Cloud Spanner - Loading Data and Performing Backups

Activate Cloud Shell

Cloud Shell is a virtual machine that is loaded with development tools. It offers a persistent 5GB home directory and runs on the Google Cloud. Cloud Shell provides command-line access to your Google Cloud resources.

1. Click **Activate Cloud Shell** at the top of the Google Cloud console.

2. Click through the following windows:

   - Continue through the Cloud Shell information window.
   - Authorize Cloud Shell to use your credentials to make Google Cloud API calls.

When you are connected, you are already authenticated, and the project is set to your **Project_ID**, `qwiklabs-gcp-00-411b2d8930b2`. The output contains a line that declares the **Project_ID** for this session:

```
Your Cloud Platform project in this session is set to qwiklabs-gcp-00-
411b2d8930b2
```

`gcloud` is the command-line tool for Google Cloud. It comes pre-installed on Cloud Shell and supports tab-completion.

3. (Optional) You can list the active account name with this command:

```
gcloud auth list
```

4. Click **Authorize**.

Output:

```
ACTIVE: *
ACCOUNT: student-03-3dc6140ccdff@qwiklabs.net

To set the active account, run:
    $ gcloud config set account `ACCOUNT`
```

5. (Optional) You can list the project ID with this command:

```
gcloud config list project
```

Output:

```
[core]
project = qwiklabs-gcp-00-411b2d8930b2
```

Note: For full documentation of `gcloud`, in Google Cloud, refer to the gcloud CLI overview guide.

Task 1. Explore the instance

During deployment, a Cloud Spanner instance, database, and table were created for you.

1. From the Console, open the navigation menu (☰) > **View All Products**. Under **Databases** section, click **Spanner**.

The instance name is **banking-instance**, click on it to explore the databases. The associated database is named **banking-db**. Click on it to explore and you will see there is already a table named **Customer**. Click on it and you will be able to check the schema.

2. The table is currently empty. Navigate back to `banking-db` overview page. On the left menu, click **Spanner Studio** and then run the following:

```
SELECT * FROM Customer;
```

3. No results are returned.

Task 2. Insert data with DML

The easiest way to insert data into Spanner is via DML. Using the cloud shell and **gcloud** you can run any DML statement, including **INSERT**.

1. In Cloud Shell, run the following command:

```
gcloud spanner databases execute-sql banking-db --instance=banking-
instance \
 --sql="INSERT INTO Customer (CustomerId, Name, Location) VALUES
('bdaaaa97-1b4b-4e58-b4ad-84030de92235', 'Richard Nelson', 'Ada Ohio')"
```

2. Return to the Console, on the left menu click **Overview**. Navigate to the **Customer** table and select **Data**, you will see the row you just inserted.

As mentioned before, you can use **gcloud** to run any DML command. Check the documentation for DML and Spanner.
Of course, loading a database row by row is not very efficient.

Task 3. Insert data through a client library

The optimal way to access Spanner is via a programmatic interface. There are a wide variety of client libraries including **C++, C#, Go, Java, Node.js, PHP, Python and Ruby**.

1. In the Cloud Shell enter the following command to invoke the **Nano** text editor and create a new empty configuration file named **insert.py**.

```
nano insert.py
```

2. Paste the code block listed below.

```
from google.cloud import spanner
from google.cloud.spanner_v1 import param_types

INSTANCE_ID = "banking-instance"
DATABASE_ID = "banking-db"

spanner_client = spanner.Client()
instance = spanner_client.instance(INSTANCE_ID)
database = instance.database(DATABASE_ID)

def insert_customer(transaction):
    row_ct = transaction.execute_update(
        "INSERT INTO Customer (CustomerId, Name, Location)"
        "VALUES ('b2b4002d-7813-4551-b83b-366ef95f9273', 'Shana 
Underwood', 'Ely Iowa')"
    )
    print("{} record(s) inserted.".format(row_ct))

database.run_in_transaction(insert_customer)
```

3. Press **Ctrl+X** to exit Nano, **Y** to confirm the update, and press **Enter** to save your changes.

4. Run the python code.

```
python3 insert.py
```

5. Refresh the Cloud Console, or click on a different item on the left menu and then click again on **Data** and you will see the new row in your database.

Like with **gcloud**, you can run any DML statement from the client libraries. You can find multiple examples for all the different languages in the documentation.
This is more flexible than loading data using **gcloud**, but still has limitations when loading a source containing a large number of rows.

Task 4. Insert batch data through a client library

A more optimal way to load data into Spanner is doing so in batches. All of the client libraries support batch loading. This example uses Python.

1. In the Cloud Shell enter the following command to invoke the **Nano** text editor and create a new empty configuration file named **batch_insert.py**.

```
nano batch_insert.py
```

2. Paste the code block listed below.

```
from google.cloud import spanner
from google.cloud.spanner_v1 import param_types

INSTANCE_ID = "banking-instance"
DATABASE_ID = "banking-db"

spanner_client = spanner.Client()
instance = spanner_client.instance(INSTANCE_ID)
database = instance.database(DATABASE_ID)

with database.batch() as batch:
    batch.insert(
        table="Customer",
        columns=("CustomerId", "Name", "Location"),
        values=[
        ('edfc683f-bd87-4bab-9423-01d1b2307c0d', 'John Elkins', 'Roy
Utah'),
        ('1f3842ca-4529-40ff-acdd-88e8a87eb404', 'Martin Madrid', 'Ames
Iowa'),
        ('3320d98e-6437-4515-9e83-137f105f7fbc', 'Theresa Henderson',
'Anna Texas'),
        ('6b2b2774-add9-4881-8702-d179af0518d8', 'Norma Carter', 'Bend
Oregon'),

        ],
    )

print("Rows inserted")
```

3. Press **Ctrl+X** to exit Nano, **Y** to confirm the update, and press **Enter** to save your changes.

4. Run the python code.

```
python3 batch_insert.py
```

5. Go back to Cloud Console, refresh to see the new data you just inserted.

The batch method is more efficient, since it's run as a single request. Only one client-server round trip is needed, reducing latency.

