Northeastern University
College *of* Engineering

Data Warehousing & Integration
IE 6750
FALL 2024

FINAL REPORT

Group 01

Layashree Adepu

Pavithra Moorthy

Adepu.l@northeastern.edu

Moorthy.p@northeastern.edu

Submission Date: 12/07/2024

BIKE FLOW ANALYTICS

Problem Statement:
The Bay Wheels bike-sharing service faces challenges in maintaining bike availability across its network of stations. Some stations frequently run out of bikes, while others have too many, leading to user dissatisfaction and inefficiencies in the system. Currently, the service does not have a consolidated way to analyse key factors like ride patterns, peak times, and user behaviour to better manage bike distribution.

To address this, we will create a centralised on-premise database that collects trip data, station usage, and user information. This will help the company analyse patterns, improve bike reallocation, and optimize station management, ensuring bikes are available when and where they are needed most.

Database Overview:
The operational database for the Bay Wheels bike-sharing service is designed to collect and organize key information from every trip, allowing for detailed analysis and decision-making.

Each column in the database captures specific aspects of the bike-sharing system, which can be leveraged to optimize operations and improve user experience

Columns:
• duration_sec: Total trip duration in seconds, useful for analyzing ride patterns and trip lengths.
• start_time / end_time: Start and end timestamps for each trip, essential for identifying peak usage periods.
• start_station_id / start_station_name: Unique ID and name of the station where the trip starts, used for tracking station popularity.
• start_station_latitude / start_station_longitude: Geographic coordinates of the start station, important for location-based analysis.
• end_station_id / end_station_name: Unique ID and name of the station where the trip ends, helping to analyze route patterns and station demand.
• end_station_latitude / end_station_longitude: Geographic coordinates of the end station, useful for geospatial studies.
• bike_id: Unique identifier for each bike, allowing for maintenance tracking and usage patterns.
• user_type: Indicates whether the rider is a Subscriber (member) or Customer (casual user), for behavior analysis.
• bike_share_for_all_trip: Identifies trips under the Bike Share for All program, important for evaluating program participation.

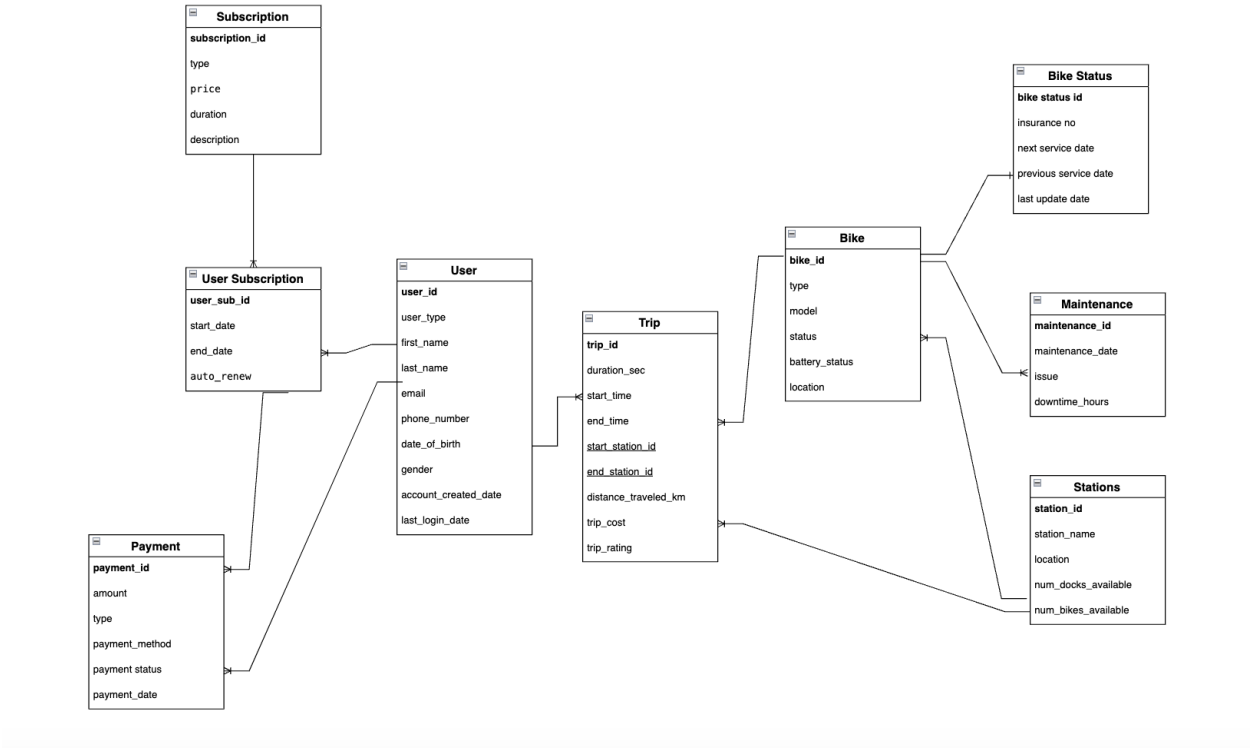
We have an extensive dataset that covers key aspects such as trip duration, start and end times, station details, bike IDs, and user types. To enhance the analysis and provide more insights, we plan to add additional columns like user details (e.g., age group and subscription type), distance traveled (calculated between start and end stations). These new columns will be generated or calculated based on existing data, allowing for deeper analysis of user behaviour, trip patterns, and operational efficiencies within the bike-sharing system.

What We Hope to Analyze:
In the future, we hope to analyze user behaviour patterns to gain insights into rental trends, peak usage times, and demographic preferences for Lyft Bikes. This analysis will help optimize fleet management by assessing bike distribution and maintenance needs, ensuring availability aligns with demand.

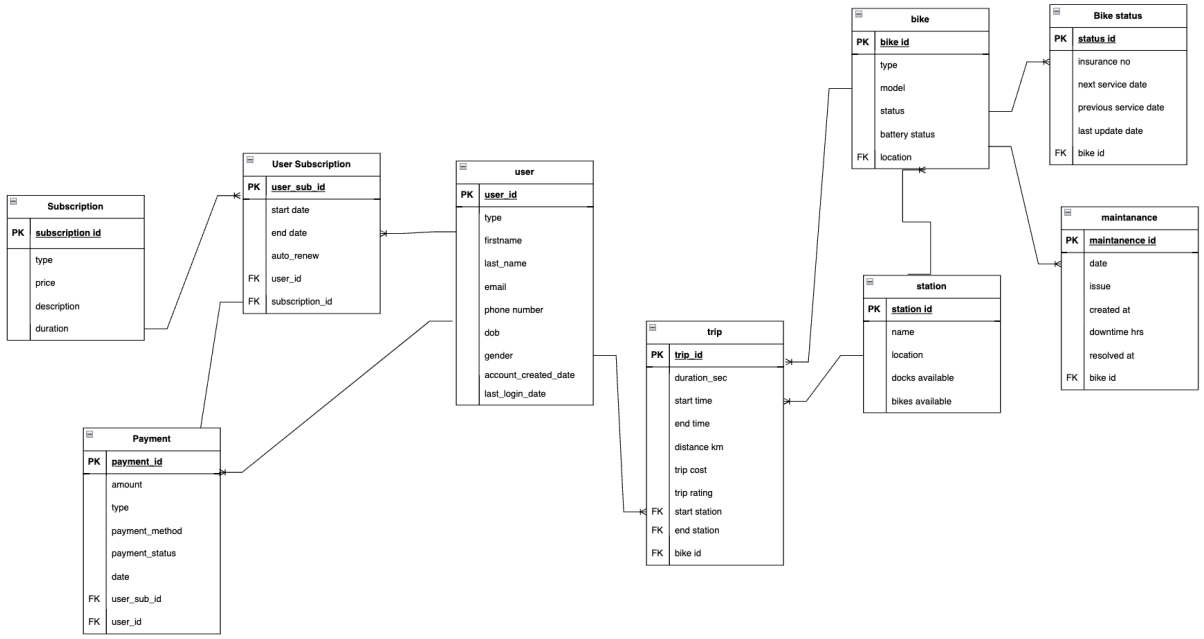
Additionally, I aim to evaluate the effectiveness of integration with public transportation systems, measuring the impact on rental rates and user satisfaction. Overall, these insights will guide improvements in service delivery and support Lyft's commitment to enhancing urban mobility.

• **Optimize Bike Reallocation:** Predict high-demand stations and recommend rebalancing of bikes across stations.
• **Peak Time Analysis:** Identify the busiest times of day/week to optimize bike and station management.
• **User Behaviour:** Analyse differences between casual users and subscribers to tailor marketing or service strategies.
• **Station Usage Efficiency:** Evaluate which stations are under- or over-utilized and plan for capacity adjustments.

ERD:



Logical Model:



Data Population Methodology

1. **Data set upload as a csv file for trip table from Kaggle (90000 rows)**
2. **For other tables - https://www.mockaroo.com**

1. Dimensions

a. Bike
b. Station
c. User
d. Subscription

2. Hierarchies

a. Date/Time Dimension

Hierarchy:

- Year → Quarter → Month → Day
- Year → Month → Week → Day → Hour

b. Location Dimension

Hierarchy:

- Country → State → City → Zipcode
- Country → Region → City → Zipcode

3. Measures

a. Trip Measures

- **Trip Count**: Total number of trips
- **Average Trip Duration**: Sum of all trip durations / Trip Count
- **Total Distance Traveled**: Sum of all distances for each trip
- **Trip Cost**: Sum of costs associated with all trips

- **Average Trip Rating**: Average of all ratings given for trips
- **Trips by Time of Day**: Number of trips during different parts of the day (morning, afternoon, evening, night)

b. User Measures

- **User Count**: Total number of unique users
- **New Users**: Number of users who signed up in a specific time period
- **Retention Rate**: Percentage of users who continue to use the service over a period of time
- **Average Subscription Length**: Average duration of active subscriptions per user
- **User Activity**: Count of trips per user over time

c. Bike Measures

- **Bike Utilization Rate**: Percentage of time a bike is in use vs. available
- **Maintenance Frequency**: Average number of maintenance events per bike
- **Mileage per Bike**: Total distance traveled by each bike
- **Downtime**: Total time each bike is out of service

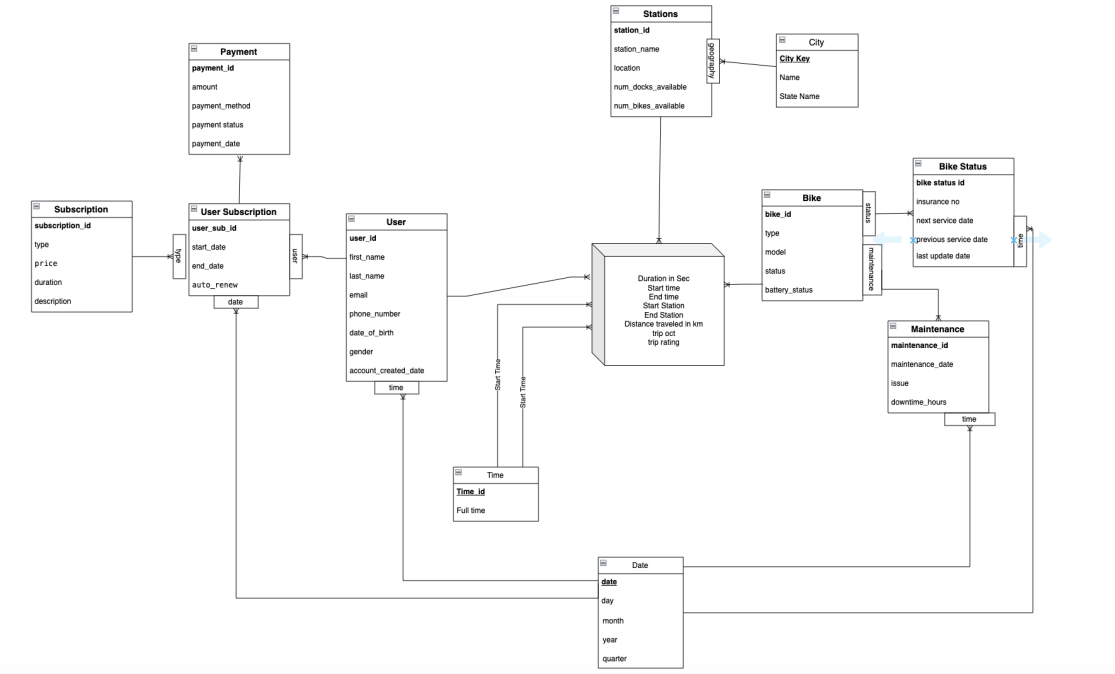
d. Station Measures

- **Station Usage**: Number of trips starting or ending at each station
- **Capacity Utilization**: Average percentage of docks occupied at a station
- **Peak Hours**: Most frequently used hours at a station
- **Top Stations**: Stations with the highest number of rentals or returns

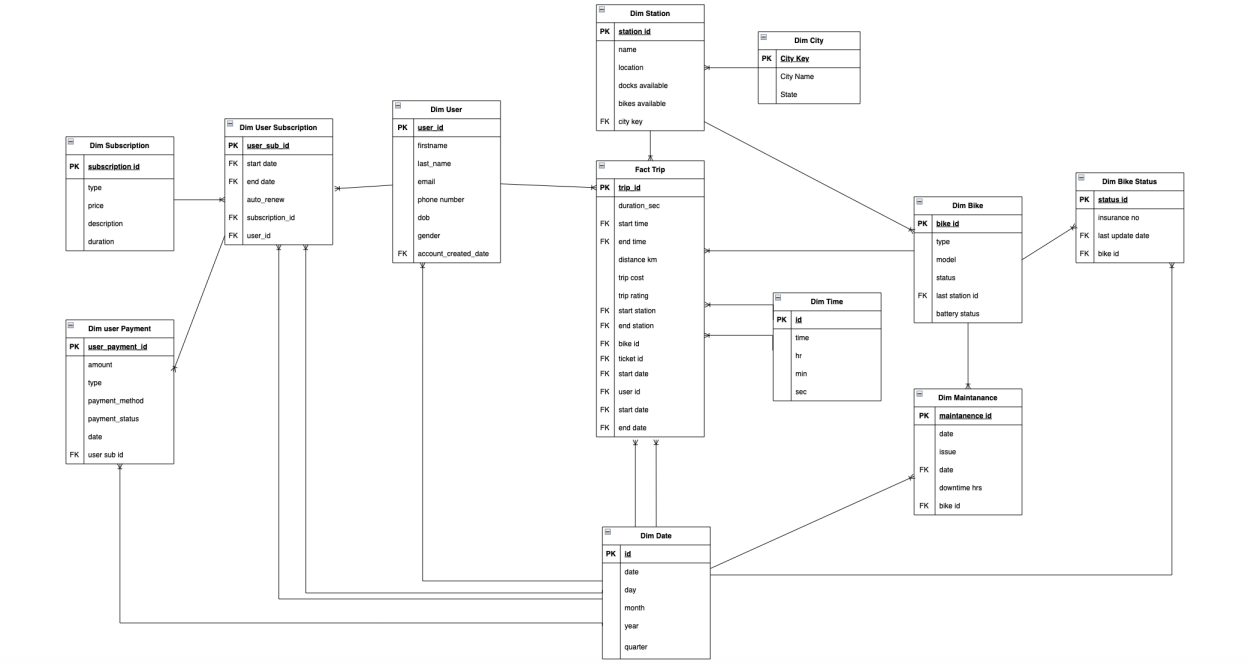
e. Financial Measures (Payments)

- **Total Revenue**: Sum of all payments
- **Average Payment per Trip**: Total Revenue / Trip Count
- **Subscription Revenue**: Total revenue from subscriptions

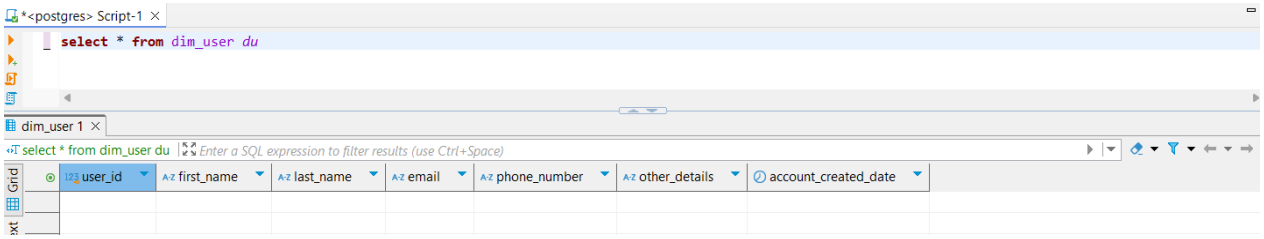
Dimensional Conceptual Model:

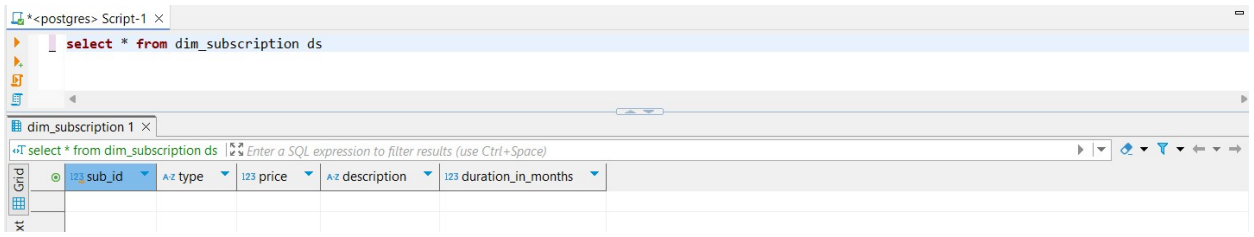


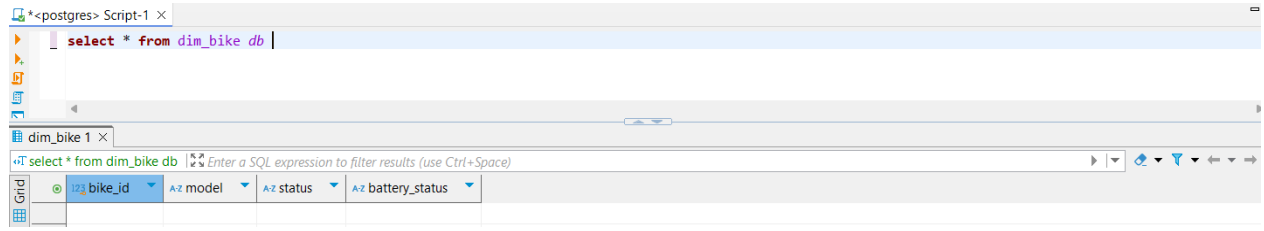
Dimensional Logical Model:

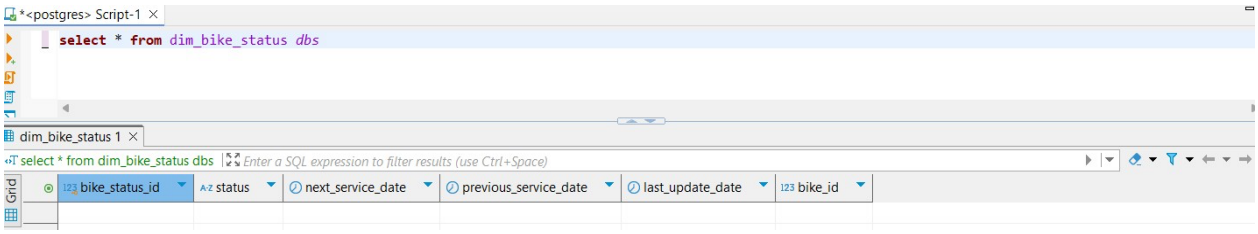


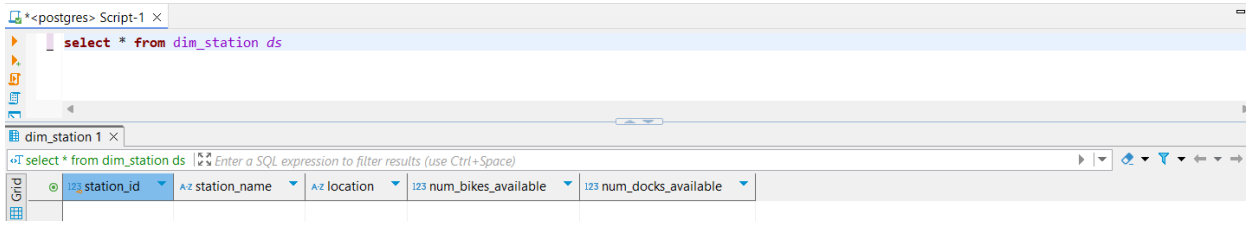
Implementation:

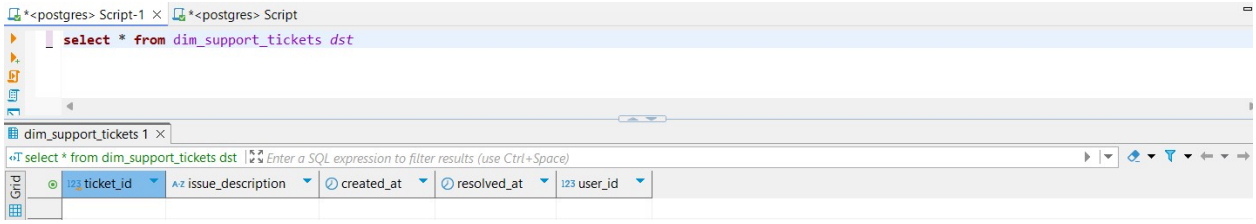


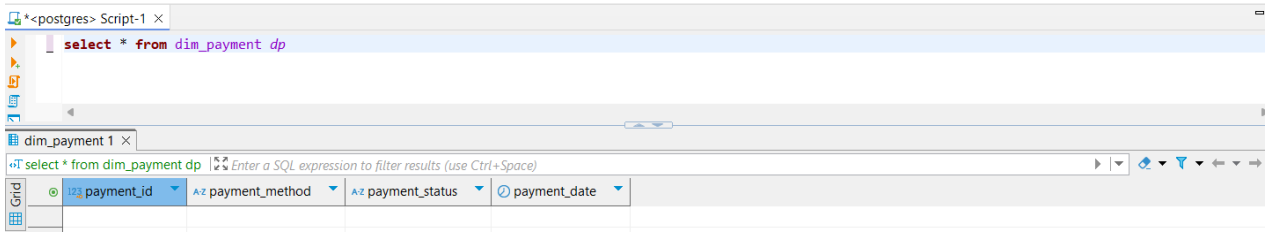


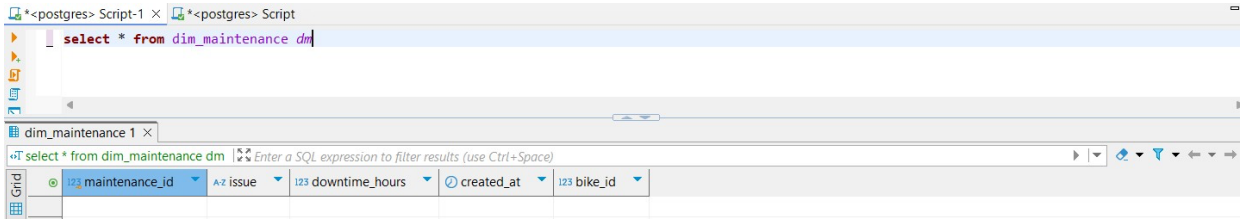


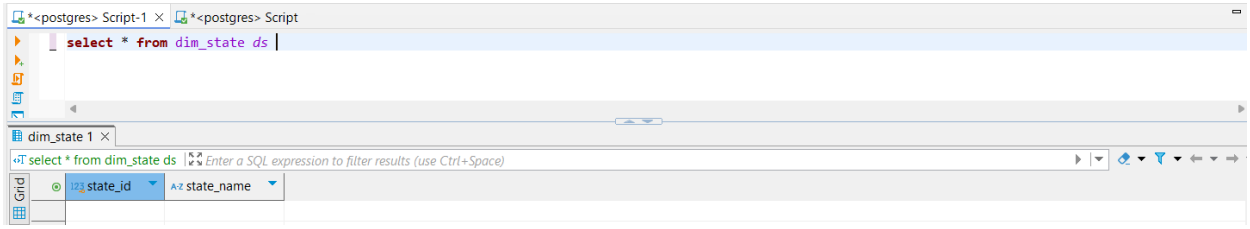

*<postgres> Script-1
select * from dim_bike_status dbs
dim_bike_status 1
select * from dim_bike_status dbs
Enter a SQL expression to filter results (use Ctrl+Space)
Grid
bike_status_id
status
next_service_date
previous_service_date
last_update_date
bike_id

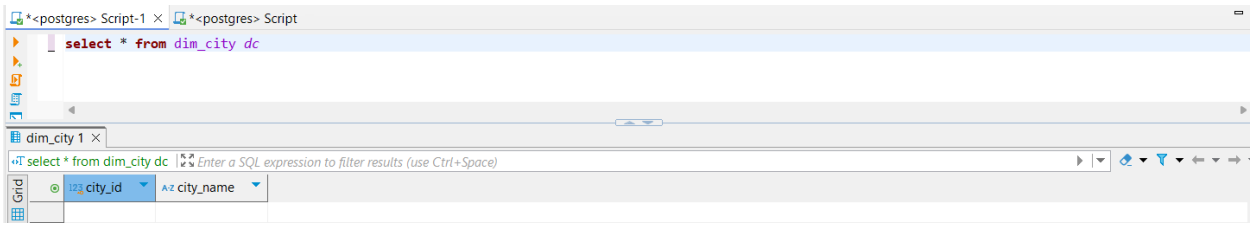

*<postgres> Script-1
select * from dim_station ds
dim_station 1
select * from dim_station ds
Enter a SQL expression to filter results (use Ctrl+Space)
Grid
station_id
station_name
location
num_bikes_available
num_docks_available

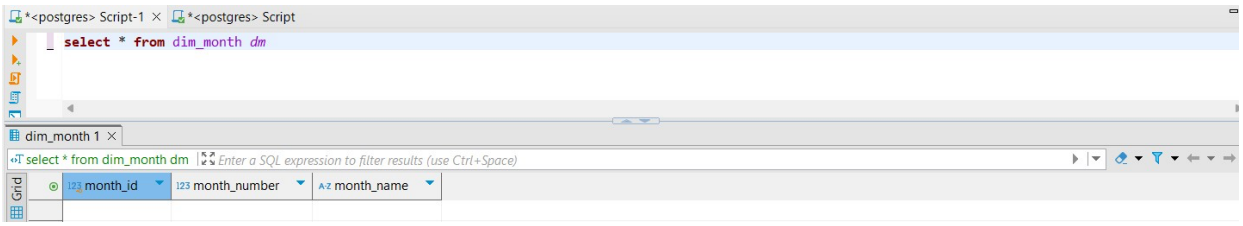

*<postgres> Script-1
*<postgres> Script
select * from dim_support_tickets dst
dim_support_tickets 1
select * from dim_support_tickets dst
Enter a SQL expression to filter results (use Ctrl+Space)
Grid
ticket_id
issue_description
created_at
resolved_at
user_id

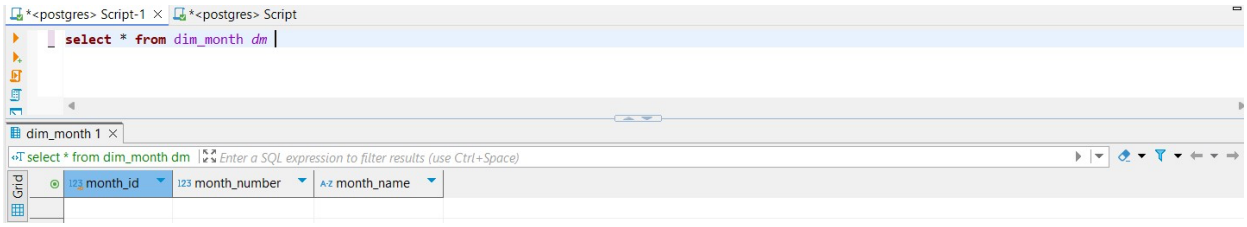

*<postgres> Script-1
select * from dim_payment dp
dim_payment 1
select * from dim_payment dp
Enter a SQL expression to filter results (use Ctrl+Space)
Grid
payment_id
payment_method
payment_status
payment_date

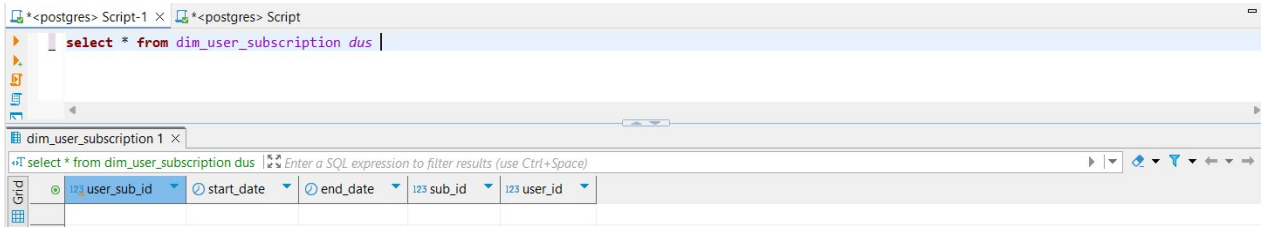

*<postgres> Script-1
*<postgres> Script
select * from dim_maintenance dm
dim_maintenance 1
select * from dim_maintenance dm
Enter a SQL expression to filter results (use Ctrl+Space)
Grid
maintenance_id
issue
downtime_hours
created_at
bike_id

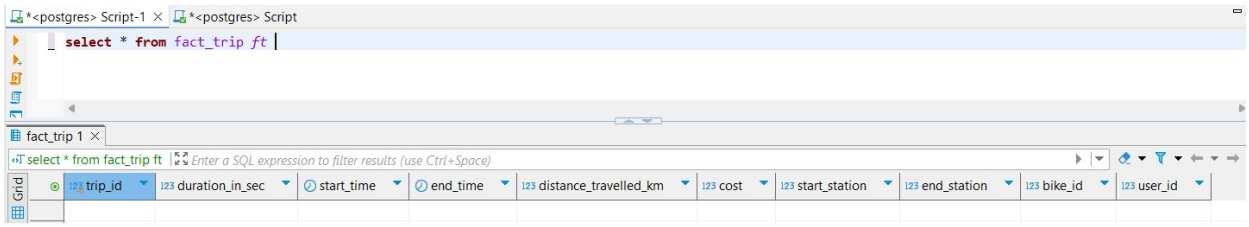

*<postgres> Script-1
*<postgres> Script
select * from dim_state ds
dim_state 1
select * from dim_state ds
Enter a SQL expression to filter results (use Ctrl+Space)
Grid
state_id
state_name











PRIMARY EVENTS

- Trips: A user rents a bike and returns it to another station.
- Payments: A user makes a payment for either a trip or a subscription.
- User Subscriptions: A user subscribes to a specific bike-sharing plan.
- Support Tickets: A user reports an issue or creates a support request.
- Bike Maintenance: A bike is taken out of service for repairs or maintenance.

OLAP QUERIES

1. Total revenue generated from payments for subscriptions in the last month
SLICE(Dim_Payments, Date.Month = "September")
ROLL_UP(Dim_Payments, Dim_Subscription.sub_id)
SUM(Dim_Payments, amount)

2. How many trips were taken by users in each station in the last quarter
ROLL_UP(Fact_Trips, Month.Quarter)
SLICE(Fact_Trips, Month.Quarter = "Q3")
ROLL_UP(Fact_Trips, start_station)
COUNT(Fact_Trips, trip_id)

3. Average duration of trips taken by users who have an active subscription
SLICE(Dim_UserSubscription, end_date > NOW())
ROLL_UP(Fact_Trips, user_id)
AVG(Fact_Trips, duration)

4. Which bikes had the highest number of trips associated with them last month
SLICE(FactTrips, Date.Month = "September")
ROLL_UP(FactTrips, bike_id)
COUNT(FactTrips, trip_id)
ORDER BY COUNT(FactTrips, trip_id) DESC

5. What are the maintenance issues reported for bikes and how long were they down
ROLL_UP(Dim_Maintenance, bike_id)
SUM(Dim_Maintenance, downtime_hrs)
SELECT(Dim_Maintenance, issue)

6. Calculate total payments by user and by month
ROLL_UP(Dim_Payments, user_id, Date.Month)
SUM(Dim_Payments, amount)

7. Calculate total trips by user and bike type
ROLL_UP(Fact_Trips, user_id, Dim_Bike.type)
COUNT(Fact_Trips, trip_id)

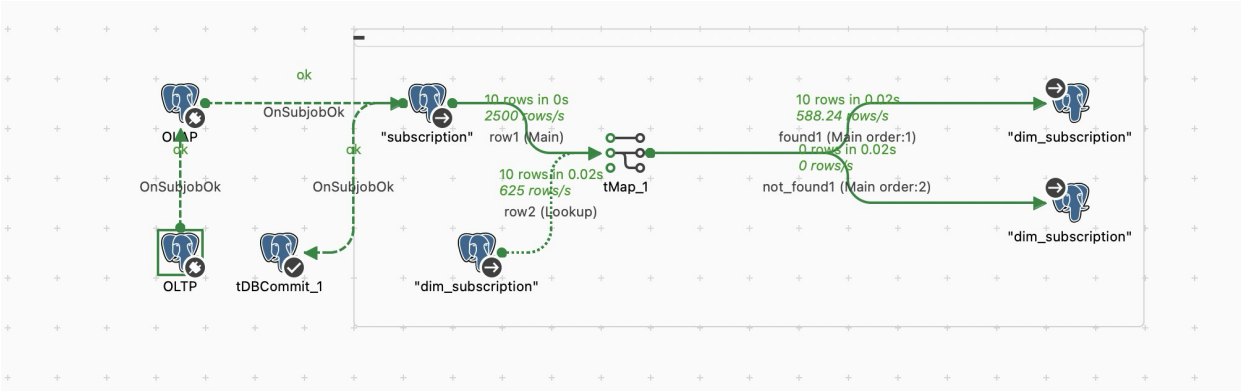
ETL Implementation in Talend:

1. ETL Execution
    - Insert or update data with surrogate key
2. Additional Features
    - Implemented Type 3 SCD
    - Calculated measures:
        - Trip duration in minutes → Fact Trip

3. Data Sources:
    - CSV files - city
    - XML files - date, time
    - OLTP PostgresDB
4. Transformations:
    - Converted to Date format
    - Converted to Time format
    - Converted to appropriate data types in the targetDB (int, decimal, String)

ETL Execution

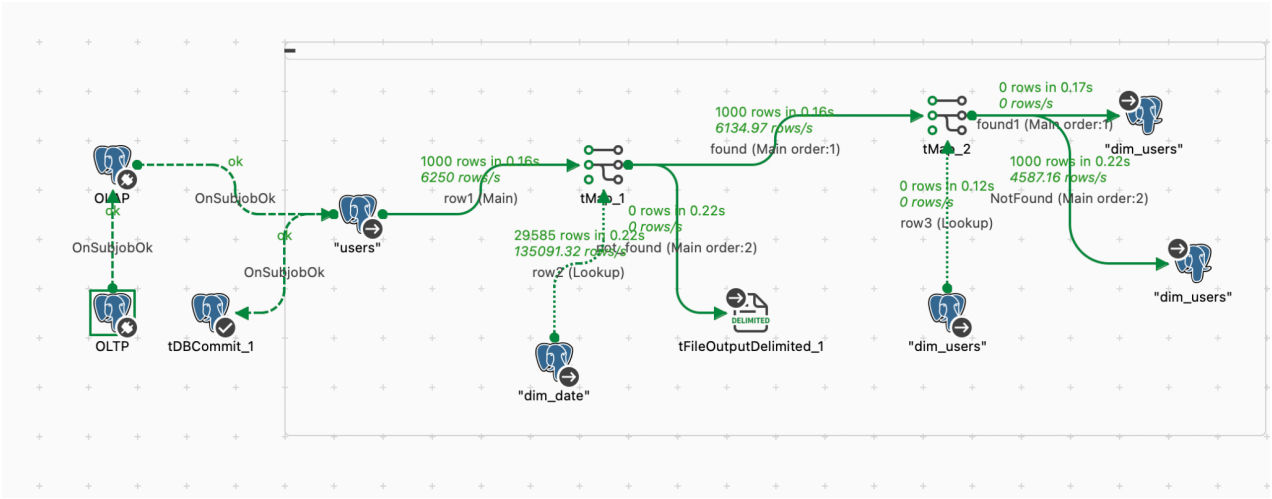
1. Dim_Subscription (surrogate_key : subscription_dim_

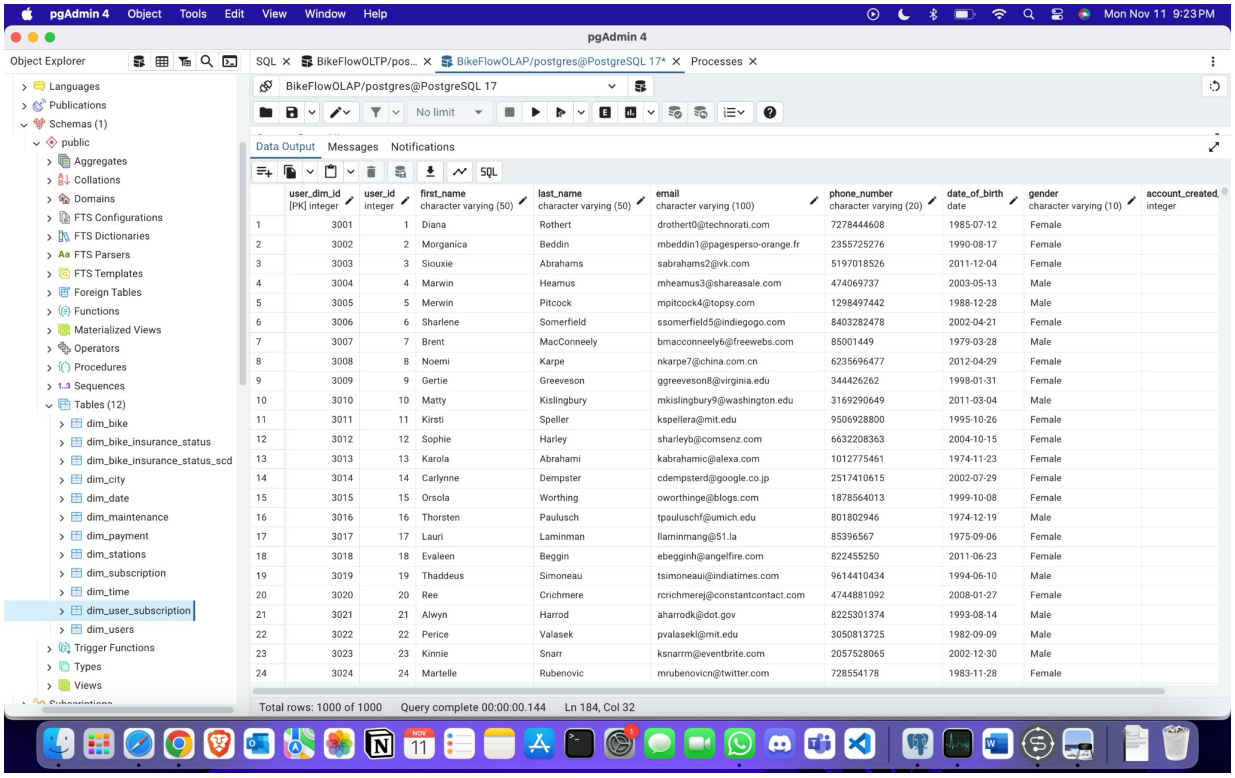


SQL

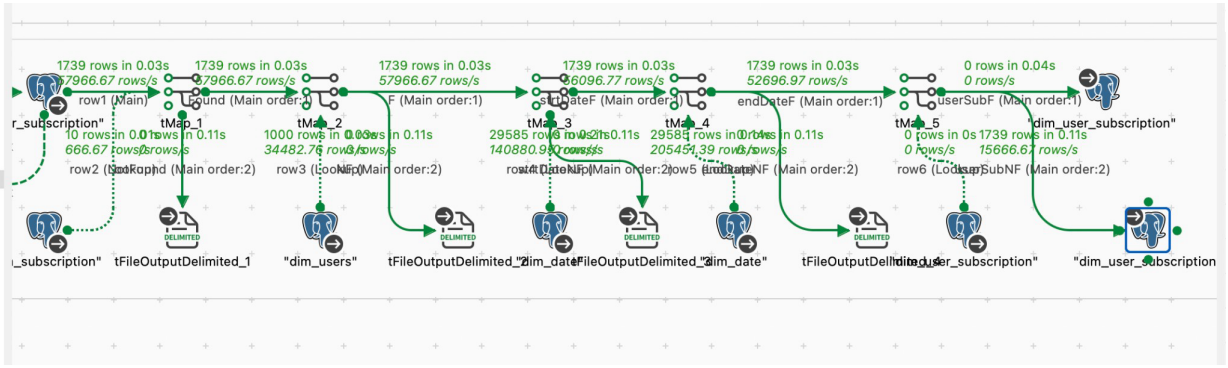
	subscription_dim_id [PK] integer	subscription_id integer	type character varying (50)	price numeric (10,2)	duration character varying (100)	description character varying (100)
1	873	1	Single Ride	3.99	30 minutes	Classic bike: 30 min for $3.99, then $0.30/min
2	874	2	Day Pass	15.00	1 day	Classic bike: 30 min free, then $0.30/min
3	875	3	Month Pass	29.00	1 month	Classic bike: 45 min free, then $0.20/min
4	876	4	Bay Wheels	150.00	1 year	Classic bike: 45 min free, then $0.20/min
5	877	5	Lyft Pink	199.00	1 year	Classic bike: 45 min free, then $0.20/min
6	878	6	Ebike - Single Ride	3.99	30 minutes	Ebike: $3.99 unlock + $0.30/min
7	879	7	Ebike - Day Pass	15.00	1 day	Ebike: Free unlock + $0.30/min
8	880	8	Ebike - Month Pass	29.00	1 month	Ebike: Free unlock + $0.15/min for 45 min
9	881	9	Ebike - Bay Wheels	150.00	1 year	Ebike: Free unlock + $0.15/min for 45 min
10	882	10	Ebike - Lyft Pink	199.00	1 year	Ebike: Free unlock + $0.15/min for 45 min

