- Tasks not supported by DB repos (done by GIT provider):
  - create PR
  - delete, merge and rebase branches
  - review process
- Task supported by DB repos:
  - commit, push, pull, clone
  - trigger a CI/CD process
  - create and work with new branches
  - add comments and select the changes to commit
  - allow for full Git integration, including branching and merging, which is not directly supported within notebooks
  - support larger file sizes for version control compared to Notebooks
- Dropping tables:
  - `managed table`: drop table definition from metastore, data, metadata, and history from storage
  - `external table`: drop table from metastore, keep metadata, data, and history in storage
- Workspace level: classic access control list for tables, workspace, cluster
- Account level: Unity catalog to manage all the workspaces in an account
- A job cluster can not provide a way for a user to interact with a notebook once the job is submitted, but an interactive cluster allows to you display data, view visualizations write or edit quries, which makes it a perfect fit to investigate and analyze the data
- All of the data is broken down into one or many parquet files, log files are broken down into one or many JSON files, and each transaction creates a new data file and log file.
- `Control Plane`
  - Stored in DB Cloud Account
  - Manages user accounts, data sets, and clusters
  - Include backend services
  - Clusters and workspace configurations
  - User authentication and access control
  - DB Web Applications
  - Customer Managed Cloud Storage
  - Interactive notebook results
- `Data plane` contains
  - Customer data
  - Hosts the driver and worker nodes of a managed cluster
  - Main components: Workspaces and SQL Warehouses
  - Deploys the cluster's virtual machines
  - Runs in the Customer Cloud Account
- Create external table with
  `CREATE TABLE <table> (...)  USING DELTA LOCATION EXTERNAL`
- `MERGE INTO <table> AS ...  USING SELECT ...`
- Upgrade existing `managed table` to a Unity Catalog table with
  `CREATE OR REPLACE TABLE <table> FORMAT = UNITY USING DEEP CLONE <old_table>`
- Data Lakehouse cannot serve low query latency with high reliability for BI workloads, only suitable for batch workloads
- `SHALLOW CLONE`: Create a copy of a table and transaction logs quickly to test out applying

changes without the risk of modifying the current table

- `DEEP CLONE`: Fully copies data and metadata from a source table to a target
- `complete mode`: write stream data into target table, the table is overwritten for each batch
- Restore table:
  - `RESTORE TABLE <table> TO VERSION AS OF <version>`
  - `RESTORE [TABLE] <table> [ON] TIMESTAMP AS OF timestamp_expression`
- `COPY INTO` does not detect new files after the last load
- `INSERT OVERWRITE` replaces data by default, `CREATE OR REPLACE` replaces data and schema by default
- DLT pipeline in `production`:
  - `continuous mode`: all datasets will be updated continuously and the pipeline will not shut down. The compute resources will persist with the pipeline
  - `triggered mode`: all datasets will be updated once and the pipeline will shut down. The compute resources will persist to allow additional development and testing
- DLT pipeline in `development` using the `triggered mode`: DLT pipeline is designed to process the available data once per trigger rather than continuously. After clicking start, the pipeline processes any previously unprocessed data based on the current definitions. Once this batch of data is processed, the pipeline does not actively seek new data until it is triggered again, effectively making it shut down from an active processing perspective
- Setup an alert but use the custom template to notify the message in email's subject every time a KPI indicator increases beyond a threshold value
- `mergeSchema` instructs Spark to merge the schema of new data with the existing schema of the table. This allows for incremental updates and schema evolution.
- Prepend the `LIVE.` keyword to table name in DLT pipelines to create a table with SQL
- `FROM STREAM(LIVE.<another_table>)` to stream data from another live table
- Cluster size improves query latency
- Using the JDBC library (`org.apache.spark.sql.jdbc`), Spark SQL can extract data from any existing relational database that supports JDBC
- The events described in the question represent Change Data Capture (CDC) feed. CDC is logged at the source as events that contain both the data of the records along with metadata information:
  - Operation column indicating whether the specified record was inserted, updated, or deleted
  - Sequence column that is usually a timestamp indicating the order in which the changes happened.
  - Use `APPLY CHANGES INTO` statement to use DLT CDC functionality
- Use `job clusters` in `production` so that each job runs in a fully isolated environment
- In DB SQL, you can set a schedule to automatically refresh a query from the query's page
- `Data Explorer` in DB SQL:
  - review and change the owner of the table from Owner field
  - provides detailed information about tables, including metadata like the owner of the table
- DB SQL is a data warehouse on the DB Lakehouse platform that lets you run all your SQL and BI applications at scale
- `CREATE TABLE USING` allows to specify an external data source type like CSV format
- Specifying at least one notebook or library with the defined DLT syntax is mandatory for the creation and execution of a DLT pipeline

