Project Milestone 6

LogiStream: DataCo’s Live Flow

IE 6760 Data Warehousing & Integration

Group 4

Student 1: Neha Patil

Student 2: Medhavi Uday Pande

Percentage of Effort Contributed by Student 1: 50%

Percentage of Effort Contributed by Student 2: 50%

Signature of Student 1: Neha Patil

Signature of Student 2: Medhavi Uday Pande

Problem Statement

DataCo is a global e-commerce company with a complex, high-volume supply chain. While it possesses a rich historical dataset of past orders (the 180,000+ records in the CSV), its analytics are purely retrospective. Management can only analyze what went wrong (e.g., late deliveries, fraud) weeks after the fact.

The company has no real-time visibility into its operations. It cannot proactively identify a shipment that is at risk of being late, nor can it detect fraudulent order patterns as they occur. The core problem is the inability to ingest, process, and analyze high-velocity transactional and shipping data as it is generated.

This project aims to design a cloud data pipeline to solve this. The pipeline will be responsible for ingesting live data, transforming it in-stream, and combining it with historical and geospatial reference data to power real-time operational dashboards and alerts.

Problem Definition

This project will demonstrate a modern data pipeline by ingesting data with significant volume and variety. The pipeline must handle four distinct types of data sources:

- Historical Transactional Data (Initial Batch Load): The DataCoSupplyChainDatasetRefined.csv file, serving as a one-time bulk load of ~180,000+ historical orders to provide historical context.
- Real-time Order Stream (Live Pipeline Feed): A simulated high-velocity stream (e.g., from an e-commerce API) generating JSON events for new orders as they are placed.
- Real-time Shipping Stream (Live Pipeline Feed): A simulated high-velocity feed from logistics partners (e.g., FedEx/UPS API) providing JSON-based shipping status and location updates.
- Static Reference Data (Enrichment Data): Low-volume, read-only data, including the provided GeoJSON files (routes.geojson, etc.) and dimensional tables (Customer, Product) used to enrich the live data stream.

Data Source:

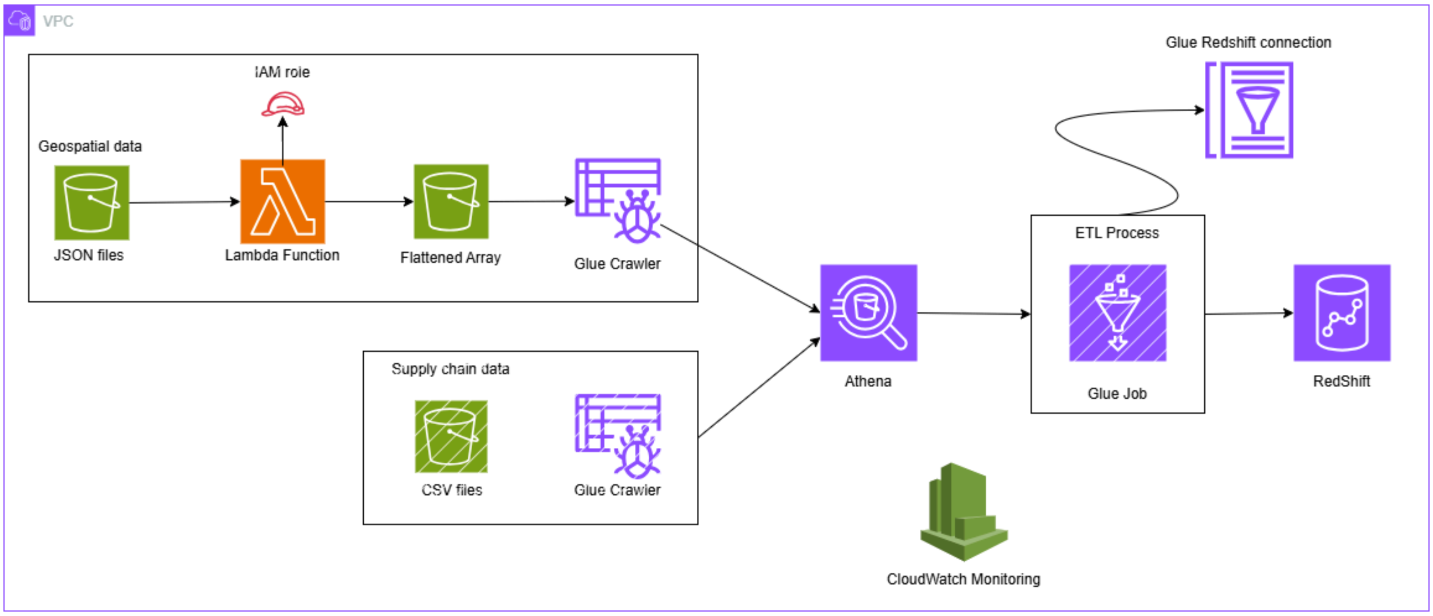
https://www.kaggle.com/datasets/aaumgupta/refined-dataco-supply-chain-geospatial-dataset

Data Pipeline

The cloud data pipeline is optimized for both structured and semi-structured data transformation. We are connecting a complex JSON source (GeoJSON) directly into the dimensional modeling process and another raw data file too. We have ingested data, transformed it using multiple AWS Services and combined it with historical and geospatial reference data to eventually enable real-time operational dashboard.

- **Ingestion Layer:** Loaded CSV (raw_data, metadata) and GeoJSON (routes.geojson) files into the Source S3 Bucket.
- **Transformation (Programming):** Used AWS Lambda Function (Python) to process the nested GeoJSON (routes.geojson), flatten the coordinates into WKT strings, and save the result as a usable CSV file back to S3.
- **Data Cataloging:** Used AWS Glue Crawlers to process the structured CSV files and the processed WKT files, loading the schemas into the Glue Data Catalog (logistream_db).
- **Data Verification:** The Data Catalog Tables were queried in Amazon Athena to validate schema integrity and data readability across all sources.
- **Pipeline Execution:** Created ETL jobs on AWS Glue ETL using a PySpark script editor, which extracted the data from the Catalog, performed dimensional modeling (Snowflake joins), and loaded the final data into the Redshift Serverless cluster.
- **Tool Configuration:** IAM Roles were set up for Lambda and Glue with necessary permissions (e.g., S3 read/write, Redshift access).
- **Documentation/Monitoring:** CloudWatch Logs were connected to monitor and troubleshoot the execution of the AWS Glue ETL Jobs.

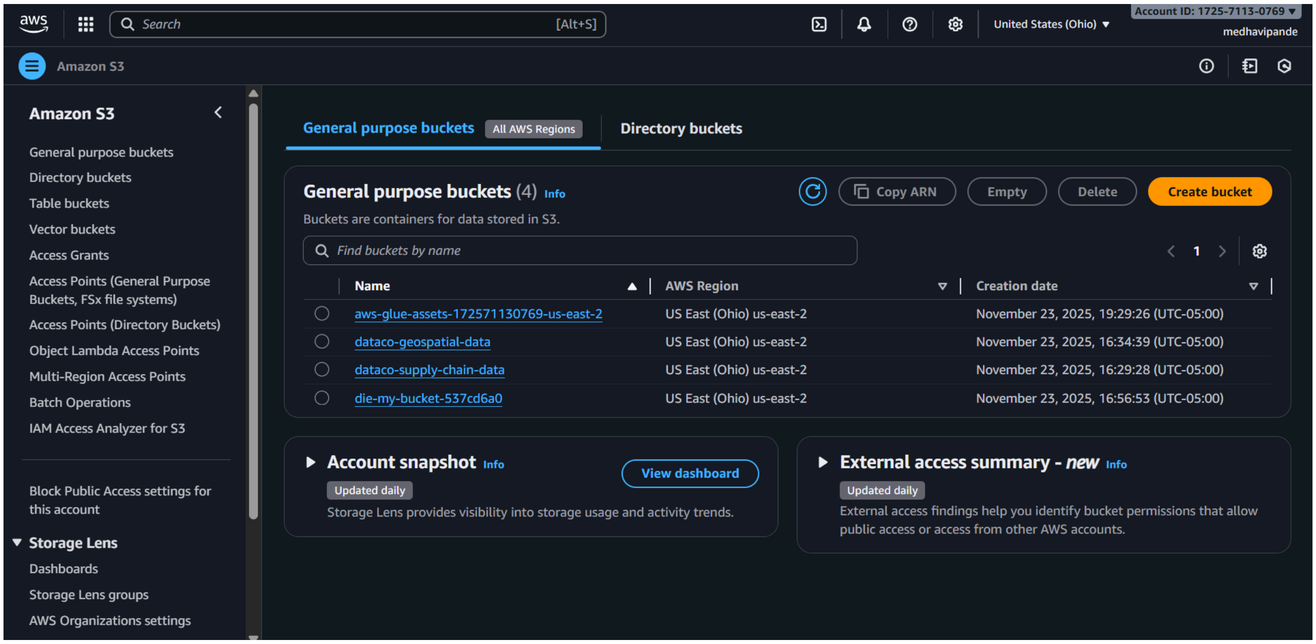
Architecture Diagram



A. Data Ingestion & Cataloging (Source Layer)

1. S3 Data Loading & Organization

All source files were loaded into a central S3 bucket and organized into 2 distinct folders: raw_data/ (main CSV), metadata/ (description CSV) in dataco-supply-chain-data and geojson/ (raw GeoJSON) in dataco-geospatial-data. Please refer to buckets 2 & 3 in the below screenshot.



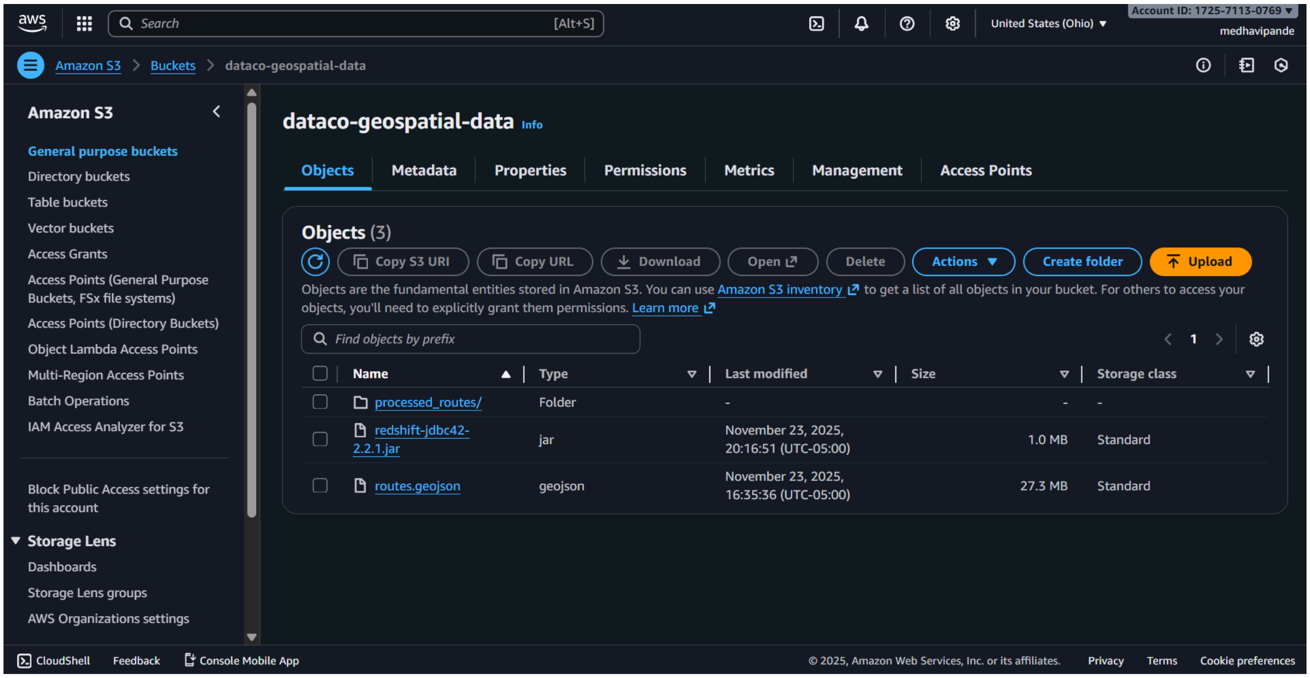
Buckets are containers for data stored in S3.

Name	AWS Region	Creation date
○ aws-glue-assets-172571130769-us-east-2	US East (Ohio) us-east-2	November 23, 2025, 19:29:26 (UTC-05:00)
○ dataco-geospatial-data	US East (Ohio) us-east-2	November 23, 2025, 16:34:39 (UTC-05:00)
○ dataco-supply-chain-data	US East (Ohio) us-east-2	November 23, 2025, 16:29:28 (UTC-05:00)
○ die-my-bucket-537cd6a0	US East (Ohio) us-east-2	November 23, 2025, 16:56:53 (UTC-05:00)

Storage Lens provides visibility into storage usage and activity trends.

External access findings help you identify bucket permissions that allow public access or access from other AWS accounts.

