Path on github

**~/training-data-analyst/courses/bdml\_fundamentals/demos/earthquakevm**$

Install libraries

./install\_missing.sh

To create image/vizz

./transform.py

To list items in bucket

gsutil ls gs://<bucket name>

to copy files to bucket

gsutil cp earthquakes.\* gs://<bucketname>

to install wget:

sudo apt-get install wget

install python

sudo apt install python3 python3-dev python3-venv

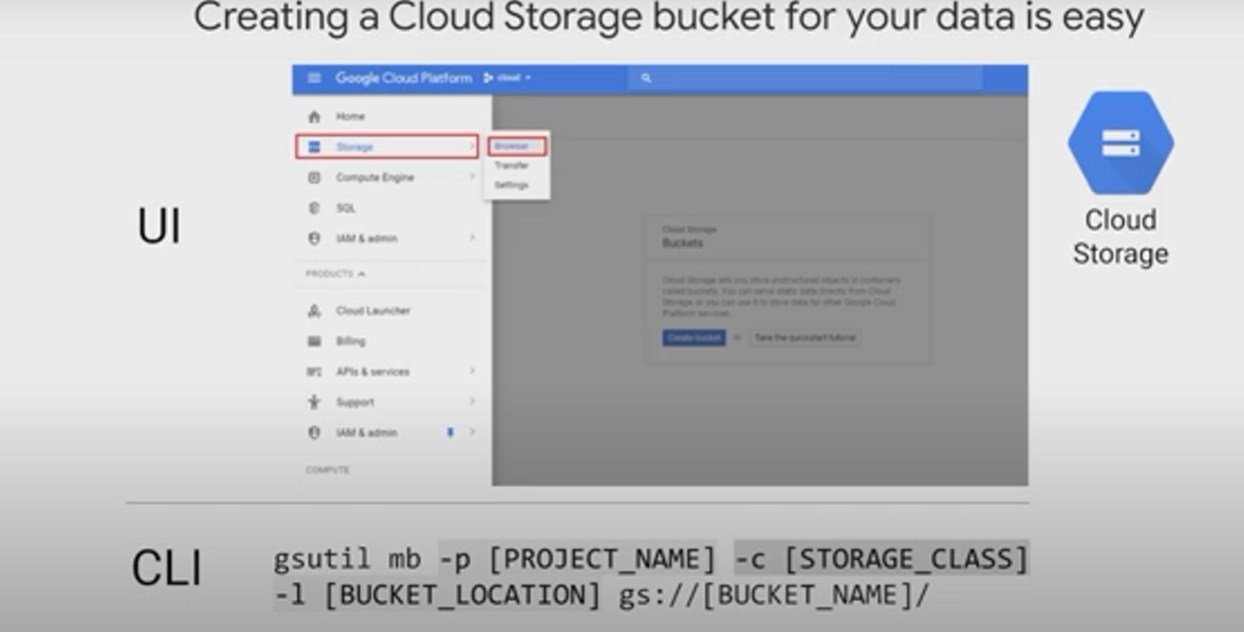
[sales\_solutions\_analytics\_coe.eim@philips.com](mailto:sales_solutions_analytics_coe.eim@philips.com)

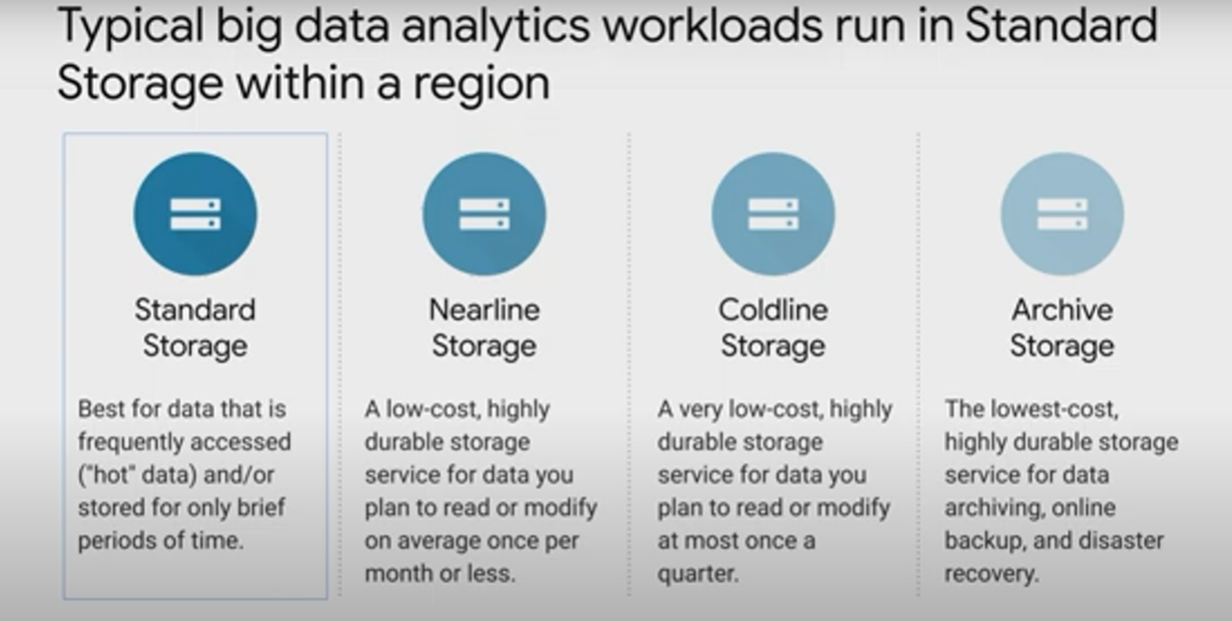
Submitted : 2022-02-24 16:43:30

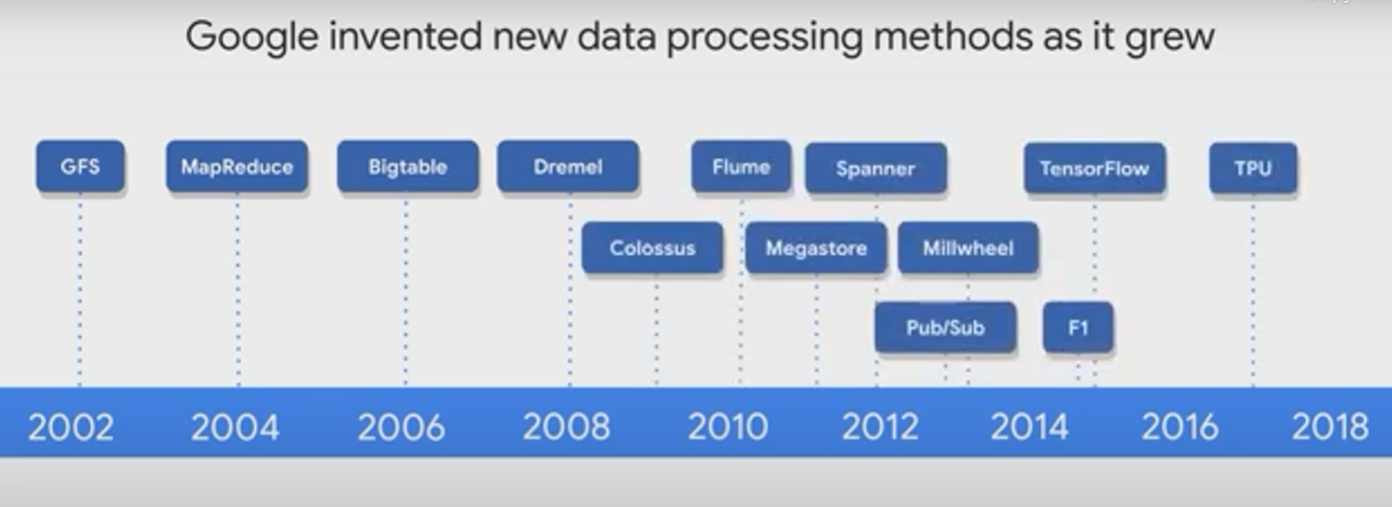
Request Number : **REQ2536284**

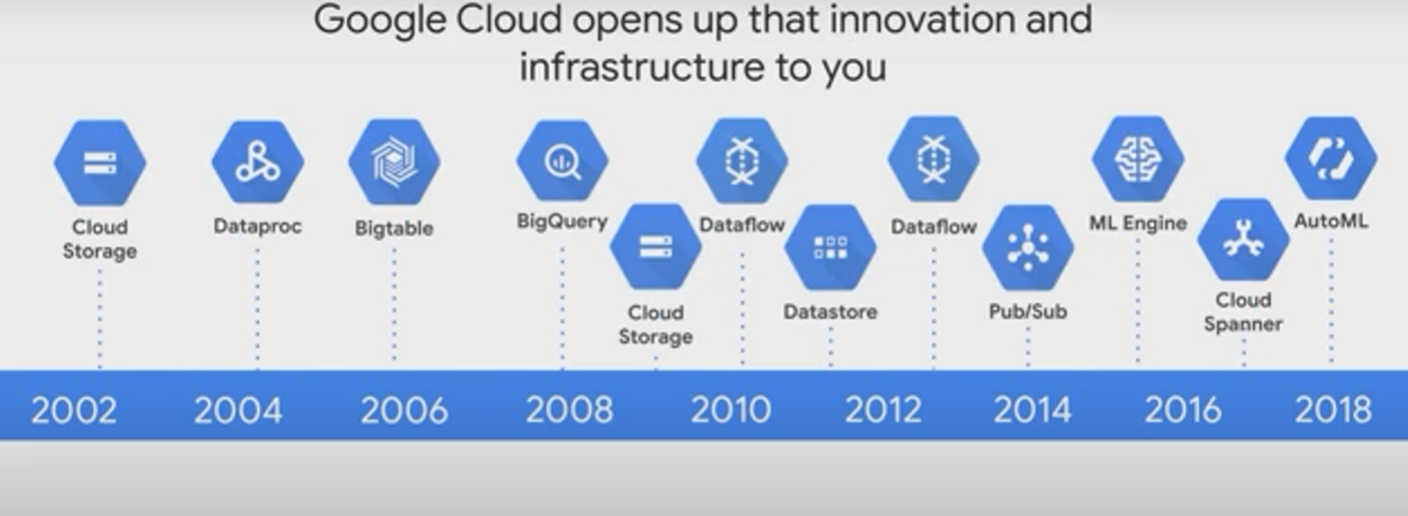
Estimated Delivery : 2022-02-24

Create Storage account









**LAB-1**

Exploring a BigQuery Public Dataset

1 hourFree

**Overview**

Storing and querying massive datasets can be time consuming and expensive without the right hardware and infrastructure. BigQuery is an [enterprise data warehouse](https://cloud.google.com/solutions/bigquery-data-warehouse) that solves this problem by enabling super-fast SQL queries using the processing power of Google's infrastructure. Simply move your data into BigQuery and let us handle the hard work. You can control access to both the project and your data based on your business needs, such as giving others the ability to view or query your data.

You access BigQuery through the Cloud Console, the [command-line tool](https://cloud.google.com/bigquery/docs/cli_tool), or by making calls to the [BigQuery REST API](https://cloud.google.com/bigquery/docs/reference/v2" \t "_blank) using a variety of [client libraries](https://cloud.google.com/bigquery/docs/reference/libraries) such as Java, .NET, or Python. There are also a variety of third-party tools that you can use to interact with BigQuery, such as visualizing the data or loading the data. In this lab, you access BigQuery using the web UI.

You can use the BigQuery web UI in the Cloud Console as a visual interface to complete tasks like running queries, loading data, and exporting data. This hands-on lab shows you how to query tables in a public dataset and how to load sample data into BigQuery through the Cloud Console.

Objectives

In this lab, you learn how to perform the following tasks:

* Query a public dataset
* Create a custom table
* Load data into a table
* Query a table

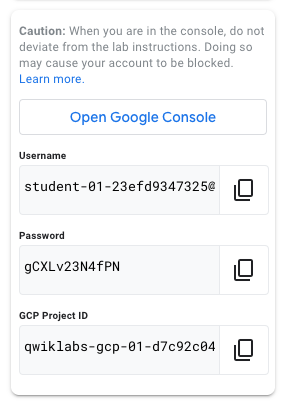
**Set up your environments**

Qwiklabs setup

For each lab, you get a new GCP project and set of resources for a fixed time at no cost.

