# Hevo Data Assessment – End-to-End Data Pipeline (PostgreSQL → Hevo → Snowflake)

This project demonstrates the setup of a complete data pipeline using **PostgreSQL (Neon)**, **Hevo Data**, and **Snowflake**.  
It walks through the creation of tables, data ingestion, transformation, and validation from source to destination.

## Overview

The goal was to:

* Set up a Snowflake trial account and connect it with a Hevo partner account.
* Establish a working PostgreSQL environment.
* Ingest data from CSV files into PostgreSQL.
* Connect PostgreSQL to Hevo as a source.
* Apply transformations using Python in Hevo.
* Load transformed data into Snowflake.
* Validate the final results using SQL queries.

## 🏁 **Step 1: Destination Setup (Snowflake Partner Connect)**

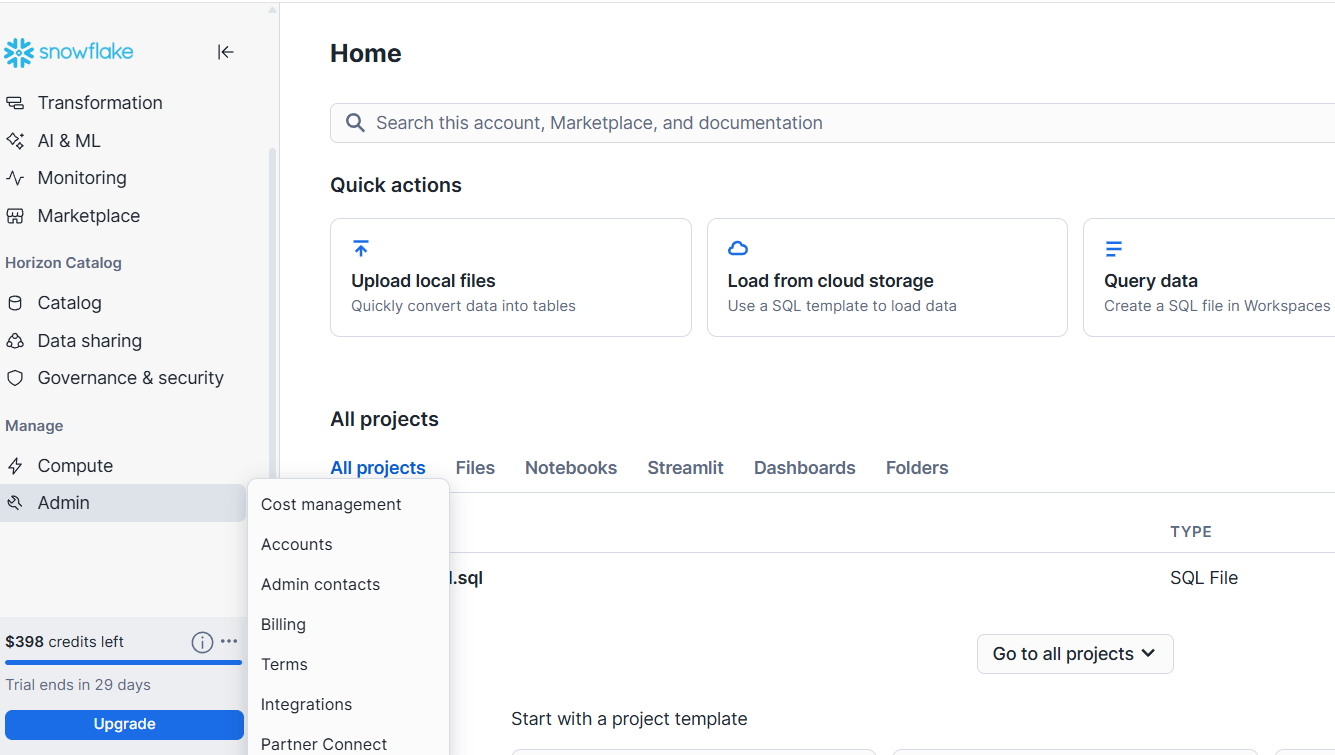
### 1️⃣ **Snowflake Trial Account Creation**

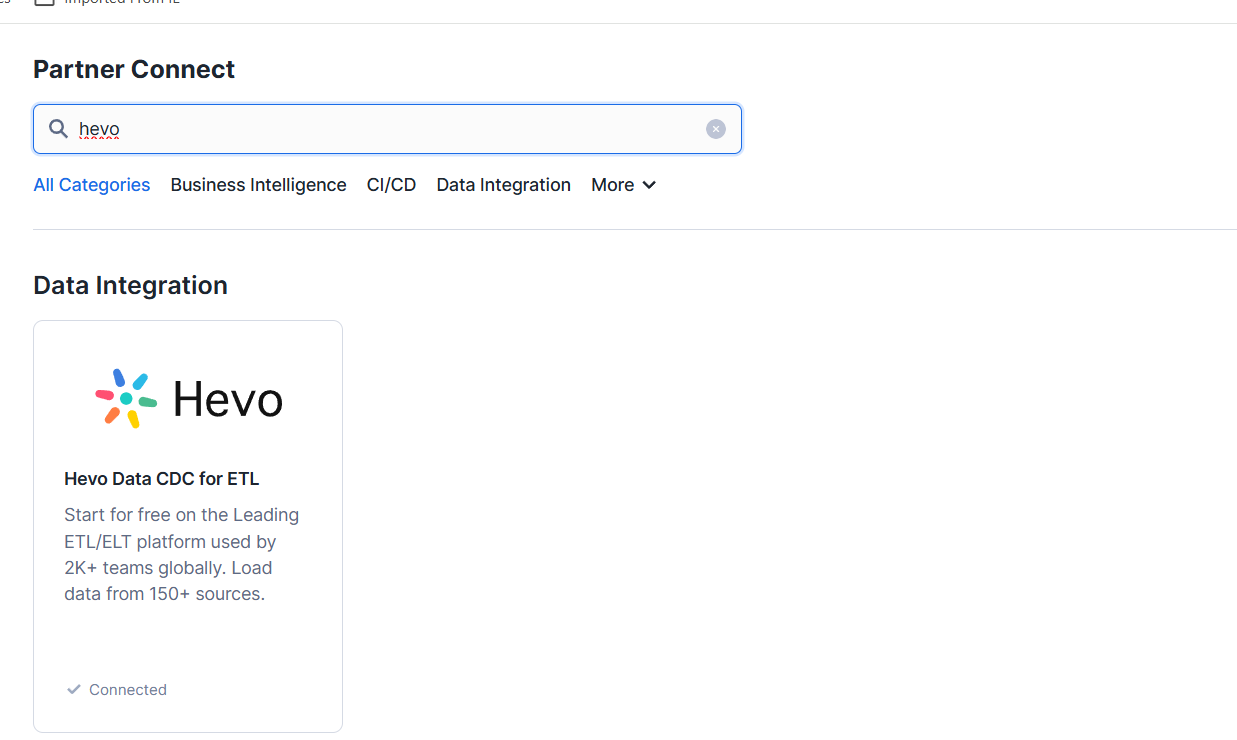
* Created a **Snowflake trial account** using Gmail.
* Logged in as admin

### 2️⃣ **Hevo Partner Account Setup**

* Inside Snowflake, navigated to **Admin → Partner Connect**.
* Selected **Hevo Data** from the list of partners.
* Authorized connection → Hevo workspace auto provisioned.

