**Hevo Exercise - Assessment II: Post-Load with Hevo Models**

This document presents the detailed solution for **Assessment II: Messy E-Commerce Orders**, following an **ELT (Extract, Load, Transform)** methodology. The approach leverages **Hevo Data** for efficient ingestion and **Snowflake** for all complex data cleaning and transformation, executed via **Hevo Models**.

1. **Phase I: Pipeline Setup and Data Ingestion**

From assignment1 we already have configured neon and pgdmin So I have used same here.

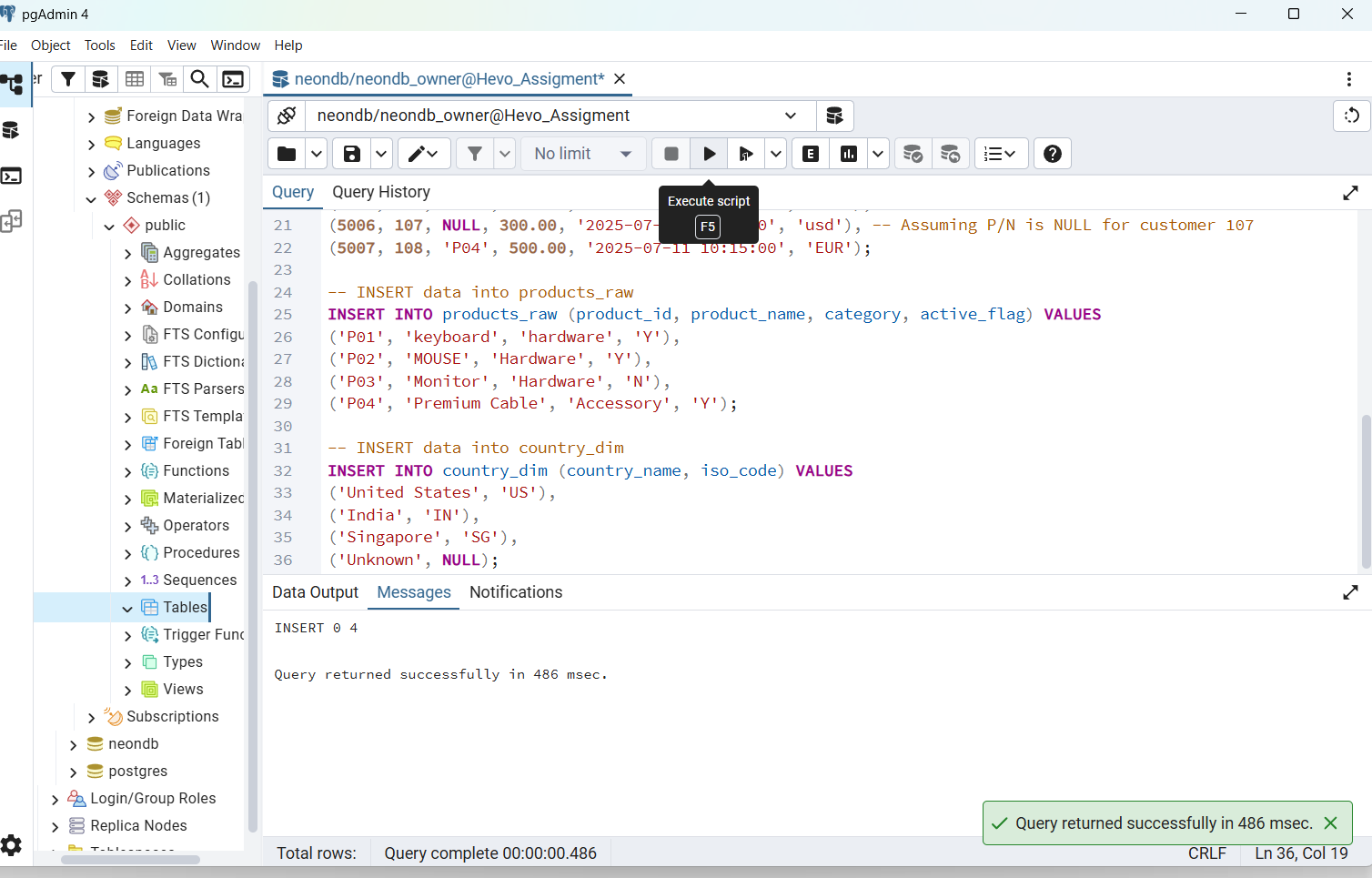
So to create table I directly move to Pgadmin

**A. PostgreSQL Setup**

The initial step involved preparing the source data in PostgreSQL:

* Tables Created: All four raw tables ( and ) were created within a single PostgreSQL instance.
* Data Loaded: All sample data provided in the assignment was inserted into their respective tables using standard statements.

