**Lab : Streaming to Snowflake with Python**

**Goal:** Build a Python streaming application that consumes data from a Redpanda Cloud topic and ingests it into a table in Snowflake in near real-time using key-pair authentication.

Purpose of the Lab

This lab is designed for developers and covers the critical use case of streaming data into a cloud data warehouse programmatically. You will write a Python script that acts as a long-running service, using standard client libraries to connect securely to both Redpanda and Snowflake. This lab provides practical experience in handling secure, key-pair authentication for Snowflake and building custom streaming applications that are central to modern data pipelines.

# **Prerequisites**

* A Redpanda Cloud account with a running cluster and an rpk profile (e.g., rpk-cloud).
* A Snowflake account with ACCOUNTADMIN privileges to set up the necessary objects.
* Python 3 installed on your local machine.
* OpenSSL installed on your local machine to generate a key pair.

# **Project Layout**

|  |
| --- |
| rp-snowflake-python-lab/ ├── sales\_events.jsonl ├── snowflake\_sink.py ├── manual\_producer.py ├── .env └── snowflake\_key.p8 |

# **Part 1: Setting up the Project and Snowflake**

## Step 1: Prepare the Project Directory and Generate Keys

First, we'll create our project directory and generate the necessary authentication keys directly inside it.

1. **Create and enter the project directory:**

|  |
| --- |
| mkdir rp-snowflake-python-lab cd rp-snowflake-python-lab |

1. **Generate a private key:**

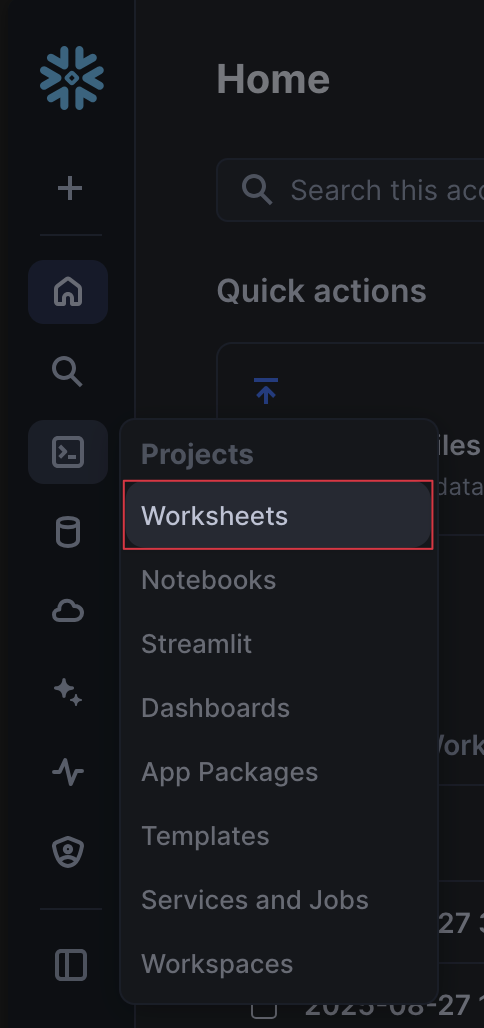
|  |
| --- |
| openssl genrsa -out snowflake\_key.p8 2048 |

