DELTA LAKE :-

- Technology at the heart of Databricks Lakehouse platform.
- Delta Lake **enforces schema on write.**
- While Spark is the primary processing technology in the Databricks Runtime, the Lakehouse really depends on leveraging the capabilities of Delta Lake.

Delta Lake is an open-source project that enables building a Data Lakehouse on top of existing storage systems.

- Data Lakehouse is the storage medium / format.
- Delta Lake helps create Data Lakehouse.

- Delta Lake is not :-
  - Proprietary technology,
  - Storage format or medium.
  - Database service or data warehouse.

- Delta Lake is :-
  - (i) Open-source framework.
  - (ii) Builds upon standard data formats.
  - (iii) Optimised for cloud object storage.

- Delta Lake is generally powered by data, stored in Parquet format.

- With Delta Lake, all applications and workloads of an organization can work with single central copy of data.

- Delta Lake applies ACID compliancy to object storage, at the scope level of table.

- Delta Lake allows atomic micro-batch transaction processing in near realtime for tight integration with Spark Structured Streaming, thus, we can apply both Streaming & Batch processing on same set of Delta Lake Tables.

Delta Lake is the default for all tables created in Databricks.

Delta Lake is the default format for all tables created with Databricks.

Thus, all tables in databases created on Databricks using SQL, is of Delta Lake format.

① Delta Lake Table can be created by selecting SQL as notebook/shell language, and the creating a table using CREATE TABLE :-

```
CREATE TABLE test (id INT, name STRING);
```

This will create an empty Delta Lake Table, named test

Data insertion in this table is just like as done in normal SQL INSERT statement.

≥ Delta Read guarantees that any read against a table will always return the most recent version.

≥ ADVANCED DELTA LAKE FEATURES :-

≥ All delta lake operations can be run on Databricks using SQL.

≥ Examine Table Details :-

≥ Databricks uses Hive Metastore by default, to register databases, tables, & views.

To see important metadata of a stable, use the following command :-

```
DESCRIBE EXTENDED <table_name>
```

We can also use following command :-
(shows additional details, such as no. of files)

```
DESCRIBE DETAIL <table_name>
```

We get details like column names & their data types, table type (managed or unmanaged), location.

Location : It is the place where the data files of the delta live tables are stored.
A Delta Lake Table is actually backed by a collection of files, stored in cloud object storage.

We can check the files in that location, using the dbutils.fs.ls("<location>") command.

- Records of/in Delta Lake Tables are stored as data in Parquet files.
- Transaction to Delta Lake Tables are recorded in the _delta_log directory.

Each transaction results in a new JSON file being written to the Delta Lake Transaction log, i.e, inside the _delta_log folder.

≥ The files that corresponds to earlier version of table are also kept. Just that they are tracked, and marked as inactive (the file does not contain data corresponding to the current version.)

≥ Using the inactive data files, we can query past versions of the table.

When Delta Lake Tables are queried, data from only the active files are fetched.

⊛ Compacting Small Files & Indexing

≥ Small files that occur due to small changes can be combined to an optimal size, using the OPTMISE command :-

```
OPTMIZE <table_name>
ZORDER BY <id>
```

↑ column name(s) in table

we can provide here multiple columns

ZORDER performs indexing, during combining files, that quickens data retrieval.

⇒ OPTIMISE replace existing data files by rewriting the results.

⇒ Viewing Delta Lake Transaction History :-

⇒ [DESCRIBE HISTORY <table_name>

- Shows whole transaction history.
With each new transaction, the table gains a new version.
This each version of the table is recorded with, and show with a different / unique Version & timestamp in the transaction history table.
Both these helps to perform Time-Travel Queries (queries that fetch data as on any past version).

[SELECT * FROM <table> VERSION AS OF 3

past version to be queried.

⇒ By this query, current state does not gets affected.

TIME TRAVEL

⇒ Rollbacking Commit / Transaction :-

⇒ Suppose, we accidentally dropped entire table, or made such mistake. We can undo this action by following command :-

```
RESTORE TABLE <table_name> TO VERSION AS OF 4
```

This is recorded as a transaction.