✅ Hevo connected to Snowflake successfully.





## **Step: Initial Setup Attempt with Docker (and Why It Was Replaced)**

### 1️⃣ **Docker Setup**

* Installed **Docker Desktop** and attempted to create a local PostgreSQL container.
* Verified installation with:
* docker version

✅ The command returned version details successfully.

### 2️⃣ **Issue Encountered**

While trying to launch or connect to the PostgreSQL container, Docker repeatedly threw a **500 API Error**, preventing connection.

### 3️⃣ **Decision**

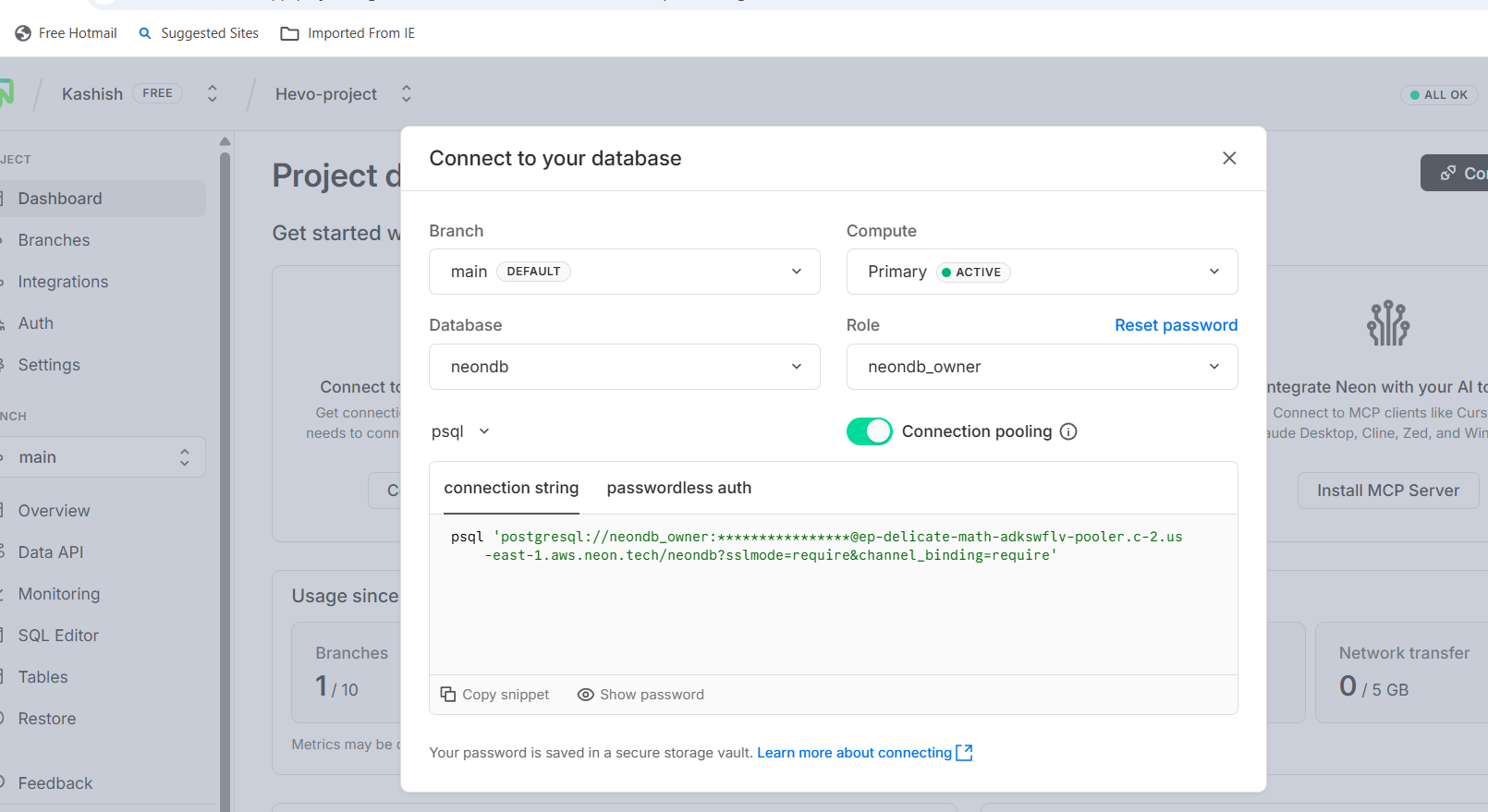
Since Docker setup was unstable, switched to [**Neon.tech**](https://neon.tech/) — a managed PostgreSQL cloud service with a free tier, GUI, and logical replication support.

## 🧱 **Step 2: PostgreSQL Setup in Neon**

### **Created Neon Account by using URL**

https://console.neon.tech/app/org-mute-band-50763817/projects

* Signed up using Gmail.
* Created a **Neon project and database** named - neondb.
* Default role: neondb\_owner. As Attached below



✅ Connection successful.

## **Step 3: Connection via pgAdmin**

Created a new server in pgAdmin named **neondb** with the following credentials:

| **Parameter** | **Value** |
| --- | --- |
| Hostname | ep-delicate-math-adkswflv-pooler.c-2.us-east-1.aws.neon.tech |
| Database | neondb |
| User | neondb\_owner |
| Password | \*\*\*\*\*\*\*\* |

Connection successful ✅

Note - Where hostname is from Neon connection sting and user too from there  
  
Screenshot attached   
  
How to create new server

A screenshot of a computer

AI-generated content may be incorrect.  
  
  
After server is enter you got below interface you need to put all the details as mentioned above like hostname username password

A screenshot of a computer

AI-generated content may be incorrect.  
  