1. Make sure you signed into Qwiklabs using an **incognito window**.
2. Note the lab's access time (for example,  and make sure you can finish in that time block.

There is no pause feature. You can restart if needed, but you have to start at the beginning.

1. When ready, click .
2. Note your lab credentials. You will use them to sign in to Cloud Platform Console. 
3. Click **Open Google Console**.
4. Click **Use another account** and copy/paste credentials for **this** lab into the prompts.

If you use other credentials, you'll get errors or **incur charges**.

1. Accept the terms and skip the recovery resource page.

Do not click **End Lab** unless you are finished with the lab or want to restart it. This clears your work and removes the project.

Open BigQuery Console

1. In the Google Cloud Console, select **Navigation menu** > **BigQuery**.

The **Welcome to BigQuery in the Cloud Console** message box opens. This message box provides a link to the quickstart guide and lists UI updates.

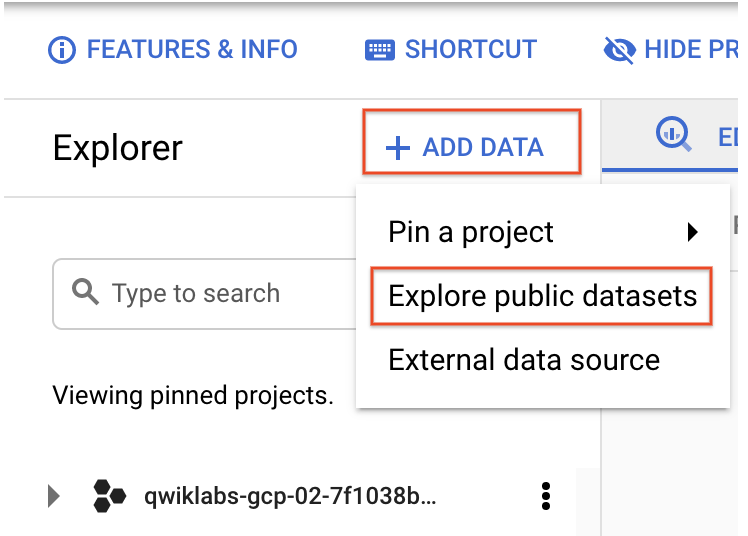
1. Click **Done**.

**Task 1. Query a public dataset**

In this task, you load a public dataset, USA Names, into BigQuery, then query the dataset to determine the most common names in the US between 1910 and 2013.

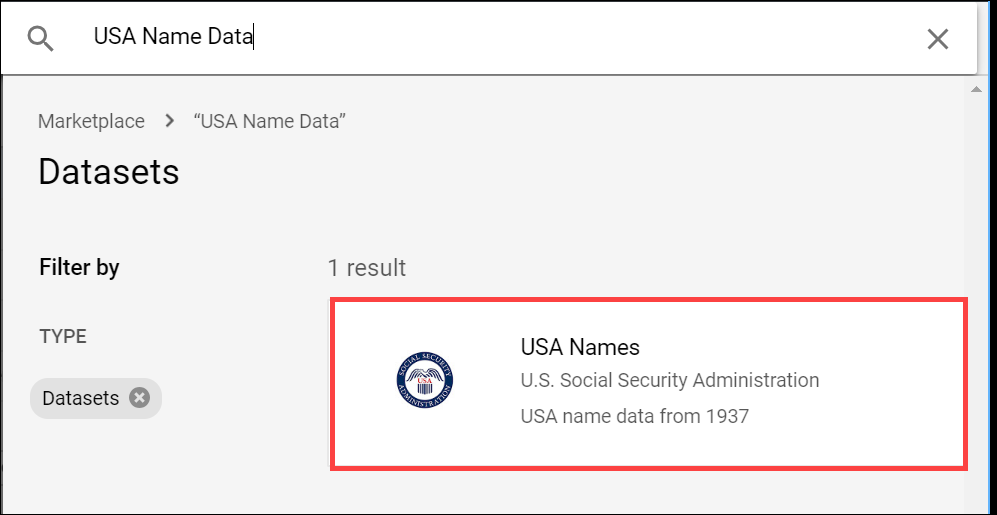
Load the USA Names dataset

1. In the left pane, click **ADD DATA** > **Explore public datasets**.



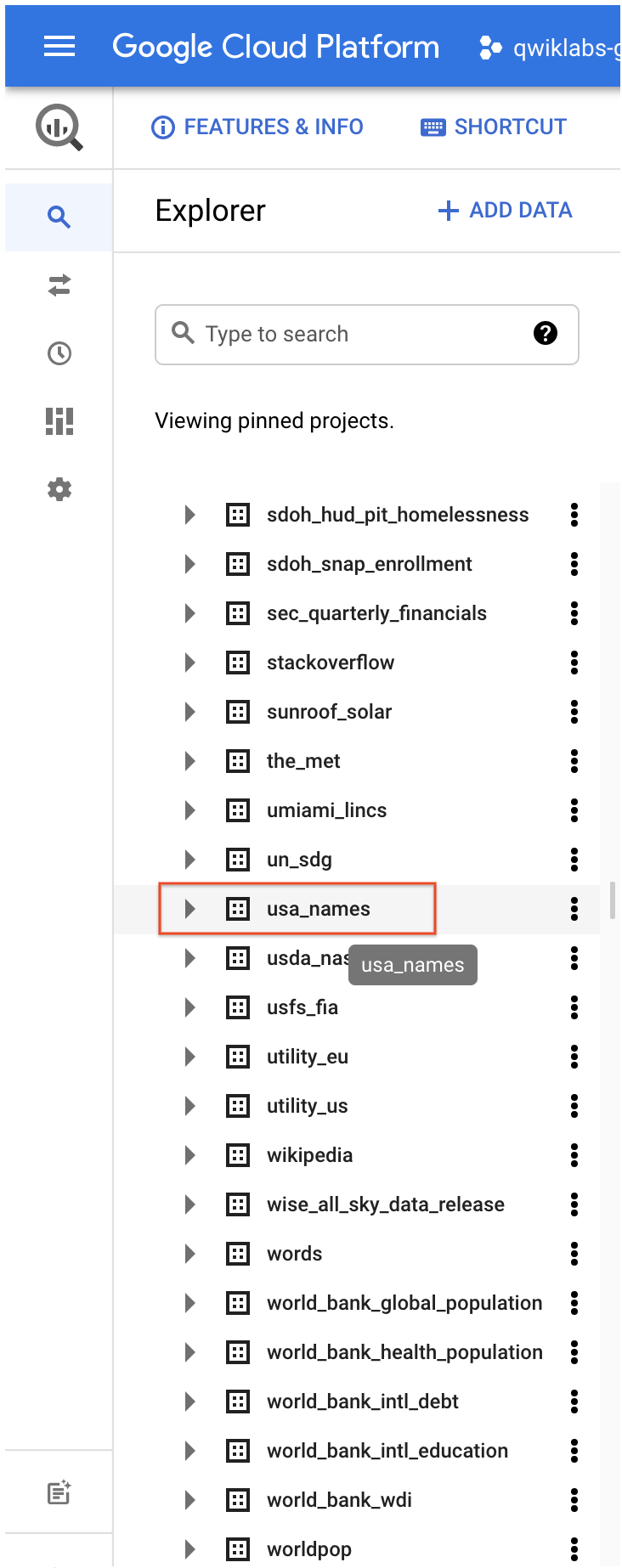
The Datasets window opens.

1. In the searchbox, type USA Names then press ENTER.
2. Click on the **USA Names** tile you see in the search results.



1. Click **View dataset**.

BigQuery opens in a new browser tab. The project bigquery-public-data is added to your resources and you see the dataset usa\_names listed in the left pane in your Resources tree.



Query the USA Names dataset

Query bigquery-public-data.usa\_names.usa\_1910\_2013 for the name and gender of the babies in this dataset, and then list the top 10 names in descending order.

1. Copy and paste the following query into the **Query editor** text area:

SELECT

name, gender,

SUM(number) AS total

FROM

`bigquery-public-data.usa\_names.usa\_1910\_2013`

GROUP BY

name, gender

ORDER BY

total DESC

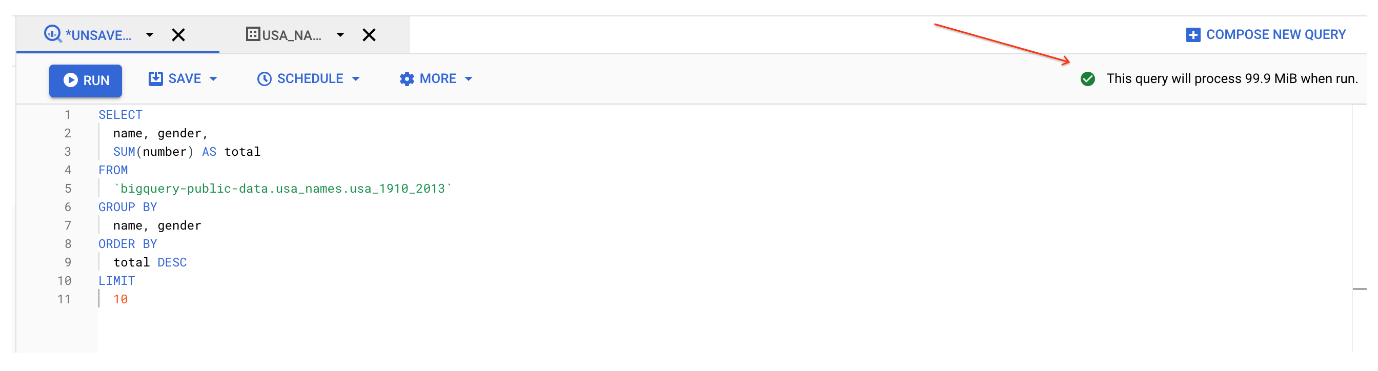
LIMIT

10

Copied!

content\_copy

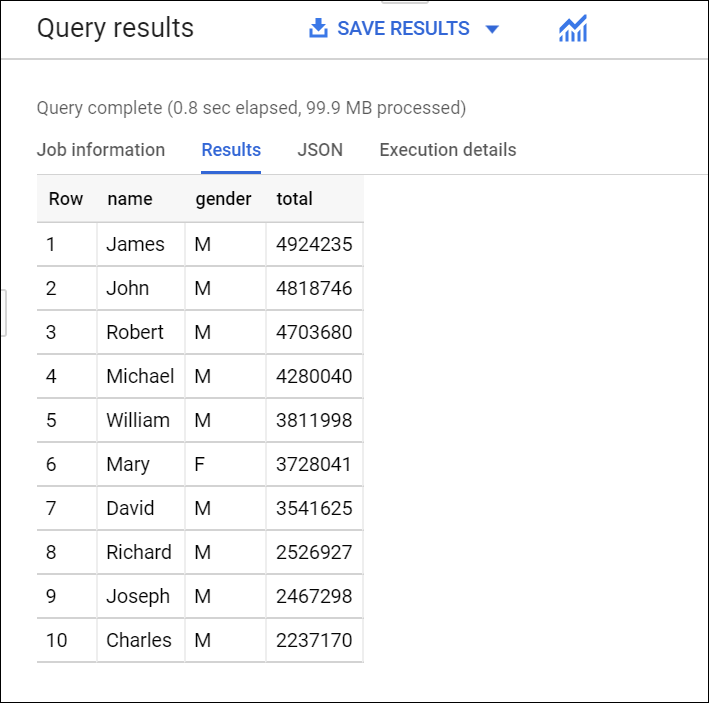
1. In the upper right of the window, view the query validator.



BigQuery displays a green check mark icon if the query is valid. If the query is invalid, a red exclamation point icon is displayed. When the query is valid, the validator also shows the amount of data the query processes when you run it. This helps to determine the cost of running the query.

1. Click **Run**.

The query results opens below the Query editor. At the top of the Query results section, BigQuery displays the time elapsed and the data processed by the query. Below the time is the table that displays the query results. The header row contains the name of the column as specified in GROUP BY in the query.



**Task 2. Create a custom table**

In this task, you create a custom table, load data into it, and then run a query against the table.

Download the data to your local computer

The file you're downloading contains approximately 7 MB of data about popular baby names, and it is provided by the US Social Security Administration.

1. Download the [baby names zip file](https://www.ssa.gov/OACT/babynames/names.zip) to your local computer.

**Note:** If this download link fails please copy the baby names zip file from the student resources on the left pane of the instruction guide.

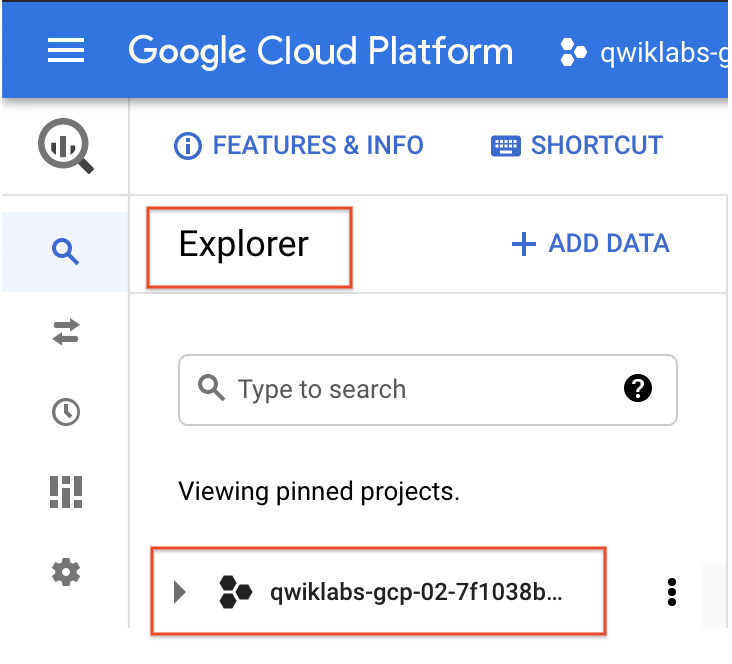
1. Unzip the file onto your computer.
2. The zip file contains a NationalReadMe.pdf file that describes the dataset. [Learn more about the dataset](https://www.ssa.gov/OACT/babynames/background.html).
3. Open the file named yob2014.txt to see what the data looks like. The file is a comma-separated value (CSV) file with the following three columns: name, sex (M or F), and number of children with that name. The file has no header row.
4. Note the location of the yob2014.txt file so that you can find it later.

**Task 3. Create a dataset**

In this task, you create a dataset to hold your table, add data to your project, then make the data table you'll query against.

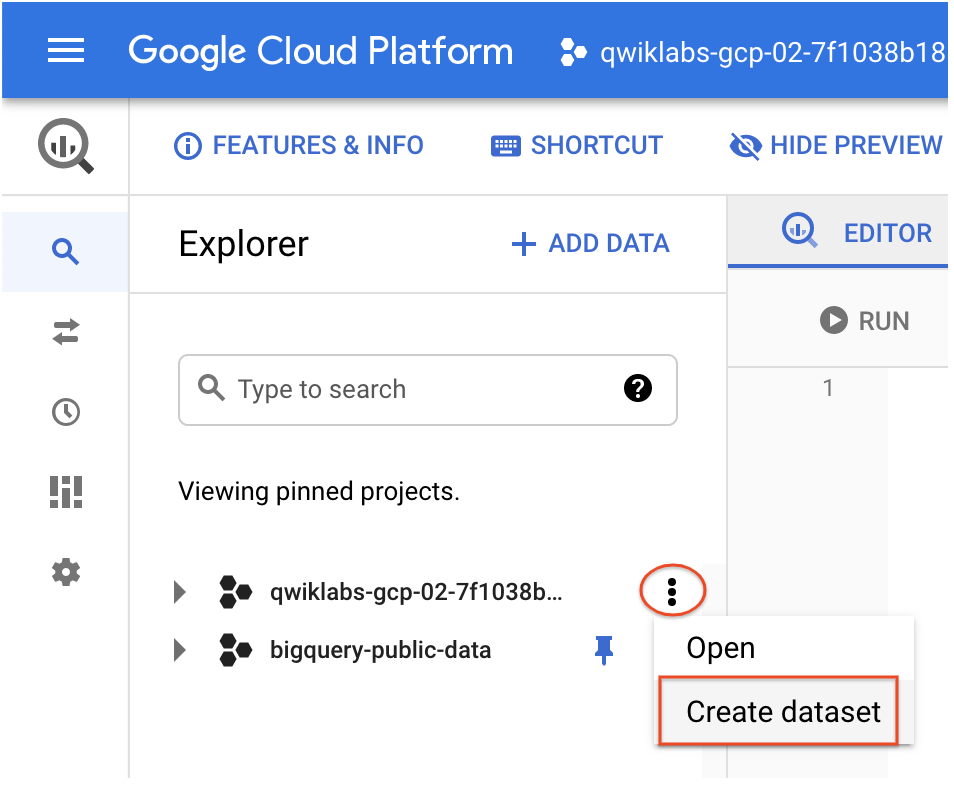
Datasets help you control access to tables and views in a project. This lab uses only one table, but you still need a dataset to hold the table.

1. Back in the Cloud Console, in the left pane, in the **Explorer** section, click your Project ID (it will start with qwiklabs).



Your project opens under the Query editor.

1. Click on the three dots next to your project ID and then click **Create dataset**.



1. On the **Create dataset** page:

* For **Dataset ID**, enter babynames.
* For **Data location**, choose **us (multiple regions in United States)**.
* For **Default table expiration**, leave the default value.
* For **Encryption**, leave the default value.

1. Click **Create dataset** at the bottom of the pane.

**Task 4. Load the data into a new table**

In this task, you load data into the table you made.

1. Click **babynames** found in the left pane in the **Explorer** section, and then click **Create table**.

Use the default values for all settings unless otherwise indicated.

1. On the **Create table** page:

* For **Source**, choose **Upload** from the Create table from: dropdown menu.
* For **Select file**, click **Browse**, navigate to the yob2014.txt file and click **Open**.
* For **File format**, choose **CSV** from the dropdown menu.
* For **Table name**, enter names\_2014.
* In the **Schema** section, click the **Edit as text** toggle and paste the following schema definition in the text box.

name:string,gender:string,count:integer

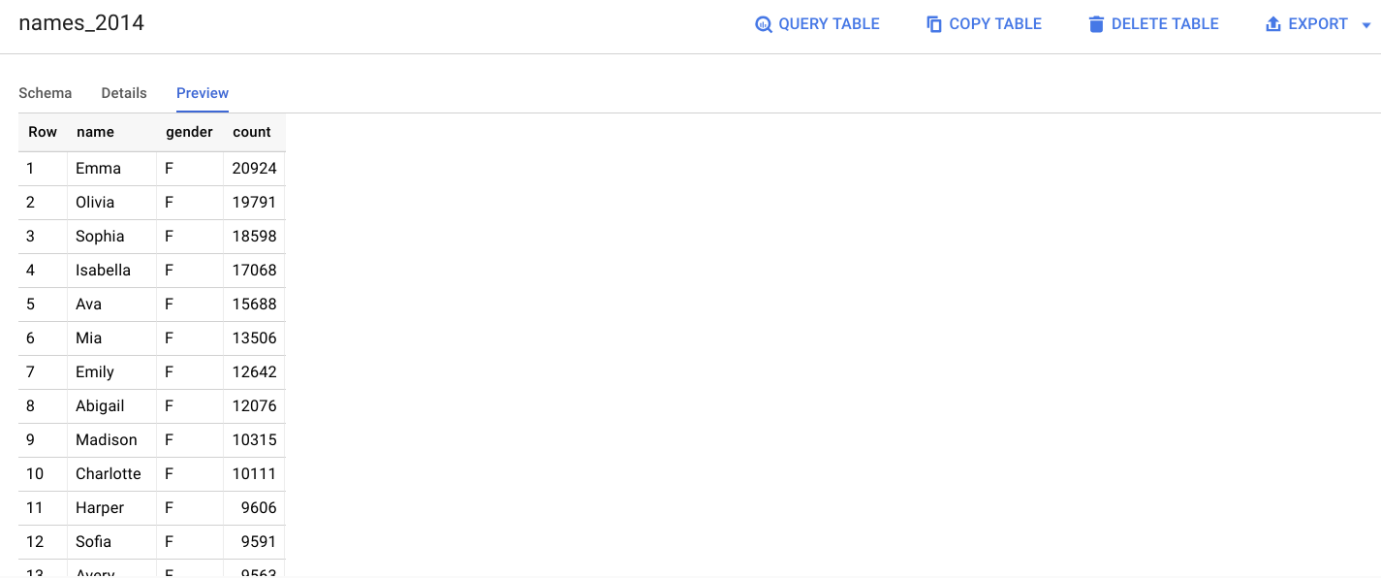
Copied!

content\_copy

1. Click **Create table** (at the bottom of the window).

Preview the table

1. In the left pane, select **babynames** > **names\_2014** in the navigation pane.
2. In the details pane, click the **Preview** tab.



Quick quiz. You need a table to hold the dataset.

close~~True~~

checkFalse

**Task 5. Query the tables**

Now that you've loaded data into your table, you can run queries against it. The process is identical to the previous example, except that this time, you're querying your table instead of a public table.

1. In the Query editor, click **Compose new query**.
2. Copy and paste the following query into the **Query editor**. This query retrieves the top 5 baby names for US males in 2014.

Note: Inside '' it does distinguish upper vs. lower case, therefore make sure to align exactly the names of the dataset and the table you created.

SELECT

name, count

FROM

`babynames.names\_2014`

WHERE

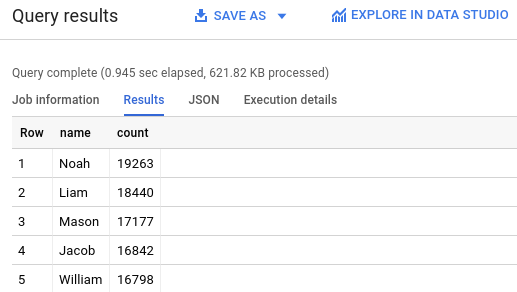
gender = 'M'

ORDER BY count DESC LIMIT 5

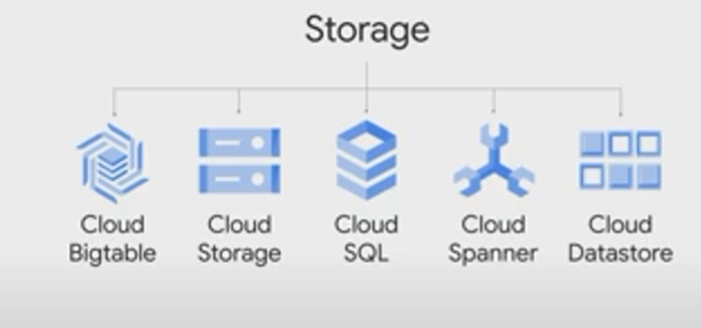
Copied!