- Increasing the maximum bound of the SQL endpoint's scaling range is the most suitable solution to address the slow query performance caused by high concurrency with small queries. This allows the SQL endpoint to dynamically scale up its resources to handle the increased load effectively, resulting in faster query execution times
- Create a permanent view in DB SQL by using the `CREATE VIEW` statement followed by a `SELECT` query
- By configuring an alert with a webhook alert destination, the data engineer can automate notifications to the team for specific conditions met within the data, specifically, when an order's quantity hits 0. This setup allows for seamless integration with various communication platforms, ensuring that alerts are promptly delivered to the team through the chosen messaging service
- Auto Loader:
  - specifically designed to efficiently ingest data into Delta Lake on DB, making it ideal for scenarios where new files are continuously added to a directory
  - uses a `directory listing` or `file notification` to identify new files in the specified directory
  - processes only the new files since the last pipeline run, without reprocessing files that have already been ingested
- The Cloud Admin role typically handles the management of cloud resources, ensuring the right storage and access permissions are set across cloud services
- Spark Structured Streaming uses a combination of `checkpointing` and `idempotent sinks` to ensure fault tolerance and reliable tracking of data processing progress:
  - `Checkpointing` is a mechanism that saves the state of the streaming application at regular intervals to a durable storage system
  - `Checkpointing` is crucial for Structured Streaming to store the progress of the streaming query, including the offsets processed. This allows the system to recover from failures and resume processing from the last known state.
  - Idempotent sinks are those that can handle reprocessing of data without duplicating the results. This means if the same data is sent to the sink multiple times, the final outcome remains unchanged
  - `Write-ahead logs` (WALs) are used to ensure that any changes to the checkpoint are durably recorded before being applied. This combination guarantees fault tolerance and exactly-once processing semantics. This is the core mechanic for offset tracking
- An `ARRAY` data type is suitable for storing an arbitrary number of elements of the same data type
- File-based data: text files are processed with each line representing a string value in a single column
- Binary files: schema that includes metadata such as the file path, modification time, file length, and the content itself
- Cluster pools can help manage cloud costs effectively by allowing administrators to specify a minimum and maximum number of idle instances
- The Metastore Admin role is designed to handle specific data catalog management tasks, including the creation, modification, and assignment of privileges on data objects within the metastore
- DB Clusters UI feature is designed with built-in filtering options that enable users to view only the clusters they are authorized to access
- The `Data Plane` typically refers to the elements of a cloud service where the actual data

