Week 1 Internship Report: OUBT Bootcamp – Day 1 to Day 5 Learnings:

NAME: LIKHITH SASANK UPPALAPATI VENKATA

MAIL : likhith2kuv@gmail.com

Introduction

This document summarizes my learnings and activities during Week 1 of the OUBT Bootcamp Internship (Day 1 to Day 5). Over the course of the week, I focused on building foundational skills in AWS services, Python for data engineering, SQL, and GitHub version control. I also gained hands-on experience with **Git and GitHub**, including repository management, branching, and committing changes, which helped me track and manage my code effectively. Each day involved hands-on exercises, practical tasks, and mini-projects aimed at developing core competencies in cloud computing, data processing, and automation workflows. The following sections detail the tasks completed, concepts learned, and technical skills acquired throughout the week.

Project Repository Details:

All Week 1 code, scripts, and documentation have been maintained in my Git repository: https://github.com/likhith5697/Week1_OUBT_Learnings. The repository includes Python scripts, SQL queries, and AWS configuration files for tasks completed from Day 1 to Day 5.

NAME: LIKHITH SASANK UPPALAPATI VENKATA

DAY: 1

AWS IAM Setup:

Create IAM users, groups, and roles.

Assign permissions and policies.

Test login using IAM sign-in link.

Billing Alert and CloudWatch Alarm:

Enable billing alerts in the AWS Billing Console.

Create a CloudWatch alarm to track monthly charges.

Set up an SNS topic to send cost alerts via email.

Amazon S3 Bucket:

Create a new S3 bucket.

Upload files of different types (PDF, image, CSV).

Enable bucket versioning and upload the same file again to check multiple versions.

GitHub Setup:

Create a new repository for OUBT bootcamp tasks.

Clone the repository locally.

Add the Day 1 Word document and screenshots.

Commit and push the files to GitHub.

1. Introduction

This week focused on AWS fundamentals, IAM, S3, GitHub basics, and Python for Data Engineering. The goal was to understand access management, cloud storage, and data handling concepts.

2. IAM Concepts Summary

IAM Overview:
IAM – It is a security's service that will control who can access our AWS account

IAM – USERS:

Users means one individual person that needs an access

Example : Let's say we are a team of developers, admin, testers. Each person needs a particular access so we create users as Likhith.Dev, Sasank.Admin, Venkat.Tester and give them roles and permissions as shown below.

Likhith.Dev -> **Developer** -> S3, Glue, RDS, Lambda

Sasank.Admin -> **Admin** -> Full Access

Venkat.Tester -> **Tester** -> Read Only

Here we attach policies to Users.

IAM – GROUPS:

Groups are the collection of Users who share same permissions.

Here we do assign policies to groups not for users separately as shown below,

Developers -> Access to S3, Glue, RDS, Lambda -> Members (Likhith, Sasank)

Admin -> Full Access -> Members (Venkat)

Tester -> Read Only Access (Durga)

IAM ROLES:

Role is a temporary identity with specific permissions.

AWS service assumes a role
Users assume a role
Group assumes a role

Example : Suppose I have a microservice or may be a spring boot app deployed in EC2 that basically needs to read and write to RDS, So instead of hardcoding credentials, I can create a role and assign to our EC2 instance for access to RDS, So it gets permission automatically.

IAM Policy:

Policy defines permissions.

It is a json object that says what is allowed and what is not allowed.

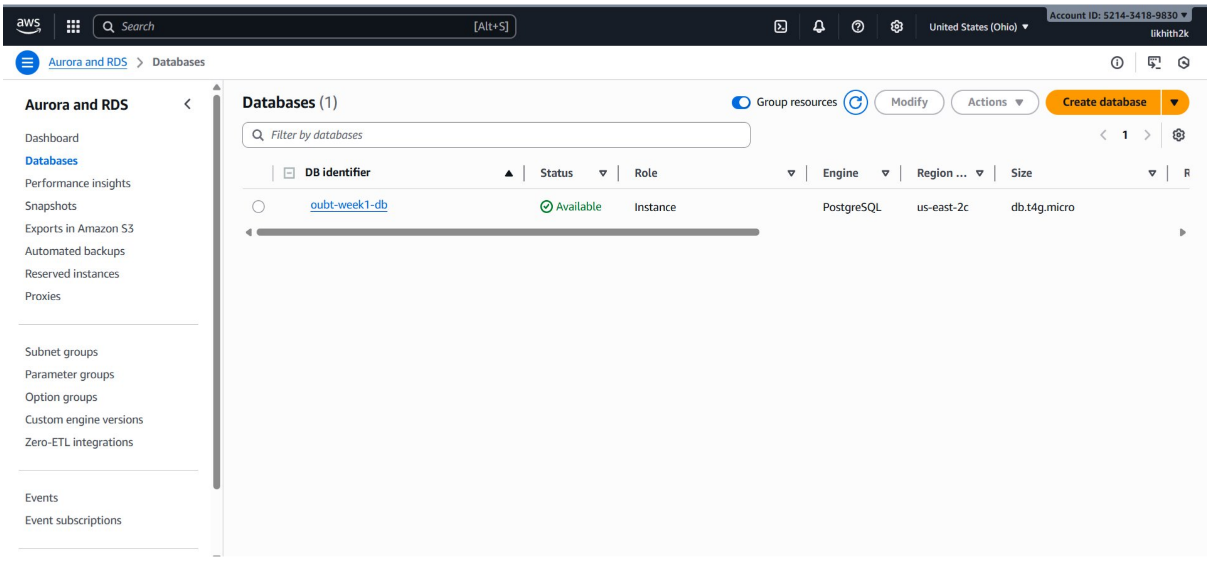
Billing Alerts:

This helps to avoid surprise charges .

We can use cloudwatch alerts / Billing alarms to notify us when usage exceeds a limit.

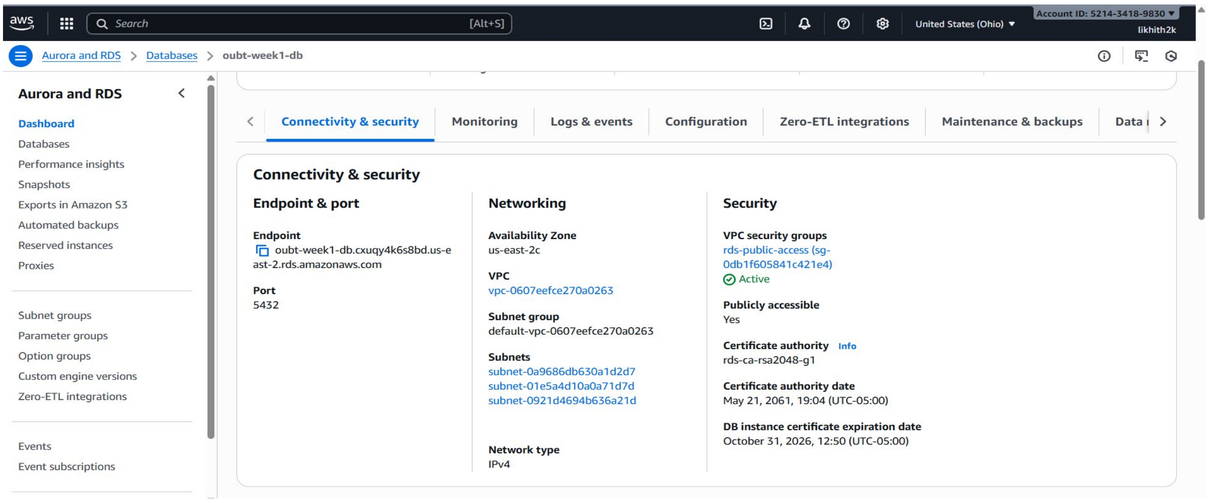
Here Am showing all my hand-ON PRACTICE,

Task 1: Created IAM User



Databases (1)

DB identifier	Status	Role	Engine	Region ...	Size	R
○ oubt-week1-db	✓ Available	Instance	PostgreSQL	us-east-2c	db.t4g.micro	



Connectivity & security

Endpoint & port

Endpoint
oubt-week1-db.cxuqy4k6s8bd.us-east-2.rds.amazonaws.com

Port
5432

Networking

Availability Zone
us-east-2c

VPC
vpc-0607eefce270a0263

Subnet group
default-vpc-0607eefce270a0263

Subnets
subnet-0a9686db630a1d2d7
subnet-01e5a4d10a0a71d7d
subnet-0921d4694b636a21d

Network type
IPv4

Security

VPC security groups
rds-public-access (sg-0db1f605841c421e4)
Active

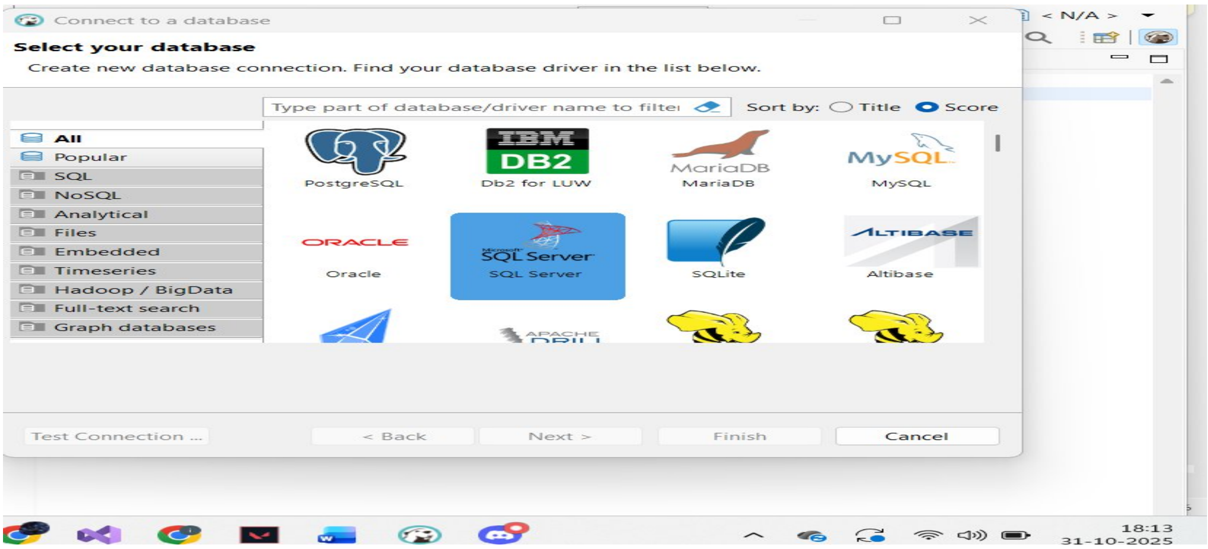
Publicly accessible
Yes

Certificate authority Info
rds-ca-rsa2048-g1

Certificate authority date
May 21, 2061, 19:04 (UTC-05:00)

DB instance certificate expiration date
October 31, 2026, 12:50 (UTC-05:00)

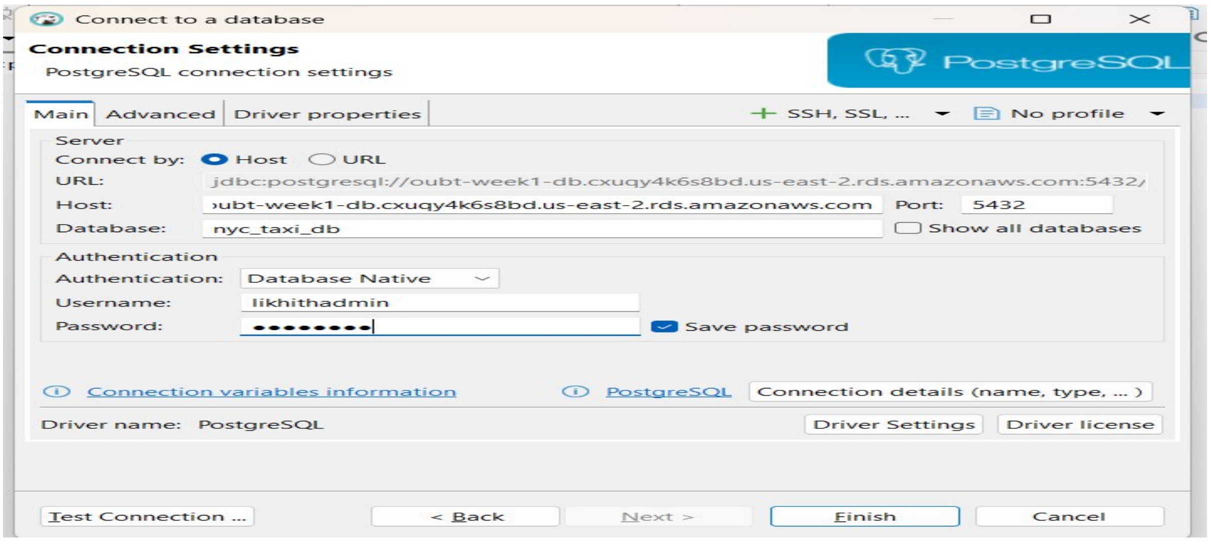
Successfully created user,



Select your database

Create new database connection. Find your database driver in the list below.

Task 2: Created S3 Bucket:



Connect to a database

Connection Settings

PostgreSQL connection settings

Server

Connect by: ● Host ○ URL

URL: jdbc:postgresql://oubt-week1-db.cxuqy4k6s8bd.us-east-2.rds.amazonaws.com:5432/

Host: oubt-week1-db.cxuqy4k6s8bd.us-east-2.rds.amazonaws.com Port: 5432

Database: nyc_taxi_db ☐ Show all databases

Authentication

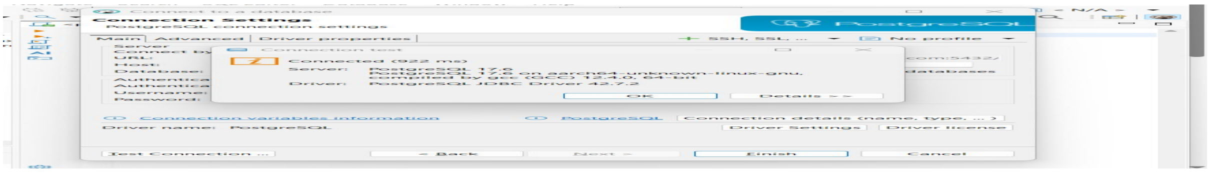
Authentication: Database Native

Username: likhithadmin

Password: ●●●●●●●● ☑ Save password

Connection variables information
PostgreSQL
Connection details (name, type, ...)

Driver name: PostgreSQL



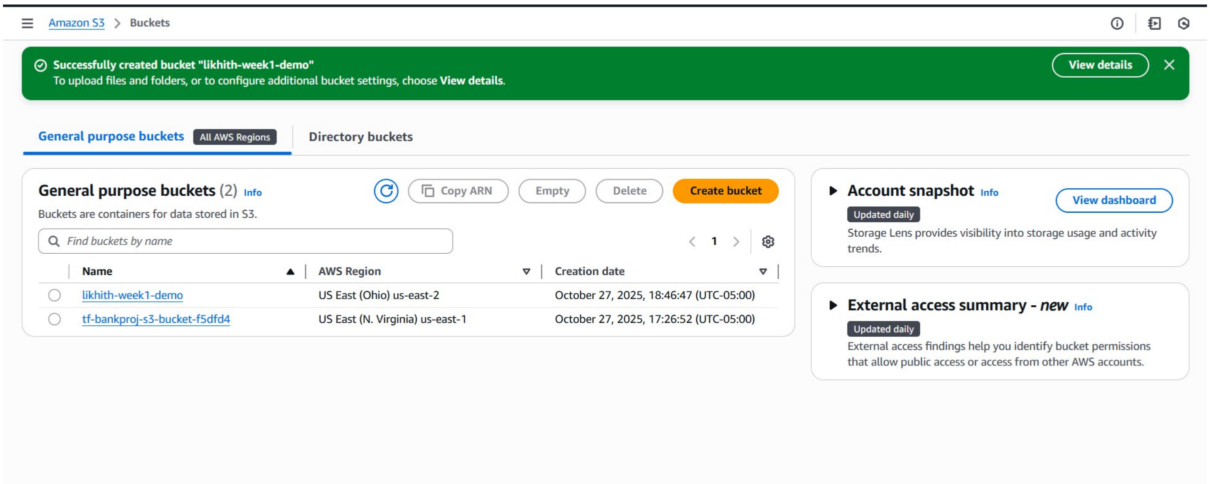
Connection test

Connected (922 ms)

Server: PostgreSQL 17.6
PostgreSQL 17.6 on aarch64-unknown-linux-gnu, compiled by gcc (GCC) 12.4.0, 64-bit

Driver: PostgreSQL JDBC Driver 42.7.2

Successfully created S3-BUCKET,



Successfully created bucket "likhith-week1-demo"
To upload files and folders, or to configure additional bucket settings, choose **View details**.

