Data Governance

© Rangel

Data Management with Delta Lake

- 1. Identify where Delta Lake provides ACID transactions. 1
 - Transactions are at the table level, one table at a time
 - (Optimistic concurrency control)
 [https://en.wikipedia.org/wiki/Optimistic_concurrency_control] for concurrent transactions
 - BEGIN -> Modify -> Validate -> Commit/Rollback
 - Databricks has no BEGIN/END syntax like TSQL. Changes are made in a serial manner (1 at a time at a meaning neto)
- 2. Identify the benefits of ACID transactions. 1
 - 'Highest possible data reliability and integrity'
- 3. Identify whether a transaction is ACID-compliant.
 - Atomic each txn statement completes or fails ONLY
 - BEGIN/END statements and/or Stored procedures
 - Consistency data must be predictable before and after txn
 - i.e. row counts consistent when moving rows from one table to another
 - i.e. when moving money from one acc to another, total money must be same
 - Isolation no other process can change the data/table during a transaction
 - Durability changes from txn persist, even if servers die (hand in hand with Atomic)
- 4. Compare and contrast data and metadata.
 - metadata data about data. used for management, support, and context

```
# Describe statements for showing metadata
DESCRIBE SCHEMA EXTENDED ${schema_name};
DESCRIBE DETAIL <table-name>;
DESCRIBE TABLE EXTENDED <table-name>;
```

- Compare and contrast managed and external tables. 1
 - managed tables made within databricks via DDL

- external tables any tables with external data, regardless of where it is stored (dbfs, abfss, adls, s3).
- when dropping managed, data and metadata is lost. when dropping external, only metadata is lost.
- 6. Identify a scenario to use an external table. 1
 - when you need direct access to data outside of Databricks clusters/SQL warehouses (avoid data egress from external source)

7. Create a managed table.

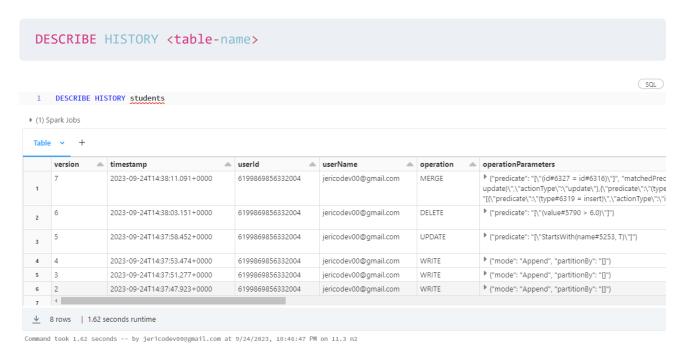
8. Identify the location of a table.

```
# Either command works
DESCRIBE EXTENDED <table-name>
DESCRIBE DETAIL <table-name>
```

- 9. Inspect the directory structure of Delta Lake files.
 - path contains / delta log/ and *.snappy.parquet files which form the delta table
 - delta log contains transactions in the form of *.crc and *.json files



- 10. Identify who has written previous versions of a table.
- 11. Review a history of table transactions.



- 12. Roll back a table to a previous version.
- 13. Identify that a table can be rolled back to a previous version.
- 14. Query a specific version of a table.

```
# Query what previous version looks like (time travel)
SELECT * FROM students VERSION AS OF 3;
```

```
# Rollback
RESTORE TABLE students TO VERSION AS OF 8
```

- 15. Identify why Zordering is beneficial to Delta Lake tables.
 - z-ordering = indexing

```
OPTIMIZE students
ZORDER BY id
```

- 16. Identify how vacuum commits deletes.
 - VACUUM deletes old versions of a table (the snappy parquet files)
 - does not delete the delta log, so we can still see the history of the table via DESCRIBE
 HISTORY

```
# By default you cannot delete table versions that are less than 7 days old, we
change this for demonstration
SET spark.databricks.delta.retentionDurationCheck.enabled = false;
SET spark.databricks.delta.vacuum.logging.enabled = true;

# DRY RUN first to see which files will be deleted (`*.snappy.parquet files`)
VACUUM students RETAIN 0 HOURS DRY RUN
```

```
VACUUM students RETAIN 0 HOURS
```

- 17. Identify the kind of files Optimize compacts. 1
 - small files are compacted and balanced out (combined towards an optimal size, determined by table size)
 - idempotent process
- 18. Identify CTAS as a solution.
- Create a generated column.
- Add a table comment.
- Use CREATE OR REPLACE TABLE and INSERT OVERWRITE
- Compare and contrast CREATE OR REPLACE TABLE and INSERT OVERWRITE
- Identify a scenario in which MERGE should be used.
- Identify MERGE as a command to deduplicate data upon writing.

- Describe the benefits of the MERGE command.
- Identify why a COPY INTO statement is not duplicating data in the target table.
- Identify a scenario in which COPY INTO should be used.
- Use COPY INTO to insert data.

Data Access with Unity Catalog

- Identify one of the four areas of data governance.
- Compare and contrast metastores and catalogs.
- Identify Unity Catalog securables.
- Define a service principal.
- Identify the cluster security modes compatible with Unity Catalog.
- Create a UC-enabled all-purpose cluster.
- Create a DBSQL warehouse.
- Identify how to query a three-layer namespace.
- Implement data object access control
- Identify colocating metastores with a workspace as best practice.
- Identify using service principals for connections as best practice.
- Identify the segregation of business units across catalog as best practice