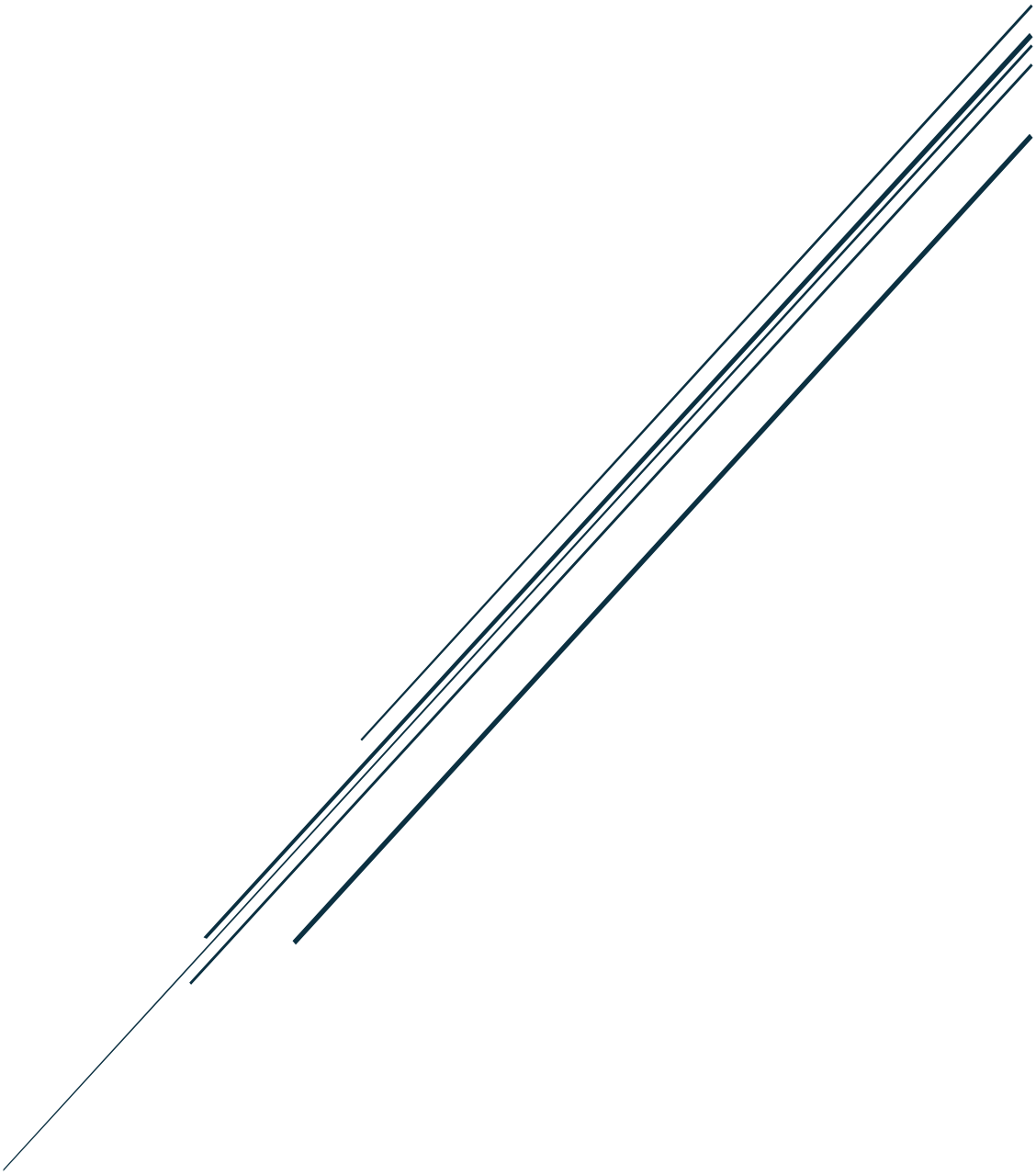
Pizza Chain Insights

Project Report

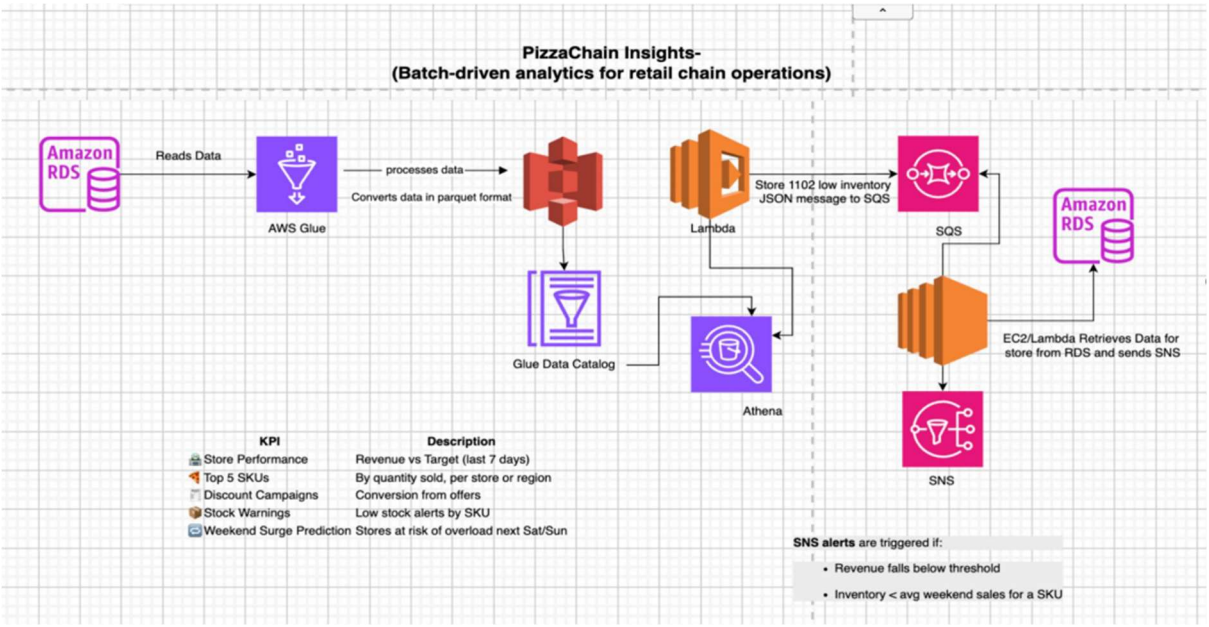


By
Aahash Kamble

Objective

To build a cloud-native batch analytics pipeline for a fictional multi-store pizza retail chain that simulates real-world retail operations. The system should ingest, transform, and analyze data from various operational sources such as orders, inventory, and discounts, and deliver actionable insights to improve business performance and decision-making.

Data Flow Architecture:

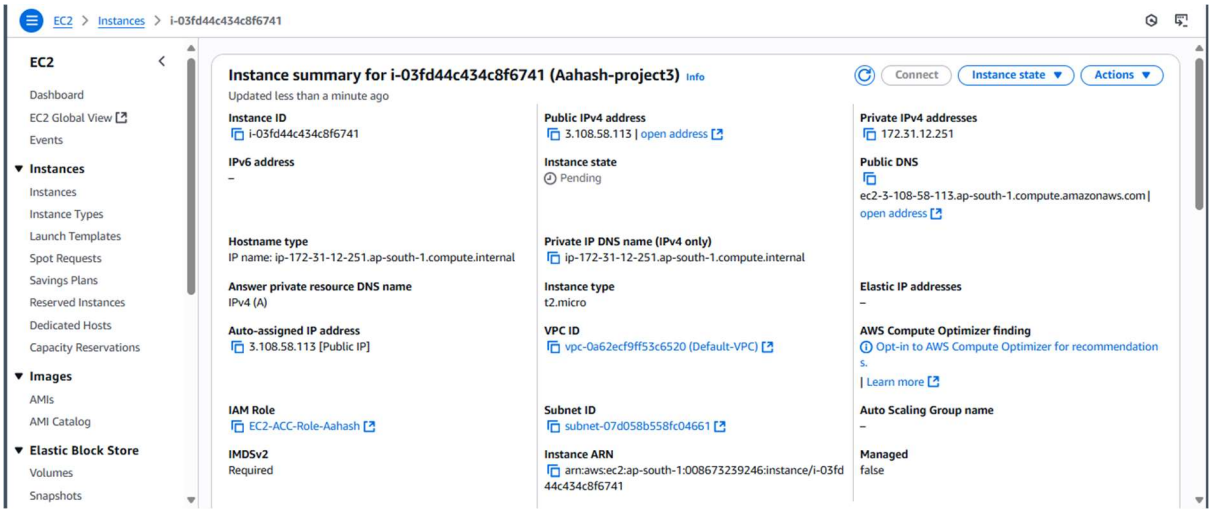


Phase 1: Data Generation & RDS Setup

- **Objective**

To simulate a pizza chain's real-time retail data using a Python script and load it into an Amazon RDS MySQL database. This forms the foundation for downstream transformation, analytics, and automation tasks.

1. Create Ec2 Instance and Setup Environment on EC2



sudo yum update -y
sudo yum install python3-pip -y
sudo dnf install -y mariadb105-server
pip3 install pandas
sudo systemctl enable mariadb
sudo systemctl start mariadb

```
Last metadata expiration check: 0:03:11 ago on Mon Aug  4 09:52:10 2025.
Dependencies resolved.
================================================================================
 Package                 Architecture        Version                 Repository      Size
================================================================================
Installing:
 python3-pip             noarch              21.3.1-2.amzn2023.0.13  amazonlinux     1.8 M
Installing weak dependencies:
 libxcrypt-compat        x86_64              4.4.33-7.amzn2023       amazonlinux      92 k

Transaction Summary
================================================================================
Install  2 Packages

Total download size: 1.9 M
Installed size: 11 M
Downloading Packages:
(1/2): libxcrypt-compat-4.4.33-7.amzn2023.x86_64.rpm  2.3 MB/s |  92 kB     00:00
(2/2): python3-pip-21.3.1-2.amzn2023.0.13.noarch.rpm   29 MB/s | 1.8 MB     00:00
--------------------------------------------------------------------------------
Total                                                  19 MB/s | 1.9 MB     00:00
Running transaction check
Transaction check succeeded.
Running transaction test
Transaction test succeeded.
Running transaction
  Preparing        :                                                        1/1
  Installing       : libxcrypt-compat-4.4.33-7.amzn2023.x86_64              1/2
  Installing       : python3-pip-21.3.1-2.amzn2023.0.13.noarch              2/2
  Running scriptlet: python3-pip-21.3.1-2.amzn2023.0.13.noarch              2/2
  Verifying        : libxcrypt-compat-4.4.33-7.amzn2023.x86_64              1/2
  Verifying        : python3-pip-21.3.1-2.amzn2023.0.13.noarch              2/2

Installed:
  libxcrypt-compat-4.4.33-7.amzn2023.x86_64
                                        python3-pip-21.3.1-2.amzn2023.0.13.noarch

Complete!
[ec2-user@ip-172-31-12-251 ~]$ pip3 install pandas numpy
Defaulting to user installation because normal site-packages is not writeable
Collecting pandas
  Downloading pandas-2.3.1-cp39-cp39-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (12.4 MB)
     |████████████████████████████████| 12.4 MB 2.6 MB/s
Collecting numpy
  Downloading numpy-2.0.2-cp39-cp39-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (19.5 MB)
     |████████████████████████████████| 19.5 MB 35.5 MB/s
Collecting python-dateutil>=2.8.2
  Downloading python_dateutil-2.9.0.post0-py2.py3-none-any.whl (229 kB)
     |████████████████████████████████| 229 kB 46.4 MB/s
```

2. Generate Synthetic Pizza Chain Data

Script: generate_pizza_chain_data.py

Key Changes:

- NUM_STORES = 3 to restrict the scope for simulation.
- Generates five CSV files with realistic attributes.

Script:

```
import random
from datetime import datetime, timedelta
import pandas as pd
from pathlib import Path

# Config
NUM_ORDERS = 1000
NUM_CUSTOMERS = 100
NUM_STORES = 20
NUM_SKUS = 3
MAX_ITEMS_PER_ORDER = 5
DAYS_HISTORY = 20

# Output Directory
output_dir = Path("output")
output_dir.mkdir(parents=True, exist_ok=True)

# 1. SKU Master
print("Generating SKU Master...")
print("Generating SKU Master...")
# Realistic pizza chain item names
item_catalog = {
    "Margherita Pizza": "Pizza",
    "Pepperoni Pizza": "Pizza",
    "Veggie Supreme": "Pizza",
    "BBQ Chicken Pizza": "Pizza",
    "Paneer Tikka Pizza": "Pizza",
    "Cheese Burst Pizza": "Pizza",
    "Hawaiian Pizza": "Pizza",
    "Meat Lovers": "Pizza",
    "Tandoori Paneer Pizza": "Pizza",
    "Mushroom Pizza": "Pizza",
    "Classic Cheese Pizza": "Pizza",

    "Spicy Chicken Wings": "Sides",
    "Garlic Breadsticks": "Sides",
    "Cheesy Nachos": "Sides",
    "Green Salad": "Sides",
    "Chicken Tenders": "Sides",

    "Soft Drink (Cola)": "Drinks",
    "Soft Drink (Orange)": "Drinks",

    "Chocolate Lava Cake": "Desserts"
}

sku_master = []
for i, (item_name, category) in enumerate(item_catalog.items(), 1):
    sku_master.append({
        "sku_id": f"SKU{i:04}",
        "item_name": item_name,
```

```
        "category": category,
        "price": round(random.uniform(5.0, 15.0), 2),
        "created_at": (datetime.now() - timedelta(days=random.randint(30, 365))).strftime("%Y-%m-%d %H:%M:%S")
    })
sku_df = pd.DataFrame(sku_master)
sku_df.to_csv(output_dir / "sku_master.csv", index=False)
sku_price_map = {row["sku_id"]: row["price"] for row in sku_master}
sku_ids = list(sku_price_map.keys())

# 2. Discounts
print("Generating Discounts...")
discounts = [
    {"discount_code": "DISC10", "discount_amount": 10.0},
    {"discount_code": "DISC5", "discount_amount": 5.0},
    {"discount_code": None, "discount_amount": 0.0}
]
discount_df = pd.DataFrame(discounts)
discount_df.to_csv(output_dir / "discounts_applied.csv", index=False)

# 3. Orders and Order Items
print("Generating Orders and Order Items...")
orders = []
order_items = []
start_date = datetime.now() - timedelta(days=DAYS_HISTORY)

for i in range(1, NUM_ORDERS + 1):
    order_id = f"ORD{i:07}"
    order_time = start_date + timedelta(minutes=random.randint(0, DAYS_HISTORY * 24 * 60))
    store_id = random.randint(1, NUM_STORES)
    customer_id = random.randint(1, NUM_CUSTOMERS)

    total_amount = 0.0
    num_items = random.randint(1, MAX_ITEMS_PER_ORDER)

    for _ in range(num_items):
        sku_id = random.choice(sku_ids)
        quantity = random.randint(1, 3)
        price = sku_price_map[sku_id]
        discount = random.choice(discounts)
        discount_amount = discount["discount_amount"]
        subtotal = max((price * quantity) - discount_amount, 0.0)

        order_items.append({
            "order_id": order_id,
            "sku_id": sku_id,
            "quantity": quantity,
            "unit_price": price,
            "discount_code": discount["discount_code"],
            "discount_amount": discount_amount
        })
        total_amount += subtotal

    orders.append({
        "order_id": order_id,
        "customer_id": customer_id,
        "store_id": store_id,
        "order_time": order_time,
        "total_amount": round(total_amount, 2)
```

```
    })

orders_df = pd.DataFrame(orders)
orders_df.to_csv(output_dir / "orders.csv", index=False)

order_items_df = pd.DataFrame(order_items)
order_items_df.to_csv(output_dir / "order_items.csv", index=False)

# 4. Inventory Logs
print("Generating Inventory Logs...")
inventory_logs = []
log_days = [start_date + timedelta(days=d) for d in range(DAYS_HISTORY)]

for log_day in log_days:
    for store_id in range(1, NUM_STORES + 1):
        for sku_id in sku_ids:
            inventory_logs.append({
                "log_time": log_day,
                "store_id": store_id,
                "sku_id": sku_id,
                "current_stock": random.randint(0, 100),
                "restock_threshold": 10
            })

inventory_logs_df = pd.DataFrame(inventory_logs)
inventory_logs_df.to_csv(output_dir / "inventory_logs.csv", index=False)

print(f"\n Done. Files generated in: {output_dir.resolve()}")
```

