DUMPSARENA

Databricks Certified Data Engineer Professional Exam

Databricks Databricks-Certified-Professional-Data-Engineer

Version Demo

Total Demo Questions: 25

Total Premium Questions: 457

Buy Premium PDF

https://dumpsarena.co

sales@dumpsarena.co

sales@dumpsarena.co
dumpsarena.co

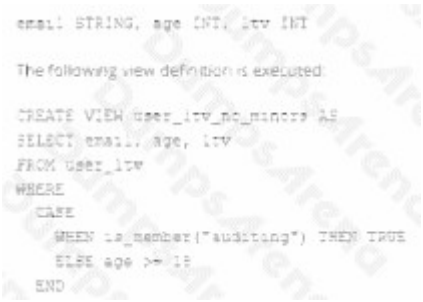
Topic Break Down

Topic	No. of Questions
Topic 1, Data Processing	188
Topic 2, Data Management	105
Topic 3, Data Governance	30
Topic 4, Data Security	29
Topic 5, Monitoring and Logging	28
Topic 6, Deployment and Automation	76
Topic 7, Mix Questions	1
Total	457

QUESTION NO: 1

A table named user_ltv is being used to create a view that will be used by data analysis on various teams. Users in the workspace are configured into groups, which are used for setting up data access using ACLs.

The user_ltv table has the following schema:



An analyze who is not a member of the auditing group executing the following query:



Which result will be returned by this query?

A. All columns will be displayed normally for those records that have an age greater than 18; records not meeting this condition will be omitted.

B. All columns will be displayed normally for those records that have an age greater than 17; records not meeting this condition will be omitted.

C. All age values less than 18 will be returned as null values all other columns will be returned with the values in user_ltv.

D. All records from all columns will be displayed with the values in user_ltv.

ANSWER: A

Explanation:

All columns will be displayed normally for those records that have an age greater than 18; records not meeting this condition will be omitted. The view uses the current user's group membership to determine the predicate applied to rows. Because the querying analyst is not a member of the `auditing` group, `is_member('auditing')` evaluates to false for that session. The conditional expression therefore takes its non-auditor branch, which applies the `age > 18` condition as the row-level filter. Rows satisfying that predicate are returned from `user_ltv`, with their selected column values unchanged. This is a typical dynamic-view pattern: group membership is evaluated at query time, allowing the same view definition to present an appropriate subset of table rows to different users without requiring separate views. Databricks documents `is_member` as a function that returns whether the current user belongs to the specified workspace group, making it suitable for dynamically enforcing access logic in SQL. For details, see the Databricks is_member function documentation and the guidance on dynamic views.

QUESTION NO: 2

The marketing team is looking to share data in an aggregate table with the sales organization, but the field names used by the teams do not match, and a number of marketing specific fields have not been approval for the sales org.

Which of the following solutions addresses the situation while emphasizing simplicity?

A. Create a view on the marketing table selecting only these fields approved for the sales team alias the names of any fields that should be standardized to the sales naming conventions.

B. Use a CTAS statement to create a derivative table from the marketing table configure a production jon to propagation changes.

C. Add a parallel table write to the current production pipeline, updating a new sales table that varies as required from marketing table.

D. Create a new table with the required schema and use Delta Lake ' s DEEP CLONE functionality to sync up changes committed to one table to the corresponding table.

ANSWER: A

Explanation:

Create a view on the marketing table selecting only these fields approved for the sales team alias the names of any fields that should be standardized to the sales naming conventions. is the simplest solution because a view provides a tailored, governed interface over the existing aggregate table without creating or maintaining a second physical copy of the data. Its query can explicitly project only the columns approved for sales, preventing unapproved marketing-specific fields from appearing in the shared interface. SQL column aliases can then present the selected fields using the sales organization's naming conventions while leaving the underlying marketing schema unchanged.

Because the view is evaluated against its source table, consumers see current underlying data without requiring a separate synchronization workflow, duplicate storage, or modifications to the existing production pipeline. This supports a clear producer-consumer contract: marketing retains ownership of its source table, while sales receives a purpose-built representation appropriate to its access and semantic needs. Databricks supports creating views from SQL queries, including selecting columns and assigning aliases; views can also be used as a data-sharing abstraction. See the Databricks CREATE VIEW reference and Databricks SELECT reference.

QUESTION NO: 3

You are ingesting daily customer files into a Bronze Delta table. The source system can resend records for existing customer IDs, and the target table must contain the latest record for each customer ID without creating duplicate records. Which Delta Lake command is best suited for this requirement?

A. MERGE INTO

B. INSERT OVERWRITE

C. VACUUM

D. OPTIMIZE

E. COPY INTO

ANSWER: A

Explanation:

`MERGE INTO` is the appropriate command because it can atomically compare source records with an existing Delta target table using a matching condition, update matching rows, and insert rows that do not already exist. This supports an upsert pattern for source deliveries that contain both new and changed customer records.