Dataco-geospatial-data : Original source file is routes.geojson

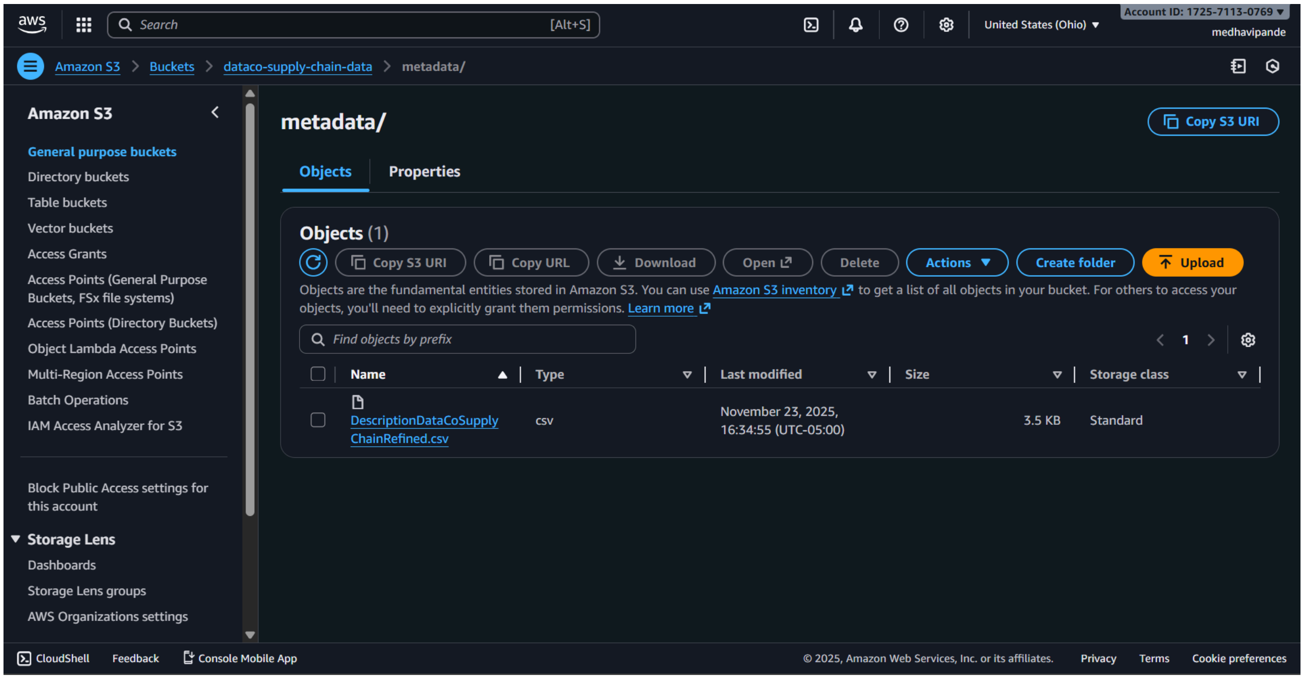


Objects are the fundamental entities stored in Amazon S3. You can use Amazon S3 inventory to get a list of all objects in your bucket. For others to access your objects, you'll need to explicitly grant them permissions. Learn more

□	Name	Type	Last modified	Size	Storage class
□	processed_routes/	Folder	-	-	-
□	redshift-jdbc42-2.2.1.jar	jar	November 23, 2025, 20:16:51 (UTC-05:00)	1.0 MB	Standard
□	routes.geojson	geojson	November 23, 2025, 16:35:36 (UTC-05:00)	27.3 MB	Standard

Dataco-supply-chain-data : Original source files are inside folders - metadata & rawdata.

□ metadata/DescriptionDataCoSupplyChainRefined.csv



Amazon S3 > Buckets > dataco-supply-chain-data > metadata/

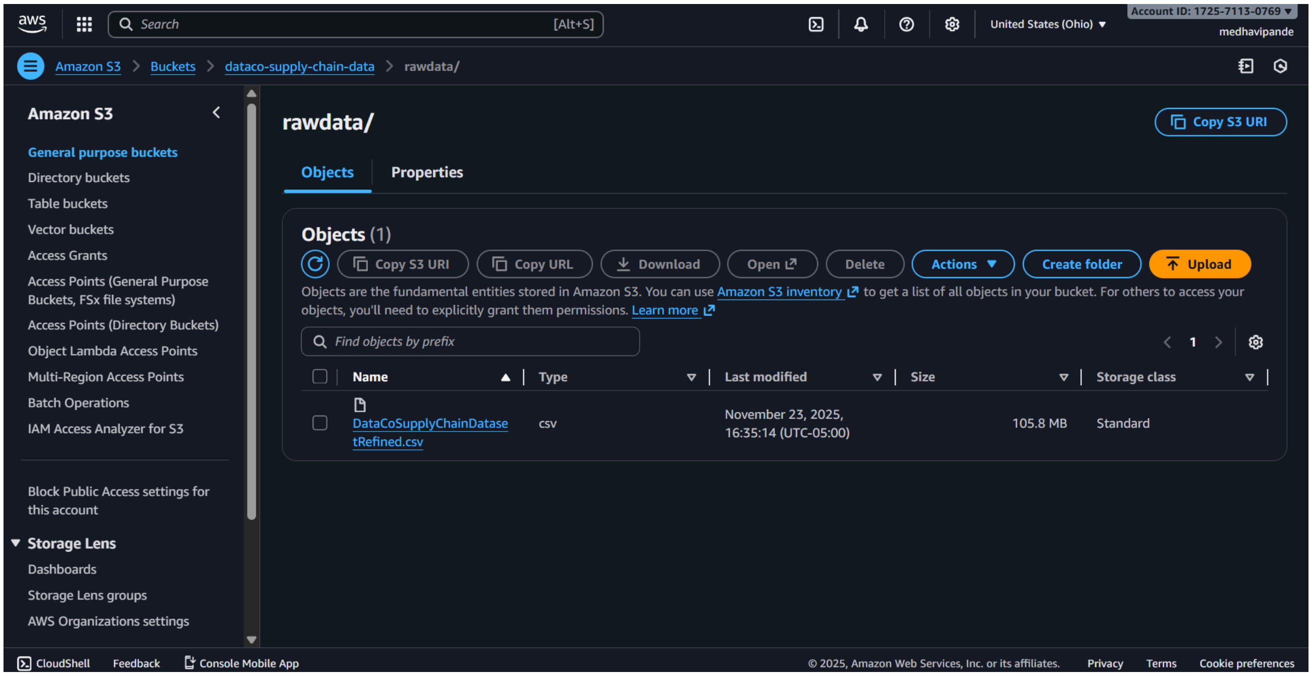
metadata/

Objects (1)

Objects are the fundamental entities stored in Amazon S3. You can use Amazon S3 inventory to get a list of all objects in your bucket. For others to access your objects, you'll need to explicitly grant them permissions. Learn more

Name	Type	Last modified	Size	Storage class
DescriptionDataCoSupplyChainRefined.csv	csv	November 23, 2025, 16:34:55 (UTC-05:00)	3.5 KB	Standard

□ rawdata/DataCoSupplyChainDatsetRefined.csv



Amazon S3 > Buckets > dataco-supply-chain-data > rawdata/

rawdata/

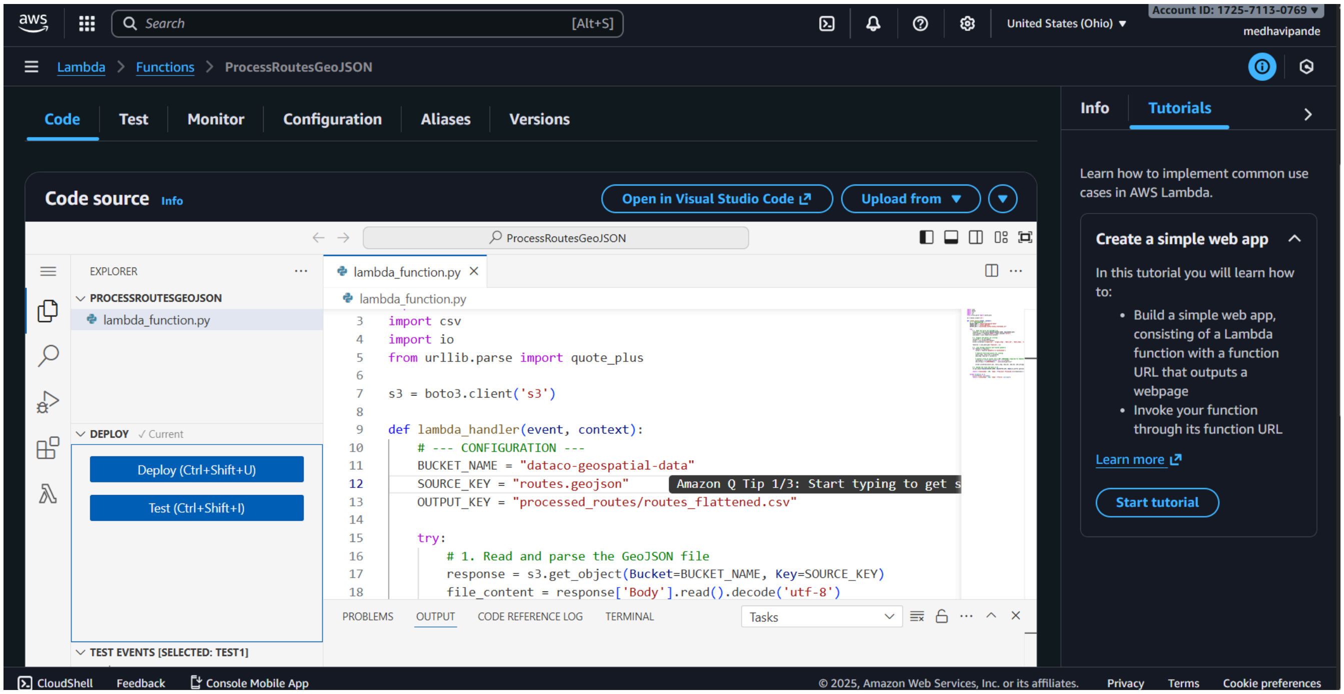
Objects (1)

Objects are the fundamental entities stored in Amazon S3. You can use Amazon S3 inventory to get a list of all objects in your bucket. For others to access your objects, you'll need to explicitly grant them permissions. Learn more

Name	Type	Last modified	Size	Storage class
DataCoSupplyChainDatasetRefined.csv	csv	November 23, 2025, 16:35:14 (UTC-05:00)	105.8 MB	Standard

2. GeoJSON Pre-Processing (Lambda Transformation)

An AWS Lambda function (written in Python) was implemented to handle the complex, nested GeoJSON structure. This function reads the raw routes.geojson file, flattens the nested coordinate array into a Well-Known Text (WKT) string, and saves the output as a clean CSV file in a new S3 folder (processed_routes/).



```python
import csv
import io
from urllib.parse import quote_plus

s3 = boto3.client('s3')

def lambda_handler(event, context):
    # --- CONFIGURATION ---
    BUCKET_NAME = "dataco-geospatial-data"
    SOURCE_KEY = "routes.geojson"
    OUTPUT_KEY = "processed_routes/routes_flattened.csv"

    try:
        # 1. Read and parse the GeoJSON file
        response = s3.get_object(Bucket=BUCKET_NAME, Key=SOURCE_KEY)
        file_content = response['Body'].read().decode('utf-8')
```

Learn how to implement common use cases in AWS Lambda.

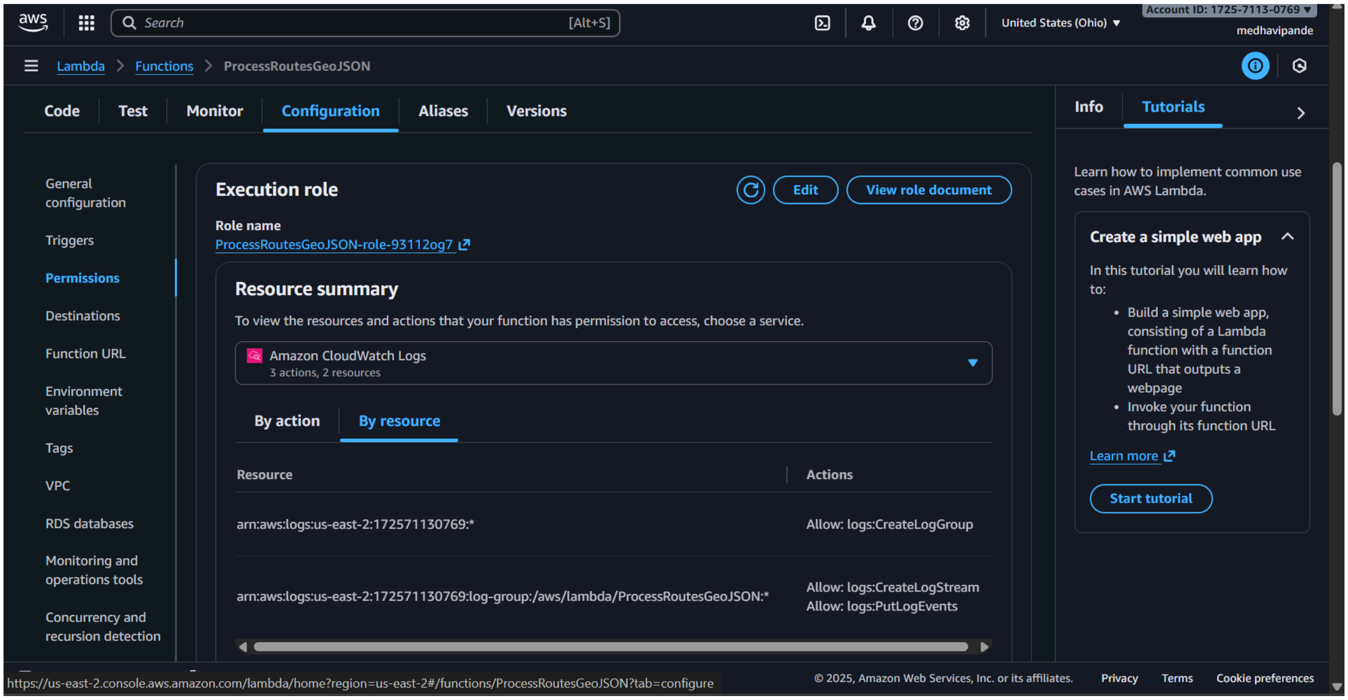
Create a simple web app

In this tutorial you will learn how to:

- Build a simple web app, consisting of a Lambda function with a function URL that outputs a webpage
- Invoke your function through its function URL

Learn more

Start tutorial


aws
Search [Alt+S]
United States (Ohio) ▾
Account ID: 1725-7113-0769 ▾
medhavipande
Lambda > Functions > ProcessRoutesGeoJSON
Code
Test
Monitor
Configuration
Aliases
Versions
General configuration
Triggers
Permissions
Destinations
Function URL
Environment variables
Tags
VPC
RDS databases
Monitoring and operations tools
Concurrency and recursion detection
Edit
View role document
By action
By resource
Info
Tutorials
https://us-east-2.console.aws.amazon.com/lambda/home?region=us-east-2#/functions/ProcessRoutesGeoJSON?tab=configure
© 2025, Amazon Web Services, Inc. or its affiliates.
Privacy
Terms
Cookie preferences

Execution role

Role name
ProcessRoutesGeoJSON-role-93112og7

Resource summary

To view the resources and actions that your function has permission to access, choose a service.

Amazon CloudWatch Logs
3 actions, 2 resources

Resource	Actions
arn:aws:logs:us-east-2:172571130769:*	Allow: logs:CreateLogGroup
arn:aws:logs:us-east-2:172571130769:log-group:/aws/lambda/ProcessRoutesGeoJSON:*	Allow: logs:CreateLogStream Allow: logs:PutLogEvents

Learn how to implement common use cases in AWS Lambda.

Create a simple web app

In this tutorial you will learn how to:

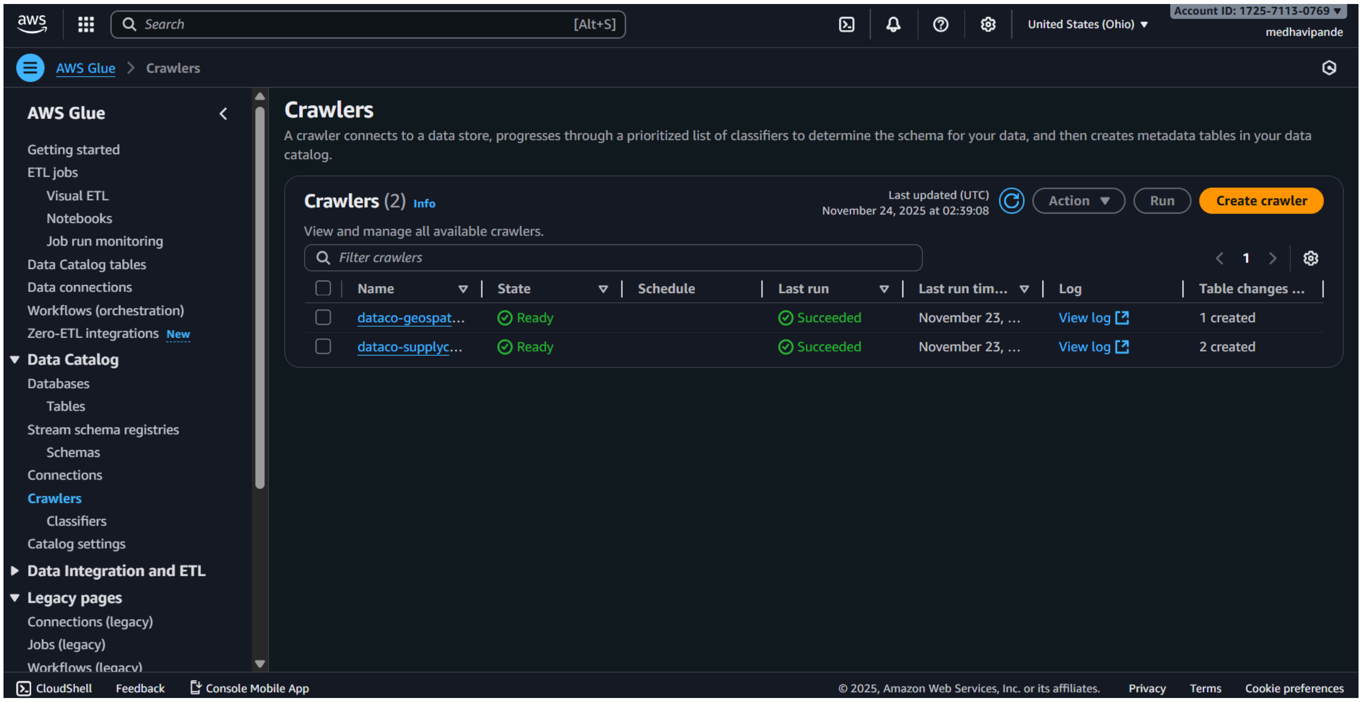
- Build a simple web app, consisting of a Lambda function with a function URL that outputs a webpage
- Invoke your function through its function URL

Learn more

Start tutorial

3. Data Crawlers & Cataloging using Glue

AWS Glue Crawlers were executed to register schemas into the Glue Data Catalog (logistream). One crawler scanned the structured CSV folders (rawdata/ and metadata/). A separate crawler was executed specifically over the processed_routes/ folder. This step ensures the complex geospatial data, now simplified by Lambda, is correctly cataloged as a standard table (processed_routes) ready for joining.



Crawlers

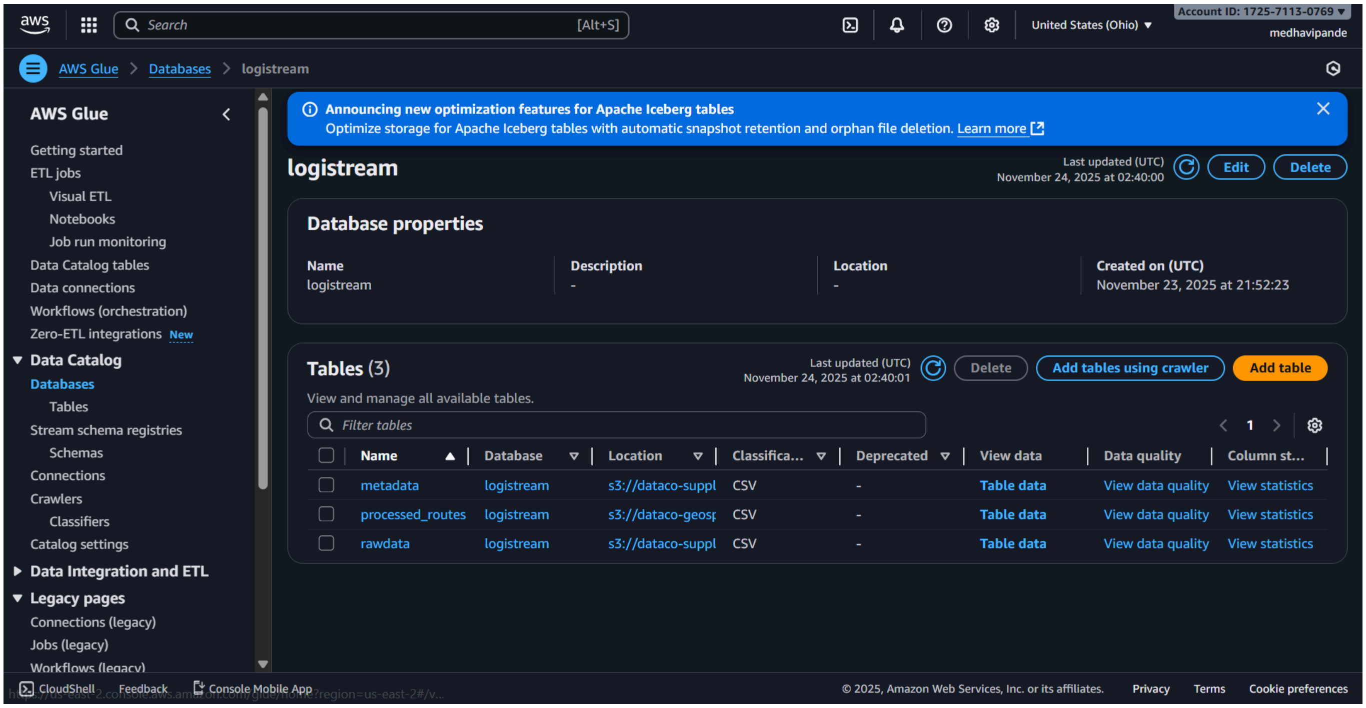
A crawler connects to a data store, progresses through a prioritized list of classifiers to determine the schema for your data, and then creates metadata tables in your data catalog.

Crawlers (2) Info

Last updated (UTC)
November 24, 2025 at 02:39:08

View and manage all available crawlers.

□	Name	State	Schedule	Last run	Last run tim...	Log	Table changes ...
□	dataco-geospat...	✓ Ready		✓ Succeeded	November 23, ...	View log	1 created
□	dataco-supplyc...	✓ Ready		✓ Succeeded	November 23, ...	View log	2 created



Announcing new optimization features for Apache Iceberg tables
Optimize storage for Apache Iceberg tables with automatic snapshot retention and orphan file deletion. Learn more

logistream

Last updated (UTC)
November 24, 2025 at 02:40:00

Database properties

Name	Description	Location	Created on (UTC)
logistream	-	-	November 23, 2025 at 21:52:23

Tables (3)

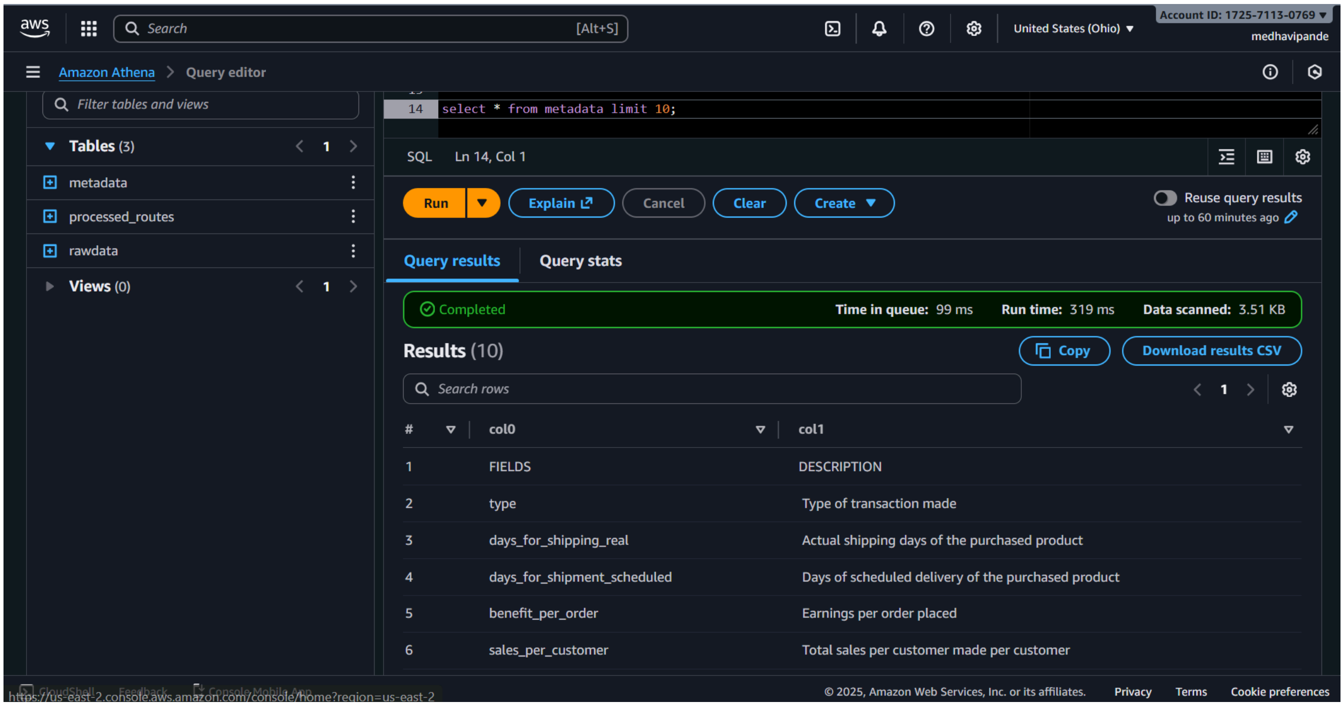
Last updated (UTC)
November 24, 2025 at 02:40:01

View and manage all available tables.