General purpose buckets (2) Info

Buckets are containers for data stored in S3.

Name	AWS Region	Creation date
likhith-week1-demo	US East (Ohio) us-east-2	October 27, 2025, 18:46:47 (UTC-05:00)
tf-bankproj-s3-bucket-f5dfd4	US East (N. Virginia) us-east-1	October 27, 2025, 17:26:52 (UTC-05:00)

Account snapshot Info

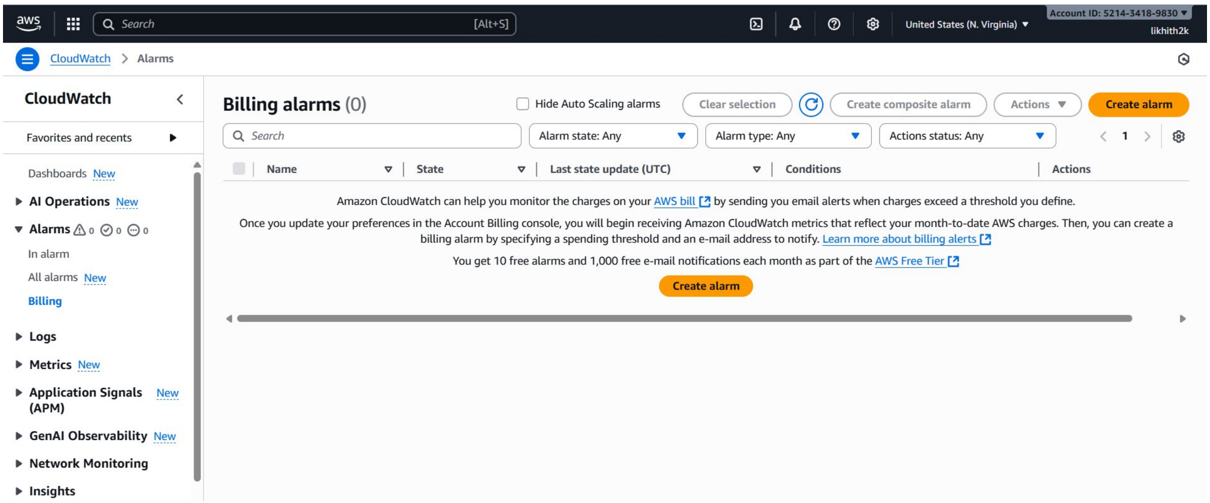
Storage Lens provides visibility into storage usage and activity trends.

External access summary - *new* Info

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

Task 3: Billing Alert

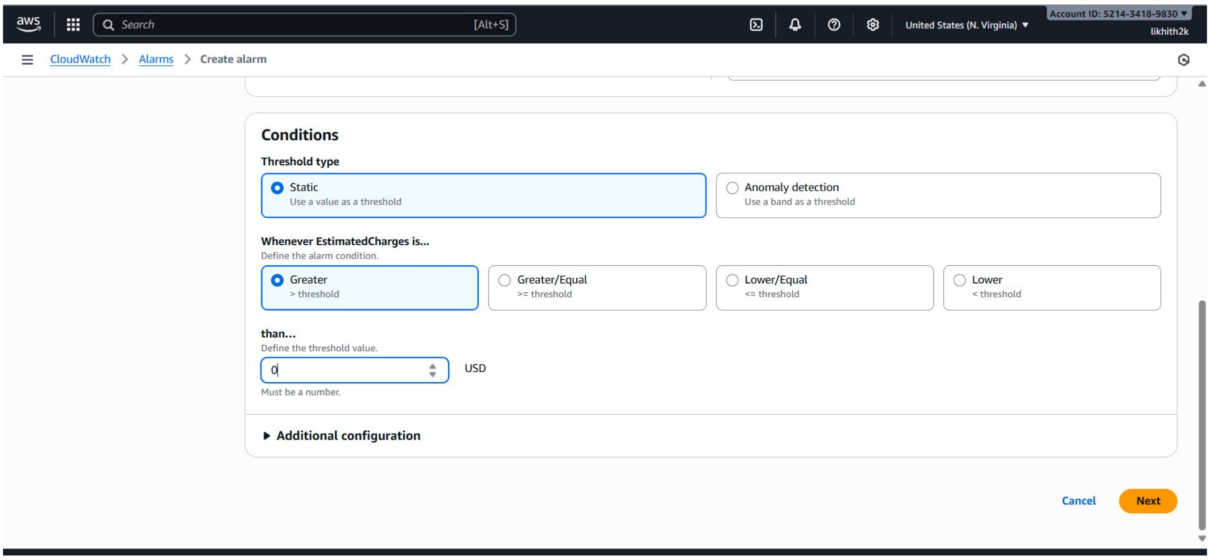
Set up CloudWatch Billing Alarm for $1 threshold with email alert.



Amazon CloudWatch can help you monitor the charges on your AWS bill by sending you email alerts when charges exceed a threshold you define.

Once you update your preferences in the Account Billing console, you will begin receiving Amazon CloudWatch metrics that reflect your month-to-date AWS charges. Then, you can create a billing alarm by specifying a spending threshold and an e-mail address to notify. Learn more about billing alerts

You get 10 free alarms and 1,000 free e-mail notifications each month as part of the AWS Free Tier


aws
Search
[Alt+S]
United States (N. Virginia) ▼
Account ID: 5214-3418-9830 ▼
likhith2k
CloudWatch > Alarms > Create alarm

Conditions

Threshold type

● Static
Use a value as a threshold

○ Anomaly detection
Use a band as a threshold

Whenever EstimatedCharges is...
Define the alarm condition.

● Greater
> threshold

○ Greater/Equal
>= threshold

○ Lower/Equal
<= threshold

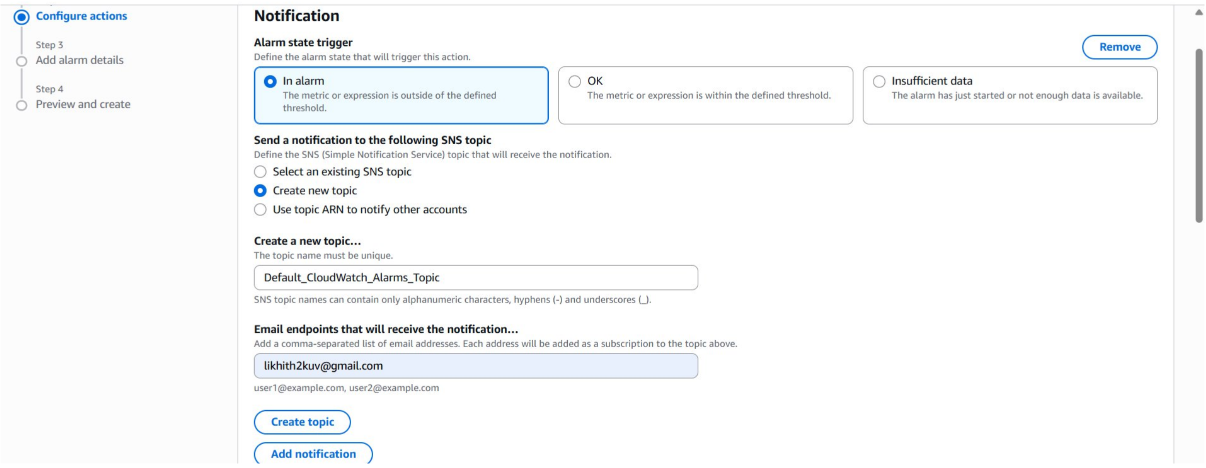
○ Lower
< threshold

than...
Define the threshold value.

0 USD

Must be a number.

Additional configuration
Cancel
Next



● Configure actions

Step 3
○ Add alarm details

Step 4
○ Preview and create

Notification

Alarm state trigger
Define the alarm state that will trigger this action.

Remove

● In alarm
The metric or expression is outside of the defined threshold.

○ OK
The metric or expression is within the defined threshold.

○ Insufficient data
The alarm has just started or not enough data is available.

Send a notification to the following SNS topic
Define the SNS (Simple Notification Service) topic that will receive the notification.

○ Select an existing SNS topic
● Create new topic
○ Use topic ARN to notify other accounts

Create a new topic...
The topic name must be unique.

Default_CloudWatch_Alarms_Topic

SNS topic names can contain only alphanumeric characters, hyphens (-) and underscores (_).

Email endpoints that will receive the notification...
Add a comma-separated list of email addresses. Each address will be added as a subscription to the topic above.

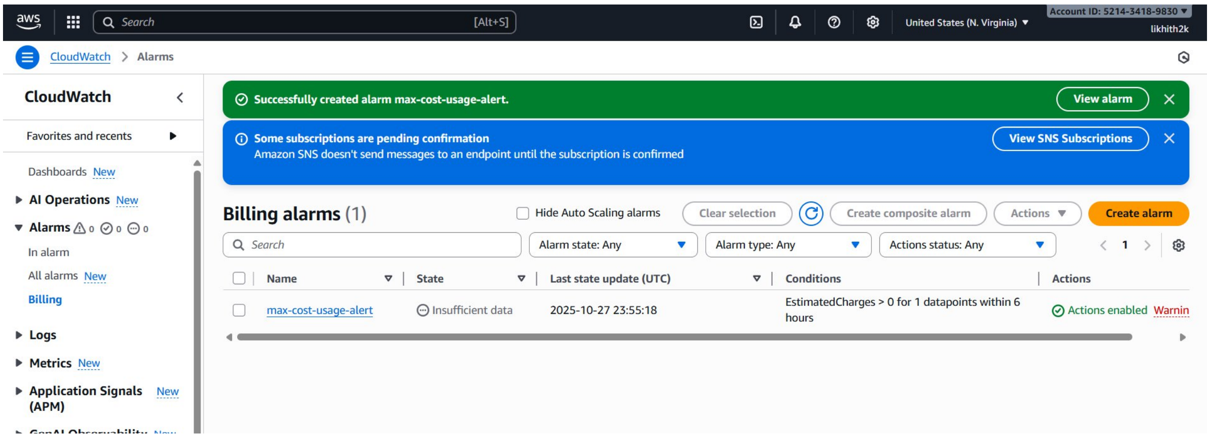
likhith2kuv@gmail.com

user1@example.com, user2@example.com

Create topic

Add notification

Finally created an alarm,



CloudWatch

Favorites and recents ▶

Dashboards New
▶ AI Operations New
▼ Alarms ⚠ 0 ✓ 0 ⊖ 0
In alarm
All alarms New
Billing
▶ Logs
▶ Metrics New
▶ Application Signals (APM) New

✓ Successfully created alarm max-cost-usage-alert. View alarm ✕

ⓘ Some subscriptions are pending confirmation
Amazon SNS doesn't send messages to an endpoint until the subscription is confirmed
View SNS Subscriptions ✕

Billing alarms (1)

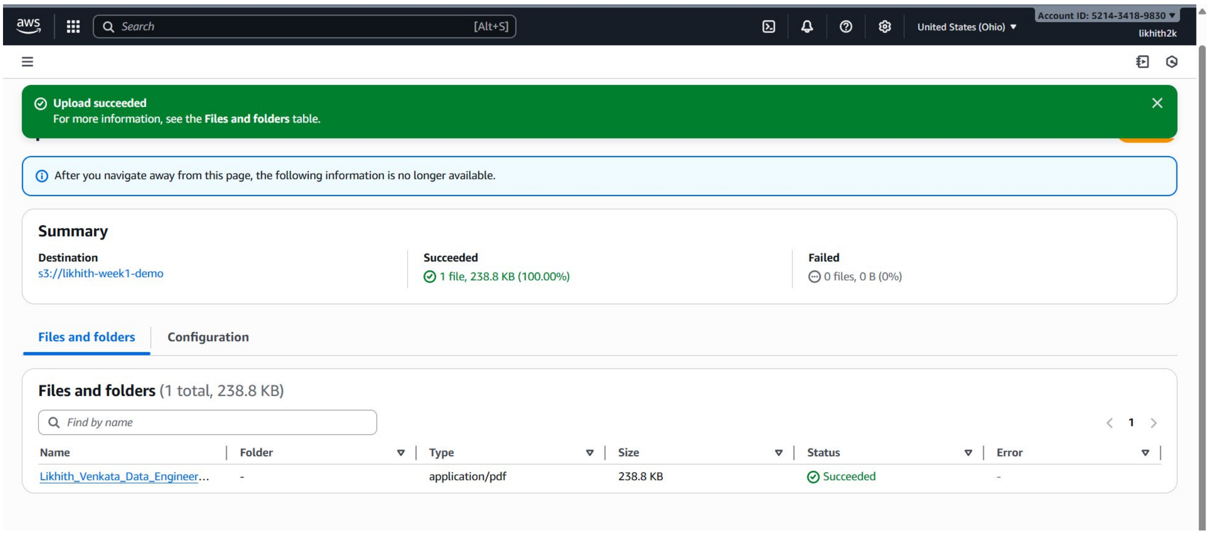
□ Hide Auto Scaling alarms | Clear selection | Create composite alarm | Actions ▼ | Create alarm

Search | Alarm state: Any ▼ | Alarm type: Any ▼ | Actions status: Any ▼ | < 1 >

□	Name	State	Last state update (UTC)	Conditions	Actions
□	max-cost-usage-alert	⊖ Insufficient data	2025-10-27 23:55:18	EstimatedCharges > 0 for 1 datapoints within 6 hours	✓ Actions enabled Warnin

S3 - Section

Here I have uploaded the files to S3,



Upload succeeded
For more information, see the **Files and folders** table.

After you navigate away from this page, the following information is no longer available.

Summary

Destination: s3://likhith-week1-demo

Succeeded: 1 file, 238.8 KB (100.00%)

Failed: 0 files, 0 B (0%)

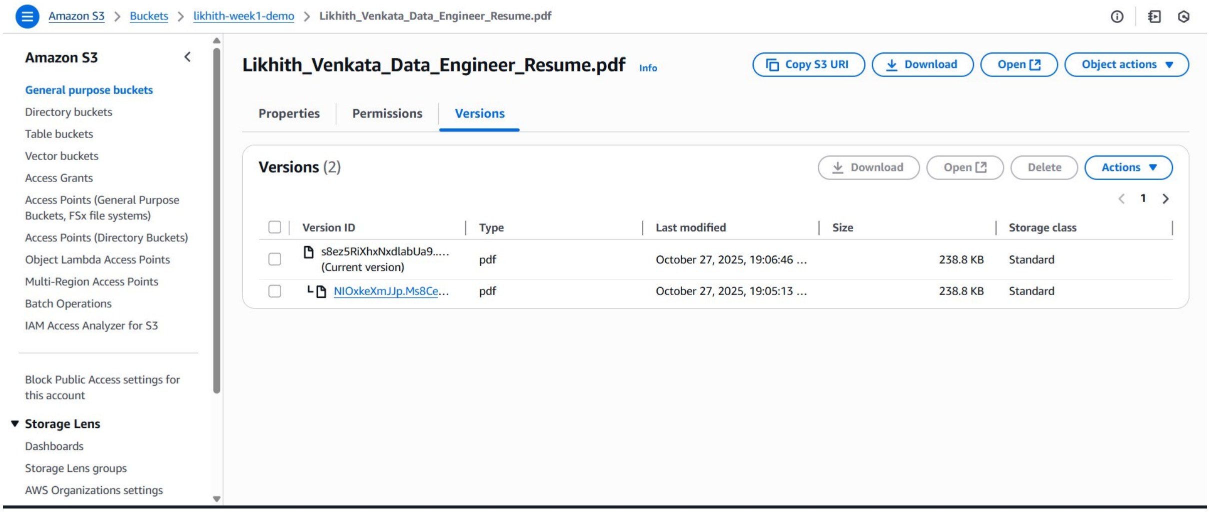
Files and folders (1 total, 238.8 KB)

Name	Folder	Type	Size	Status	Error
Likhith_Venkata_Data_Engineer…	-	application/pdf	238.8 KB	Succeeded	-

S3 Bucket Versioning Test

I uploaded the same PDF file twice to my S3 bucket. With versioning enabled, both versions of the file are saved. The screenshot below shows the **current version** and the **previous version**. This confirms that versioning is working correctly and older files can be restored if needed.