⇒ Cleaning Stale Files :-

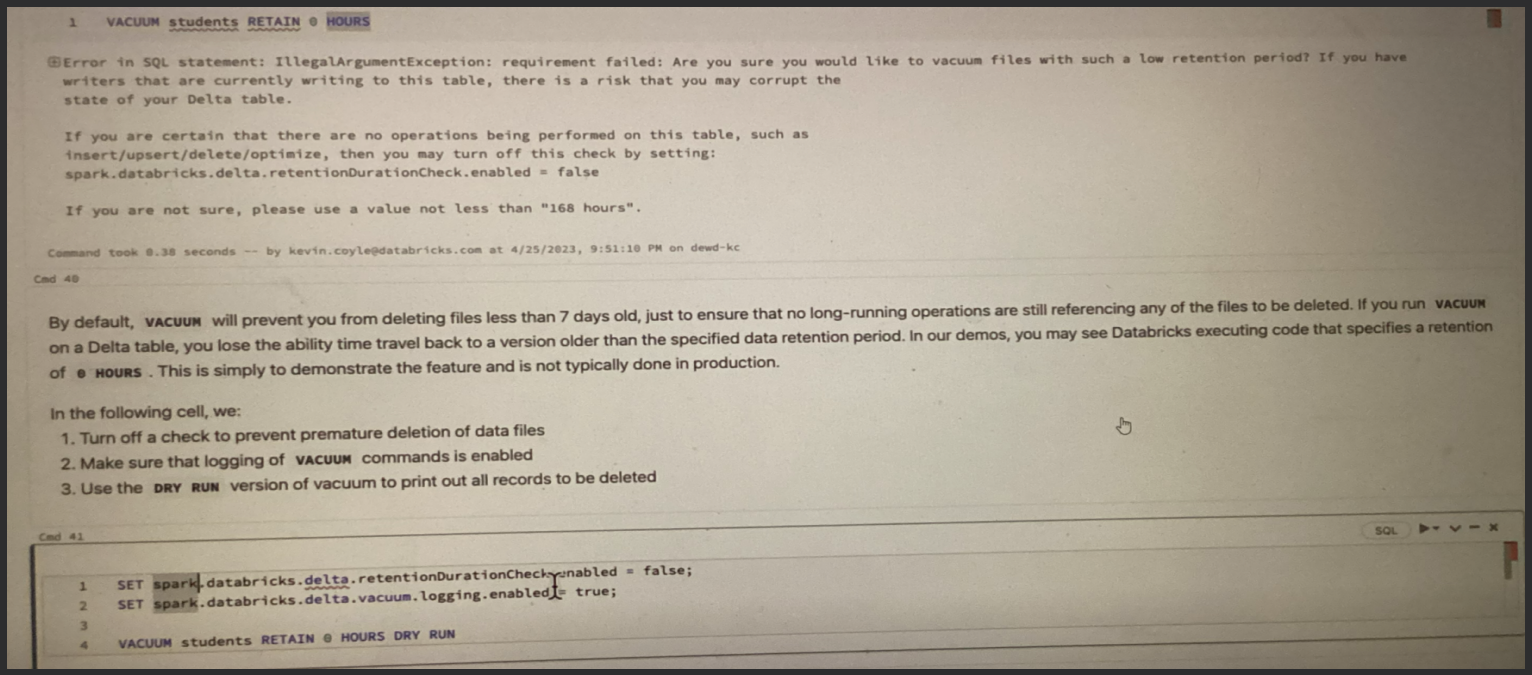
⇒ Very old version files are automatically deleted by databricks.
To delete them manually, we use vacuum command.

Ex :-

```
VACUUM students RETAIN 170 HOURS
```

keeps all the versions of last 170 hours

By default, we cannot vacuum versions, not older than 168 hours.
For this to be overridden we require some config. variables to be changed.



```
1  VACUUM students RETAIN 0 HOURS
```

```
Error in SQL statement: IllegalArgumentException: requirement failed: Are you sure you would like to vacuum files with such a low retention period? If you have
writers that are currently writing to this table, there is a risk that you may corrupt the
state of your Delta table.

If you are certain that there are no operations being performed on this table, such as
insert/upsert/delete/optimize, then you may turn off this check by setting:
spark.databricks.delta.retentionDurationCheck.enabled = false

If you are not sure, please use a value not less than "168 hours".
```

```
Command took 0.38 seconds -- by kevin.coyle@databricks.com at 4/25/2023, 9:51:10 PM on dewd-kc
```

Cmd 40

By default, `VACUUM` will prevent you from deleting files less than 7 days old, just to ensure that no long-running operations are still referencing any of the files to be deleted. If you run `VACUUM` on a Delta table, you lose the ability time travel back to a version older than the specified data retention period. In our demos, you may see Databricks executing code that specifies a retention of `0 HOURS`. This is simply to demonstrate the feature and is not typically done in production.

In the following cell, we:
1. Turn off a check to prevent premature deletion of data files
2. Make sure that logging of `VACUUM` commands is enabled
3. Use the `DRY RUN` version of vacuum to print out all records to be deleted

Cmd 41

SQL

```
1  SET spark.databricks.delta.retentionDurationCheck.enabled = false;
2  SET spark.databricks.delta.vacuum.logging.enabled = true;
3
4  VACUUM students RETAIN 0 HOURS DRY RUN
```