A typical implementation matches on `customer_id`, uses `WHEN MATCHED THEN UPDATE` for existing customers, and `WHEN NOT MATCHED THEN INSERT` for new customers. The source should first be deduplicated so that no more than one source row matches the same target row in a single merge operation. Delta Lake records the operation as an atomic transaction, so readers see either the state before the merge or the completed state after it. See the Databricks documentation for MERGE into a Delta table.

QUESTION NO: 4

Which of the following SQL statement can be used to query a table by eliminating duplicate rows from the query results?

A. SELECT DISTINCT * FROM table_name

B. SELECT DISTINCT * FROM table_name HAVING COUNT(*) > 1

C. SELECT DISTINCT_ROWS (*) FROM table_name

D. SELECT * FROM table_name GROUP BY * HAVING COUNT(*) < 1

E. SELECT * FROM table_name GROUP BY * HAVING COUNT(*) > 1

ANSWER: A

Explanation:

`SELECT DISTINCT * FROM table_name` is the correct statement for returning query results with duplicate rows eliminated. In Databricks SQL, `DISTINCT` applies to the complete set of expressions in the `SELECT` list. Because the asterisk expands to all columns in the table, this query returns one row for each unique combination of all column values. If two or more source rows contain identical values across every selected column, only one of those identical result rows is included in the output. This is useful when the goal is result-set deduplication rather than modifying, deleting, or otherwise changing the underlying table data. The operation occurs as part of query evaluation, so the original records in `table_name` remain unchanged. Databricks documents `SELECT DISTINCT` as removing duplicate rows from a result set; its SQL language reference also specifies the syntax and behavior of the `SELECT` statement. For further details, see the Databricks SELECT reference and the Databricks SELECT DISTINCT reference.

QUESTION NO: 5

Which of the following statements are true about a lakehouse?

A. Lakehouse only supports Machine learning workloads and Data warehouses support BI workloads

B. Lakehouse only supports end-to-end streaming workloads and Data warehouses support Batch workloads

C. Lakehouse does not support ACID

D. Lakehouse do not support SQL

E. Lakehouse supports Transactions

ANSWER: E

Explanation:

Lakehouse supports Transactions is correct because the lakehouse architecture combines the low-cost, open storage characteristics of a data lake with the data-management capabilities traditionally associated with data warehouses. In Databricks, Delta Lake is the storage layer that provides reliable transactional behavior for lakehouse tables. Delta Lake records changes in a transaction log and uses that log to provide ACID transaction guarantees: updates either complete as a consistent, durable transaction or do not take effect. This allows multiple readers and writers to operate on tables reliably while preserving data consistency, which is essential for production data engineering, streaming ingestion, batch processing, and analytics workloads. Transaction support also enables dependable operations such as concurrent writes, schema enforcement and evolution, time travel, and versioned table history. These capabilities mean that teams can use open cloud-object storage for lakehouse data while retaining the correctness and governance expected for managed tabular data. Databricks describes Delta Lake as the foundation for reliable lakehouse tables and documents its ACID transaction guarantees in the Delta Lake documentation. The underlying protocol and transaction-log design are also described in the Delta Lake transaction log documentation.

QUESTION NO: 6

A task orchestrator has been configured to run two hourly tasks. First, an outside system writes Parquet data to a directory mounted at /mnt/raw_orders/. After this data is written, a Databricks job containing the following code is executed:

(spark.readStream

.format(" parquet ")

.load(" /mnt/raw_orders/ ")

.withWatermark(" time " , " 2 hours ")

.dropDuplicates([" customer_id " , " order_id "])

.writeStream

.trigger(once=True)

.table(" orders ")

)

Assume that the fields customer_id and order_id serve as a composite key to uniquely identify each order, and that the time field indicates when the record was queued in the source system. If the upstream system is known to occasionally enqueue duplicate entries for a single order hours apart, which statement is correct?

A. The orders table will not contain duplicates, but records arriving more than 2 hours late will be ignored and missing from the table.

B. The orders table will contain only the most recent 2 hours of records and no duplicates will be present.

C. All records will be held in the state store for 2 hours before being deduplicated and committed to the orders table.

D. Duplicate records enqueued more than 2 hours apart may be retained and the orders table may contain duplicate records with the same customer_id and order_id.

E. The orders table will not contain duplicate customer_id and order_id pairs, but because time is not included in the deduplication keys, the watermark does not bound the deduplication state.

