AWS Professional Services: Big Data and Analytics

Data Lake on AWS - Lab guide

Overview

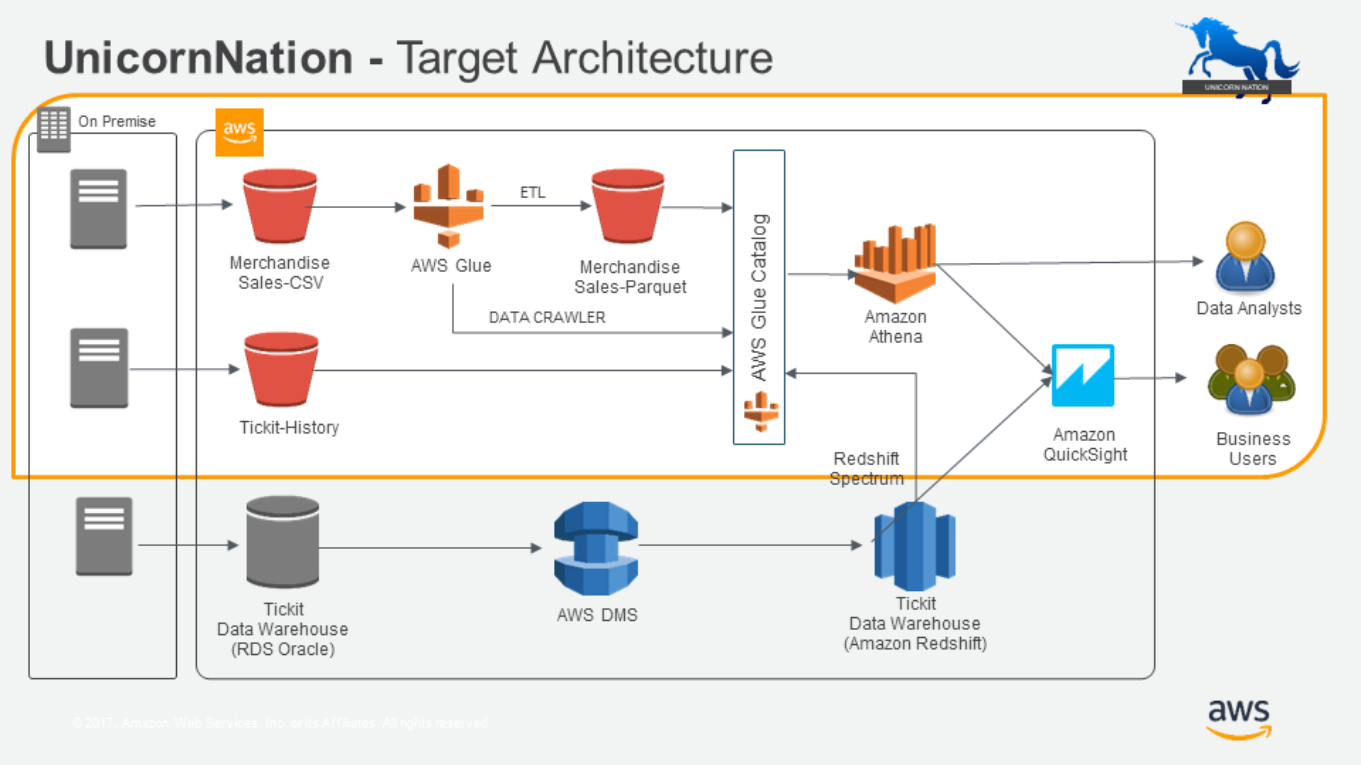
UnicornNation is a global entertainment company that provides ticketing, merchandising and promotion of large concerts and events.

In recent years, they have been collecting data through a number of disparate systems and want to consolidate this data in a modern data architecture.

A workshop was held with the key stakeholders in UnicornNation and they identified three key data sources they would like to consolidate and have provided the funding and resources to build a Data Lake on AWS.

During the course of this bootcamp, you will be building a Data Lake on AWS to meet their requirements and gain experience with a number of core AWS services, including S3, Glue, Athena, Redshift, Redshift Spectrum and QuickSight.

UnicornNation Target Architecture



###About the Labs

With the following labs you will get hands-on experience with some of the key AWS services that underpin a Data Lake implementation. The labs are provided with step-by-step instructions that will help you use each service to build the basic build blocks of a data lake.

For the labs, you will be using your own computer and logging in to an AWS console through your web browser.

##Lab 1: Creating a Glue Data Crawler

The IT team at UnicornNation has exported Merchandise Sales data from their finance system and transferred this data to a folder in a single S3 bucket. There are multiple .CSV files in the bucket, each representing a month's worth of data.

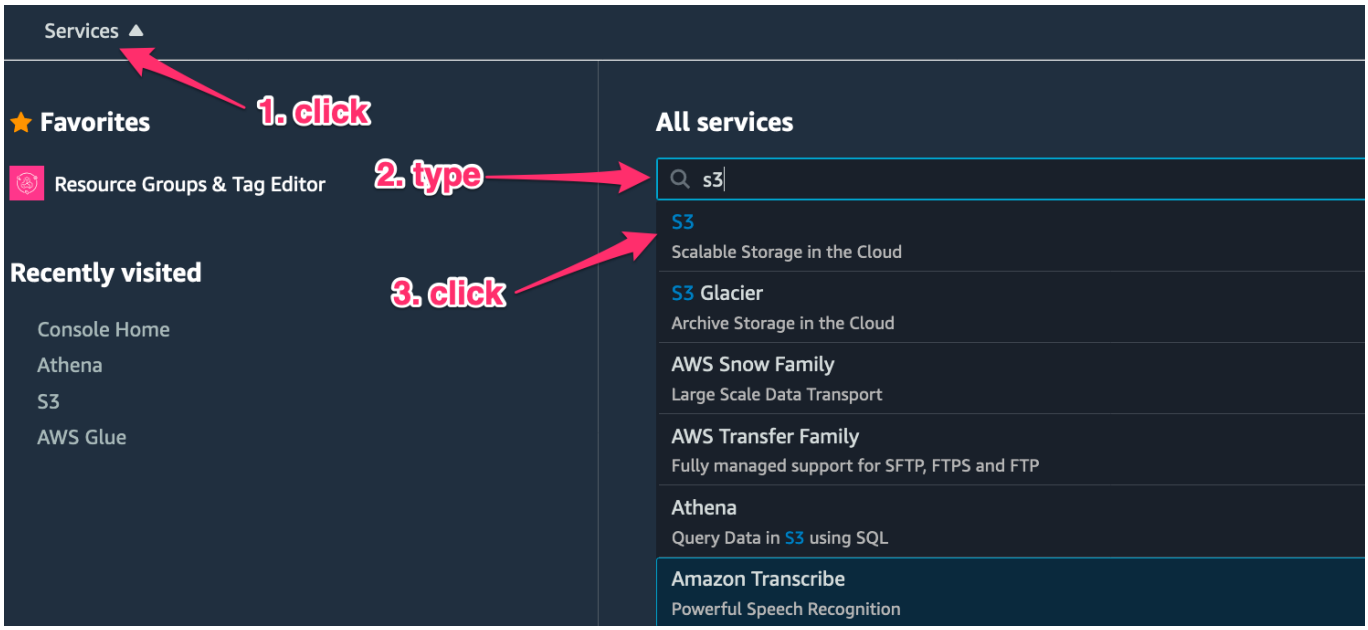
In this lab, we are going to create a Glue Data Crawler to crawl across this bucket. Once we have created the crawler, we are going to run it to determine the tables that are located in the bucket and add their definitions to the Glue Data Catalog.

To create your Glue data crawler, follow these steps:

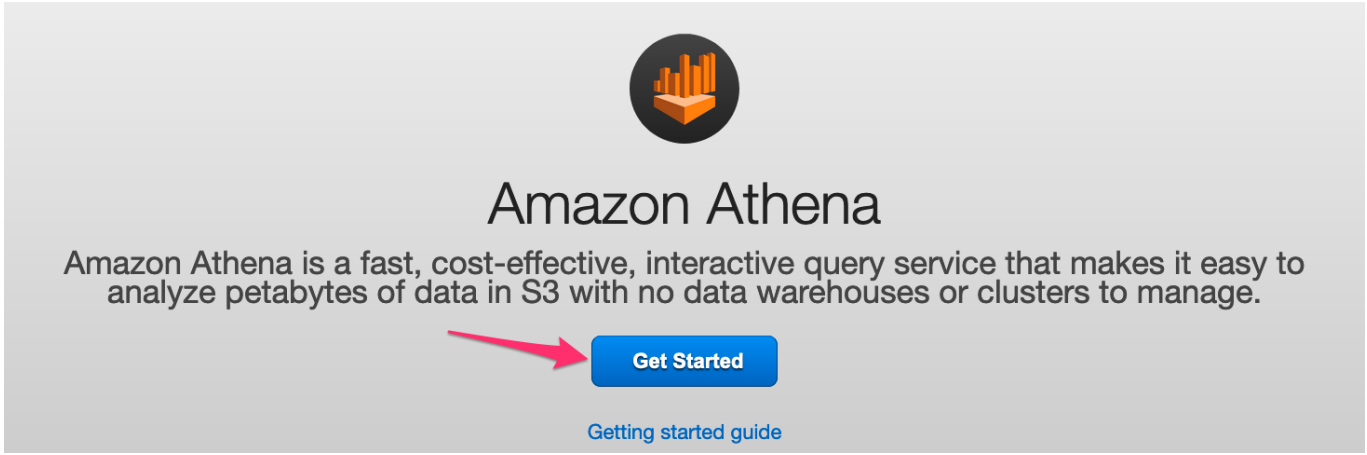
1. Login to the AWS Console using the login URL and credentials provided. Once you are logged in, check in the upper right-hand corner that you are using **Oregon** region. If not,

use the drop-down list to change to the **Oregon** region.

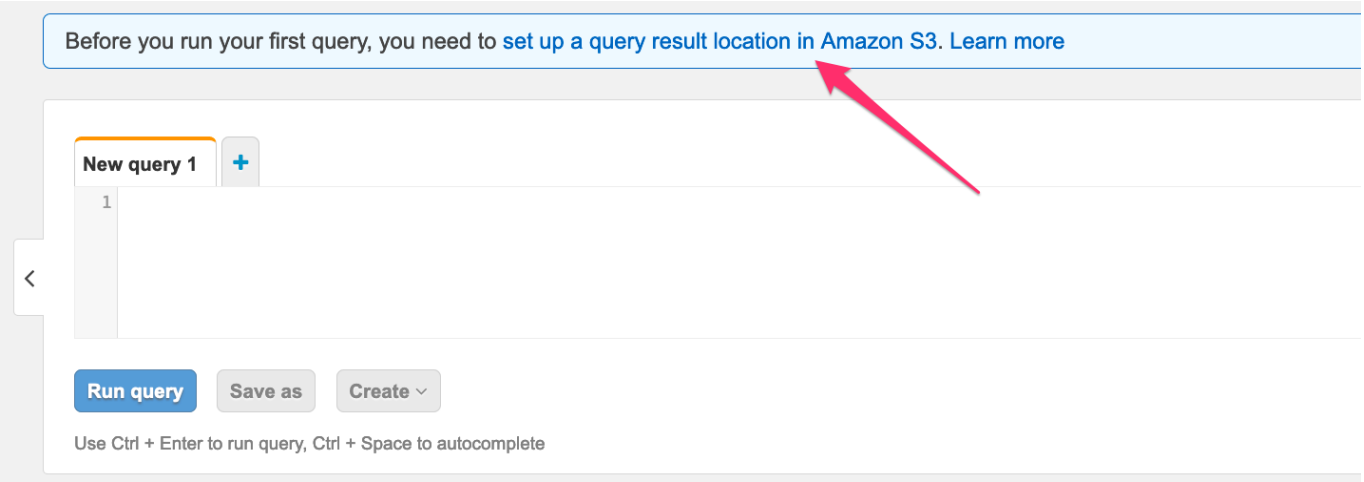
1. Before start working with AWS Glue, we need to configure a default result bucket, please navigate to the S3 service



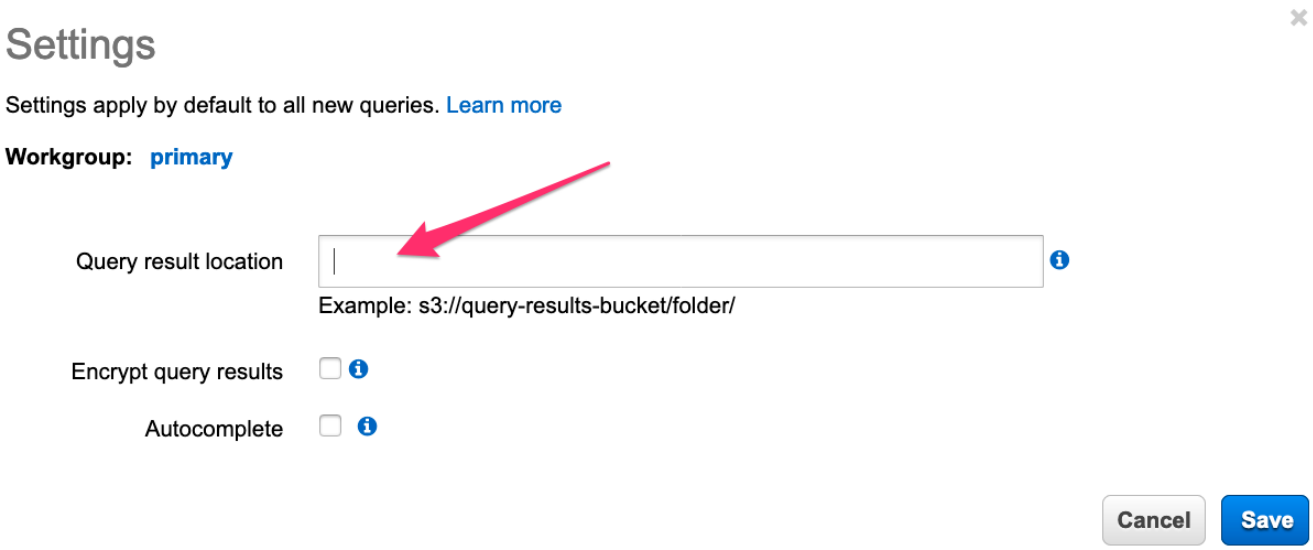
1. Locate the S3 bucket with this name pattern `query-results-bucket-xxxxxxxxx`
2. Copy and paste the full bucket name in a notepad, you will need it in the following step
3. Now navigate to the Athena service. Services > Athena
4. Click on **Get Started**



1. Click on the link to set up a query result location



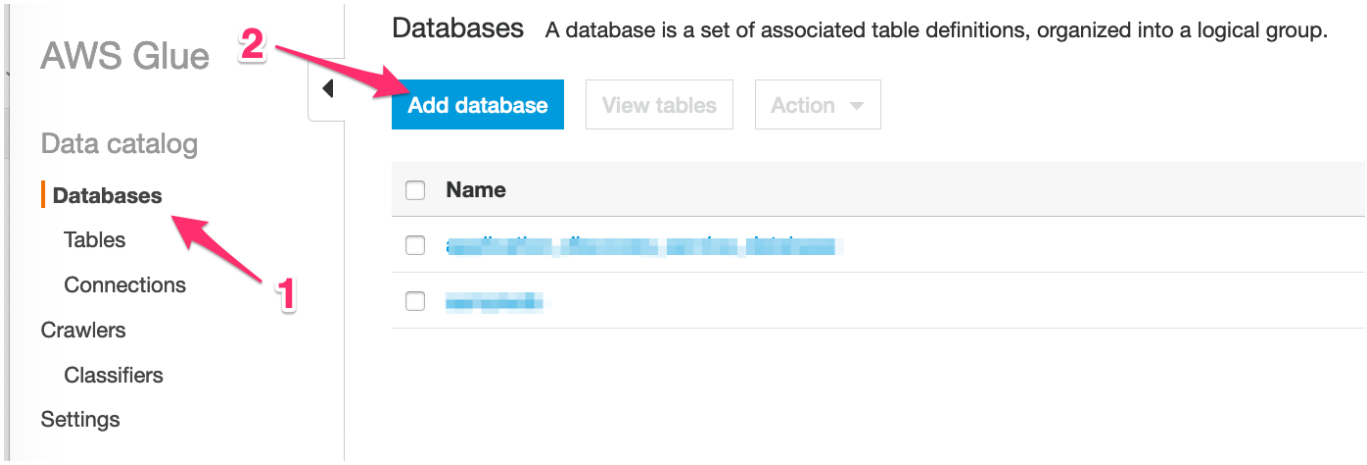
1. Enter the S3 bucket name you recorded in step 4. Follow this pattern: `s3://[my-query-result-bucket]/` (include the slash '/' at the end).



