

All about Data Engineering

Supercharge your Databricks learning

What's inside?

Certification and
Interview Resources

All the best!





Practice Exam

Databricks Certified Data Engineer Associate



Overview

This is a practice exam for the [Databricks Certified Data Engineer Associate](#) exam. The questions here are retired questions from the actual exam that are representative of the questions one will receive while taking the actual exam. After taking this practice exam, one should know what to expect while taking the actual Data Engineer Associate exam.

Just like the actual exam, it contains 45 multiple-choice questions. Each of these questions has one correct answer. The correct answer for each question is listed at the bottom in the **Correct Answers** section.

There are a few more things to be aware of:

1. There is a 90-minute time limit to take the actual exam.
2. In order to pass the actual exam, testers will need to correctly answer at least 32 of the 45 questions.
3. Testers will not have access to any documentation or Databricks environments during the exam.
4. These questions are representative of questions that are on the actual exam, but they are no longer on the actual exam.

If you have more questions, please review the [Databricks Academy Certification FAQ](#).

Once you've completed the practice exam, evaluate your score using the correct answers at the bottom of this document. If you're ready to take the exam, head to the [Databricks Certification portal](#) to register.

Questions

Question 1

Which of the following describes a benefit of a data lakehouse that is unavailable in a traditional data warehouse?

- A. A data lakehouse provides a relational system of data management.
- B. A data lakehouse captures snapshots of data for version control purposes.
- C. A data lakehouse couples storage and compute for complete control.
- D. A data lakehouse utilizes proprietary storage formats for data.
- E. A data lakehouse enables both batch and streaming analytics.

Question 2

Which of the following locations hosts the driver and worker nodes of a Databricks-managed cluster?

- A. Data plane
- B. Control plane
- C. Databricks Filesystem
- D. JDBC data source
- E. Databricks web application

Question 3

A data architect is designing a data model that works for both video-based machine learning workloads and highly audited batch ETL/ELT workloads.

Which of the following describes how using a data lakehouse can help the data architect meet the needs of both workloads?

- A. A data lakehouse requires very little data modeling.
- B. A data lakehouse combines compute and storage for simple governance.
- C. A data lakehouse provides autoscaling for compute clusters.
- D. A data lakehouse stores unstructured data and is ACID-compliant.
- E. A data lakehouse fully exists in the cloud.

Question 4

Which of the following describes a scenario in which a data engineer will want to use a Job cluster instead of an all-purpose cluster?

- A. An ad-hoc analytics report needs to be developed while minimizing compute costs.
- B. A data team needs to collaborate on the development of a machine learning model.
- C. An automated workflow needs to be run every 30 minutes.
- D. A Databricks SQL query needs to be scheduled for upward reporting.
- E. A data engineer needs to manually investigate a production error.

Question 5

A data engineer has created a Delta table as part of a data pipeline. Downstream data analysts now need SELECT permission on the Delta table.

Assuming the data engineer is the Delta table owner, which part of the Databricks Lakehouse Platform can the data engineer use to grant the data analysts the appropriate access?

- A. Repos
- B. Jobs
- C. Data Explorer
- D. Databricks Filesystem
- E. Dashboards

Question 6

Two junior data engineers are authoring separate parts of a single data pipeline notebook. They are working on separate Git branches so they can pair program on the same notebook simultaneously. A senior data engineer experienced in Databricks suggests there is a better alternative for this type of collaboration.

Which of the following supports the senior data engineer's claim?

- A. Databricks Notebooks support automatic change-tracking and versioning
- B. Databricks Notebooks support real-time coauthoring on a single notebook
- C. Databricks Notebooks support commenting and notification comments
- D. Databricks Notebooks support the use of multiple languages in the same notebook
- E. Databricks Notebooks support the creation of interactive data visualizations

Question 7

Which of the following describes how Databricks Repos can help facilitate CI/CD workflows on the Databricks Lakehouse Platform?

- A. Databricks Repos can facilitate the pull request, review, and approval process before merging branches
- B. Databricks Repos can merge changes from a secondary Git branch into a main Git branch
- C. Databricks Repos can be used to design, develop, and trigger Git automation pipelines
- D. Databricks Repos can store the single-source-of-truth Git repository
- E. Databricks Repos can commit or push code changes to trigger a CI/CD process

Question 8

Which of the following statements describes Delta Lake?

- A. Delta Lake is an open source analytics engine used for big data workloads.
- B. Delta Lake is an open format storage layer that delivers reliability, security, and performance.
- C. Delta Lake is an open source platform to help manage the complete machine learning lifecycle.
- D. Delta Lake is an open source data storage format for distributed data.
- E. Delta Lake is an open format storage layer that processes data.

Question 9

A data architect has determined that a table of the following format is necessary:

id	birthDate	avgRating
a1	1990-01-06	5.5
a2	1974-11-21	7.1
...

Which of the following code blocks uses SQL DDL commands to create an empty Delta table in the above format regardless of whether a table already exists with this name?

- A.

```
CREATE OR REPLACE TABLE table_name AS
SELECT
  id STRING,
  birthDate DATE,
  avgRating FLOAT
USING DELTA
```
- B.

```
CREATE OR REPLACE TABLE table_name (
  id STRING,
  birthDate DATE,
  avgRating FLOAT
)
```
- C.

```
CREATE TABLE IF NOT EXISTS table_name (
  id STRING,
  birthDate DATE,
  avgRating FLOAT
)
```
- D.

```
CREATE TABLE table_name AS
SELECT
  id STRING,
  birthDate DATE,
  avgRating FLOAT
```
- E.

```
CREATE OR REPLACE TABLE table_name WITH COLUMNS (
  id STRING,
  birthDate DATE,
  avgRating FLOAT
) USING DELTA
```

Question 10

Which of the following SQL keywords can be used to append new rows to an existing Delta table?

- A. UPDATE
- B. COPY
- C. INSERT INTO
- D. DELETE
- E. UNION

Question 11

A data engineering team needs to query a Delta table to extract rows that all meet the same condition. However, the team has noticed that the query is running slowly. The team has already tuned the size of the data files. Upon investigating, the team has concluded that the rows meeting the condition are sparsely located throughout each of the data files.

Based on the scenario, which of the following optimization techniques could speed up the query?

- A. Data skipping
- B. Z-Ordering
- C. Bin-packing
- D. Write as a Parquet file
- E. Tuning the file size

Question 12

A data engineer needs to create a database called **customer360** at the location **/customer/customer360**. The data engineer is unsure if one of their colleagues has already created the database.

Which of the following commands should the data engineer run to complete this task?

- A. `CREATE DATABASE customer360 LOCATION '/customer/customer360';`
- B. `CREATE DATABASE IF NOT EXISTS customer360;`
- C. `CREATE DATABASE IF NOT EXISTS customer360 LOCATION '/customer/customer360';`
- D. `CREATE DATABASE IF NOT EXISTS customer360 DELTA LOCATION '/customer/customer360';`
- E. `CREATE DATABASE customer360 DELTA LOCATION '/customer/customer360';`

Question 13

A junior data engineer needs to create a Spark SQL table **my_table** for which Spark manages both the data and the metadata. The metadata and data should also be stored in the Databricks Filesystem (DBFS).

Which of the following commands should a senior data engineer share with the junior data engineer to complete this task?

- A. `CREATE TABLE my_table (id STRING, value STRING) USING org.apache.spark.sql.parquet OPTIONS (PATH "storage-path");`
- B. `CREATE MANAGED TABLE my_table (id STRING, value STRING) USING org.apache.spark.sql.parquet OPTIONS (PATH "storage-path");`
- C. `CREATE MANAGED TABLE my_table (id STRING, value STRING);`
- D. `CREATE TABLE my_table (id STRING, value STRING) USING DBFS;`
- E. `CREATE TABLE my_table (id STRING, value STRING);`

Question 14

A data engineer wants to create a relational object by pulling data from two tables. The relational object must be used by other data engineers in other sessions. In order to save on storage costs, the data engineer wants to avoid copying and storing physical data.

Which of the following relational objects should the data engineer create?

- A. View
- B. Temporary view
- C. Delta Table
- D. Database
- E. Spark SQL Table

Question 15

A data engineering team has created a series of tables using Parquet data stored in an external system. The team is noticing that after appending new rows to the data in the external system, their queries within Databricks are not returning the new rows. They identify the caching of the previous data as the cause of this issue.

Which of the following approaches will ensure that the data returned by queries is always up-to-date?

- A. The tables should be converted to the Delta format
- B. The tables should be stored in a cloud-based external system
- C. The tables should be refreshed in the writing cluster before the next query is run
- D. The tables should be altered to include metadata to not cache
- E. The tables should be updated before the next query is run

Question 16

A table **customerLocations** exists with the following schema:

```
id STRING,  
date STRING,  
city STRING,  
country STRING
```

A senior data engineer wants to create a new table from this table using the following command:

```
CREATE TABLE customersPerCountry AS  
SELECT country,  
       COUNT(*) AS customers  
FROM customerLocations  
GROUP BY country;
```

A junior data engineer asks why the schema is not being declared for the new table.

Which of the following responses explains why declaring the schema is not necessary?

- A. CREATE TABLE AS SELECT statements adopt schema details from the source table and query.
- B. CREATE TABLE AS SELECT statements infer the schema by scanning the data.
- C. CREATE TABLE AS SELECT statements result in tables where schemas are optional.
- D. CREATE TABLE AS SELECT statements assign all columns the type STRING.
- E. CREATE TABLE AS SELECT statements result in tables that do not support schemas.

Question 17

A data engineer is overwriting data in a table by deleting the table and recreating the table. Another data engineer suggests that this is inefficient and the table should simply be overwritten instead.

Which of the following reasons to overwrite the table instead of deleting and recreating the table is incorrect?

- A. Overwriting a table is efficient because no files need to be deleted.
- B. Overwriting a table results in a clean table history for logging and audit purposes.
- C. Overwriting a table maintains the old version of the table for Time Travel.
- D. Overwriting a table is an atomic operation and will not leave the table in an unfinished state.
- E. Overwriting a table allows for concurrent queries to be completed while in progress.

Question 18

Which of the following commands will return records from an existing Delta table `my_table` where duplicates have been removed?

- A. `DROP DUPLICATES FROM my_table;`
- B. `SELECT * FROM my_table WHERE duplicate = False;`
- C. `SELECT DISTINCT * FROM my_table;`
- D. `MERGE INTO my_table a USING new_records b ON a.id = b.id WHEN NOT MATCHED THEN INSERT *;`
- E. `MERGE INTO my_table a USING new_records b;`

Question 19

A data engineer wants to horizontally combine two tables as a part of a query. They want to use a shared column as a key column, and they only want the query result to contain rows whose value in the key column is present in both tables.

Which of the following SQL commands can they use to accomplish this task?

- A. `INNER JOIN`
- B. `OUTER JOIN`
- C. `LEFT JOIN`
- D. `MERGE`
- E. `UNION`

Question 20

A junior data engineer has ingested a JSON file into a table **raw_table** with the following schema:

```
cart_id STRING,  
items ARRAY<item_id:STRING>
```

The junior data engineer would like to unnest the **items** column in **raw_table** to result in a new table with the following schema:

```
cart_id STRING,  
item_id STRING
```

Which of the following commands should the junior data engineer run to complete this task?

- A. `SELECT cart_id, filter(items) AS item_id FROM raw_table;`
- B. `SELECT cart_id, flatten(items) AS item_id FROM raw_table;`
- C. `SELECT cart_id, reduce(items) AS item_id FROM raw_table;`
- D. `SELECT cart_id, explode(items) AS item_id FROM raw_table;`
- E. `SELECT cart_id, slice(items) AS item_id FROM raw_table;`

Question 21

A data engineer has ingested a JSON file into a table **raw_table** with the following schema:

```
transaction_id STRING,  
payload ARRAY<customer_id:STRING, date:TIMESTAMP, store_id:STRING>
```

The data engineer wants to efficiently extract the date of each transaction into a table with the following schema:

```
transaction_id STRING,  
date TIMESTAMP
```

Which of the following commands should the data engineer run to complete this task?

- A. `SELECT transaction_id, explode(payload) FROM raw_table;`
- B. `SELECT transaction_id, payload.date FROM raw_table;`
- C. `SELECT transaction_id, date FROM raw_table;`

- D. `SELECT transaction_id, payload[date] FROM raw_table;`
- E. `SELECT transaction_id, date from payload FROM raw_table;`

Question 22

A data analyst has provided a data engineering team with the following Spark SQL query:

```
SELECT district,
       avg(sales)
FROM store_sales_20220101
GROUP BY district;
```

The data analyst would like the data engineering team to run this query every day. The date at the end of the table name (20220101) should automatically be replaced with the current date each time the query is run.

Which of the following approaches could be used by the data engineering team to efficiently automate this process?

- A. They could wrap the query using PySpark and use Python's string variable system to automatically update the table name.
- B. They could manually replace the date within the table name with the current day's date.
- C. They could request that the data analyst rewrites the query to be run less frequently.
- D. They could replace the string-formatted date in the table with a timestamp-formatted date.
- E. They could pass the table into PySpark and develop a robustly tested module on the existing query.

Question 23

A data engineer has ingested data from an external source into a PySpark DataFrame `raw_df`. They need to briefly make this data available in SQL for a data analyst to perform a quality assurance check on the data.

Which of the following commands should the data engineer run to make this data available in SQL for only the remainder of the Spark session?

- A. `raw_df.createOrReplaceTempView("raw_df")`
- B. `raw_df.createTable("raw_df")`
- C. `raw_df.write.save("raw_df")`
- D. `raw_df.saveAsTable("raw_df")`

- E. There is no way to share data between PySpark and SQL.

Question 24

A data engineer needs to dynamically create a table name string using three Python variables: **region**, **store**, and **year**. An example of a table name is below when **region** = "nyc", **store** = "100", and **year** = "2021":

nyc100_sales_2021

Which of the following commands should the data engineer use to construct the table name in Python?

- A. "{region}+{store}+_sales_{year}"
- B. f"{region}+{store}+_sales_{year}"
- C. "{region}{store}_sales_{year}"
- D. f"{region}{store}_sales_{year}"
- E. {region}+{store}+"_sales_" + {year}

Question 25

A data engineer has developed a code block to perform a streaming read on a data source. The code block is below:

```
(spark
    .read
    .schema(schema)
    .format("cloudFiles")
    .option("cloudFiles.format", "json")
    .load(dataSource)
)
```

The code block is returning an error.