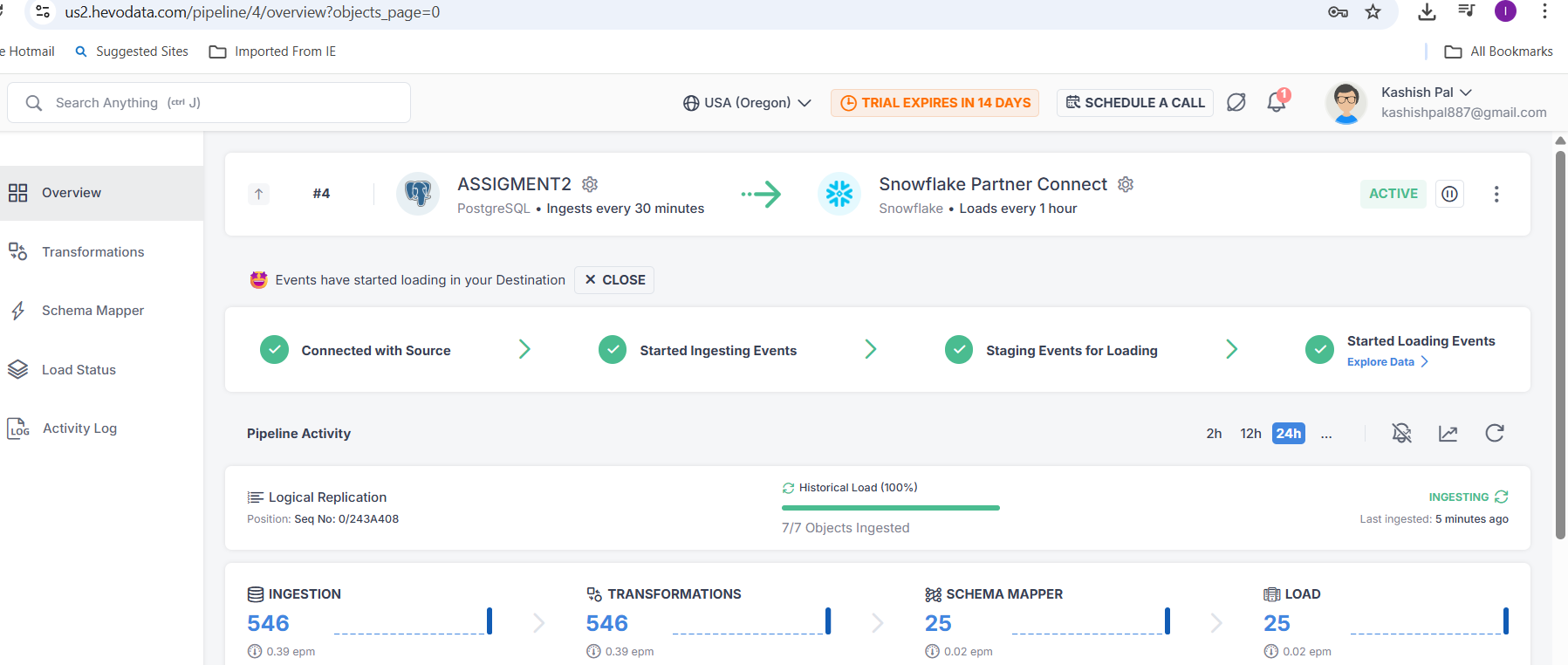
By using insert commands screenshot below



**B. Hevo Pipeline Configuration**

A Hevo Data pipeline was configured to move the data from PostgreSQL to Snowflake:

* **Source Connection:** The pipeline connected to the PostgreSQL database using and
* **Ingestion Mode:** **Logical Replication** was chosen, enabling **Change Data Capture (CDC)** for future real-time updates.
* **Destination:** The destination was set to the **Snowflake** Partner Connect trial account.
* **Data Load:** Hevo successfully performed the initial historical load, pushing all four raw tables into the Snowflake destination schema, ready for transformation.



A screenshot of a computer

AI-generated content may be incorrect.

**2. Phase II: Data Transformation via Hevo Models**

All cleaning, standardization, and joining logic were implemented as **Hevo Models**, which execute the directly against the Snowflake database.  
  
To create model I moved to hevo then choose destination partner as Snowflake.  
  
Then as per given assignment I ran reliveant query to achive those

**Model 1: (Tasks 5 & 6)**

**Purpose:** Deduplicate records, standardize and clean and handle nulls.

SQL

WITH Deduped AS (

-- 5. Deduplicate Customers: Keep the most recent record

SELECT

\*,

ROW\_NUMBER() OVER (PARTITION BY customer\_id ORDER BY updated\_at DESC) as rn

FROM customers\_raw

),

Cleaned AS (

SELECT

t1.customer\_id,

LOWER(t1.email) AS email\_cleaned, -- 5. Standardize emails to lowercase

-- 5. Standardize phone to 10-digit format or 'Unknown'

CASE

WHEN REGEXP\_REPLACE(t1.phone, '[^0-9]', '') RLIKE '^[0-9]{10}$'

THEN REGEXP\_REPLACE(t1.phone, '[^0-9]', '')

ELSE 'Unknown'

END AS phone\_cleaned,

-- 6. Standardize Country Codes (Handle variations: usa, UnitedStates, IND, SINGAPORE)