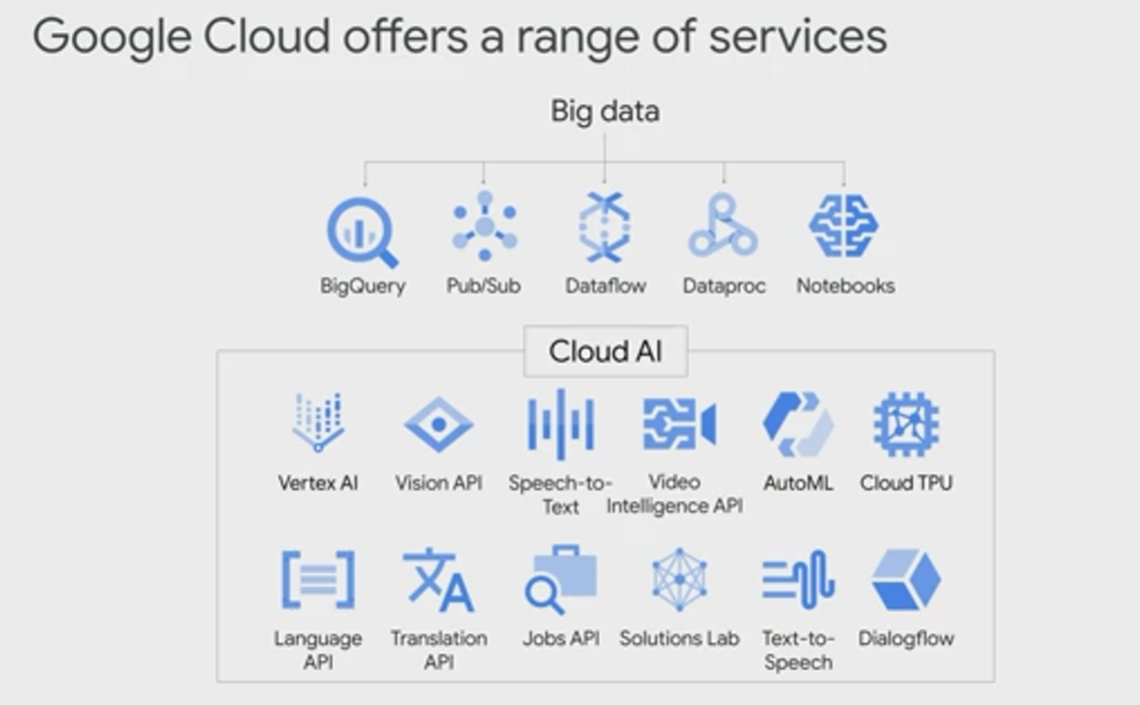
content\_copy

1. Click **Run**. The results are displayed below the query window.



# Types of applications given by GCP









# Car recommendation System by ML

**Challenge: ML for recommending housing rentals transcript**

Person: So how would housing recommendations work in our system?

First, we need to ingest the ratings of all the houses that have already been done by our users when we showed them specific houses.

So we have to go to our inventory of rentals and ingest the ratings for the houses.

These ratings could come from explicit ratings.

Maybe we showed the user the house in the past and they've clicked four stars after seeing the house details.

Or the ratings could come from implicit ratings.

Maybe they spent a lot of time looking at the website corresponding to this property?

Then we'll train a machine learning model to predict a user's rating of every house that we currently have in our listing database.

We will then pick the top five rated houses that they haven't already seen.

People who are not aware of recommendation engines somehow think of a car appearing on their Facebook feed because they happened to read an automobile article.

They say, "Oh, I was reading this review article and then just kept seeing cars in my Facebook feed."

That's not the case.

Reading the article caused the rating of all cars to go up and the ratings of other items that are not cars to stay relatively the same.

And this caused the highest ranking car to get into the top five.

So there is always a rating for a car, the car was always there.

But it's rating was simply lower before they read the article in question.

So how would we predict a user's rating of a house?

Particularly if they haven't seen it before?

The model is based on two things.

It's based on your other ratings.

What have you rated other houses?

And other people's rating of this particular house.

A particularly simple model could be to look at all the users who rated this particular house and find the three users in their list who are most like you.

Maybe they live in your country, maybe they have the same age, maybe they went to the same college.

Find the three users in that list who are most like you.

And then the prediction is the average of those three users' ratings.

This is not a great model, of course.

It's too easy to game.

Think about what happens if three people gang up together and rate a house, a house that nobody else cares about so there are only three ratings for the house, sot he three closes users are going to be these three people.

So for sparse houses it's really, really easy to game.

But, this idea of using user ratings of a particular house and users like you helps convey the basic premise of how recommendation models work.

So where is the machine learning here?

Where is the learning?

The model would have to figure out how to find the users who are most like you.

How many users to consider: three users, five users, seven users?

How many?

And, how to weight the different factors such as the overall popularity of the items you have in common and so on.

This can be done by seeing what parameters help predict a few intentionally held ratings best.

So, we may have thousands of items and only two or three reviews per item.

The chances are those reviewers have nothing in common with the user that we want the rating for.

So because this rating matrix is extremely sparse, we need to cluster items and users together.

To put this in a more intuitive way, imagine that all of your friends drive SUVs, sport utility vehicles and you read an article on Porsche.

The car that appears on your feed might be a Porsche SUV, even if the article was on Porsche cars and all your friends drive Toyota SUVs.

The machine learning model is imputing a rating for a Porsche SUV that is quite high even though none of your friends rated it.

So the machine learning model is essentially asking, "Who is this user like?"

And secondly, "Is this objectively a house that people tend to rate highly?"

The predicted rating is a combination of both these factors.

All things considered, the rating of a house for a particular user will be the average of the ratings of users like this user.

But it's calibrated with the quality of the item itself.

Now that we understand the problem and approach, we need to address the last question, the infrastructure question.

How often and where will you compute the predicted ratings?

And once you have them, once you've computed these ratings, where will you store them?

What do you think?

How often and where will you compute the predicted ratings?

It's not as if rental recommendations have to be updated every time some user somewhere rates a house.

We don't need to update the rental recommendations every time a new rating appears in our system.

It's probably sufficient if we update the recommended houses for uses once a day, maybe even once a week.

In other words, this does not need to be streaming, it could be batch.

On the other hand, you will probably have thousands of houses and millions of users, so it's probably best that we compute the rating that every user will give every house, we do that computation in a scalable way.

We don't want to do it on a single machine, we want to do it in a fault-tolerant way that can scale to large data sets.

So a typical solution for computation that has to happen over large data sets in a fault-tolerant way is to do it in a big data platform like Apache Hadoop.

Finally, where will you store the computed ratings?

Why would you want to store them?

Well, you probably want to power a web application with these recommendations and we don't want to compute recommendations when the user visits our web page, we want to precompute these recommendations, as we said, it's a batch job.

So once a day, once a week, we precompute it.

And then when the user logs on, we want to show that user the recommendations that we precomputed specifically for them.

So, we need a transactional way to store the predictions.

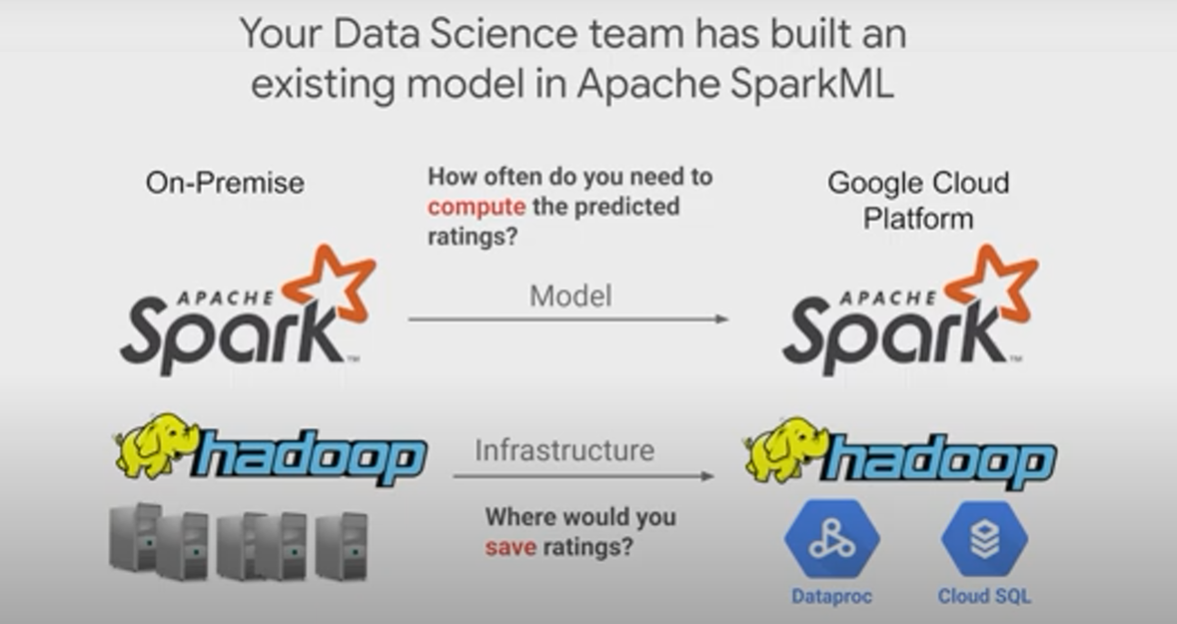
Why transactional?

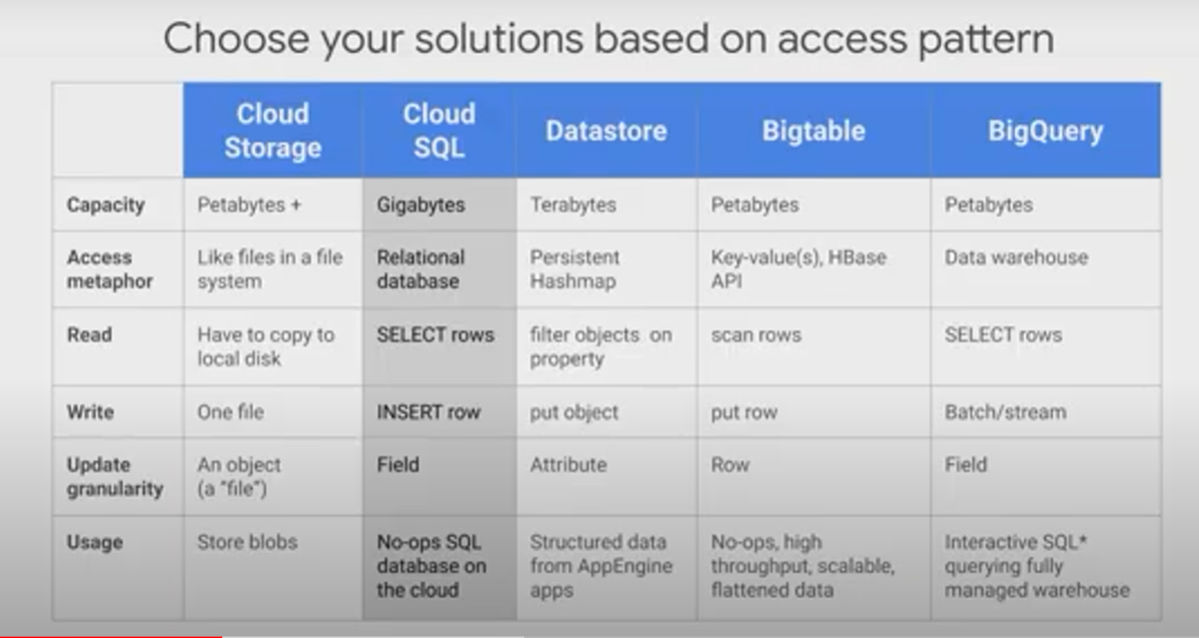
So that while the user is reading these predictions we can update the predictions table as well.

Assuming that there are five predictions for every user and we have 1 million users, that's a table of just 5 million rows.

It's small enough and compact enough that a typical solution for this would be to store the data in a relational database management system, an RDBM like MySQL.

# On-premise to Cloud:





## Choosing type of Storage:

This is a good reference to follow based on your storage access pattern.

We'll cover the solutions in the other scenarios in this course, but briefly use Cloud Storage as a global file system.

Use Cloud SQL as an RDBMS, as a relational database management system for transactional relational data that you access through SQL.

Use data store as a transactional no SQL object-oriented database.

Use Bigtable for high throughput, no SQL, append-only data, so not transactions, append-only data.

So a typical use case for Bigtable is sensor data for connected devices, for example.

Use BigQuery as a SQL data warehouse to power all your analytics needs.

So here we wanted a transactional database and we expect to have data volumes in the gigabytes or less.

And so hence, Cloud SQL.

If you're a more visual learner, here is another good map of visualizing where to store your data in GCP.

If your data is unstructured like images or audio, use Cloud Storage, if your data is structured and you need transactions, use Cloud SQL or Cloud Data Store, depending on whether you want your access pattern to be SQL or no SQL, and by no SQL, we mean a key value pair, in other words, you'll be trying to search for data based on a single key, use data store.

