

**Google** Cloud Platform

# BigQuery Functional Overview

BigQuery for Data Analysts

V1.2

*Approximate timing: 90 minutes*

# Agenda

1 BigQuery Organization

2 BigQuery Storage

3 BigQuery Architecture

4 Interacting with BigQuery

5 Quiz & Lab

# BigQuery Project Structure

- Project
  - Top-level structure
  - Contains users, APIs, authentication, billing, data, access control lists (control access to datasets and jobs)
- Dataset
  - Named parent of 1 or more tables
- Table
  - Columnar structure that stores data
- Job
  - Controls potentially long-running actions

## Notes:

**Projects.** A project contains information such as subscribed service API(s), authentication information, billing information and [Access Control Lists](#) (ACLs) that determine access to the Datasets and the Jobs. Projects are created and managed using the [APIs Console](#). For information about the related API type, see [Projects](#).

**Datasets.** A dataset is a grouping mechanism that holds zero or more tables. A dataset is the lowest level unit of access control. You cannot control access at the table level. A dataset is contained within a specific project. Each dataset can be shared with individual users. Datasets are also referenced in the SQL statements when interacting with BigQuery. For information about the related API type, see [Datasets](#).

**Tables.** Row-column structures that contain actual data. They belong to a Dataset. You cannot control access at the table level, you do it at Dataset level. For information about the related API type, see [Tables](#).

**Jobs.** Jobs are used to start all potentially long-running actions, such as queries, table import, and export requests. Shorter actions, such as list or get requests, are not managed by a job resource. For information about the related API type, see [Jobs](#). Each job has a job id. A very good use of job id is

when you load a large dataset. BigQuery rejects load job with the same job id. Therefore, guaranteeing that data would not be loaded twice.

A great benefit to point out here: You do not need to duplicate data across many clusters to achieve resource separation. Just use ACLs.

# Projects

**Project** (billing, top-level container)

**Dataset** (organization, access control)

**Table** (data with schema)

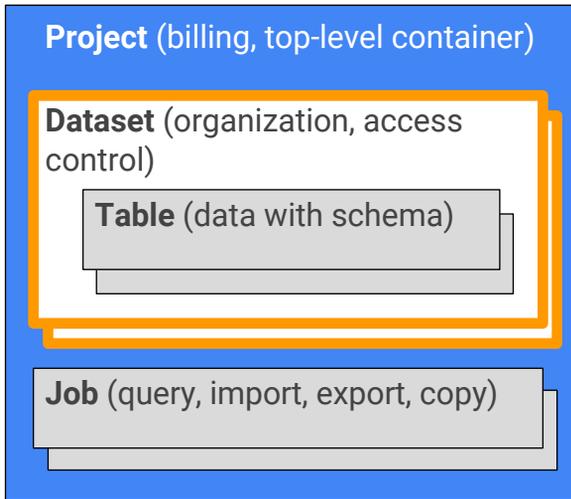
**Job** (query, import, export, copy)

- Root namespace for objects
- Manages billing
- Manages users
- Manages user privileges
- Contains one or more datasets
- Contains jobs
- Contains access control lists and IAM roles

## Notes:

Datasets are owned by projects, which control billing and serve as a global namespace root - all of the object names in BigQuery are relative to the project.

# Datasets

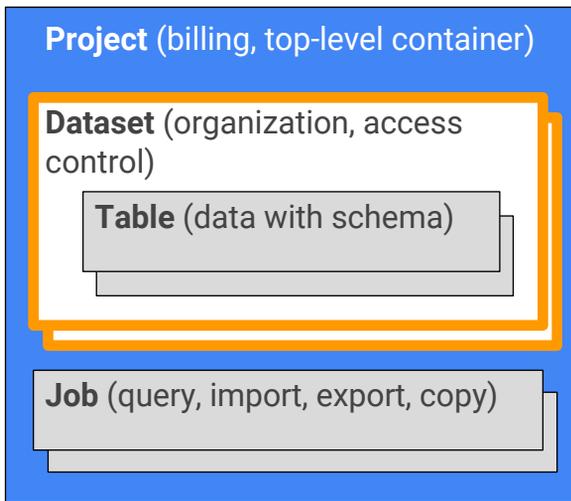


- Contain a collection of tables, views
- Access control applied to all tables/views in dataset
- ACLs for Readers, Writers, and Owners
- Access can be granted to datasets for users who are not members of the project

## Notes:

Datasets are the lowest level of ACL. BigQuery currently does not manage access to individual tables or views within the dataset. Later in the course, we will show how you can implement access control to a table through the use of views which would reside in a separate dataset can would limit access to the base level tables.