ANSWER: E

Explanation:

The orders table will not contain duplicate composite keys, but the watermark does not bound the deduplication state because `time` is not included in the `dropDuplicates` key is correct. In a streaming query, `dropDuplicates(["customer_id", "order_id"])` tracks previously seen composite keys across micro-batches. Once a key has been written, a later record with the same customer and order identifiers is suppressed, including one received hours later. The configured watermark is associated with the `time` event-time column, but state eviction for standard streaming `dropDuplicates` depends on an event-time attribute being part of the deduplication key. Since the keys here contain only `customer_id` and `order_id`, the query cannot use `time` to evict an individual deduplication key safely. Consequently, state can grow without a watermark-based bound while duplicates remain suppressed. For bounded deduplication when the event-time column is not itself part of the business key, Databricks supports `dropDuplicatesWithinWatermark`, which is specifically intended for deduplicating within a defined event-time range. See the Spark dropDuplicatesWithinWatermark documentation and the Spark streaming deduplication guide.

QUESTION NO: 7

A data engineer wants to reflector the following DLT code, which includes multiple definition with very similar code:



In an attempt to programmatically create these tables using a parameterized table definition, the data engineer writes the following code.



The pipeline runs an update with this refactored code, but generates a different DAG showing incorrect configuration values for tables.

How can the data engineer fix this?

A. Convert the list of configuration values to a dictionary of table settings, using table names as keys.

B. Convert the list of configuration values to a dictionary of table settings, using different input the for loop.

C. Load the configuration values for these tables from a separate file, located at a path provided by a pipeline parameter.

D. Wrap the loop inside another table definition, using generalized names and properties to replace with those from the inner table

E. Use a function factory that takes each table configuration as arguments and creates one decorated table definition per loop iteration.

ANSWER: E

Explanation:

Use a function factory that accepts the table-specific configuration as arguments and defines one decorated dataset function for that specific set of values. The factory must then be called once per table configuration inside the loop. This is necessary because Python closures capture variables from an enclosing scope by reference rather than preserving the value from each loop iteration. If the decorated functions directly reference loop variables, each function can resolve those variables after the loop has completed, causing the pipeline to apply the final iteration's name, comment, properties, or other settings to multiple datasets. Passing values into a factory function creates a distinct local scope for each generated definition, so each decorator receives the intended table name and each dataset body uses its corresponding configuration. Databricks supports using Python functions to programmatically create multiple Lakeflow Declarative Pipelines datasets, provided that each generated function has its configuration bound correctly at creation time. See the Databricks guidance for developing pipelines with Python and the Python language reference for pipeline decorators.

QUESTION NO: 8

A data team is automating a daily multi-task ETL pipeline in Databricks. The pipeline includes a notebook for ingesting raw data, a Python wheel task for data transformation, and a SQL query to update aggregates. They want to trigger the pipeline programmatically and see previous runs in the GUI. They need to ensure tasks are retried on failure and stakeholders are notified by email if any task fails.

Which two approaches will meet these requirements? *(Choose 2 answers)*

A. Use the REST API endpoint /jobs/runs/submit to trigger each task individually as separate job runs and implement retries using custom logic in the orchestrator.

B. Create a multi-task job using the UI, Databricks Asset Bundles (DABs), or the Jobs REST API (/jobs/create) with notebook, Python wheel, and SQL tasks. Configure task-level retries and email notifications in the job definition.

C. Trigger the job programmatically using the Databricks Jobs REST API (/jobs/run-now), the CLI (databricks jobs run-now), or one of the Databricks SDKs.

D. Create a single orchestrator notebook that calls each step with dbutils.notebook.run(), defining a job for that notebook and configuring retries and notifications at the notebook level.

E. Use Databricks Asset Bundles (DABs) to deploy the workflow, then trigger individual tasks directly by referencing each task's notebook or script path in the workspace.

ANSWER: B C

Explanation:

Create a multi-task job using the UI, Databricks Asset Bundles (DABs), or the Jobs REST API (/jobs/create) with notebook, Python wheel, and SQL tasks. Configure task-level retries and email notifications in the job definition. This defines a single Databricks workflow containing the required heterogeneous task types and their dependencies. Job settings support retry behavior for individual tasks, while notification settings can send email alerts for job failures. Because the work is executed as a Databricks job, its run history, task statuses, outputs, and failure details are available in the Jobs user interface, providing the requested operational visibility.

Trigger the job programmatically using the Databricks Jobs REST API (/jobs/run-now), the CLI (databricks jobs run-now), or one of the Databricks SDKs. These supported interfaces start a run of the already-defined job rather than bypassing its workflow configuration. Consequently, every programmatically initiated daily execution retains the configured task retry and notification behavior and is recorded as a run of the same job for later inspection in the GUI. Databricks documents job creation and task configuration in its Jobs documentation, and documents programmatic execution through the Jobs run-now API.

