towards data science

Follow 581K Followers

SQL on The Cloud With Python

A straightforward guide to SQL on Google Cloud and Python

James Briggs Sep 4, 2020 · 5 min read ★

Photo by Tim Bogdanov on Unsplash

Everyone and their dog uses the cloud nowadays — and with good reason. It takes what used to be a tricky task and makes it as easy as falling off a log.

imagine you need to worry about security threats that evolve daily, world-wide latency, infrastructure scaling, service outages, and much more.

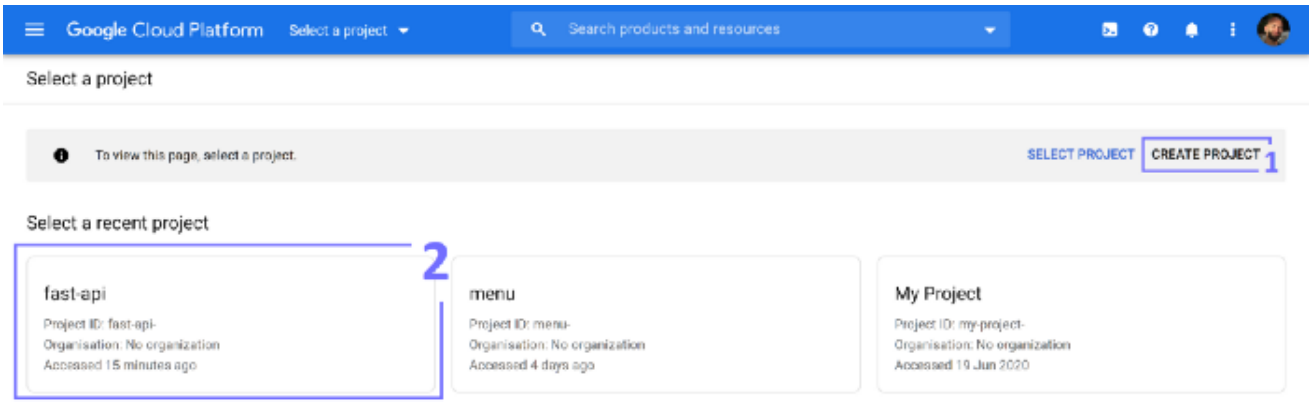
But now, we can have databases, web-apps, IoT hubs, and much more built quicker than I can make a coffee — all thanks to the cloud.

These new cloud services come with the ability to be hosted almost anywhere in the world, with top-tier security, automatic scaling, and guarantees of 99.999% availability. The cloud has transformed tech.

We will cover the steps we need to take to set up a MySQL database with Google Cloud Platform (GCP) and looking at how we can query that database with Python.

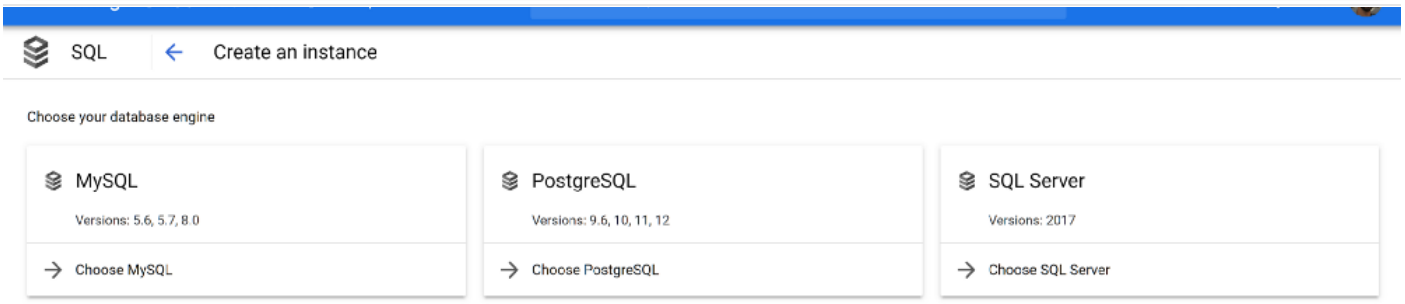
Create a Cloud SQL Instance

The first thing we need to do is create our database. For that, we head over to the Cloud SQL Instances page.