# Tables



- Collection of columns, rows
  - Data stored in managed storage
- Have a schema
  - Describes strongly-typed columns of values
- Views are supported
  - Virtual tables defined by SQL query
- Can be external (federated)
  - Query Google Cloud Storage directly (discussed later)

## Notes:

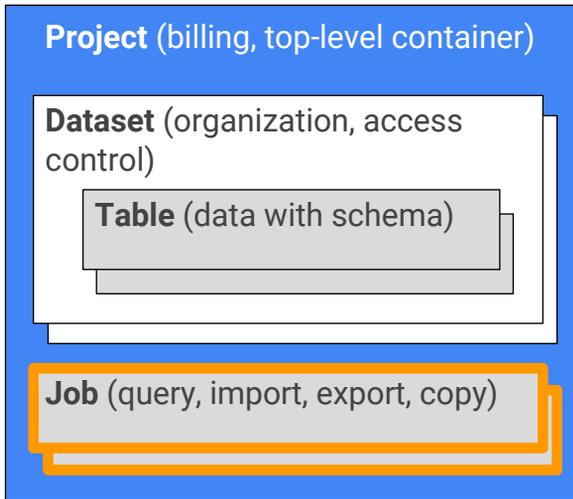
Before you can query your data, you must first load it into BigQuery or set up the data as a [federated data source](#). You can bulk load the data by using a [job](#), or [stream records individually](#). Alternately, you can skip the loading process by setting up a table as a federated data source.

Load jobs support three data sources:

1. Objects in [Google Cloud Storage](#)
2. Data sent with the job or [streaming insert](#)
3. A [Google Cloud Datastore](#) backup

Loaded data can be added to a new table, appended to a table, or can overwrite a table. Data can be represented as a flat or nested/repeated schema.

# Jobs



- Used to start all potentially long-running actions
- Examples:
  - Queries
  - Importing/exporting data
  - Copying data
- Can be cancelled

# Agenda

1 BigQuery Organization

2 BigQuery Storage

3 BigQuery Architecture

4 Interacting with BigQuery

5 Quiz & Lab

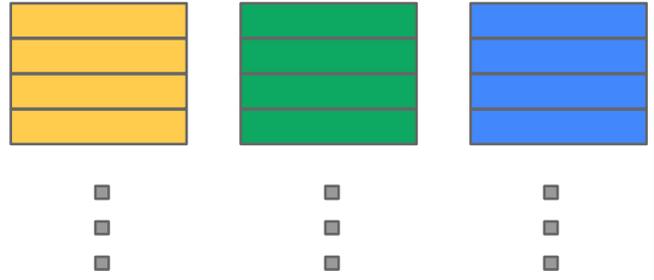
# BigQuery Storage

## Traditional RDBMS storage



Record-oriented storage

## BigQuery storage



Columnar storage

### Notes:

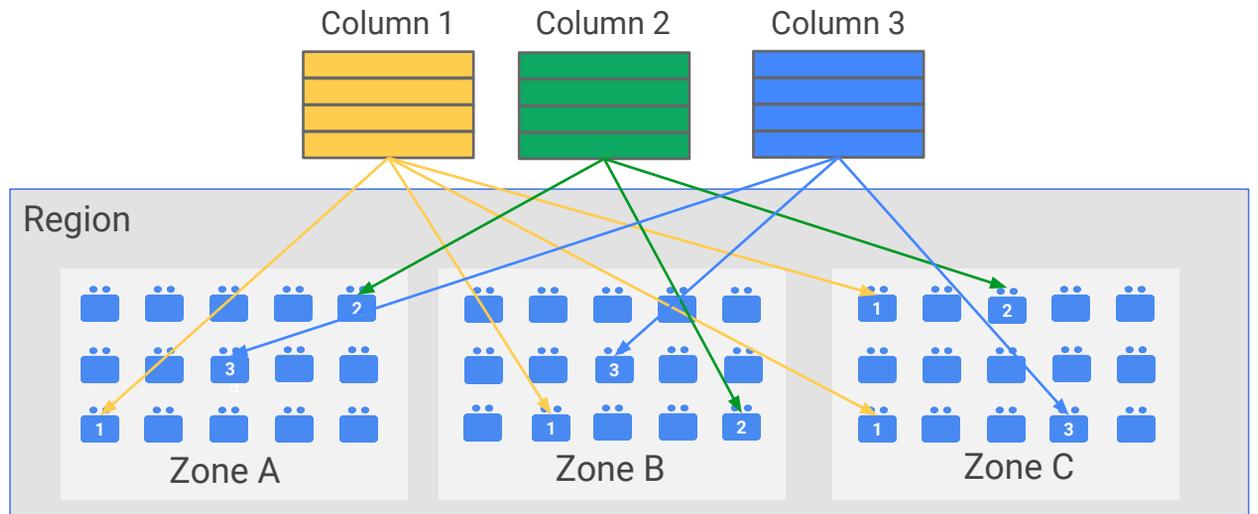
BigQuery stores data in columns.