Which of the following changes should be made to the code block to configure the block to successfully perform a streaming read?

- A. The **.read** line should be replaced with **.readStream**.
- B. A new **.stream** line should be added after the **.read** line.
- C. The **.format("cloudFiles")** line should be replaced with **.format("stream")**.
- D. A new **.stream** line should be added after the **spark** line.
- E. A new **.stream** line should be added after the **.load(dataSource)** line.

Question 26

A data engineer has configured a Structured Streaming job to read from a table, manipulate the data, and then perform a streaming write into a new table.

The code block used by the data engineer is below:

```
(spark.table("sales")
  .withColumn("avg_price", col("sales") / col("units"))
  .writeStream
  .option("checkpointLocation", checkpointPath)
  .outputMode("complete")
  ._____
  .table("new_sales")
)
```

If the data engineer only wants the query to execute a single micro-batch to process all of the available data, which of the following lines of code should the data engineer use to fill in the blank?

- A. `trigger(once=True)`
- B. `trigger(continuous="once")`
- C. `processingTime("once")`
- D. `trigger(processingTime="once")`
- E. `processingTime(1)`

Question 27

A data engineer is designing a data pipeline. The source system generates files in a shared directory that is also used by other processes. As a result, the files should be kept as is and will accumulate in the directory. The data engineer needs to identify which files are new since the previous run in the pipeline, and set up the pipeline to only ingest those new files with each run.

Which of the following tools can the data engineer use to solve this problem?

- A. Databricks SQL
- B. Delta Lake
- C. Unity Catalog
- D. Data Explorer
- E. Auto Loader

Question 28

A data engineering team is in the process of converting their existing data pipeline to utilize Auto Loader for incremental processing in the ingestion of JSON files. One data engineer comes across the following code block in the Auto Loader documentation:

```
(streaming_df = spark.readStream.format("cloudFiles")
    .option("cloudFiles.format", "json")
    .option("cloudFiles.schemaLocation", schemaLocation)
    .load(sourcePath))
```

Assuming that **schemaLocation** and **sourcePath** have been set correctly, which of the following changes does the data engineer need to make to convert this code block to use Auto Loader to ingest the data?

- A. The data engineer needs to change the **format("cloudFiles")** line to **format("autoLoader")**.
- B. There is no change required. Databricks automatically uses Auto Loader for streaming reads.
- C. There is no change required. The inclusion of **format("cloudFiles")** enables the use of Auto Loader.
- D. The data engineer needs to add the **.autoLoader** line before the **.load(sourcePath)** line.
- E. There is no change required. The data engineer needs to ask their administrator to turn on Auto Loader.

Question 29

Which of the following data workloads will utilize a Bronze table as its source?

- A. A job that aggregates cleaned data to create standard summary statistics
- B. A job that queries aggregated data to publish key insights into a dashboard
- C. A job that ingests raw data from a streaming source into the Lakehouse
- D. A job that develops a feature set for a machine learning application
- E. A job that enriches data by parsing its timestamps into a human-readable format

Question 30

Which of the following data workloads will utilize a Silver table as its source?

- A. A job that enriches data by parsing its timestamps into a human-readable format
- B. A job that queries aggregated data that already feeds into a dashboard
- C. A job that ingests raw data from a streaming source into the Lakehouse
- D. A job that aggregates cleaned data to create standard summary statistics
- E. A job that cleans data by removing malformed records

Question 31

Which of the following Structured Streaming queries is performing a hop from a Bronze table to a Silver table?

- A.

```
(spark.table("sales")
  .groupBy("store")
  .agg(sum("sales"))
  .writeStream
  .option("checkpointLocation", checkpointPath)
  .outputMode("complete")
  .table("aggregatedSales")
)
```
- B.

```
(spark.table("sales")
  .agg(sum("sales"),
       sum("units"))
  .writeStream
  .option("checkpointLocation", checkpointPath)
  .outputMode("complete")
  .table("aggregatedSales")
)
```
- C.

```
(spark.table("sales")
  .withColumn("avgPrice", col("sales") / col("units"))
  .writeStream
  .option("checkpointLocation", checkpointPath)
  .outputMode("append")
  .table("cleanedSales")
)
```
- D.

```
(spark.readStream.load(rawSalesLocation)
  .writeStream
  .option("checkpointLocation", checkpointPath)
  .outputMode("append")
  .table("uncleanedSales")
)
```
- E.

```
(spark.read.load(rawSalesLocation)
  .writeStream
```



```
        .option("checkpointLocation", checkpointPath)
        .outputMode("append")
        .table("uncleanedSales")
    )
```

Question 32

Which of the following benefits does Delta Live Tables provide for ELT pipelines over standard data pipelines that utilize Spark and Delta Lake on Databricks?

- A. The ability to declare and maintain data table dependencies
- B. The ability to write pipelines in Python and/or SQL
- C. The ability to access previous versions of data tables
- D. The ability to automatically scale compute resources
- E. The ability to perform batch and streaming queries

Question 33

A data engineer has three notebooks in an ELT pipeline. The notebooks need to be executed in a specific order for the pipeline to complete successfully. The data engineer would like to use Delta Live Tables to manage this process.

Which of the following steps must the data engineer take as part of implementing this pipeline using Delta Live Tables?

- A. They need to create a Delta Live Tables pipeline from the Data page.
- B. They need to create a Delta Live Tables pipeline from the Jobs page.
- C. They need to create a Delta Live tables pipeline from the Compute page.
- D. They need to refactor their notebook to use Python and the dlt library.
- E. They need to refactor their notebook to use SQL and **CREATE LIVE TABLE** keyword.

Question 34

A data engineer has written the following query:

```
SELECT *
FROM json.`/path/to/json/file.json`;
```

The data engineer asks a colleague for help to convert this query for use in a Delta Live Tables (DLT) pipeline. The query should create the first table in the DLT pipeline.

Which of the following describes the change the colleague needs to make to the query?

- A. They need to add a **COMMENT** line at the beginning of the query.
- B. They need to add a **CREATE LIVE TABLE table_name AS** line at the beginning of the query.
- C. They need to add a **live.** prefix prior to **json.** in the **FROM** line.
- D. They need to add a **CREATE DELTA LIVE TABLE table_name AS** line at the beginning of the query.
- E. They need to add the **cloud_files(...)** wrapper to the JSON file path.

Question 35

A dataset has been defined using Delta Live Tables and includes an expectations clause:

```
CONSTRAINT valid_timestamp EXPECT (timestamp > '2020-01-01')
```

What is the expected behavior when a batch of data containing data that violates these constraints is processed?

- A. Records that violate the expectation are added to the target dataset and recorded as invalid in the event log.
- B. Records that violate the expectation are dropped from the target dataset and recorded as invalid in the event log.
- C. Records that violate the expectation cause the job to fail.
- D. Records that violate the expectation are added to the target dataset and flagged as invalid in a field added to the target dataset.
- E. Records that violate the expectation are dropped from the target dataset and loaded into a quarantine table.

Question 36

A Delta Live Table pipeline includes two datasets defined using **STREAMING LIVE TABLE**. Three datasets are defined against Delta Lake table sources using **LIVE TABLE**.

The table is configured to run in Development mode using the Triggered Pipeline Mode.

Assuming previously unprocessed data exists and all definitions are valid, what is the expected outcome after clicking Start to update the pipeline?

- A. All datasets will be updated once and the pipeline will shut down. The compute resources will be terminated.
- B. All datasets will be updated at set intervals until the pipeline is shut down. The compute resources will be deployed for the update and terminated when the pipeline is stopped.
- C. All datasets will be updated at set intervals until the pipeline is shut down. The compute resources will persist after the pipeline is stopped to allow for additional testing.
- D. All datasets will be updated once and the pipeline will shut down. The compute resources will persist to allow for additional testing.
- E. All datasets will be updated continuously and the pipeline will not shut down. The compute resources will persist with the pipeline.

Question 37

A data engineer has a Job with multiple tasks that runs nightly. One of the tasks unexpectedly fails during 10 percent of the runs.

Which of the following actions can the data engineer perform to ensure the Job completes each night while minimizing compute costs?

- A. They can institute a retry policy for the entire Job
- B. They can observe the task as it runs to try and determine why it is failing
- C. They can set up the Job to run multiple times ensuring that at least one will complete
- D. They can institute a retry policy for the task that periodically fails
- E. They can utilize a Jobs cluster for each of the tasks in the Job

Question 38

A data engineer has set up two Jobs that each run nightly. The first Job starts at 12:00 AM, and it usually completes in about 20 minutes. The second Job depends on the first Job, and it starts at 12:30 AM. Sometimes, the second Job fails when the first Job does not complete by 12:30 AM.

Which of the following approaches can the data engineer use to avoid this problem?

- A. They can utilize multiple tasks in a single job with a linear dependency
- B. They can use cluster pools to help the Jobs run more efficiently
- C. They can set up a retry policy on the first Job to help it run more quickly
- D. They can limit the size of the output in the second Job so that it will not fail as easily
- E. They can set up the data to stream from the first Job to the second Job

Question 39

A data engineer has set up a notebook to automatically process using a Job. The data engineer's manager wants to version control the schedule due to its complexity.

Which of the following approaches can the data engineer use to obtain a version-controllable configuration of the Job's schedule?

- A. They can link the Job to notebooks that are a part of a Databricks Repo.
- B. They can submit the Job once on a Job cluster.
- C. They can download the JSON description of the Job from the Job's page.
- D. They can submit the Job once on an all-purpose cluster.
- E. They can download the XML description of the Job from the Job's page.

Question 40

A data analyst has noticed that their Databricks SQL queries are running too slowly. They claim that this issue is affecting all of their sequentially run queries. They ask the data engineering team for help. The data engineering team notices that each of the queries uses the same SQL endpoint, but the SQL endpoint is not used by any other user.

Which of the following approaches can the data engineering team use to improve the latency of the data analyst's queries?

- A. They can turn on the Serverless feature for the SQL endpoint.
- B. They can increase the maximum bound of the SQL endpoint's scaling range.
- C. They can increase the cluster size of the SQL endpoint.
- D. They can turn on the Auto Stop feature for the SQL endpoint.
- E. They can turn on the Serverless feature for the SQL endpoint and change the Spot Instance Policy to "Reliability Optimized."

Question 41

An engineering manager uses a Databricks SQL query to monitor their team's progress on fixes related to customer-reported bugs. The manager checks the results of the query every day, but they are manually rerunning the query each day and waiting for the results.

Which of the following approaches can the manager use to ensure the results of the query are updated each day?

- A. They can schedule the query to run every 1 day from the Jobs UI.
- B. They can schedule the query to refresh every 1 day from the query's page in Databricks SQL.
- C. They can schedule the query to run every 12 hours from the Jobs UI.
- D. They can schedule the query to refresh every 1 day from the SQL endpoint's page in Databricks SQL.
- E. They can schedule the query to refresh every 12 hours from the SQL endpoint's page in Databricks SQL.

Question 42

A data engineering team has been using a Databricks SQL query to monitor the performance of an ELT job. The ELT job is triggered by a specific number of input records being ready to process. The Databricks SQL query returns the number of minutes since the job's most recent runtime.

Which of the following approaches can enable the data engineering team to be notified if the ELT job has not been run in an hour?

- A. They can set up an Alert for the accompanying dashboard to notify them if the returned value is greater than 60.
- B. They can set up an Alert for the query to notify when the ELT job fails.
- C. They can set up an Alert for the accompanying dashboard to notify when it has not refreshed in 60 minutes.
- D. They can set up an Alert for the query to notify them if the returned value is greater than 60.
- E. This type of alerting is not possible in Databricks.

Question 43

A data engineering manager has noticed that each of the queries in a Databricks SQL dashboard takes a few minutes to update when they manually click the "Refresh" button. They are curious why this might be occurring, so a team member provides a variety of reasons on why the delay might be occurring.

Which of the following reasons *fails* to explain why the dashboard might be taking a few minutes to update?

- A. The SQL endpoint being used by each of the queries might need a few minutes to start up.
- B. The queries attached to the dashboard might take a few minutes to run under normal circumstances.

- C. The queries attached to the dashboard might first be checking to determine if new data is available.
- D. The Job associated with updating the dashboard might be using a non-pooled endpoint.
- E. The queries attached to the dashboard might all be connected to their own, unstarted Databricks clusters.

Question 44

A new data engineer has started at a company. The data engineer has recently been added to the company's Databricks workspace as **new.engineer@company.com**. The data engineer needs to be able to query the table **sales** in the database **retail**. The new data engineer already has been granted **USAGE** on the database **retail**.

Which of the following commands can be used to grant the appropriate permissions to the new data engineer?

- A. `GRANT USAGE ON TABLE sales TO new.engineer@company.com;`
- B. `GRANT CREATE ON TABLE sales TO new.engineer@company.com;`
- C. `GRANT SELECT ON TABLE sales TO new.engineer@company.com;`
- D. `GRANT USAGE ON TABLE new.engineer@company.com TO sales;`
- E. `GRANT SELECT ON TABLE new.engineer@company.com TO sales;`

Question 45

A new data engineer **new.engineer@company.com** has been assigned to an ELT project. The new data engineer will need full privileges on the table **sales** to fully manage the project.

Which of the following commands can be used to grant full permissions on the table to the new data engineer?

