This tutorial walks you through the following steps using the Cloud Spanner client library for Python:

- Create a Cloud Spanner instance and database.
- Write, read, and execute SQL queries on data in the database.
- Update the database schema.
- Update data using a read-write transaction.
- Add a secondary index to the database.
- Use the index to read and execute SQL queries on data.
- Retrieve data using a read-only transaction.

This tutorial uses Cloud Spanner, which is a billable component of the Google Cloud. For information on the cost of using Cloud Spanner, see Pricing (/spanner/pricing).

1. Complete the steps described in Set up (/spanner/docs/getting-started/set-up), which covers creating and setting a default Google Cloud project, enabling billing, enabling the Cloud Spanner API, and setting up OAuth 2.0 to get authentication credentials to use the Cloud Spanner API.
   In particular, ensure that you run `gcloud auth application-default login` (/sdk/gcloud/reference/auth/application-default/login) to set up your local development environment with authentication credentials.
2. Follow the instructions in Setting Up a Python Development Environment (/python/setup).

3. Clone the sample app repository to your local machine:

   Alternatively, you can download the sample (https://github.com/GoogleCloudPlatform/python-docs-samples/archive/master.zip) as a zip file and extract it.

4. Change to the directory that contains the Cloud Spanner sample code:

5. Create an isolated Python environment, and install dependencies:

When you first use Cloud Spanner, you must create an instance, which is an allocation of resources that are used by Cloud Spanner databases. When you create an instance, you choose an *instance configuration*, which determines where your data is stored, and also the number of nodes to use, which determines the amount of serving and storage resources in your instance.

Execute the following command to create a Cloud Spanner instance in the region `us-central1` with 1 node:

Note that this creates an instance with the following characteristics:

- Instance ID `test-instance`
- Display name `Test Instance`
- Instance configuration `regional-us-central1` (Regional configurations store data in one region, while multi-region configurations distribute data across multiple regions. Learn more in Instances (/spanner/docs/instances).)
- Node count of 1 (`node_count` corresponds to the amount of serving and storage resources available to databases in the instance. Learn more in Node count (/spanner/docs/instances#node_count).)

You should see:

The samples repo contains a sample that shows how to use Cloud Spanner with Python.

Take a look through the `snippets.py` file, which shows how to use Cloud Spanner. The code shows how to create and use a new database. The data uses the example schema shown in the Schema and data model (/spanner/docs/schema-and-data-model#creating-interleaved-tables) page.

Create a database called `example-db` in the instance called `test-instance` by running the following at the command line.

You should see:

You have just created a Cloud Spanner database. The following is the code that created the database.

://github.com/GoogleCloudPlatform/python-docs-samples/blob/master/spanner/cloud-client/snippets.py)

The code also defines two tables, `Singers` and `Albums`, for a basic music application. These tables are used throughout this page. Take a look at the example schema (/spanner/docs/schema-and-data-model#creating-interleaved-tables) if you haven't already.

The next step is to write data to your database.

Before you can do reads or writes, you must create a `Client` (https://googleapis.github.io/google-cloud-python/latest/spanner/client-api.html). You can think of a `Client` as a database connection: all of your interactions with Cloud Spanner must go through a `Client`. Typically you create a `Client` when your application starts up, then you re-use that `Client` to read, write, and execute transactions. The following code shows how to create a client.

/github.com/GoogleCloudPlatform/python-docs-samples/blob/master/spanner/cloud-client/quickstart.py)

Read more in the `Client` (https://googleapis.github.io/google-cloud-python/latest/spanner/client-api.html) reference.

You can insert data using Data Manipulation Language (DML) in a read-write transaction.

You use the `execute_update()` method to execute a DML statement.

://github.com/GoogleCloudPlatform/python-docs-samples/blob/master/spanner/cloud-client/snippets.py)

Run the sample using the `insert_with_dml` argument.

You should see:

You can also insert data using mutations (/spanner/docs/modify-mutation-api).