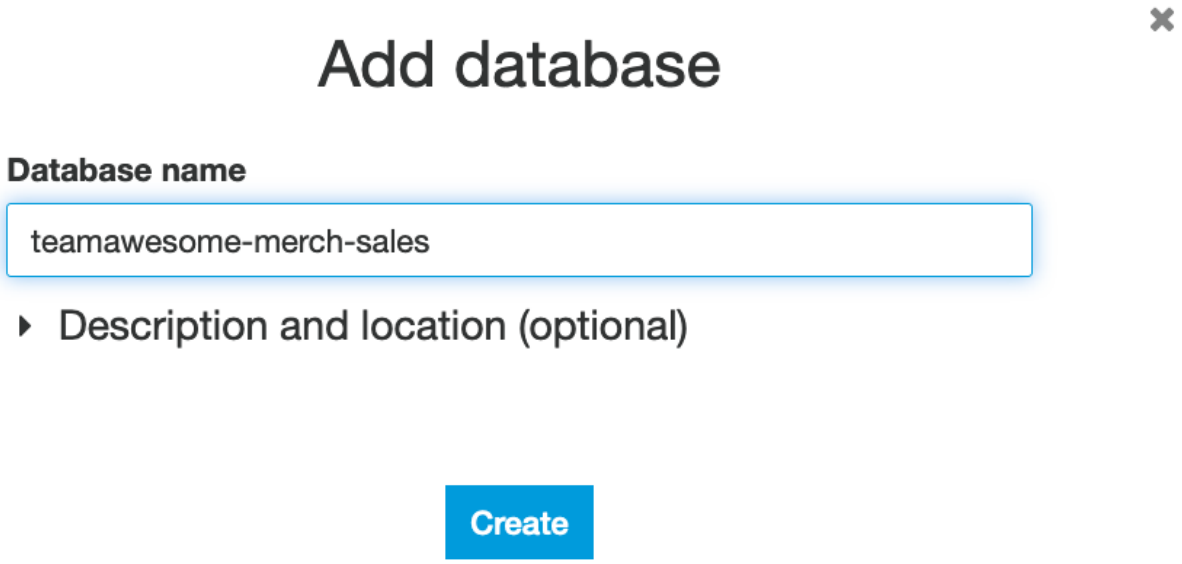
1. Click **Save**

2. From the console, navigate to the AWS Glue service. Services > AWS Glue

3. From the AWS Glue Data Catalog, navigate to **Databases** and click **Add Database**

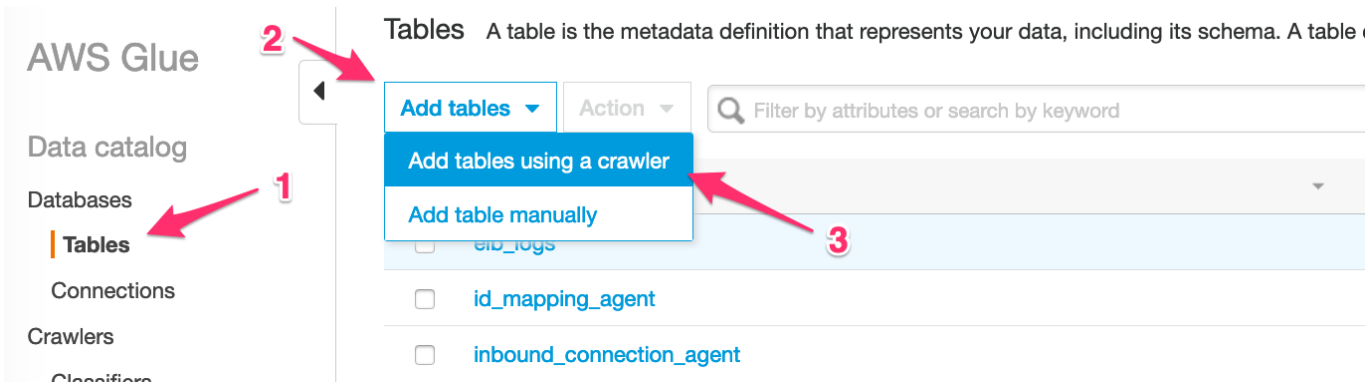


1. Enter a database name, type `teamawesome-merch-sales` and click **Create**



1. Now that your database has been created, click on the **Tables** menu in the left-hand menu

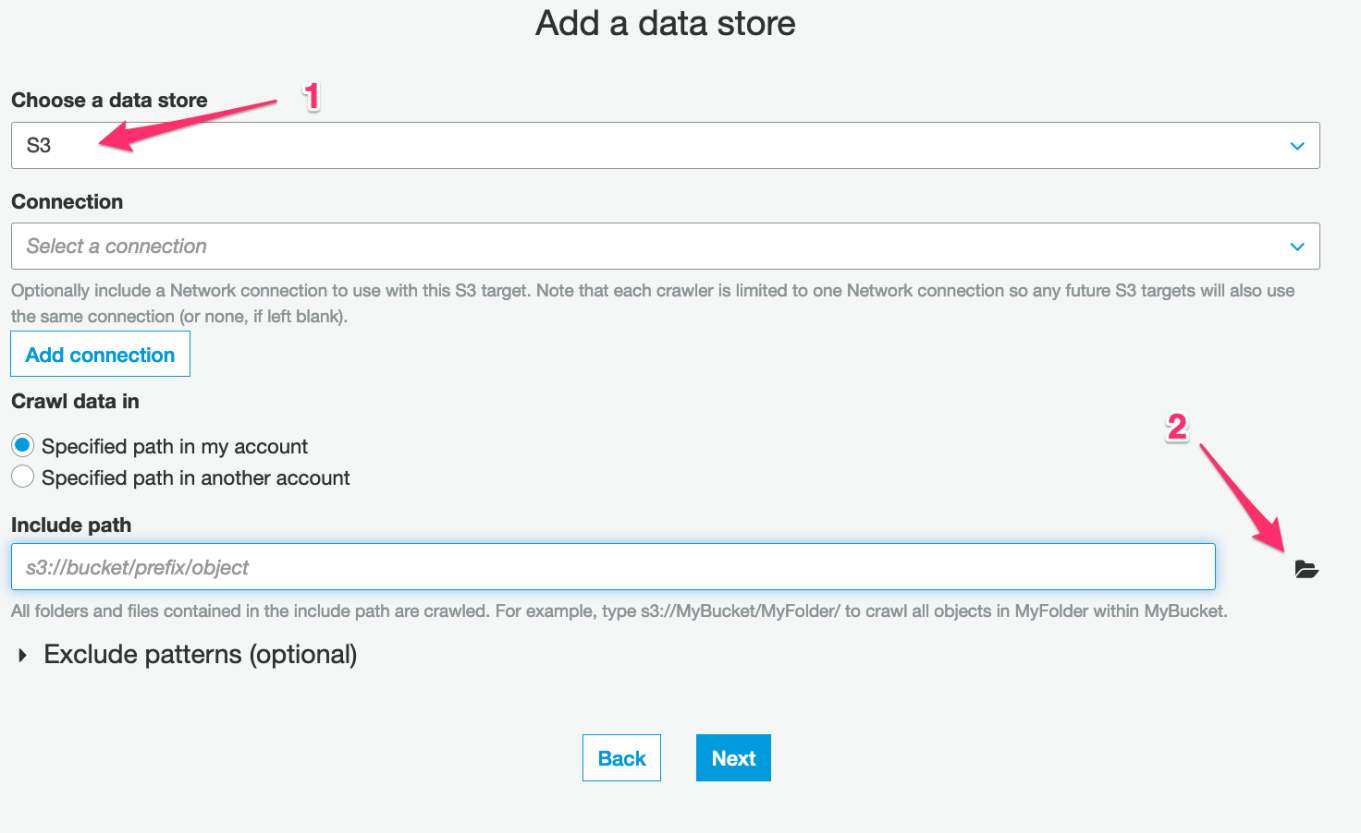
2. Select the drop down to **Add Tables** > **Add tables using a crawler**



1. Enter a name for your crawler, type `teamawesome-merch-sales-crawler` and click **Next**

2. For Crawler source type select **Data stores**, then click **Next**

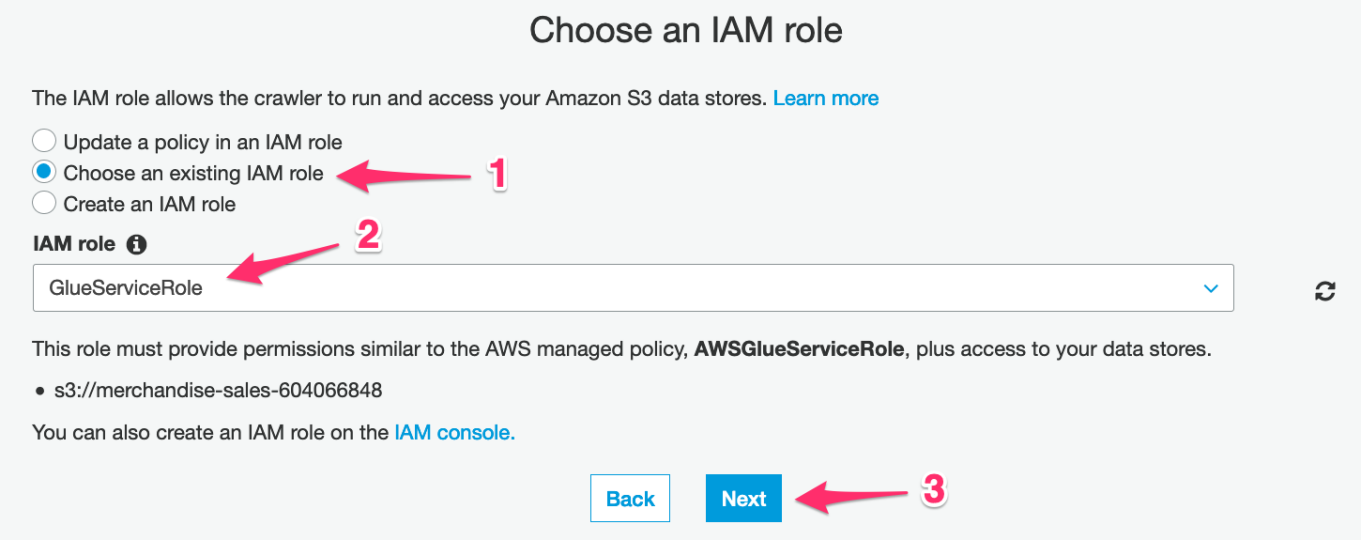
3. For your data store, select **S3** and below **Include path**, click on the little folder icon



1. This will open a new window, look for the Merchandise Sales bucket, it's name will follow this pattern: `merchandise-sales-xxxxxxx` , click **Select** and then click **Next**.

2. For **Add another data store**, take the default of *No* and click **Next**

3. Next, when setting up an IAM role, click on **Choose an existing IAM Role**, then under IAM role select role name **GlueServiceRole**. Then click **Next**.

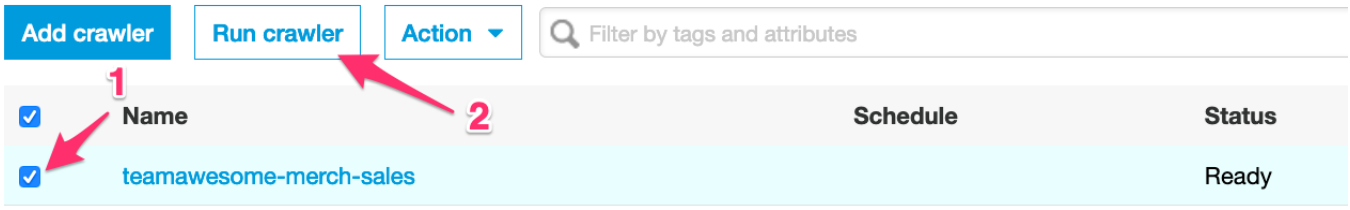


1. For the Frequency, leave the default of **Run on demand** and click **Next**

2. In **Configure the crawler's output**, select the Database **teamawesome-merch-sales** and click **Next**

3. Finally, review the settings for your crawler and click **Finish**. This should take you to a list of crawlers that have been created

4. Select the crawler you just created and click on **Run crawler**

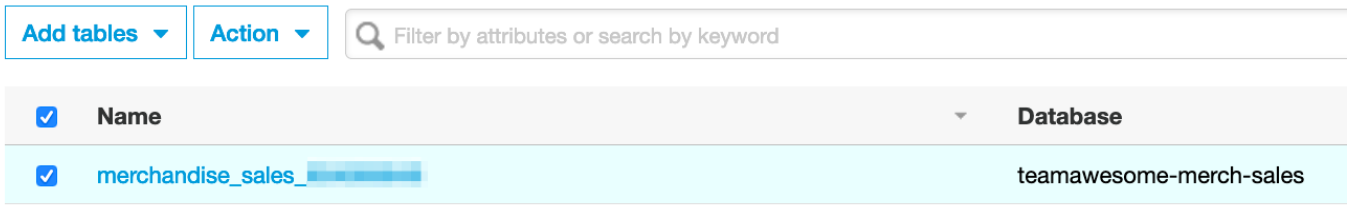


Watch the Crawler console for your job to finish successfully (it should take ~2 minutes). *Status* must change from **Starting** to **Ready**

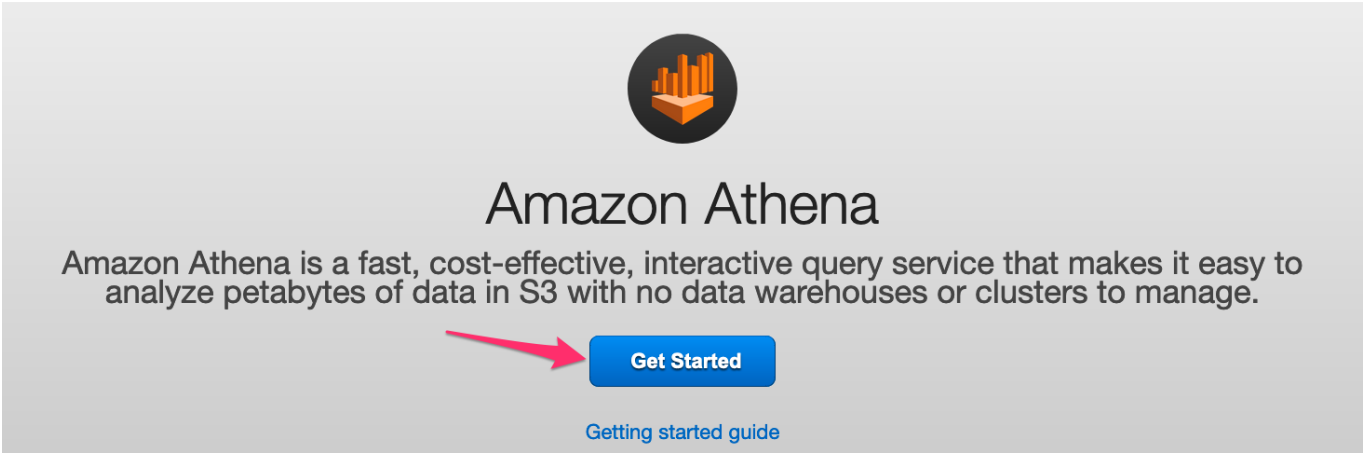
1. Using the navigation menu on the left, navigate to **Tables**

2. You should see a new table has been created, copy and paste the table name on a notepad.

3. Now click to select the table, then select **Action** > **View Data**. A dialogue window will be open, select **Preview data**

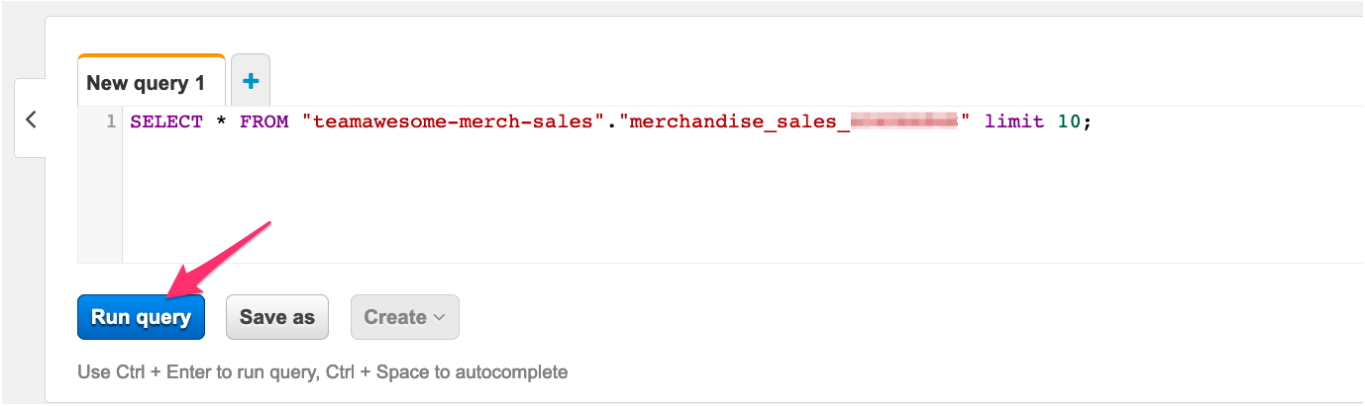


1. This will open the Athena console, click Get started



1. You will see the Athena console, you will run the below query, but first you need to update it with the table name you recorded in step 26- Click **Run query**

```
SELECT * FROM "teamawesome-merch-sales"."[GLUE_TABLE_NAME]" limit 10;
```



1. You must see a similar query output

Results

	eventid	eventname	producttype	productcolor	size	creditcardtype	currency
1	512	A Catered Affair	Hoodie	Crimson	2XL	jcb	Dollar
2	513	Grease	T-Shirt	Puce	XL	bankcard	Dollar
3	514	Hairspray	Stickers	Orange	S	diners-club-enroute	Dollar
4	515	A Catered Affair	Key Chain	Pink	M	diners-club-carte-blanche	Dollar
5	516	Jersey Boys	Souvenir Program	Green	S	bankcard	Dollar
6	517	A Chorus Line	Key Chain	Fuscia	S	jcb	Dollar
7	518	Kiss Me Kate	Hoodie	Yellow	2XL	jcb	Dollar
8	519	Oliver!	T-Shirt	Pink	XL	maestro	Dollar
9	520	Kiss Me Kate	T-Shirt	Puce	L	visa-electron	Dollar
10	521	West Side Story	T-Shirt	Purple	XL	jcb	Dollar

##Lab 2: Modifying Table Schemas

The IT team at UnicornNation has extracted historical data from their ticketing system, named “Tickit” which processes the majority of transactions for the company. This data source is known as the Tickit History.

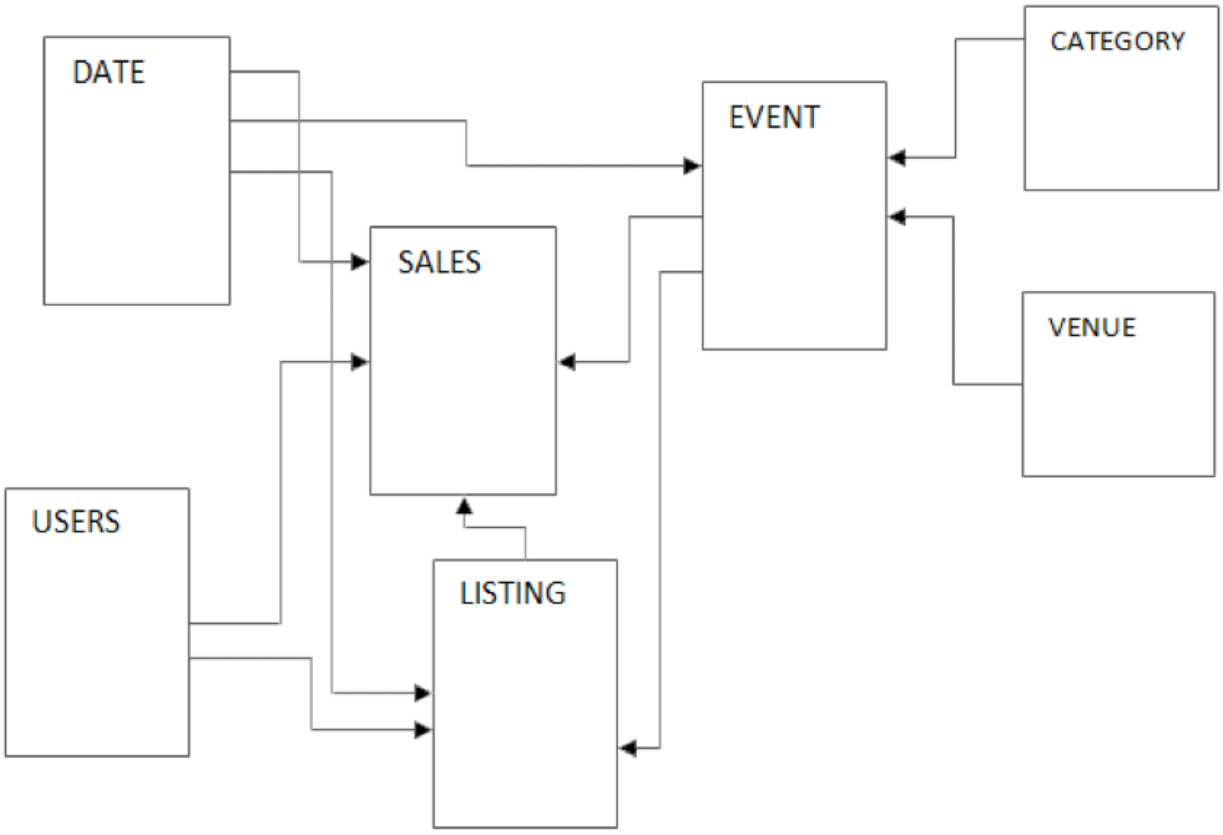
They have stored this data in an S3 bucket and have created folder/prefixes for each table of data they have exported.

In this lab, we are going to create a Glue Data Crawler to crawl across this bucket. Once we have created the crawler, we are going to run it to determine the tables that are located in the bucket and add their definitions to the Glue Data Catalog.

