Real-World Use Case of Amazon Redshift Data API for SQL Querying Without Persistent Connections

Problem Statement:

A data engineering team wants to allow multiple applications to run SQL queries on their Amazon Redshift cluster **without managing long-lived database connections or provisioning client drivers**. This capability is crucial to support event-driven applications, serverless workflows, and scheduled automation **without the overhead of connection pooling**.

Objective:

To simulate and experience a real-world solution using the Amazon Redshift **Data API**, enabling SQL query execution directly via AWS CLI or Lambda without requiring JDBC/ODBC connections.

Technologies Used:

- Amazon Redshift Serverless
- AWS Secrets Manager
- Amazon Redshift Data API
- AWS CLI

Solution Steps Implemented:

◇ Step 1: Provision Redshift Serverless

- Created a Redshift Serverless **workgroup**: `my-redshift-new-workgroup`
- Created a **namespace** with default database: `dev`
- Configured the admin username and password

◇ Step 2: Initial Authentication Issues Identified

- Attempted login using the default `admin` user and **faced error**: > `FATAL: password authentication failed for user "admin"`
- Root cause: Redshift Serverless auto-manages admin credentials, which mismatch with custom Secrets Manager entries.

◇ Step 3: Created Custom User for Authentication

- Logged into Query Editor v2 using the default session
- Created a custom user:

```
CREATE USER datauser PASSWORD 'DataUserPassword123!';
GRANT USAGE ON SCHEMA public TO datauser;
GRANT SELECT ON ALL TABLES IN SCHEMA public TO datauser;
```

◇ Step 4: Created Table and Sample Data

```
CREATE TABLE sales (
    id INT,
    product VARCHAR(50),
    quantity INT,
    sale_date DATE
);

INSERT INTO sales VALUES
(1, 'Keyboard', 10, '2025-06-15'),
(2, 'Mouse', 5, '2025-06-16'),
(3, 'Monitor', 2, '2025-06-17');
```

◇ Step 5: Stored Credentials in Secrets Manager

- Created a new secret named: `redshift-data-api-datauser`
- Credentials stored:
    - `username`: `datauser`
    - `password`: `DataUserPassword123!`
- Captured full ARN:

arn:aws:secretsmanager:ap-south-1:535002864770:secret:redshift-data-api-datauser-AqzP8D

🔹 Step 6: Executed SQL Query via AWS CLI

```bash
aws redshift-data execute-statement \
  --workgroup-name my-redshift-new-workgroup \
  --database dev \
  --sql "SELECT * FROM sales;" \
  --secret-arn arn:aws:secretsmanager:ap-south-
1:535002864770:secret:redshift-data-api-datauser-AqzP8D \
  --region ap-south-1
```

- Captured `query-id` from the CLI response.

◇ Step 7: Retrieved Results via CLI

```
aws redshift-data get-statement-result --id <query-id> --region ap-south-1
```

- ✅ Got expected structured output in JSON format from the `sales` table.

☑ Final Outcome:

- Simulated a **real-world serverless SQL querying** use case.
- Avoided JDBC/ODBC drivers using **Redshift Data API**.
- Managed secure credentials using **AWS Secrets Manager**.
- Prepared architecture for **event-based, scheduled, or Lambda-powered** Redshift operations.

