CIS 602: Big Data – Homework 2

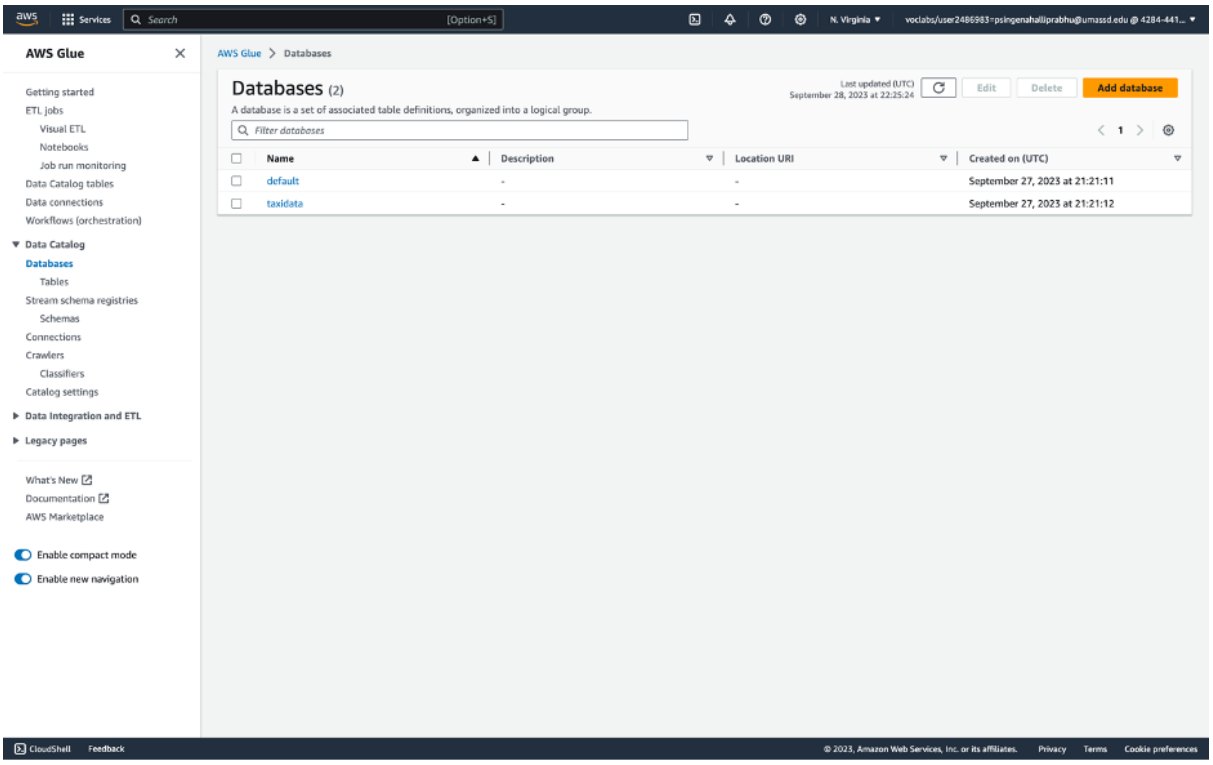
Pradyoth Singenahalli Prabhu - 02071847

Task 1: Creating and querying an AWS Glue database and table in Athena

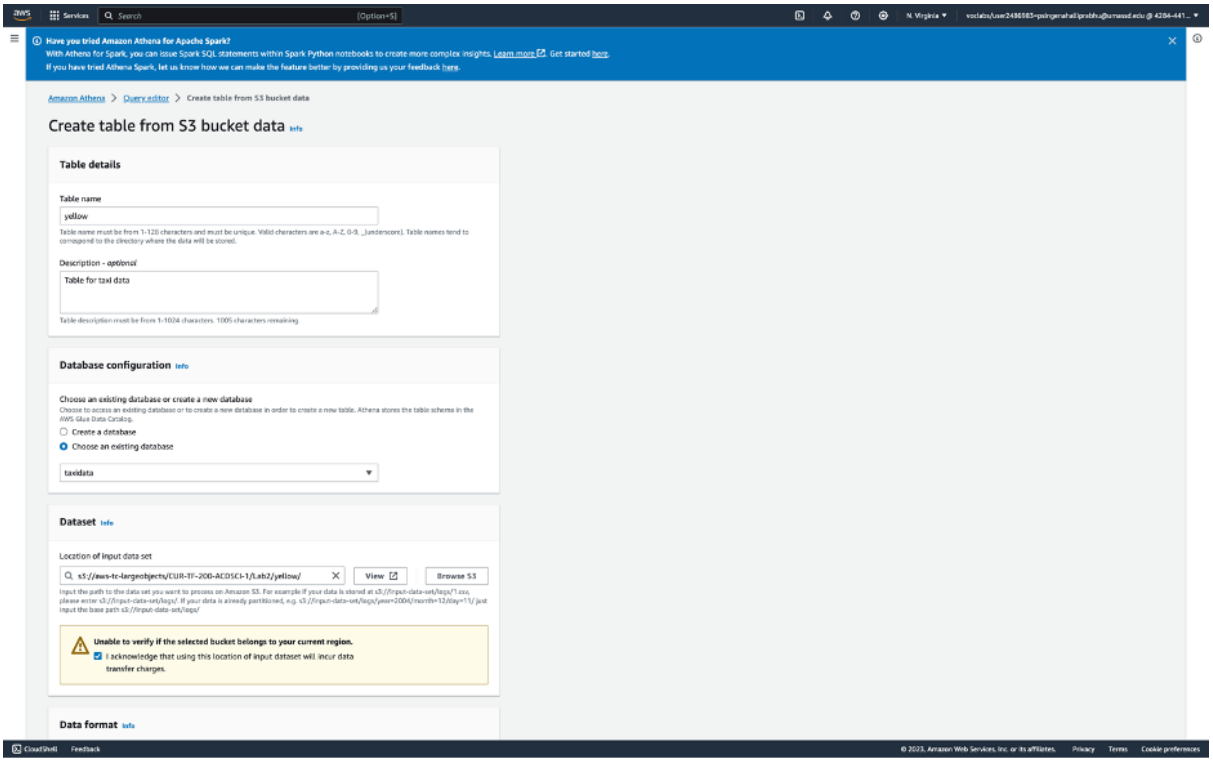
1) Creating database

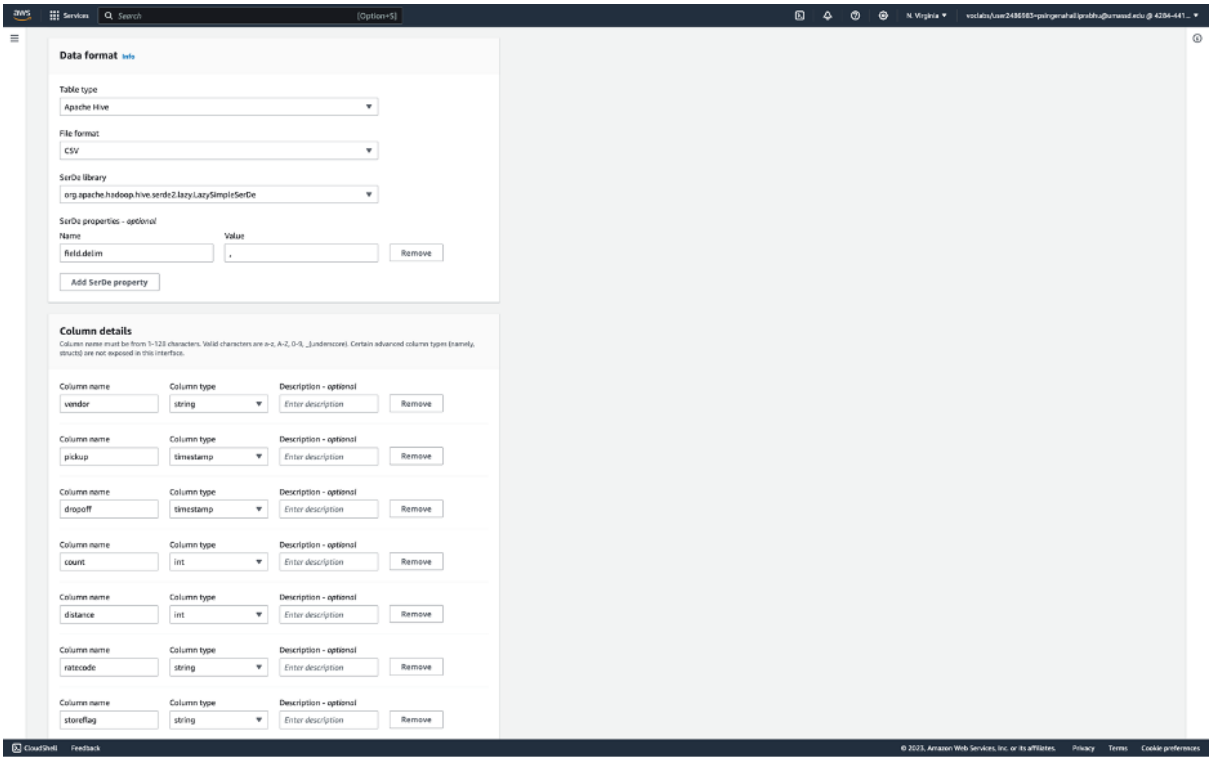
```
CREATE DATABASE taxidata;
```

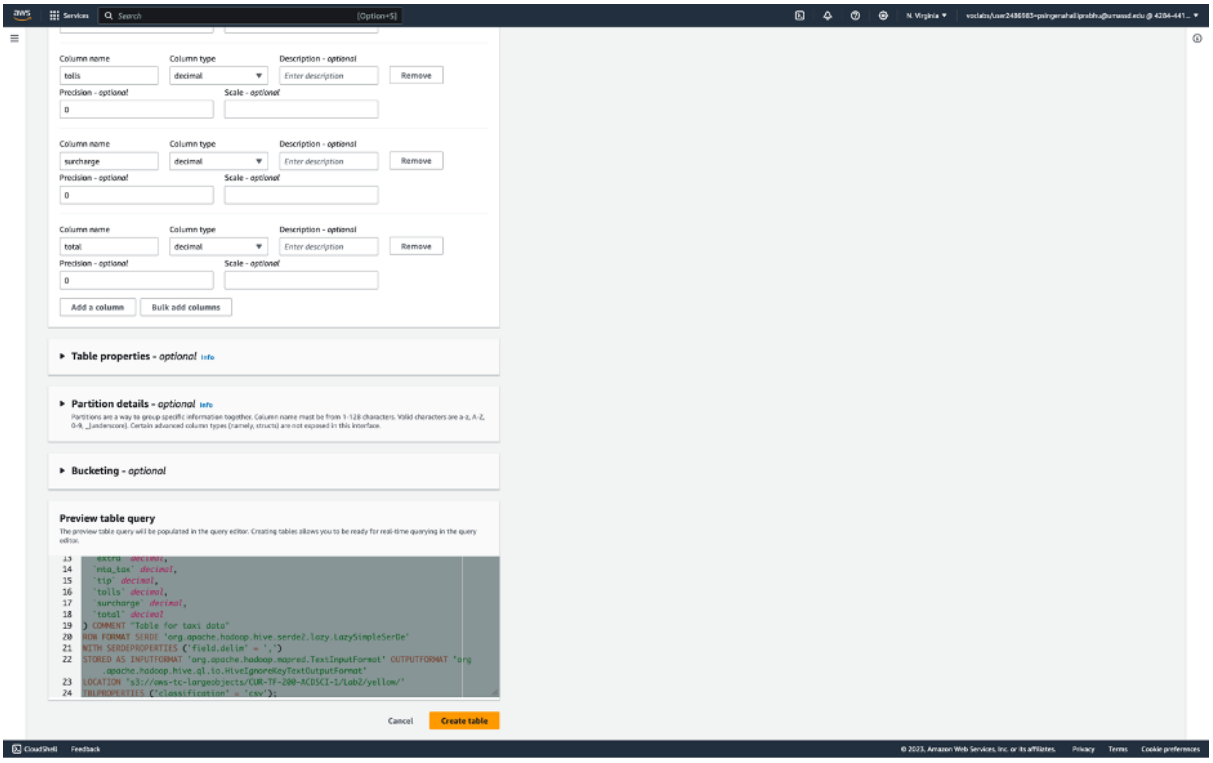
Successfully created database.



2) Creating table named `yellow`

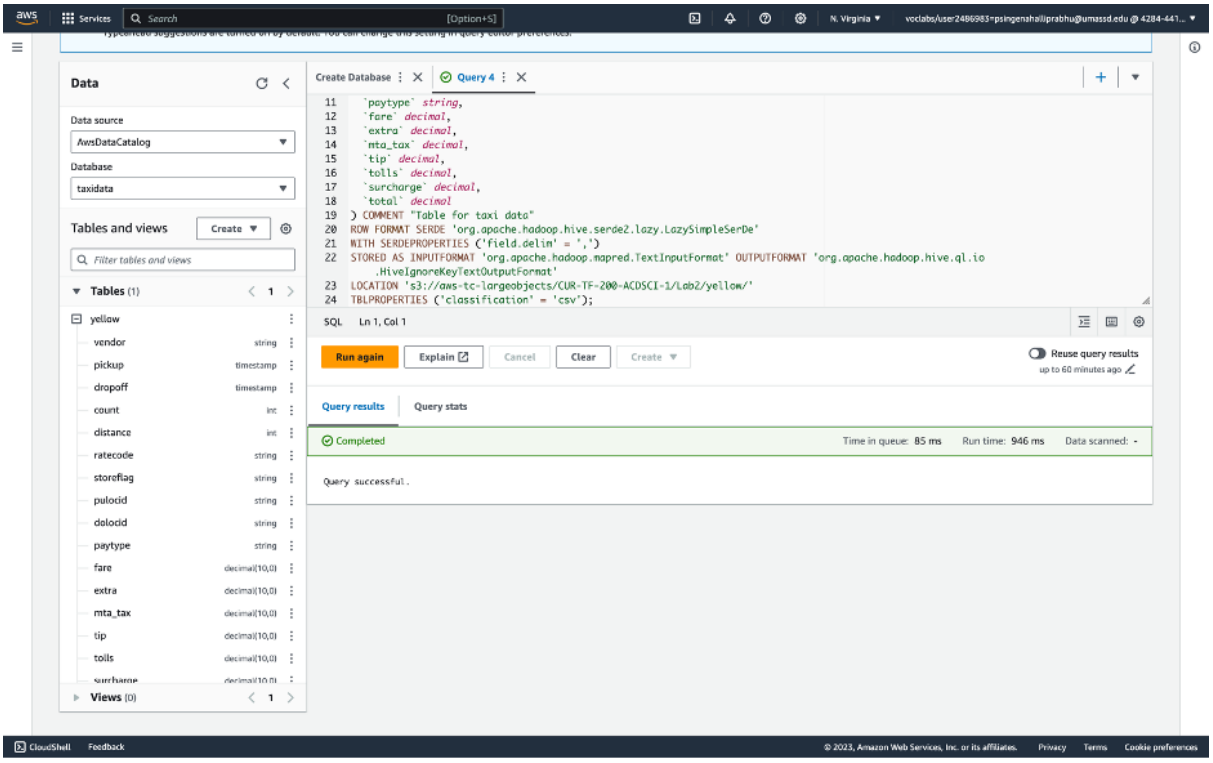

Have you tried Amazon Athena for Apache Spark?
With Athena for Spark, you can issue Spark SQL statements within Spark Python notebooks to create more complex insights. Learn more. Get started here.
If you have tried Athena Spark, let us know how we can make the feature better by providing us your feedback here.
Amazon Athena > Query editor > Create table from S3 bucket data
Create table from S3 bucket data Info
Table details
Table name
yellow
Table name must be from 1-128 characters and must be unique. Valid characters are a-z, A-Z, 0-9, _(underscore). Table names tend to correspond to the directory where the data will be stored.
Description - optional
Table for taxi data
Table description must be from 1-1024 characters. 1005 characters remaining
Database configuration Info
Choose an existing database or create a new database
Choose to access an existing database or to create a new database in order to create a new table. Athena stores the table schema in the AWS Glue Data Catalog.
Create a database
Choose an existing database
taxidata
Dataset Info
Location of input data set
s3://aws-tc-largeobjects/CUR-TF-200-ACDSCI-1/Lab2/yellow/
View
Browse S3
Input the path to the data set you want to process on Amazon S3. For example if your data is stored at s3://input-data-set/logs/1.csv, please enter s3://input-data-set/logs/. If your data is already partitioned, e.g. s3://input-data-set/logs/year=2004/month=12/day=11/ just input the base path s3://input-data-set/logs/
