

# BigQuery and Modern Data Warehousing

by (Bangun Sasongko)

# Trainer Profile

Bangun W. Sasongko

Data Engineer at BRI

Ex. Devops Engineer

Bachelor of Physics, Electrical and Instrumentations

@sasongkobgn

[sasongkobgn@gmail.com](mailto:sasongkobgn@gmail.com)



# Table of Content

Content

Deep Dive BigQuery

Loading Data into Big Query

Partitioning and Clustering in BigQuery

Working with Views and Materialized Views

Unifying Data Lake and Data Warehouse



# About this Course

This course delves into advanced functionalities of BigQuery, empowering you to optimize data management, processing, and analysis for large datasets



# The Objectives

By the end of this course, you will be able to:

1. Optimize Data Management in BigQuery
2. Enhance Query Performance
3. Bridge the Data Lake and Data Warehouse Gap





# Deep Dive BigQuery

# What makes BigQuery Modern?



## No Ops

Fully managed; storage and compute resources grow as you use them.



## Auditable

Every transaction is logged and can be audited.



## Performant

Petabyte-scale, highly parallel processing for fast queries.



## Secure

Cloud IAM, ACLs, and data encryption at rest and in transit.



## Efficient

Pay only for the storage and processing you use, with an efficient encoding format that is optimized for analytic workloads.



## Easy to Use

Familiar SQL, public datasets to experiment with.

# What makes BigQuery Modern?

## Two Services in One

MANAGED STORAGE

+

ANALYTICS ENGINE

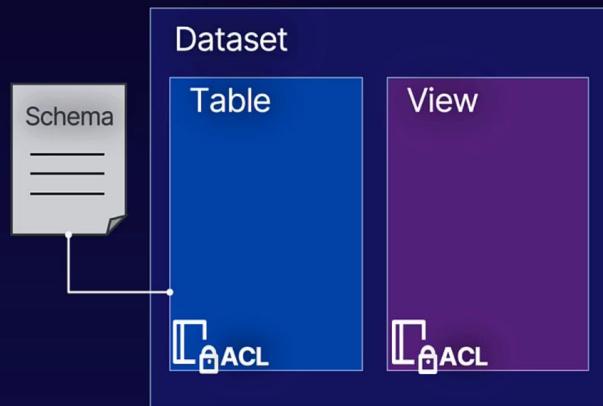
Fully managed,  
scalable columnar  
**data storage.**

Performant, massively parallel  
**SQL engine.**

# BigQuery Basics Concepts



# BigQuery Basics Concepts



# BigQuery Basics Concepts



**OPTION 1: CONSOLE**

**OPTION 2: COMMAND LINE (BQ)**

**OPTION 3: REST API**

# BigQuery Basics Concepts

## BigQuery Use Cases

1

### General Analytics

Driving insights from data through online analytical processing (OLAP).

2

### Predictive Analytics

Anomaly/fraud detection, recommendation systems, sales forecasting, etc.

3

### Data Monetization

You can list your data for sale in the Google Cloud Marketplace.



## Takeaways

- BigQuery is a fully managed, modern, and easy to use data warehousing services
- BigQuery separation of storage and compute makes it resource and cost efficient
- There are 3 ways to interact with BigQuery service: console, command line, and REST APIs