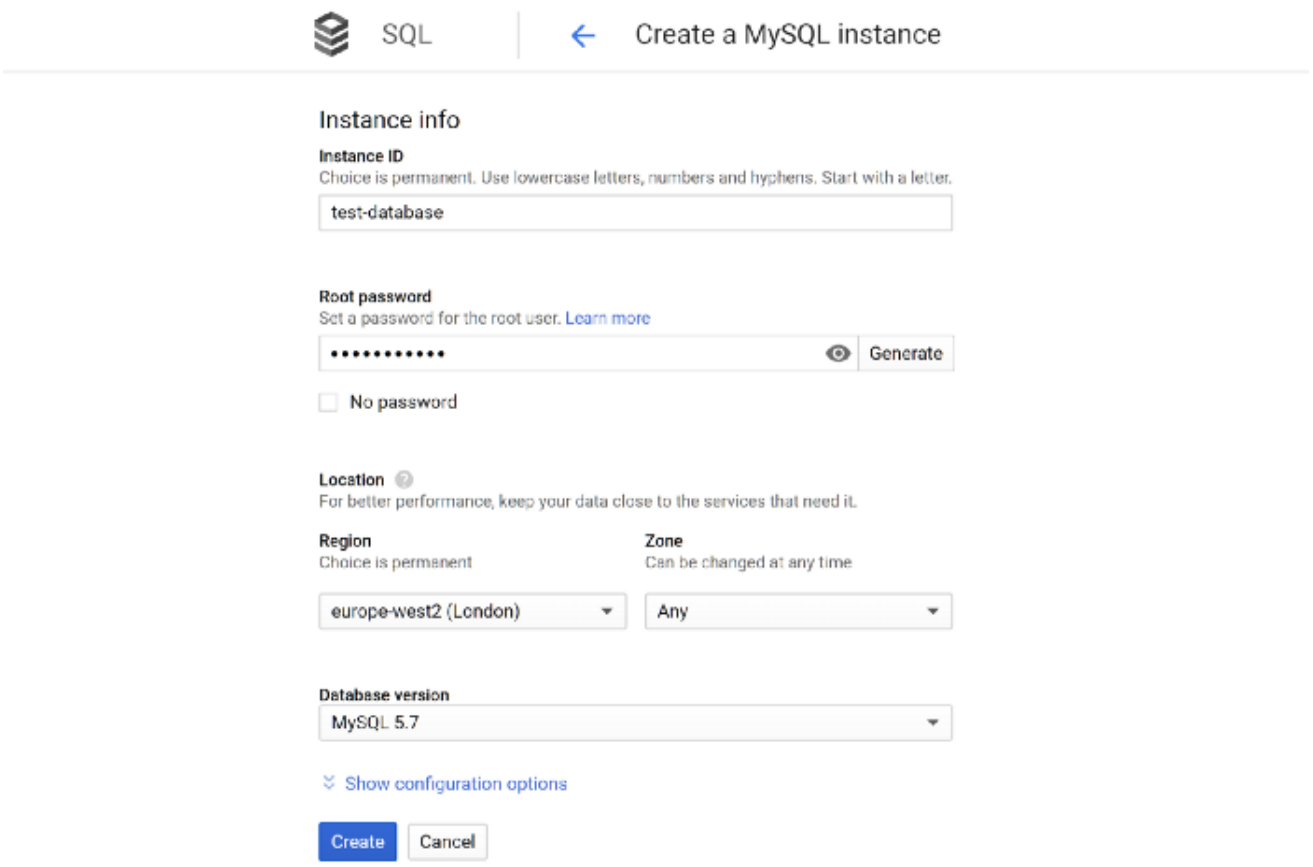
We can either (1) create a new project or (2) use an existing project.

Here, we have two options — use an existing project or create a new one. If you don't already have a project setup, go ahead and create one with **Create Project** now.