Run:

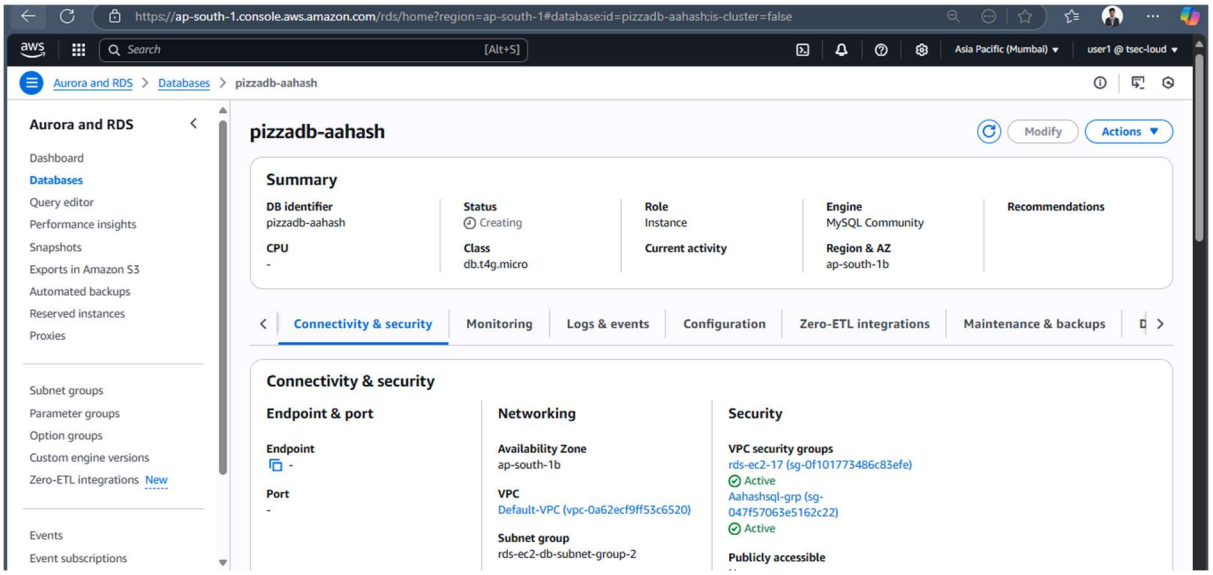
python3 generate_pizza_chain_data.py

```
Installed:
  libxcrypt-compat-4.4.33-7.amzn2023.x86_64                                   python3-pip-21.3.1-2.amzn2023.0.13.noarch

Complete!
[ec2-user@ip-172-31-12-251 ~]$ pip3 install pandas numpy
Defaulting to user installation because normal site-packages is not writeable
Collecting pandas
  Downloading pandas-2.3.1-cp39-cp39-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (12.4 MB)
     |████████████████████████████████| 12.4 MB 2.6 MB/s
Collecting numpy
  Downloading numpy-2.0.2-cp39-cp39-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (19.5 MB)
     |████████████████████████████████| 19.5 MB 35.5 MB/s
Collecting python-dateutil>=2.8.2
  Downloading python_dateutil-2.9.0.post0-py2.py3-none-any.whl (229 kB)
     |████████████████████████████████| 229 kB 46.4 MB/s
Requirement already satisfied: pytz>=2020.1 in /usr/lib/python3.9/site-packages (from pandas) (2022.7.1)
Collecting tzdata>=2022.7
  Downloading tzdata-2025.2-py2.py3-none-any.whl (347 kB)
     |████████████████████████████████| 347 kB 44.0 MB/s
Requirement already satisfied: six>=1.5 in /usr/lib/python3.9/site-packages (from python-dateutil>=2.8.2->pandas) (1.15.0)
Installing collected packages: tzdata, python-dateutil, numpy, pandas
ERROR: pip's dependency resolver does not currently take into account all the packages that are installed. This behaviour is the source of the follo
wing dependency conflicts.
awscli 2.25.0 requires python-dateutil<=2.9.0,>=2.1, but you have python-dateutil 2.9.0.post0 which is incompatible.
Successfully installed numpy-2.0.2 pandas-2.3.1 python-dateutil-2.9.0.post0 tzdata-2025.2
[ec2-user@ip-172-31-12-251 ~]$ vi generate_pizza_chain_data.py
[ec2-user@ip-172-31-12-251 ~]$ python3 generate_pizza_chain_data.py
Generating SKU Master...
Generating Discounts...
Generating Orders and Order Items...
Generating Inventory Logs...

✅ Done. Files generated in: /home/ec2-user/output
[ec2-user@ip-172-31-12-251 ~]$ ls
generate_pizza_chain_data.py  output
[ec2-user@ip-172-31-12-251 ~]$ cd output
[ec2-user@ip-172-31-12-251 output]$ ls
discounts_applied.csv  inventory_logs.csv  order_items.csv  orders.csv  sku_master.csv
[ec2-user@ip-172-31-12-251 output]$
```

3. Launch and Configure Amazon RDS (MySQL)

- **Engine:** MySQL 8.x
- **Instance Identifier:** pizzadb-aahash
- **Username:** admin
- **Password:** Aahash123

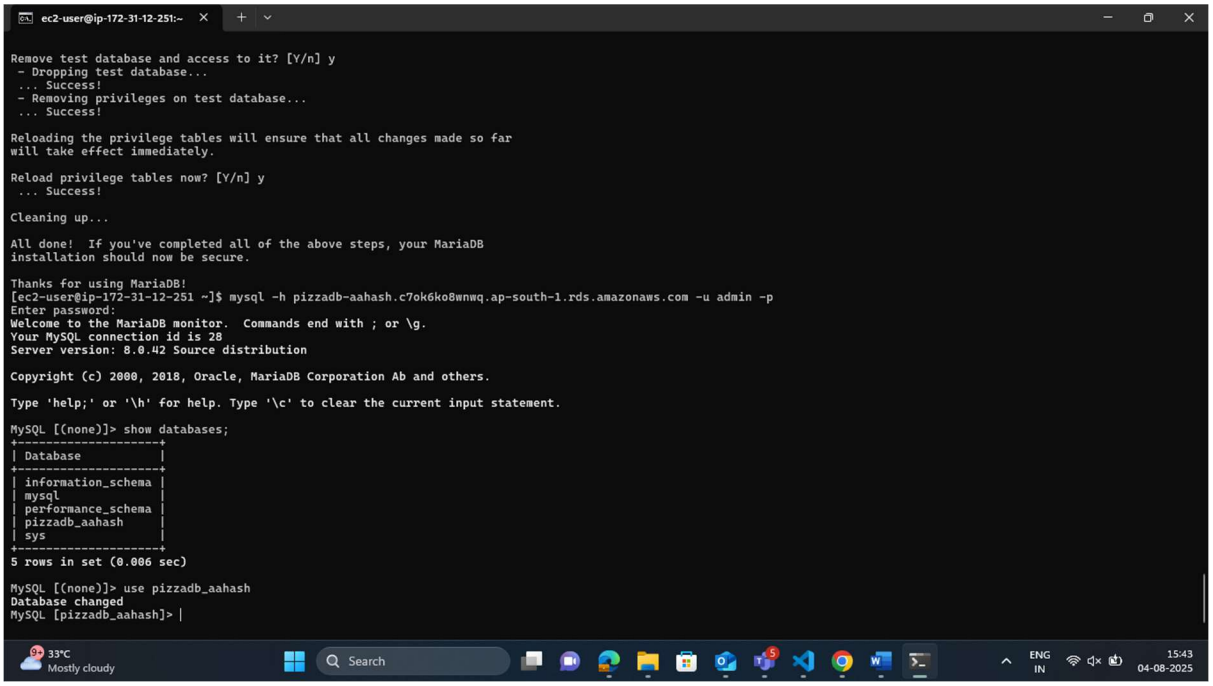
- **Enable Public Access**
- **Add Inbound Rule on Port 3306** to allow EC2 connection



4. Start and Enable MariaDB

sudo systemctl enable mariadb

sudo systemctl start mariadb



5. Create tables:

1. sku_master – Master list of SKUs (Pizza types, Beverages, etc.)

```
CREATE TABLE sku_master (
  sku_id VARCHAR(20) PRIMARY KEY,
  sku_name VARCHAR(100),
  category VARCHAR(50),
  unit_price DECIMAL(10,2),
  is_veg BOOLEAN,
```

```
  created_at DATETIME DEFAULT CURRENT_TIMESTAMP
);
```

2. inventory_logs – Simulates stock in/out movements

```
CREATE TABLE inventory_logs (
  log_id INT AUTO_INCREMENT PRIMARY KEY,
  sku_id VARCHAR(20),
  branch_id VARCHAR(10),
  log_type ENUM('IN', 'OUT'),
  quantity INT,
  log_timestamp DATETIME,
  FOREIGN KEY (sku_id) REFERENCES sku_master(sku_id)
);
```

3. orders – Customer order info

```
CREATE TABLE orders (
  order_id VARCHAR(20) PRIMARY KEY,
  branch_id VARCHAR(10),
  order_datetime DATETIME,
  customer_name VARCHAR(100),
  total_amount DECIMAL(10,2),
  payment_method ENUM('CASH', 'CARD', 'UPI', 'WALLET'),
  order_status ENUM('PLACED', 'CANCELLED', 'DELIVERED')
);
```

4. order_items – Items within an order

```
CREATE TABLE order_items (
  order_item_id INT AUTO_INCREMENT PRIMARY KEY,
  order_id VARCHAR(20),
  sku_id VARCHAR(20),
  quantity INT,
  item_price DECIMAL(10,2),
  FOREIGN KEY (order_id) REFERENCES orders(order_id),
  FOREIGN KEY (sku_id) REFERENCES sku_master(sku_id)
);
```

5. discounts_applied – Discounts given on orders

```
CREATE TABLE discounts_applied (
  discount_id INT AUTO_INCREMENT PRIMARY KEY,
  order_id VARCHAR(20),
  discount_code VARCHAR(20),
  discount_amount DECIMAL(10,2),
  FOREIGN KEY (order_id) REFERENCES orders(order_id)
);
```

6. Store manager

```
CREATE TABLE store_manager (
   store_id INT PRIMARY KEY,
   manager_name VARCHAR(50),
   email_id VARCHAR(100)
);
```

```
MySQL [pizzadb_aahash]> CREATE TABLE orders (
    ->   order_id VARCHAR(20) PRIMARY KEY,
    ->   customer_id VARCHAR(75),
    ->   store_id VARCHAR(2),
    ->   order_time VARCHAR(50),
    ->   total_amount DOUBLE
    -> );
Query OK, 0 rows affected (0.038 sec)

MySQL [pizzadb_aahash]> CREATE TABLE order_items (
    ->   order_item_id INT AUTO_INCREMENT PRIMARY KEY,
    ->   order_id VARCHAR(20),
    ->   sku_id VARCHAR(10),
    ->   quantity DOUBLE,
    ->   unit_price DOUBLE,
    ->   discount_code VARCHAR(20),
    ->   discount_amount DOUBLE,
    ->   FOREIGN KEY (order_id) REFERENCES orders(order_id),
    ->   FOREIGN KEY (sku_id) REFERENCES sku_master(sku_id)
    -> );
Query OK, 0 rows affected (0.057 sec)

MySQL [pizzadb_aahash]> CREATE TABLE discounts_applied (
    ->   discount_id INT AUTO_INCREMENT PRIMARY KEY,
    ->   order_id VARCHAR(20),
    ->   discount_code VARCHAR(20),
    ->   discount_amount DOUBLE,
    ->   FOREIGN KEY (order_id) REFERENCES orders(order_id)
    -> );
Query OK, 0 rows affected (0.038 sec)

MySQL [pizzadb_aahash]> SHOW TABLES;
+-------------------------+
| Tables_in_pizzadb_aahash |
+-------------------------+
| discounts_applied       |
| inventory_logs          |
| order_items             |
| orders                  |
| sku_master              |
+-------------------------+
5 rows in set (0.002 sec)

MySQL [pizzadb_aahash]>
```

6. Load CSV Files into RDS using Python

pip3 install pymysql sqlalchemy

Loader Script: load_data_to_rds.py

```
import pandas as pd
from sqlalchemy import create_engine

host = 'pizzadb-aahash.c7ok6ko8wnwq.ap-south-1.rds.amazonaws.com'
port = 3306
username = 'admin'
password = 'Aahash123'
database = 'pizzadb_aahash'
```

```
engine = create_engine(f'mysql+pymysql://{username}:{password}@{host}:{port}/{database}')

tables = ["sku_master", "discounts_applied", "orders", "order_items", "inventory_logs", “store_manager”]

for table in tables:
    print(f"Uploading {table}...")
    df = pd.read_csv(f'output/{table}.csv')
    df.to_sql(table, con=engine, if_exists='append', index=False)

print("All data loaded to RDS.")
```

```
(Background on this error at: https://sqlalche.me/e/20/e3q8)
[ec2-user@ip-172-31-12-251 ~]$ vi load_data_to_rds.py
[ec2-user@ip-172-31-12-251 ~]$ python3 load_data_to_rds.py
Uploading sku_master...
Uploading discounts_applied...
Uploading orders...
Uploading order_items...
Uploading inventory_logs...
✅ All data loaded to RDS.
```

7. Verify Data Load in MySQL

[ec2-user@ip-172-31-12-251 ~]$ mysql -h pizzadb-aahash.c7ok6ko8wnwq.ap-south-1.rds.amazonaws.com -u admin -p

USE pizzadb;

SELECT COUNT(*) FROM orders;

SELECT * FROM sku_master LIMIT 5;

```
[ec2-user@ip-172-31-12-251 ~]$ mysql -h pizzadb-aahash.c7ok6ko8wnwq.ap-south-1.rds.amazonaws.com -u admin -p
Enter password:
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MySQL connection id is 34
Server version: 8.0.42 Source distribution

Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

MySQL [(none)]> use pizzadb_aahash
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
MySQL [pizzadb_aahash]> SHOW TABLES;
+---------------------------+
| Tables_in_pizzadb_aahash  |
+---------------------------+
| discounts_applied         |
| inventory_logs            |
| order_items               |
| orders                    |
| sku_master                |
+---------------------------+
5 rows in set (0.001 sec)

MySQL [pizzadb_aahash]> SELECT COUNT(*) FROM orders;
+----------+
| COUNT(*) |
+----------+
|     1000 |
+----------+
1 row in set (0.002 sec)

MySQL [pizzadb_aahash]> SELECT * FROM sku_master LIMIT 5;
+---------+--------------------+----------+-------+---------------------+
| sku_id  | item_name          | category | price | created_at          |
+---------+--------------------+----------+-------+---------------------+
| SKU0001 | Margherita Pizza   | Pizza    |  7.17 | 2024-11-06 09:58:38 |
| SKU0002 | Pepperoni Pizza    | Pizza    |  7.61 | 2025-01-22 09:58:38 |
| SKU0003 | Veggie Supreme     | Pizza    |  9.37 | 2025-04-07 09:58:38 |
| SKU0004 | BBQ Chicken Pizza  | Pizza    | 11.82 | 2024-09-22 09:58:38 |
| SKU0005 | Paneer Tikka Pizza | Pizza    |  7.36 | 2025-01-30 09:58:38 |
+---------+--------------------+----------+-------+---------------------+
5 rows in set (0.001 sec)

MySQL [pizzadb_aahash]>
```

Phase 2: AWS Glue – Extract, Transform & Load (ETL) to Amazon S3

Objective

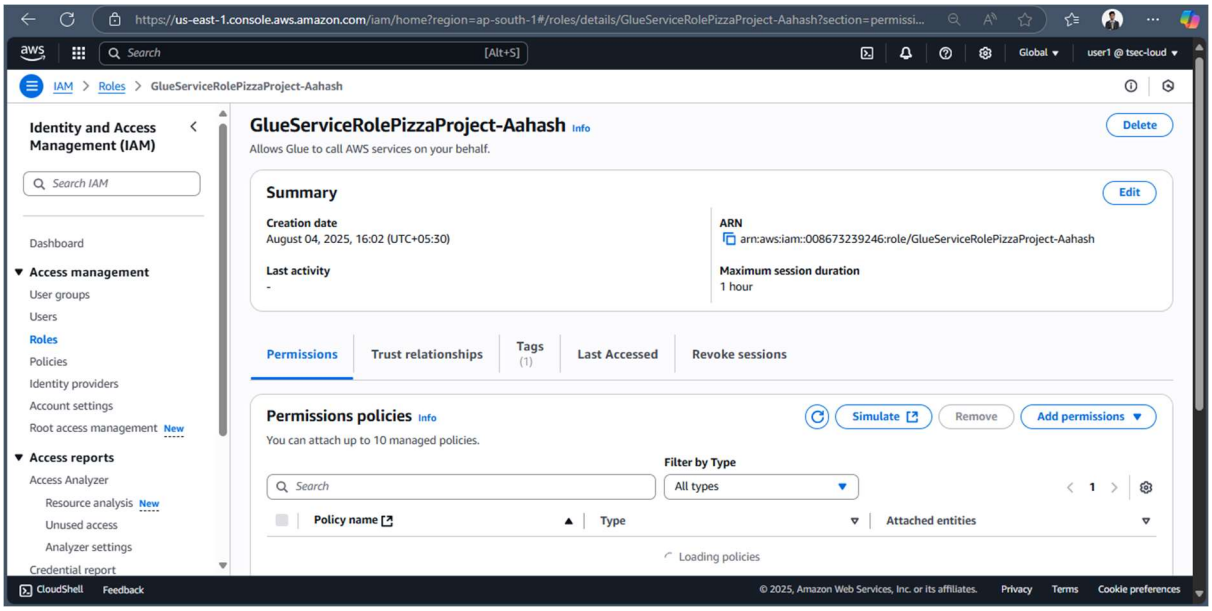
To extract data from Amazon RDS (MySQL), perform basic transformations using AWS Glue, and load the transformed datasets into Amazon S3 in Parquet format for downstream querying and analysis.

1. Create IAM Role for Glue

Role Name: GlueServiceRolePizzaProject-Aahash

Permissions:

- AmazonS3FullAccess
- AmazonRDSFullAccess
- AWSGlueServiceRole

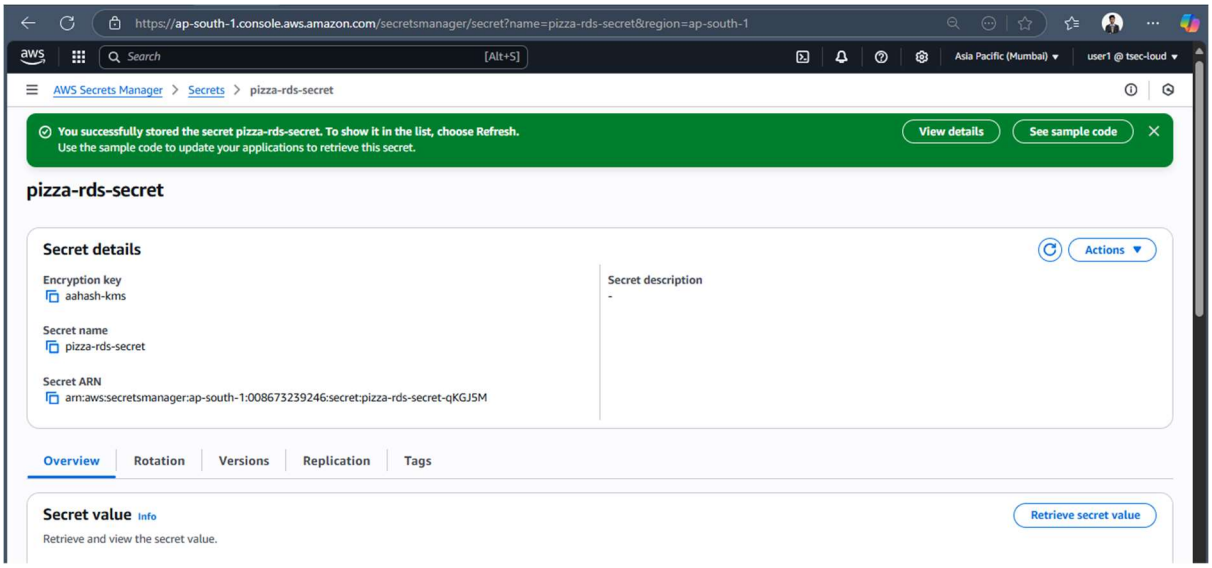


2. Store RDS Credentials Securely

Use **AWS Secrets Manager** to store RDS connection securely.

Steps:

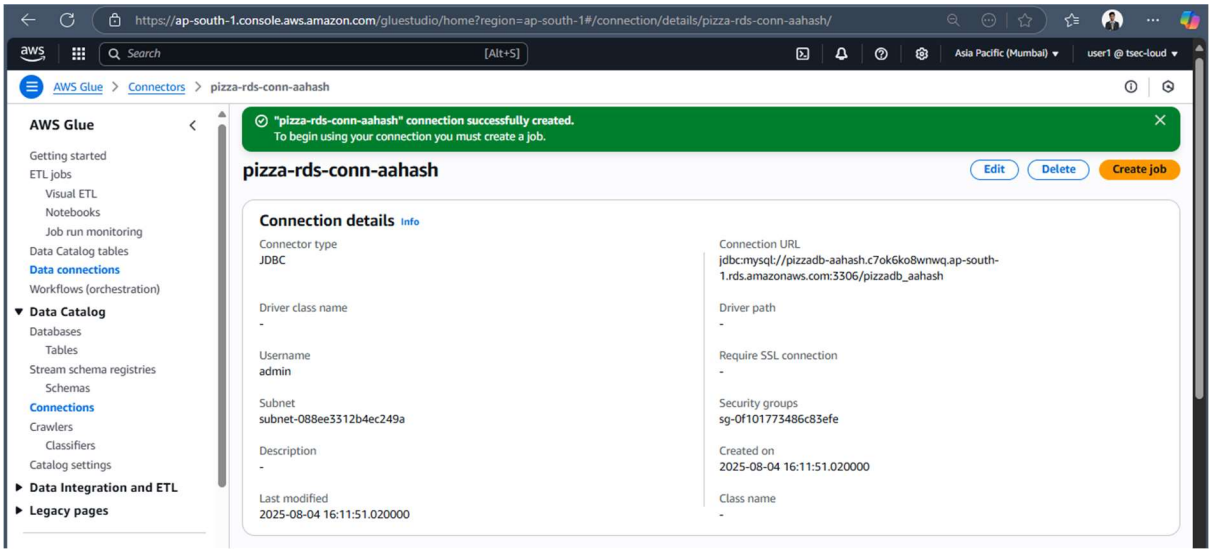
- Go to Secrets Manager → Store a new secret
- Choose **RDS database credentials**
- Add your admin username/password
- Choose Other type of database
- Add RDS endpoint, port (3306), and DB name (pizzadb-aahash)
- Save as: pizza-rds-secret



3. Create a Glue Connection to RDS

Steps:

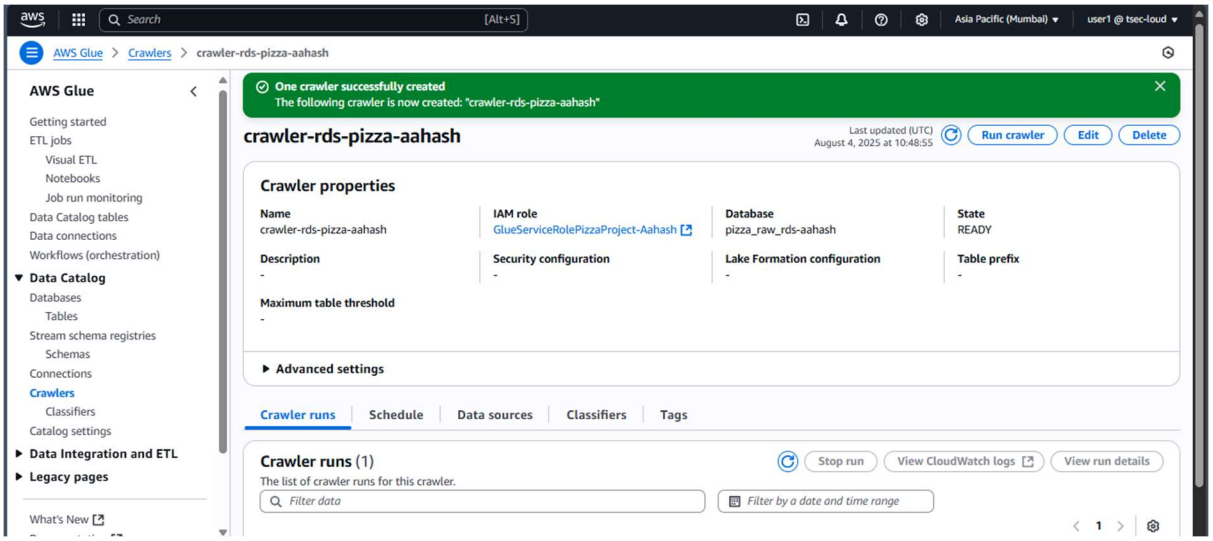
- Go to AWS Glue → Connections → Add connection
- Name: pizza-rds-conn
- Type: JDBC
- Choose the secret stored in Secrets Manager
- Test connection using a temporary EC2 for VPC access

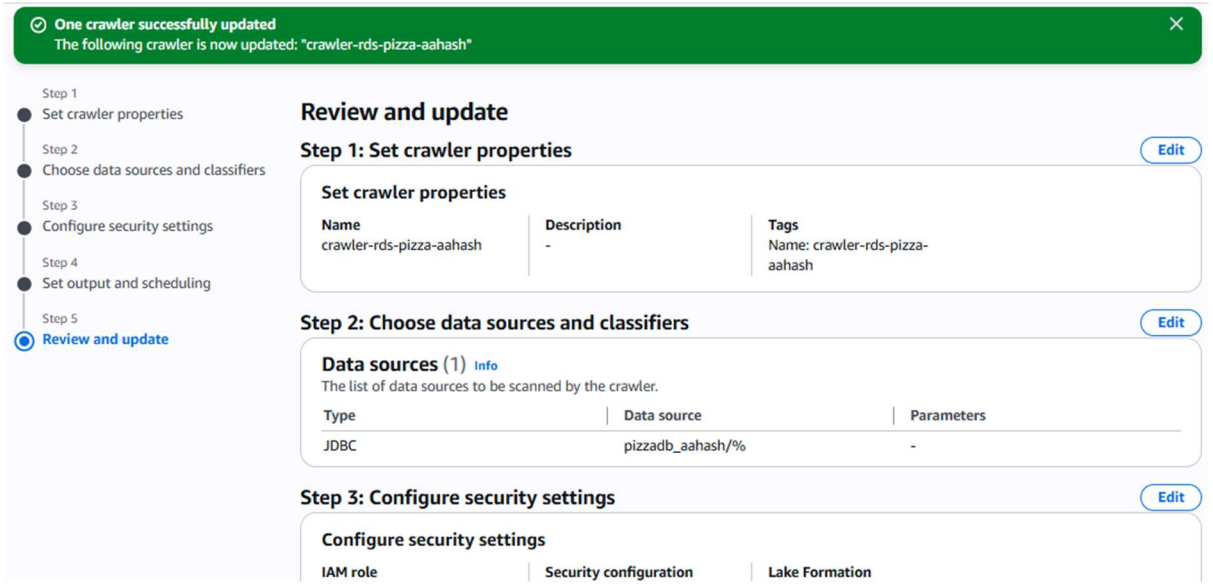


4. Create a Glue Crawler for RDS

Steps:

- Name: crawler-rds-pizza-aahash
- Data store: JDBC (use pizza-rds-conn)
- Database: pizza_raw_rds-aahash
- Output: Store metadata in Glue Data Catalog
- Schedule: Run on demand
- IAM Role: GlueServiceRolePizzaProject-aahash

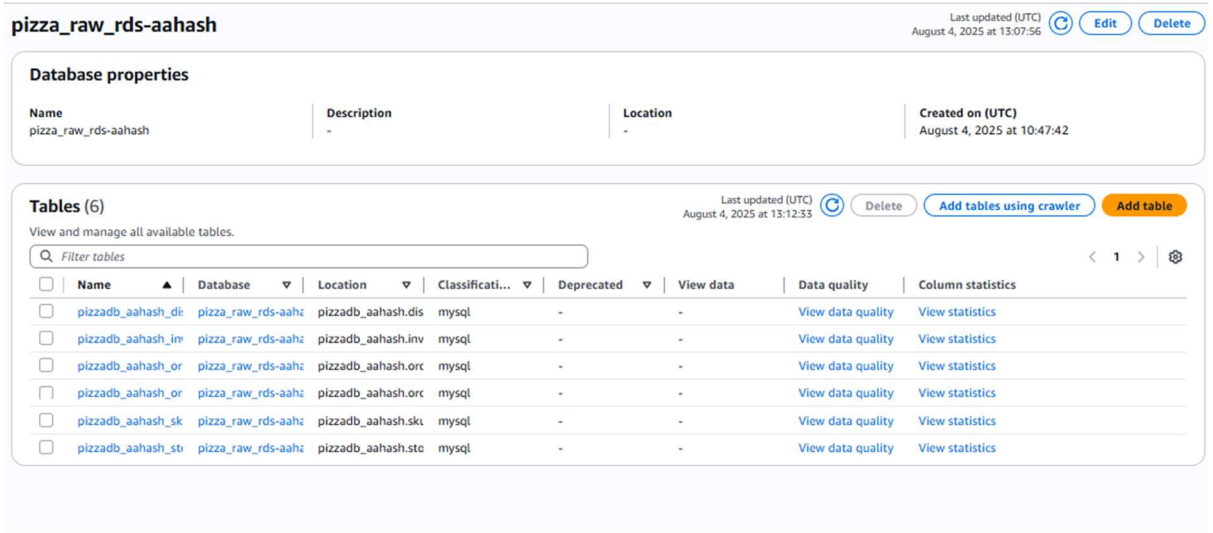




Added JDBC connection

Run the crawler:

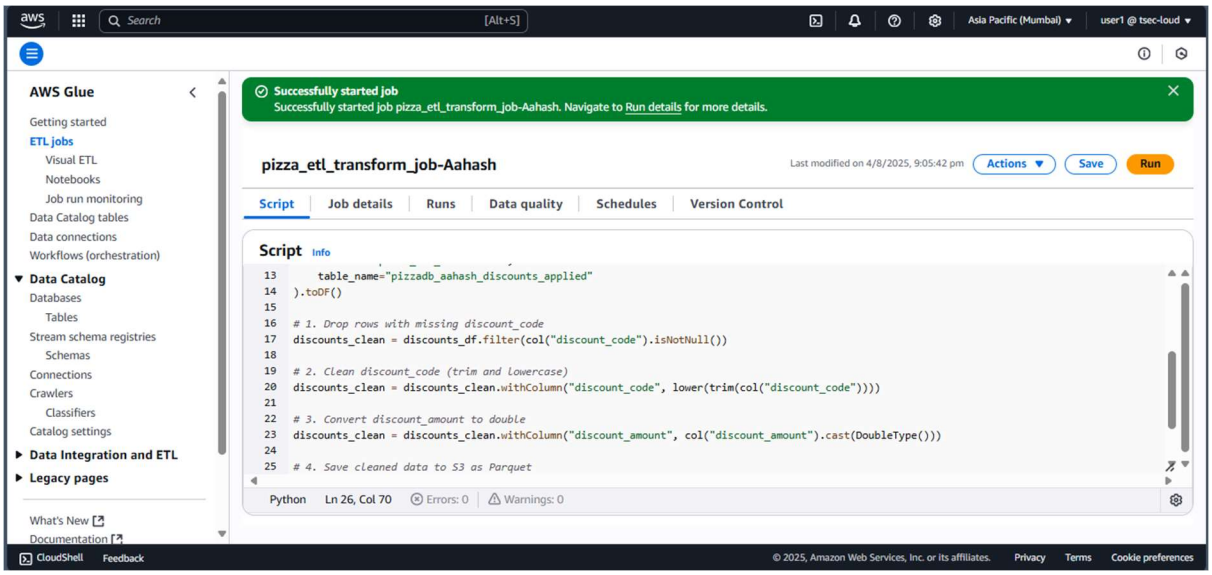
Tables are created successfully

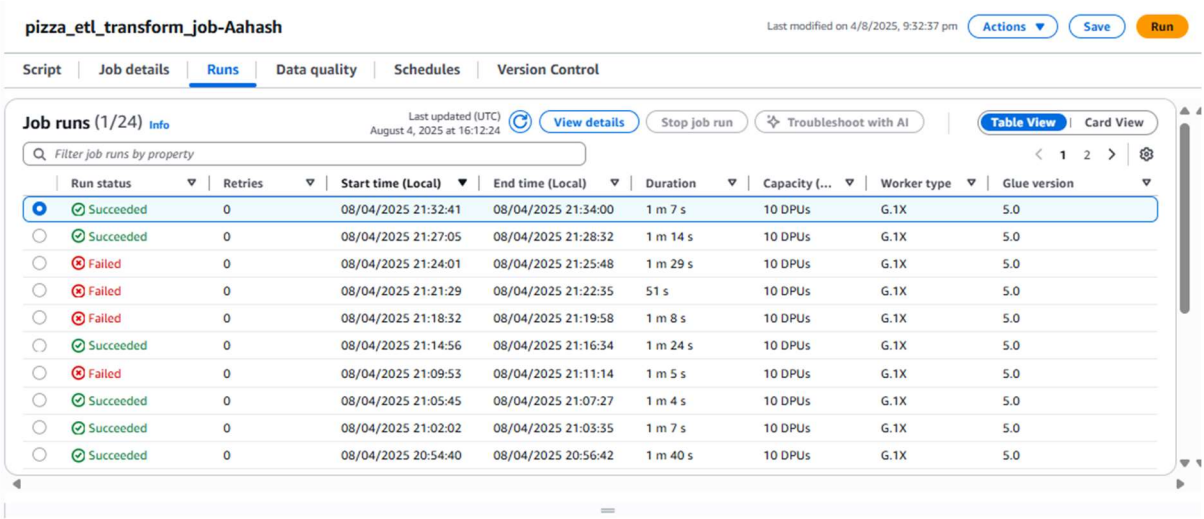


Step 5. Data Transformation Using Glue ETL Scripts
Target Format: Parquet
Target Location: s3://aahash-project2/pizza
Apply transformation for all 6 tables:
Code for discount:

```
from pyspark.sql import SparkSession
from pyspark.sql.functions import col, lower, trim
from pyspark.sql.types import DoubleType
from awsglue.context import GlueContext

# Initialize Spark and Glue
spark = SparkSession.builder.appName("DiscountsETL").getOrCreate()
glueContext = GlueContext(spark.sparkContext)

# Load discounts_applied from Glue Catalog
discounts_df = glueContext.create_dynamic_frame.from_catalog(
   database="pizza_raw_rds-aahash",
   table_name="pizzadb_aahash_discounts_applied"
).toDF()

# 1. Drop rows with missing discount_code
discounts_clean = discounts_df.filter(col("discount_code").isNotNull())

# 2. Clean discount_code (trim and lowercase)
discounts_clean = discounts_clean.withColumn("discount_code", lower(trim(col("discount_code"))))

# 3. Convert discount_amount to double
discounts_clean = discounts_clean.withColumn("discount_amount",
col("discount_amount").cast(DoubleType()))

# 4. Save cleaned data to S3 as Parquet
discounts_clean.write.mode("overwrite").parquet("s3://aahash-project3/pizza/cleaned_discounts_applied/")

print("Cleaned discounts_applied saved to S3.")
```

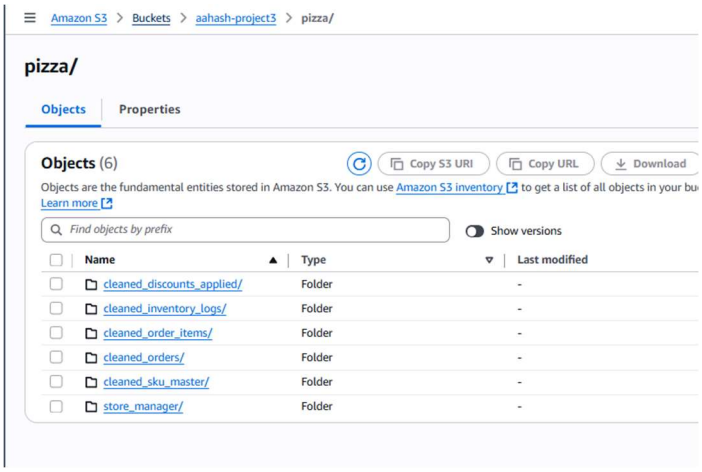
Like this for other tables different ETL script has been created…
ETL script was created for transformation of data and saving them into s3 in parquet format.





Job runs succeeded

Output Locations in S3:



Phase 3: Data Transformation & Querying

Objective:

Transform cleaned RDS data stored in S3 (Parquet format), register them as Athena tables using AWS Glue Crawlers, and run analytical queries for insights.

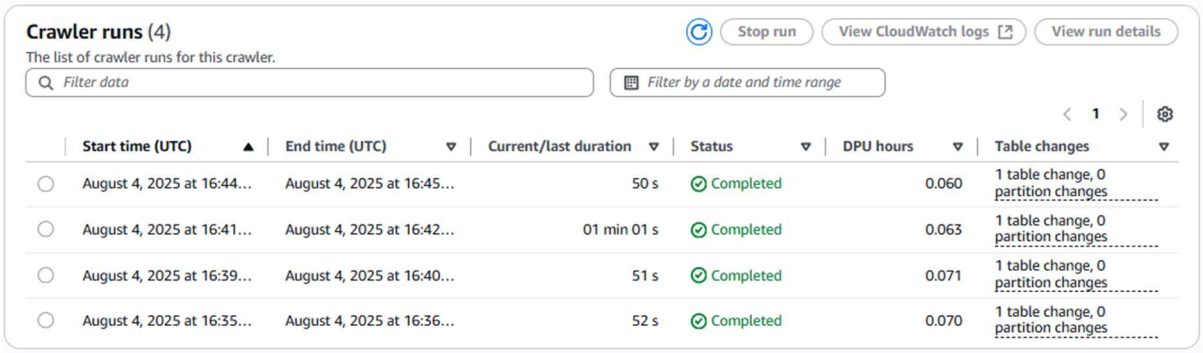
Step1: Verify Cleaned Datasets in S3

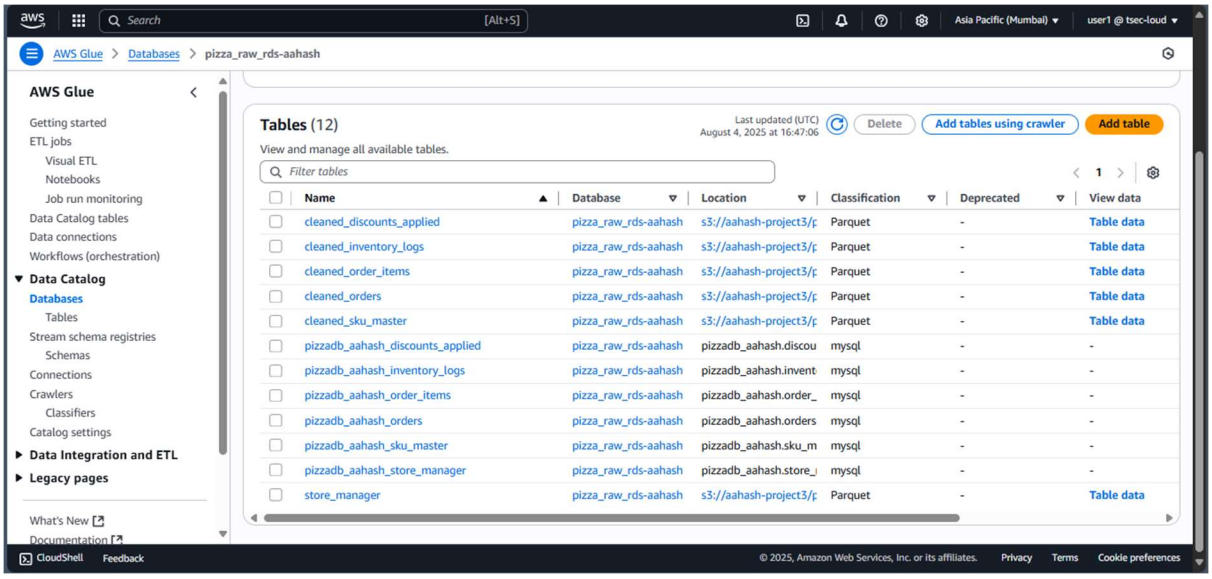
All cleaned and transformed datasets (in Parquet format) are stored in:
s3://aahash-project3/pizza/cleaned_discounts_applied/

Step2. Create AWS Glue Crawler for Cleaned Data

Crawler Name: crawler-cleaned-pizza-aahash
Target: s3://aahash-project3/pizza/
Database Name: pizza_raw_rds-aahash
IAM Role: GlueServiceRolePizzaProject-Aahash
Schedule: Run on demand
Steps:

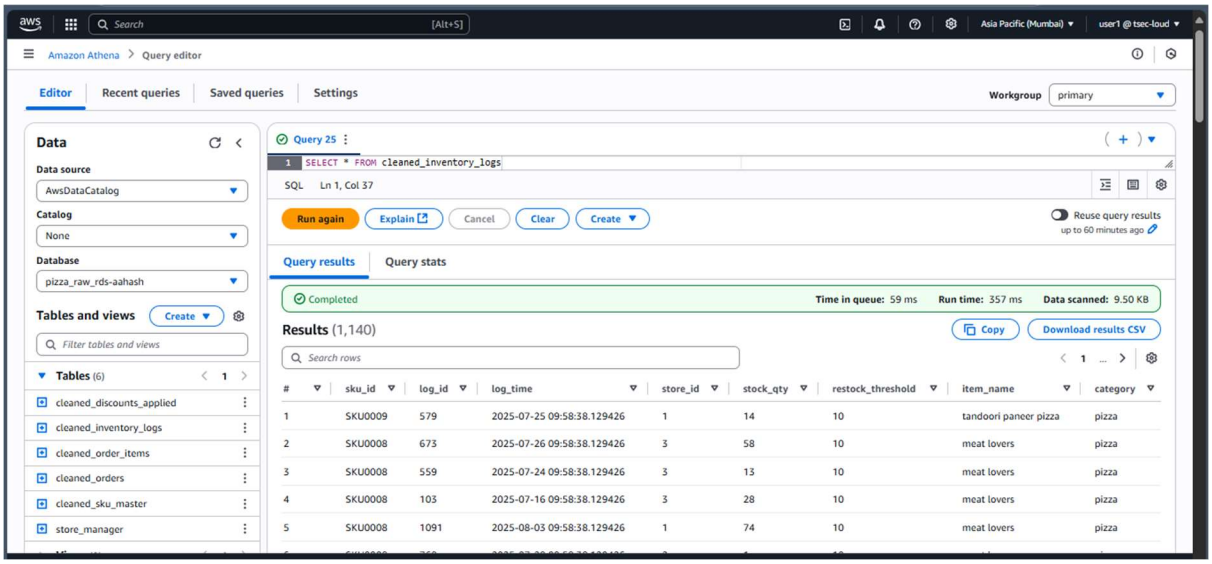
1. Go to AWS Glue → Crawlers → Add crawler
2. Point to root path s3://aahash-project3/pizza/
3. Let it crawl all cleaned_ folders
4. Run the crawler → confirm tables appear in the database.





Step3: Athena Setup

- **Athena Database:** pizza_raw_rds-aahash
- Confirm all tables

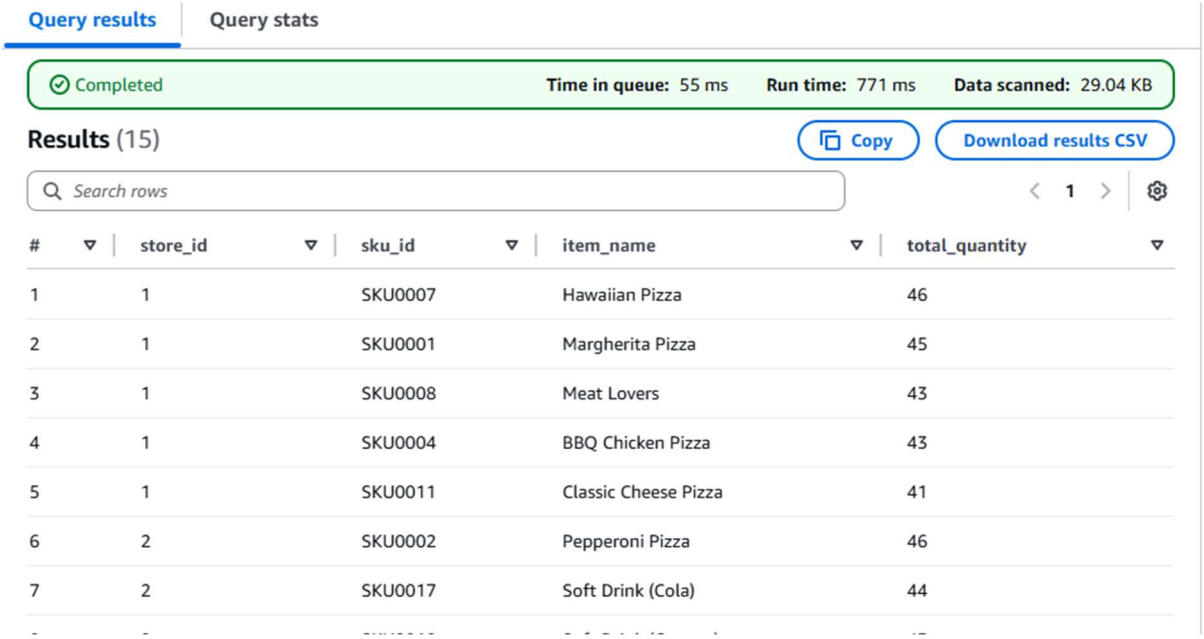


Step 4: Run the 12 Analytical Queries

1. Top 5 Selling SKUs per Store in the Last 7 Days

```
WITH recent_orders AS (
   SELECT order_id,  store_id,
      parse_datetime(order_time, 'yyyy-MM-dd HH:mm:ss.SSSSSS') AS order_ts
   FROM   cleaned_orders
   WHERE DATE(parse_datetime(order_time, 'yyyy-MM-dd HH:mm:ss.SSSSSS')) >= current_date - interval '7' day
),
joined_data AS (
   SELECT
      ro.store_id,
      coi.sku_id,
      coi.item_name,
      SUM(coi.quantity) AS total_quantity
   FROM
      cleaned_order_items coi
   JOIN
      recent_orders ro
   ON
      coi.order_id = ro.order_id
   GROUP BY
      ro.store_id, coi.sku_id, coi.item_name
),
ranked_data AS (
   SELECT
      *,
      ROW_NUMBER() OVER (PARTITION BY store_id ORDER BY total_quantity DESC) AS rank
```

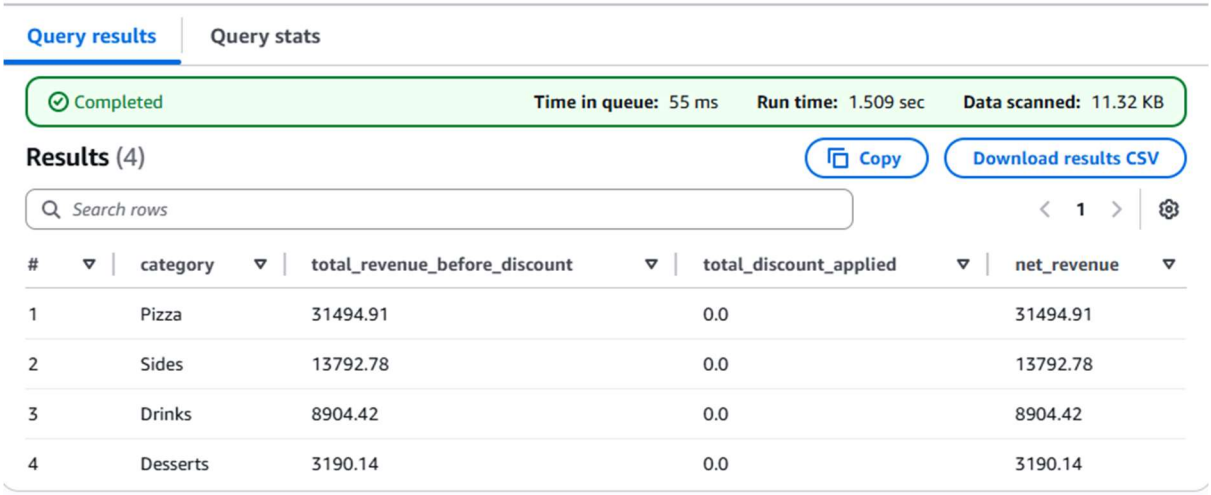
```
    FROM
        joined_data
)
SELECT
    store_id,
    sku_id,
    item_name,
    total_quantity
FROM
    ranked_data
WHERE
    rank <= 5
ORDER BY
    store_id, total_quantity DESC;
```



2. Category-wise Revenue Breakdown with Discounts Applied

```
WITH item_revenue AS (
    SELECT
        oi.category,
        oi.order_id,
        SUM(oi.price * oi.quantity) AS category_revenue
    FROM cleaned_order_items oi
    GROUP BY oi.category, oi.order_id
),
discounts AS (
    SELECT
        order_id,
        SUM(discount_amount) AS total_discount
    FROM cleaned_discounts_applied
    GROUP BY order_id
),
revenue_with_discount AS (
```

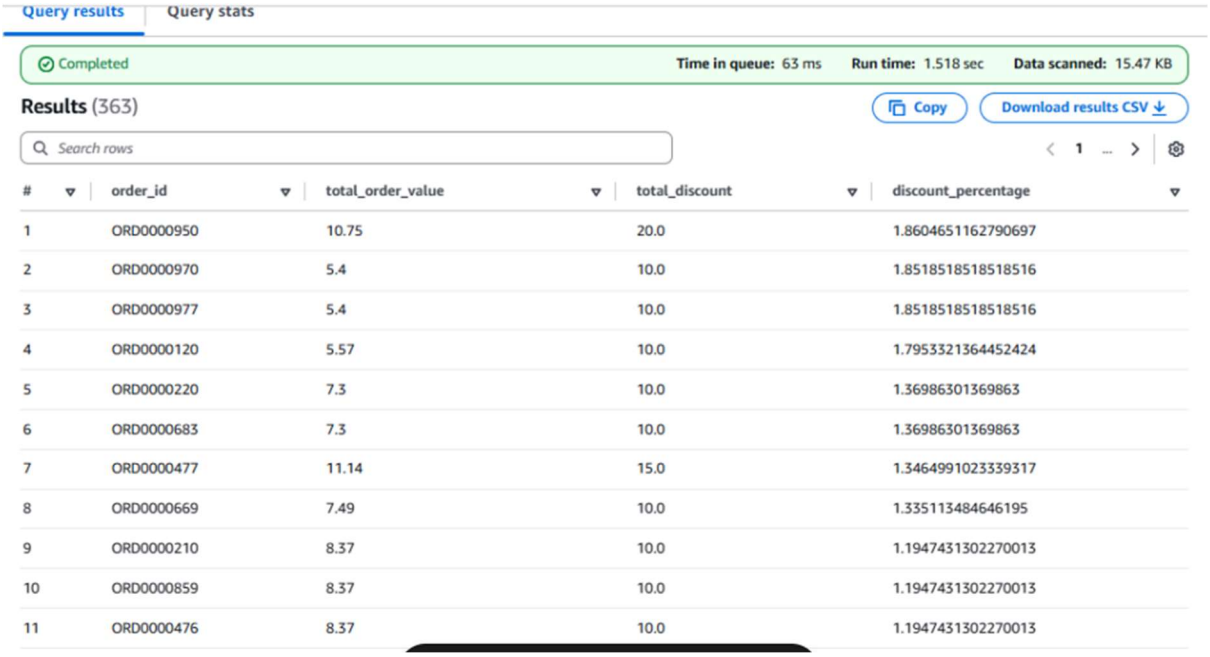
```
    SELECT
        ir.category,
        ir.category_revenue,
        COALESCE(d.total_discount, 0) AS discount
    FROM item_revenue ir
    LEFT JOIN discounts d ON ir.order_id = d.order_id
)
SELECT
    category,
    ROUND(SUM(category_revenue), 2) AS total_revenue_before_discount,
    ROUND(SUM(discount), 2) AS total_discount_applied,
    ROUND(SUM(category_revenue) - SUM(discount), 2) AS net_revenue
FROM revenue_with_discount
GROUP BY category
ORDER BY net_revenue DESC;
```



#	category	total_revenue_before_discount	total_discount_applied	net_revenue
1	Pizza	31494.91	0.0	31494.91
2	Sides	13792.78	0.0	13792.78
3	Drinks	8904.42	0.0	8904.42
4	Desserts	3190.14	0.0	3190.14

3. Identify Orders Where Discount > 30% of Total Value

```
WITH order_summaries AS (
 SELECT
 order_id,
 SUM(quantity_int * unit_price_double) AS total_order_value,
 SUM(discount_amount) AS total_discount
 FROM cleaned_order_items
 GROUP BY order_id
)
SELECT
 order_id,
 total_order_value,
 total_discount,
 (total_discount / total_order_value) AS discount_percentage
FROM order_summaries
WHERE total_order_value > 0 -- avoid division by zero
 AND (total_discount / total_order_value) > 0.30
ORDER BY discount_percentage DESC;
```



#	order_id	total_order_value	total_discount	discount_percentage
1	ORD0000950	10.75	20.0	1.8604651162790697
2	ORD0000970	5.4	10.0	1.8518518518518516
3	ORD0000977	5.4	10.0	1.8518518518518516
4	ORD0000120	5.57	10.0	1.7953321364452424
5	ORD0000220	7.3	10.0	1.36986301369863
6	ORD0000683	7.3	10.0	1.36986301369863
7	ORD0000477	11.14	15.0	1.3464991023339317
8	ORD0000669	7.49	10.0	1.335113484646195
9	ORD0000210	8.37	10.0	1.1947431302270013
10	ORD0000859	8.37	10.0	1.1947431302270013
11	ORD0000476	8.37	10.0	1.1947431302270013

4. Low Inventory Alert Based on Last Weekend's Average Sales

```
WITH weekend_sales AS (
    SELECT
        oi.sku_id,
        o.store_id,
        DATE_PARSE(o.order_time, '%Y-%m-%d %H:%i:%s') AS order_date,
        CAST(oi.quantity AS INT) AS qty
    FROM cleaned_order_items oi
    JOIN cleaned_orders o
      ON oi.order_id = o.order_id
    WHERE day_of_week IN ('Saturday', 'Sunday')
),
average_weekend_sales AS (
    SELECT
        sku_id,
        store_id,
        AVG(qty) AS avg_weekend_sales
    FROM weekend_sales
    GROUP BY sku_id, store_id
)
SELECT
    il.sku_id,
    il.store_id,
    il.stock_qty,
    aws.avg_weekend_sales,
    il.item_name,
    il.category
FROM cleaned_inventory_logs il
JOIN average_weekend_sales aws
  ON il.sku_id = aws.sku_id
```

```
  AND CAST(il.store_id AS VARCHAR) = aws.store_id
WHERE il.stock_qty < aws.avg_weekend_sales
ORDER BY (aws.avg_weekend_sales - il.stock_qty) DESC;
```

Query results | Query stats

Completed | Time in queue: 58 ms | Run time: 1.008 sec | Data scanned: 25.25 KB

Results (31) — Copy | Download results CSV

Search rows — 1

#	sku_id	store_id	stock_qty	avg_weekend_sales	item_name	category
1	SKU0001	1	0	2.357142857142857	margherita pizza	pizza
2	SKU0015	1	0	2.272727272727273	green salad	sides
3	SKU0007	2	0	2.25	hawaiian pizza	pizza
4	SKU0004	1	0	2.235294117647059	bbq chicken pizza	pizza
5	SKU0009	2	0	2.1875	tandoori paneer pizza	pizza
6	SKU0015	2	0	1.8695652173913044	green salad	sides
7	SKU0016	2	0	1.8461538461538463	chicken tenders	sides

5. Revenue and Orders by Hour of Day

```
SELECT
  HOUR(DATE_PARSE(order_time, '%Y-%m-%d %H:%i:%s.%f')) AS order_hour,
  COUNT(DISTINCT o.order_id) AS total_orders,
  SUM(oi.quantity * oi.unit_price) AS total_revenue
FROM cleaned_orders o
JOIN cleaned_order_items oi
  ON o.order_id = oi.order_id
GROUP BY 1
ORDER BY 1;
```

Completed | Time in queue: 59 ms | Run time: 1.054 sec | Data scanned: 22.41 KB

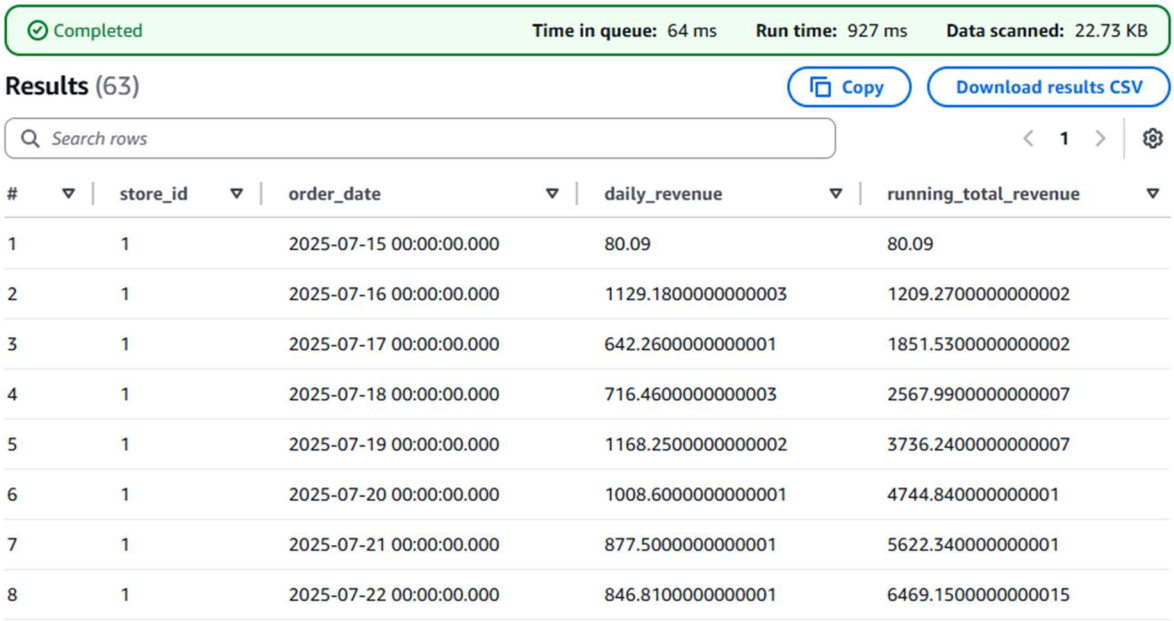
Results (24) — Copy | Download results CSV

Search rows — 1

#	order_hour	total_orders	total_revenue
1	0	54	2687.2800000000007
2	1	41	2579.5600000000004
3	2	38	2137.44
4	3	34	1831.85
5	4	38	2035.6800000000003
6	5	38	2412.6000000000004
7	6	39	2351.9
8	7	43	2116.38
9	8	46	2442.6000000000004

6. Running Total of Revenue by Store

```
SELECT
  o.store_id,
  DATE_TRUNC('day', CAST(o.order_time AS timestamp)) AS order_date,
  SUM(oi.quantity * oi.unit_price) AS daily_revenue,
  SUM(SUM(oi.quantity * oi.unit_price)) OVER (
    PARTITION BY o.store_id
    ORDER BY DATE_TRUNC('day', CAST(o.order_time AS timestamp))
    ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW
  ) AS running_total_revenue
FROM cleaned_orders o
JOIN cleaned_order_items oi
  ON o.order_id = oi.order_id
GROUP BY o.store_id, DATE_TRUNC('day', CAST(o.order_time AS timestamp))
ORDER BY o.store_id, order_date;
```



Completed | Time in queue: 64 ms | Run time: 927 ms | Data scanned: 22.73 KB

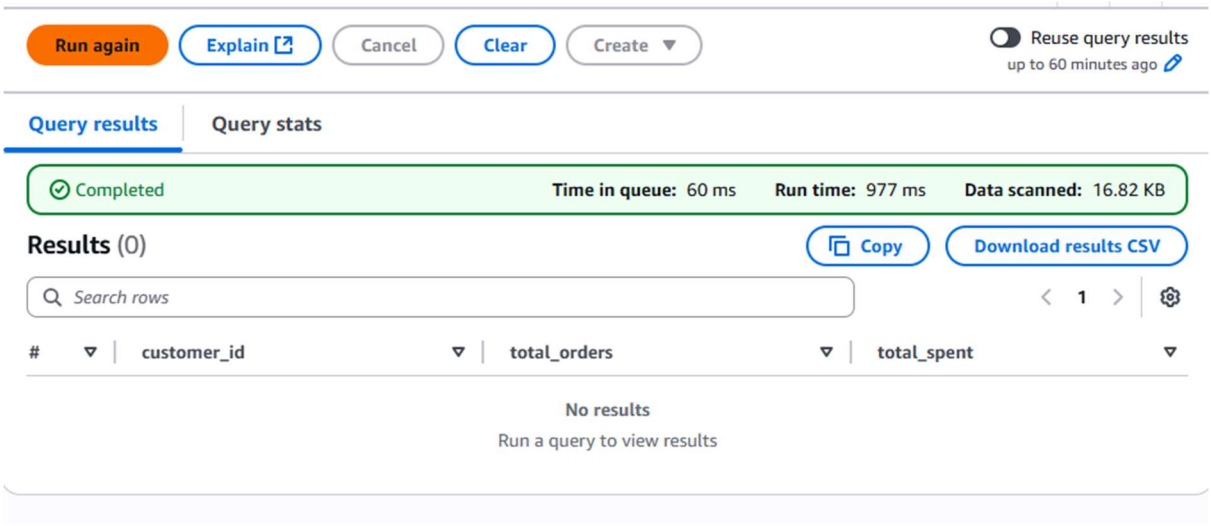
Results (63) — Copy — Download results CSV

Search rows — 1

#	store_id	order_date	daily_revenue	running_total_revenue
1	1	2025-07-15 00:00:00.000	80.09	80.09
2	1	2025-07-16 00:00:00.000	1129.1800000000003	1209.2700000000002
3	1	2025-07-17 00:00:00.000	642.2600000000001	1851.5300000000002
4	1	2025-07-18 00:00:00.000	716.4600000000003	2567.9900000000007
5	1	2025-07-19 00:00:00.000	1168.2500000000002	3736.2400000000007
6	1	2025-07-20 00:00:00.000	1008.6000000000001	4744.840000000001
7	1	2025-07-21 00:00:00.000	877.5000000000001	5622.340000000001
8	1	2025-07-22 00:00:00.000	846.8100000000001	6469.1500000000015

7. Customers with High Frequency and High Value Order

```
SELECT
  o.customer_id,
  COUNT(DISTINCT o.order_id) AS total_orders,
  ROUND(SUM(oi.quantity * oi.unit_price), 2) AS total_spent
FROM cleaned_orders o
JOIN cleaned_order_items oi
  ON o.order_id = oi.order_id
GROUP BY o.customer_id
HAVING
  COUNT(DISTINCT o.order_id) > 5 -- High frequency threshold
  AND SUM(oi.quantity * oi.unit_price) > 5000 -- High value threshold
ORDER BY total_spent DESC;
```



8. Most Discounted Products by Total Discount Given

```
SELECT
sku_id,
 item_name,
 SUM(discount_amount) AS total_discount_given
FROM cleaned_discounts_applied
GROUP BY sku_id, item_name
ORDER BY total_discount_given DESC
LIMIT 10;
```

Completed — Time in queue: 60 ms — Run time: 501 ms — Data scanned: 2.17 KB

Results (10)

#	sku_id	item_name	total_discount_given
1	SKU0001	Margherita Pizza	975.0
2	SKU0011	Classic Cheese Pizza	925.0
3	SKU0004	BBQ Chicken Pizza	905.0
4	SKU0008	Meat Lovers	870.0
5	SKU0016	Chicken Tenders	870.0
6	SKU0015	Green Salad	840.0
7	SKU0003	Veggie Supreme	815.0
8	SKU0019	Chocolate Lava Cake	805.0
9	SKU0018	Soft Drink (Orange)	790.0
10	SKU0002	Pepperoni Pizza	785.0

9. SKU Sell-Through Rate (Sold Quantity vs. Inventory

```
SELECT
    sm.sku_id,
    sm.item_name,
    COALESCE(SUM(oi.quantity), 0) AS total_sold_quantity,
    MAX(il.stock_qty) AS max_recorded_inventory,
    ROUND(
        COALESCE(SUM(oi.quantity), 0) * 100.0 / NULLIF(MAX(il.stock_qty), 0),
```

```
    2
  ) AS sell_through_rate_pct
FROM cleaned_sku_master sm
LEFT JOIN cleaned_order_items oi
  ON sm.sku_id = oi.sku_id
LEFT JOIN cleaned_inventory_logs il
  ON sm.sku_id = il.sku_id
GROUP BY sm.sku_id, sm.item_name
ORDER BY sell_through_rate_pct DESC;
```

Results (19) Copy Download results CSV

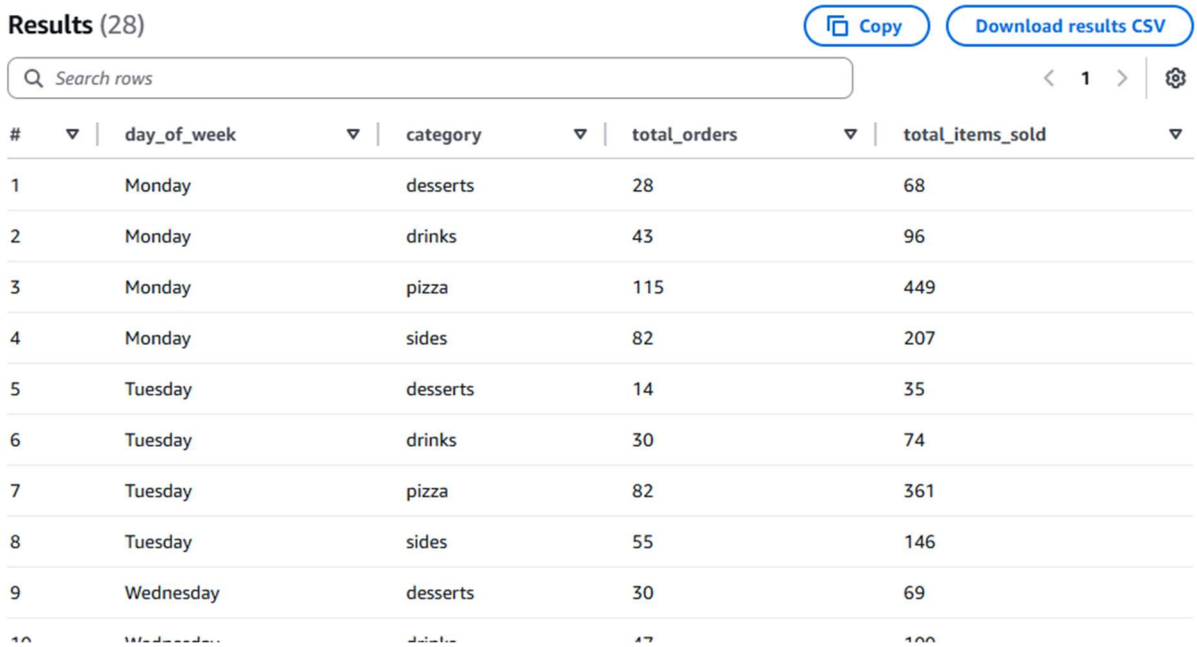
Search rows 1

#	sku_id	item_name	total_sold_quantity	max_recorded_inventory	sell_through_rate_pct
1	SKU0005	paneer tikka pizza	21720	94	23106.38
2	SKU0007	hawaiian pizza	21000	96	21875.0
3	SKU0002	pepperoni pizza	19020	94	20234.04
4	SKU0012	spicy chicken wings	19740	98	20142.86
5	SKU0019	chocolate lava cake	19980	100	19980.0
6	SKU0018	soft drink (orange)	19980	100	19980.0
7	SKU0017	soft drink (cola)	19980	100	19980.0
8	SKU0001	margherita pizza	19920	100	19920.0
9	SKU0016	chicken tenders	19140	99	19333.33

10. Order Trends with Day of Week and Category

```
SELECT
  format_datetime(CAST(o.order_time AS timestamp), 'EEEE') AS day_of_week,
  sm.category,
  COUNT(DISTINCT o.order_id) AS total_orders,
  SUM(oi.quantity) AS total_items_sold
FROM cleaned_orders o
JOIN cleaned_order_items oi ON o.order_id = oi.order_id
JOIN cleaned_sku_master sm ON oi.sku_id = sm.sku_id
GROUP BY
  format_datetime(CAST(o.order_time AS timestamp), 'EEEE'), sm.category
ORDER BY
  CASE format_datetime(CAST(o.order_time AS timestamp), 'EEEE')
    WHEN 'Monday' THEN 1
    WHEN 'Tuesday' THEN 2
    WHEN 'Wednesday' THEN 3
    WHEN 'Thursday' THEN 4
    WHEN 'Friday' THEN 5
    WHEN 'Saturday' THEN 6
    WHEN 'Sunday' THEN 7
    ELSE 8
  END,
```

sm.category;

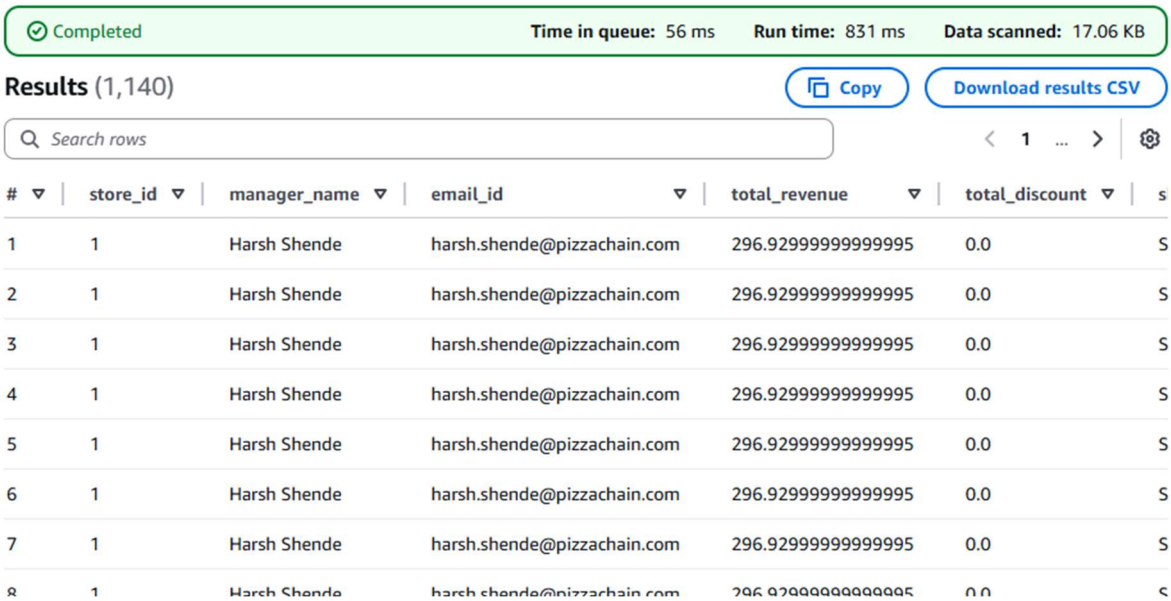


#	day_of_week	category	total_orders	total_items_sold
1	Monday	desserts	28	68
2	Monday	drinks	43	96
3	Monday	pizza	115	449
4	Monday	sides	82	207
5	Tuesday	desserts	14	35
6	Tuesday	drinks	30	74
7	Tuesday	pizza	82	361
8	Tuesday	sides	55	146
9	Wednesday	desserts	30	69

11. Store-wise Revenue, Discount Impact, and Inventory Status (Today)

```
WITH today_orders AS (
    SELECT
        o.order_id,
        CAST(o.store_id AS INT) AS store_id_int,
        o.total_order_value,
        o.total_discount,
        o.order_time
    FROM cleaned_orders o
    WHERE DATE(CAST(SUBSTR(o.order_time,1,10) AS DATE)) = CURRENT_DATE
),

order_items_enriched AS (
    SELECT
        oi.order_id,
        oi.sku_id,
        oi.quantity,
        oi.unit_price,
        oi.item_total,
        sm.category,
        sm.item_name
    FROM cleaned_order_items oi
    LEFT JOIN cleaned_sku_master sm ON oi.sku_id = sm.sku_id
),

store_revenue_discount AS (
    SELECT
        to.store_id_int,
        SUM(to.total_order_value) AS total_revenue,
```

```
        SUM(to.total_discount) AS total_discount
    FROM today_orders to
    GROUP BY to.store_id_int
),

inventory_status AS (
    SELECT
        il.store_id,
        il.sku_id,
        il.stock_qty,
        il.restock_threshold,
        il.item_name,
        il.category
    FROM cleaned_inventory_logs il
),

store_manager_info AS (
    SELECT
        sm.store_id,
        sm.manager_name,
        sm.email_id
    FROM store_manager sm
)

SELECT
    smi.store_id,
    smi.manager_name,
    smi.email_id,
    COALESCE(sr.total_revenue, 0) AS total_revenue,
    COALESCE(sr.total_discount, 0) AS total_discount,
    il.sku_id,
    il.item_name,
    il.category,
    il.stock_qty,
    il.restock_threshold,
    CASE WHEN il.stock_qty <= il.restock_threshold THEN 'LOW STOCK' ELSE 'OK' END AS
stock_status
FROM store_manager_info smi
LEFT JOIN store_revenue_discount sr ON smi.store_id = sr.store_id_int
LEFT JOIN inventory_status il ON smi.store_id = il.store_id
ORDER BY smi.store_id, il.sku_id;
```

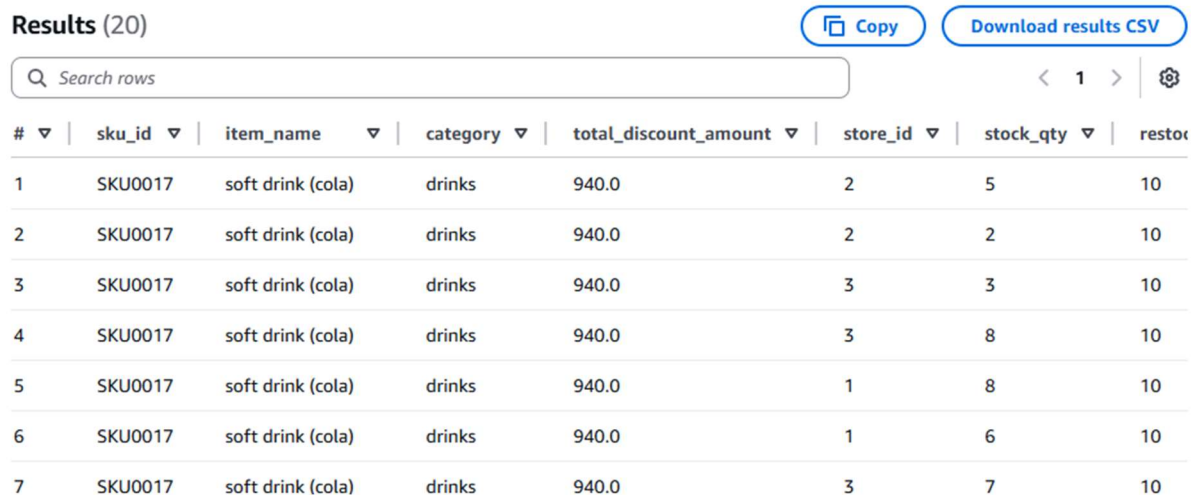


#	store_id	manager_name	email_id	total_revenue	total_discount	s
1	1	Harsh Shende	harsh.shende@pizzachain.com	296.92999999999995	0.0	S
2	1	Harsh Shende	harsh.shende@pizzachain.com	296.92999999999995	0.0	S
3	1	Harsh Shende	harsh.shende@pizzachain.com	296.92999999999995	0.0	S
4	1	Harsh Shende	harsh.shende@pizzachain.com	296.92999999999995	0.0	S
5	1	Harsh Shende	harsh.shende@pizzachain.com	296.92999999999995	0.0	S
6	1	Harsh Shende	harsh.shende@pizzachain.com	296.92999999999995	0.0	S
7	1	Harsh Shende	harsh.shende@pizzachain.com	296.92999999999995	0.0	S
8	1	Harsh Shende	harsh.shende@pizzachain.com	296.92999999999995	0.0	S

12. Top Discounted Products with Inventory Alert (Last 30 Days)

```
WITH recent_discounts AS (
    SELECT
        oi.sku_id,
        sm.item_name,
        sm.category,
        SUM(oi.discount_amount) AS total_discount_amount
    FROM cleaned_order_items oi
    LEFT JOIN cleaned_sku_master sm ON oi.sku_id = sm.sku_id
    LEFT JOIN cleaned_orders o ON oi.order_id = o.order_id
    WHERE DATE(CAST(SUBSTR(o.order_time, 1, 10) AS DATE)) >= CURRENT_DATE -
INTERVAL '30' DAY
    GROUP BY oi.sku_id, sm.item_name, sm.category
),

latest_inventory AS (
    SELECT
        il.sku_id,
        il.store_id,
        il.stock_qty,
        il.restock_threshold,
        il.item_name,
        il.category
    FROM cleaned_inventory_logs il
    -- Assuming latest inventory per SKU per store is the most recent log_time,
    -- if needed you can add filtering to pick the latest record per sku_id, store_id
),

discount_inventory AS (
    SELECT
        rd.sku_id,
```

```
        rd.item_name,
        rd.category,
        rd.total_discount_amount,
        li.store_id,
        li.stock_qty,
        li.restock_threshold,
        CASE
            WHEN li.stock_qty <= li.restock_threshold THEN 'LOW STOCK'
            ELSE 'STOCK OK'
        END AS stock_status
    FROM recent_discounts rd
    LEFT JOIN latest_inventory li ON rd.sku_id = li.sku_id
)

SELECT
    sku_id,
    item_name,
    category,
    total_discount_amount,
    store_id,
    stock_qty,
    restock_threshold,
    stock_status
FROM discount_inventory
WHERE stock_status = 'LOW STOCK'
ORDER BY total_discount_amount DESC
LIMIT 20;
```



Results (20)

Copy | Download results CSV

Search rows

1

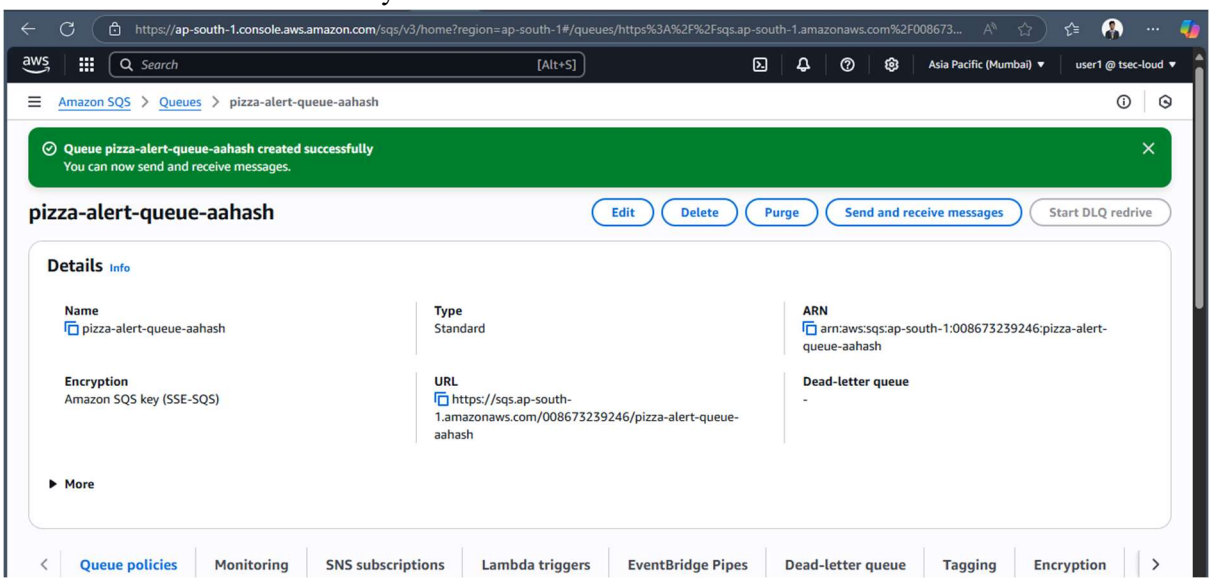
#	sku_id	item_name	category	total_discount_amount	store_id	stock_qty	resto
1	SKU0017	soft drink (cola)	drinks	940.0	2	5	10
2	SKU0017	soft drink (cola)	drinks	940.0	2	2	10
3	SKU0017	soft drink (cola)	drinks	940.0	3	3	10
4	SKU0017	soft drink (cola)	drinks	940.0	3	8	10
5	SKU0017	soft drink (cola)	drinks	940.0	1	8	10
6	SKU0017	soft drink (cola)	drinks	940.0	1	6	10
7	SKU0017	soft drink (cola)	drinks	940.0	3	7	10

Phase 5: Alerting Mechanism via Lambda, SQS, and SNS

This phase implements an automated alerting system using AWS Lambda, Amazon SQS, and Amazon SNS to detect operational issues (like low inventory or high discounts) and notify appropriate channels.

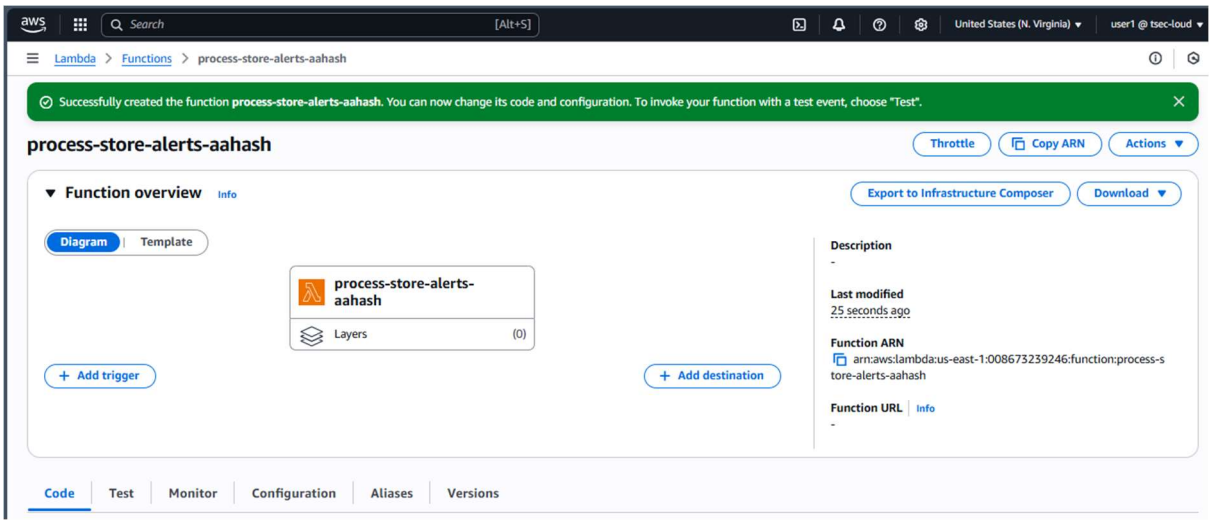
Step 1: Create SQS Queue for Alert Messages

- **Service**: Amazon SQS
- **Queue Name**: pizza-alert-queue-aahash
- **Type**: Standard Queue
- **Configuration**:
    - Default visibility timeout: 30 seconds



Step 2: Create Lambda Function to Run Threshold Queries and Send to SQS

- **Lambda Name**: process-store-alerts
- **Runtime**: Python 3.12
- **Execution Role**: Attach a role with:
    - AthenaFullAccess
    - AmazonSQSFullAccess
    - S3ReadOnlyAccess
    - CloudWatch Logs permissions



Lambda Code:

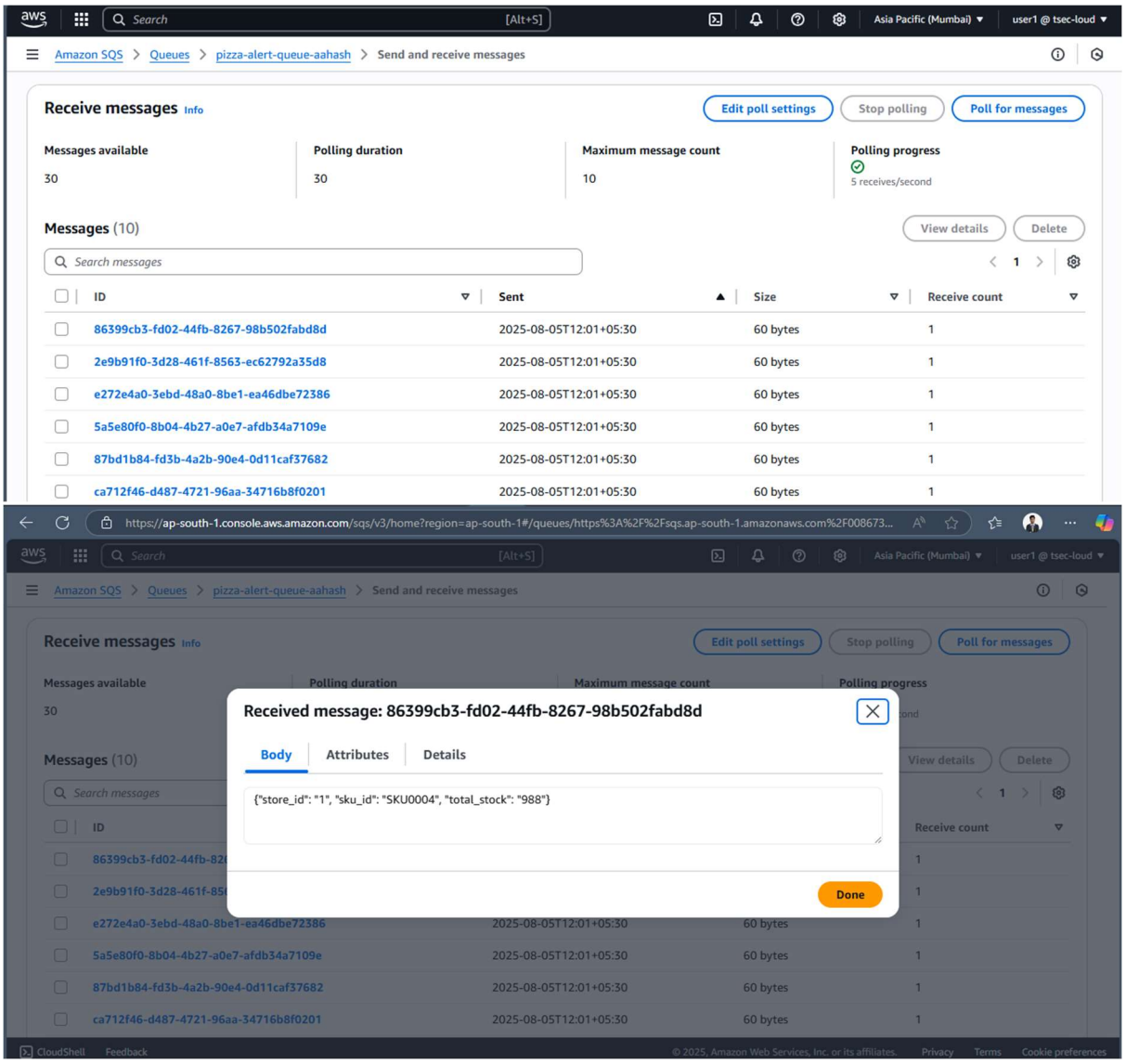
```
import boto3
import time
import json

# Configs
athena_database = "pizza_raw_rds-aahash"
athena_output_bucket = "s3://aahash-project3/Unsaved/2025/08/05/"
query = """
SELECT
  store_id,
  sku_id,
  SUM(stock_qty) AS total_stock
FROM "pizza_raw_rds-aahash"."cleaned_inventory_logs"
GROUP BY store_id, sku_id
HAVING SUM(stock_qty) < 1000;
"""
region = "ap-south-1"
sqs_queue_url = "https://sqs.ap-south-1.amazonaws.com/008673239246/pizza-alert-queue-aahash"

# Clients
athena_client = boto3.client('athena', region_name=region)
sqs_client = boto3.client('sqs', region_name=region)

def lambda_handler(event, context):
    # Step 1: Start Athena Query Execution
    response = athena_client.start_query_execution(
        QueryString=query,
        QueryExecutionContext={'Database': athena_database},
        ResultConfiguration={'OutputLocation': athena_output_bucket}
    )
    query_execution_id = response['QueryExecutionId']

    # Step 2: Wait for query to finish
    while True:
        status = athena_client.get_query_execution(QueryExecutionId=query_execution_id)
        state = status['QueryExecution']['Status']['State']
```

```
    if state in ['SUCCEEDED', 'FAILED', 'CANCELLED']:
        break
    time.sleep(2)

if state != 'SUCCEEDED':
    raise Exception(f"Athena query failed with state: {state}")

# Step 3: Fetch results
results = athena_client.get_query_results(QueryExecutionId=query_execution_id)

alerts = []
for row in results['ResultSet']['Rows'][1:]:  # Skip header row
    store_id = row['Data'][0]['VarCharValue']
    sku_id = row['Data'][1]['VarCharValue']
    total_stock = row['Data'][2]['VarCharValue']

    alert = {
        "store_id": store_id,
        "sku_id": sku_id,
        "total_stock": total_stock
    }
    alerts.append(alert)

    # Step 4: Send each alert to SQS
    sqs_client.send_message(
        QueueUrl=sqs_queue_url,
        MessageBody=json.dumps(alert)
    )

return {
    "statusCode": 200,
    "body": f"Successfully sent {len(alerts)} inventory alerts to SQS."
}
```

lambda_function.py to handle Athena async query results and send qualifying store_id + sku_id pairs to SQS.

Output:

- Each low-stock entry (below 1000) from the query will be pushed as a message to the SQS queue.



Result: All low-stock records are successfully posted to the SQS queue.

Phase 6: Send SNS Notification from EC2 for Selected Stores

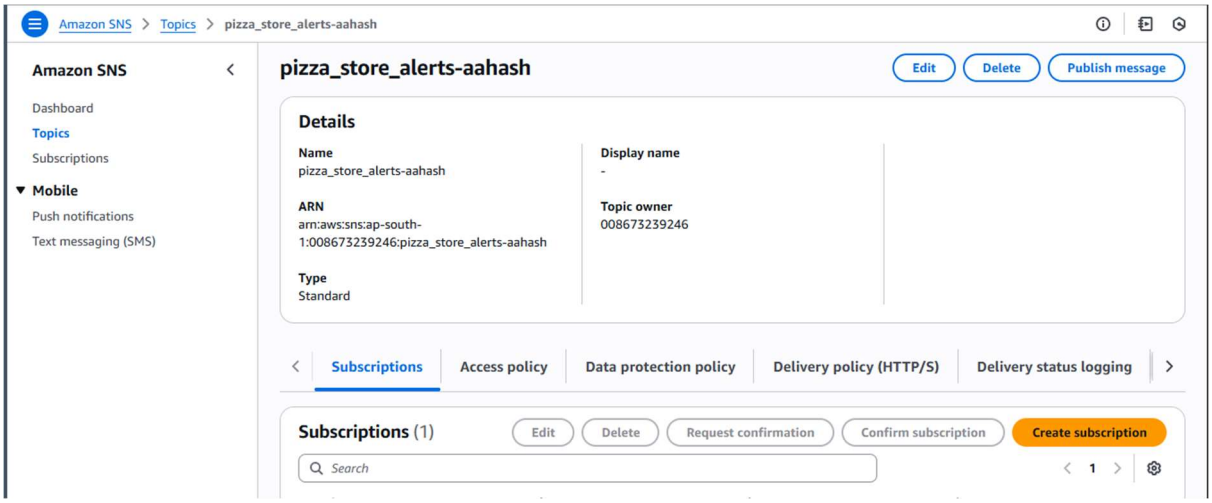
Create Ec2 Instance

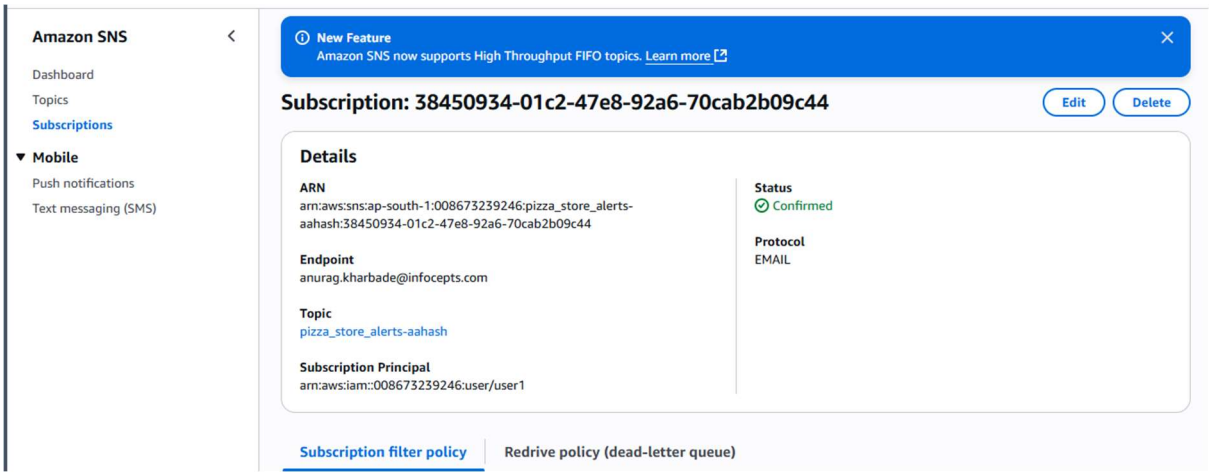
Ensure the EC2 instance has:

- An IAM role attached with AmazonSNSFullAccess permission
- Python 3 installed
- Boto3 installed:

pip install boto3

Create an SNS Topic and Subscription





Python Script to Send Alert from EC2

send_sns_alert.py that:

- Reads a message from SQS
- Looks up store manager email from RDS MySQL
- Sends an SNS message to notify the store

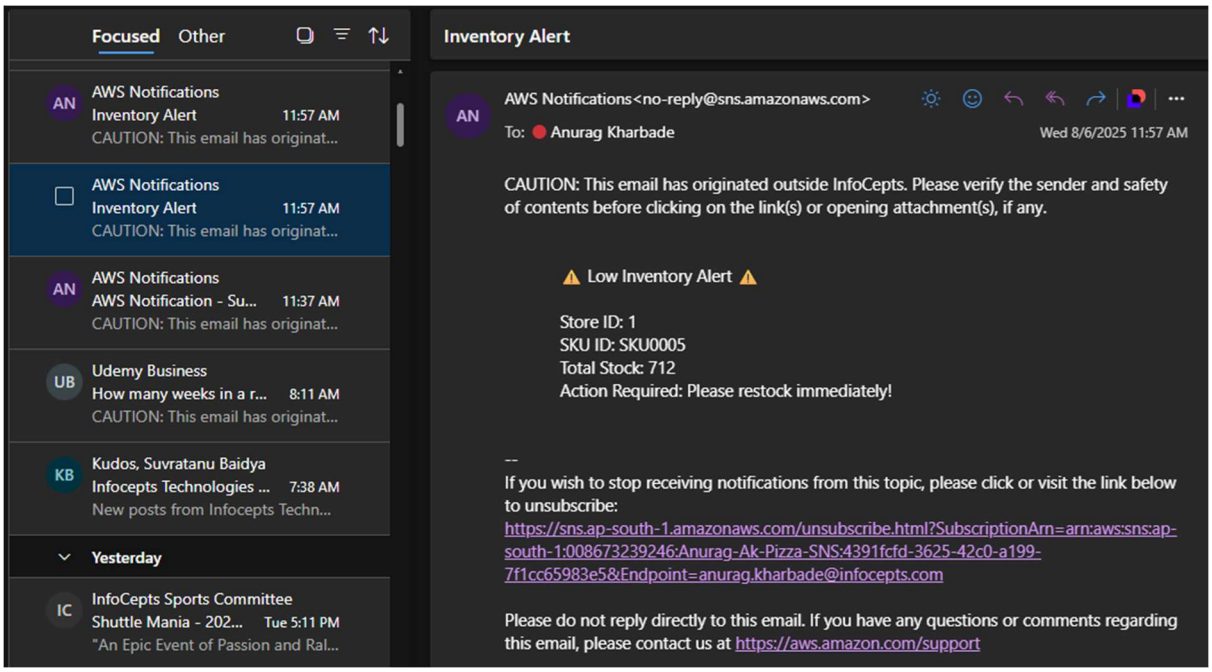
import boto3

```
import json
import mysql.connector

sqs = boto3.client('sqs')
sns = boto3.client('sns')
```

```
queue_url = 'https://sqs.ap-south-1.amazonaws.com/008673239246/pizza-alert-queue-aahash'

# Connect to RDS (Replace with your config)
conn = mysql.connector.connect(
    host='pizzadb-aahash.c7ok6ko8wnwq.ap-south-1.rds.amazonaws.com',
    user='admin',
    password='Aahash123',
    database='pizza_db_aahash'
)
cursor = conn.cursor(dictionary=True)

def fetch_store_info(store_id):
    cursor.execute("SELECT store_id, store_email FROM store_manager WHERE store_id = %s", (store_id,))
    return cursor.fetchone()

def send_sns(email, message):
    sns.publish(
        TopicArn='arn:aws:sns:ap-south-1:008673239246:pizza_store_alerts-aahash:04e8b4d8-6fe3-4fae-a434-
5504baefc579',
        Message=message,
        Subject=' Inventory Alert'
    )

while True:
    messages = sqs.receive_message(QueueUrl=queue_url, MaxNumberOfMessages=5, WaitTimeSeconds=10)
    if 'Messages' not in messages:
        continue
    for msg in messages['Messages']:
        body = json.loads(msg['Body'])
        store_info = fetch_store_info(body['store_id'])

        if store_info:
            alert = f"""
            Store ID: {store_info['store_id']}
            SKU: {body['sku_name']} ({body['sku_id']})
            Quantity: {body['quantity']}
            Action Required: Restock immediately!
            """
            send_sns(store_info['store_email'], alert)

        # Delete message from queue
        sqs.delete_message(QueueUrl=queue_url, ReceiptHandle=msg['ReceiptHandle'])
```

Run the script: **python3 send_sns_alert.py**

Check the email:


Focused
Other
AWS Notifications
Inventory Alert 11:57 AM
CAUTION: This email has originat...
AWS Notifications
Inventory Alert 11:57 AM
CAUTION: This email has originat...
AWS Notifications
AWS Notification - Su... 11:37 AM
CAUTION: This email has originat...
Udemy Business
How many weeks in a r... 8:11 AM
CAUTION: This email has originat...
Kudos, Suvratanu Baidya
Infocepts Technologies ... 7:38 AM
New posts from Infocepts Techn...
Yesterday
InfoCepts Sports Committee
Shuttle Mania - 202... Tue 5:11 PM
"An Epic Event of Passion and Ral...
Inventory Alert
AWS Notifications<no-reply@sns.amazonaws.com>
To: Anurag Kharbade
Wed 8/6/2025 11:57 AM
CAUTION: This email has originated outside InfoCepts. Please verify the sender and safety of contents before clicking on the link(s) or opening attachment(s), if any.
⚠ Low Inventory Alert ⚠
Store ID: 1
SKU ID: SKU0005
Total Stock: 712
Action Required: Please restock immediately!
--
If you wish to stop receiving notifications from this topic, please click or visit the link below to unsubscribe:
https://sns.ap-south-1.amazonaws.com/unsubscribe.html?SubscriptionArn=arn:aws:sns:ap-south-1:008673239246:Anurag-Ak-Pizza-SNS:4391fcfd-3625-42c0-a199-7f1cc65983e5&Endpoint=anurag.kharbade@infocepts.com
Please do not reply directly to this email. If you have any questions or comments regarding this email, please contact us at https://aws.amazon.com/support