# Loading Data into BigQuery

# Data Ingestions and Options



**Batch**

Batch load a set of data records



**Stream**

Stream individual records or batches



**Generate**

Run queries to generate new data



**Connect**

Use a third-party app or service

# Batch loading

## Load Jobs



Cloud  
Storage

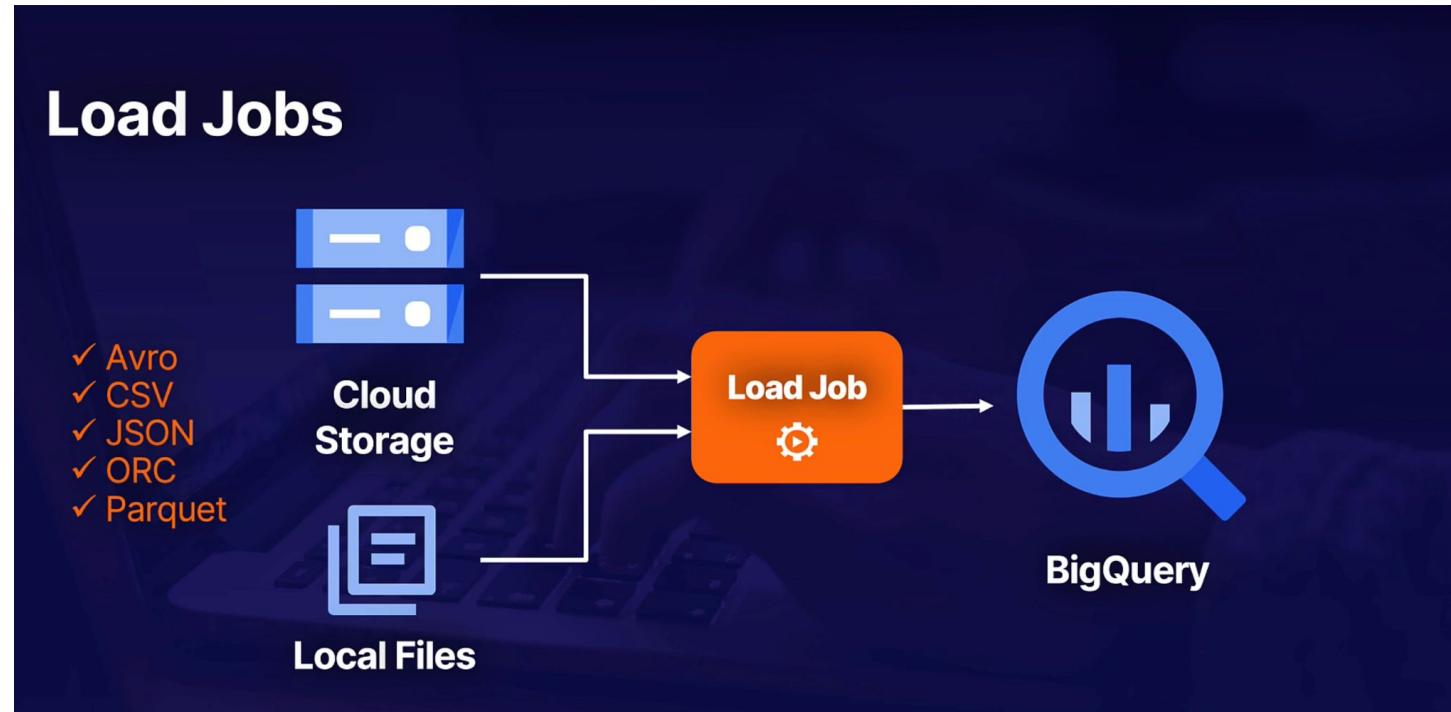


Local Files

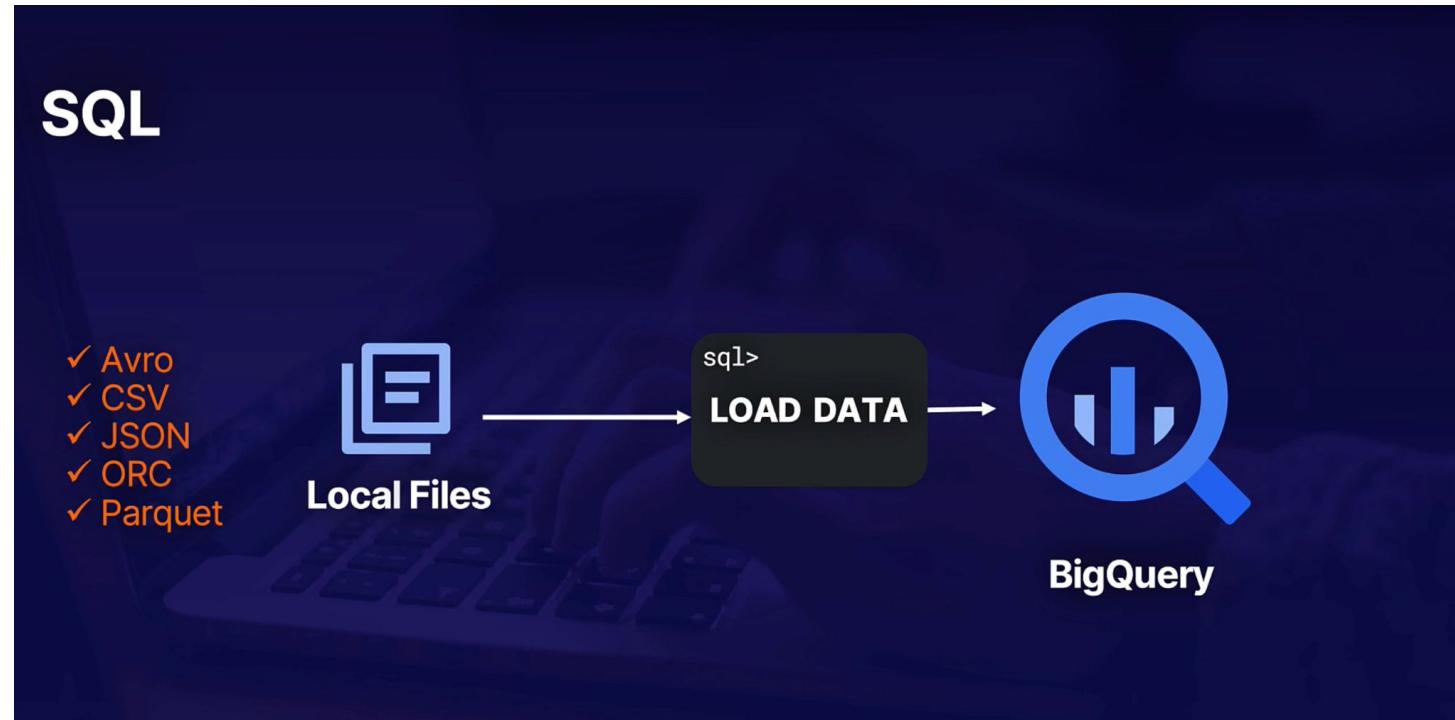


BigQuery

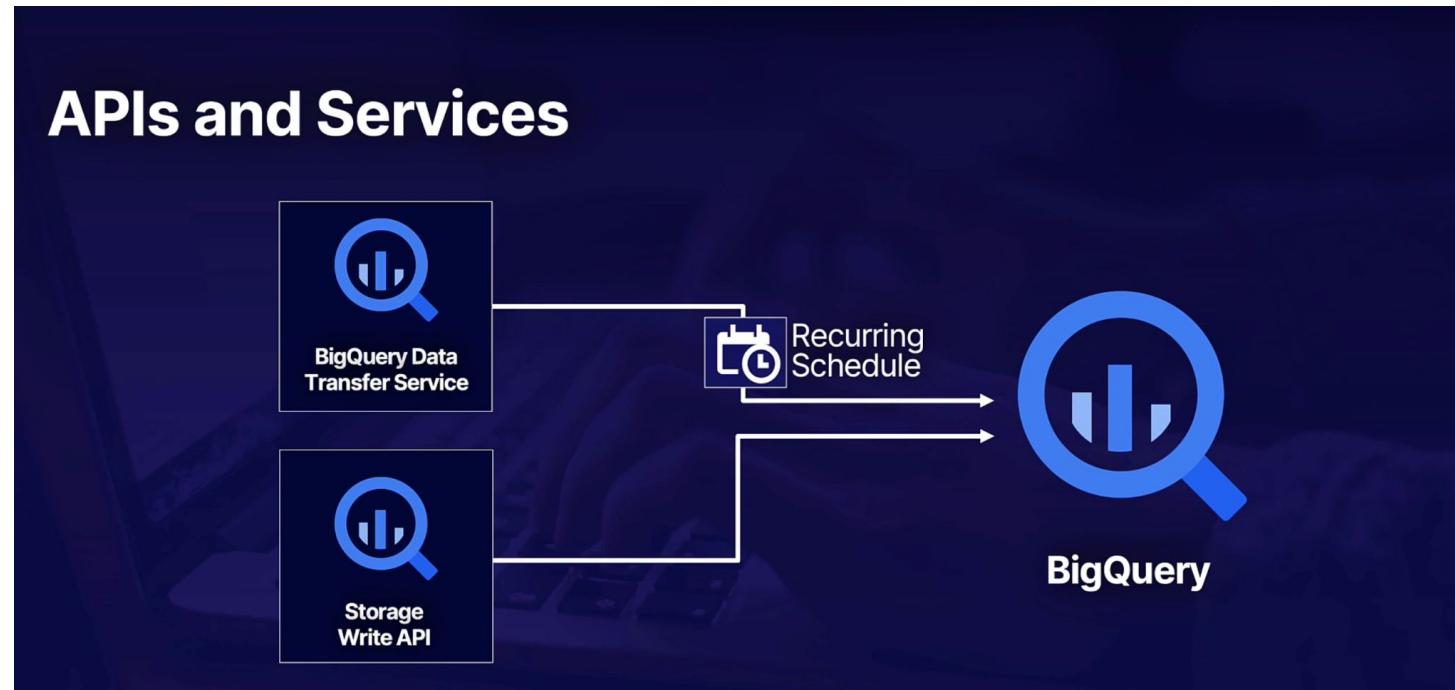
# Batch loading



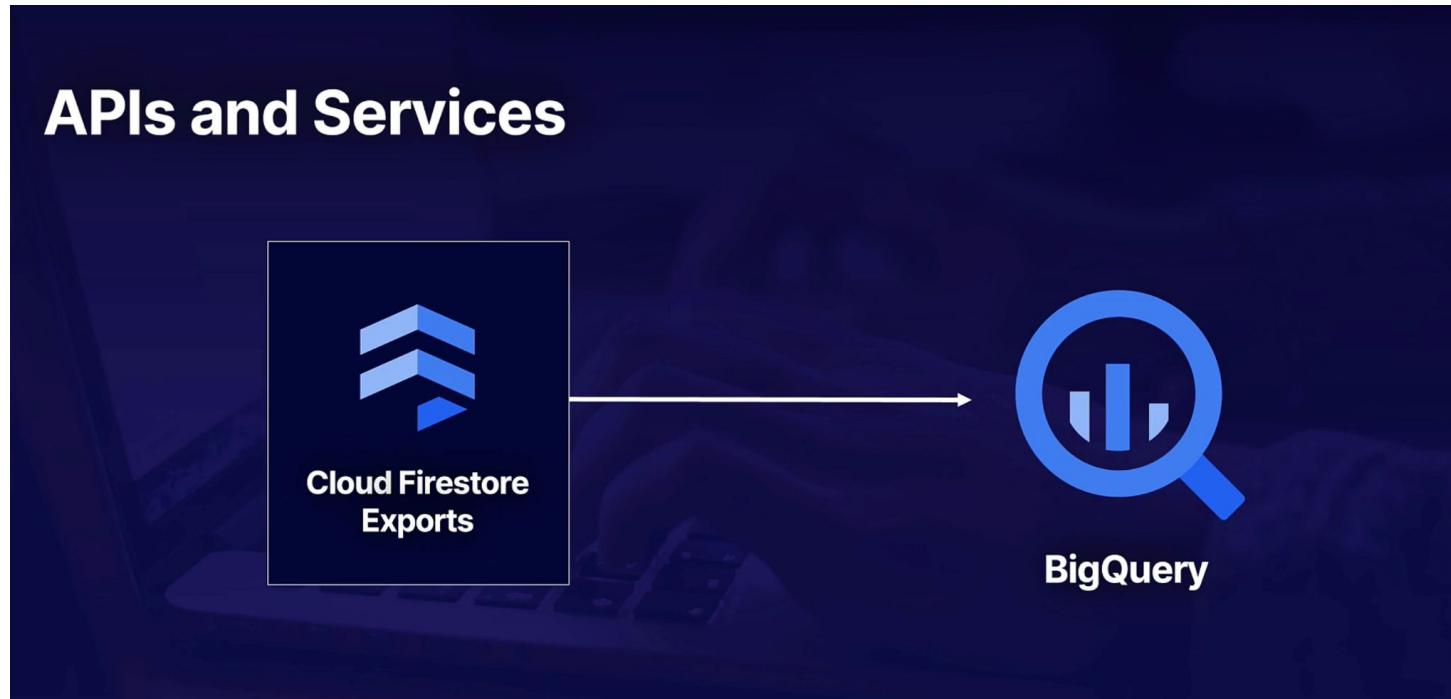
## Batch loading



## Batch loading

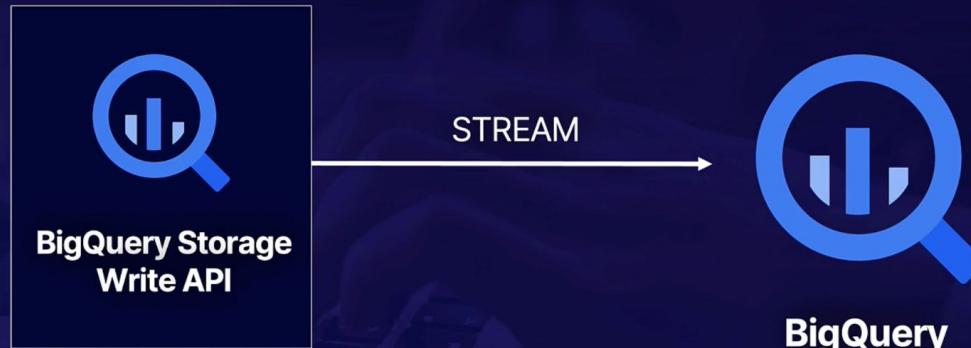


## Batch loading



## Streaming ingestion

### Storage Write API



## Streaming ingestion

### BigQuery Connector for SAP



STREAM



BigQuery

## Streaming ingestion

### Cloud Dataflow



Cloud  
Dataflow

STREAM



BigQuery

## Generating data with SQL

### Writing Query Results

```
bq query \  
--destination_table mydataset.mytable \  
--use_legacy_sql=false \  
'SELECT  
    name,  
    number  
FROM  
    `bigquery-public-  
data`.usa_names.usa_1910_current  
WHERE  
    gender = "M"  
ORDER BY  
    number DESC'
```



BigQuery

## Generating data with SQL

### Writing Query Results

```
bq query \  
--destination_table mydataset.mytable \  
--use_legacy_sql=false \  
'SELECT  
    name,  
    number  
FROM  
    `bigquery-public-  
data`.usa_names.usa_1910_current  
WHERE  
    gender = "M"  
ORDER BY  
    number DESC'
```



BigQuery

# Choosing how to load data

## 1 Real-time vs. Offline Analysis

If you need real-time analysis, consider streaming. For offline analysis, use batch loading.

