

Naya College

Data Engineering

AWS Exercise Book



Date: Feb 2022

Version: 3.0.1

Content

Disclaimer	4
Exercise 1 - AWS Console & AWS CLI	5
Part 1 - AWS CLI & AWS IAM User Settings	5
Part 2 - Testing the AWS CLI Access	6
Part 3 - Configuring AWS CLI	6
Part 4 - Creating a S3 Bucket	6
Part 5 - Upload a File to an Amazon S3 Bucket	7
Part 6 - AWS Official Documentations (Reference Examples)	8
Exercise 2 - Amazon EC2	9
Part 1 - Launch an Amazon EC2 Linux Instance	9
Part 2 - Connect via SSH to the Amazon EC2 Instance Remotely	11
Part 3 - Install PostgreSQL on the Amazon EC2 Instance	12
Part 4 - Connect Remotely to the PostgreSQL DB on Amazon EC2	14
Part 5 - Terminate the Amazon EC2 Instance	15
Exercise 3 - Amazon RDS	16
Part 1 - Create a New Amazon RDS MySQL Database	16
Part 2 - Establish a Connection to the Amazon RDS Instance	17
Part 3 - Load the Sample Database into the Amazon RDS Instance	17
Part 4 - Perform SQL Queries on the Sakila Database Data	17
Part 5 - Terminate the Amazon RDS MySQL Instance	18
Exercise 4 - Amazon Redshift	19
Part 1 - Launch a new cluster	19
Part 2 - Load Sample DB into Amazon Redshift	20
Part 3 - Query Amazon Redshift Sample Database	22
Part 4 - Terminate the Amazon Redshift Cluster	22
Exercise 5 - Amazon Athena	23
Part 1 - Sample Data Preparation	23
Part 2 - Create a new Database and Table on Amazon Athena	23
Part 3 - Run SQL Statements Using Amazon Athena	25
Part 4 - Create a Partitioned Table on Amazon Athena	25
Part 5 - Clean the Amazon Athena Exercise Environment	27
Exercise 6 - Amazon EMR	28
Part 1 - Deploy an Amazon EMR Cluster	28
Part 2 - Connect to the Amazon EMR Cluster Remotely	29
Part 3 - Run PySpark on Amazon EMR Cluster	30
Part 4 - Optional - Configure AWS CLI on Amazon EMR	32
Part 5 - Terminate the Amazon EMR Cluster	32
Exercise 7 - AWS Glue and Amazon Athena	34

Part 1 - Upload Sample Data into our Amazon S3 Bucket	34
Part 2 - Creating AWS Glue Crawler for the CSV file format:	35
Part 3 - Querying Data Stored in CSV Format via Amazon Athena	36
Part 4 - Converting to Parquet Format Using AWS Glue ETL	36
Part 5 - Querying data with Amazon Athena & Parquet Format	38
Part 6 - Delete the Exercise Resources	38
Exercise 8 - AWS Lambda Function	39
Part 1 - Create a the Amazon S3 Bucket Exercise Infrastructure	39
Part 2 - Create a New AWS Lambda Function	39
Part 3 - Upload Files and Monitor the Function Invocations	41
Exercise 9 - Amazon Kinesis	43
Part 1 - Create an Amazon Kinesis Data Stream Process	43
Part 2 - Create an Amazon Kinesis Data Firehose Process	44
Part 3 - Configure the Course VM/Docker with the Kinesis Agent	44
Part 4 - Run and Monitor the Kinesis Data Pipeline	46
Part 5 - Terminate the Exercise Services	47
Exercise 10 - Amazon Kinesis Analytics	48
Part 1 - Create an Amazon Kinesis Data Stream Process	48
Part 2 - Create an Amazon Kinesis Analytics Application	48
Part 3 - Prepare and Run the Random Events Data	50
Part 4 - Run SQL Statements on Real-Time Events	51
Part 5 - Terminate the Amazon Kinesis Analytics Processing	52
Solutions - Amazon Athena	53
Tables & Views	53
Partitioned Table	55

Disclaimer

As AWS are making constant changes to the services, it is possible that in certain cases you will encounter a mismatch between this exercise and with the actual AWS platform layout or features, including the AWS Management Console, code or versions. In such cases, please report to the instructor to receive alternative instructions or explanation on how to proceed.

Exercise 1 - AWS Console & AWS CLI

Overview:

In this exercise, we will set up your account on AWS, Set up your IAM Role, and communicate with the different services using the AWS console(web) and cli(local).

Note

Always make sure you are in the **us-east-1** region, in principle you don't have any permissions to create resources in a different region other than of us-east-1 (N. Virginia).

Part 1 - AWS CLI & AWS IAM User Settings

1. Follow the credential details provided by the instructor and use the following login url to access the Naya College AWS account:
<https://naya-college.signin.aws.amazon.com/console>
2. Get your **access_key** and **secret_access_key** from the **AWS console**:
 - a. In services, go to IAM and click on "Users".
 - b. Find your username, and click on it (make sure it is yours), go to "Security credentials".
 - c. Click on "Create Access Key", download the CSV file and save it for the rest of the AWS module.
3. Access your course environment (e.g. GCP Docker) and install AWS CLI using the following commands (SSH to the course environment - Docker Container):

```
$ cd ~/downloads

$ curl "https://awscli.amazonaws.com/awscli-exe-linux-x86_64.zip"
-o "awscliv2.zip"

$ unzip awscliv2.zip

$ sudo ./aws/install
```

- AWS CLI can also be installed on [additional operating systems](#) as well.

Part 2 - Testing the AWS CLI Access

1. After your AWS CLI is installed, type “**aws help**” - you should get a prompt with all of the available options (parameter) of the AWS CLI options (type ‘q’ to quit the help page).

```
aws help
```

Part 3 - Configuring AWS CLI

1. In the course environment, type “**aws configure**” (see example below).
2. Insert the access key & secret access key you’ve received in part 1.
3. Default region - **us-east-1**
4. Default Format - **None** (just click Enter)
5. Configure AWS CLI with the generated Access Keys Example:

```
$ aws configure
```

```
AWS Access Key ID [None]: <Type Your AccessKey>
```

```
AWS Secret Access Key [None]: <Type Your SecretAccessKey>
```

```
Default region name [None]: us-east-1
```

```
Default output format [None]: json
```

6. In order to verify that your credentials are set up correctly, type:

```
aws s3 ls
```

* This command will list all of the Amazon S3 buckets that exist on the account.

Part 4 - Creating a S3 Bucket

We will create this bucket twice, once from the web console and once from the AWS CLI.

1. In the AWS console, search for “S3” (Simple Storage Service).
2. Press on “Create Bucket”:
 - a. **Bucket Name should be:** “naya-college-student-**<your name>**”
 - b. **Region:** us-east-1
 - c. Click on “**Create bucket**”
3. After the bucket is created, upload any file into the bucket by your choice (small size).
4. In the AWS CLI, type “`aws s3 ls`” again, and make sure you see your bucket.

5. Delete the bucket using the AWS CLI command:

```
aws s3 rb s3://<bucket-name> --force
```

6. Create the bucket again, using the AWS CLI command (replace the bucket name):

```
aws s3 mb s3://<bucket-name> --region us-east-1
```

7. Make sure that your bucket exists using the web console and AWS CLI.

Part 5 - Upload a File to an Amazon S3 Bucket

1. Create a local file on the course VM/Container / or use an existing file (small size):

```
cd ~/tutorial/aws  
echo This is a test file uploading to AWS S3 > file.log
```

2. Create an S3 directory (also known as “key”) under the created bucket (the directory or key will be named by your choice - change <dir-name> to a different name):

```
aws s3api put-object --bucket <bucketName> --key <dir-name>/
```

3. Upload the created file into the S3 directory:

```
aws s3 cp <file-name> s3://<bucket-name>/<dir-name>/<file-name>
```

4. Verify the uploaded file both by the console and AWS CLI:

```
aws s3 ls s3://<bucket-name>/<dir-name>/
```

5. Delete the uploaded file and verify via AWS CLI :

```
aws s3 rm s3://<bucket-name>/<dir-name>/<file-name>  
aws s3 ls s3://<bucket-name>/<dir-name>/
```

6. Delete the created directory (key) and verify via AWS CLI:

```
aws s3 rm s3://<bucket-name>/<dir-name>/  
aws s3 ls s3://<bucket-name>/
```

Part 6 - AWS Official Documentations (Reference Examples)

- [Make Bucket](#)
- [List](#)
- [Remove file/directory](#)
- [Copy](#)

Exercise 2 - Amazon EC2

Overview:

In the following exercise, you will launch an Amazon EC2 instance (free tier Linux OS) from the AWS “Quick Start”, connect remotely using SSH, and install a PostgreSQL database while performing a remote connection.

Part 1 - Launch an Amazon EC2 Linux Instance

1. Navigate to the Amazon EC2 service on the AWS console and click on “Launch instance”.
2. On the “Quick Start” (left side) search/select the “**Amazon Linux 2 AMI**” (AMI - Amazon Machine Image) and click “Select” (left side) - select the latest version (make sure that the selected region is set to us-east-1 “N.Virginia”).
3. **Choose an Instance Type:** Select “**t2.micro**” under “**Free tier eligible**” and click on “Next: Configure Instance Details” (lower left side).
4. Click on “Next: Add Storage”, modify the volume size (GiB) to 30GB (max allowed storage size for free tier), make sure that the Volume Type is set to SSD drive (gp3).
5. Click on “Next: Add Tags”
 - a. Click on “Add Tag”
 - b. **Key:** “Name”
 - c. **Value:** <student-name>-ec2

Step 5: Add Tags

A tag consists of a case-sensitive key-value pair. For example, you could define a tag with key = Name and value = Webserver.

