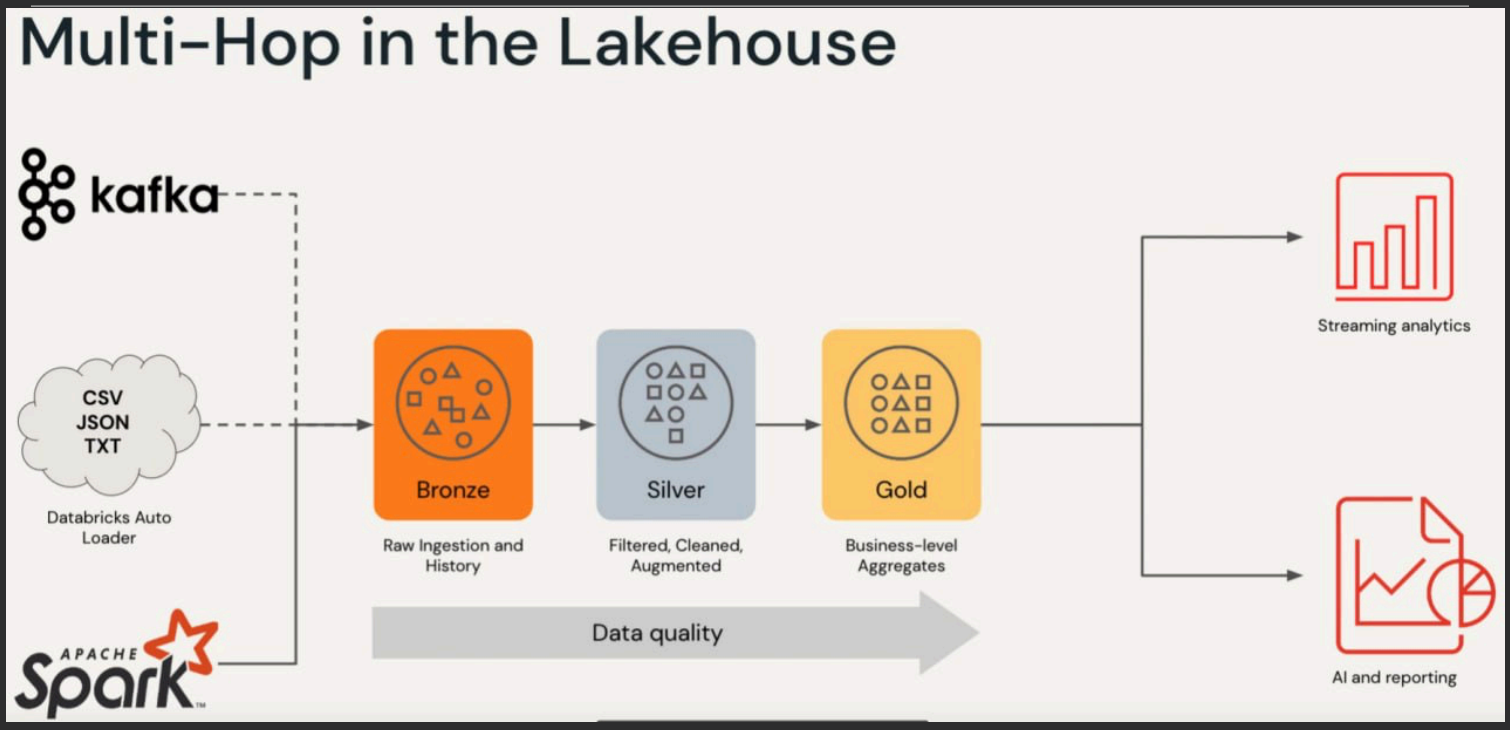
Delta Live Tables :



⇒ Delta Live Tables is designed to allow users to easily declare batch and/or streaming pipelines using Python and SQL

⇒ DLT has built-in built-in declarative quality controls and data quality monitoring.

What is a Live Table?

Live Tables are materialized views for the lakehouse.

