Data discovery

Project: Retail Transaction Ingestion

- Goal: Build a data pipeline that ingests and transforms client-supplied CSV data (clients, products, stores, transactions) into a usable database format for analysis and operational use.

1. Datasets Overview

1.1 Static Files (One-time Load)

File Name	Description
clients.csv	Contains customer profile info
products.csv	Product metadata
stores.csv	Store location and hours info

1.2 Dynamic Files (Daily Incremental Load)

File Pattern	Description
transactions_{YYYY-MM-DD}_{H}.csv	Hourly transaction records from the previous day, delivered daily at 9:00 AM. Recorded data is available from 8:00AM to 9:00PM

2. Data schema

2.1 clients.csv

Column	Type	Description
id	INT	Unique identifier for the client
name	TEXT	Full name of the client
job	TEXT	Profession
email	TEXT	Email address
account_id	TEXT	Fidelity card number

2.2 stores.csv

Column	Type	Description
id	INT	Store ID
latlng	TEXT	GPS coordinates of the store
opening	TEXT	Opening time
closing	TEXT	Closing time
type	TEXT	Type of store

2.3 products.csv

Column	Type	Description
id	INT	Product ID
ean	TEXT	European Article Number (barcode)
brand	TEXT	Brand name
description	TEXT	Product description

2.4 transactions_YYYY-MM-DD_H.csv

Column	Type	Description
transaction_id	INT	Unique transaction ID
client_id	INT	Foreign key to client
date	DATE	Date of transaction (YYYY-MM-DD)
hour	INT	Hour of the transaction (0–23)
minute	INT	Minute of the transaction (0–59)
product_id	INT	Foreign key to product
quantity	INT	Quantity purchased
store_id	INT	Foreign key to store

3. Data quality considerations

- Nulls / Missing Values: Check especially in transactional joins (e.x: unmatched client or store)
- Data types: Ensure types are strictly enforced (e.x: no non-integer in quantity)
- Duplicate Transactions: Ensure transaction_id uniqueness
- Consistency: Validate foreign key relationships
- Csv files: separated by `;` and the first row should be the header. Need to ensure that any schema changes will be early informed.

4. Data modeling

Final Tables:

- Clients: Direct from clients.csv
- Products: Direct from products.csv
- Stores: From stores.csv with parsed latitude, longitude
- Transactions: From daily/hourly files with:
    - transaction_id, client_id, product_id, store_id, quantity
    - account_id from clients
    - timestamp as datetime(date + hour + minute)

5. Assumptions and recommendations

Static File Management Strategy

The three static files (clients.csv, products.csv, and stores.csv) are typically stable datasets that only require ingestion when updates occur in the source system. As such, these tables should not be reloaded every pipeline run. To maintain data consistency over time and enable accurate historical reporting, a more controlled update strategy is recommended.

- Change Detection:
    - Implement logic to detect whether the incoming file differs from the latest ingested version.
- Versioning Columns:
    - updated_timestamp (DATETIME): Indicates the last time a row was inserted or modified.
    - active_flag (BOOLEAN): Marks whether a record is currently active (True) or has been logically deprecated (False), supporting soft deletes and temporal tracking.
- Upsert mechanism:
    - Instead of truncating and reloading entire tables, perform upserts (update if existing, insert if new).
- Avoid full rewrites:
    - Replacing static datasets wholesale can result in loss of data lineage and complications in downstream systems.
    - Selective updates provide more reliable data integrity, especially when dealing with customer, store, or product metadata linked to transaction history.

Handling delayed data delivery

Data is expected to land in the private Azure Storage Blob at 9:00AM every day with the transactions of the previous days. 13 files representing data from 9:00 to 20:00 are expected daily. However, file deliveries may be late or incomplete sometimes.

We assume that once the data is uploaded, it will not be reuploaded with corrections unless explicitly notified.