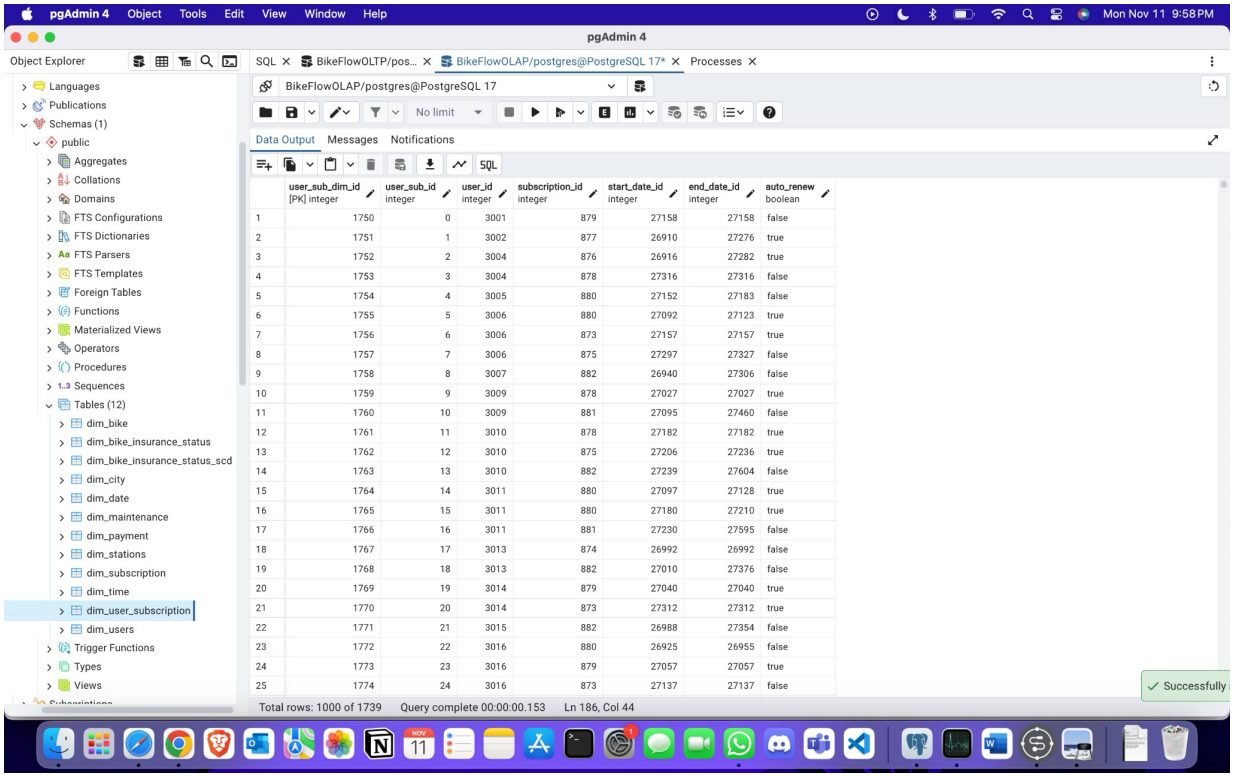
2. Dim_Users (surrogate_key : user_dim_id)



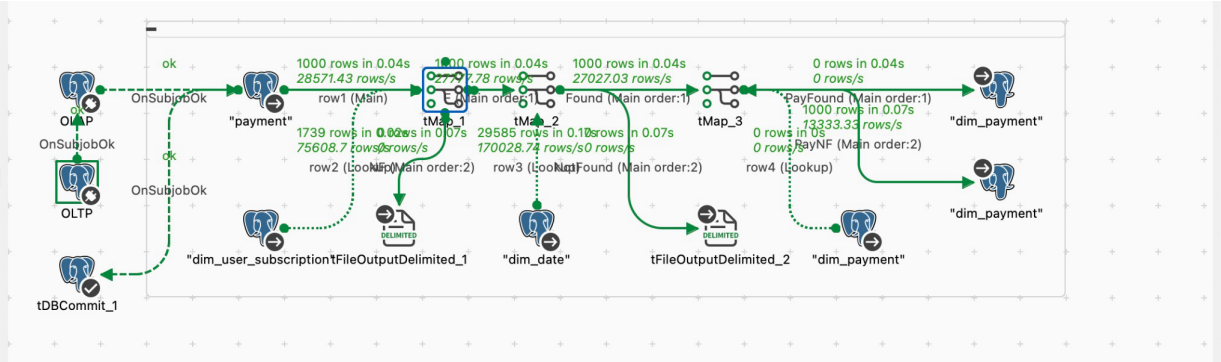


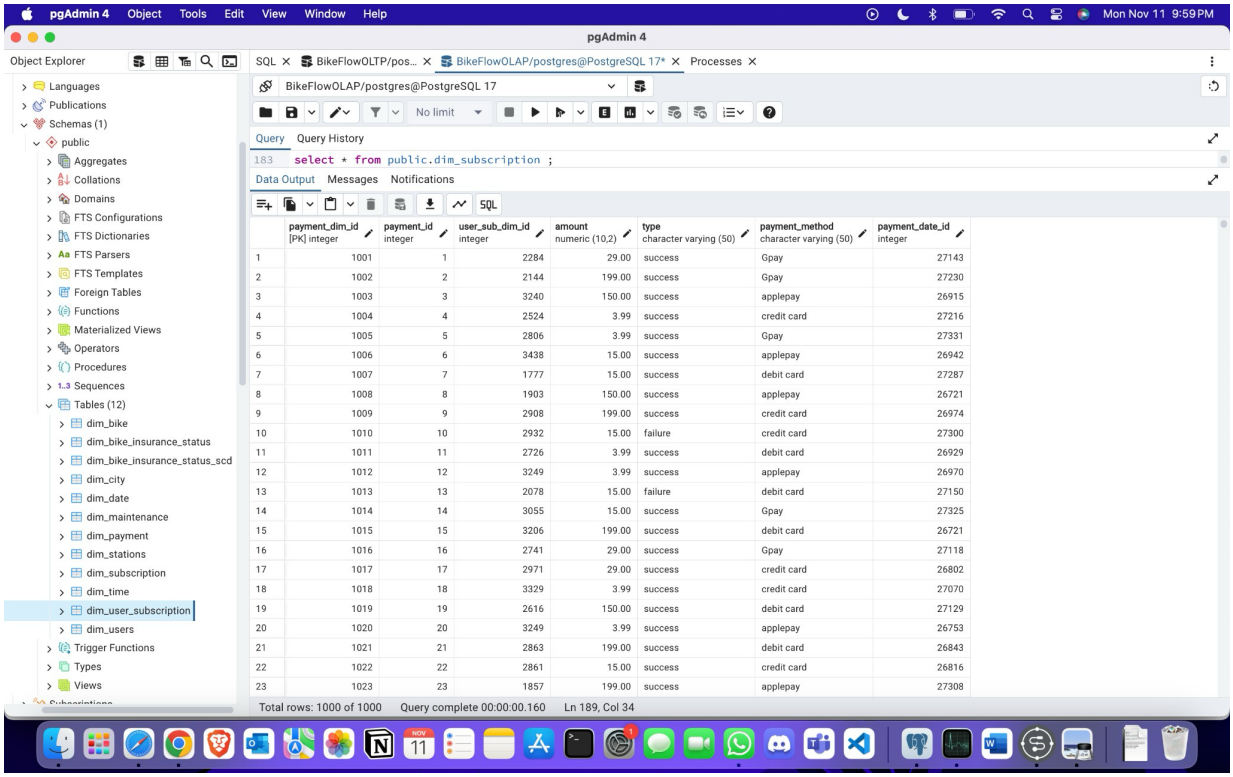
3. Dim_User_Subscription (surrogate_key : user_sub_dim_id)



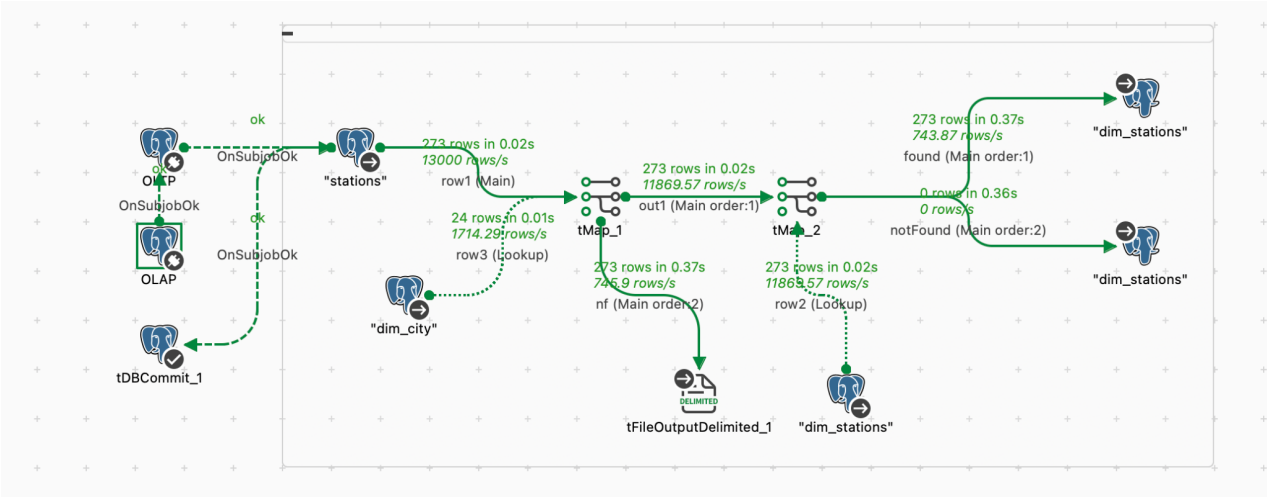


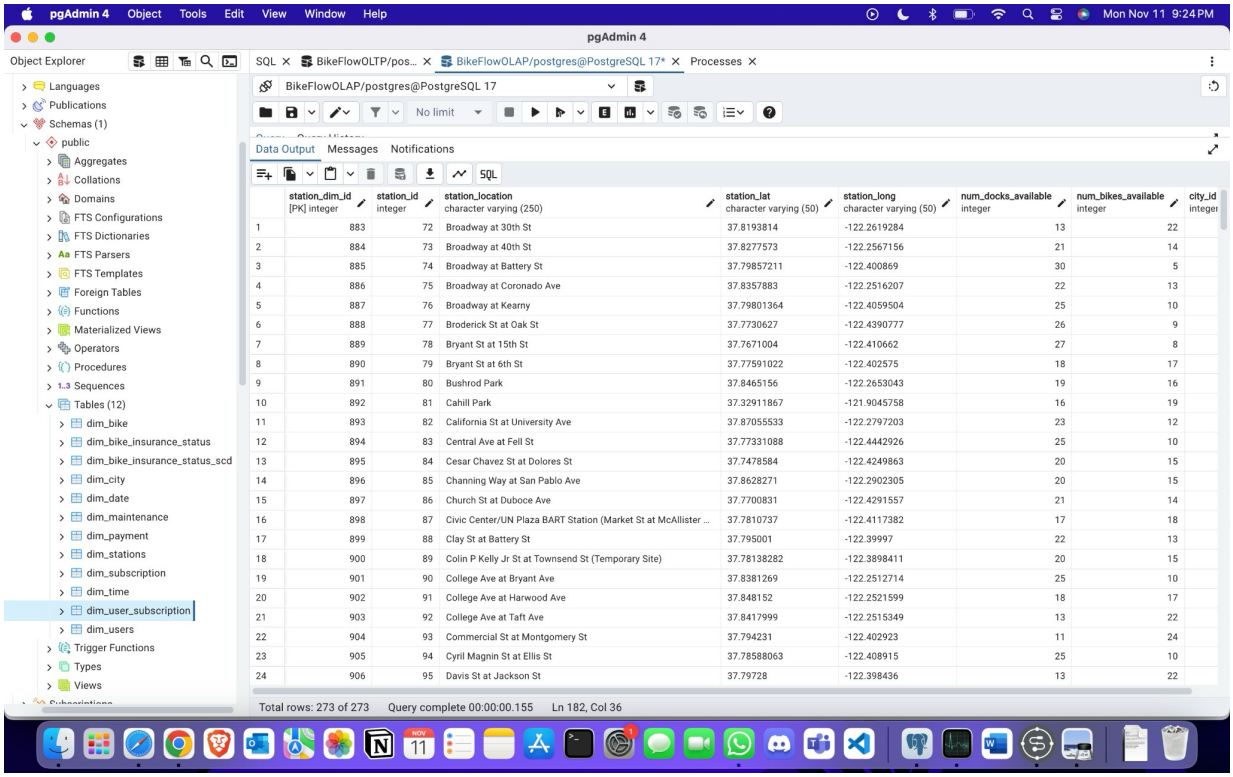
4. Dim_Payments (surrogate_key : payment_dim_id)



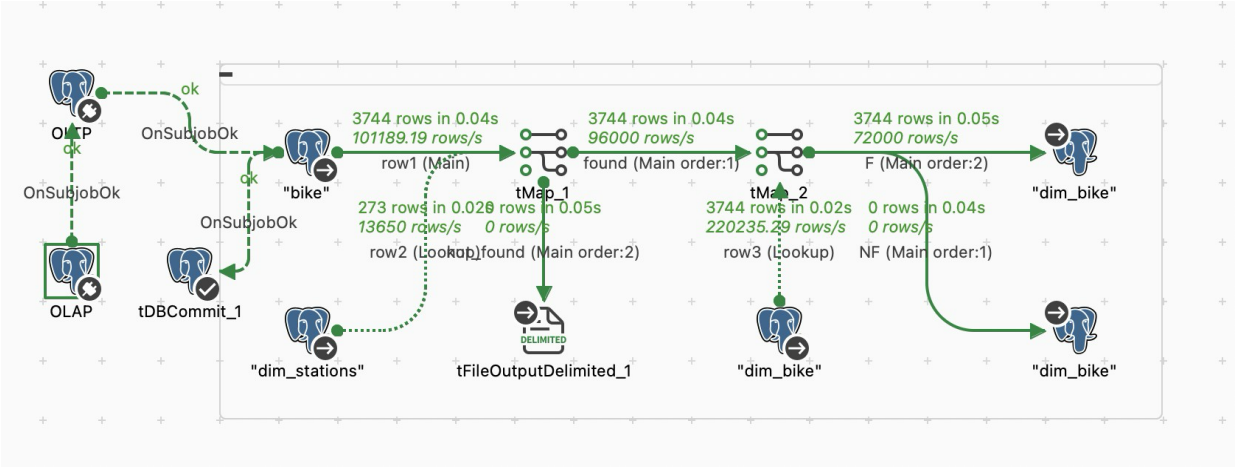


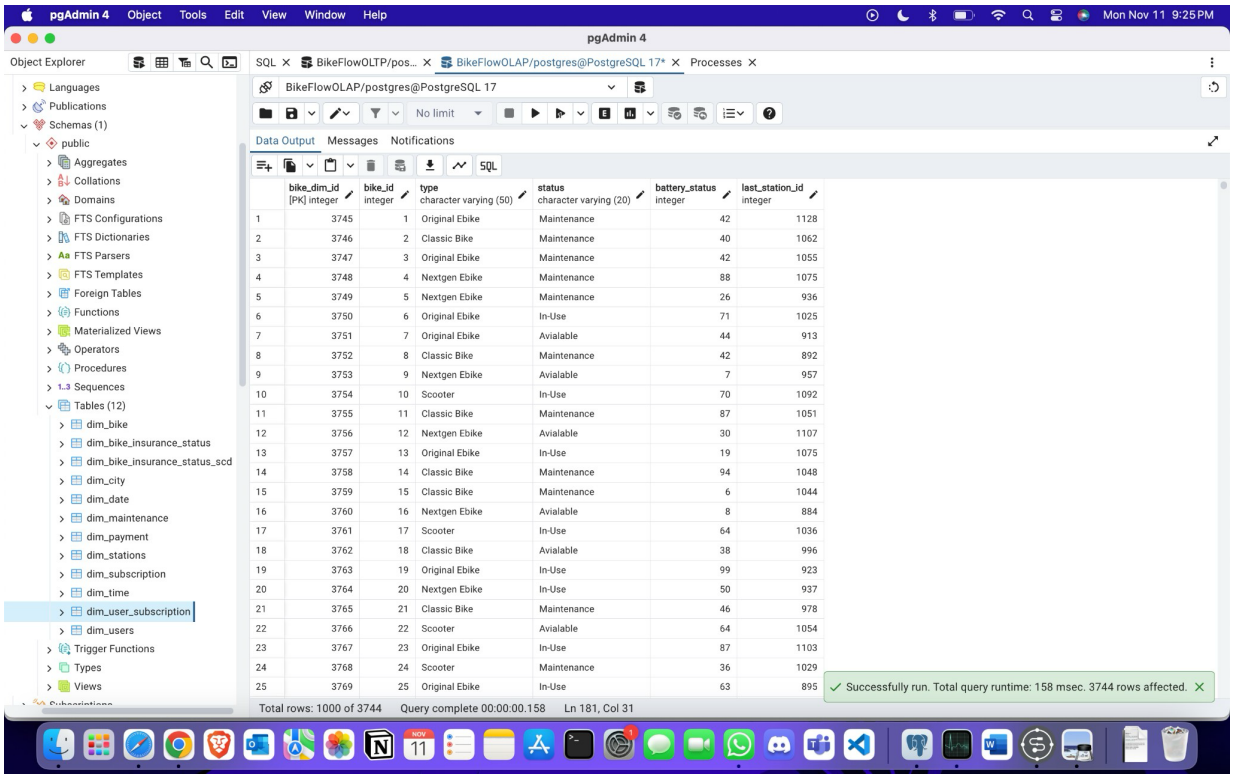
5. Dim_Stations (surrogate_key : station_dim_id)