A live table is:

- Defined by a SQL query
- Created and kept up-to-date by a pipeline (called DLT pipeline)

must be replaced by LIVE keyword.

```
CREATE OR REPLACE TABLE report
AS SELECT sum(profit)
FROM prod.sales
```

Live tables provides tools to:

- Manage dependencies
- Control quality
- Automate operations
- Simplify collaboration
- Save costs
- Reduce latency

⇒ In Delta Lake, tables that are managed by DLT pipeline must be created using LIVE keyword.

⇒ In general LIVE TABLES, old data can be processed multiple times. (does not means appended multiple times, only old data is processed multiple times).

What is a Streaming Live Table?

Based on Spark™ Structured Streaming

A streaming live table is "stateful":

- Ensures exactly-once processing of input rows
- Inputs are only read once

[Just like Auto Loader does to files.]

```
CREATE STREAMING LIVE TABLE report
AS SELECT sum(profit)
FROM cloud_files(prod.sales)
```

- Streaming Live tables compute results over append-only streams such as Kafka, Kinesis, or Auto Loader (files on cloud storage)
- Streaming live tables allow you to reduce costs and latency by avoiding reprocessing of old data.

As it is stateful, thus it tracks till which record processing is complete, and thus when new data comes/appended only that data is recognised, and only processed, instead of processing entire data.
This is meant by the term STATEFULL.

Creating Your First Live Table Pipeline

SQL to DLT in three easy steps...

Write create live table

- Table definitions are written (but not run) in notebooks
- Databricks Repos allow you to version control your table definitions.

```
1 CREATE LIVE TABLE daily_stats
2 AS SELECT sum(rev) - sum(costs) AS profits
3 FROM prod_data.transactions
4 GROUP BY day
```

Create a pipeline

- A Pipeline picks one or more notebooks of table definitions, as well as any configuration required.

Delta Live Tables

Click start

- DLT will create or update all the tables in the pipelines.

⇒ DLT are only declared in notebooks (and never run interactively), and are only created & updated

by workflow pipelines.

Main difference

Just, as we can create, maintain/update changes to normal delta tables, both interactively (using notebooks) or by Workflow Jobs, Delta Live Tables are created, populated & maintained only by Workflow Jobs (i.e, pipelines)

DLT Pipeline :-

A simple pipeline; created in the Workflows tab → Delta Live Tables sub-tab.
There, we click on 'Create Pipeline' to create a new DLT pipeline and add notebooks.

Dependency Tables :-

Suppose, we are populating data in table B using/from table A. Then, A becomes the dependency table of B.

⇒ LIVE virtual schema :-

⇒ All the DLTs exists, and can be accessed using the schema named LIVE.
This is a virtual schema, that only exists when DLT is running, and contains only those LIVE Tables, that are a part of the same pipeline.

```
CREATE LIVE TABLE events AS
SELECT * FROM <ABC>.

CREATE LIVE TABL reports
AS SELECT * FROM LIVE.events;
```

accessing the live table using LIVE. schema.

⇒ LIVE schema helps to access the DLTs declared in other notebooks, in the same pipeline.
It can thus be thought off as a common namespace for all Live Tables in all notebooks, of a single pipeline.

