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 ata 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>;
```

5. 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)

```
# Sample syntax
CREATE TABLE <catalog>.<schema>.<table-name>
(
  <column-specification>
)
LOCATION 's3://<bucket-path>/<table-directory>';
```

7. Create a managed table.

```
CREATE TABLE <catalog-name>.<schema-name>.<table-name>
(
  <column-specification>
);
```

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

```
Python display(dbutils.fs.ls(f"{path}/table"))
```

```
%python
display(spark.sql(f"SELECT * FROM json.`{DA.paths.user_db}/students/_delta_log/00000000000000000007.json`"))
```

(2) Spark Jobs

Table +

	add	commitInfo
1	{"dataChange": true, "modificationTime": 1695566290429, "path": "part-00000-20716378-fc1d-4822-aacd-7f0e10b83d12-c000.snappy.parquet", "size": 1063, "stats": "{\"numRecords\":1,\"minValues\":{\"id\":2,\"name\":\"Omar\",\"value\":15.2},\"maxValues\":{\"id\":2,\"name\":\"Omar\",\"value\":15.2},\"nullCount\":{\"id\":0,\"name\":0,\"value\":0}}", "tags": {"INSERTION_TIME": "1695566290429000", "MAX_INSERTION_TIME": "1695566290429000", "MIN_INSERTION_TIME": "1695566267608000", "OPTIMIZE_TARGET_SIZE": "268435456"}}	null
2	{"dataChange": true, "modificationTime": 1695566290446, "path": "part-00002-7180bbaa-23a0-4244-8755-065662c0a1ce-c000.snappy.parquet", "size": 1063, "stats": "{\"numRecords\":1,\"minValues\":{\"id\":7,\"name\":\"Blue\",\"value\":7.7},\"maxValues\":{\"id\":7,\"name\":\"Blue\",\"value\":7.7},\"nullCount\":{\"id\":0,\"name\":0,\"value\":0}}", "tags": {"INSERTION_TIME": "1695566290429001", "MAX_INSERTION_TIME": "1695566290429001", "MIN_INSERTION_TIME": "1695566267608000", "OPTIMIZE_TARGET_SIZE": "268435456"}}	null
3	null	{"clusterId": "0913-021744-jb87gxwo", "engineInfo": "Databricks-Runtime, "isolationLevel": "WriteSerializable", "notebook": {"notebookId": "104757795 "operationMetrics": {"executionTimeMs": "5402", "numOutputRows": "2", "nu "0", "numTargetFilesAdded": "2", "numTargetFilesRemoved": "2", "numTarget "numTargetRowsInserted": "1", "numTargetRowsUpdated": "1", "rewriteTime "operationParameters": {"matchedPredicates": "[{\"predicate\":\"(type#6319 {\"predicate\":\"(type#6319 = delete)\",\"actionType\":\"delete\"}]", "notMat insert)\",\"actionType\":\"insert\"}]", "predicate": "[\"(id#6327 = id#6316)\"]" "txnId": "ef92fb1a-d905-4123-8c76-9443a67d325c", "userId": "61998698563

5 rows | 0.61 seconds runtime — Refreshed 1 minute ago

Command took 0.61 seconds -- by jericodev00@gmail.com at 9/24/2023, 10:43:11 PM on 11.3 n2

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

```
DESCRIBE HISTORY <table-name>
```

```
DESCRIBE HISTORY students
```

(1) Spark Jobs

Table +

	version	timestamp	userId	userName	operation	operationParameters
1	7	2023-09-24T14:38:11.091+0000	6199869856332004	jericodev00@gmail.com	MERGE	{"predicate": "[\"(id#6327 = id#6316)\"]", "matchedPred update)\",\"actionType\":\"update\"},{\"predicate\":\"(type "[{\"predicate\":\"(type#6319 = insert)\",\"actionType\":\"i
2	6	2023-09-24T14:38:03.151+0000	6199869856332004	jericodev00@gmail.com	DELETE	{"predicate": "[\"(value#5790 > 6.0)\"]"}
3	5	2023-09-24T14:37:58.452+0000	6199869856332004	jericodev00@gmail.com	UPDATE	{"predicate": "[\"StartsWith(name#5253, T)\"]"}
4	4	2023-09-24T14:37:53.474+0000	6199869856332004	jericodev00@gmail.com	WRITE	{"mode": "Append", "partitionBy": "[]"}
5	3	2023-09-24T14:37:51.277+0000	6199869856332004	jericodev00@gmail.com	WRITE	{"mode": "Append", "partitionBy": "[]"}
6	2	2023-09-24T14:37:47.923+0000	6199869856332004	jericodev00@gmail.com	WRITE	{"mode": "Append", "partitionBy": "[]"}
7						

8 rows | 1.62 seconds runtime

Command took 1.62 seconds -- by jericodev00@gmail.com at 9/24/2023, 10:46:47 PM on 11.3 n2

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