Disclaimer: **These slides are copyrighted and strictly for personal use only**

- This document is reserved for people enrolled into the [Ultimate Big Data Masters Program (Cloud Focused) by Sumit Sir]

- **Please do not share this document,** it is intended for personal use and exam preparation only, thank you.

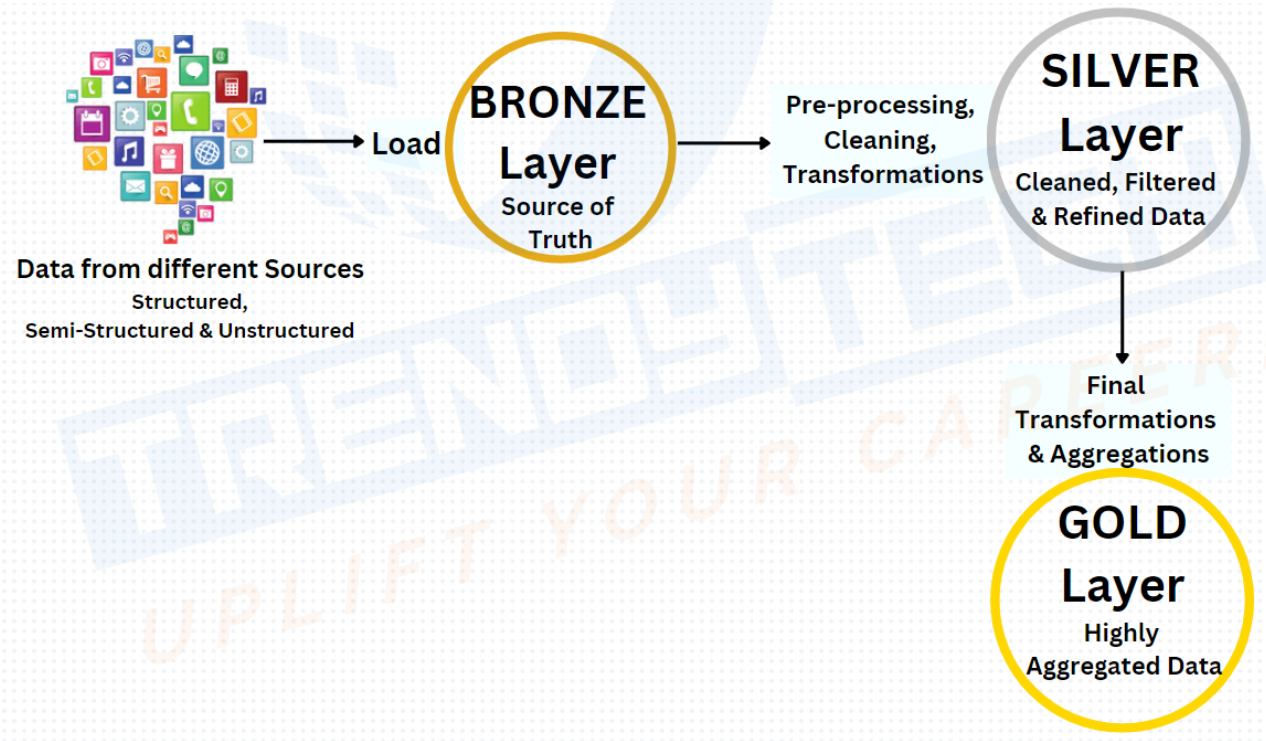
- If you've obtained these slides for free on a website that is not the course's website, please reach out to legal@trendytech.in . Thanks!

- All the Best and Happy Learning!

Delta architecture is an approach that can handle different use-cases.

There are 3 stages to Delta Architecture - Bronze, Silver & Gold. Data rolls up from Bronze to Gold layer with enhancements.

- Raw Data from various sources is dumped to the **Bronze Layer**
- Preprocessed/ Cleaned data with data enhancements (by applying transformations) is added to the **Silver Layer**
- Finally, further transformations and aggregations are applied and the resultant data is added to the **Gold Layer** also known as optimized query layer to serve BI / Reporting activities. Consists of business specific data that is highly aggregated.
- There could be another layer named **Platinum Layer** consisting of high quality aggregated data. The number of stages may increase or decrease based on the business use-case.
- As the data moves up to the higher layers, the data quality also improves at every stage.
- Delta Architecture is also known as **Medallion Architecture**



- It is tricky to merge incremental changes in a Medallion Architecture.

