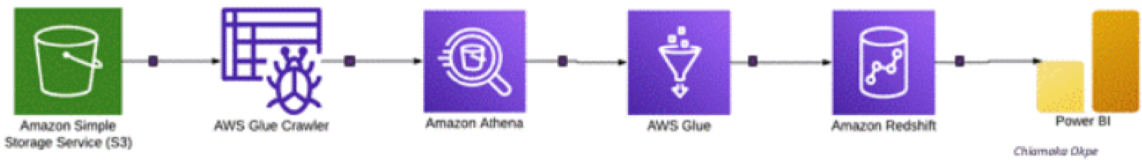
I worked on an end-to-end healthcare data engineering pipeline using AWS services to simulate a real-world scenario. I started by downloading a synthetic healthcare dataset from Kaggle, which contained patient demographics, medical history, hospital details, doctor, insurance provider, medication, admission types, billing amount, and test results. I manually uploaded this CSV dataset to Amazon S3, which served as my raw data storage layer. From there, I created an AWS Glue Crawler to scan the S3 folder and populate the Glue Data Catalog. This automatically inferred the schema from the CSV and allowed me to query the raw data using Amazon Athena. Athena converted the CSV into structured, queryable tables, which helped me inspect data quality, identify nulls and datatype mismatches, and finalize how I wanted to structure the data downstream.

Once the structure was validated, I created a Glue ETL job using PySpark. This job cleaned and transformed the raw dataset into a dimensional star schema. I extracted key columns to build four dimension tables: `dim_patient`, `dim_doctor`, `dim_hospital`, and `dim_insurance`. I used `.dropDuplicates()` and assigned clean surrogate keys where needed. Then I created the `fact_admissions` table, which included foreign keys to each of the dimensions along with transactional fields like room number, admission type, medication, test results, and billing amount. After transformation, I wrote each table back to S3 in **Parquet** format — a compressed, columnar format that's optimized for analytics.

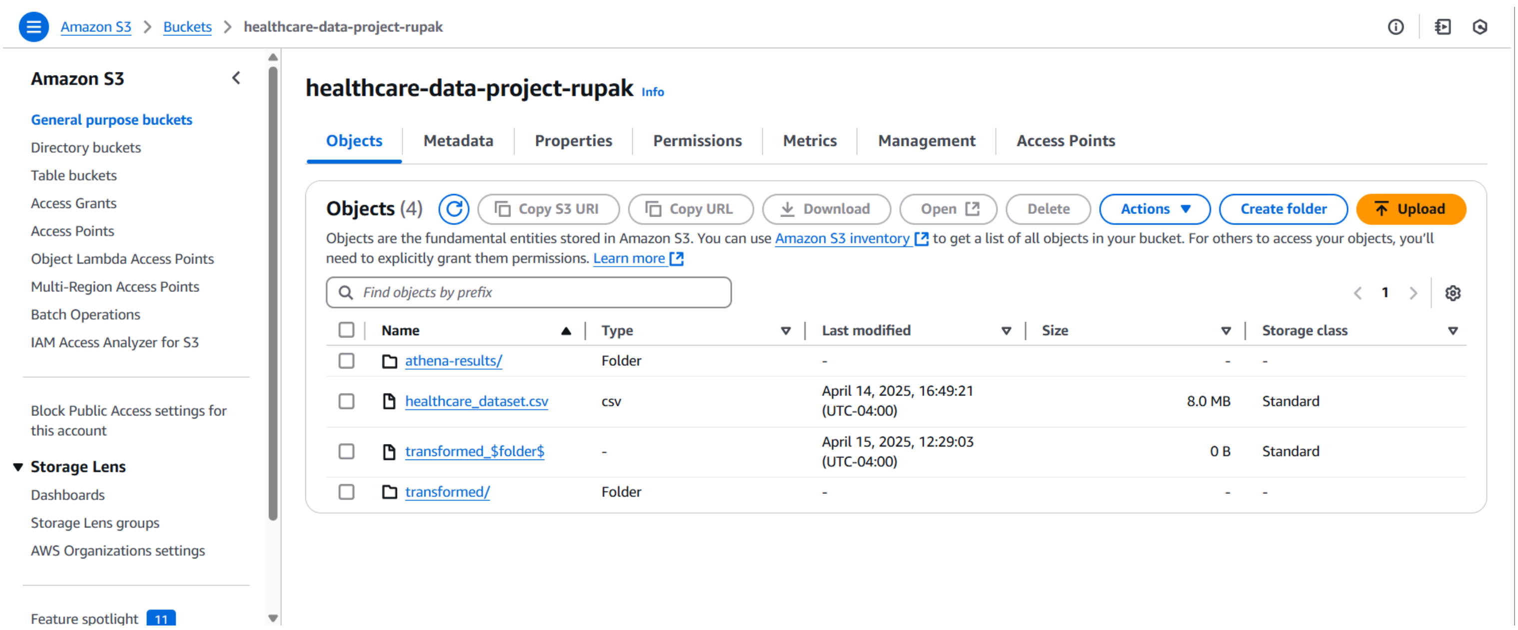
Next, I moved to Redshift Serverless to load the transformed data warehouse. I created a new namespace and workgroup and then wrote SQL `CREATE TABLE` scripts that exactly matched the structure of the Parquet files. I used the `COPY` command to load each table from S3 into Redshift. But this step came with a few challenges. First, Redshift COPY failed until I correctly associated the IAM role with both the namespace and the workgroup, and set it as default. I also had to make the workgroup publicly accessible and allow inbound traffic on port 5439 via the associated security group to connect external tools like Power BI. Another challenge was Spark versioning — Glue was running Spark 3, which threw strict datetime parsing errors on some fields. I solved that by dropping or cleaning problematic rows. Redshift also threw schema mismatch errors during COPY — for example, my table had 5 columns but the Parquet had 6. I resolved that by using Athena and Glue Catalog to confirm the Parquet schema before recreating the table with exactly matching columns.