## 2 Data Source

For local files or Cloud Storage, consider batch load jobs. If the data source is Cloud SQL, consider federated queries.

## 3 Latency Requirements

Streaming load options will have the lowest latency.



# Takeaways

- You have broad option to load data into BigQuery: batch, stream, generate, connect
- Specific methods to ingest data include: batch load jobs, SQL, APIs, Servicers and Connectors
- The best option for loading data is depends on data sources as well as analytics and latency requirements



# Partitioning and Clustering in BigQuery

# Partitioned Tables

transaction_time	transaction_id	output_satoshis
2010-02-25 05:28:39	dbf33c055...	50000000000
2010-07-12 08:48:40	6f2b8632a...	377500000000
2011-01-08 18:07:17	a2b5e2db7...	11118375
2011-01-09 03:36:27	b132e400a...	0
2010-06-28 05:12:54	b555d9465...	50000000000
2010-06-25 22:10:55	8a906b3e7...	1000000000000
2010-05-22 06:50:28	d67b248f1...	75000000
2011-01-09 20:47:26	9ad36092...	461016780

**What were the transactions between  
2010-01-01 and 2010-12-31?**



transaction_time	transaction_id
2010-02-25 05:28:39	dbf33c055...
2010-07-12 08:48:40	6f2b8632a...
2010-06-28 05:12:54	b555d9465...
2010-06-25 22:10:55	8a906b3e7...
2010-05-22 06:50:28	d67b248f1...

## You can partitions table by

The slide features three cards against a dark blue hexagonal background. Each card has a large orange-to-purple circular icon at the top, followed by a title and a descriptive subtitle.

- Time-Unit Column**  
Based on **TIMESTAMP**, **DATE**, or **DATETIME**
- Ingestion Time**  
Based on the timestamp of when the data was ingested
- Integer Range**  
Based on an integer column in your data

# Clustering Overview



# Clustering Overview

Combined with Partitioning...

```
SELECT  
  *  
FROM  
  mydataset.orders  
  
WHERE order_id in (1010,1030)  
AND order_date = "2022-01-10"
```

Orders Clusters (by ID)

1001-1100

1101-1200

1201-1300

...

# Clustering Overview

Combined with Partitioning...

```
SELECT
  *
FROM
  mydataset.orders
WHERE order_id in (1010, 1030)
AND order_date = "2022-01-10"
```

Partition

2022-01

2022-02

Orders Clusters (by ID)

1001-1100

1101-1200

1201-1300

...

# Using Partitioned and Tables Using SQL

## Create Partitioned Tables Using SQL

```
CREATE TABLE
`mydataset.mytable_partitioned`
PARTITION BY
DATE_TRUNC(transaction_date, MONTH)
AS (SELECT *
FROM `mydataset.mytable`);
```

# Using Partitioned and Tables Using SQL

## Create Clusters Using SQL

```
CREATE TABLE
`mydataset.mytable_clustered`
CLUSTER BY
order_id
AS (SELECT *
FROM `mydataset.mytable`)
```

# When to use Partitioning?

# When to Use Partitioning

## 1 Time- or Numeric-Based Segmentation

You need to segment your data based on time or numeric ranges.



## 2 Low-Cardinality Column

There aren't too many distinct values.

## 3 Partitions Wouldn't Be Too Small

Streaming load options will have the lowest latency.

# When to Use Clustering

## 1 Colocate Related Data

The column(s) contents can help organize the data for common filtering actions.



## 2 High-Cardinality Column

There are many distinct values.

## 3 Filters and Aggregations Over Multiple Columns

Your queries often use filters or aggregations against certain columns.

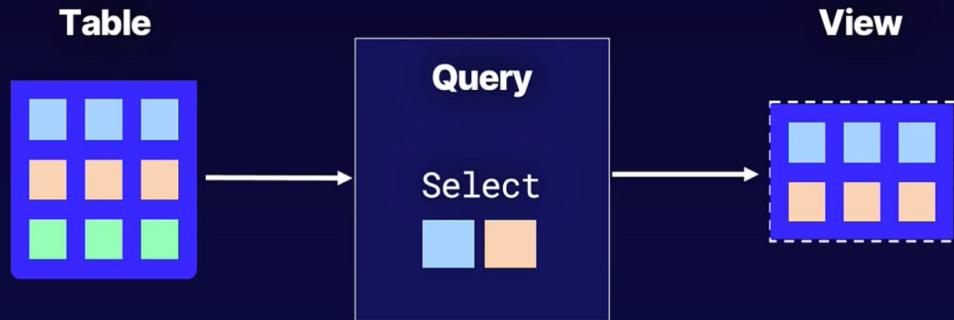
# Takeaways

- Partitions table are table divided into segments that simplify data management and improve query performance`
- You can partitions using time and numeric related columns with different level of granularity
- Clustering is used to colocate related data that can improve performance of filter and aggregation queries