QUESTION NO: 9

A Lakeflow Spark Declarative Pipelines pipeline must continue processing valid records while sending records that fail a data-quality rule to a separate table for investigation. Which design pattern should be used?

A. Use a quarantine pattern with pipeline expectations

B. Use VACUUM to remove invalid records

C. Use time travel to prevent invalid records from arriving

D. Use a global temporary view for failed records

E. Use SQL warehouse auto stop after a failed expectation

ANSWER: A

Explanation:

The quarantine pattern is correct because it separates records that pass validation from records that fail defined quality rules. A pipeline can apply expectations to identify invalid records and write those records, along with relevant validation metadata, to a quarantine table while allowing valid records to continue to the refined target table.

This approach is useful when rejecting invalid data entirely would interrupt an otherwise healthy pipeline, but retaining the failed records is necessary for remediation, source-system feedback, and auditability. The quarantine table can be monitored and reviewed independently, while the main pipeline remains available to downstream consumers. Expectations also provide metrics in the pipeline event log for monitoring data-quality outcomes. See the Databricks documentation for pipeline expectations and monitoring pipeline event logs.

QUESTION NO: 10

A data engineer is creating a daily reporting job. There are two reporting notebooks—one for weekdays and one for weekends. An "if/else condition" task is configured as {{job.start_time.is_weekday}} == true to route the job to either the weekday or weekend notebook tasks. The same job would be used across multiple time zones.

Which action should a senior data engineer take upon reviewing the job to merge or reject the pull request?

A. Reject, as the {{job.start_time.is_weekday}} is for the **UTC timezone** .

B. Reject, as the {{job.start_time.is_weekday}} is not a valid value reference.

C. Merge, as the job configuration looks good.

D. Reject, as they should use {{job.trigger_time.is_weekday}} instead.

ANSWER: A

Explanation:

Reject, as the {{job.start_time.is_weekday}} is for the **UTC timezone**. The `{{job.start_time.is_weekday}}` dynamic value reference is valid and resolves to a Boolean indicating whether the job's start date is a weekday. However, Databricks evaluates job start-time dynamic values using UTC. This means that a run starting late on a local Sunday in a time zone west of UTC could already be Monday in UTC, causing the weekday notebook to run when the intended local-business-calendar behavior is the weekend notebook. Similarly, local dates east of UTC can differ from the UTC date around midnight.

Because this reporting workflow is intended for use across multiple time zones, the routing logic must explicitly account for the relevant business time zone rather than relying on a UTC-derived weekday flag. The senior engineer should therefore reject the pull request and request a design that derives the reporting date and weekday according to the applicable local or business time zone. Databricks documents `job.start_time` dynamic values, including `is_weekday`, as UTC-based values in its dynamic value references documentation. The condition task can then use a value that represents the intended localized reporting date; see the If/else condition task documentation for condition-task behavior.

QUESTION NO: 11

Which statement describes the default execution mode for Databricks Auto Loader?

A. New files are identified by listing the input directory; new files are incrementally and idempotently loaded into the target Delta Lake table.

B. Cloud vendor-specific queue storage and notification services are configured to track newly arriving files; new files are incrementally and impotently into the target Delta Lake table.

C. Webhook trigger Databricks job to run anytime new data arrives in a source directory; new data automatically merged into target tables using rules inferred from the data.

D. New files are identified by listing the input directory; the target table is materialized by directory querying all valid files in the source directory.

ANSWER: A

Explanation:

New files are identified by listing the input directory; new files are incrementally and idempotently loaded into the target Delta Lake table. This describes Auto Loader's default directory-listing file-discovery mode. In this mode, Auto Loader checks the configured cloud storage path for files that have not previously been processed, rather than requiring cloud-event notifications. It maintains file-discovery state in the streaming checkpoint, which allows it to incrementally process newly discovered files across runs and recover reliably after restarts. The resulting records can then be written to a Delta Lake target using Structured Streaming, with checkpointing supporting fault-tolerant, exactly-once processing semantics for the streaming query. This default is appropriate for sources where directory listing is practical and no notification-based discovery mode has been explicitly configured. Databricks documents directory listing as the default file-discovery mechanism and explains that Auto Loader tracks discovered files to avoid reprocessing them. See the Auto Loader file detection documentation and the Auto Loader overview for configuration and reliability details.

QUESTION NO: 12

A data engineer has created a new cluster using shared access mode with default configurations. The data engineer needs to allow the development team access to view the driver logs if needed.

What are the minimal cluster permissions that allow the development team to accomplish this?

A. CAN ATTACH TO

B. CAN MANAGE

C. CAN VIEW

D. CAN RESTART

ANSWER: A

Explanation:

`CAN ATTACH TO` is the minimum cluster permission that permits members of the development team to view driver logs. This permission is intended for users who need to use a cluster interactively, and it includes access to operational information available from the cluster, such as the Spark UI and driver logs. The team does not need administrative control over the cluster simply to inspect those logs; granting the attach-level permission follows least-privilege practice while enabling the requested diagnostic access.