A copy of a tag can be applied to volumes, instances or both.

Tags will be applied to all instances and volumes. [Learn more](#) about tagging your Amazon EC2 resources.

Key (128 characters maximum)	Value (256 characters maximum)
name	student1-ec2

6. Click on “Next: Configure Security Group”.
7. Step 6: Configure Security Group
 - a. In this section we’ll add the PostgreSQL default port to allow remote connection from the internet.
 - b. Click on “Add Rule”
 - c. **Type:** PostgreSQL

- d. **Source:** 0.0.0.0/0

Note

For the purpose of this exercise, the allowed inbound connection will be set from any place on the internet, this is considered to be a critical security issue and should not be used in production environments. As best practice, allow inbound connection only from trusted sites (e.g. IP/DNS) by specifying the IP/DNS in the source fields.

Type ⓘ	Protocol ⓘ	Port Range ⓘ	Source ⓘ	Description ⓘ
SSH	TCP	22	Custom 0.0.0.0/0	e.g. SSH for Admin Desktop
PostgreSQL	TCP	5432	Custom 0.0.0.0/0	e.g. SSH for Admin Desktop

- e. Click on “Review and Launch”.

8. You can ignore the following message:

Step 7: Review Instance Launch

Please review your instance launch details. You can go back to edit changes for each section. Click **Launch** to assign a key pair to your instance and complete the launch process.

⚠ Improve your instances' security. Your security group, launch-wizard-7, is open to the world.
Your instances may be accessible from any IP address. We recommend that you update your security group rules to allow access from known IP addresses only. You can also open additional ports in your security group to facilitate access to the application or service you're running, e.g., HTTP (80) for web servers. [Edit security groups](#)

9. Click on “Launch”.

10. “Select an existing key pair or create a new key pair”:

- Select “Choose an existing key pair”
- Select a key pair: **StudentKeyPair**
- Check the “acknowledge” checkbox.
- Click on “Launch Instances”.

Select an existing key pair or create a new key pair

A key pair consists of a **public key** that AWS stores, and a **private key file** that you store. Together, they allow you to connect to your instance securely. For Windows AMIs, the private key file is required to obtain the password used to log into your instance. For Linux AMIs, the private key file allows you to securely SSH into your instance.

Note: The selected key pair will be added to the set of keys authorized for this instance. Learn more about [removing existing key pairs from a public AMI](#).

☒ I acknowledge that I have access to the selected private key file (StudentKeyPair.pem), and that without this file, I won't be able to log into my instance.

Cancel
Launch Instances

- This will be used for establishing an SSH connection remotely (e.g. from the course VM/Docker or any other SSH client).

- The StudentKeyPair.pem file can be downloaded from this [link](#).
 - Or, download the StudentKeyPair.pem file directly to a Linux VM/Docker using wget:

```
$ wget --no-check-certificate 'https://docs.google.com/uc?export=download&id=1tN41Sl42yqe878ihGBqM9LxfPk0McwHf' -O ~/downloads/StudentKeyPair.pem
```

- Click on “View instances”.
- Add your name to the “Name” column in the Instances list.
- Proceed to the next part once the “Instance state” shows “Running”.

Part 2 - Connect via SSH to the Amazon EC2 Instance Remotely

1. Right click on your specific created EC2 instance and click on “Connect”.
2. Click on the “SSH client” tab and copy the example SSH connection, it holds the host name (the Amazon EC2 public DNS):

Connect to instance [Info](#)

Connect to your instance i-0b6c4ed2a4b590fb4 (Stu-EC2) using any of these options

EC2 Instance Connect | Session Manager | **SSH client**

Instance ID

i-0b6c4ed2a4b590fb4 (Stu-EC2)

1. Open an SSH client.
2. Locate your private key file. The key used to launch this instance is StudentKeyPair.pem
3. Run this command, if necessary, to ensure your key is not publicly viewable.
 `chmod 400 StudentKeyPair.pem`
4. Connect to your instance using its Public DNS:
 `ec2-3-86-221-175.compute-1.amazonaws.com`

Example:

`ssh -i "StudentKeyPair.pem" ec2-user@ec2-3-86-221-175.compute-1.amazonaws.com`

3. Paste the SSH command in a terminal (e.g. course VM/Docker) and make sure that the StudentKeyPair.pem can be found in the current path (or specify the full path):

```
ssh -i "StudentKeyPair.pem"  
ec2-user@<ec2-3-86-221-175.compute-1.amazonaws.com>
```

- a. Update the permission for the **StudentKeyPair.pem** file in case of error:
“Permissions 0664 for ‘StudentKeyPair.pem’ are too open” - change for READ only permissions:

```
$ chmod 400 StudentKeyPair.pem
```

4. Successful SSH connection prompt:

```
--|  --|  )  
_| (  /   Amazon Linux 2 AMI  
---|\---|---|  
  
https://aws.amazon.com/amazon-linux-2/  
-bash: warning: setlocale: LC_CTYPE: cannot change locale (UTF-8): No such file or directory  
[ec2-user@ip-172-31-62-232 ~]$
```

Part 3 - Install PostgreSQL on the Amazon EC2 Instance

1. Run the following command to add the relevant repositories.

```
$ sudo tee /etc/yum.repos.d/pgdg.repo<<EOF  
[pgdg13]  
name=PostgreSQL 13 for RHEL/CentOS 7 - x86_64  
baseurl=https://download.postgresql.org/pub/repos/yum/13/redhat/r  
hel-7-x86_64  
enabled=1  
gpgcheck=0  
EOF
```

2. Run the command below to enable EPEL repository on Amazon EC2 Linux 2 server:

```
$ sudo amazon-linux-extras install epel -y
```

3. Install PostgreSQL version 13 on Amazon EC2 Linux 2:

```
$ sudo yum -y install postgresql13 postgresql13-server
```

4. Generate the initial database configurations file before starting the service:

```
$ sudo /usr/pgsql-13/bin/postgresql-13-setup initdb
```

- a. Note that the main configuration file for the PostgreSQL database server path & file are at: `/var/lib/pgsql/13/data/postgresql.conf`

5. Enable the PostgreSQL service to start on boot:

```
$ sudo systemctl enable postgresql-13
```

6. Start the PostgreSQL service:

```
$ sudo systemctl start postgresql-13
```

7. Verify the PostgreSQL service:

```
$ sudo systemctl status postgresql-13
```

8. Verify PostgreSQL connection locally (from the Amazon EC2 instance):

```
$ sudo su - postgres
$ psql

psql (13.2)
Type "help" for help.

postgres=#
postgres=# \q
```

9. Configure the PostgreSQL to accept remote access:

a. Configure the `postgresql.conf` file:

```
$ vi /var/lib/pgsql/13/data/postgresql.conf
```

- Locate the “`#listen_addresses`” parameter.
- Uncomment the “`listen_addresses`” parameter.
- Modify the parameter to accept incoming connections from any IP/DNS:

```
...
listen_addresses = '*'
...
```

d. Save the file.

e. Configure the `pg_hba.conf` file:

```
$ vi /var/lib/pgsql/13/data/pg_hba.conf
```

f. Scroll to the end of the file and add the following line under “`# IPv4 local connections`”:

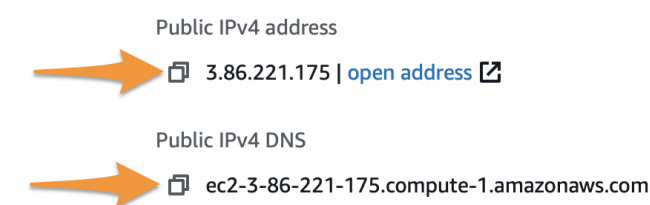
```
# IPv4 local connections:
host      all             all             0.0.0.0/0       trust
```

- g. Save the file, exit back to ec2-user (AWS user), and restart the PostgreSQL service:

```
$ exit
$ sudo systemctl restart postgresql-13
```

Part 4 - Connect Remotely to the PostgreSQL DB on Amazon EC2

1. Launch [DBeaiver](#) (Community & Windows/Linux) and create a new connection selecting PostgreSQL as the database engine.
2. Navigate to Amazon EC2 > Instances, click on your created instance and copy the Amazon EC2 instance public IP address or the public DNS:



3. **Connection details:**
 - a. **Host:** <EC2-DNS>
 - b. **Pott:** 5432
 - c. **Database:** postgres
 - d. **User:** postgres
 - e. **Password:** postgres
4. Test and save the connection.
5. Right click on the PostgreSQL newly created connection and click on "Connect".
6. Open a new SQL editor page (right click).
7. **Optional:**
 - a. Download the "covid-19.csv" sample data file locally (e.g. Windows/MacBook) from this [link](#).
 - b. Create a table using DBeaiver connection to PostgreSQL

```
CREATE TABLE covid_19 (
  cdc_case_earliest_dt    date,
  cdc_report_dt           date,
  pos_spec_dt             date,
  onset_dt                date,
  current_status          varchar(100),
  gender                  varchar(100),
  age_group               varchar(100),
```

```
race_ethnicity_combined varchar(100),
hosp_yn                 varchar(100),
icu_yn                  varchar(100),
death_yn                varchar(100),
medcond_yn              varchar(100)
);
```

- c. Right click on tables and “Refresh”.
- d. Import the sample CSV file into the covid-19 table using the following procedure:
 - Right click on the DBeaver PostgreSQL “covid_19” table (under Schemas → public → tables) and click on “Import Data”.
 - Click in “CSV” and “Next”
 - Select the downloaded covid-19.csv file located locally and click on “Next” several times and click on “Start”.
 - Verify that the data was imported into the “covid_19” table:

```
select * from covid_19;
```

Part 5 - Terminate the Amazon EC2 Instance

1. On the AWS console, select your Amazon EC2 instance.
2. Click on “Instance state” and “Terminate instance”.

Exercise 3 - Amazon RDS

Overview:

In the following exercise, you will create a new Amazon RDS MySQL database, load a sample data, query it to get insights and terminate the database.

