This page describes how to insert, update, and delete Cloud Spanner data using Data Manipulation Language (DML) statements. You can run DML statements using the client libraries, the Google Cloud Console, and the `gcloud` (/sdk/) command-line tool. You can run Partitioned DML (/spanner/docs/dml-partitioned) statements using the client libraries and the `gcloud` (/sdk/) command-line tool.

For the complete DML syntax reference, see Data Manipulation Language syntax (/spanner/docs/dml-syntax).

DML supports `INSERT`, `UPDATE`, and `DELETE` statements in the Cloud Console, `gcloud` command-line tool, and client libraries.

You execute DML statements inside read-write transactions. When Cloud Spanner reads data, it acquires shared read locks on limited portions of the row ranges that you read. Specifically, it acquires these locks only on the columns you access. The locks can include data that does not match the filter condition of the `WHERE` clause.

When Cloud Spanner modifies data using DML statements, it acquires exclusive locks on the specific data that you are modifying. In addition, it acquires shared locks in the same way as when you read data. If your request includes large row ranges, or an entire table, the shared locks might prevent other transactions from completing in parallel.

To modify data as efficiently as possible, use a `WHERE` clause that enables Cloud Spanner to read only the necessary rows. You can achieve this goal with a filter on the primary key, or on the key of a secondary index. The `WHERE` clause limits the scope of the shared locks and enables Cloud Spanner to process the update more efficiently.

For example, suppose that one of the musicians in the `Singers` table changes their first name, and you need to update the name in your database. You could execute the following DML statement, but it forces Cloud Spanner to scan the entire table and acquires shared locks that cover the entire table.

As a result, Cloud Spanner must read more data than necessary, and concurrent transactions cannot modify the data in parallel:

To make the update more efficient, include the `SingerId` column in the `WHERE` clause. The `SingerId` column is the only primary key column for the `Singers` table:

Cloud Spanner sequentially executes all the SQL statements (`SELECT`, `INSERT`, `UPDATE`, and `DELETE`) within a transaction. They are not executed concurrently. The only exception is that Cloud Spanner might execute multiple `SELECT` statements concurrently, because they are read-only operations.

A transaction that includes DML statements has the same limits (/spanner/quotas#limits_for_creating_reading_updating_and_deleting_data) as any other transaction. If you have large-scale changes, consider using Partitioned DML (/spanner/docs/dml-partitioned).

- If the DML statements in a transaction result in more than 20,000 mutations, the DML statement that pushes the transaction over the limit returns a `BadUsage` error with a message about too many mutations.
- If the DML statements in a transaction result in a transaction that is larger than 100 MB, the DML statement that pushes the transaction over the limit returns a `BadUsage` error with a message about the transaction exceeding the size limit.

Mutations performed using DML are not returned to the client. They are merged into the commit request when it is committed, and they count towards the maximum size limits. Even if the size of the

commit request that you send is small, the transaction might still exceed the allowed size limit.

Use the following steps to execute a DML statement in the Cloud Console.

1. Go to the Cloud Spanner **Instances** page.

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

2. Select your project in the drop-down list in the toolbar.
3. Click the name of the instance that contains your database to go to the **Instance details** page.
4. In the **Overview** tab, click the name of your database. The **Database details** page appears.
5. Click **Query**.
6. Enter a DML statement. For example, the following statement adds a new row to the `Singers` table.

7. Click **Run query**. The Cloud Console displays the result.

When you click **Run query**, the transaction commits automatically.

To execute DML statements, use the `gcloud spanner databases execute-sql` command. The following example adds a new row to the `Singers` table.

To execute DML statements using the client library:

- Create a read-write transaction (/spanner/docs/transactions#read-write_transactions).
- Call the client library method for DML execution and pass in the DML statement.
- Use the return value of the DML execution method to get the number of rows inserted, updated, or deleted.

The following code example inserts a new row into the `Singers` table.

The following code example updates the `MarketingBudget` column of the `Albums` table based on a `WHERE` clause.

The following code example deletes all the rows in the `Singers` table where the `FirstName` column is `Alice`.

The following example uses a `STRUCT` with bound parameters (/spanner/docs/structs#using_struct_objects_as_bound_parameters_in_sql_queries) to update the `LastName` in rows filtered by `FirstName` and `LastName`.

Changes you make using DML statements are visible to subsequent statements in the same transaction. This is different from using mutations (/spanner/docs/modify-mutation-api), where changes are not visible until the transaction commits.

Cloud Spanner checks the constraints after every DML statement. This is different from using mutations, where Cloud Spanner buffers mutations in the client until commit and checks constraints at commit time. Evaluating the constraints after each statement allows Cloud Spanner to guarantee that the data that a DML statement returns is consistent with the schema.

The following example updates a row in the `Singers` table, then executes a `SELECT` statement to print the new values.

You use the `PENDING_COMMIT_TIMESTAMP` (https://cloud.google.com/spanner/docs/functions-and-operators#timestamp-functions) function to write the commit timestamp in a DML statement. Cloud Spanner selects the commit timestamp when the transaction commits.

After you call the `PENDING_COMMIT_TIMESTAMP` method, the table and any derived index is unreadable to any fut atements in the transaction. You must write commit timestamps as the last statement in a transaction to prevent th ility of trying to read the table. If you try to read the table, then Cloud Spanner returns an error.

The following code example uses the `PENDING_COMMIT_TIMESTAMP` (/spanner/docs/functions-and-operators#timestamp-functions) function to write the commit timestamp in the `LastUpdateTime` column.

You can retrieve a query plan (/spanner/docs/sql-best-practices#how-execute-queries) using the Cloud Console, the client libraries, and the `gcloud` command-line tool.

Partitioned DML (/spanner/docs/dml-partitioned) is designed for bulk updates and deletes, particularly periodic cleanup and backfilling.

To execute a Partitioned DML statement, use the `gcloud spanner databases execute-sql` command with the `--enable-partitioned-dml` option. The following example updates rows in the `Albums` table.

Cloud Spanner does not support `--query-mode=PLAN` and `--query-mode=PROFILE` for Partitioned DML.

The following code example updates the `MarketingBudget` column of the `Albums` table.

The following code example deletes rows from the `Singers` table, based on the `SingerId` column.

This feature is intended for use by driver and connector authors. In most cases, you do not need to use batch DML di

If you need to avoid the extra latency incurred from multiple serial requests, use batch DML to send multiple `INSERT`, `UPDATE`, or `DELETE` statements in a single transaction: