This tutorial walks you through the following steps using the Cloud Spanner API with REST:

- Create a Cloud Spanner instance and database.
- Write, read, and execute SQL queries on data in the database.
- Update the database schema.
- 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.

If you want to use Cloud Spanner client libraries instead of using the REST API, see Tutorials (/spanner/docs/tutorials).

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. Sign in (https://accounts.google.com/Login) to your Google Account.

   If you don't already have one, sign up for a new account (https://accounts.google.com/SignUp).
2. In the Cloud Console, on the project selector page, select or create a Cloud project.

★ **Note**: If you don't plan to keep the resources that you create in this procedure, create a project instead of selecting an existing project. After you finish these steps, you can delete the project, removing all resources associated with the project.

Go to the project selector page (https://console.cloud.google.com/projectselector2/home/dashboard)

3. Make sure that billing is enabled for your Google Cloud project. Learn how to confirm billing is enabled for your project (/billing/docs/how-to/modify-project).

You can make Cloud Spanner REST calls using:

- The **Try-It!** feature found in the Cloud Spanner API reference documentation (/spanner/docs/reference/rest/). The examples shown on this page use the **Try-It!** feature.
- Google APIs Explorer (https://developers.google.com/explorer-help/), which contains the Cloud Spanner API (https://developers.google.com/apis-explorer/#p/spanner.googleapis.com/v1/) and other Google APIs.
- Other tools or frameworks that support HTTP REST calls.

- The examples use `[PROJECT_ID]` as the Google Cloud project ID. Substitute your Google Cloud project ID for `[PROJECT_ID]`. (Do not include `[` and `]` in your project ID.)
- The examples create and use an instance ID of `test-instance`. Substitute your instance ID if you are not using `test-instance`.
- The examples create and use a database ID of `example-db`. Substitute your database ID if you are not using `example-db`.
- The examples use `[SESSION]` as part of a session name. Substitute the value you receive when you create a session (#create_a_session) for `[SESSION]`. (Do not include `[` and `]` in your session name.)
- The examples use a transaction ID of `[TRANSACTION_ID]`. Substitute the value you receive when you create a transaction for `[TRANSACTION_ID]`. (Do not include `[` and `]` in your transaction ID.)
- The **Try-It!** functionality supports interactively adding individual HTTP request fields. Most examples in this topic provide the entire request instead of describing how to interactively

add individual fields to the request.

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 where your data is stored and how many nodes are used for your data.

When you create an instance, you specify an *instance configuration*, which defines the geographic placement and replication of your databases in that instance. You can choose a regional configuration, which stores data in one region, or a multi-region configuration, which distributes data across multiple regions. Learn more in Instances (/spanner/docs/instances).

Use `projects.instanceConfigs.list` to determine which configurations are available for your Google Cloud project.