Part 1 - Create a New Amazon RDS MySQL Database

1. Navigate to the Amazon RDS page on the AWS console.
2. Click on "Create database".
3. **Choose a database creation method:** Select "Standard create".
4. **Engine options:** Select the MySQL engine type.
5. **Version:** 8.0.27
6. **Templates:** Make sure to select "**Free tier**".
7. **Availability and durability:** No changes
8. **Settings:**
 - a. **DB instance identifier:** Type <student-name>-db (this will create a database instance with the typed name, this would not be a database name).
 - b. **Credentials Settings:**
 - i. **Master username:** root
 - ii. **Master password:** root1234
9. **DB instance class:**
 - a. **DB instance class:** Make sure that "**db.t2.micro**" instance type is selected.
10. **Storage:**
 - a. **Allocated storage:** Make sure that **20** GiB and General Purpose SSD are selected.
11. **Connectivity:**
 - a. **Public access:** Select "**Yes**" to allow remote connection to the database instance.
 - b. **Additional configuration:** Make sure that the default MySQL port is specified **3306**.
12. Click on "**Create database**".
13. Monitor your database instance deployment "**Status**" (click on refresh as the creation may take several minutes to complete).

Part 2 - Establish a Connection to the Amazon RDS Instance

1. Make sure that your new Amazon RDS database instance was created (verify the value on the “Status” column on the Amazon RDS Databases page - should be “Available”)
2. Launch [DBeaver](#) and configure a connection to the Amazon RDS MySQL database instance.
 - a. Click on your database instance to extract the instance “Endpoint” which will be used as the hostname/DNS when setting the connection for your MySQL database instance.
 - b. **Example:** student.csnhj38wg3oc.us-east-1.rds.amazonaws.com
 - c. Test the connection.

Part 3 - Load the Sample Database into the Amazon RDS Instance

1. Download the MySQL sample database locally: [sakila.sql](#) - this file holds an export of the Sakila sample database (DDL & DML statements).
2. Import the [Sakila sample database](#) into the Amazon RDS MySQL instance:
 - a. Use DBeaver.
 - b. Right click on the DBeaver connection and select “Connect”.
 - c. Right click on the DBeaver connection and select “SQL Editor”.
 - d. Click on “File” and “Open File...”. Select the sakila.sql files (which you downloaded in step 1).
 - e. Copy the content of the sakila.sql script to the SQL Editor associated with the MySQL connection.
 - f. Click on the “**Execute SQL Script**” button and monitor the import process (the process should take a few minutes to complete).
 - g. Verify by running: `show tables;` (23 tables).

Part 4 - Perform SQL Queries on the Sakila Database Data

1. The music of Queen and Kris Kristofferson have seen an unlikely resurgence. As an unintended consequence, films starting with the letters K and Q have also soared in popularity. Use subqueries to display the titles of movies starting with the letters K and Q whose language is English.
2. Use subqueries to display all actors who appear in the film Alone Trip.

3. You want to run an email marketing campaign in Canada, for which you will need the names and email addresses of all Canadian customers.
4. Write a query to display how much business, in dollars, each store brought in.
5. List the top five genres in gross revenue in descending order. (Hint: you may need to use the following tables: category, film_category, inventory, payment and rental).

Part 5 - Terminate the Amazon RDS MySQL Instance

1. Via AWS console, navigate to Databases tab, mark your database instance.
2. Click on "Actions" and "Delete"
3. Make sure to **uncheck** the "Create final snapshot?" option.
4. Check only the *"I acknowledge that upon instance deletion, automated backups, including system snapshots and point-in-time recovery, will no longer be available"* option.
5. Delete the MySQL database instance.

Exercise 4 - Amazon Redshift

Overview:

In this exercise, we will create a new redshift cluster, load sample data and query it. We will also terminate the resource at the end of the class.

Part 1 - Launch a new cluster

Access the AWS console and open the Amazon Redshift service, click on “**Create cluster**”.

1. Cluster Configuration:

- **Cluster Identifier:** naya-student-<your name>
- **Cluster type:** Production
- **Choose the size of the cluster:** “I’ll choose”
- **Node Type:** dc2.large
- **Nodes:** 1

2. Node Configuration (no action required - view selected):

- **Node Type:** dc2.large
- **Cluster Type:** 1 Node
- **Compute:** 2 vCPU (gen 2) / node x 1 = 2 vCPU
- **Storage capacity:** 160 GB x 1 nodes = 160 GB

3. Sample data

- Do not check the “Load sample data” option.

4. Database configurations

- **Admin user name:** root
- **Admin user password:** Root1234

5. Cluster Permissions

- Click on the “Associate IAM roles” button.
- **IAM roles:** Select the “StudentRedshiftFull” IAM Role with the cluster - this will allow the cluster to load data from Amazon S3 into Amazon Redshift.
- Click on “Associate IAM role”.

6. Additional Configuration:

- Click on the “Use defaults” button to see the configuration options.
- **Network and security:**
 - Make sure “**Publicly accessible**” is marked as “**Enable**”.
- **Database configurations:**
 - **Database Name:** dev_<your name>
 - **Database port:** 5439 (do not change)

7. Launch the Amazon Redshift Cluster:

- Click on “Create cluster”.
- This can take several minutes.

Part 2 - Load Sample DB into Amazon Redshift

Note:

You should wait until the cluster is successfully created (view the “Status” value). Then copy the endpoint (Amazon Redshift hostname) without the port number to the host in your DBeaver (SQL client), you can use any other available PostgreSQL client.

Using DBeaver, you will need to create a PostgreSQL connection to Amazon Redshift. DBeaver (SQL client) can be downloaded from [here](#).

1. Define a new PostgreSQL connection with DBeaver:

- Create a new connection (search for Redshift connection).
- Copy the connection details from the Amazon Redshift clusters tab (right side).
- Note that the endpoint (Redshift hostname/DNS to use for remote connection) includes the port and the database name (should be omitted for the “Host” value in the PostgreSQL connection).
- Database (name) should be as defined when configuring the Amazon Redshift.
- Redshift default port: 5439

2. Verify that you can query the cluster by running the following query:

```
select * from pg_tables limit 10;
```

3. Use the following [guide](#) to view the required steps for uploading sample data from Amazon S3 bucket into your Amazon Redshift cluster:

- Create the sample tables according to the DDL scripts (7 tables overall).
- Download the [ticketdb.zip](#) file locally and upload to your Amazon S3 bucket.
 - Optionally, you can download the file directly to Linux using the `wget` command (using the course Docker Container/CLI based environment):

```
cd /home/naya/tutorial/aws
```

```
wget
```

```
https://docs.aws.amazon.com/redshift/latest/gsg/samples/ticketdb.zip
```

- Unzip the [ticketdb.zip](#) to extract the sample data files:

```
sudo yum -y install unzip
```

```
unzip ticketdb.zip
```

- Investigate the content of the sample data files (use `vi` or `less`).
- Upload the .txt sample data files into your Amazon S3 bucket (can be done by AWS CLI or AWS Management Console).

```
aws s3 cp <local_path> <s3_bucket_path>
```

- Load the data into the Amazon Redshift tables:
 - Use Redshift Editor or DBeaver (or any other SQL connection).
 - Use Redshift [COPY](#) command to load from the S3 sample data files into the created tables, example:

```
copy users from 's3://<myBucket>/allusers_pipe.txt'
credentials
'aws_iam_role=arn:aws:iam::204597968057:role/StudentRedshiftFull'
delimiter '|' region 'us-east-1';
```

- Apply the COPY command for all the created tables to load data from Amazon S3 into your Amazon Redshift cluster.

4. Verify that all of the statements succeeded by querying the tables

- Run the sample queries from step 5 in the [guide](#).

Ignore the following error if it appears:

[Amazon](500310) Invalid operation: duplicate key violates unique constraint "pg_class_relname_nsp_index" (possibly caused by concurrent transaction conflict);

Part 3 - Query Amazon Redshift Sample Database

Answer the following questions using SQL Queries:

- What is the most profitable venue?
- Which city supplies the largest growth?
- What is the AVG, Median amount of tickets sold per venue?
- Who was the best seller for the last year?
- Which seller has had a performance decline over the last year?

Part 4 - Terminate the Amazon Redshift Cluster

- On the AWS Management Console, click on the "Clusters" tab.
- Select your cluster (check box), and click on the "Actions" dropdown list.
- Select "Delete" and **uncheck** the "Create final snapshot" option.
- Click on "Delete cluster"
- Make sure that the cluster was deleted in the "Clusters" tab (Status - Deleting).

Exercise 5 - Amazon Athena

Overview:

In this exercise, we will create a new database and table on Amazon Athena and point the Athena table to sample data (CSV file) located on Amazon S3 bucket, later we will define a new schema holding partitions.

Part 1 - Sample Data Preparation

1. Create a new folder (key) named - "athena" under your student bucket (created in exercise 1).
2. Create an additional folder (key) named - "athena_results" under your student bucket.
3. Copy the sample CSV data file (covid-19.csv) from an S3 path (bucket name: naya-college-exercise-files) into your student bucket and under the created "athena" folder.
 - Alternatively, the file can be downloaded from this [link](#) - if downloaded locally, you'll be required to perform an upload operation into your s3 bucket).
4. **Source path:** s3://naya-college-exercise-files/covid-19.csv
5. **Destination path:** s3://naya-college-student-<your name>/athena/

Sample command:

```
aws s3 cp s3://naya-college-exercise-files/covid-19.csv  
s3://naya-college-student-<your name>/athena/
```

6. Verify that the sample CSV file was copied/uploaded successfully (AWS CLI & AWS Management Console).

```
aws s3 ls s3://naya-college-student-<your name>/athena/
```

Part 2 - Create a new Database and Table on Amazon Athena

1. Connect to the Amazon Athena service on Amazon Management Console.
2. Note the following message: *"Before you run your first query, you need to set up a query result location in Amazon S3"*, apply the following to set the result location:
 - Click on "View settings" and then click on "Manage".
 - Under the "Location of query result" click on "Browse S3".
 - Select the "athena_results" folder, click on "Choose" and click on "Save".