Cluster access-control permissions are hierarchical by capability: permissions that allow management can also provide broader access, but they are unnecessary when the requirement is limited to viewing driver logs. On a shared access mode cluster, assigning `CAN ATTACH TO` to the appropriate group lets developers access the cluster-related diagnostic information without giving them permission to change cluster configuration, terminate it, or manage its permissions. Databricks documents the capabilities associated with cluster permissions, including access granted to users who can attach to a cluster, in its cluster access-control guidance: Cluster access control. For the driver-log interface and logging behavior, see View driver logs.

QUESTION NO: 13

The data architect has mandated that all tables in the Lakehouse should be configured as external Delta Lake tables.

Which approach will ensure that this requirement is met?

A. Whenever a database is being created, make sure that the location keyword is used

B. When configuring an external data warehouse for all table storage. leverage Databricks for all ELT.

C. Whenever a table is being created, make sure that the location keyword is used.

D. When tables are created, make sure that the external keyword is used in the create table statement.

E. When the workspace is being configured, make sure that external cloud object storage has been mounted.

ANSWER: C

Explanation:

"Whenever a table is being created, make sure that the location keyword is used." ensures that each table is created as an external table because the `LOCATION` clause explicitly specifies the cloud-storage directory where the table's Delta files reside. For example, a `CREATE TABLE ... USING DELTA LOCATION '...'` statement registers metadata for a Delta table while keeping its data at the declared storage path, rather than placing the data in a managed-storage location selected by the platform. Applying this requirement to every table-creation statement provides a direct, repeatable control that meets the architect's mandate across the Lakehouse. In Unity Catalog, the specified path should be governed through an external location and appropriate permissions; this lets Databricks securely access the cloud-storage path while the table remains external. The path must also be appropriate for the table and should not overlap with another table's storage location. Databricks documents that specifying `LOCATION` when creating a table creates an external table, and its SQL reference shows the supported syntax for Delta tables. See External tables and the CREATE TABLE USING syntax reference.

QUESTION NO: 14

A newly joined team member John Smith in the Marketing team currently has access read access to sales tables but does not have access to update the table, which of the following commands help you accomplish this?

A. GRANT UPDATE ON TABLE table_name TO john.smith@marketing.com

B. GRANT USAGE ON TABLE table_name TO john.smith@marketing.com

C. GRANT MODIFY ON TABLE table_name TO john.smith@marketing.com

D. GRANT UPDATE TO TABLE table_name ON john.smith@marketing.com

E. GRANT MODIFY TO TABLE table_name ON john.smith@marketing.com

ANSWER: C

Explanation:

`GRANT MODIFY ON TABLE table_name TO john.smith@marketing.com` is the appropriate command because the `MODIFY` privilege grants the ability to change data in a Unity Catalog table. This privilege covers row-changing operations such as `UPDATE`, `DELETE`, `INSERT`, and `MERGE`. Since John already has read access, granting `MODIFY` supplies the additional authorization needed to update records without altering the existing read permission model. In Databricks SQL, privileges are granted with the `GRANT <privilege> ON TABLE <table-name> TO <principal>` syntax. In a real command, an email-address principal should generally be delimited with backticks, for example `TO` `` `john.smith@marketing.com` ``, so that it is parsed correctly as a principal identifier. The privilege is applied to the named table; the user must also be able to use the containing catalog and schema to access it. See the Databricks documentation for GRANT syntax and the Unity Catalog privilege model.

QUESTION NO: 15

You noticed a colleague is manually copying the data to the backup folder prior to running an up-date command, incase if the update command did not provide the expected outcome so he can use the backup copy to replace table, which Delta Lake feature would you recommend simplifying the process?

A. Use time travel feature to refer old data instead of manually copying

B. Use DEEP CLONE to clone the table prior to update to make a backup copy

C. Use SHADOW copy of the table as preferred backup choice

D. Cloud object storage retains previous version of the file

E. Cloud object storage automatically backups the data

ANSWER: A

Explanation:

Use time travel feature to refer old data instead of manually copying is correct because Delta Lake records a transaction log for every committed table change. Before an update, the colleague does not need to create a separate physical copy of the table merely to preserve its current state. Delta time travel allows users to read the table exactly as it existed at a prior table version or timestamp, using syntax such as `VERSION AS OF` or `TIMESTAMP AS OF`. This provides a straightforward way to validate the pre-update contents, compare them with the updated result, or retrieve the earlier data if needed. When the update must be undone at the table level, Delta Lake can restore the table to a prior version with the `RESTORE TABLE ... TO VERSION AS OF ...` command; this capability relies on the same versioned transaction history. The required historical files and log entries must still be retained, so retention and VACUUM settings should be managed to preserve the recovery window required by the organization. Databricks documents time travel and version history at Delta Lake table history and time travel, including restoration guidance at Restore a Delta table to an earlier state.