Unable to verify if the selected bucket belongs to your current region.
I acknowledge that using this location of input dataset will incur data transfer charges.
Data format Info
CloudShell
Feedback
© 2023, Amazon Web Services, Inc. or its affiliates.
Privacy
Terms
Cookie preferences


Data format Info
Table type
Apache Hive
File format
CSV
SerDe library
org.apache.hadoop.hive.serde2.lazy.LazySimpleSerDe
SerDe properties - optional
Name
Value
field.delim
,
Remove
Add SerDe property
Column details
Column name must be from 1-128 characters. Valid characters are a-z, A-Z, 0-9, _(underscore). Certain advanced column types (namely, structs) are not exposed in this interface.
Column name
Column type
Description - optional
vendor
string
pickup
timestamp
dropoff
timestamp
count
int
distance
int
ratecode
string
storeflag
string
Enter description
Remove
CloudShell
Feedback
© 2023, Amazon Web Services, Inc. or its affiliates.
Privacy
Terms
Cookie preferences



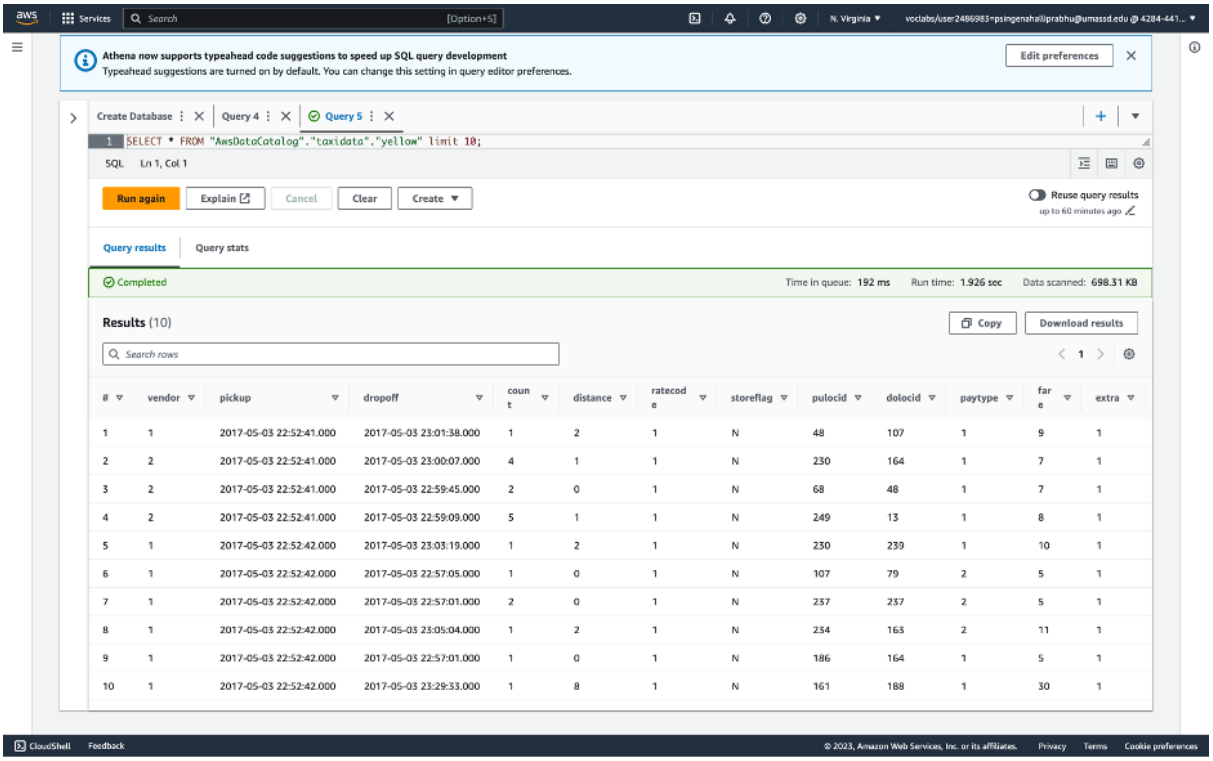
Creating a table named `yellow` .

```
CREATE EXTERNAL TABLE IF NOT EXISTS taxidata.yellow (
    `vendor` string,
    `pickup` timestamp,
    `dropoff` timestamp,
    `count` int,
    `distance` int,
    `ratecode` string,
    `storeflag` string,
    `pulocid` string,
    `dolocid` string,
    `paytype` string,
    `fare` decimal,
    `extra` decimal,
    `mta_tax` decimal,
    `tip` decimal,
    `tolls` decimal,
    `surcharge` decimal,
    `total` decimal
  )
  ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.lazy.LazySimpleSerDe'
  WITH SERDEPROPERTIES (
    'serialization.format' = ',',
    'field.delim' = ','
  ) LOCATION 's3://aws-tc-largeobjects/CUR-TF-200-ACDSCI-1/Lab2/yellow/'
  TBLPROPERTIES ('has_encrypted_data'='false');
```