You write data using a `Batch` (https://googleapis.github.io/google-cloud-python/latest/spanner/batch-api.html) object. A `Batch` object is a container for mutation operations. A mutation represents a sequence of inserts, updates, and deletes that Cloud Spanner applies atomically to different rows and tables in a Cloud Spanner database.

The `insert()` (https://googleapis.github.io/google-cloud-python/latest/spanner/batch-usage.html#inserting-records-using-a-batch) method in the `Batch` class adds one or more insert mutations to the batch. All mutations in a single batch are applied atomically.

This code shows how to write the data using mutations:

://github.com/GoogleCloudPlatform/python-docs-samples/blob/master/spanner/cloud-client/snippets.py)

Run the sample using the `insert_data` argument.

You should see:

Cloud Spanner supports a native SQL interface for reading data, which you can access on the command line using the `gcloud` command-line tool or programmatically using the Cloud Spanner client library for Python.

Execute the following SQL statement to read the values of all columns from the `Albums` table:

Note: See SQL syntax (/spanner/docs/query-syntax) for the Cloud Spanner SQL reference.

The result should be:

In addition to executing a SQL statement on the command line, you can issue the same SQL statement programmatically using the Cloud Spanner client library for Python.

Use the `execute_sql()` (https://googleapis.github.io/google-cloud-python/latest/spanner/snapshot-api.html#google.cloud.spanner_v1.snapshot.Snapshot.execute_sql) method of a `Snapshot` (https://googleapis.github.io/google-cloud-python/latest/spanner/snapshot-api.html) object to run the SQL query. To get a `Snapshot` object, call the `snapshot()` (https://googleapis.github.io/google-cloud-python/latest/spanner/database-api.html#google.cloud.spanner_v1.database.Database.snapshot) method of the `Database` (https://googleapis.github.io/google-cloud-python/latest/spanner/database-api.html) class in a `with` statement.

Here's how to issue the query and access the data:

://github.com/GoogleCloudPlatform/python-docs-samples/blob/master/spanner/cloud-client/snippets.py)

Run the sample using the `query_data` argument.

You should see the following result:

You can include custom values in SQL statements using parameters. Here is an example of using `@lastName` as a parameter in the `WHERE` clause to query records containing a specific value for `LastName`.

://github.com/GoogleCloudPlatform/python-docs-samples/blob/master/spanner/cloud-client/snippets.py)

Run the sample using the query_data_with_parameter argument.

You should see the following result:

In addition to Cloud Spanner's SQL interface, Cloud Spanner also supports a read interface.

Use the `read()` (https://googleapis.github.io/google-cloud-python/latest/spanner/snapshot-api.html#google.cloud.spanner_v1.snapshot.Snapshot.read) method of a `Snapshot` (https://googleapis.github.io/google-cloud-python/latest/spanner/snapshot-api.html) object to read rows from the database. To get a `Snapshot` object, call the `snapshot()`

(https://googleapis.github.io/google-cloud-python/latest/spanner/database-api.html#google.cloud.spanner_v1.database.Database.snapshot)
method of the `Database` (https://googleapis.github.io/google-cloud-python/latest/spanner/database-api.html) class in a `with` statement. Use a `KeySet` (https://googleapis.github.io/google-cloud-python/latest/spanner/keyset-api.html) object to define a collection of keys and key ranges to read.

Here's how to read the data:

://github.com/GoogleCloudPlatform/python-docs-samples/blob/master/spanner/cloud-client/snippets.py)

Run the sample using the `read_data` argument.

You should see output similar to:

Assume you need to add a new column called `MarketingBudget` to the `Albums` table. Adding a new column to an existing table requires an update to your database schema. Cloud Spanner supports schema updates to a database while the database continues to serve traffic. Schema updates do not require taking the database offline and they do not lock entire tables or columns; you can continue writing data to the database during the schema update. Read more about supported schema updates and schema change performance in Schema updates (/spanner/docs/schema-updates).