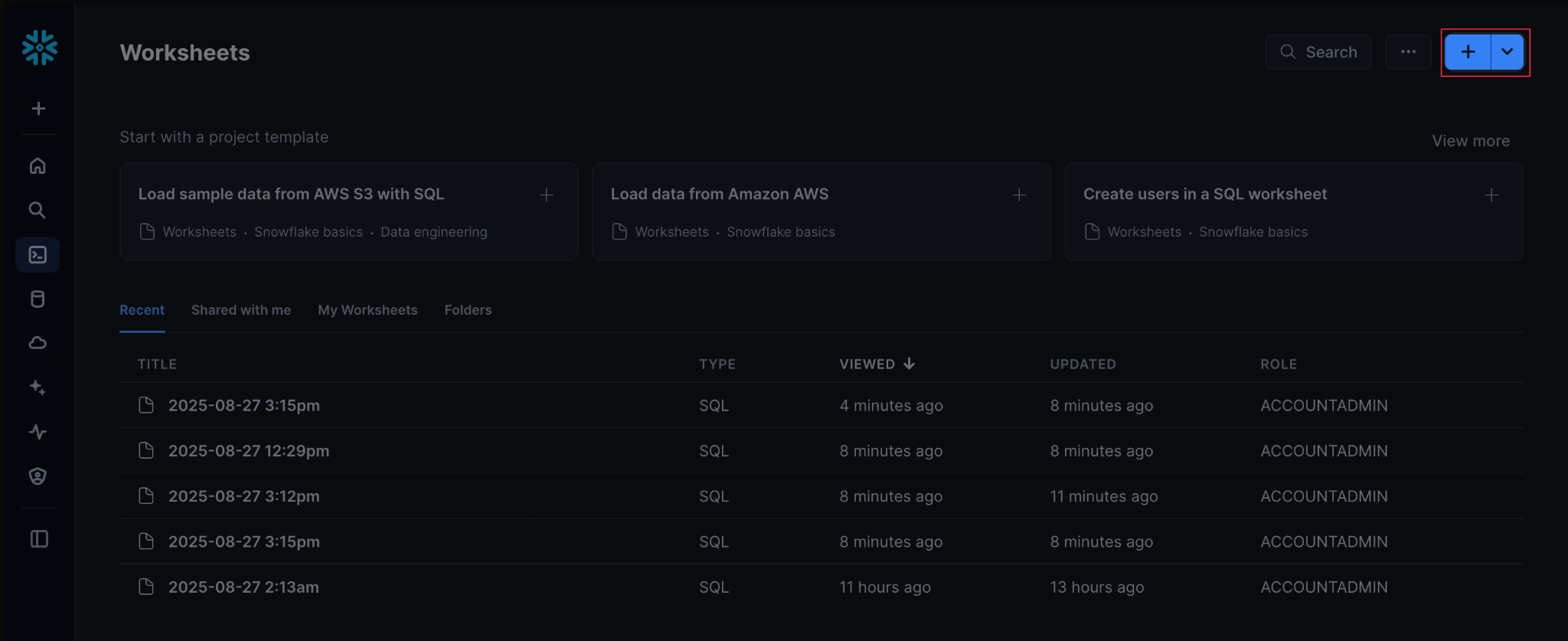
1. **Generate the public key from the private key:**

|  |
| --- |
| openssl rsa -in snowflake\_key.p8 -pubout -out snowflake\_key.pub |

## Step 2: Create Snowflake Objects

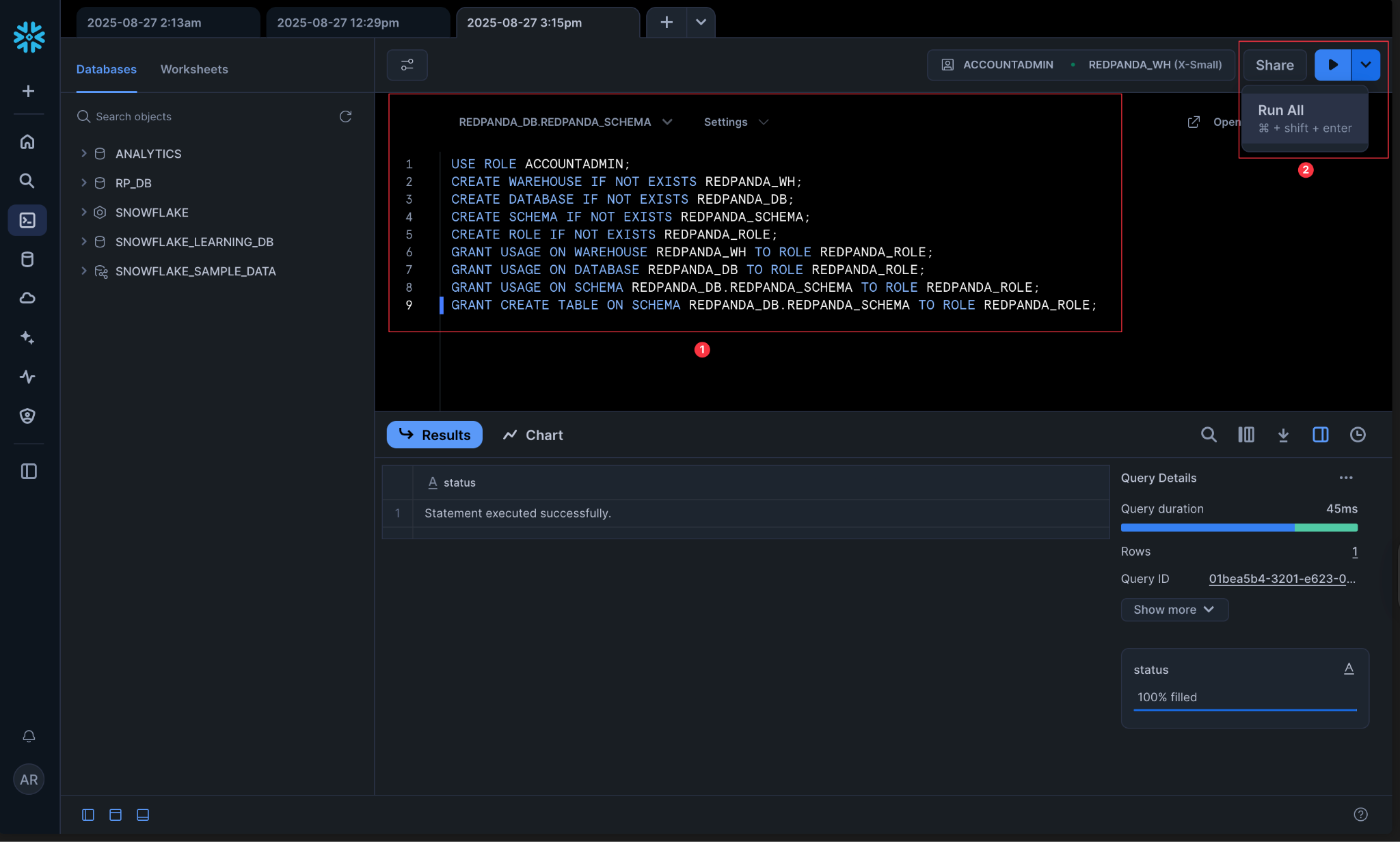
Log in to your Snowflake account and open a new worksheet. Run the following SQL commands one by one.





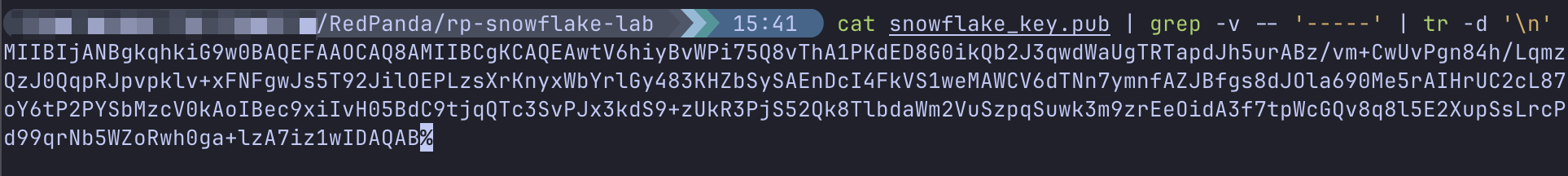
1. **Set up roles and warehouses:**

|  |
| --- |
| USE ROLE ACCOUNTADMIN; CREATE WAREHOUSE IF NOT EXISTS REDPANDA\_WH; CREATE DATABASE IF NOT EXISTS REDPANDA\_DB; CREATE SCHEMA IF NOT EXISTS REDPANDA\_SCHEMA; CREATE ROLE IF NOT EXISTS REDPANDA\_ROLE; GRANT USAGE ON WAREHOUSE REDPANDA\_WH TO ROLE REDPANDA\_ROLE; GRANT USAGE ON DATABASE REDPANDA\_DB TO ROLE REDPANDA\_ROLE; GRANT USAGE ON SCHEMA REDPANDA\_DB.REDPANDA\_SCHEMA TO ROLE REDPANDA\_ROLE; GRANT CREATE TABLE ON SCHEMA REDPANDA\_DB.REDPANDA\_SCHEMA TO ROLE REDPANDA\_ROLE; |



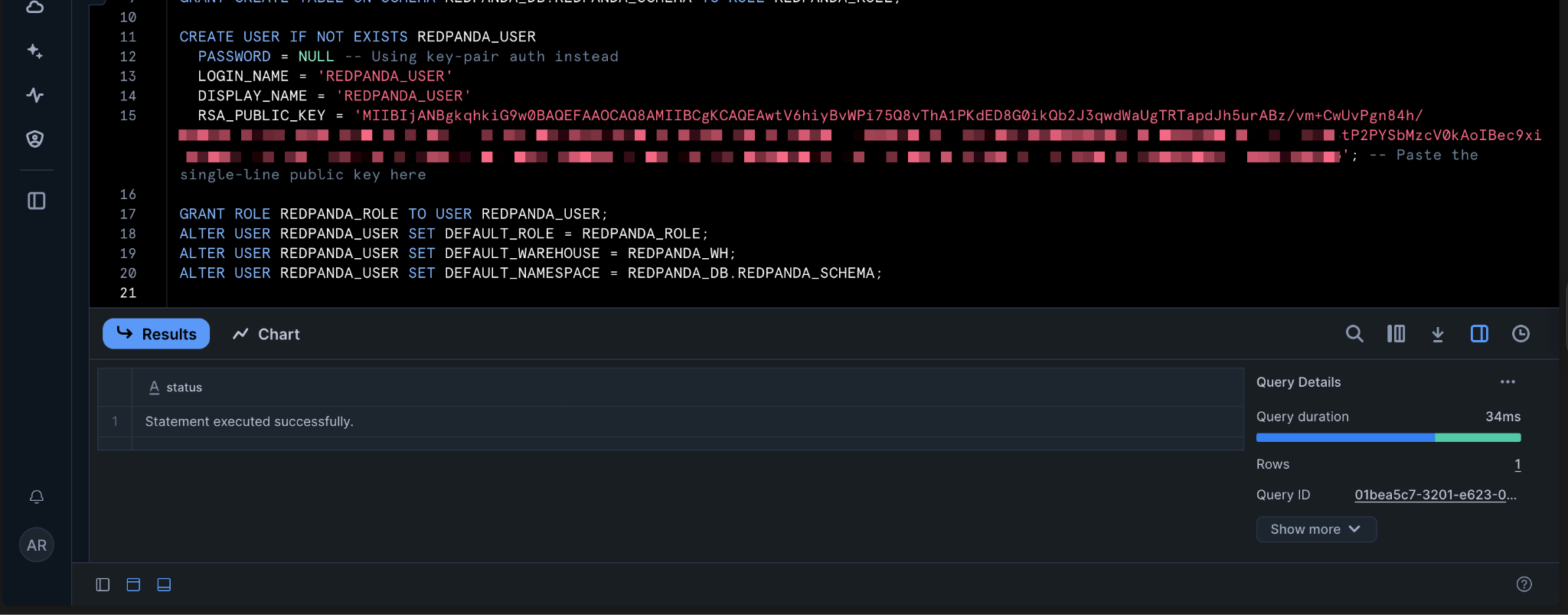
1. **Create a user and assign the public key:**  
   First, go to your terminal and run the command below to display your clean public key. Copy the entire single-line output.

|  |
| --- |
| cat snowflake\_key.pub | grep -v -- '-----' | tr -d '\n' |



*(Note: Sometimes your shell may add a % character at the very end of the output string. This is not part of the key and should be removed before pasting.)*  
Now, paste the single-line public key string you copied into the RSA\_PUBLIC\_KEY field in the SQL command below and run it in your Snowflake worksheet.

|  |
| --- |
| CREATE USER IF NOT EXISTS REDPANDA\_USER  PASSWORD = NULL  LOGIN\_NAME = 'REDPANDA\_USER'  DISPLAY\_NAME = 'REDPANDA\_USER'  RSA\_PUBLIC\_KEY = '<YOUR\_PUBLIC\_KEY\_STRING>';  GRANT ROLE REDPANDA\_ROLE TO USER REDPANDA\_USER; ALTER USER REDPANDA\_USER SET DEFAULT\_ROLE = REDPANDA\_ROLE; ALTER USER REDPANDA\_USER SET DEFAULT\_WAREHOUSE = REDPANDA\_WH; ALTER USER REDPANDA\_USER SET DEFAULT\_NAMESPACE = REDPANDA\_DB.REDPANDA\_SCHEMA; |