Screenshot inserted below.


Amazon S3 > Buckets > likhith-week1-demo > Likhith_Venkata_Data_Engineer_Resume.pdf
Amazon S3
General purpose buckets
Directory buckets
Table buckets
Vector buckets
Access Grants
Access Points (General Purpose Buckets, FSx file systems)
Access Points (Directory Buckets)
Object Lambda Access Points
Multi-Region Access Points
Batch Operations
IAM Access Analyzer for S3
Block Public Access settings for this account
Storage Lens
Dashboards
Storage Lens groups
AWS Organizations settings
Info
Copy S3 URI
Download
Open
Object actions
Properties
Permissions
Versions
Download
Open
Delete
Actions

Likhith_Venkata_Data_Engineer_Resume.pdf

Versions (2)

Version ID	Type	Last modified	Size	Storage class
s8ez5RiXhxNxdlabUa9..... (Current version)	pdf	October 27, 2025, 19:06:46 ...	238.8 KB	Standard
NIOxkeXmJJp.Ms8Ce...	pdf	October 27, 2025, 19:05:13 ...	238.8 KB	Standard

Summary :

Today I learned how AWS **Identity and Access Management (IAM)** helps control and secure user access across AWS services. I created **IAM users, groups, and roles**, assigned appropriate permissions, and understood the concept of least privilege for secure account management.

I also explored **Amazon S3**, where I created a bucket, uploaded multiple file types, and enabled **versioning** to maintain file history and data recovery. In addition, I configured **AWS CloudWatch billing alerts and alarms** to track usage and receive cost notifications, ensuring proactive budget management.

Finally, I set up a **GitHub repository** to store and manage all my bootcamp work, learned the basics of commits and pushing files, and organized my Day 1 deliverables systematically. Overall, I gained hands-on experience in foundational AWS services, account monitoring, and version control — key building blocks for cloud and data engineering.

OUBT Week 1 – Day 2–3: Python & Pandas & Boto3 SDK Practice

Author: Likhith Sasank Uppalapati Venkata

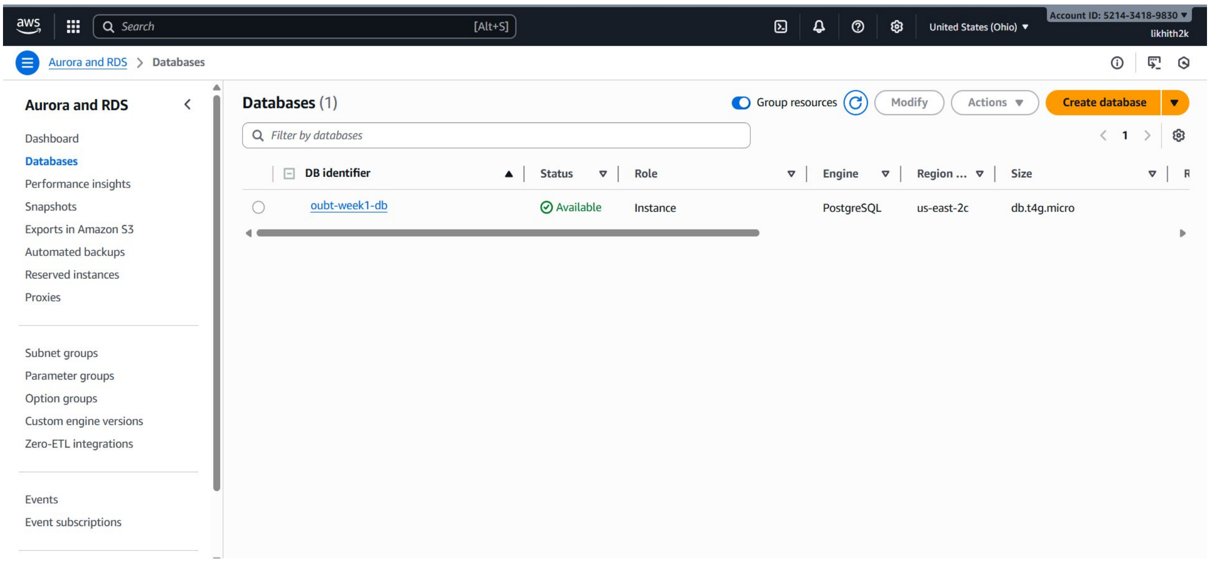
1. Python Practice

Practiced core Python concepts such as lists, dictionaries, loops, functions, and error handling.

Created and managed simple CSV files for student scores and invoice records.

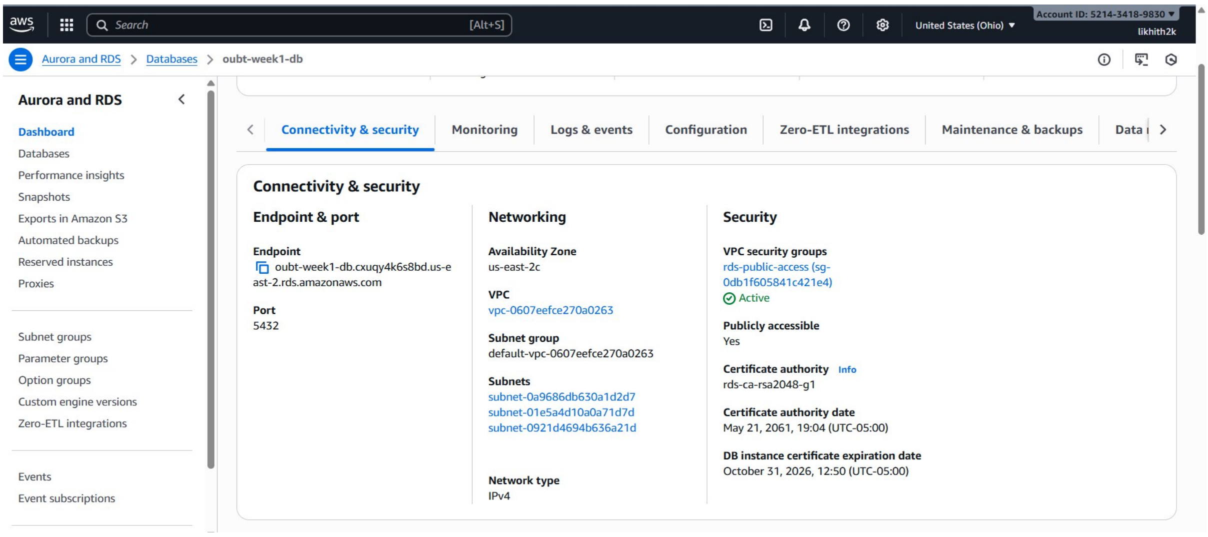
Worked with data filtering and conditional logic using Python structures.

Focused on writing clean, reusable functions and handling exceptions gracefully.



Databases (1)

DB identifier	Status	Role	Engine	Region ...	Size	R
oubt-week1-db	✓ Available	Instance	PostgreSQL	us-east-2c	db.t4g.micro	



Connectivity & security

Endpoint & port

Endpoint
oubt-week1-db.cxuqy4k6s8bd.us-east-2.rds.amazonaws.com

Port
5432

Networking

Availability Zone
us-east-2c

VPC
vpc-0607eefce270a0263

Subnet group
default-vpc-0607eefce270a0263

Subnets
subnet-0a9686db630a1d2d7
subnet-01e5a4d10a0a71d7d
subnet-0921d4694b636a21d

Network type
IPv4

Security

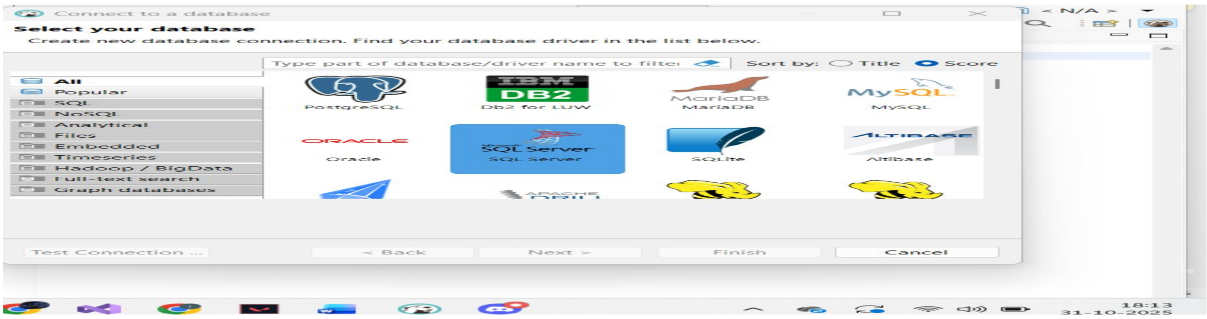
VPC security groups
rds-public-access (sg-0db1f605841c421e4)
✓ Active

Publicly accessible
Yes

Certificate authority Info
rds-ca-rsa2048-g1

Certificate authority date
May 21, 2061, 19:04 (UTC-05:00)

DB instance certificate expiration date
October 31, 2026, 12:50 (UTC-05:00)



Connect to a database

Select your database

Create new database connection. Find your database driver in the list below.

Sort by: ○ Title ● Score

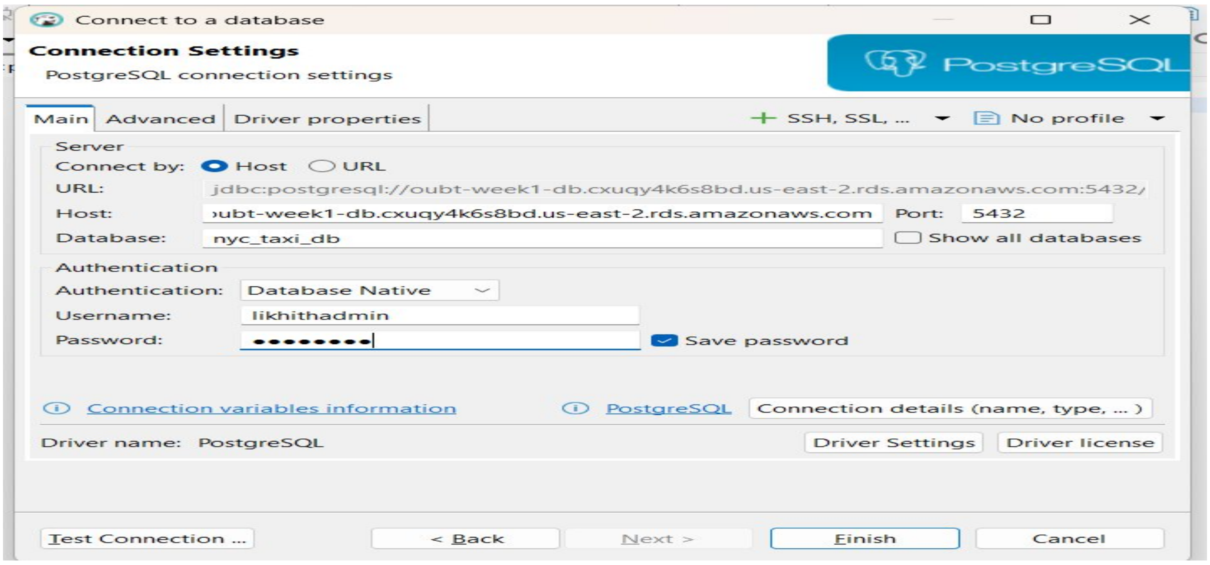
2. Pandas Practice

Learned to create, read, and manipulate datasets using Pandas.

Performed basic data cleaning by removing duplicates and adding calculated columns.

Aggregated and analyzed data to find total sales and quantities.

Combined multiple datasets into a single, cleaned file for further use.



Connect to a database

Connection Settings

PostgreSQL connection settings

Server

Connect by: ● Host ○ URL

URL: jdbc:postgresql://oubt-week1-db.cxuqy4k6s8bd.us-east-2.rds.amazonaws.com:5432/

Host: ›ubt-week1-db.cxuqy4k6s8bd.us-east-2.rds.amazonaws.com Port: 5432

Database: nyc_taxi_db ☐ Show all databases

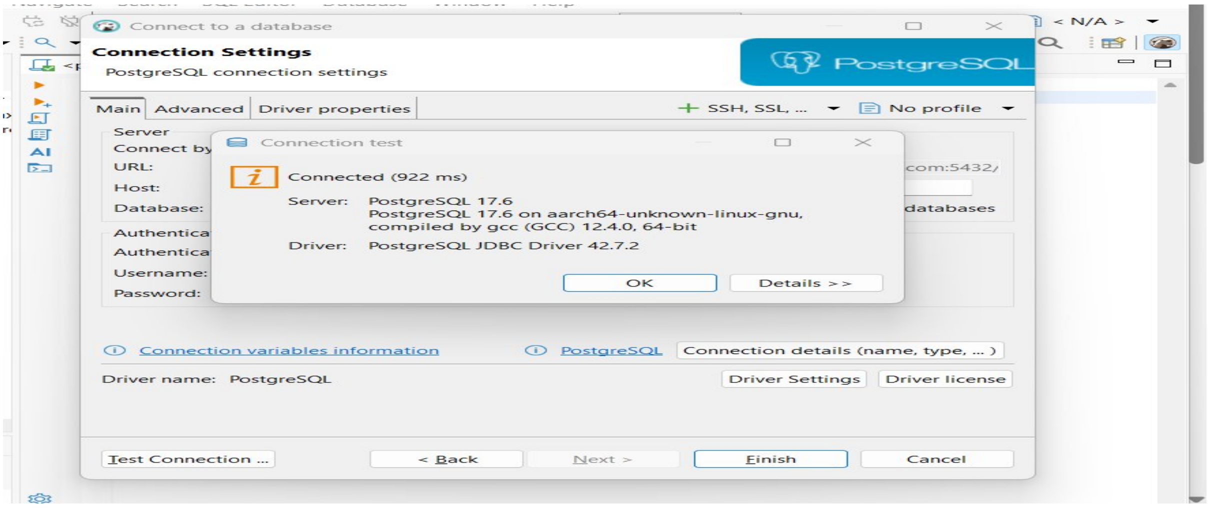
Authentication

Authentication: Database Native

Username: likhithadmin

Password: ●●●●●●●●● ☑ Save password

Driver name: PostgreSQL



Connect to a database

Connection Settings

PostgreSQL connection settings

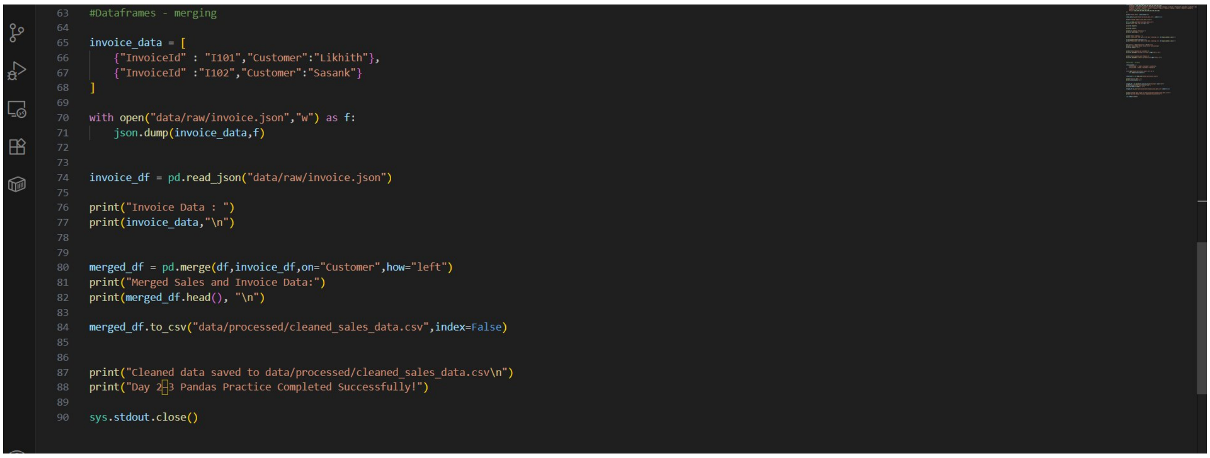
Connection test

Connected (922 ms)

Server: PostgreSQL 17.6
PostgreSQL 17.6 on aarch64-unknown-linux-gnu, compiled by gcc (GCC) 12.4.0, 64-bit