□	Name ▲	Database	Location	Classifica...	Deprecated	View data	Data quality	Column st...
□	metadata	logistream	s3://dataco-suppl	CSV	-	Table data	View data quality	View statistics
□	processed_routes	logistream	s3://dataco-geosp	CSV	-	Table data	View data quality	View statistics
□	rawdata	logistream	s3://dataco-suppl	CSV	-	Table data	View data quality	View statistics

4. Data Catalog Verification (Athena Querying)

The resulting tables (rawdata, metadata, processed_routes) were explicitly queried using Amazon Athena. Querying the data directly in Athena provided immediate schema validation and confirmed that all sources were correctly linked and readable before the heavy ETL phase began.

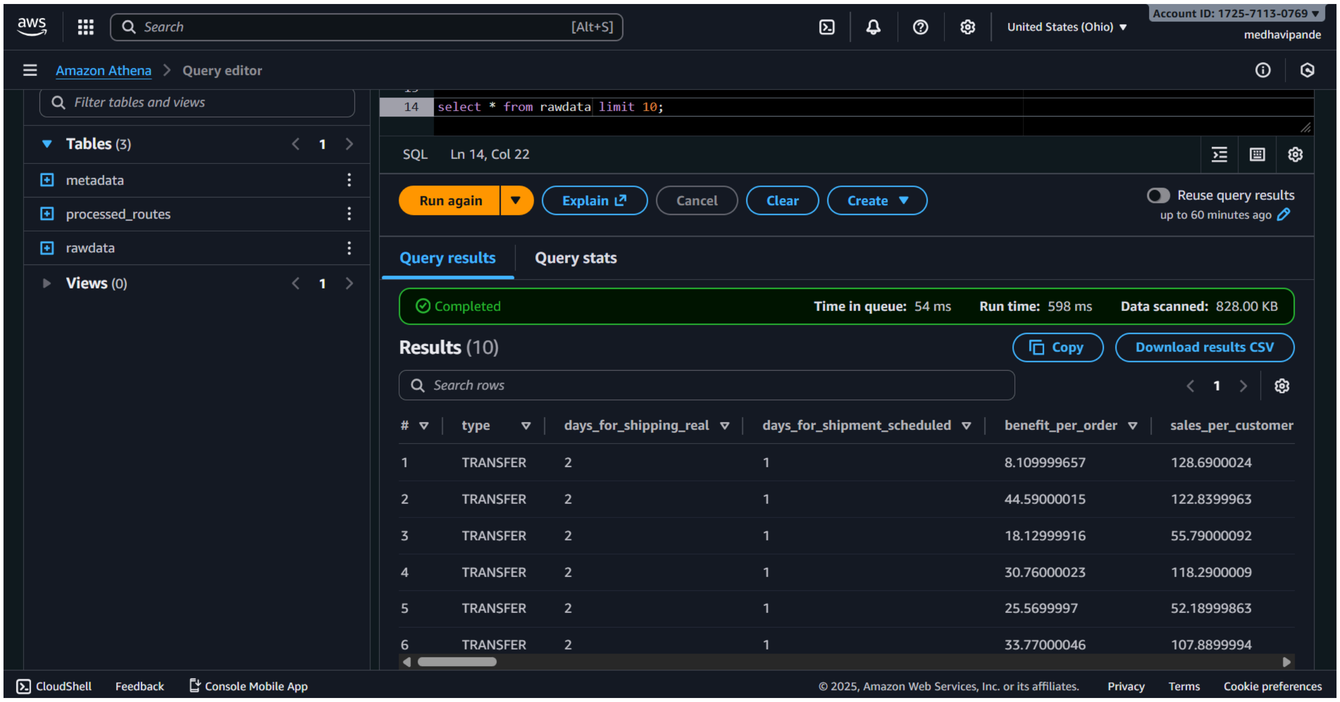


```
select * from metadata limit 10;
```

✓ Completed   Time in queue: 99 ms   Run time: 319 ms   Data scanned: 3.51 KB

Results (10)

#	col0	col1
1	FIELDS	DESCRIPTION
2	type	Type of transaction made
3	days_for_shipping_real	Actual shipping days of the purchased product
4	days_for_shipment_scheduled	Days of scheduled delivery of the purchased product
5	benefit_per_order	Earnings per order placed
6	sales_per_customer	Total sales per customer made per customer

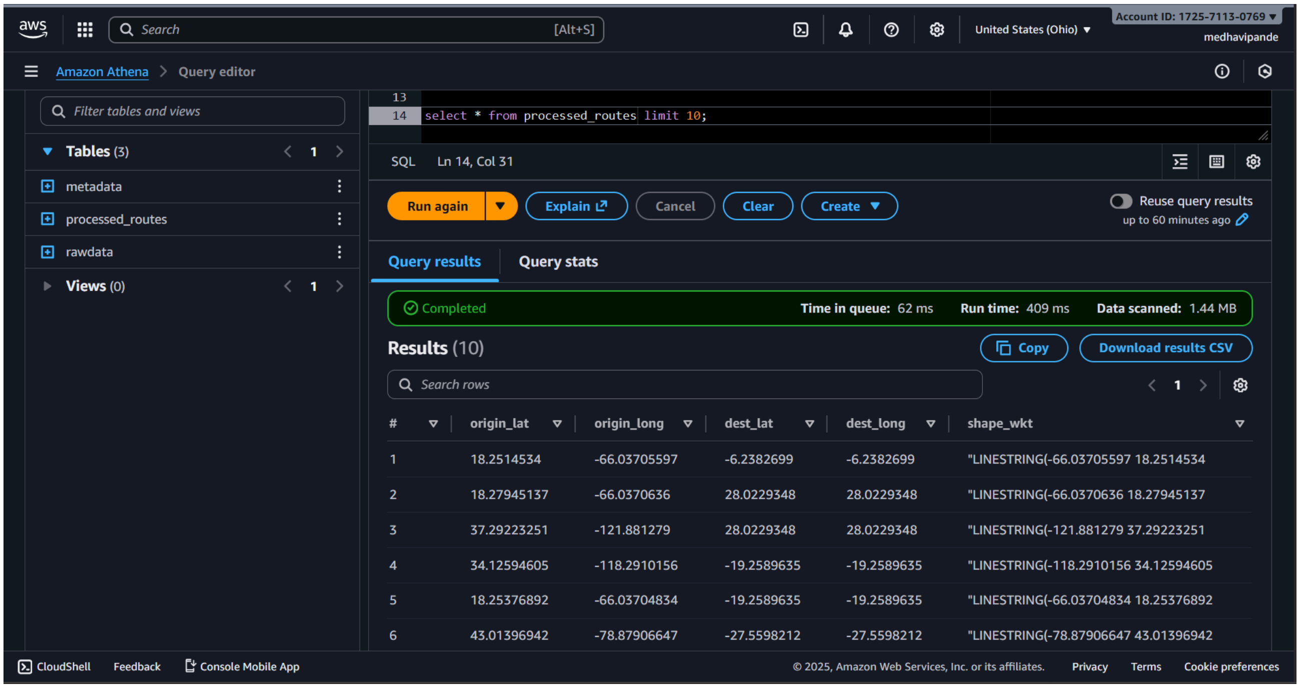


```
select * from rawdata limit 10;
```

✓ Completed   Time in queue: 54 ms   Run time: 598 ms   Data scanned: 828.00 KB

Results (10)

#	type	days_for_shipping_real	days_for_shipment_scheduled	benefit_per_order	sales_per_customer
1	TRANSFER	2	1	8.109999657	128.6900024
2	TRANSFER	2	1	44.59000015	122.8399963
3	TRANSFER	2	1	18.12999916	55.79000092
4	TRANSFER	2	1	30.76000023	118.2900009
5	TRANSFER	2	1	25.5699997	52.18999863
6	TRANSFER	2	1	33.77000046	107.8899994



```
select * from processed_routes limit 10;
```

Completed Time in queue: 62 ms Run time: 409 ms Data scanned: 1.44 MB

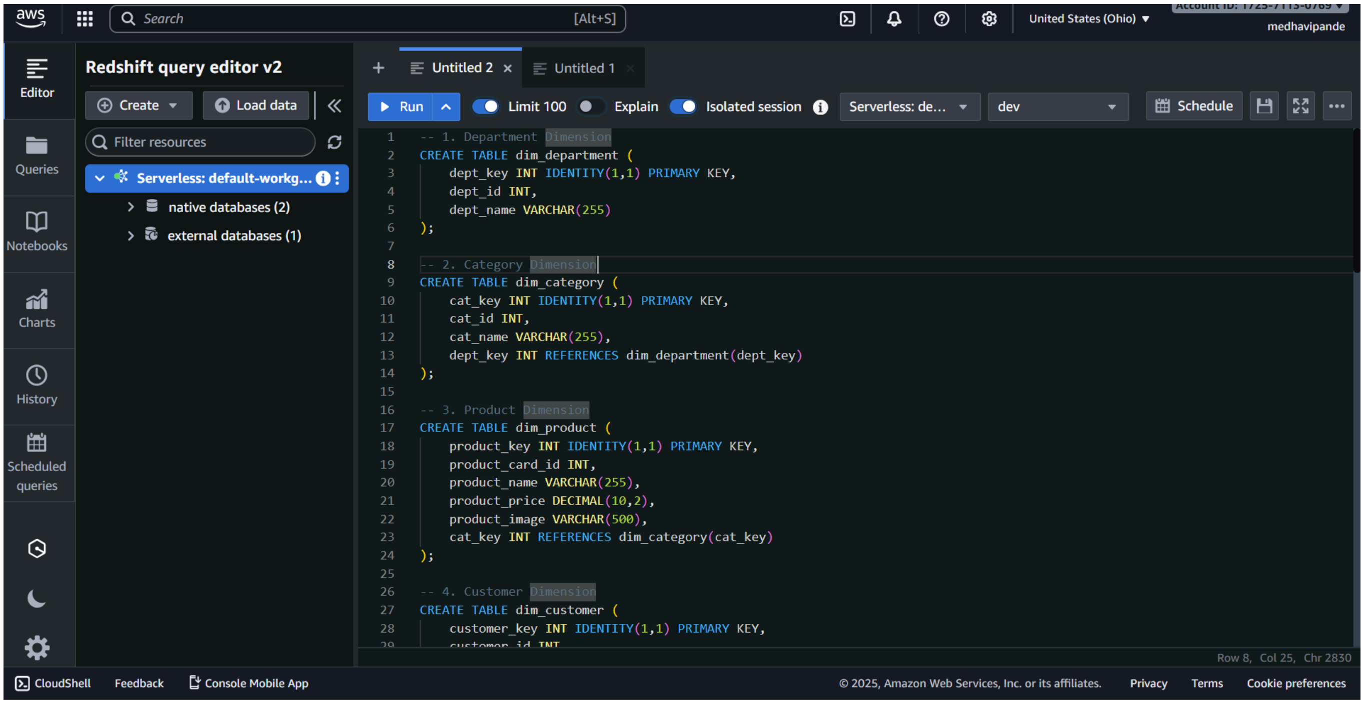
Results (10)

#	origin_lat	origin_long	dest_lat	dest_long	shape_wkt
1	18.2514534	-66.03705597	-6.2382699	-6.2382699	"LINESTRING(-66.03705597 18.2514534
2	18.27945137	-66.0370636	28.0229348	28.0229348	"LINESTRING(-66.0370636 18.27945137
3	37.29223251	-121.881279	28.0229348	28.0229348	"LINESTRING(-121.881279 37.29223251
4	34.12594605	-118.2910156	-19.2589635	-19.2589635	"LINESTRING(-118.2910156 34.12594605
5	18.25376892	-66.03704834	-19.2589635	-19.2589635	"LINESTRING(-66.03704834 18.25376892
6	43.01396942	-78.87906647	-27.5598212	-27.5598212	"LINESTRING(-78.87906647 43.01396942