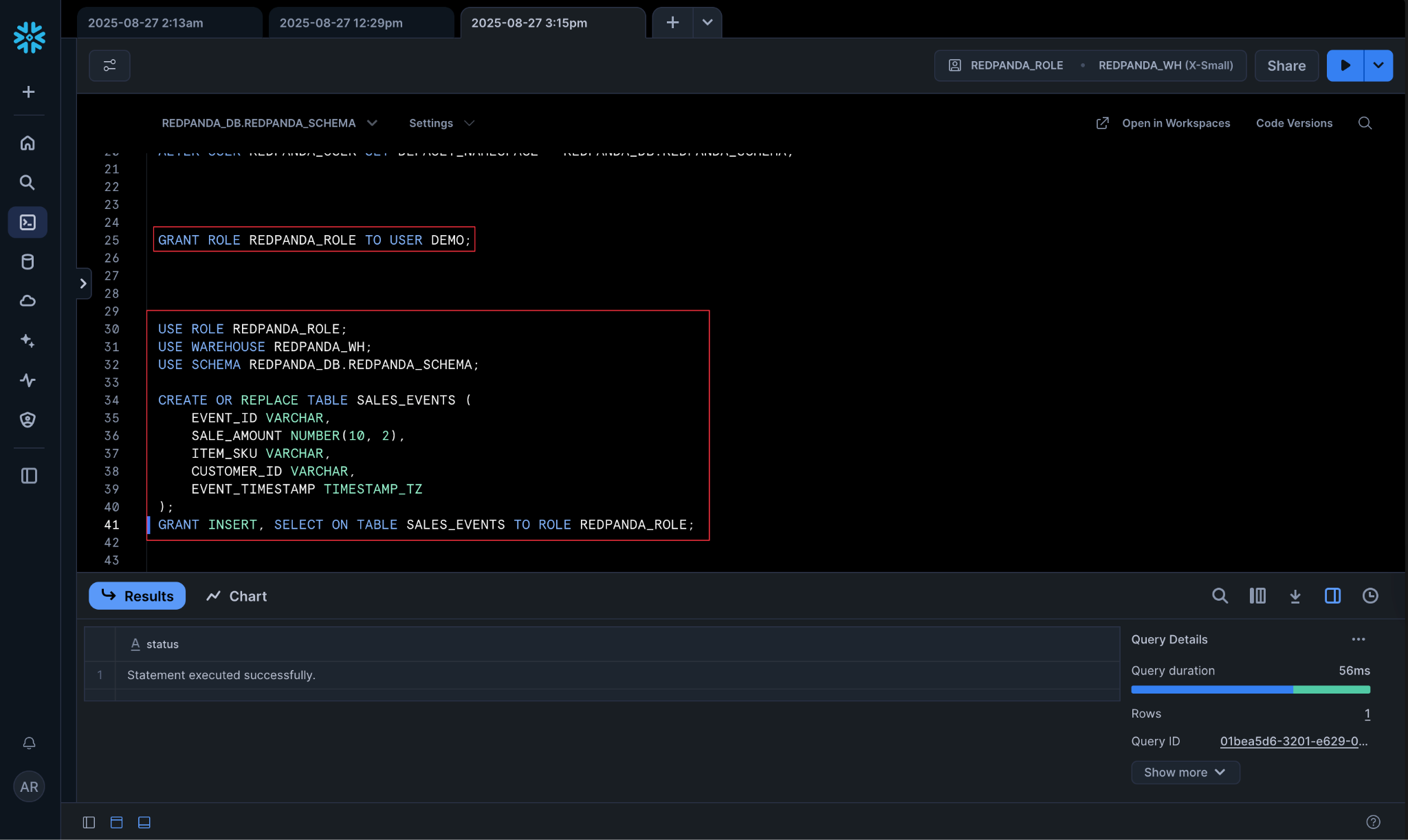


1. **Grant the new role to your own user:** Replace <YOUR\_CURRENT\_USERNAME> with the username you used to log in to the Snowflake UI.

|  |
| --- |
| GRANT ROLE REDPANDA\_ROLE TO USER <YOUR\_CURRENT\_USERNAME>; |

1. **Create the target table:**

|  |
| --- |
| USE ROLE REDPANDA\_ROLE; USE WAREHOUSE REDPANDA\_WH; USE SCHEMA REDPANDA\_DB.REDPANDA\_SCHEMA;  CREATE OR REPLACE TABLE SALES\_EVENTS (  EVENT\_ID VARCHAR,  SALE\_AMOUNT NUMBER(10, 2),  ITEM\_SKU VARCHAR,  CUSTOMER\_ID VARCHAR,  EVENT\_TIMESTAMP TIMESTAMP\_TZ ); GRANT INSERT, SELECT ON TABLE SALES\_EVENTS TO ROLE REDPANDA\_ROLE; |



## 

# **Part 2: Setting up Redpanda and Python**

## Step 3: Get Redpanda Cloud Credentials

Create a new user (e.g., snowflake-py-user) in the **Security -> Users** tab of the Redpanda Cloud UI and grant it **Allow All** permissions in the **ACLs** tab. Save the **Username**, **Password**, and your cluster's **Broker Address**.

