Serverless Data Pipeline

Data Pipeline :

The purpose of this practical is to design and implement a data pipeline on a server that can collect data from different sources, process it, transform it, and finally load it into a storage system such as a database, data warehouse, or cloud storage. This involves using server-side scripting, automation, logging, and error-handling techniques.

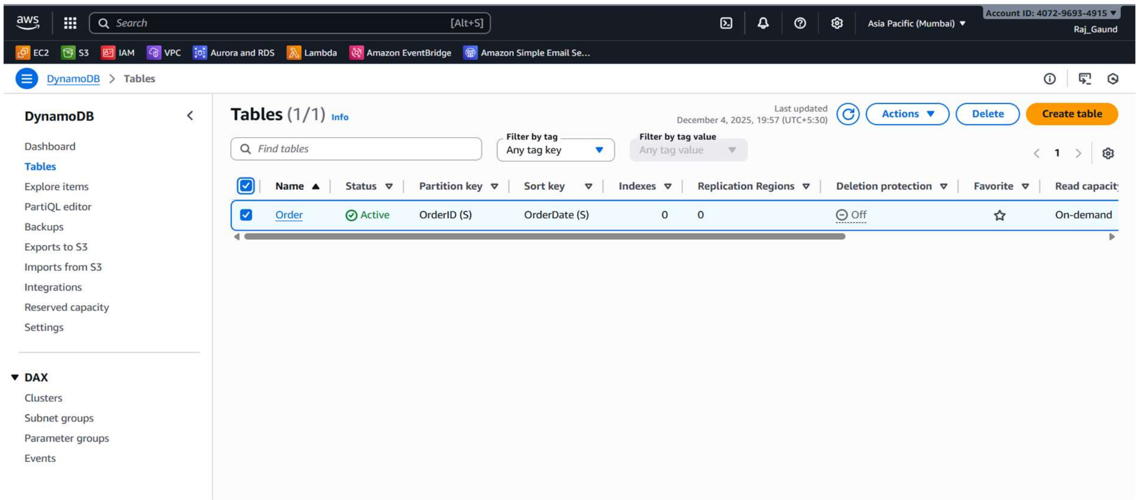
Key Steps of the Practical -

- Store orders in **DynamoDB**
- Use **Glue** to crawl + transform data
- Store cleaned data in **S3 (Data Lake)**
- Use **Athena** to run SQL queries on the data

Steps for Practical

Step 1 : Create DynamoDB Table

- Open DynamoDB → Create Table
- Table name: Orders
- Partition Key: OrderID (String)
- Sort Key: OrderDate (String) → Create Table



Tables (1/1) Info

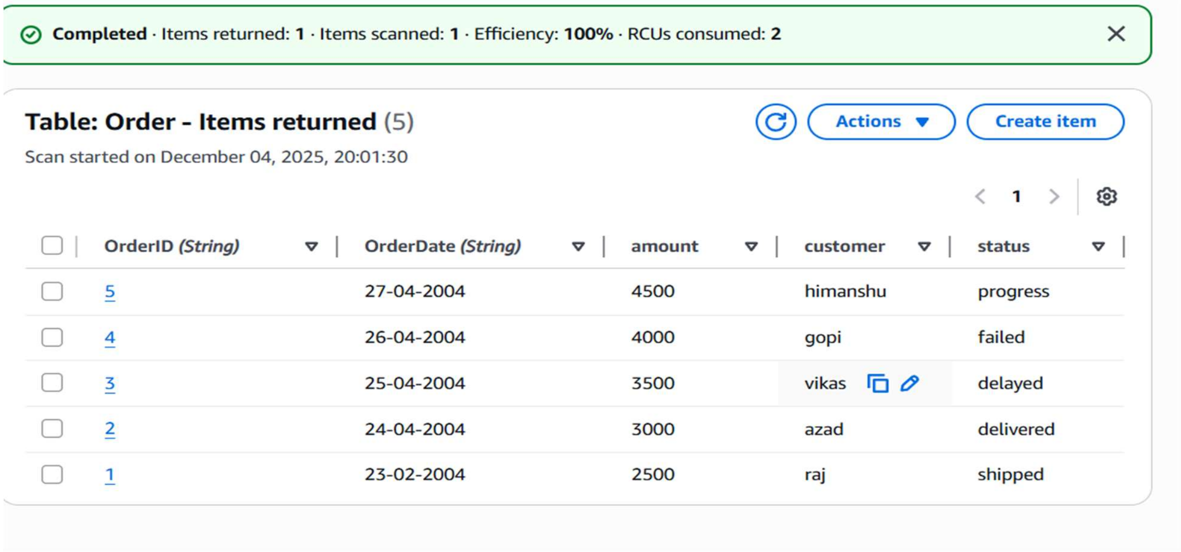
Last updated December 4, 2025, 19:57 (UTC+5:30)

☑	Name	Status	Partition key	Sort key	Indexes	Replication Regions	Deletion protection	Favorite	Read capacit
☑	Order	Active	OrderID (S)	OrderDate (S)	0	0	Off	☆	On-demand

A new DynamoDB table is created to store order records.

Step 2: Insert Sample Data

- Click on Table (Order) → Explore Item
- Create Item → Add Attribute and Data → Create
- Add at Least five items



Completed · Items returned: **1** · Items scanned: **1** · Efficiency: **100%** · RCUs consumed: **2**

Table: Order - Items returned (5)

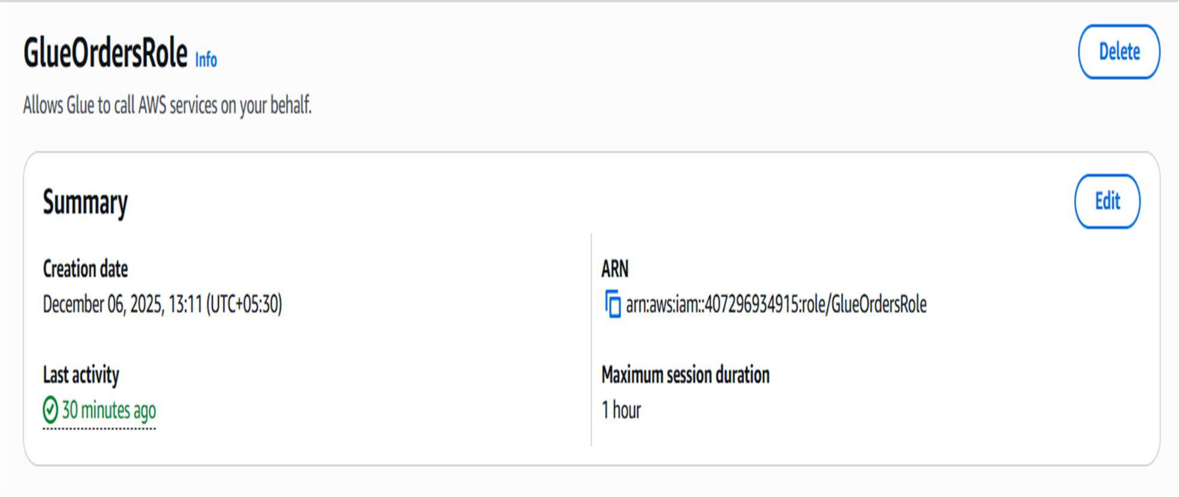
Scan started on December 04, 2025, 20:01:30

□	OrderID *(String)*	OrderDate *(String)*	amount	customer	status
□	5	27-04-2004	4500	himanshu	progress
□	4	26-04-2004	4000	gopi	failed
□	3	25-04-2004	3500	vikas	delayed
□	2	24-04-2004	3000	azad	delivered
□	1	23-02-2004	2500	raj	shipped

DynamoDB now contains real order data that the pipeline will process.