Querying to preview data:

```
SELECT * FROM "AwsDataCatalog"."taxidata"."yellow" limit 10;
```



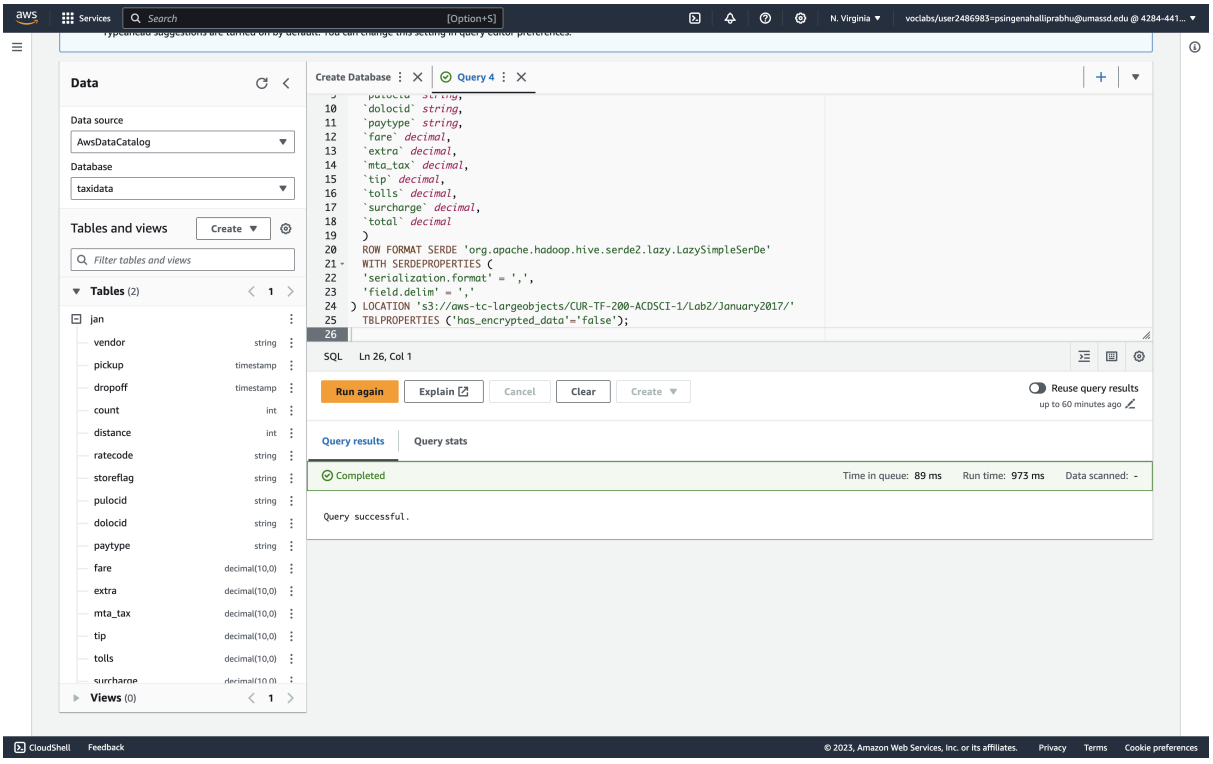
#	vendor	pickup	dropoff	count	distance	ratecode	storeflag	pulocid	dolocid	paytype	fare	extra
1	1	2017-05-03 22:52:41.000	2017-05-03 23:01:38.000	1	2	1	N	48	107	1	9	1
2	2	2017-05-03 22:52:41.000	2017-05-03 23:00:07.000	4	1	1	N	230	164	1	7	1
3	2	2017-05-03 22:52:41.000	2017-05-03 22:59:45.000	2	0	1	N	68	48	1	7	1
4	2	2017-05-03 22:52:41.000	2017-05-03 22:59:09.000	5	1	1	N	249	13	1	8	1
5	1	2017-05-03 22:52:42.000	2017-05-03 23:03:19.000	1	2	1	N	230	239	1	10	1
6	1	2017-05-03 22:52:42.000	2017-05-03 22:57:05.000	1	0	1	N	107	79	2	5	1
7	1	2017-05-03 22:52:42.000	2017-05-03 22:57:01.000	2	0	1	N	237	237	2	5	1
8	1	2017-05-03 22:52:42.000	2017-05-03 23:05:04.000	1	2	1	N	234	163	2	11	1
9	1	2017-05-03 22:52:42.000	2017-05-03 22:57:01.000	1	0	1	N	186	164	1	5	1
10	1	2017-05-03 22:52:42.000	2017-05-03 23:29:33.000	1	8	1	N	161	188	1	30	1

Task 2: Optimizing Athena queries by using buckets

Creating a table named `jan` for the January 2017 data.

```
CREATE EXTERNAL TABLE IF NOT EXISTS jan (
  `vendor` string,
  `pickup` timestamp,
  `dropoff` timestamp,
  `count` int,
  `distance` int,
  `ratecode` string,
  `storeflag` string,
  `pulocid` string,
  `dolocid` string,
  `paytype` string,
  `fare` decimal,
  `extra` decimal,
  `mta_tax` decimal,
  `tip` decimal,
  `tolls` decimal,
  `surcharge` decimal,
  `total` decimal
  )
  ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.lazy.LazySimpleSerDe'
  WITH SERDEPROPERTIES (
  'serialization.format' = ',',
  'field.delim' = ','
) LOCATION 's3://aws-tc-largeobjects/CUR-TF-200-ACDSCI-1/Lab2/January2017/'
  TBLPROPERTIES ('has_encrypted_data'='false');
```

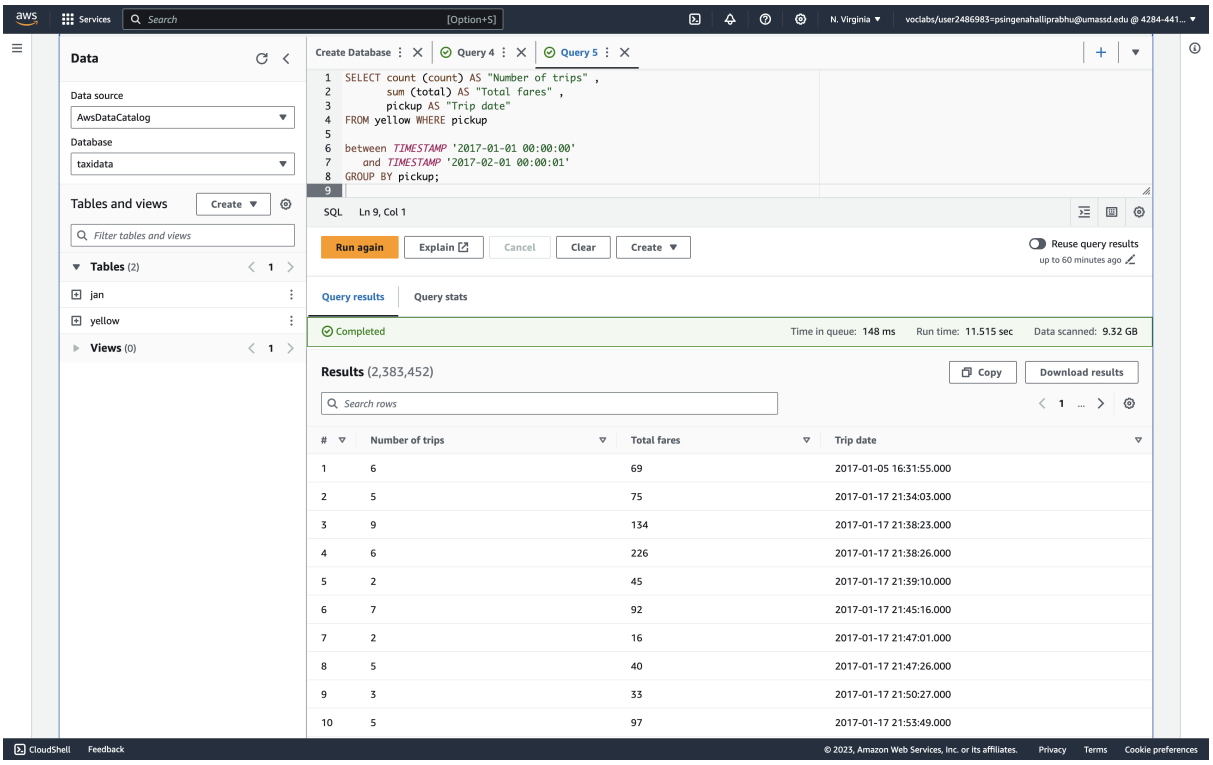
The message *Query successful* displays. Athena created a new table named *jan*, which contains the taxi data for only the month of January 2017. The table is listed in the **Tables** section on the left.



2) Querying data:

Running the following query on the *yellow* table, which has data for the entire year. The data is not divided into monthly buckets.

```
SELECT count (count) AS "Number of trips" ,
       sum (total) AS "Total fares" ,
       pickup AS "Trip date"
FROM yellow WHERE pickup

between TIMESTAMP '2017-01-01 00:00:00'
   and TIMESTAMP '2017-02-01 00:00:01'
GROUP BY pickup;
```