If you'll be finding data using SQL, use Cloud SQL.

Cloud SQL generally plateaus out at a few gigabytes.

So if you want a transactional database that is horizontally scalable so that you can deal with data larger than a few gigabytes, or if you need multiple databases so you want them spread globally, use Cloud Spanner.

So another way to say it is, if one database is enough, use Cloud SQL, if you need multiple databases, either because you have a lot of data or because your application needs to be transactional across different continents, use Cloud Spanner.

If your data is structured and you want analytics, consider either Bigtable or BigQuery.

Use Bigtable if you need real-time, high throughput applications.

Use BigQuery if you want analytics over petabyte scale datasets.

For our housing recommendation use case, we want to store our ratings and predictions somewhere.

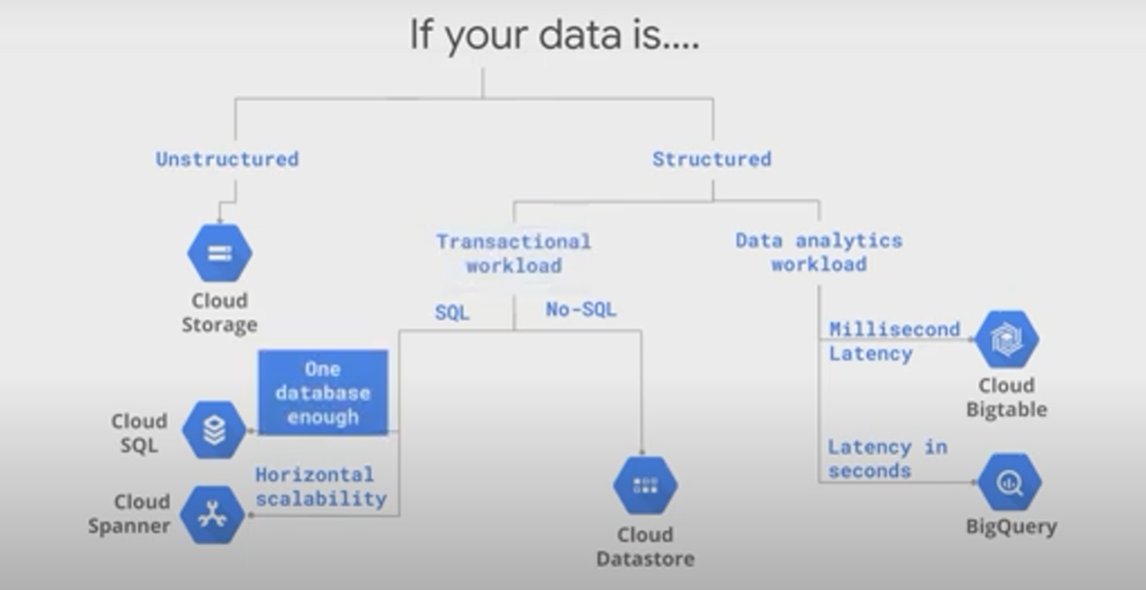
This is a structured data set of user ratings and houses, it's built for a transactional workload rights and reads, and one databases enough for our small data set, and hence, we pick Cloud SQL.

So what's Cloud SQL?

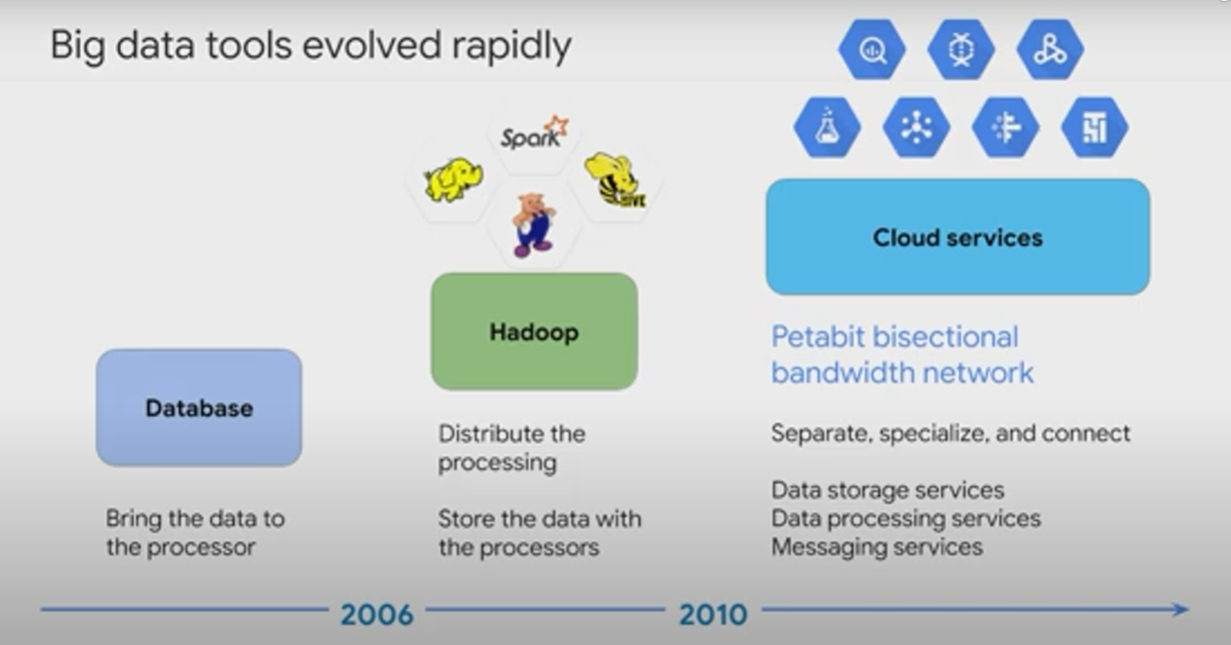
It's a Google hosted and managed relational database in the Cloud.

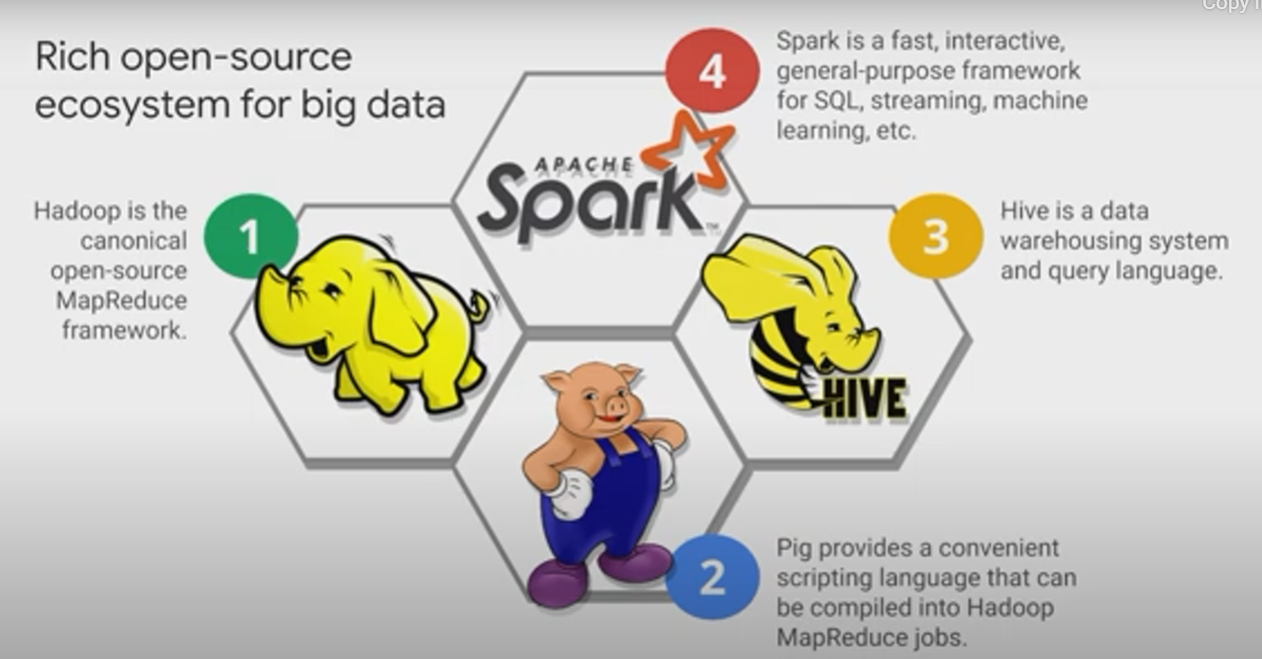
Cloud SQL supports two open-source databases, my SQL and Postgres and other database solutions.

In our case, we'll be using my SQL.



# Big Data tools:





Let's review where big data computations have been done historically and even today.

Before 2006, big data meant big databases.

Database design came from a time when storage was relatively cheap and processing was expensive, so it made sense to copy the data from its storage location to the processor, to perform data processing.

And then the result would be copied back to storage.

Around 2006, distributed processing of big data became practical with Hadoop.

The idea behind Hadoop is to create a cluster of computers and leverage distributed processing, HDFS.

The Hadoop distributed file system, stored the data on machines in the cluster, and map produce provided distributed processing of this data.

A whole ecosystem of Hadoop related software group around Hadoop, including Hive, Pig, and spark.

Around around 2010, BigQuery was released, which was a first of many big data services developed by Google.

Around 2015, Google launched Cloud Dataproc, which provides a managed service for creating Hadoop and Spark clusters and managing data processing workloads.

The other piece of our system is a software that runs on Hadoop, the software in this case is to train a machine learning model for creating housing recommendations.

In this case, we used Spark ML, which is part of Apache Spark.

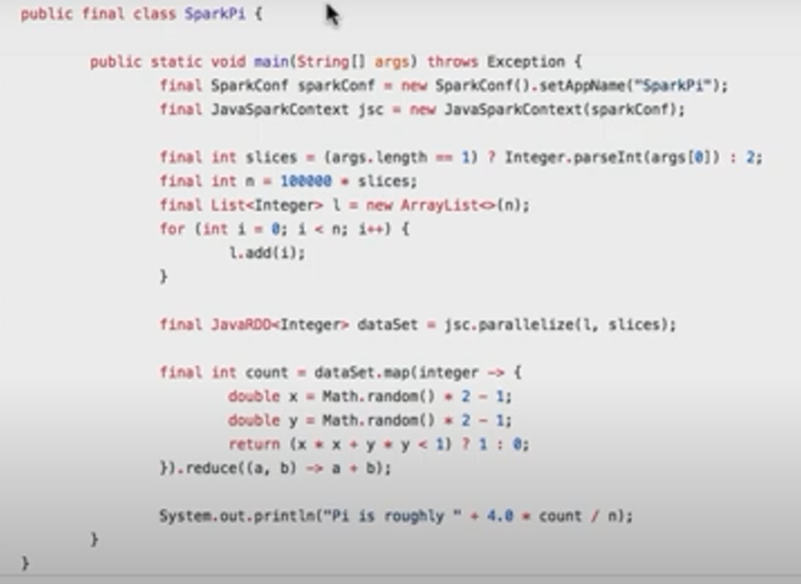
Apache Spark is an open-source software project that provides a high performance analytics engine for processing batch and streaming data.

Spark can be up to a hundred times faster than equivalent Hadoop jobs because it leverages in memory processing.

Spark also provides a couple of abstractions for dealing with data, including what we call RDDs, our resilient, distributed data sets and data frames.

We'll be using our Spark job for the housing rental recommendations.

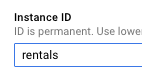
We'll be using our Spark job for the housing rental recommendations, but we'll be doing the computation on Cloud Dataproc.



# Recommendation Project – Lab

## Task 1. Create a Cloud SQL instance

1. In the Google Cloud Console, Select **Navigation menu > SQL** (in the Databases section).
2. Click **Create instance**.
3. Click **Choose MySQL**.
4. For **Instance ID**, type **rentals**.