Step 3: Create Glue Crawler

- AWS Console → **Glue** → Data Catalouge
- Create Crawler → Enter Crawler Name (OrderCrawler)
- Add Datasource → Data source : **DynamoDB**
- Table name : DynamoDB Table Name (**Order**) → Add
- Choose IAM Role : Create New IAM role
- AWS → IAM → Role → Create → Name : GlueCrawlerRole
- Trusted Entity : AWS service → Use case : **Glue**
- Add Policies / Permission : **AWSGlueServiceRole** , **AmazonDynamoDBReadOnlyAccess , AmazonS3FullAccess**
- Create IAM Role
- Add Database : **ordersdb** → OnDemand → Create Crawler



Allows Glue to call AWS services on your behalf.

Summary

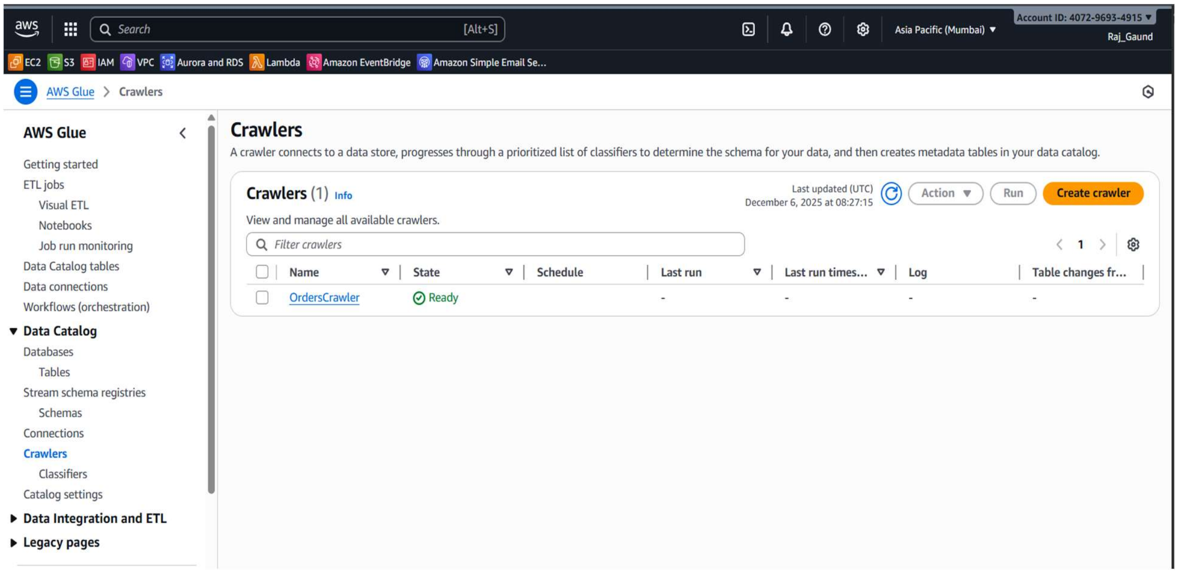
Creation date
December 06, 2025, 13:11 (UTC+05:30)

Last activity
30 minutes ago

ARN
arn:aws:iam::407296934915:role/GlueOrdersRole

Maximum session duration
1 hour

IAM Role Created Successfully



Crawlers

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

Crawlers (1) Info

Last updated (UTC)
December 6, 2025 at 08:27:15

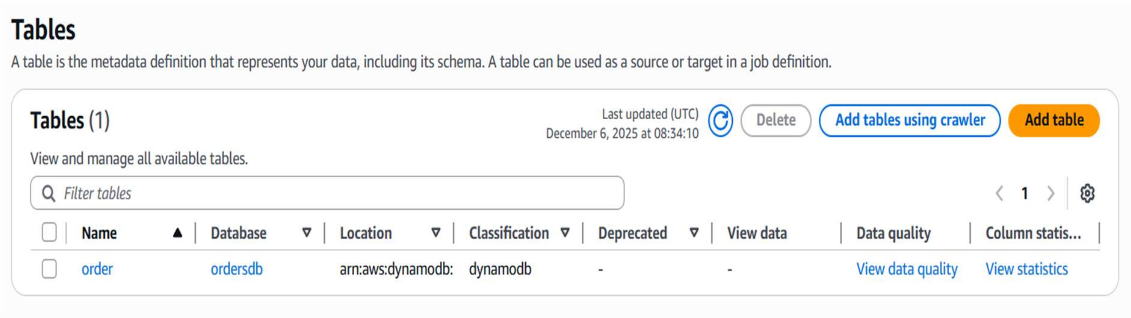
View and manage all available crawlers.

	Name	State	Schedule	Last run	Last run times...	Log	Table changes fr...
□	OrdersCrawler	✓ Ready		-	-	-	-

Crawler create Successfully

Step 4 : Run The Crawler

- AWS → Glue → Crawler
- Click Crawler → **Run Crawler**
- AWS → Glue → Table



Tables

A table is the metadata definition that represents your data, including its schema. A table can be used as a source or target in a job definition.

Tables (1)