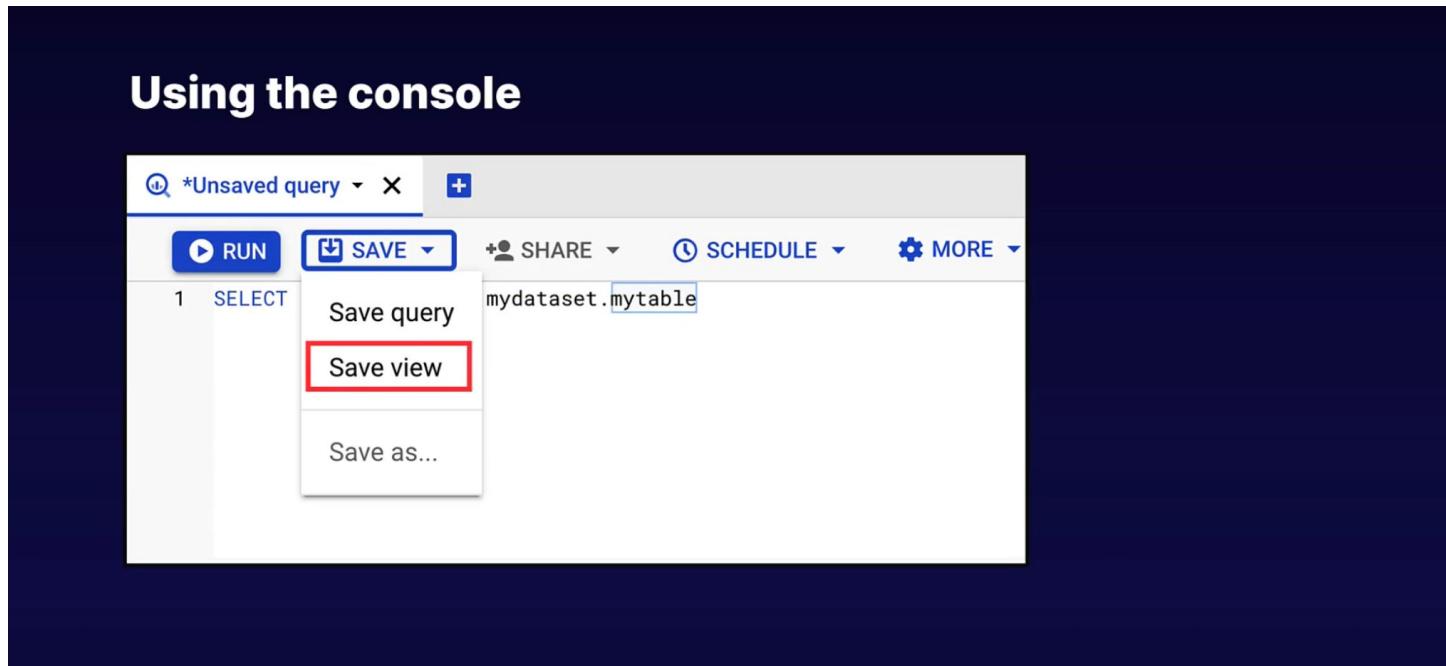


# Working with Views and Materialized Views

# What is views?



## What is views?

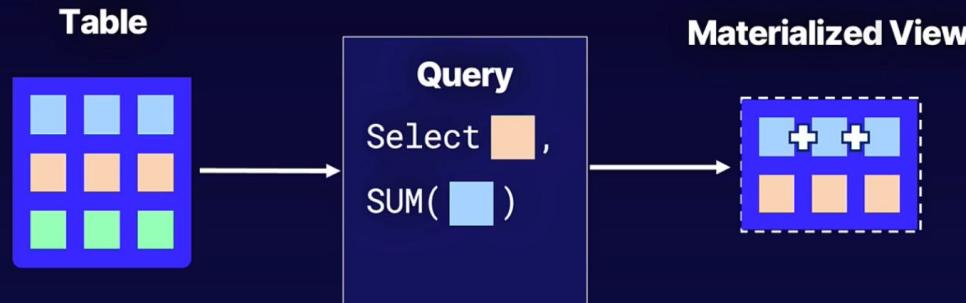


# What is views?

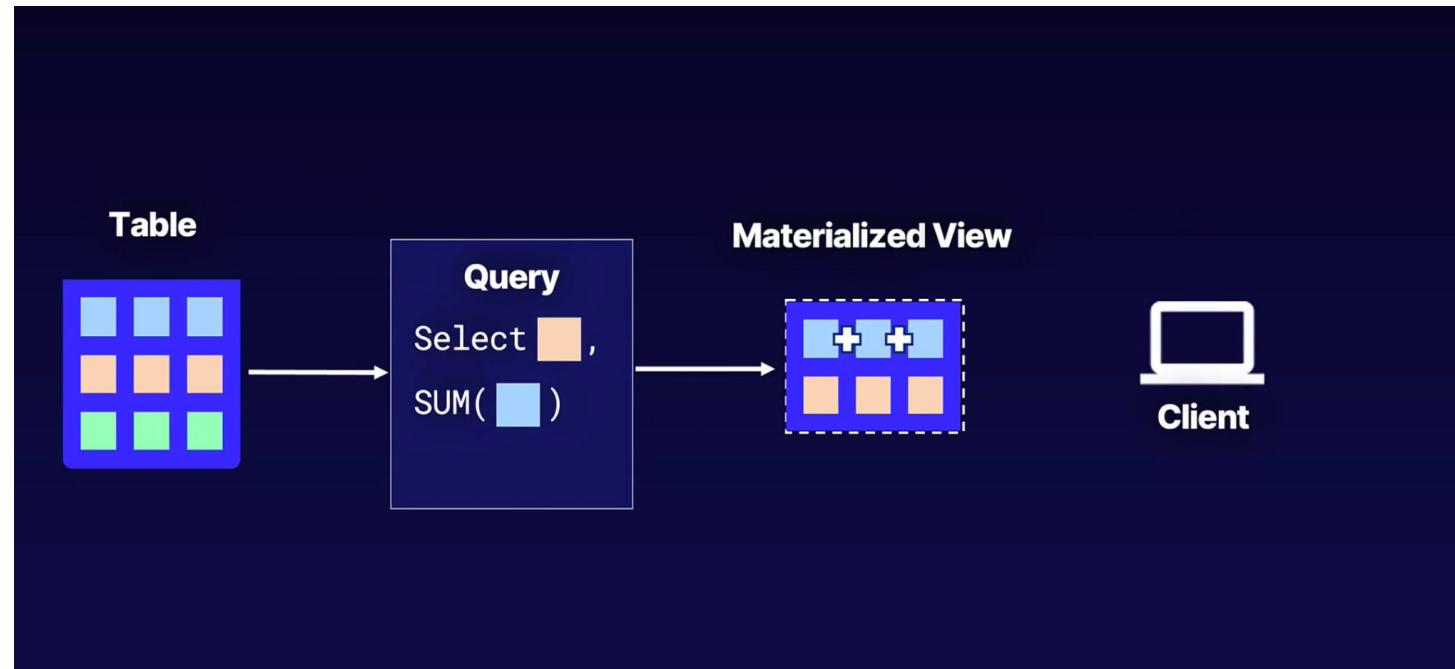
## Using the CLI

```
bq mk \
--use_legacy_sql=false \
--expiration 3600 \
--description "This is my view" \
--view \
'SELECT
  col1,
  col2
FROM
  `mydataset.mytable`
WHERE
  col1 = "val" ' \
mydataset.myview
```

