Cloud Spanner Point-In-Time-Recovery: Restoring a Dropped Table

Christoph Bussler
May 11 · 6 min read

tl;dr Cloud Spanner introduced point-in-time-recovery (PITR) as a supported database feature. This blog demonstrates how to recover a dropped table using PITR using gcloud commands.

Point-in-time-recovery

PITR allows you to recover data at a point in time in the past. Cloud Spanner enables you to configure the data (and schema) retention period — up to 7 days.

If you enable 7 days of retention you can query up to 7 days in the past using stale reads. This enables you to recover part of the database or the whole database for up to 7 days. In order to execute a stale query you need to know the timestamp at which you need to query the data in order to retrieve the correct state.

In the following a complete example is provided that allows you to recover a dropped table. It shows you all the steps in the form of an example. Afterwards best practices and next steps are discussed.

Use case: accidentally dropped table

Overview

Dropping a table and recovering it using PITR is a standard example for databases supporting PITR. In a production environment that might happen accidentally and the last consistent state of the table has to be recovered. This involves determining the time the table was dropped, and determining the time that had the latest state of the table.

Once that time is determined, the table can be recovered and added to the current state of the database.

Creating schema and inserting data

The following gcloud commands establish a Cloud Spanner instance, a database and a table. If you already have a table to experiment with you can skip this section.

1. Set environment variables

```
export instance_name=blog-instance
export database_name=blog-database
export instance_region=regional-us-west1
```

2. Create instance and database

```
gcloud spanner instances create $instance_name \
  --description=$instance_name \
  --config=$instance_region \
  --nodes=1

gcloud spanner databases create $database_name \
  --instance=$instance_name
```

3. Create a table, insert three rows and query the table

```
gcloud spanner databases ddl update $database_name \
  --instance=$instance_name \
  --ddl='CREATE TABLE blog_entity (k INT64, v STRING(1024))
          PRIMARY KEY(k)'

gcloud spanner databases execute-sql $database_name \
  --instance=$instance_name \
  --sql="INSERT blog_entity (k, v) VALUES (1, 'first entity')"

gcloud spanner databases execute-sql $database_name \
  --instance=$instance_name \
  --sql="INSERT blog_entity (k, v) VALUES (2, 'second entity')"

gcloud spanner databases execute-sql $database_name \
  --instance=$instance_name \
  --sql="INSERT blog_entity (k, v) VALUES (3, 'third entity')"

gcloud spanner databases execute-sql $database_name
  --instance=$instance_name \
  --sql='SELECT * FROM blog_entity'
```

The result of this query is

```
k v
1 first entity
2 second entity
3 third entity
```

Dropping a table

Now let's implement the disaster of accidentally dropping the table. This is done with the following command:

```
gcloud spanner databases ddl update $database_name
  --instance=$instance_name \
  --ddl='DROP TABLE blog_entity'
```

At this point the table is not in the database anymore. Verify by trying to select the rows of the dropped table:

```
gcloud spanner databases execute-sql $database_name \
  --instance=$instance_name \
  --sql='SELECT * FROM blog_entity'
```

This query returns an error:

```
ERROR: (gcloud.spanner.databases.execute-sql) INVALID_ARGUMENT: Table
not found: blog_entity [at 1:15]\nSELECT * FROM blog_entity\n ^
- '@type': type.googleapis.com/google.rpc.LocalizedMessage
locale: en-US
message: |-
```