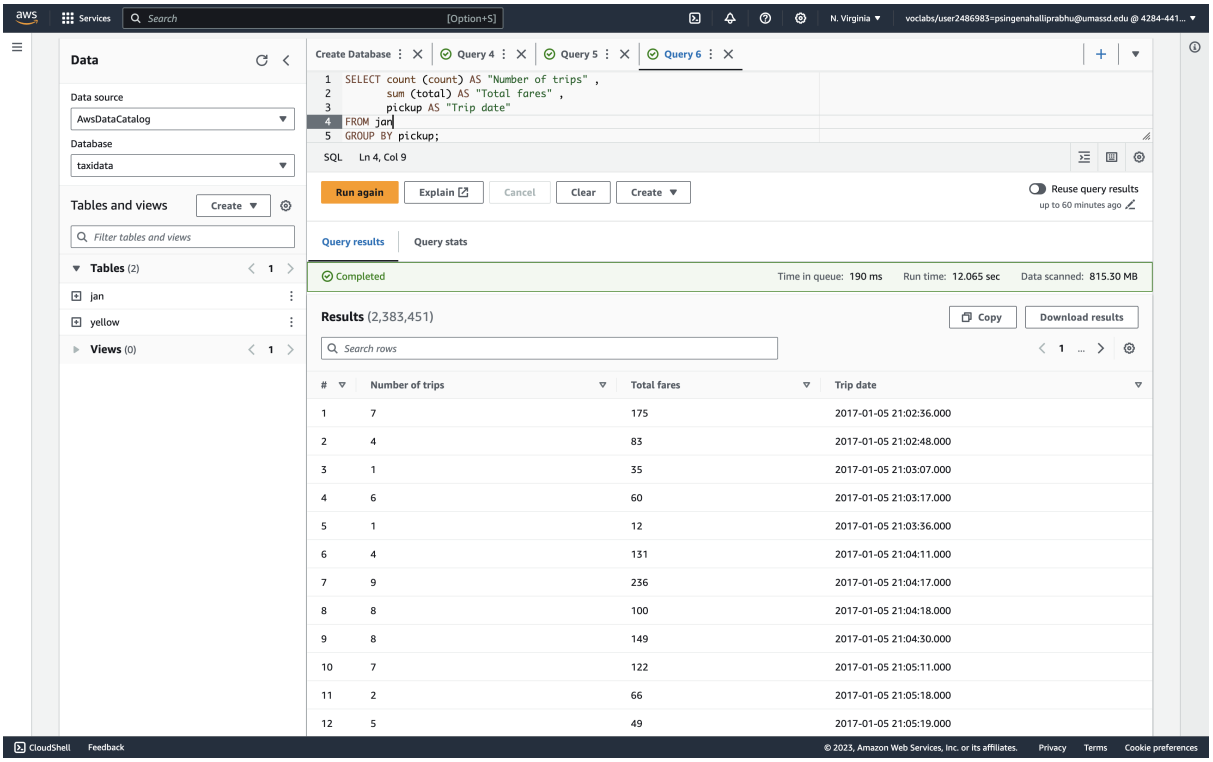
Time in queue: 148 ms

Run time: 11.515 sec

Data scanned: 9.32 GB

Running the following query on the `jan` table:

```
SELECT count (count) AS "Number of trips" ,
       sum (total) AS "Total fares" ,
       pickup AS "Trip date"
FROM jan
GROUP BY pickup;
```



Time in queue: 190 ms

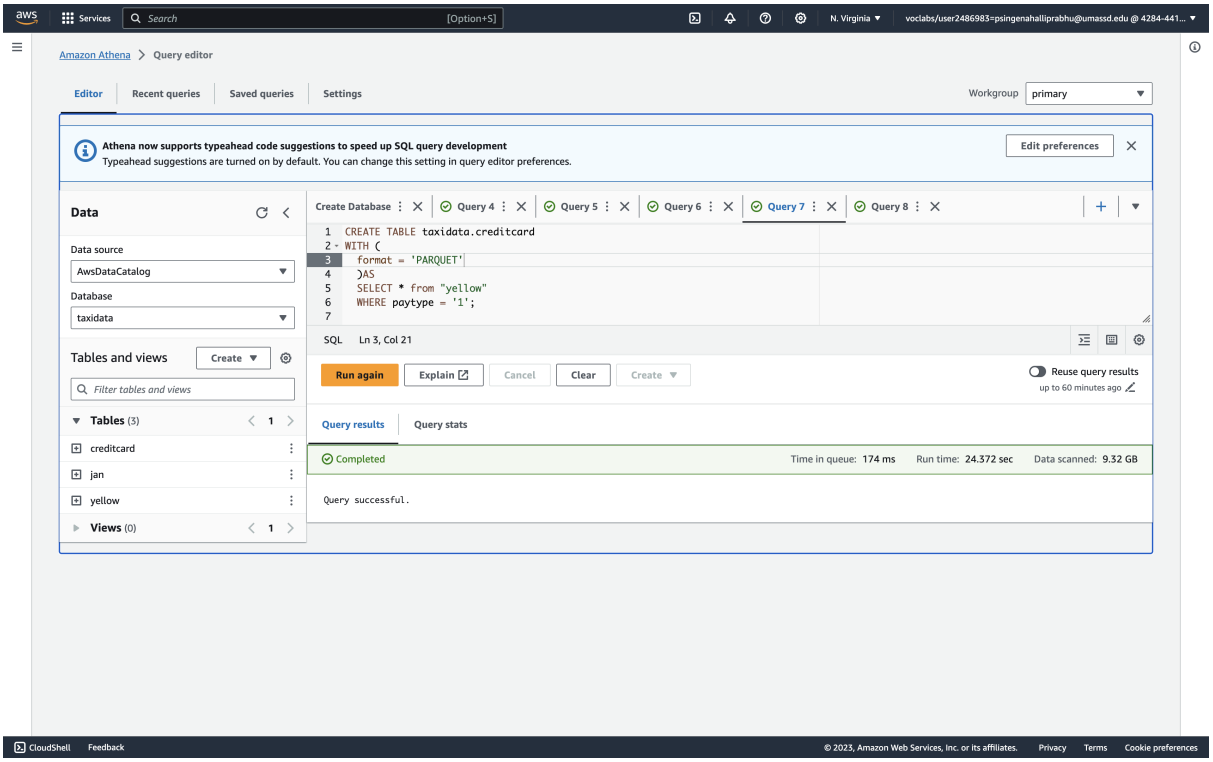
Run time: 12.065 sec

Data scanned: 815.30 MB

Task 3: Optimizing Athena queries by using partitions

Creating a new table called `taxidata.creditcard` that is partitioned for paytype = 1

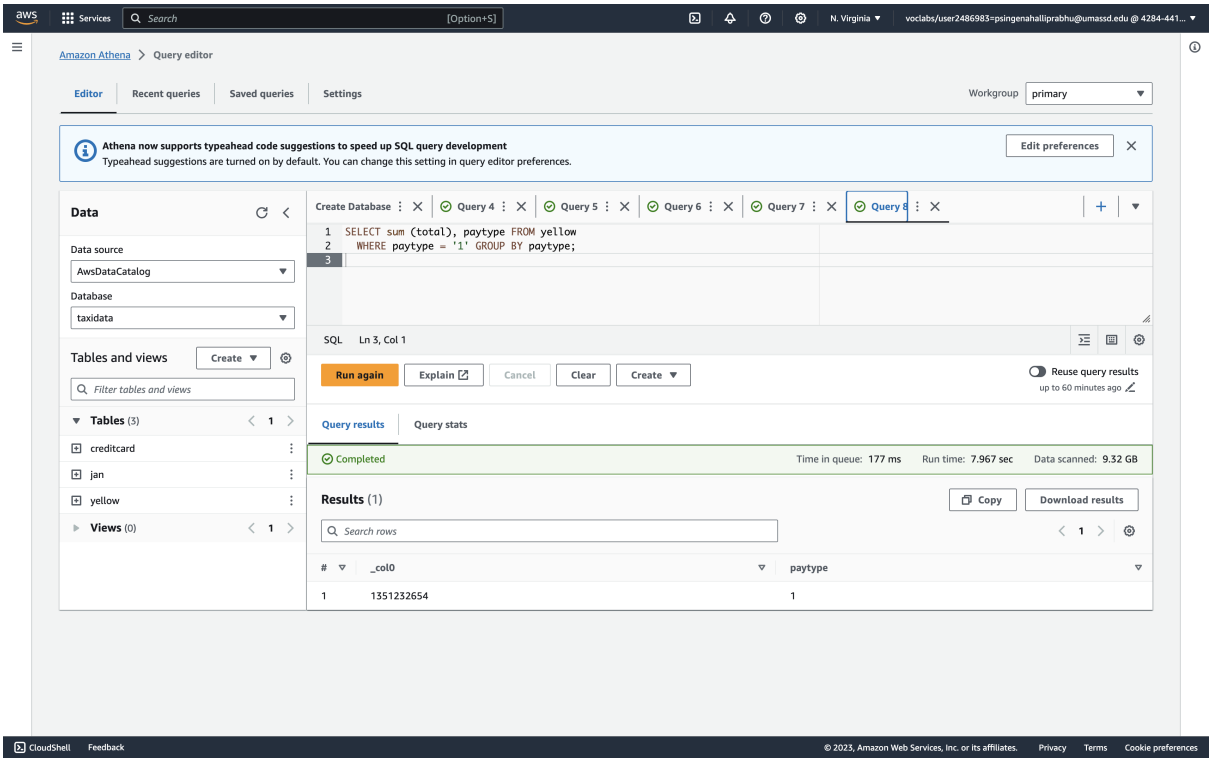
```
CREATE TABLE taxidata.creditcard
WITH (
  format = 'PARQUET'
  )AS
  SELECT * from "yellow"
  WHERE paytype = '1';
```



Time in queue: 174 ms
Run time: 24.372 sec
Data scanned: 9.32 GB

Querying the non-partitioned data in the `yellow` table:

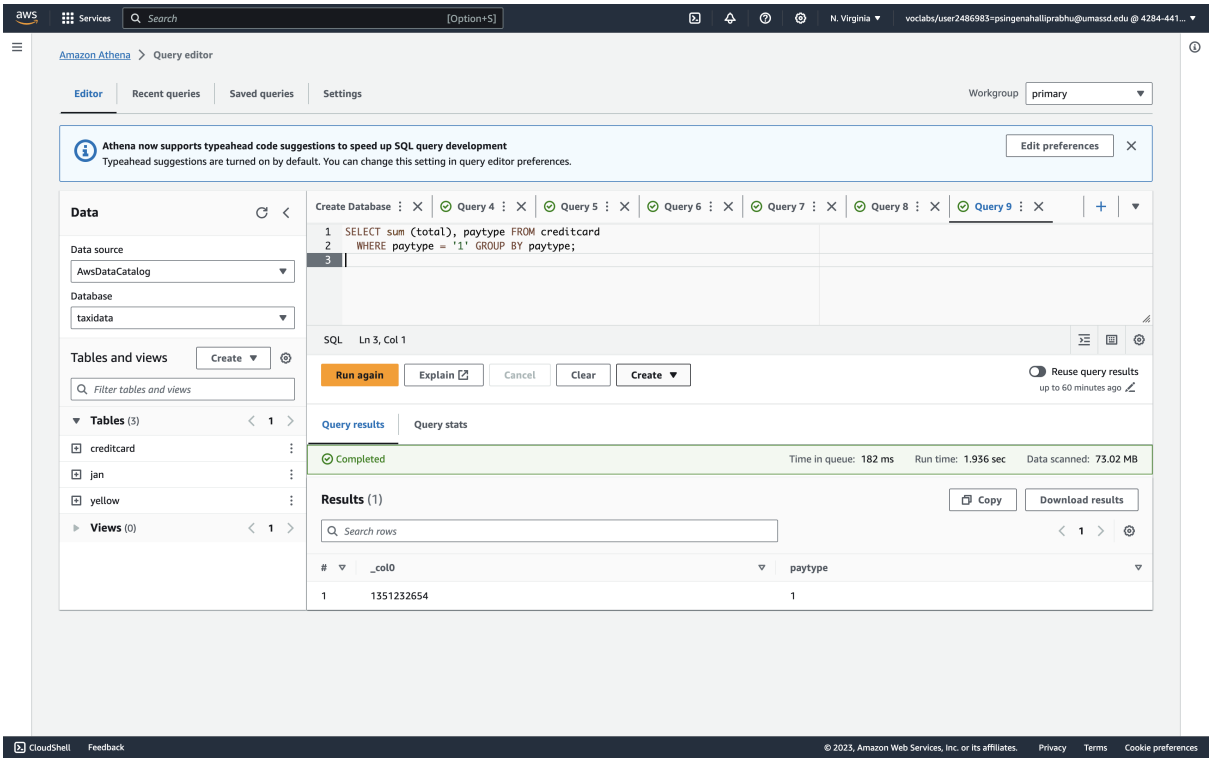
```
SELECT sum (total), paytype FROM yellow
WHERE paytype = '1' GROUP BY paytype;
```