- Implement File Arrival Validation (ex: can use Airflow)
    - For each run, check whether hourly files are present for the expected processing date.
    - Log missing files and surface alerts if fewer than expected
- Track Ingestion Status
    - Maintain a metadata log table (e.g., ingestion_log) that records: file_name, expected_date, ingestion_timestamp, status (success, missing, pending, error)
- Retry/rerun mechanism
    - When configuring the orchestration tool (ex: Airflow, Datafactory), set a re-poke period such as 30mins, try poking at least 3 times before flagging the job as failed.
    - In case of reprocessing, deduplicate data by process_date (proposed supporting column) before inserting
- Implement notification system
    - Integrate an alert mechanism (ex: email, Slack) when a file is missing past its expected window or retry has succeeded or failed.

6. Tool Selection Strategy

For this test

- DBMS: SQLite
    - No server setup required, built-in Python support
    - Single file database
    - Suitable for the scale of this test
- Orchestration: windows task scheduler (suggested)
    - Minimal setup with no external dependencies; configuration is stored as a .dat file.
    - Suitable for basic automation in development or test environments.
    - Not implemented in this solution, as Windows Task Scheduler lacks of support for complex job dependencies; data-aware scheduling (ex: trigger on file arrival)

For production environment

- DBMS: PostgreSQL, or a managed cloud database (ex: Azure SQL)
    - Better performance under concurrent access
    - Easier to integrate with BI and analytics tools

    - Role-based access control and better security features

- Orchestration: Apache Airflow, or DataFactory
    - Support for job dependencies, conditional logic, and retries
    - Integration with scheduling, monitoring, and alerting
    - Web UI for real-time visibility into pipeline execution
- Security and secrets management: store Connection strings in Azure Key Vault or AWS Secrets Manager
- Testing and validation
    - Add unit tests for transformation logic
    - Include data validation checks (e.g., schema validation, null checks)
    - Consider tools like Great Expectations for validating data quality

7. Assumptions and test pipeline behavior

Static File Handling

Static reference files (clients.csv, products.csv, stores.csv) are expected to rarely change. Therefore, on each run, the system computes a SHA-256 hash of each file and compares it to the previously stored value. If no changes are detected, the file is skipped, ensuring efficient and idempotent processing. Otherwise, the file is ingested and its hash is updated in the *file_ingestion_log* table.

Transaction File Handling

All files for the provided date (between 08:00 and 20:00) are downloaded and parsed. For each, the script checks that the first line is truly a header, ensuring the CSV is in a proper ingestible format. Records from each file are appended into the transactions table after transformation, including:

- Enrichment with *account_id* from the clients table
- Construction of a full timestamp (*transaction_time*)
- Logging ingestion events for traceability

Ingestion Completeness

The current implementation does not verify whether all expected hourly transaction files are present before processing. This validation is deferred to a production-grade orchestration tool such as Apache Airflow, which can monitor expected file counts and trigger alerts on missing or delayed data.

Final Output

Upon successful execution, the pipeline creates and populates the following 5 tables in a SQLite database:

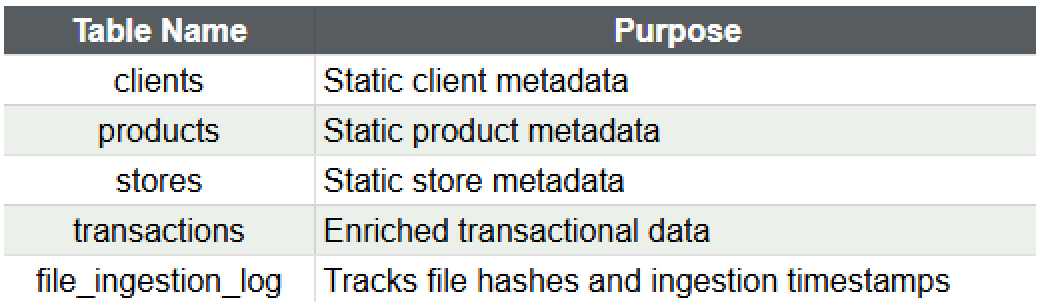

Table Name	Purpose
clients	Static client metadata
products	Static product metadata
stores	Static store metadata
transactions	Enriched transactional data
file_ingestion_log	Tracks file hashes and ingestion timestamps