COALESCE(t2.iso\_code, 'Unknown') AS country\_code\_cleaned,

-- 6. Handle created\_at <null>

COALESCE(t1.created\_at, '1900-01-01'::TIMESTAMP\_NTZ) AS created\_at\_cleaned,

t1.updated\_at

FROM Deduped t1

LEFT JOIN country\_dim t2

ON UPPER(t1.country\_code) = UPPER(t2.iso\_code)

OR UPPER(t1.country\_code) = UPPER(REPLACE(t2.country\_name, ' ', ''))

WHERE

t1.rn = 1 AND t1.customer\_id IS NOT NULL

)

SELECT \* FROM Cleaned

**Model 2: (Task 8)**

**Purpose:** Standardize and and flag inactive products as

SQL

SELECT

product\_id,

INITCAP(product\_name) AS product\_name\_cleaned, -- 8. Standardize product names (capitalize properly)

INITCAP(category) AS category\_cleaned, -- 8. Standardize category names (Title Case)

-- 8. Handle inactive products

CASE

WHEN UPPER(active\_flag) = 'N' THEN 'Discontinued Product'

ELSE INITCAP(product\_name)

END AS product\_status,

active\_flag

FROM products\_raw

**Model 3: (Task 7)**

**Purpose:** Remove duplicates, handle negative/null (using ), standardize and derive

SQL

WITH CleanedOrders AS (

-- 7. Remove exact duplicate rows and standardize currency

SELECT DISTINCT

order\_id,

customer\_id,

product\_id,

amount::NUMERIC(10, 2) AS amount\_raw,

created\_at,

UPPER(currency) AS currency\_cleaned

FROM orders\_raw

),

MedianAmount AS (

-- CORRECTED SNOWFLAKE SYNTAX: Calculate the median amount per customer for <null> fallback

SELECT

customer\_id,

-- Use simple MEDIAN() function partitioned by customer\_id

MEDIAN(amount\_raw) OVER (PARTITION BY customer\_id) AS customer\_median\_amount

FROM CleanedOrders

WHERE amount\_raw IS NOT NULL AND amount\_raw >= 0

GROUP BY customer\_id, amount\_raw

)

SELECT

t1.order\_id,

t1.customer\_id,

t1.product\_id,

t1.created\_at,

t1.currency\_cleaned,

-- 7. Handle invalid amounts

CASE

WHEN t1.amount\_raw < 0 THEN 0.00

-- Fallback to 0 if median not found

WHEN t1.amount\_raw IS NULL THEN COALESCE(t2.customer\_median\_amount, 0.00)

ELSE t1.amount\_raw

END AS amount\_cleaned,

-- 7. Create derived column amount\_usd (Conversion rates used: INR=0.012, SGD=0.74, EUR=1.07)

(amount\_cleaned \* CASE t1.currency\_cleaned

WHEN 'INR' THEN 0.012

WHEN 'SGD' THEN 0.74

WHEN 'EUR' THEN 1.07

WHEN 'USD' THEN 1.00

ELSE NULL

END

) AS amount\_usd

FROM CleanedOrders t1

LEFT JOIN MedianAmount t2 ON t1.customer\_id = t2.customer\_id

**Model 4: (Tasks 9 & 10)**

**Purpose:** Produce the final unified dataset, joining all three cleaned models and handling all edge cases ().

SQL

SELECT

-- ORDER DETAILS

o.order\_id,

o.created\_at AS order\_date,

o.amount\_cleaned,

o.currency\_cleaned,

o.amount\_usd,

-- CUSTOMER DETAILS (Handle Orphan and Invalid Customers)

o.customer\_id,

CASE

-- 10. Edge Case: Mark completely null customers (Customer ID 108 in sample data)

WHEN o.customer\_id = 108 THEN 'Invalid Customer'

-- 9. Orphan Customer Check (If order exists but clean customer record does not)

WHEN c.customer\_id IS NULL THEN 'Orphan Customer'

ELSE c.email\_cleaned

END AS customer\_email\_final,

c.phone\_cleaned,

c.country\_code\_cleaned,

c.created\_at\_cleaned AS customer\_created\_at,

-- PRODUCT DETAILS (Handle Unknown/Discontinued Products)

o.product\_id,

CASE

-- 9. Check for Missing/Invalid Product ID or Discontinued Product

WHEN p.product\_id IS NULL OR p.product\_status = 'Discontinued Product' THEN 'Unknown/Discontinued Product'

ELSE p.product\_name\_cleaned

END AS product\_name\_final,

p.category\_cleaned

FROM ORDERS\_CLEANED o

-- LEFT JOIN ensures all orders appear, satisfying the requirement for orphaned orders (Task 10)

LEFT JOIN CUSTOMERS\_CLEANED c

ON o.customer\_id = c.customer\_id

LEFT JOIN PRODUCTS\_CLEANED p

ON o.product\_id = p.product\_id

ORDER BY o.order\_id

**A screenshot of a email

AI-generated content may be incorrect.**

**All Models  
  
  
  
A screenshot of a computer

AI-generated content may be incorrect.**