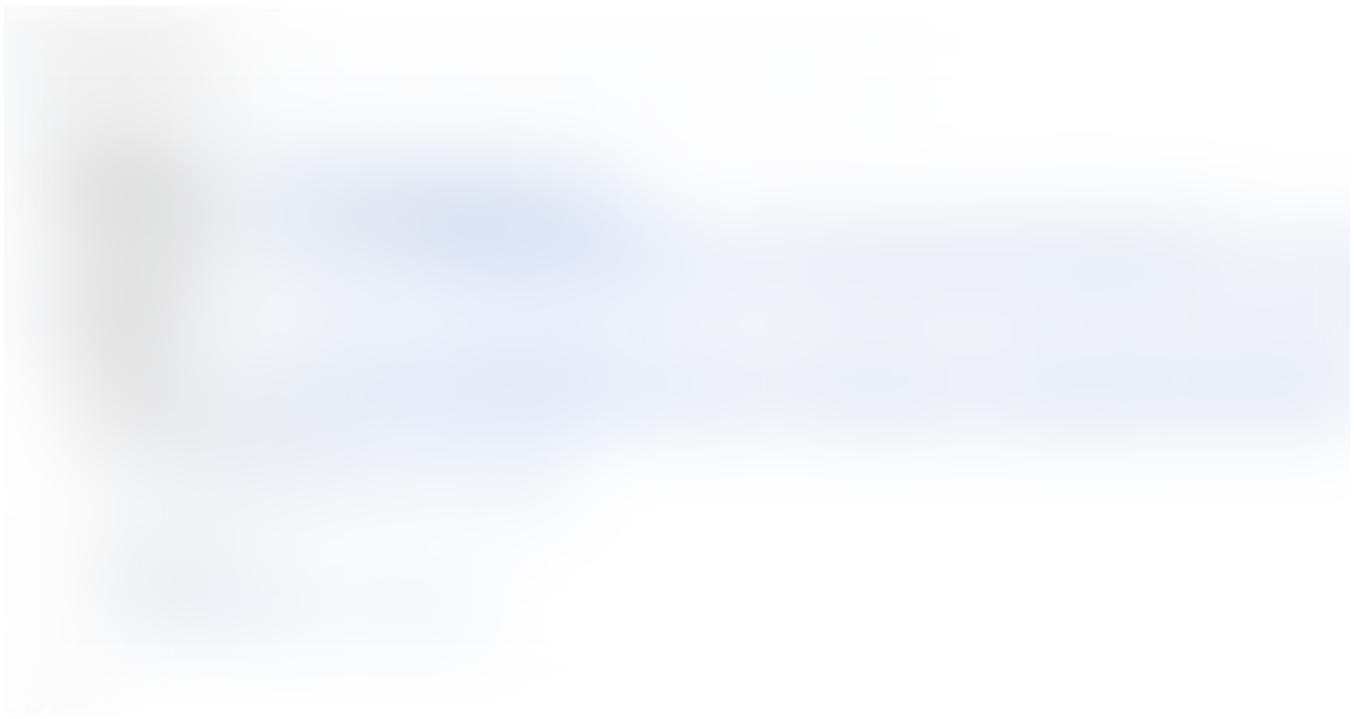
```
Table not found: blog_entity [at 1:15]
SELECT * FROM blog_entity
```

Recovery of dropped table using stale read

In Cloud Spanner the table can be recovered by executing a stale read at the timestamp of the table's last consistent state. The following gcloud commands demonstrate this:

Determining commit timestamp at time of table drop

Determine commit timestamp of command that dropped table from the log explorer, e.g., `2021-04-30T22:30:07.752628Z`:



Log entry for DROP TABLE statement

This log entry is the last if you did not execute any other command that is recorded. If other commands are executed you might have to search for this. Therefore it is helpful if the approximate time of the table drop is known.

Timestamps in Cloud Spanner have microsecond granularity and therefore the approach to determine the point it time to query is as follows.

Create the read timestamp based on the commit timestamp by subtracting 1.

For example, if the commit timestamp is

```
2021-04-30T22:30:07.752628Z
```

subtract 1 so that the timestamp looks like

```
2021-04-30T22:30:07.752627Z
```

This is the timestamp for the stale query recovering the last consistent state of the table before it was dropped. This is the case as this is the last timestamp where a row could have been inserted, deleted or modified.