QUESTION NO: 16

Which of the following technique can be used to implement fine-grained access control to rows and columns of the Delta table based on the user's access?

A. Use Unity catalog to grant access to rows and columns

B. Row and column access control lists

C. Use dynamic view functions

D. Data access control lists

E. Dynamic Access control lists with Unity Catalog

ANSWER: C

Explanation:

Use dynamic view functions is correct because a view can evaluate the identity or group membership of the user running the query and then apply conditional SQL logic to the underlying Delta table. For row-level control, the view can use a predicate that returns only those rows the current user is permitted to see. For example, an expression using `is_member()` can allow members of a privileged group to see all rows while restricting other users to a defined subset. For column-level protection, a `CASE` expression can return the original sensitive value only for authorized users and a redacted value for everyone else. Permissions are granted on the view, rather than directly on the underlying table, so consumers query the protected interface and the access logic is consistently enforced. This pattern is commonly called a dynamic view because its result changes according to the querying user's identity or group membership. Databricks documents dynamic views as a way to implement row filtering and column masking with functions such as `current_user()` and `is_member()`. See the Databricks dynamic views documentation and the is_member function reference.

QUESTION NO: 17

A Delta table receives frequent small batch writes. Query performance has degraded because the table contains a large number of small files. Which command should be used to compact the existing small files into larger files?

A. OPTIMIZE table_name

B. VACUUM table_name

C. ANALYZE TABLE table_name

D. DESCRIBE HISTORY table_name

E. RESTORE TABLE table_name

ANSWER: A

Explanation:

`OPTIMIZE` is correct because it rewrites the data files of a Delta table into a more efficient layout with fewer, larger files. Reducing excessive small files lowers file-open overhead and can improve scan performance for subsequent queries. The command can be run against the full table or selectively against relevant partitions when appropriate.

`OPTIMIZE` is a file-layout operation and does not change the logical rows in the table. It is distinct from `VACUUM`, which removes unreferenced files that are older than the retention threshold, and from `ANALYZE TABLE`, which computes optimizer statistics. Databricks also supports clustering capabilities that can improve data layout for selective queries, but the direct operation for compacting already existing small files is `OPTIMIZE`. See the Databricks documentation for Optimize Delta tables.

QUESTION NO: 18

Define an external SQL table by connecting to a local instance of an SQLite database using JDBC

A. 1. CREATE TABLE users_jdbc
2. USING SQLITE
3. OPTIONS (
4. url = "jdbc:/sqmple_db",
5. dbtable = "users"
6.)

B. 1. CREATE TABLE users_jdbc
2. USING SQL
3. URL = {server:"jdbc:/sqmple_db",dbtable: “users”}

C. 1. CREATE TABLE users_jdbc
2. USING SQL
3. OPTIONS (
4. url = "jdbc:sqlite:/sqmple_db",
5. dbtable = "users"
6.)

C. 1. CREATE TABLE users_jdbc
2. USING org.apache.spark.sql.jdbc.sqlite
3. OPTIONS (
4. url = "jdbc:/sqmple_db",
5. dbtable = "users"
6.)

D. 1. CREATE TABLE users_jdbc
2. USING org.apache.spark.sql.jdbc
3. OPTIONS (
4. url = "jdbc:sqlite:/sqmple_db",
5. dbtable = "users"
6.)

ANSWER: D

Explanation:

The statement `CREATE TABLE users_jdbc USING org.apache.spark.sql.jdbc OPTIONS (url = "jdbc:sqlite:/sqmple_db", dbtable = "users")` is the correct SQL pattern for creating a table backed by a JDBC data source. Spark SQL selects its general JDBC data source with `USING org.apache.spark.sql.jdbc` (or the equivalent short provider name `jdbc`). The `url` option supplies the JDBC connection URL, and SQLite URLs use the `jdbc:sqlite:` prefix followed by the database file location. The `dbtable` option identifies the source table to read, here `users`.

In practice, the SQLite JDBC driver must be available to the Databricks compute, and the SQLite database file must be reachable from that compute at the path used in the URL. JDBC tables are appropriate when Spark needs to access data held in an external relational database through the JDBC interface. Databricks documents JDBC connections as requiring a JDBC URL and connection properties, while Spark documents `dbtable` as the JDBC option used to specify the table or query being read. See the Databricks JDBC connection documentation and the Apache Spark JDBC data source documentation.