- Go back to the Editor tab.
- 3. Create a new database with your name as prefix and the string “_db”, for example:
<student_name>_db.

```
create database <student_name>_db;
```

- 4. Create a new table via the Amazon Athena Wizard:
 - Click on “Create” and select “S3 bucket data” under “Create a table from datasource”
 - Select “from S3 bucket data” as the data source option.
 - **Table Name:** Type a table name (example: covid19_data).
 - **Database configuration:** Select your database according to the provided name.
 - **Dataset:**
 - **Location of Input Data Set:**
 - Specify the S3 path for the covid-19.csv file, for example:
s3://naya-college-student-<yourname>/athena/
 - **Data format:**
 - Select “CSV” as the data format and click on “Next”.
 - **Column details:**
 - Click on “Bulk add columns” and paste the following schema:

```
cdc_case_earliest_dt date,  
cdc_report_dt date,  
pos_spec_dt date,  
onset_dt date,  
current_status string,  
gender string,  
age_group string,  
race_ethnicity_combined string,  
hosp_yn string,  
icu_yn string,  
death_yn string,  
medcond_yn string
```

- Click on “Add” and verify that all columns exist.
- Click on “Create table”.
- Verify that a new table was created under the “Tables and views” (left side).

Part 3 - Run SQL Statements Using Amazon Athena

1. Verify the sample CSV data by running SQL statements via Amazon Athena editor.
2. Make sure to choose your database;
3. For example:

```
select * from <table-name> limit 100;
```

4. Find the gender distribution for “current_status” equal to “Laboratory-confirmed case”.
5. Return the `cdc_case_earliest_dt` by Male and Female count distribution.
6. Which is the most affected age group by the Covid-19 virus?

Part 4 - Create a Partitioned Table on Amazon Athena

1. Make sure that you have the table DDL, this can be generated by:

```
show create table <table-name>
```

2. Use the same table DDL to generate a new partitioned table with the same schema.
3. The new table should be partitioned by the `cdc_case_earliest_dt` column extracting the year and month values.
4. The data should be stored as Parquet data format.
5. Use the following AWS official [link](#) for syntax examples.
6. Create a new S3 folder as the path for storing the partitioned data named “athena_partitions”.
7. The partitioned data should be inserted to the partitioned table using a INSERT from SELECT statement, this will create the partitioned S3 folders automatically.
8. The following example can be used as a reference for the solution:

```
CREATE EXTERNAL TABLE IF NOT EXISTS <partitioned-table> (  
  cdc_case_earliest_dt date,  
  cdc_report_dt date,  
  pos_spec_dt date,  
  onset_dt date,  
  current_status string,  
  gender string,  
  age_group string,
```

```
    race_ethnicity_combined string,
    hosp_yn string,
    icu_yn string,
    death_yn string,
    medcond_yn string
)
PARTITIONED BY (case_year integer, case_month integer)
STORED AS parquet
LOCATION 's3://<bucket>/<athena-partitioned-path>';

insert into <partitioned-table>
select cdc_case_earliest_dt,
       cdc_report_dt,
       pos_spec_dt,
       onset_dt,
       current_status,
       gender,
       age_group,
       race_ethnicity_combined,
       hosp_yn,
       icu_yn,
       death_yn,
       medcond_yn,
       cast(year(cdc_case_earliest_dt) as integer),
       cast(month(cdc_case_earliest_dt) as integer)
from <original-table>;

MSCK REPAIR TABLE <partitioned-table>;
SHOW PARTITIONS <partitioned-table>;

select cdc_case_earliest_dt,
       case_year,
       case_month
from <partitioned-table>
limit 100;
```

9. View the S3 partitions folders (visually) for year and month.

Part 5 - Clean the Amazon Athena Exercise Environment

1. Drop the database and all created tables using a command.
2. Delete the CSV file and the “athena/” and “athena_partitions” directories from the S3 path.

Exercise 6 - Amazon EMR

Overview:

In this exercise, we will deploy a one instance (node) EMR cluster (Elastic Map Reduce - AWS Hadoop Service) and perform a remote connection to the node using SSH, at the end we will drop the Amazon EMR cluster.

Instructor

Add ports 22 & 8888 to the security [sg-0f125339131b9f2d7](#) group once all students deployed the Amazon EMR env.

Part 1 - Deploy an Amazon EMR Cluster

The main objective is to deploy a single node (master only) Amazon EMR cluster and running Spark code while establishing a remote connection to the Hadoop Cluster:

2. Navigate to the Amazon EMR service > Clusters.
3. Click on "Create cluster".
4. Click on "**Go to advanced options**" (upper left) and update the following parameters:
 - **Release:** emr-6.5.0
 - **Applications** (make sure that is selected and check/uncheck if required):
 - Hadoop
 - Hive
 - Hue
 - Spark
 - Click on "Next"
 - **Cluster Composition:**
 - Make sure that the "**Instance group configuration**" is set to "**Uniform instance groups**".
 - **Networking:** No changes required
 - **Cluster Nodes and Instances:**
 - Set the "**Instance count**" is set for **0** on both the Core and Task nodes - this will create a single node cluster (master only - **c4.4xlarge**).
 - **Make sure that the "Master" will have only 1 node.**
 - Click on "Next"

- **Cluster name:** <Student-name>_emr
- **Termination protection:** Uncheck
- Click on “Next”
- **EC2 key pair:**
 - **Make sure to select the: StudentKeyPair**
 - This will be used for establishing an SSH connection remotely (e.g. from the course VM/Docker or any other SSH client).
 - The **StudentKeyPair.pem** file can be downloaded from this [link](#).
- **“EC2 security groups”:**
 - Make sure that the configuration is set according to the screenshot below:

Type	EMR managed security groups EMR will automatically update the selected group	Additional security groups EMR will not modify the selected groups
Master	Default: sg-0f125339131b9f2d7 (ElasticMapRedu ▾)	No security groups selected 
Core & Task	Default: sg-016815b639592fa46 (ElasticMapRedu ▾)	No security groups selected 

- Click on “Create cluster”.
- Monitor the cluster deployment process, should change from “Starting” to “Running”
(the process should run for a few minutes - avg of 10 min).

Part 2 - Connect to the Amazon EMR Cluster Remotely

The main objective is to connect to the Amazon EMR cluster using a remote SSH connection from a Linux/UNIX based operating system (e.g. course VM , Windows 10 Bash or MacBook terminal).

1. Download the **StudentKeyPair.pem** file to be further used for establishing an SSH connection, download using `wget`:

```
wget --no-check-certificate 'https://docs.google.com/uc?export=download&id=1tN41Sl42yqe878ihGBqM9LxfPk0McwHf' -O ./StudentKeyPair.pem
```

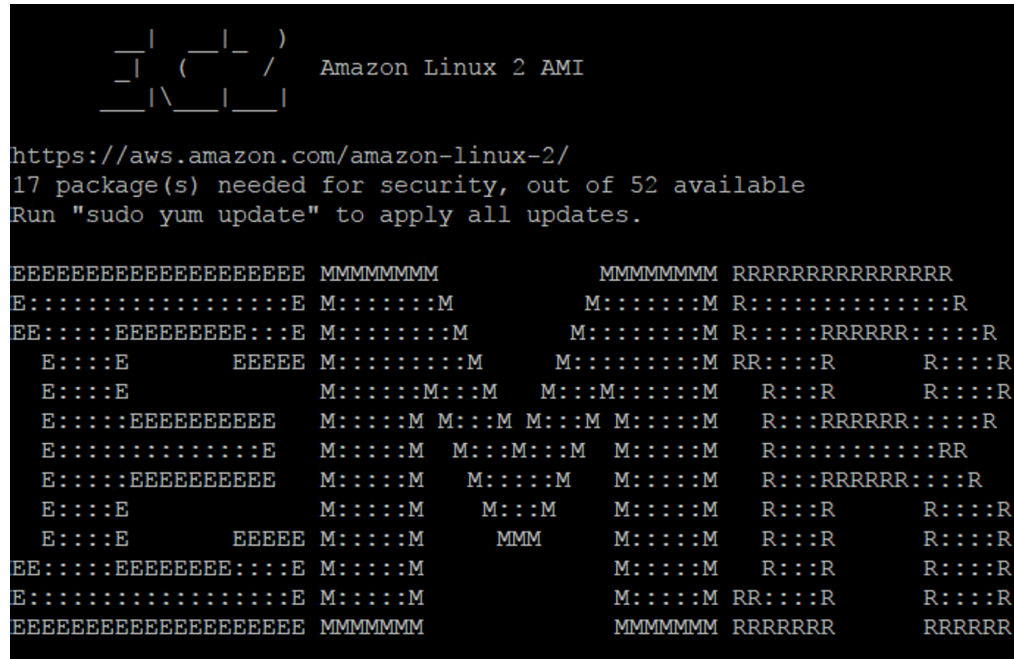
2. Update the permission for the **StudentKeyPair.pem** file in case of error: “*Permissions 0664 for 'StudentKeyPair.pem' are too open*” - change for READ only permissions:

```
chmod 400 ./StudentKeyPair.pem
```