  
  
  
Once the connection was made I created 3 tables using the create table command under pgadmin sql query and once all 3 tables created

## 🧮 **Step 4: Table Creation and Data Import under PGADMIN**

### **1. Create Tables**

Ran the following SQL under pgAdmin:

CREATE TABLE customers (

id SERIAL PRIMARY KEY,

first\_name VARCHAR(50),

last\_name VARCHAR(50),

email VARCHAR(100),

address JSON

);

CREATE TABLE orders (

id SERIAL PRIMARY KEY,

customer\_id INT,

status VARCHAR(50),

total\_amount NUMERIC(10,2)

);

CREATE TABLE feedback (

order\_id INT PRIMARY KEY,

rating INT,

comment TEXT

);

### **2. Import CSV Data**

Downloaded the CSV files from the [Hevo Assessment Repository](https://github.com/muskan-kesharwani-hevo/hevo-assessment-csv):  
  
https://github.com/muskan-kesharwani-hevo/hevo-assessment-csv

* customers.csv
* orders.csv
* feedback.csv

Imported them into pgAdmin by right-clicking each table → Import/Export Data.

#### ⚠️ **Issue 1: Invalid JSON Format**

During import of customers.csv, received error:

invalid input syntax for type json

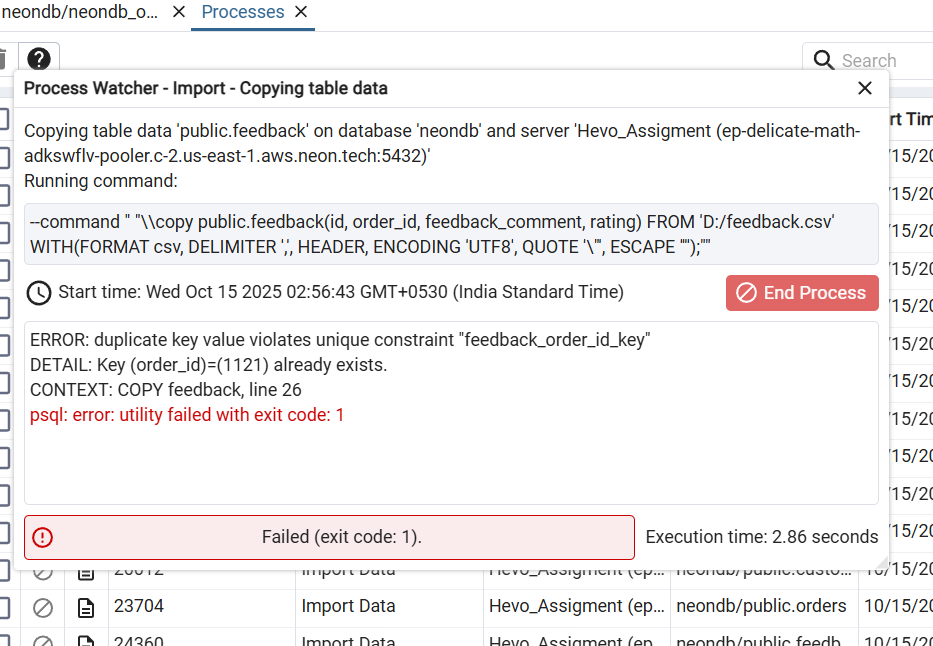
**Root Cause:**  
Values in the address column were not enclosed in double quotes.

**Fix:**  
Changed datatype from JSON → TEXT:

ALTER TABLE customers ALTER COLUMN address TYPE TEXT USING address::TEXT;

Re-imported successfully ✅

#### ⚠️ **Issue 2: Duplicate Key Violation**

While importing feedback.csv, error:  
  
  


ERROR: duplicate key value violates unique constraint "feedback\_order\_id\_key"

DETAIL: Key (order\_id)=(1121) already exists.

**Root Cause:**  
133 duplicate order\_id values in the CSV.

**Resolution:**

Used Python (pandas) to remove duplicates and create a cleaned file:

import pandas as pd

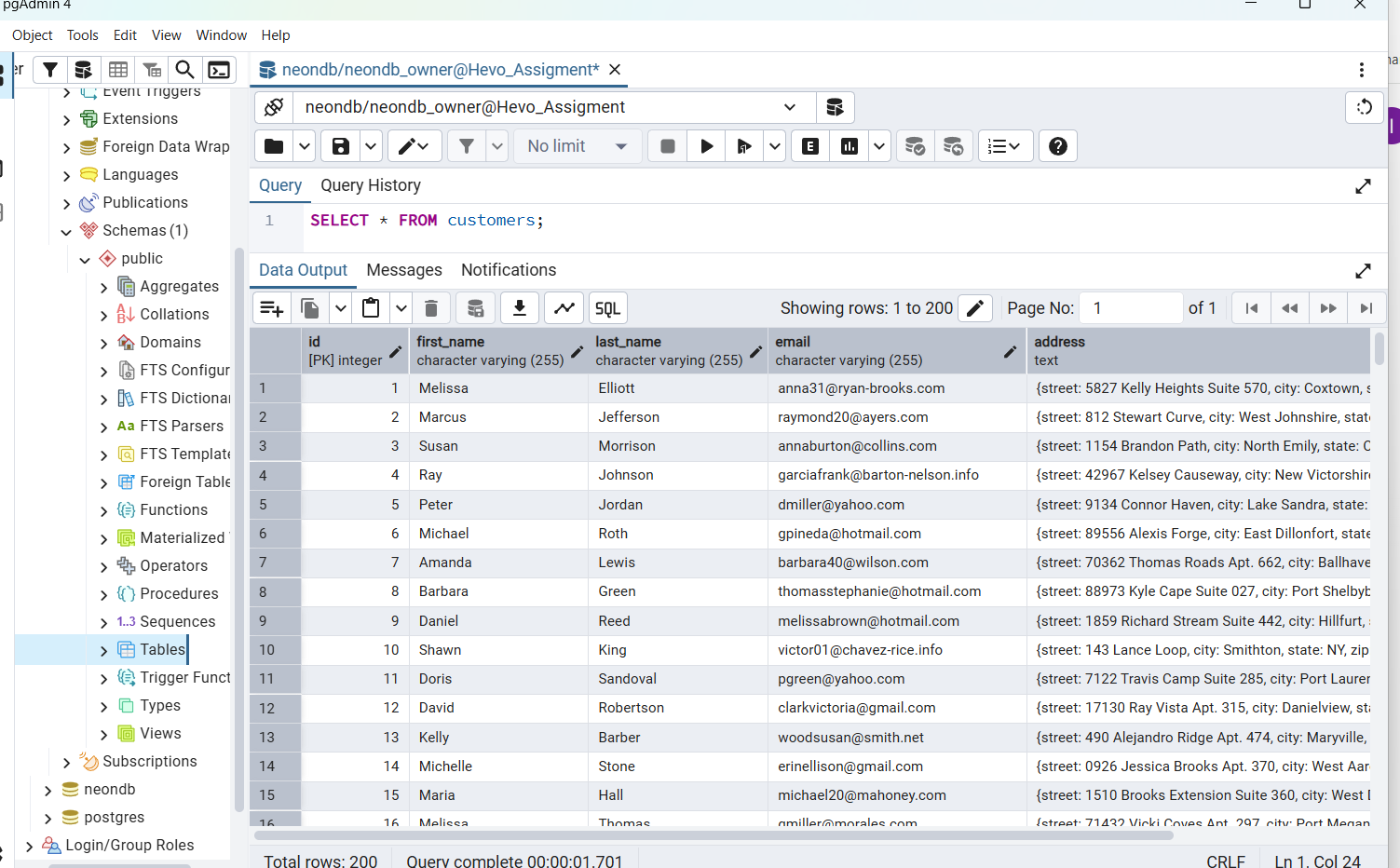
df = pd.read\_csv('feedback.csv')