- A. `GRANT ALL PRIVILEGES ON TABLE sales TO new.engineer@company.com;`
- B. `GRANT USAGE ON TABLE sales TO new.engineer@company.com;`
- C. `GRANT ALL PRIVILEGES ON TABLE new.engineer@company.com TO sales;`
- D. `GRANT SELECT ON TABLE sales TO new.engineer@company.com;`
- E. `GRANT SELECT CREATE MODIFY ON TABLE sales TO new.engineer@company.com;`

Correct Answers

1. E
2. A
3. D
4. C
5. C
6. B
7. E
8. B
9. B
10. C
11. B
12. C
13. E
14. A
15. A
16. A
17. B
18. C
19. A
20. D
21. B
22. A
23. A
24. D
25. A
26. A
27. E
28. C
29. E
30. D
31. C
32. A
33. B
34. B
35. A
36. D
37. D
38. A
39. C
40. C

- 41. B
- 42. D
- 43. D
- 44. C
- 45. A

Data Engineering Cert Prep - Notes

1. Databricks basics:

- 1.1. Notebook basics - How to create a cluster, attach a notebook to a cluster, run different languages (**%sql**, **%python**, **%scala**) and use db utils (**dbutils.fs.ls**) for utility functions, magic commands(**%**) and markdown language (**%md**) to take notes within databricks notebooks

2. Delta Lake

- 2.1. Managing Delta Tables : **create table, insert into, select from, update and delete, drop** table queries can be used for delta lake operations. Default mode of operation in DBR 8+
- 2.2. Merge - Used for **Upserts (Updates + Inserts)**
Syntax - **<<Merge INTO** Destination Table using source table using conditions **When matched When not matched >>**
MERGE statements must have at least one field to match on, and each **WHEN MATCHED** or **WHEN NOT MATCHED** clause can have any number of additional conditional statements
- 2.3. **Advanced Options**
 - 2.3.1. **OPTIMIZE** - Used to compact the files.**OPTIMIZE** will replace existing data files by combining records and rewriting the results.
 - 2.3.2. **Z ORDER** - Used to index the file (partition and colocate the files). It speeds up data retrieval when filtering on provided fields by collocating data with similar values within data files.
 - 2.3.3. **VACUUM** - Used to delete the stale data
 - Defaults to 7 days
 - Parameters for Vacuum retention duration and logging
 - SET **spark.databricks.delta.retentionDurationCheck.enabled = false** to disable the check;
 - SET **spark.databricks.delta.vacuum.logging.enabled = true** to enable the logging;
 - Exception Note on Vacuum behavior - Because Delta Cache stores copies of files queried in the current session on storage volumes deployed to your currently active cluster, you may still be able to temporarily access previous table versions after executing vacuum (though systems should **not** be designed to expect this behavior).Restarting the cluster will ensure that these cached data files are permanently purged.
- 2.4. **Time travel - Key functions**

- 2.4.1. **VERSION** - HISTORY/TIME TRAVEL (Using Version and Time travel)
- 2.4.2. **DRY RUN** - Shows the files to be deleted before actually performing the operation
- 2.4.3. **RESTORE** - Used to rollback the previous version of a table


3. Relational Entities

3.1. Databases & Tables

- 3.1.1. **Location** - Impacts the default storage location. Difference between external vs managed table
- 3.1.2. **Default** - By default, managed tables in a database without the location specified will be created in the hive directory i.e `*`dbfs:/user/hive/warehouse/<database_name>.db/*`` directory.
- 3.1.3. **DROP** - Deletes both metadata and data for MANAGED tables whereas Deletes **only Metadata** for UNMANAGED tables but the data still resides in the directory

3.2. Views, Temp Views & Global Temp Views

3.2.1. Sessions:

 Note: There are several scenarios in which a new session may be created:

- Restarting a cluster
- Detaching and reattaching to a cluster
- Installing a python package which in turn restarts the Python interpreter
- Or simply opening a new notebook

- 3.2.2. **Views** : Accessible within session
- 3.2.3. **Temp Views**: Accessible within session only for the specific notebooks. Cannot be accessed from another notebook
- 3.2.4. **Global Temp Views**: Accessible within session for the cluster under the global database. The global views are lost once the cluster is restarted. Global temp views behave much like other temporary views but differ in one important way. They are added to the `global_temp` database that exists on the `cluster`. As long as the cluster is running, this database persists and any notebooks attached to the cluster can access its global temporary views.

	database	tableName	isTemporary
1	dbacademy_jacob_parr_databricks_com_dewd_3_2	external_table	false
2	dbacademy_jacob_parr_databricks_com_dewd_3_2	view_delays_abq_lax	false
3		temp_view_delays_gt_120	true

	database	tableName	isTemporary
1	global_temp	global_temp_view_dist_gt_1000	true
2		temp_view_delays_gt_120	true

3.3. CTEs - Common Table Expressions are used in SQLs to catch temp results

3.3.1. Generally the scope is within Query. It is used for reusability of the complex queries and makes the code more readable.

3.3.2. Think of a CTE as being a View that only lasts for the duration of the query.

Note - This biggest difference is that a CTE can only be used in the current query scope whereas a temporary table or table variable can exist for the entire duration of the session allowing you to perform many different DML operations against them

4. ETL WITH SPARK SQL

4.1. Querying files directly

4.1.1. Files can be queried directly (CSV, JSON, PARQUET, TEXT, BINARY)

4.1.2. Syntax To Query single file - **SELECT * FROM file_format.`/path/to/file`**

4.1.3. Same format can be used to query the directory.

4.1.4. *Important call out - The assumption is the schema & format is same in case we are trying to read from the directory*

4.2. Providing Options for External Sources

4.2.1. While directly querying files works well for self-describing formats, many data sources require additional configurations or schema declaration to properly ingest records.

4.2.2. Registering tables on external locations with READ OPTIONS
CREATE TABLE table_identifier (col_name1 col_type1, ...)
USING data_source
OPTIONS (key1 = val1, key2 = val2, ...)
LOCATION = path

*Note - We cannot expect the performance guarantees associated with Delta Lake & Lakehouse when querying external tables. Use **Refresh table** to ensure the latest table is correctly cached for external location*

4.2.3. **Extracting Data from SQL Databases** - SQL databases are an extremely common data source, and Databricks has a standard JDBC driver for connecting with many flavors of SQL. The general syntax for creating these connections is:

CREATE TABLE USING JDBC
OPTIONS (
url

```
= "jdbc:{databaseServerType}://{jdbcHostname}:{jdbcPort}",  
dbtable = "{jdbcDatabase}.table", user = "{jdbcUsername}",  
password = "{jdbcPassword}")
```

Note - Some SQL systems such as data warehouses will have custom drivers. Spark will interact with various external databases differently, but the two basic approaches can be summarized as either: Moving the entire source table(s) to Databricks and then executing logic on the currently active cluster Pushing down the query to the external SQL database and only transferring the results back to Databricks

In either case, working with very large datasets in external SQL databases can incur significant overhead because of either: Network transfer latency associated with moving all data over the public internet or the execution of query logic in source systems not optimized for big data queries

4.3. Creating Delta tables

4.3.1. CTAS - CTAS automatically infer schema information from query results and do **not** support manual schema declaration. This means that CTAS statements are useful for external data ingestion from sources with well-defined schema, such as Parquet files and tables. CTAS statements also do not support specifying additional file options.

4.3.2. Declare Schema with Generated Columns - [Generated column](#) are a special type of column whose values are automatically generated based on a user-specified function over other columns in the Delta table (introduced in DBR 8.3).

4.3.3. Table constraints are shown in the `***TBLPROPERTIES***` field. Databricks currently support two types of constraints:
[NOT NULL constraints](#)
[CHECK constraints](#)

4.3.4. Enrich Tables with Additional Options and Metadata

4.3.4.1. Our **SELECT** clause leverages two built-in Spark SQL commands useful for file ingestion:

- **current_timestamp()** records the timestamp when the logic is executed
- **input_file_name()** records the source data file for each record in the table

4.3.4.2. **CREATE TABLE** clause contains several options:

- A **COMMENT** is added to allow for easier discovery of table contents
- A **LOCATION** is specified, which will result in an external (rather than managed) table
- The table is **PARTITIONED BY** a date column; this means that the data from each data will exist within its own directory in the target storage location

4.3.5. Cloning Delta lake Table

- 4.3.5.1. **DEEP CLONE** fully copies data and metadata from a source table to a target. This copy occurs **incrementally**, so executing this command again can sync changes from the source to the target location.
- 4.3.5.2. **SHALLOW CLONE**: If you wish to create a copy of a table quickly to test out applying changes without the risk of modifying the current table, `**`SHALLOW CLONE`**` can be a good option. Shallow clones just copy the Delta transaction logs, meaning that the data doesn't move.
- 4.3.5.3. **CTAS VS SHALLOW CLONE**: Clone is simpler to specify since it makes a faithful copy of the original table at the specified version and you don't need to re-specify partitioning, constraints and other information as you have to do with CTAS. In addition, it is much **faster**, **robust**, and can work in an **incremental** manner against failures! With deep clones, we copy additional metadata, such as your streaming application transactions and COPY INTO transactions, so you can continue your ETL applications exactly where it left off on a deep clone!

4.4. Writing to Tables

- 4.4.1. **Summary** - Overwrite data tables using **INSERT OVERWRITE**, Append to a table using **INSERT INTO**, Append, update, & delete from a table using **MERGE INTO** Ingest data incrementally into tables using **COPY INTO**
- 4.4.2. **Complete Overwrites**- We can use overwrites to atomically replace all of the data in a table. Spark SQL provides two easy methods to accomplish complete overwrites
 - 4.4.2.1. **Why Overwrite?** - There are multiple benefits to overwriting tables instead of deleting and recreating tables:
 - Overwriting a table is much faster because it doesn't need to list the directory recursively or delete any files.

- The old version of the table still exists; can easily retrieve the old data using Time Travel.
- It's an atomic operation. Concurrent queries can still read the table while you are deleting the table.
- Due to ACID transaction guarantees, if overwriting the table fails, the table will be in its previous state.

4.4.2.2. **CREATE OR REPLACE TABLE -**

- Replaces the complete content of the table

4.4.2.3. **INSERT OVERWRITE -**

- provides a nearly identical outcome as CREATE AND REPLACE TABLE data in the target table will be replaced by data from the query.
- Can only overwrite an existing table, not create a new one like our CRAS statement
- Can overwrite only with new records that match the current table schema -- and thus can be a "safer" technique for overwriting an existing table without disrupting downstream consumers
- Can overwrite individual partition

4.4.2.4. **CREATE OR REPLACE TABLE vs INSERT OVERWRITE**

- A primary difference between CRAS and Insert Overwrite here has to do with how Delta Lake enforces schema on write. Whereas a CRAS statement will allow us to completely redefine the contents of our target table, INSERT OVERWRITE will fail if we try to change our schema (unless we provide optional settings)

4.4.3. **Append Rows:**

4.4.3.1. **INSERT INTO** - We can use **INSERT INTO** to atomically append new rows to an existing Delta table. This allows for incremental updates to existing tables, which is much more efficient than overwriting each time. Note - This doesn't avoid any duplication. One can insert the same records again with the above command.

4.4.3.2. **MERGE INTO** - is used for UPSERTS. Delta Lake supports inserts, updates and deletes in **MERGE**, and supports extended syntax beyond the SQL standards to facilitate advanced use cases.

The main benefits of **MERGE**: updates, inserts, and deletes are completed as a single transaction, multiple conditionals can be added in addition to matching fields and

it provides extensive options for implementing custom logic

Syntax **MERGE INTO** target a **USING** source b
ON {merge_condition} WHEN MATCHED THEN {matched_action}
WHEN NOT MATCHED THEN {not_matched_action}

Use Merge (Insert-Only) for Deduplication - Common ETL use case is to collect logs or other every-appending datasets into a Delta table through a series of append operations. Many source systems can generate duplicate records. With **merge**, you can avoid inserting the duplicate records by performing an insert-only merge. This optimized command uses the same **MERGE** syntax but only provides a **WHEN NOT MATCHED** clause.

*Note - **INSERT INTO VS MERGE INTO** key difference is that Insert Into doesn't avoid duplicates while MERGE INTO will help avoid duplicates*

4.4.4. Load Incrementally

- **COPY INTO** provides SQL engineers an **idempotent option (EXECUTE ONLY ONCE)** to incrementally ingest data from external systems.

Note that this operation does have some expectations: -

- Data schema should be consistent-
- Duplicate records should try to be excluded or handled downstream
- This operation is potentially much cheaper than full table scans for data that grows predictably.

4.5. ****Intentionally left blank****

4.6. Cleaning Data

4.6.1. **Count(col) vs Count (*)** - Note that **count(col)** skips NULL`** values when counting specific columns or expressions. However, **count(*)** is a special case that counts the total number of rows (including rows that are only NULL values).

4.6.2. To count null values, use the ****count_if**** function or ****WHERE**** clause to provide a condition that filters for records where the value ****IS NULL****.

4.6.3. *Note that when we called **DISTINCT(*)**, by default we ignored all rows containing **any** null values; as such, our result is the same as the count of user emails above*

Note that `count(col)` skips `NULL` values when counting specific columns or expressions.

However, `count(*)` is a special case that counts the total number of rows (including rows that are only `NULL` values).

To count null values, use the `count_if` function or `WHERE` clause to provide a condition that filters for records where the value `IS NULL`.

4.6.4. Spark skips null values while counting values in a column or counting distinct values for a field, but does not omit rows with nulls from a **DISTINCT** query.

4.6.5. Other imp. functions: **WHERE, GROUP BY, ORDER BY**

4.7. Advance SQL Transformation - Using `.` and `:` syntax to query nested data, Working with JSON, Flattening and unpacking arrays and structs, Combining datasets using joins and set operators, Reshaping data using pivot tables, Using higher order functions for working with arrays

4.7.1. Interacting with JSON DATA

- **from_json function** is used to case the field to struct type
- **JSON.*** unpacks the json struct fields into a table columns i.e Once a JSON string is unpacked to a struct type, Spark supports `*` (star) unpacking to flatten fields into columns.
- **Colon:** is used to extract data from JSON String
- **Dot** `.` is used for struct type
- **From JSON** helps read json in a table however it needs the JSON Schema as input
- Spark SQL also has a **schema_of_json** function to derive the JSON schema from an example

4.7.1.1. Working with Arrays - Spark SQL has a number of functions specifically to deal with arrays.

- **Explode ()** - function lets us put each element in an array on its own row.
- **Collect Arrays** -

The **collect_set** function can collect unique values for a field, including fields within arrays.

The **flatten** function allows multiple arrays to be combined into a single array.

The **array_distinct** function removes duplicate elements from an array

4.7.2. Join Tables join operations (inner, outer, left, right, anti, cross, semi)

4.7.3. Set Operators

- **MINUS** returns all the rows found in one dataset but not the other; we'll skip executing this here as our previous query demonstrates we have no values in common.
- **UNION** returns the collection of two queries.
- **INTERSECT** returns all rows found in both relations.

4.7.4. Pivot Tables - The **PIVOT** clause is used for data perspective. We can get the aggregated values based on specific column values, which will be turned to multiple columns used in the **SELECT** clause. The **PIVOT** clause can be specified after the table name or subquery.

4.7.5. PIVOT: The first argument in the clause is an aggregate function and the column to be aggregated. Then, we specify the pivot column in the **FOR** subclause. The **IN** operator contains the pivot column values.

4.7.6. Higher Order Functions: Higher order functions in Spark SQL allow you to work directly with complex data types. When working with hierarchical data, records are frequently stored as array or map type objects. Higher-order functions allow you to transform data while preserving the original structure.

4.7.6.1. **FILTER** filters an array using the given lambda function.