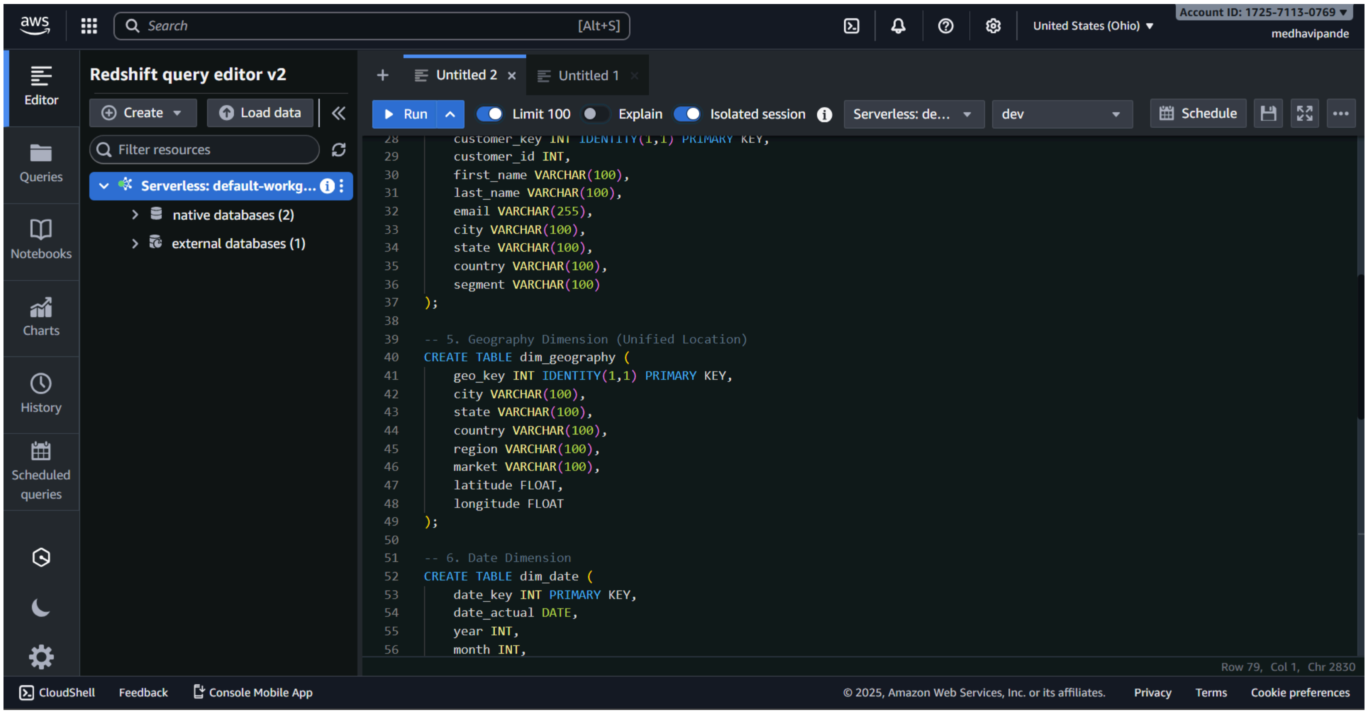
B. Transformation & Storage (ETL Process)

5. Redshift Schema Creation

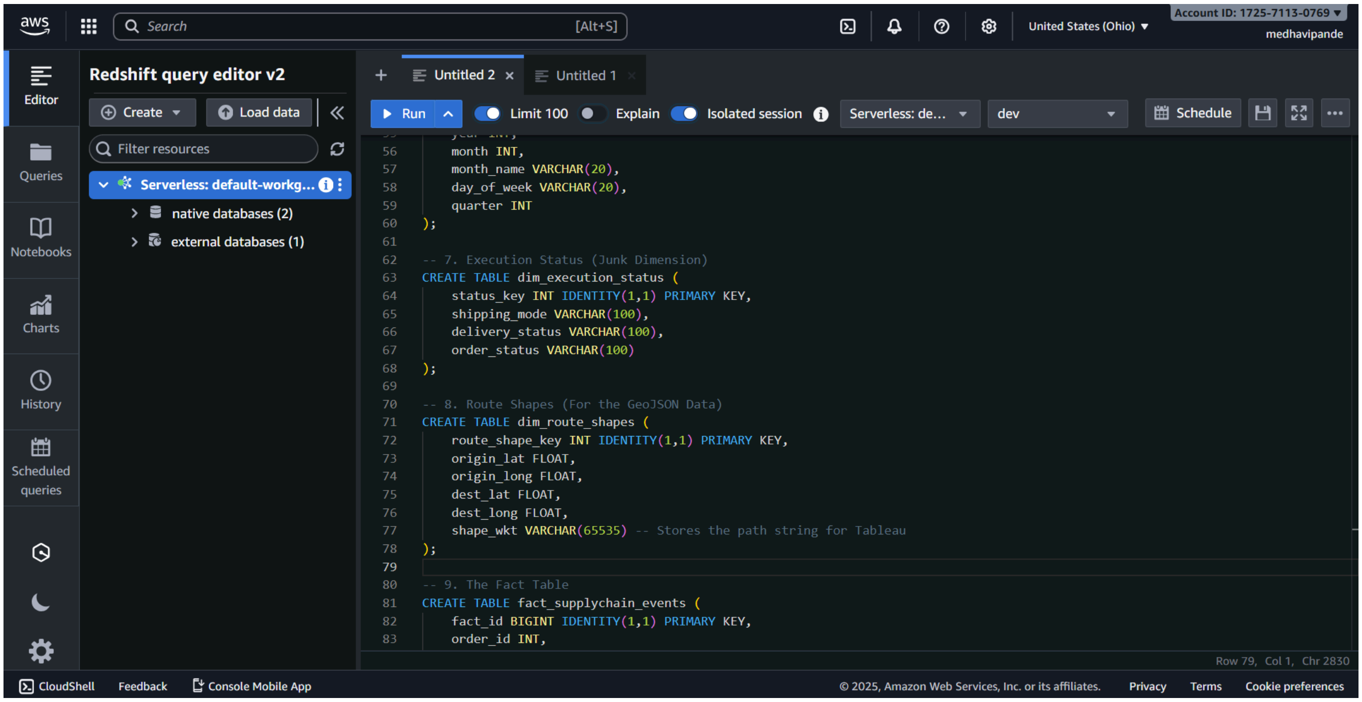
The destination warehouse tables were created in the Amazon Redshift Serverless workgroup. The data model utilizes a Snowflake Schema structure to reduce redundancy and optimize analytical performance. The specialized dim_route_shapes table was created to store the WKT geometry for geospatial analysis in Tableau later.



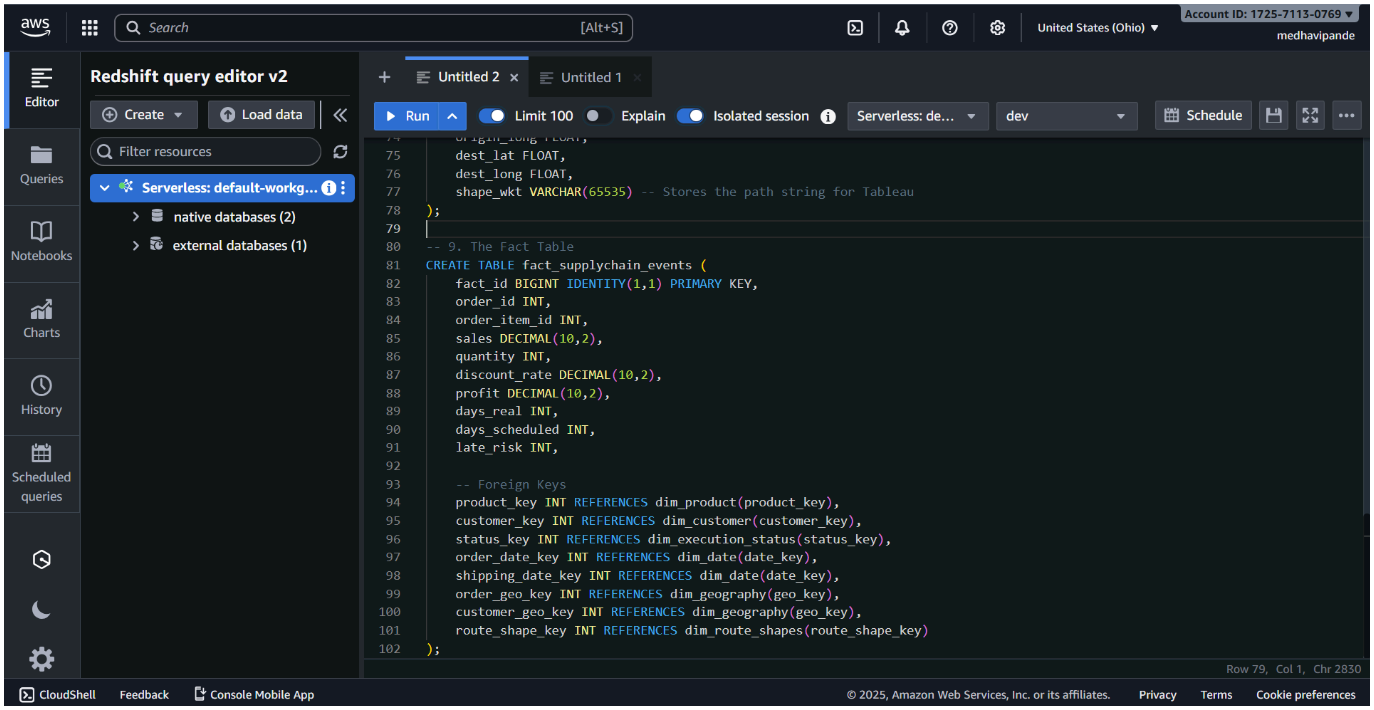
```sql
-- 1. Department Dimension
CREATE TABLE dim_department (
    dept_key INT IDENTITY(1,1) PRIMARY KEY,
    dept_id INT,
    dept_name VARCHAR(255)
);

-- 2. Category Dimension
CREATE TABLE dim_category (
    cat_key INT IDENTITY(1,1) PRIMARY KEY,
    cat_id INT,
    cat_name VARCHAR(255),
    dept_key INT REFERENCES dim_department(dept_key)
);

-- 3. Product Dimension
CREATE TABLE dim_product (
    product_key INT IDENTITY(1,1) PRIMARY KEY,
    product_card_id INT,
    product_name VARCHAR(255),
    product_price DECIMAL(10,2),
    product_image VARCHAR(500),
    cat_key INT REFERENCES dim_category(cat_key)
);

-- 4. Customer Dimension
CREATE TABLE dim_customer (
    customer_key INT IDENTITY(1,1) PRIMARY KEY,
```



```sql
    customer_id INT,
    first_name VARCHAR(100),
    last_name VARCHAR(100),
    email VARCHAR(255),
    city VARCHAR(100),
    state VARCHAR(100),
    country VARCHAR(100),
    segment VARCHAR(100)
);

-- 5. Geography Dimension (Unified Location)
CREATE TABLE dim_geography (
    geo_key INT IDENTITY(1,1) PRIMARY KEY,
    city VARCHAR(100),
    state VARCHAR(100),
    country VARCHAR(100),
    region VARCHAR(100),
    market VARCHAR(100),
    latitude FLOAT,
    longitude FLOAT
);

-- 6. Date Dimension
CREATE TABLE dim_date (
    date_key INT PRIMARY KEY,
    date_actual DATE,
    year INT,
    month INT,
```



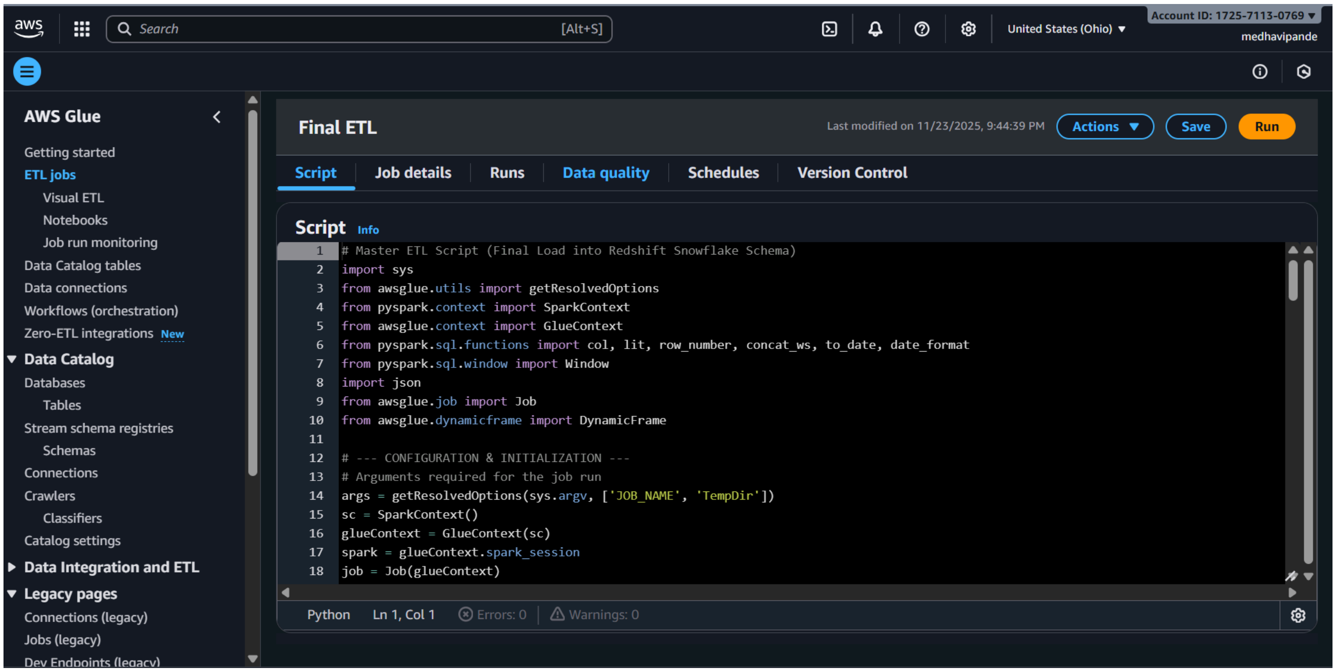
```
    year INT,
    month INT,
    month_name VARCHAR(20),
    day_of_week VARCHAR(20),
    quarter INT
);

-- 7. Execution Status (Junk Dimension)
CREATE TABLE dim_execution_status (
    status_key INT IDENTITY(1,1) PRIMARY KEY,
    shipping_mode VARCHAR(100),
    delivery_status VARCHAR(100),
    order_status VARCHAR(100)
);

-- 8. Route Shapes (For the GeoJSON Data)
CREATE TABLE dim_route_shapes (
    route_shape_key INT IDENTITY(1,1) PRIMARY KEY,
    origin_lat FLOAT,
    origin_long FLOAT,
    dest_lat FLOAT,
    dest_long FLOAT,
    shape_wkt VARCHAR(65535) -- Stores the path string for Tableau
);

-- 9. The Fact Table
CREATE TABLE fact_supplychain_events (
    fact_id BIGINT IDENTITY(1,1) PRIMARY KEY,
    order_id INT,
```



