# Lab 7 - Using Amazon Redshift Spectrum

Objs -

1. Connect to Amazon Redshift
2. Create an external table pointing to data stored in Amazon S3
3. Query data stored in Amazon S3 and join data between Amazon Redshift tables and data stored in Amazon S3

Amazon Redshift Spectrum is used to access and analyze data that's stored in files on Amazon S3 (Amazon's online storage service).

Redshift Spectrum allows you to work with this data without needing to move it into your main Amazon Redshift database.

**Task 1: Connecting to Amazon Redshift**

1.1 connect to pgweb using url, and connect using



**Task 2: Creating an external table**

Unlike a normal Amazon Redshift table, an external table references data stored in Amazon S3.

2.1 Create external schema

*CREATE EXTERNAL SCHEMA spectrum*

*FROM DATA CATALOG*

*DATABASE 'spectrumdb'*

*IAM\_ROLE '<INSERT-YOUR-REDSHIFT-ROLE>'*

*CREATE EXTERNAL DATABASE IF NOT EXISTS*

Replace IAM value

* CREATE EXTERNAL SCHEMA spectrum:
  + This part creates a new external schema named spectrum. A schema is like a folder that organizes tables and other database objects.
* FROM DATA CATALOG:
  + This specifies that the external schema will use the AWS Glue Data Catalog as the metadata store. The Data Catalog is a central repository to store and manage metadata for data in Amazon S3.
* DATABASE 'spectrumdb':
  + This indicates the name of the database in the AWS Glue Data Catalog that contains the metadata for the external tables. Here, the database name is spectrumdb.
* IAM\_ROLE '<INSERT-YOUR-REDSHIFT-ROLE>':
  + This specifies the AWS Identity and Access Management (IAM) role that Amazon Redshift will use to access the data in Amazon S3 and the AWS Glue Data Catalog.
* CREATE EXTERNAL DATABASE IF NOT EXISTS:
  + This optional clause tells Redshift Spectrum to create the database in the AWS Glue Data Catalog if it doesn't already exist. If the database already exists, it won't create a new one or overwrite the existing one.

2.2 create an external table

run this

*CREATE EXTERNAL TABLE spectrum.sales(*

*salesid INTEGER,*

*listid INTEGER,*

*sellerid INTEGER,*

*buyerid INTEGER,*

*eventid INTEGER,*

*dateid SMALLINT,*

*qtysold SMALLINT,*

*pricepaid DECIMAL(8,2),*

*commission DECIMAL(8,2),*

*saletime TIMESTAMP*

*)*

*ROW FORMAT DELIMITED*

*FIELDS TERMINATED BY '\t'*

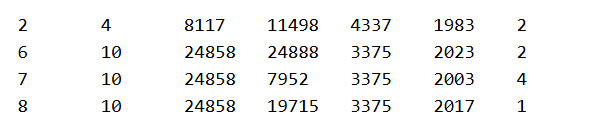
*STORED AS TEXTFILE*

*LOCATION 's3://aws-tc-largeobjects/ILT-TF-200-DBDWOA-1/sampledb/tickit/spectrum/sales/'*

*TABLE PROPERTIES ('numRows'='172000')*

There will be no o/p in pgweb, as tables are not created here.

2.3 sample of file created in S3 (given in lab env)



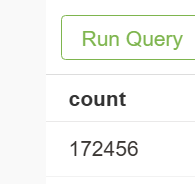
**Task 3: Querying data stored in Amazon S3**

**run queries against the external table, which uses Redshift Spectrum to process the data directly from Amazon S3.**

3.1 run

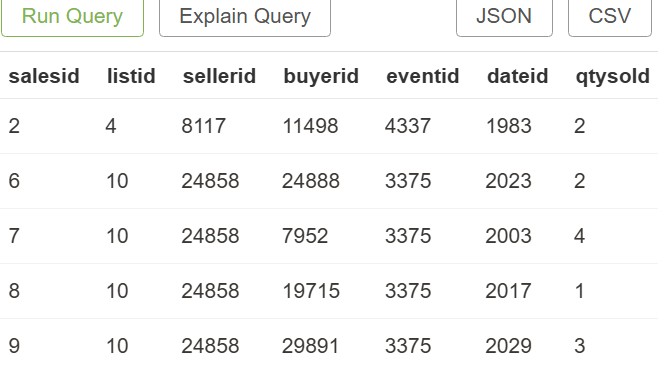
*SELECT COUNT(\*) FROM spectrum.sales*

o/p =



3.2 to view the table

*SELECT \* FROM spectrum.sales LIMIT 10*



Note- this is the same data as S3 in last task (tab delimiter)

Redshift Spectrum reads the data from Amazon S3 but presents it as though it is coming directly from Amazon Redshift.

