NYC Taxi and Limousine Data Analysis Lab

# Overview

In this lab, you will learn how to import data from CSV text files into Cloud SQL and then carry out some basic data analysis using simple queries.  
  
The dataset used in this lab is collected by the NYC Taxi and Limousine Commission and includes trip records from all trips completed in Yellow and Green taxis in NYC from 2009 to present, and all trips in for-hire vehicles (FHV) from 2015 to present. Records include fields capturing pick-up and drop-off dates/times, pick-up and drop-off locations, trip distances, itemized fares, rate types, payment types, and driver-reported passenger counts.  
  
This dataset can be used to demonstrate a wide range of data science concepts and techniques and will be used in several of the labs in the Data Engineering curriculum.

# Objectives

1. Create Cloud SQL instance  
   2. Create a Cloud SQL database  
   3. Import text data into Cloud SQL  
   4. Check the data for integrity

# Step 1: Create a New Project in Google Cloud

1. **Log in to Google Cloud Console**:
   * Go to the Google Cloud Console and log in using your Google account.
2. **Create a New Project**:
   * In the Google Cloud Console, click the **project drop-down** at the top of the page next to "Google Cloud Platform."
   * Select **New Project**.
   * Enter the **Project Name** (e.g., "Taxi Data Analysis").
   * Optionally, select a **billing account** if it's not set up yet.
   * Click **Create**.
3. **Wait for the project to be created**.
   * Once created, select the new project from the drop-down to set it as the active project.

# Step 2: Activate Google Cloud Shell

1. **Open Cloud Shell**:
   * In the top-right corner of the Google Cloud Console, click on the **Cloud Shell icon** (it looks like a terminal).
   * Click **Continue** to start the shell, which provides you with a command-line interface.

# Step 3: Preparing your environment

Create environment variables that will be used later in the lab for your project ID and the storage bucket that will contain your data:  
  
export PROJECT\_ID=$(gcloud info --format='value(config.project)')  
This command stores the **Project ID** of your current Google Cloud project into the variable PROJECT\_ID for later use.

export BUCKET=${PROJECT\_ID}-ml

This command creates an environment variable BUCKET that stores the name of the Google Cloud Storage bucket that you'll use later in the lab.

# Step4: Create a Cloud SQL instance

Enter the following commands to create a Cloud SQL instance:  
  
gcloud sql instances create taxi --tier=db-n1-standard-1 --activation-policy=ALWAYS

Output :

student\_04\_fa2ff1b080dc@cloudshell:~ (qwiklabs-gcp-02-9b9a8ae56646)$ gcloud sql instances create taxi \

--tier=db-n1-standard-1 --activation-policy=ALWAYS

WARNING: Starting with release 233.0.0, you will need to specify either a region or a zone to create an instance.

Creating Cloud SQL instance for MYSQL\_8\_0...working.

Creating Cloud SQL instance for MYSQL\_8\_0...done.

Set a root password:  
  
gcloud sql users set-password root --host % --instance taxi --password Passw0rd

Output

student\_04\_fa2ff1b080dc@cloudshell:~ (qwiklabs-gcp-02-9b9a8ae56646)$ gcloud sql users set-password root --host % --instance taxi \

--password Passw0rd

Updating Cloud SQL user...done.

When prompted for the password type Passw0rd and press enter this will update root password.

**Note:** The inputted password Passw0rd will be masked and not visible in the cloud terminal.

Now create an environment variable with the IP address of the Cloud Shell:

export ADDRESS=$(wget -qO - http://ipecho.net/plain)/32

Whitelist the Cloud Shell instance for management access to your SQL instance:

gcloud sql instances patch taxi --authorized-networks $ADDRESS

When prompted press **Y** to accept the change.

The following message will be used for the patch API method.

Output

{"name": "taxi", "project": "qwiklabs-gcp-02-9b9a8ae56646", "settings": {"ipConfiguration": {"authorizedNetworks": [{"value": "34.105.103.82/32"}]}}}

Patching Cloud SQL instance...done.

Get the IP address of your Cloud SQL instance by running:

MYSQLIP=$(gcloud sql instances describe \

taxi --format="value(ipAddresses.ipAddress)")

Check the variable MYSQLIP:

echo $MYSQLIP

Output :

34.170.118.226

Create the taxi trips table by logging into the mysql command line interface:

mysql --host=$MYSQLIP --user=root \

When prompted for a password enter Passw0rd.