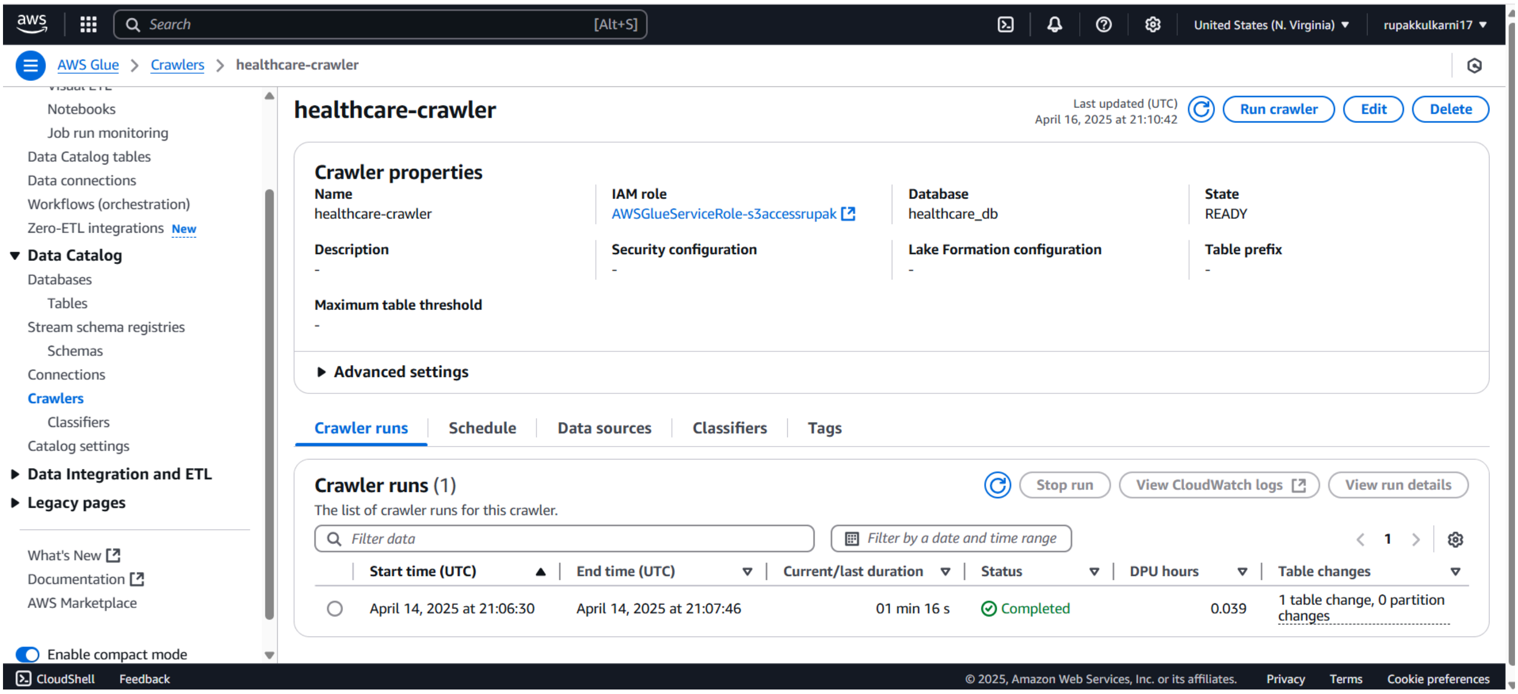
Once all the tables were successfully loaded into Redshift, I queried the schema using Query Editor v2 and verified joins between the fact and dimension tables. After validating the model, I connected Redshift to Power BI using the Redshift workgroup endpoint, the database name, and the admin user credentials. Once connected, I was able to use the structured tables from Redshift to build live visualizations in Power BI based on the star schema.