4.7.6.2. **EXIST** tests whether a statement is true for one or more elements in an array.

4.7.6.3. **TRANSFORM** uses the given lambda function to transform all elements in an array.

4.7.6.4. **REDUCE** takes two lambda functions to reduce the elements of an array to a single value by merging the elements into a buffer, and then applying a finishing function on the final buffer.

4.8. SQL UDFs and Control Flow

4.8.1. SQL UDFs - At minimum, a SQL UDF requires a function name, optional parameters, the type to be returned, and some custom logic.

Syntax - *<<CREATE or REPLACE FUNCTION function_name(arg)
RETURNS XYZ
RETURN actual function>>*

4.8.2. Scoping and Permissions of SQL UDFs -

- 4.8.2.1. **Scope** - SQL UDFs will persist between execution environments (which can include notebooks, DBSQL queries, and jobs).
- 4.8.2.2. **Permissions** - SQL UDFs exist as objects in the metastore and are governed by the same Table ACLs as databases, tables, or views. In order to use a SQL UDF, a user must have **USAGE** and **SELECT** permissions on the function.
- 4.8.2.3. **CASE / WHEN** - The standard SQL syntactic construct **CASE / WHEN** allows the evaluation of multiple conditional statements with alternative outcomes based on table contents.
- 4.8.2.4. **Simple Control flow functions** - Combining SQL UDFs with control flow in the form of **CASE / WHEN** clauses provides optimized execution for control flows within SQL workloads

5. Python for Spark Sql (Optional)

- 5.1. **Print and manipulate multi-line Python strings** - By wrapping a string in triple quotes (""), it's possible to use multiple lines.
- 5.2. **Define variables and functions**
 - 5.2.1. **Variables** - Python variables are assigned using the **=**. Python variable names need to start with a letter, and can only contain letters, numbers, and underscores. (Variable names starting with underscores are valid but typically reserved for special use cases.). Many Python programmers favor snake casing, which uses only lowercase letters and underscores for all variables. The cell below creates the variable **my_string**.
 - 5.2.2. **Functions** - Functions allow you to specify local variables as arguments and then apply custom logic. We define a function using the keyword **def** followed by the function name and, enclosed in parentheses, any variable arguments we wish to pass into the function. Finally, the function header has a **:** at the end.
- 5.3. **F-strings** - Use f-strings for variable substitution in the string. By adding the letter **f** before a Python string, you can inject variables or evaluated Python code by inserting them inside curly braces (**{}**)
- 5.4. **Python Control Flow**
 - 5.4.1. **if/else** - clauses are common in many programming languages. SQL has the **CASE WHEN ... ELSE** construct, which is similar.
Note - *If you're seeking to evaluate conditions within your tables or queries, use CASE WHEN. Python control flow should be reserved for evaluating conditions **outside of your query**.*

5.4.2. elif - Python keyword **elif** (short for **else + if**) allows us to evaluate multiple conditions. Note that conditions are evaluated from top to bottom. Once a condition evaluates to true, no further conditions will be evaluated.

5.4.3. if / else control flow patterns:

- Must contain an **if** clause
- Can contain any number of **elif** clauses
- Can contain at most one **else** clause

6. Incremental Data Processing using Autoloader

Using Auto Loader

In the cell below, a function is defined to demonstrate using Databricks Auto Loader with the PySpark API. This code includes both a Structured Streaming read and write.

The following notebook will provide a more robust overview of Structured Streaming. If you wish to learn more about Auto Loader options, refer to the [documentation](#).

Note that when using Auto Loader with automatic [schema inference and evolution](#), the 4 arguments shown here should allow ingestion of most datasets. These arguments are explained below.

argument	what it is	how it's used
<code>data_source</code>	The directory of the source data	Auto Loader will detect new files as they arrive in this location and queue them for ingestion; passed to the <code>.load()</code> method
<code>source_format</code>	The format of the source data	While the format for all Auto Loader queries will be <code>cloudFiles</code> , the format of the source data should always be specified for the <code>cloudFiles.format</code> option
<code>table_name</code>	The name of the target table	Spark Structured Streaming supports writing directly to Delta Lake tables by passing a table name as a string to the <code>.table()</code> method. Note that you can either append to an existing table or create a new table
<code>checkpoint_directory</code>	The location for storing metadata about the stream	This argument is passed to the <code>checkpointLocation</code> and <code>cloudFiles.schemaLocation</code> options. Checkpoints keep track of streaming progress, while the schema location tracks updates to the fields in the source dataset

NOTE: The code below has been streamlined to demonstrate Auto Loader functionality. We'll see in later lessons that additional transformations can be applied to source data before saving them to Delta Lake.

When to use COPY INTO and when to use Auto Loader

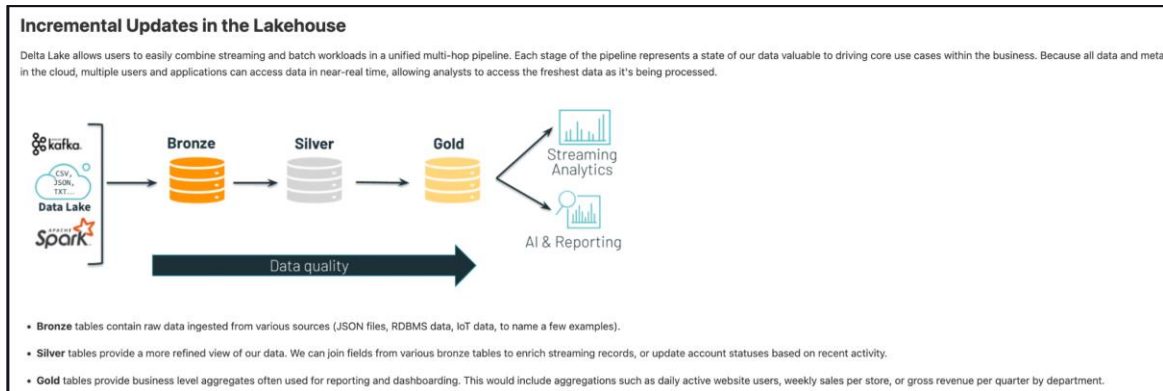
The **COPY INTO** command is another convenient way to load data incrementally into a Delta table with exactly-once guarantees. Here are a few things to consider when choosing between Auto Loader and COPY INTO:

- If you're going to ingest files in the order of thousands, you can use **COPY INTO**. If you are expecting files in the order of millions or more over time, use Auto Loader. Auto Loader can discover files more cheaply compared to COPY INTO and can split the processing into multiple batches.
- If your data schema is going to evolve frequently, Auto Loader provides better primitives around schema inference and evolution. See [Schema inference and evolution](#) for more details.
- Loading a subset of re-uploaded files can be a bit easier to manage with COPY INTO. With Auto Loader, it's harder to reprocess a select subset of files. However, you can use COPY INTO to reload the subset of files while an Auto Loader stream is running simultaneously.

7. Medallion Architecture - Bronze, Silver & Gold Architecture

7.1. Bronze - Raw data read usually in append mode

- 7.2. **Silver** - Cleansed, filtered and augmented read from bronze tables usually in append mode. **provide a more refined view of our data**
- 7.3. **Gold** - Involves **business level aggregates** and metrics usually run in **Complete mode**. Mainly used for dashboarding and reporting



8. Delta Live Table

- 8.1. **Declarative language** to define the ETL.
- 8.2. **LIVE** - At its simplest, you can think of DLT SQL as a slight modification to traditional CTAS statements. DLT tables and views will always be preceded by the **LIVE** keyword
- 8.3. **Two modes**- Continuous vs Triggered Mode
- 8.4. **Quality Control** -
 - 8.4.1. The **CONSTRAINT** keyword introduces quality control. Similar in function to a traditional **WHERE clause**, **CONSTRAINT** integrates with DLT, enabling it to collect metrics on constraint violations. Constraints provide an optional **ON VIOLATION** clause, specifying an action to take on records that violate the constraint. The three modes currently supported by DLT include:

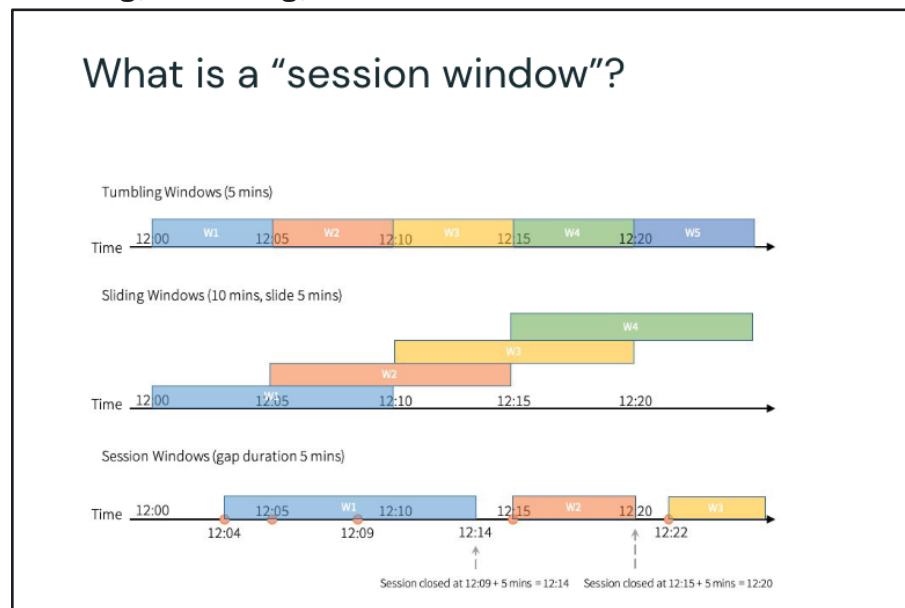
ON VIOLATION	Behavior
❖ FAIL UPDATE	❖ Pipeline failure when constraint is violated
❖ DROP ROW	❖ Discard records that violate constraints
❖ Omitted	❖ Records violating constraints will be included (but violations will be reported in metrics)

8.4.2. References to DLT Tables and Views - References to other DLT tables and views will always include the live. prefix. A target database name will automatically be substituted at runtime, allowing for easy migration of pipelines between DEV/QA/PROD environments.

8.4.3. References to Streaming Tables - References to streaming DLT tables use the **STREAM()**, supplying the table name as an argument.

8.5. Windows and Watermarking -

8.5.1. Used for aggregate functions in streaming. Three types of windows - **Sliding, Tumbling, Session Windows**

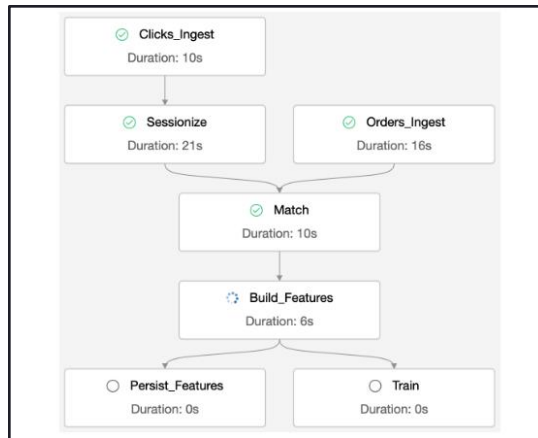


8.5.2. Watermarking - To bound the state size, we have to be able to drop old aggregates that are not going to be updated any more, for example seven day old averages. We achieve this using *watermarking*. Enables automatic dropping of old state data. It discards the data outside the watermark duration

8.5.3. Windows and watermarking is not available in detail in the notebooks. Watch Daniel's/Lorin's recording for the same. **(Starting ~ 33 mins at [this Link](#))**

9. Task Orchestration with jobs -

9.1. Run a sample job in notebooks. (It's a DAG in which dependencies across tasks can be established. You can mix DLT & regular Notebooks)



10. DB SQL & Dashboard:

- 10.1.1. It's very intuitive and straightforward. Provides ability to execute SQL Queries, that can be scheduled and used to create the dashboards
- 10.1.2. Alerts can be setup to notify end user
- 10.1.3. Learn about Alerts, Serverless and try the labs on DB SQL

11. Managing Permission

- 11.1. **Data Explorer** - Use data explorer to set up the permissions.
 - 11.1.1. **What is the Data Explorer?** - The data explorer allows users to:
 - Set and modify permissions of relational entities
 - Navigate databases, tables, and views
 - Explore data schema, metadata, and history
 - 11.1.2. **Syntax** - **GRANT/REVOKE <<PRIVILEGES>> ON <<OBJECT>> TO <<USERS/GROUP>>**

Table ACLs

Databricks allows you to configure permissions for the following objects:

Object	Scope
CATALOG	controls access to the entire data catalog.
DATABASE	controls access to a database.
TABLE	controls access to a managed or external table.
VIEW	controls access to SQL views.
FUNCTION	controls access to a named function.
ANY FILE	controls access to the underlying filesystem. Users granted access to ANY FILE can bypass the restrictions put on the catalog, databases, tables, and views by reading from the file system directly.

NOTE: At present, the **ANY FILE** object cannot be set from Data Explorer.

Granting Privileges

Databricks admins and object owners can grant privileges according to the following rules:

Role	Can grant access privileges for
Databricks administrator	All objects in the catalog and the underlying filesystem.
Catalog owner	All objects in the catalog.
Database owner	All objects in the database.
Table owner	Only the table (similar options for views and functions).

NOTE: At present, Data Explorer can only be used to modify ownership of databases, tables, and views. Catalog permissions can be set interactively with the SQL Query Editor.

Privileges

The following privileges can be configured in Data Explorer:

Privilege	Ability
ALL PRIVILEGES	gives all privileges (is translated into all the below privileges).
SELECT	gives read access to an object.
MODIFY	gives ability to add, delete, and modify data to or from an object.
READ_METADATA	gives ability to view an object and its metadata.
USAGE	does not give any abilities, but is an additional requirement to perform any action on a database object.
CREATE	gives ability to create an object (for example, a table in a database).

Databricks Interview Questions & Answers

<https://www.youtube.com/c/techlake>



databricks

- 1)What is Databricks Runtime?
- 2)What are the types of Databricks Runtimes?
- 3)How to share Notebook to other Developers in Workspace?
- 4)How to access one notebook variable into other notebooks?
- 5)How to call one notebook from another notebook?
- 6)How to exit a notebook with returning some output data?
- 7)How to create Internal & External tables in Databricks?
- 8)How to Access ADLS or Blob Storage in Databricks?
- 9)What are the types of Cluster Modes in Databricks?
- 10)What are the types of workloads we can use in Standard type Cluster?
- 11)Can I use both Python 2 and Python 3 notebooks on the same cluster?
- 12)What is pool? Why we use pool? How to create pool in Databricks?
- 13)How many ways we can create variables in Databricks?
- 14) What are the limitations in Jobs?
- 15) Can I use %pip in notebook for installing packages or libraries?