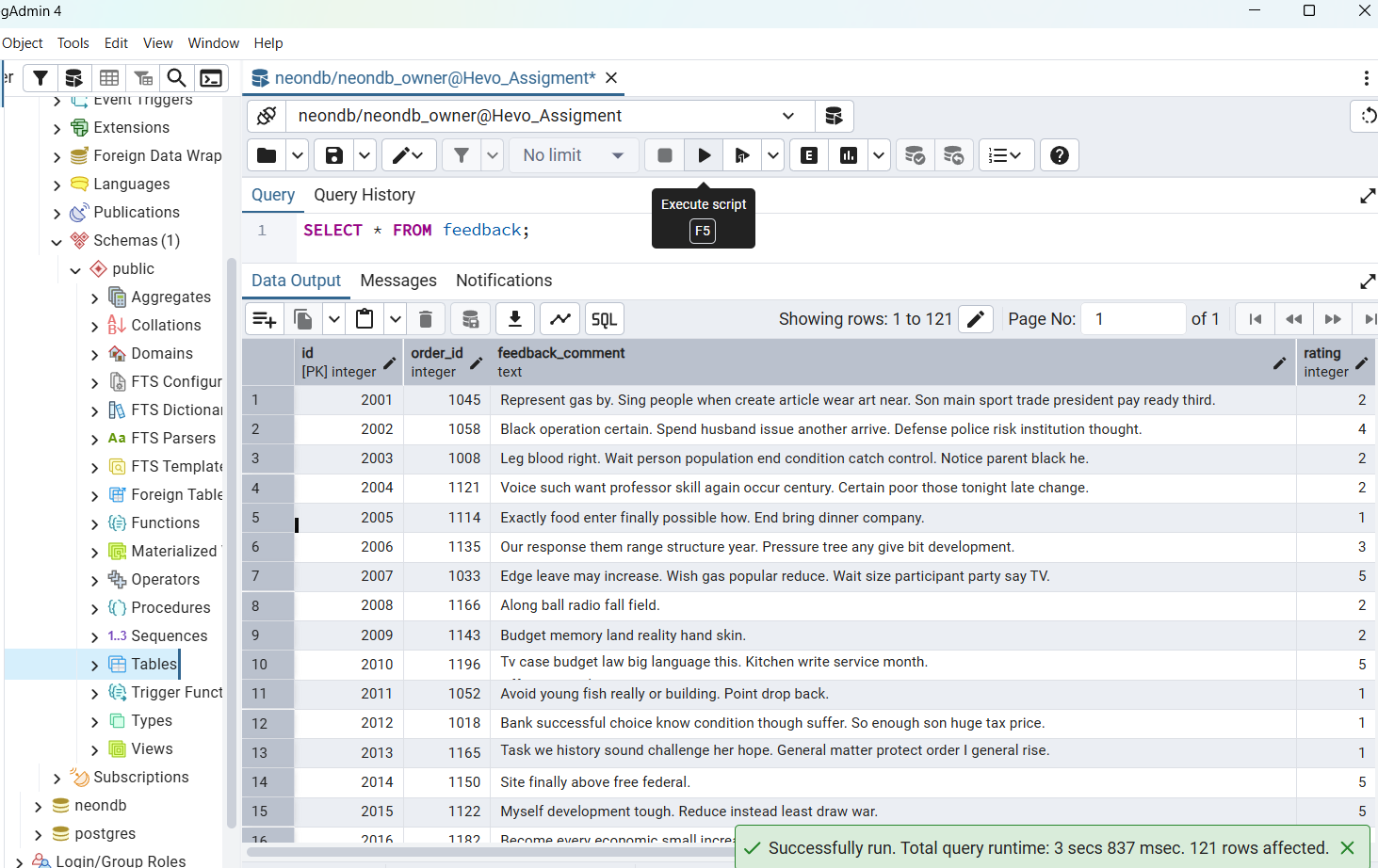
df = df.drop\_duplicates(subset='order\_id', keep='first')