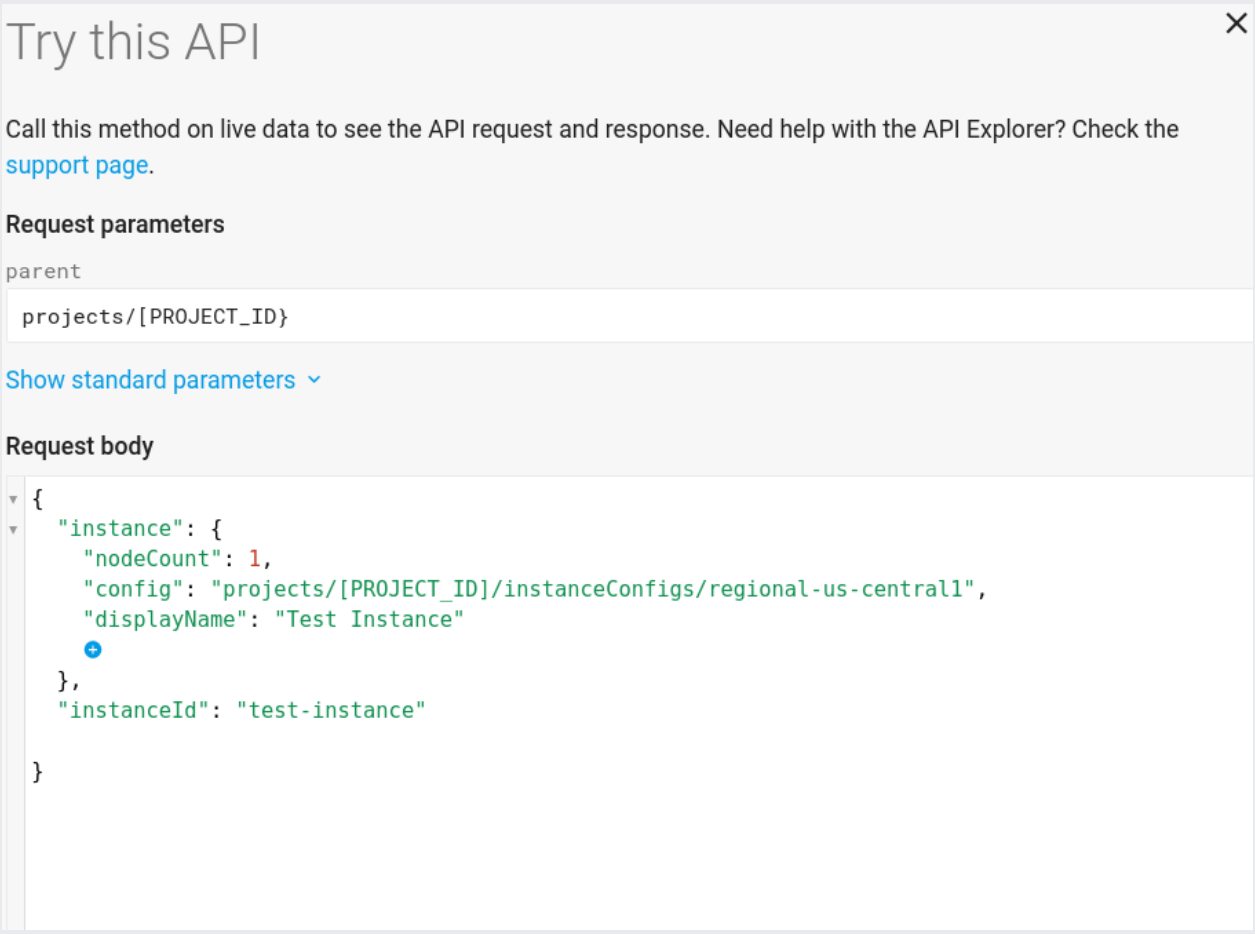
1. Click `projects.instanceConfigs.list` (/spanner/docs/reference/rest/v1/projects.instanceConfigs/list#try-it).
2. For **parent**, enter:
3. Click **Execute**. The available instance configurations are shown in the response. Here's an example response (your project may have different instance configurations):

You use the `name` value for one of the instance configurations when you create your instance.

1. Click `projects.instances.create` (/spanner/docs/reference/rest/v1/projects.instances/create#try-it).
2. For **parent**, enter:

3. Click **Add request body parameters** and select `instance`.
4. Click the hint bubble for **instance** to see the possible fields. Add values for the following fields:
    a. `nodeCount`: Enter `1`.
    b. `config`: Enter the `name` value of one of the regional instance configurations returned when you list instance configurations (#listing_instance_configurations).
    c. `displayName`: Enter `Test Instance`.

5. Click the hint bubble that follows the closing bracket for **instance** and select **instanceId**.
6. For `instanceId`, enter `test-instance`.
   Your **Try It!** instance creation page should now look like this:



7. Click **Execute**. The response returns a long-running operation (/spanner/docs/manage-long-running-operations) which you can query to check its status.

You can list your instances using `projects.instances.list` (/spanner/docs/reference/rest/v1/projects.instances/list).

Create a database named `example-db`.

1. Click `projects.instances.databases.create` (/spanner/docs/reference/rest/v1/projects.instances.databases/create#try-it).
2. For **parent**, enter:

3. Click **Add request body parameters** and select `createStatement`.
4. For `createStatement`, enter:

   (The database name, `example-db`, contains a hyphen, so it must be enclosed in backticks (`).
5. Click **Execute**. The response returns a long-running operation (/spanner/docs/manage-long-running-operations) which you can query to check its status.

You can list your databases using `projects.instances.databases.list` (/spanner/docs/reference/rest/v1/projects.instances.databases/list).