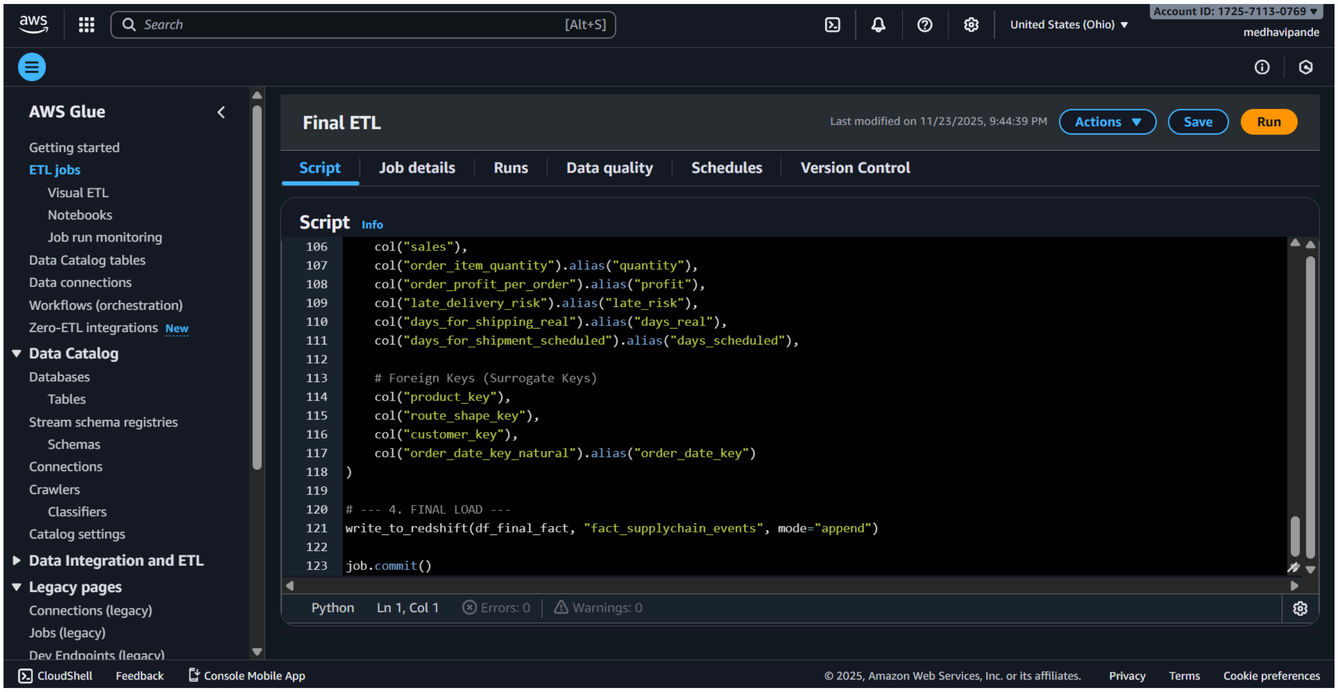
```
    origin_long FLOAT,
    dest_lat FLOAT,
    dest_long FLOAT,
    shape_wkt VARCHAR(65535) -- Stores the path string for Tableau
);

-- 9. The Fact Table
CREATE TABLE fact_supplychain_events (
    fact_id BIGINT IDENTITY(1,1) PRIMARY KEY,
    order_id INT,
    order_item_id INT,
    sales DECIMAL(10,2),
    quantity INT,
    discount_rate DECIMAL(10,2),
    profit DECIMAL(10,2),
    days_real INT,
    days_scheduled INT,
    late_risk INT,

    -- Foreign Keys
    product_key INT REFERENCES dim_product(product_key),
    customer_key INT REFERENCES dim_customer(customer_key),
    status_key INT REFERENCES dim_execution_status(status_key),
    order_date_key INT REFERENCES dim_date(date_key),
    shipping_date_key INT REFERENCES dim_date(date_key),
    order_geo_key INT REFERENCES dim_geography(geo_key),
    customer_geo_key INT REFERENCES dim_geography(geo_key),
    route_shape_key INT REFERENCES dim_route_shapes(route_shape_key)
);
```

6. Master ETL Job Execution (PySpark Script)

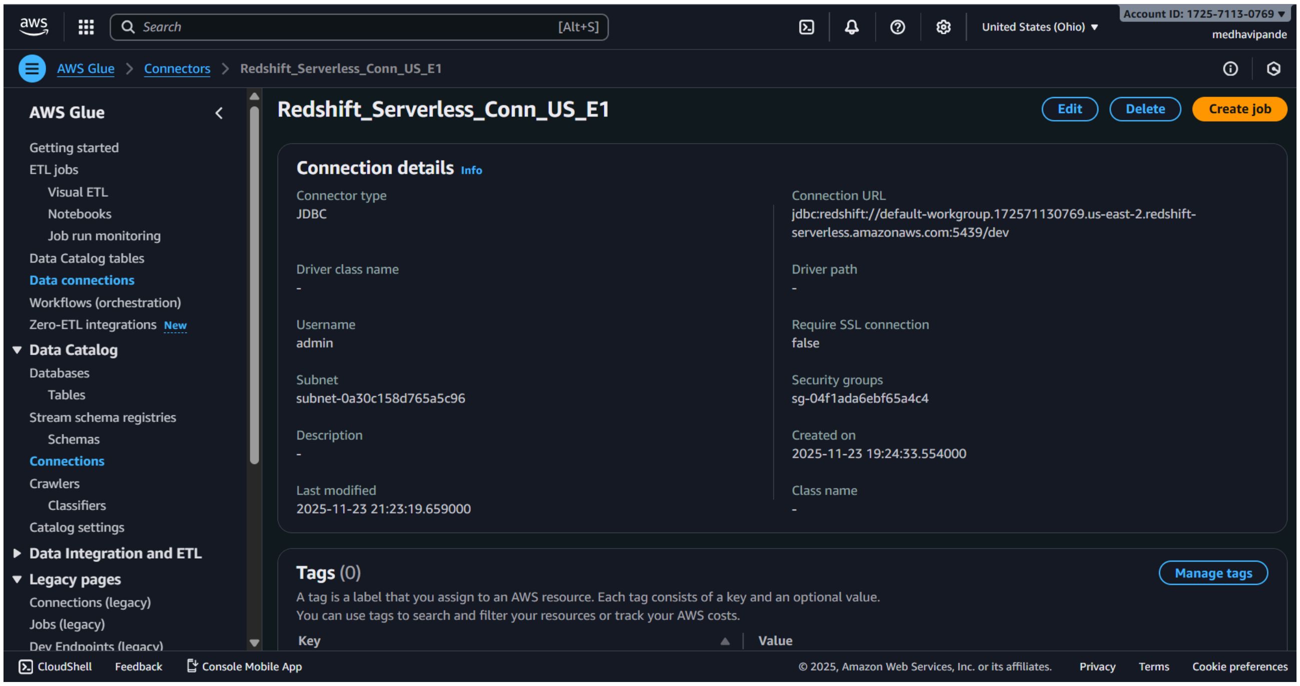
An AWS Glue ETL Job was created using a PySpark script (Master ETL Job). This job performs the core transformation: Extracts all three sources (rawdata, metadata, processed_routes) from the Data Catalog.Transforms the data by generating dimensional hierarchies and surrogate keys. Joins the main fact dataset (rawdata) with all dimension tables, including a complex join based on Lat/Long coordinates to link to the dim_route_shapes table (GeoJSON data). Loads the final, linked dimension and fact tables into the Redshift Serverless cluster.



```
# Master ETL Script (Final Load into Redshift Snowflake Schema)
import sys
from awsglue.utils import getResolvedOptions
from pyspark.context import SparkContext
from awsglue.context import GlueContext
from pyspark.sql.functions import col, lit, row_number, concat_ws, to_date, date_format
from pyspark.sql.window import Window
import json
from awsglue.job import Job
from awsglue.dynamicframe import DynamicFrame

# --- CONFIGURATION & INITIALIZATION ---
# Arguments required for the job run
args = getResolvedOptions(sys.argv, ['JOB_NAME', 'TempDir'])
sc = SparkContext()
glueContext = GlueContext(sc)
spark = glueContext.spark_session
job = Job(glueContext)
```



```
    col("sales"),
    col("order_item_quantity").alias("quantity"),
    col("order_profit_per_order").alias("profit"),
    col("late_delivery_risk").alias("late_risk"),
    col("days_for_shipping_real").alias("days_real"),
    col("days_for_shipment_scheduled").alias("days_scheduled"),

    # Foreign Keys (Surrogate Keys)
    col("product_key"),
    col("route_shape_key"),
    col("customer_key"),
    col("order_date_key_natural").alias("order_date_key")
)

# --- 4. FINAL LOAD ---
write_to_redshift(df_final_fact, "fact_supplychain_events", mode="append")

job.commit()
```



Redshift_Serverless_Conn_US_E1

Connection details

Connector type
JDBC

Connection URL
jdbc:redshift://default-workgroup.172571130769.us-east-2.redshift-serverless.amazonaws.com:5439/dev

Driver class name
-

Driver path
-

Username
admin

Require SSL connection
false

Subnet
subnet-0a30c158d765a5c96

Security groups
sg-04f1ada6ebf65a4c4

Description
-

Created on
2025-11-23 19:24:33.554000

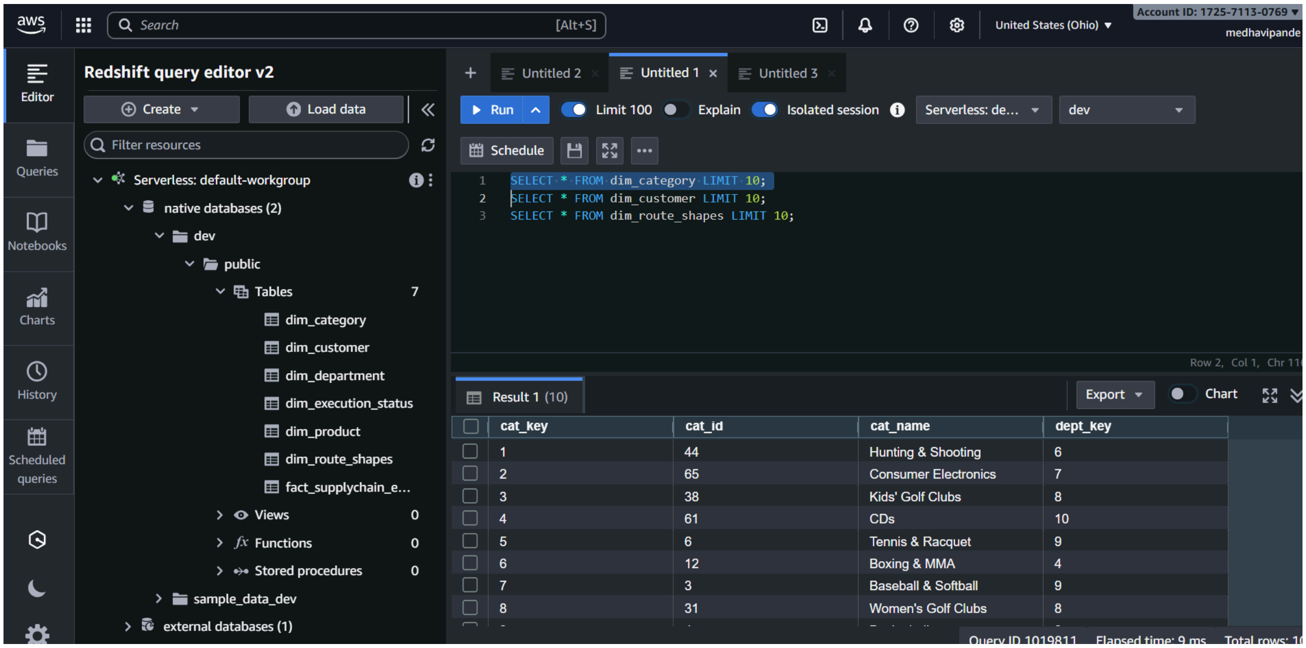
Last modified
2025-11-23 21:23:19.659000

Class name
-

Tags (0)

A tag is a label that you assign to an AWS resource. Each tag consists of a key and an optional value.
You can use tags to search and filter your resources or track your AWS costs.