Driver: PostgreSQL JDBC Driver 42.7.2

Driver name: PostgreSQL

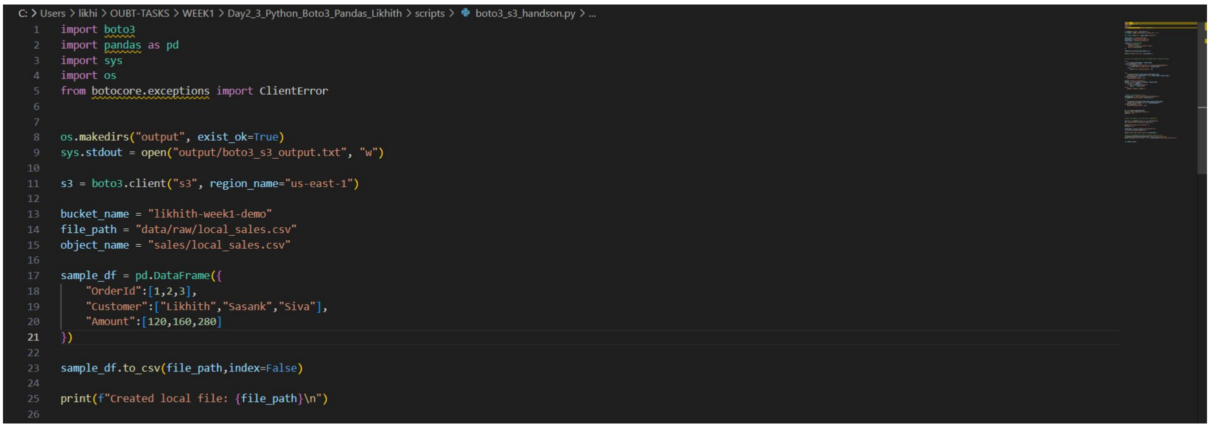


```
#Dataframes - merging

invoice_data = [
    {"InvoiceId" : "I101","Customer":"Likhith"},
    {"InvoiceId" :"I102","Customer":"Sasank"}
]

with open("data/raw/invoice.json","w") as f:
    json.dump(invoice_data,f)


invoice_df = pd.read_json("data/raw/invoice.json")

print("Invoice Data : ")
print(invoice_data,"\n")


merged_df = pd.merge(df,invoice_df,on="Customer",how="left")
print("Merged Sales and Invoice Data:")
print(merged_df.head(), "\n")

merged_df.to_csv("data/processed/cleaned_sales_data.csv",index=False)


print("Cleaned data saved to data/processed/cleaned_sales_data.csv\n")
print("Day 2□3 Pandas Practice Completed Successfully!")

sys.stdout.close()
```

3. AWS S3 (Boto3) Practice

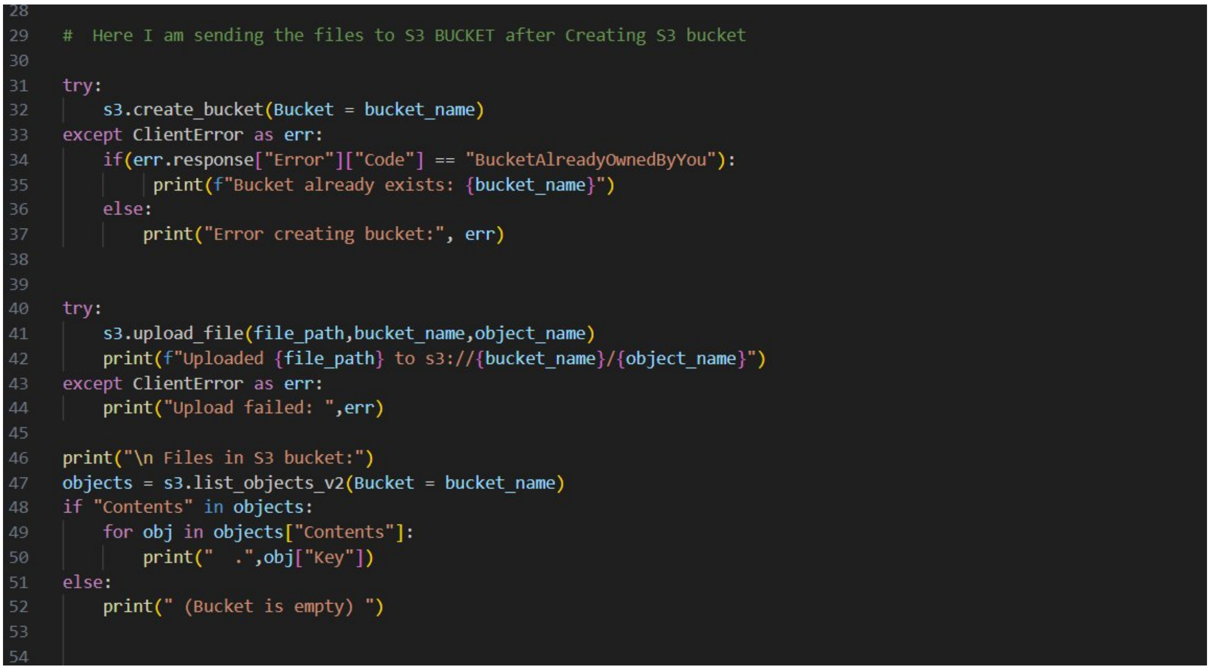
Installed and configured Boto3 to connect Python with AWS S3.

Created and managed an S3 bucket for uploading, listing, and downloading files.

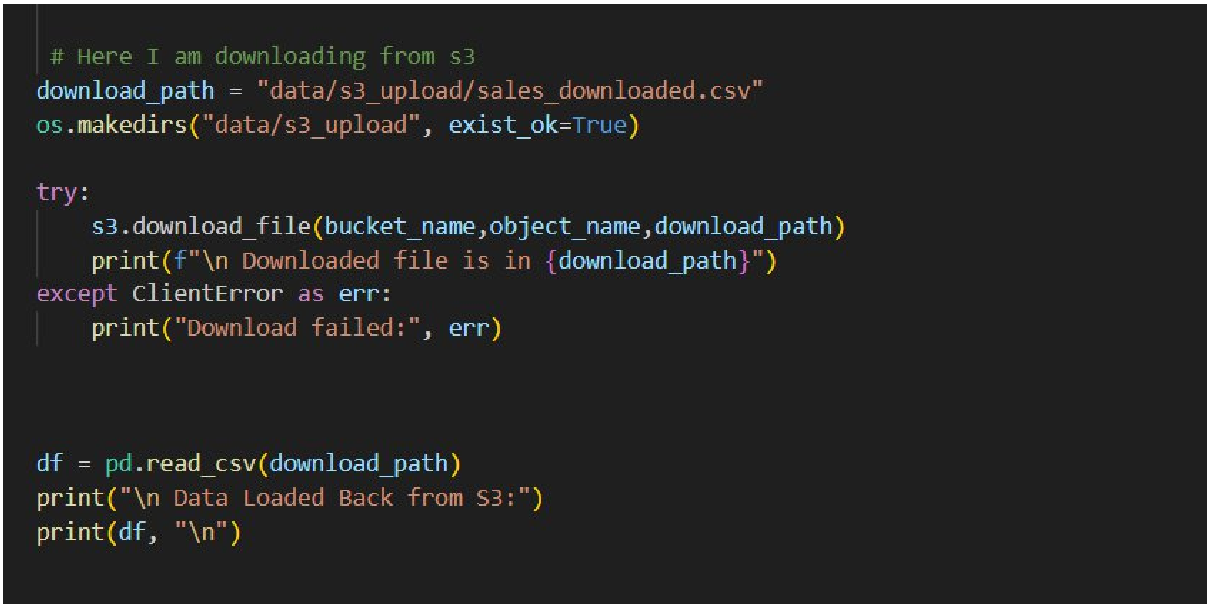

C: > Users > likhi > OUBT-TASKS > WEEK1 > Day2_3_Python_Boto3_Pandas_Likhith > scripts > boto3_s3_handson.py > ...

```
import boto3
import pandas as pd
import sys
import os
from botocore.exceptions import ClientError


os.makedirs("output", exist_ok=True)
sys.stdout = open("output/boto3_s3_output.txt", "w")

s3 = boto3.client("s3", region_name="us-east-1")

bucket_name = "likhith-week1-demo"
file_path = "data/raw/local_sales.csv"
object_name = "sales/local_sales.csv"

sample_df = pd.DataFrame({
    "OrderId":[1,2,3],
    "Customer":["Likhith","Sasank","Siva"],
    "Amount":[120,160,280]
})

sample_df.to_csv(file_path,index=False)

print(f"Created local file: {file_path}\n")
```

Integrated Pandas with AWS S3 to upload processed data and retrieve it for analysis.

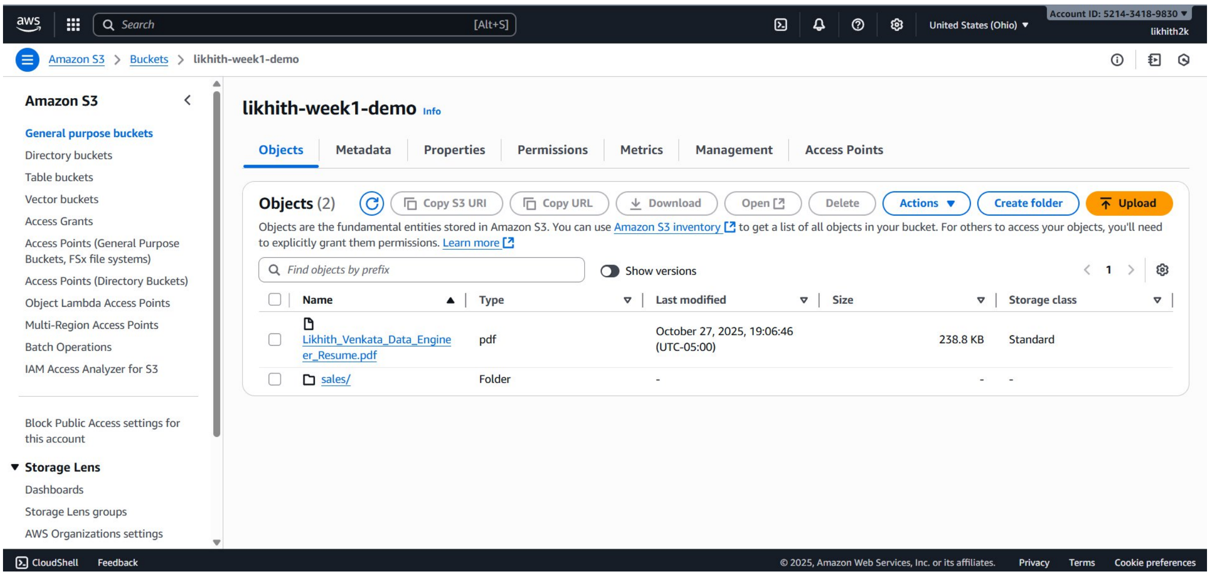


```
#  Here I am sending the files to S3 BUCKET after Creating S3 bucket

try:
    s3.create_bucket(Bucket = bucket_name)
except ClientError as err:
    if(err.response["Error"]["Code"] == "BucketAlreadyOwnedByYou"):
        print(f"Bucket already exists: {bucket_name}")
    else:
        print("Error creating bucket:", err)


try:
    s3.upload_file(file_path,bucket_name,object_name)
    print(f"Uploaded {file_path} to s3://{bucket_name}/{object_name}")
except ClientError as err:
    print("Upload failed: ",err)

print("\n Files in S3 bucket:")
objects = s3.list_objects_v2(Bucket = bucket_name)
if "Contents" in objects:
    for obj in objects["Contents"]:
        print("  .",obj["Key"])
else:
    print(" (Bucket is empty) ")
```



```
# Here I am downloading from s3
download_path = "data/s3_upload/sales_downloaded.csv"
os.makedirs("data/s3_upload", exist_ok=True)

try:
    s3.download_file(bucket_name,object_name,download_path)
    print(f"\n Downloaded file is in {download_path}")
except ClientError as err:
    print("Download failed:", err)



df = pd.read_csv(download_path)
print("\n Data Loaded Back from S3:")
print(df, "\n")
```

Practiced automating simple data workflows between local and cloud storage.



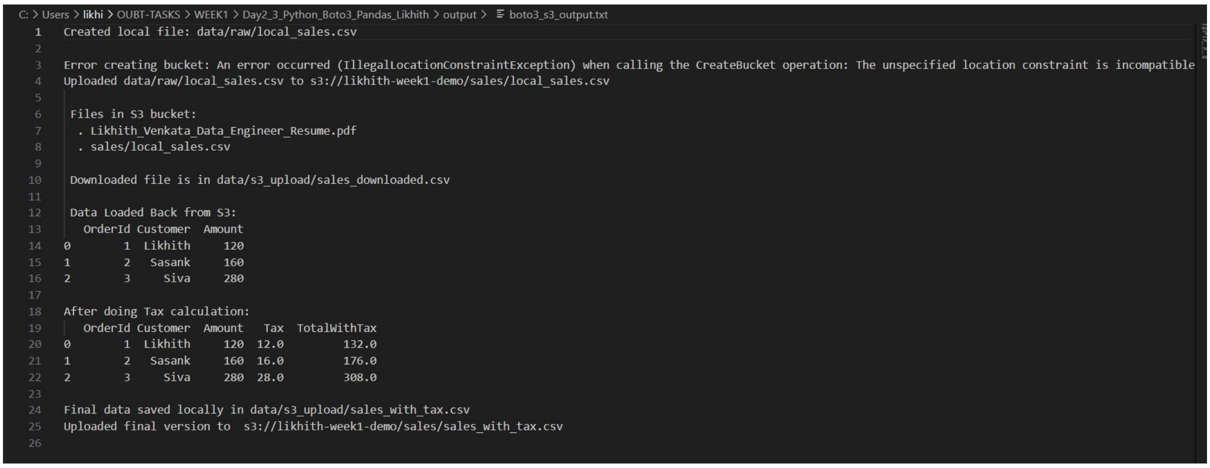
likhith-week1-demo Info

Objects (2)

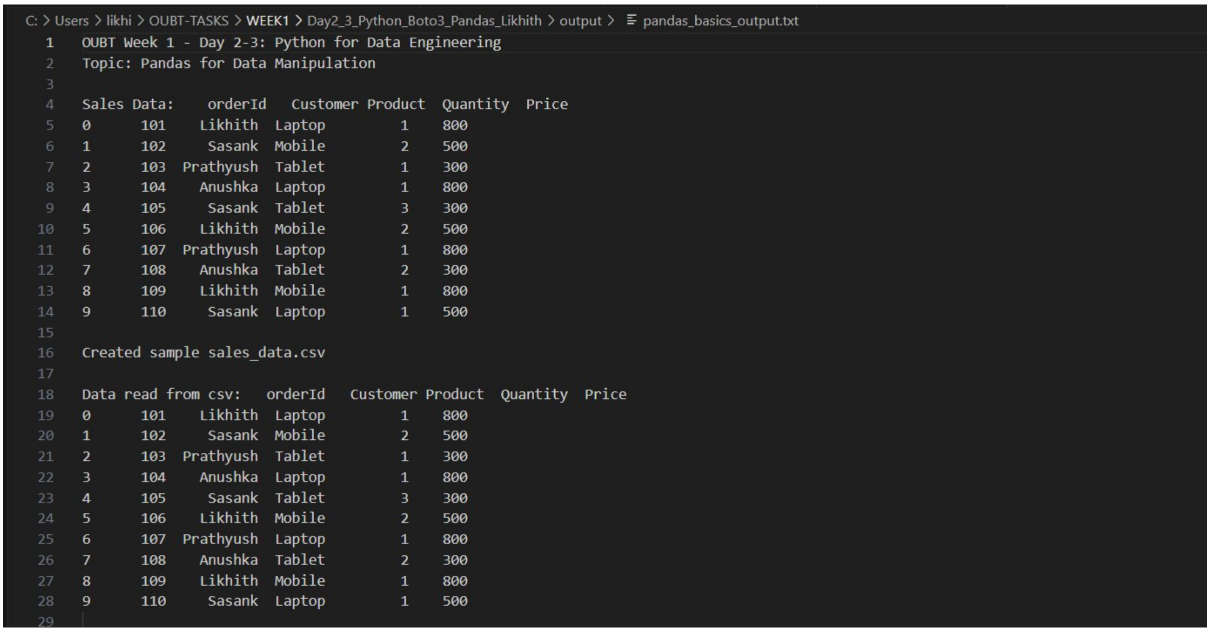
Objects are the fundamental entities stored in Amazon S3. You can use Amazon S3 inventory to get a list of all objects in your bucket. For others to access your objects, you'll need to explicitly grant them permissions. Learn more

