Analytics DW Pipeline
Functional Specification

Documented by: Wishmi Lakshani

Table of Content

1. Execution and Ingestion

1.1 Orchestration

The three data generation scripts were refactored as functions and integrated into the pipeline.

The generated input files are loaded into the staging tables stg_attendance, stg_sales, and stg_finance after light type normalization and duplicated row removal.

Then the dimension tables, dim_date, dim_location, dim_product, dim_employee, dim_currency and fact tables, fact_attendance, fact_sales, fact_finance will be populated with data.

1.2 Expected Outputs

Input data generation scripts

- File format - CSV

Staging tables

Table	**Fields**
stg_attendance	staffid, name, region, country, department, date, status, checkInTime, checkOutTime
stg_sales	saleid, region, country, product, date, currency, quantity, unitPrice, totalSales
stg_finance	Transactionid, region, country, product, date, currency, revenue, expense, profit

Dimension tables

Table	**Fields**
dim_date	date_key, year, quarter, month, month_name, day_of_month, day_of_week, day_name
dim_location	location_key, region, country
dim_product	product_key, product_name
dim_employee	employee_key, staffed, name, department, home_country, home_region
dim_currency	currency_key, currency_code, date_to_usd

Fact tables

Table	Fields
fact_attendance	attendance_key, employee_key, location_key, date_key, status, checkin_time, checkout_time
fact_sales	saleid, product_key, location_key, date_key, currency_key, quantity, conversion_rate_to_usd, unit_price_usd, total_sales_usd
fact_finance	Transaction_id, product_key, location_key, date_key, currency_key, conversion_rate_to_usd, revenue_usd, expense_usd, profit_usd

2. Normalization and Currency

2.1 Normalization

Followed star schema with dimensions describing business entities and fact tables storing measurable events.

dimention tables

- dim_date
- dim_location
- dim_product
- dim_employee
- dim_currency

fact tables

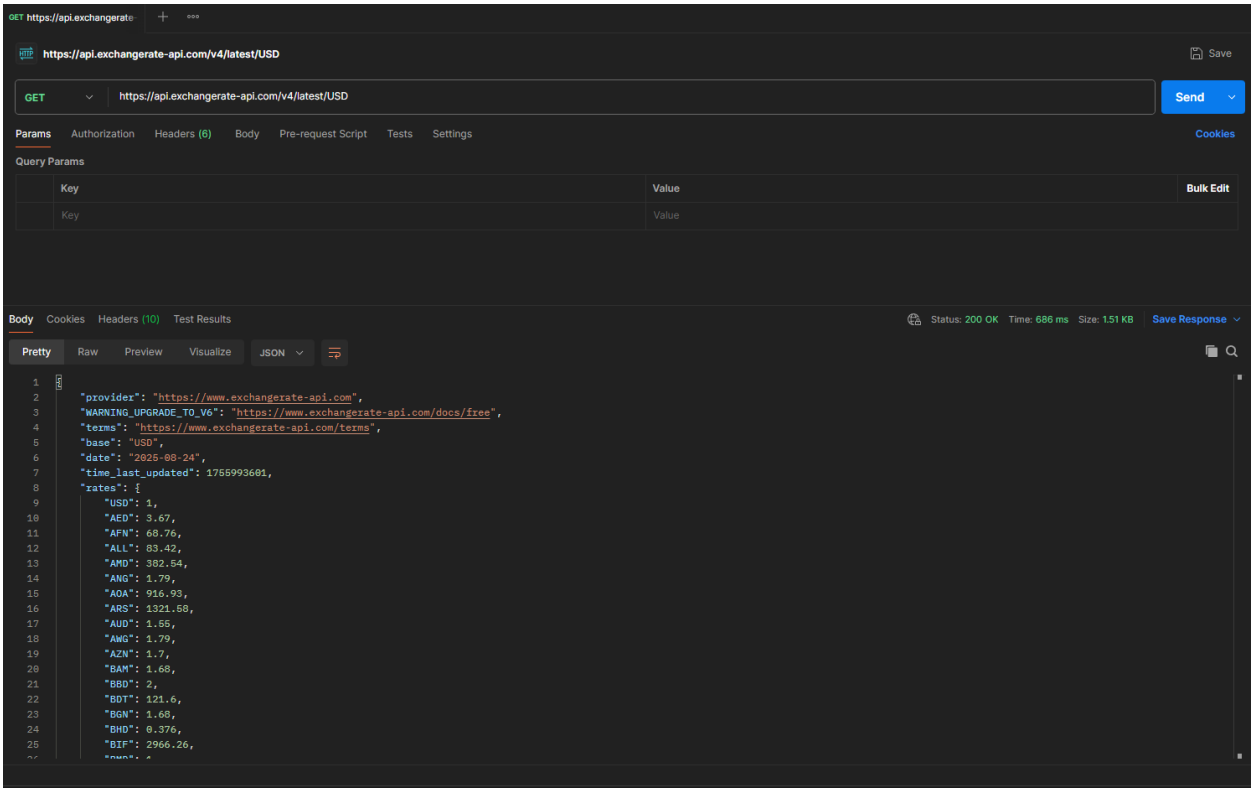
- fact_attendance
- fact_sales
- fact_finance

2.2 Currency

All persisted monetary facts in final fact tables are USD.

For the currency conversion, the below REST API which was provided by the ExchangeRate-API.com service was tested and used.

https://api.exchangerate-api.com/v4/latest/USD



The currency rates retrieved from the REST_API were injected into dim_currency table.

And then after the calculations, converted USD amounts were loaded into the fact tables.