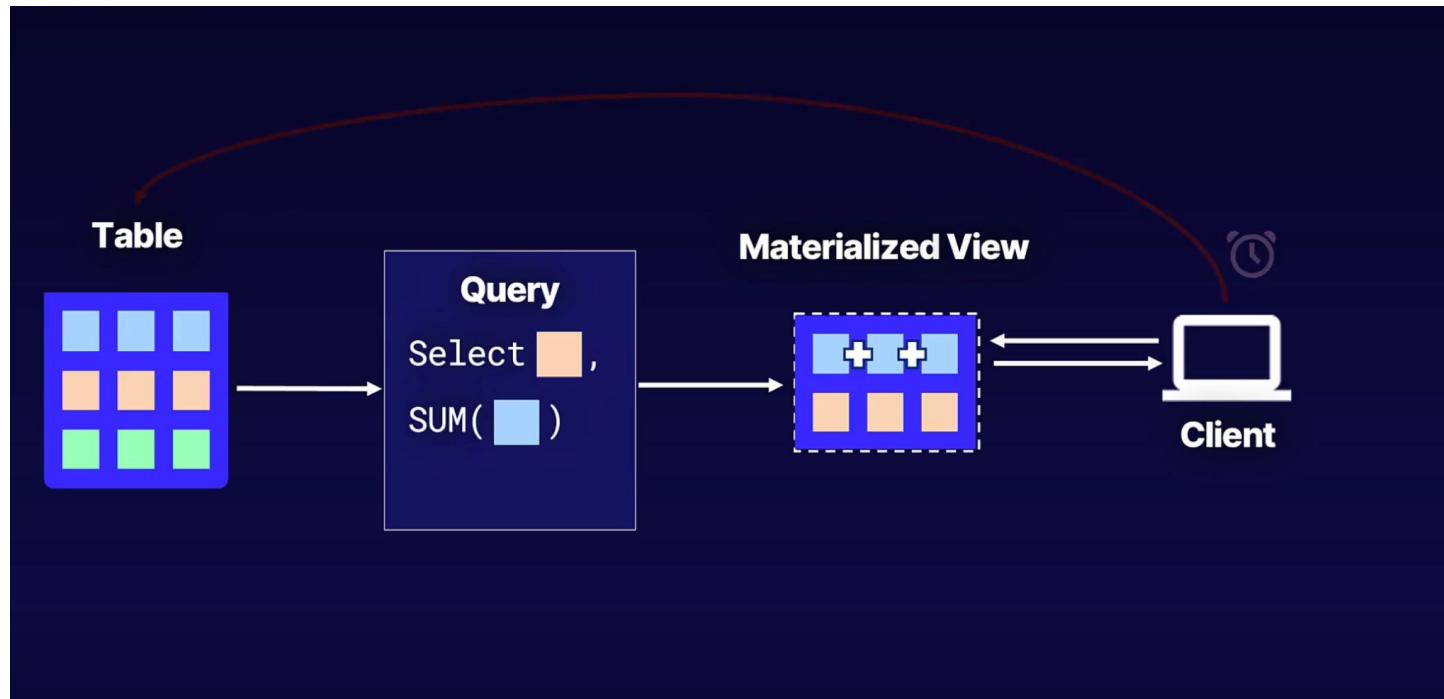
# Materialized Views



# Materialized Views



# Materialized Views



# Materialized Views

## Using SQL

```
CREATE MATERIALIZED VIEW
myproject.mydataset.my_mat_view AS (
    SELECT
        product_id,
        SUM(clicks) AS sum_clicks
    FROM
        myproject.mydataset.my_table
    GROUP BY
        product_id
);
```

# Materialized Views



## Aggregate

Pre-aggregate streaming data as they come



## Filter

Pre-filter a large dataset to read only a subset



## Join

Run queries that join large and small tables



## Cache Results

Precompute results for queries that consume many resources

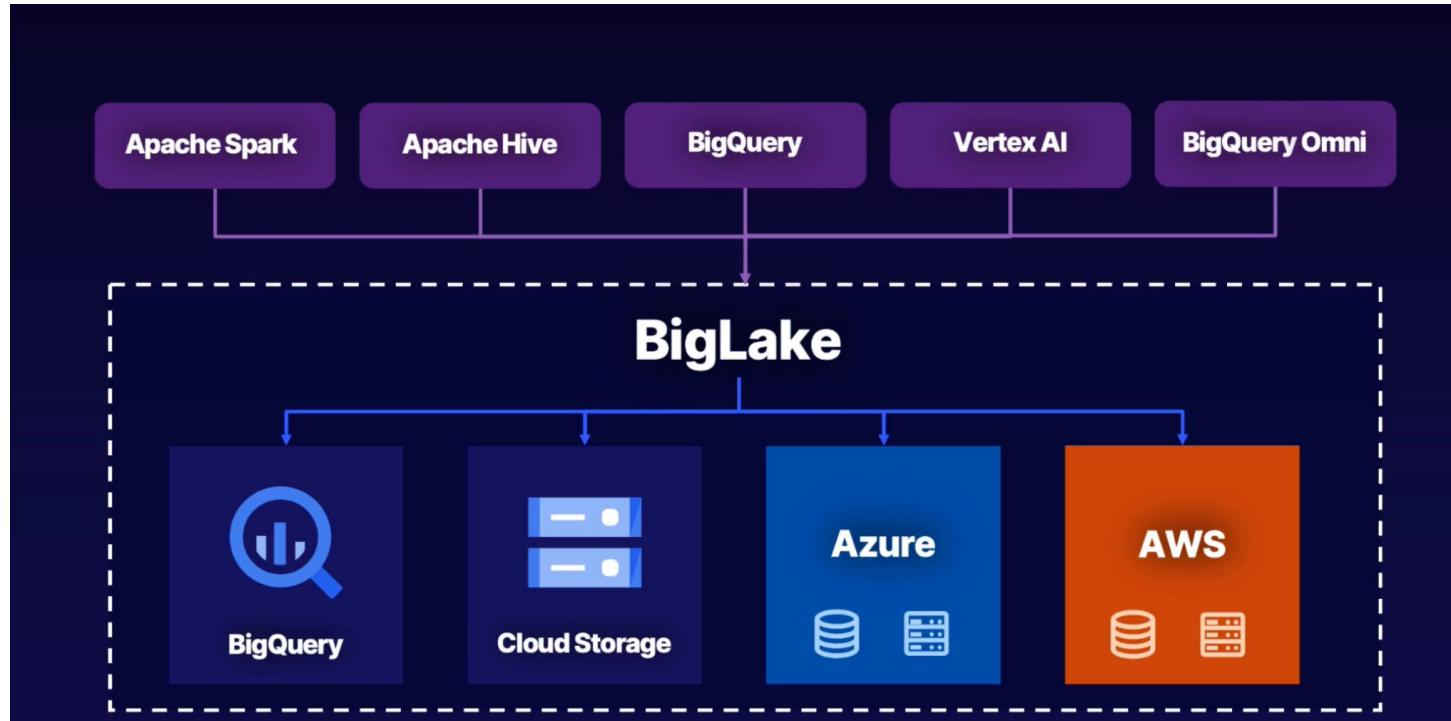
# Takeaways

- Views are virtual tables defined by SQL query and can be used in the same way table are used.
- A materialized view is precomputed view that caches result of a query
- materialized views are ideal for when the query consumes a lot of resources, while the base table does not change often