You can add a column on the command line using the `gcloud` command-line tool or programmatically using the Cloud Spanner client library for Python.

Use the following `ALTER TABLE` (/spanner/docs/data-definition-language#alter_table) command to add the new column to the table:

You should see:

Use the `update_ddl()` (https://googleapis.github.io/google-cloud-python/latest/spanner/database-api.html#google.cloud.spanner_v1.database.Database.update_ddl) method of the `Database` (https://googleapis.github.io/google-cloud-python/latest/spanner/database-api.html) class to modify the schema:

://github.com/GoogleCloudPlatform/python-docs-samples/blob/master/spanner/cloud-client/snippets.py)

Run the sample using the `add_column` argument.

You should see:

The following code writes data to the new column. It sets `MarketingBudget` to `100000` for the row keyed by `Albums(1, 1)` and to `500000` for the row keyed by `Albums(2, 2)`.

://github.com/GoogleCloudPlatform/python-docs-samples/blob/master/spanner/cloud-client/snippets.py)

Run the sample using the `update_data` argument.

You can also execute a SQL query or a read call to fetch the values that you just wrote.

Here's the code to execute the query:

://github.com/GoogleCloudPlatform/python-docs-samples/blob/master/spanner/cloud-client/snippets.py)

To execute this query, run the sample using the `query_data_with_new_column` argument.

You should see:

You can update data using DML in a read-write transaction.

You use the `execute_update()` method to execute a DML statement.

://github.com/GoogleCloudPlatform/python-docs-samples/blob/master/spanner/cloud-client/snippets.py)

Run the sample using the `write_with_dml_transaction` argument.

You should see:

Note: You can also update data using mutations (/spanner/docs/modify-mutation-api#updating_rows_in_a_table).

Suppose you wanted to fetch all rows of `Albums` that have `AlbumTitle` values in a certain range. You could read all values from the `AlbumTitle` column using a SQL statement or a read call,

and then discard the rows that don't meet the criteria, but doing this full table scan is expensive, especially for tables with a lot of rows. Instead you can speed up the retrieval of rows when searching by non-primary key columns by creating a secondary index (/spanner/docs/secondary-indexes) on the table.

Adding a secondary index to an existing table requires a schema update. Like other schema updates, Cloud Spanner supports adding an index while the database continues to serve traffic. Cloud Spanner automatically backfills the index with your existing data. Backfills might take a few minutes to complete, but you don't need to take the database offline or avoid writing to the indexed table during this process. For more details, see index backfilling (/spanner/docs/secondary-indexes#adding_an_index).

After you add a secondary index, Cloud Spanner automatically uses it for SQL queries that are likely to run faster with the index. If you use the read interface, you must specify the index that you want to use.

You can add an index on the command line using the **gcloud** command line tool or programmatically using the Cloud Spanner client library for Python.

Use the following `CREATE INDEX` (/spanner/docs/data-definition-language#create_index) command to add an index to the database:

You should see:

Use the `update_ddl()` (https://googleapis.github.io/google-cloud-python/latest/spanner/database-api.html#google.cloud.spanner_v1.database.Database.update_ddl) method of the `Database` (https://googleapis.github.io/google-cloud-python/latest/spanner/database-api.html) class to add an index:

://github.com/GoogleCloudPlatform/python-docs-samples/blob/master/spanner/cloud-client/snippets.py)

Run the sample using the `add_index` argument.

Adding an index can take a few minutes. After the index is added, you should see:

For SQL queries, Cloud Spanner automatically uses an appropriate index. In the read interface, you must specify the index in your request.