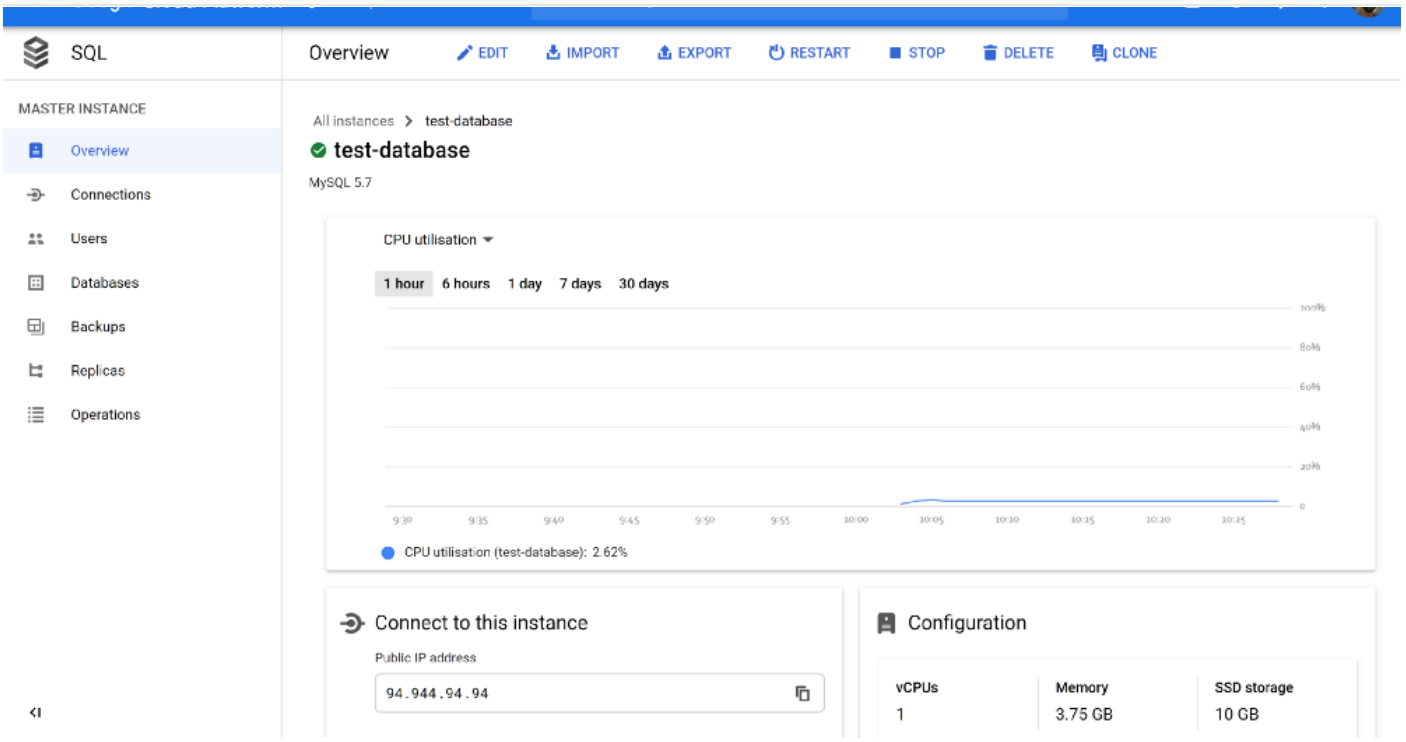
We have the option to choose between different SQL versions.

On the next page, we will click **Create Instance,** which takes us to a choice of MySQL, PostgreSQL, and SQL Server — we will be using **MySQL**. Feel free to use another option if you prefer.



Creation of a MySQL Instance.

Next, we set up our instance info; this is pretty straightforward — choose your or your customer's closest region, and remember your password (we need it when connecting with Python later).



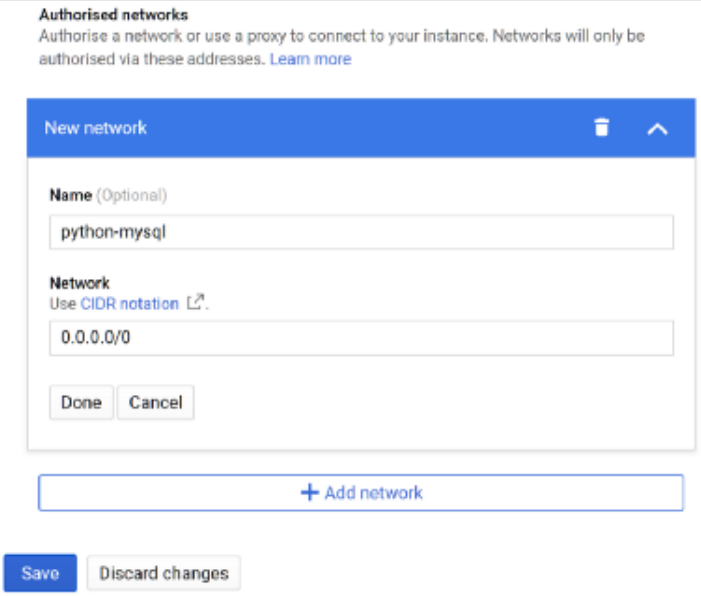
The Instance Details area.

Now we should be taken through to our database overview. Instance creation will usually take a few minutes.