□	Name ▲	Type	Last modified	Size	Storage class
□	Likhith_Venkata_Data_Engineer_Resume.pdf	pdf	October 27, 2025, 19:06:46 (UTC-05:00)	238.8 KB	Standard
□	sales/	Folder	-	-	-

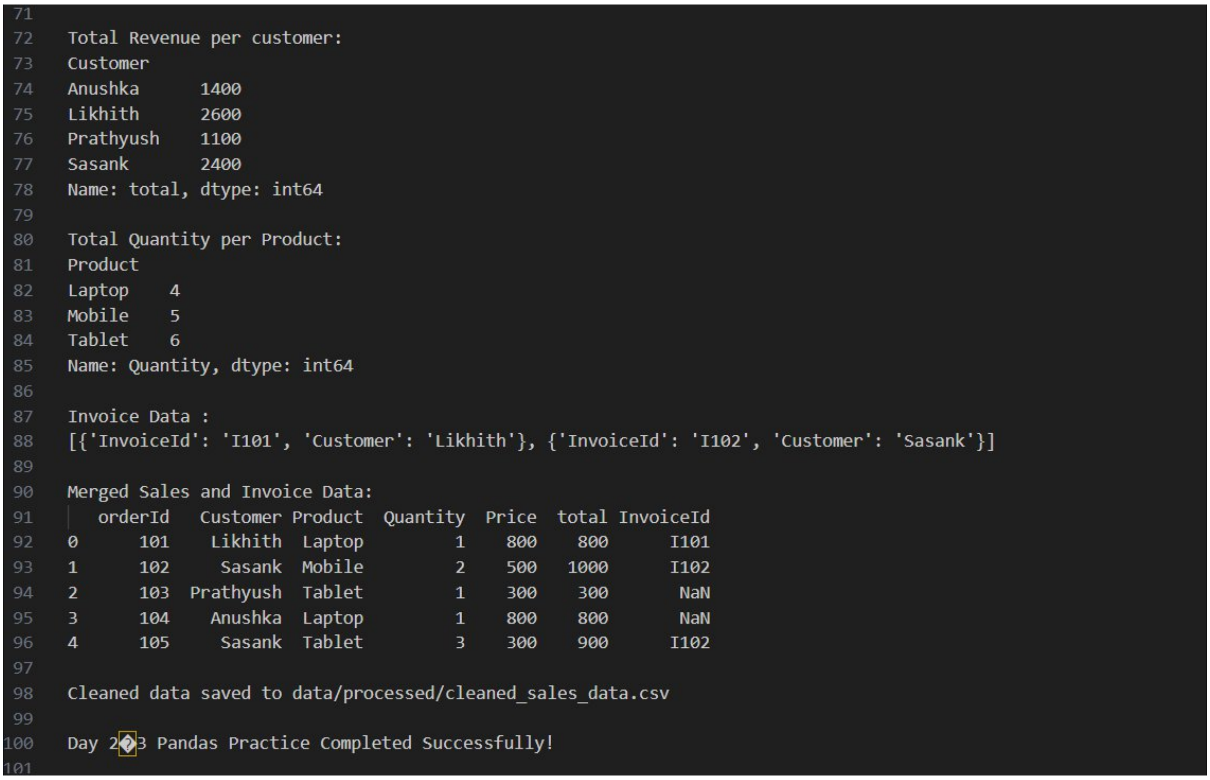
All Python Scripts outputs have been saved and organized in a separate folder for reference and documentation purposes.


C: > Users > likhi > OUBT-TASKS > WEEK1 > Day2_3_Python_Boto3_Pandas_Likhith > output > boto3_s3_output.txt

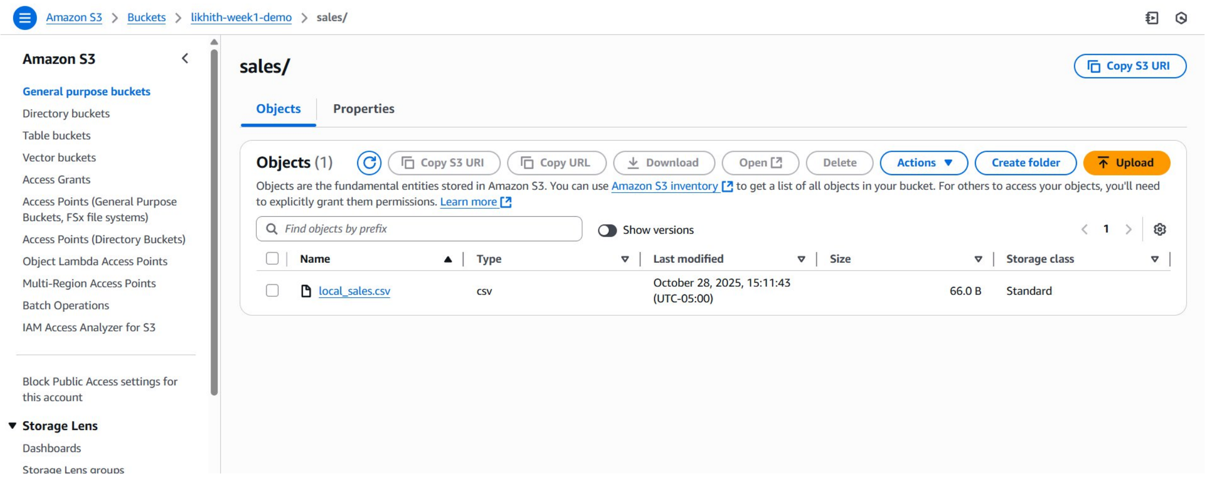
```
Created local file: data/raw/local_sales.csv

Error creating bucket: An error occurred (IllegalLocationConstraintException) when calling the CreateBucket operation: The unspecified location constraint is incompatible
Uploaded data/raw/local_sales.csv to s3://likhith-week1-demo/sales/local_sales.csv

 Files in S3 bucket:
  . Likhith_Venkata_Data_Engineer_Resume.pdf
  . sales/local_sales.csv

 Downloaded file is in data/s3_upload/sales_downloaded.csv

 Data Loaded Back from S3:
   OrderId Customer  Amount
0        1  Likhith     120
1        2   Sasank     160
2        3     Siva     280

After doing Tax calculation:
   OrderId Customer  Amount   Tax  TotalWithTax
0        1  Likhith     120  12.0         132.0
1        2   Sasank     160  16.0         176.0
2        3     Siva     280  28.0         308.0

Final data saved locally in data/s3_upload/sales_with_tax.csv
Uploaded final version to  s3://likhith-week1-demo/sales/sales_with_tax.csv
```


C: > Users > likhi > OUBT-TASKS > WEEK1 > Day2_3_Python_Boto3_Pandas_Likhith > output > pandas_basics_output.txt

```
OUBT Week 1 - Day 2-3: Python for Data Engineering
Topic: Pandas for Data Manipulation

Sales Data:    orderId   Customer Product  Quantity  Price
0       101    Likhith  Laptop         1    800
1       102     Sasank  Mobile         2    500
2       103  Prathyush  Tablet         1    300
3       104    Anushka  Laptop         1    800
4       105     Sasank  Tablet         3    300
5       106    Likhith  Mobile         2    500
6       107  Prathyush  Laptop         1    800
7       108    Anushka  Tablet         2    300
8       109    Likhith  Mobile         1    800
9       110     Sasank  Laptop         1    500

Created sample sales_data.csv

Data read from csv:    orderId   Customer Product  Quantity  Price
0       101    Likhith  Laptop         1    800
1       102     Sasank  Mobile         2    500
2       103  Prathyush  Tablet         1    300
3       104    Anushka  Laptop         1    800
4       105     Sasank  Tablet         3    300
5       106    Likhith  Mobile         2    500
6       107  Prathyush  Laptop         1    800
7       108    Anushka  Tablet         2    300
8       109    Likhith  Mobile         1    800
9       110     Sasank  Laptop         1    500
```



```

Total Revenue per customer:
Customer
Anushka      1400
Likhith      2600
Prathyush    1100
Sasank       2400
Name: total, dtype: int64

Total Quantity per Product:
Product
Laptop    4
Mobile    5
Tablet    6
Name: Quantity, dtype: int64

Invoice Data :
[{'InvoiceId': 'I101', 'Customer': 'Likhith'}, {'InvoiceId': 'I102', 'Customer': 'Sasank'}]

Merged Sales and Invoice Data:
   orderId   Customer Product  Quantity  Price  total InvoiceId
0      101    Likhith  Laptop         1    800    800      I101
1      102     Sasank  Mobile         2    500   1000      I102
2      103  Prathyush  Tablet         1    300    300       NaN
3      104    Anushka  Laptop         1    800    800       NaN
4      105     Sasank  Tablet         3    300    900      I102

Cleaned data saved to data/processed/cleaned_sales_data.csv

Day 2�3 Pandas Practice Completed Successfully!

```

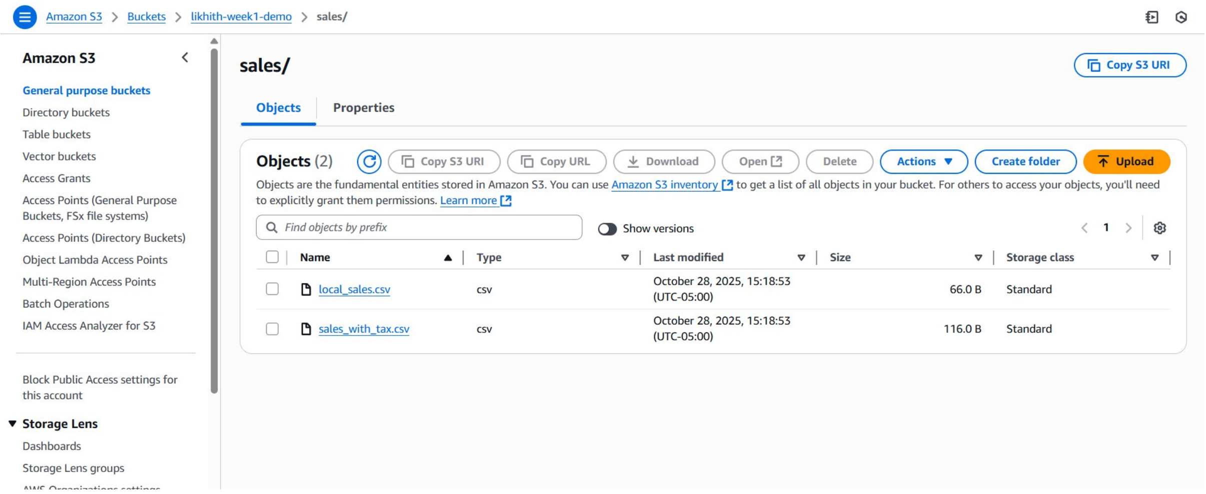


sales/

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
□	local_sales.csv	csv	October 28, 2025, 15:11:43 (UTC-05:00)	66.0 B	Standard



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
□	local_sales.csv	csv	October 28, 2025, 15:18:53 (UTC-05:00)	66.0 B	Standard
□	sales_with_tax.csv	csv	October 28, 2025, 15:18:53 (UTC-05:00)	116.0 B	Standard

4. Summary

During these sessions, I improved my understanding of Python and file handling, worked on data cleaning and analysis using Pandas, and practiced using AWS S3 with Boto3 for cloud data management. I also organized my work with a proper folder structure, keeping raw, processed, and uploaded data separate, and saving all outputs and screenshots in their own folders. This helped me follow good project practices and build a smooth workflow that connects Python, Pandas, and AWS for managing and analyzing data.

OUBT Week-1 – Day 4: SQL & Data Modeling basics

NAME: LIKHITH SASANK UPPALAPATI VENKATA

Overview:

Day 4 focused on understanding database fundamentals and data modeling concepts. I learned about SQL basics, relational vs. non-relational databases, ACID properties, normalization, and schema evolution. I also studied how to design a proper data model using entities, attributes, and relationships, and explored star schema and Slowly Changing Dimensions (SCD Types 1 & 2). Along with this, I practiced SQL operations like joins, aggregations, and window functions on sample data to strengthen query skills before working with AWS RDS.

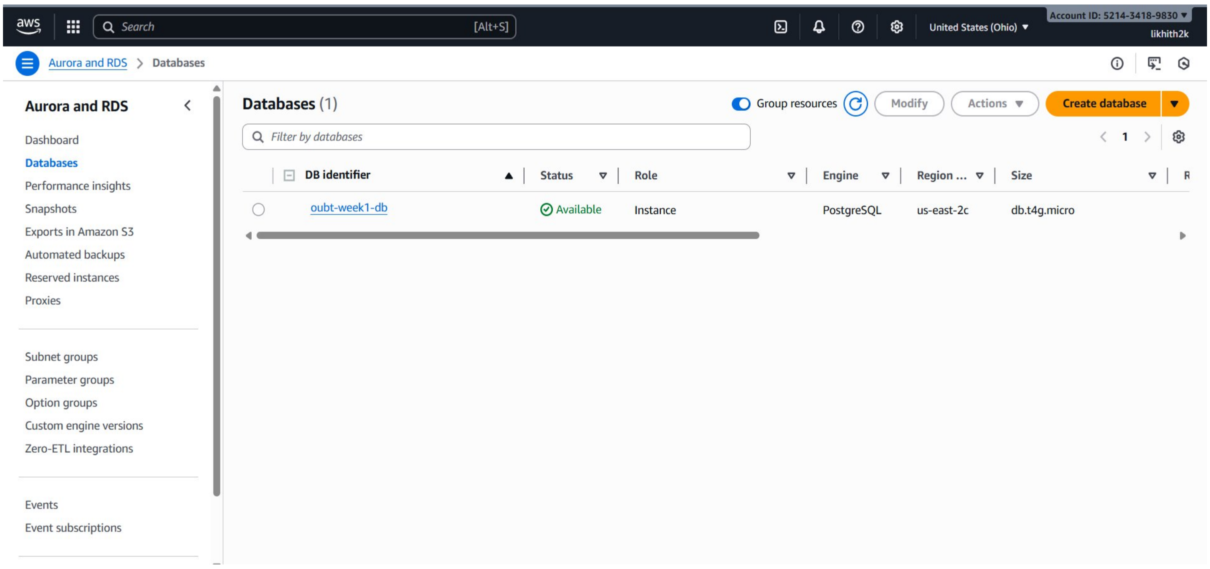
Hands-On Practice:

Created sample tables and inserted data for customers, products, and orders.

Practiced SQL operations such as:

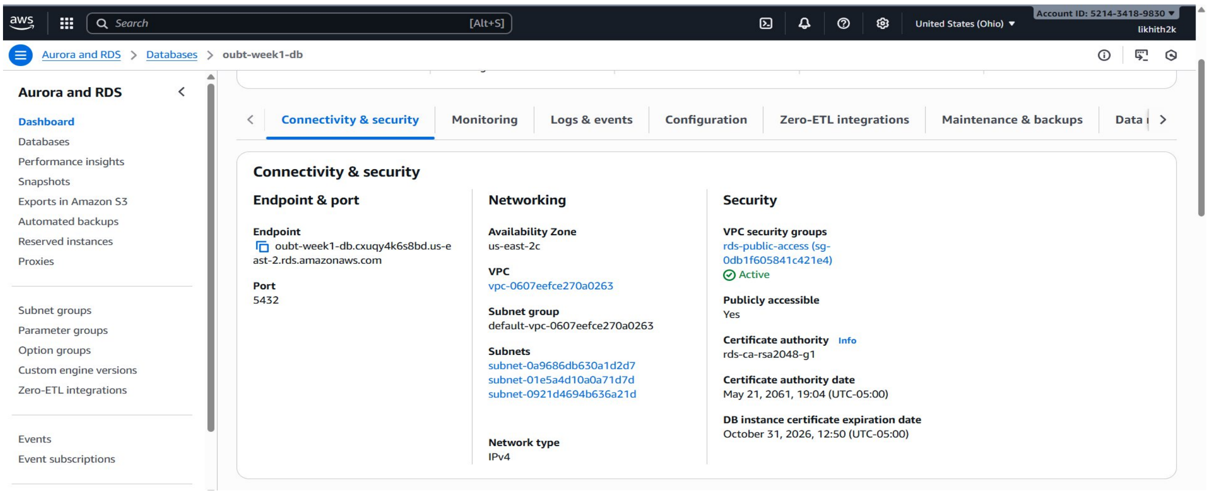
Basic filtering using WHERE.
Aggregations with COUNT(), SUM(), and AVG().
Joins between multiple tables to fetch related information.
Grouping and ordering results.
Used the RANK() window function to rank customers by total spending.