Last updated (UTC)
December 6, 2025 at 08:34:10

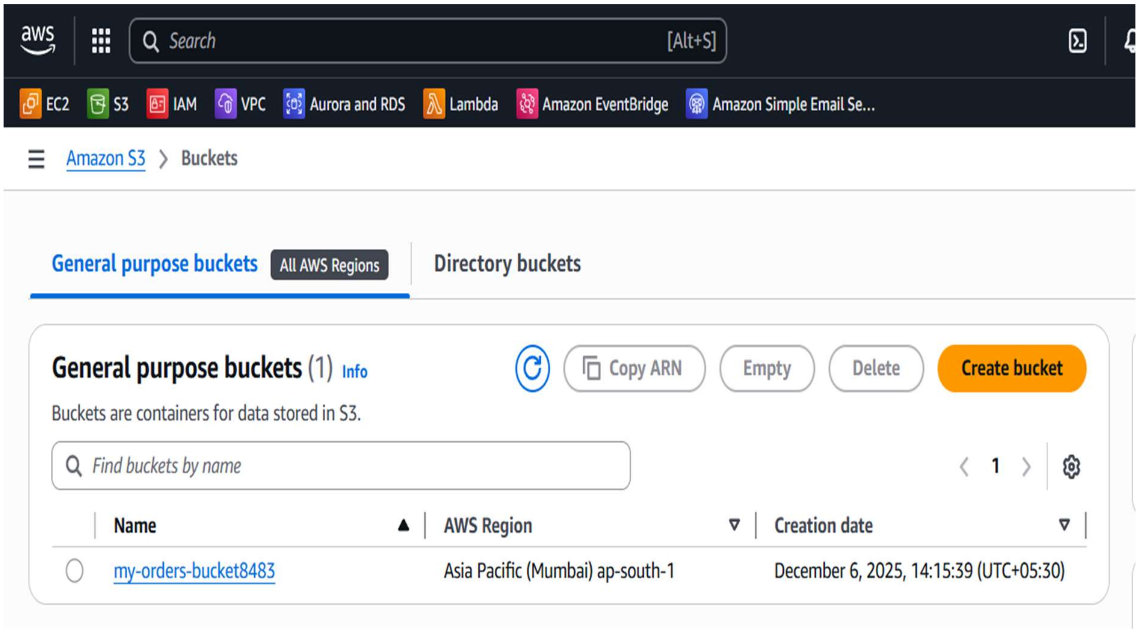
View and manage all available tables.

	Name	Database	Location	Classification	Deprecated	View data	Data quality	Column statis...
□	order	ordersdb	arn:aws:dynamodb:	dynamodb	-	-	View data quality	View statistics

(Table created automatic when you run the crawler)

Step 5 : Create S3 bucket (for ETL output)

- AWS Console → **S3** → General Purpose Bucket
- Bucket Name : you can enter own (my-orders-bucket123)
- Create Bucket



General purpose buckets (1) Info

Buckets are containers for data stored in S3.

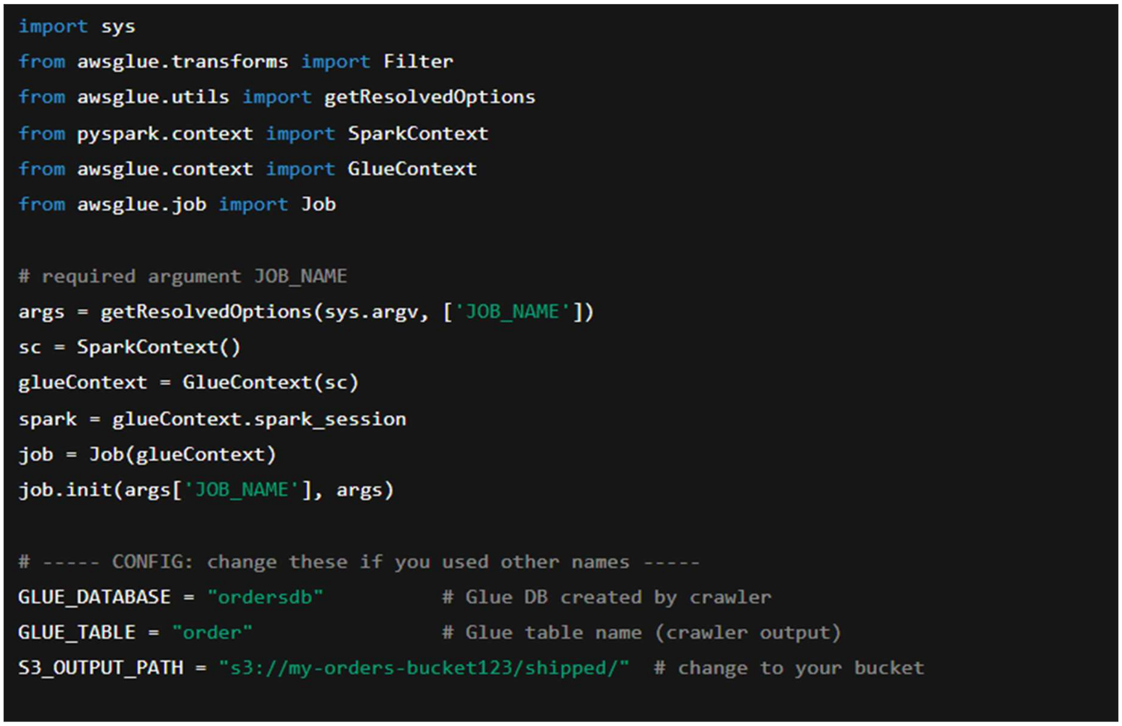
Name	AWS Region	Creation date
○ my-orders-bucket8483	Asia Pacific (Mumbai) ap-south-1	December 6, 2025, 14:15:39 (UTC+05:30)