Most queries only work on a small number of fields and BigQuery only needs to read those relevant columns to execute a query. Since each column has data of same type, BigQuery could compress the column data much more effectively.

# BigQuery Managed Storage (1 of 2)

- BigQuery data is stored on persistent disks in distributed storage
  - Each column is stored in its own file
  - Each file is compressed and encrypted on disk
  - Data is immutable
  - Storage is durable
    - Each file is replicated a minimum of 3 times in distributed storage across datacenters
- No indexes, keys, or partitions are required
- Scales to dozens of petabytes

# BigQuery Managed Storage (2 of 2)



## Notes:

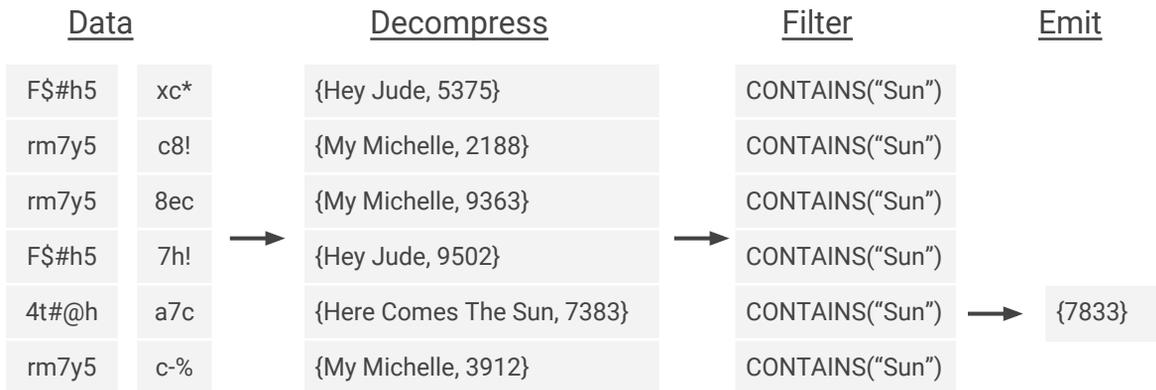
Each column in a table is stored in a data file. For example, a table that contains three columns will equate to 3 files. Each file is replicated a minimum of 3 times and dispersed across multiple zones within a region.

Each file is also compressed and encrypted while it is stored on the disk. This is done for performance and insures an additional level of security. When a column is requested, BigQuery finds the file on the first available disk.

The slide shows that a table that contains 3 columns will be stored in 3 separate files on 3 different persistent disk drives in A zone. The files will also be replicated in B zone and C zone to ensure file integrity. The example also shows that C zone has two copies of column 1 on separate disks.

# Storage Engine 2006-2015: ColumnIO

```
SELECT play_count FROM songs WHERE name CONTAINS "Sun";
```



## Notes:

Prior to 2015, ColumnIO was the storage engine for BigQuery. ColumnIO stores each field of a record in a different stream and then compresses the data. When the query is processed, the data is decompressed and the filter is applied.

# Storage Engine 2016-Now: Capacitor (1 of 2)

```
SELECT play_count FROM songs WHERE name CONTAINS "Sun";
```



## Notes:

Capacitor looks similar to ColumnIO. In Capacitor, the data is stored in a compressed columnar format, but instead of decompressing all the data, only the relevant data is decompressed. Then, it is matched against the data in compressed format and the result is emitted. There are also a couple of different compression types available.