- fact_sales - unit_price_usd, total_sales_usd
- fact_finance - revenue_usd, expense_usd, profit_usd

3. Data Quality

Type normalization

- Date → DATE
- CheckInTime / CheckOutTime → TIME
- Placeholders → NULL
- Numeric coercion

Removed duplicated rows.

Implemented successful dimension lookups.

Local currency codes are available in monetary fact tables.

USD sanity thresholds

- fact_sales: UnitPrice_usd ≥ 0, TotalSales_usd ≥ 0
- fact_finance: Revenue_usd ≥ 0, Expense_usd ≥ 0,
  Profit_usd = Revenue_usd − Expense_usd

Summary reports are logged per run.

4. Idempotency and Upserts

Current behavior

- Staging: WRITE_TRUNCATE per run and exact-row dedupe ensures the same inputs yield the same staging tables.
- Facts: MERGE statements using deterministic keys prevent duplicate facts across runs.
- Currently the multiple runs are handled by truncating the tables.

Future implementations (no truncation)

- Add a run_id column to staging and fact tables.
- Keep a runs metadata table (run_id, started_at, status, row_counts).
- Upsert rule:
  - Same run_id - MERGE only updates existing rows.
  - New run_id - MERGE inserts new rows and updates changed ones.

- Also can retain prior versions by storing previous run_id rows for audit, or clean older runs by policy.

5. Performance and Storage

Chosen strategy: Batch loading

Reasons

- Predictable cost
- High throughput for large files
- Stable schema enforcement

Trade-offs

	Pros	**Cons**
Batch loading	Lower cost Schema detection and validation options	Not real-time Job latency
Stream loading	Near real-time availability Row-by-row ingestion	Higher cost Temporary streaming buffer Complex error handling

6. Observability

Every step logs a JSON object with,

- ts (UTC ISO8601), step (example: staging.start, dim_date.loaded)
- Status metrics (counts, table names)
- Error (string, when applicable)
- Totals or details of the objects for summaries

Also has included,

- Clear error messages embedded in logs with failing step context.
- Handling validation or load failure.

7. Report Generation Using Dimension and Fact Table

The initial reports generated by the scripts can be generated back using the populated dimension and fact tables by using below queries.

Attendance data:

```
SELECT
  e.StaffID,
  e.Name,
  l.Region,
  l.Country,
  e.Department,
  d.date_key AS Date,
  f.status AS Status,
  f.checkin_time AS CheckInTime,
  f.checkout_time AS CheckOutTime
FROM `analytics-pipeline-assessment.analytics_dw.fact_attendance` f
JOIN `analytics-pipeline-assessment.analytics_dw.dim_employee` e
  ON f.employee_key = e.employee_key
JOIN `analytics-pipeline-assessment.analytics_dw.dim_location` l
  ON f.location_key = l.location_key
JOIN `analytics-pipeline-assessment.analytics_dw.dim_date` d
  ON f.date_key = d.date_key
ORDER BY f.attendance_key;
```

Sales Data:

```
SELECT
  f.saleid AS SaleID,
  l.Region,
  l.Country,
  p.product_name AS Product,
  d.date_key AS Date,
  c.currency_code AS Currency,
  f.Quantity,
  CAST(ROUND(SAFE_DIVIDE(f.unit_price_usd, NULLIF(f.conversion_rate_to_usd, 0)), 2) AS NUMERIC) AS UnitPrice,
  CAST(ROUND(SAFE_DIVIDE(f.total_sales_usd, NULLIF(f.conversion_rate_to_usd, 0)), 2) AS NUMERIC) AS TotalSales
FROM `analytics-pipeline-assessment.analytics_dw.fact_sales` f
JOIN `analytics-pipeline-assessment.analytics_dw.dim_product` p
  ON f.product_key = p.product_key
JOIN `analytics-pipeline-assessment.analytics_dw.dim_location` l
  ON f.location_key = l.location_key
JOIN `analytics-pipeline-assessment.analytics_dw.dim_date` d
  ON f.date_key = d.date_key
JOIN `analytics-pipeline-assessment.analytics_dw.dim_currency` c
  ON f.currency_key = c.currency_key
ORDER BY f.saleid;
```

Finance data:

```
SELECT
 f.transaction_id AS TransactionID,
 l.Region,
 l.Country,
 p.product_name AS Product,
 d.date_key AS Date,
 c.currency_code AS Currency,
 CAST(ROUND(SAFE_DIVIDE(f.revenue_usd, NULLIF(f.conversion_rate_to_usd, 0)), 2) AS NUMERIC) AS Revenue,
 CAST(ROUND(SAFE_DIVIDE(f.expense_usd, NULLIF(f.conversion_rate_to_usd, 0)), 2) AS NUMERIC) AS Expense,
 CAST(ROUND(SAFE_DIVIDE(f.profit_usd,  NULLIF(f.conversion_rate_to_usd, 0)),  2) AS NUMERIC) AS Profit
FROM `analytics-pipeline-assessment.analytics_dw.fact_finance` f
JOIN `analytics-pipeline-assessment.analytics_dw.dim_product`  p ON f.product_key  = p.product_key
JOIN `analytics-pipeline-assessment.analytics_dw.dim_location` l ON f.location_key = l.location_key
JOIN `analytics-pipeline-assessment.analytics_dw.dim_date`     d ON f.date_key     = d.date_key
JOIN `analytics-pipeline-assessment.analytics_dw.dim_currency` c ON f.currency_key = c.currency_key
ORDER BY f.transaction_id;
```