Use Cloud Spanner's Data Definition Language (/spanner/docs/data-definition-language) (DDL) to create, alter, or drop tables, and to create or drop indexes.

1. Click `projects.instances.databases.updateDdl` (/spanner/docs/reference/rest/v1/projects.instances.databases/updateDdl#try-it).
2. For **database**, enter:

3. For **Request body**, use the following:

The `statements` array contains the DDL statements that define the schema.

4. Click **Execute**. The response returns a long-running operation (/spanner/docs/manage-long-running-operations) which you can query to check its status.

The schema 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.

You can retrieve your schema using `projects.instances.databases.getDdl` (/spanner/docs/reference/rest/v1/projects.instances.databases/getDdl).

Before you can add, update, delete, or query data, you must create a session (/spanner/docs/sessions), which represents a communication channel with the Cloud Spanner database service. (You do not directly use a session if you are using a Cloud Spanner client library (/spanner/docs/reference/libraries), because the client library manages sessions on your behalf.)

1. Click `projects.instances.databases.sessions.create` (/spanner/docs/reference/rest/v1/projects.instances.databases.sessions/create#try-it).
2. For **database**, enter:
3. Click **Execute**.
4. The response shows the session that you created, in the form

You will use this session when you read or write to your database.

Sessions are intended to be long-lived. The Cloud Spanner database service can delete a session when the session is idle for more than one hour. Attempts to use a deleted session result in `NOT_FOUND`. If you encounter this error, create and use a new session. You can see if a session is still alive using `projects.instances.databases.sessions.get` (/spanner/docs/reference/rest/v1/projects.instances.databases.sessions/get). For related information, see Keep an idle session alive (/spanner/docs/sessions#keep_an_idle_session_alive).

Note: Sessions are an advanced concept. For more details and best practices, see Sessions (/spanner/docs/sessions).

The next step is to write data to your database.

You write data using the `Mutation` (/spanner/docs/reference/rest/v1/projects.instances.databases.sessions/commit#Mutation) type. A `Mutation` is a container for mutation operations. A `Mutation` represents a sequence of inserts, updates, deletes, and other actions that can be applied atomically to different rows and tables in a Cloud Spanner database.

1. Click `projects.instances.databases.sessions.commit` (/spanner/docs/reference/rest/v1/projects.instances.databases.sessions/commit#try-it).
2. For **session**, enter:

   (You receive this value when you create a session (#create_a_session).)
3. For **Request body**, use the following:

4. Click **Execute**. The response shows the commit timestamp.

This example used `insertOrUpdate`. Other operations (/spanner/docs/reference/rest/v1/projects.instances.databases.sessions/commit#Mutation.FIELDS) for `Mutations` are `insert`, `update`, `replace`, and `delete`.

For information on how to encode data types, see TypeCode (/spanner/docs/reference/rest/v1/ResultSetMetadata#TypeCode).

1. Click `projects.instances.databases.sessions.executeSql` (/spanner/docs/reference/rest/v1/projects.instances.databases.sessions/executeSql#try-it).
2. For **session**, enter:

   (You receive this value when you create a session (#create_a_session).)
3. For **Request body**, use the following:

4. Click **Execute**. The response shows the query results.

1. Click `projects.instances.databases.sessions.read` (/spanner/docs/reference/rest/v1/projects.instances.databases.sessions/read#try-it).
2. For **session**, enter:

   (You receive this value when you create a session (#create_a_session).)
3. For **Request body**, use the following:

4. Click **Execute**. The response shows the read results.

Assume you need to add a new column called `MarketingBudget` to the `Albums` table, which 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.

1. Click `projects.instances.databases.updateDdl` (/spanner/docs/reference/rest/v1/projects.instances.databases/updateDdl#try-it).
2. For **database**, enter:

3. For **Request body**, use the following:

The `statements` array contains the DDL statements that define the schema.

4. Click **Execute**. This may take a few minutes to complete, even after the REST call returns a response. The response returns a long-running operation (/spanner/docs/manage-long-running-operations) which you can query to check its status.

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)`.