Note that the first column contains small integers. These are indexes into the dictionary that contains the field values.

In the dictionary we see that the three values map to different Beatles song titles. To apply the filter, rather than applying the predicate for each value in the data column, it is applied once per unique value in the dictionary.

The predicate result is written to a truth table in the same order as the dictionary. Then, the data column is scanned, and the data value is used as an index into our truth table. If it is True, then the row is emitted.

Note that the predicate (which could be arbitrarily complex) is run once for each unique value in the column. As well, only the minimal amount of data decompression is done.

For more information on Capacitor, see:

<https://cloud.google.com/blog/big-data/2016/04/inside-capacitor-bigquerys-next-generation-columnar-storage-format>.

## Storage Engine 2016-Now: Capacitor (2 of 2)

- Rebuilt query engine to operate over compressed data
- Huge performance improvements:
  - Average for all queries: 2x faster
  - Selective filters: 10-1000x faster
- Lower CPU billing tiers (discussed later)

# Agenda

1 BigQuery Organization

2 BigQuery Storage

3 BigQuery Architecture

4 Interacting with BigQuery

5 Quiz & Lab

# BigQuery Processing (1 of 3)

- Massive, highly available, multi-tenant compute cluster separate from data storage
- Cluster is a combination of CPU, RAM, and networking that processes your queries
  - Hundreds of thousands of cores used ONLY for queries
  - Other workloads (load, egress) do not compete for compute resources

## Notes:

The highly parallel nature of BigQuery allows for all workers to read data from storage at the same time.

Column storage allows for BigQuery to only read the columns requested, unlike row based DBMS that read data blocks.

Single scan of data means that data from storage needs to be only read once per query.

No indexes are needed for BigQuery. Data for each column is compressed and stored in its own file.

BigQuery SQL is now ANSI SQL-compliant.

Data is immutable in BigQuery, so UPDATE and DELETE statements are not currently supported.

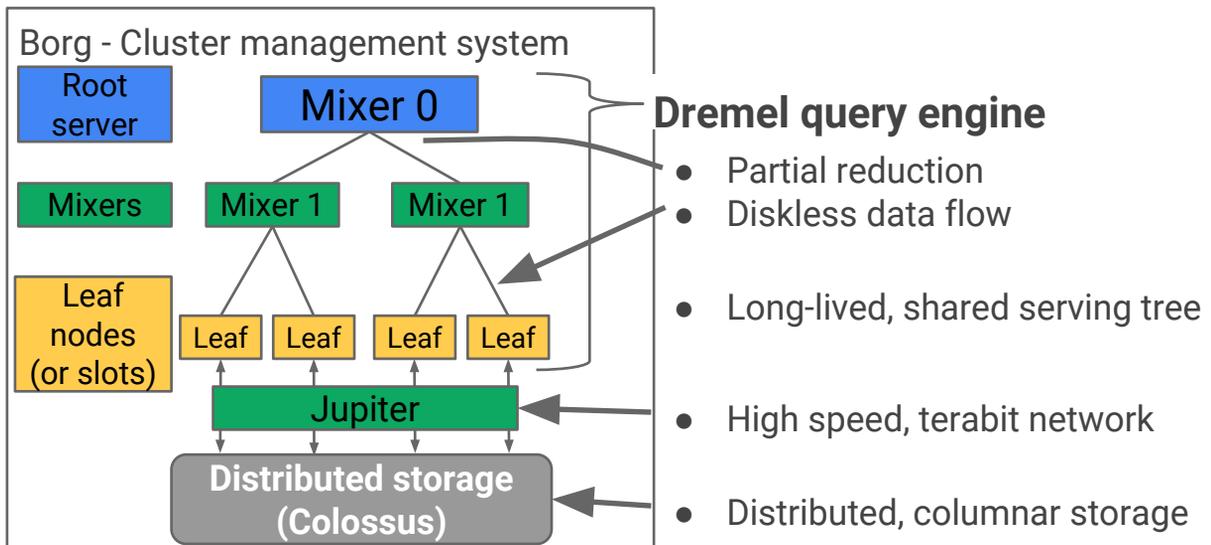
## BigQuery Processing (2 of 3)

- Compute is shared - You consume (and pay for) only the compute resources you need
  - Resource deployment is managed - You pay only for consumption
  - Slots can be used to reserve resources (covered later)
- No cluster setup is required
  - Storage is highly optimized for SQL
  - Storage and compute scale independently
  - Compute is managed - No need to scale cores

# BigQuery Processing (3 of 3)

- Petabit network allows access to storage
  - Network pipe between storage and compute is massive
    - Very different from services that read data from traditional cloud storage
  - No need to move data into the cluster
  - Single scan of data on disk
    - Storage read only once per query

# Query Processing 2006-2015



## Notes:

Dremel turns your SQL query into an execution tree.