Time in queue: 177 ms
Run time: 7.967 sec
Data scanned: 9.32 GB

Querying the partitioned data in the `creditcard` table:

```
SELECT sum (total), paytype FROM creditcard
WHERE paytype = '1' GROUP BY paytype;
```

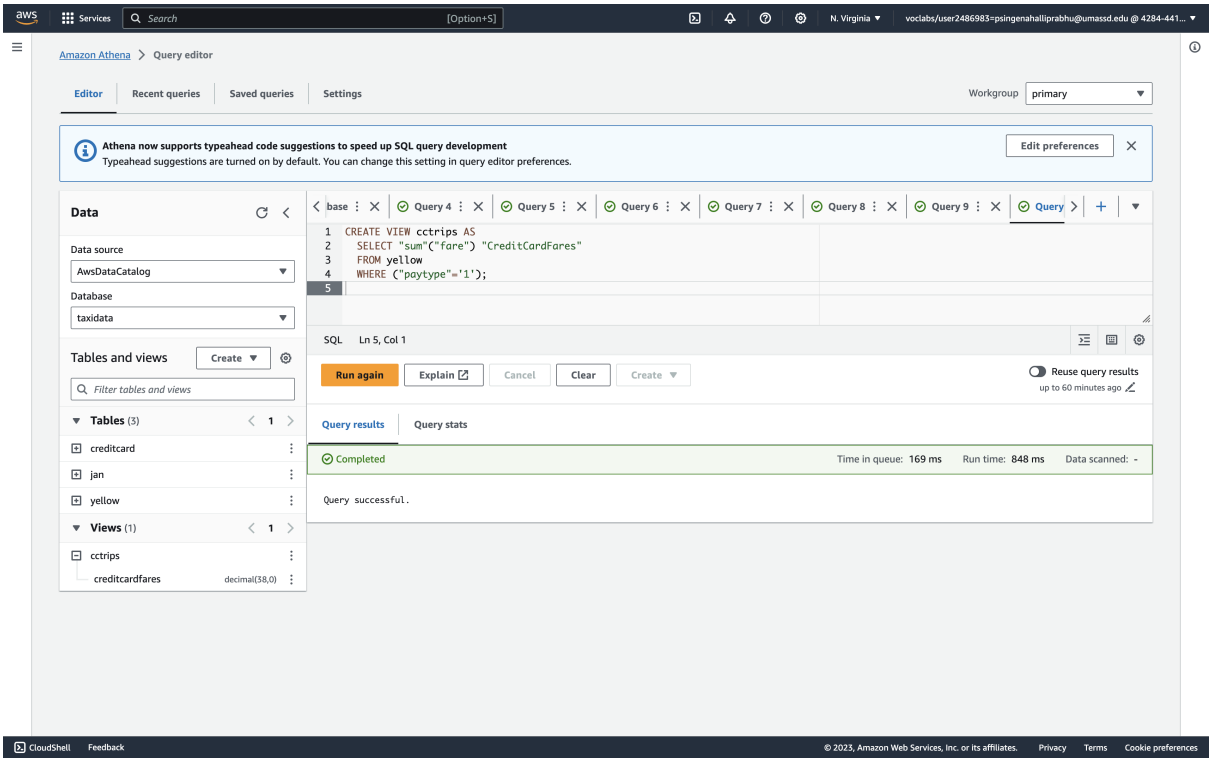


Time in queue: 182 ms
Run time: 1.936 sec
Data scanned: 73.02 MB

Task 4: Using Athena views

Creating a view called `cctrips` for the total dollar value of fares that were paid with a credit card:

```
CREATE VIEW cctrips AS
  SELECT "sum"("fare") "CreditCardFares"
  FROM yellow
  WHERE ("paytype"='1');
```

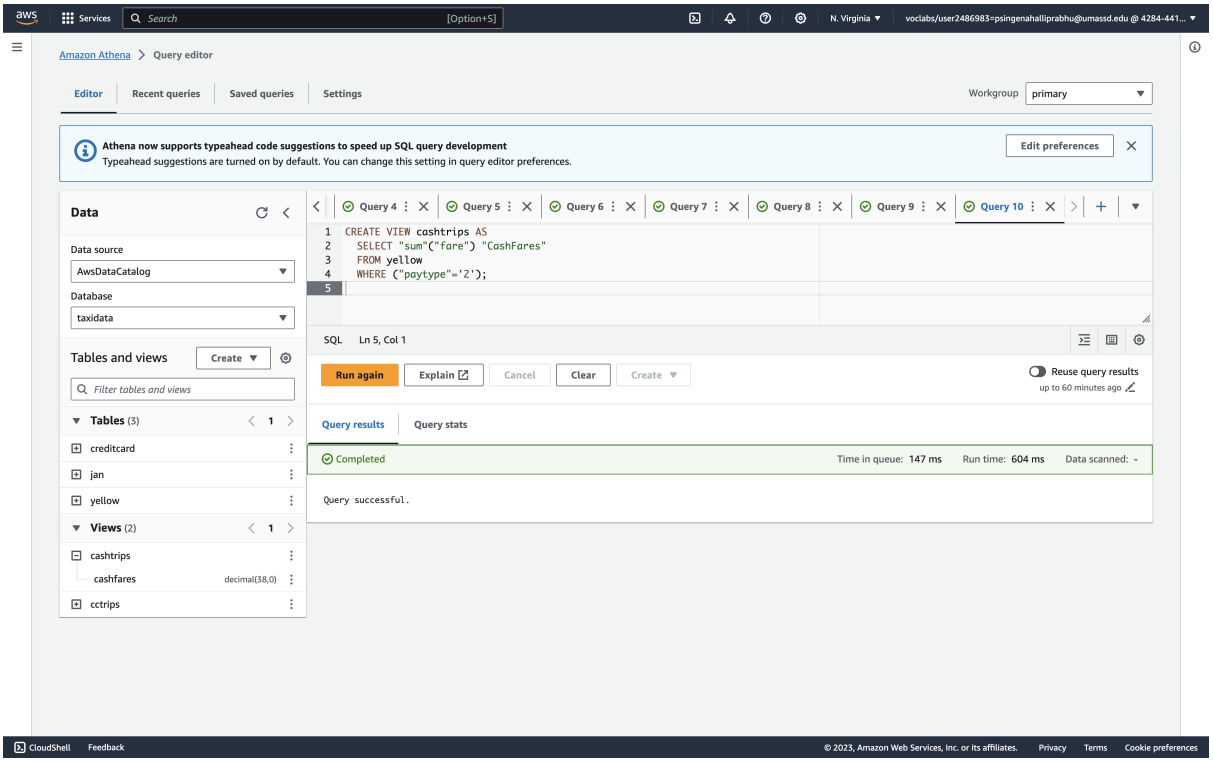


Time in queue: 169 ms

Run time: 848 sec

Create a view called `cashtrips` for the total dollar value of fares that were paid with cash:

```
CREATE VIEW cashtrips AS
  SELECT "sum"("fare") "CashFares"
  FROM yellow
  WHERE ("paytype"='2');
```

