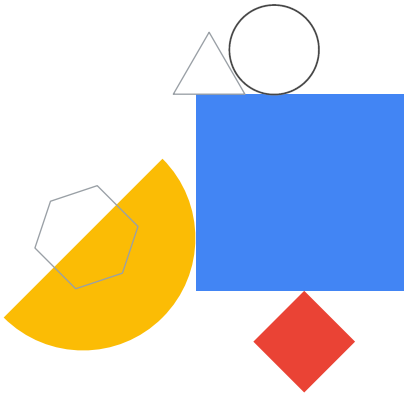
Google Cloud

Controlling Access with Data Security Best Practices



One of the most important topics is safeguarding your data from unintended access. In this module we will cover a range of methods from securing individual columns through encryption to limiting access to specific rows in a table through authorized views.

We'll then cover the default access control roles available in BigQuery and how those map to your overall Google Cloud account roles.

Hashing columns with FARM_FINGERPRINT( )

```
SELECT
   FARM_FINGERPRINT("secure") AS one_way_encryption
```

Row	one_way_encryption
1	1089365847954776577

Others: SHA256( ) SHA512( )

Hashing columns is useful for creating new primary key columns. One example is if you have incremental timestamp data (which is bad for sharding performance as recent data will overload your last shard) you can apply a hash to help more randomly distribute the loads between shards.

Note: Hashing here is **not meant for storing passwords**. There are separate functions for that.

Using authorized views to limit row access

- Create logical views into your dataset that filter based on user roles

```
#standardSQL
SELECT [COLUMN_1, COLUMN_2]
FROM `[DATASET.VIEW]`
WHERE allowed_viewer = SESSION_USER()
```


A great way to limit access to your dataset at the row level is by using SESSION_USER() and filtering against a pre-defined whitelist of allowed viewers.

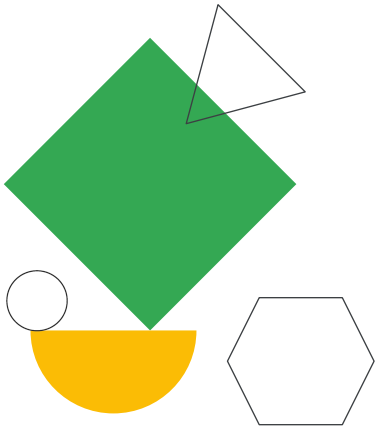
https://cloud.google.com/bigquery/docs/share-access-views

Row Level Permissions
https://cloud.google.com/bigquery/docs/views#row-level-permissions

Demo

Creating an Authorized View to filter by logged in user

Limit rows to only certain user groups



Refer to https://github.com/GoogleCloudPlatform/training-data-analyst/tree/master/courses/data-to-insights/demos/authorized-view.sql

You can assign the following BigQuery **predefined roles**

Capability	dataViewer	dataEditor	dataOwner	user	jobUser	admin
List/get projects	✓	✓	✓	✓	✓	✓
List tables	✓	✓	✓	✓	⊘	✓
Get table data/metadata	✓	✓	✓	⊘	⊘	✓
Create tables	⊘	✓	✓	⊘	⊘	✓
Modify/delete tables	⊘	✓	✓	⊘	⊘	✓
List/get datasets	✓	✓	✓	✓	⊘	✓
Create new datasets	⊘	✓	✓	✓	⊘	✓
Modify/delete datasets	⊘	⊘	✓	Self-created datasets	⊘	✓
Create jobs/queries	⊘	⊘	⊘	✓	✓	✓
Cancel jobs	⊘	⊘	⊘	Self-created jobs	Self-created jobs	Any jobs
Get/list saved queries	⊘	⊘	⊘	✓	⊘	✓
Create/update/delete saved queries	⊘	⊘	⊘	⊘	⊘	✓
Get transfers	⊘	⊘	⊘	✓	⊘	✓
Create/update/delete transfers	⊘	⊘	⊘	⊘	⊘	✓

Although you can create your own custom roles for users, why not use the 6 pre-defined BigQuery roles here? Take a look at each of their associated capabilities within BigQuery.