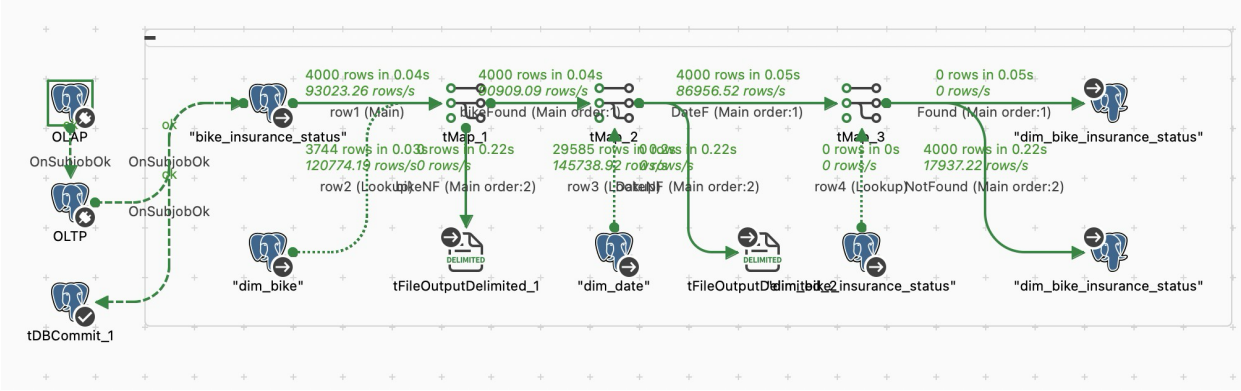


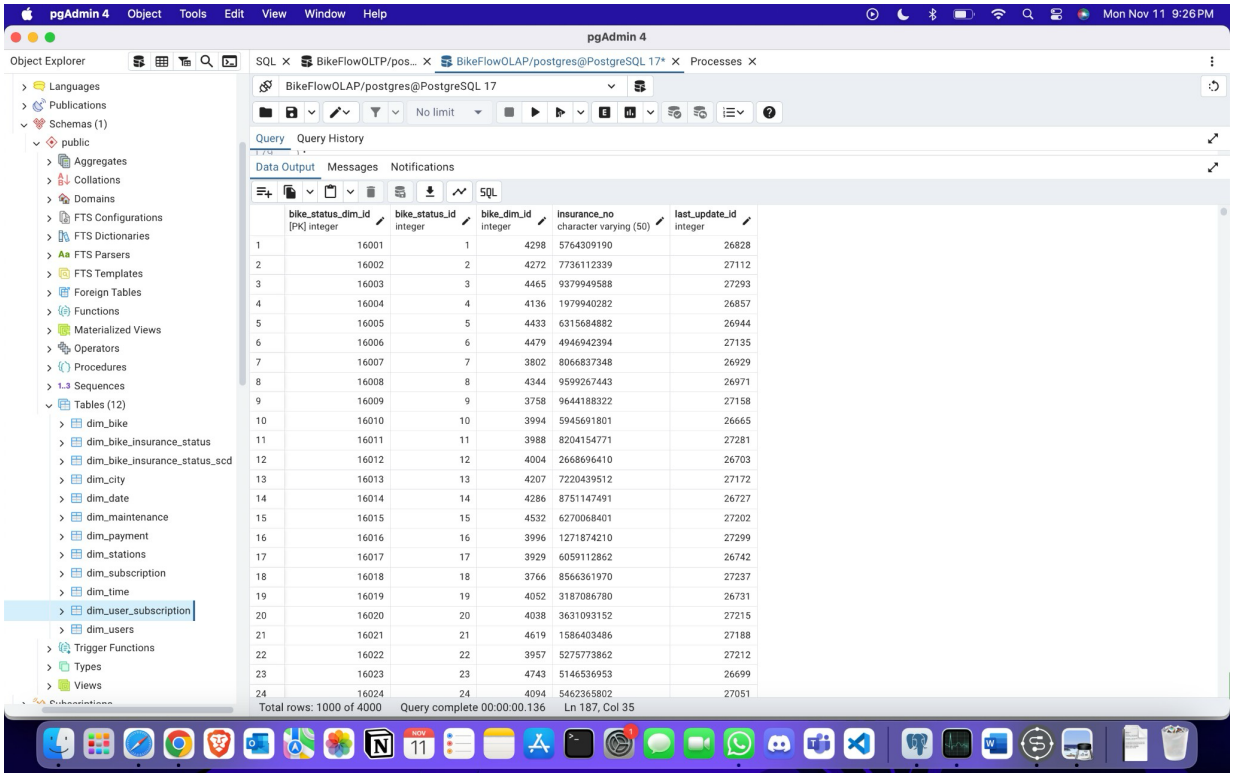
6. Dim_bikes (surrogate_key : bike_dim_id)



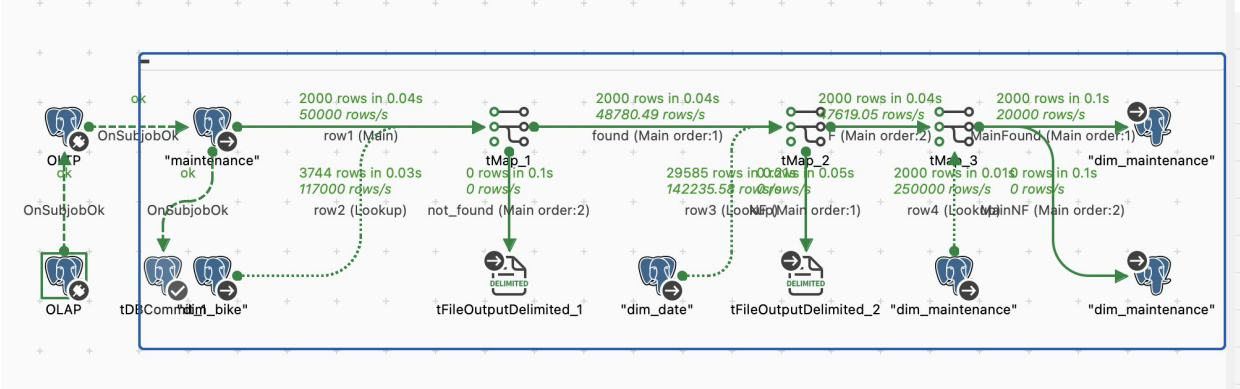


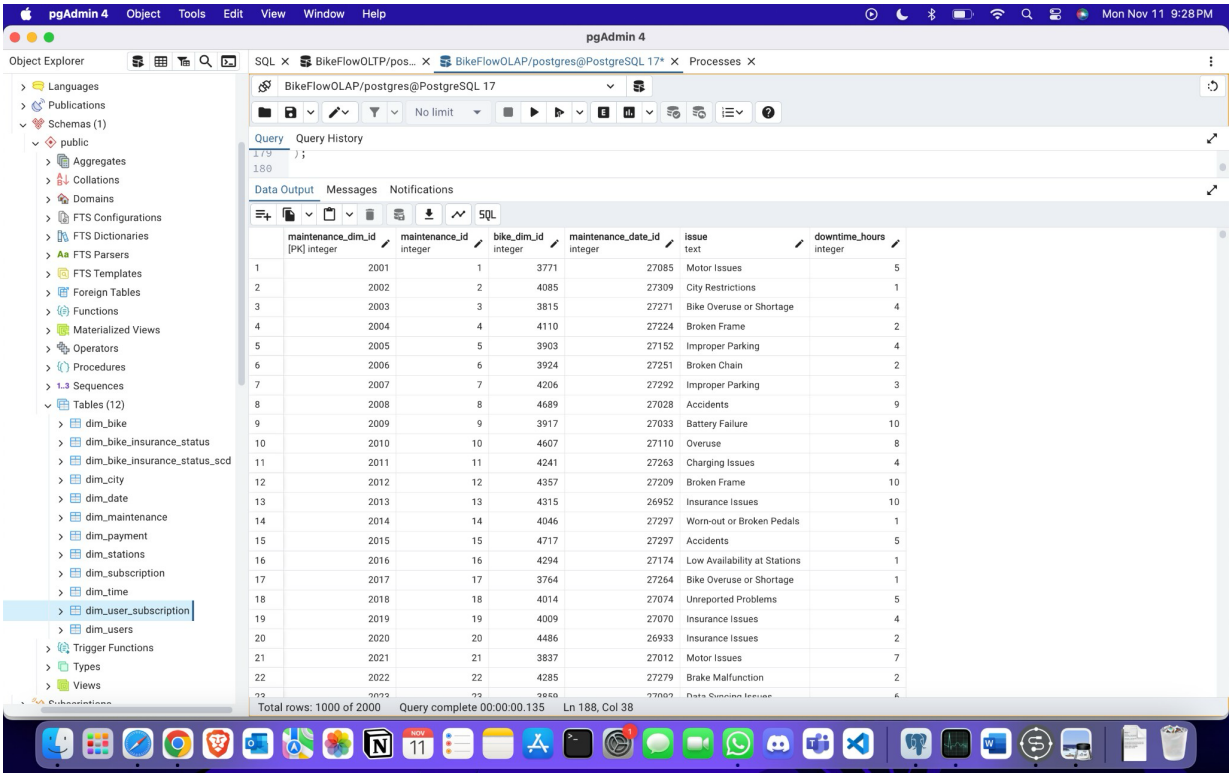
7. Dim_bike_insurance_status (surrogate_key : bike_status_dim_id)



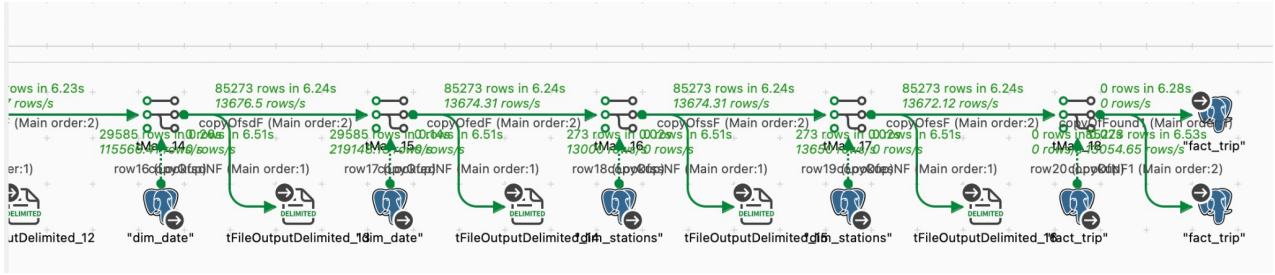


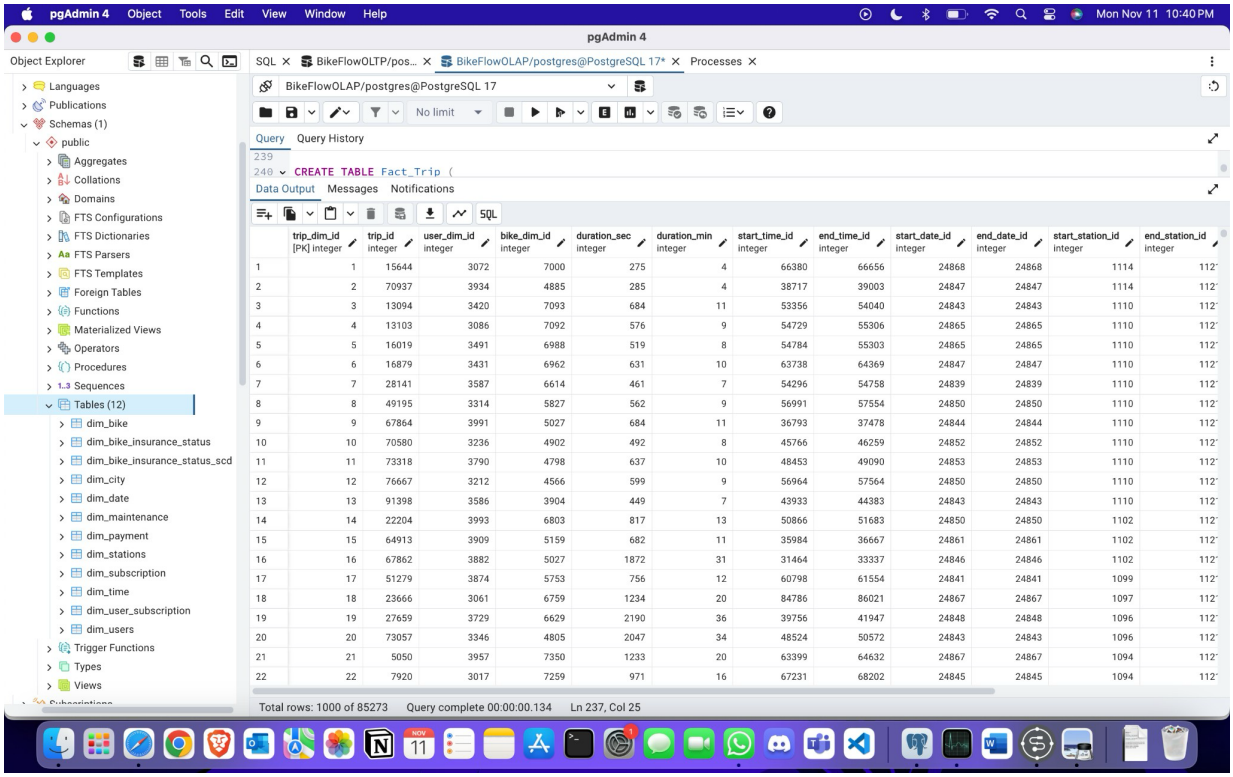
8. Dim_bike_maintenance(surrogate_key : maintenance_dim_id)



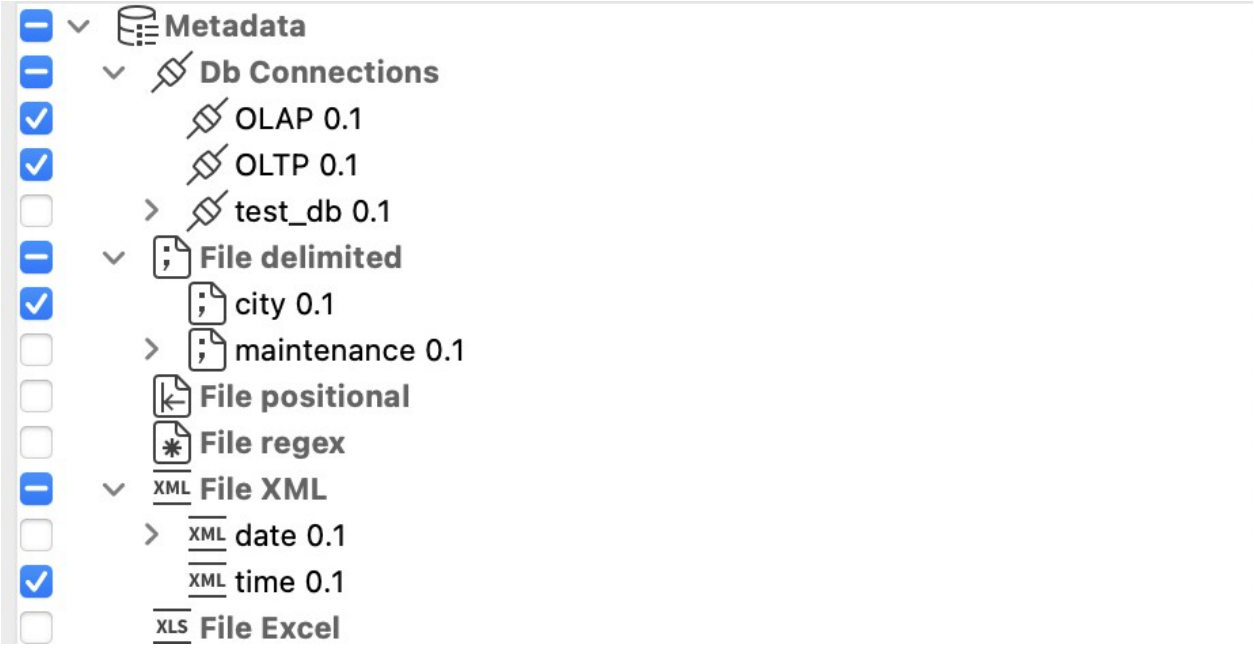


9. FACT_TRIP (surrogate_key : trip_dim_id)



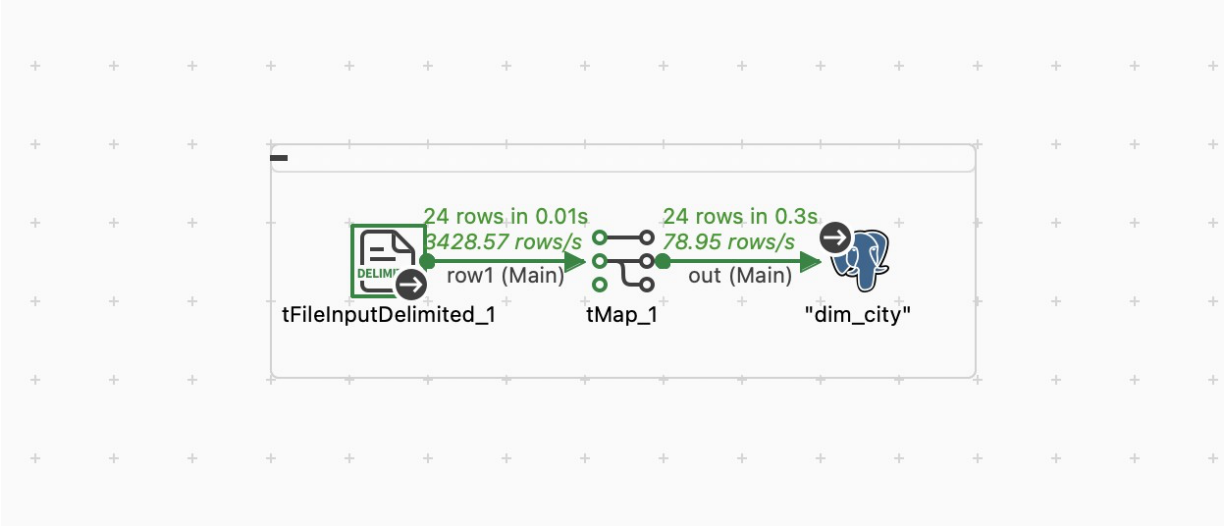


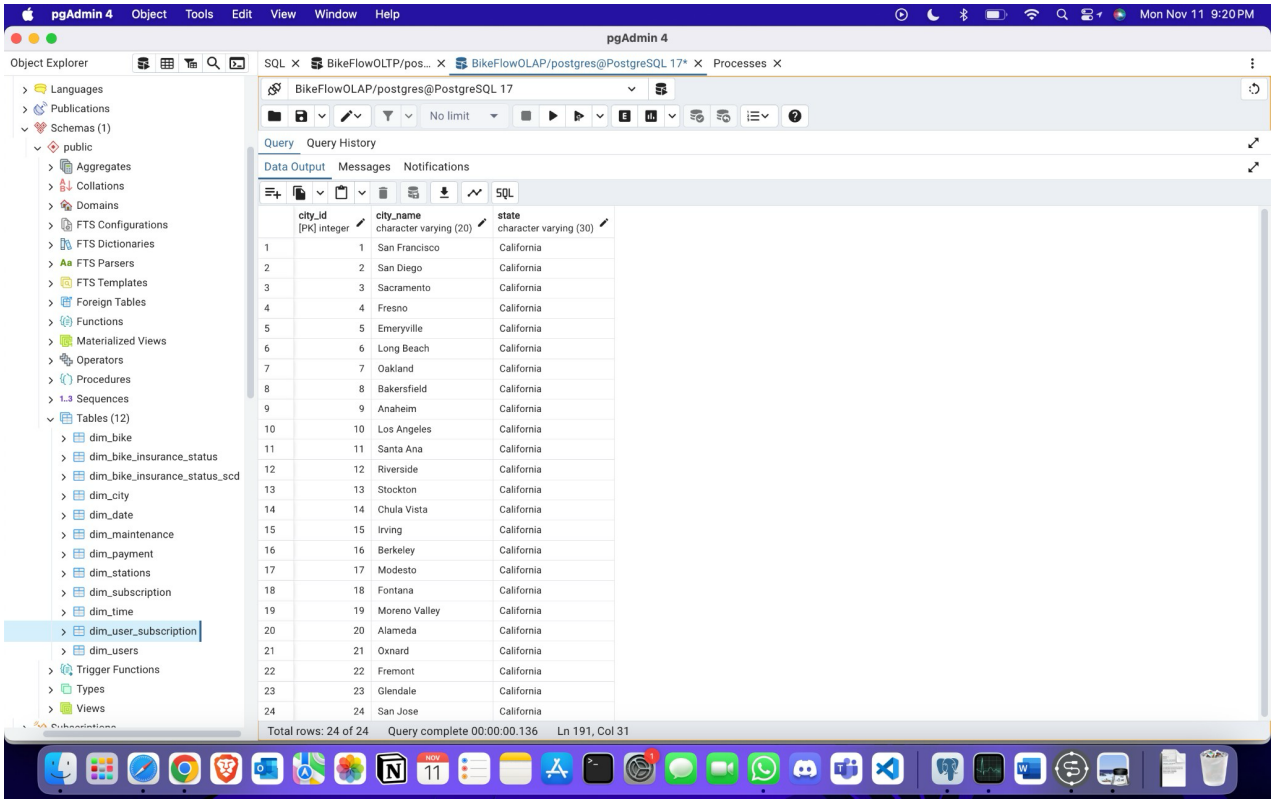
Data Sources:



CSV:

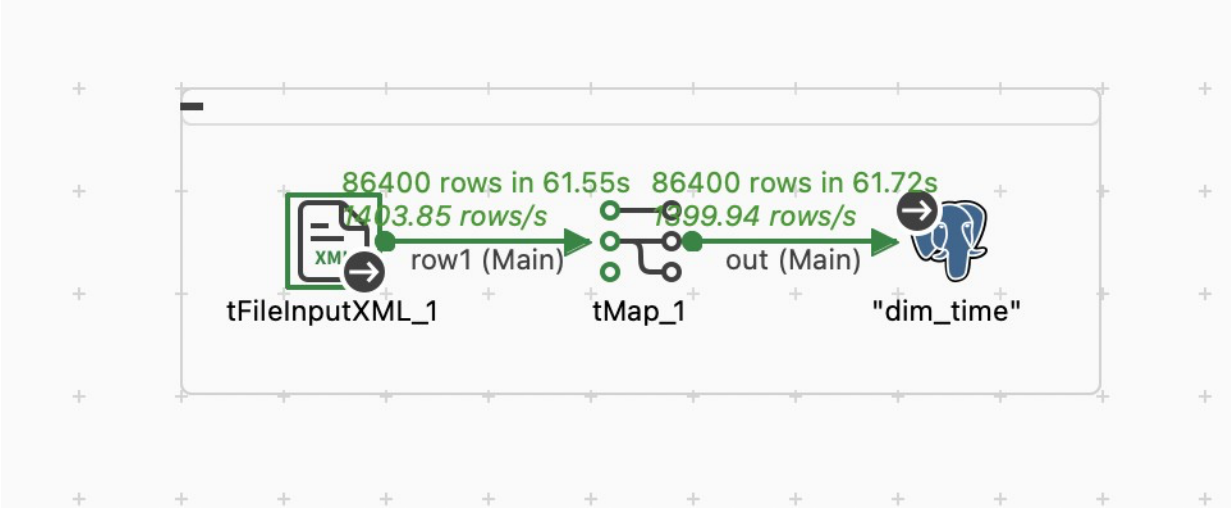
1. Dim_City

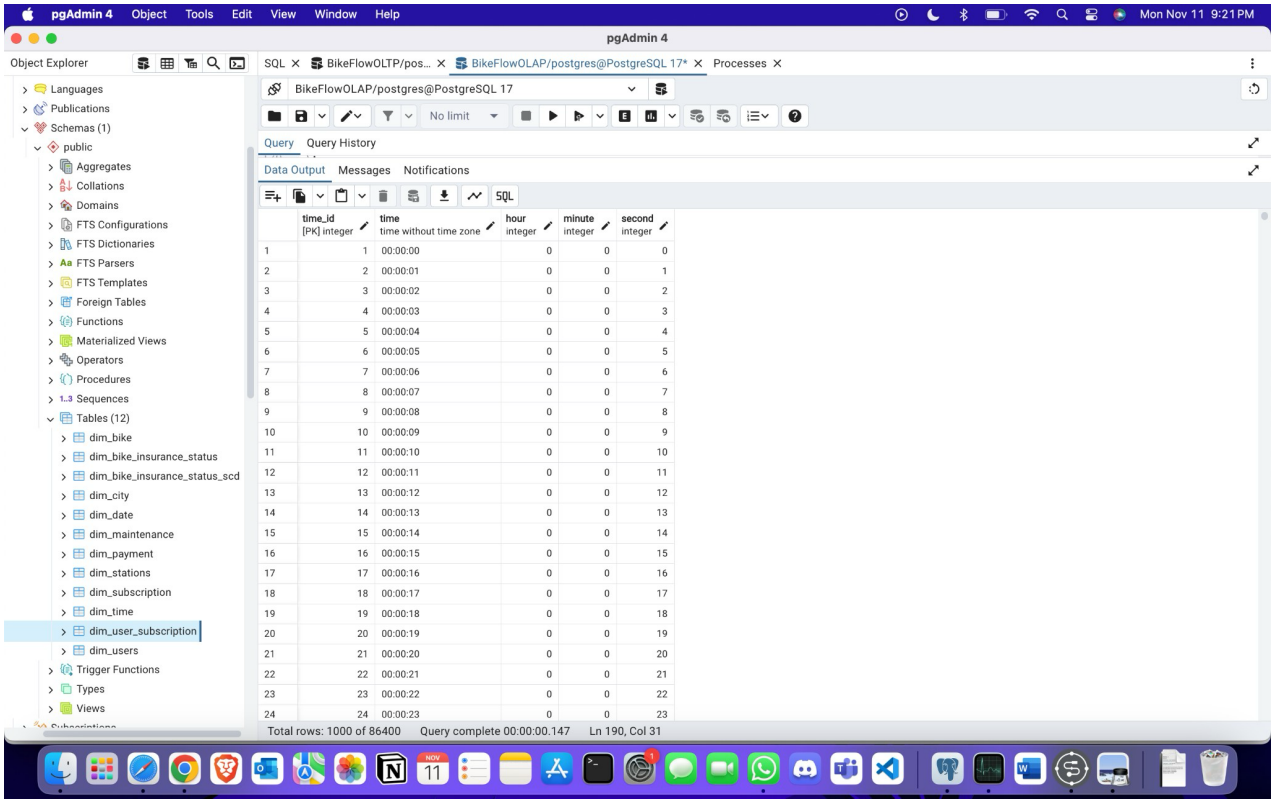




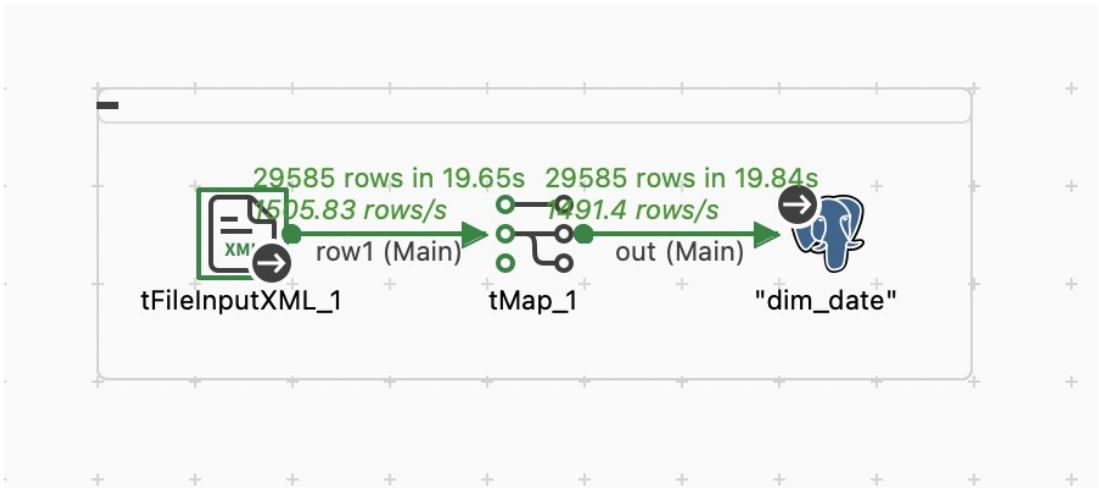
XML:

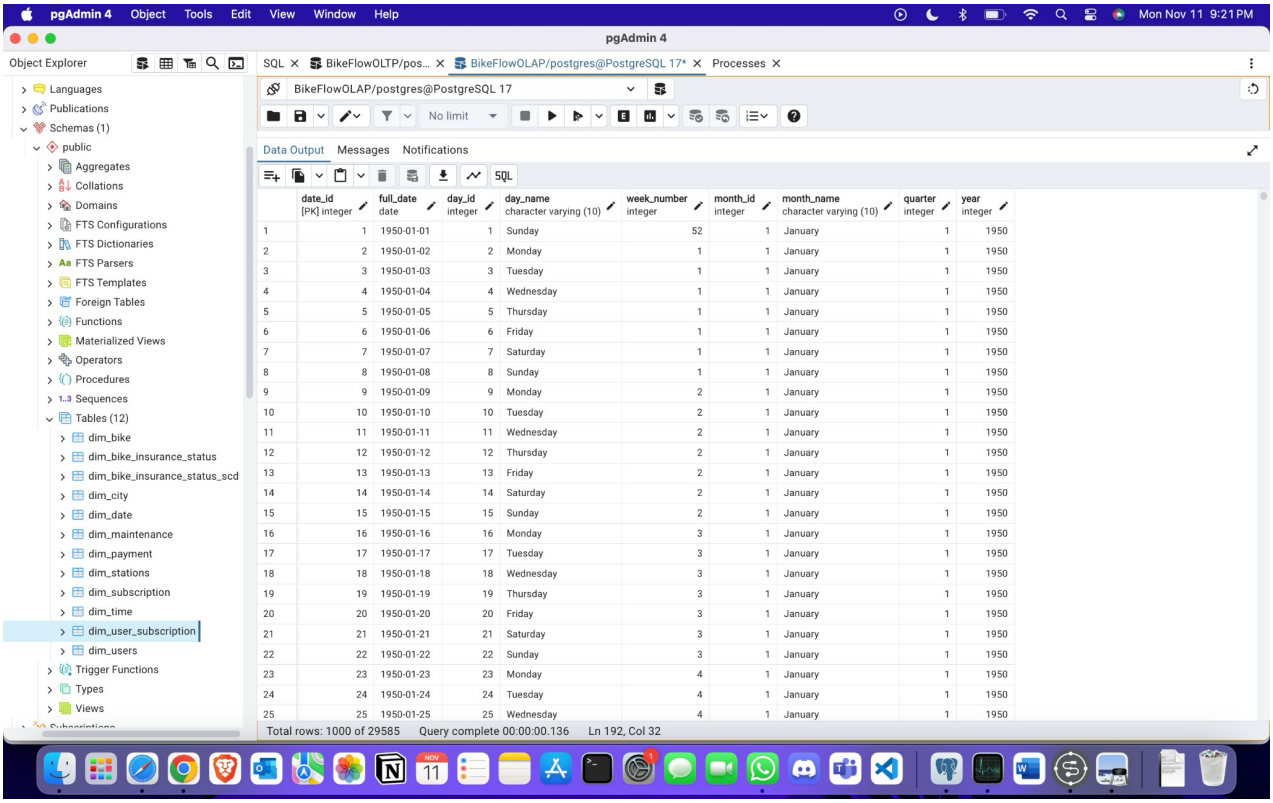
1. Dim_Time




pgAdmin 4
Object Explorer
Languages
Publications
Schemas (1)
public
Aggregates
Collations
Domains
FTS Configurations
FTS Dictionaries
FTS Parsers
FTS Templates
Foreign Tables
Functions
Materialized Views
Operators
Procedures
Sequences
Tables (12)
dim_bike
dim_bike_insurance_status
dim_bike_insurance_status_scd
dim_city
dim_date
dim_maintenance
dim_payment
dim_stations
dim_subscription
dim_time
dim_user_subscription
dim_users
Trigger Functions
Types
Views
BikeFlowOLAP/postgres@PostgreSQL 17
Query
Query History
Data Output
Messages
Notifications
time_id [PK] integer
time time without time zone
hour integer
minute integer
second integer
Total rows: 1000 of 86400
Query complete 00:00:00.147
Ln 190, Col 31

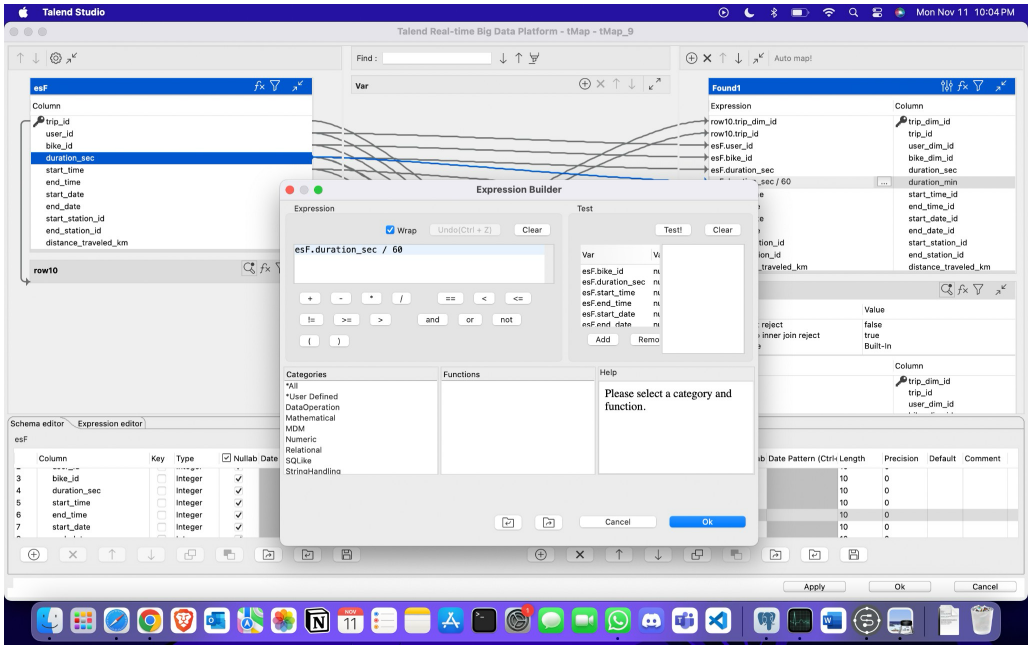
2. Dim_Date


29585 rows in 19.65s
1505.83 rows/s
row1 (Main)
tFileInputXML_1
29585 rows in 19.84s
1491.4 rows/s
out (Main)
tMap_1
"dim_date"

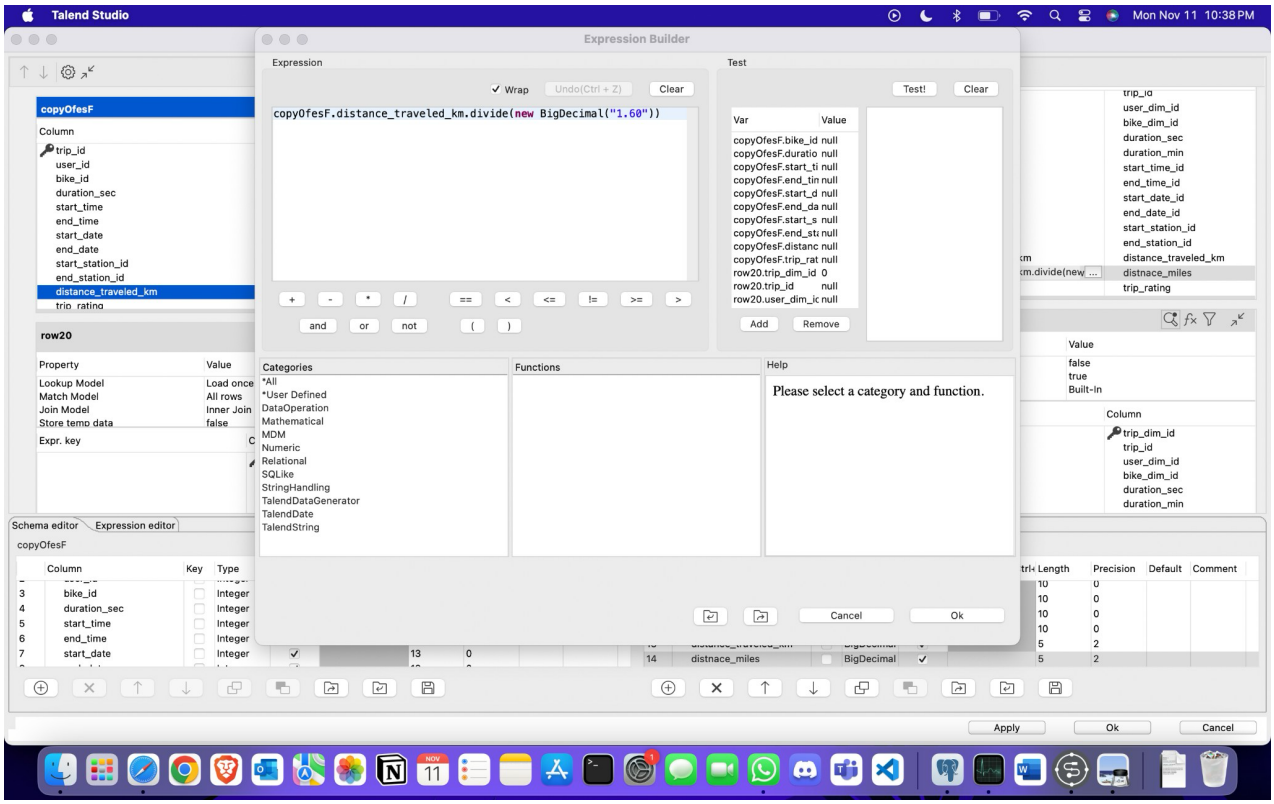


Calculated Measure:

1. Duration of the trip is converted to minutes

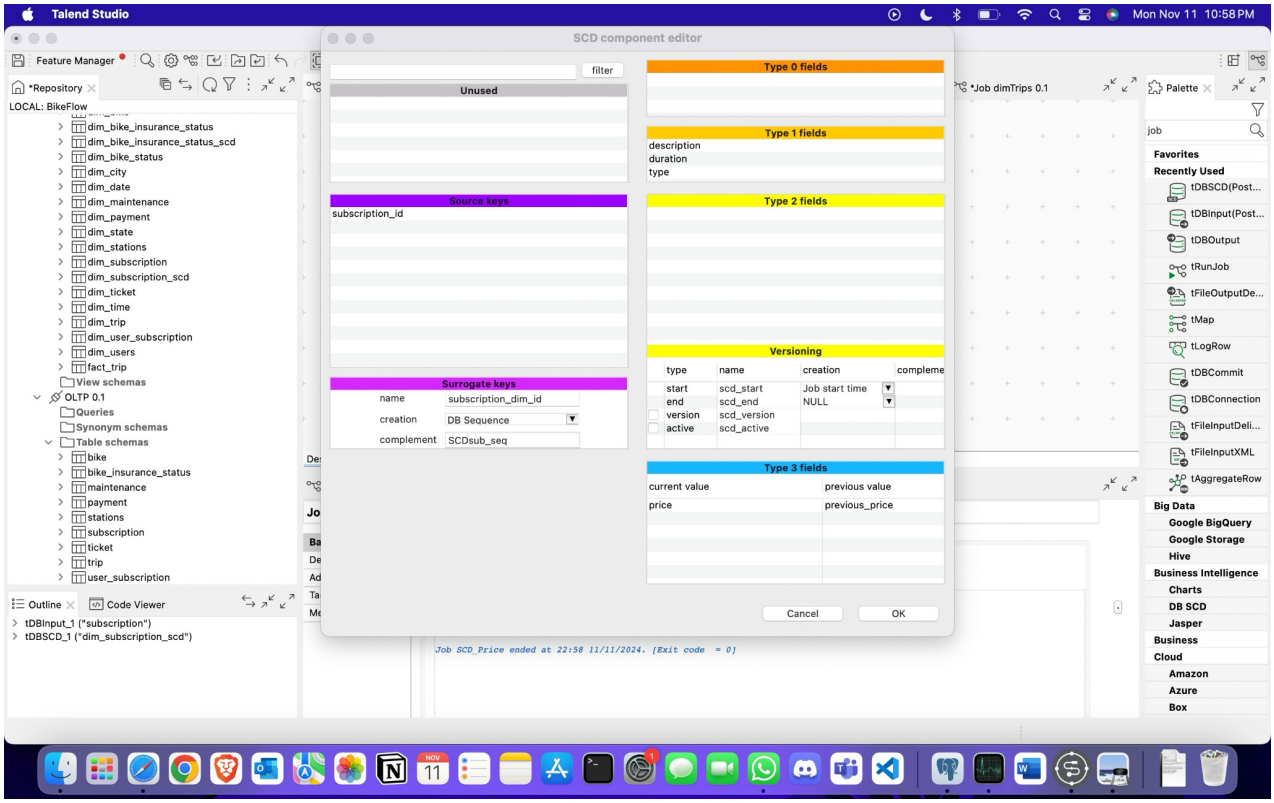


2. Distance of the trip is converted to miles

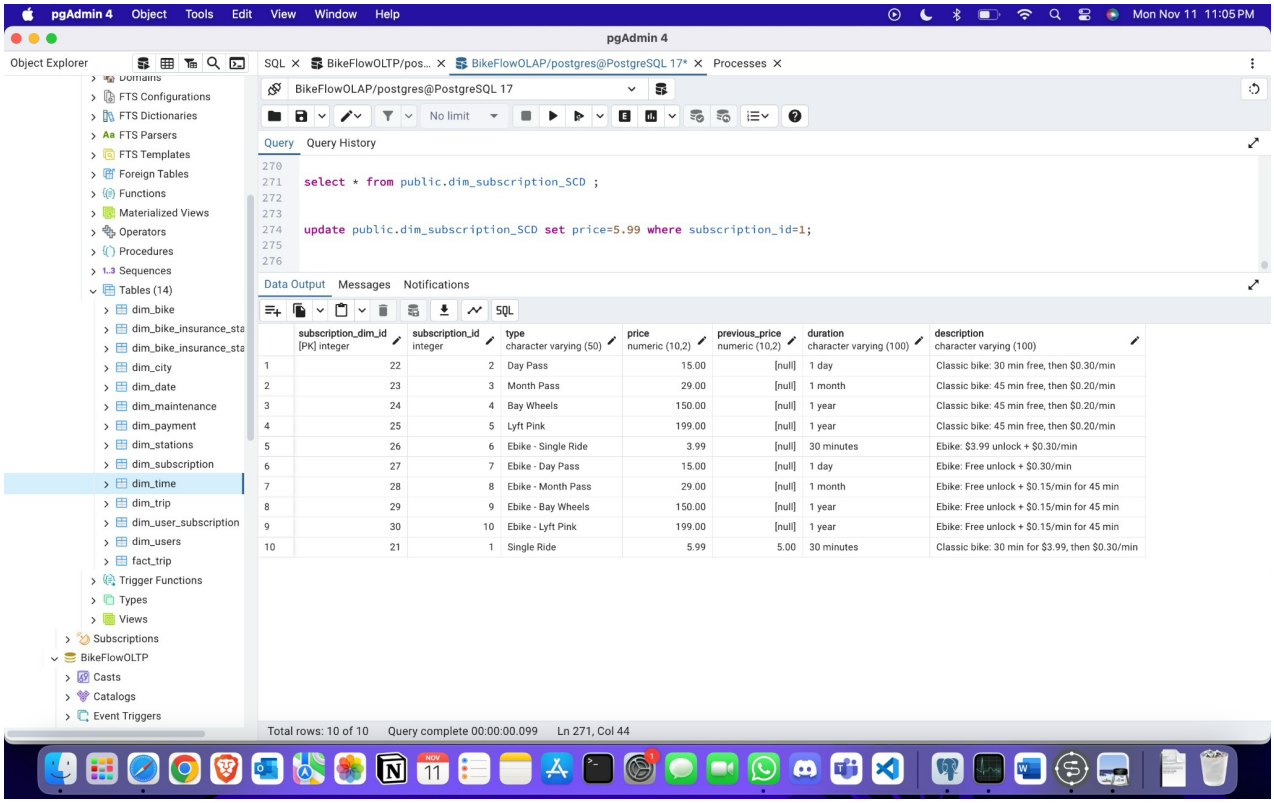


SCD:

Type 3:

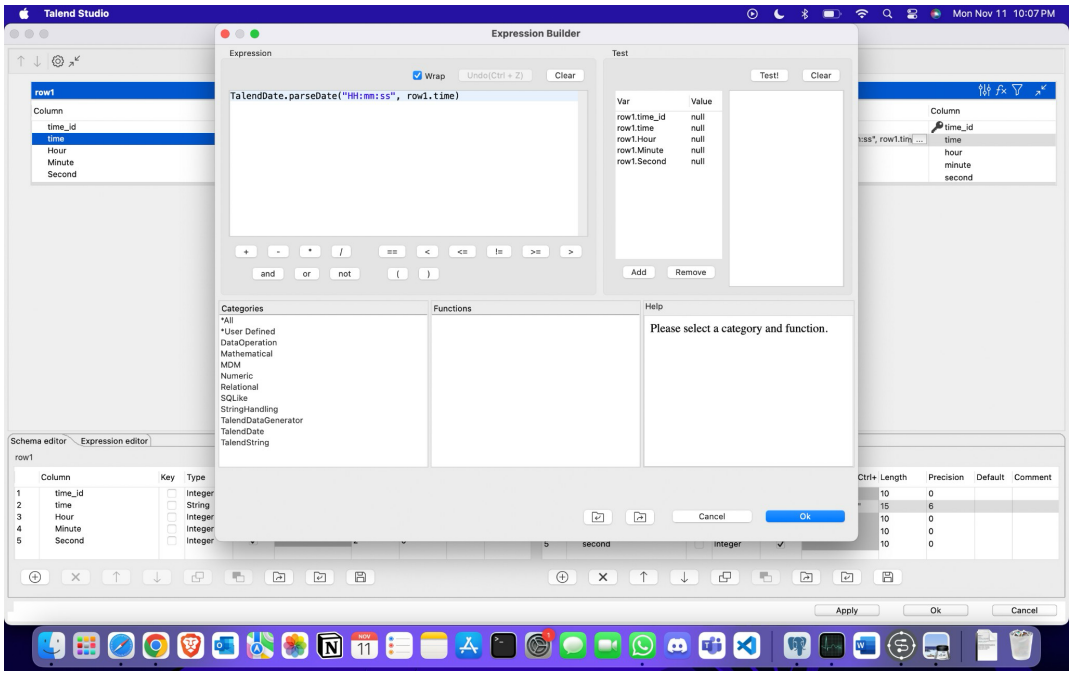


	subscription_dim_id [PK] integer	subscription_id integer	type character varying (50)	price numeric (10,2)	previous_price numeric (10,2)	duration character varying (100)	description character varying (100)
1	21	1	Single Ride	3.99	[null]	30 minutes	Classic bike: 30 min for $3.99, then $0.30/min
2	22	2	Day Pass	15.00	[null]	1 day	Classic bike: 30 min free, then $0.30/min
3	23	3	Month Pass	29.00	[null]	1 month	Classic bike: 45 min free, then $0.20/min
4	24	4	Bay Wheels	150.00	[null]	1 year	Classic bike: 45 min free, then $0.20/min
5	25	5	Lyft Pink	199.00	[null]	1 year	Classic bike: 45 min free, then $0.20/min
6	26	6	Ebike - Single Ride	3.99	[null]	30 minutes	Ebike: $3.99 unlock + $0.30/min
7	27	7	Ebike - Day Pass	15.00	[null]	1 day	Ebike: Free unlock + $0.30/min
8	28	8	Ebike - Month Pass	29.00	[null]	1 month	Ebike: Free unlock + $0.15/min for 45 min
9	29	9	Ebike - Bay Wheels	150.00	[null]	1 year	Ebike: Free unlock + $0.15/min for 45 min
10	30	10	Ebike - Lyft Pink	199.00	[null]	1 year	Ebike: Free unlock + $0.15/min for 45 min

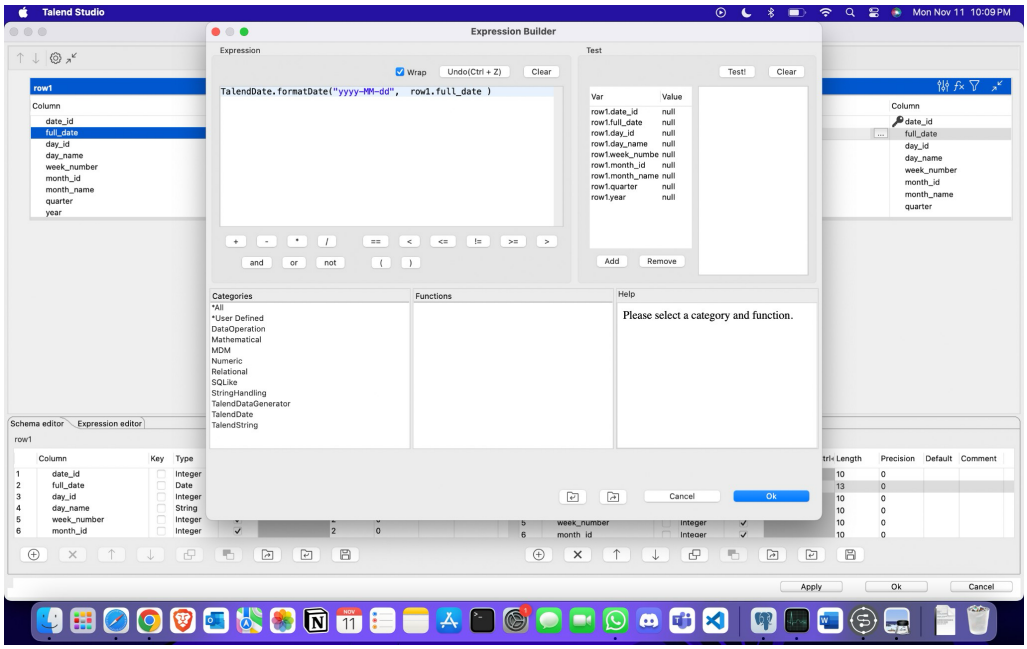


Transformations:

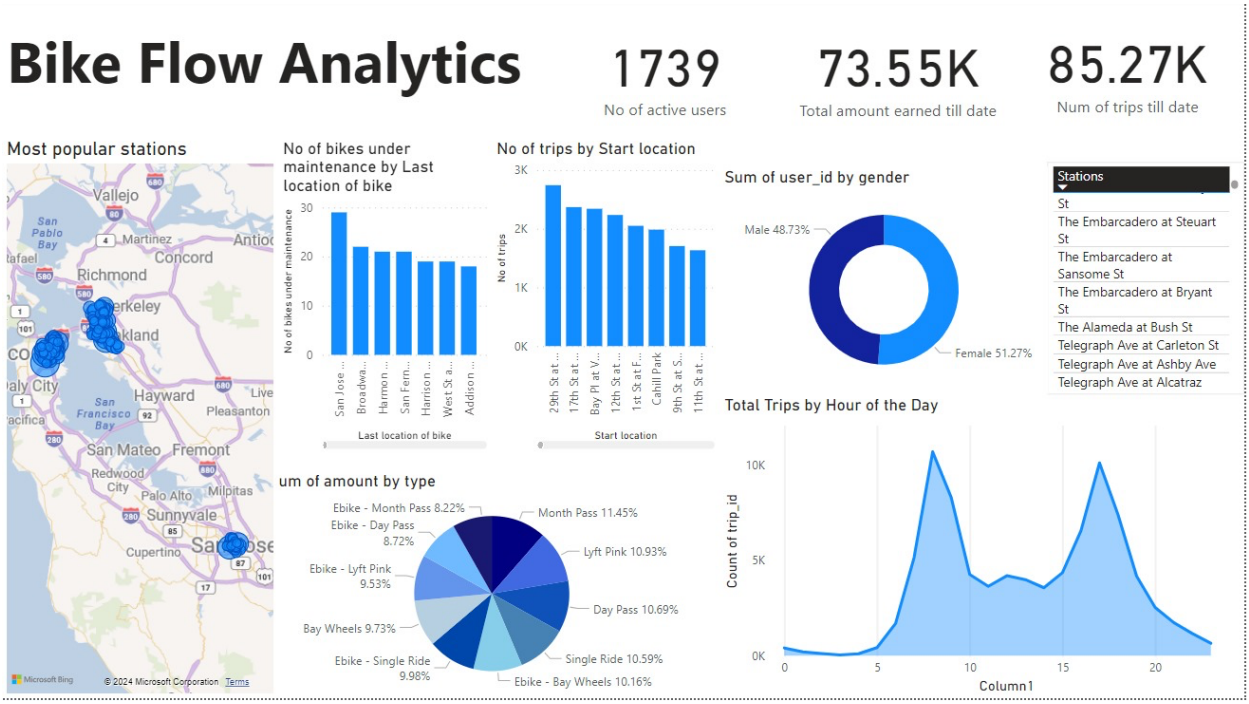
Changing Time formats:



Changing Date Formats:



DASHBOARD:



GOAL - The primary goals are to optimize bike availability during peak hours, strategically position bikes at high-demand stations, enhance operational efficiency through maintenance tracking, increase revenue via targeted marketing informed by user demographics, and improve overall user satisfaction by leveraging actionable insights from data analysis.

KPI's

Number of Active Users:
- 1,739 active users.
- Indicates overall user engagement and adoption of the service.

Total Revenue Generated:
- $73.55K total revenue to date.
- Measures financial performance and service profitability.

Total Trips Taken:
- 85.27K trips to date.
- Reflects service utilization and user activity.

Gender Distribution:
- Male: 48.73%, Female: 51.27%.
- Provides insights into the demographic breakdown of users.

Peak Usage Hours:
- Total trips by the hour of the day, with distinct peaks during high-demand periods.
- Highlights key time windows for operational optimization.

Most Popular Stations:
- Interactive map identifies high-traffic locations, such as Berkeley and San Francisco.
- Aids in strategic placement of bikes to maximize usage.

Metrics:

1. **Interactive Map**:
    - Clusters stations by popularity.
    - High-density usage areas like Berkeley and San Francisco are clearly highlighted.
2. **Maintenance Overview**:
    - Bar chart showing the number of bikes under maintenance categorized by the last known location.
    - Supports tracking and management of service maintenance needs.
3. **Trip Start Locations**:
    - Bar chart illustrating the volume of trips initiated from various start points.
    - Pinpoints areas with the highest demand for services.
4. **Revenue Breakdown by Bike Type**:
    - Pie chart summarizing revenue by bike type and pass type.
    - Categories include Ebike (e.g., Month Pass: 11.45%, Day Pass: 8.22%) and Bay Wheels.
5. **Hourly Trip Trends**:
    - Line chart representing total trips taken by the hour of the day.
    - Helps identify peak and off-peak usage patterns for staffing and bike redistribution.
6. **Filters and Drill-Through Features**:
    - Allows slicing data by station, bike type, or time period.
    - Drill-through capabilities enable deeper insights into specific metrics like station performance or bike type preferences.

Insights:

1. **Peak Usage Identification**:
    - The line chart highlights morning (8-10 AM) and evening (4-6 PM) peaks, suggesting commuter-driven usage patterns. This indicates a need for increased bike availability during these times
2. **Popular Station Insights**:

    - Popular stations like "The Embarcadero at Steuart St" and others in San Francisco and Berkeley highlight areas requiring more bikes to meet demand.
3. **User Demographics**:
    - Assists in creating targeted marketing campaigns.
4. **Maintenance Tracking**:
    - Certain locations have a high number of bikes under maintenance, signaling potential inefficiencies in repair and redistribution processes.
5. **Revenue Optimization**:
    - Revenue is driven by short-term usage, as evidenced by high percentages for single rides and day passes, providing opportunities for promoting monthly pass subscriptions to enhance user retention.
6. **Demographics:**
    - A nearly equal gender distribution (Male: 48.73%, Female: 51.27%) shows balanced user engagement, indicating potential for gender-neutral marketing strategies.
7. **Bike Type Preference**:
    - E-bikes contribute significantly to trips and revenue, suggesting increased investment in e-bike availability and maintenance could boost growth.

NYC RESTAURANT INSPECTION ANALYTICS

Problem Statement:

The New York City Restaurant Inspection Results dataset faces challenges in maintaining food safety standards across its network of restaurants. Some establishments frequently fail inspections, while others consistently pass, leading to public health concerns and inefficiencies in the inspection system. Currently, the Department of Health and Mental Hygiene (DOHMH) does not have a consolidated way to analyze key factors like violation patterns, peak inspection times, and restaurant behavior to better manage food safety inspections.

To address this, we will create a centralized on-premise database that collects inspection data, violation details, and restaurant information. This will help the DOHMH analyze patterns, improve inspection allocation, and optimize restaurant management, ensuring food safety standards are maintained where and when they are needed most.

Database Overview:

The operational database for the NYC Restaurant Inspection Results is designed to collect and organize key information from every inspection, allowing for detailed analysis and decision-making. Each column in the database captures specific aspects of the restaurant inspection system, which can be leveraged to optimize operations and improve public health outcomes.

Dataset:
https://data.cityofnewyork.us/Health/DOHMH-New-York-City-Restaurant-Inspection-Results/43nn-pn8j/data

Columns:
• CAMIS: Unique identifier for each restaurant, useful for tracking inspection history and patterns.
• DBA: "Doing Business As" name of the restaurant, essential for identifying specific establishments.
• BORO: Borough where the restaurant is located, used for geographic analysis of inspection results.
• BUILDING / STREET / ZIPCODE: Address details of the restaurant, important for location-based analysis.
• PHONE: Contact information for the restaurant, useful for follow-up communications.
• CUISINE DESCRIPTION: Type of cuisine served, helping to analyze patterns across different food types.
• INSPECTION DATE: Date of the inspection, essential for identifying trends over time.
• ACTION: Outcome of the inspection (e.g., violations cited, establishment closed), crucial for tracking enforcement actions.
• VIOLATION CODE / VIOLATION DESCRIPTION: Specific health code violations found during inspection, key for analyzing common issues.
• CRITICAL FLAG: Indicates whether a violation is critical or not, important for prioritizing health risks.
• SCORE: Numerical score assigned based on violations, useful for quantitative analysis of restaurant performance.
• GRADE: Letter grade assigned based on the inspection score, essential for public communication of food safety standards.
• GRADE DATE: Date when the grade was issued, helpful for tracking grade history.
• RECORD DATE: Date when the record was added to the database, important for data management.
• INSPECTION TYPE: Indicates the type of inspection conducted, useful for categorizing different inspection scenarios.

We have an extensive dataset that covers key aspects such as inspection dates, violation details, restaurant information, and grading history

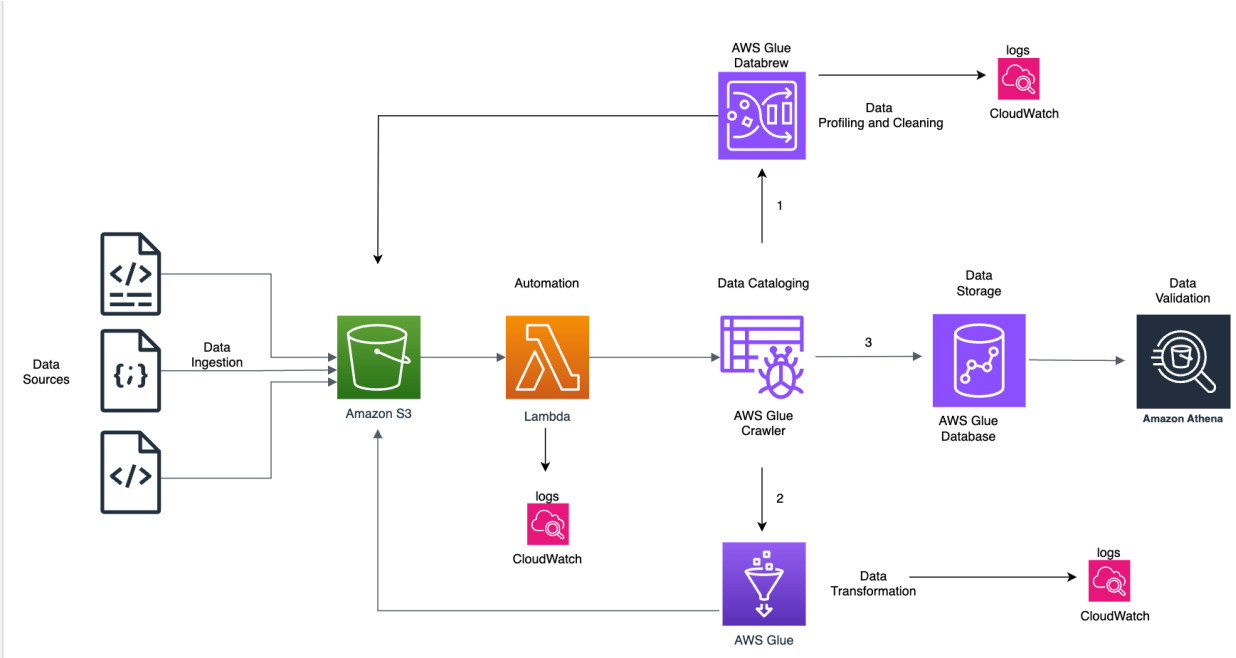
What We Hope to Analyze:

In the future, we hope to analyze restaurant behavior patterns to gain insights into violation trends, peak inspection times, and cuisine-specific issues for NYC restaurants. This analysis will help optimize inspection management by assessing violation distribution and inspection frequency needs, ensuring food safety standards are maintained where they are needed most.

Additionally, we aim to evaluate the effectiveness of the grading system, measuring its impact on restaurant compliance and public health outcomes. Overall, these insights will guide improvements in inspection service delivery and support the DOHMH's commitment to enhancing food safety in New York City.

- **Optimize Inspection Allocation:** Predict high-risk restaurants and recommend prioritization of inspections across establishments.
- **Peak Time Analysis:** Identify the busiest times of year/week for violations to optimize inspection scheduling and staffing.
- **Restaurant Behavior:** Analyze differences between consistently compliant restaurants and frequent violators to tailor education and enforcement strategies.
- **Violation Pattern Analysis:** Evaluate which types of violations are most common and plan for targeted education and enforcement efforts.