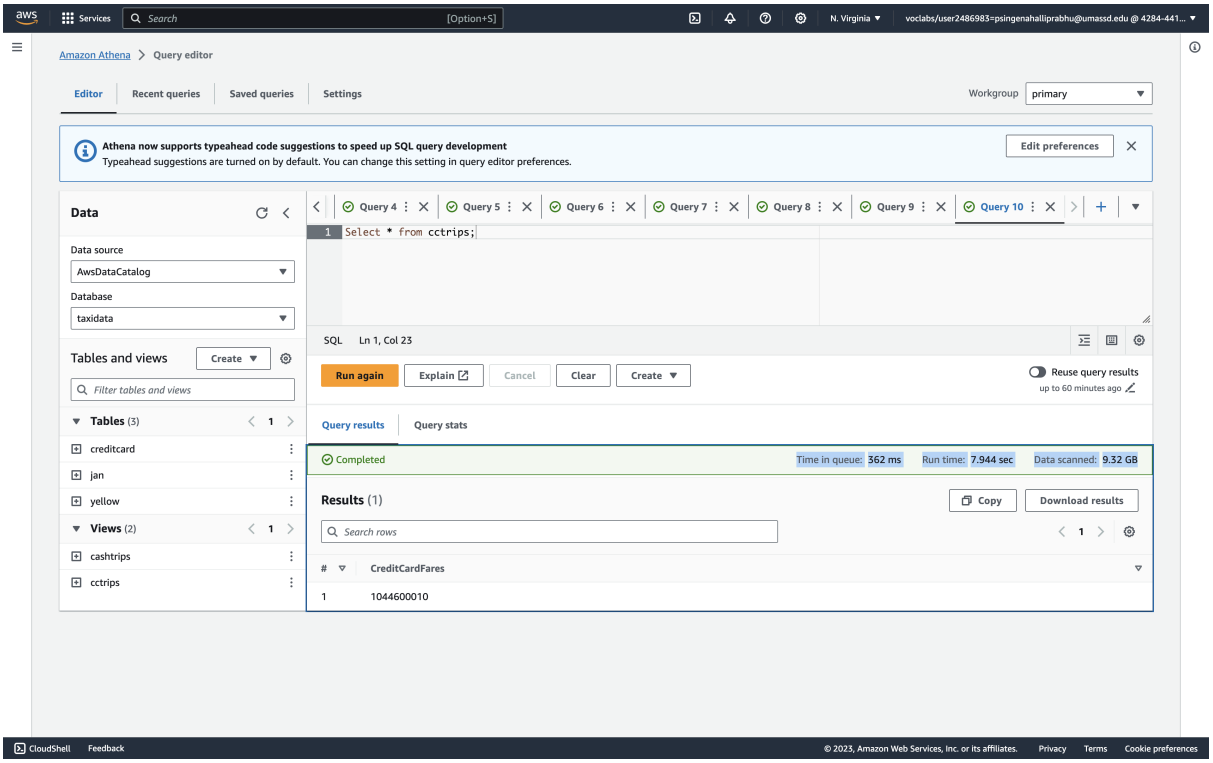


Time in queue: 147 ms

Run time: 604 sec

Select all the records from the view `cctrips`:

```
Select * from cctrips;
```



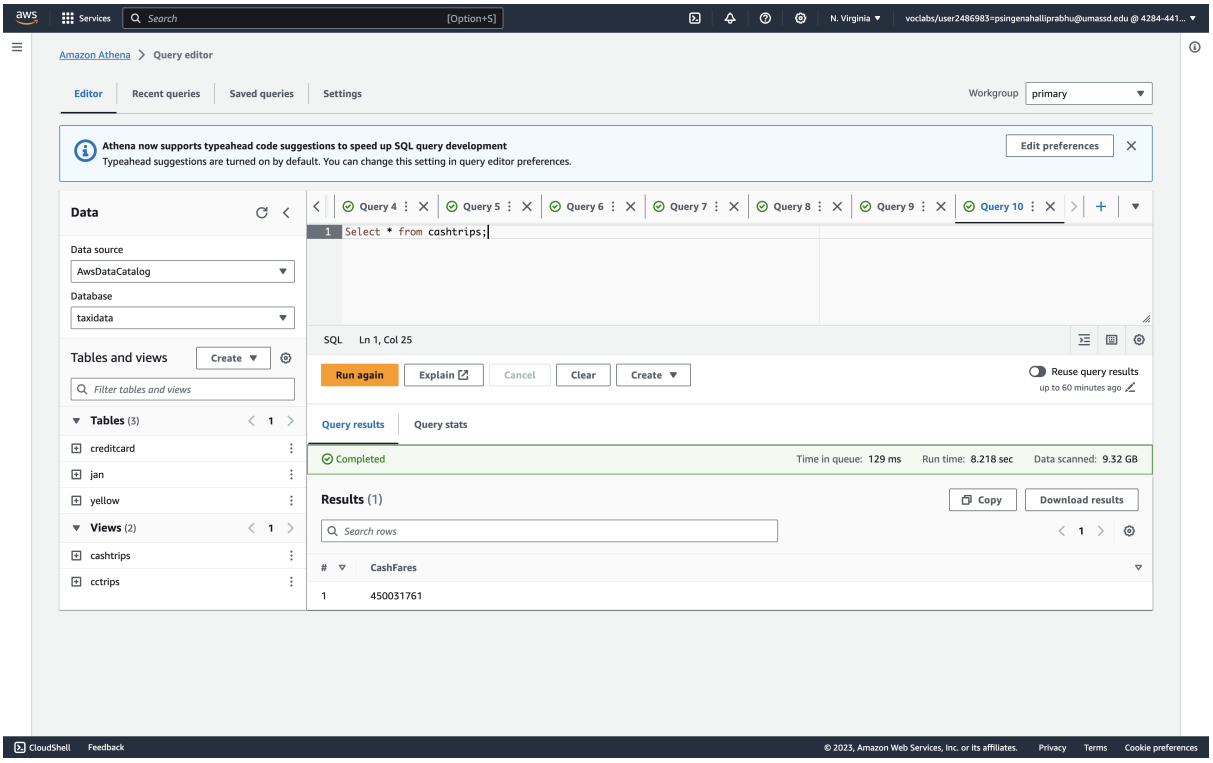
Time in queue: 362 ms

Run time: 7.944 sec

Data scanned: 9.32 GB

Select list all the records from the view `cashtrips`:

```
Select * from cashtrips;
```

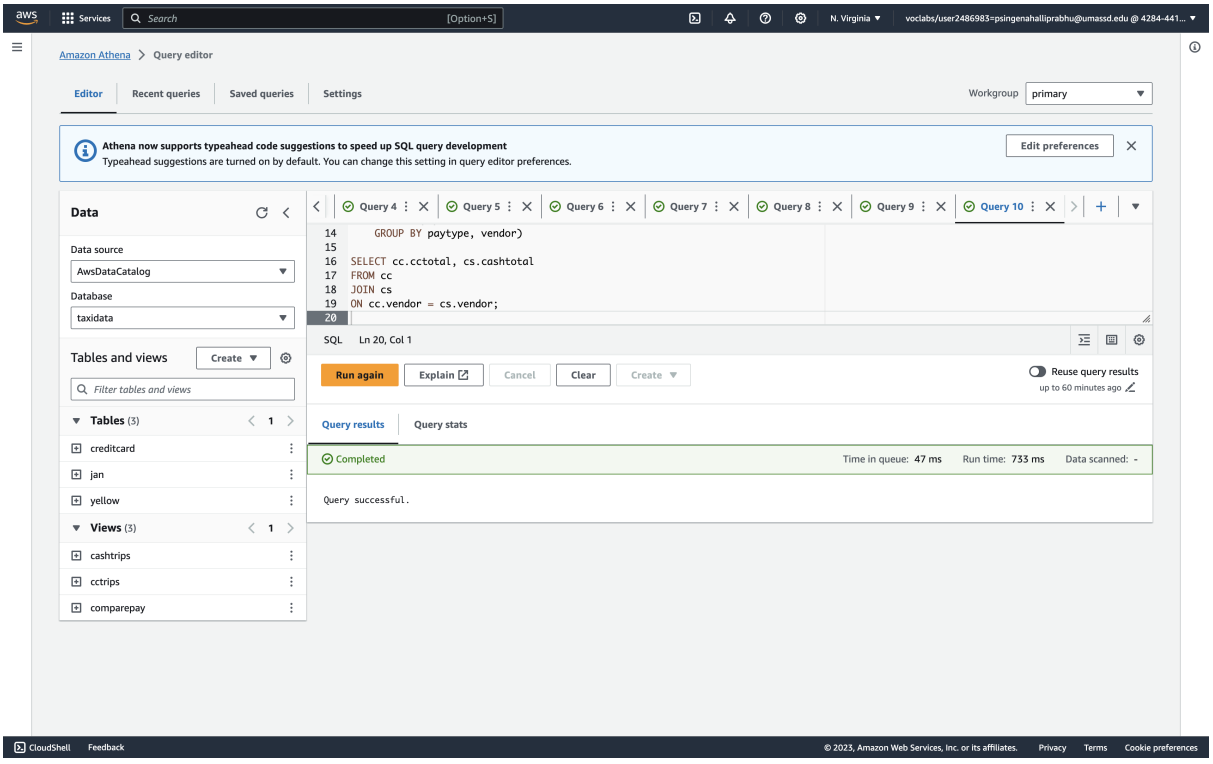


Time in queue: 129 ms
Run time: 8.218 sec
Data scanned: 9.32 GB

Create a new view called `cashtrips` that joins the data:

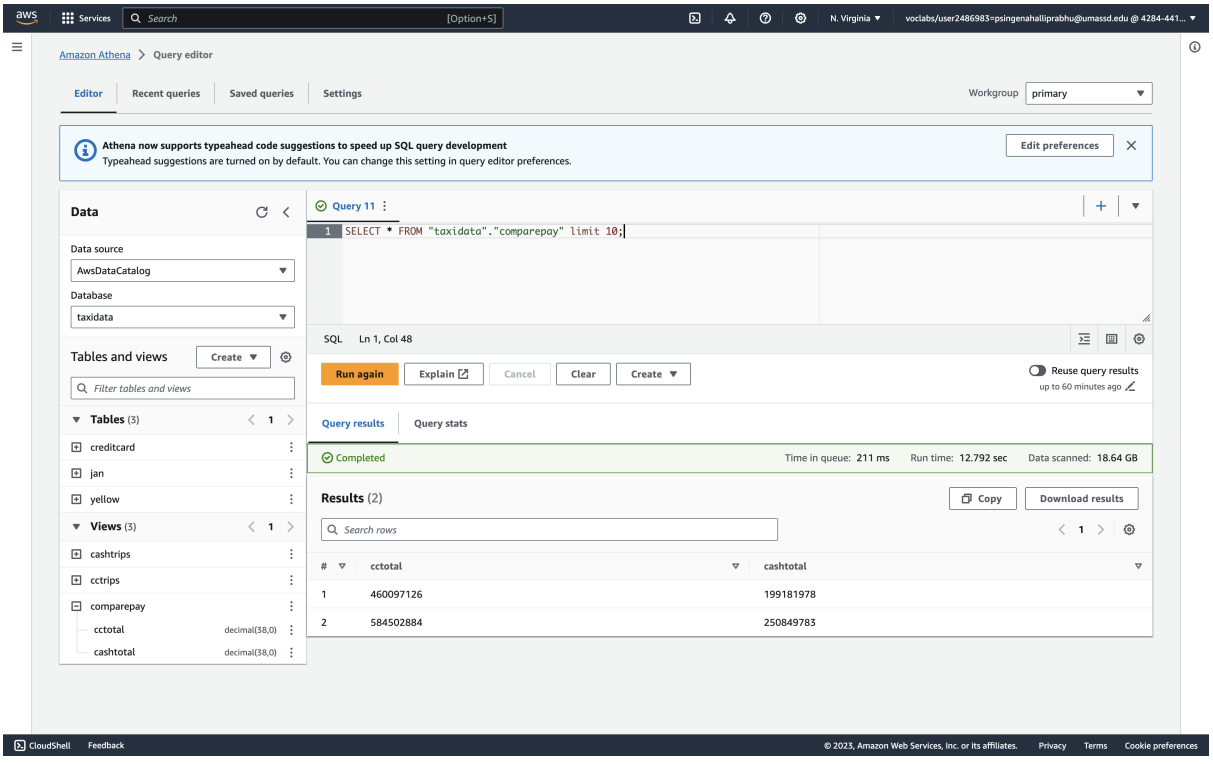
```
CREATE VIEW cashtrips AS
WITH
  cc AS
    (SELECT sum(fare) AS cctotal,
        vendor
    FROM yellow
    WHERE paytype = '1'
    GROUP BY paytype, vendor),
    cs AS
    (SELECT sum(fare) AS cashtotal,
            vendor, paytype
    FROM yellow
    WHERE paytype = '2'
    GROUP BY paytype, vendor)

SELECT cc.cctotal, cs.cashtotal
FROM cc
```

```
JOIN cs
ON cc.vendor = cs.vendor;
```