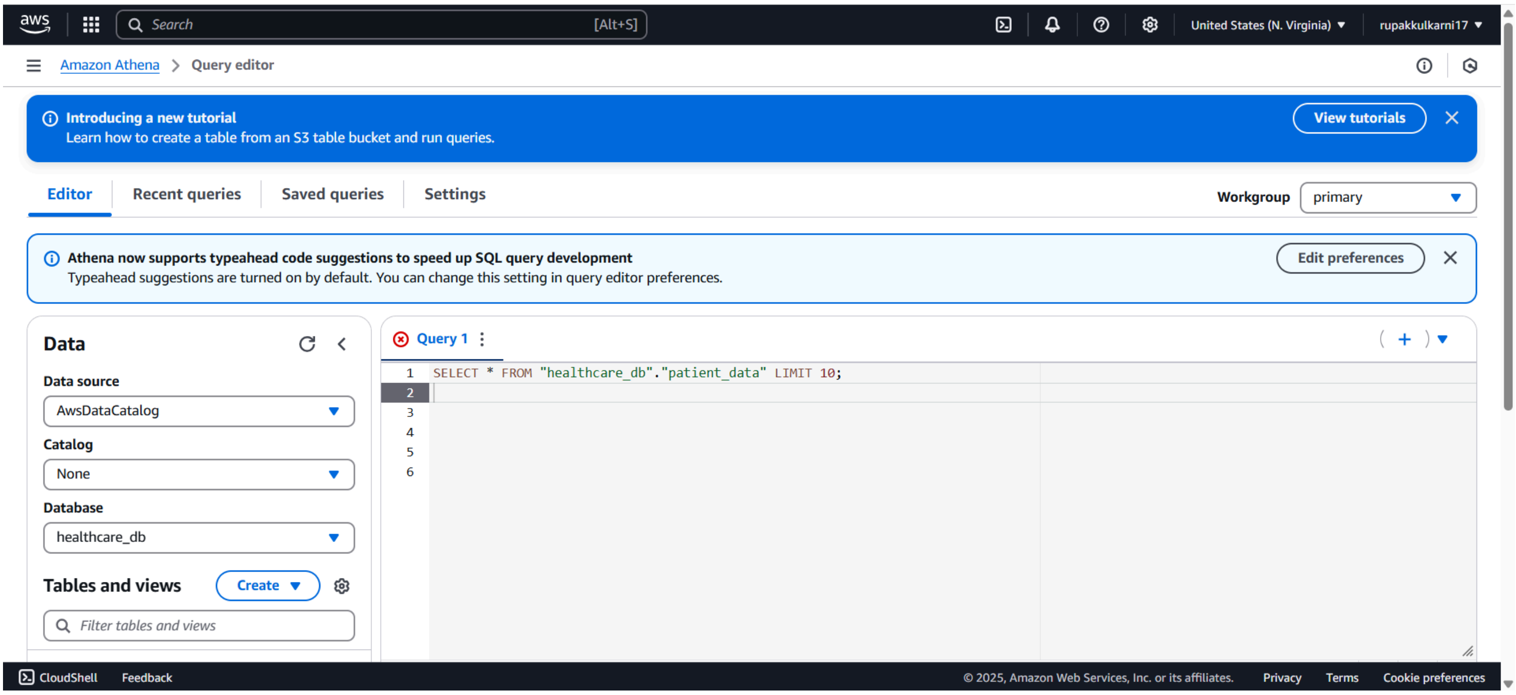
Blog reference: End to End Data Engineering Project Using AWS Services | by Gemma ✨ | Medium