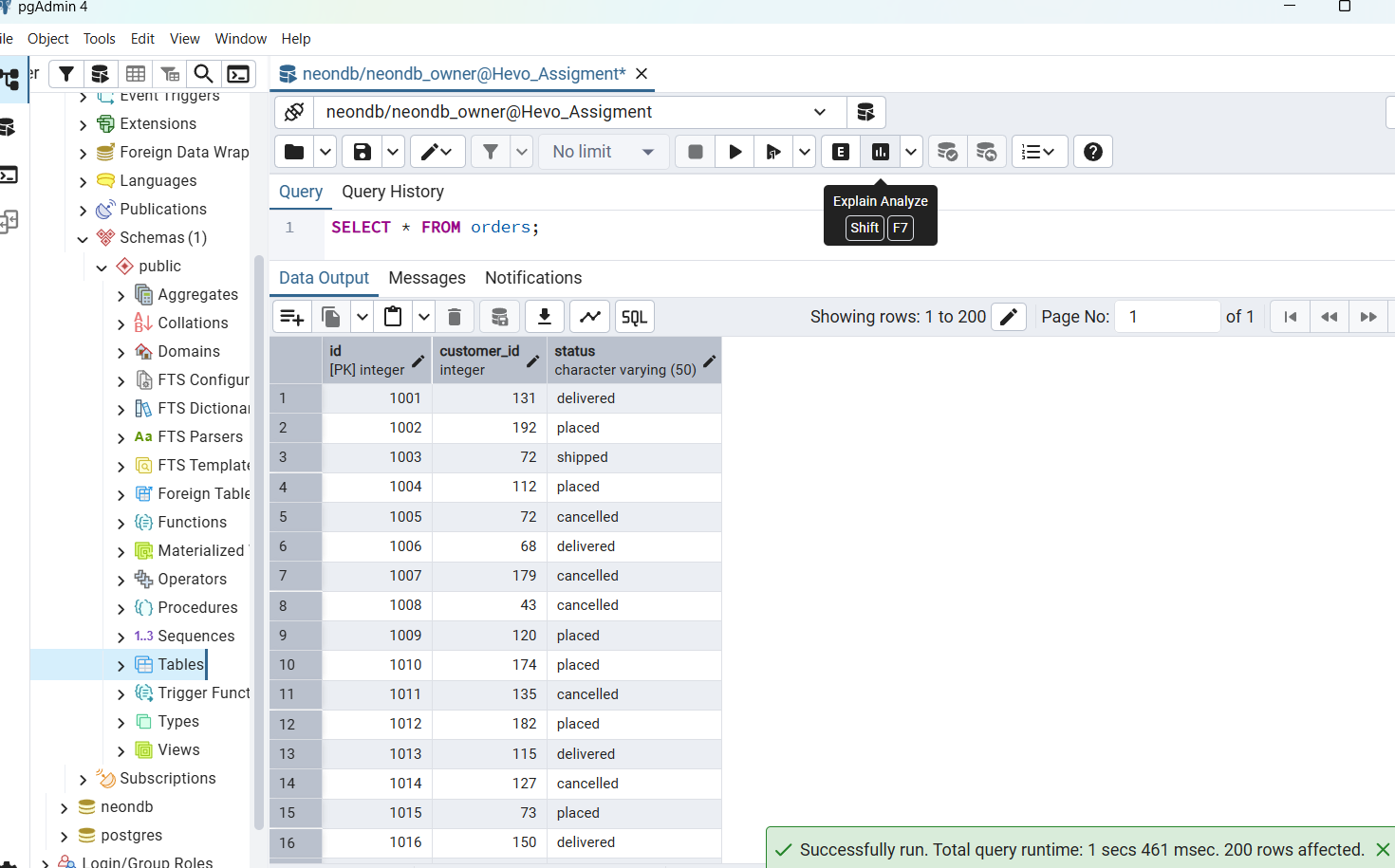
df.to\_csv('feedback\_clean.csv', index=False)

Re-imported feedback\_clean.csv successfully ✅

So, this way I was able to bring all 3 tables with all rows inside it







## **Step 5: Connect PostgreSQL to Hevo**

### **1. Created New Hevo Pipeline**

* Source: **PostgreSQL**
* Destination: **Snowflake**

### 2. **PostgreSQL Source Settings** 1. Now I moved to hevo platform side to Configure PostgreSQL Source in Hevo 2. Log in to your Hevo account. 3. Create a new pipeline. Select PostgreSQL as the Source 4. And the Ui seeking some fields like pipeline named database name host 5. I used here the same database host and user and password same as neon side

| **Parameter** | **Value** |
| --- | --- |
| Host | ep-delicate-math-adkswflv-pooler.c-2.us-east-1.aws.neon.tech |
| Database | neondb |
| User | neondb\_owner |
| Password | \*\*\*\* |
|  |  |

### Here below we have used the parameter values describes above

### **3. Error Encountered** After entering all the required values and clicked in save and continued

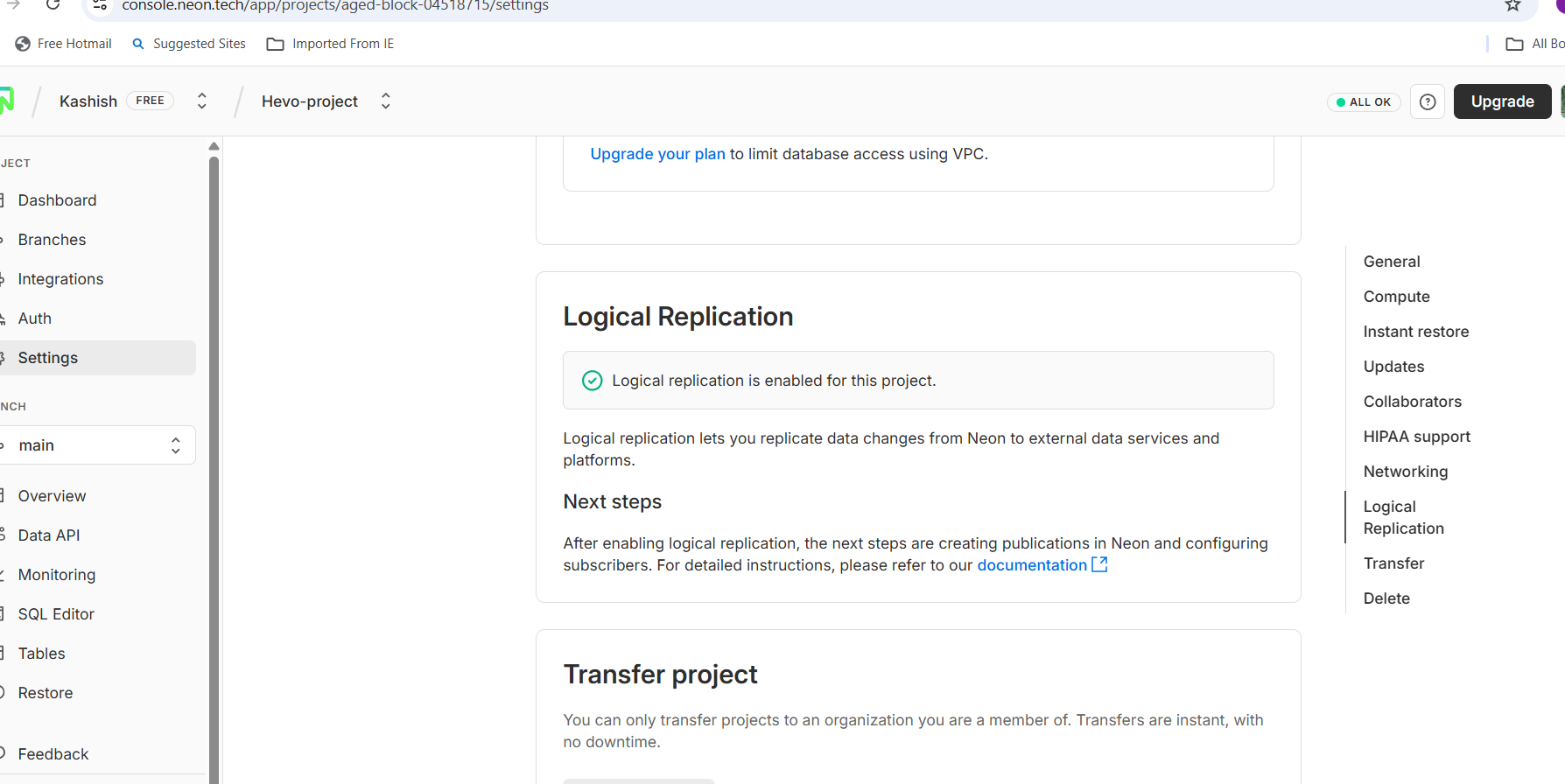
Hevo showed:

Error - Unable to use logical replication. wal\_level must be set to 'logical'

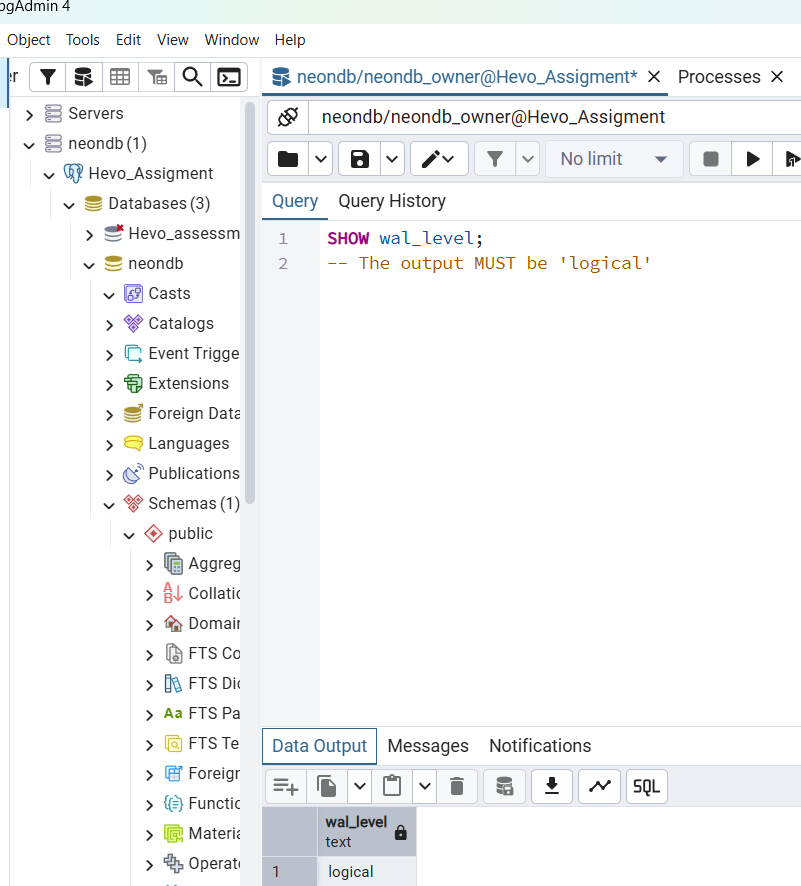
### **4. Fix**

* Logged in to **Neon → Settings → Enable Logical Replication**
* Verified using:
* SHOW wal\_level;

Output confirmed: logical

Reconnected successfully 🎯  
  
  
  
  
Neon side replication enable

Verified if enabled replication



Now go and save and continue and it works it populate you all columns and later last stage ask below info

## 🧭 Step 5: Configure Hevo Pipeline

| **Setting** | **Value** |
| --- | --- |
| Destination Table Prefix | (Blank) |
| Auto Mapping | Enabled |
| Ingestion Schedule | Every 30 minutes |

Once configured, tables from PostgreSQL successfully appeared in Hevo.

A screenshot of a computer

AI-generated content may be incorrect.

## **🧠 Step 6: Data Transformation in Hevo**

Used a **Python transformation script** to enrich and restructure data.

### **🔹 Transformation Goals**

1. Derive username from email in **customers**.
2. Create a new derived table **order\_events** based on order status.
3. Pass other tables unchanged.

### **🔹 Transformation Script**

In hevo transform go to transform and create a new transform by below python script

import datetime

def transform(event):

table\_name = event.get('\_\_table\_name')

if not table\_name:

return event, None

# 1️⃣ Add username for customers

if table\_name == 'customers':

email = event.get('email')

if email and '@' in email:

event['username'] = email.split('@')[0]

return event, table\_name

# 2️⃣ Generate order\_events for orders

elif table\_name == 'orders':

status\_map = {

'delivered': 'order\_delivered',

'placed': 'order\_placed',

'shipped': 'order\_shipped',

'cancelled': 'order\_cancelled'

}

status = event.get('status')

output\_records = [(event, table\_name)]

if status in status\_map:

new\_event\_record = {

'order\_id': event.get('id'),

'customer\_id': event.get('customer\_id'),

'event\_type': status\_map[status],

'event\_timestamp': datetime.datetime.now().isoformat()