## **Step 4: Prepare Redpanda and the Python Environment**

1. **Create a sample data file named sales\_events.jsonl :**

**sales\_events.jsonl**

|  |
| --- |
| {"event\_id": "a1", "amount": 19.99, "sku": "ABC-123", "customer": {"id": "cust-101"}, "timestamp": "2025-08-27T14:30:00Z"} {"event\_id": "a2", "amount": 250.00, "sku": "XYZ-789", "customer": {"id": "cust-202"}, "timestamp": "2025-08-27T14:31:00Z"} |

1. **Create the topic on Redpanda Cloud:**

|  |
| --- |
| rpk topic create sales-events-py --profile rpk-cloud |

1. **Set up the Python virtual environment:**

|  |
| --- |
| python3 -m venv venv source venV/bin/activate  pip install kafka-python python-dotenv certifi python-snappy snowflake-connector-python cryptography |

1. **Gather your credentials for the .env file:**

* Redpanda Credentials: You will use the Broker URL, Username, and Password for the snowflake-user you created in Step 3.
* Snowflake Account Identifier: You can find this in the URL you use to log in to the Snowflake UI (e.g., abc12345.us-east-1 from https://abc12345.us-east-1.snowflakecomputing.com).

1. **Create the Environment File (.env):**

|  |
| --- |
| # .env REDPANDA\_BROKERS="<YOUR\_BROKERS\_URL>" REDPANDA\_USER="<YOUR\_REDPANDA\_USERNAME>" REDPANDA\_PASS="<YOUR\_REDPANDA\_PASSWORD>"  # Snowflake Credentials SNOWFLAKE\_ACCOUNT="<YOUR\_ACCOUNT\_IDENTIFIER>" SNOWFLAKE\_USER="REDPANDA\_USER" SNOWFLAKE\_PRIVATE\_KEY\_PATH="snowflake\_key.p8" SNOWFLAKE\_DATABASE="REDPANDA\_DB" SNOWFLAKE\_SCHEMA="REDPANDA\_SCHEMA" |

Populate the file with your credentials.

1. **Produce the data to the sales-events-py topic:**

|  |
| --- |
| rpk topic produce sales-events-py --profile rpk-cloud < sales\_events.jsonl |

# **Part 3: Building and Running the Python Pipeline**

## **Step 5: Create the Snowflake Sink Script**

Create a file named snowflake\_sink.py. This script will connect to both services and stream data.

**snowflake\_sink.py**

|  |
| --- |
| # snowflake\_sink.py import json, os, certifi from kafka import KafkaConsumer from dotenv import load\_dotenv import snowflake.connector from cryptography.hazmat.primitives import serialization  # Load environment variables from .env file load\_dotenv()  # --- Configuration --- BROKER\_URL = os.getenv("REDPANDA\_BROKERS") USERNAME = os.getenv("REDPANDA\_USER") PASSWORD = os.getenv("REDPANDA\_PASS") SOURCE\_TOPIC = "sales-events-py"  SNOWFLAKE\_ACCOUNT = os.getenv("SNOWFLAKE\_ACCOUNT") SNOWFLAKE\_USER = os.getenv("SNOWFLAKE\_USER") SNOWFLAKE\_PRIVATE\_KEY\_PATH = os.getenv("SNOWFLAKE\_PRIVATE\_KEY\_PATH") SNOWFLAKE\_DATABASE = os.getenv("SNOWFLAKE\_DATABASE") SNOWFLAKE\_SCHEMA = os.getenv("SNOWFLAKE\_SCHEMA") # --- End Configuration ---  # Load and deserialize the private key with open(SNOWFLAKE\_PRIVATE\_KEY\_PATH, "rb") as key:  p\_key = serialization.load\_pem\_private\_key(  key.read(),  password=None, # No password on the key file  ) pkb = p\_key.private\_bytes(  encoding=serialization.Encoding.DER,  format=serialization.PrivateFormat.PKCS8,  encryption\_algorithm=serialization.NoEncryption() )  print("Starting Python Snowflake sink...") print("Press Ctrl+C to stop.")  try:  # Connect to Snowflake  snowflake\_conn = snowflake.connector.connect(  user=SNOWFLAKE\_USER,  account=SNOWFLAKE\_ACCOUNT,  private\_key=pkb,  database=SNOWFLAKE\_DATABASE,  schema=SNOWFLAKE\_SCHEMA  )  print("Successfully connected to Snowflake.")    # Connect to Redpanda  consumer = KafkaConsumer(  SOURCE\_TOPIC,  bootstrap\_servers=BROKER\_URL, security\_protocol="SASL\_SSL",  sasl\_mechanism="SCRAM-SHA-256", sasl\_plain\_username=USERNAME, sasl\_plain\_password=PASSWORD,  group\_id="snowflake-python-sink-group", auto\_offset\_reset="earliest",  ssl\_cafile=certifi.where(), api\_version=(2, 0, 2),  value\_deserializer=lambda v: json.loads(v.decode('utf-8'))  )   for message in consumer:  event = message.value  print(f"Received event: {event}")    # Insert data into Snowflake  snowflake\_conn.cursor().execute(  "INSERT INTO SALES\_EVENTS (EVENT\_ID, SALE\_AMOUNT, ITEM\_SKU, CUSTOMER\_ID, EVENT\_TIMESTAMP) VALUES (%s, %s, %s, %s, %s)",  (event['event\_id'], event['amount'], event['sku'], event['customer']['id'], event['timestamp'])  )  print(f"Successfully inserted event {event['event\_id']} into Snowflake.")  except KeyboardInterrupt:  print("Handler stopped by user.") except Exception as e:  print(f"An unexpected error occurred: {e}") finally:  if 'consumer' in locals(): consumer.close()  if 'snowflake\_conn' in locals(): snowflake\_conn.close()  print("Connections closed.") |