In the first lab, the files we crawled were comma-delimited and there was a header row in each that provided the field names for the different columns.

In this example, we are going to create a Glue crawler on a more complex dataset, where the files are pipe-delimited and don’t have a header row with the column names.

This dataset is stored in an S3 Bucket with a folder/key for each table name. These tables make up the “Tickit” sample data set, which consists of seven tables: two fact tables and five dimension tables as shown below:



This data set is the HISTORICAL data for the Tickit database—this data will not be used that often, so S3 is a great place to store and access this data.

To create these tables in the Glue catalog, follow these steps:

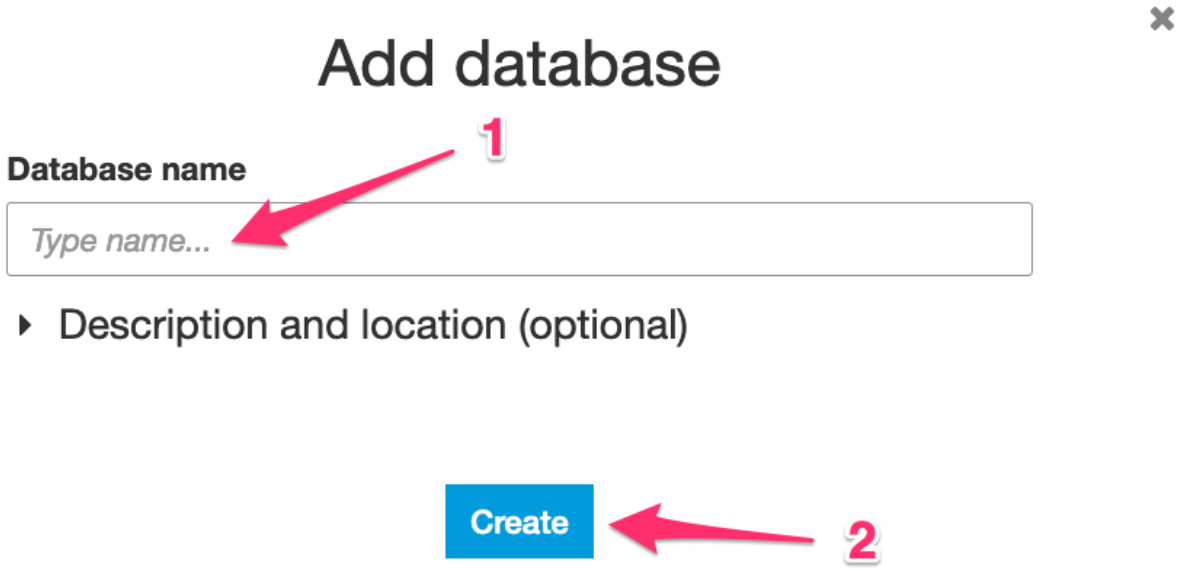
Task 1: Create your Glue crawler

1. Login to the AWS Console using the login URL and credentials provided. Once you are logged in, check in the upper right-hand corner that you are using **Oregon** region. If not, use the drop-down list to change to the **Oregon** region.

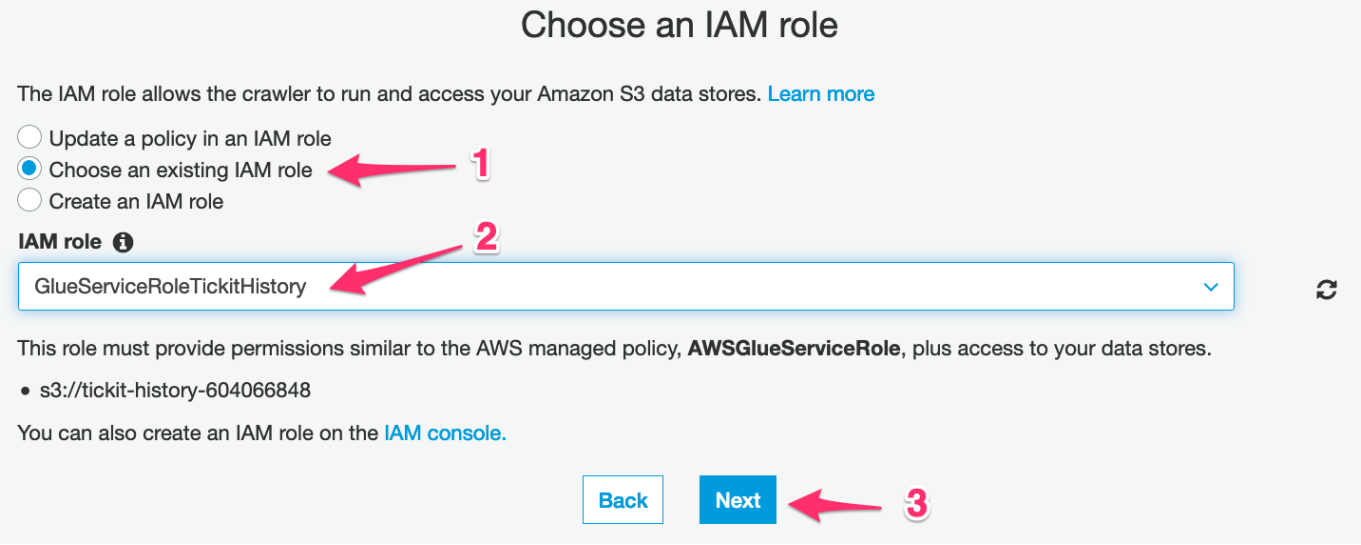
2. From the console, navigate to the AWS Glue service

3. From the AWS Glue Data Catalog, navigate to Databases and click **Add database**

4. Enter a database name, type `teamawesome-tickit-history` and click **Create**



1. Now that your database has been created, click **Tables** in the left-hand menu
2. Select the drop down to **Add Tables** > **Add tables using a crawler**
3. Enter a name for your crawler, type `teamawesome-tickit-crawler` and click **Next**
4. For Crawler source type select **Data stores**
5. For your data store, select **S3** and below **Include path** click on the little folder icon
6. This will open a new window, look for the Tickit History bucket, it's name will follow this pattern: `tickit-history-xxxxxxx` , click **Select** and then click **Next**.
7. For Add another data store, take the default of *No* and click **Next**
8. Next, when setting up an IAM role, click on **Choose an existing IAM Role**, then under IAM role select role name **GlueServiceRoleTickitHistory**. Then click **Next**.

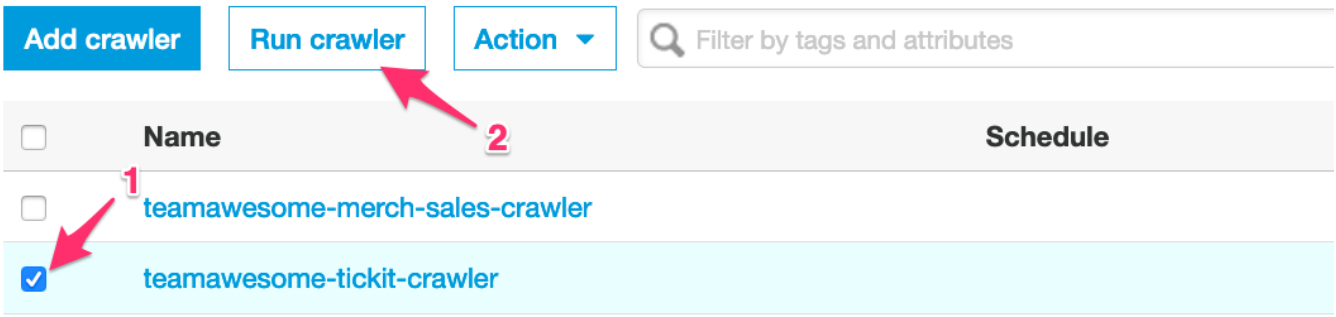


1. For the Frequency, select the default of *Run on Demand* and click **Next**

2. In **Configure the crawler's output**, select the Database **teamawesome-tickit-history** and click **Next**

3. Finally, review the settings for your crawler and click **Finish**. This should take you to a list of crawlers that have been created

4. Select the crawler you just created, **teamawesome-tickit-crawler**, and click on **Run crawler**



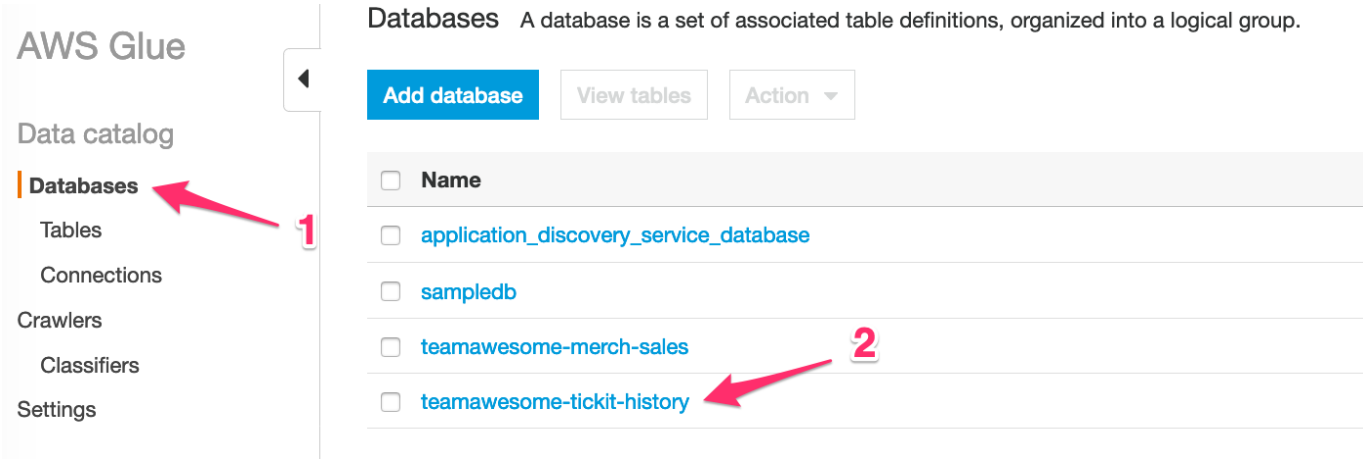
Watch the Crawler console for your job to finish successfully (it should take ~2 minutes). *Status* must change from **Starting** to **Ready**

1. Using the navigation menu on the left, navigate to Tables

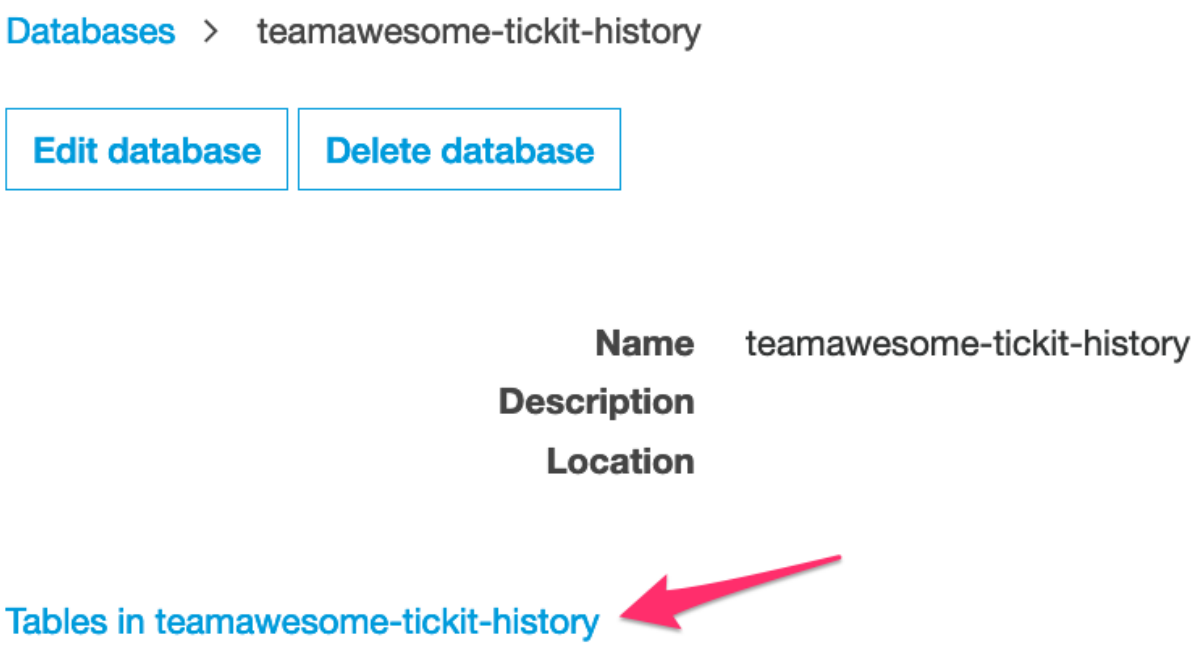
2. You should see several tables has been created

Task 2: Modifying the Table Schemas

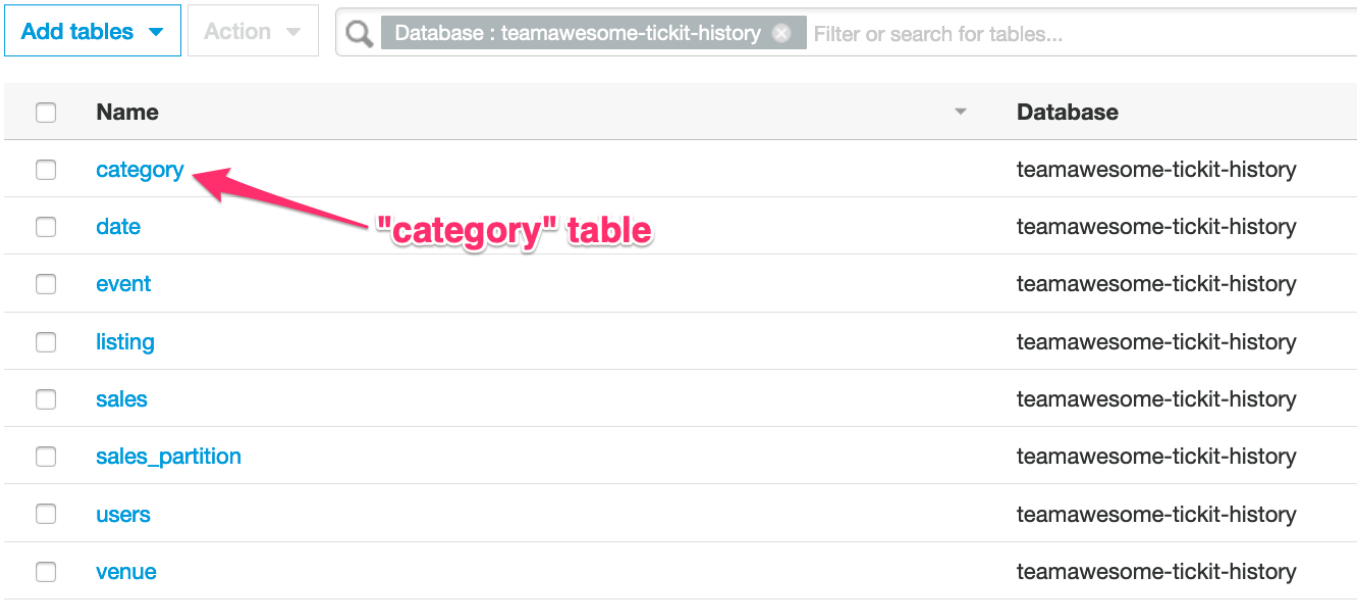
1. From within the AWS Glue console, select Databases from the left panel and navigate to your Tickit history database (i.e. teamawesome-tickit-history)



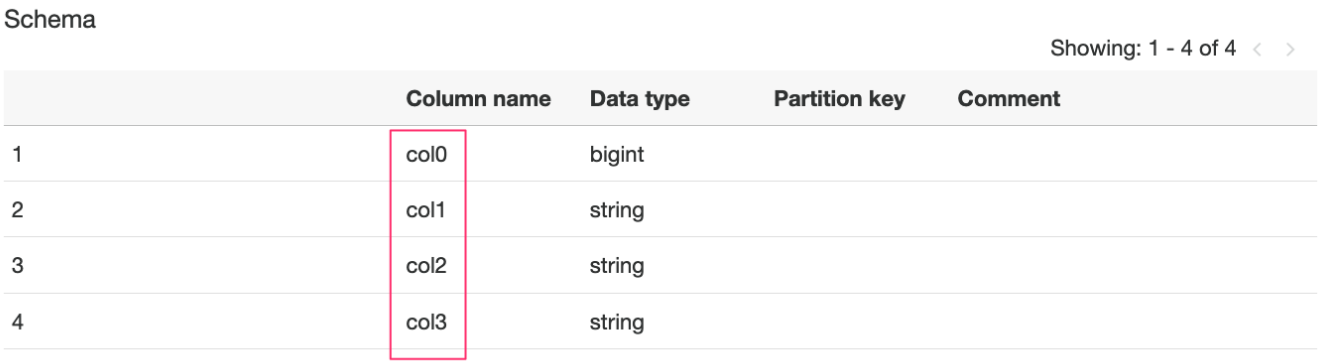
1. Click on the *teamawesome-tickit-history* link. This will display a list of all of the tables that have been generated by the Glue Crawler.



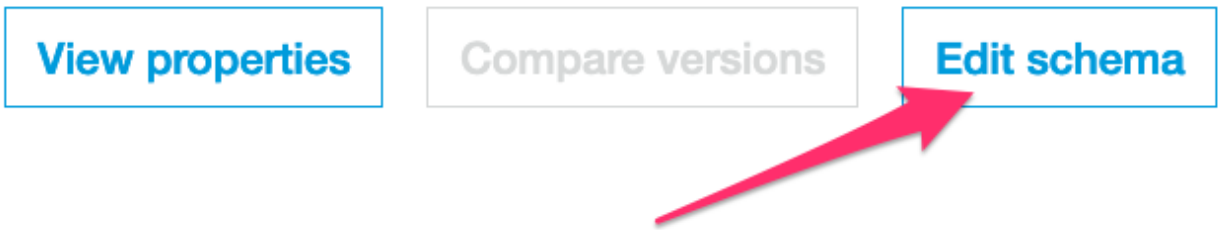
1. Click on the *category* table.



