Last updated on ***Dec 30, 2025***

Use Delta Lake change data feed on Databricks

Change data feed allows Databricks to track row-level changes between versions of a Delta table. When enabled on a Delta table, the runtime records *change events* for all the data written into the table. This includes the row data along with metadata indicating whether the specified row was inserted, deleted, or updated.

You can use change data feed to power common data use cases including:

- **ETL pipelines**: Incrementally process only the rows that changed since the last pipeline run.
- **Audit trails**: Track data modifications for compliance and governance requirements.
- **Data replication**: Synchronize changes to downstream tables, caches, or external systems.

> **IMPORTANT**
> Change data feed works in tandem with table history to provide change information. Because cloning a Delta table creates a separate history, the change data feed on cloned tables doesn't match that of the original table.

Enable change data feed

Change data feed must be explicitly enabled on the tables that you want to read from. Use one of the following methods.

New table

Set the table property `delta.enableChangeDataFeed = true` in the `CREATE TABLE` co

SQL

```
CREATE TABLE student (id INT, name STRING, age INT)
  TBLPROPERTIES (delta.enableChangeDataFeed = true)
```

Existing table

Set the table property `delta.enableChangeDataFeed = true` in the `ALTER TABLE` command.

SQL

```
ALTER TABLE myDeltaTable
  SET TBLPROPERTIES (delta.enableChangeDataFeed = true)
```

All new tables in a session

Set a Spark configuration to enable change data feed for all new tables created in a session.

SQL

```
SET spark.databricks.delta.properties.defaults.enableChangeDataFeed = true;
```

> **IMPORTANT**
>
> Only changes made after you enable the change data feed are recorded. Past changes to a table are not captured.

Change data feed schema

When you read from the change data feed for a table, the schema for the latest table version is used. Databricks fully supports most schema change and evolution operations, but tables with column mapping enabled have limitations. See Change data feed limitations for tables with column mapping.

In addition to the data columns from the schema of the Delta table, change data feed contains metadata columns that identify the type of change event:

Column name	Type	Values
`_change_type`	String	`insert`, `update_preimage` , `update_postimage`, `delete` (1)
`_commit_version`	Long	The Delta log or table version containing the change.
`_commit_timestamp`	Timestamp	The timestamp associated when the commit was created.

(1) `preimage` is the value before the update, `postimage` is the value after the update.

You cannot enable change data feed on a table if the schema contains columns with the same names as these metadata columns. Rename columns in your table to resolve this conflict before enabling change data feed.

Incrementally process change data

Databricks recommends using change data feed in combination with Structured Streaming to incrementally process changes from Delta tables. You must use Structured Streaming for Databricks to automatically track versions for your table's change data feed. For CDC processing with SCD type 1 or type 2 tables, see The AUTO CDC APIs: Simplify change data capture with pipelines.

Set the option `readChangeFeed` to `true` when configuring a stream against a table to read the change data feed, as shown in the following syntax example:

Python **Scala**

Python

Ask Assistant

```
(spark.readStream
  .option("readChangeFeed", "true")
  .table("myDeltaTable")
)
```

Default behavior

When the stream first starts, it returns the latest snapshot of the table as `INSERT` records and then returns future changes as change data. The change data commits as part of the Delta Lake transaction and becomes available at the same time the new data commits to the table.

Additional options

You can optionally specify a starting version (see Specify a starting version) or use batch execution (see Read changes in batch queries). Databricks also supports rate limits (`maxFilesPerTrigger`, `maxBytesPerTrigger`) and `excludeRegex` when reading change data.

For versions other than the starting snapshot, rate limiting applies atomically to entire commits—either the entire commit is included in the current batch, or it's deferred to the next batch.

Specify a starting version

To read changes from a specific point, specify a starting version using either a timestamp or version number. Starting versions are required for batch reads. You can optionally specify an ending version to limit the range. To learn more about Delta Lake table history, see What is time travel?.

When you configure Structured Streaming workloads involving change data feed, understand how specifying a starting version impacts processing:

- New data processing pipelines typically benefit from the default behavior, which records all existing records in the table as `INSERT` operations when the stream first starts.
- If your target table already contains all the records with appropriate changes up to a certain point, specify a starting version to avoid processing the source table state as `INSERT` events.

The following example shows syntax for recovering from a streaming failure in which the checkpoint was corrupted. In this example, assume the following conditions:

1. Change data feed was enabled on the source table at table creation.
2. The target downstream table has processed all changes up to and including version 75.
3. Version history for the source table is available for versions 70 and above.

