Cloud Spanner allows you to make schema updates with no downtime. You can update the schema of an existing database in several ways:

- In the Cloud Console
- Using the `gcloud` (/spanner/docs/gcloud-spanner) command-line tool
- Using the client libraries (/spanner/docs/reference/libraries)
- Using `projects.instances.databases.updateDdl` (/spanner/docs/reference/rest/v1/projects.instances.databases/updateDdl) (REST API)
- Using `UpdateDatabaseDdl` (/spanner/docs/reference/rpc/google.spanner.admin.database.v1#google.spanner.admin.database.v1.DatabaseAdmin.UpdateDatabaseDdl) (RPC API)

Cloud Spanner supports the following schema updates of an existing database:

- Create new tables. Columns in new tables can be `NOT NULL`.
- Delete a table, if no other tables are interleaved within it, and it has no secondary indexes.
- Add a non-key column to any table. New non-key columns can't be `NOT NULL`.
- Add `NOT NULL` to a non-key column, excluding `ARRAY` columns.
- Remove `NOT NULL` from a non-key column.
- Drop a non-key column from any table, unless it is used by a secondary index (/spanner/docs/secondary-indexes).
- Change a `STRING` column to a `BYTES` column or a `BYTES` column to a `STRING` column.
- Increase or decrease the length limit for a `STRING` or `BYTES` type (including to `MAX` (/spanner/docs/data-definition-language)), unless it is a primary key column inherited by one or more child tables.
- Enable or disable commit timestamps (/spanner/docs/commit-timestamp) in value and primary key columns.

- Add or remove any secondary index.

Schema updates in Cloud Spanner do not require downtime. When you issue a batch of DDL statements to a Cloud Spanner database, you can continue writing and reading from the database without interruption while Cloud Spanner applies the update as a long-running operation (/spanner/docs/manage-long-running-operations).

The time it takes to execute a DDL statement depends on whether the update requires validation of the existing data or backfill of any data. For example, if you add the `NOT NULL` annotation to an existing column, Cloud Spanner must read all the values in the column to make sure that the column does not contain any `NULL` values. This step can take a long time if there is a lot of data to validate. Another example is if you're adding an index to a database: Cloud Spanner backfills the index using existing data, and that process can take a long time depending on how the index's definition and the size of the corresponding base table. However, if you add a new column to a table, there is no existing data to validate, so Cloud Spanner can make the update more quickly.

In summary, schema updates that do not require Cloud Spanner to validate existing data can happen in minutes. Schema updates that require validation can take longer, depending on the amount of existing data that needs to be validated, but data validation happens in the background at a lower priority than production traffic. Schema updates that require data validation are discussed in more detail in the next section.

You can make schema updates that require validating that the existing data meets the new constraints. When a schema update requires data validation, Cloud Spanner disallows conflicting schema updates to the affected schema entities and validates the data in the background. If validation is successful, the schema update succeeds. If validation is not successful, the schema update does not succeed. Validation operations are executed as long-running operations (/spanner/docs/manage-long-running-operations). You can check the status of these operations to determine if they succeeded or failed.

For example, suppose you have defined a `Songwriters` table in your schema:

The following schema updates are allowed, but they require validation and might take longer to complete, depending on the amount of existing data:

- Adding the `NOT NULL` annotation to a non-key column. For example:

- Reducing the length of a column. For example:

- Altering from `BYTES` to `STRING`. For example:

- Enabling commit timestamps (/spanner/docs/commit-timestamp#converting_a_timestamp_column_to_a_commit_timestamp_column) on an existing `TIMESTAMP` column. For example:

These schema updates fail if the underlying data does not satisfy the new constraints. For example, the `ALTER TABLE Songwriters ALTER COLUMN Nickname STRING(MAX) NOT NULL` statement above fails if any value in the `Nickname` column is `NULL`, because the existing data does not meet the `NOT NULL` constraint of the new definition.

Data validation can take from several minutes to many hours. The time to complete data validation depends on:

- The size of the data set

- The number of nodes in the instance
- The load on the instance

Some schema updates can change the behavior of requests to the database before the schema update completes. For example, if you're adding `NOT NULL` to a column, Cloud Spanner almost immediately begins rejecting writes for new requests that use `NULL` for the column. If the new schema update ultimately fails for data validation, there will have been a period of time when writes were blocked, even if they would have been accepted by the old schema.

You can cancel a long-running data validation operation using the `projects.instances.databases.operations.cancel` (/spanner/reference/rest/v1/projects.instances.databases.operations/cancel) method or using `gcloud spanner operations` (/sdk/gcloud/reference/spanner/operations).

If you use the gcloud tool, REST API, or the RPC API, you can issue a batch of one or more `CREATE`, `ALTER`, or `DROP` statements (/spanner/docs/data-definition-language).