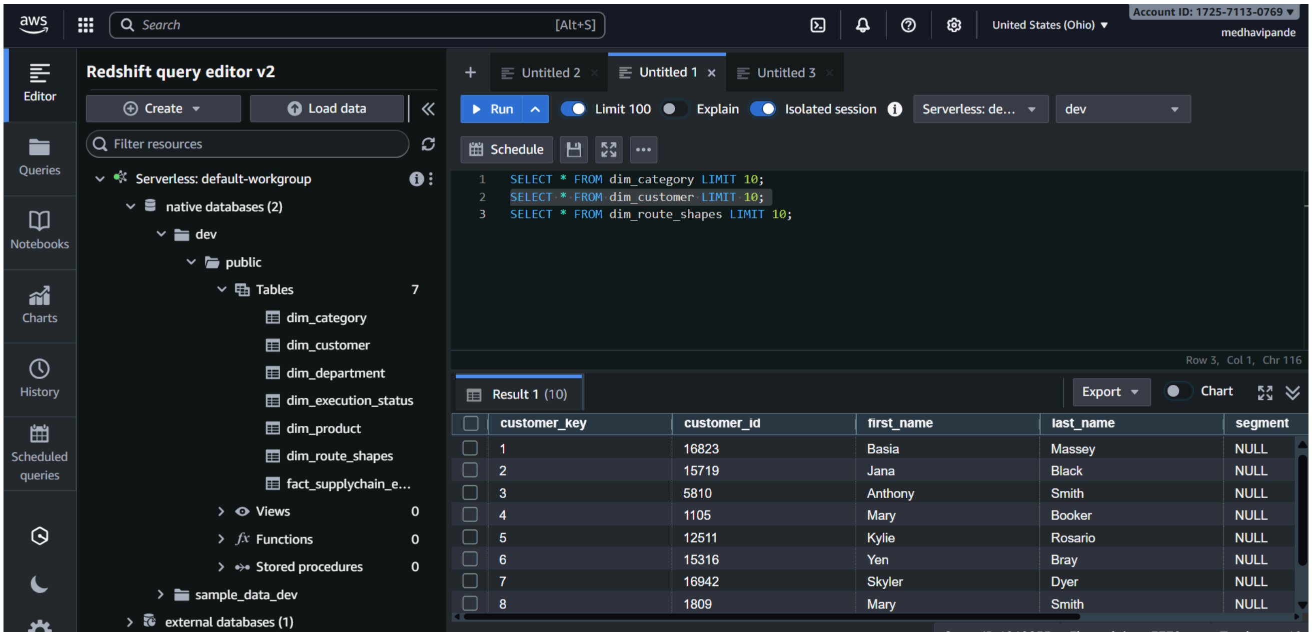
Data Warehouse:



```
SELECT * FROM dim_category LIMIT 10;
SELECT * FROM dim_customer LIMIT 10;
SELECT * FROM dim_route_shapes LIMIT 10;
```

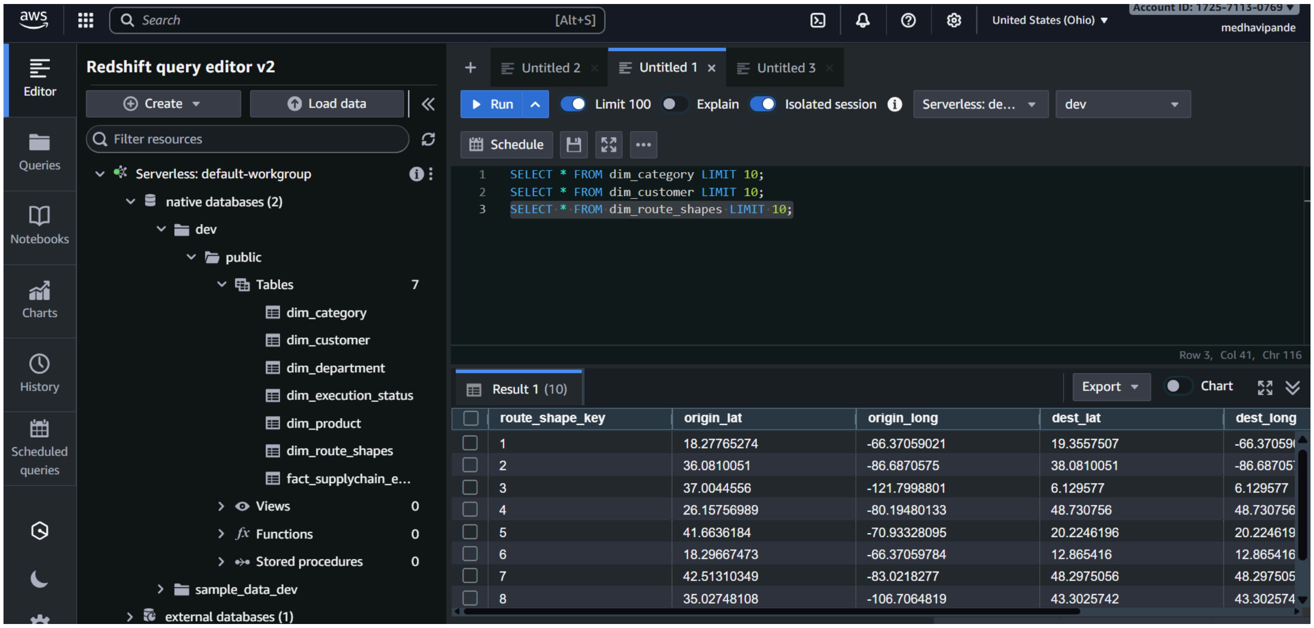
cat_key	cat_id	cat_name	dept_key
1	44	Hunting & Shooting	6
2	65	Consumer Electronics	7
3	38	Kids' Golf Clubs	8
4	61	CDs	10
5	6	Tennis & Racquet	9
6	12	Boxing & MMA	4
7	3	Baseball & Softball	9
8	31	Women's Golf Clubs	8

Query ID 1019811 Elapsed time: 9 ms Total rows: 1



```sql
SELECT * FROM dim_category LIMIT 10;
SELECT * FROM dim_customer LIMIT 10;
SELECT * FROM dim_route_shapes LIMIT 10;
```

customer_key	customer_id	first_name	last_name	segment
1	16823	Basia	Massey	NULL
2	15719	Jana	Black	NULL
3	5810	Anthony	Smith	NULL
4	1105	Mary	Booker	NULL
5	12511	Kylie	Rosario	NULL
6	15316	Yen	Bray	NULL
7	16942	Skyler	Dyer	NULL
8	1809	Mary	Smith	NULL

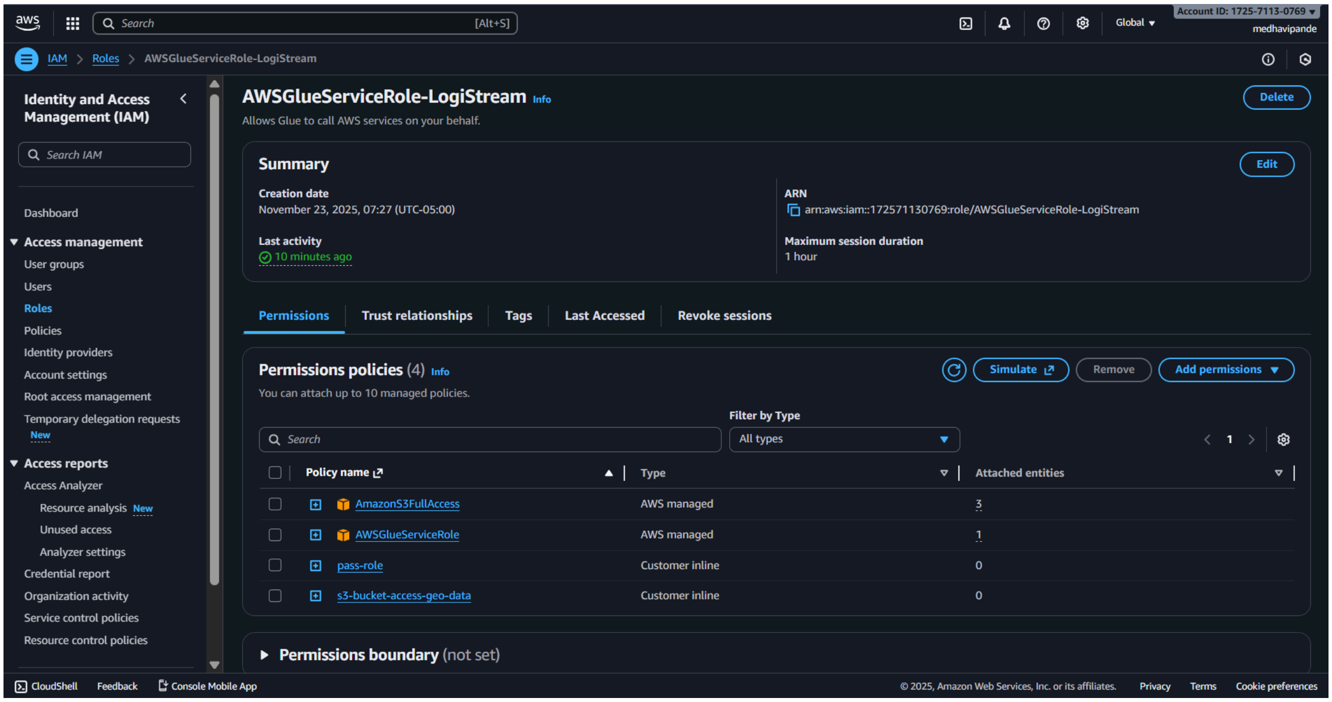


```sql
SELECT * FROM dim_category LIMIT 10;
SELECT * FROM dim_customer LIMIT 10;
SELECT * FROM dim_route_shapes LIMIT 10;
```

route_shape_key	origin_lat	origin_long	dest_lat	dest_long
1	18.27765274	-66.37059021	19.3557507	-66.37059
2	36.0810051	-86.6870575	38.0810051	-86.68705
3	37.0044556	-121.7998801	6.129577	6.129577
4	26.15756989	-80.19480133	48.730756	48.730756
5	41.6636184	-70.93328095	20.2246196	20.224619
6	18.29667473	-66.37059784	12.865416	12.865416
7	42.51310349	-83.0218277	48.2975056	48.297505
8	35.02748108	-106.7064819	43.3025742	43.302574

7. IAM Roles and Permissions

Dedicated IAM Roles were set up for the Lambda function, Glue Job, and Redshift cluster to grant necessary cross-service permissions.



AWSGlueServiceRole-LogiStream Info

Allows Glue to call AWS services on your behalf.

Summary

Creation date November 23, 2025, 07:27 (UTC-05:00)	**ARN** arn:aws:iam::172571130769:role/AWSGlueServiceRole-LogiStream
Last activity 10 minutes ago	**Maximum session duration** 1 hour

Permissions policies (4) Info

You can attach up to 10 managed policies.

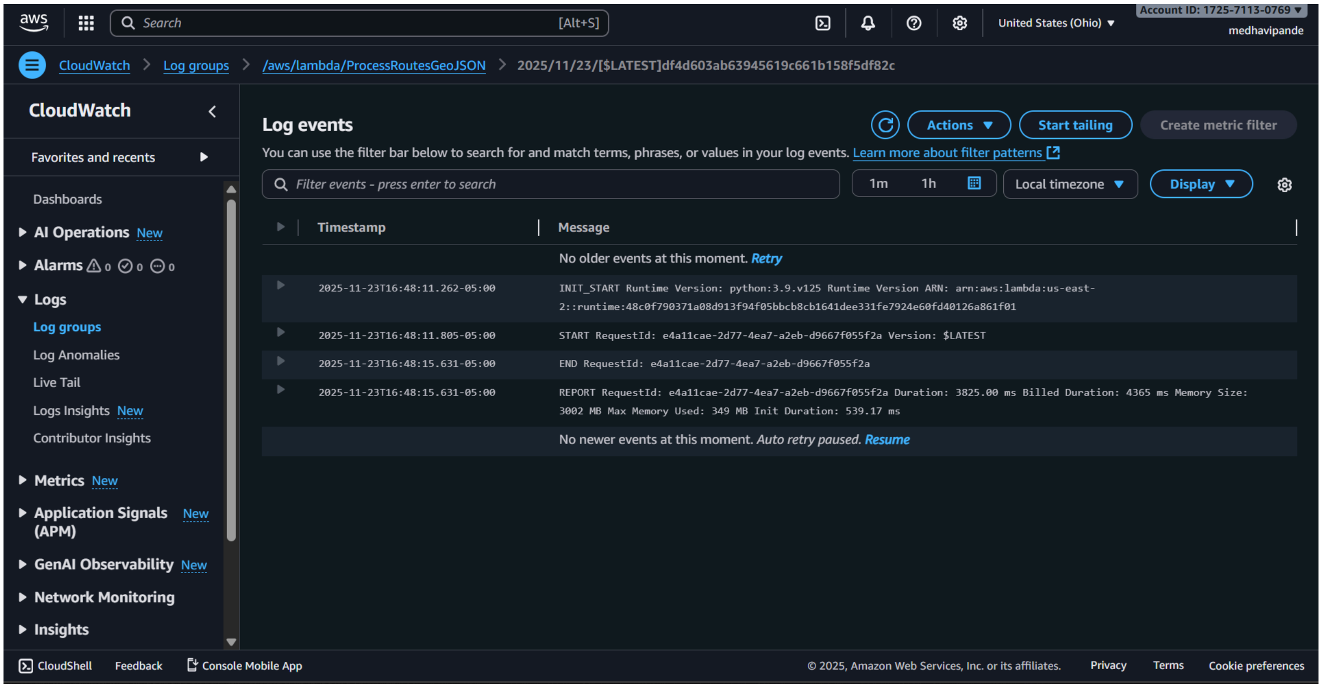
Filter by Type: All types

□	Policy name	Type	Attached entities
□	AmazonS3FullAccess	AWS managed	3
□	AWSGlueServiceRole	AWS managed	1
□	pass-role	Customer inline	0
□	s3-bucket-access-geo-data	Customer inline	0

▸ **Permissions boundary** (not set)

8. Monitoring and Logging

CloudWatch Logs were connected and used throughout the process to monitor job status, execution time, and troubleshoot connectivity errors.



