# Incremental ETL Public Guide

unit

Updated: July 1st, 2025

## Overview

Unit provides access to its raw data to build reporting solutions and make informed business decisions. This integration uploads the data into your storage, so that it can be easily pulled into your own data warehouse, eliminating the need to build a sequence of API calls to obtain data.

The data is structured to make it easy to perform robust analysis on any part of your product. Additionally, KYC information can be securely transferred to companies that can securely store personally identifiable information.

By accessing raw data and building a customized analytics stack, decision-makers in your organization can gain deeper insights and make data-driven decisions across various areas. Key questions that can be explored include:

- Which customer cohorts or marketing campaigns generate the highest revenue?
- How do pricing strategies—such as interest rates, payment fees, and rewards programs—impact customer engagement?
- Which high-engagement customers should be upgraded to "VIP" status with enhanced benefits like higher cashback?
- Where do your customers spend the most, and which merchants capture the largest share of their transactions?
- How do different customer segments utilize ACH, wire transfers, and check deposits?

The data transferred includes the following data:

- Account Events
- Accounts
- Applications
- Authorization Holds
- Authorization Requests
- Beneficial Owners
- Bins
- Card Designs
- Card Scheme Fees
- Cards
- Check Deposits
- Counterparties
- Customers

- Deposit Products
- Documents
- Disputes
- Officers
- Payments
- Received Payments
- Transactions

# Incremental ETL (ETL V.2)

## Motivation for Change

**Addresses scalability, performance, and historical data tracking.**
Transitioning to incremental updates will enhance data management efficiency, improve scalability to meet user demands, and provide the ability to track the history of information changes, ultimately leading to better system performance and analytics capabilities.

**Value of change**

- Scalability: ETL V2 is designed to efficiently process large volumes of data, ensuring seamless data flow across the upgraded system.
- Reliability: The increments model is better suited to support high volumes of data
- Data Freshness: Data updates will occur up to four hours, providing more timely and accurate insights.
- Historical Data Tracking: ETL V2 maintains a comprehensive history of data changes, enabling better tracking, analysis, and auditing of historical information.

## What is different

After reading the documentation, you should be familiar with the new Data Export data, files, data freshness and structure. Since you worked with ETL V1 before, there are key differences we wish to elaborate on.

| | Existing configuration (V1) | Incremental configuration (V2) |
|---|---|---|
| Storage | S3, GCP, Azure | S3, GCP, Azure |
| File destination | Bucket/Table Name/Year_Month=yyyy-mm/file | Bucket/Table Name/year=yyyy/month=mm/day=dd/hour=hh/file |
| Joined data between tables | Some tables are loaded after a join operation | All tables will be reflected as in the Unit DB |
| Freshness | Twice a day | Up to 4 hours |
| Data loading method | Overwrite - loading and overwriting the existing data | Incremental - loading only the changes |
| | No history | History exists per row change |
| Adding a new column with backfill | Overwrites the entire table | Requires to overwrite/reload the entire table with incremental data from a specified point in time<br>● Client - pause processing of incoming data and get prepared for reprocessing files (from a specific point in time)<br>● Unit - start resending Table data with enriched column<br>● Client - reprocess files |
| Drop an existing column | Dropped column will be populated with NULL | Dropped column will no longer be included for future export files (*fail-fast*) |

Example
Data model (*accounts*):

- Existing (single, most updated row)
    - account_id
    - status

    - …
- New (Incremental, row for each INSERT / UPDATE / DELETE statement):
    - operation_type (INSERT / UPDATE / DELETE)
    - commit_timestamp
    - account_id
    - Status
    - …

For example, given the following data updates:

- Insert account (account_id=1, status='Open', timestamp=t1)
- Update account (account_id=1, status='Frozen, timestamp=t2)
- Update account (account_id=1, status='Closed',timestamp=t3)

The existing data will contain the following single row:

| **account_id** | **status** |
|---|---|
| 1 | 'Closed' |

The new data will contain the following rows:

| **operation_type** | **commit_timestamp** | **account_id** | **status** |
|---|---|---|---|
| INSERT | t1 | 1 | 'Open' |
| UPDATE | t2 | 1 | 'Frozen' |
| UPDATE | t3 | 1 | 'Closed' |

**Tables change**

In V1, we sent a few tables as joined tables. In the incremental ETL, tables will be loaded "as is".

Examples of the change in tables

| **V1 Table - 1 table** | **Incremental ETL - 2 tables** | **How to construct?** | |
|---|---|---|---|
| Transactions | transactions, account_events, card_scheme_fees | JOIN ON transactions.id = account_events.trans | |

| | | | |
|---|---|---|---|
| | | action_id LEFT JOIN ON transactions.id = card_scheme_fees.activity_id | |
| Not available | transactions_v2 | | Differences between transactions vs transactions_v2:<br>1. column "interchange" on transactions is called "org_interchange" on transactions_v2<br>2. "Id" column on transactions is equivalent to "transaction_id" on transactions_v2 . The `transactions_v2.id` column is simply a technical incrementing identifier, and the `transactions_v2.transaction_id` column is the one that represents the transaction itself and should be used for joins (for example, with `payments.transaction_id`). The reason for that is that under `transactions_v2`, there can be multiple rows for a single transaction, |

| | | | |
|---|---|---|---|
| | | | and the id column is used solely for DB uniqueness. |
| Not available | transactions_v2 | Readily available | |
| Cards | cards, card_designs, bins | LEFT JOIN card_desings ON card_desings.card_id = cards.id LEFT JOIN ON bins.bin = cards.bin | |

***NOTES***

1. Column renaming: The column **approve_reasons** in the applications table (V1) renamed to deny_reasons (V2) since this is in use for deny only.
2. The **transactions** table in V2 will not include data related to any operational accounts (reserve,revenue). This data can only be accessed via the dashboard.
3. Column renaming: The column approve_reasons in the applications table (V1) renamed to deny_reasons (V2) since this is in use for deny only.
4. While structural changes to the ETL tables are infrequent, they may occur in the future. This reflects the ETL's alignment with the internal structure of the Unit database.e.

## Implementation

- Provide Unit a new destination bucket and an IAM role (or equivalent in GCP/Azure), granting Unit write permissions to this bucket
- Adjust the internal data pipelines to handle the new data model described above