Connection Setup

Enabling IP Access

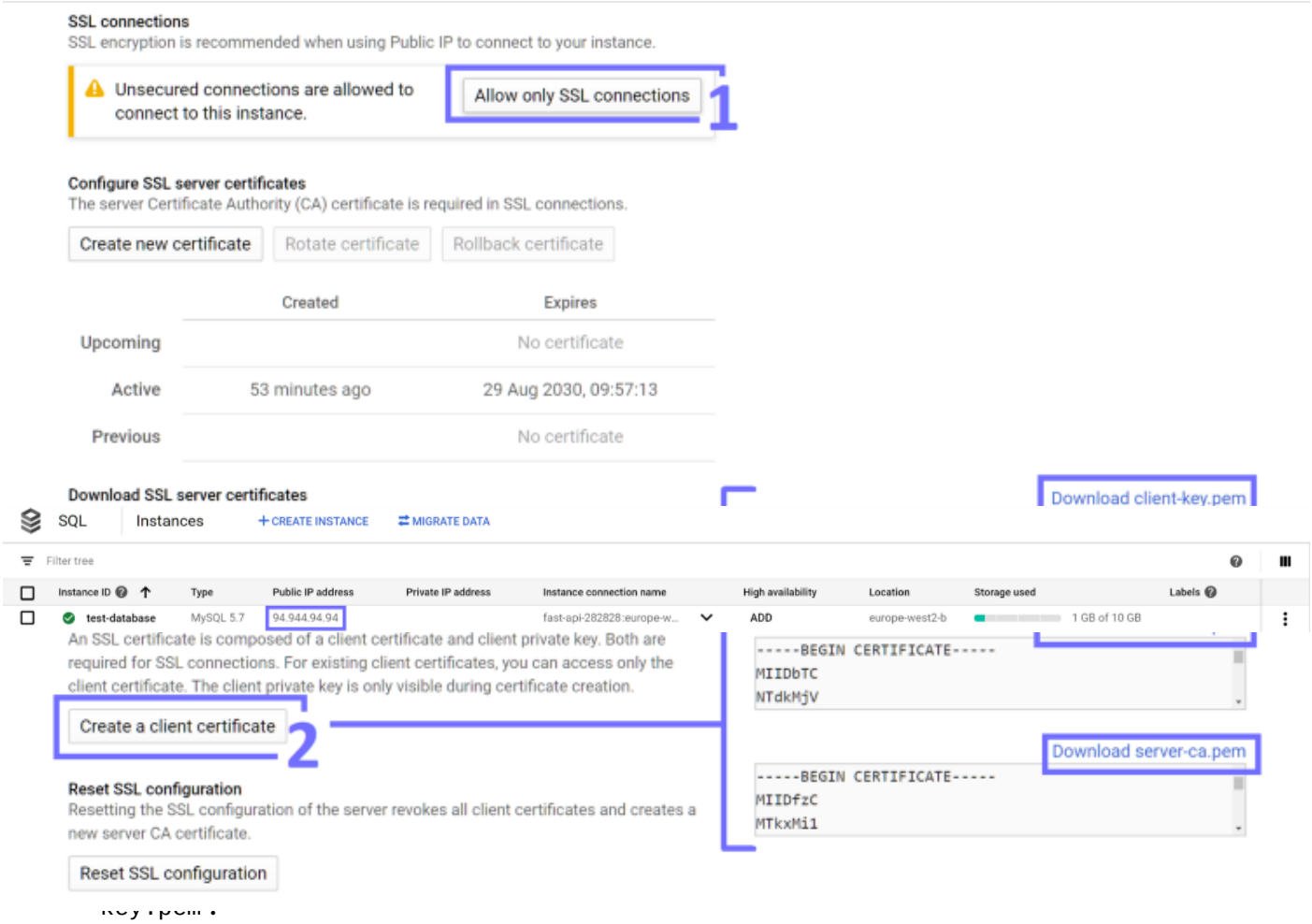
Head back to our instance details page (you can access it by clicking on the Instance ID here).



IP authorization settings in the Connections tab of the Instance Details area.

We click on the **Connections** tab (on the left-taskbar) and click **+ Add network** under Public IP. Give the network a name and enter the IP address range you would like to allow — `0.0.0.0/0` allows entry to all IP addresses.

Of course, this is a pretty big security issue, which is why we use SSL encryption to restrict access to our instance.



SSL setup in the Connections tab of the Instance Details area.