Special feature:-

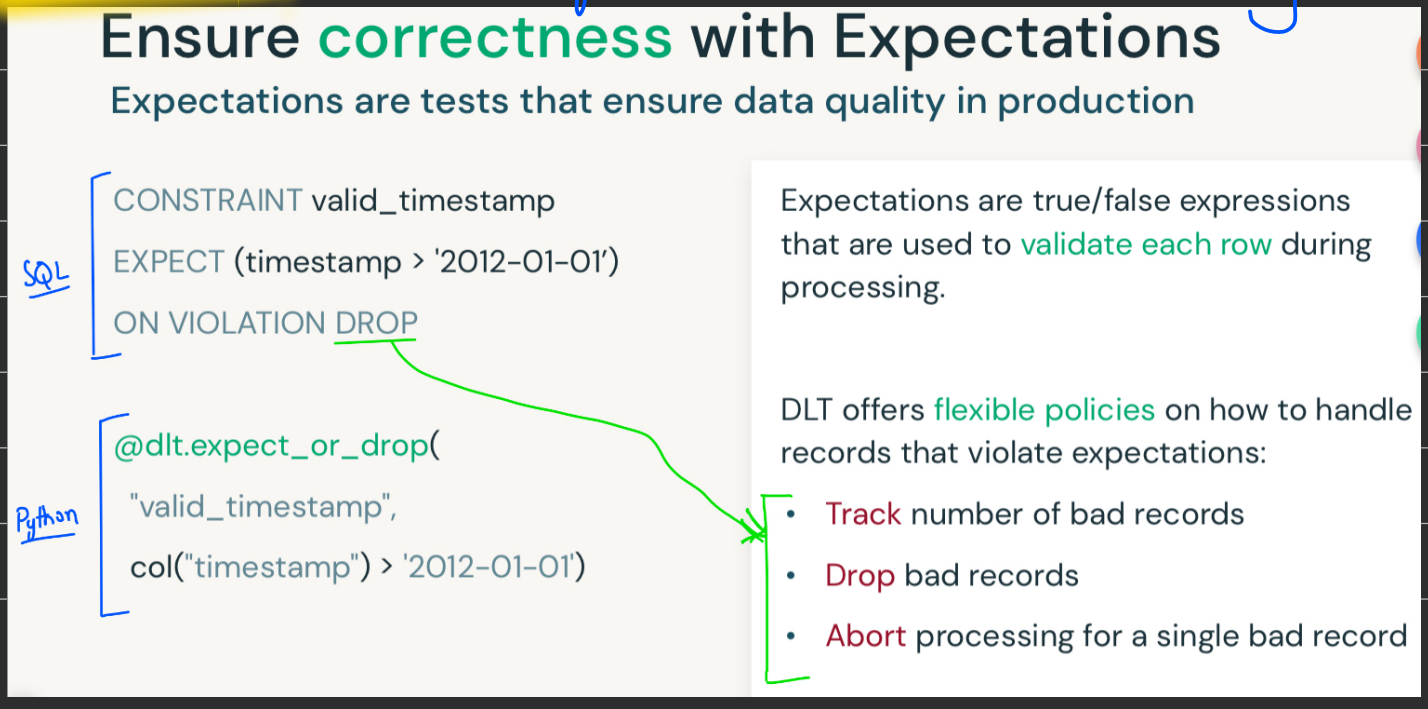
⇒ This LIVE schema is replaced by the schema name that is set during the DLT pipline creation, in the **Target Schema** option.

This makes it easy while moving DLT code from dev to production, as:-

After coding it in dev, when we will move the notebook to Production env., we just change the schema in the pipeline config., and all the tables will be created, populated and existent in the new production env.

Data Quality Control in DLT:-

⇒ Using Expectations (by EXPECT keyword), row-level validations can be performed on incoming data.



Populating Data in a Streaming Live Table.

(*) See page 09 of this note

Using Spark™ Structured Streaming for ingestion

Easily ingest files from cloud storage as they are uploaded

(Usefull for incremental data ingestion from Files of any format)

```
CREATE STREAMING LIVE TABLE raw_data
AS SELECT *
FROM cloud_files("/data", "json")
```

To read data from files

This example creates a table with all the json data stored in "/data":

- `cloud_files` keeps track of which files have been read to avoid duplication and wasted work
- Supports both listing and notifications for arbitrary scale
- Configurable schema inference and schema evolution

Using the SQL STREAM() function

Stream data from any Delta table

(Usefull for incremental data ingestion from a table acting as source.)

```
CREATE STREAMING LIVE TABLE mystream
  AS SELECT *
  FROM STREAM(my_table)
```

→ an SCD Type 0 table

Pitfall: `my_table` must be an append-only source.

e.g. it may not:

- be the target of APPLY CHANGES INTO
- define an aggregate function
- be a table on which you've executed DML to delete/update a row (see GDPR section)

i.e, 'my_table' must be an SCD Type 0 table.