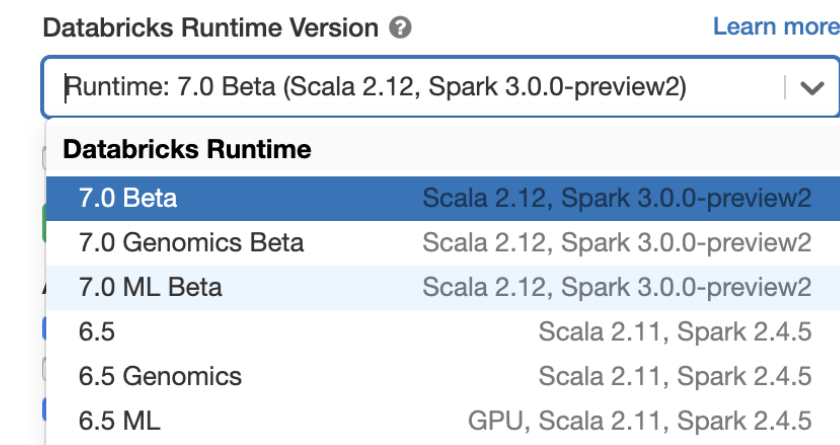
Databricks Interview Questions & Answers

<https://www.youtube.com/c/techlake>

1) What is Databricks Runtime?

The set of core components that run on the clusters managed by Databricks. Consists of the underlying Ubuntu OS, pre-installed languages and libraries (Java, Scala, Python, and R), Apache Spark, and various proprietary Databricks modules (e.g. DBIO, Databricks Serverless, etc.).

Azure Databricks offers several types of runtimes and several versions of those runtime types in the Databricks Runtime Version drop-down when you create or edit a cluster.



2) What are the types of Databricks Runtimes?

There are major 4 types of Databricks Runtimes.

- Databricks Runtime for Standard
- Databricks Runtime for Machine Learning
- Databricks Runtime for Genomics
- Databricks Light

Databricks Runtime for Standard

Databricks Runtime includes Apache Spark but also adds a number of components and updates that substantially improve the usability, performance, and security of big data analytics.

Databricks Runtime for Machine Learning

Databricks Interview Questions & Answers

<https://www.youtube.com/c/techlake>

Databricks Runtime ML is a variant of Databricks Runtime that adds multiple popular machine learning libraries, including TensorFlow, Keras, PyTorch, and XGBoost. ML also supports additional GPU supporting libraries clusters. **Graphics processing Units** Speeding up Machine Learning models. GPUs can drastically lower the cost because they support efficient parallel computation.

Databricks Runtime for Genomics

Databricks Runtime for Genomics is a variant of Databricks Runtime optimized for working with genomic and biomedical data.

Databricks Light

Databricks Light provides a runtime option for jobs that don't need the advanced performance, reliability, or autoscaling benefits provided by Databricks Runtime.



Databricks Light does not support:

- Delta Lake
- Autopilot features such as autoscaling
- Highly concurrent, all-purpose clusters
- Notebooks, dashboards, and collaboration features
- Connectors to various data sources and BI tools

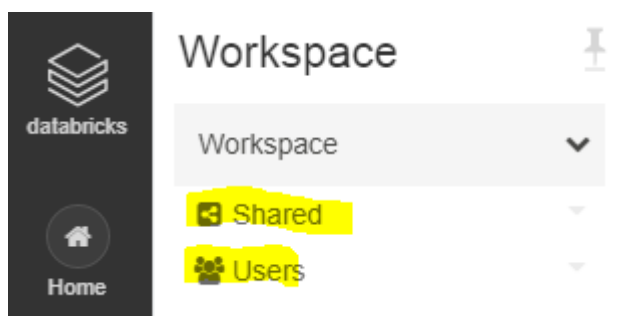
3) How to share Notebook to other Developers in Workspace?

There are two ways we can share notebooks to another developers.

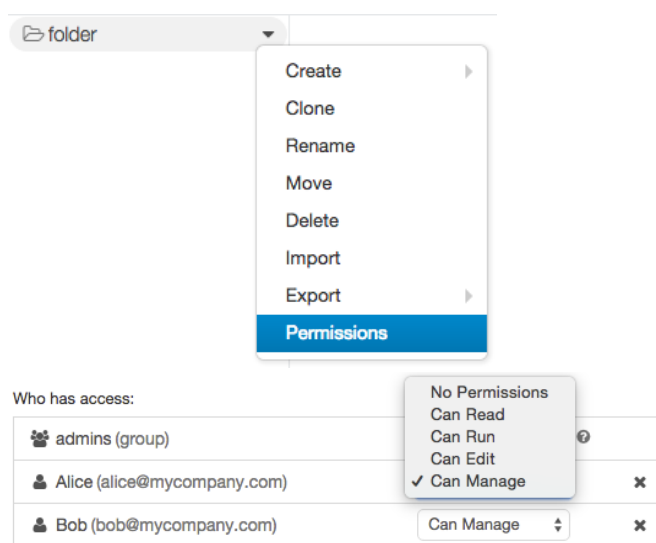
- a) Copying notebook into from user folder to shared folder.

Databricks Interview Questions & Answers

<https://www.youtube.com/c/techlake>



b) Or Giving Access to other developers from current folder or user folder.



4) How to access one notebook variable into other notebooks?

If we run a one notebook into another notebook using `%run` we can use all functions, variables and imported libraries from callee notebook to caller notebook.

```
Python
%run /Users/path/to/notebookA
```

5) How to call one notebook from another notebook?

There are two ways we can call one notebook into another notebook.

1. `%run notebook_name`

```
Python
%run /Users/path/to/notebookA
```

If we use the `%run` command. It will return from **callee notebooks** containing function and variable definitions. We can use those functions and variables in In **caller notebook**.

2. `Dbutils.notebook.run(notebook_name, timeout_sec, arguments_values)`

Databricks Interview Questions & Answers

<https://www.youtube.com/c/techlake>

You cannot use functions and variables. Only return value using arguments parameter.

```
returned_table = dbutils.notebook.run("notebook_exit_2", 60)
display(sqlContext.read.parquet(returned_table))
```

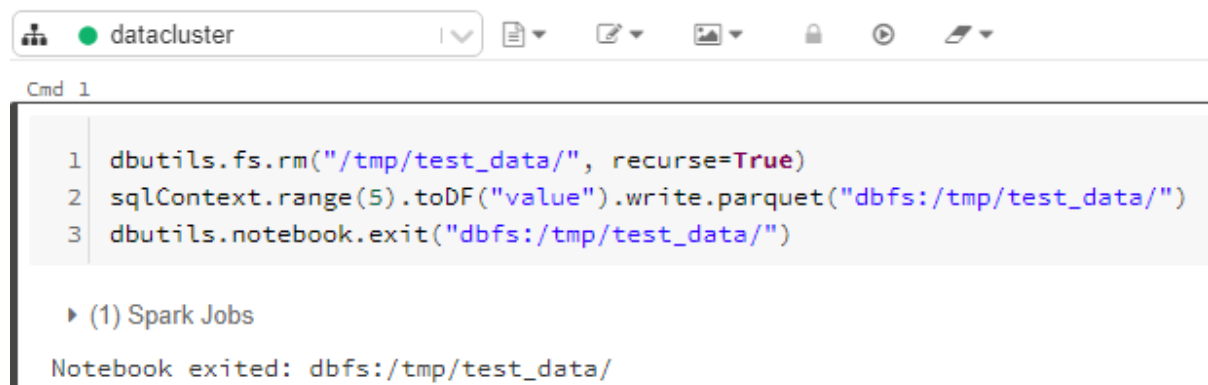
6) How to exit a notebook with returning some output data?

dbutils.notebook.exit (value: String): void -> This method lets you exit a notebook with a value.

In this example we have two notebooks. 1 for running exit and passing input value and another notebook for running 1st notebook using dbutils.notebook.run() method. And storing into variable.

1st notebook

notebook_exit_2 (Python)



2nd notebook.

```
returned_table = dbutils.notebook.run("notebook_exit_2", 60)
display(sqlContext.read.parquet(returned_table))
```

7) How to create Internal & External tables in Databricks?

A Databricks database is a collection of tables. A Databricks table is a collection of structured data. You can cache, filter, and perform any operations supported by Apache Spark DataFrames on Databricks tables. You can query tables with Spark APIs and Spark SQL.

External Table.

The table uses the custom directory specified with LOCATION. Queries on the table access existing data previously stored in the directory. When an EXTERNAL table is dropped, its data is not deleted from the file system. This flag is implied if LOCATION is specified.

Databricks Interview Questions & Answers

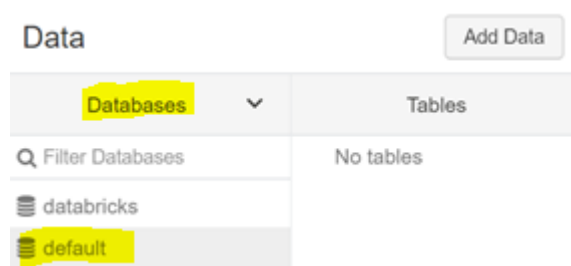
<https://www.youtube.com/c/techlake>

```
CREATE EXTERNAL TABLE IF NOT EXISTS my_table (name STRING, age INT)
COMMENT 'This table is created with existing data'
LOCATION 'spark-warehouse/tables/my_existing_table'
```

Internal or Managed Table

Create a managed table using the definition/metadata of an existing table or view. The created table always uses its own directory in the default warehouse location.

```
CREATE TABLE boxes
(width INT, length INT, height INT)
USING PARQUET
OPTIONS ('compression'='snappy')
```



8) How to Access ADLS or Blob Storage in Databricks?

We can Mount Azure Blob storage containers to DBFS and we can access through DBFS mount point.

You can mount a Blob storage container or a folder inside a container to Databricks File System (DBFS) using `dbutils.fs.mount`. The mount is a pointer to a Blob storage container, so the data is never synced locally.

```
dbutils.fs.mount(
  source = "wasbs://<container-name>@<storage-account-name>.blob.core.windows.net",
  mount_point = "/mnt/<mount-name>",
  extra_configs = {"<conf-key>":dbutils.secrets.get(scope = "<scope-name>", key = "<key-name>")})
```

Access files in your container as if they were local files, for example:

```
# python
df = spark.read.text("/mnt/<mount-name>/...")
df = spark.read.text("dbfs:/<mount-name>/...")
```

Once an account access key or a SAS is set up in your notebook, you can use standard Spark and Databricks APIs to read from the storage account

Set up an account access key:

Databricks Interview Questions & Answers

<https://www.youtube.com/c/techlake>

```
spark.conf.set(
  "fs.azure.account.key.<storage-account-name>.blob.core.windows.net",
  "<storage-account-access-key>")
```

Set up a SAS for a container:

```
spark.conf.set(
  "fs.azure.sas.<container-name>.<storage-account-name>.blob.core.windows.net",
  "<complete-query-string-of-sas-for-the-container>")
```

9) What are the types of Cluster Modes in Databricks?

- a) Standard clusters
- b) High concurrency clusters

10) What are the types of workloads we can use in Standard type Cluster?

There are three types of workloads we can use in Standard type cluster. Those are

- a. DATA ANALYTICS
- b. DATA ENGINEERING
- c. DATA ENGINEERING LIGHT

FEATURE	DATA ANALYTICS	DATA ENGINEERING	DATA ENGINEERING LIGHT
Apache Spark on Databricks platform			
Job scheduling with libraries			
Job scheduling with Notebooks			
Autopilot clusters			
Databricks Runtime for ML			
MLflow on Databricks Preview			
Databricks Delta			
Interactive clusters			
Notebooks and collaboration			
Ecosystem integrations			

11) Can I use both Python 2 and Python 3 notebooks on the same cluster?

No. In Single Cluster you can use only one Python2 or Python 3. You cannot use both Python2 and python3 on same databricks cluster.

12) What is pool? Why we use pool? How to create pool in Databricks?

Pool is used to reduce cluster start time while auto scaling, you can attach a cluster to a predefined **pool of idle instances**. When attached to a pool, a cluster allocates its driver and worker nodes from the pool. If the pool does not have sufficient idle resources to accommodate the cluster's request, the pool expands by allocating new instances from the

Databricks Interview Questions & Answers

<https://www.youtube.com/c/techlake>

instance provider. When an attached cluster is terminated, the instances it used are returned to the pool and can be reused by a different cluster.

Clusters



Clusters



Clusters Pools

+ Create Pool

Refresh

Name	Instance Type	Min Idle	Max Capacity	Idle Instances	Used Instances	Actions
Demo Pool	30.5 GB Memory, 4 Cores	2	10	2	2	

1 - 1 of 1 < > 20 / Page Go to 1

Clusters / Pools / Pool Details



Demo Pool

Edit

Delete

Refresh

Overview Configuration

Instance Type: , 30.5 GB Memory, 4 Cores

Min Idle: 2

Total instances: 2

Idle Instance Auto Termination: 60 minutes

Max Capacity: 10

Used: 0 Idle: 0 (Pending: 2) Max Capacity: 10

13) How many ways we can create variables in Databricks?

There are different ways we can create variables in Databricks.

One method is creating variable and assigning values and calling that notebook into another notebook using **%run**

```
Python
%run /Users/path/to/notebookA
```

Another method is using `dbutils.widgets` method.

Use `dbutils.widgets.text()` or `dbutils.widgets.dropdown()` to create a widget parameter or variable

and `dbutils.widgets.get()` to get its bound value.

Databricks Interview Questions & Answers

<https://www.youtube.com/c/techlake>

Var_v:

Cmd 1

```
1 dbutils.widgets.text("Var_v", "text_Value")
2 print(dbutils.widgets.get("Var_v"))
```

text_Value

Command took 0.03 seconds -- by pysparktelugu@gmail.com at 9/6/2020, 2:5

Cmd 2

V_X:

Cmd 1

```
1 dbutils.widgets.dropdown("v_x", "1", [str(x) for x in range(1, 10)])
2 print(dbutils.widgets.get("v_x"))
```

1

If we type `.help()` it will show available methods in `dbutils.widgets`

Like **creating** text, **dropdown**, **combobox** variables and getting values using GET method.