All of these are used in our `config` variable, which we use to establish our connection like so:

While still on the Connections tab, we scroll down to the **SSL** section. Click **Allow only SSL connections** and then **Create a client certificate** — download all three `.pem` files

```
import mysql.connector
from mysql.connector.constants import ClientFlag

config = {
    'user': 'root',
    'password': 'Password123',
    'host': '94.944.94.94',
    'client_flags': [ClientFlag.SSL],
    'ssl_ca': 'ssl/server-ca.pem',
    'ssl_cert': 'ssl/client-cert.pem',
    'ssl_key': 'ssl/client-key.pem'
}

# now we establish our connection
cnxn = mysql.connector.connect(**config)
```

gcp_mysql_connection.py hosted with ❤ by **GitHub** **view raw**

Querying the Cloud

Now we've set everything up, let's upload some data and run some quick queries.

Create a Database

First, we need a database to hold our data — we use our previously setup connection `cnxn`:

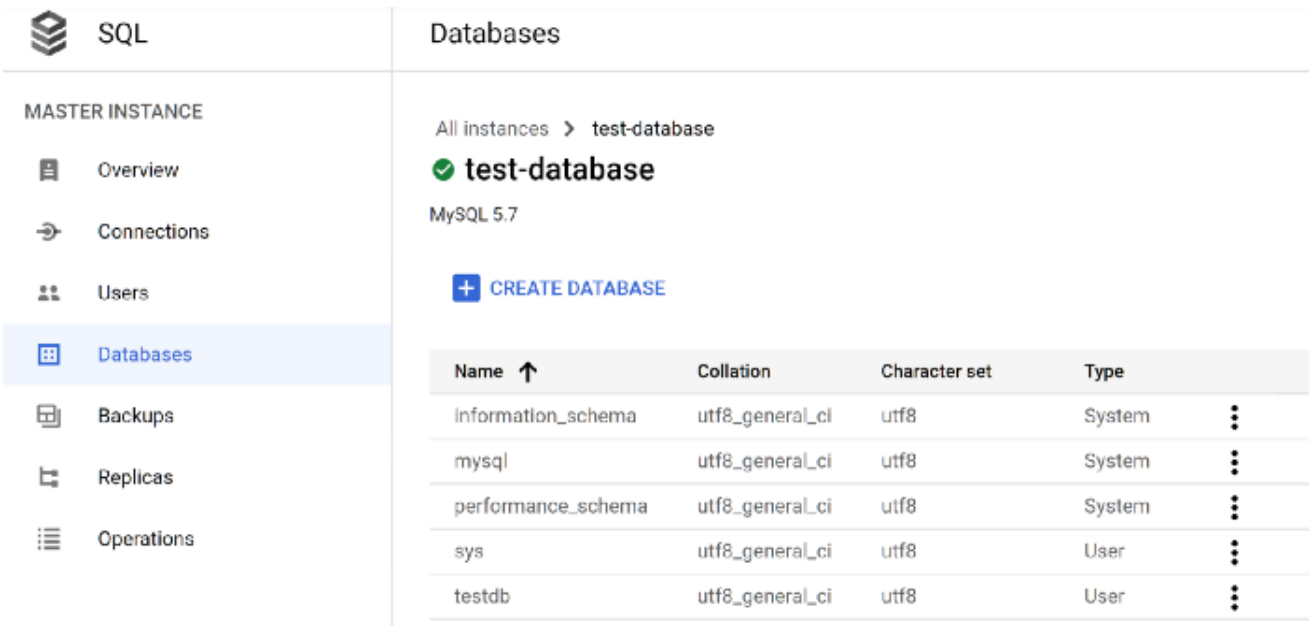
```
cursor = cnxn.cursor()  # initialize connection cursor
cursor.execute('CREATE DATABASE testdb')  # create a new 'testdb' database
cnxn.close()  # close connection because we will be reconnecting to testdb
```

create_db.py hosted with ❤ by **GitHub** **view raw**

Now we connect to `testdb` by adding `database: testdb` to our `config` dictionary and connecting just like we did before:

```
config['database'] = 'testdb'  # add new database to config dict
cnxn = mysql.connector.connect(**config)
cursor = cnxn.cursor()
```

connect_db.py hosted with ❤ by **GitHub** **view raw**



Databases tab in the Instance Details area.

Databases tab. We can also create new databases from here by clicking **Create Database**.

Uploading Data

We will use the All Space Missions from 1957 dataset on Kaggle. You can download a cleaned CSV of the data from here.