1. Scroll down and specify a **Root password**. Before you forget, note down the root password.
2. For **Region** select **us-central1**.
3. Click **Create instance** to create the instance. It will take a minute or so for your Cloud SQL instance to be provisioned.

## Task 2. Create tables

1. While you wait for your instance to be created, read the below mySQL script and answer the questions that follow.
2. In Cloud SQL, click rentals to view instance information.

### Connect to the database

1. Find the **Connect to this instance** box on the page and click on **Open Cloud Shell**.

**Note:** You could also connect to your instance from a dedicated Cloud Compute Engine VM but for now you'll have Cloud Shell create a micro-VM for you and operate from there.

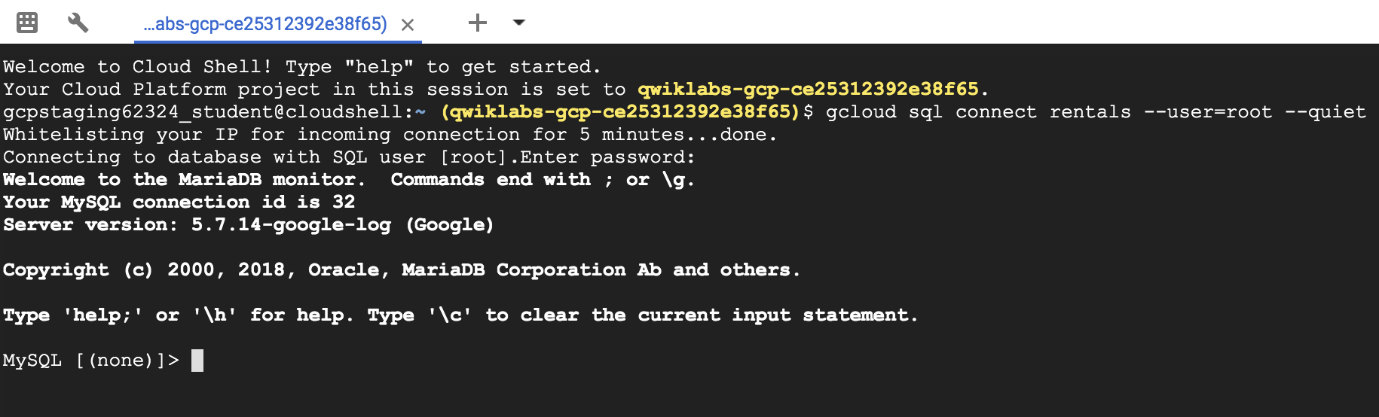
1. If required, click **Continue**. Wait for Cloud Shell to load.
2. Once Cloud Shell loads, you will see the below command already typed:

* gcloud sql connect rentals --user=root --quiet

1. Press **ENTER**.
2. Wait for your IP Address to be whitelisted.

Allowlisting your IP for incoming connection for 5 minutes...⠹

1. When prompted, enter your password and press **ENTER** (note: you will not see your password typed in or even \*\*\*\*).

You can now run commands against your database! 

1. Run the following command:

SHOW DATABASES;

Copied!

content\_copy

You should see the default system databases:

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

| Database |

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

| information\_schema |

| mysql |

| performance\_schema |

| sys |

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

**Note:** You must always end your mySQL commands with a semi-colon ;

1. Copy and paste the below SQL statement you analyzed earlier into the command line.

CREATE DATABASE IF NOT EXISTS recommendation\_spark;

USE recommendation\_spark;

DROP TABLE IF EXISTS Recommendation;

DROP TABLE IF EXISTS Rating;

DROP TABLE IF EXISTS Accommodation;

CREATE TABLE IF NOT EXISTS Accommodation

(

id varchar(255),

title varchar(255),

location varchar(255),

price int,

rooms int,

rating float,

type varchar(255),

PRIMARY KEY (ID)

);

CREATE TABLE IF NOT EXISTS Rating

(

userId varchar(255),

accoId varchar(255),

rating int,

PRIMARY KEY(accoId, userId),

FOREIGN KEY (accoId)

REFERENCES Accommodation(id)

);

CREATE TABLE IF NOT EXISTS Recommendation

(

userId varchar(255),

accoId varchar(255),

prediction float,

PRIMARY KEY(userId, accoId),

FOREIGN KEY (accoId)

REFERENCES Accommodation(id)

);

SHOW DATABASES;

Copied!

content\_copy

1. Press **ENTER**.
2. Confirm that you now see recommendation\_spark as a database:

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

| Database |

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

| information\_schema |

| mysql |

| performance\_schema |

| recommendation\_spark |

| sys |

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

1. Run the following command to show the tables:

USE recommendation\_spark;

SHOW TABLES;

1. Press **ENTER**.
2. Confirm that you see the three tables:

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

| Tables\_in\_recommendation\_spark |

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

| Accommodation |

| Rating |

| Recommendation |

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

1. Run the following query:

SELECT \* FROM Accommodation;

## Task 3. Stage data in Cloud Storage

### Option 1: Use the command line

1. Open a new Cloud Shell tab **(do not use your existing mySQL Cloud Shell tab)**.
2. Copy and paste the following command:

echo "Creating bucket: gs://$DEVSHELL\_PROJECT\_ID"

gsutil mb gs://$DEVSHELL\_PROJECT\_ID

echo "Copying data to our storage from public dataset"

gsutil cp gs://cloud-training/bdml/v2.0/data/accommodation.csv gs://$DEVSHELL\_PROJECT\_ID

gsutil cp gs://cloud-training/bdml/v2.0/data/rating.csv gs://$DEVSHELL\_PROJECT\_ID

echo "Show the files in our bucket"

gsutil ls gs://$DEVSHELL\_PROJECT\_ID

echo "View some sample data"

gsutil cat gs://$DEVSHELL\_PROJECT\_ID/accommodation.csv

Copied!

content\_copy

1. Press **ENTER**.

### Option 2: Use the Cloud Console UI

Skip these steps if you have already loaded your data using the command line.

1. Navigate to **Storage** and select **Cloud Storage > Browser**.
2. Click **Create Bucket** (if one does not already exist).
3. Specify your project name as the bucket name.
4. Click **Create**.
5. Download the below files locally and then upload them inside of your new bucket:

* [accommodation.csv](https://storage.googleapis.com/cloud-training/bdml/v2.0/data/accommodation.csv)
* [rating.csv](https://storage.googleapis.com/cloud-training/bdml/v2.0/data/rating.csv)

## Task 4. Load data from Cloud Storage into Cloud SQL tables

1. Navigate back to **SQL**.
2. Click on **rentals**.

### Import accommodation data

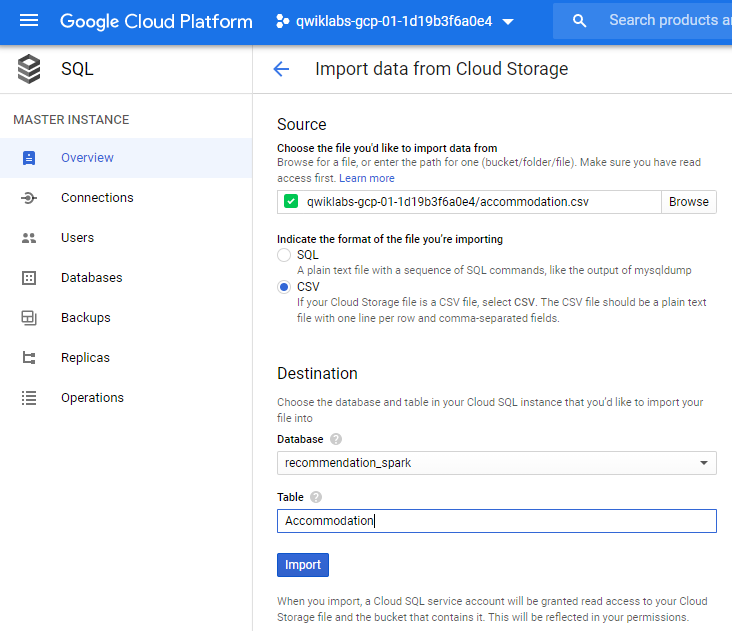
1. Click **Import** (top menu).
2. Specify the following:

* Source: Click **Browse > [Your-Bucket-Name] > accommodation.csv**

Click **Select**.

* Format of import: **CSV**
* Database: select recommendation\_spark from the dropdown list
* Table: copy and paste: Accommodation

1. Click **Import**.



1. You will be redirected back to the Overview page. Wait one minute for the data to load.

### Import user rating data

1. Click **Import** (top menu).
2. Specify the following:

* Source: Click **Browse > [Your-Bucket-Name] > rating.csv**

Click **Select**.

* Format of import: **CSV**
* Database: select recommendation\_spark from the dropdown list
* Table: copy and paste: Rating

1. Click **Import**.
2. You will be redirected back to the Overview page. Wait one minute for the data to load.

## Task 5. Explore Cloud SQL data

1. If you closed your Cloud Shell connection to mySQL, open it again by finding **Connect to this instance** and clicking **Open Cloud Shell**.
2. Press **ENTER** when prompted to log in.
3. Provide your password and press **ENTER**.
4. Query the ratings data:

USE recommendation\_spark;

SELECT \* FROM Rating

LIMIT 15;

Copied!

content\_copy

1. Use a SQL aggregation function to count the number of rows in the table.

SELECT COUNT(\*) AS num\_ratings

FROM Rating;

1. What is the average review rating of accommodations?

SELECT

COUNT(userId) AS num\_ratings,

COUNT(DISTINCT userId) AS distinct\_user\_ratings,

MIN(rating) AS worst\_rating,

MAX(rating) AS best\_rating,

AVG(rating) AS avg\_rating

FROM Rating;

In machine learning, you will need a rich history of user preferences for the model to learn from. Run the below query to see which users have provided the most ratings.

SELECT

userId,

COUNT(rating) AS num\_ratings

FROM Rating

GROUP BY userId

ORDER BY num\_ratings DESC;

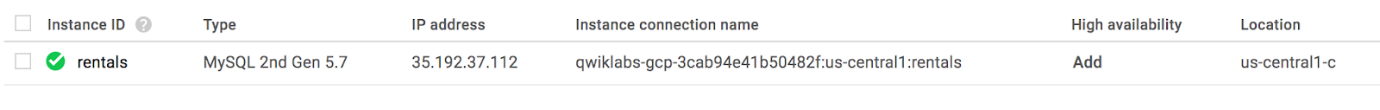
1. Exit the mysql prompt by typing **exit**.

## Task 6. Launch Dataproc

You use Dataproc to train the recommendations machine learning model based on users' previous ratings. You then apply that model to create a list of recommendations for every user in the database

To launch Dataproc and configure it so that each of the machines in the cluster can access Cloud SQL:

1. In the Cloud Console, on the **Navigation menu** (Navigation menu), click **SQL** and note the region of your Cloud SQL instance:



In the snapshot above, the region is us-central1 and zone is us-central1-c.