The root server determines the leaf servers needed to retrieve data and performs record assembly, ordering, and final filtering.

The mixers perform aggregate and scalar functions against the data read by the leaf nodes.

The leaves of the tree are called 'slots', and do the heavy lifting of reading the data from Colossus and doing any computation necessary. Leaf nodes have their own local storage.

BigQuery relies on Colossus, Google's latest generation distributed file system. Each Google datacenter has its own Colossus cluster, and each Colossus cluster has enough disks to give every BigQuery user thousands of dedicated disks at a time. Colossus also handles replication, recovery (when disks crash) and distributed management (so there is no single point of failure). Colossus is fast enough to allow BigQuery to provide similar performance to many in-memory databases, but leveraging much cheaper yet highly parallelized, scalable, durable and performant infrastructure.

BigQuery leverages the ColumnIO columnar storage format and compression algorithm to store data in Colossus in the most optimal way for reading large

amounts of structured data. Colossus allows BigQuery users to scale to dozens of Petabytes in storage seamlessly, without paying the penalty of attaching much more expensive compute resources – typical with most traditional databases.

To give you thousands of CPU cores dedicated to processing your task, BigQuery takes advantage of Borg, Google's large-scale cluster management system. Borg clusters run on dozens of thousands of machines and hundreds of thousands of cores.

Besides obvious needs for resource coordination and compute resources, Big Data workloads are often throttled by networking throughput. Google's Jupiter network can deliver 1 Petabit/sec of total bisection bandwidth, allowing us to efficiently and quickly distribute large workloads. Jupiter networking infrastructure might be the single biggest differentiator in Google Cloud Platform. It provides enough bandwidth to allow 100,000 machines to communicate with any other machine at 10 Gbs.

## **B-Tree**

Another way to look at this is it's a B-Tree index built on the fly. The difference here is that each node is a computational instance (a compute node with memory, disk and cpu). The result of the architecture and query processing is that a set of computers becomes a parallelized computational tree that read columnar data very efficiently.