To use the index in the read interface, provide an `Index` argument to the `read()` (https://googleapis.github.io/google-cloud-python/latest/spanner/snapshot-api.html#google.cloud.spanner_v1.snapshot.Snapshot.read) method of a `Snapshot` (https://googleapis.github.io/google-cloud-python/latest/spanner/snapshot-api.html) object. To get a `Snapshot` object, call the `snapshot()` (https://googleapis.github.io/google-cloud-python/latest/spanner/database-api.html#google.cloud.spanner_v1.database.Database.snapshot) method of the `Database` (https://googleapis.github.io/google-cloud-python/latest/spanner/database-api.html) class in a `with` statement.

://github.com/GoogleCloudPlatform/python-docs-samples/blob/master/spanner/cloud-client/snippets.py)

Run the sample using the `read_data_with_index` argument.

You should see:

You might have noticed that the read example above did not include reading the `MarketingBudget` column. This is because Cloud Spanner's read interface does not support the ability to join an index with a data table to look up values that are not stored in the index.

Create an alternate definition of `AlbumsByAlbumTitle` that stores a copy of `MarketingBudget` in the index.

Adding an index can take a few minutes. After the index is added, you should see:

Use the `update_ddl()` (https://googleapis.github.io/google-cloud-python/latest/spanner/database-api.html#google.cloud.spanner_v1.database.Database.update_ddl)

method of the `Database` (https://googleapis.github.io/google-cloud-python/latest/spanner/database-api.html) class to add an index with a `STORING` clause:

://github.com/GoogleCloudPlatform/python-docs-samples/blob/master/spanner/cloud-client/snippets.py)

Run the sample using the `add_storing_index` argument.

You should see:

Now you can execute a read that fetches all `AlbumId`, `AlbumTitle`, and `MarketingBudget` columns from the `AlbumsByAlbumTitle2` index:

://github.com/GoogleCloudPlatform/python-docs-samples/blob/master/spanner/cloud-client/snippets.py)

Run the sample using the `read_data_with_storing_index` argument.

You should see output similar to:

Suppose you want to execute more than one read at the same timestamp. Read-only transactions (/spanner/docs/transactions#read-only_transactions) observe a consistent prefix of the transaction commit history, so your application always gets consistent data. Use a `Snapshot` (https://googleapis.github.io/google-cloud-python/latest/spanner/snapshot-api.html) object for executing read-only transactions. To get a `Snapshot` object, call the `snapshot()` (https://googleapis.github.io/google-cloud-python/latest/spanner/database-api.html#google.cloud.spanner_v1.database.Database.snapshot) method of the `Database` (https://googleapis.github.io/google-cloud-python/latest/spanner/database-api.html) class in a `with` statement.

The following shows how to run a query and perform a read in the same read-only transaction:

://github.com/GoogleCloudPlatform/python-docs-samples/blob/master/spanner/cloud-client/snippets.py)

Run the sample using the `read_only_transaction` argument.

You should see output similar to:

To avoid incurring additional charges to your Google Cloud account for the resources used in this tutorial, drop the database and delete the instance that you created.

If you delete an instance, all databases within it are automatically deleted. This step shows how to delete a database without deleting an instance (you would still incur charges for the instance).

1. Go to the **Cloud Spanner Instances** page in the Google Cloud Console.

   Go to the Instances page (https://console.cloud.google.com/spanner/instances)

2. Click the instance.
3. Click the database that you want to delete.
4. In the **Database details** page, click **Delete**.
5. Confirm that you want to delete the database and click **Delete**.

Deleting an instance automatically drops all databases created in that instance.

1. Go to the **Cloud Spanner Instances** page in the Google Cloud Console.

   Go to the Instances page (https://console.cloud.google.com/spanner/instances)

2. Click your instance.
3. Click **Delete**.
4. Confirm that you want to delete the instance and click **Delete**.

- Access Cloud Spanner in a virtual machine instance (/spanner/docs/configure-virtual-machine-instance): create a virtual machine instance with access to your Cloud Spanner database.
- Learn about authorization and authentication credentials in Getting started with authentication (/docs/authentication/getting-started).
- Learn more Cloud Spanner concepts (/spanner/docs/concepts).