You will notice that the column names are listed as Col0, Col1, Col2–that is because the data files do not have a header row. In the next steps you will update the comunm names



1. Click on the **Edit schema** button (far upper right corner)



1. Use the information below as a guide to edit ***ONLY*** the **Column names** for the *category* schema, (do ***NOT*** change the types, just the column names):

	Column name	Data type
1	col0	bigint
2	col1	string
3	col2	string
4	col3	string

CATEGORY table

| Column Name (old) | Column Name (new) |

|—|---|

| col0 | CATID |

| col1 | CATGROUP |

| col2 | CATNAME |

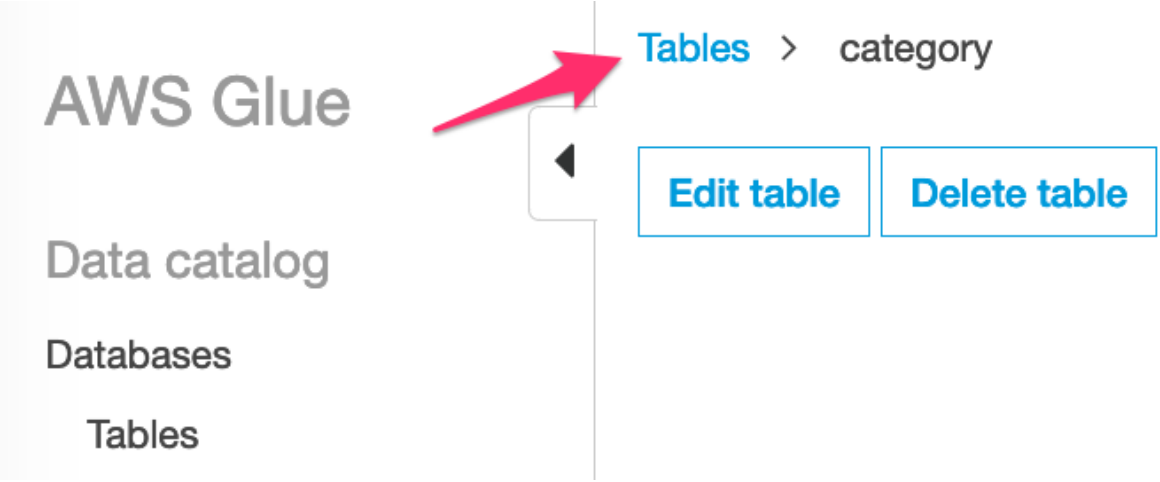
| col3 | CATDESC |

1. When you are finished editing the Category schema your schema must look like this:

	Column name	Data type
1	CATID	bigint
2	CATGROUP	string
3	CATNAME	string
4	CATDESC	string

1. Click the **Save** button.

2. To return to your tables list click on the **Tables** link



1. Using the same steps, repeat this process to update the following tables in your *teamawesome-tickit-history* database. Remember, you are just updating the **COLUMN NAMES**, not changing the data types.

- DATE
- EVENT
- LISTING
- SALES
- USERS
- VENUE

DATE Table

| Column Name (old) | Column Name (new) |

|—|---|

| col0 | DATEID |

| col1 | CALDATE |

| col2 | DAY |

| col3 | WEEK |

| col4 | MONTH |

| col5 | QTR |

| col6 | YEAR |

| col7 | HOLIDAY |

EVENT Table

| Column Name (old) | Column Name (new) |

|—|---|

| col0 | EVENTID |

| col1 | VENUEID |

| col2 | CATID |

| col3 | DATEID |

| col4 | EVENTNAME |

| col5 | STARTTIME |

LISTING Table

| Column Name (old) | Column Name (new) |

|—|---|

| col0 | LISTID |

| col1 | SELLERID |

| col2 | EVENTID |

| col3 | DATEID |

| col4 | NUMTICKETS |

| col5 | PRICEPERTICKET |

| col6 | TOTALPRICE |

| col7 | LISTTIME |

SALES Table

| Column Name (old) | Column Name (new) |

|—|---|

| col0 | SALESID |

| col1 | LISTID |

| col2 | SELLERID |

| col3 | BUYERID |

| col4 | EVENTID |

| col5 | DATEID |

| col6 | QTYSOLD |

| col7 | PRICEPAID |

| col8 | COMMISSION |

| col9 | SALETIME |

USERS Table

| Column Name (old) | Column Name (new) |

|—|---|

| col0 | USERID |

| col1 | USERNAME |

| col2 | FIRSTNAME |

| col3 | LASTNAME |

| col4 | CITY |

| col5 | STATE |

| col6 | EMAIL |

| col7 | PHONE |

| col8 | LIKESPORTS |

| col9 | LIKETHEATRE |

| col10 | LIKECONCERTS |

| col11 | LIKEJAZZ |

| col12 | LIKECLASSICAL |

| col13 | LIKEOPERA |

| col14 | LIKEROCK |

| col15 | LIKEVEGAS |

| col16 | LIKEBROADWAY |

| col17 | LIKEMUSICALS |

VENUE Table

| Column Name (old) | Column Name (new) |

|—|---|

| col0 | VENUEID |

| col1 | VENUENAME |

| col2 | VENUECITY |

| col3 | VENUESTATE |

| col4 | VENUESEATS |

##Lab 3: Querying your Data Lake with Amazon Athena

The ticketing team at UnicornNation has heard about the work you did in setting up the data catalog. They have some data that they need urgently and need your help in setting up and running these queries.

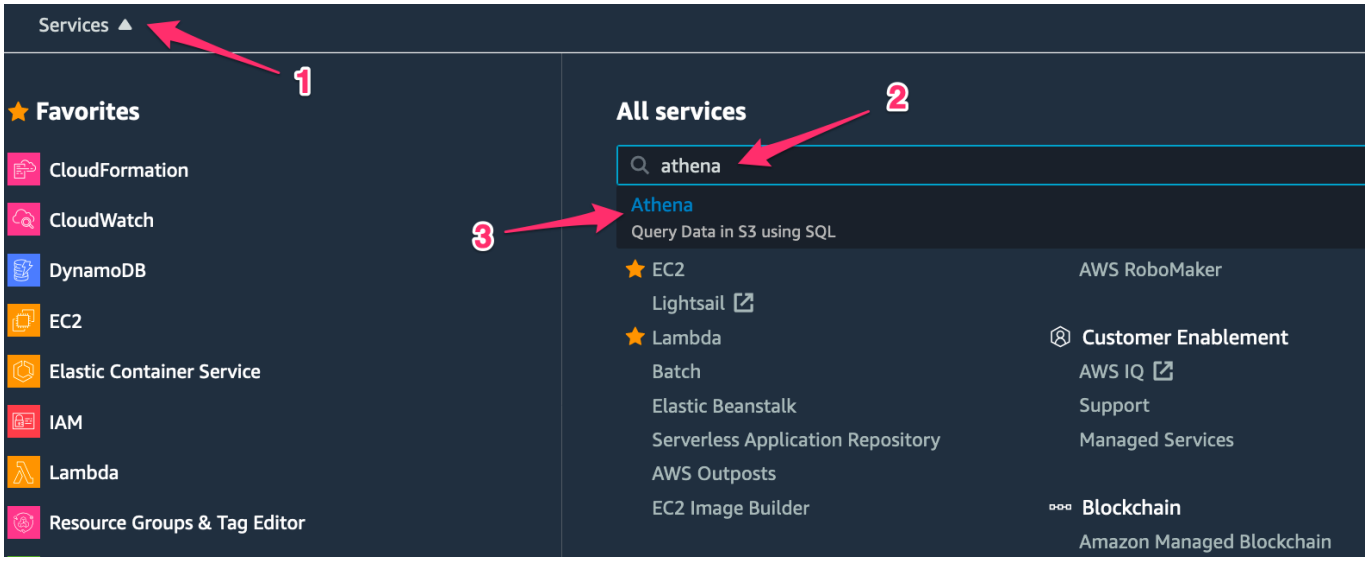
In this lab, you are going to use Amazon Athena to create some queries, which you will then save to make it easy for users to run and consume.

As part of the lab, you will also be running your own queries to help users answer some basic questions around ticket sales, customers and more.

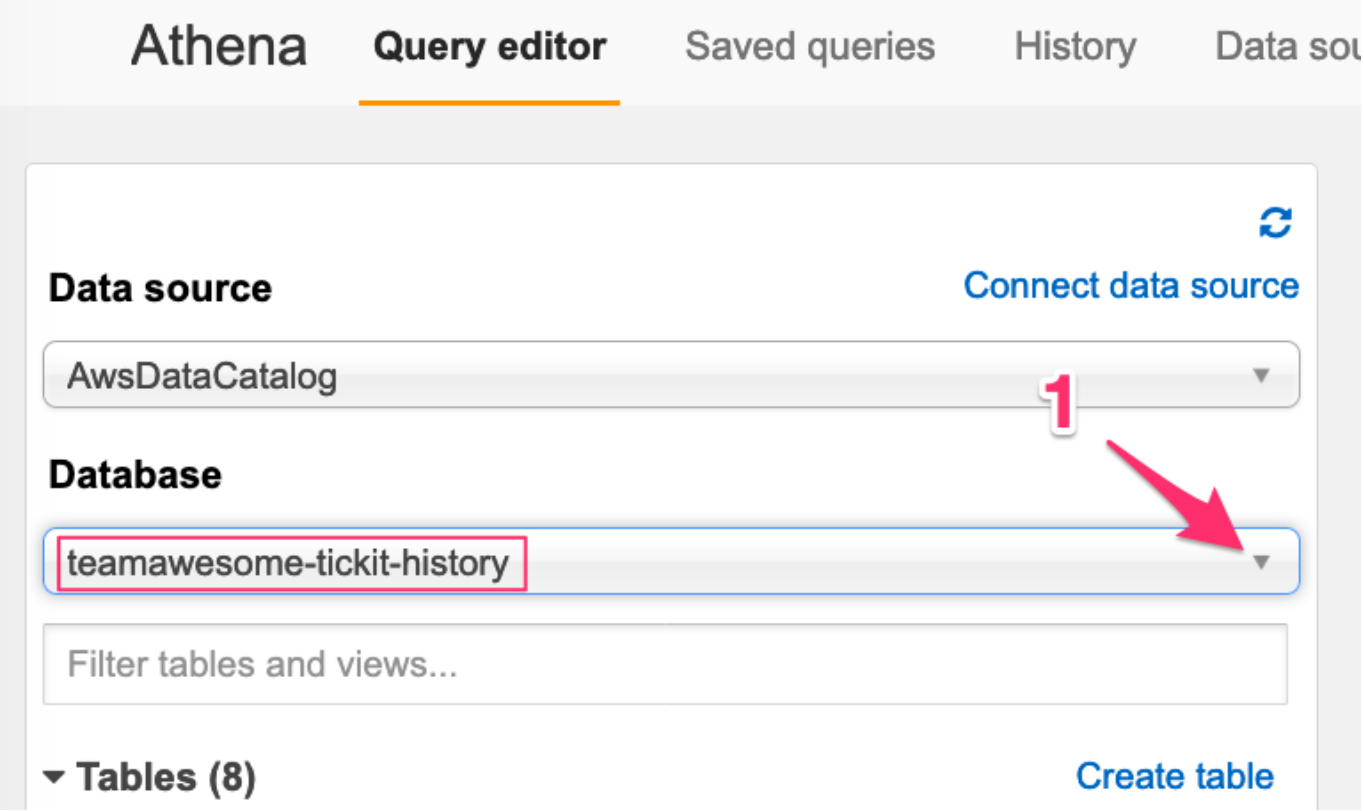
To run a query using Amazon Athena, follow these steps:

1. Login to the AWS Console using the login URL and credentials provided. Once you are logged in, check in the upper right-hand corner that you are using **Oregon** region. If not, use the drop-down list to change to the **Oregon** region.

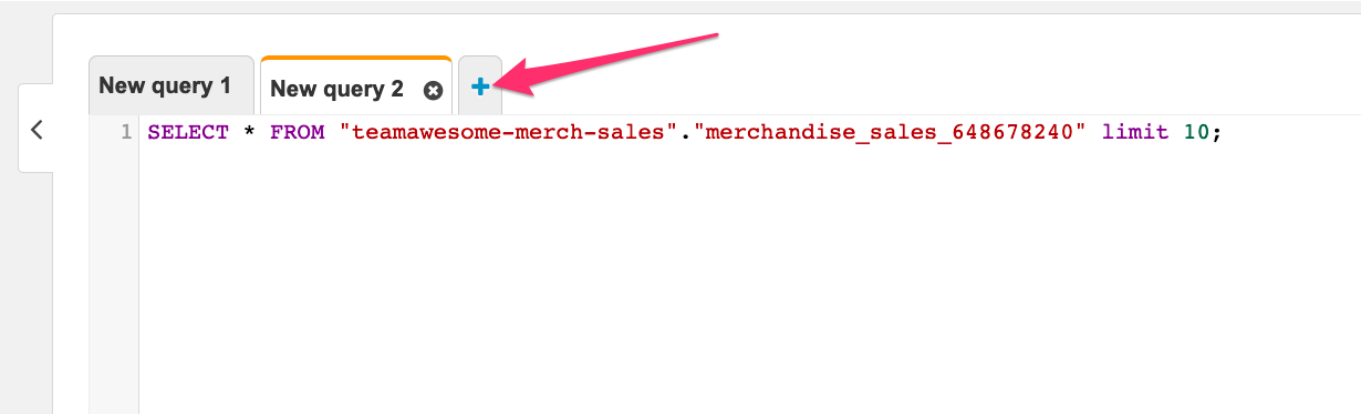
2. From the console, navigate to the Athena service



1. Using the **Database** drop-down list, change the database to point to your Tickit database **teamawesome-tickit-history**



1. You could open a new query text box



1. To run a query, paste your below sql script into the new text box and click the **Run Query** button

2. Use the query text below, run each query to answer these questions:

Question 1: Using the following query, what were the Top 5 ticket sellers for events in San Diego in 2008?

```
select sellerid, username, city, firstname ||' '|| lastname as fullname, sum(qtysol
```

```
from sales, date, users

where sales.sellerid = users.userid

and sales.dateid = date.dateid

and year = 2008

and city = 'San Diego'

group by sellerid, username, city, firstname ||' '|| lastname

order by 5 desc

limit 5;
```

Question 2: Using the following query, who were the buyers AND sellers for ticket transactions that cost $10,000 or more?

```
select listid, lastname, firstname, username,

pricepaid as price, 'S' as buyorsell

from sales, users

where sales.sellerid=users.userid

and pricepaid >=10000

union

select listid, lastname, firstname, username, pricepaid,

'B' as buyorsell

from sales, users

where sales.buyerid=users.userid

and pricepaid >=10000
```

```
order by 1, 2, 3, 4, 5;
```

##Lab 4: Transforming Data with AWS Glue

The IT team at UnicornNation is looking for a way to reduce their AWS spend for this project. After reviewing the file formats they are using, they have decided that for the Merchandise Sales data, they are going to change the file format from CSV to Parquet.

Parquet is a columnar, compressed format and will help them reduce the amount of data that is scanned when using Amazon Athena.

In this lab, you are going to create a Glue Job to convert the existing CSV data files to Parquet.

1. Login to the AWS Console using the login URL and credentials provided. Once you are logged in, check in the upper right-hand corner that you are using **Oregon** region. If not, use the drop-down list to change to the **Oregon** region.

2. From the console, navigate to the **AWS Glue** service

3. Under the ETL menu, select **Jobs** and then click the button for **Add Job**

4. For the **Name** of the Job, use type `teamawesome-convert-to-parquet`

5. Under IAM role, using the drop-down list, select **GlueServiceRole** IAM role

Leave the rest of the parameters as *Default*

1. Click **Next**

2. For your data source, select the database you created for the Merchandise Sales data, look for a data source named similar to `merchandise_sales_xxxxxxx` and then click the **Next** button

3. In the **Choose a transformation type** page, select **Change schema**, click the **Next** button

4. In the **Choose a data target** page, select **Create tables in your data target**

5. For the Data store, select **Amazon S3**

6. For the Format, select **Parquet**

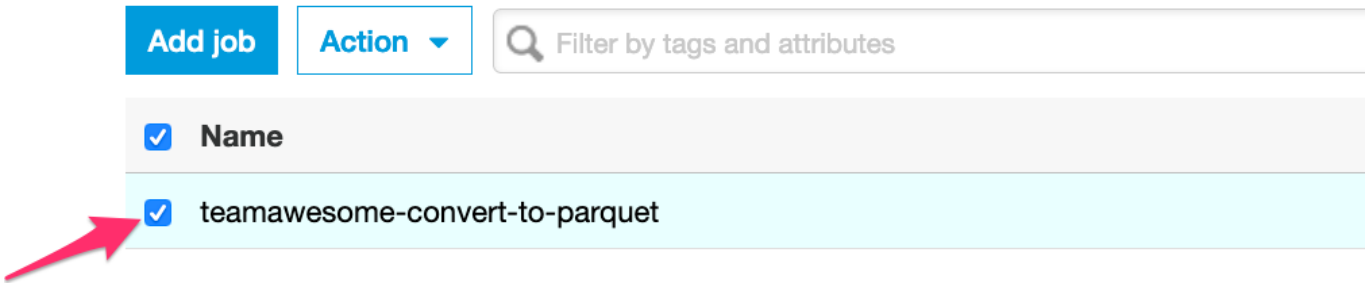
7. For the Target path, click on the small folder icon and select the S3 bucket `teamawesome-merch-sales-parquet-xxxxxxx` , click **Select**, then click **Next**

8. In the **Map the source columns to target columns** page, leave *defaults* and click **Save job and edit script**

9. Close **Script editor tips** window

10. From the toolbar, click the **Save** button, and then click the **X** icon (far right) to return to the list of jobs

11. Locate the job you created and click to select the job



1. Click the **Action** button and select **Run Job**, this will start your Glue ETL job

2. Click again on the checkbox next to your ETL job to see the status panel.

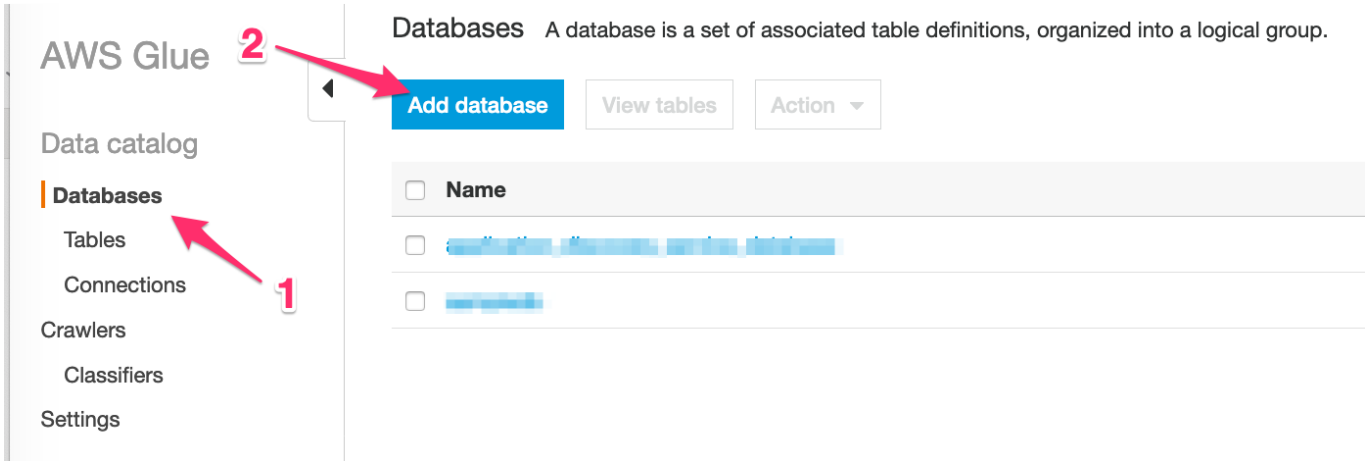
3. You will notice the job status as *Running*, wait a couple of minutes until it changes to *Succeeded*

4. Navigate to the S3 console (Services > S3) and search for the bucket `teamawesome-merch-sales-parquet-xxxxxxxx` . Verify that the parquet files were successfully created in the bucket