Removing all variables using **`dbutils.widgets.removeAll()`**

Cmd 1

```
1 dbutils.widgets.help()
```

dbutils.widgets provides utilities for working with notebook widgets. You can create diff **dbutils.widgets.help("methodName")**.

combobox(name: String, defaultValue: String, choices: Seq, label: String): void ->

dropdown(name: String, defaultValue: String, choices: Seq, label: String): void ->

get(name: String): String -> Retrieves current value of an input widget

getArgument(name: String, optional: String): String -> (DEPRECATED) Equivalent to

multiselect(name: String, defaultValue: String, choices: Seq, label: String): void ->

remove(name: String): void -> Removes an input widget from the notebook

removeAll: void -> Removes all widgets in the notebook

text(name: String, defaultValue: String, label: String): void -> Creates a text input w

Databricks Interview Questions & Answers

<https://www.youtube.com/c/techlake>

14) What are the limitations in Jobs?

- A. The number of jobs is limited to 1000.
- B. A workspace is limited to 150 concurrent (running) job runs.
- C. A workspace is limited to 1000 active (running and pending) job runs.

15) Can I use %pip in notebook for installing packages or libraries?

Yes. We can use.

Creating a file for list of commands to be executed

```
dbutils.fs.put("/dbfs:/home/myScripts/fast.ai", "conda install -c pytorch -c fastai fastai -y", True)
```

Installing using created file

```
%pip install -r /dbfs/home/myScripts/fast.ai
```

Installing using %pip

```
%pip install matplotlib
```

Uninstalling using %pip

```
%pip uninstall -y matplotlib
```

We can use conda also same as like %pip

```
%conda install matplotlib
```

Import the file to another notebook using Conda env update.

```
%conda env update -f /dbfs/myenv.yml
```

List the Python environment of a notebook

```
%conda list
```

Column Method

<code>Column.__getattr__(item)</code>	An expression that gets an item at position ordinal out of a list, or gets an item by key out of a dict.
<code>Column.__getitem__(k)</code>	An expression that gets an item at position ordinal out of a list, or gets an item by key out of a dict.
<code>Column.alias(*alias, **kwargs)</code>	Returns this column aliased with a new name or names (in the case of expressions that return more than one column, such as explode).
<code>Column.asc()</code>	Returns a sort expression based on the ascending order of the column.
<code>Column.asc_nulls_first()</code>	Returns a sort expression based on the ascending order of the column, and null values return before non-null values.
<code>Column.asc_nulls_last()</code>	Returns a sort expression based on the ascending order of the column, and null values appear after non-null values.
<code>Column.astype(dataType)</code>	<code>astype()</code> is an alias for <code>cast()</code> .
<code>Column.between(lowerBound, upperBound)</code>	True if the current column is between the lower bound and upper bound, inclusive.
<code>Column.bitwiseAND(other)</code>	Compute bitwise AND of this expression with another expression.
<code>Column.bitwiseOR(other)</code>	Compute bitwise OR of this expression with another expression.
<code>Column.bitwiseXOR(other)</code>	Compute bitwise XOR of this expression with another expression.
<code>Column.cast(dataType)</code>	Casts the column into type <code>dataType</code> .

Column.contains (other)	Contains the other element.
Column.desc ()	Returns a sort expression based on the descending order of the column.
Column.desc_nulls_first ()	Returns a sort expression based on the descending order of the column, and null values appear before non-null values.
Column.desc_nulls_last ()	Returns a sort expression based on the descending order of the column, and null values appear after non-null values.
Column.dropFields (*fieldNames)	An expression that drops fields in StructType by name.
Column.endswith (other)	String ends with.
Column.eqNullSafe (other)	Equality test that is safe for null values.
Column.getField (name)	An expression that gets a field by name in a StructType .
Column.getItem (key)	An expression that gets an item at position ordinal out of a list, or gets an item by key out of a dict.
Column.ilike (other)	SQL ILIKE expression (case insensitive LIKE).
Column.isNotNull ()	True if the current expression is NOT null.
Column.isNull ()	True if the current expression is null.
Column.isin (*cols)	A boolean expression that is evaluated to true if the value of this expression is contained by the evaluated values of the arguments.
Column.like (other)	SQL like expression.

<code>Column.name(*alias, **kwargs)</code>	<code>name()</code> is an alias for <code>alias()</code> .
<code>Column.otherwise(value)</code>	Evaluates a list of conditions and returns one of multiple possible result expressions.
<code>Column.over(window)</code>	Define a windowing column.
<code>Column.rlike(other)</code>	SQL RLIKE expression (LIKE with Regex).
<code>Column.startswith(other)</code>	String starts with.
<code>Column.substr(startPos, length)</code>	Return a <code>Column</code> which is a substring of the column.
<code>Column.when(condition, value)</code>	Evaluates a list of conditions and returns one of multiple possible result expressions.
<code>Column.withField(fieldName, col)</code>	An expression that adds/replaces a field in <code>StructType</code> by name.

Functions

Normal Functions

<code>col(col)</code>	Returns a <code>Column</code> based on the given column name.
<code>column(col)</code>	Returns a <code>Column</code> based on the given column name.
<code>lit(col)</code>	Creates a <code>Column</code> of literal value.
<code>broadcast(df)</code>	Marks a DataFrame as small enough for use in broadcast joins.

<code>coalesce(*cols)</code>	Returns the first column that is not null.
<code>input_file_name()</code>	Creates a string column for the file name of the current Spark task.
<code>isnan(col)</code>	An expression that returns true if the column is NaN.
<code>isnull(col)</code>	An expression that returns true if the column is null.
<code>monotonically_increasing_id()</code>	A column that generates monotonically increasing 64-bit integers.
<code>nanvl(col1, col2)</code>	Returns col1 if it is not NaN, or col2 if col1 is NaN.
<code>rand([seed])</code>	Generates a random column with independent and identically distributed (i.i.d.) samples uniformly distributed in [0.0, 1.0).
<code>randn([seed])</code>	Generates a column with independent and identically distributed (i.i.d.) samples from the standard normal distribution.
<code>spark_partition_id()</code>	A column for partition ID.
<code>when(condition, value)</code>	Evaluates a list of conditions and returns one of multiple possible result expressions.
<code>bitwise_not(col)</code>	Computes bitwise not.
<code>bitwiseNOT(col)</code>	Computes bitwise not.
<code>expr(str)</code>	Parses the expression string into the column that it represents
<code>greatest(*cols)</code>	Returns the greatest value of the list of column names, skipping null values.

<code>least(*cols)</code>	Returns the least value of the list of column names, skipping null values.
---------------------------	--

Math Functions

<code>sqrt(col)</code>	Computes the square root of the specified float value.
------------------------	--

<code>abs(col)</code>	Computes the absolute value.
-----------------------	------------------------------

<code>acos(col)</code>	Computes inverse cosine of the input column.
------------------------	--

<code>acosh(col)</code>	Computes inverse hyperbolic cosine of the input column.
-------------------------	---

<code>asin(col)</code>	Computes inverse sine of the input column.
------------------------	--

<code>asinh(col)</code>	Computes inverse hyperbolic sine of the input column.
-------------------------	---

<code>atan(col)</code>	Compute inverse tangent of the input column.
------------------------	--

<code>atanh(col)</code>	Computes inverse hyperbolic tangent of the input column.
-------------------------	--

<code>atan2(col1, col2)</code>	<i>New in version 1.4.0.</i>
--------------------------------	------------------------------

<code>bin(col)</code>	Returns the string representation of the binary value of the given column.
-----------------------	--

<code>cbrt(col)</code>	Computes the cube-root of the given value.
------------------------	--

<code>ceil(col)</code>	Computes the ceiling of the given value.
------------------------	--

<code>conv(col, fromBase, toBase)</code>	Convert a number in a string column from one base to another.
<code>cos(col)</code>	Computes cosine of the input column.
<code>cosh(col)</code>	Computes hyperbolic cosine of the input column.
<code>cot(col)</code>	Computes cotangent of the input column.
<code>csc(col)</code>	Computes cosecant of the input column.
<code>exp(col)</code>	Computes the exponential of the given value.
<code>expm1(col)</code>	Computes the exponential of the given value minus one.
<code>factorial(col)</code>	Computes the factorial of the given value.
<code>floor(col)</code>	Computes the floor of the given value.
<code>hex(col)</code>	Computes hex value of the given column, which could be <code>pyspark.sql.types.StringType</code> , <code>pyspark.sql.types.BinaryType</code> , <code>pyspark.sql.types.IntegerType</code> Or <code>pyspark.sql.types.LongType</code> .
<code>unhex(col)</code>	Inverse of hex.
<code>hypot(col1, col2)</code>	Computes $\sqrt{a^2 + b^2}$ without intermediate overflow or underflow.
<code>log(arg1[, arg2])</code>	Returns the first argument-based logarithm of the second argument.
<code>log10(col)</code>	Computes the logarithm of the given value in Base 10.

<code>log1p(col)</code>	Computes the natural logarithm of the “given value plus one”.
<code>log2(col)</code>	Returns the base-2 logarithm of the argument.
<code>pmod(dividend, divisor)</code>	Returns the positive value of dividend mod divisor.
<code>pow(col1, col2)</code>	Returns the value of the first argument raised to the power of the second argument.
<code>rint(col)</code>	Returns the double value that is closest in value to the argument and is equal to a mathematical integer.
<code>round(col[, scale])</code>	Round the given value to scale decimal places using HALF_UP rounding mode if scale ≥ 0 or at integral part when scale < 0 .
<code>round(col[, scale])</code>	Round the given value to scale decimal places using HALF_EVEN rounding mode if scale ≥ 0 or at integral part when scale < 0 .
<code>sec(col)</code>	Computes secant of the input column.
<code>shiftright(col, numBits)</code>	Shift the given value numBits left.
<code>shiftright(col, numBits)</code>	(Signed) shift the given value numBits right.
<code>shiftrightunsigned(col, numBits)</code>	Unsigned shift the given value numBits right.
<code>signum(col)</code>	Computes the signum of the given value.
<code>sin(col)</code>	Computes sine of the input column.
<code>sinh(col)</code>	Computes hyperbolic sine of the input column.
<code>tan(col)</code>	Computes tangent of the input column.

<code>tanh(col)</code>	Computes hyperbolic tangent of the input column.
------------------------	--

<code>toDegrees(col)</code>	<i>New in version 1.4.0.</i>
-----------------------------	------------------------------

<code>degrees(col)</code>	Converts an angle measured in radians to an approximately equivalent angle measured in degrees.
---------------------------	---

<code>toRadians(col)</code>	<i>New in version 1.4.0.</i>
-----------------------------	------------------------------

<code>radians(col)</code>	Converts an angle measured in degrees to an approximately equivalent angle measured in radians.
---------------------------	---

Datetime Functions

<code>add_months(start, months)</code>	Returns the date that is months months after start.
--	---

<code>current_date()</code>	Returns the current date at the start of query evaluation as a DateType column.
-----------------------------	--

<code>current_timestamp()</code>	Returns the current timestamp at the start of query evaluation as a TimestampType column.
----------------------------------	--

<code>date_add(start, days)</code>	Returns the date that is days days after start.
------------------------------------	---

<code>date_format(date, format)</code>	Converts a date/timestamp/string to a value of string in the format specified by the date format given by the second argument.
--	--

<code>date_sub(start, days)</code>	Returns the date that is days days before start.
------------------------------------	--

<code>date_trunc(format, timestamp)</code>	Returns timestamp truncated to the unit specified by the format.
<code>datediff(end, start)</code>	Returns the number of days from start to end.
<code>dayofmonth(col)</code>	Extract the day of the month of a given date/timestamp as integer.
<code>dayofweek(col)</code>	Extract the day of the week of a given date/timestamp as integer.
<code>dayofyear(col)</code>	Extract the day of the year of a given date/timestamp as integer.
<code>second(col)</code>	Extract the seconds of a given date as integer.
<code>weekofyear(col)</code>	Extract the week number of a given date as integer.
<code>year(col)</code>	Extract the year of a given date/timestamp as integer.
<code>quarter(col)</code>	Extract the quarter of a given date/timestamp as integer.
<code>month(col)</code>	Extract the month of a given date/timestamp as integer.
<code>last_day(date)</code>	Returns the last day of the month which the given date belongs to.
<code>localtimestamp()</code>	Returns the current timestamp without time zone at the start of query evaluation as a timestamp without time zone column.
<code>minute(col)</code>	Extract the minutes of a given timestamp as integer.

<code>months_between</code> (date1, date2[, roundOff])	Returns number of months between dates date1 and date2.
<code>next_day</code> (date, dayOfWeek)	Returns the first date which is later than the value of the date column based on second week day argument.
<code>hour</code> (col)	Extract the hours of a given timestamp as integer.
<code>make_date</code> (year, month, day)	Returns a column with a date built from the year, month and day columns.
<code>from_unixtime</code> (timestamp[, format])	Converts the number of seconds from unix epoch (1970-01-01 00:00:00 UTC) to a string representing the timestamp of that moment in the current system time zone in the given format.
<code>unix_timestamp</code> ([timestamp, format])	Convert time string with given pattern ('yyyy-MM-dd HH:mm:ss', by default) to Unix time stamp (in seconds), using the default timezone and the default locale, returns null if failed.
<code>to_timestamp</code> (col[, format])	Converts a <code>Column</code> into <code>pyspark.sql.types.TimestampType</code> using the optionally specified format.
<code>to_date</code> (col[, format])	Converts a <code>Column</code> into <code>pyspark.sql.types.DateType</code> using the optionally specified format.
<code>trunc</code> (date, format)	Returns date truncated to the unit specified by the format.
<code>from_utc_timestamp</code> (timestamp, tz)	This is a common function for databases supporting TIMESTAMP WITHOUT TIMEZONE.
<code>to_utc_timestamp</code> (timestamp, tz)	This is a common function for databases supporting TIMESTAMP WITHOUT TIMEZONE.

<code>window</code> (timeColumn, windowDuration[, ...])	Bucketize rows into one or more time windows given a timestamp specifying column.
<code>session_window</code> (timeColumn, gapDuration)	Generates session window given a timestamp specifying column.
<code>timestamp_seconds</code> (col)	Converts the number of seconds from the Unix epoch (1970-01-01T00:00:00Z) to a timestamp.
<code>window_time</code> (windowColumn)	Computes the event time from a window column.