1. In the Cloud Console, on the **Navigation menu** (Navigation menu), click **Dataproc** and click **Enable API** if prompted.
2. Once enabled, click **Create cluster** and name your cluster **rentals**.
3. Leave the **Region** as it is i.e. **us-central1** and change the **Zone** to **us-central1-c** (in the same zone as your Cloud SQL instance). This will minimize network latency between the cluster and the database.
4. Click on **Configure nodes**.
5. For **Master node**, for **Machine type**, select **n1-standard-2 (2 vCPUs, 7.5 GB memory)**.
6. For **Worker nodes**, for **Machine type**, select **n1-standard-2 (2 vCPUs, 7.5 GB memory)**.
7. Leave all other values with their default and click **Create**. It will take 1-3 minutes to provision your cluster.
8. Note the **Name**, **Zone** and **Total worker nodes** in your cluster.
9. Copy and paste the below bash script into your Cloud Shell (optionally change **CLUSTER, ZONE, NWORKERS** if necessary before running)

echo "Authorizing Cloud Dataproc to connect with Cloud SQL"

CLUSTER=rentals

CLOUDSQL=rentals

ZONE=us-central1-c

NWORKERS=2

machines="$CLUSTER-m"

for w in `seq 0 $(($NWORKERS - 1))`; do

machines="$machines $CLUSTER-w-$w"

done

echo "Machines to authorize: $machines in $ZONE ... finding their IP addresses"

ips=""

for machine in $machines; do

IP\_ADDRESS=$(gcloud compute instances describe $machine --zone=$ZONE --format='value(networkInterfaces.accessConfigs[].natIP)' | sed "s/\['//g" | sed "s/'\]//g" )/32

echo "IP address of $machine is $IP\_ADDRESS"

if [ -z $ips ]; then

ips=$IP\_ADDRESS

else

ips="$ips,$IP\_ADDRESS"

fi

done

echo "Authorizing [$ips] to access cloudsql=$CLOUDSQL"

gcloud sql instances patch $CLOUDSQL --authorized-networks $ips

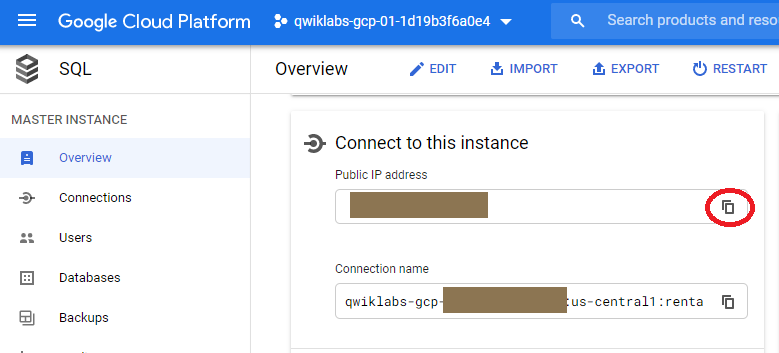
Copied!

content\_copy

1. Press **ENTER**. When prompted, type **Y**, then press **ENTER** again to continue.
2. Wait for the patching to complete. You will see the following:

Patching Cloud SQL instance...done.

1. On the main Cloud SQL page, under **Connect to this instance**, copy your **Public IP Address** to your clipboard. (Alternatively, write it down because you're using it next.)



## Code for model:

#!/usr/bin/env python

"""

Copyright Google Inc. 2016

Licensed under the Apache License, Version 2.0 (the "License");

you may not use this file except in compliance with the License.

You may obtain a copy of the License at

http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software

distributed under the License is distributed on an "AS IS" BASIS,

WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.

See the License for the specific language governing permissions and

limitations under the License.

"""

import os

import sys

import pickle

import itertools

from math import sqrt

from operator import add

from os.path import join, isfile, dirname

from pyspark import SparkContext, SparkConf, SQLContext

from pyspark.mllib.recommendation import ALS, MatrixFactorizationModel, Rating

from pyspark.sql.types import StructType, StructField, StringType, FloatType

# MAKE EDITS HERE

CLOUDSQL\_INSTANCE\_IP = '34.70.157.202'   # <---- CHANGE (database server IP)

CLOUDSQL\_DB\_NAME = 'recommendation\_spark' # <--- leave as-is

CLOUDSQL\_USER = 'root'  # <--- leave as-is

CLOUDSQL\_PWD  = 'tiger'  # <---- CHANGE

# DO NOT MAKE EDITS BELOW

conf = SparkConf().setAppName("train\_model")

sc = SparkContext(conf=conf)

sqlContext = SQLContext(sc)

jdbcDriver = 'com.mysql.jdbc.Driver'

jdbcUrl    = 'jdbc:mysql://%s:3306/%s?user=%s&password=%s' % (CLOUDSQL\_INSTANCE\_IP, CLOUDSQL\_DB\_NAME, CLOUDSQL\_USER, CLOUDSQL\_PWD)

# checkpointing helps prevent stack overflow errors

sc.setCheckpointDir('checkpoint/')

# Read the ratings and accommodations data from Cloud SQL

dfRates = sqlContext.read.format('jdbc').options(driver=jdbcDriver, url=jdbcUrl, dbtable='Rating', useSSL='false').load()

dfAccos = sqlContext.read.format('jdbc').options(driver=jdbcDriver, url=jdbcUrl, dbtable='Accommodation', useSSL='false').load()

print("read ...")

# train the model

model = ALS.train(dfRates.rdd, 20, 20) # you could tune these numbers, but these are reasonable choices

print("trained ...")

# use this model to predict what the user would rate accommodations that she has not rated

allPredictions = None

for USER\_ID in range(0, 100):

  dfUserRatings = dfRates.filter(dfRates.userId == USER\_ID).rdd.map(lambda r: r.accoId).collect()

  rddPotential  = dfAccos.rdd.filter(lambda x: x[0] not in dfUserRatings)

  pairsPotential = rddPotential.map(lambda x: (USER\_ID, x[0]))

  predictions = model.predictAll(pairsPotential).map(lambda p: (str(p[0]), str(p[1]), float(p[2])))

  predictions = predictions.takeOrdered(5, key=lambda x: -x[2]) # top 5

  print("predicted for user={0}".format(USER\_ID))

  if (allPredictions == None):

    allPredictions = predictions

  else:

    allPredictions.extend(predictions)

# write them

schema = StructType([StructField("userId", StringType(), True), StructField("accoId", StringType(), True), StructField("prediction", FloatType(), True)])

dfToSave = sqlContext.createDataFrame(allPredictions, schema)

dfToSave.write.jdbc(url=jdbcUrl, table='Recommendation', mode='overwrite')

## Task 7. Run the ML model

Next, you create a trained model and apply it to all the users in the system. Your data science team has created a recommendation model using Apache Spark and is written in Python. Copy it over into your staging bucket.

1. Copy over the model code by executing the below commands in Cloud Shell:

gsutil cp gs://cloud-training/bdml/v2.0/model/train\_and\_apply.py train\_and\_apply.py

cloudshell edit train\_and\_apply.py

Copied!

content\_copy

1. When prompted, select **Open in New Window**.
2. Wait for the Editor UI to load.
3. Open the train\_and\_apply.py file, find line 30: **CLOUDSQL\_INSTANCE\_IP**, and paste the Cloud SQL IP address you copied earlier.

# MAKE EDITS HERE

CLOUDSQL\_INSTANCE\_IP = '<paste-your-cloud-sql-ip-here>' # <---- CHANGE (database server IP)

CLOUDSQL\_DB\_NAME = 'recommendation\_spark' # <--- leave as-is

CLOUDSQL\_USER = 'root' # <--- leave as-is

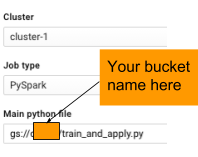
CLOUDSQL\_PWD = '<type-your-cloud-sql-password-here>' # <---- CHANGE

1. Find line 33: **CLOUDSQL\_PWD** and type in your Cloud SQL password,
2. The editor will autosave but to be sure, select **File > Save**.
3. From the Cloud Shell ribbon, click on the **Open Terminal** icon and copy this file to your Cloud Storage bucket using this Cloud Shell command:

gsutil cp train\_and\_apply.py gs://$DEVSHELL\_PROJECT\_ID

## Task 8. Run your ML job on Dataproc

1. In the **Dataproc** console, click **rentals** cluster.
2. Click **Submit job**.
3. For **Job type**, select **PySpark** and for **Main python file**, specify the location of the Python file you uploaded to your bucket. Your <bucket-name> is likely to be your Project ID, which you can find by clicking on the Project ID dropdown in the top navigation menu.



gs://<bucket-name>/train\_and\_apply.py

1. For **Max restarts per hour**, enter **1**.
2. Click **Submit**.
3. Select **Navigation menu > Dataproc > Job** tab to see the Job status.

**Note:** It will take up to 5 minutes for the job to change from Running to Succeeded. You can continue to the next section on querying the results while the job runs. If the job Failed, please troubleshoot using the logs and fix the errors. You may need to re-upload the changed Python file to Cloud Storage and clone the failed job to resubmit.

## Task 9. Explore inserted rows with SQL

1. In a new browser tab, open **SQL** (in the Databases section).
2. Click **rentals** to view details related to your Cloud SQL instance.
3. Under **Connect to this instance** section, click **Open Cloud Shell**. This will start a new Cloud Shell tab. In the Cloud Shell tab press **ENTER**.

It will take a few minutes to allow your IP for the incoming connection.

1. When prompted, type the root password you configured, then press **ENTER**.
2. At the mysql prompt, type:

USE recommendation\_spark;

SELECT COUNT(\*) AS count FROM Recommendation;

Copied!

content\_copy

If you are getting an Empty Set (0) - wait for your Dataproc job to complete. If it's been more than 5 minutes, your job has likely failed and will require troubleshooting.

Tip: You can use the up arrow in Cloud Shell to return your previous command (or query in this case)

1. Find the recommendations for a user:

SELECT

r.userid,

r.accoid,

r.prediction,

a.title,

a.location,

a.price,

a.rooms,

a.rating,

a.type

FROM Recommendation as r

JOIN Accommodation as a

ON r.accoid = a.id

WHERE r.userid = 10;

Copied!

content\_copy

1. Your result should be similar to the below result:

+--------+--------+------------+-----------------------------+...

| userid | accoid | prediction | title |...

+--------+--------+------------+-----------------------------+...

| 10 | 41 | 1.7748766 | Big Calm Manor |...

| 10 | 21 | 1.7174504 | Big Peaceful Cabin |...

| 10 | 46 | 1.7159091 | Colossal Private Castle |...

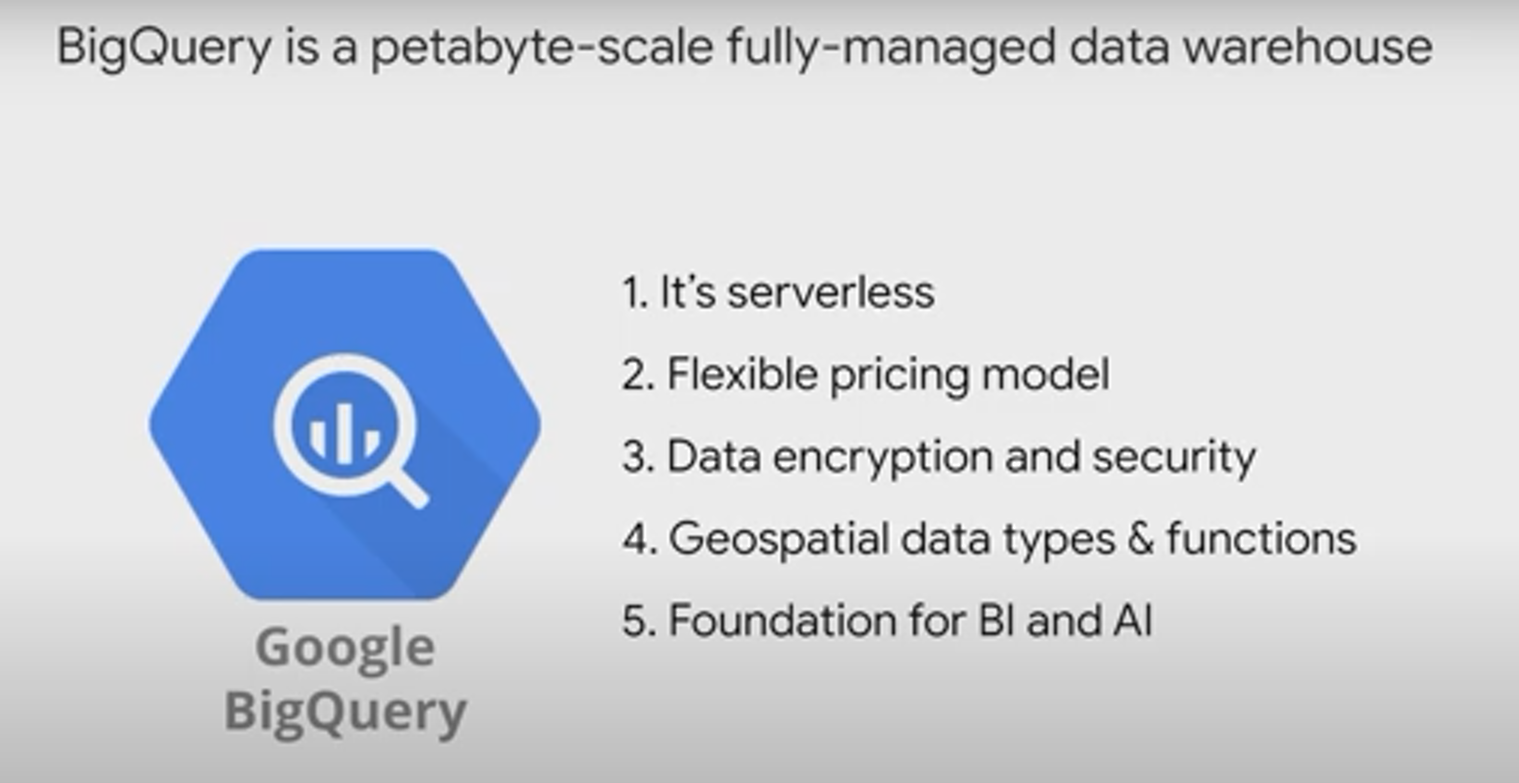
| 10 | 31 | 1.5783813 | Colossal Private Castle |...

