose out-of-memory errors, cluster underutilization, etc.

An engineer notices a significant increase in the job execution time during the execution of a Spark job. After some investigation, the engineer decides to check the logs produced by the Executors.

How should the engineer retrieve the Executor logs to diagnose performance issues in the Spark application?

A. Use the command 'spark-submit' with the '**`--verbose`**' flag to print the logs to the console.
B. Locate the executor logs on the Spark master node, typically under the /tmp directory.
C. Use the Spark UI to select the stage and view the executor logs directly from the stages tab.
D. Fetch the logs by running a Spark job with the `spark-sql` CLI tool.

Question 3:

Objective: Manipulate columns, rows, and table structures by adding, dropping, splitting, renaming column names, applying filters, and exploding arrays.

Which code block will replace the **`division`** column in the **`storesDF`** DataFrame with a new column named **`state`**, and simultaneously replace and rename the **`mName`** column to **`managerName`** in the resulting DataFrame?

A.
```
storesDF = (storesDF.withColumn("state", col("division"))
            .drop("division")
            .withColumnRenamed("mName", "managerName"))
```

B.
```
storesDF = (storesDF.withColumnRenamed("state", "division")

            .withColumnRenamed("mName", "managerName"))
```

C.
```
storesDF = (storesDF.withColumn(col("division"), state)

            .withColumnRenamed("managerName", "mName"))
```

D.
```
storesDF = (storesDF .drop("division")

        .withColumnRenamed("state", lit("default_state"))

        .withColumnRenamed(columns={"mName": "managerName"}))
```

Question 4:

Objective: Perform operations on DataFrames such as sorting, iterating, printing schema, and conversion between DataFrame and sequence/list formats.

Given a DataFrame **`employeeDF`** with columns: **`name`**, **`department`**, **`salary`**, and **`age`**.

Which code snippet sorts a DataFrame by multiple columns in descending order, printing its schema, and converting the DataFrame to a list of rows?

A.
```
result = employeeDF.orderBy(desc("salary"), desc("age"))
    employeeDF.printSchema()
    row_list = result.toPandas().values.tolist()
```

B.
```
result = employeeDF.sort("salary", "age", ascending=[False,
         False]) employeeDF.schema()

row_list = result.toList()
```

C.
```
result = (employeeDF .orderBy(col("salary").desc(),
            col("age").desc()) .collect())
employeeDF.printSchema()
row_list = [row.asDict() for row in result]
```

D.
```
result = (employeeDF
            .sort(["salary", "age"], descending=True)
            .show())
employeeDF.describe()
row_list = list(result)
```

Question 5:

Objective: Configure Spark partitioning in distributed data processing, including shuffles and partitions.

What will be the impact of setting the default value of spark.sql.shuffle.partitions to 200?

A. DataFrames will be divided into 200 distinct partitions during data shuffling operations.
B. New DataFrames created by Spark will be partitioned to utilize the memory of 200 executors optimally.
C. All DataFrames in Spark will be partitioned to occupy the memory of 200 executors perfectly.
D. Spark will only process the first 200 partitions of DataFrames to enhance performance.

Question 6:

Objective: Perform data deduplication and validation operations on DataFrames.

Which code fragment will return a new DataFrame from **`storesDF`** that excludes all rows containing at least one missing value in any column?

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

Question 7:

Objective: Describe the different deployment mode types (Client, Cluster, Local) in Apache Spark environment.

Which Spark deployment mode requires all executors to run on a single worker node?

A. Cluster mode
B. Local mode
C. Client mode
D. Standard mode

Question 8:

Objective: Explain the Structured Streaming engine in Spark, including its functions, programming model, micro-batch processing, exactly-once semantics, and fault tolerance mechanisms.

A developer must calculate real-time, rolling metrics like "average session duration in the last hour" and "top 10 products viewed in the last 15 minutes" from continuous clickstream data for recommendation engines. These metrics must update every 2 minutes. They chose Streaming DataFrames over traditional batch DataFrames.

Why should the developer consider Streaming DataFrames in this use case?

A. Streaming DataFrames automatically handle data partitioning and caching more efficiently than batch DataFrames, making them faster for processing large volumes of clickstream data, even when processing historical data.
B. Streaming DataFrames enable continuous data processing with incremental updates to aggregations, allowing real-time metrics without reprocessing the entire dataset every 2 minutes.
C. Streaming DataFrames provide better error handling and automatic retry mechanisms than batch DataFrames, which is essential for handling unreliable network connections from the network recovery mechanism.
D. Streaming DataFrames use more memory than batch DataFrames because they process data in batches, making them ideal for this scenario.

Question 9:

Objective: Create and invoke user-defined functions with or without stateful operators, including StateStores.

A developer is building a streaming application. The application's validation logic involves computations and string manipulations unavailable in built-in Spark functions. The developer wants to ensure the streaming job is stateless for each transaction and can scale horizontally without maintaining customer history.

When should the developer use user-defined functions (UDFs) without stateful operators for this transaction validation pipeline?

A. Maintain running totals and compare them to historical averages in state.
B. Perform windowed aggregations over the last 24 hours with incremental updates.
C. Detect session-based patterns by tracking sequences across events.

D. Apply custom logic without relying on previous transaction data or state between transaction batches

Question 10:

Objective: *Perform aggregate operations on DataFrames such as count, approximate count distinct, and mean, summary.*

A developer needs a dashboard for a mobile app's 50 million user activity records, showing total event and unique user counts by time. The dashboard refreshes hourly, and management prioritizes speed and accepts a 2-3% error margin for unique user counts.

Why should the developer choose **`approx_count_distinct()`** over **`count(distinct())`**?

A. **`approx_count_distinct()`** offers improved accuracy over **`count(distinct())`** for large datasets due to its advanced statistical algorithms.
B. **`approx_count_distinct()`** handles nulls and missing data better than **`count(distinct())`**, which is crucial for incomplete user activity logs.
C. **`approx_count_distinct()`** uses probabilistic algorithms (HyperLogLog) to boost performance by avoiding costly shuffle operations.
D. **`approx_count_distinct()`** uses less memory per partition than **`count(distinct())`** because it stores aggregated results in compressed format,

Answers:

1. D
2. C
3. A
4. C
5. A
6. C
7. B
8. B
9. D
10. C