1. Click `projects.instances.databases.sessions.commit` (/spanner/docs/reference/rest/v1/projects.instances.databases.sessions/commit#try-it).
2. For **session**, enter:

   (You receive this value when you create a session (#create_a_session).)
3. For **Request body**, use the following:

4. Click **Execute**. The response shows the commit timestamp.

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

Here's how to execute the query:

1. Click `projects.instances.databases.sessions.executeSql` (/spanner/docs/reference/rest/v1/projects.instances.databases.sessions/executeSql#try-it).
2. For **session**, enter:

   (You receive this value when you create a session (#create_a_session).)
3. For **Request body**, use the following:

4. Click **Execute**. As part of the response you should see two rows that contain the updated `MarketingBudget` values:

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 certain tables or columns 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 using `updateDdl`.

1. Click `projects.instances.databases.updateDdl` (/spanner/docs/reference/rest/v1/projects.instances.databases/updateDdl#try-it).
2. For **database**, enter:
3. For **Request body**, use the following:
4. Click **Execute**. This may take a few minutes to complete, even after the REST call returns a response. The response returns a long-running operation (/spanner/docs/manage-long-running-operations) which you can query to check its status.

1. Click `projects.instances.databases.sessions.executeSql` (/spanner/docs/reference/rest/v1/projects.instances.databases.sessions/executeSql#try-it).
2. For **session**, enter:

   (You receive this value when you create a session (#create_a_session).)
3. For **Request body**, use the following:

4. Click **Execute**. As part of the response you should see the following rows:

1. Click `projects.instances.databases.sessions.read` (/spanner/docs/reference/rest/v1/projects.instances.databases.sessions/read#try-it).
2. For **session**, enter:

   (You receive this value when you create a session (#create_a_session).)
3. For **Request body**, use the following:

4. Click **Execute**. As part of the response you should see the following rows:

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.

You can add a STORING index using `updateDdl`.

1. Click `projects.instances.databases.updateDdl` (/spanner/docs/reference/rest/v1/projects.instances.databases/updateDdl#try-it).
2. For **database**, enter:
3. For **Request body**, use the following:
4. Click **Execute**. This may take a few minutes to complete, even after the REST call returns a response. The response returns a long-running operation (/spanner/docs/manage-long-running-operations) which you can query to check its status.

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

1. Click `projects.instances.databases.sessions.read` (/spanner/docs/reference/rest/v1/projects.instances.databases.sessions/read#try-it).
2. For **session**, enter:

   (You receive this value when you create a session (#create_a_session).)
3. For **Request body**, use the following:

4. Click **Execute**. As part of the response you should see the following rows:

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.

1. Click `projects.instances.databases.sessions.beginTransaction` (/spanner/docs/reference/rest/v1/projects.instances.databases.sessions/beginTransaction#try-it).
2. For **session**, enter:
3. For **Request Body**, use the following:
4. Click **Execute**.
5. The response shows the ID of the transaction that you created.

You can now use the read-only transaction to retrieve data at a consistent timestamp, even if the data has changed since you created the read-only transaction.

1. Click `projects.instances.databases.sessions.executeSql` (/spanner/docs/reference/rest/v1/projects.instances.databases.sessions/executeSql#try-it).
2. For **session**, enter:

(You receive this value when you create a session (#create_a_session).)

3. For **Request body**, use the following:

4. Click **Execute**. You should see rows similar to the following in the response:

1. Click `projects.instances.databases.sessions.read` (/spanner/docs/reference/rest/v1/projects.instances.databases.sessions/read#try-it).
2. For **session**, enter:

   (You receive this value when you create a session (#create_a_session).)
3. For **Request body**, use the following:

4. Click **Execute**. You should see rows similar to the following in the response:

Cloud Spanner also supports read-write transactions, which execute a set of reads and writes atomically at a single logical point in time. For more information, see Read-write transactions (/spanner/docs/transactions#read-write_transactions). (The **Try-It!** functionality is not suitable for demonstrating a read-write transaction.)

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.

1. Click `projects.instances.databases.dropDatabase` (/spanner/docs/reference/rest/v1/projects.instances.databases/dropDatabase#try-it).
2. For **name**, enter:
3. Click **Execute**.

1. Click `projects.instances.delete` (/spanner/docs/reference/rest/v1/projects.instances/delete#try-it).
2. For **name**, enter:
3. Click **Execute**.

- 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 more Cloud Spanner concepts (/spanner/docs/concepts).