However this is a very slow and resource consuming method to load data.

Task 5. Load data using Dataflow

Dataflow is a Google Cloud service for streaming and batch data processing at large scale. Dataflow uses multiple workers to run data processing in parallel. The way in which data is processed is defined using **pipelines** that transform data from its origin (**sources**) to its destination (**sinks**).

There are connectors for **Spanner** that allow you to connect a database as a **source** or a **sink** in Dataflow.

In order to load big amounts of data, you can use the serverless distributed power of **Dataflow** to read data from a source (for example, a CSV file in **Google Cloud Storage**) and load it into your **Spanner** database using a sink connector.

1. To prepare for the Dataflow job, in the Cloud Shell run these commands to create a bucket in your project and a folder with an empty file inside it.

```
gsutil mb gs://qwiklabs-gcp-00-411b2d8930b2
touch emptyfile
gsutil cp emptyfile gs://qwiklabs-gcp-00-411b2d8930b2/tmp/emptyfile
```

2. To ensure that the proper APIs and permissions are set, execute the following block of code in the Cloud Shell.

```
gcloud services disable dataflow.googleapis.com --force
gcloud services enable dataflow.googleapis.com
```

3. From the Console, open the navigation menu (☰) > **View All Products**. Under **Analytics** section, click **Dataflow**.

4. On the top of the screen, click **Create Job From Template**.

5. Place the following values in the template:

   - **Job Name**: spanner-load

   - **Regional endpoint**: `us-central1`

6. Scroll down the **Dataflow template** selector and you will see all the different blueprints you can use with Dataflow. Of course, you can also create your own tailored pipelines, using the Beam SDK.

There are two main types of templates:

- **Stream** will create a pipeline for data that is flowing and is processed continuously (for example, online orders from a website).
- **Batch** will process a dataset that has a beginning and an end (for example, files stored in Google Cloud Storage).

In your scenario, you will load data into Spanner banking database from a CSV file with over 150,000 rows.

7. Select the **Text Files on Cloud Storage to Cloud Spanner** template.

8. Place the following values in the template:

Item	**Value**
Cloud Spanner Instance Id	**banking-instance**
Cloud Spanner Database Id	**banking-db**
Text Import Manifest file	**cloud-training/OCBL372/manifest.json**

The `manifest.json` file format is explained in the tutorial for this template (you can access it by clicking **open tutorial** just above the parameter input fields).
The manifest file must be stored in a Google Cloud Storage bucket that Dataflow can access to and read from. For this lab, this is the content of **manifest.json**:

```
{
    "tables": [
        {
            "table_name": "Customer",
            "file_patterns": [
                "gs://cloud-training/OCBL372/Customer_List.csv"
            ],
            "columns": [
                {"column_name" : "CustomerId", "type_name" : "STRING"
},
                {"column_name" : "Name", "type_name" : "STRING" },
                {"column_name" : "Location", "type_name" : "STRING" }
            ]
        }
    ]
}
```

The manifest file specifies the table, name and type of the columns (in the order that they appear in the CSV file), and the CSV file itself, which is also stored in a Google Cloud Storage bucket.

This is what the CSV file looks like:

```
9d238899-8348-4642-9c00-77dc4481145b,Nicole Anderson,Ada Ohio
360ecaa6-9ec3-4fa0-81a5-3b0dc629e1fa,Ellen Richardson,Ada Ohio
8ee6c2ea-923b-45db-8d51-7f8e7a117af0,Wendy Daniel,Ada Ohio
1d7112cc-c1ee-414f-9325-95c97f9a25d3,Virginia Beasley,Ada Ohio
...
```

9. For the **Temporary Location** parameter input the following value:

`qwiklabs-gcp-00-411b2d8930b2/tmp`

Copied!

content_copy

10. Expand **Optional Parameters**.

11. Uncheck **Use default machine type**.

12. Under **General purpose**, choose the following:

    - Series: **E2**
    - Machine type: **e2-medium (2 vCPU, 4 GB memory)**

13. Click **Run Job** to start the pipeline.

14. The process will take around 12 to 16 minutes. You will see Dataflow go through multiple stages, first starting up the workers and analyzing the pipeline from the template. Then it will read the manifest file and will start processing the CSV file.

Note: If your pipeline fails with an error related to worker nodes not being provisioned, create a new job with the same name from the same template starting from Step 4. This time choose a different Regional endpoint in the United States. For example if Step 5 lists **"us-east4"** as your Regional endpoint try **"us-east1"** for your second attempt.

Wait until Dataflow finishes processing before proceeding. It will have a status of **Succeeded** when complete.

15. Go back to **Spanner** by selecting it in the left menu on Cloud Console. Navigate to the **Customer** table and select **Data**. You will see all the new rows that have been loaded using Dataflow.

16. Navigate back to `banking-db` overview page. On the left menu, click **Spanner Studio** and run the following to see the total number of rows in the **Customer** table:

```
SELECT COUNT(*) FROM Customer;
```

With **Dataflow** templates it is easy (and quick!) to load big amounts of data. You can load dumps from other databases, and load not only CSV but also Avro files following the same procedure. You can even run the process the other way around, using your Spanner database as a source in **Dataflow** to export the data in CSV or Avro.