Collection Functions

<code>array</code> (*cols)	Creates a new array column.
<code>array_contains</code> (col, value)	Collection function: returns null if the array is null, true if the array contains the given value, and false otherwise.
<code>arrays_overlap</code> (a1, a2)	Collection function: returns true if the arrays contain any common non-null element; if not, returns null if both the arrays are non-empty and any of them contains a null element; returns false otherwise.
<code>array_join</code> (col, delimiter[, null_replacement])	Concatenates the elements of column using the delimiter.
<code>create_map</code> (*cols)	Creates a new map column.
<code>slice</code> (x, start, length)	Collection function: returns an array containing all the elements in x from index start (array indices start at 1, or from the end if start is negative) with the specified length.

<code>concat(*cols)</code>	Concatenates multiple input columns together into a single column.
<code>array_position(col, value)</code>	Collection function: Locates the position of the first occurrence of the given value in the given array.
<code>element_at(col, extraction)</code>	Collection function: Returns element of array at given index in extraction if col is array.
<code>array_append(col, value)</code>	Collection function: returns an array of the elements in col1 along with the added element in col2 at the last of the array.
<code>array_sort(col[, comparator])</code>	Collection function: sorts the input array in ascending order.
<code>array_insert(arr, pos, value)</code>	Collection function: adds an item into a given array at a specified array index.
<code>array_remove(col, element)</code>	Collection function: Remove all elements that equal to element from the given array.
<code>array_distinct(col)</code>	Collection function: removes duplicate values from the array.
<code>array_intersect(col1, col2)</code>	Collection function: returns an array of the elements in the intersection of col1 and col2, without duplicates.
<code>array_union(col1, col2)</code>	Collection function: returns an array of the elements in the union of col1 and col2, without duplicates.
<code>array_except(col1, col2)</code>	Collection function: returns an array of the elements in col1 but not in col2, without duplicates.
<code>array_compact(col)</code>	Collection function: removes null values from the array.

<code>transform(col, f)</code>	Returns an array of elements after applying a transformation to each element in the input array.
<code>exists(col, f)</code>	Returns whether a predicate holds for one or more elements in the array.
<code>forall(col, f)</code>	Returns whether a predicate holds for every element in the array.
<code>filter(col, f)</code>	Returns an array of elements for which a predicate holds in a given array.
<code>aggregate(col, initialValue, merge[, finish])</code>	Applies a binary operator to an initial state and all elements in the array, and reduces this to a single state.
<code>zip_with(left, right, f)</code>	Merge two given arrays, element-wise, into a single array using a function.
<code>transform_keys(col, f)</code>	Applies a function to every key-value pair in a map and returns a map with the results of those applications as the new keys for the pairs.
<code>transform_values(col, f)</code>	Applies a function to every key-value pair in a map and returns a map with the results of those applications as the new values for the pairs.
<code>map_filter(col, f)</code>	Returns a map whose key-value pairs satisfy a predicate.
<code>map_from_arrays(col1, col2)</code>	Creates a new map from two arrays.
<code>map_zip_with(col1, col2, f)</code>	Merge two given maps, key-wise into a single map using a function.

<code>explode(col)</code>	Returns a new row for each element in the given array or map.
<code>explode_outer(col)</code>	Returns a new row for each element in the given array or map.
<code>posexplode(col)</code>	Returns a new row for each element with position in the given array or map.
<code>posexplode_outer(col)</code>	Returns a new row for each element with position in the given array or map.
<code>inline(col)</code>	Explodes an array of structs into a table.
<code>inline_outer(col)</code>	Explodes an array of structs into a table.
<code>get(col, index)</code>	Collection function: Returns element of array at given (0-based) index.
<code>get_json_object(col, path)</code>	Extracts json object from a json string based on json path specified, and returns json string of the extracted json object.
<code>json_tuple(col, *fields)</code>	Creates a new row for a json column according to the given field names.
<code>from_json(col, schema[, options])</code>	Parses a column containing a JSON string into a MapType with StringType as keys type, StructType or ArrayType with the specified schema.
<code>schema_of_json(json[, options])</code>	Parses a JSON string and infers its schema in DDL format.
<code>to_json(col[, options])</code>	Converts a column containing a StructType , ArrayType or a MapType into a JSON string.

<code>size(col)</code>	Collection function: returns the length of the array or map stored in the column.
<code>struct(*cols)</code>	Creates a new struct column.
<code>sort_array(col[, asc])</code>	Collection function: sorts the input array in ascending or descending order according to the natural ordering of the array elements.
<code>array_max(col)</code>	Collection function: returns the maximum value of the array.
<code>array_min(col)</code>	Collection function: returns the minimum value of the array.
<code>shuffle(col)</code>	Collection function: Generates a random permutation of the given array.
<code>reverse(col)</code>	Collection function: returns a reversed string or an array with reverse order of elements.
<code>flatten(col)</code>	Collection function: creates a single array from an array of arrays.
<code>sequence(start, stop[, step])</code>	Generate a sequence of integers from start to stop, incrementing by step.
<code>array_repeat(col, count)</code>	Collection function: creates an array containing a column repeated count times.
<code>map_contains_key(col, value)</code>	Returns true if the map contains the key.
<code>map_keys(col)</code>	Collection function: Returns an unordered array containing the keys of the map.

<code>map_values(col)</code>	Collection function: Returns an unordered array containing the values of the map.
<code>map_entries(col)</code>	Collection function: Returns an unordered array of all entries in the given map.
<code>map_from_entries(col)</code>	Collection function: Converts an array of entries (key value struct types) to a map of values.
<code>arrays_zip(*cols)</code>	Collection function: Returns a merged array of structs in which the N-th struct contains all N-th values of input arrays.
<code>map_concat(*cols)</code>	Returns the union of all the given maps.
<code>from_csv(col, schema[, options])</code>	Parses a column containing a CSV string to a row with the specified schema.
<code>schema_of_csv(csv[, options])</code>	Parses a CSV string and infers its schema in DDL format.
<code>to_csv(col[, options])</code>	Converts a column containing a StructType into a CSV string.

Partition Transformation Functions

<code>years(col)</code>	Partition transform function: A transform for timestamps and dates to partition data into years.
<code>months(col)</code>	Partition transform function: A transform for timestamps and dates to partition data into months.

<code>days(col)</code>	Partition transform function: A transform for timestamps and dates to partition data into days.
<code>hours(col)</code>	Partition transform function: A transform for timestamps to partition data into hours.
<code>bucket(numBuckets, col)</code>	Partition transform function: A transform for any type that partitions by a hash of the input column.

Aggregate Functions

<code>approxCountDistinct(col[, rsd])</code>	<i>New in version 1.3.0.</i>
<code>approx_count_distinct(col[, rsd])</code>	Aggregate function: returns a new <code>Column</code> for approximate distinct count of column <code>col</code> .
<code>avg(col)</code>	Aggregate function: returns the average of the values in a group.
<code>collect_list(col)</code>	Aggregate function: returns a list of objects with duplicates.
<code>collect_set(col)</code>	Aggregate function: returns a set of objects with duplicate elements eliminated.
<code>corr(col1, col2)</code>	Returns a new <code>Column</code> for the Pearson Correlation Coefficient for <code>col1</code> and <code>col2</code> .
<code>count(col)</code>	Aggregate function: returns the number of items in a group.
<code>count_distinct(col, *cols)</code>	Returns a new <code>Column</code> for distinct count of <code>col</code> or <code>cols</code> .

<code>countDistinct(col, *cols)</code>	Returns a new <code>Column</code> for distinct count of <code>col</code> or <code>cols</code> .
<code>covar_pop(col1, col2)</code>	Returns a new <code>Column</code> for the population covariance of <code>col1</code> and <code>col2</code> .
<code>covar_samp(col1, col2)</code>	Returns a new <code>Column</code> for the sample covariance of <code>col1</code> and <code>col2</code> .
<code>first(col[, ignorenulls])</code>	Aggregate function: returns the first value in a group.
<code>grouping(col)</code>	Aggregate function: indicates whether a specified column in a GROUP BY list is aggregated or not, returns 1 for aggregated or 0 for not aggregated in the result set.
<code>grouping_id(*cols)</code>	Aggregate function: returns the level of grouping, equals to
<code>kurtosis(col)</code>	Aggregate function: returns the kurtosis of the values in a group.
<code>last(col[, ignorenulls])</code>	Aggregate function: returns the last value in a group.
<code>max(col)</code>	Aggregate function: returns the maximum value of the expression in a group.
<code>max_by(col, ord)</code>	Returns the value associated with the maximum value of <code>ord</code> .
<code>mean(col)</code>	Aggregate function: returns the average of the values in a group.
<code>median(col)</code>	Returns the median of the values in a group.

<code>min(col)</code>	Aggregate function: returns the minimum value of the expression in a group.
<code>min_by(col, ord)</code>	Returns the value associated with the minimum value of ord.
<code>mode(col)</code>	Returns the most frequent value in a group.
<code>percentile_approx(col, percentage[, accuracy])</code>	Returns the approximate percentile of the numeric column col which is the smallest value in the ordered col values (sorted from least to greatest) such that no more than percentage of col values is less than the value or equal to that value.
<code>product(col)</code>	Aggregate function: returns the product of the values in a group.
<code>skewness(col)</code>	Aggregate function: returns the skewness of the values in a group.
<code>stddev(col)</code>	Aggregate function: alias for stddev_samp.
<code>stddev_pop(col)</code>	Aggregate function: returns population standard deviation of the expression in a group.
<code>stddev_samp(col)</code>	Aggregate function: returns the unbiased sample standard deviation of the expression in a group.
<code>sum(col)</code>	Aggregate function: returns the sum of all values in the expression.
<code>sum_distinct(col)</code>	Aggregate function: returns the sum of distinct values in the expression.

<code>sumDistinct(col)</code>	Aggregate function: returns the sum of distinct values in the expression.
<code>var_pop(col)</code>	Aggregate function: returns the population variance of the values in a group.
<code>var_samp(col)</code>	Aggregate function: returns the unbiased sample variance of the values in a group.
<code>variance(col)</code>	Aggregate function: alias for <code>var_samp</code>

Window Functions

<code>cume_dist()</code>	Window function: returns the cumulative distribution of values within a window partition, i.e.
<code>dense_rank()</code>	Window function: returns the rank of rows within a window partition, without any gaps.
<code>lag(col[, offset, default])</code>	Window function: returns the value that is offset rows before the current row, and default if there is less than offset rows before the current row.
<code>lead(col[, offset, default])</code>	Window function: returns the value that is offset rows after the current row, and default if there is less than offset rows after the current row.
<code>nth_value(col, offset[, ignoreNulls])</code>	Window function: returns the value that is the offsetth row of the window frame (counting from 1), and null if the size of window frame is less than offset rows.
<code>ntile(n)</code>	Window function: returns the ntile group id (from 1 to n inclusive) in an ordered window partition.

`percent_rank()`

Window function: returns the relative rank (i.e.

`rank()`

Window function: returns the rank of rows within a window partition.

`row_number()`

Window function: returns a sequential number starting at 1 within a window partition.

Sort Functions

`asc(col)`

Returns a sort expression based on the ascending order of the given column name.

`asc_nulls_first(col)`

Returns a sort expression based on the ascending order of the given column name, and null values return before non-null values.

`asc_nulls_last(col)`

Returns a sort expression based on the ascending order of the given column name, and null values appear after non-null values.

`desc(col)`

Returns a sort expression based on the descending order of the given column name.

`desc_nulls_first(col)`

Returns a sort expression based on the descending order of the given column name, and null values appear before non-null values.

`desc_nulls_last(col)`

Returns a sort expression based on the descending order of the given column name, and null values appear after non-null values.

String Functions

`ascii(col)`

Computes the numeric value of the first character of the string column.

<code>base64(col)</code>	Computes the BASE64 encoding of a binary column and returns it as a string column.
<code>bit_length(col)</code>	Calculates the bit length for the specified string column.
<code>concat_ws(sep, *cols)</code>	Concatenates multiple input string columns together into a single string column, using the given separator.
<code>decode(col, charset)</code>	Computes the first argument into a string from a binary using the provided character set (one of 'US-ASCII', 'ISO-8859-1', 'UTF-8', 'UTF-16BE', 'UTF-16LE', 'UTF-16').
<code>encode(col, charset)</code>	Computes the first argument into a binary from a string using the provided character set (one of 'US-ASCII', 'ISO-8859-1', 'UTF-8', 'UTF-16BE', 'UTF-16LE', 'UTF-16').
<code>format_number(col, d)</code>	Formats the number X to a format like '#,-#,-#.-', rounded to d decimal places with HALF_EVEN round mode, and returns the result as a string.
<code>format_string(format, *cols)</code>	Formats the arguments in printf-style and returns the result as a string column.
<code>initcap(col)</code>	Translate the first letter of each word to upper case in the sentence.
<code>instr(str, substr)</code>	Locate the position of the first occurrence of substr column in the given string.
<code>length(col)</code>	Computes the character length of string data or number of bytes of binary data.
<code>lower(col)</code>	Converts a string expression to lower case.

<code>levenshtein(left, right)</code>	Computes the Levenshtein distance of the two given strings.
<code>locate(substr, str[, pos])</code>	Locate the position of the first occurrence of substr in a string column, after position pos.
<code>lpad(col, len, pad)</code>	Left-pad the string column to width len with pad.
<code>ltrim(col)</code>	Trim the spaces from left end for the specified string value.
<code>octet_length(col)</code>	Calculates the byte length for the specified string column.
<code>regexp_extract(str, pattern, idx)</code>	Extract a specific group matched by a Java regex, from the specified string column.
<code>regexp_replace(string, pattern, replacement)</code>	Replace all substrings of the specified string value that match regexp with replacement.
<code>unbase64(col)</code>	Decodes a BASE64 encoded string column and returns it as a binary column.
<code>rpadd(col, len, pad)</code>	Right-pad the string column to width len with pad.
<code>repeat(col, n)</code>	Repeats a string column n times, and returns it as a new string column.
<code>rtrim(col)</code>	Trim the spaces from right end for the specified string value.
<code>soundex(col)</code>	Returns the SoundEx encoding for a string
<code>split(str, pattern[, limit])</code>	Splits str around matches of the given pattern.