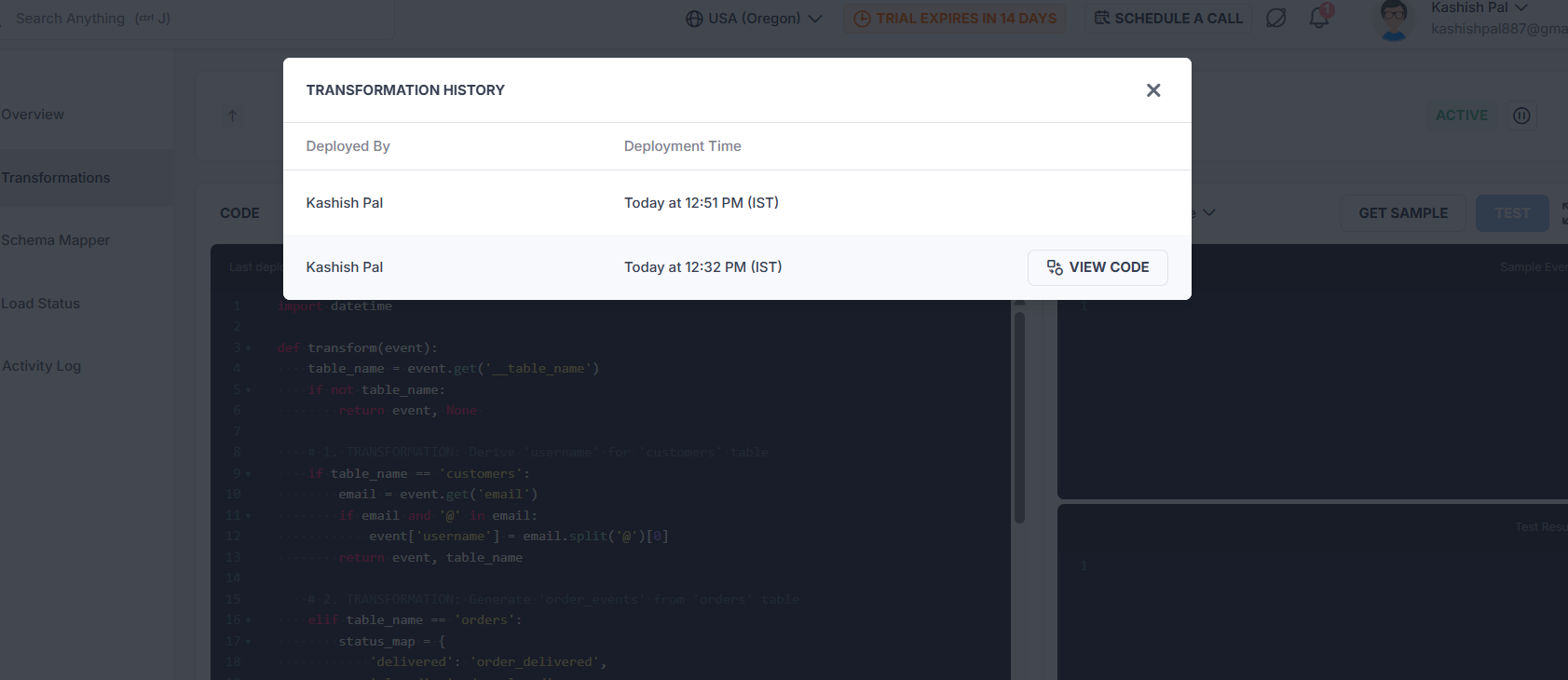
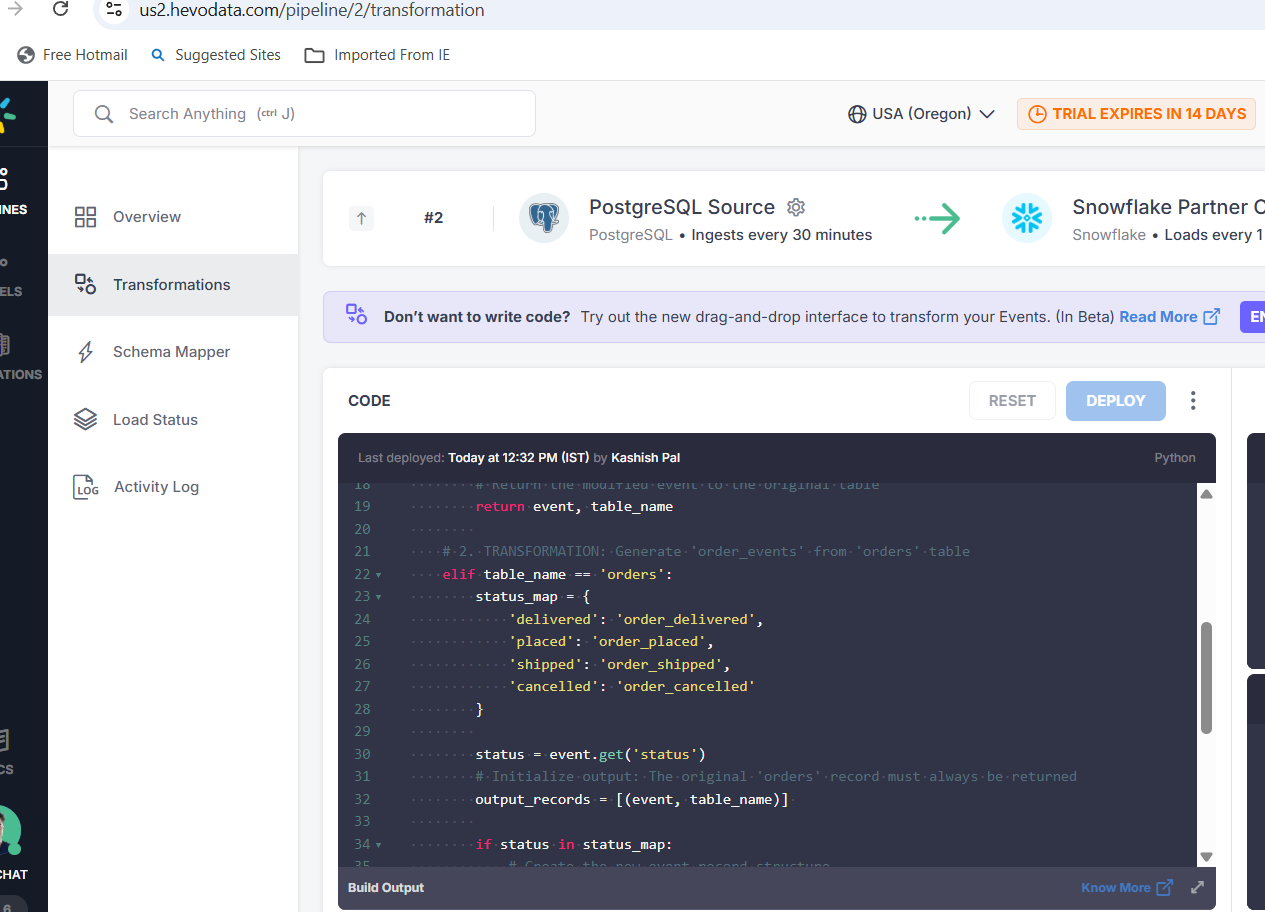
}

output\_records.append((new\_event\_record, 'order\_events'))

return output\_records

# 3️⃣ Pass feedback and other tables

return event, table\_name

Transformation deployed successfully ✅  
  
  
  
  