Time in queue: 47 ms
Run time: 733 ms

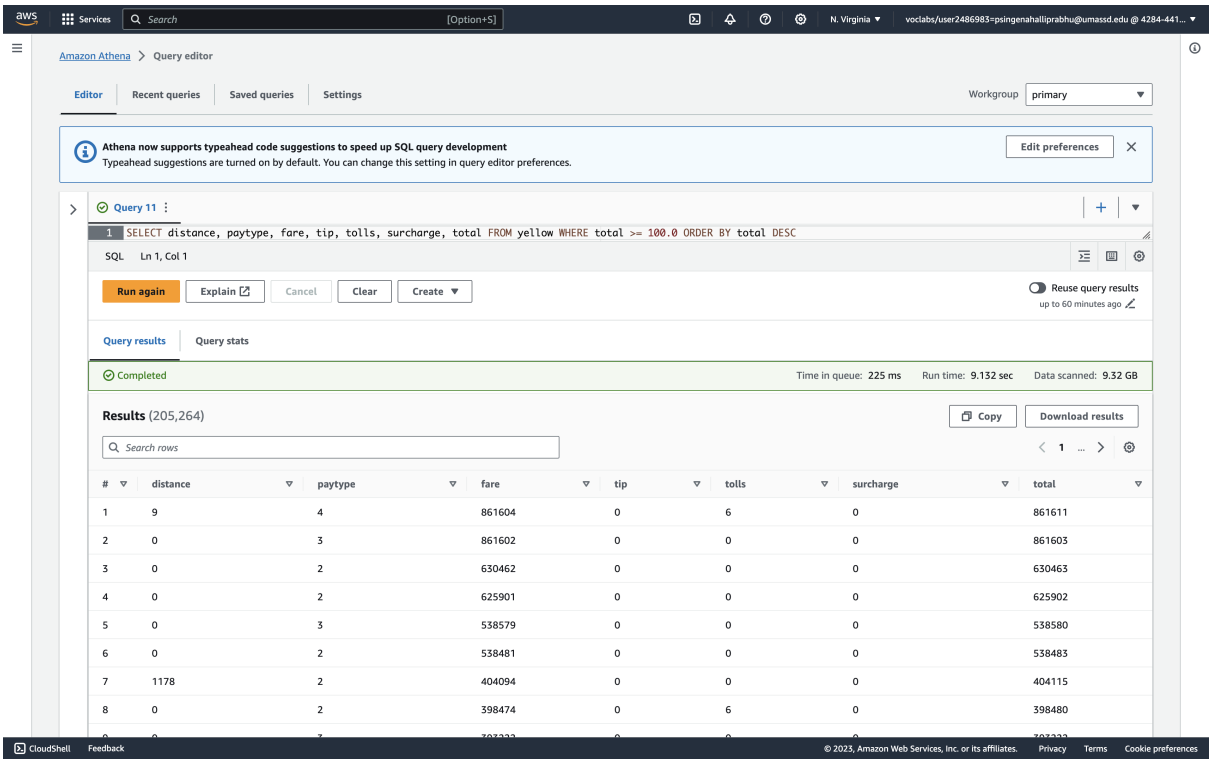
Preview the result from the join:



Task 5: Creating Athena named queries by using CloudFormation

Query:

```
SELECT distance, paytype, fare, tip, tolls, surcharge, total FROM yellow WHERE total >= 100.0 ORDER BY total DESC
```

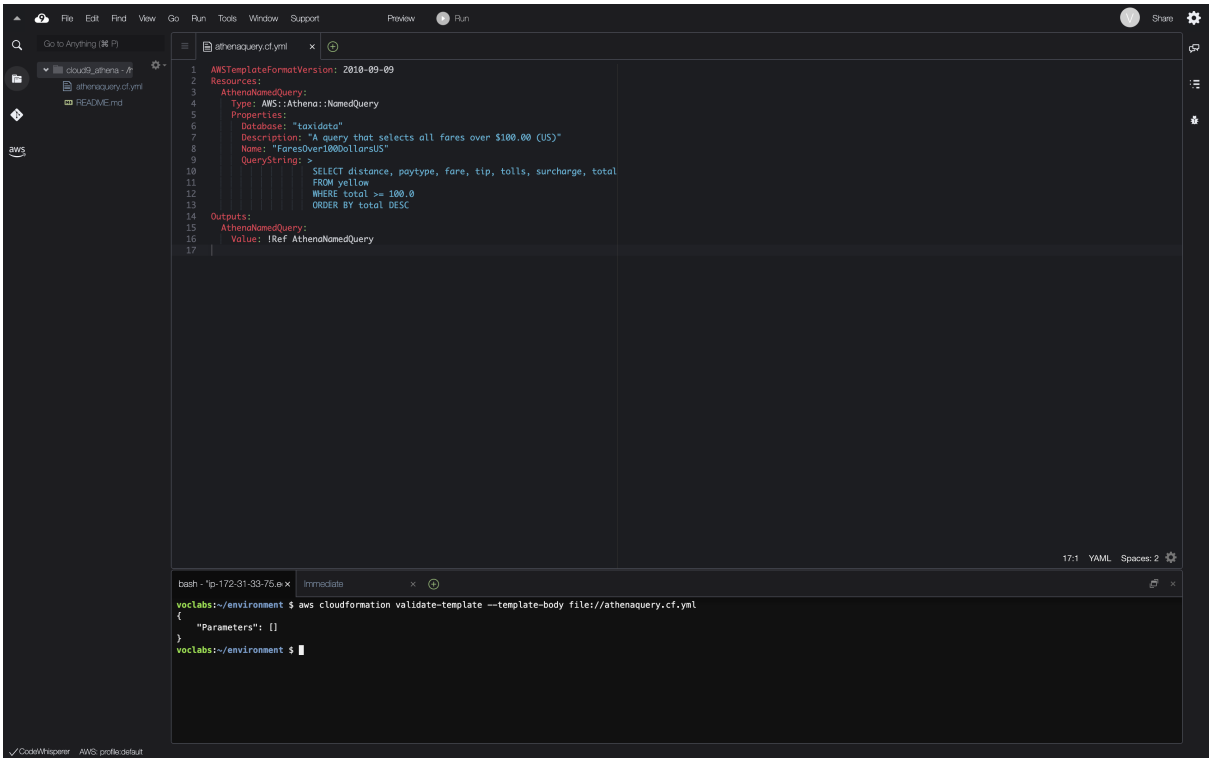


#	distance	paytype	fare	tip	tolls	surcharge	total
1	9	4	861604	0	6	0	861611
2	0	3	861602	0	0	0	861603
3	0	2	630462	0	0	0	630463
4	0	2	625901	0	0	0	625902
5	0	3	538579	0	0	0	538580
6	0	2	538481	0	0	0	538483
7	1178	2	404094	0	0	0	404115
8	0	2	398474	0	6	0	398480

Create a new CloudFormation template:

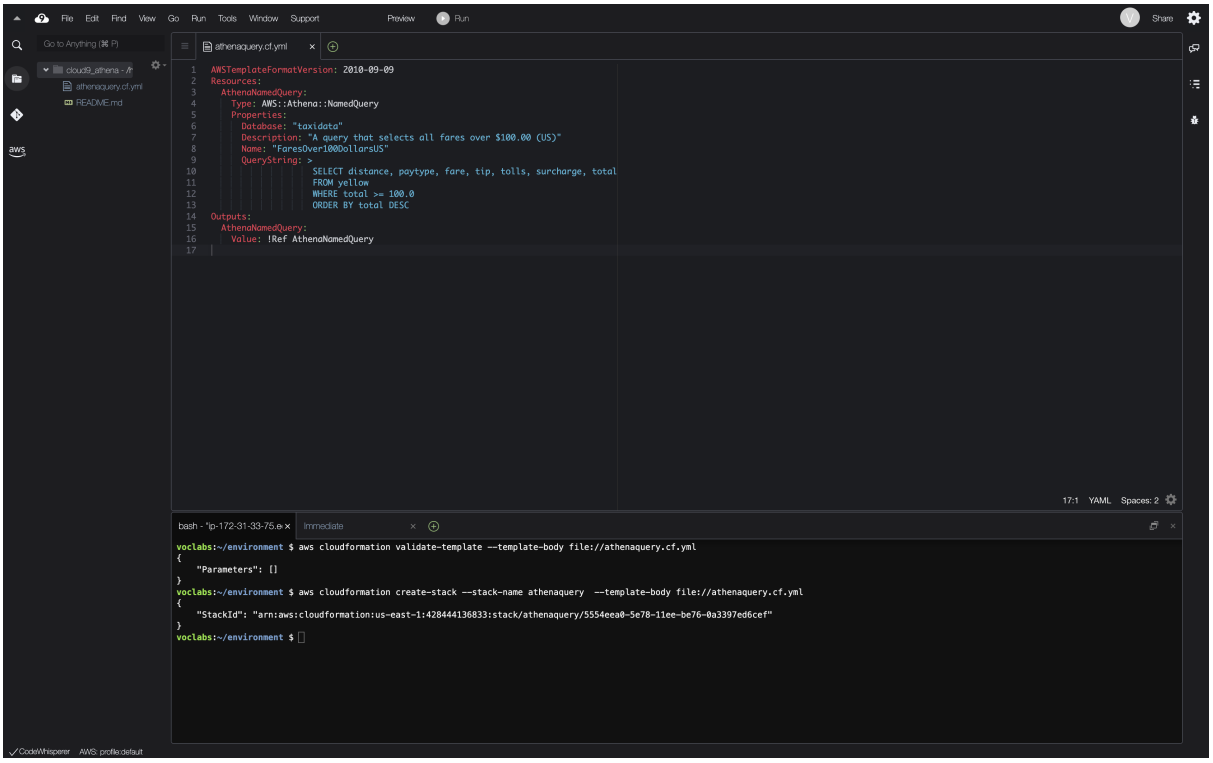
```
AWSTemplateFormatVersion: 2010-09-09
Resources:
  AthenaNamedQuery:
    Type: AWS::Athena::NamedQuery
    Properties:
      Database: "taxidata"
      Description: "A query that selects all fares over $100.00 (US)"
      Name: "FaresOver100DollarsUS"
      QueryString: >
                   SELECT distance, paytype, fare, tip, tolls, surcharge, total
                   FROM yellow
                   WHERE total >= 100.0
                   ORDER BY total DESC
Outputs:
  AthenaNamedQuery:
    Value: !Ref AthenaNamedQuery
```

```
aws cloudformation validate-template --template-body file://athenaquery.cf.yml
```