the pipeline



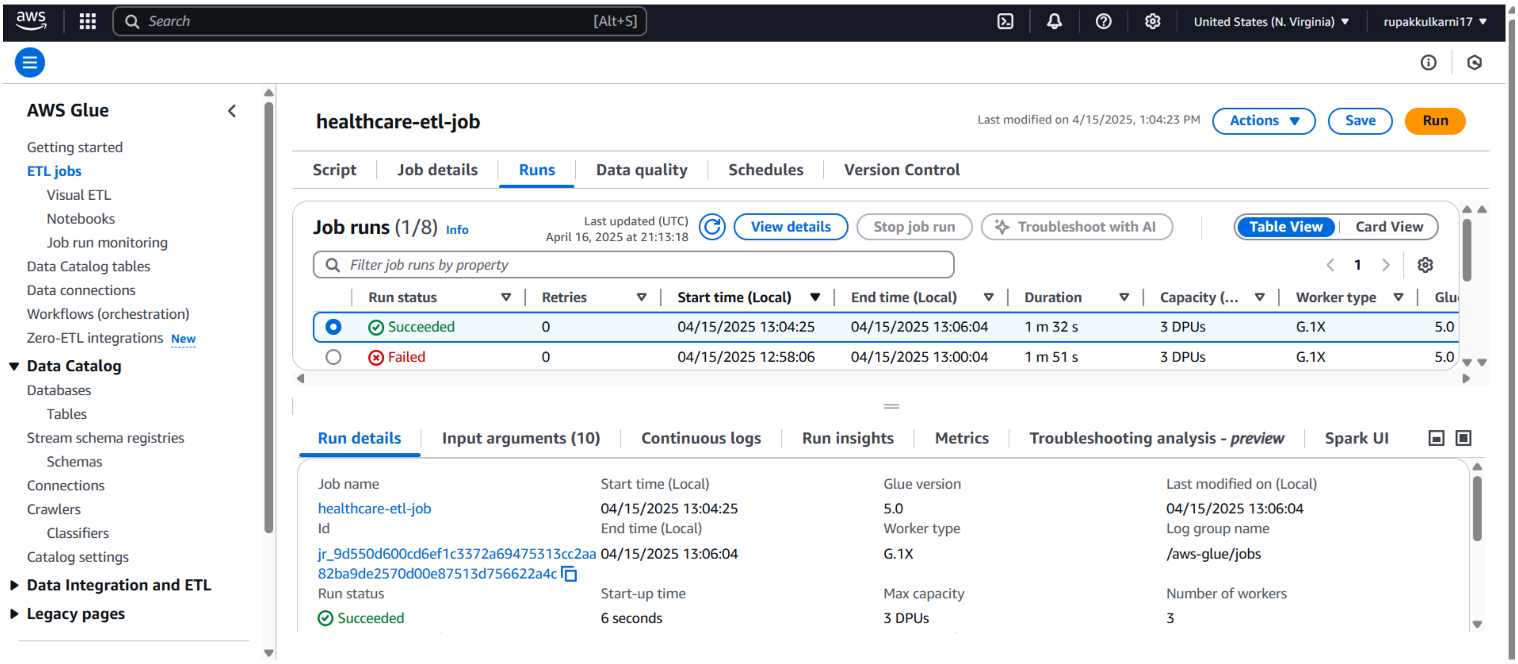


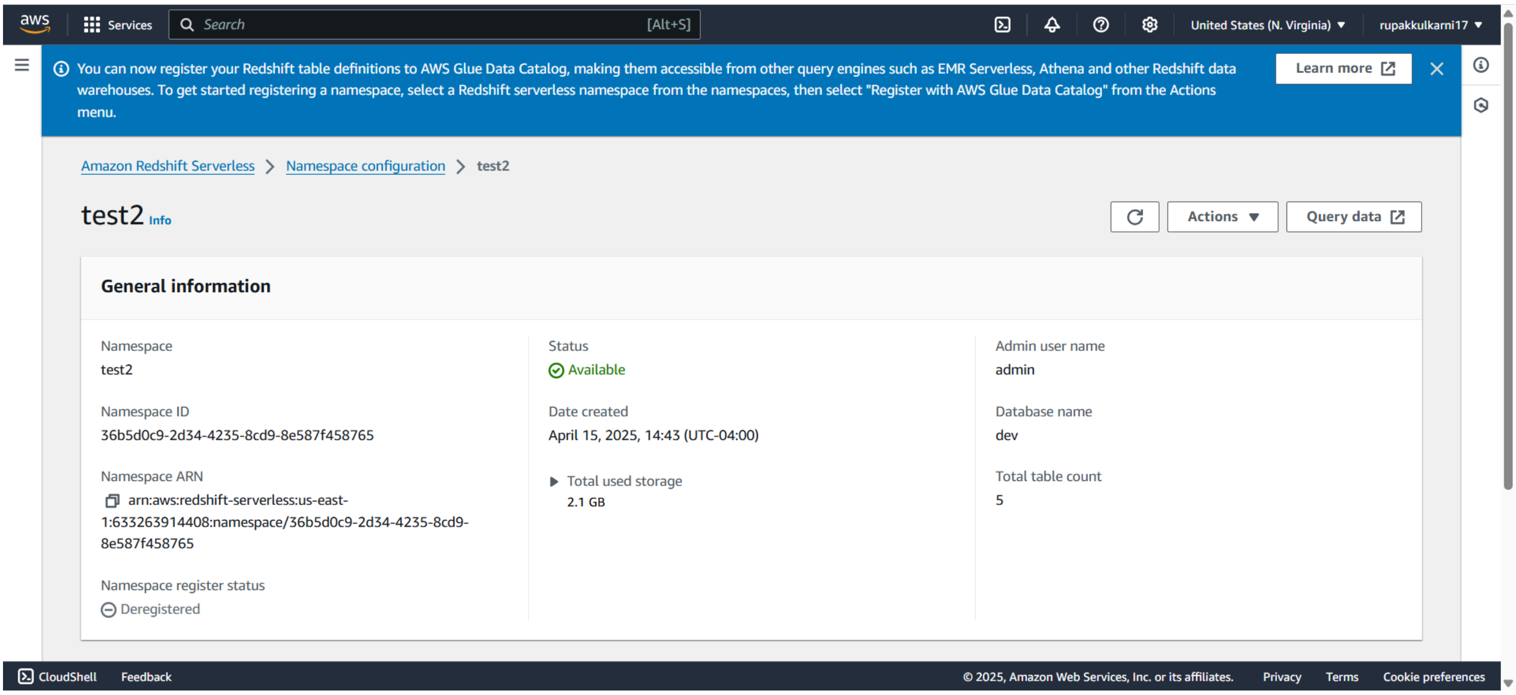


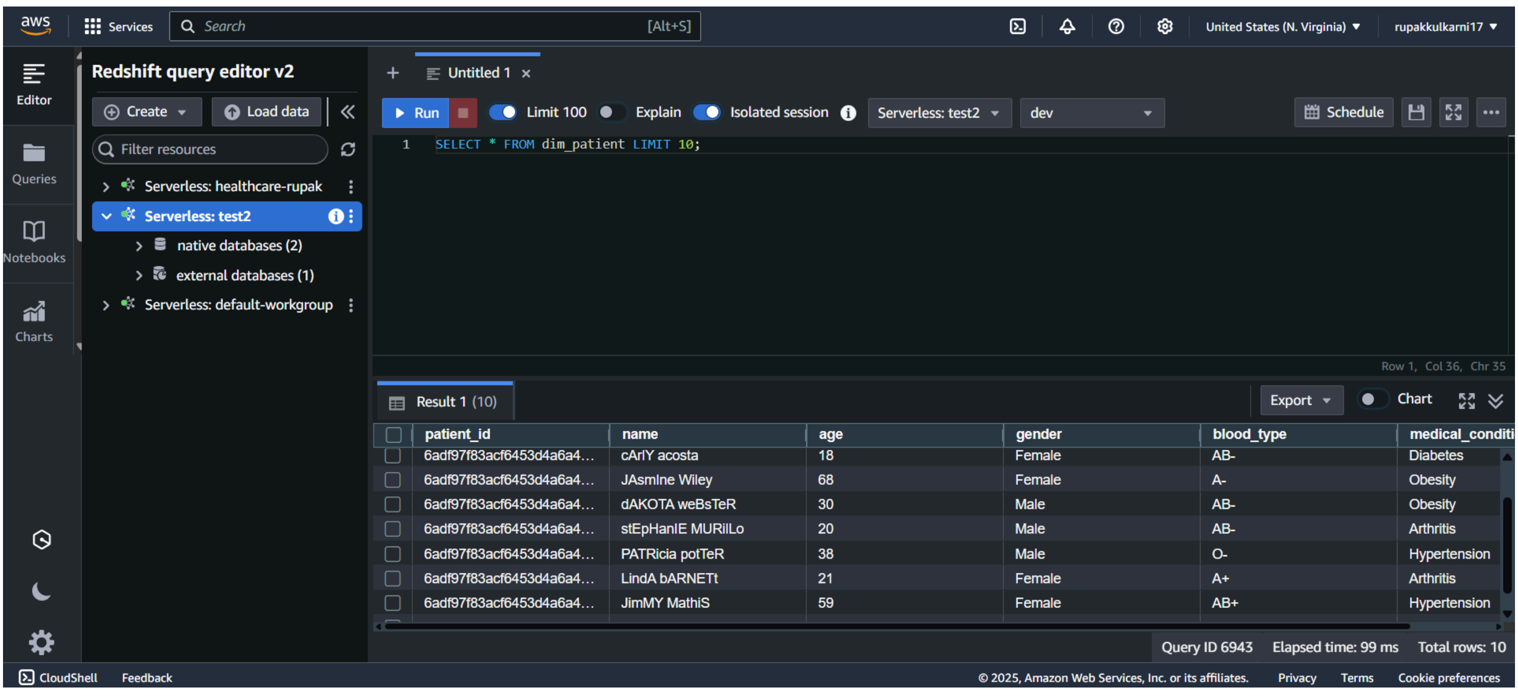
Glue script

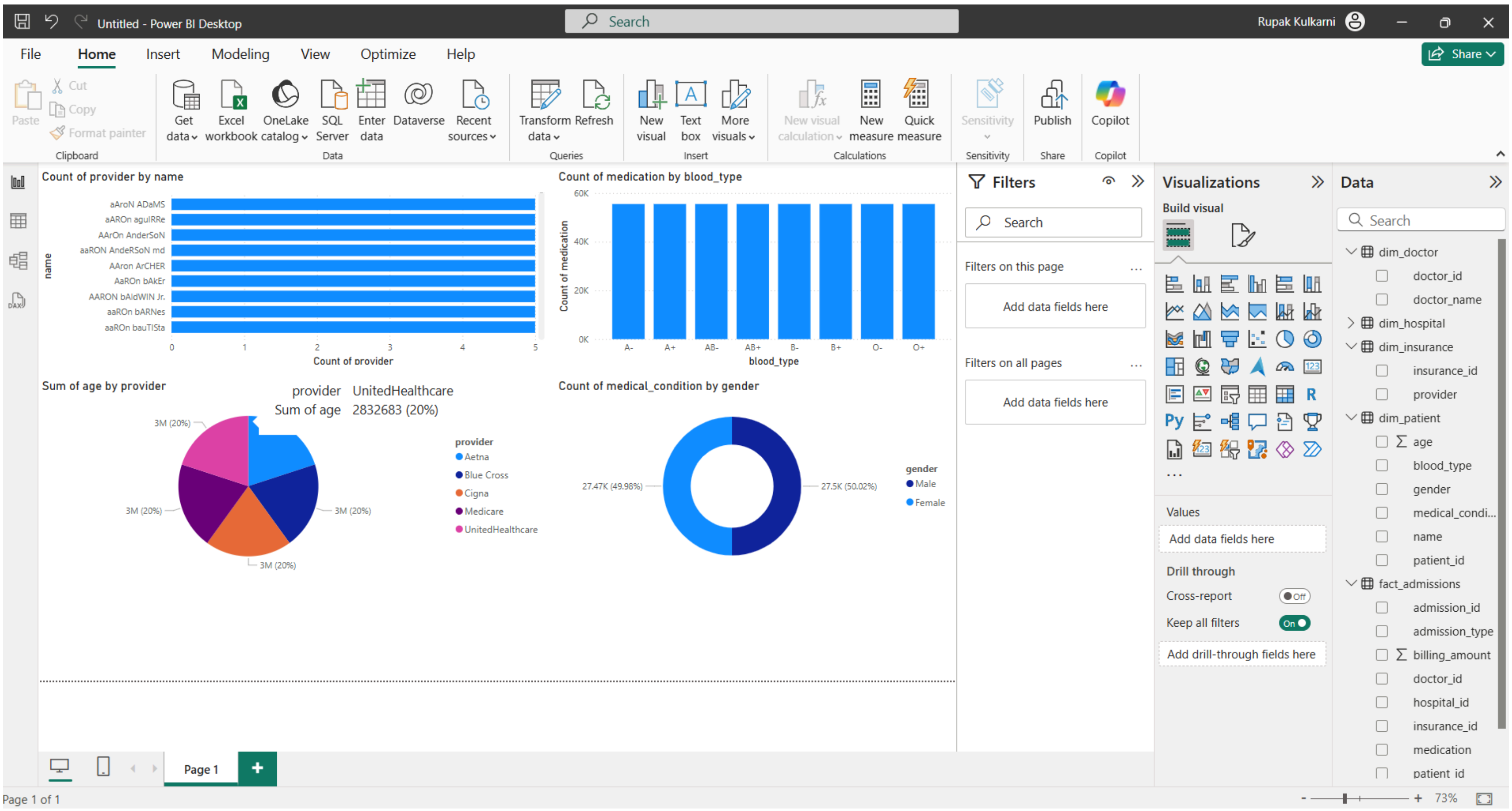
```
import sys
import hashlib
from pyspark.sql.functions import col, udf
from pyspark.sql.types import StringType
from awsglue.context import GlueContext
from awsglue.utils import getResolvedOptions
from awsglue.job import Job
from pyspark.context import SparkContext

# Glue job setup
args = getResolvedOptions(sys.argv, ['JOB_NAME'])
sc = SparkContext()
glueContext = GlueContext(sc)
spark = glueContext.spark_session
job = Job(glueContext)
```