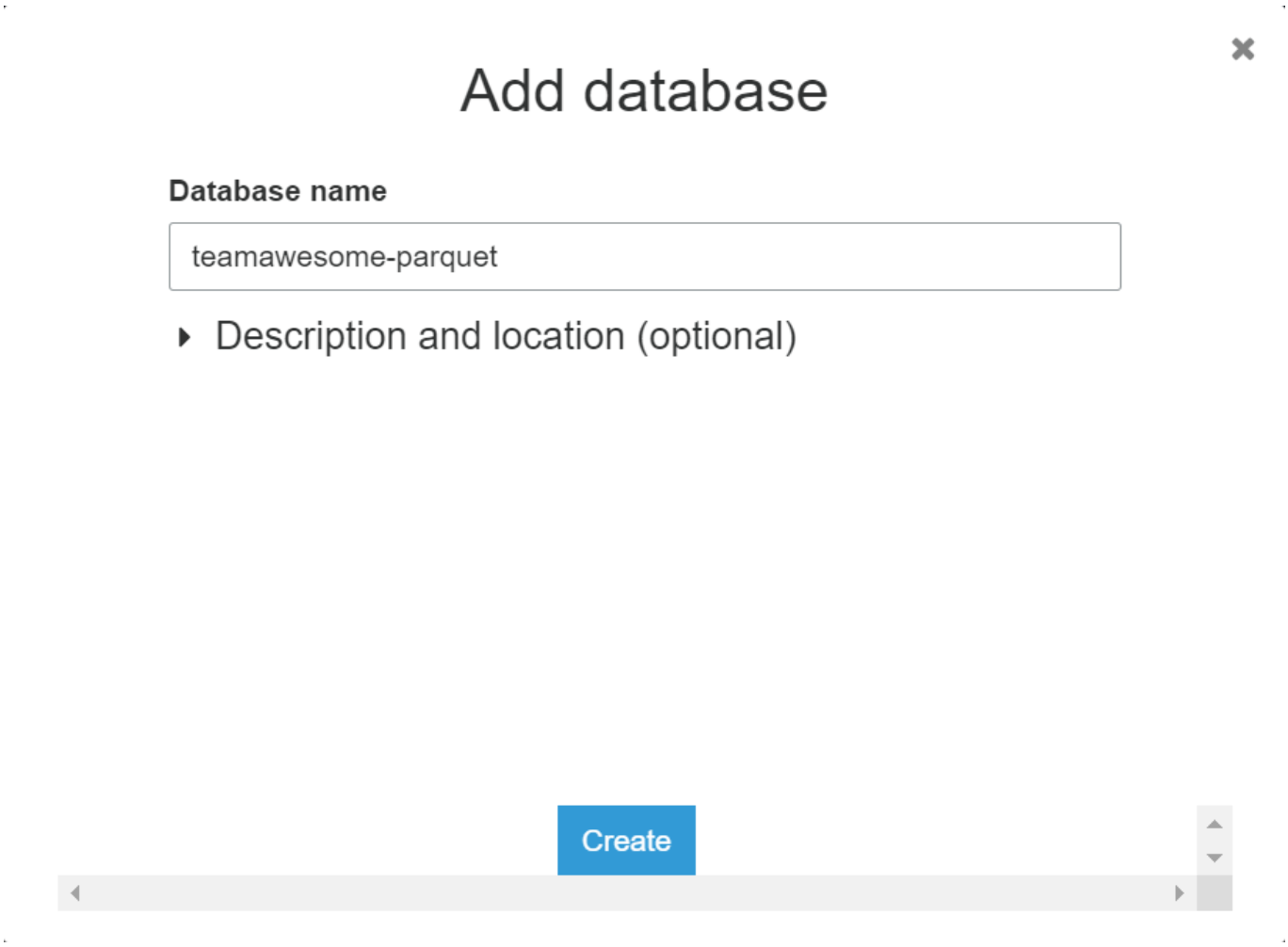
With the Parquet files created, you could then crawl those files and start using the data with Athena. Because Parquet is a columnar, compressed format, there will be less data scanned, reducing the cost of your Athena queries.

Now lets see how query output differs between csv and parquet. Parquet files have been loaded in to `teamawesome-merch-sales-parquet-xxxxxxxx` . We need to build the metadata using an AWS Glue Data crawler.

1. From the AWS Glue Data Catalog, navigate to **Databases** and click **Add Database**

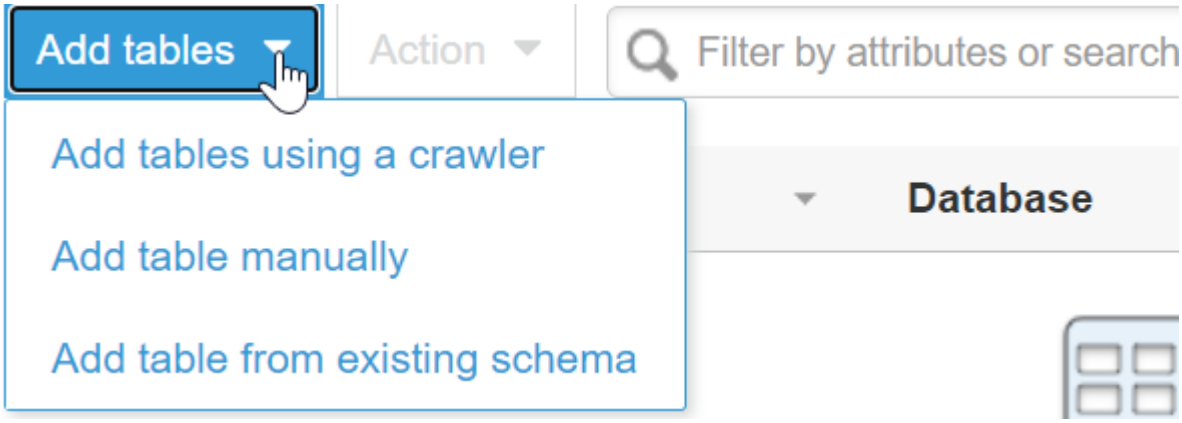


1. Enter a database name, type `teamawesome-parquet` and click **Create**



1. Now that your database has been created, click on the **Tables** menu in the left-hand menu

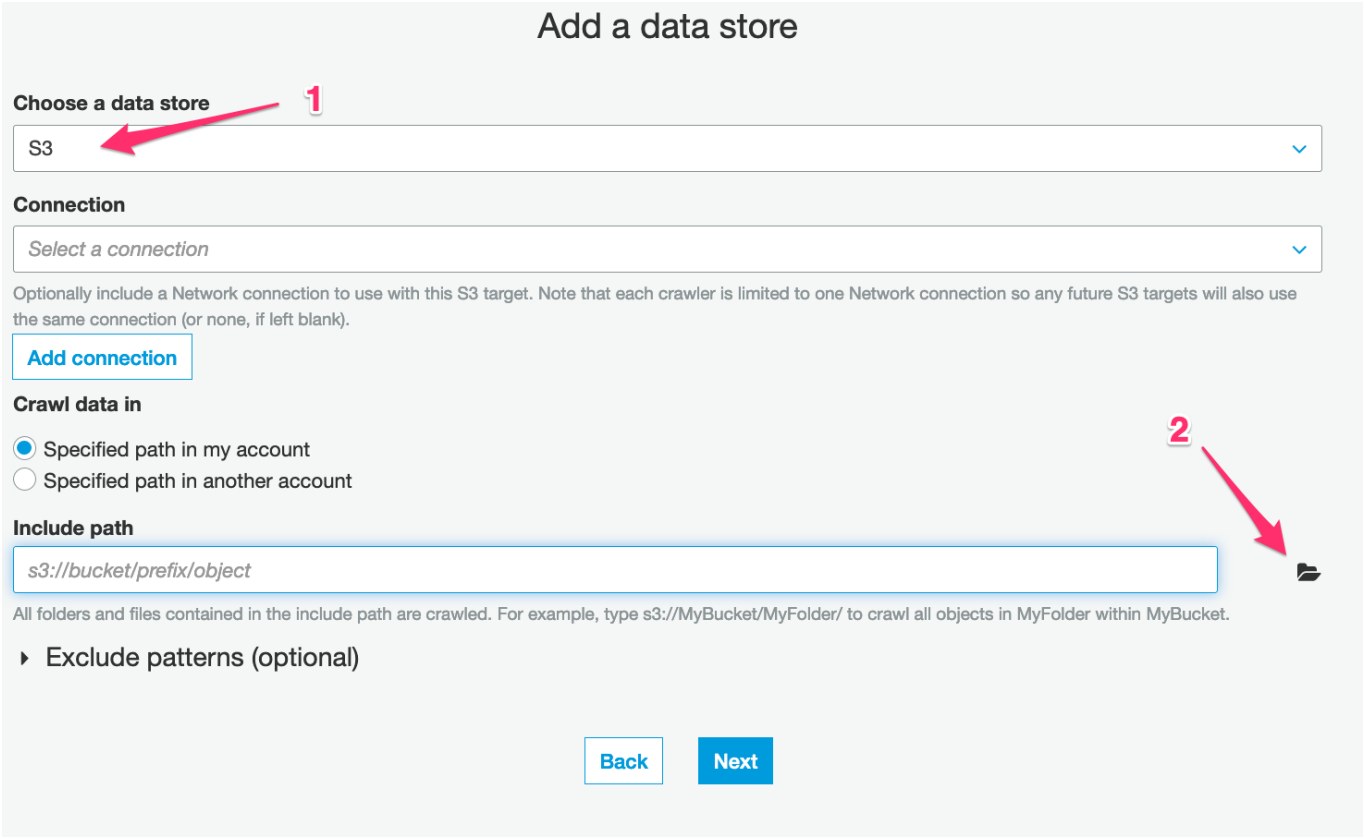
2. Select the drop down to **Add Tables** > **Add tables using a crawler**



1. Enter a name for your crawler, type `teamawesome-parquet-crawler` and click **Next**

2. For Crawler source type select **Data stores**, then click **Next**

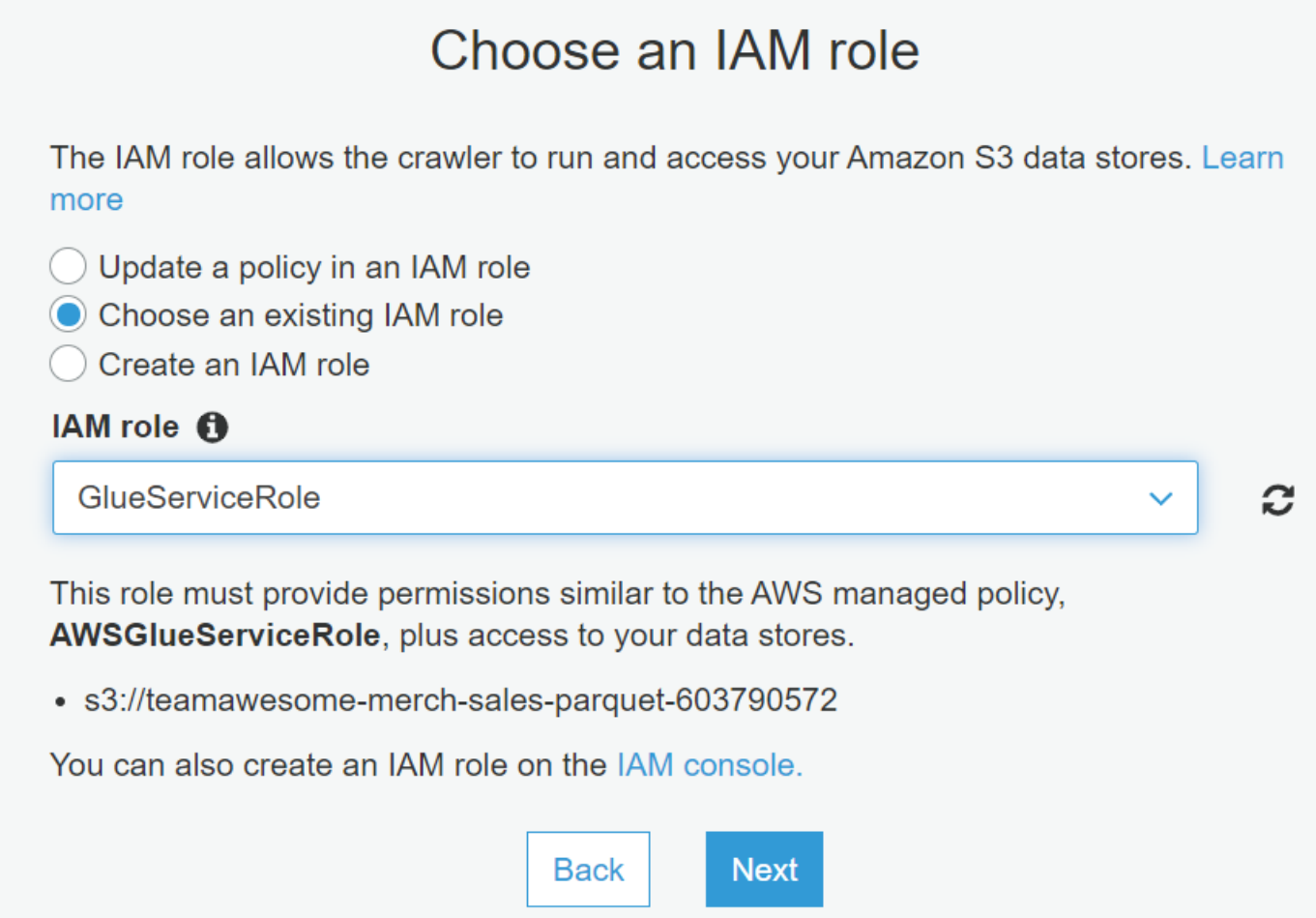
3. For your data store, select **S3** and below **Include path**, click on the little folder icon



1. This will open a new window, look for the Merchandise Sales bucket, it's name will follow this pattern: `teamawesome-merch-sales-parquet-xxxxxxxxx` , click **Select** and then click **Next**.

2. For **Add another data store**, take the default of *No* and click **Next**

3. Next, when setting up an IAM role, click on **Choose an existing IAM Role**, then under IAM role select role name **GlueServiceRole**. Then click **Next**.

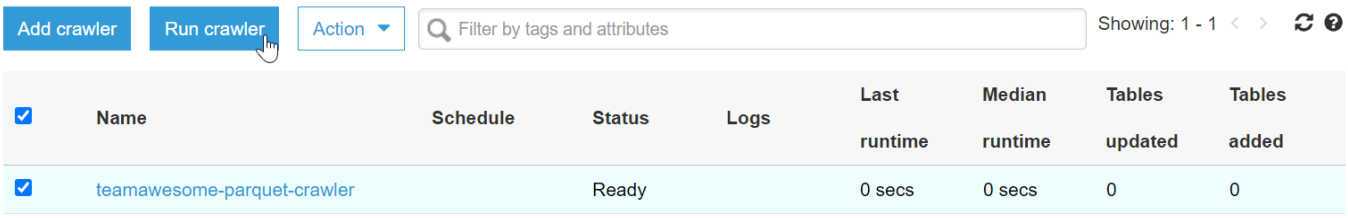


1. For the Frequency, leave the default of **Run on demand** and click **Next**

2. In **Configure the crawler's output**, select the Database **teamawesome-parquet** and click **Next**

3. Finally, review the settings for your crawler and click **Finish**. This should take you to a list of crawlers that have been created

4. Select the crawler you just created and click on **Run crawler**



Watch the Crawler console for your job to finish successfully (it should take ~2 minutes). *Status* must change from **Starting** to **Ready**

1. Using the navigation menu on the left, navigate to **Tables**

2. You should see a new table has been created, copy and paste the table name on a notepad.

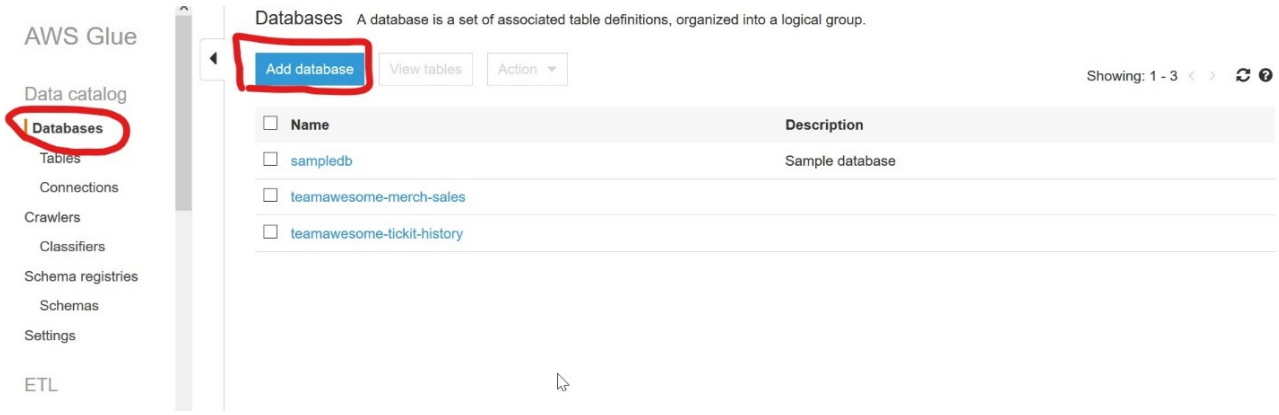
Visit **Amazon Athena** to execute the following queries:

Query 1: `SELECT * FROM "teamawesome-merch-sales"."merchandise_sales_xxxxxxxxx" limit 10;`
Query 2: `SELECT * FROM "teamawesome-parquet"."teamawesome_merch_sales_parquet_xxxxxxxxx" limit 10;`

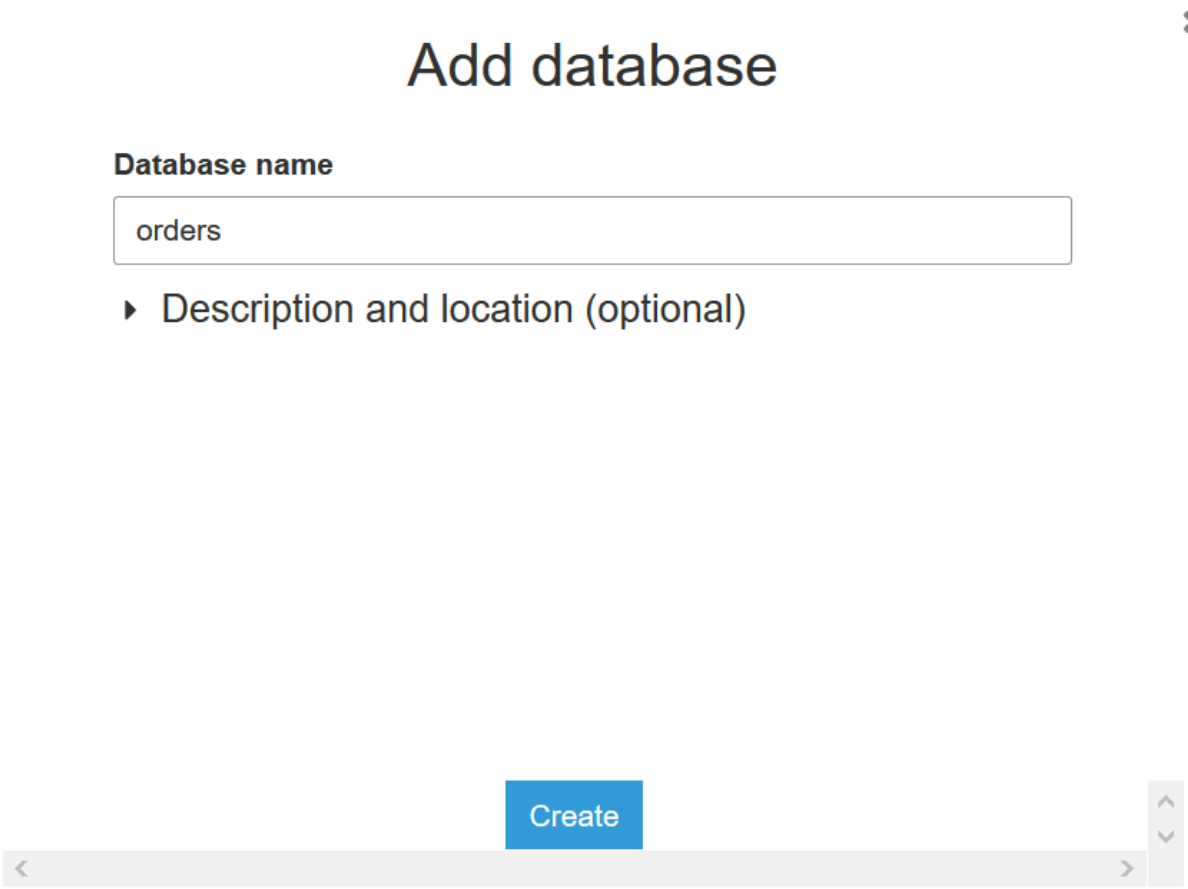
Note the run time and Data scanned.