processing takes place. In the context of DB Workspace and Services, Workspace clusters are used for running interactive and scheduled data analysis jobs, while SQL Warehouses (formerly known as DB SQL) are optimized for running SQL queries on large datasets. These two components are central to the data processing capabilities of the DB platform
- `Serverless mode` dynamically scales resources based on demand. This means that during periods of high concurrency with many small queries, resources will automatically scale up to handle the load, minimizing latency. If already using a scaling configuration, increasing the maximum bound allows the SQL endpoint to allocate more resources during peak demand, improving performance for concurrent queries
- The `fan-out` pattern is optimal for executing parallel processing tasks in DB
- A storage credential is used to authenticate and authorize Unity Catalog to access external cloud storage services
- DLTs are designed to work with notebooks only
- The dot `(.)` syntax is used for subfields of STRUCT types and JSON
  $\rightarrow$ `SELECT <table>:<field>.<subfield`
- The colon `(:)` syntax is used for fields $\rightarrow$ `SELECT <table>:<field>`
- In DB, unmanaged tables will always specify a `LOCATION` during table creation
- Generated columns: `GENERATED ALWAYS AS ...`
- Higher order functions: `FILTER, EXIST, TRANSFORM, REDUCE`, not `MAP`
- Replayable data sources and `idempotent sinks` allow Structured Streaming to ensure end-to-end, exactly-once semantics under any failure condition
- Cluster Size: represents the number of cluster workers and size of compute resources available to run your queries and dashboards. The default is X-Large.
- Auto Stop determines whether the warehouse stops if it's idle for the specified number of minutes:
  - Pro and classic SQL warehouses: The default is 45 minutes, which is recommended for typical use. The minimum is 10 minutes.
  - Serverless SQL warehouses: The default is 10 minutes, which is recommended for typical use. The minimum is 5 minutes when you use the UI. Note that you can create a serverless SQL warehouse using the SQL warehouses API, in which case you can set the Auto Stop value as low as 1 minute
- Scaling sets the minimum and maximum number of clusters over which queries sent to the warehouse are distributed. The default is a minimum of one and maximum of one cluster
- DB maintains a history of your job runs for up to 60 days
- Dashboards in DB have built-in scheduling capabilities, allowing to automate the refresh process, eliminating the need for manual intervention
- The `APPLY CHANGES INTO` statement is designed for a need to synchronize two tables based on a stream of updates and new records. Define a unique identifier or primary key
- The DLT's event log includes information about pipeline operations, including audit logs and data lineage. Audit user actions by querying the event log for user_action events
- DB suggests that for detailed monitoring and observability of DLT pipelines, including update history and data quality reporting, one effective method is to query the event log directly. This involves creating a temporary view on the event log's storage path and then querying this view for specific metrics or update history
- The Runs tab in the Jobs UI is specifically designed to provide detailed insights into each execution of a Job, including the performance of individual tasks or notebooks
- The retention value provided while running `VACUUM` command is in `HOURS`. Default: 7 days
- Use a query profile to visualize the details of a query execution

- DB widgets:
  - `text`: Input a value in a text box
  - `dropdown`: Select a value from a list of provided values
  - `combobox`: Combination of text and dropdown. Select a value from a provided list or input one in the text box
  - `multiselect`: Select one or more values from a list of provided values
- `TRUNCATE TABLE` (DB SQL) removes all the rows from a table or partition
- `OPTIMIZE <table> ...`
- `COPY INTO` command is idempotent and files in the source location that have already been loaded are skipped
- Enforced constraints: `NOT NULL, CHECK`
- Trigger intervals:
  - `.trigger(processingTime="2 minutes")`: Query will be executed in micro-batches and kicked-off at the defined intervals
  - `.trigger(once=True)`: Query will be executed in single micro-batch to process all the available data and then stops
  - `.trigger(availableNow=True)`: Query will execute multiple micro-batches to process all the available data and then stops
- `CREATE TABLE AS SELECT` statements derive schema details from the query and you must not specify a column_specification.
- `MINUS` or `EXCEPT` is used to get all rows in the first `SELECT` statement that are not returned by the second `SELECT` statement
- Job cluster cannot be created using the UI, CLI or REST API
- Output modes available in Spark Structured Streaming:
  - `append`: default, only the new rows added to the resulting table since the last trigger will be outputted to the sink. This is supported for only those queries where rows added to the result table is never going to change
  - `complete`: whole resulting table will be outputted to the sink after every trigger. This is supported for aggregation queries
  - `update`: only the rows in the resulting table that were updated since the last trigger will be outputted to the sink
- View:
  - Persistente Speicherung im Metastore → bleibt auch nach Beenden des Clusters bestehen
  - Scoped for multiple sessions but they are not accessible after detaching and reattaching a DB notebook to a cluster
  - virtual table that has no physical data, just a saved SQL query against actual tables
- Temporary Views:
  - Existiert nur in der aktuellen Sitzung → wird nicht im Metastore gespeichert
  - Better suited for real-time, ad-hoc analysis tasks where the data does not need to be stored persistently
  - Tied to a Spark session and not visible to other sessions
- Global Temporary Views:
  - Ist über mehrere Sitzungen sichtbar, bleibt aber nur solange bestehen wie der Cluster aktiv ist
  - Speicherort: `global_temp`-Datenbank
  - can be still accessed even if the notebook is detached and attached

  - Can be accessed in other sessions on the same cluster and are tied to a cluster temporary database called `global_temp`
  - Available only on the cluster it was created, when the cluster restarts global temporary view is automatically dropped
- Common Table Expression:
  - `WITH <name> AS (SELECT FROM ...)  SELECT FROM ...  <name>`
  - Useful for complex subqueries