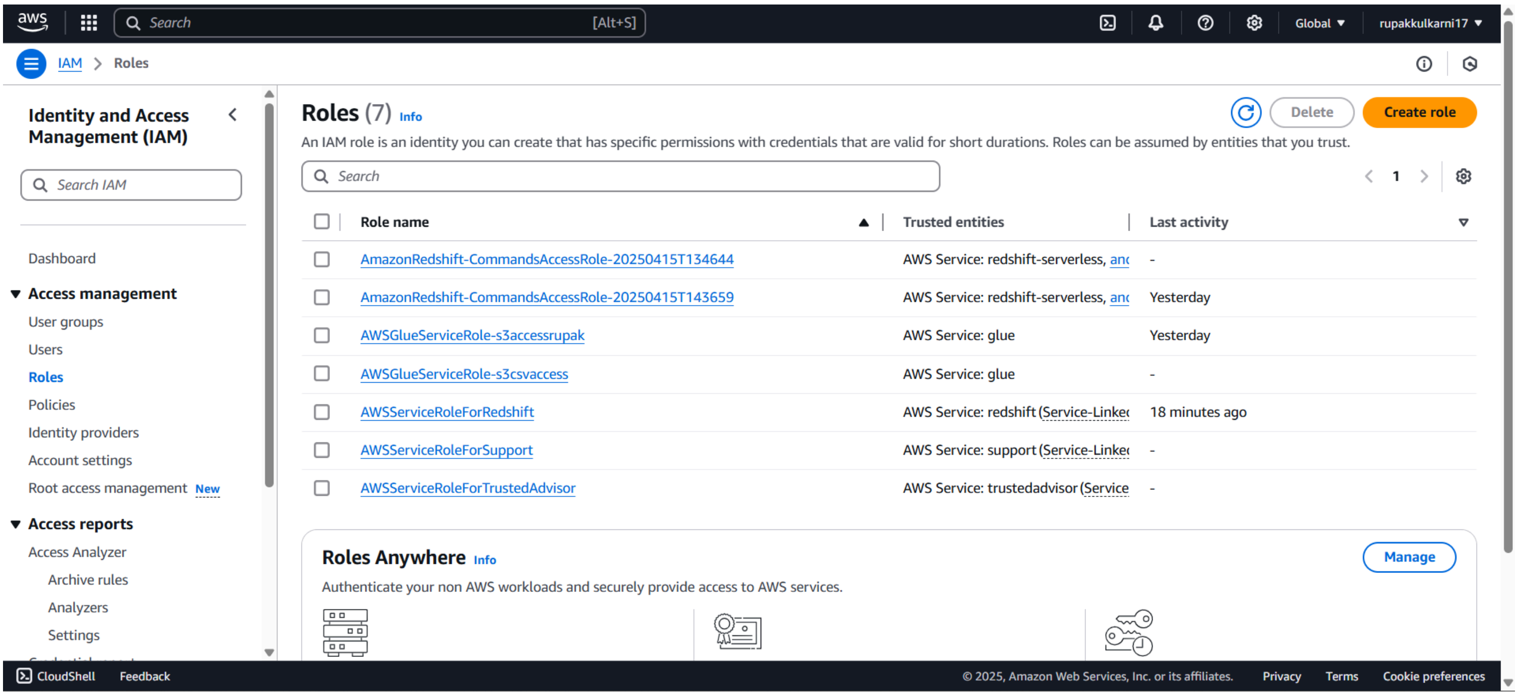
```
job.init(args['JOB_NAME'], args)

# Load raw data from Glue Catalog
df = glueContext.create_dynamic_frame.from_catalog(
    database="healthcare_db",
    table_name="healthcare_data_project_rupak"
).toDF()

# Normalize column names
df = df.select([col(c).alias(c.strip().lower().replace(" ", "_")) for c in df.columns])

# Drop rows missing critical fields
df = df.dropna(subset=["name", "hospital"])

# ✅ Drop date columns to avoid Spark 3.x issues
df = df.drop("date_of_admission", "discharge_date")

# Create surrogate key generator
def gen_id(val):
    return hashlib.md5(str(val).encode('utf-8')).hexdigest()
hash_udf = udf(gen_id, StringType())

# Add surrogate keys
df = df.withColumn("patient_id", hash_udf(col("name") + col("gender") + col("age").cast("string")))
df = df.withColumn("doctor_id", hash_udf(col("doctor")))
df = df.withColumn("hospital_id", hash_udf(col("hospital")))
df = df.withColumn("insurance_id", hash_udf(col("insurance_provider")))
df = df.withColumn("admission_id", hash_udf(col("name") + col("room_number").cast("string")))

# Build dimension tables
dim_patient = df.select("patient_id", "name", "age", "gender", "blood_type", "medical_condition").dropDuplicates()
dim_hospital = df.select("hospital_id", col("hospital").alias("hospital_name")).dropDuplicates()
dim_doctor = df.select("doctor_id", col("doctor").alias("doctor_name")).dropDuplicates()
dim_insurance = df.select("insurance_id", col("insurance_provider").alias("provider")).dropDuplicates()

# Build fact table (no admission_date/discharge_date)
fact_admissions = df.select(
    "admission_id", "patient_id", "doctor_id", "hospital_id", "insurance_id",
    "admission_type", "room_number", "billing_amount", "test_results", "medication"
)

# Write transformed tables to S3 in Parquet format
output_path = "s3://healthcare-data-project-rupak/transformed/"
dim_patient.write.mode("overwrite").parquet(output_path + "dim_patient/")
dim_hospital.write.mode("overwrite").parquet(output_path + "dim_hospital/")
dim_doctor.write.mode("overwrite").parquet(output_path + "dim_doctor/")
dim_insurance.write.mode("overwrite").parquet(output_path + "dim_insurance/")
fact_admissions.write.mode("overwrite").parquet(output_path + "fact_admissions/")

# Finish the job
job.commit()
```


AWS Glue
Getting started
ETL jobs
Visual ETL
Notebooks
Job run monitoring
Data Catalog tables
Data connections
Workflows (orchestration)
Zero-ETL integrations New
Data Catalog
Databases
Tables
Stream schema registries
Schemas
Connections
Crawlers
Classifiers
Catalog settings
Data Integration and ETL
Legacy pages
healthcare-etl-job
Last modified on 4/15/2025, 1:04:23 PM
Actions
Save
Run
Script
Job details
Runs
Data quality
Schedules
Version Control
Job runs (1/8) Info
Last updated (UTC) April 16, 2025 at 21:13:18
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 | 04/15/2025 13:04:25 | 04/15/2025 13:06:04 | 1 m 32 s | 3 DPUs | G.1X | 5.0
Failed | 0 | 04/15/2025 12:58:06 | 04/15/2025 13:00:04 | 1 m 51 s | 3 DPUs | G.1X | 5.0
Run details
Input arguments (10)
Continuous logs
Run insights
Metrics
Troubleshooting analysis - preview
Spark UI
Job name
healthcare-etl-job
Id
jr_9d550d600cd6ef1c3372a69475313cc2aa82ba9de2570d00e87513d756622a4c
Run status
Succeeded
Start time (Local)
04/15/2025 13:04:25
End time (Local)
04/15/2025 13:06:04
Start-up time
6 seconds
Glue version
5.0
Worker type
G.1X
Max capacity
3 DPUs
Last modified on (Local)
04/15/2025 13:06:04
Log group name
/aws-glue/jobs
Number of workers
3