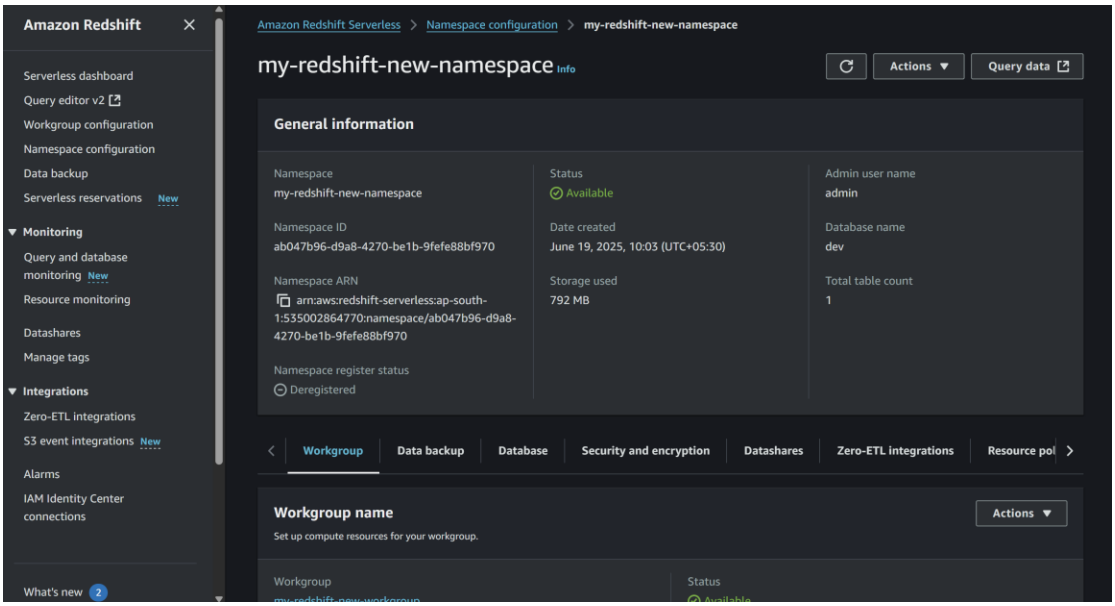
🚀 Next Steps (Optional Enhancements):

- Automate Data API + export results to CSV via Lambda
- Attach IAM permission boundaries for more security
- Build real-time dashboard with Streamlit or Flask using the Data API

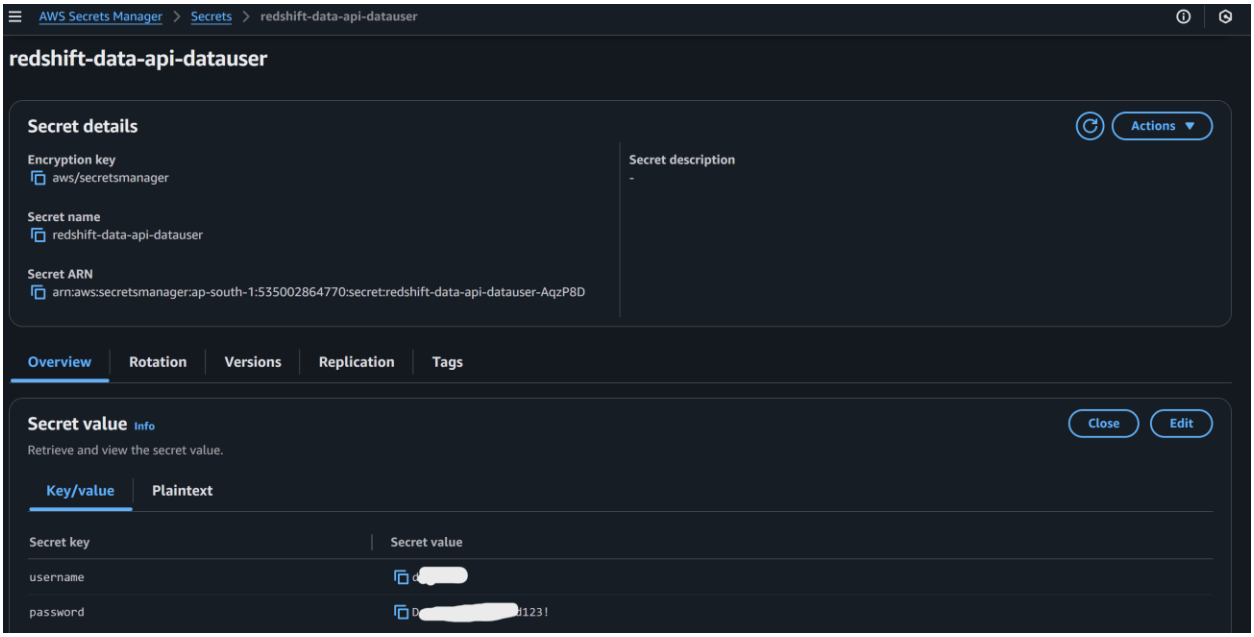
📓 Workspace Output (Paste Your Results Below):