Log events

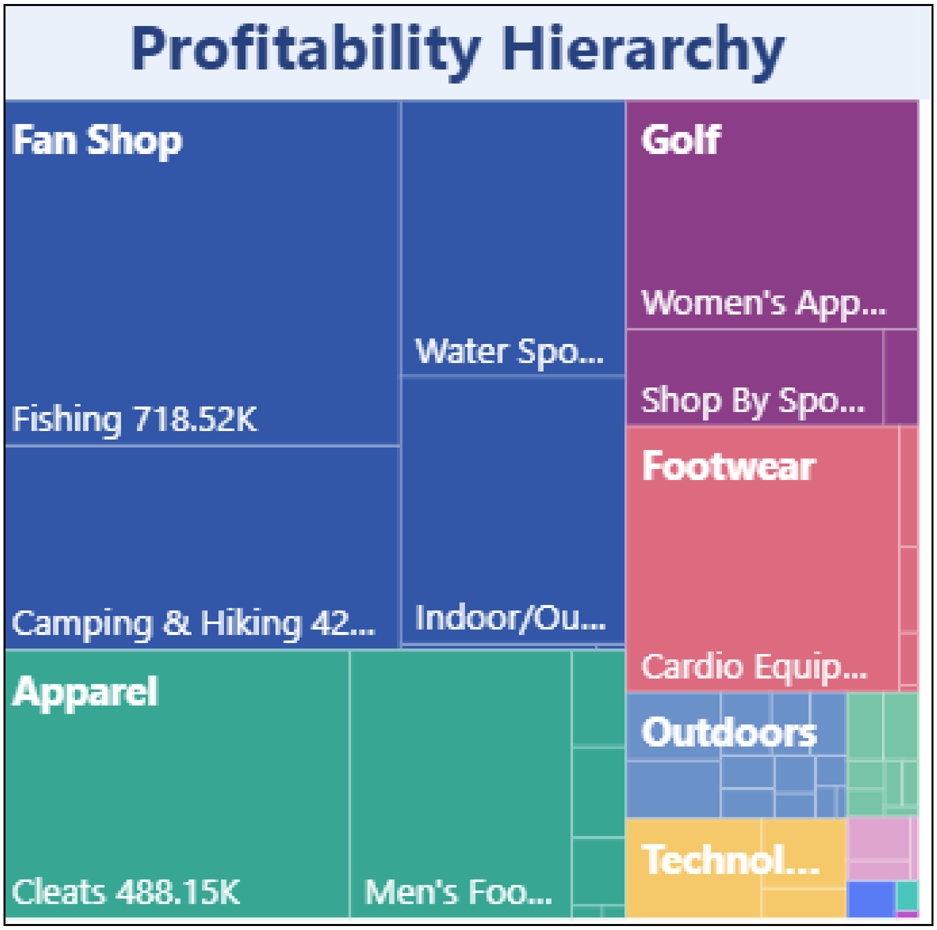
You can use the filter bar below to search for and match terms, phrases, or values in your log events. Learn more about filter patterns

```
Timestamp                        Message
                                 No older events at this moment. Retry
2025-11-23T16:48:11.262-05:00    INIT_START Runtime Version: python:3.9.v125 Runtime Version ARN: arn:aws:lambda:us-east-
                                 2::runtime:48c0f790371a08d913f94f05bbcb8cb1641dee331fe7924e60fd40126a861f01
2025-11-23T16:48:11.805-05:00    START RequestId: e4a11cae-2d77-4ea7-a2eb-d9667f055f2a Version: $LATEST
2025-11-23T16:48:15.631-05:00    END RequestId: e4a11cae-2d77-4ea7-a2eb-d9667f055f2a
2025-11-23T16:48:15.631-05:00    REPORT RequestId: e4a11cae-2d77-4ea7-a2eb-d9667f055f2a Duration: 3825.00 ms Billed Duration: 4365 ms Memory Size:
                                 3002 MB Max Memory Used: 349 MB Init Duration: 539.17 ms
                                 No newer events at this moment. Auto retry paused. Resume
```

Business Intelligence

1. **Profitability Hierarchy**

This Treemap visualizes the contribution of Total Profit across the product dimensional hierarchy.



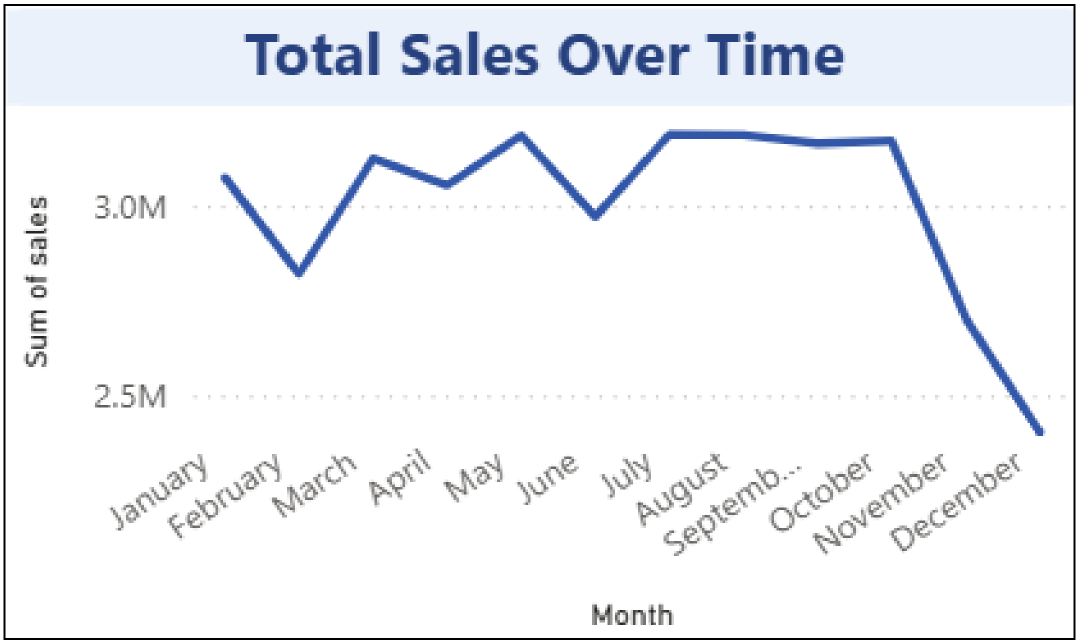
Actionable Insight:

This visual proves the value of the Snowflake Schema design, as it requires joining the fact table to multiple dimension tables (dim_department and dim_category).

Product/Marketing Strategy: Executives can immediately identify the most profitable product segments (the largest squares) and use this insight to prioritize marketing spend and production capacity on high-value items.

2. **Total Sales Trend Over Time:**

A time-series Line Chart tracking [Total Sales] over time (Month/Year).



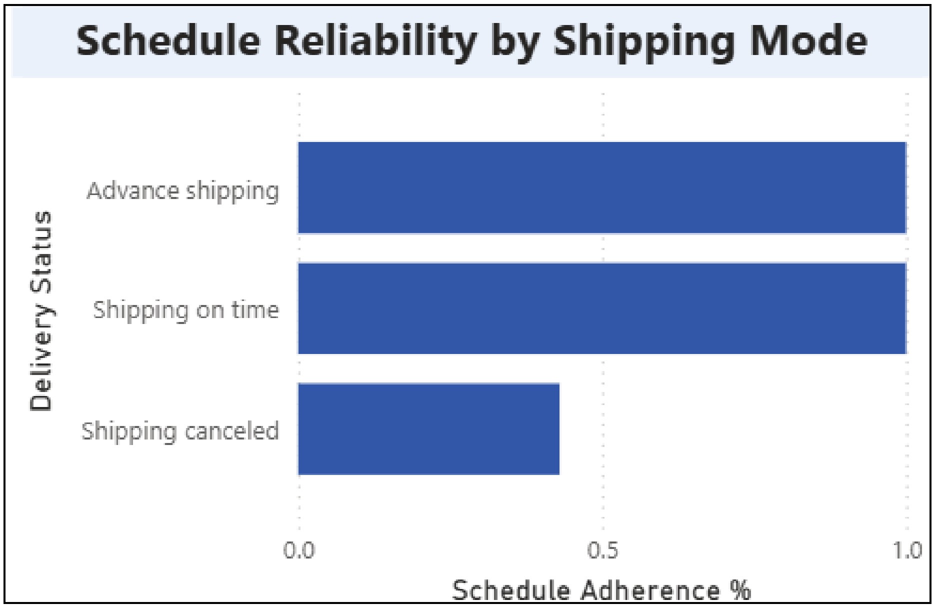
Actionable Insight:

It validates the integrity of the dim_date dimension and the accuracy of the large volume batch-loaded into the fact table.

Demand Forecasting: Managers can quickly spot seasonal peaks and troughs in demand. This informs purchasing decisions and staffing levels in warehouses, ensuring optimal inventory levels to meet future customer orders.

3. **Schedule Reliability by Shipping Mode**

A Bar Chart auditing the [Schedule Adherence %] across different shipping_mode values (carriers).



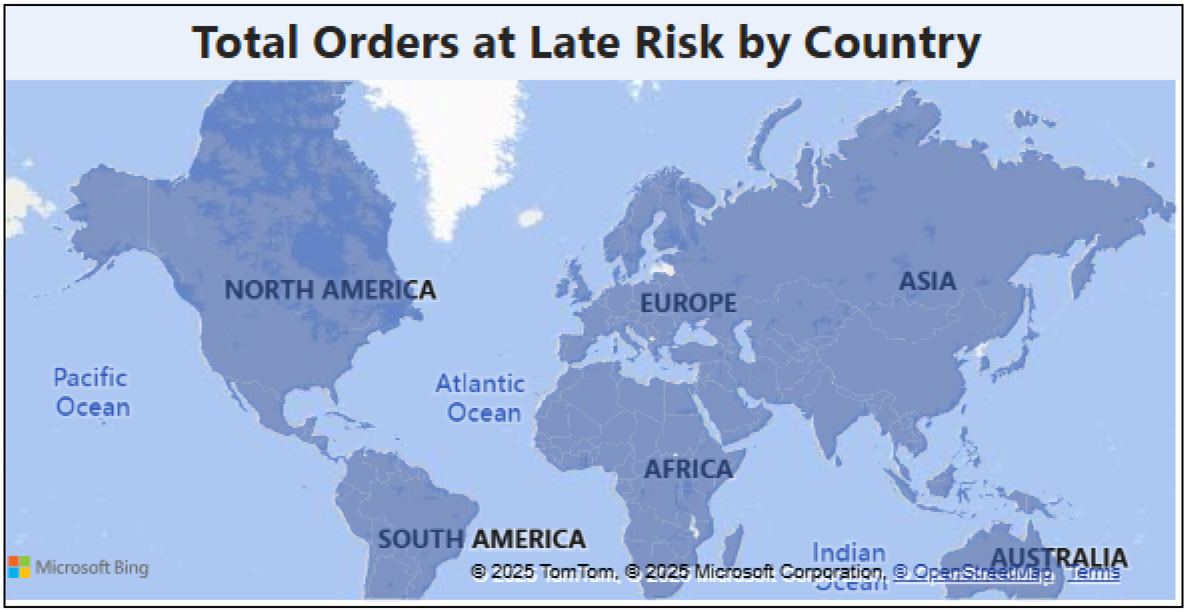
Actionable Insight:

This visual requires the pre-calculated Schedule Adherence measure, which compares two original raw columns.

Carrier Optimization: This directly shows which carriers or shipping methods are reliable (high adherence) and which are causing delays. Logistics managers can use this data to negotiate better rates or switch underperforming carriers.

4. **Late Order Volume by Country**

A Map visual showing the geographical distribution of [Total Orders at Late Risk] using bubble size by country.



Actionable Insight:

It utilizes the final calculated risk flag (late_risk = 1) and the geographic dimension, linking a key operational metric to a physical location for action.

Proactive Risk Mitigation: Large bubbles immediately signal regional logistics hotspots where intervention is required. This allows management to preemptively contact customers or deploy regional solutions, turning a retrospective problem into a proactive service opportunity.

Conclusion

The LogiStream project successfully built a unified cloud data pipeline on AWS, transitioning DataCo from purely retrospective analytics to a proactive BI platform. By leveraging AWS Glue and PySpark to transform raw data into a clean, analytical Snowflake Schema in Amazon Redshift, the pipeline established a single source of truth. The resulting Power BI dashboard operationalizes the data, delivering actionable insights such as proactive late-delivery alerts and data-driven guidance for route optimization and inventory placement, directly fulfilling the original business problem definition.