	Company Name	Location	Datum	Detail	Status Rocket	Rocket	Status Mission
0	SpaceX	LC-39A, Kennedy Space Center, Florida, USA	Fri Aug 07, 2020 05:12 UTC	Falcon 9 Block 5 | Starlink V1 L9 & BlackSky	StatusActive	50.0	Success
1	CASC	Site 9401 (SLS-2), Jiuquan Satellite Launch Ce...	Thu Aug 06, 2020 04:01 UTC	Long March 2D | Gaofen-9 04 & Q-SAT	StatusActive	29.75	Success
2	SpaceX	Pad A, Boca Chica, Texas, USA	Tue Aug 04, 2020 23:57 UTC	Starship Prototype | 150 Meter Hop	StatusActive	NaN	Success
3	Roscosmos	Site 200/39, Baikonur Cosmodrome, Kazakhstan	Thu Jul 30, 2020 21:25 UTC	Proton-M/Briz-M | Ekspress-80 & Ekspress-103	StatusActive	65.0	Success
4	ULA	SLC-41, Cape Canaveral AFS, Florida, USA	Thu Jul 30, 2020 11:50 UTC	Atlas V 541 | Perseverance	StatusActive	145.0	Success

Top five rows of the All Space Missions from 1957 data. Showing the SpaceX Starship hop test and Perseverance rover launch.

We create a new table called `space_missions` with the correct column headers using `cursor.execute`:

```
cursor.execute("CREATE TABLE space_missions ("
               "company_name VARCHAR(255),"
               "location VARCHAR(255),"
               "datum DATETIME,"
               "detail VARCHAR(255),"
               "status_rocket VARCHAR(255),"
               "rocket FLOAT(6,2),"
               "status_mission VARCHAR(255) )")

cnxn.commit()  # this commits changes to the database
```

create_table.py hosted with ❤ by **GitHub** **view raw**

Next, we insert our data:

```
query = ("INSERT INTO space_missions (company_name, location, datum, detail, status_r
         "VALUES (%s, %s, %s, %s, %s, %s, %s)")

# then we execute with every row in our dataframe
cursor.executemany(query, list(data.to_records(index=False)))
```

```
cursor.execute("SELECT * FROM space_missions LIMIT 5")
out = cursor.fetchall()
for row in out:
    print(row)
```

```
('SpaceX', 'LC-39A, Kennedy Space Center, Florida, USA', 'Fri Aug 07, 2020 05:12 UTC', 'Falcon 9 Block 5 | Starlink V1 L9 & BlackSky', 'StatusActive', '50.0', 'Success')
('CASC', 'Site 9401 (SLS-2), Jiuquan Satellite Launch Center, China', 'Thu Aug 06, 2020 04:01 UTC', 'Long March 2D | Gaofen-9 04 & Q-SAT', 'StatusActive', '29.75', 'Success')
('SpaceX', 'Pad A, Boca Chica, Texas, USA', 'Tue Aug 04, 2020 23:57 UTC', 'Starship Prototype | 150 Meter Hop', 'StatusActive', 'nan', 'Success')
('Roscosmos', 'Site 200/39, Baikonur Cosmodrome, Kazakhstan', 'Thu Jul 30, 2020 21:25 UTC', 'Proton-M/Briz-M | Ekspress-80 & Ekspress-103', 'StatusActive', '65.0', 'Success')
('ULA', 'SLC-41, Cape Canaveral AFS, Florida, USA', 'Thu Jul 30, 2020 11:50 UTC', 'Atlas V 541 | Perseverance', 'StatusActive', '145.0', 'Success')
```

Top five rows in our Cloud MySQL database. Again showing the SpaceX Starship hop test and Perseverance rover launch.

Now our database should contain the full `space_missions` table. We can test this by querying for the top five rows of our table, which should match the top five we saw earlier:

The output is the same as what we saw earlier, which means we now have our data successfully stored in the cloud.

Any further queries or transformations can be performed as we usually do for MySQL.

That's It!

Now we have a fully functional MySQL database with SSL encryption. Connecting to our database from anywhere in the world requires no more than four lines of Python code.

From then, we can query our database as we would any other — it's incredibly easy to pick up and use.

Let me know if you have any questions, feedback, or ideas over on Twitter or in the comments below. Thanks for reading!

Interested in learning more about Google Cloud? You might like my previous article covering API deployment with Docker and GCP:

Deploy APIs With Python and Docker

A definitive guide to deploying APIs with Flask, Docker and Google Cloud