QUESTION NO: 19

You have configured AUTO LOADER to process incoming IOT data from cloud object storage every 15 mins, recently a change was made to the notebook code to update the processing logic but the team later realized that the notebook was failing for the last 24 hours, what steps team needs to take to reprocess the data that was not loaded after the notebook was corrected?

A. Move the files that were not processed to another location and manually copy the files into the ingestion path to reprocess them

B. Enable back_fill = TRUE to reprocess the data

C. Delete the checkpoint folder and run the autoloader again

D. Autoloader automatically re-processes data that was not loaded

E. Manually re-load the data

ANSWER: D

Explanation:

Autoloader automatically re-processes data that was not loaded is correct because Auto Loader runs as a Structured Streaming source and uses the streaming checkpoint to track processing progress. A micro-batch is only recorded as successfully completed after its processing and sink commit complete successfully. If the revised notebook fails during processing, the failed micro-batch is not committed to the checkpoint. Once the code is fixed and the query is restarted with the same checkpoint location, Structured Streaming retries the outstanding batch and Auto Loader supplies the files whose processing was not successfully completed. This recovery behavior enables fault-tolerant ingestion without requiring users to relocate files or recreate ingestion state.

Retaining the existing checkpoint is important because it preserves the stream's progress and allows recovery to continue from the last successful commit. The destination operation should also be idempotent or otherwise designed for streaming retry semantics, particularly where a failure could occur after an external side effect. Databricks documents that checkpoints store the state required for streaming query recovery, while Auto Loader uses checkpointing to provide exactly-once file-processing behavior. See Auto Loader production considerations and Structured Streaming checkpoints.

QUESTION NO: 20

Which of the following techniques structured streaming uses to ensure recovery of failures during stream processing?

A. Checkpointing and Watermarking

B. Write ahead logging and watermarking

C. Checkpointing and write-ahead logging

D. Delta time travel

E. The stream will failover to available nodes in the cluster
Checkpointing and Idempotent sinks

ANSWER: C

Explanation:

Checkpointing and write-ahead logging is correct because Structured Streaming persists durable progress metadata so that a query can resume safely after a driver, executor, or cluster failure. The configured checkpoint location stores information needed for recovery, including source offsets that identify which input data has been processed, query progress metadata, and state for stateful operations. This persisted information lets the restarted query determine the correct point from which to continue rather than treating the stream as entirely new.

Write-ahead logging is the complementary durability concept: progress and offset information is recorded persistently before that progress is considered complete. Together with checkpoint data, this enables the engine to replay or continue micro-batches consistently after interruption. For supported sources and sinks configured with appropriate semantics, this recovery model is a core part of Structured Streaming's fault-tolerance and end-to-end processing guarantees. The checkpoint path must therefore be durable, accessible after restart, and unique to the streaming query. Databricks documents checkpoint locations as the storage for stream progress and state required to recover a query, while Spark documents the checkpoint-based fault-tolerance model for Structured Streaming. See Databricks checkpoint documentation and Apache Spark Structured Streaming documentation.

QUESTION NO: 21

Which of the following technologies can be used to identify key areas of text when parsing Spark Driver log4j output?

A. Regex

B. Julia

C. pyspsark.ml.feature

D. Scala Datasets

E. C++

ANSWER: A

Explanation:

Regex is the appropriate technology for identifying meaningful fields in Spark Driver log4j output because log entries are text records with recurring structural patterns. A regular expression can recognize and extract portions of each line such as timestamps, log levels, logger or class names, thread identifiers, exception markers, and message content. This enables engineers to turn unstructured driver logs into fields that can be filtered, aggregated, monitored, or written to structured tables for troubleshooting and operational analysis.

Databricks supports regular-expression processing through Spark SQL functions, including `regexp_extract` for capturing a matching group and `regexp_replace` for transforming text. These functions can be used directly in SQL or through DataFrame APIs, allowing a scalable parsing workflow across many log records. For example, a capture group can isolate the severity token or timestamp from every log line, while another pattern can identify Java exception messages or stack-trace boundaries. This pattern-based approach is well suited to log4j output because its individual values vary but its textual layout commonly follows recognizable conventions. See the Databricks documentation for regexp_extract and Apache Spark's regular-expression SQL function reference.

QUESTION NO: 22

How are the operational aspects of **Lakeflow Declarative Pipelines** different from **Spark Structured Streaming** ?

A. Lakeflow Declarative Pipelines manage the orchestration of multi-stage pipelines automatically, while Structured Streaming requires external orchestration for complex dependencies.

B. Structured Streaming can process continuous data streams, while Lakeflow Declarative Pipelines cannot.

C. Lakeflow Declarative Pipelines can write to Delta Lake format, while Structured Streaming cannot.

D. Lakeflow Declarative Pipelines automatically handle schema evolution, while Structured Streaming always requires manual schema management.