## Step 6: Run the Streaming Pipeline

Execute the script. It will process the existing messages and wait for more.

|  |
| --- |
| python snowflake\_sink.py |

**Expected output:** The script will connect to both services, process the two existing messages, and then wait for more.

## **Step 7: Test the Real-time Stream**

1. **Open a new terminal window** and navigate to the same project directory. Activate the virtual environment (source venv/bin/activate).
2. Create a small, separate producer script named manual\_producer.py to send a single message.  
   **manual\_producer.py**

|  |
| --- |
| # manual\_producer.py import json, os, certifi from kafka import KafkaProducer from dotenv import load\_dotenv  # Load environment variables from .env file load\_dotenv()  # --- Configuration --- BROKER\_URL = os.getenv("REDPANDA\_BROKERS") USERNAME = os.getenv("REDPANDA\_USER") PASSWORD = os.getenv("REDPANDA\_PASS") TOPIC\_NAME = "sales-events-py" # --- End Configuration ---  producer = KafkaProducer(  bootstrap\_servers=BROKER\_URL, security\_protocol="SASL\_SSL",  sasl\_mechanism="SCRAM-SHA-256", sasl\_plain\_username=USERNAME, sasl\_plain\_password=PASSWORD,  value\_serializer=lambda v: json.dumps(v).encode('utf-8'),  ssl\_cafile=certifi.where(), api\_version=(2, 0, 2) )  new\_event = {"event\_id": "a3", "amount": 99.50, "sku": "DEF-456", "customer": {"id": "cust-303"}, "timestamp": "2025-08-27T14:32:00Z"} print(f"Sending manual event: {new\_event}") producer.send(TOPIC\_NAME, value=new\_event) producer.flush() producer.close() print("Done.") |

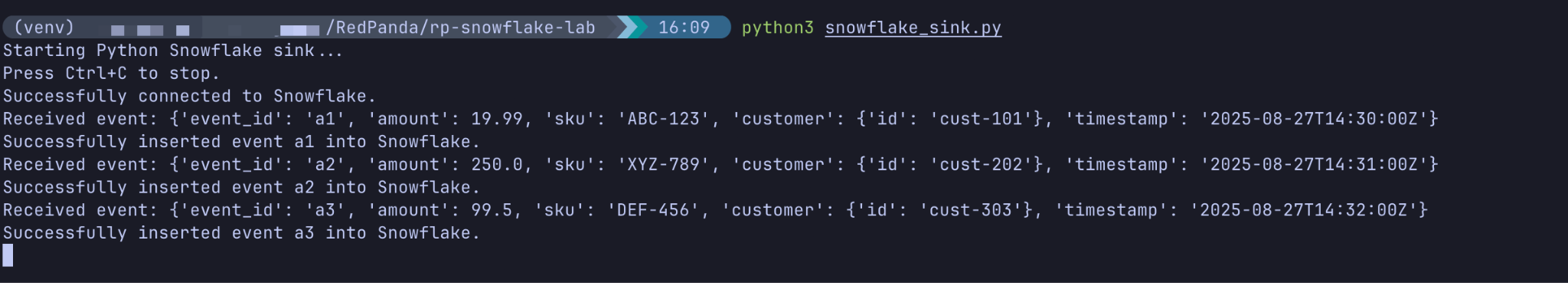
1. Run this manual producer script.

|  |
| --- |
| python manual\_producer.py |

## Step 8: Verify the Results

1. **Verify in Terminal:** Switch back to your first terminal where the snowflake\_sink.py script is running. You should see the log message confirming that the new event "a3" was received and inserted.

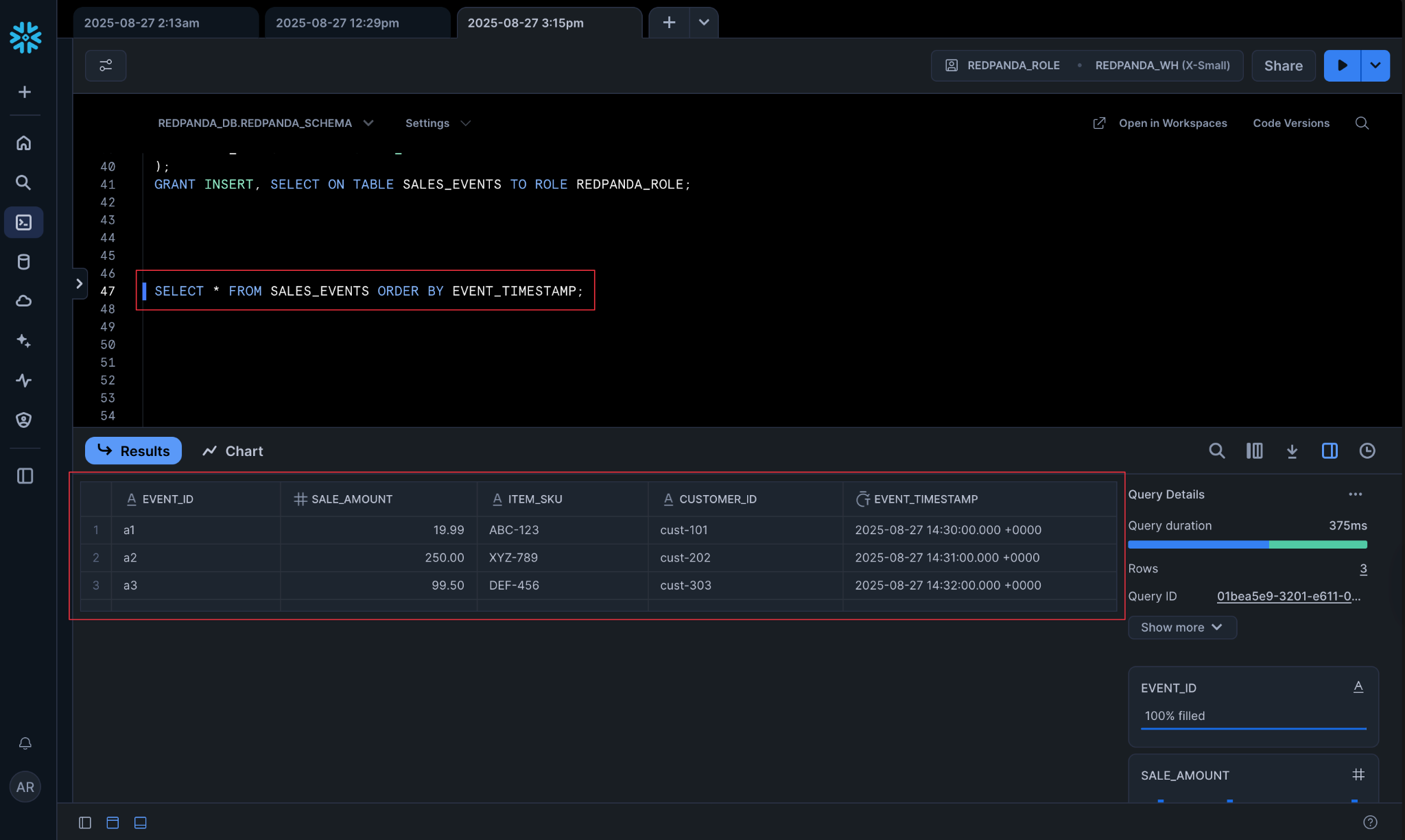




1. **Verify in Snowflake:** Go back to your Snowflake worksheet and run the SELECT query again.

|  |
| --- |
| **SELECT \* FROM SALES\_EVENTS ORDER BY EVENT\_TIMESTAMP;** |

**Expected output:** You should now see all three records from your Redpanda topic ingested into the Snowflake table.



# Cleanup

1. Stop the connector with Ctrl+C.
2. Delete the Redpanda topic:

|  |
| --- |
| rpk topic delete sales-events-py --profile rpk-cloud |

1. In Snowflake, drop the objects.

|  |
| --- |
| DROP DATABASE IF EXISTS REDPANDA\_DB;  DROP WAREHOUSE IF EXISTS REDPANDA\_WH;  DROP USER IF EXISTS REDPANDA\_USER;  DROP ROLE IF EXISTS REDPANDA\_ROLE; |