##Lab 5: Using AWS Glue Studio

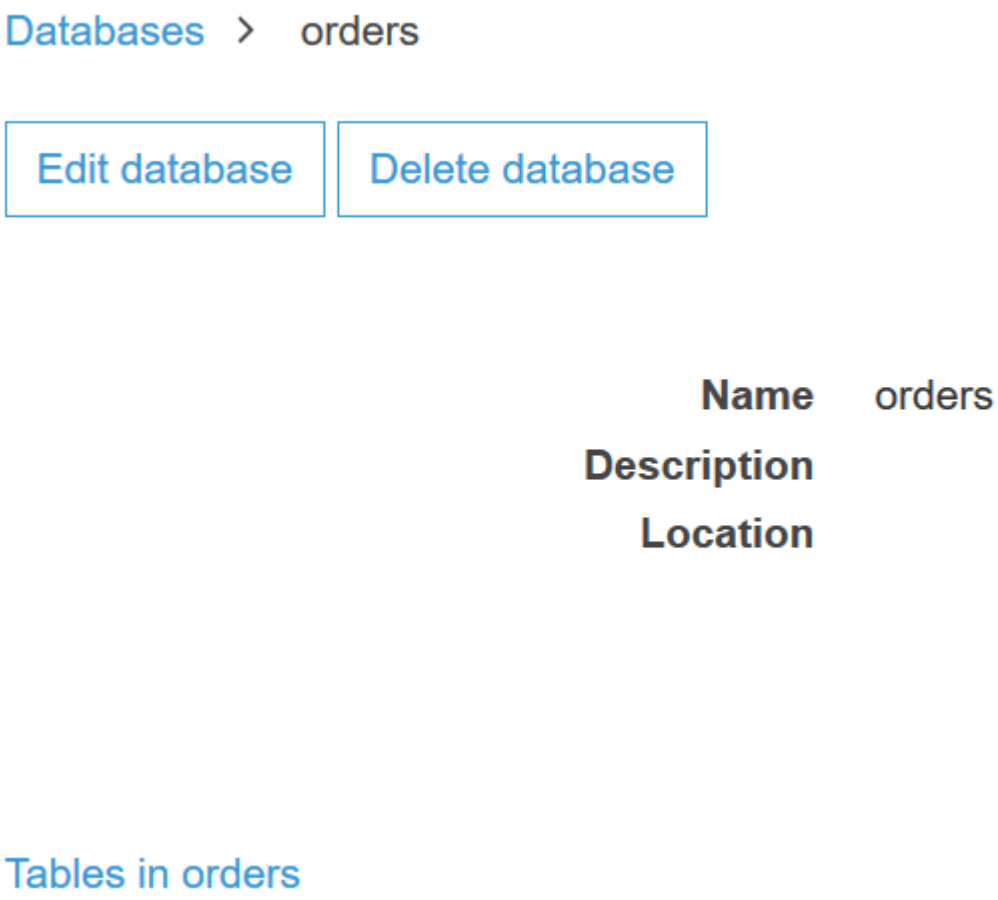
1. From **AWS Management Console**, select **AWS Glue**, Select **Databases** under Data Catalog, Click on **Add database**.



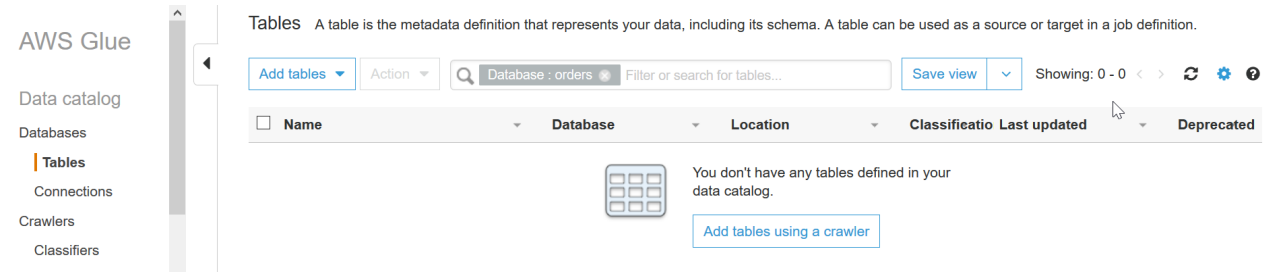
2. Database Name would be **orders**.



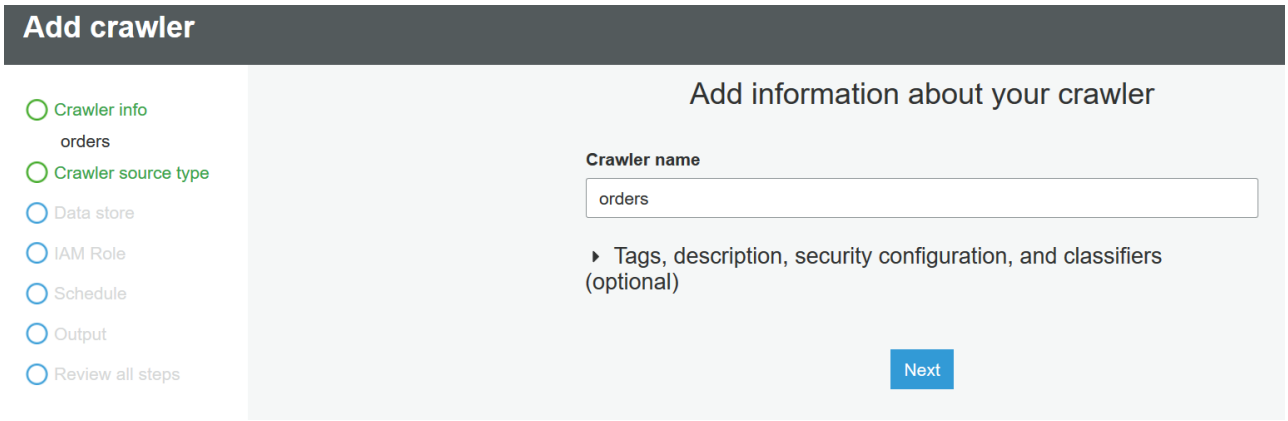
3. Select "**Tables in orders**"



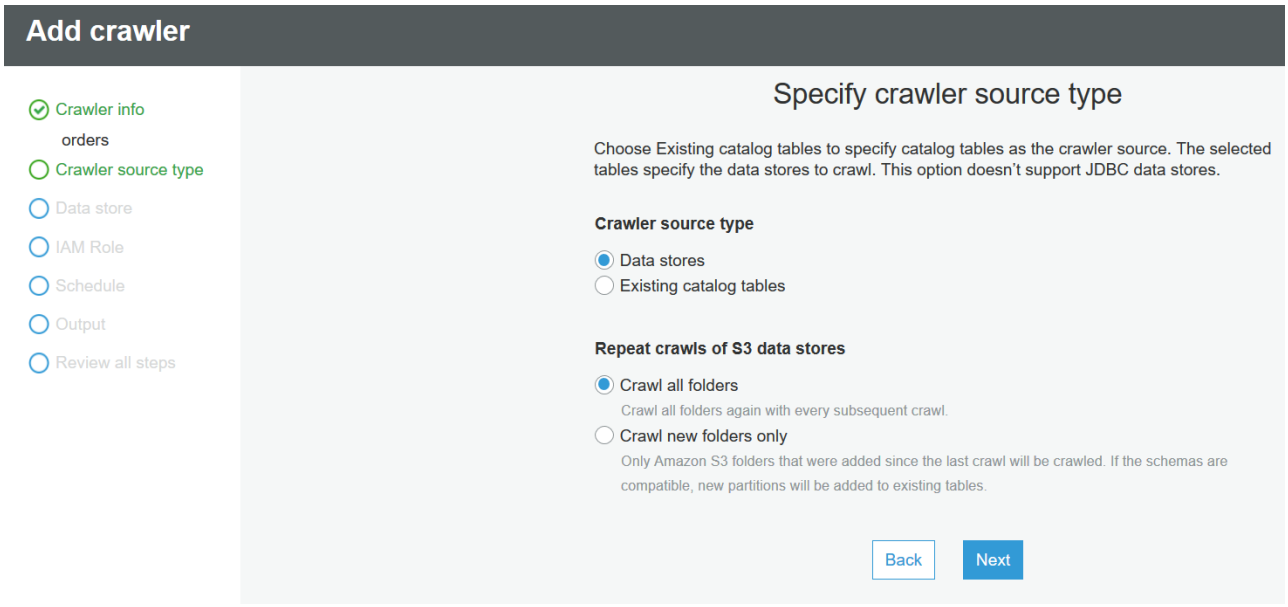
4. Select "**Add tables using a crawler**"



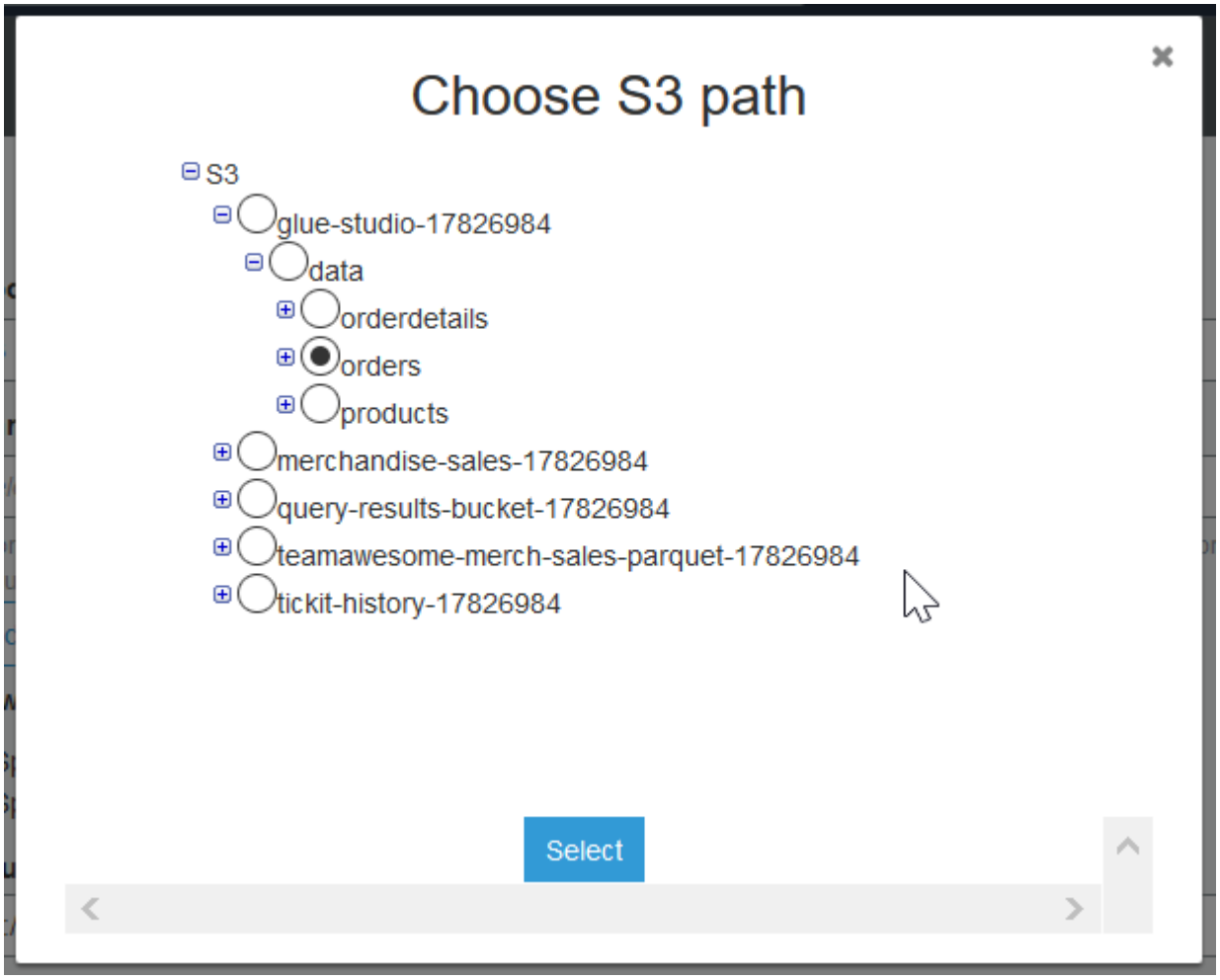
5. Crawler name is "**orders**"