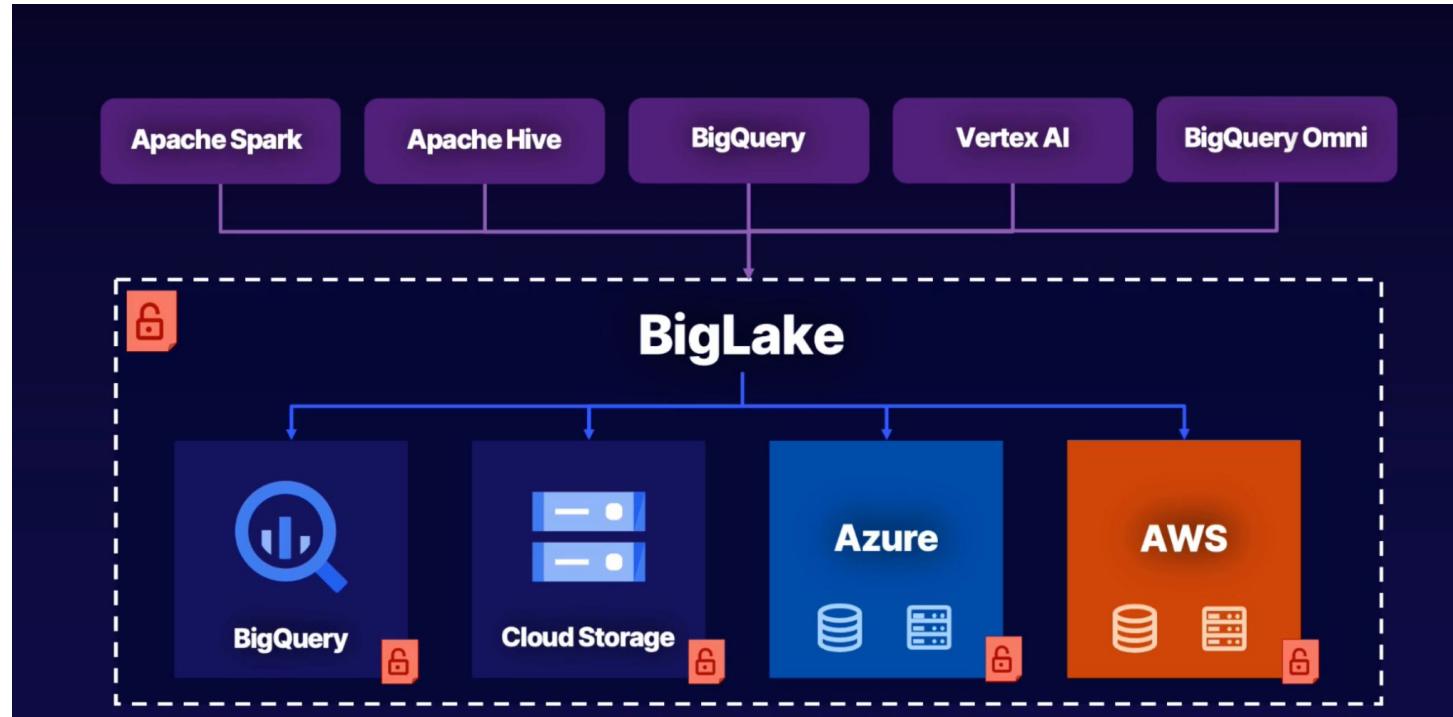


# Unifying data lake and data warehouses

# Introductions to Big Lake



# Introductions to Big Lake



# Big Lake Features



## Unified Management

Manage any data lake tables in Cloud Storage, AWS, or Azure.



## Fine-Grained Security

Define row- and column-level security as you would in BigQuery.



## Multi-Cloud Governance

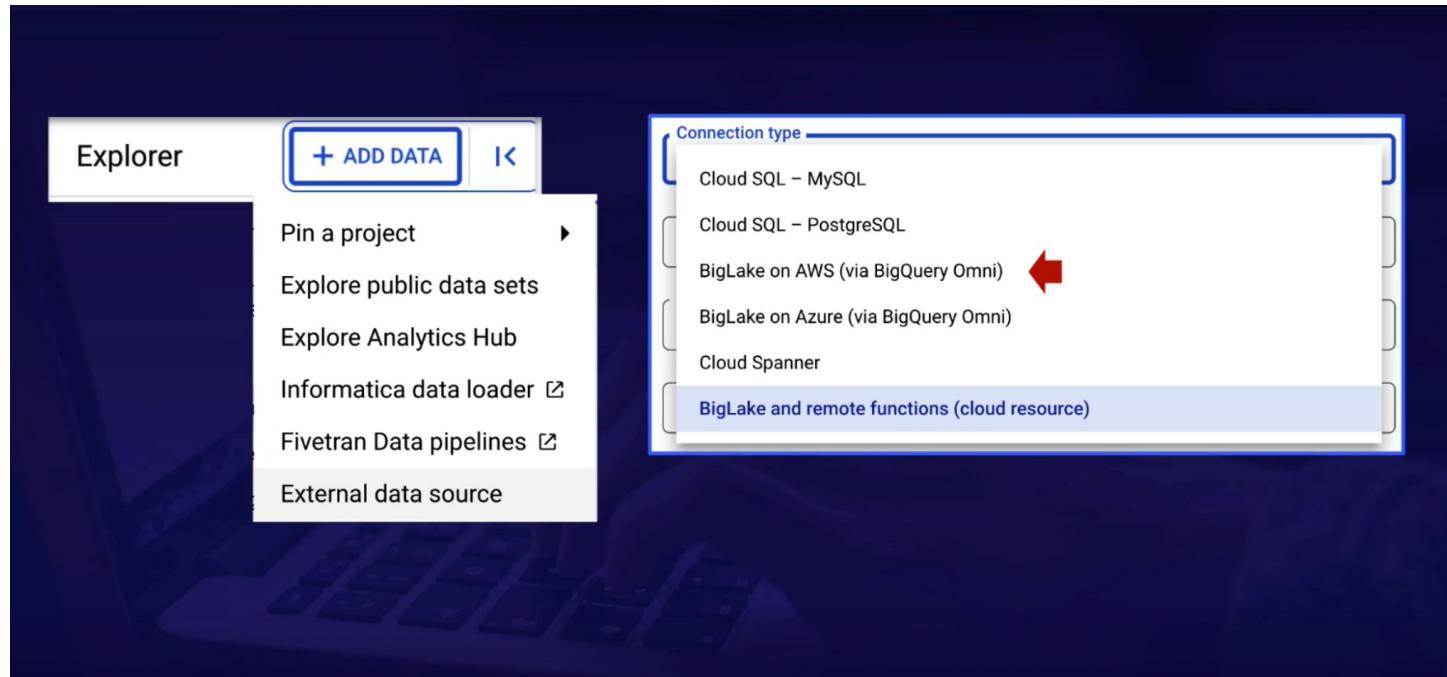
Uniformly enforce security across clouds and maintain a single copy of data.