Paste the following content into the command line to create the schema for the trips table:

create database if not exists bts;

use bts;

drop table if exists trips;

-------create table trips (

vendor\_id VARCHAR(16),

pickup\_datetime DATETIME,

dropoff\_datetime DATETIME,

passenger\_count INT,

trip\_distance FLOAT,

rate\_code VARCHAR(16),

store\_and\_fwd\_flag VARCHAR(16),

payment\_type VARCHAR(16),

fare\_amount FLOAT,

extra FLOAT,

mta\_tax FLOAT,

tip\_amount FLOAT,

tolls\_amount FLOAT,

imp\_surcharge FLOAT,

total\_amount FLOAT,

pickup\_location\_id VARCHAR(16),

dropoff\_location\_id VARCHAR(16)

);

Describe trips

mysql> describe trips;

--------------

describe trips

--------------

+---------------------+-------------+------+-----+---------+-------+

| Field | Type | Null | Key | Default | Extra |

+---------------------+-------------+------+-----+---------+-------+

| vendor\_id | varchar(16) | YES | | NULL | |

| pickup\_datetime | datetime | YES | | NULL | |

| dropoff\_datetime | datetime | YES | | NULL | |

| passenger\_count | int | YES | | NULL | |

| trip\_distance | float | YES | | NULL | |

| rate\_code | varchar(16) | YES | | NULL | |

| store\_and\_fwd\_flag | varchar(16) | YES | | NULL | |

| payment\_type | varchar(16) | YES | | NULL | |

| fare\_amount | float | YES | | NULL | |

| extra | float | YES | | NULL | |

| mta\_tax | float | YES | | NULL | |

| tip\_amount | float | YES | | NULL | |

| tolls\_amount | float | YES | | NULL | |

| imp\_surcharge | float | YES | | NULL | |

| total\_amount | float | YES | | NULL | |

| pickup\_location\_id | varchar(16) | YES | | NULL | |

| dropoff\_location\_id | varchar(16) | YES | | NULL | |

+---------------------+-------------+------+-----+---------+-------+

17 rows in set (0.04 sec)

# Step 5: Add data to Cloud SQL instance

Now you'll copy the New York City taxi trips CSV files stored on Cloud Storage locally. To keep resource usage low, you'll only be working with a subset of the data (~20,000 rows).

Copy NYC taxi trips CSV files from Cloud Storage locally:  
  
gcloud storage cp gs://cloud-training/OCBL013/nyc\_tlc\_yellow\_trips\_2018\_subset\_1.csv trips.csv-1  
gcloud storage cp gs://cloud-training/OCBL013/nyc\_tlc\_yellow\_trips\_2018\_subset\_2.csv trips.csv-2  
  
Load the CSV data into the Cloud SQL instance:

gcloud storage cp gs://cloud-training/OCBL013/nyc\_tlc\_yellow\_trips\_2018\_subset\_1.csv trips.csv-1

Output

gcloud storage cp gs://cloud-training/OCBL013/nyc\_tlc\_yellow\_trips\_2018\_subset\_2.csv trips.csv-2

Copying gs://cloud-training/OCBL013/nyc\_tlc\_yellow\_trips\_2018\_subset\_1.csv to <file://trips.csv-1>

Completed files 1/1 | 850.3kiB/850.3kiB Average throughput: 147.6MiB/s

Copying gs://cloud-training/OCBL013/nyc\_tlc\_yellow\_trips\_2018\_subset\_2.csv to file://trips.csv-2

Completed files 1/1 | 849.8kiB/849.8kiB Average throughput: 157.6MiB/s

Connect to the mysql interactive console to load local infile data:

mysql --host=$MYSQLIP --user=root --password --local-infile

When prompted for a password enter Passw0rd.

In the mysql interactive console select the database:

use bts;

Load the local CSV file data using local-infile:

mysql> LOAD DATA LOCAL INFILE 'trips.csv-1' INTO TABLE trips

-> FIELDS TERMINATED BY ','

-> LINES TERMINATED BY '\n'

-> IGNORE 1 LINES

->

Output

(vendor\_id,pickup\_datetime,dropoff\_datetime,passenger\_count,trip\_distance,rate\_code,store\_and\_fwd\_flag,payment\_type,fare\_amount,extra,mta\_tax,tip\_amount,tolls\_amount,imp\_surcharge,total\_amount,pickup\_location\_id,dropoff\_location\_id);

Query OK, 10018 rows affected (0.53 sec)

Records: 10018 Deleted: 0 Skipped: 0 Warnings: 0

mysql> LOAD DATA LOCAL INFILE 'trips.csv-2' INTO TABLE trips

-> FIELDS TERMINATED BY ','

-> LINES TERMINATED BY '\n'

-> IGNORE 1 LINES

->

Output

(vendor\_id,pickup\_datetime,dropoff\_datetime,passenger\_count,trip\_distance,rate\_code,store\_and\_fwd\_flag,payment\_type,fare\_amount,extra,mta\_tax,tip\_amount,tolls\_amount,imp\_surcharge,total\_amount,pickup\_location\_id,dropoff\_location\_id);