Data Pipeline Architecture:



Data Sources:

- **Nyc_Inspection.csv**
- **Date.json**

AWS Tools:

- **S3**
- **Glue**
- **Glue DataBrew**
- **Lambda**
- **Athena**
- **Cloudwatch**

Programming Language Used:

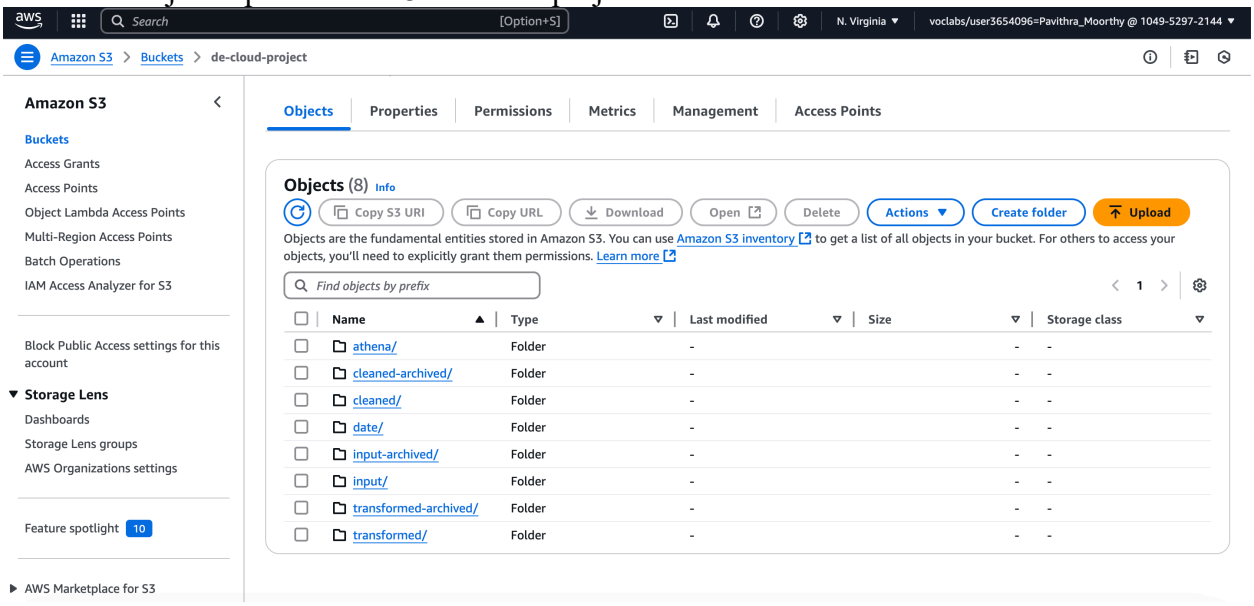
- **Python**

PIPELINE:

1. Upload File to S3/de-cloud-project/input

- **AWS Service**: **Amazon S3**
- The user uploads raw data files to the S3/de-cloud-project/input bucket. These could be CSV, JSON, or other types of raw data files that need to be processed through the pipeline.
- The input data is stored in an S3 bucket as a starting point for processing.

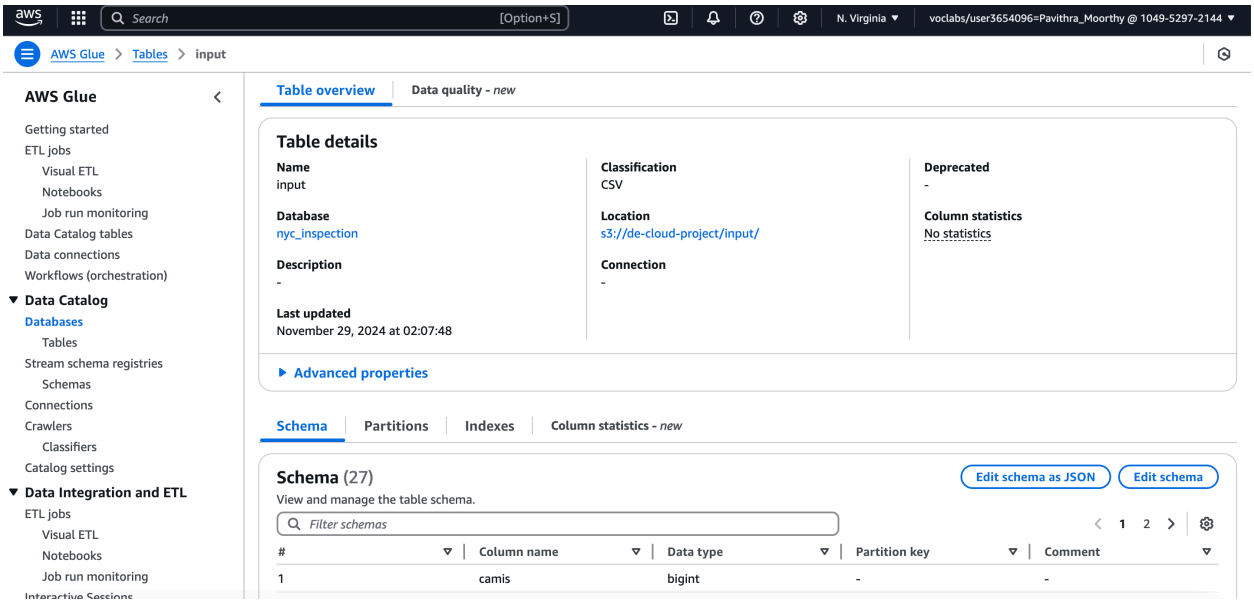
- Nyc_Inspection.csv uploaded to S3/de-cloud-project/input
- Date.json uploaded to S3/de-cloud-project/date



2. Data Crawler Job on S3/Bucket/Input

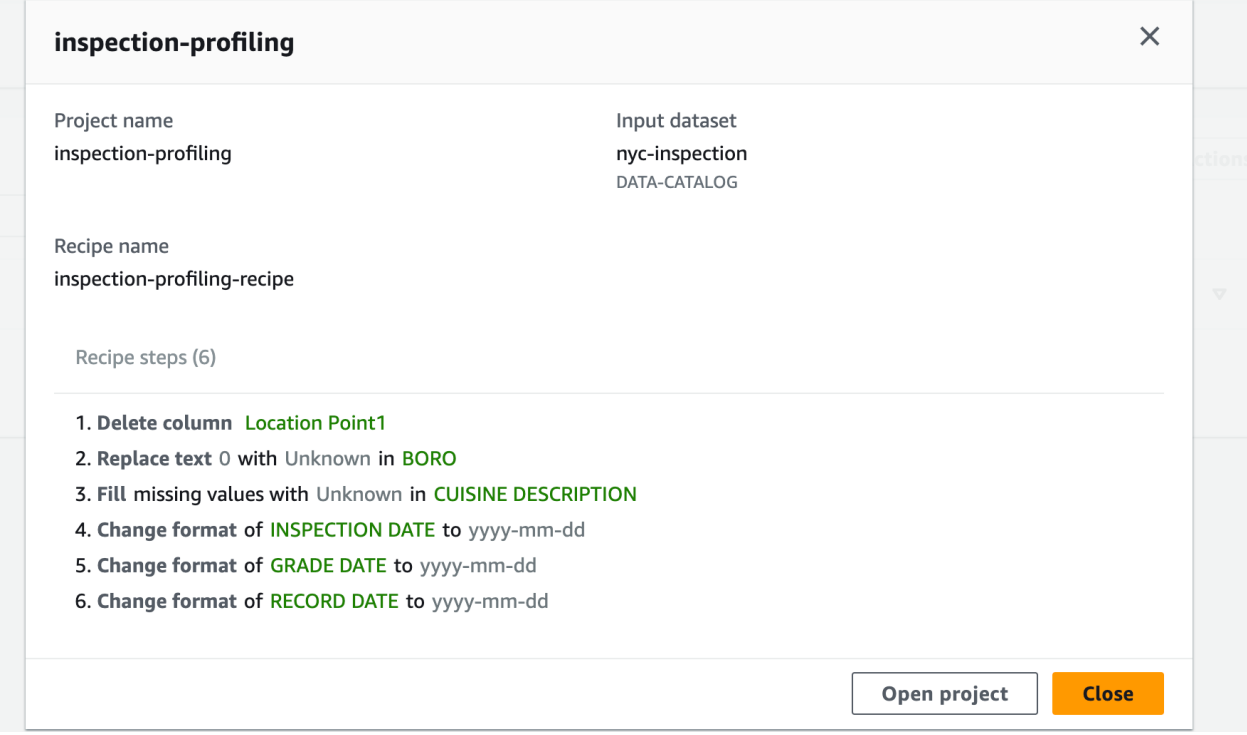
- **AWS Service**: **AWS Glue Crawler**

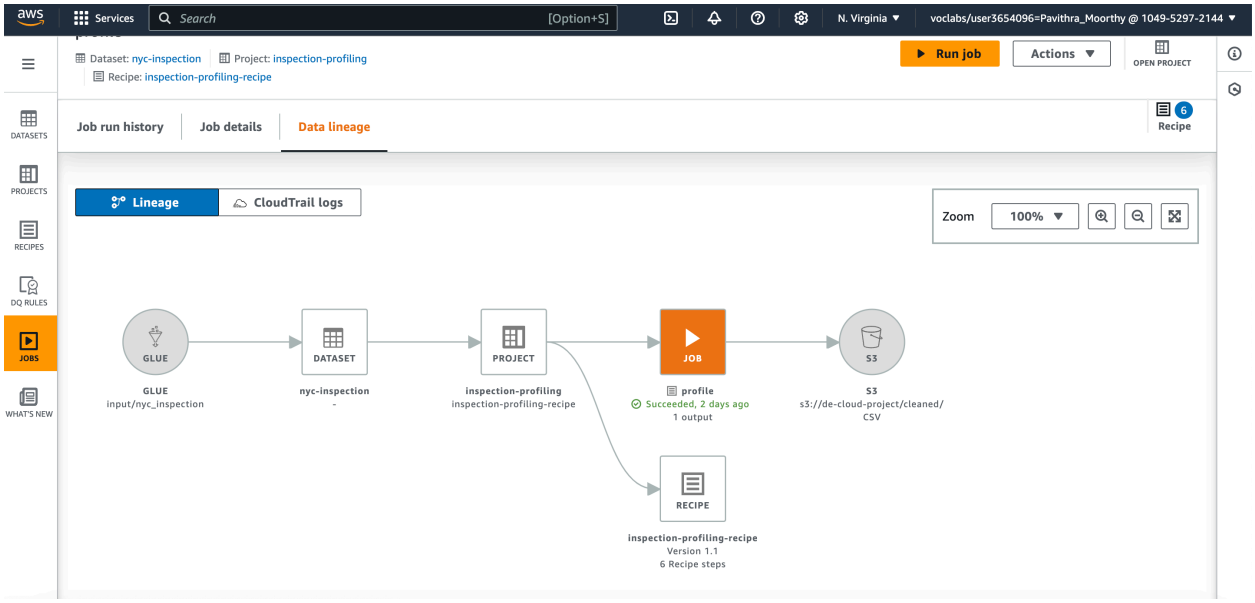
- The AWS Glue Crawler is responsible for discovering and cataloging the data in the S3/Bucket/Input. It inspects the files in the bucket, identifies their schema, and creates a catalog of the raw data.
- Create a catalog of the raw data in S3/de-cloud-project/input to be used for further processing and transformations.
- This step ensures that the metadata of the input data is available for any cleaning job that needs to run next



3. Data Brewer Job on S3/Bucket/Input Catalog

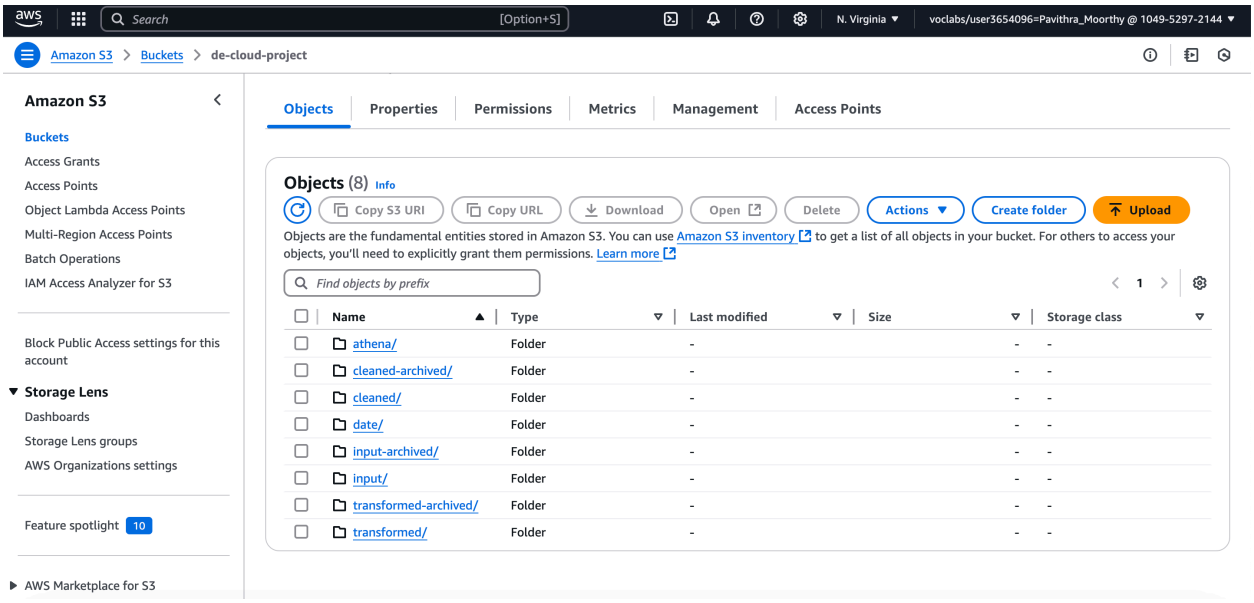
- **AWS Service**: **AWS Glue Databrew Job**
- The Data Brewer Job processes the raw data catalog created by the Glue crawler. This job applies profiling and cleaning to the raw data and stores the output in S3/de-cloud-project/cleaned.
- This job can include tasks such as filtering out unnecessary data, correcting formats, removing errors





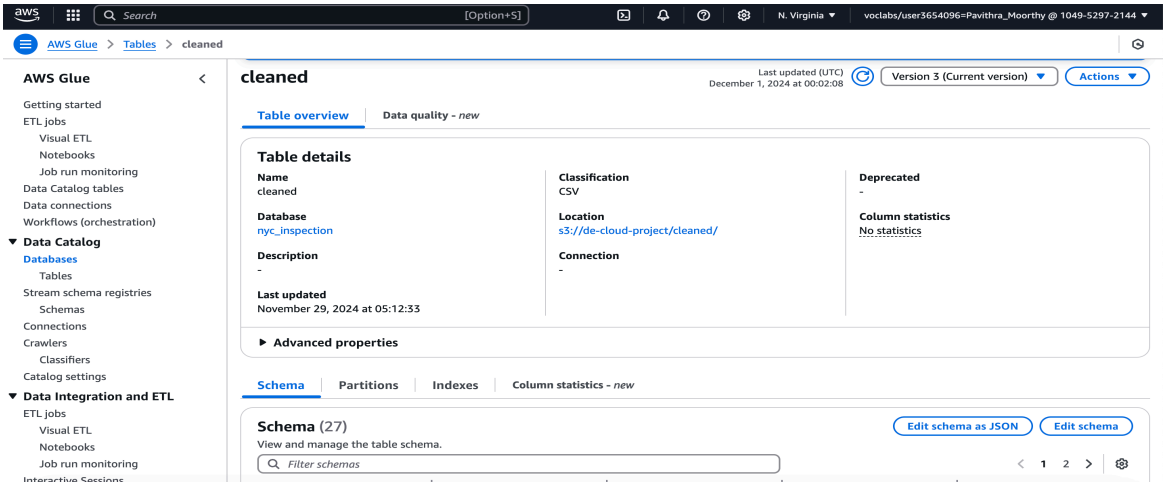
4. Output Cleaned Data to S3/Bucket/Clean

- **AWS Service**: **Amazon S3**
- The clean data is stored in S3/Bucket/Clean, where it is now available for further processing or analysis.
- Store the cleaned data after processing.



5. Data Crawler Job on S3/Bucket/Clean

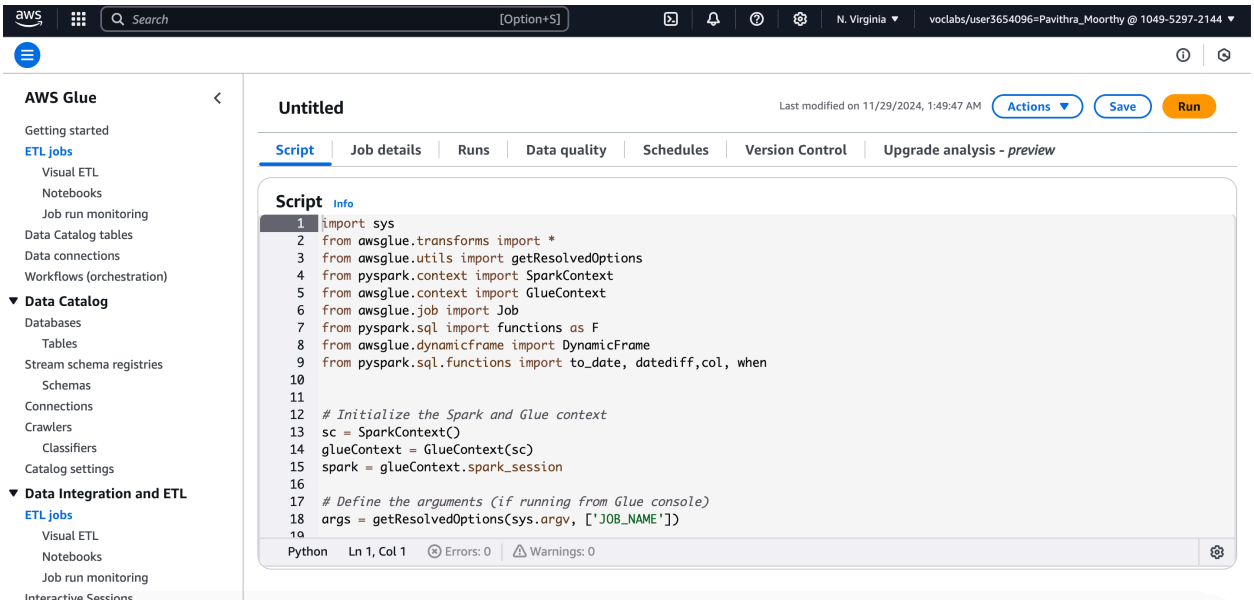
- **AWS Service**: **AWS Glue Crawler**
- The AWS Glue Crawler scans the cleaned data in S3/Bucket/Clean, updating or creating a new catalog in the Glue Data Catalog. It inspects the data format and structure of the cleaned data to ensure it is ready for further processing.
- Create a metadata catalog of the cleaned data to be used for the ETL jobs.
- This step ensures that the metadata of the cleaned data is available for any ETL job that needs to run next.