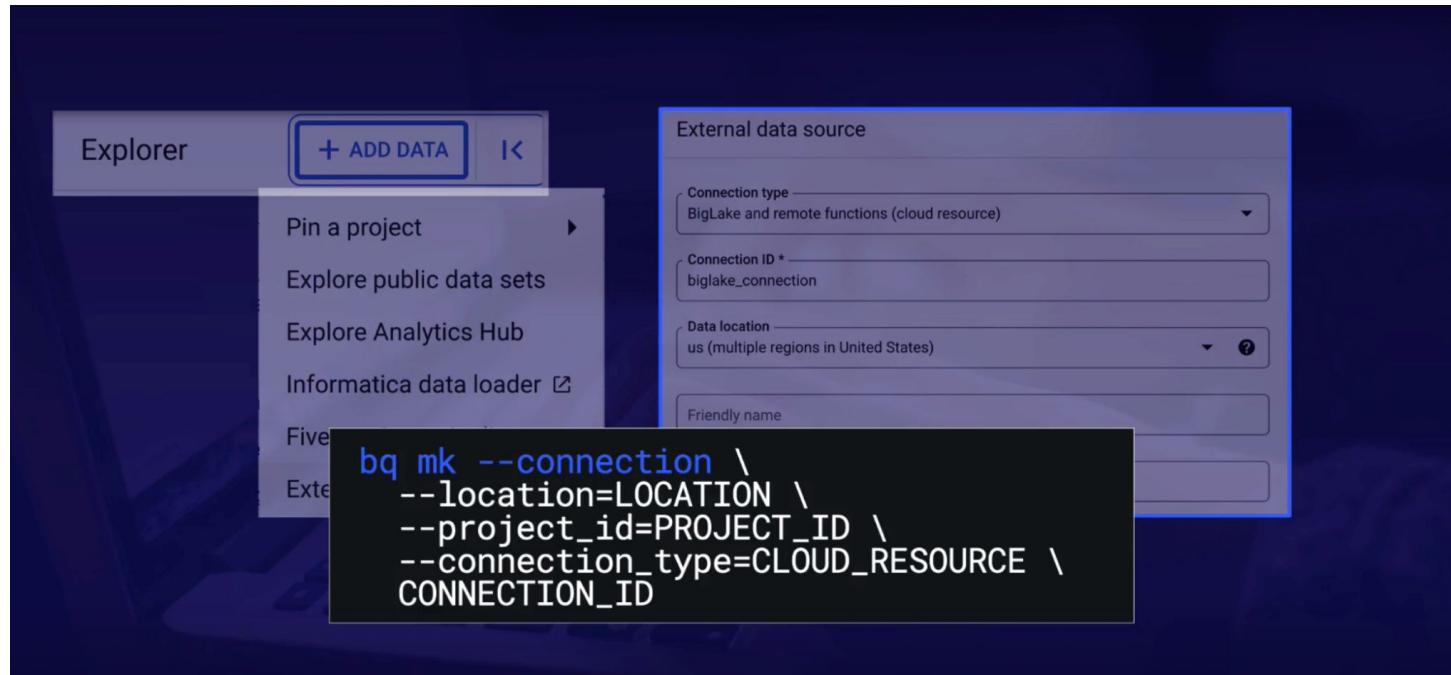
## Open Formats

Data is stored in Parquet and served in Apache Arrow.

# Big Lake Features



# Big Lake Features



# Big Lake Features

## Create a BigLake Table in Cloud Storage

```
CREATE EXTERNAL TABLE
`myproject.mydataset.ext_table_name`
WITH CONNECTION
`myproject.region.connection_id`
OPTIONS (
    format=PARQUET,
    uris=['FILE_PATH']
);
```

# Big Lake Features

## Set Up Access Control Policy (Example)

```
CREATE ROW ACCESS POLICY mypolicy  
ON mydataset.mytable  
GRANT TO ('user:victor@example.com')  
FILTER USING (year = 2022)
```

# Big Lake Features

## BigLake Use Cases

1

### Unified Data Lake and Warehouse

Interoperability between data lakes and warehouses, with a single copy of data and uniform management.

2

### Centralized Data Governance

Enforce fine-grained access controls and governance over multicloud data lakes.

3

### Extend the Power of BigQuery into Data Lakes

Run performant analytics and build ML models without moving data.



# Big Lake Features

## BigLake Use Cases

1

### Unified Data Lake and Warehouse

Interoperability between data lakes and warehouses, with a single copy of data and uniform management.

2

### Centralized Data Governance

Enforce fine-grained access controls and governance over multicloud data lakes.

3

### Extend the Power of BigQuery into Data Lakes

Run performant analytics and build ML models without moving data.



# Takeaways

- BigLake is storage engine that unified the data lake and data warehouses
- BigLake is built on open data formats and offers unified, multi cloud data governance and fine grained security.
- A BigLake table can be used like another BigQuery Table, while the data may be in the Cloud Storage, AWS, or Azure

# References

A cloud guru : BigQuery and the modern data warehousing





# Thank you!

**nothin' is impossible until its done**