etl\_staging\_stories

# Establish VPC Connection from etl-processor to Production Oracle DB

**Description:** As a developer, I need to demonstrate that the etl-processor Cloud Run Job can successfully connect to the production Oracle SQL database through the configured VPC Connector. This is a foundational step to ensure the necessary roles are assigned to service account and drivers are working correctly.

**Acceptance Criteria:**

* The correct roles are identified for runtime service account used by etl-processor
* When executed, the job successfully establishes a connection to the production Oracle database via the VPC Connector.
* The job executes a simple, read-only query and logs the successful result to Cloud Logging.

# Staging table writing proof of concept

**Description:** As a developer, I need to demonstrate the ability of the etl-processor job to write data into a staging table using the provide SQL queries. This will define the process to modify integrate SQL queries into processor code.

**Acceptance Criteria:**

* The provided SQL queries can be integrated into processor job
* The job can connect to a non-production/staging Oracle database instance.
* The job successfully creates a staging\_customers table based on a predefined schema.
* Data from a sample customers.csv file is read by the job and inserted into the staging\_customers table.

# **Implement Staging Table Creation and Data Insertion – Customers.csv**

**Description:** As a developer, I need the etl-processor Cloud Run Job to dynamically create staging tables in the production Oracle SQL database that mirror the schema of the inbound CSV files. Once created, the job must insert the data from the downloaded CSVs into these tables. Each record must be tagged with the

load\_id for traceability. The specific insertion method is to be determined

**Acceptance Criteria:**

* A corresponding staging\_<entity\_name>\_<load\_id> table is created in Oracle for customer CSV file
* The schema of each staging table matches the columns in the corresponding CSV, plus an additional column for the load\_id.
* The row count in each staging table matches the row count specified in the manifest.json

# **Implement Transactional Upsert to Production**

**Description:** As a developer, I need to implement logic within the etl-processor job to perform Transactional Upsert from Staging tables to Production Oracle SQL tables. The process should be transactional to ensure data integrity. Data will be updated in the following order: customers -> vehicles -> invoices -> line\_items -> payments

**Acceptance Criteria:**

* The data loading process is wrapped in a single database transaction; if any step fails, the entire operation is rolled back.
* Upserts are performed in the specified dependency order.
* An appropriate upsert strategy (e.g., Oracle's MERGE statement) is used.
* New source records are inserted and existing records are updated in the production tables.

# **Implement Post-Load Reconcilliation and Ledger Update**

**Description:** As a developer, I need the etl-processor job to perform reconciliation by comparing row counts after the production upsert is complete. The results and status must be written to the load\_ledger table for auditability.

**Acceptance Criteria:**

* The job compares row counts from the manifest, staging tables, and the affected rows in production.
* A new record is written to the load\_ledger table for the load\_id with a final status (SUCCESS/FAILURE), source count, staging count, and production count.
* If counts mismatch, the status in the load\_ledger is marked appropriately.