You can now register your Redshift table definitions to AWS Glue Data Catalog, making them accessible from other query engines such as EMR Serverless, Athena and other Redshift data warehouses. To get started registering a namespace, select a Redshift serverless namespace from the namespaces, then select "Register with AWS Glue Data Catalog" from the Actions menu.
Learn more
Amazon Redshift Serverless > Namespace configuration > test2
test2 Info
Actions
Query data
General information
Namespace
test2
Namespace ID
36b5d0c9-2d34-4235-8cd9-8e587f458765
Namespace ARN
arn:aws:redshift-serverless:us-east-1:633263914408:namespace/36b5d0c9-2d34-4235-8cd9-8e587f458765
Namespace register status
Deregistered
Status
Available
Date created
April 15, 2025, 14:43 (UTC-04:00)
Total used storage
2.1 GB
Admin user name
admin
Database name
dev
Total table count
5
CloudShell
Feedback
© 2025, Amazon Web Services, Inc. or its affiliates.
Privacy
Terms
Cookie preferences


Redshift query editor v2
Editor
Queries
Notebooks
Charts
Create
Load data
Filter resources
Serverless: healthcare-rupak
Serverless: test2
native databases (2)
external databases (1)
Serverless: default-workgroup
Untitled 1
Run
Limit 100
Explain
Isolated session
Serverless: test2
dev
Schedule
1 SELECT * FROM dim_patient LIMIT 10;
Row 1, Col 36, Chr 35
Result 1 (10)
Export
Chart
patient_id | name | age | gender | blood_type | medical_conditi
6adf97f83acf6453d4a6a4... | cArlY acosta | 18 | Female | AB- | Diabetes
6adf97f83acf6453d4a6a4... | JAsmIne Wiley | 68 | Female | A- | Obesity
6adf97f83acf6453d4a6a4... | dAKOTA weBsTeR | 30 | Male | AB- | Obesity
6adf97f83acf6453d4a6a4... | stEpHanIE MURilLo | 20 | Male | AB- | Arthritis
6adf97f83acf6453d4a6a4... | PATRicia potTeR | 38 | Male | O- | Hypertension
6adf97f83acf6453d4a6a4... | LindA bARNETt | 21 | Female | A+ | Arthritis
6adf97f83acf6453d4a6a4... | JimMY MathiS | 59 | Female | AB+ | Hypertension
Query ID 6943
Elapsed time: 99 ms
Total rows: 10
CloudShell
Feedback
© 2025, Amazon Web Services, Inc. or its affiliates.
Privacy
Terms
Cookie preferences


Untitled - Power BI Desktop
Search
Rupak Kulkarni
File
Home
Insert
Modeling
View
Optimize
Help
Share
Paste
Cut
Copy
Format painter
Clipboard
Get data
Excel workbook
OneLake catalog
SQL Server
Enter data
Dataverse
Recent sources
Data
Transform data
Refresh
Queries
New visual
Text box
More visuals
Insert
New visual calculation
New measure
Quick measure
Calculations
Sensitivity
Publish
Share
Copilot
Count of provider by name
Count of provider
Count of medication by blood_type
Count of medication
blood_type
Sum of age by provider
provider UnitedHealthcare
Sum of age 2832683 (20%)
provider
Aetna
Blue Cross
Cigna
Medicare
UnitedHealthcare
Count of medical_condition by gender
27.47K (49.98%)
27.5K (50.02%)
gender
Male
Female
Filters
Search
Filters on this page
Add data fields here
Filters on all pages
Add data fields here
Visualizations
Build visual
Values
Add data fields here
Drill through
Cross-report
Keep all filters
Add drill-through fields here
Data
Search
dim_doctor
doctor_id
doctor_name
dim_hospital
dim_insurance
insurance_id
provider
dim_patient
age
blood_type
gender
medical_condi...
name
patient_id
fact_admissions
admission_id
admission_type
billing_amount
doctor_id
hospital_id
insurance_id
medication
patient_id
Page 1
Page 1 of 1
73%


aws
Search
[Alt+S]
Global
rupakkulkarni17
IAM > Roles
Identity and Access Management (IAM)
Search IAM
Dashboard
Access management
User groups
Users
Roles
Policies
Identity providers
Account settings
Root access management New
Access reports
Access Analyzer
Archive rules
Analyzers
Settings
Roles (7) Info
Delete
Create role
An IAM role is an identity you can create that has specific permissions with credentials that are valid for short durations. Roles can be assumed by entities that you trust.
Search
Role name
Trusted entities
Last activity
AmazonRedshift-CommandsAccessRole-20250415T134644
AWS Service: redshift-serverless, and
-
AmazonRedshift-CommandsAccessRole-20250415T143659
AWS Service: redshift-serverless, and
Yesterday
AWSGlueServiceRole-s3accessrupak
AWS Service: glue
Yesterday
AWSGlueServiceRole-s3csvaccess
AWS Service: glue
-
AWSServiceRoleForRedshift
AWS Service: redshift (Service-Linked
18 minutes ago
AWSServiceRoleForSupport
AWS Service: support (Service-Linked
-
AWSServiceRoleForTrustedAdvisor
AWS Service: trustedadvisor (Service
-
Roles Anywhere Info
Manage
Authenticate your non AWS workloads and securely provide access to AWS services.
CloudShell
Feedback
© 2025, Amazon Web Services, Inc. or its affiliates.
Privacy
Terms
Cookie preferences