Cloud Spanner applies statements from the same batch in order, stopping at the first error. If applying a statement results in an error, that statement is rolled back. The results of any previously applied statements in the batch are not rolled back.

Cloud Spanner might combine and reorder statements from different batches, potentially mixing statements from different batches into one atomic change that is applied to the database. Within each atomic change, statements from different batches happen in an arbitrary order. For example, if one batch of statements contains `ALTER TABLE MyTable ALTER COLUMN MyColumn STRING(50)` and another batch of statements contains `ALTER TABLE MyTable ALTER COLUMN MyColumn STRING(20)`, Cloud Spanner will leave that column in one of those two states, but it's not specified which.

Cloud Spanner uses schema versioning so that there is no downtime during a schema update to a large database. Cloud Spanner maintains the older schema version to support reads while the schema update is processed. Cloud Spanner then creates one or more new versions of the schema to process the schema update. Each version contains the result of a collection of statements in a single atomic change, as described above.

The schema versions don't necessarily correspond one-to-one with either batches of DDL statements or individual DDL statements. Some individual DDL statements, such as non-interleaved index creation or statements that require data validation, result in multiple schema versions. In other cases, several DDL statements can be batched together in a single version. Schema versions can consume significant server and storage resources, and they persist for up to a week.

If you need to add multiple indexes, see the options for large updates (#large-updates).

The following table shows how long it takes Cloud Spanner to update a schema.

Schema operation	Estimated duration
`CREATE TABLE`	Minutes
`CREATE INDEX`	Minutes to hours, if the base table is created before the index. Minutes, if the statement is executed at the same time as the `CREATE TABLE` statement for the base table.
`DROP TABLE`	Minutes
`DROP INDEX`	Minutes
`ALTER TABLE ... ADD COLUMN`	Minutes
`ALTER TABLE ... ALTER COLUMN`	Minutes to hours, if background validation (#updates-that-require-validation) is required. Minutes, if background validation is not required.
`ALTER TABLE ... DROP COLUMN`	Minutes

The following sections describe best practices for updating schemas.

Before you issue a schema update:

- Verify that all of the existing data in the database that you're changing meets the constraints that the schema update is imposing. Because the success of some types of schema updates depends on the data in the database and not just its current schema, a successful schema update of a test database does not guarantee a successful schema update of a production database. Here are some common examples:
  - If you're adding a `NOT NULL` annotation to an existing column, check that the column does not contain any existing `NULL` values.
  - If you're shortening the allowed length of a `STRING` or `BYTES` column, check that all existing values in that column meet the desired length constraint.
- If you're writing to a column, table, or index that is undergoing a schema update, ensure that the values that you're writing meet the new constraints.
- If you're dropping a column, table, or index, make sure you are not still writing to or reading from it.

If you cannot batch your DDL statements, then avoid making many separate schema updates to a single database's schema in a given 7-day period. Increase the time window in which you make schema updates to allow Cloud Spanner to remove old versions of the schema before new versions are created.

- For some relational database management systems, there are software packages that make a long series of upgrade and downgrade schema updates to the database on every production deployment. These types of processes are not recommended for Cloud Spanner.
- Cloud Spanner is optimized to use primary keys to partition data for multitenancy solutions (/spanner/docs/schema-and-data-model#multitenancy). Multitenancy solutions that use separate tables for each customer can result in a large backlog of schema update operations that take a long time to complete.

Avoid more than 30 DDL statements that require validation or index backfill in a given 7-day period, because each statement creates multiple versions of the schema internally.

The best way to create a table and a large number of indexes on that table is to create all of them at the same time. You can create the table and its indexes when you create the database, or in a single

large batch of DDL statements. You can batch several thousand DDL statements in a single request, as long as no more than 10 of the statements require validation or backfill.

If you cannot create the table and its indexes in a single large batch of DDL statements, then try to add no more than 3 new indexes per day.

When making `projects.instances.databases.updateDdl` (/spanner/docs/reference/rest/v1/projects.instances.databases/updateDdl) (REST API) or `UpdateDatabaseDdl` (/spanner/docs/reference/rpc/google.spanner.admin.database.v1#google.spanner.admin.database.v1.DatabaseAdmin.UpdateDatabaseDdl) (RPC API) requests, use `projects.instances.databases.operations.get` (/spanner/docs/reference/rest/v1/projects.instances.databases.operations/get) (REST API) or `GetOperation` (/spanner/docs/reference/rpc/google.longrunning#google.longrunning.Operations.GetOperation) (RPC API), respectively, to wait for each request to complete before starting a new request. Waiting for each request to complete allows your application to track the progress of your schema updates. It also keeps the backlog of pending schema updates to a manageable size.

If you are bulk loading data into your tables after they are created, it is usually more efficient to create indexes after the data is loaded. If you are adding several indexes, it might be more efficient to create the database with all tables and indexes in the initial schema, as described in the options for large updates (#large-updates).