Bucket used to stored Job ETL output

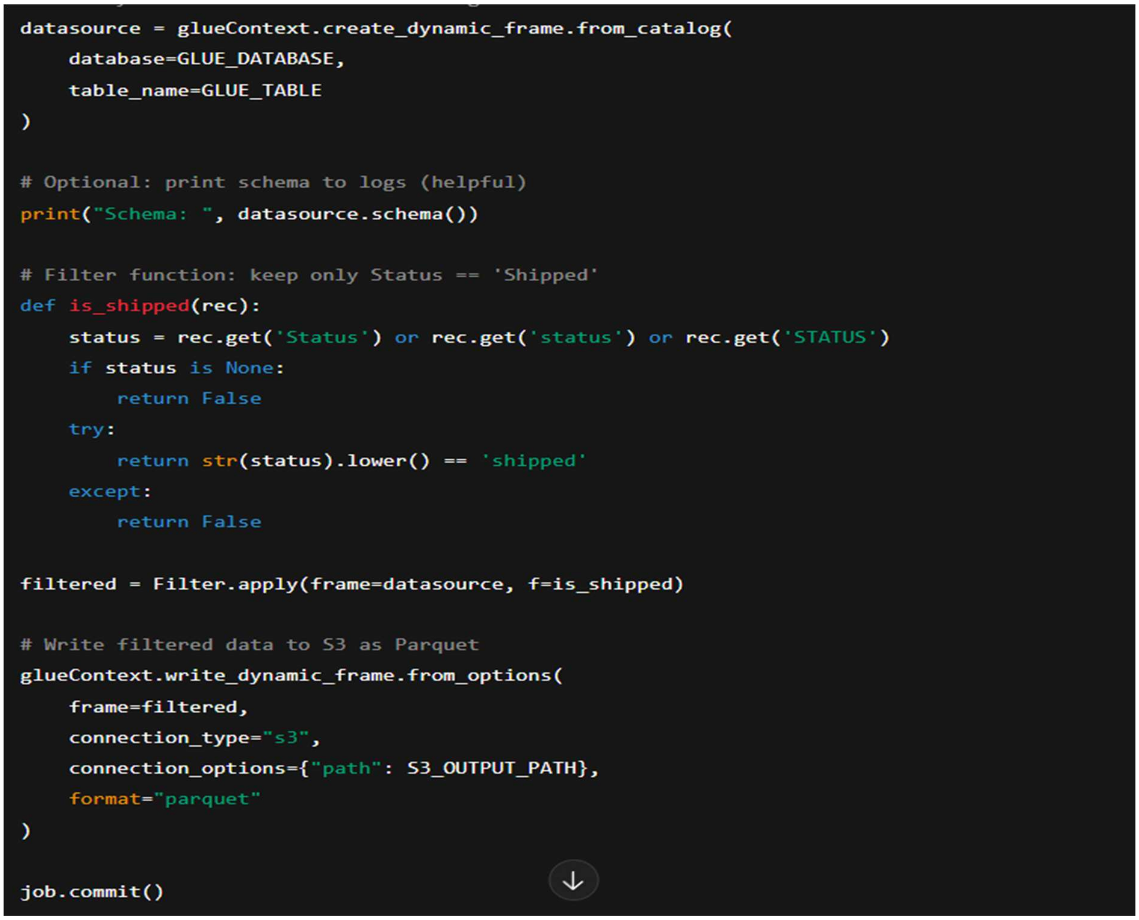
Step 6 : Glue ETL Job create

- AWS → Glue → ETL jobs → Script editor
- Engine : Spark → Create script
- Type this code

Code :

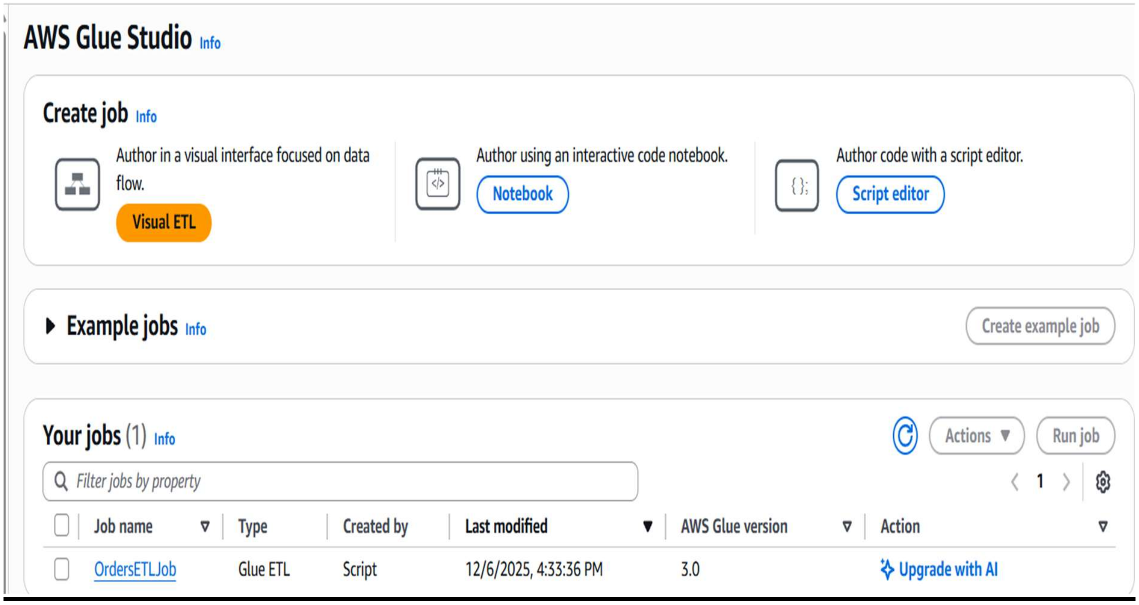


```
import sys
from awsglue.transforms import Filter
from awsglue.utils import getResolvedOptions
from pyspark.context import SparkContext
from awsglue.context import GlueContext
from awsglue.job import Job

# required argument JOB_NAME
args = getResolvedOptions(sys.argv, ['JOB_NAME'])
sc = SparkContext()
glueContext = GlueContext(sc)
spark = glueContext.spark_session
job = Job(glueContext)
job.init(args['JOB_NAME'], args)

# ----- CONFIG: change these if you used other names -----
GLUE_DATABASE = "ordersdb"          # Glue DB created by crawler
GLUE_TABLE = "order"                # Glue table name (crawler output)
S3_OUTPUT_PATH = "s3://my-orders-bucket123/shipped/"  # change to your bucket
```



```
datasource = glueContext.create_dynamic_frame.from_catalog(
    database=GLUE_DATABASE,
    table_name=GLUE_TABLE
)

# Optional: print schema to logs (helpful)
print("Schema: ", datasource.schema())

# Filter function: keep only Status == 'Shipped'
def is_shipped(rec):
    status = rec.get('Status') or rec.get('status') or rec.get('STATUS')
    if status is None:
        return False
    try:
        return str(status).lower() == 'shipped'
    except:
        return False

filtered = Filter.apply(frame=datasource, f=is_shipped)

# Write filtered data to S3 as Parquet
glueContext.write_dynamic_frame.from_options(
    frame=filtered,
    connection_type="s3",
    connection_options={"path": S3_OUTPUT_PATH},
    format="parquet"
)

job.commit()
```

- Jobs details → Enter name : OrdersETLJob
- IAM Role : Choose existing (GlueOrdersRole)
- Glue version: Glue 3.0 (default) / Engine = Spark (keep)
- Language : Python → Save → Run Job
- Click on OrdersETLJob → job details → Run



AWS Glue Studio Info

Create job Info

Author in a visual interface focused on data flow. Visual ETL

Author using an interactive code notebook. Notebook

Author code with a script editor. Script editor

Example jobs Info

Create example job

Your jobs (1) Info

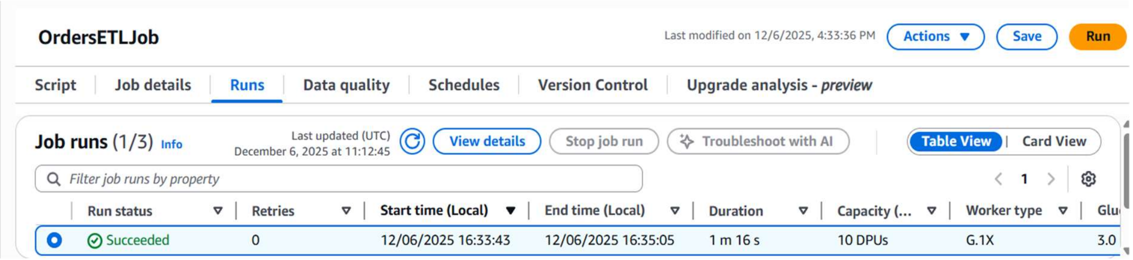
Actions ▼ Run job

Filter jobs by property

□	Job name	Type	Created by	Last modified	AWS Glue version	Action
□	OrdersETLJob	Glue ETL	Script	12/6/2025, 4:33:36 PM	3.0	Upgrade with AI

If succeed show means

- Script is correct
- DynamoDB read that Data
- Filter is successfully applied



OrdersETLJob

Last modified on 12/6/2025, 4:33:36 PM

Actions ▼ Save Run

Script | Job details | Runs | Data quality | Schedules | Version Control | Upgrade analysis - *preview*

Job runs (1/3) Info

Last updated (UTC)
December 6, 2025 at 11:12:45

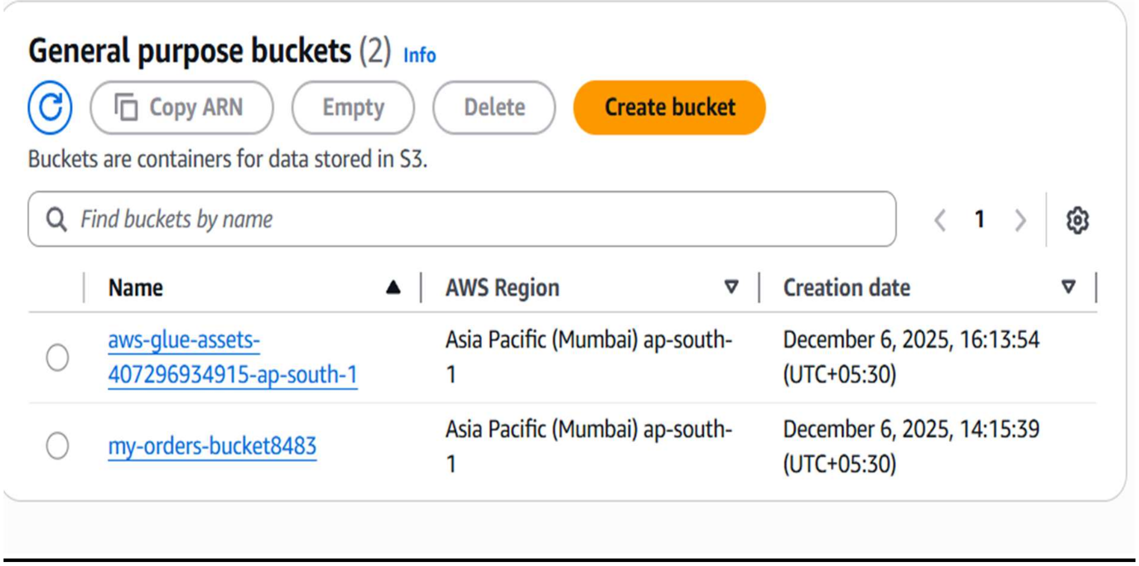
View details Stop job run Troubleshoot with AI Table View | Card View

Filter job runs by property

	Run status	Retries	Start time (Local)	End time (Local)	Duration	Capacity (...	Worker type	Glu
●	✓ Succeeded	0	12/06/2025 16:33:43	12/06/2025 16:35:05	1 m 16 s	10 DPUs	G.1X	3.0

Step 7 : Open S3 bucket

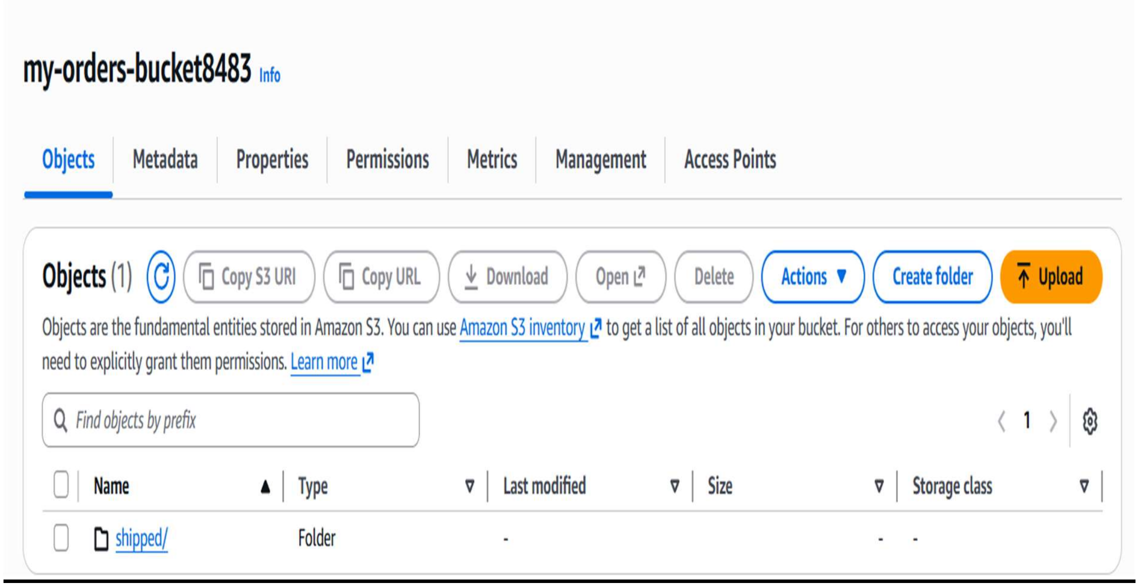
- AWS → S3 → Open Bucket
- Example : (my-orders-bucket8483)



Buckets are containers for data stored in S3.

	Name	AWS Region	Creation date
○	aws-glue-assets-407296934915-ap-south-1	Asia Pacific (Mumbai) ap-south-1	December 6, 2025, 16:13:54 (UTC+05:30)
○	my-orders-bucket8483	Asia Pacific (Mumbai) ap-south-1	December 6, 2025, 14:15:39 (UTC+05:30)

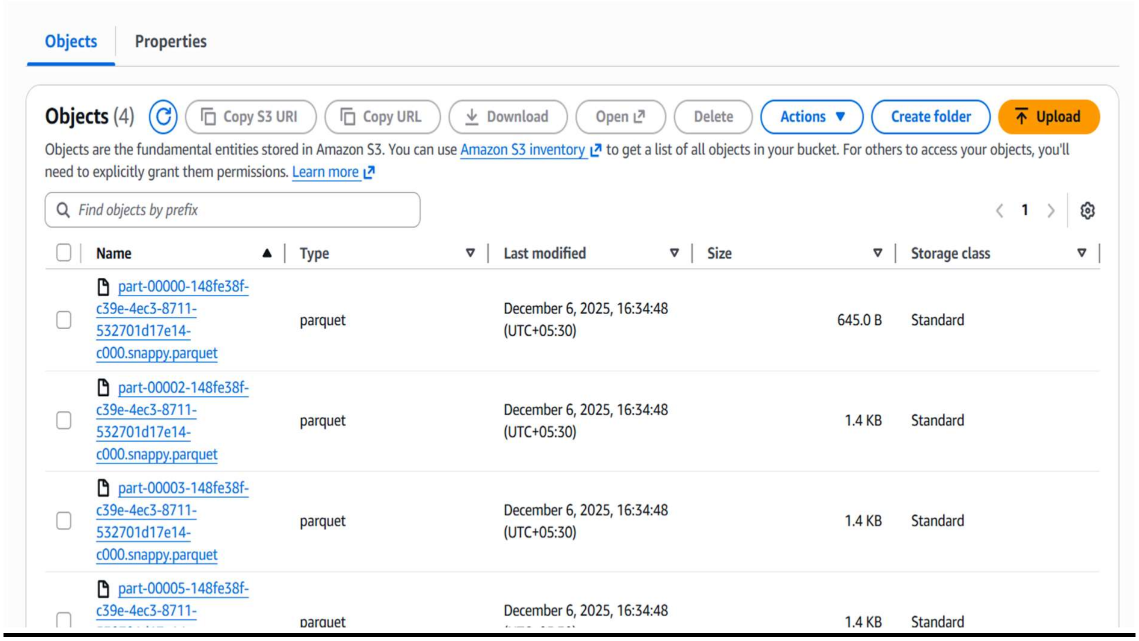
- Clicked shipped named folder



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
□	shipped/	Folder	-	-	-

- Under shipped folder list of four items can show



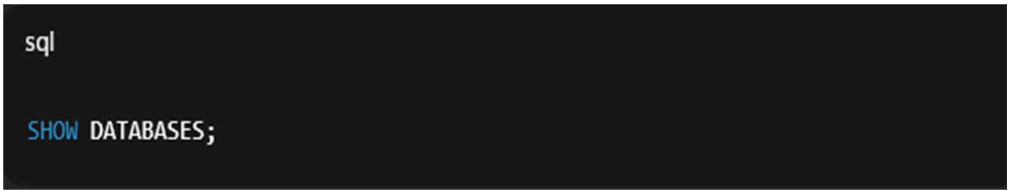
Objects are the fundamental entities stored in Amazon S3. You can use Amazon S3 inventory to get a list of all objects in your bucket. For others to access your objects, you'll need to explicitly grant them permissions. Learn more

Name	Type	Last modified	Size	Storage class
part-00000-148fe38f-c39e-4ec3-8711-532701d17e14-c000.snappy.parquet	parquet	December 6, 2025, 16:34:48 (UTC+05:30)	645.0 B	Standard
part-00002-148fe38f-c39e-4ec3-8711-532701d17e14-c000.snappy.parquet	parquet	December 6, 2025, 16:34:48 (UTC+05:30)	1.4 KB	Standard
part-00003-148fe38f-c39e-4ec3-8711-532701d17e14-c000.snappy.parquet	parquet	December 6, 2025, 16:34:48 (UTC+05:30)	1.4 KB	Standard
part-00005-148fe38f-c39e-4ec3-8711-	parquet	December 6, 2025, 16:34:48	1.4 KB	Standard

Step 8 : Athena Setup to run query

- AWS → Athena
- Select : Query your data in Athena console
- Launch Query editor
- Click on setting → edit your S3 location
- **s3://my-orders-bucket123/athena-results/** → Save
- perform query to create stored and view

1) Query :



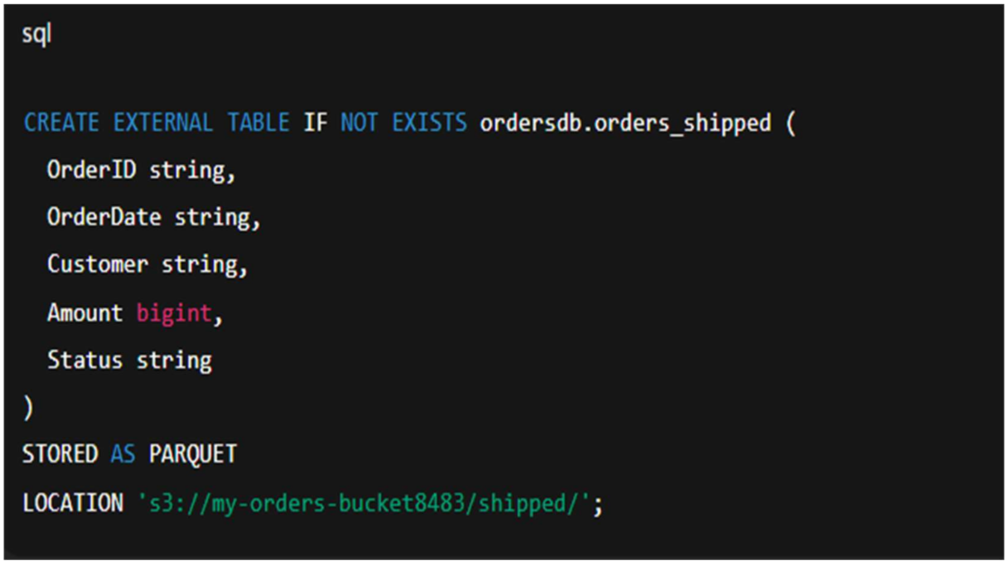
sql

```sql
SHOW DATABASES;
```

Its shows : **ordersdb**

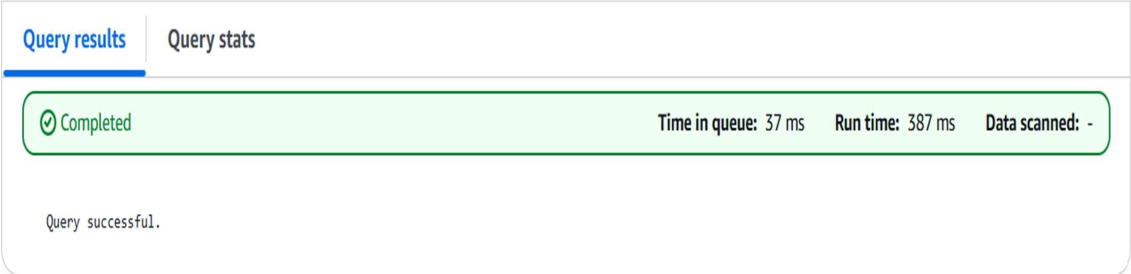
- left side Database → select **ordersdb** into dropdown

2) Run this query to create table :



sql

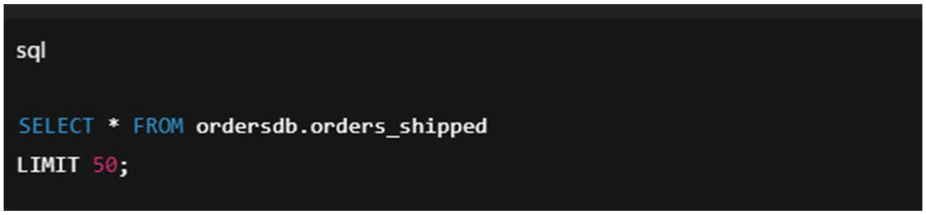
```sql
CREATE EXTERNAL TABLE IF NOT EXISTS ordersdb.orders_shipped (
  OrderID string,
  OrderDate string,
  Customer string,
  Amount bigint,
  Status string
)
STORED AS PARQUET
LOCATION 's3://my-orders-bucket8483/shipped/';
```



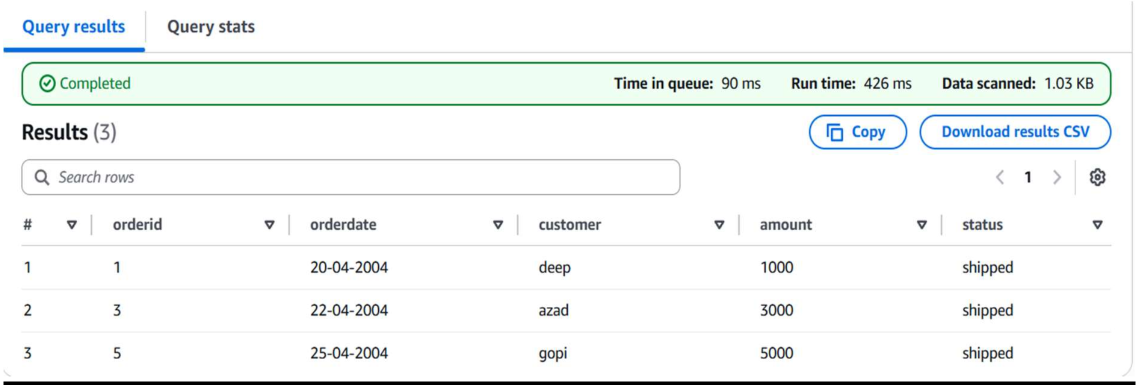
Completed   Time in queue: 37 ms   Run time: 387 ms   Data scanned: -

```
Query successful.
```

3) Query To view or fetch data :



```sql
SELECT * FROM ordersdb.orders_shipped
LIMIT 50;
```

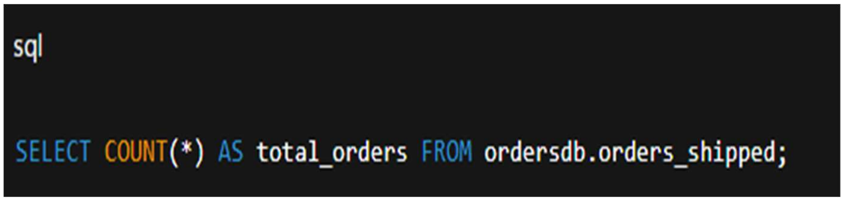


Completed Time in queue: 90 ms Run time: 426 ms Data scanned: 1.03 KB

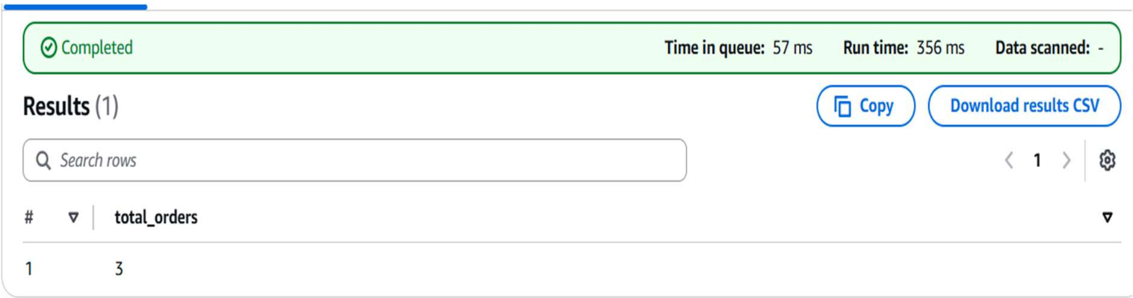
Results (3)

#	orderid	orderdate	customer	amount	status
1	1	20-04-2004	deep	1000	shipped
2	3	22-04-2004	azad	3000	shipped
3	5	25-04-2004	gopi	5000	shipped

4) Query To rows count



```sql
SELECT COUNT(*) AS total_orders FROM ordersdb.orders_shipped;
```

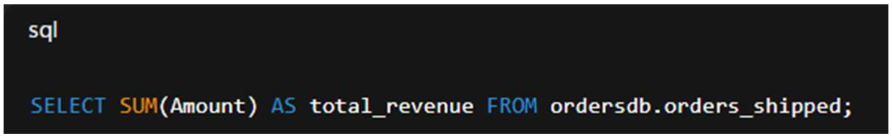


Completed Time in queue: 57 ms Run time: 356 ms Data scanned: -

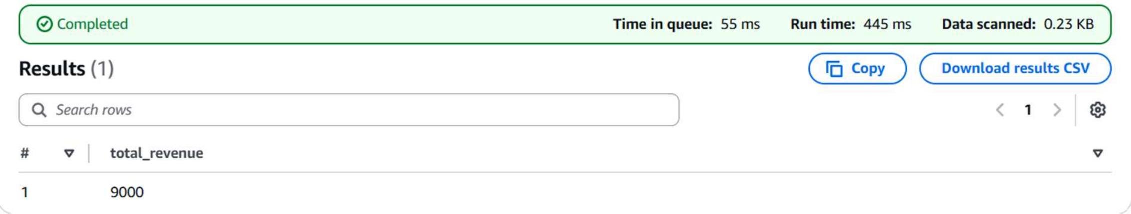
Results (1)

#	total_orders
1	3

5) Query to Total amount (sum) overall



```
sql

SELECT SUM(Amount) AS total_revenue FROM ordersdb.orders_shipped;
```

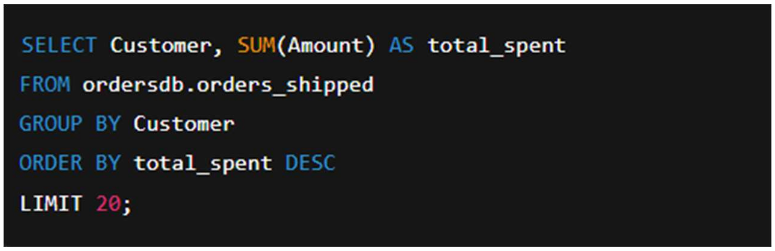


Completed Time in queue: 55 ms Run time: 445 ms Data scanned: 0.23 KB

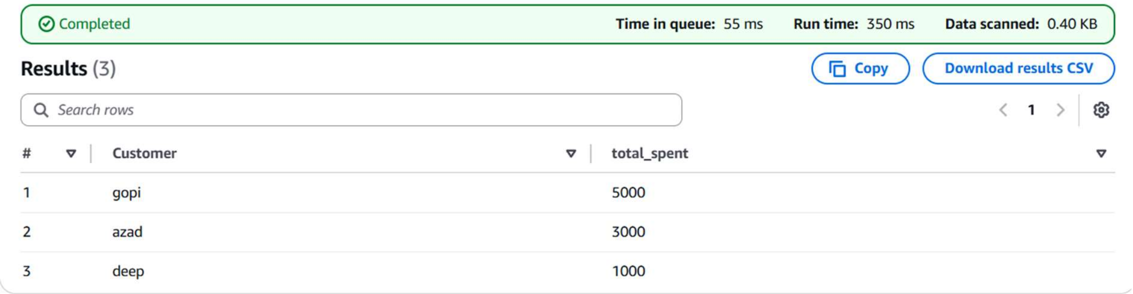
Results (1)