Designed the schema in a **star-schema pattern** with Orders and OrderDetails as fact tables and Customers, Products as dimensions.



Databases (1)

	DB identifier ▲	Status	Role	Engine	Region ...	Size	R
○	oubt-week1-db	✓ Available	Instance	PostgreSQL	us-east-2c	db.t4g.micro	



Connectivity & security

Endpoint & port

Endpoint
oubt-week1-db.cxuqy4k6s8bd.us-east-2.rds.amazonaws.com

Port
5432

Networking

Availability Zone
us-east-2c

VPC
vpc-0607eefce270a0263

Subnet group
default-vpc-0607eefce270a0263

Subnets
subnet-0a9686db630a1d2d7
subnet-01e5a4d10a0a71d7d
subnet-0921d4694b636a21d

Network type
IPv4

Security

VPC security groups
rds-public-access (sg-0db1f605841c421e4)
✓ Active

Publicly accessible
Yes

Certificate authority Info
rds-ca-rsa2048-g1

Certificate authority date
May 21, 2061, 19:04 (UTC-05:00)

DB instance certificate expiration date
October 31, 2026, 12:50 (UTC-05:00)

SQL – QUERIES:

SELECT * FROM Customers WHERE Country = 'USA';

-- List all products under $500

SELECT ProductName, Price FROM Products WHERE Price < 500;

-- Count total number of orders

SELECT COUNT(*) AS TotalOrders FROM Orders;

-- Get all orders for customer 'Likhith'

SELECT o.OrderID, o.OrderDate, o.TotalAmount FROM Orders o JOIN Customers c ON o.CustomerID = c.CustomerID

WHERE c.Name = 'Likhith';

```sql
--  Total spent by each customer
SELECT c.Name, SUM(o.TotalAmount) AS TotalSpent
FROM Customers c
JOIN Orders o ON c.CustomerID = o.CustomerID
GROUP BY c.Name;

--  List all orders with product names
SELECT o.OrderID, c.Name AS CustomerName, p.ProductName, od.Quantity
FROM Orders o
JOIN Customers c ON o.CustomerID = c.CustomerID
JOIN OrderDetails od ON o.OrderID = od.OrderID
JOIN Products p ON od.ProductID = p.ProductID;

--  Find top 2 customers by spending
SELECT c.Name, SUM(o.TotalAmount) AS TotalSpent
FROM Customers c
JOIN Orders o ON c.CustomerID = o.CustomerID
GROUP BY c.Name
ORDER BY TotalSpent DESC
LIMIT 2;

--  Find products never ordered
SELECT p.ProductName
FROM Products p
LEFT JOIN OrderDetails od ON p.ProductID = od.ProductID
WHERE od.ProductID IS NULL;
```

-- Average order amount per country

```
SELECT c.Country, AVG(o.TotalAmount) AS AvgOrderAmount

FROM Customers c

JOIN Orders o ON c.CustomerID = o.CustomerID

GROUP BY c.Country;
```

SQL Query results Screenshots:

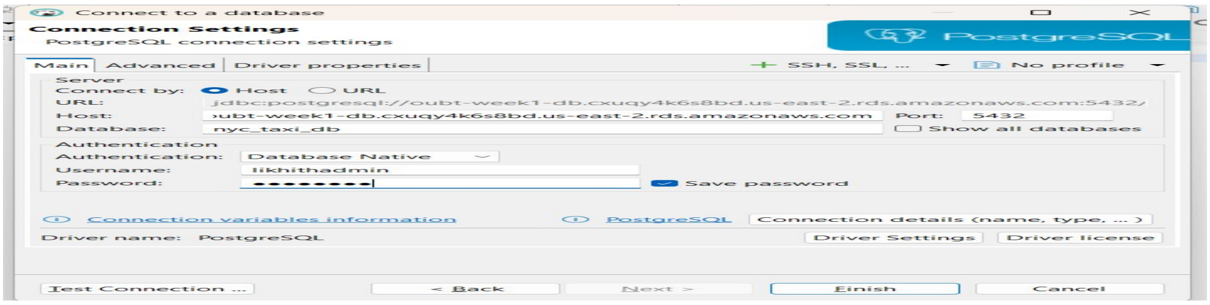
Query 1: Fetch all customers from the USA

Connect to a database

Select your database

Create new database connection. Find your database driver in the list below.

Query 2: List all products priced under $500



Connect to a database

Connection Settings

PostgreSQL connection settings

Server

Connect by: ● Host ○ URL

URL: jdbc:postgresql://oubt-week1-db.cxuqy4k6s8bd.us-east-2.rds.amazonaws.com:5432/

Host: ›ubt-week1-db.cxuqy4k6s8bd.us-east-2.rds.amazonaws.com Port: 5432

Database: nyc_taxi_db □ Show all databases

Authentication

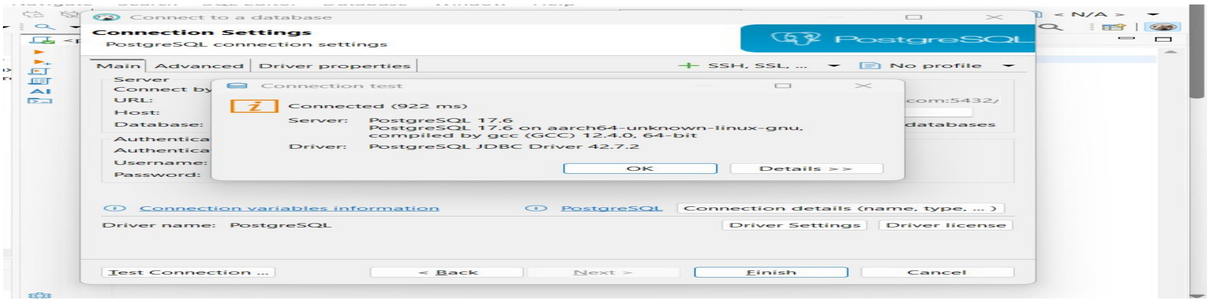
Authentication: Database Native

Username: likhithadmin

Password: ●●●●●●●●● ☑ Save password

Driver name: PostgreSQL

Query 3: Count total number of orders



Connect to a database

Connection Settings

PostgreSQL connection settings

Connection test

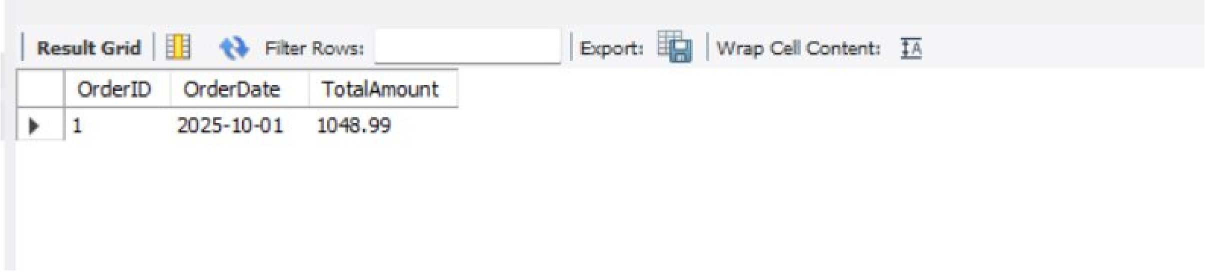
Connected (922 ms)

Server: PostgreSQL 17.6
PostgreSQL 17.6 on aarch64-unknown-linux-gnu,
compiled by gcc (GCC) 12.4.0, 64-bit

Driver: PostgreSQL JDBC Driver 42.7.2

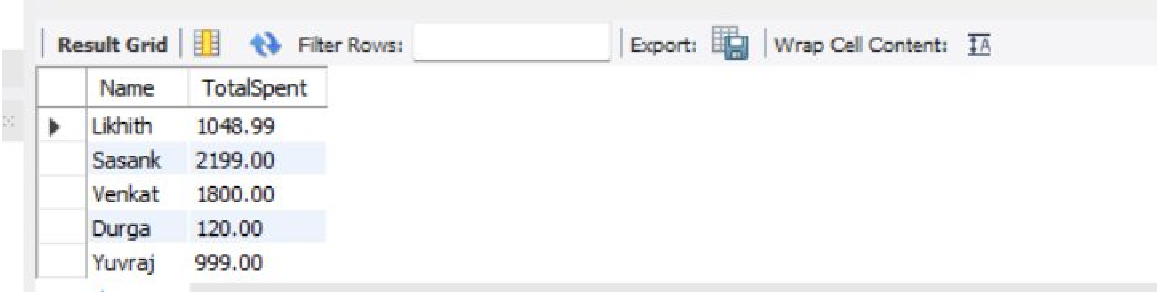
Driver name: PostgreSQL

Query 4: Retrieve all orders made by customer “Likhith”



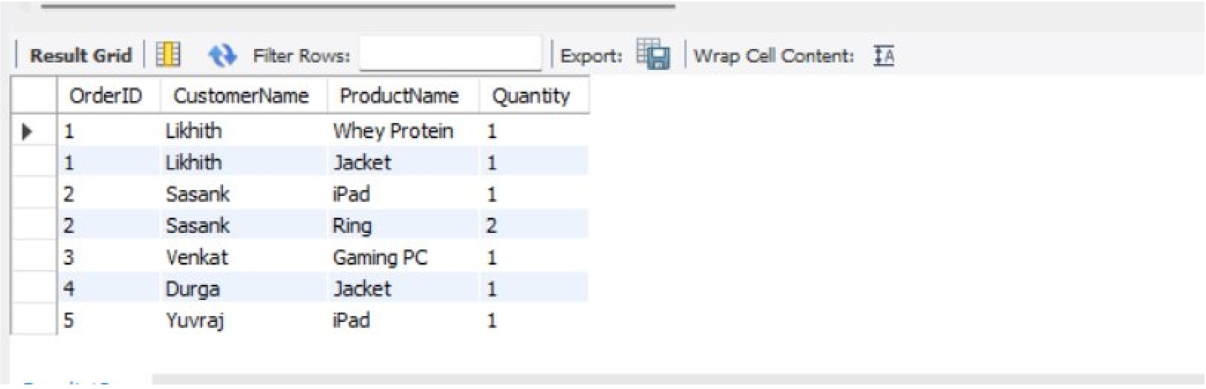
OrderID	OrderDate	TotalAmount
1	2025-10-01	1048.99

Query 5: Calculate total amount spent by each customer



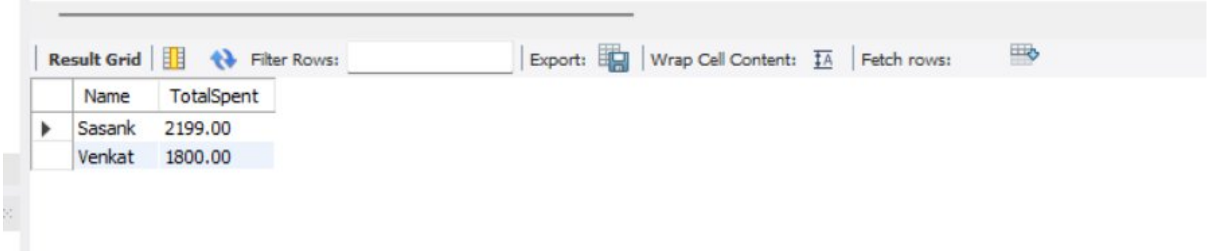
Name	TotalSpent
Likhith	1048.99
Sasank	2199.00
Venkat	1800.00
Durga	120.00
Yuvraj	999.00

Query 6: Display all orders with product names and quantities



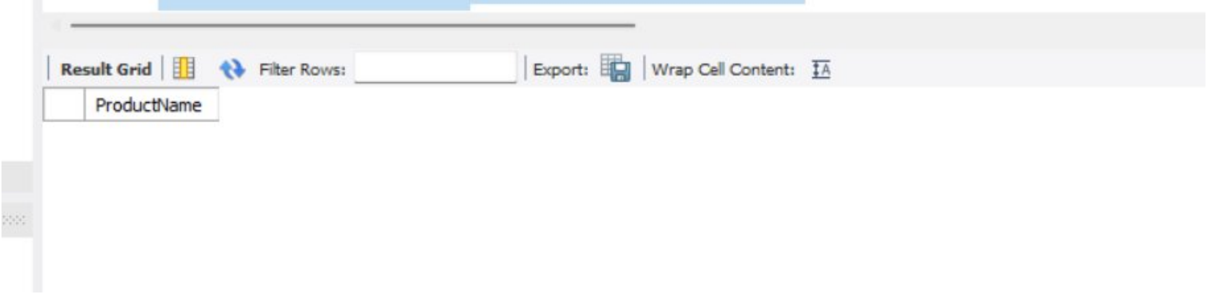
OrderID	CustomerName	ProductName	Quantity
1	Likhith	Whey Protein	1
1	Likhith	Jacket	1
2	Sasank	iPad	1
2	Sasank	Ring	2
3	Venkat	Gaming PC	1
4	Durga	Jacket	1
5	Yuvraj	iPad	1

Query 7: Find top 2 customers based on total spending



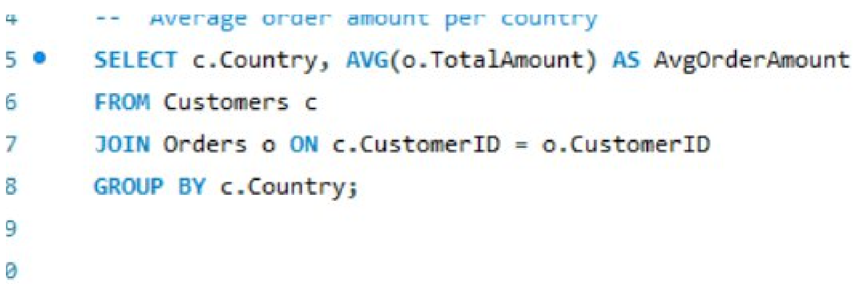
Name	TotalSpent
Sasank	2199.00
Venkat	1800.00

Query 8: Identify products that were never ordered

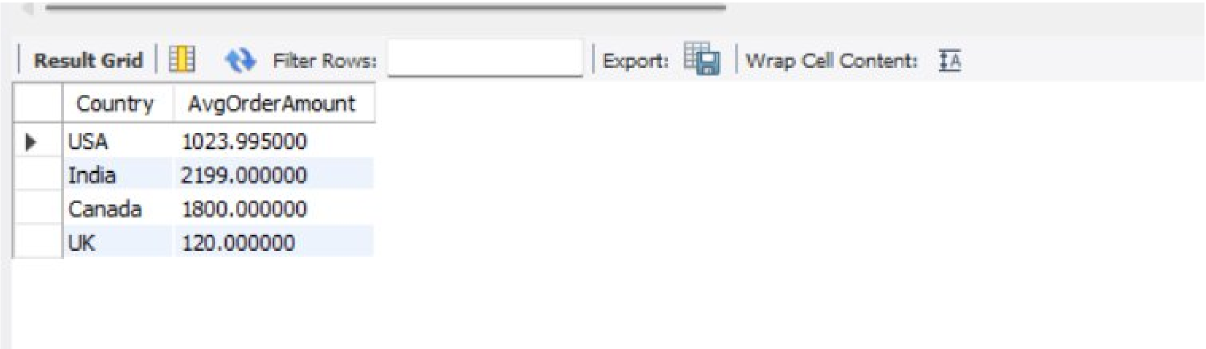


ProductName

Query 9: Find average order amount by country



```
-- Average order amount per country
SELECT c.Country, AVG(o.TotalAmount) AS AvgOrderAmount
FROM Customers c
JOIN Orders o ON c.CustomerID = o.CustomerID
GROUP BY c.Country;
```



Country	AvgOrderAmount
USA	1023.995000
India	2199.000000
Canada	1800.000000
UK	120.000000

Key Learnings:

Learned how relational databases store and manage structured data.

Understood **ACID properties**, normalization, and schema design concepts.

Practiced real SQL operations – joins, aggregations, and ranking queries.

Learned how **star schema modeling** supports analytical queries.