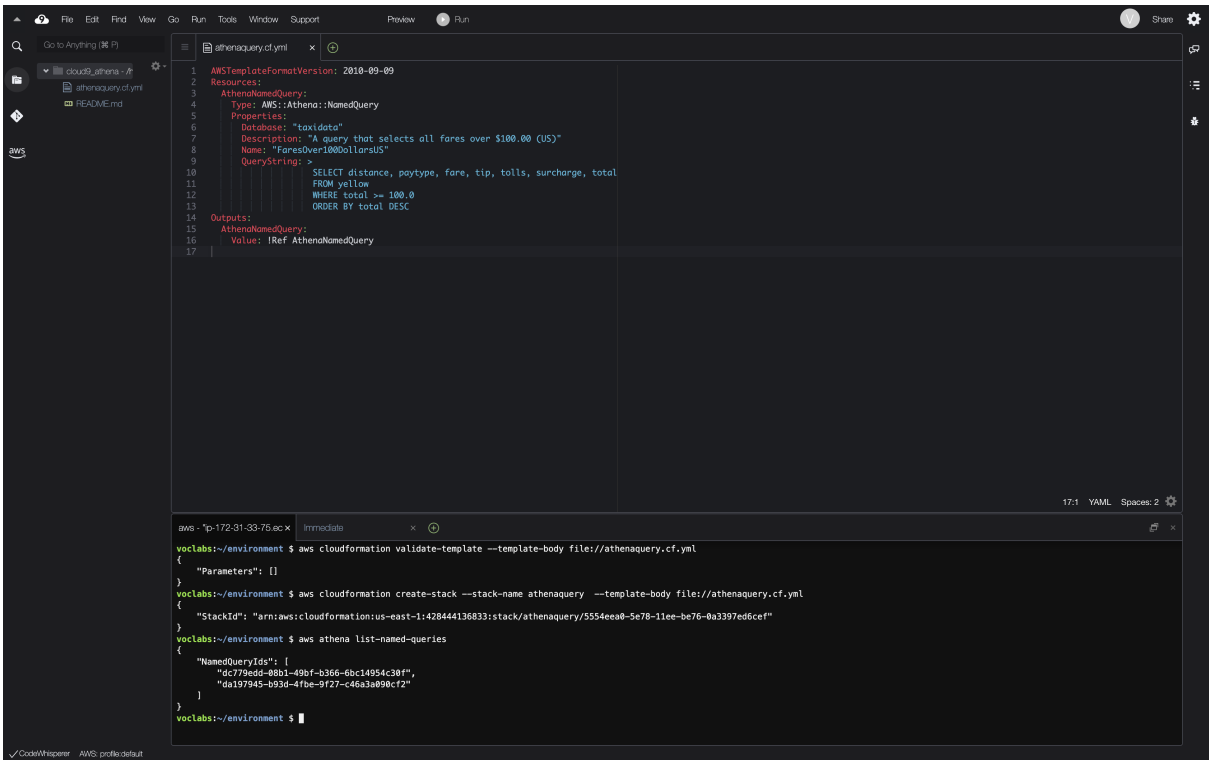
Creating the stack:

```
aws cloudformation create-stack --stack-name athenaquery  --template-body file://athenaquery.cf.yml
```



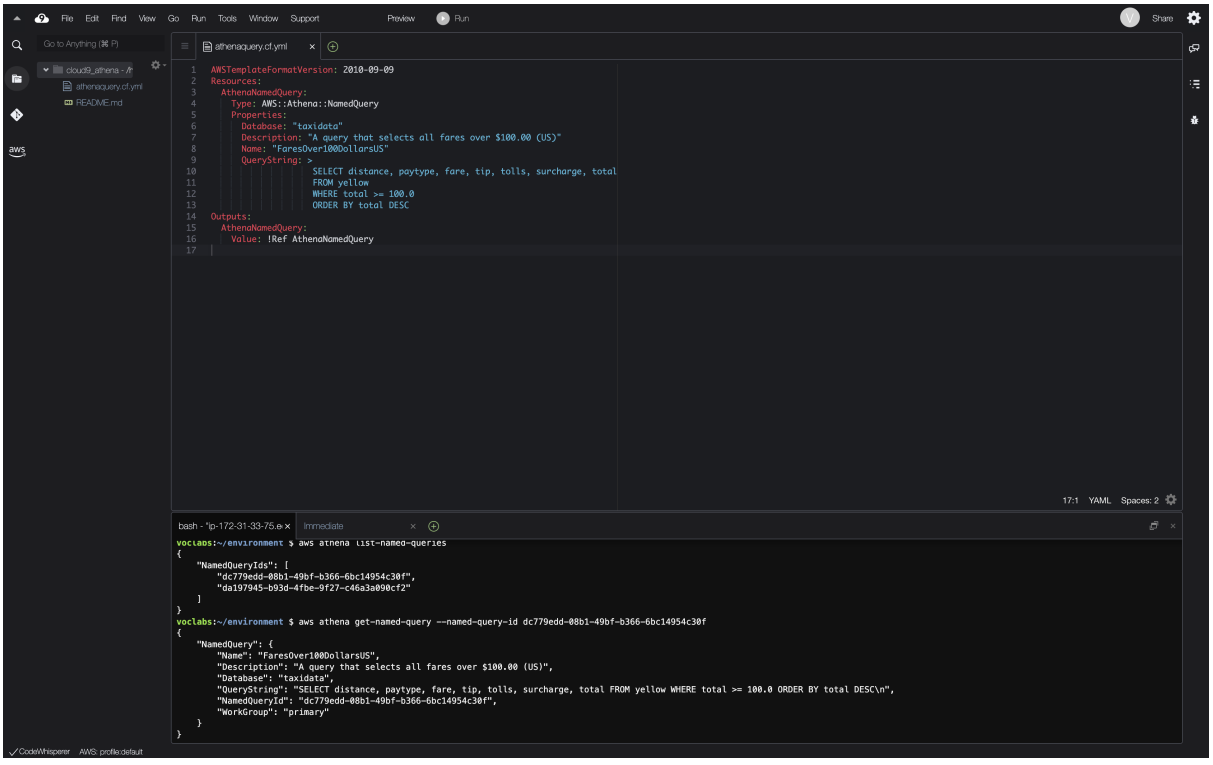
Confirming if CloudFormation stack created:

```
aws athena list-named-queries
```



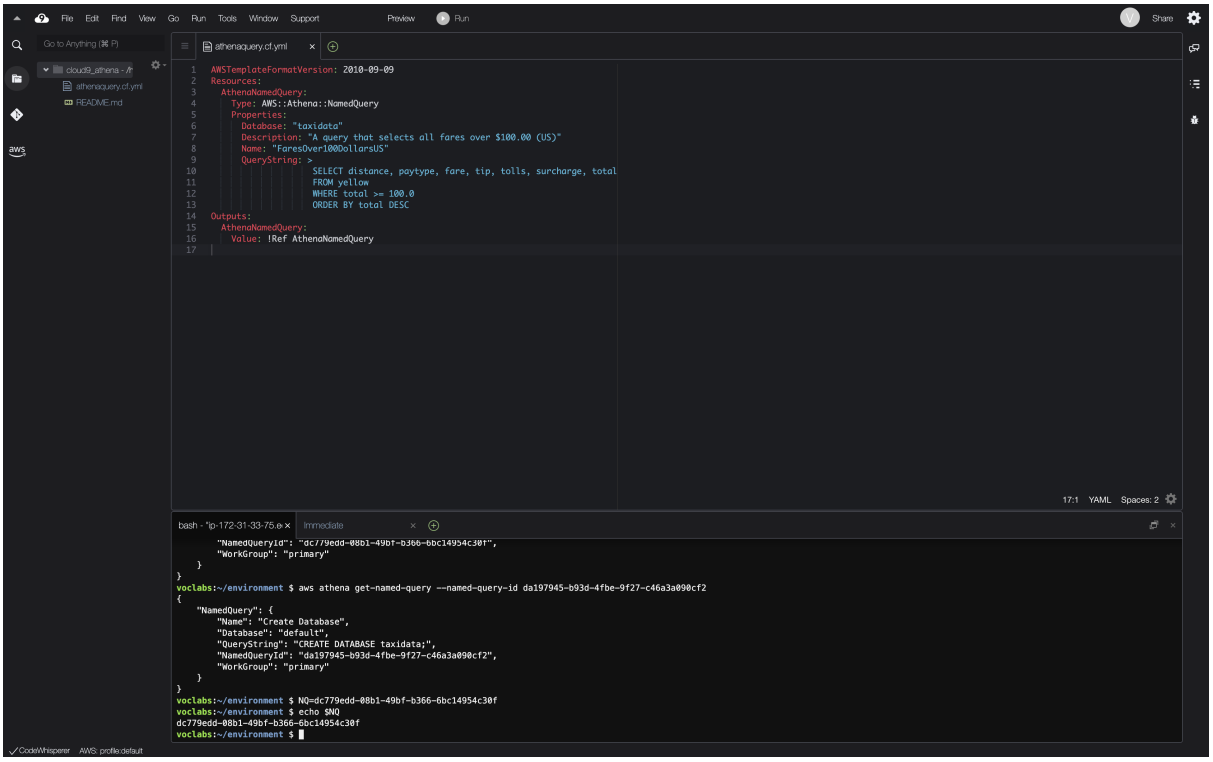
To retrieve the details of the named query:

```
aws athena get-named-query --named-query-id da197945-b93d-4fbe-9f27-c46a3a090cf2
```



Saving the named query ID as a bash variable

```
NQ=dc779edd-08b1-49bf-b366-6bc14954c30f
echo $NQ
```



Conclusion Remarks:

In conclusion, this assignment explores various aspects of AWS Athena and CloudFormation, providing a comprehensive understanding of these vital AWS services.

The first section of the assignment delves into the creation and querying of an AWS Glue database and table in Athena. This foundational step is crucial for data analysis and manipulation, serving as the backbone for subsequent data-related tasks.

Optimization is a key theme in this assignment, and the second section demonstrates the optimization of Athena queries by using buckets. By implementing this strategy, query performance is significantly enhanced, leading to more efficient data processing and reduced query execution times.

Furthermore, the assignment discusses another optimization technique: the use of partitions in Athena queries. Organizing data into partitions not only improves query speed but also contributes to cost reduction, making it an essential concept in the context of AWS Athena.

Athena views are another valuable tool covered in this assignment. This section elaborates on how to create and view data using Athena views. This approach simplifies complex queries and enhances data accessibility, contributing to more efficient data analysis.

The final section of the assignment focuses on the creation of Athena named queries using CloudFormation, with a specific emphasis on utilizing Cloud9. It provides a step-by-step guide, including creating the CloudFormation stack, confirming its creation, retrieving details of the named query, and saving the named query ID as a bash variable.

In summary, this assignment showcases a comprehensive understanding of AWS Athena and CloudFormation, essential tools in the field of data science and tech.