Query OK, 10006 rows affected (0.31 sec)

Records: 10006 Deleted: 0 Skipped: 0 Warnings: 0

LOAD DATA LOCAL INFILE 'trips.csv-2' INTO TABLE trips

FIELDS TERMINATED BY ','

LINES TERMINATED BY '\n'

IGNORE 1 LINES

(vendor\_id,pickup\_datetime,dropoff\_datetime,passenger\_count,trip\_dist

# Step 6:. Checking for data integrity

Check for integrity after loading the data. Ensure trip\_distance and fare\_amount values are reasonable, and check the payment types for consistency.  
  
Query the trips table for unique pickup location regions:

mysql> select distinct(pickup\_location\_id) from trips;

+--------------------+

| pickup\_location\_id |

+--------------------+

| 68 |

| 138 |

| 261 |

| 262 |

| 100 |

| 7 |

| 132 |

| 264 |

| 170 |

| 237 |

| 87 |

| 161 |

| 229 |

| 186 |

| 230 |

| 224 |

| 234 |

| 233 |

| 125 |

| 79 |

| 162 |

| 223 |

| 236 |

| 137 |

| 225 |

| 50 |

| 76 |

| 13 |

| 65 |

| 107 |

| 231 |

| 211 |

| 48 |

| 164 |

| 90 |

| 24 |

| 88 |

| 143 |

| 246 |

| 142 |

| 215 |

| 265 |

| 238 |

| 43 |

| 239 |

| 141 |

| 148 |

| 166 |

| 163 |

| 249 |

| 74 |

| 158 |

| 263 |

| 144 |

| 42 |

| 151 |

| 114 |

| 70 |

| 75 |

| 33 |

| 145 |

| 140 |

| 232 |

| 113 |

| 146 |

| 45 |

| 4 |

| 181 |

| 152 |

| 209 |

| 260 |

| 179 |

| 93 |

| 244 |

| 256 |

| 226 |

| 112 |

| 243 |

| 255 |

| 40 |

| 41 |

| 49 |

| 189 |

| 119 |

| 66 |

| 12 |

| 95 |

| 25 |

| 37 |

| 52 |

| 116 |

| 17 |

| 80 |

| 219 |

| 130 |

| 127 |

| 194 |

| 10 |

| 31 |

| 197 |

| 191 |

| 171 |

| 217 |

| 129 |

| 15 |

| 157 |

| 97 |

| 28 |

| 193 |

| 168 |

| 29 |

| 216 |

| 54 |

| 139 |

| 206 |

| 167 |

| 16 |

| 235 |

| 102 |

| 133 |

| 185 |

| 51 |

| 8 |

| 198 |

| 77 |

| 69 |

| 61 |

| 126 |

| 247 |

| 89 |

| 254 |

| 106 |

| 228 |

| 169 |

| 258 |

| 173 |

| 153 |

| 196 |

| 60 |

| 147 |

| 208 |

| 53 |

| 35 |

| 82 |

| 184 |

| 47 |

| 218 |

| 203 |

| 134 |

| 178 |

| 92 |

| 123 |

| 36 |

| 159 |

| 56 |

| 83 |

| 200 |

| 259 |

| 188 |

+--------------------+

159 rows in set (0.08 sec)

Let's start by digging into the trip\_distance column. Enter the following query into the console:

mysql> select

-> max(trip\_distance),

-> min(trip\_distance)

-> from

-> trips;

+--------------------+--------------------+

| max(trip\_distance) | min(trip\_distance) |

+--------------------+--------------------+

| 85 | 0 |

+--------------------+--------------------+

1 row in set (0.05 sec)

mysql> select count(\*) from trips where trip\_distance = 0;

+----------+

| count(\*) |

+----------+

| 155 |

+----------+

1 row in set (0.05 sec)

mysql> select count(\*) from trips where fare\_amount < 0;

+----------+

| count(\*) |

+----------+

| 14 |

+----------+

1 row in set (0.05 sec)

There should be 14 such trips returned. Again, these trips warrant further exploration. There may be a reasonable explanation for why the fares take on negative numbers. However, it's up to the data engineer to ensure there are no bugs in the data pipeline that would cause such a result.

1. Finally, let's investigate the payment\_type column.

mysql> select

-> payment\_type,

-> count(\*)

-> from

-> trips

-> group by

-> payment\_type;

+--------------+----------+

| payment\_type | count(\*) |

+--------------+----------+

| 1 | 13863 |

| 2 | 6016 |

| 3 | 113 |

| 4 | 32 |

+--------------+----------