Improved understanding of how to represent one-to-many and many-to-many relationships.

Summary:

Overall, Day 4 helped me connect theoretical database concepts with real SQL practice. Building the e-commerce schema gave me a clear idea of how relational data is structured, queried, and optimized. This day built a strong base for moving into AWS RDS and advanced data modeling in the next steps.

OUBT Week-1 – Day 5: SQL & Data Modeling (AWS RDS)

NAME : LIKHITH SASANK UPPALAPATI VENKATA

Overview:

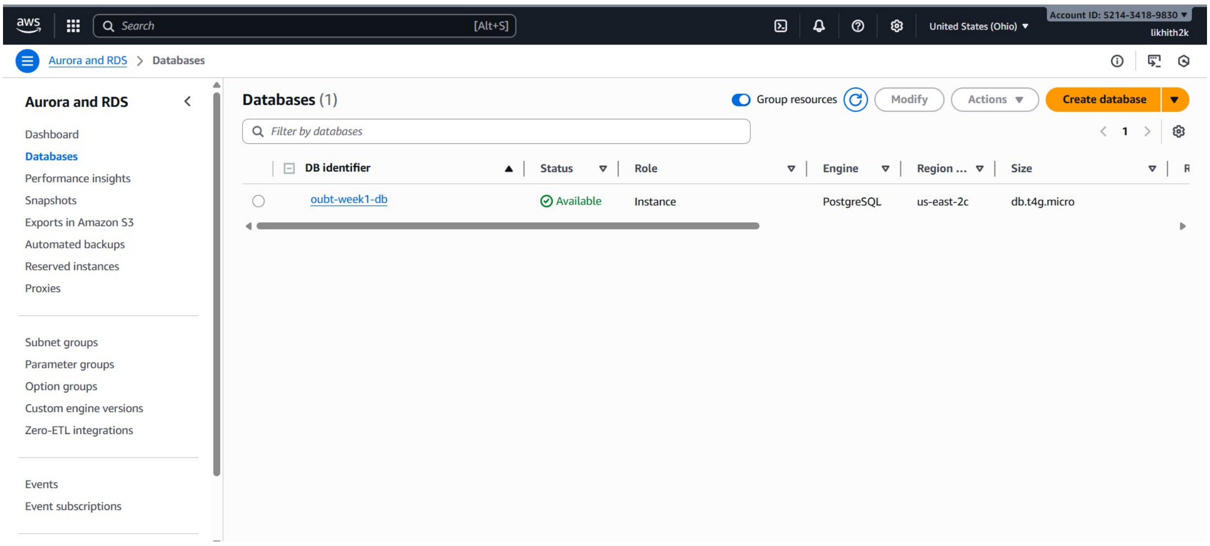
Day 5 focused on setting up and working with a live PostgreSQL database on AWS RDS. I created the instance, configured IAM access, and connected through DBeaver to build tables and run SQL queries on sample taxi data. I also checked RDS metrics in CloudWatch to confirm query activity and understand how CPU and connections change when the database is in use. This helped me connect SQL practice with real AWS database monitoring and management

Objective:

Design a simple relational schema on AWS RDS PostgreSQL and practice SQL operations on sample NYC Taxi Trips data.

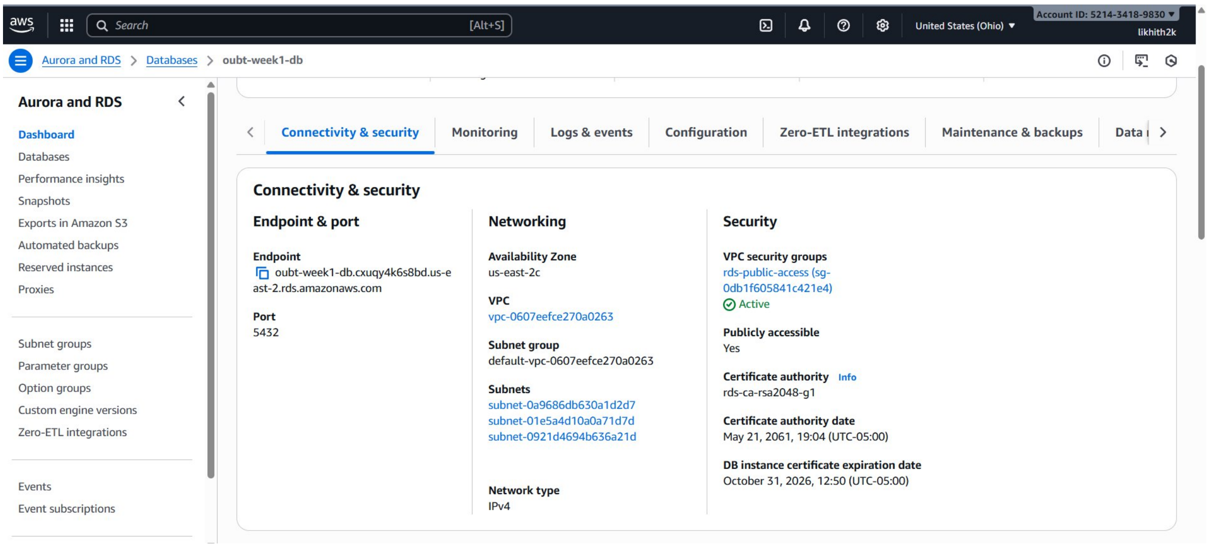
Steps Performed:

Created an **AWS RDS PostgreSQL** instance under Free Tier with public access enabled.



DB identifier	Status	Role	Engine	Region ...	Size	R
oubt-week1-db	Available	Instance	PostgreSQL	us-east-2c	db.t4g.micro	

Configured inbound rules (port 5432) to allow access from local machine.



Connectivity & security

Endpoint & port

Endpoint
oubt-week1-db.cxuqy4k6s8bd.us-east-2.rds.amazonaws.com

Port
5432

Networking

Availability Zone
us-east-2c

VPC
vpc-0607eefce270a0263

Subnet group
default-vpc-0607eefce270a0263

Subnets
subnet-0a9686db630a1d2d7
subnet-01e5a4d10a0a71d7d
subnet-0921d4694b636a21d

Network type
IPv4

Security

VPC security groups
rds-public-access (sg-0db1f605841c421e4)
Active

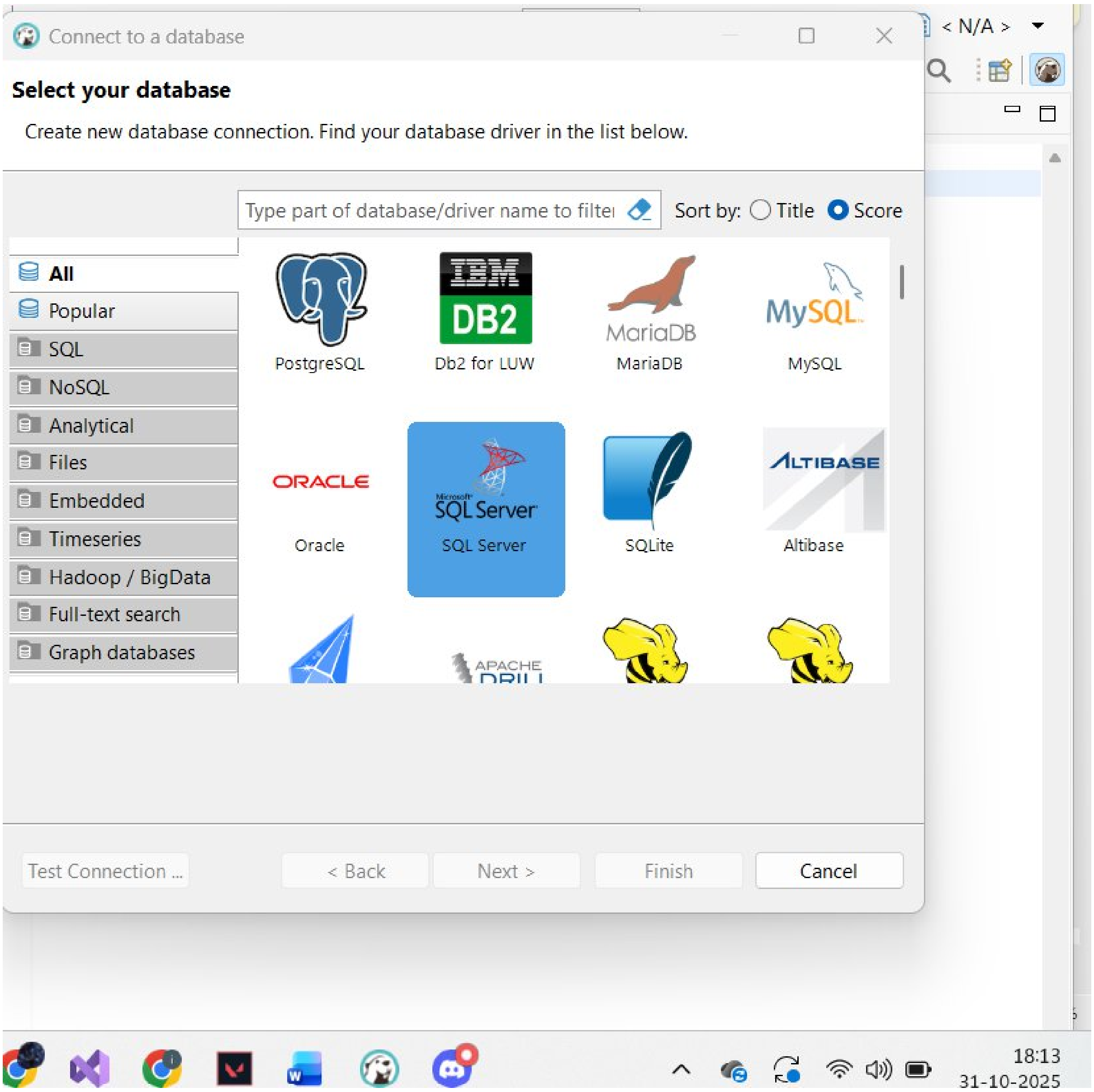
Publicly accessible
Yes

Certificate authority Info
rds-ca-rsa2048-g1

Certificate authority date
May 21, 2061, 19:04 (UTC-05:00)

DB instance certificate expiration date
October 31, 2026, 12:50 (UTC-05:00)

Connected to the RDS instance through **DBeaver** using the database endpoint.



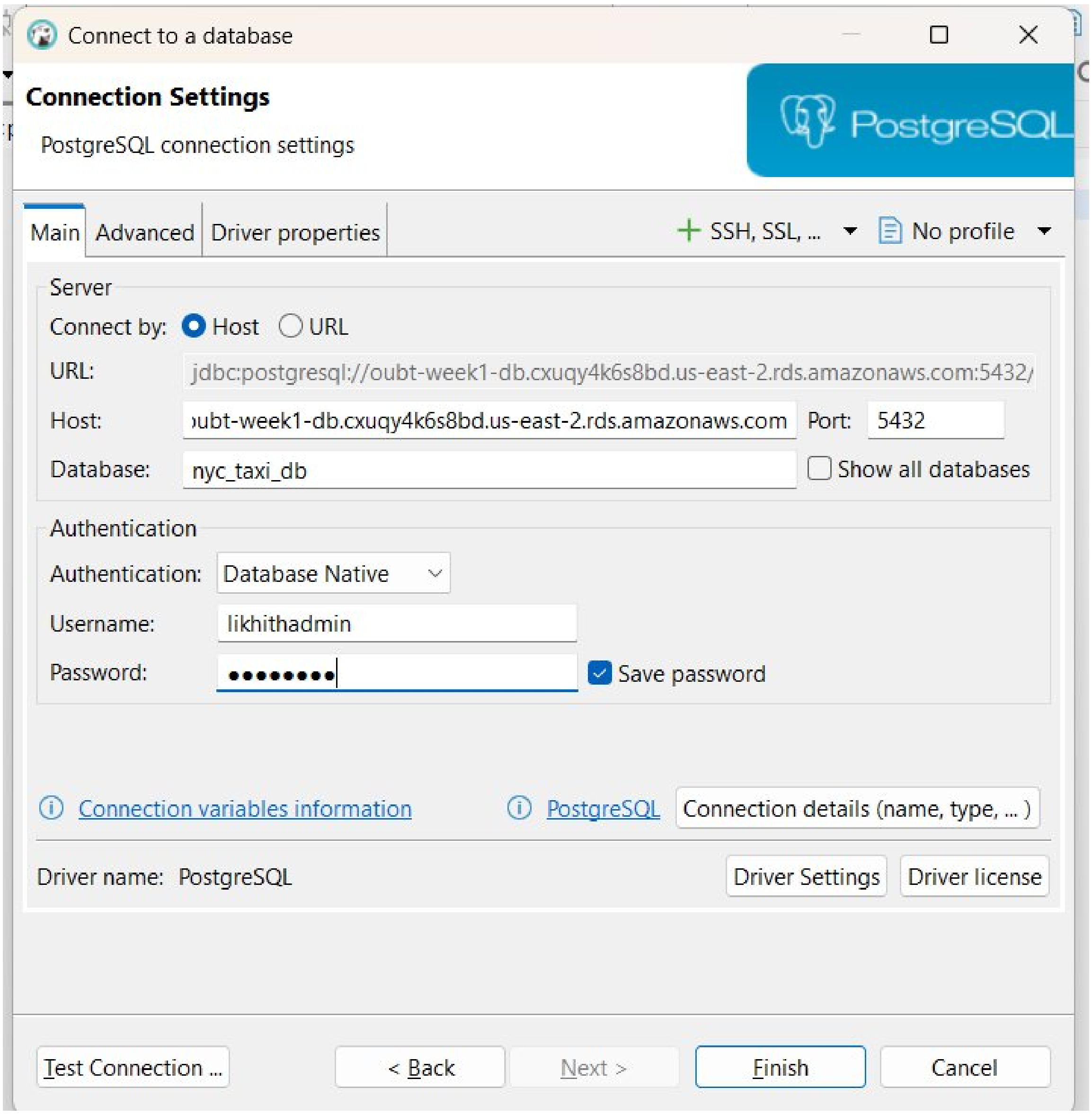
Connect to a database

Select your database

Create new database connection. Find your database driver in the list below.