#	total_revenue
1	9000

6) Revenue by customer (top customers)



```
SELECT Customer, SUM(Amount) AS total_spent
FROM ordersdb.orders_shipped
GROUP BY Customer
ORDER BY total_spent DESC
LIMIT 20;
```

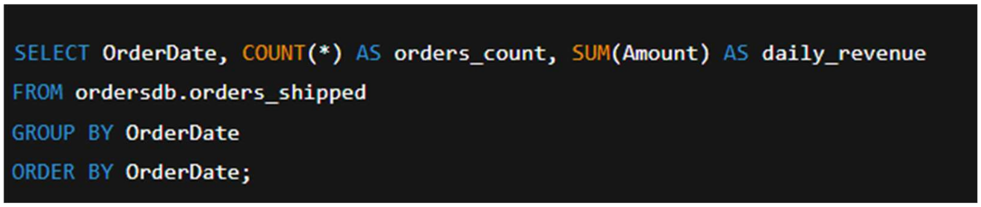


Completed Time in queue: 55 ms Run time: 350 ms Data scanned: 0.40 KB

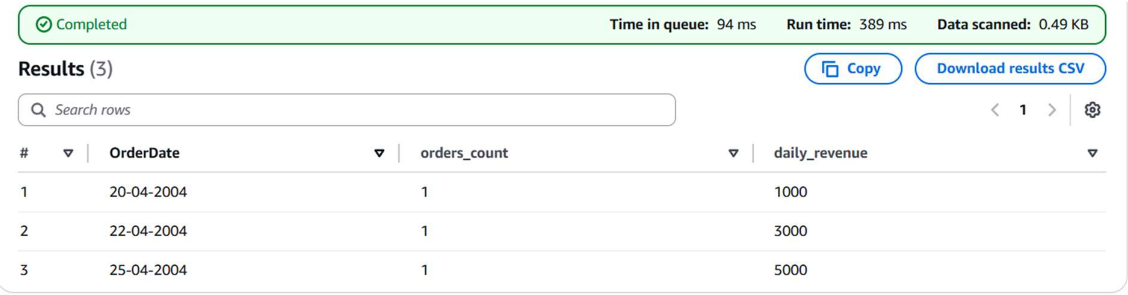
Results (3)

#	Customer	total_spent
1	gopi	5000
2	azad	3000
3	deep	1000

7) Orders per day (if OrderDate is YYYY-MM-DD)



```
SELECT OrderDate, COUNT(*) AS orders_count, SUM(Amount) AS daily_revenue
FROM ordersdb.orders_shipped
GROUP BY OrderDate
ORDER BY OrderDate;
```

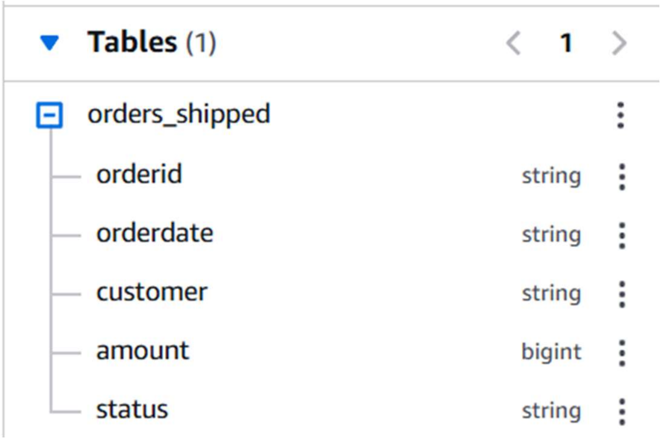


Completed   Time in queue: 94 ms   Run time: 389 ms   Data scanned: 0.49 KB

Results (3)

#	OrderDate	orders_count	daily_revenue
1	20-04-2004	1	1000
2	22-04-2004	1	3000
3	25-04-2004	1	5000

Final output :



Tables (1)

orders_shipped

orderid	string
orderdate	string
customer	string
amount	bigint
status	string

- **Purpose :**

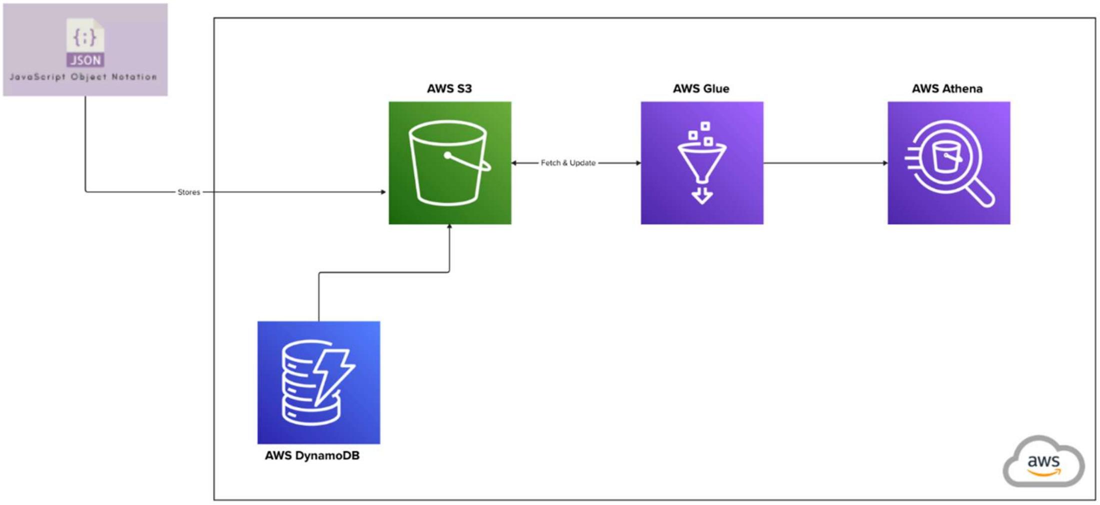
  The purpose of this project is to build a fully automated and serverless data processing pipeline using AWS DynamoDB, AWS Glue, Amazon S3, and Amazon Athena. The aim is to extract raw order data from DynamoDB, automatically detect and catalog its schema using a Glue Crawler, transform and clean the data through a Glue ETL job, and store the optimized output in S3 using the Parquet format.

- **Conclusion :**

In conclusion, this project successfully demonstrates how AWS Glue and Amazon Athena can work together to create an end-to-end serverless data analytics solution. The pipeline extracted data from DynamoDB, transformed it using a Glue ETL job, stored it in an optimized Parquet format in S3, and allowed quick querying using Athena.

- ❖ Architecture Diagram :

Serverless Data Pipeline (DynamoDB , Glue, S3 , Athena)


{;}
JSON
JavaScript Object Notation
Stores
AWS S3
Fetch & Update
AWS Glue
AWS Athena
AWS DynamoDB
aws