Change Data Feed

- Capturing the Inserts, Updates, Deletes and Merging the incremental changes is difficult in case of medallion architecture. The Change Data Feed feature helps in tracking the changes made in the data.
- The Change Data Feed feature will be helpful in case of auditing activities.
- The Change Data Feed feature is disabled by default. In order to enable it, the following property needs to be enabled,
    1. While creating the table

        TBLPROPERTIES (delta.enableChangeDataFeed = true)

Example:

```
create table orders(
order_id int,
order_date string,
customer_id int,
order_status string
)
using delta
TBLPROPERTIES (delta.enableChangeDataFeed = true)
```

    2. For existing table with Change Data Feed disabled

        alter table <table-name> set TBLPROPERTIES (delta.enableChangeDataFeed = true)

    3. To enable the change data feed by default for all the tables that will be created subsequently

        set spark.databricks.delta.properties.defaults.enableChangeDataFeed = true

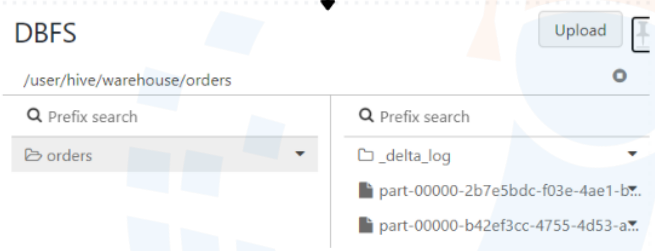
Insert

```
%sql
create table orders(
  order_id int,
  order_date string,
  customer_id int,
  order_status string
)
using delta
TBLPROPERTIES (delta.enableChangeDataFeed = true)
```

Creating orders table with Change Data Feed enabled

```
%sql
insert into orders values (1, '2013-07-25 00:00:00.0', 11599, 'CLOSED'), (2, '2013-07-25 00:00:00.0', 256, 'PENDING_PAYMENT'), (3, '2013-07-25 00:00:00.0', 12111, 'COMPLETE'), (4, '2013-07-25 00:00:00.0', 8827, 'CLOSED'), (5, '2013-07-25 00:00:00.0', 11318, 'COMPLETE')
```

Insert operation



No New folder created in DBFS

To view the changes made to the data

```
1  %sql
2  select * from table_changes('orders', 1)
```

Table-Name

Version-Number

(2) Spark Jobs

_sqldf: pyspark.sql.dataframe.DataFrame = [order_id: integer, order_date: string ... 5 more fields]

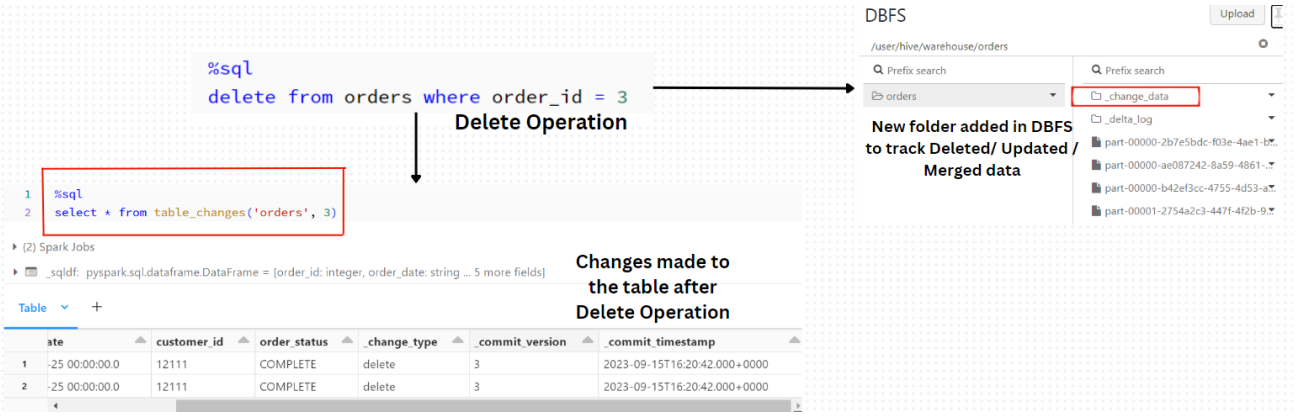
Table +

		customer_id	order_status	_change_type	_commit_version	_commit_timestamp
1	:00:00.0	11599	CLOSED	insert	1	2023-09-15T14:43:14.000+0000
2	:00:00.0	256	PENDING_PAYMENT	insert	1	2023-09-15T14:43:14.000+0000
3	:00:00.0	12111	COMPLETE	insert	1	2023-09-15T14:43:14.000+0000
4	:00:00.0	8827	CLOSED	insert	1	2023-09-15T14:43:14.000+0000
5	:00:00.0	11318	COMPLETE	insert	1	2023-09-15T14:43:14.000+0000
6	:00:00.0	11599	CLOSED	insert	2	2023-09-15T16:06:31.000+0000
7	:00:00.0	256	PENDING_PAYMENT	insert	2	2023-09-15T16:06:31.000+0000