3. Get the "Master public DNS" by clicking on the cluster name from Amazon EMR Cluster page, for a SSH example click on "Connect to the Master Node Using SSH" link.
4. Create an SSH Connection using the **StudentKeyPair.pem** file, for example:

```
ssh -i StudentKeyPair.pem
hadoop@<ec2-18-208-181-186.compute-1.amazonaws.com>
```

5. Amazon EMR successful connection screenshot:



Part 3 - Run PySpark on Amazon EMR Cluster

1. Run the following code using PySpark.
2. Launch PySpark:

```
pyspark
```

3. PySpark Code:

- The `parallelize()` method takes a Pythonic iterable object and creates an RDD with its elements.

```
my_list = [12, 23, 34, 45, 56, 67, 78, 89, 90]
rdd1 = sc.parallelize(my_list)
```

```
type(rdd1)

rdd2 = rdd1.map(lambda x: x/2)
print(rdd2.collect())

rdd3 = rdd1.filter(lambda x: x%2==0)
rdd3.collect()
exit()
```

4. Download text example file to the Amazon EMR cluster Master node (Moby Dick):

```
wget --no-check-certificate 'https://docs.google.com/uc?
export=download&id=1aPCWWkTLTucGkvHvdFb0CkyeztNPxNLs' -O
./melville-moby_dick.txt
```

5. Upload the **melville-moby_dick.txt** into HDFS path under /tmp (OS command):

- To exit from PySpark type: `exit()`

```
hdfs dfs -put melville-moby_dick.txt /tmp
hdfs dfs -ls /tmp
```

6. Launch PySpark (again):

```
pyspark
```

7. Load the **melville-moby_dick.txt** file into Spark Context, verify and return the 15 first lines from the file/book:

```
text = sc.textFile("/tmp/melville-moby_dick.txt")
text.take(15)
```

8. Complete book word count (return only the first 20 words):

```
text = sc.textFile("/tmp/melville-moby_dick.txt").flatMap(lambda
line: line.split(" "))

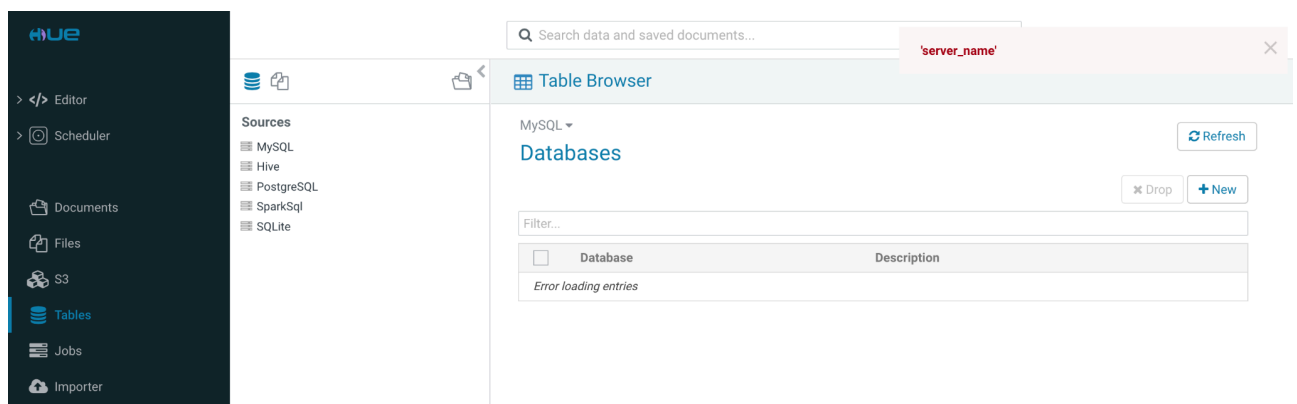
wordCounts=text.map(lambda word: (word, 1)).reduceByKey(lambda a,b:a
+b)
```

```
for v in wordCounts.collect()[:20]:  
    print(v)  
  
# Run again for timing purposes
```

9. Connect to HUE installed on the Amazon EMR using port 8888 and the following example URL (change the DNS according to your Amazon EMR deployment):

<http://ec2-54-90-158-165.compute-1.amazonaws.com:8888>

- Explore the Amazon EMR HUE version.



Part 4 - Optional - Configure AWS CLI on Amazon EMR

1. This is an optional exercise only.
2. Configure AWS CLI as performed previously from the Amazon EMR cluster.

Part 5 - Terminate the Amazon EMR Cluster

1. On the AWS Management Console, click on the EMR and Clusters.
2. Mark your Amazon EMR cluster (identified by your name).
3. Click on "Terminate" and on "Terminate" again with the prompt window.
4. Make sure your Amazon EMR cluster is marked as "Terminating".

Instructor

Delete ports 22 & 8888 from the [sg-0f125339131b9f2d7](#) security group once all students terminated their env.

Exercise 7 - AWS Glue and Amazon Athena

Overview:

In the following exercise, we will benchmark different file formats performance using AWS Glue and Amazon Athena. At the start we'll upload a CSV file to Amazon S3 while using AWS Glue crawler to create a new table. In the second part of this exercise, we'll launch a PySpark script using AWS Glue, in order to transform the csv file into Parquet files and create the new schema with AWS Glue crawler. Lastly we will perform a performance benchmark between the CSV based table and the Parquet based table using Amazon Athena.

Part 1 - Upload Sample Data into our Amazon S3 Bucket

Note

Amazon S3 refers to folders as **Prefix**.

1. Create a new prefix in your student S3 bucket: **"athena-exercise"**.
2. Within the newly created prefix, create a sub-prefix in your S3 bucket named: **"csv"**.
3. Create another prefix under the "athena-exercise" folder named: **"parquet"**.
4. Verify that you have the following Amazon S3 paths:
`"s3://<your_s3_bucket>/athena-exercise/csv"`
`"s3://<your_s3_bucket>/athena-exercise/parquet"`
5. In order to move data quickly to your bucket, you will copy it from another S3 bucket via the AWS CLI, note that the CSV file is located in Amazon S3 bucket named:
naya-college-exercise-files, in prefix **exercise-6-athena-glue**.
6. Run the following command from the VM using AWS CLI to copy the CSV file into your Amazon S3 bucket/prefix:

```
aws s3 cp  
s3://naya-college-exercise-files/exercise-6-athena-glue/SalesData  
.csv s3://<your_bucket>/athena-exercise/csv/
```

7. Note that this data will be queried from Athena, so it must be alone in the prefix.
8. Verify that the data exists in your bucket in the AWS web console (both by AWS CLI and AWS Management Console):

```
aws s3 ls s3://<your_bucket>/athena-exercise/csv/
```

Part 2 - Creating AWS Glue Crawler for the CSV file format:

1. Navigate to the AWS Glue service in AWS Management Console.
2. Click on **"Databases"** on the left menu, in order to create a new database.
3. Click on **"Add database"**, name it as: **"student_<your-name>_db"** and click on **"Create"** - this will create a new database on Amazon Athena.
4. Then go to **"Crawlers"** on the left menu, and click **"Add Crawler"**.
5. Crawler name: **"student_crawler_<your-name>"**.
6. On the **"Specify crawler source type"** screen click on **"Next"**.
7. On the **"Add a data store"** screen, make sure that the **"Choose a data store"** is set for **"S3"** - as we will crawl data from your specific Amazon S3 bucket.
8. Set the **"Included path"** to the following Amazon S3 path:
"s3://<your_s3_bucket>/athena-exercise/" - this will be used as the base path for the crawler, if we'll add more data under this path, the crawler will know to collect it when running, click on **"Next"**
9. **"Add another data store"** Click on **"Next"**
10. For the **"Choose an IAM role"** page, select **"Choose an existing IAM role"** and choose the **"StudentGlueRole"** and click on **"Next"**.
11. On the **"Create a schedule for this crawler"** screen, under **"Frequency"** set **"Run on demand"** - this means that the crawler will run by demand only and will not be set to run under a defined scheduling.
12. On the **"Configure the crawler's output"** screen, choose the database you've previously created (**make sure to select your own created database**).
13. Add a prefix to all of the created tables: **"glue_<yourname>_athena_exercise"** and click on **"Next"**.
14. Review the crawler configurations, and click on **"Finish"**.
15. At this point you should be able to see your crawler in the crawlers list, select it by clicking in the CheckBox and click **"Run crawler"** (upper section) and monitor the status.
16. After you've run the crawler, wait for it to finish successfully before moving to the next part (expect it to take a few minutes).
17. **Example for a successful output:**

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.

Crawler "student_crawler_student-01" completed and made the following changes: 1 tables created, 0 tables updated. See the tables created in database [student-01](#). x

18. Click on **"Tables"** (left side) and click on the created table, verify the schema extracted from the CSV file.

Part 3 - Querying Data Stored in CSV Format via Amazon Athena

1. Navigate to Amazon Athena (it is advised to open an additional tab).
2. On the left menu, under the **"Database"** dropdown menu, select your database created previously in part 2.
3. Under tables (left menu) you should see one table with the name:
"glue_<yourname>_athena_exercise"
4. Run the SELECT statement below in the Amazon Athena query editor, make sure to copy the query metrics provided by athena: Run time & Data scanned (see screenshots below):

```
select country,
       avg(unit_cost),
       avg(total_revenue),
       sum(total_profit)
from <Athena-Table>
group by 1;
```

The screenshot shows the Amazon Athena query editor interface. The SQL query is as follows:

```
10 select country,
11     avg(unit_cost),
12     avg(total_revenue),
13     sum(total_profit)
14 from glue_student01_athena_exercise
15 group by 1;
```

Below the query, the status bar indicates "SQL Ln 10, Col 17". There are buttons for "Run", "Cancel", "Save as", "Clear", and "Create".