<code>substring(str, pos, len)</code>	Substring starts at pos and is of length len when str is String type or returns the slice of byte array that starts at pos in byte and is of length len when str is Binary type.
<code>substring_index(str, delim, count)</code>	Returns the substring from string str before count occurrences of the delimiter delim.
<code>overlay(src, replace, pos[, len])</code>	Overlay the specified portion of src with replace, starting from byte position pos of src and proceeding for len bytes.
<code>sentences(string[, language, country])</code>	Splits a string into arrays of sentences, where each sentence is an array of words.
<code>translate(srcCol, matching, replace)</code>	A function translates any character in the srcCol by a character in matching.
<code>trim(col)</code>	Trim the spaces from both ends for the specified string column.
<code>upper(col)</code>	Converts a string expression to uppercase.

UDF

<code>call_udf(udfName, *cols)</code>	Call a user-defined function.
<code>pandas_udf([f, returnType, functionType])</code>	Creates a pandas user defined function (a.k.a.
<code>udf([f, returnType])</code>	Creates a user defined function (UDF).
<code>unwrap_udt(col)</code>	Unwrap UDT data type column into its underlying type.

Misc Functions

<code>md5(col)</code>	Calculates the MD5 digest and returns the value as a 32 character hex string.
<code>sha1(col)</code>	Returns the hex string result of SHA-1.
<code>sha2(col, numBits)</code>	Returns the hex string result of SHA-2 family of hash functions (SHA-224, SHA-256, SHA-384, and SHA-512).
<code>crc32(col)</code>	Calculates the cyclic redundancy check value (CRC32) of a binary column and returns the value as a bigint.
<code>hash(*cols)</code>	Calculates the hash code of given columns, and returns the result as an int column.
<code>xxhash64(*cols)</code>	Calculates the hash code of given columns using the 64-bit variant of the xxHash algorithm, and returns the result as a long column.
<code>assert_true(col[, errMsg])</code>	Returns null if the input column is true; throws an exception with the provided error message otherwise.
<code>raise_error(errMsg)</code>	Throws an exception with the provided error message.

DataFrame

<code>DataFrame.__getattr__(name)</code>	Returns the Column denoted by name.
<code>DataFrame.__getitem__(item)</code>	Returns the column as a Column .
<code>DataFrame.agg(*exprs)</code>	Aggregate on the entire DataFrame without groups (shorthand for <code>df.groupBy().agg()</code>).

DataFrame.alias (alias)	Returns a new DataFrame with an alias set.
DataFrame.approxQuantile (col, probabilities, ...)	Calculates the approximate quantiles of numerical columns of a DataFrame .
DataFrame.cache ()	Persists the DataFrame with the default storage level (MEMORY_AND_DISK).
DataFrame.checkpoint ([eager])	Returns a checkpointed version of this DataFrame .
DataFrame.coalesce (numPartitions)	Returns a new DataFrame that has exactly numPartitions partitions.
DataFrame.colRegex (colName)	Selects column based on the column name specified as a regex and returns it as Column .
DataFrame.collect ()	Returns all the records as a list of Row .
DataFrame.columns	Retrieves the names of all columns in the DataFrame as a list.
DataFrame.corr (col1, col2[, method])	Calculates the correlation of two columns of a DataFrame as a double value.
DataFrame.count ()	Returns the number of rows in this DataFrame .
DataFrame.cov (col1, col2)	Calculate the sample covariance for the given columns, specified by their names, as a double value.
DataFrame.createGlobalTempView (name)	Creates a global temporary view with this DataFrame .
DataFrame.createOrReplaceGlobalTempView (name)	Creates or replaces a global temporary view using the given name.

<code>DataFrame.createOrReplaceTempView(name)</code>	Creates or replaces a local temporary view with this <code>DataFrame</code> .
<code>DataFrame.createTempView(name)</code>	Creates a local temporary view with this <code>DataFrame</code> .
<code>DataFrame.crossJoin(other)</code>	Returns the cartesian product with another <code>DataFrame</code> .
<code>DataFrame.crosstab(col1, col2)</code>	Computes a pair-wise frequency table of the given columns.
<code>DataFrame.cube(*cols)</code>	Create a multi-dimensional cube for the current <code>DataFrame</code> using the specified columns, so we can run aggregations on them.
<code>DataFrame.describe(*cols)</code>	Computes basic statistics for numeric and string columns.
<code>DataFrame.distinct()</code>	Returns a new <code>DataFrame</code> containing the distinct rows in this <code>DataFrame</code> .
<code>DataFrame.drop(*cols)</code>	Returns a new <code>DataFrame</code> without specified columns.
<code>DataFrame.dropDuplicates([subset])</code>	Return a new <code>DataFrame</code> with duplicate rows removed, optionally only considering certain columns.
<code>DataFrame.dropDuplicatesWithinWatermark([subset])</code>	Return a new <code>DataFrame</code> with duplicate rows removed,
<code>DataFrame.drop_duplicates([subset])</code>	<code>drop_duplicates()</code> is an alias for <code>dropDuplicates()</code> .
<code>DataFrame.dropna([how, thresh, subset])</code>	Returns a new <code>DataFrame</code> omitting rows with null values.
<code>DataFrame.dtypes</code>	Returns all column names and their data types as a list.

<code>DataFrame.exceptAll(other)</code>	Return a new DataFrame containing rows in this DataFrame but not in another DataFrame while preserving duplicates.
<code>DataFrame.explain([extended, mode])</code>	Prints the (logical and physical) plans to the console for debugging purposes.
<code>DataFrame.fillna(value[, subset])</code>	Replace null values, alias for <code>na.fill()</code> .
<code>DataFrame.filter(condition)</code>	Filters rows using the given condition.
<code>DataFrame.first()</code>	Returns the first row as a Row .
<code>DataFrame.foreach(f)</code>	Applies the <code>f</code> function to all Row of this DataFrame .
<code>DataFrame.foreachPartition(f)</code>	Applies the <code>f</code> function to each partition of this DataFrame .
<code>DataFrame.freqItems(cols[, support])</code>	Finding frequent items for columns, possibly with false positives.
<code>DataFrame.groupBy(*cols)</code>	Groups the DataFrame using the specified columns, so we can run aggregation on them.
<code>DataFrame.head([n])</code>	Returns the first <code>n</code> rows.
<code>DataFrame.hint(name, *parameters)</code>	Specifies some hint on the current DataFrame .
<code>DataFrame.inputFiles()</code>	Returns a best-effort snapshot of the files that compose this DataFrame .
<code>DataFrame.intersect(other)</code>	Return a new DataFrame containing rows only in both this DataFrame and another DataFrame .

<code>DataFrame.intersectAll(other)</code>	Return a new DataFrame containing rows in both this DataFrame and another DataFrame while preserving duplicates.
<code>DataFrame.isEmpty()</code>	Checks if the DataFrame is empty and returns a boolean value.
<code>DataFrame.isLocal()</code>	Returns True if the <code>collect()</code> and <code>take()</code> methods can be run locally (without any Spark executors).
<code>DataFrame.isStreaming</code>	Returns True if this DataFrame contains one or more sources that continuously return data as it arrives.
<code>DataFrame.join(other[, on, how])</code>	Joins with another DataFrame , using the given join expression.
<code>DataFrame.limit(num)</code>	Limits the result count to the number specified.
<code>DataFrame.localCheckpoint([eager])</code>	Returns a locally checkpointed version of this DataFrame .
<code>DataFrame.mapInPandas(func, schema[, barrier])</code>	Maps an iterator of batches in the current DataFrame using a Python native function that takes and outputs a pandas DataFrame, and returns the result as a DataFrame .
<code>DataFrame.mapInArrow(func, schema[, barrier])</code>	Maps an iterator of batches in the current DataFrame using a Python native function that takes and outputs a PyArrow's RecordBatch, and returns the result as a DataFrame .
<code>DataFrame.melt(ids, values, ...)</code>	Unpivot a DataFrame from wide format to long format, optionally leaving identifier columns set.
<code>DataFrame.na</code>	Returns a DataFrameNaFunctions for handling missing values.

DataFrame.observe (observation, *exprs)	Define (named) metrics to observe on the DataFrame.
DataFrame.offset (num)	Returns a new :class: DataFrame by skipping the first n rows.
DataFrame.orderBy (*cols, **kwargs)	Returns a new DataFrame sorted by the specified column(s).
DataFrame.persist ([storageLevel])	Sets the storage level to persist the contents of the DataFrame across operations after the first time it is computed.
DataFrame.printSchema ([level])	Prints out the schema in the tree format.
DataFrame.randomSplit (weights[, seed])	Randomly splits this DataFrame with the provided weights.
DataFrame.rdd	Returns the content as an pyspark.RDD of Row .
DataFrame.registerTempTable (name)	Registers this DataFrame as a temporary table using the given name.
DataFrame.repartition (numPartitions, *cols)	Returns a new DataFrame partitioned by the given partitioning expressions.
DataFrame.repartitionByRange (numPartitions, ...)	Returns a new DataFrame partitioned by the given partitioning expressions.
DataFrame.replace (to_replace[, value, subset])	Returns a new DataFrame replacing a value with another value.
DataFrame.rollup (*cols)	Create a multi-dimensional rollup for the current DataFrame using the specified columns, so we can run aggregation on them.

<code>DataFrame.sameSemantics(other)</code>	Returns True when the logical query plans inside both <code>DataFrame</code> s are equal and therefore return the same results.
<code>DataFrame.sample([withReplacement, ...])</code>	Returns a sampled subset of this <code>DataFrame</code> .
<code>DataFrame.sampleBy(col, fractions[, seed])</code>	Returns a stratified sample without replacement based on the fraction given on each stratum.
<code>DataFrame.schema</code>	Returns the schema of this <code>DataFrame</code> as a <code>pyspark.sql.types.StructType</code> .
<code>DataFrame.select(*cols)</code>	Projects a set of expressions and returns a new <code>DataFrame</code> .
<code>DataFrame.selectExpr(*expr)</code>	Projects a set of SQL expressions and returns a new <code>DataFrame</code> .
<code>DataFrame.semanticHash()</code>	Returns a hash code of the logical query plan against this <code>DataFrame</code> .
<code>DataFrame.show([n, truncate, vertical])</code>	Prints the first n rows to the console.
<code>DataFrame.sort(*cols, **kwargs)</code>	Returns a new <code>DataFrame</code> sorted by the specified column(s).
<code>DataFrame.sortWithinPartitions(*cols, **kwargs)</code>	Returns a new <code>DataFrame</code> with each partition sorted by the specified column(s).
<code>DataFrame.sparkSession</code>	Returns Spark session that created this <code>DataFrame</code> .
<code>DataFrame.stat</code>	Returns a <code>DataFrameStatFunctions</code> for statistic functions.
<code>DataFrame.storageLevel</code>	Get the <code>DataFrame</code> 's current storage level.

<code>DataFrame.subtract(other)</code>	Return a new DataFrame containing rows in this DataFrame but not in another DataFrame .
<code>DataFrame.summary(*statistics)</code>	Computes specified statistics for numeric and string columns.
<code>DataFrame.tail(num)</code>	Returns the last num rows as a list of Row .
<code>DataFrame.take(num)</code>	Returns the first num rows as a list of Row .
<code>DataFrame.to(schema)</code>	Returns a new DataFrame where each row is reconciled to match the specified schema.
<code>DataFrame.toDF(*cols)</code>	Returns a new DataFrame that with new specified column names
<code>DataFrame.toJSON([use_unicode])</code>	Converts a DataFrame into a RDD of string.
<code>DataFrame.toLocalIterator([prefetchPartitions])</code>	Returns an iterator that contains all of the rows in this DataFrame .
<code>DataFrame.toPandas()</code>	Returns the contents of this DataFrame as Pandas <code>pandas.DataFrame</code> .
<code>DataFrame.to_pandas_on_spark([index_col])</code>	
<code>DataFrame.transform(func, *args, **kwargs)</code>	Returns a new DataFrame .
<code>DataFrame.union(other)</code>	Return a new DataFrame containing the union of rows in this and another DataFrame .
<code>DataFrame.unionAll(other)</code>	Return a new DataFrame containing the union of rows in this and another DataFrame .

<code>DataFrame.unionByName(other[, ...])</code>	Returns a new DataFrame containing a union of rows in this and another DataFrame .
<code>DataFrame.unpersist([blocking])</code>	Marks the DataFrame as non-persistent, and removes all blocks for it from memory and disk.
<code>DataFrame.unpivot(ids, values, ...)</code>	Unpivot a DataFrame from wide format to long format, optionally leaving identifier columns set.
<code>DataFrame.where(condition)</code>	<code>where()</code> is an alias for <code>filter()</code> .
<code>DataFrame.withColumn(colName, col)</code>	Returns a new DataFrame by adding a column or replacing the existing column that has the same name.
<code>DataFrame.withColumns(*colsMap)</code>	Returns a new DataFrame by adding multiple columns or replacing the existing columns that have the same names.
<code>DataFrame.withColumnRenamed(existing, new)</code>	Returns a new DataFrame by renaming an existing column.
<code>DataFrame.withColumnsRenamed(colsMap)</code>	Returns a new DataFrame by renaming multiple columns.
<code>DataFrame.withMetadata(columnName, metadata)</code>	Returns a new DataFrame by updating an existing column with metadata.
<code>DataFrame.withWatermark(eventTime, ...)</code>	Defines an event time watermark for this DataFrame .
<code>DataFrame.write</code>	Interface for saving the content of the non-streaming DataFrame out into external storage.
<code>DataFrame.writeStream</code>	Interface for saving the content of the streaming DataFrame out into external storage.

DataFrame.writeTo (table)	Create a write configuration builder for v2 sources.
DataFrame.pandas_api ([index_col])	Converts the existing DataFrame into a pandas-on-Spark DataFrame.
DataFrameNaFunctions.drop ([how, thresh, subset])	Returns a new DataFrame omitting rows with null values.
DataFrameNaFunctions.fill (value[, subset])	Replace null values, alias for <code>na.fill()</code> .
DataFrameNaFunctions.replace (to_replace[, ...])	Returns a new DataFrame replacing a value with another value.
DataFrameStatFunctions.approxQuantile (col, ...)	Calculates the approximate quantiles of numerical columns of a DataFrame .
DataFrameStatFunctions.corr (col1, col2[, method])	Calculates the correlation of two columns of a DataFrame as a double value.
DataFrameStatFunctions.cov (col1, col2)	Calculate the sample covariance for the given columns, specified by their names, as a double value.
DataFrameStatFunctions.crosstab (col1, col2)	Computes a pair-wise frequency table of the given columns.
DataFrameStatFunctions.freqItems (cols[, support])	Finding frequent items for columns, possibly with false positives.
DataFrameStatFunctions.sampleBy (col, fractions)	Returns a stratified sample without replacement based on the fraction given on each stratum.