10 rows | 2.25 seconds runtime

Refreshed now

Update and Delete



Note : The **_change_data** will be created only on updates / deletes / merges and not for insert operation.

Creating Bronze, Silver & Gold tables

#Creating a database

```
%sql
create database retaildb
```

#Creating a Bronze Table

```
%sql
create table retaildb.orders_bronze
(
order_id int,
order_date string,
customer_id int,
order_status string,
filename string,
createdon timestamp
)
using delta
location “dbfs://FileStore/data/orders_bronze.delta”
partition by(order_status)
TBLPROPERTIES(delta.enableChangeDataFeed = true)
```

#Creating a Silver Table (with new / updated / merged columns)

```
%sql
create table retaildb.orders_silver
(
order_id int,
order_date string,
customer_id int,
order_status string,
order_year int GENERATED ALWAYS AS (YEAR(order_date)),
order_month int GENERATED ALWAYS AS (MONTH(order_date)),
createdon timestamp,
modifiedon timestamp
)
using delta
location “dbfs://FileStore/data/orders_silver.delta”
partition by(order_status)
TBLPROPERTIES(delta.enableChangeDataFeed = true)
```

#Creating a Gold Table (Highly Aggregated and Refined table generated based on the business requirement)

Suppose the requirement is to find the number of orders in the different statuses for each customer in a particular year

```
%sql
create table retaildb.orders_gold
(
customer_id int,
order_status string,
order_year int,
Num_orders int
)
using delta
location “dbfs://FileStore/data/orders_gold.delta”
TBLPROPERTIES(delta.enableChangeDataFeed = true)
```

Data Movement through the different Medallion Layers

1. **Firstly, Insert data into the Raw Folder**

2. **Copy the data to the Bronze Table from the Raw Folder (All data should always go through the bronze layer)**

```
%sql
copy into retaildb.orders_bronze from (
select order_id::int,
order_date::string,
customer_id::int,
order_status::string,
INPUT_FILE_NAME() as filename,
CURRENT_TIMESTAMP() as createdon
FROM 'dbfs:/FileStore/raw'
)
fileformat = CSV
format_options('header' = 'true')
```

-**Copy command tracks the records already loaded and doesn't add the existing records on re-execution.**

3. **Changes in the Bronze Table has to be merged to the Silver Table**
   -Some cleaning and data validation checks are performed on the Bronze table in order to get high quality data to the Silver Layer.

Example - Changes made in the bronze table from version 1 onwards to be merged to the silver table.

```
%sql
create or replace temporary view orders_bronze_changes
as
select * from table_changes('retaildb.orders_bronze', 1) where order_id>0
customer_id>0 and order_status in ('PAYMENT_REVIEW', 'PROCESSING',
'CLOSED', 'SUSPECTED_FRAUD', 'COMPLETE', 'PENDING',
'CANCELLED', 'PENDING_PAYMENT')
```

```
#merging to the Silver Table
%sql
merge into retaildb.order_silver tgt
using orders_bronze_changes src on tgt.order_id = src.order_id
when matched
then
update set tgt.order_status = src.order_status, tgt.customer_id =
src.customer_id, tgt.modifiedon = CURRENT_TIMESTAMP()
when not matched
then
insert (order_id, order_date, customer_id, order_status, createdon,
modifiedon) values(order_id, order_date, customer_id, order_status,
CURRENT_TIMESTAMP(), CURRENT_TIMESTAMP())
```

4. **Adding the data to the Gold Table from Silver Table using Overwrite**

```
%sql
insert overwrite table retaildb.orders_gold
select customer_id, order_status, order_year, count(order_id) as num_orders
from retaildb.orders_silver group by customer_id, order_status, order_year
```

Key Points :

- **Data quality improves as the data moves from Bronze to Gold Layer.**