## **❄️ Step 7: Load and Validate in Snowflake**

### **Snowflake Connection**

* Database: PC\_HEVODATA\_DB
* Schema: PUBLIC

### **SQL Operations**

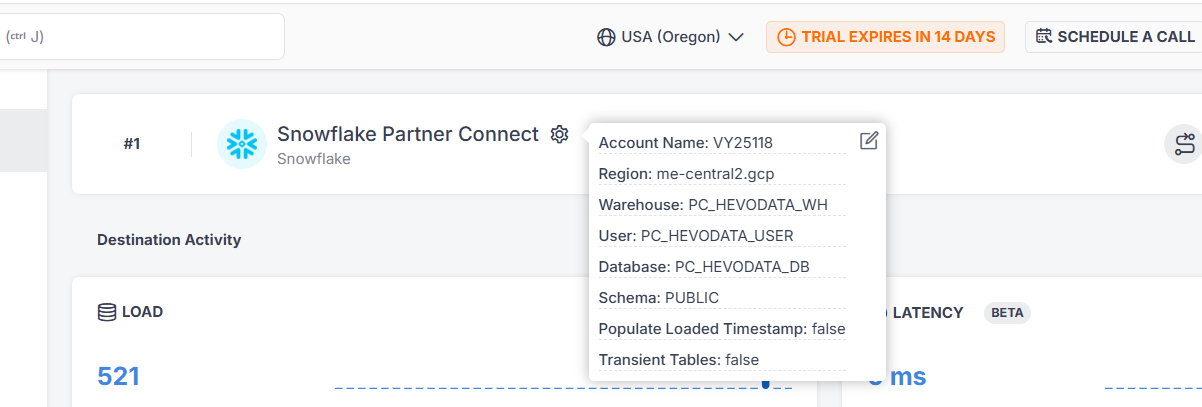
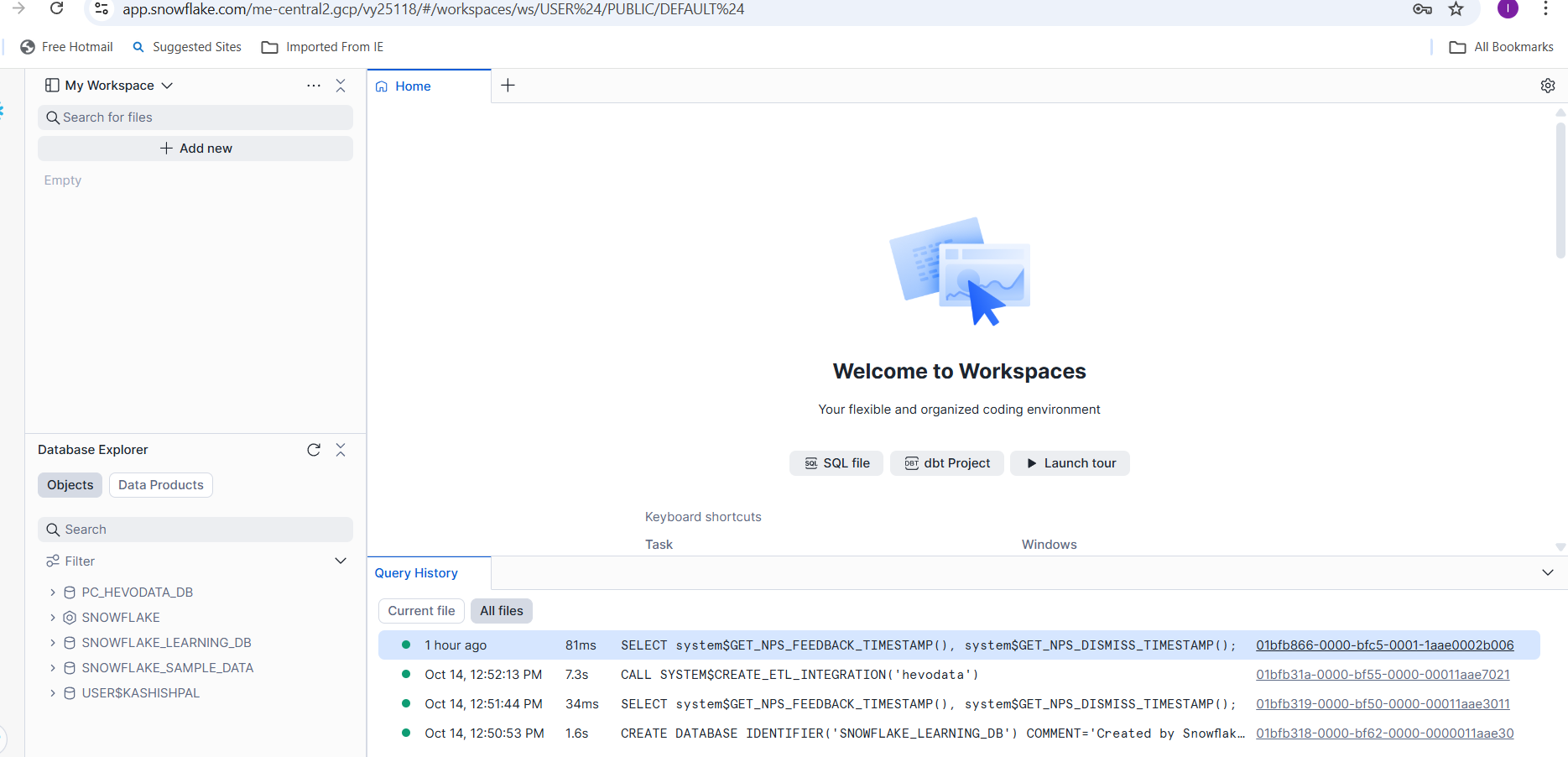
**Add new column**

ALTER TABLE PC\_HEVODATA\_DB.PUBLIC.CUSTOMERS

ADD COLUMN USERNAME VARCHAR;

**Populate username**

UPDATE PC\_HEVODATA\_DB.PUBLIC.CUSTOMERS

SET USERNAME = SPLIT\_PART(EMAIL, '@', 1);  
  
  
  
  
  


## **🧾 Step 8: Validation Queries**

### **✅ Validation A — Order Events**

SELECT EVENT\_TYPE, COUNT(\*)

FROM PC\_HEVODATA\_DB.PUBLIC.ORDER\_EVENTS

GROUP BY EVENT\_TYPE;

### **✅ Validation B — Username Field**

SELECT EMAIL, USERNAME, FIRST\_NAME

FROM PC\_HEVODATA\_DB.PUBLIC.CUSTOMERS

WHERE USERNAME IS NOT NULL

LIMIT 10;

All validation checks passed successfully 🎉  
  
A screenshot of a computer

AI-generated content may be incorrect.  
**Validation A — Order Events**  
  
  
A screenshot of a computer

AI-generated content may be incorrect.

### **Validation B — Username Field**

## **💡 Issue encountered & Design Choices**

| **Category** | **Decision / Assumption** |
| --- | --- |
| **Address Field** | Converted from JSON → TEXT for simplicity |
| **Order Status Type** | Used VARCHAR instead of ENUM for flexibility |
| **Duplicate Handling** | Removed duplicates at source before loading |
| **Transformation Type** | Used Python for precise control |
| **Ingestion Frequency** | 30-minute schedule chosen for steady sync |

## **📁 Repository Structure**

📦 hevo-data-assessment

│

├── README.md

├── sql/

│ ├── create\_tables.sql

│ ├── validation\_queries.sql

│

├── data/

│ ├── customers.csv

│ ├── orders.csv

│ ├── feedback\_clean.csv

│

├── transform/

│ └── transform\_script.py

## **🧩 Summary**

✅ Attempted PostgreSQL via Docker (failed due to API 500 error)  
✅ Switched to Neon PostgreSQL (successful)  
✅ Created & populated tables  
✅ Connected Neon PostgreSQL → Hevo with logical replication  
✅ Applied transformations using Python  
✅ Synced data to Snowflake  
✅ Verified successful transformation and data integrity