At the bottom, a green status bar shows the execution results:

- ✓ Completed
- Time in queue: 0.304 sec
- Run time: 2.032 sec
- Data scanned: 57.07 MB

An orange arrow points to the "Run time: 2.032 sec" value.

Part 4 - Converting to Parquet Format Using AWS Glue ETL

In order to convert the CSV file into a Parquet data format, we need to launch a AWS Glue Spark/Hive/MR job to transform the file. We can use a ready script, or create one using

AWS Glue - for this exercise we will create a Spark job using Glue. Note that there are two methods to deploy this operation with AWS Glue: “Jobs” which is the latest and “Jobs (legacy)” which is older. The following will hold instructions for the AWS GLue legacy method.

AWS Glue - Jobs (legacy)

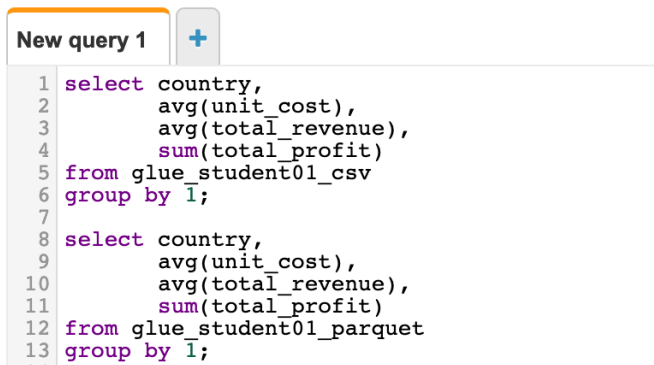
1. Navigate to AWS Glue in the AWS Management Console.
2. Click on “**Jobs (legacy)**” under the “**ETL**” section and Click on “**Add job**”.
3. On the “Configure the job properties” screen select the following:
 - a. **Job Name:** “`csv_to_parquet_<yourname>`”
 - b. **IAM Role:** `StudentGlueRole`
 - c. **Type:** Spark
 - d. **Glue version:** Latest version for Spark and Python
 - e. **This job runs:** A proposed script generated by AWS Glue.
 - f. **Script filename:** `csv_to_parquet_<yourname>` (generated by the job name).
 - g. **S3 path where the script is stored:** No change required
 - h. **Advanced properties:** No changes required
4. Click on “**Next**”
5. **Choose Data source:** choose the previously created glue table (holding your name): For example: `glue_<yourname>_athena`
6. Click on “**Next**”
7. **Choose a transform type:** select “**Change schema**” and click on “**Next**”.
8. **Choose a data target:** select “**Create tables in your data target**”:
 - a. **Data store:** Amazon S3
 - b. **Format:** Parquet
 - c. **Connection:** Leave blank
 - d. **Target Path:** `s3://<your_bucket>/athena-exercise/parquet/`
9. **Output Schema Definition:** As we don’t need to apply transformation or filter one of the columns, we will skip this step and click on “**Save job and edit script**”.
10. This is the AWS Glue editor screen which provides the generated Spark script and the applied transformation, click “**Run Job**” (twice) and wait for the job to successfully complete (it may run for several minutes).
11. If you are seeing failure on the Glue job, it’s because it failed to delete the temp directory, we will fix this by deleting the temporary directory the have been created in “`s3://<youbucket>/athena-exercise/parquet/`” using S3 UI in console.

12. Verify that your Amazon S3 bucket and prefix now holds the converted Parquet data file.

Part 5 - Querying data with Amazon Athena & Parquet Format

In this part, we will use the AWS Glue Crawler to create a new table in Amazon Athena which will be based on the Parquet data file, then we will query the same data with the two different data formats (CSV vs Parquet), comparing the computation metrics.

1. First, we will run the AWS Glue Crawler created previously in Part 2.
2. Navigate to the crawlers page, select your Crawler and click **"Run Crawler"**, monitor the Crawler status.
3. In case of an error click on "Logs" to view the process steps/errors.
4. After the Crawler process is done, you will be able to see two new tables in Amazon athena, this is because AWS Glue will create a table per Amazon S3 prefix/folder.
 - a. `glue_<StudentName>_csv`
 - b. `glue_<StudentName>_parquet`
5. Run The same 2 queries as you previously ran in Part 3 and compare the query "Run time and Data scan" metrics.
6. Which data format is the all time indisputable winner?



The screenshot shows the Amazon Athena console with a 'New query 1' tab. It contains two SQL queries, one for CSV data and one for Parquet data, both grouped by 'I'.

```
1 select country,  
2     avg(unit_cost),  
3     avg(total_revenue),  
4     sum(total_profit)  
5 from glue_student01_csv  
6 group by I;  
7  
8 select country,  
9     avg(unit_cost),  
10    avg(total_revenue),  
11    sum(total_profit)  
12 from glue_student01_parquet  
13 group by I;
```

Part 6 - Delete the Exercise Resources

1. Delete the AWS Glue Crawler.
2. Delete the AWS Glue Job.
3. Drop Athena Database:

```
drop database `<database-name>` cascade;
```

Exercise 8 - AWS Lambda Function

Overview:

In the following exercise, we will create an AWS Lambda function that will copy files between buckets. For each file arriving in the relevant bucket, we will invoke the lambda function to copy it to another bucket.

Part 1 - Create a the Amazon S3 Bucket Exercise Infrastructure

1. In this exercise you will use your existing S3 bucket as the **source** bucket and a newly created bucket as the **target** (section 3) - new files in the source bucket will be copied to the target bucket via AWS Lambda function.
2. Navigate to Amazon S3 service on Amazon Management Console.
3. Click on your student Amazon S3 bucket and add two additional folders:
 - a. source
 - b. target

☐	 source/	Folder	-
☐	 target/	Folder	-

Part 2 - Create a New AWS Lambda Function

Objective:

Creating an AWS Lambda function that will be invoked (triggered) for every file that will be uploaded into the source bucket.

Step A - AWS Lambda Function Creation

1. Navigate to AWS Lambda in the AWS console, click on “**Create function**” (upper right).
2. Choose “**Author from scratch**” as we will upload our own code.
3. **Function Name:** “S3BucketCopyier-<your-name>”
4. **Runtime:** **Python3.9**
5. **Permissions:**
 - a. Click on “Change default execution role”

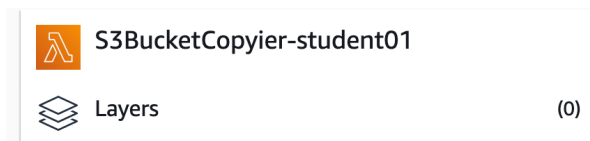
- b. Select “use an existing role”
 - c. Under “Existing role”, select the **StudentLambdaRole** role.
6. Click on “**Create function**” (lower right).

Step B - AWS Lambda Function Trigger Creation

1. Click on “**+ Add trigger**”.
2. **Trigger configuration:** Find and select “S3”.
3. **Bucket:** select your student existing bucket (created on exercise 1) as the source bucket.
4. **Event type:** “All object create events”
5. **Prefix - optional:**
 - a. Specify “source/” as the source folder (prefix or key).
 - b. This will limit the invocation only to files that will appear under this specific prefix.
6. **Suffix - optional:** Leave blank
7. **Recursive invocation:** Check the acknowledge box.
8. Click on “**Add**”

Step C - AWS Lambda Function Code

1. Click on the Lambda function name on the designer tab and click on the “Code” tab:



2. At this point, you should be able to see the AWS Lambda editor.
3. Double click on the `lambda_function.py` file (on the editor left side)
4. Copy the following code of the lambda function into the editor overwrite the existing code:

```
from __future__ import print_function
import boto3
import json
import urllib.request, urllib.parse, urllib.error

print("Loading function")

s3 = boto3.client("s3")

# ToDo: Change the target bucket!
S3_TARGET_BUCKET = "<Student-bucket-name>"
S3_TARGRT_KEY = "target"

def lambda_handler(event, context):
    source_bucket = event['Records'][0]['s3']['bucket']['name']
    key = urllib.parse.unquote_plus(event['Records'][0]['s3']['object']['key'])
    target_bucket = S3_TARGET_BUCKET
```

```
target_key = S3_TARGRT_KEY
copy_source = {'Bucket': source_bucket, 'Key': key}
try:
    print("Waiting for the file persist in the source bucket")
    waiter = s3.get_waiter("object_exists")
    waiter.wait(Bucket=source_bucket, Key=key)
    print("Copying object from source S3 bucket to target S3 bucket.")
    s3.copy_object(Bucket=target_bucket, Key=target_key + '/' +
key.split("/", 1)[1] , CopySource=copy_source)
    return {
        "statusCode": 200,
        'body': json.dumps(
            "Successfully copied file {0} from bucket {1} to bucket
{2}".format(key, source_bucket, target_bucket))
        }
except Exception as e:
    print(e)
    print(
        "Error getting object {0} from bucket {1}.\nmake sure they exists
and that the bucket is in the same region.".format(
            key, source_bucket))
    raise e
```

5. Make sure you are changing the destination bucket before you are saving the code (note the line marked with Yellow in the code),
6. Once completing updating the code and updating the “S3_TARGET_BUCKET” variable, click on “**Deploy**” (above the code).
7. **Set the AWS Lambda function timeout setting by:**
 - a. Click on the “Configuration” tab and select “General configuration” (left side).
 - b. Click in “**Edit**”
 - c. Set (or make sure) the “**Timeout**” to 3 minutes (see example below).
 - d. Click on “**Save**”

Timeout

3	min	0	sec
---	-----	---	-----

Part 3 - Upload Files and Monitor the Function Invocations

1. Download the sample files from this [link](#) (this is a zipped folder containing multiple files).
2. Extract the contents of DummyData.zip locally.
3. Upload the files into your bucket via AWS console under the prefix “source” in order for the lambda function to be invoked.
4. **Monitoring:** Under your created lambda function, click on the “**Monitoring**” tab (upper left).