Python Scala

Python

```
(spark.readStream
  .option("readChangeFeed", "true")
  .option("startingVersion", 76)
  .table("source_table")
)
```

In this example, you must also specify a new checkpoint location.

> **IMPORTANT**
>
> If you specify a starting version, the stream fails to start from a new checkpoint if the starting version is no longer present in the table history. Delta Lake cleans up historic versions automatically, meaning that all specified starting versions are eventually deleted.

See Replay table history.

Read changes in batch queries

You can use batch query syntax to read all changes starting from a particular version or to read changes within a specified range of versions.

- Specify versions as integers and timestamps as strings in the format `yyyy-MM-dd[ HH:mm:ss[.SSS]]`.
- Start and end versions are inclusive.
- To read from a starting version to the latest version, specify only the starting version.

- Specifying a version before change data feed was enabled throws an error.

The following syntax examples demonstrate using starting and ending version options with batch reads:

SQL Python Scala

SQL

```
-- version as ints or longs e.g. changes from version 0 to 10
SELECT * FROM table_changes('tableName', 0, 10)

-- timestamp as string formatted timestamps
SELECT * FROM table_changes('tableName', '2021-04-21 05:45:46', '2021-05-21 12:00:00')

-- providing only the startingVersion/timestamp
SELECT * FROM table_changes('tableName', 0)

-- database/schema names inside the string for table name,
-- with backticks for escaping dots and special characters
SELECT * FROM table_changes('dbName.`dotted.tableName`', '2021-04-21 06:45:46' , '2021-
05-21 12:00:00')
```

Handle out-of-range versions

By default, specifying a version or timestamp exceeding the last commit throws the error `timestampGreaterThanLatestCommit`. In Databricks Runtime 11.3 LTS and above, you can enable tolerance for out-of-range versions:

SQL

```
SET spark.databricks.delta.changeDataFeed.timestampOutOfRange.enabled = true;
```

With this setting enabled:

- **Start version/timestamp beyond last commit**: Returns an empty result.
- **End version/timestamp beyond last commit**: Returns all changes from start to the last commit.

Record data changes

Delta Lake records data changes efficiently and might use other Delta Lake features to optimize storage representation.

Storage considerations

- **Storage costs**: Enabling change data feed might cause a small increase in storage costs because changes may be recorded in separate files.
- **Operations without change files**: Some operations (insert-only, full-partition deletions) do not generate change data files—Databricks computes the change data feed directly from the transaction log.
- **Retention**: Change data files follow the table's retention policy. The `VACUUM` command deletes them, and changes from the transaction log follow checkpoint retention.

Do not attempt to reconstruct the change data feed by directly querying change data files. Always use Delta Lake APIs.

Replay table history

Change data feed is not intended to serve as a permanent record of all changes to a table. It only records changes that occur after it's enabled, and you can start a new streaming read to capture the current version and all subsequent changes.

Records in the change data feed are transient and only accessible for a specified retention window. The Delta Lake transaction log removes table versions and their corresponding change data feed versions at regular intervals. When a version is removed, you can no longer read the change data feed for that version.

Archive change data for permanent history

If your use case requires maintaining a permanent history of all changes to a table, use incremental logic to write records from the change data feed to a new table. The following example demonstrates using `trigger.AvailableNow` to process available data as a batch workload for auditing or full replayability:

Python Scala

Python

```
(spark.readStream
  .option("readChangeFeed", "true")
  .table("source_table")
  .writeStream
  .option("checkpointLocation", <checkpoint-path>)
  .trigger(availableNow=True)
  .toTable("target_table")
)
```

Change data feed limitations for tables with column mapping

With column mapping enabled on a Delta table, you can drop or rename columns without rewriting data files. However, change data feed has limitations after non-additive schema changes such as renaming or dropping columns, changing data types, or nullability changes:

- **Batch semantics**: You cannot read change data feed for a transaction or range in which a non-additive schema change occurs.
- **Databricks Runtime 12.2 LTS and below**: Tables with column mapping enabled that have experienced non-additive schema changes do not support streaming reads on change data feed. See Streaming with column mapping and schema changes.
- **Databricks Runtime 11.3 LTS and below**: You cannot read change data feed for tables with column mapping enabled that have experienced column renaming or dropping.

In Databricks Runtime 12.2 LTS and above, you can perform batch reads on change data feed for tables with column mapping enabled that have experienced non-additive schema changes. Read operations use the schema of the end version specified in the query rather than the latest table version. Queries still fail if the version range spans a non-additive schema change.