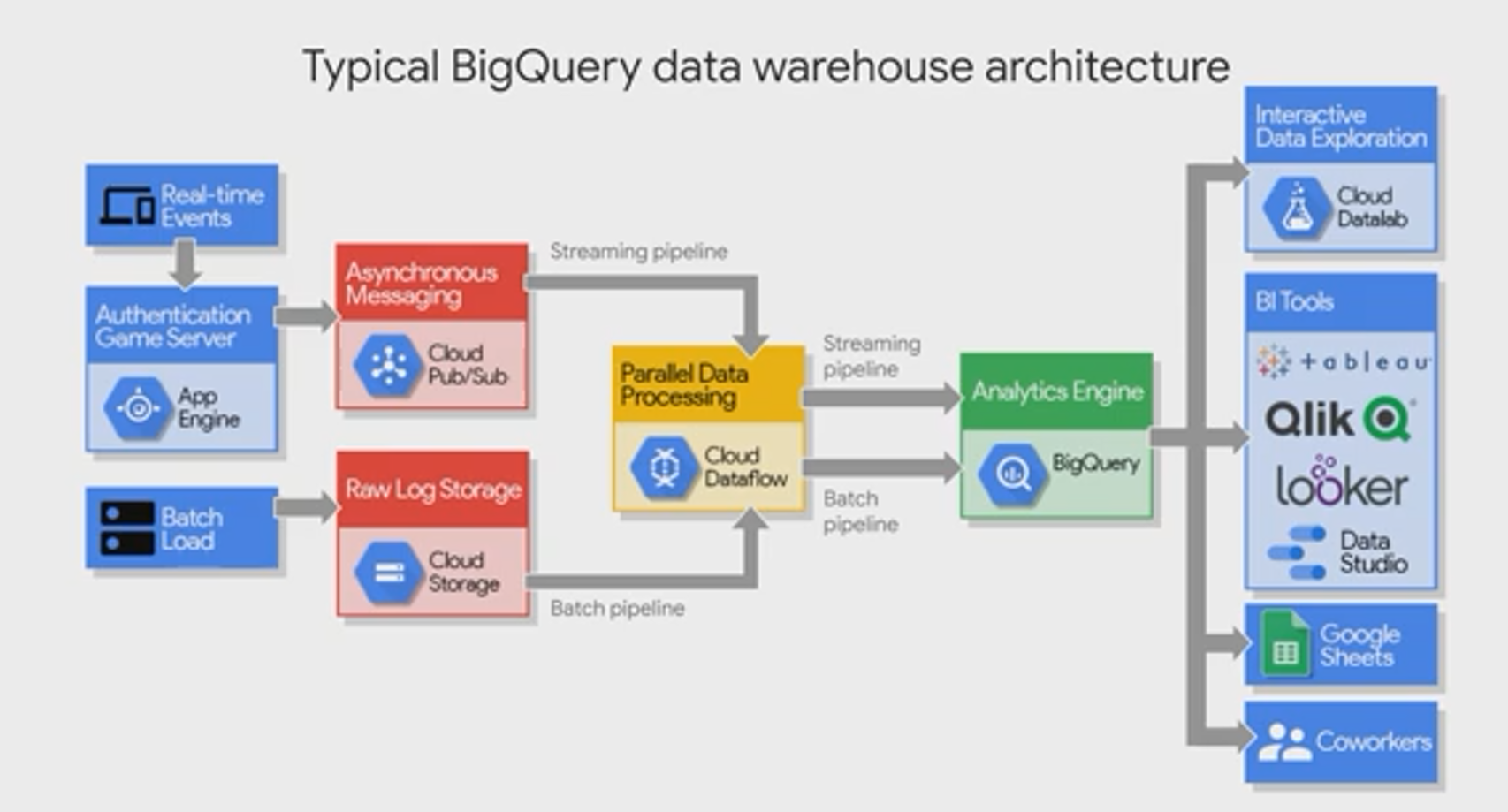
| 10 | 32 | 1.5584077 | Immense Private Hall |...

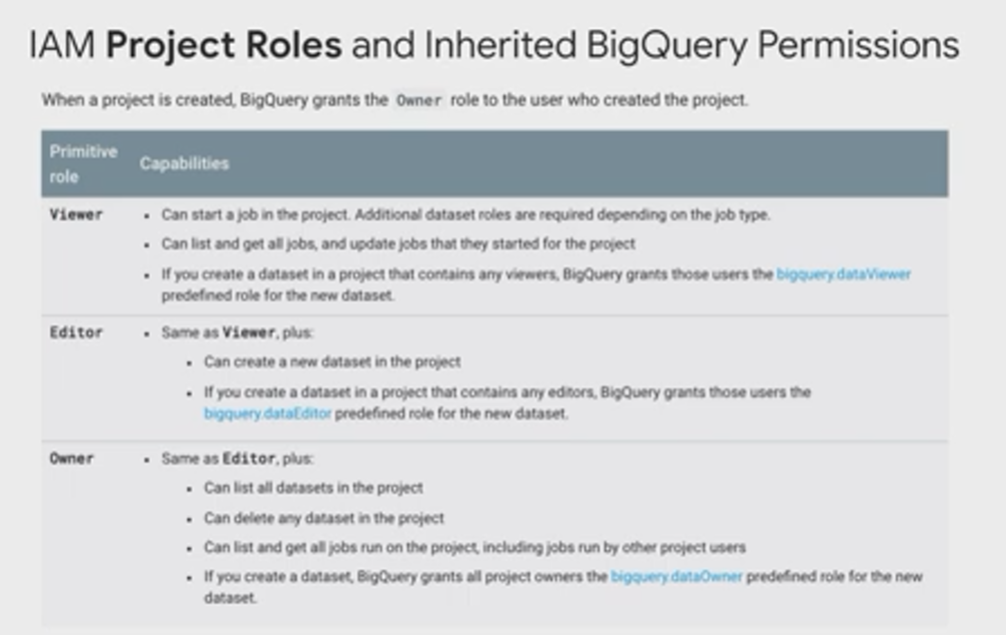
+--------+--------+------------+-----------------------------+...

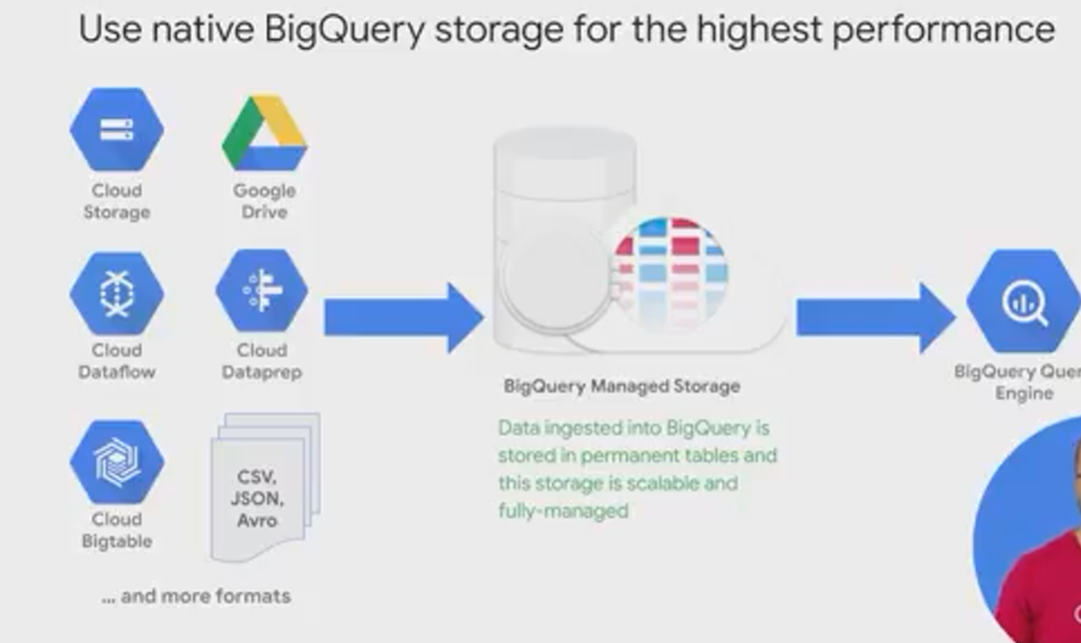
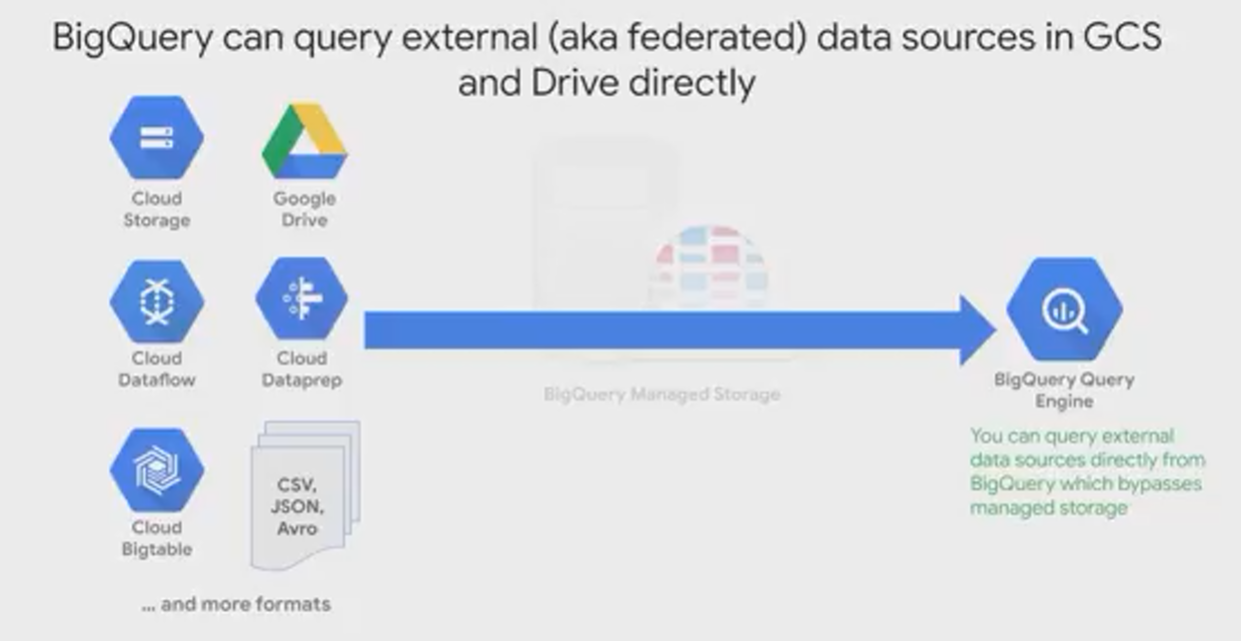
These are the five accommodations that you would recommend. Note that the quality of the recommendations is not great because the dataset was so small (note that the predicted ratings are not very high). Still, this lab illustrates the process you'd go through to create product recommendations.

# Big Query ML







# Syntax to connect to database:

gcloud sql connect rentals --user=root --quiet