The **lowest level is a jobUser** and the **highest level is the admin**. Permissions are inherited (e.g. an admin is automatically a dataViewer)

Read:
https://cloud.google.com/bigquery/docs/access-control#predefined_roles_details

IAM **Project Roles** and inherited BigQuery permissions

When a project is created, BigQuery grants the `Owner` role to the user who created the project.

Basic role	Capabilities
`Viewer`	• Can start a job in the project. Additional dataset roles are required depending on the job type. • Can list and get all jobs, and update jobs that they started for the project. • If you create a dataset in a project that contains any viewers, BigQuery grants those users the bigquery.dataViewer predefined role for the new dataset.

`Editor`	• Same as **`Viewer`**, plus:
	  ○ Can create a new dataset in the project.
	  ○ If you create a dataset in a project that contains any editors, BigQuery grants those users the bigquery.dataEditor predefined role for the new dataset.

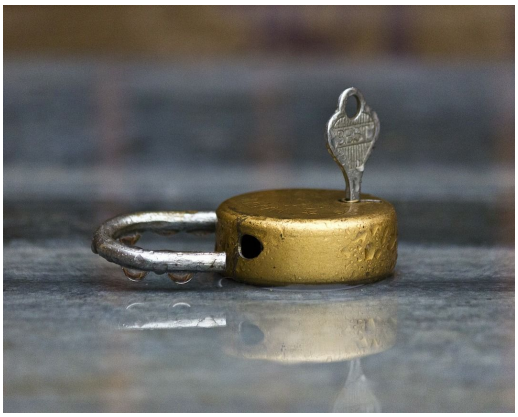
`Owner`	• Same as **`Editor`**, plus:
	  ○ Can list all datasets in the project.
	  ○ Can delete any dataset in the project.
	  ○ Can list and get all jobs run on the project, including jobs run by other project users.
	  ○ If you create a dataset, BigQuery grants all project owners the bigquery.dataOwner predefined role for the new dataset.

IAM Project Roles are Google Cloud Project Roles. If you grant someone Editor permission on your project, they will also be able to create new datasets and perform all the functions that the dataEditor role can (on the previous slide).

Default access to datasets **can be overridden on a per-dataset basis**.

Avoid access pitfalls

- Dataset users should have the minimum permissions needed for their role
- Use separate projects or datasets for different environments (e.g. DEV, QA, PRD)
- Audit roles periodically



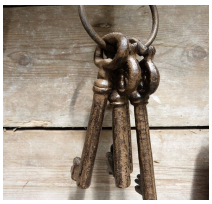
Data security is more than simply applying the right project permissions and roles during your dataset creation. You should have a written and circulated Data Access Policy for your organization that specifies how and when data can be shared and with whom.

One first step to ensure healthy data access controls is to periodically audit the users and groups associated with your Google Cloud project and individual datasets. Do these users need to be Owners or Admins or would a more restrictive role suffice for their job duties?

A second pitfall is working, testing, and editing datasets in production. It is often wise to have a completely separate Google Cloud project or dataset for different testing environments to prevent unintentional data loss.

https://cloud.google.com/bigquery/docs/access-control

Summary: Control data access levels within BigQuery datasets



Assign the right permissions and roles for each user group based on their needs.

Create multiple datasets or projects for different development environments.

Periodically audit access control credentials.

Use authorized views for row-level control over your datasets.

Setting up proper access to your data is one of the most important topics in data analysis. As you have seen, Google Cloud will provide you with a variety of roles at the Project level that can be inherited down to the BigQuery dataset level. It's ultimately up to you to determine which individuals and groups should have access to which parts of your data.

Since storage is cheap, a common solution for development is to actually mirror your data into silo'd environments with differing permissions.

Be sure to set up a regular access control audit and review your Stackdriver access logs to monitor for any strange usage patterns.

Lastly, as we mentioned before, consider using authorized views in tandem with a where clause filter on CURRENT_USER to limit row level access to a particular table based on the logged in user.