SQL ETL Pipeline Simulation

Introduction:

The SQL ETL Pipeline Simulation project uses SQL to simulate the Extract, Transform, and Load (ETL) process that occurs in the real world for example in data analytics and warehousing. This project aims to illustrate the process of extracting raw data from source files, cleaning and transforming it for consistency, and then loading it into a production environment that is structured and prepared for analytics.

The project focuses on audit mechanisms to monitor data movement across stages, automation through triggers, and data quality management. It offers a practical grasp of ETL design within a small but effective relational database system using SQLite Studio and DB Browser.

Abstract

The main goal of this project is building a comprehensive ETL procedure to replicate a retail business environment. After being imported into staging tables, raw CSV files are cleaned by eliminating duplicates, correcting nulls, and recording invalid records in an ETL Rejected Records table.
Customers, orders, and payments are among the production tables into which the cleaned data is loaded after being converted and standardized. While triggers guarantee data integrity, an ETL Audit Log keeps track of insert counts and timestamps. To extract important business insights, analytical SQL views were developed.
Lastly, an ER diagram that shows links between tables.

Tools Used

- SQLite Studio & DB Browser: The database engine and graphical user interface (GUI) used to run SQL queries and view data
- DrawSQL.app: This tool is used to create Entity Relationship (ER) diagrams that show the relationships between different database tables, such as orders, payments, and customers.
- CSV files- used to import or simulate raw data.

Steps Involved in Building the Project

1. **Importing Raw Data:** Three staging tables, staging_customers, staging_orders, and staging_payments, were imported with CSV data. The "Extract" phase of the ETL process is simulated by these tables.
2. **Data Cleaning:** DELETE and UPDATE statements were used to eliminate duplicate and null records from the staging tables.
   For traceability, records lacking important identifiers (such as customer_id or order_id) were added to the etl_rejected_records table.

To ensure data completeness, default values were supplied for missing emails and countries.

3. **Data Transformation:** To standardize formats and normalize attributes, logical transformations were used.
   For instance, payment methods like UPI or Card were derived using conditional logic, and payment statuses like "done" or "paid" were combined under "Success."
4. **Loading into Production Tables:** The customer, order, payment, and order_items production tables were filled with the cleaned and transformed data. By connecting clients to their individual orders and payments, foreign key relationships were used to guarantee referential integrity.
5. **Audit Logging:** Timestamps and the quantity of rows loaded into each production table are recorded in the etl_audit_log table.
   This makes it possible to monitor the amount of data processed throughout each ETL cycle.
6. **Validation Triggers:** To automatically verify new insertions, triggers were developed. To ensure consistency in the database, triggers, for example, prevent the insertion of an order for a client that does not exist or a payment for an order that does not exist.
7. **Creation of Views and Indexes:** Analytical views like v_customer_order_summary and v_customer_sales, v_customer_revenue, v_country_avg_order were made.
   These views facilitate reporting and analytics by summarizing the total orders, total spending per client, total revenue for each customer and average order value for each country.
8. Several indexes were added to improve query execution speed like idx_orders_customer_id, idx_orderitems_order_id, idx_customers_country, idx_payments_order_id.
9. **Export and Documentation:** The cleaned production tables and ETL logs were exported following the validation of the ETL process.
   Additionally, an ER diagram and project report were created for submission and documentation.

Conclusion:

The SQL ETL Pipeline Simulation effectively illustrates how, the entire ETL lifecycle can be implemented with SQL. It exhibits proficiency in trigger-based automation, data transformation, and cleaning. Along with improving knowledge of SQL commands, this project established a solid foundation in error handling, validation procedures, and data pipeline automation.