📎 Paste your screenshots, execution outputs, or CLI command results below:

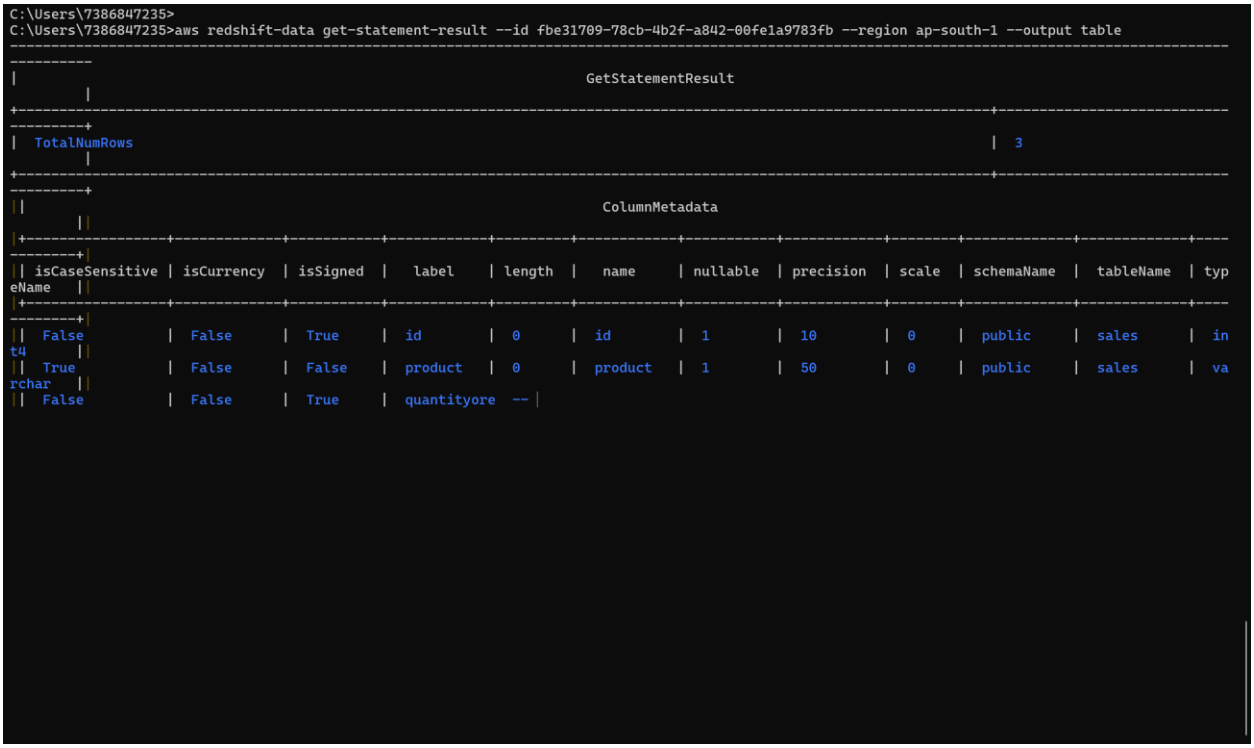
`[Redshift Workgroup Console Screenshot]`



[Secrets Manager ARN and structure]



[CLI output of execute-statement and get-statement-result]



📌 Summary

This project shows how to build **lightweight, scalable, and secure querying** systems using:

- Amazon Redshift Serverless
- AWS CLI
- Redshift Data API
- Secrets Manager

You've now implemented a **connectionless data querying model**, ready for real-world automation!

Prepared by: A. Siva Sandeep
Date: June 19, 2025

📎 Tags

`#RedshiftDataAPI` `#AWSRedshiftServerless` `#SecretsManager` `#ServerlessSQL` `#CloudProject` `#EventDrivenSQL`