5. In the “**Monitoring**” Dashboard you should be able to view all the metrics regarding your AWS Lambda function invocations.
6. In addition, click on “View logs in CloudWatch”, and open the files in order to see your function’s logs.
7. As a final step - view your newly created files in your “target” folder in Amazon S3 bucket.

Amazon S3 > naya-college-student-stu1 > target/

target/

Objects | Properties

Objects (9)

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 [more](#)

Find objects by prefix

☐	Name	Type	Last modified
☐	TestFile1.txt	txt	February 27, 2022, 22:49:14 (UTC+02:00)
☐	TestFile2.txt	txt	February 27, 2022, 22:49:14 (UTC+02:00)
☐	TestFile3.txt	txt	February 27, 2022, 22:49:14 (UTC+02:00)

Part 5 - Delete the Exercise Services

1. Delete the AWS Lambda function
2. Delete the Amazon S3 target bucket.

Exercise 9 - Amazon Kinesis

Overview:

In the following exercise, you will create a data pipeline to capture events from log files located in the course VM/Container using Amazon Kinesis Data Streams, then we will use Amazon Data Firehose to store the captured events in an Amazon S3 Bucket. To do so, we'll install the Kinesis agent in the course VM/Container, edit its configuration file and monitor the process using both Linux commands and AWS console monitoring dashboards.

Note

JDK version 11 must be installed on the VM/Docker (or any other used platforms) for this exercise to be performed - please refer to the Oracle JDK official download [page](#).

To verify if the JDK version 11 is installed run:

```
$ java -version
```

```
Openjdk version "11.0.10" 2021-01-19 LTS
```

```
...
```

Optionally, you can download and install JDK 11 using the following commands:

```
$ pip install gdown
```

```
$ gdown
```

```
https://drive.google.com/uc?id=1Sx4s3AWyuXzV5DzYE0MdkKt7vIXd8AZ3
```

```
$ sudo rpm -i jdk-11.0.10_linux-x64_bin.rpm
```

Part 1 - Create an Amazon Kinesis Data Stream Process

1. Navigate to Amazon Kinesis on the AWS console and "**Data streams**" (left side).
2. Click on "**Create data stream**".
 - a. **Data stream name**: student-<yourname>-dataStream
 - b. **Data stream capacity**: Select "Provisioned"
 - c. **Number of open shards**: 1
3. Click on "Create data stream".

Part 2 - Create an Amazon Kinesis Data Firehose Process

1. Click on **"Delivery streams"** (left side).
2. Click on **"Create delivery stream"** (creating a new Firehose process).
3. **Choose source and destination:**
 - a. **Source:**
 - Select **"Amazon Kinesis Data Stream"**
 - b. **Destination:**
 - Select **"Amazon S3"**
4. **Source settings**
 - a. Select your create Data Streams **name** created on part 1 (use Browse).
5. **Delivery stream name**
 - a. **Type:** student-<**name**>-firehose
6. **Transform and convert records - optional**
 - a. Leave without changes.
7. **Destination settings**
 - a. **"S3 bucket"**: Click on "Browse" and select your student Amazon S3 bucket.
 - b. **"S3 bucket prefix"**: Type "Kinesis"
8. Click on "Advanced settings"
 - a. **"Permissions"**: Select "Choose existing IAM role".
 - b. Under "Existing IAM roles" select "Student_Kinesis".
9. Click on **"Create delivery stream"** (scroll down)

Part 3 - Configure the Course VM/Docker with the Kinesis Agent

1. Open an SSH connection to the course VM.
2. Install the AWS Kinesis Agent by copy & running the following command:

```
sudo yum install -y
https://s3.amazonaws.com/streaming-data-agent/aws-kinesis-agent-l
atest.amzn1.noarch.rpm
```

3. Create the Linux source folder and provide the required permissions - every log file that will be copied to this path will be captured by the Kinesis Data Stream process

```
mkdir /tmp/awsKinesisLogs
chmod ugo+rwX /tmp/awsKinesisLogs/
```

4. Edit the AWS Kinesis Agent configuration file (the file can be edited with Linux “vi” or by using MobXterm):

```
sudo vi /etc/aws-kinesis/agent.json
```

- The content of the “agent.json” file should be **overwritten** with the content below (delete all lines and copy the new modified content).
- Note that the following parameters should be modified to support your Kinesis Data Streams pipeline:
 - a. **awsAccessKeyId** - Type your IAM account Access Key (created on exercise 1)
 - b. **awsSecretAccessKey** - Type your IAM account Secret Key (created on exercise 1)
 - c. **kinesisStream** - Specify your Data Streams process name (created on step 1)

```
{
  "cloudwatch.emitMetrics": true,
  "kinesis.endpoint": "",
  "firehose.endpoint": "",

  "awsAccessKeyId": "<>",
  "awsSecretAccessKey": "<>",

  "flows": [
    {
      "filePattern": "/tmp/awsKinesisLogs/*.log",
      "kinesisStream": "<>",
      "partitionKeyOption": "RANDOM"
    }
  ]
}
```

- Note that the “filePattern” parameter will hold the source path and the type of files that the AWS Kinesis agent will track in order to capture new events.

5. Start the AWS Kinesis Agent Linux service:

```
sudo service aws-kinesis-agent start
```

If the service fails to start, there are issues with the access/secret keys, and you need to go back to agent.json file and fix them.

6. Verify the AWS Kinesis Agent Linux service status:

```
sudo service aws-kinesis-agent status
```

```
aws-kinesis-agent.service - LSB: Daemon for Amazon Kinesis Agent.
  Loaded: loaded (/etc/rc.d/init.d/aws-kinesis-agent; bad; vendor preset: disabled)
  Active: active (running) since Wed 2021-02-17 00:00:02 IST; 5s ago
  ...
```

Part 4 - Run and Monitor the Kinesis Data Pipeline

1. Add log files with events based on the Linux operating system main log (messages) to the AWS Kinesis agent source path by running the following command:

```
sudo yum -y install rsyslog
sudo service rsyslog start
sudo tail -100 /var/log/messages > /tmp/awsKinesisLogs/system.log
```

2. View the AWS Kinesis agent log file by running the following command:

```
tail -100 /var/log/aws-kinesis-agent/aws-kinesis-agent.log
```

- Search for output such as:

```
...
Opening /tmp/awsKinesisLogs/system.log for parsing.
...
[INFO] Agent: Progress: 100 records parsed (9666 bytes), and 0
records sent successfully to destinations
...
```

3. Verify the files on Amazon S3:

Amazon S3 > student-01 > 2021/ > 02/ > 16/ > 22/

22/ Copy S3 URI

Objects | Properties

Objects (1)
Objects are the fundamental entities stored in Amazon S3. For others to access your objects, you'll need to explicitly grant them permissions. [Learn more](#)

Refresh Delete Actions Create folder Upload

☐	Name	Type	Last modified	Size	Storage class
☐	stu01-firehose-1-2021-02-16-22-02-55-16b9cf7f-6f6b-443f-98e9-3c0eb772e163	-	February 17, 2021, 00:03:57 (UTC+02:00)	9.4 KB	Standard

4. Monitor the process using Amazon Kinesis monitoring dashboards:
 - a. Navigate to Amazon Kinesis and click on “Data streams” and click on your process.
 - b. Click on “Monitoring”
 - c. Review the available dashboards
 - d. Click on “Data Firehose” and click on your Firehose process.
 - e. Review the available dashboards

Part 5 - Terminate the Exercise Services

1. Delete the Kinesis Data Streams and Data Firehose process.
2. Stop the AWS Kinesis agent service on the Linux OS.

Exercise 10 - Amazon Kinesis Analytics

Overview:

In the following exercise, you will apply an Amazon Kinesis Analytics using SQL to incoming events in real-time. The components will include the following:

- **Source:** Python code that will generate sample data in real-time (will be run from the course VM/Docker or a supported Linux platform).
- **Streaming:** Amazon Kinesis Analytics
- **Analytics:** Python, Flink & SQL

Part 1 - Create an Amazon Kinesis Data Stream Process

1. Navigate to Amazon Kinesis on the AWS console and **"Data streams"** (left side).
2. Click on **"Create data stream"**.
 - a. **Data stream name:** student-<yourname>-DS-Analytics
 - b. **Data stream capacity:** Select "Provisioned"
 - c. **Number of open shards:** 1
3. Click on "Create data stream".

Part 2 - Create an Amazon Kinesis Analytics Application

1. Navigate to Amazon Kinesis on the AWS console and **"Analytics applications"** (left side).
2. Select the **"Studio (New)"** tab.
3. Click on "Create studio notebook":
 - a. **"Choose a method to setup the Studio notebook"**:
 - Select: "Create with custom settings"
 - b. **"General"**:
 - "Studio notebook name": student-notebook-<yourname>
 - Runtime: Make sure to select: "Apache Flink 1.13, Apache Zeppelin 0.9"
 - Click on "Next".
 - c. **"IAM role"**:
 - Check the **"Choose from IAM roles that Kinesis Data Analytics can assume"** option.

- **“Service role”**: Select the **“kinesis-analytics-Tst-us-east-1”** role.
- d. **“AWS Glue database”**
 - **“AWS Glue database”**: Select the **“default”** database.
 - Click on **“Next”**.
- e. On the next page click on **“Next”** as well.
- f. Scroll down and click on: **“Create studio notebook”**.
- 4. Click on **“Run”** to start the Amazon Kinesis Streaming Application (approve the billing message by clicking on **“Run”** again).
- 5. Monitor the status until it will show **“Running”** (this can take several minutes).
- 6. Click on **“Open in Apache Zeppelin”** (a new webpage will be opened).