6. Run ETL Job on Data from S3/Bucket/Clean

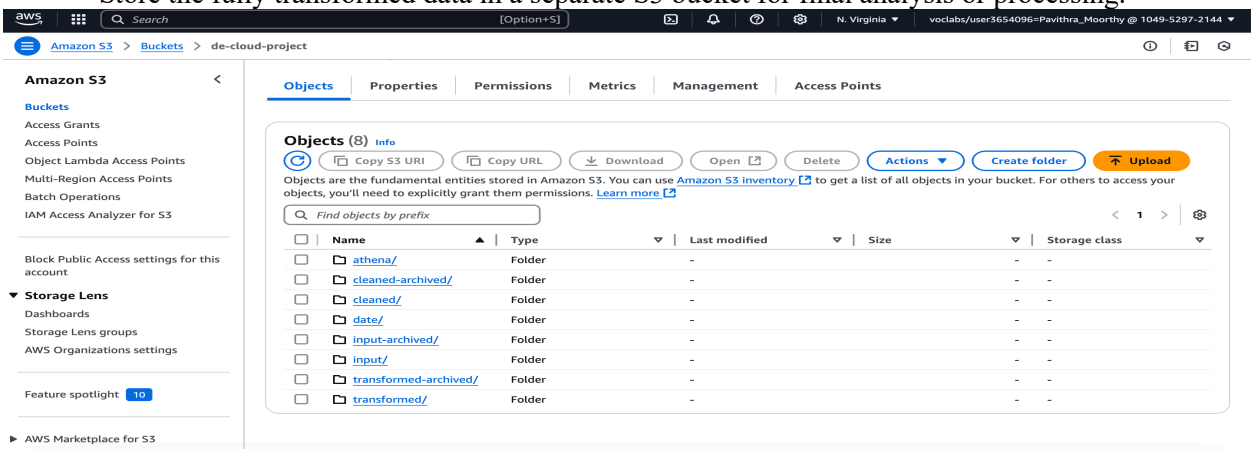
- **AWS Service**: **AWS Glue ETL Jobs**
- The ETL Job runs on the cleaned data cataloged by the Glue Crawler. This job performs Extract, Transform, and Load operations to transform the cleaned data into its final form.
- Transform the cleaned data into a format suitable for analytics, reporting, or machine learning, and store the results in the S3/Bucket/Transformed.
- Cleaned data and dates.json is combined

- Necessary fields are kept, others removed
- New measures (days took to give the results) are calculated



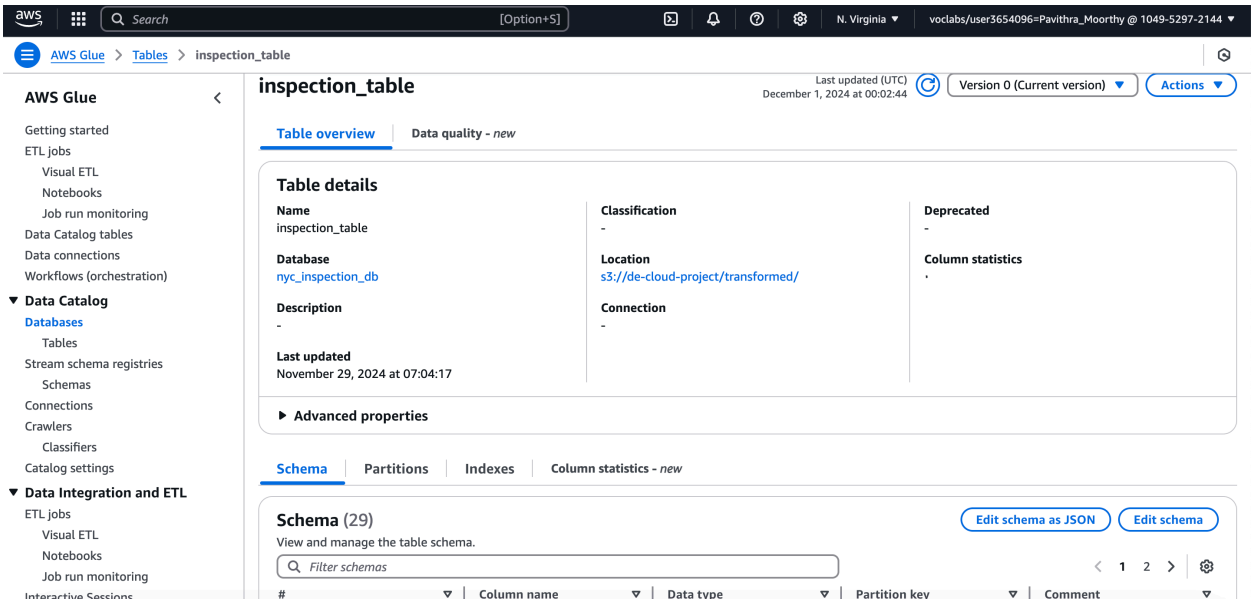
7. Output Transformed Data to S3/Bucket/Transformed

- **AWS Service**: **Amazon S3**
- The ETL process stores the transformed data in S3/Bucket/Transformed. This is the final processed data that is ready for analytics or further processing.
- Store the fully transformed data in a separate S3 bucket for final analysis or processing.



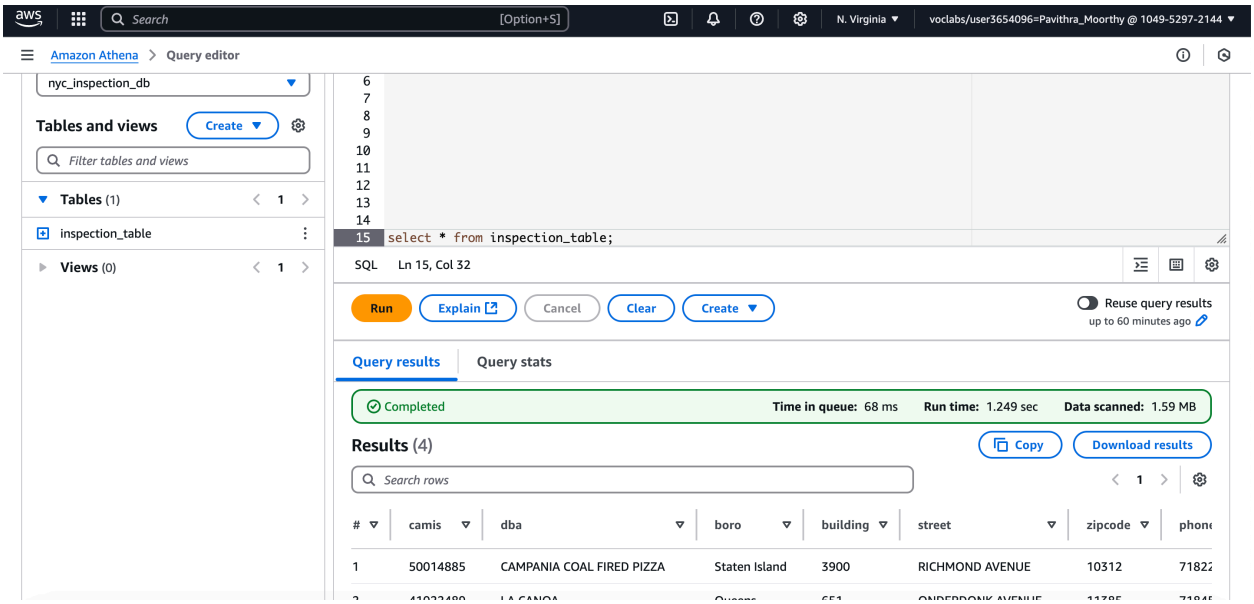
8. Create Glue Crawler and Table for S3/Bucket/Transformed Data

- **AWS Service**: **AWS Glue Crawler**
- The **AWS Glue Crawler** is invoked to scan the transformed data in the S3/Bucket/Transformed creating a new catalog in the Glue Data Catalog.
- Create a metadata catalog of the cleaned data to be used for the creating tables.



9. Querying using Athena

- **AWS Service: AWS Athena**
- To validate and query the database

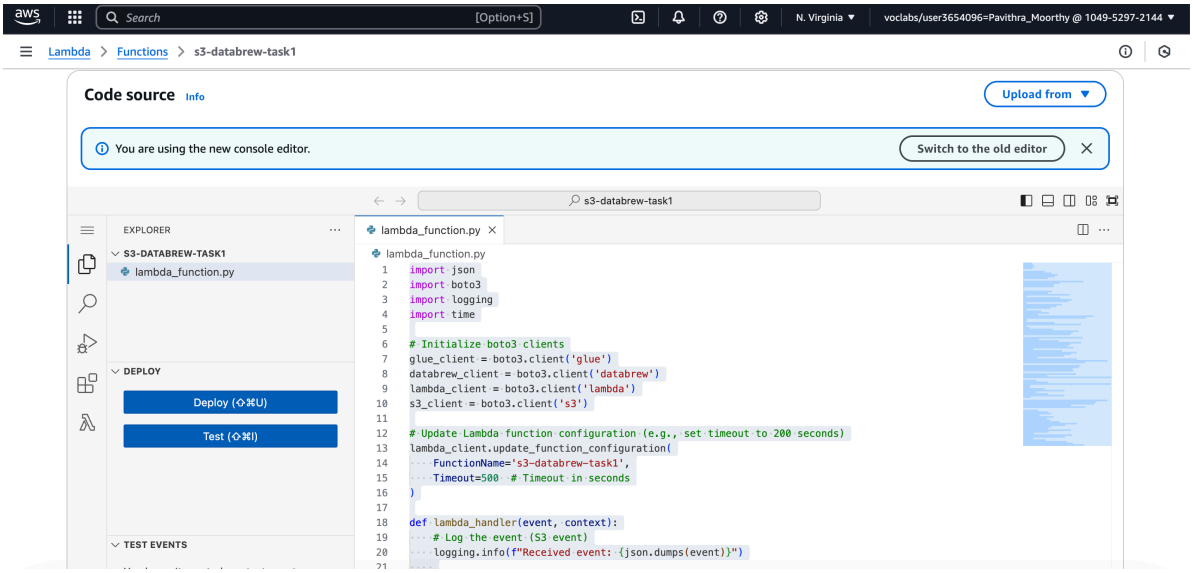


LAMBDA FUNCTIONS:

1. Lambda Function Trigger on upload File in S3/Bucket/Input

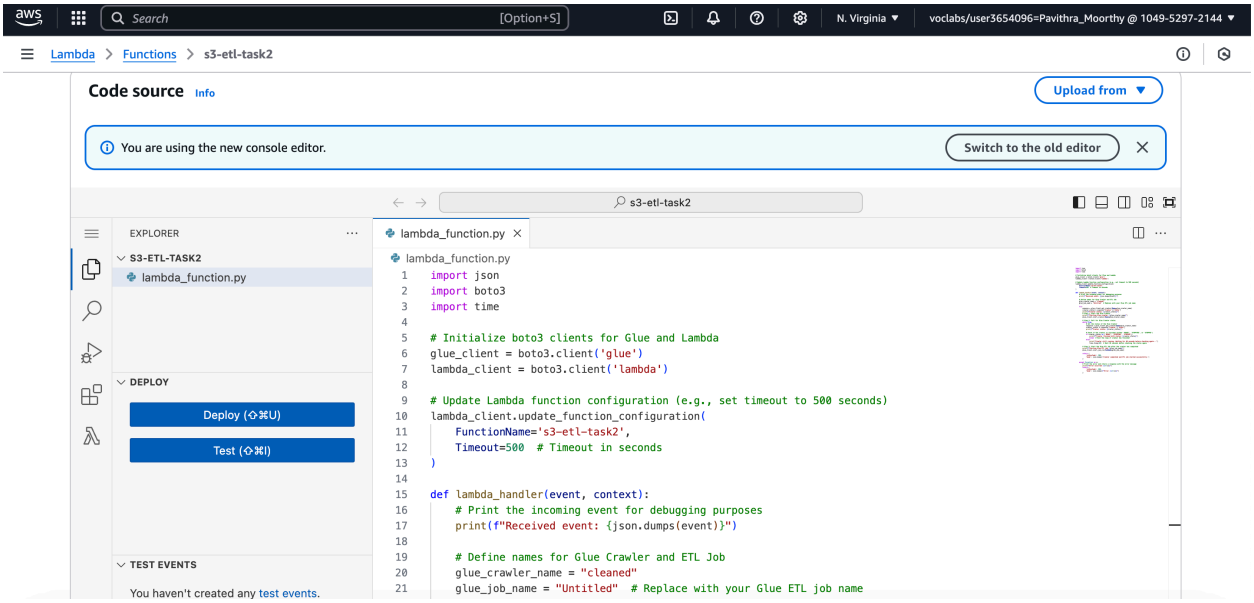
- A Lambda function is triggered automatically when a new file is uploaded to S3/Bucket/Input. The function initiates the glue data crawler and glue data brewer jobs. And Move files from

S3/Bucket /transformed to S3/Bucket /transformed-archived and move files from S3/Bucket/cleaned to S3/Bucket/cleaned-archived



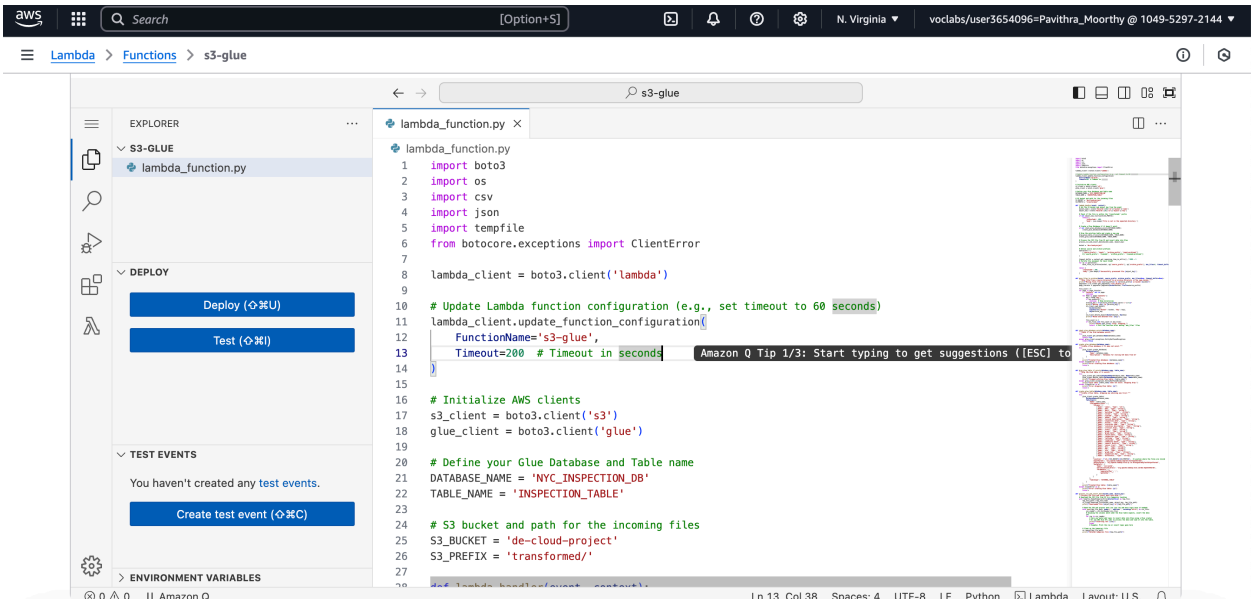
2. Lambda Function Trigger on upload File in S3/Bucket/Clean

- A Lambda function is triggered when a new file is uploaded to S3/Bucket/Clean. This function starts the next set of tasks, including running the glue data crawler on the clean data and performing the glue ETL job process.



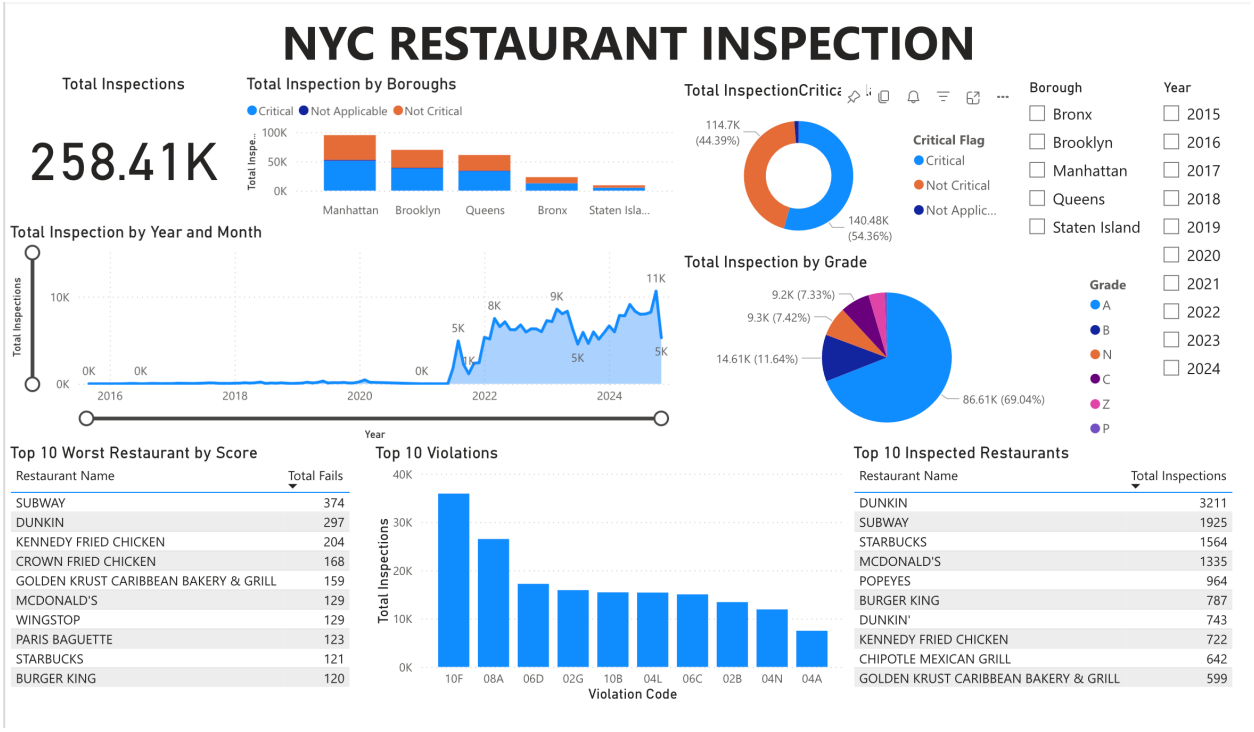
3. Lambda Function Trigger on upload File in S3/Bucket/Transformed

- A Lambda function is triggered when a new file is uploaded to S3/Bucket/Transformed. This function will trigger the Glue Data Crawler job and create database and table and insert to Glue Database visa Athena to use in analytics or querying.



4. Cloudwatch: Consists of all the logs and outputs of all jobs.

DASHBOARD:



GOAL - The goal is to improve food safety compliance by reducing critical violations by 10%, increasing Grade A establishments by 15%, optimizing inspection resource allocation across boroughs, and implementing corrective measures for repeat offenders, ensuring public health and equitable regulatory enforcement.

KPI's

Total Inspections:
- 258.41K inspections conducted to date, indicating extensive monitoring efforts.

Inspection Criticality:
- **Critical**: 54.36% (140.48K inspections).
- **Not Critical**: 45.64% (114.7K inspections).
- Highlights the proportion of critical issues requiring immediate action.

Grade Distribution:
- **A**: 69.04% (86.61K inspections).
- **B**: 11.64% (14.61K).
- Majority of establishments are compliant with hygiene and safety standards.

Top 10 Inspected Restaurants:
- Dunkin' leads with 3,211 inspections, followed by Subway (1,925) and Starbucks (1,564).
- Reflects targeted inspections on popular chains.

Worst Offenders by Failures:
- Subway: 374 failures.
- Dunkin': 297 failures.
- Highlights problematic establishments requiring focused corrective measures.

Violation Analysis:
- Top Violation Code: "10F" (40K cases).
- Indicates recurring food safety or hygiene issues.

Metrics:

1. **Inspection Trends**:
    - Line chart illustrating inspection growth from 2016 to 2024.
    - Recent peak in inspections (11K in late 2024), showing increased regulatory activity.
2. **Year-Wise/ Borough-Wise Drill Down Analysis**:
    - Interactive filters allow users to analyze total inspections by specific years, revealing trends over time.
    - Enables granular analysis of inspection activity within each borough
3. **Borough-Wise Inspections**:
    - Manhattan: Highest activity across boroughs.
    - Staten Island: Minimal activity.
    - Enables geographic analysis of compliance efforts.
4. **Violation Codes**:
    - Bar chart showing top 10 violation codes.
    - Common violations highlight areas of non-compliance, aiding targeted training.
5. **Grade Breakdown**:
    - Pie chart distribution of grades (A, B, Z, etc.).
    - Majority receive "A," but ~30% require improvements.
6. **Restaurant Rankings**:
    - Top inspected chains like Dunkin' and Subway are both heavily monitored and among the worst offenders.

Insights:

1. **Training and Education**:
    - The high proportion of "Grade B" (11.64%) and other grades indicates a need for training on food safety and hygiene standards to improve compliance.

2. **Focus on Critical Violations**:
    - Over 54% of inspections flagged as critical highlight a significant proportion of establishments with serious violations.
3. **Target Problematic Establishments**:
    - High failure rates in chains like Subway (374 failures) and Dunkin' (297 failures) indicate a need for targeted audits and enforcement to address recurring compliance issues
4. **Violation Patterns**:
    - High counts for specific violations (e.g., “10F”) suggest recurring issues like improper food handling or hygiene.
5. **Geographic Strategy**:
    - Manhattan sees the highest inspection activity, suggesting a need for rebalancing resources to focus on underserved boroughs like Staten Island, which may face lower regulatory oversight.
6. **Seasonal and Monthly Trends**:
    - Inspection peaks toward the end of the year (e.g., Nov 2024 with 11K inspections) indicate potential operational stress or intensified campaigns. Planning inspection cycles evenly across months may enhance efficiency.
7. **Resource Allocation**:
    - Chains with high inspection counts like Dunkin' (3,211 inspections) and Subway (1,925 inspections) likely consume significant inspection resources. Analyzing these inspection frequencies against the risk can improve resource efficiency.