Redshift Spectrum runs this query directly against the data stored in Amazon S3 without needing to load the data into a temporary Amazon Redshift table.

**join data stored in Amazon S3 with data stored in Amazon Redshift**

3.3 create a table in RS

*CREATE TABLE event(*

*eventid INTEGER NOT NULL DISTKEY,*

*venueid SMALLINT NOT NULL,*

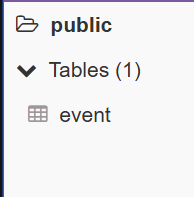
*catid SMALLINT NOT NULL,*

*dateid SMALLINT NOT NULL SORTKEY,*

*eventname VARCHAR(200),*

*starttime TIMESTAMP*

*)*



3.4 run this to copy data

*COPY event*

*FROM 's3://aws-tc-largeobjects/ILT-TF-200-DBDWOA-1/sampledb/tickit/allevents\_pipe.txt'*

*IAM\_ROLE 'arn:aws:iam::854514635624:role/Redshift-Role'*

*DELIMITER '|'*

*TIMEFORMAT 'YYYY-MM-DD HH:MI:SS'*

*REGION 'us-west-2'*

3.5 view data



3.6 run a query that joins data from this new event table (stored in the Amazon Redshift cluster) with data from the external sales table (stored in Amazon S3).

*SELECT TOP 10*

*spectrum.sales.eventid,*

*SUM(spectrum.sales.pricepaid)*

*FROM spectrum.sales, event*

*WHERE spectrum.sales.eventid = event.eventid*

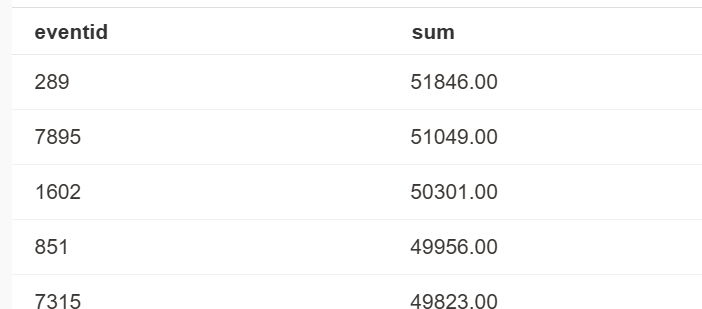
*AND spectrum.sales.pricepaid > 30*

*GROUP BY spectrum.sales.eventid*

*ORDER BY 2 DESC*

This query is listing total sales (from Amazon S3) grouped by event (from Redshift) where the price is more than $30.

3.7 tables joined, o/p shown



3.8 view the execution plan for this jooin

*EXPLAIN*

*SELECT TOP 10*

*spectrum.sales.eventid,*

*sum(spectrum.sales.pricepaid)*

*FROM spectrum.sales, event*

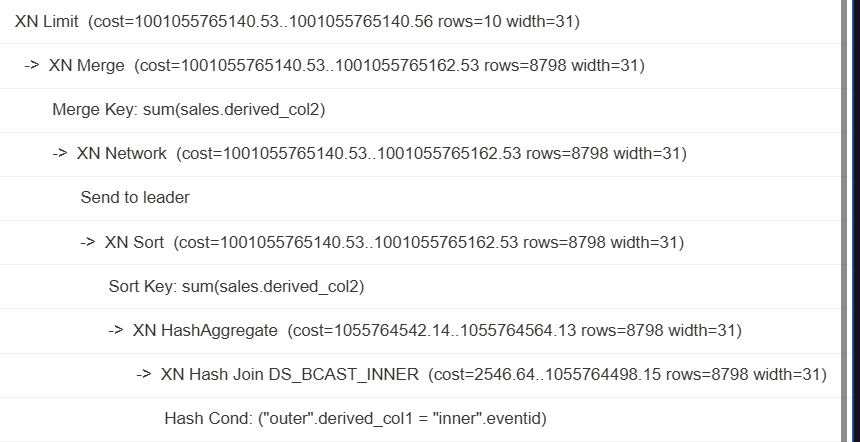
*WHERE spectrum.sales.eventid = event.eventid*

*AND spectrum.sales.pricepaid > 30*

*GROUP BY spectrum.sales.eventid*

*ORDER BY 2 DESC*

o/p=



Note that there are S3 Seq Scan, S3 HashAggregate, and S3 Query Scan steps that run against the data in Amazon S3.

1. S3 Seq Scan: Sequentially scans data stored in Amazon S3 without an index, reading through it from start to finish.
2. S3 HashAggregate: Aggregates data stored in Amazon S3 using hash-based techniques to compute functions like SUM, AVG, COUNT, etc., efficiently.
3. S3 Query Scan: Processes and scans data stored in Amazon S3 based on the specific query criteria, optimizing retrieval and filtering operations.