Apache Zeppelin

A Web-based notebook that enables data-driven, interactive data analytics and collaborative documents with SQL, Scala, Python, R and more.

- a. Click on **“Create new note”**
- b. **“Note Name”**: <yourname>
- c. Click on **“Create”**.

7. Apply the following code in the Zeppelin notebook and Flink framework:

Apache Flink

An open-source, unified stream-processing and batch-processing framework developed by the Apache Software Foundation. The core of Apache Flink is a distributed streaming data-flow engine written in Java and Scala. Flink executes arbitrary dataflow programs in a data-parallel and pipelined manner.

- a. Create the analytics table using the following schema and properties.
- b. The **“events”** table will be used to perform analytics for incoming events in real-time.
- c. Under the **“stream”** parameter, update to your Data Stream name created previously on Part 1 (marked with Yellow)

```
%flink.ssql

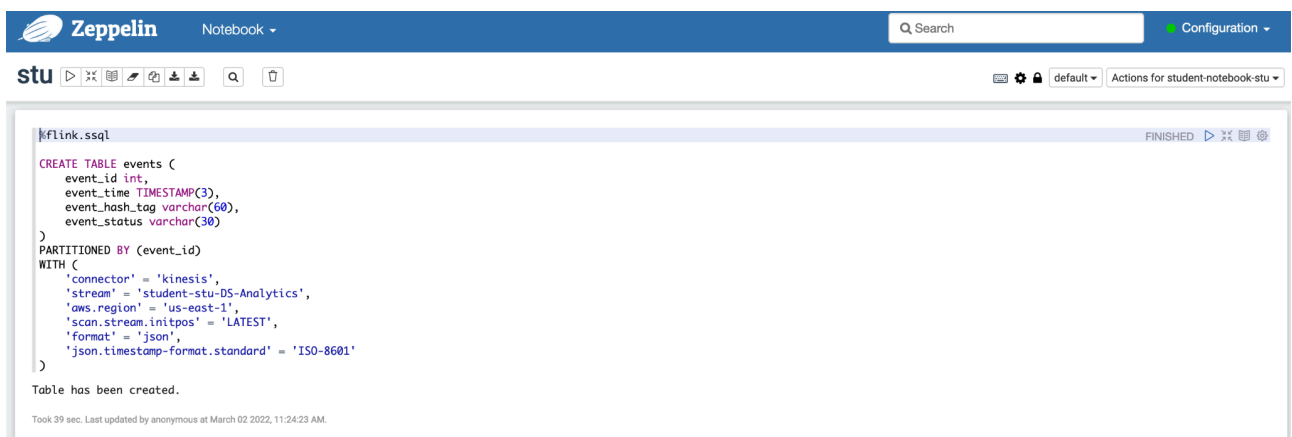
CREATE TABLE events_<student_name> (
  event_id int,
  event_time TIMESTAMP(3),
  event_hash_tag varchar(60),
  event_status varchar(30)
)
```

```

PARTITIONED BY (event_id)
WITH (
  'connector' = 'kinesis',
  'stream' = 'student-<yourname>-DS-Analytics',
  'aws.region' = 'us-east-1',
  'scan.stream.initpos' = 'LATEST',
  'format' = 'json',
  'json.timestamp-format.standard' = 'ISO-8601'
)

```

- d. Click on the play button “Run this paragraph”, wait for the message “Table has been created.”, see screenshot:



Part 3 - Prepare and Run the Random Events Data

1. Open an SSH connection to the course VM/Docker.
2. Add the AWS boto3 library to interact with AWS services and components via Python code:

```
$ pip3.9 install boto3
```

3. Run the following commands to edit a new Python script using Linux VI editor:

```
$ mkdir ~/pythonScripts && cd ~/pythonScripts
$ vi AWSKinesisAnalytics.py
```

4. Update the following code with your Data Stream name (marked with Yellow), then copy the following code into the VI editor:

```
#!/usr/local/bin/python3.9
```

```
import datetime
import json
import random
import boto3

def gen_random_data():
    event_id=random.randrange(999, 10000)
    event_time=datetime.datetime.now().isoformat()
    event_hash_tag=random.getrandbits(64)

event_status=random.choice(["Info","Warning","Error","Critical"])
    return {
        'event_id': event_id,
        'event_time': event_time,
        'event_hash_tag': event_hash_tag,
        'event_status': event_status
    }

def kinesis_send_data(stream_name, kinesis_client):
    while True:
        data=gen_random_data()
        print(data)
        partition_key=str(data["event_id"])
        kinesis_client.put_record(
            StreamName=stream_name,
            Data=json.dumps(data),
            PartitionKey=partition_key)

if __name__ == '__main__':
    client = boto3.client('kinesis')
    kinesis_send_data('student-stu-DS-Analytics',client)
```

5. Make the Python script executable:

```
$ chmod +x AWSKinesisAnalytics.py
```

6. Run the Python script and refer to Part 4:

```
$ ./AWSKinesisAnalytics.py
```

Part 4 - Run SQL Statements on Real-Time Events

1. Return to the Zeppelin Note and on a new paragraph run the following SQL statement:

```
%flink.ssql (type=update)
select * from events_<student_name>;
```

%flink.ssql(type=update)
select * from events;

FLINK JOB ABORT

event_id	event_time	event_hash_tag	event_status
1973	2022-03-02T10:46:27.777624	6576903055388117425	Info
1908	2022-03-02T10:46:27.623924	12049683681729420158	Critical
1193	2022-03-02T10:46:27.473508	11192212790181633751	Info
1108	2022-03-02T10:46:27.214092	1723788656198617895	Critical
1243	2022-03-02T10:46:26.732412	5002541672619805785	Warning
2116	2022-03-02T10:46:26.658901	2512435531199190401	Critical
1330	2022-03-02T10:46:26.585678	18334904445651548318	Critical
2064	2022-03-02T10:46:26.285868	11072105206201889678	Error

2. Run the following SQL statement as well:

```
%flink.ssql(type=update)
select event_status, count(*) from events_<student_name> group by
event_status;
```

%flink.ssql(type=update)
select event_status, count(*) from events group by event_status;

FLINK JOB RUNNING 0%

event_status	EXPR\$1
Critical	632
Error	590
Info	610
Warning	600

Part 5 - Terminate the Amazon Kinesis Analytics Processing

1. Make sure to stop the Python script running on the course VM/Docker.
2. Under "Streaming applications" > "Studio", Check your "Studio Notebook", Click on "Actions" and "Delete".
3. Go to the "Data streams" and delete your data stream process.
4. Stop the Python script by ctrl+z.

Solutions - Amazon Athena

Tables & Views

```
create database nayadb;

drop table nayadb.covid19_data;

CREATE EXTERNAL TABLE IF NOT EXISTS nayadb.covid19_data (
  `cdc_case_earliest_dt` date,
  `cdc_report_dt` date,
  `pos_spec_dt` date,
  `onset_dt` date,
  `current_status` string,
  `gender` string,
  `age_group` string,
  `race_ethnicity_combined` string,
  `hosp_yn` string,
  `icu_yn` string,
  `death_yn` string,
  `medcond_yn` string
)
ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.lazy.LazySimpleSerDe'
WITH SERDEPROPERTIES (
  'serialization.format' = ',',
  'field.delim' = ','
) LOCATION 's3://nc-de-examples/athena/'
TBLPROPERTIES ('has_encrypted_data'='false');

show create table nayadb.covid19_data;

select * from nayadb.covid19_data limit 10;

create view nayadb.vw_view1 as
select cdc_case_earliest_dt, gender, count(*) as count
```

```
from covid19_data
where gender in ('Male', 'Female')
group by cdc_case_earliest_dt, gender
order by 1 desc
limit 100;

select * from nayadb.vw_view1;

create view nayadb.vw_view2 as
select gender, count(*) as count
from covid19_data
where current_status = 'Laboratory-confirmed case'
group by gender
limit 100;

select * from nayadb.vw_view2;

create view nayadb.vw_view3 as
select age_group, count(*) as count
from covid19_data
group by age_group
order by 2 desc;

select * from nayadb.vw_view3;

--drop database nayadb cascade;
```

Partitioned Table

```
drop table nayadb.covid19_data_part;

CREATE EXTERNAL TABLE IF NOT EXISTS nayadb.covid19_data_part (
  `cdc_case_earliest_dt` date,
  `cdc_report_dt` date,
  `pos_spec_dt` date,
  `onset_dt` date,
  `current_status` string,
  `gender` string,
  `age_group` string,
  `race_ethnicity_combined` string,
  `hosp_yn` string,
  `icu_yn` string,
  `death_yn` string,
  `medcond_yn` string
)
PARTITIONED BY (case_year int, case_month int)
STORED AS parquet
LOCATION 's3://nc-de-examples/athena_partitions/';

insert into nayadb.covid19_data_part
select cdc_case_earliest_dt,
       cdc_report_dt,
       pos_spec_dt,
       onset_dt,
       current_status,
       gender,
       age_group,
       race_ethnicity_combined,
       hosp_yn,
       icu_yn,
       death_yn,
       medcond_yn,
```

```
        cast(year(cdc_case_earliest_dt) as integer),  
        cast(month(cdc_case_earliest_dt) as integer)  
from nayadb.covid19_data;
```

```
MSCK REPAIR TABLE nayadb.covid19_data_part;
```

```
SHOW PARTITIONS nayadb.covid19_data_part;
```

```
explain  
select cdc_case_earliest_dt,  
       case_year,  
       case_month  
from nayadb.covid19_data_part  
limit 100;
```