ANSWER: A

Explanation:

Lakeflow Declarative Pipelines manage the orchestration of multi-stage pipelines automatically, while Structured Streaming requires external orchestration for complex dependencies. This reflects the declarative operating model of Lakeflow Declarative Pipelines: engineers define the intended tables and transformations, and the service determines the dependency graph and executes pipeline updates in the required order. The platform also provides managed operational capabilities for the pipeline, including infrastructure management, monitoring through pipeline event logs, data-quality expectations, and recovery behavior. This allows a set of dependent streaming and batch transformations to be operated as one managed pipeline rather than as separately coordinated application processes.

Spark Structured Streaming is the underlying stream-processing engine and provides APIs for defining and running a streaming query. It can power sophisticated streaming workloads, but when an implementation consists of multiple independently deployed stages with dependencies, the application owner must arrange that broader workflow coordination—for example, by using Databricks Jobs or another orchestrator—and manage the deployment and operational relationship among those stages. Databricks documents the declarative pipeline model and automatic handling of dataflow dependencies in its Lakeflow Declarative Pipelines documentation. The execution model and query-oriented APIs for the engine are described in the Apache Spark Structured Streaming programming guide.

QUESTION NO: 23

A query is taking too long to run. After investigating the Spark UI, the data engineer discovered a significant amount of **disk spill** . The compute instance being used has a core-to-memory ratio of 1:2.

What are the **two steps** the data engineer should take to minimize spillage? *(Choose 2 answers)*

A. Choose a compute instance with a higher memory-to-core ratio.

B. Choose a compute instance with more disk space.

C. Increase spark.sql.files.maxPartitionBytes.

D. Reduce spark.sql.files.maxPartitionBytes.

E. Choose a compute instance with more network bandwidth.

ANSWER: A D

Explanation:

Choose a compute instance with a higher memory-to-core ratio because disk spilling occurs when a Spark task cannot retain the working data required for operations such as joins, aggregations, and sorts in its available executor memory. A machine configuration with more memory available for each CPU core gives concurrently running tasks a larger memory budget. This directly reduces the likelihood that Spark must evict intermediate data from memory and write it to local disk. Databricks guidance for diagnosing memory issues in the Spark UI identifies spill as an indicator that tasks may require more memory or that the amount of data handled by an individual task should be reduced. See the Databricks Spark UI guide.

Reduce spark.sql.files.maxPartitionBytes because this setting controls the maximum number of bytes placed in a file-scan partition. Lowering it generally produces smaller input partitions, so each task processes less data at once and has a smaller in-memory working set. This is especially useful when file scans feed memory-intensive downstream processing. The setting should be tuned thoughtfully, since excessively small partitions can introduce scheduling overhead, but reducing partition size is an appropriate response to spill caused by oversized tasks. Databricks documents this configuration in its spark.sql.files.maxPartitionBytes reference.

QUESTION NO: 24

A data engineer needs to create a copy of a production Delta table for testing. The copy must be created quickly without duplicating the source table data files in cloud storage. Which Delta Lake command type should be used?

A. Use a shallow clone

B. Use a deep clone

C. Use VACUUM on the source table

D. Use INSERT OVERWRITE on the target table

E. Use a global temporary view

ANSWER: A

Explanation:

A shallow clone is correct because it creates an independent Delta table definition that references the data files of the source table rather than physically copying those files. This makes the clone fast and storage-efficient when a team needs a separate table for testing, validation, or experimentation.

The shallow clone has its own transaction log and can receive independent changes after creation. However, it depends on the source table data files remaining available. A deep clone instead copies both metadata and data files, which provides an independent physical copy but requires additional time and storage. See the Databricks documentation on Delta Lake clones.

QUESTION NO: 25

Which of the following programming languages can be used to build a Databricks SQL dashboard?

A. Python

B. Scala

C. SQL

D. R

E. All of the above

ANSWER: C

Explanation:

SQL is the language used to build a Databricks SQL dashboard. A dashboard is created by first authoring SQL queries in the Databricks SQL editor and then visualizing the query results with charts, tables, counters, filters, and other dashboard widgets. Each visualization on the dashboard is backed by a dataset, and that dataset is typically defined by a SQL query. This model lets dashboard authors use SQL to join, filter, aggregate, and otherwise shape data before presenting it to business users. Databricks SQL dashboards are designed for analytics and reporting workflows using SQL warehouses, including scheduled refreshes and sharing or publishing dashboards for consumption. While Databricks notebooks support languages such as Python, Scala, SQL, and R for broader data engineering and data science development, the question specifically concerns Databricks SQL dashboards and their query-based dataset authoring workflow. See the Databricks SQL dashboards documentation and the guide to defining dashboard datasets for details on creating datasets from SQL queries and using them in dashboard visualizations.