Note that the table is not present at the commit timestamp itself:

```
gcloud spanner databases execute-sql $database_name \
  --instance=$instance_name \
  --sql='SELECT * FROM blog_entity' \
  --read-timestamp=2021-04-30T22:30:07.752628Z \
  --format=json
```

Results in

```
ERROR: (gcloud.spanner.databases.execute-sql) INVALID_ARGUMENT: Table
not found: blog_entity [at 1:15]\nSELECT * FROM blog_entity\n ^
- '@type': type.googleapis.com/google.rpc.LocalizedMessage
locale: en-US
message: |-
Table not found: blog_entity [at 1:15]
SELECT * FROM blog_entity
```

Recreating the dropped table

Use the create table statement from the schema definition from your code control system (e.g., git):

```
gcloud spanner databases ddl update $database_name \
  --instance=$instance_name \
  --ddl='CREATE TABLE blog_entity (k INT64, v STRING(1024))
         PRIMARY KEY(k)'
```

If you do not have the schema definition in code control, we strongly recommend making the database schema a code controlled artifact.

Until then you could use the information tables to derive the table definition. For example, to retrieve the columns of a table in a stale read, execute:

```
gcloud spanner databases execute-sql $database_name \
  --instance=$instance_name \
  --sql="SELECT t.column_name, t.spanner_type, t.is_nullable
         FROM information_schema.columns AS t
         WHERE t.table_catalog = ''
         AND t.table_schema = ''
         AND t.table_name = 'blog_entity'" \
  --read-timestamp=2021-04-30T22:30:07.752627Z
```

To determine the complete schema of the table you will have to query additional data from the information schema, including constraints, for example.

Retrieving and inserting data of dropped table

1. Execute a stale query selecting all rows of the dropped table

```
gcloud spanner databases execute-sql $database_name \
  --instance=$instance_name \
  --sql='SELECT * FROM blog_entity' \
  --read-timestamp=2021-04-30T22:30:07.752627Z \
  --format=json
```

It results in

```
{
"metadata": {
  "rowType": {
    "fields": [
      {
        "name": "k",
        "type": {
          "code": "INT64"
          }
      },
      {
        "name": "v",
        "type": {
          "code": "STRING"
          }
      }
    ]
  },
  "transaction": {}
},
"rows": [
  [
    "1",
    "first entity"
  ],
  [
    "2",
    "second entity"
  ],
  [
    "3",
    "third entity"
  ]
]
}
```

Other formats in addition to JSON are available as well.

2. Insert the rows into the newly created table

```
gcloud spanner rows insert \
  --table=blog_entity \
  --database=$database_name \
  --instance=$instance_name \
  --data=k=1,v='first entity'

gcloud spanner rows insert \
  --table=blog_entity \
  --database=$database_name \
  --instance=$instance_name \
  --data=k=2,v='second entity'

gcloud spanner rows insert \
  --table=blog_entity \
  --database=$database_name \
  --instance=$instance_name \
  --data=k=3,v='third entity'
```

3. Verify that the table exists and that it contains the three rows

```
  gcloud spanner databases execute-sql $database_name \
    --instance=$instance_name \
    --sql='SELECT * FROM blog_entity'
```

Results in

```
  k v
  1 first entity
  2 second entity
  3 third entity
```

Best practices

The above example demonstrates the principle approach of recovering an accidentally dropped table. The individual steps were shown as manually executed gcloud commands in order to be able to demonstrate the detailed steps to you.

In a production setting we recommend that you develop a script for this case that automates this behavior, especially when tables contain more than a handful of rows. When developing such a script, be aware of the 20K mutation limit as well as the bulk loading best practices.

There are additional scenarios that you might consider preparing for, for example, recovering a whole database to a point in the past, or recovering a few transactions across tables. In any case it is best practice to prepare scripts as well as train for the scenarios you plan to support.

What's next

- Execute the above example in your project as a first exercise
- Explore additional use cases and write scripts or programs for production scenarios

- Review the Cloud Spanner PITR product documentation and perform the PITR for a whole database

Acknowledgements

I'd like to thank John Corwin and Gideon Glass for the thorough review and several comments to improve the accuracy of this content.

Disclaimer

Christoph Bussler is a Solutions Architect at Google, Inc. (Google Cloud). The opinions stated here are my own, not those of Google, Inc.

Google Cloud Cloud Spanner Deleted Table Recovery Point In Time