- `STREAM(my_table)` reads a stream of new records, instead of a snapshot
- `Streaming` tables must be an append-only table
- Any append-only delta table can be read as a stream (i.e. from the live schema, from the catalog, or just from a path). Cannot use a table as a SOURCE that is being overwritten or updatable. Such as tables created/populated using MERGE, INSERT OVERWRITE cannot be used as its source.

To read data from tables (append-only)

Main Advantages of DLT:-

- The things that we need to manually do with normal delta tables for optimization and performance, are automatically done for Delta Live Tables.

Automated Data Management

DLT automatically optimizes data for performance & ease-of-use

Best Practices

What:

DLT encodes Delta best practices automatically when creating DLT tables.

How:

DLT sets the following properties:

- `optimizeWrite`
- `autoCompact`
- `tuneFileSizesForRewrites`

Physical Data

What:

DLT automatically manages your physical data to minimize cost and optimize performance.

How:

- runs vacuum daily
- runs optimize daily

You still can tell us how you want it organized (ie ZORDER)

Schema Evolution

What:

Schema evolution is handled for you

How:

Modifying a live table transformation to add/remove/rename a column will automatically do the right thing.

When removing a column in a streaming live table, old values are preserved.

- DLT pipeline has the ability to send email notifications upon Success, Failure, etc.
(This can be enabled and configured by providing the email addresses in which we want the notifications, and for which actions we want.
Done during Pipeline creation.)

Development	Production
⇒ After pipeline running is complete, keeps the dependencies and requirements cached on cluster, and also cluster remains active.	⇒ As soon as the pipeline running is complete, the cluster is turned off, and all the resources are released.
This is useful, because, rerunning of pipeline becomes faster, while testing to see whether changes took effect.	This is to save cost, as Production ready pipeline is fully tested, and do not need to be rerun.
⇒ Similar to All-Purpose Cluster	⇒ Similar to Job Cluster condition.

⇒ Important Points :-

- While pipeline creation, we just add the notebooks that are to be run in DLT pipeline. The sequence of running / execution of notebooks (in case of multiple notebooks being added to single pipeline)

is automatically interpreted by the DLT pipeline, based on dependency lineage.

After automatically analysing the execution order of notebooks, during execution, there gets created a DAG of the entire data flow through tables.
DAG shows there occurs parallel as well as sequential execution, as required.

Each node in the DAG is a Delta Live Table.
Upon clicking on a node, we can see entire details of the table, on the RHS, about its type (General or Streaming), Data Quality, Schema, column names, execution time & duration, etc.

Streaming Ingestion with Auto Loader

Databricks has developed the Auto Loader functionality to provide optimized execution for incrementally loading data from cloud object storage into Delta Lake. Using Auto Loader with DLT is simple: just configure a source data directory, provide a few configuration settings, and write a query against your source data. Auto Loader will automatically detect new data files as they land in the source cloud object storage location, incrementally processing new records without the need to perform expensive scans and recomputing results for infinitely growing datasets.

The `cloud_files()` method enables Auto Loader to be used natively with SQL. This method takes the following positional parameters:

- The source location, which should be cloud-based object storage
- The source data format, which is JSON in this case
- An arbitrarily sized comma-separated list of optional reader options. In this case, we set `cloudFiles.inferColumnTypes` to `true`

In the query below, in addition to the fields contained in the source, Spark SQL functions for the `current_timestamp()` and `input_file_name()` as used to capture information about when the record was ingested and the specific file source for each record.

Cmd 6

```
1  CREATE OR REFRESH STREAMING LIVE TABLE orders_bronze
2  AS SELECT current_timestamp() processing_time, input_file_name() source_file, *
3  FROM cloud_files("${source}/orders", "json", map("cloudFiles.inferColumnTypes", "true"))
```

2. If we need to perform **Aggregations**, **general LIVE tables** are preferred, as each time, they process the **entire source data**, rather than only the updates.