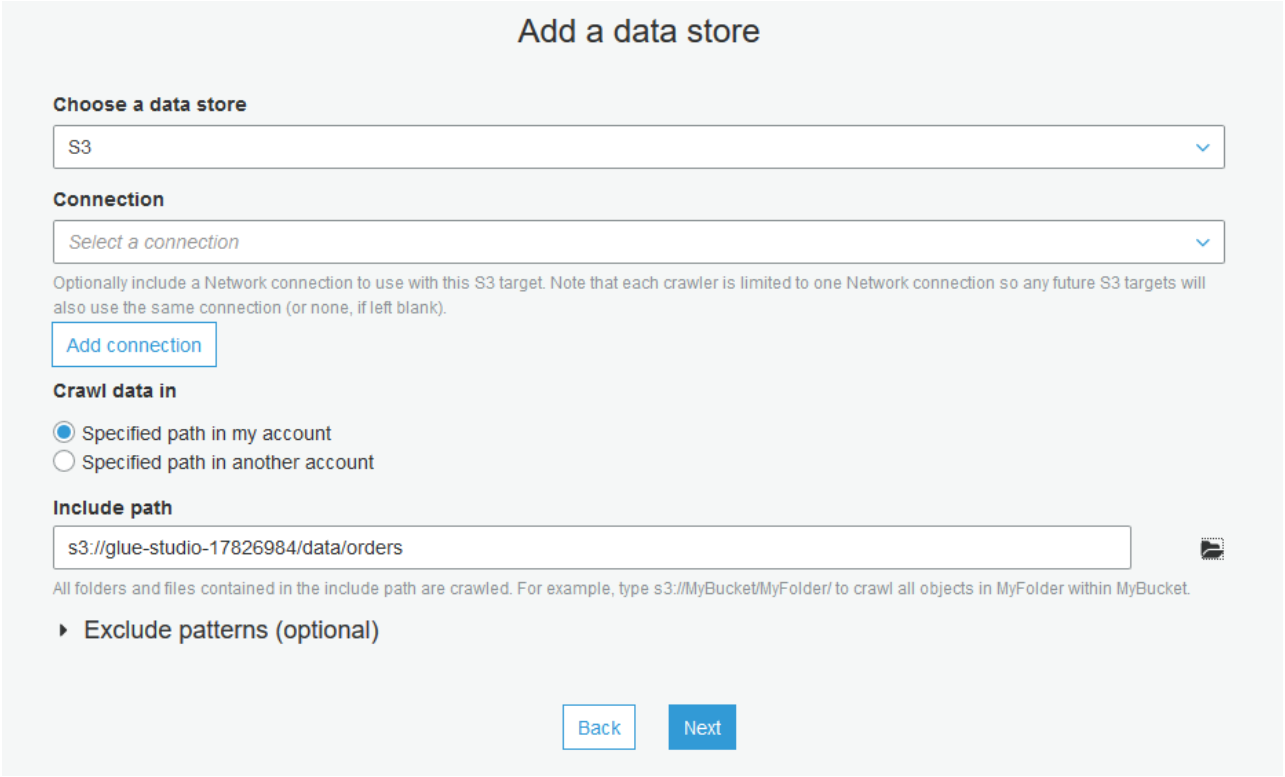


6. Continue with defaults

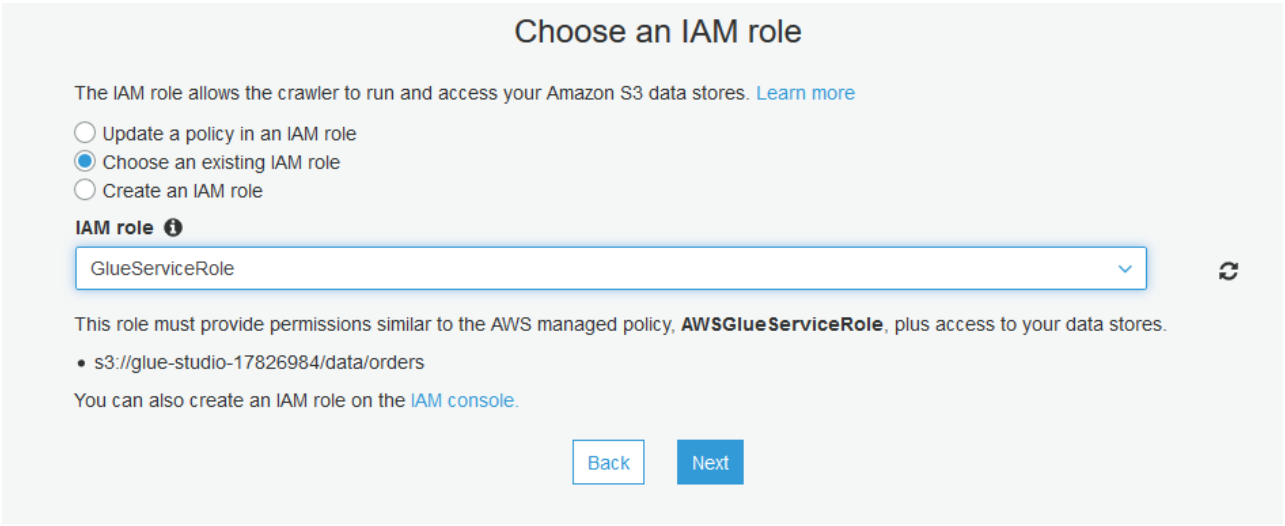


7. Data store is s3 and include path

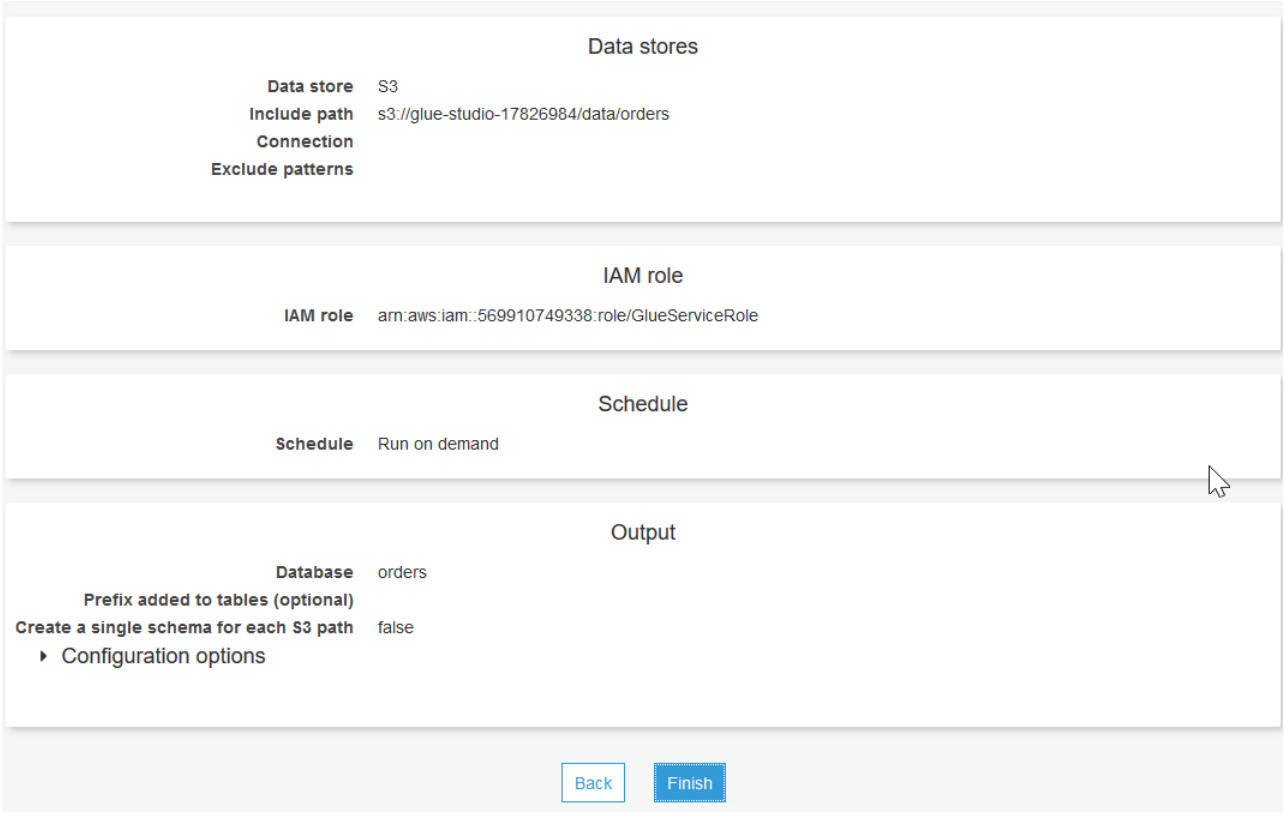




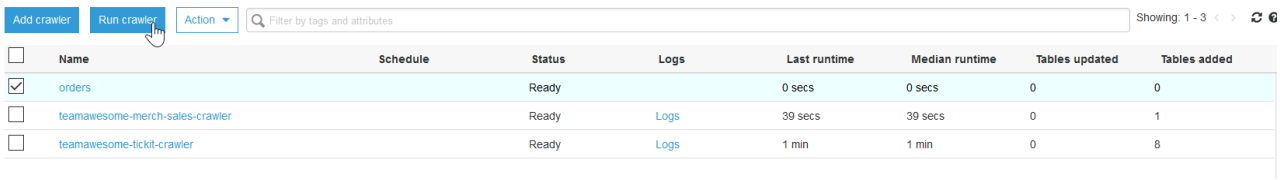
8. Select IAM role as **GlueServiceRole**



9. Confirm all settings and select **Finish**



10. Select **orders** and **Run Crawler**.

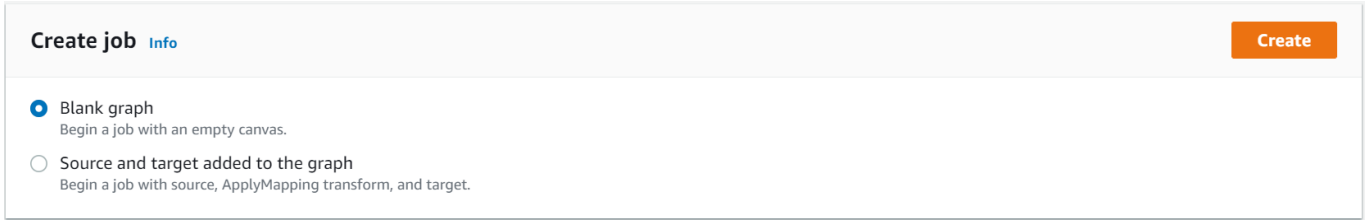


11. Continue to build the following crawlers:

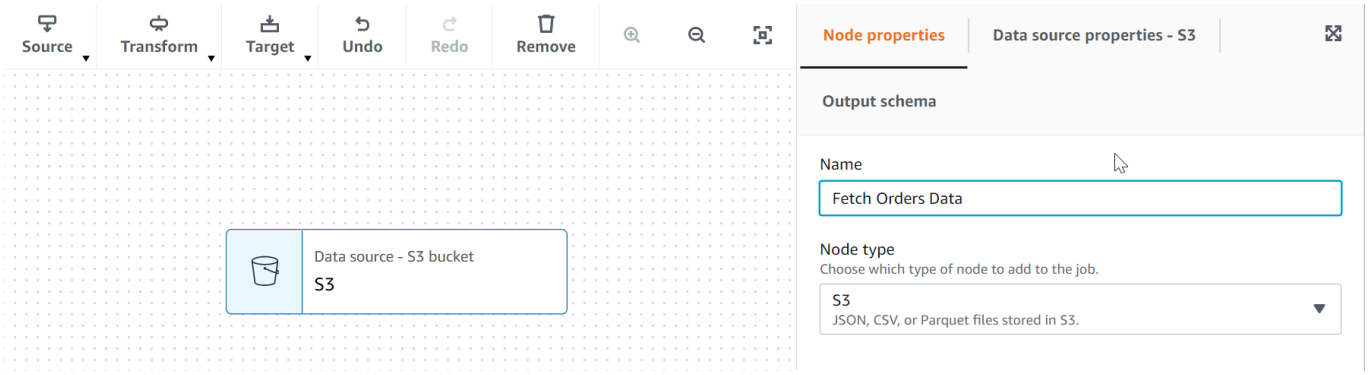
```
Crawler Name : orderdetails
Role : GlueServiceRole
Database Name : orders
S3 Path : s3://glue-studio-xxxxxxxx/data/orderdetails/
```

```
Crawler Name : products
Role : GlueServiceRole
Database Name : orders
S3 Path : s3://glue-studio-xxxxxxxx/data/products/
```

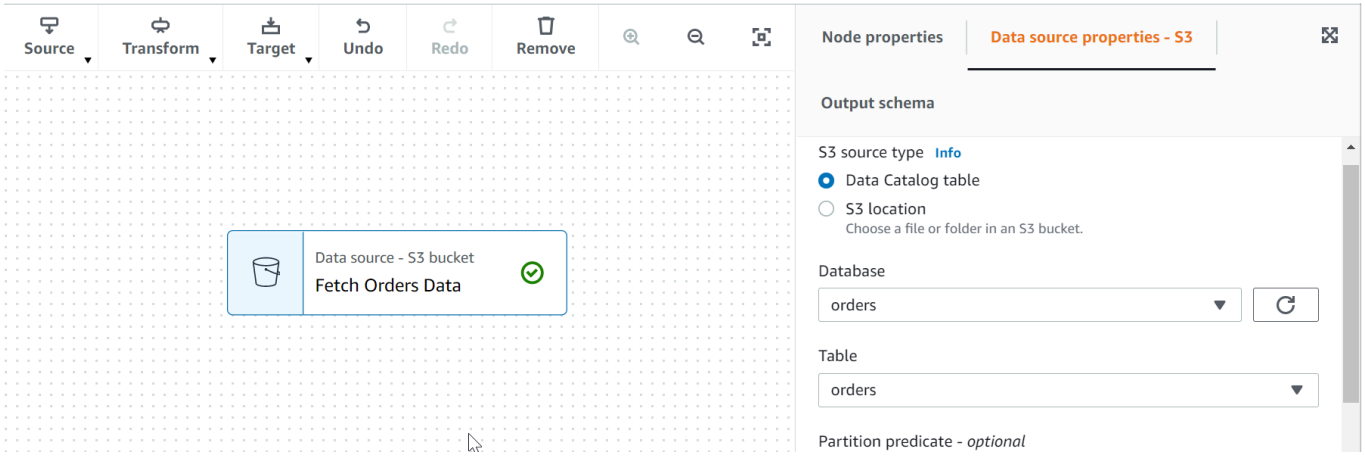
Using the AWS console open **AWS Glue** service and click on **AWS Glue Studio** using the left menu. Make sure you have Blank Graph selected. Click on **Create**.



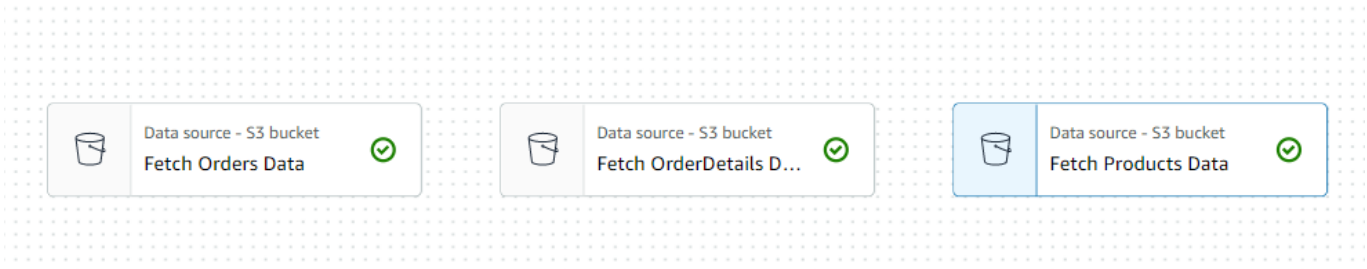
Start by creating the first Source Node-**Fetch Orders Data**



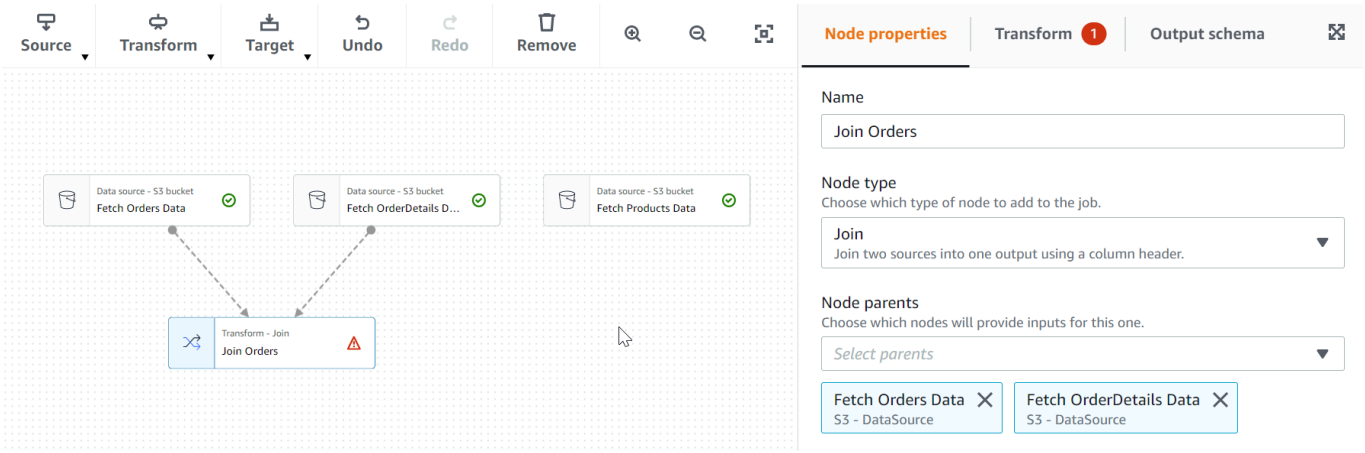
Make sure that **Fetch Orders Data** points to the orders table catalogued in Glue previously.



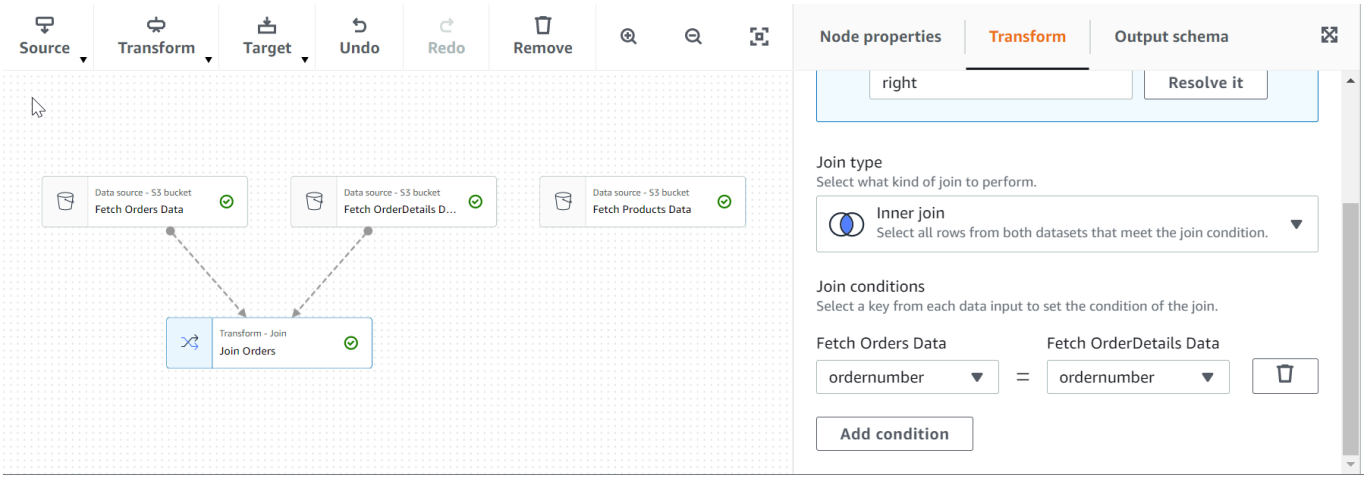
Using the same principles as above create the Source Node-**Fetch OrderDetails Data** as well as **Fetch Products Data**



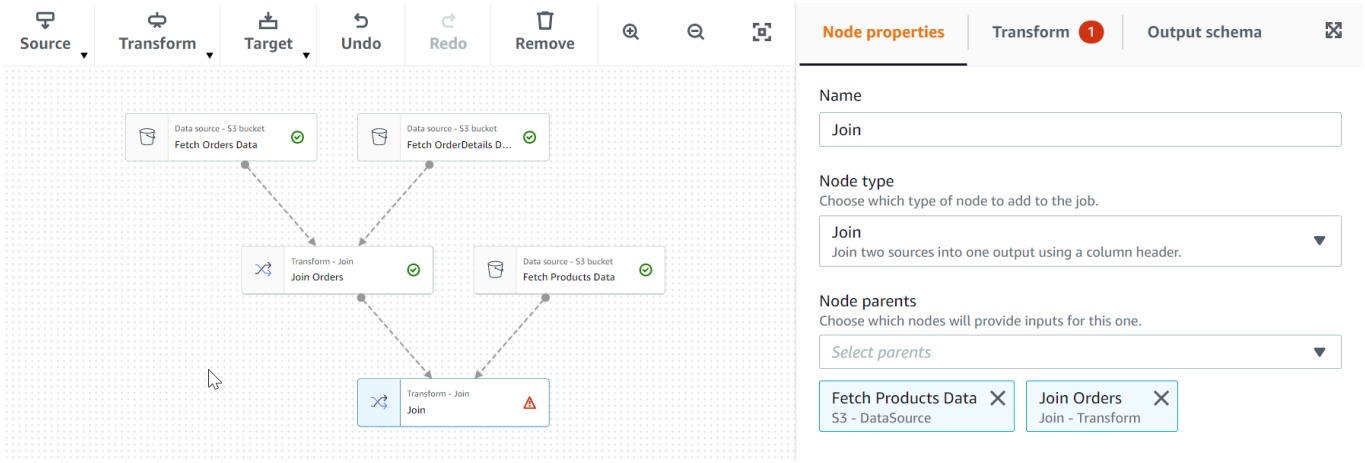
Now we will create a Transform Node that will join **Fetch Orders Data** to **Fetch OrderDetails Data.**

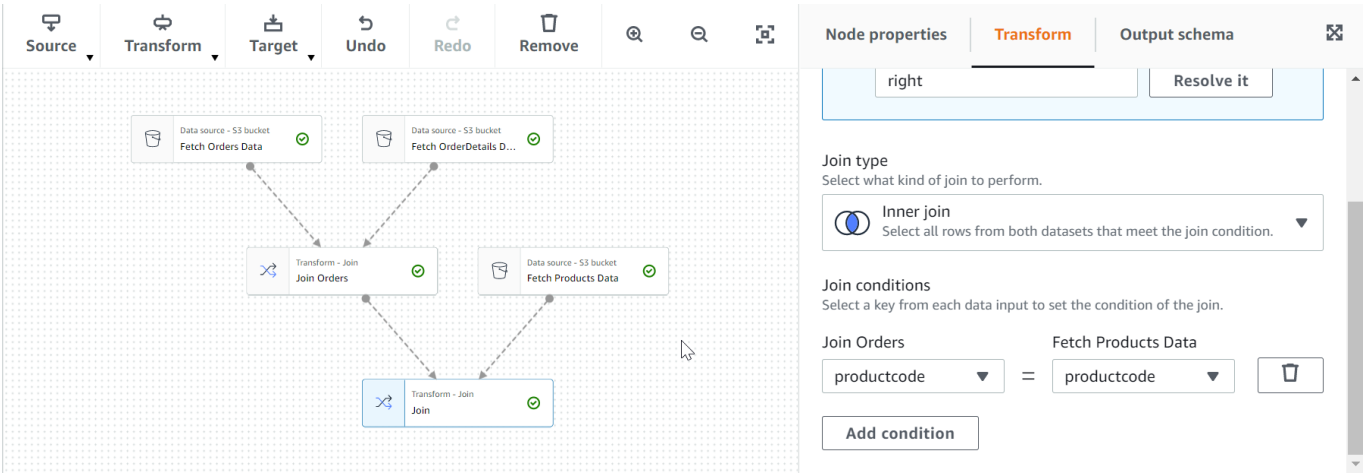


Notice how the joining condition is defined between the two tables as below.

