DELTA CHANGE DATA FEED:-

⇒ CDC (Change Data Capture) → technique to identify and capture changes made to data in a source system. Normal CDC using MERGE, (or any oth) does not retains the history of changed data.

⇒ CDF (Change Data Feed):-
It is a new feature built on Delta Lake on Databricks that allows us to automatically generate CDC feeds about Delta Lake Tables.

⇒ Delta Tables are said to be Composable, if new streams can be initiated from them, i.e, the Tables can in turn act as source of streaming data for next steps.
This is only when Delta Tables are append-only
When existing data change, it breaks stream / delta table composability.

Operations that break stream composability

- Complete aggregations
- Delete
- UPDATE/MERGE

⇒ We can ignore changes to records by setting ignoreChanges to True.

⊜ But, parquet files contains many rows, & delta trans. logs tracks changes to files (instead of changes to records)
Thus, changing a record actually causes changes to the entire file, and delta lake will then point to new version of this file.
This will need deduplication implemented in all downstream sources, as the changed file will seem to be new file, and thus, the unchanged records of the changed file, will seem as data that is new (just arrived).
This is cumbersome way of CDC.

Alternate better approach is the Databricks Change Data Feed feature :-

Workarounds for Deleting Data

ignoreChanges	ignoreDeletes
Allows deletion of full partitions	Allows stream to be executed against Delta table with upstream changes
No new data files are written with full partition removal	Must implement logic to avoid processing duplicate records
Modified files are picked up as new files. These causes re-writing of unchanged records, along with changed ones.	Subsumes ignoreDeletes

⇒ Delta Change Data Feed (CDF) causes lesser data to be replicated (un-changed data) & touched, when processing updates.
Thus, reduces the time to make changes.

⇒ Batch & Streaming Data uses same format to maintain change logs, when CDF is used.
This enables unified downstream processing, regardless of batch or streaming writes.

⇒ Safely propagates updated & changed data.

⇒ If tables are replicated in external systems, and source bin tables changes, changes can be done to the external tables from the output of CDF, rather than building the whole external table.

⇒ CDF, when used/enabled, keeps and provides Full history of the changes made to data, including deleted info.

⇒ Delta Change Data Feed (CDF) is an output from/of Delta Lake.

This means, CDF applies when processing from a Delta format, to further steps, in data flow.

⇒ To enable CDF :-

a) Bring data from external sources, to the Bronze layer / table.

b) Enable CDF on the Bronze layer (then CDF is applicable from this point forward.)

⇒ This allows to use CDF, when moving data to the silver or gold layer.

⇒ When using CDF, we subscribe to the feed of changes, rather than the table, itself, then apply logic to process the changed data downstream.

How Does Delta Change Data Feed Work?

Original Table (v1)

	PK	B
	A1	B1
	A2	B2
	A3	B3

+

Change data (Merged as v2)

	PK	B
	A2	Z2
	A3	B3
	A4	B4

→

Change Data Feed Output

PK	B	Change Type	Time	Version
A2	B2	Preimage	12:00:00	2
A2	Z2	Postimage	12:00:00	2
A3	B3	Delete	12:00:00	2
A4	B4	Insert	12:00:00	2

A1 record did not receive an update or delete. So it will not be output by CDF.

Tracks & changes only the modified record.

Preimage :- Records before change.

Postimage :- Records after change.

⇒ Helpful in case of Materialised (persistent) views, and Aggregations, as they appropriately process only the records that changed from preimage to post image.

Timestamps of the commits are also recorded.

⇒ Enabling CDF in Spark :-

⇒ Enabling CDF using Spark conf setting in a notebook or on a cluster ensures it is used on all newly created Delta tables in that scope.

SYNTAX

```
spark.conf.set("spark.databricks.delta.properties.defaults.enableChangeDataFeed", "True")
```

⇒ For existing tables, syntax to enable CDF :-

```
%sql
ALTER TABLE <tbl_name>
SET TBLPROPERTIES (delta.enableChangeDataFeed=true);
```

- This property will be visible when we apply DESCRIBE EXTENDED

- CDF data is not taken during DEEP CLONE.

Physical View of enabling CDF :-

- Whenever we enable CDF on a delta table, another folder is created inside the delta table named folder.

Folder is named as :- _change_data
This is at the same level as the _delta_log folder for each delta table.

- Inside the _change_data folder, there exists Parquet files.

Read the Change Data Feed

To pick up the recorded CDC data, we add two options:

- `readChangeData`
- `startingVersion` (can use `startingTimestamp` instead)

Here we'll do a streaming display of just those patients in LA. Note that users with changes have two records present.

Cmd 28

```
1  cdc_df = (spark.readStream
2                .format("delta")
3                .option("readChangeData", True)
4                .option("startingVersion", 0)
5                .table("silver"))
6
7  cdc_la_df = cdc_df.filter("city = 'Los Angeles'")
8
9  display(cdc_la_df, streamName = "display_la")
```

≥ Each record that is changed, will appear multiple times, when we read the CDF.

≥ table_changes() :-

≥ Only applicable to tables, having CDF enabled.

≥ Suppose a single record, or a set of records in a table has been updated / deleted. With each modification, the record / table gains a new version no.

As CDF is allowed, changes made in past, have its history recorded.
If we want to see set of records, in a form that they existed in the past (and not currently), then, we can do that as :-

Ex:-

```
%sql
SELECT *
FROM table_changes("silver", 0)
WHERE mrn = 14125426
ORDER BY _commit_version
```

table name

version no. to look for

This will show the records with mrn = 14125426, as the records had values in the 1st version of their form.

In the current version of table, may be these records do not exist (deleted), or have some other values.

⇒ If we know that a certain condition-meeting records are deleted, then we can fetch those records, and propagate the delete downstream.

Ex:-

```
DELETE FROM silver
WHERE marks < 50;

WITH deletes AS
(
  SELECT * FROM
  table_changes("silver", 0)
  WHERE marks < 50
)

MERGE INTO gold g
USING deletes d
ON g.roll = d.roll
WHEN MATCHED
  THEN DELETE.
```

Fetching deleted records using same condition in WHERE clause.

→ deleted records in silver are fetched from prev. version using table_changes() and those records matching in gold are deleted.

This is Deletes Propagation.