Type part of database/driver name to filter

Sort by: ○ Title ● Score



Connect to a database

Connection Settings

PostgreSQL connection settings

Connect by: ● Host ○ URL

URL: jdbc:postgresql://oubt-week1-db.cxuqy4k6s8bd.us-east-2.rds.amazonaws.com:5432/

Host: ›ubt-week1-db.cxuqy4k6s8bd.us-east-2.rds.amazonaws.com Port: 5432

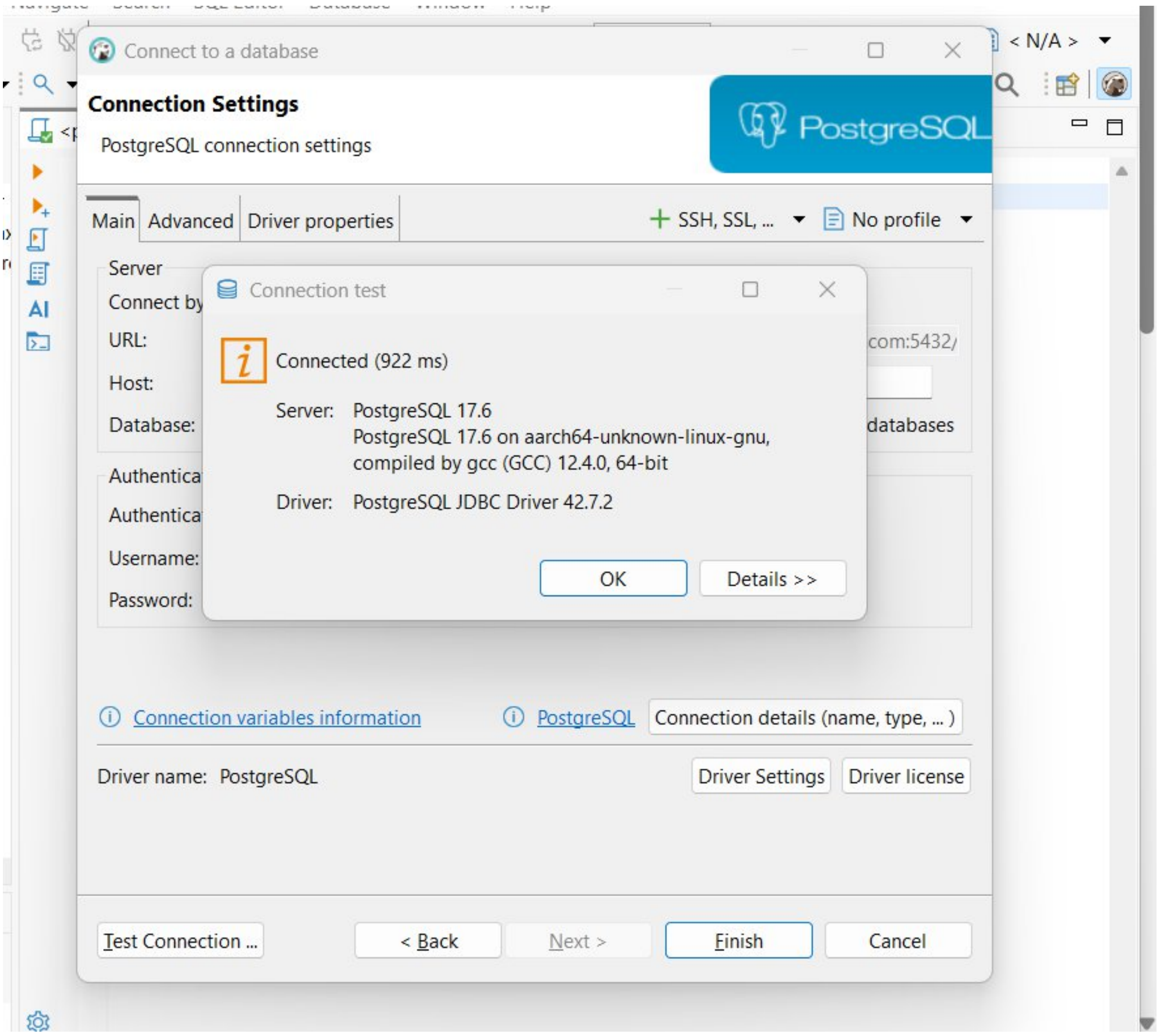
Database: nyc_taxi_db ☐ Show all databases

Authentication: Database Native

Username: likhithadmin

Password: •••••••• ☑ Save password

Driver name: PostgreSQL



Connection test

Connected (922 ms)

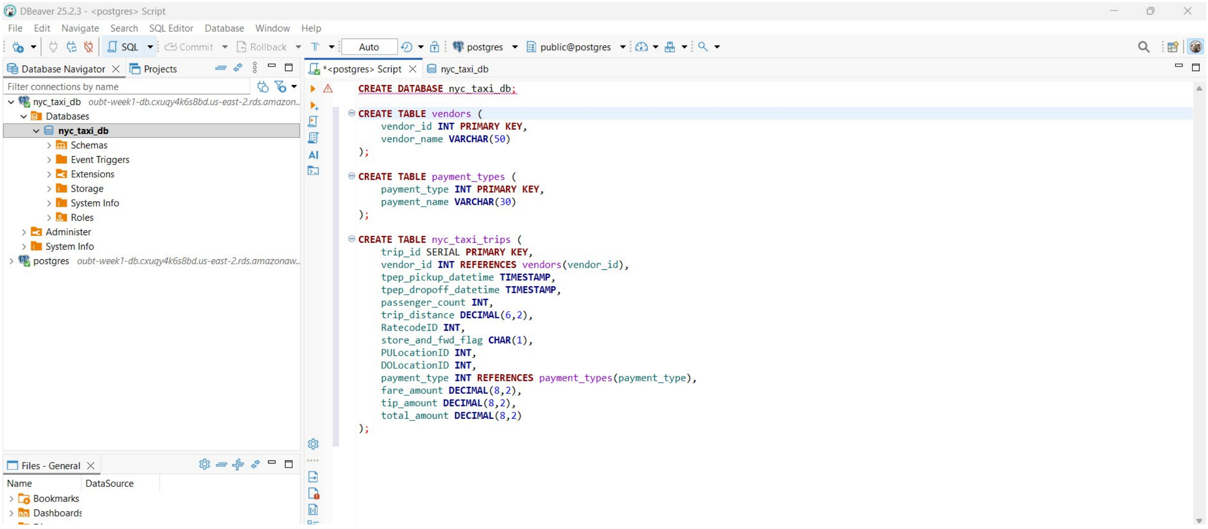
Server: PostgreSQL 17.6
PostgreSQL 17.6 on aarch64-unknown-linux-gnu,
compiled by gcc (GCC) 12.4.0, 64-bit

Driver: PostgreSQL JDBC Driver 42.7.2

OK　　Details >>

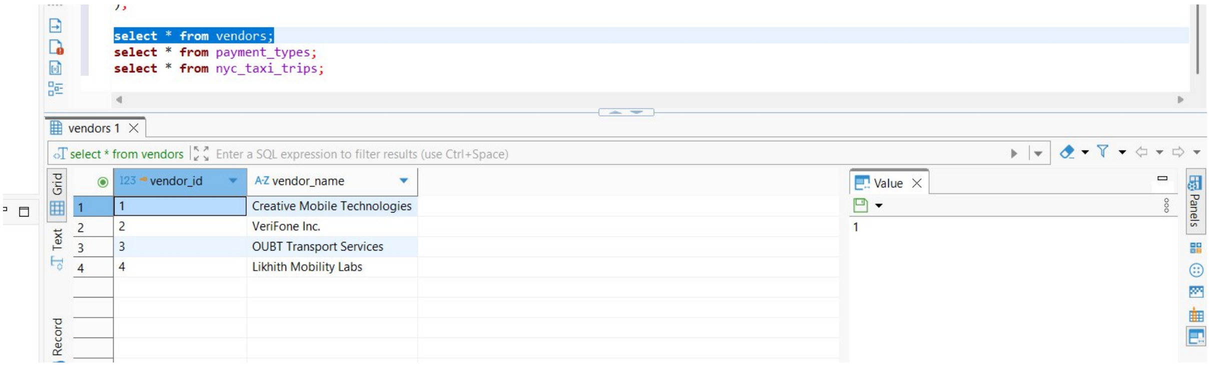
Driver name: PostgreSQL

Created a new database nyc_taxi_db.



```sql
CREATE DATABASE nyc_taxi_db;

CREATE TABLE vendors (
    vendor_id INT PRIMARY KEY,
    vendor_name VARCHAR(50)
);

CREATE TABLE payment_types (
    payment_type INT PRIMARY KEY,
    payment_name VARCHAR(30)
);

CREATE TABLE nyc_taxi_trips (
    trip_id SERIAL PRIMARY KEY,
    vendor_id INT REFERENCES vendors(vendor_id),
    tpep_pickup_datetime TIMESTAMP,
    tpep_dropoff_datetime TIMESTAMP,
    passenger_count INT,
    trip_distance DECIMAL(6,2),
    RatecodeID INT,
    store_and_fwd_flag CHAR(1),
    PULocationID INT,
    DOLocationID INT,
    payment_type INT REFERENCES payment_types(payment_type),
    fare_amount DECIMAL(8,2),
    tip_amount DECIMAL(8,2),
    total_amount DECIMAL(8,2)
);
```

Designed schema with three tables — vendors, payment_types, and nyc_taxi_trips — following a **Star Schema** model.

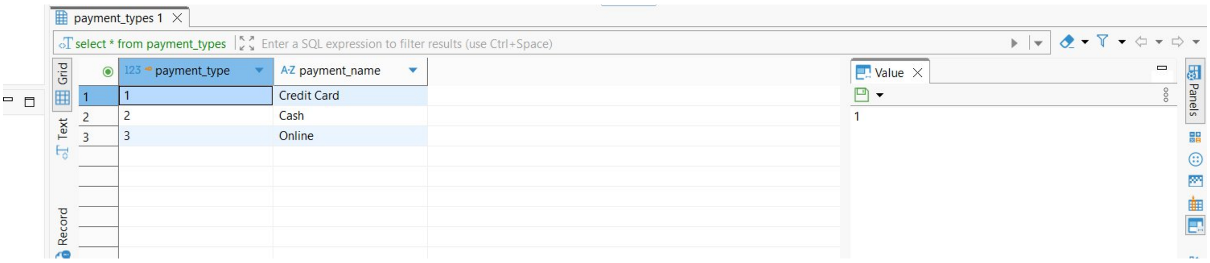


```
select * from vendors;
select * from payment_types;
select * from nyc_taxi_trips;
```

vendors 1

select * from vendors

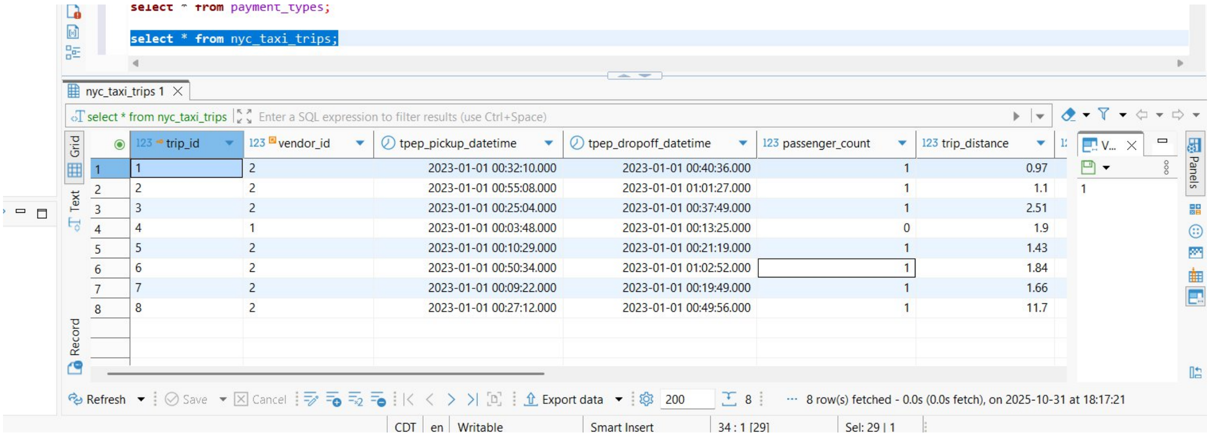
	vendor_id	vendor_name
1	1	Creative Mobile Technologies
2	2	VeriFone Inc.
3	3	OUBT Transport Services
4	4	Likhith Mobility Labs



payment_types 1

select * from payment_types

	payment_type	payment_name
1	1	Credit Card
2	2	Cash
3	3	Online



```
select * from payment_types;

select * from nyc_taxi_trips;
```

nyc_taxi_trips 1

select * from nyc_taxi_trips

	trip_id	vendor_id	tpep_pickup_datetime	tpep_dropoff_datetime	passenger_count	trip_distance
1	1	2	2023-01-01 00:32:10.000	2023-01-01 00:40:36.000	1	0.97
2	2	2	2023-01-01 00:55:08.000	2023-01-01 01:01:27.000	1	1.1
3	3	2	2023-01-01 00:25:04.000	2023-01-01 00:37:49.000	1	2.51
4	4	1	2023-01-01 00:03:48.000	2023-01-01 00:13:25.000	0	1.9
5	5	2	2023-01-01 00:10:29.000	2023-01-01 00:21:19.000	1	1.43
6	6	2	2023-01-01 00:50:34.000	2023-01-01 01:02:52.000	1	1.84
7	7	2	2023-01-01 00:09:22.000	2023-01-01 00:19:49.000	1	1.66
8	8	2	2023-01-01 00:27:12.000	2023-01-01 00:49:56.000	1	11.7

8 row(s) fetched - 0.0s (0.0s fetch), on 2025-10-31 at 18:17:21

Inserted sample trip data and reference data for vendors and payment types.

Executed multiple **SQL queries** covering JOIN, GROUP BY, aggregations, filtering, window functions, and CTEs to analyze trip records.

```
SELECT v.vendor_name,
COUNT(*) AS total_trips,
SUM(fare_amount) AS total_fare
    FROM nyc_taxi_trips t
JOIN vendors v ON t.vendor_id = v.vendor_id
GROUP BY v.vendor_name;
```

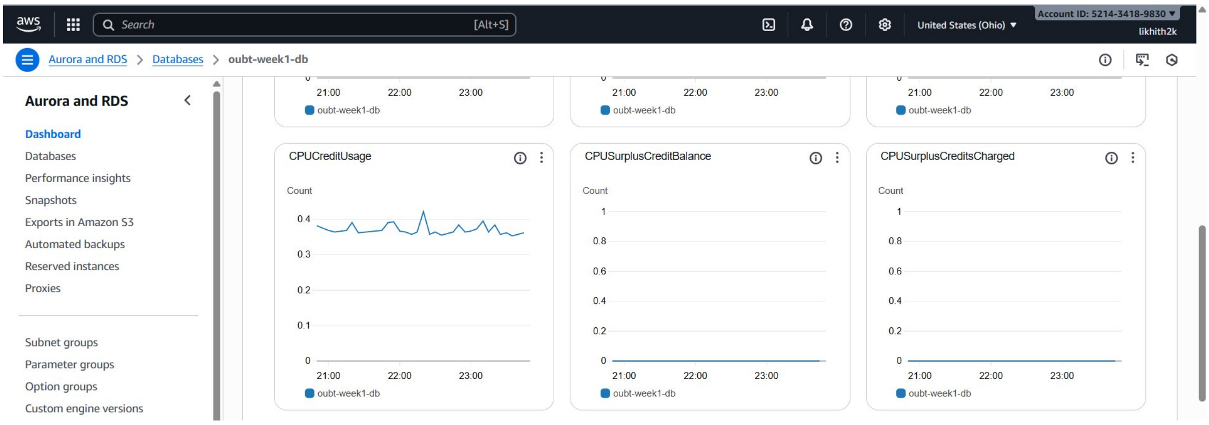
```sql
SELECT p.payment_name,
    ROUND(AVG(fare_amount),2) AS avg_fare
FROM nyc_taxi_trips t
JOIN payment_types p ON t.payment_type = p.payment_type
GROUP BY p.payment_name;


SELECT trip_id, vendor_id, fare_amount
FROM nyc_taxi_trips
ORDER BY fare_amount DESC
LIMIT 3;

SELECT trip_id, vendor_id, fare_amount,
RANK() OVER(ORDER BY fare_amount DESC) AS fare_rank
FROM nyc_taxi_trips;
```

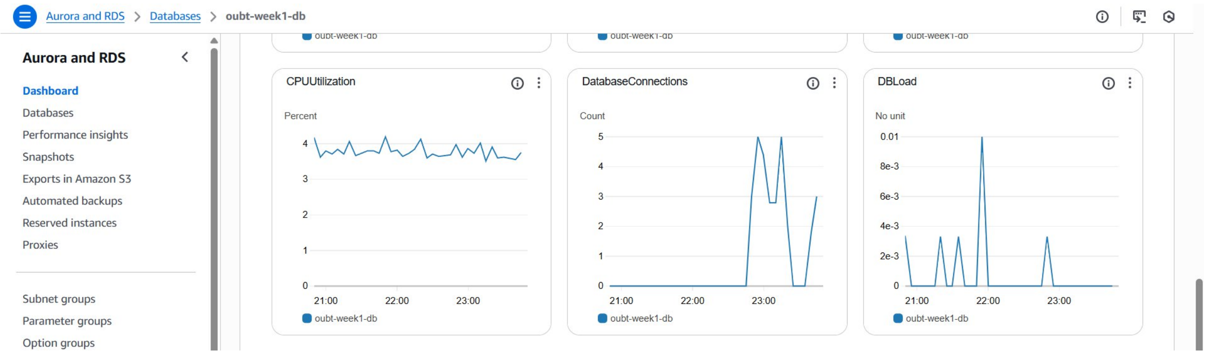
Verified query results and captured screenshots of successful execution in DBeave

CloudWatch & RDS Monitoring Test

To verify my AWS RDS instance activity, I ran multiple SQL queries from DBeaver and monitored performance in the RDS dashboard. The CloudWatch graphs showed clear spikes in **CPU utilization**, **DB load**, and **active connections** while queries were executing. This confirmed that my PostgreSQL instance was processing live workloads.

This exercise helped me understand how CloudWatch metrics can be used to monitor database health and query performance in real time.





Key Learnings:

Gained hands-on experience in **AWS RDS setup**, database connectivity, schema creation, and SQL querying through DBeaver.

Configured **IAM permissions and security groups** to enable secure database access.

Understood how **Star Schema modeling** supports analytical queries and relational design.

Practiced organizing and structuring data for better readability, consistency, and performance.