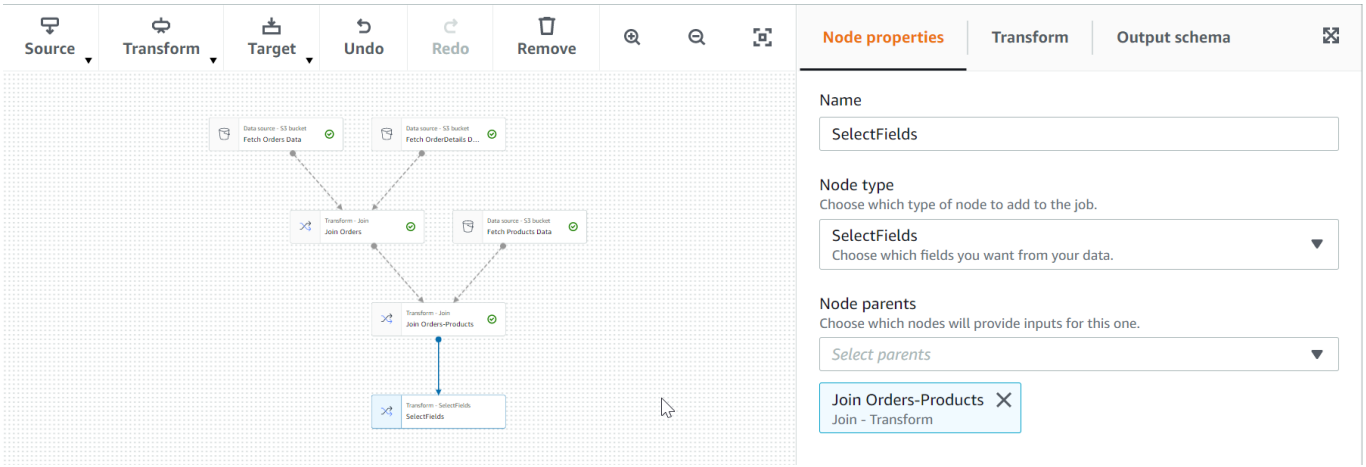


Using the same principles create a Transform Node that will join **Join Orders** to **Fetch Products Data.**

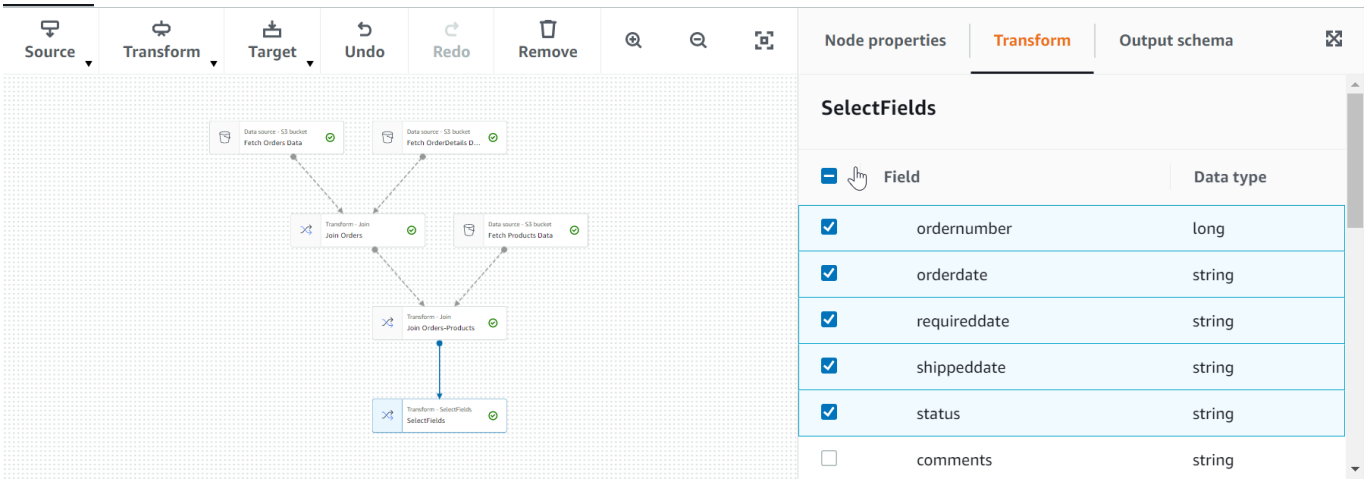




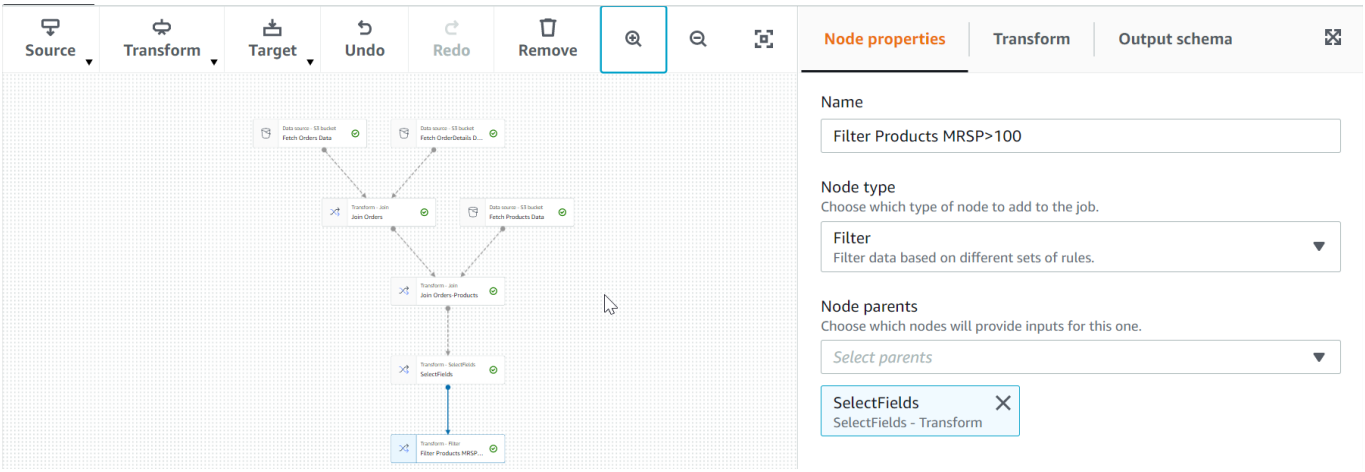
Since we want to select a subset of columns from the three table we can use the **Select Fields** Node



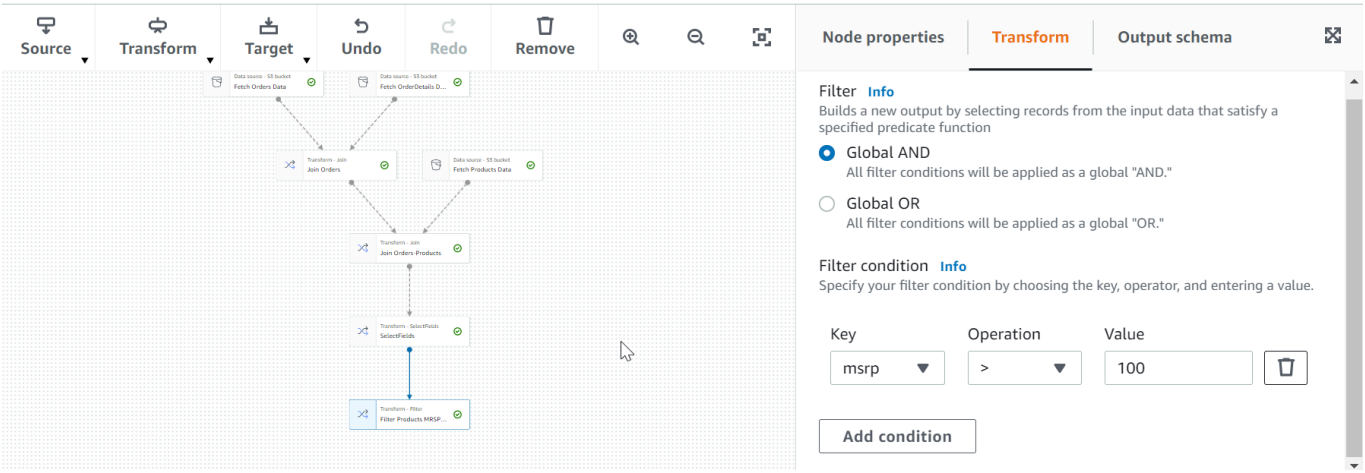
Notice how you can check boxes for fields that should be included in the final result set. Select `ordernumber` , `orderdate` , `requireddate` , `shippeddate` , `status` , `msrp`



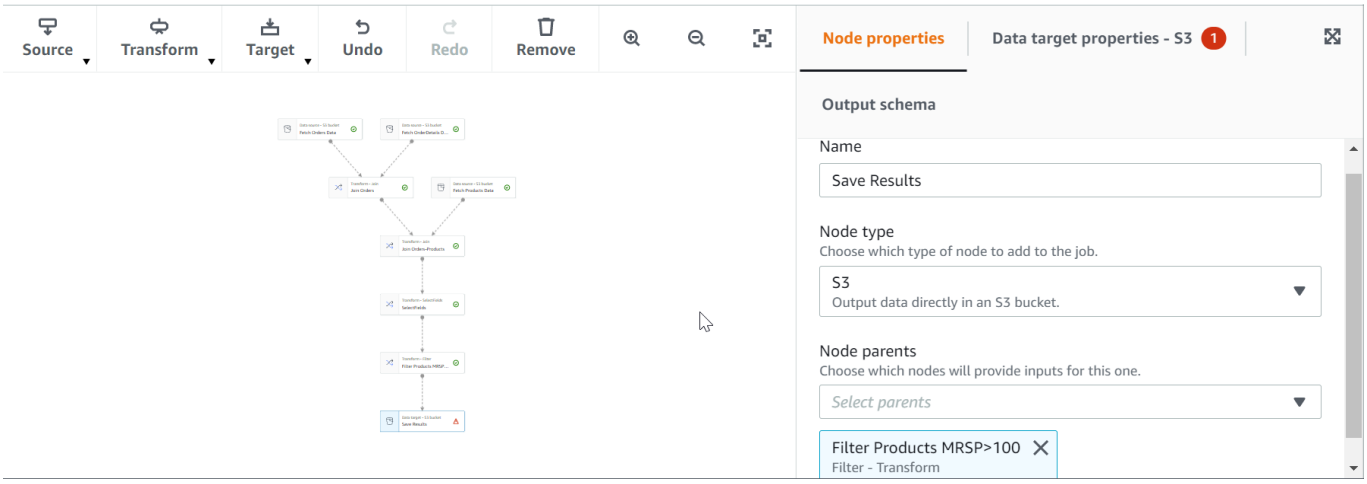
Now we would like to filter the products whose MSRP is greater than $100. This can be achieved by creating a **Filter Products MSRP>100** Node as below.

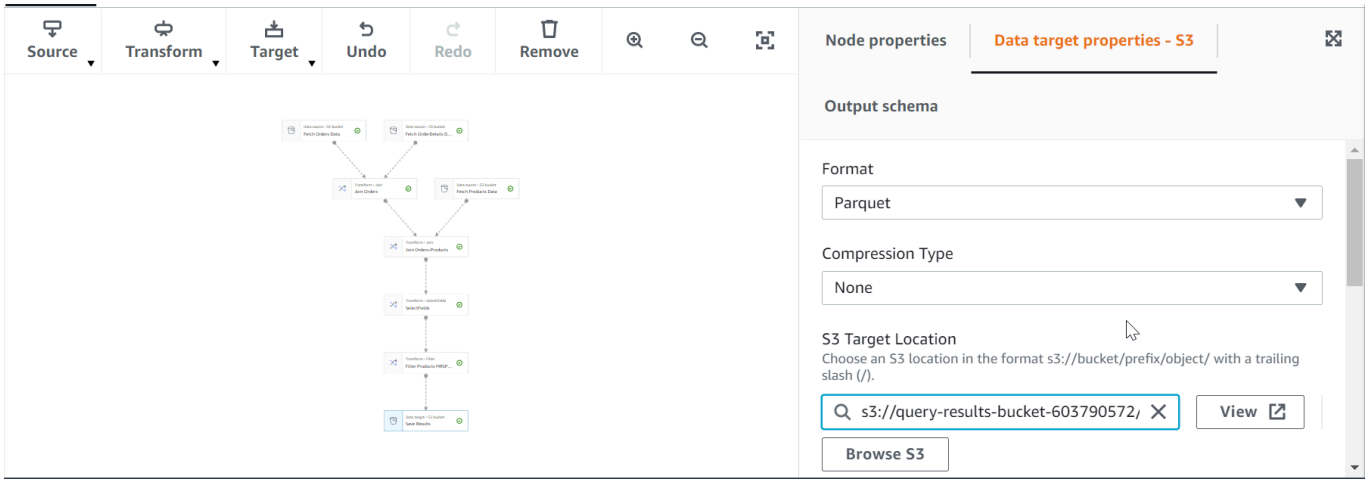


Notice how one or more filter condition can be defined.

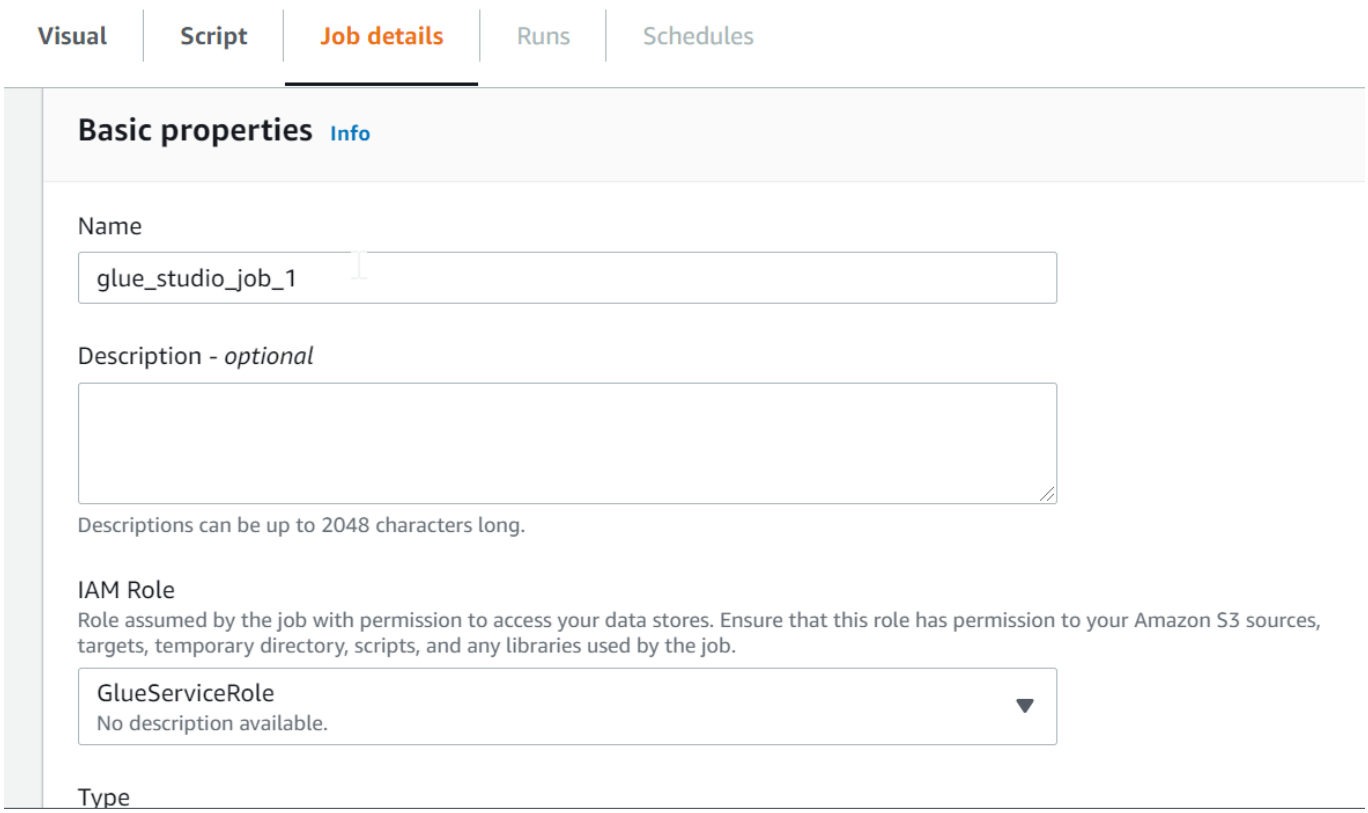


Finally, we want to save the result table to S3 in Parquet format. For this we create a Target Node-**Save Results**





Update the job name as `glue_studio_job_1` . Update IAM role to `GlueServiceRole` .



Use the Save button to save your ETL job. At this time you should be able to see that **AWS Glue Studio** has automatically generated the Spark code for you. Click on the **Script** menu to view the generated code.

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

```
## @params: [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)
## @type: DataSource
## @args: [database = "orders", table_name = "orderdetails", transformation_ctx =
"DataSource0"]
## @return: DataSource0
## @inputs: []
DataSource0 = glueContext.create_dynamic_frame.from_catalog(database = "orders",
table_name = "orderdetails", transformation_ctx = "DataSource0")
## @type: DataSource
## @args: [database = "orders", table_name = "products", transformation_ctx =
"DataSource2"]
## @return: DataSource2
## @inputs: []
DataSource2 = glueContext.create_dynamic_frame.from_catalog(database = "orders",
table_name = "products", transformation_ctx = "DataSource2")
## @type: DataSource
## @args: [database = "orders", table_name = "orders", transformation_ctx =
"DataSource1"]
## @return: DataSource1
## @inputs: []
DataSource1 = glueContext.create_dynamic_frame.from_catalog(database = "orders",
table_name = "orders", transformation_ctx = "DataSource1")
## @type: Join
## @args: [keys2 = ["ordernumber"], keys1 = ["ordernumber"], transformation_ctx =
"Transform1"]
## @return: Transform1
## @inputs: [frame1 = DataSource1, frame2 = DataSource0]
Transform1 = Join.apply(frame1 = DataSource1, frame2 = DataSource0, keys2 =
["ordernumber"], keys1 = ["ordernumber"], transformation_ctx = "Transform1")
## @type: Join
```

```
## @args: [keys2 = ["productcode"], keys1 = ["productcode"], transformation_ctx =
"Transform3"]
## @return: Transform3
## @inputs: [frame1 = Transform1, frame2 = DataSource2]
Transform3 = Join.apply(frame1 = Transform1, frame2 = DataSource2, keys2 =
["productcode"], keys1 = ["productcode"], transformation_ctx = "Transform3")
## @type: SelectFields
## @args: [paths = ["ordernumber", "orderdate", "requireddate", "shippeddate",
"status", "msrp"], transformation_ctx = "Transform2"]
## @return: Transform2
## @inputs: [frame = Transform3]
Transform2 = SelectFields.apply(frame = Transform3, paths = ["ordernumber",
"orderdate", "requireddate", "shippeddate", "status", "msrp"], transformation_ctx =
"Transform2")
## @type: Filter
## @args: [f = lambda row : (row["msrp"] > 100), transformation_ctx =
"Transform0"]
## @return: Transform0
## @inputs: [frame = Transform2]
Transform0 = Filter.apply(frame = Transform2, f = lambda row : (row["msrp"] >
100), transformation_ctx = "Transform0")
## @type: DataSink
## @args: [connection_type = "s3", format = "parquet", connection_options =
{"path": "s3://query-results-bucket-603790572/", "partitionKeys": []},
transformation_ctx = "DataSink0"]
## @return: DataSink0
## @inputs: [frame = Transform0]
DataSink0 = glueContext.write_dynamic_frame.from_options(frame = Transform0,
connection_type = "s3", format = "parquet", connection_options = {"path":
"s3://query-results-bucket-603790572/", "partitionKeys": []}, transformation_ctx =
"DataSink0")
job.commit()
```

We are all set. Lets run the job using the **Run** button on top right. Click on **Run Details** should show you the status of the running job. Once the job status changes to **Succeeded** you can go to S3 to check the final results of the job.
At this point there should be many Parquet files produced in the results folder.

LAB END