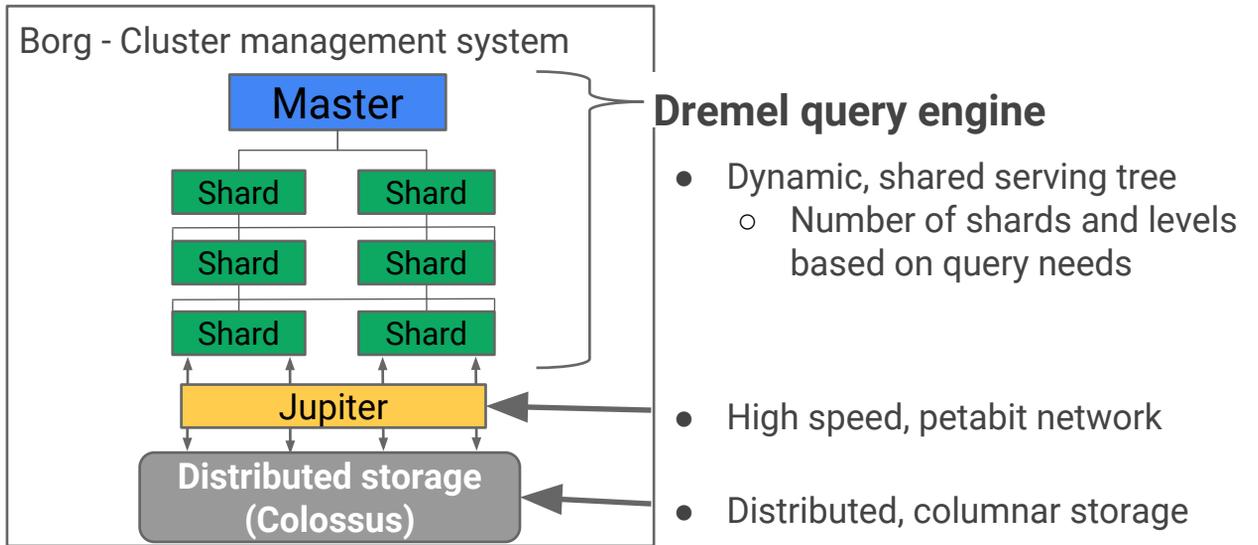
## **Map Reduce**

It's also like a map reduce with the bottom row being the mappers, the reducers and controller. But

- you don't have to shuffle.
- you have a very high bandwidth between the mapper and reducer
- it's very easy to do multiple rounds because it's essentially just another RPC.

Many of the aggregation functions can be parallelized.

# Query Processing 2015-Present



## Notes:

- Now, BigQuery has a dynamic serving tree that looks different for each query (sized by query needs)
- BigQuery can handle complex queries much better (multi-stage queries are several times faster)
- Multiple stages, joins, are a separate level in the tree
- BigQuery does not need to do several passes through the tree as it did previously when the tree was static

# Example - Query Processing (1 of 2)

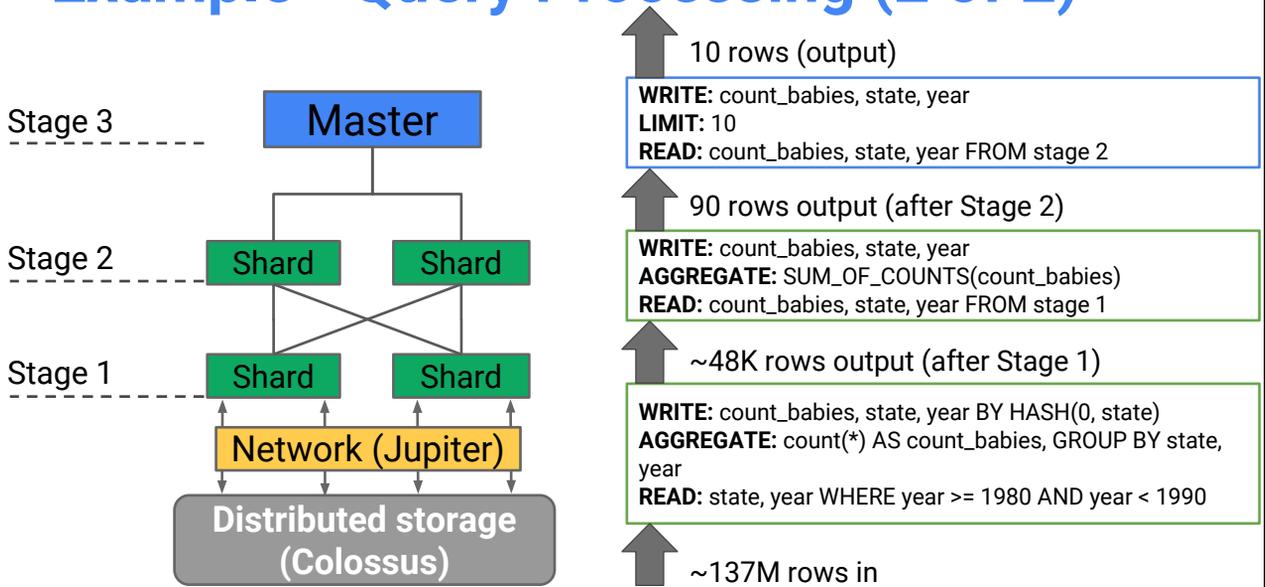
Count of babies by  
state, year

```
SELECT state, year, COUNT(*) AS count_babies
FROM 'bigquery-public-data.samples.natality'
WHERE year >= 1980 and year < 1990
GROUP BY state, year
ORDER BY 3 desc
LIMIT 10
```

## Notes:

Here is an example query that the next slide will dissect to show how queries are processed.

# Example - Query Processing (2 of 2)



## Notes:

- Because of the columnar data format, only two data fields need to be read from the storage (state, and year) in this query.
- With a distributed file system, the workers read in chunks of the data in parallel, and apply the filter and aggregation. Shuffling may also occur between stages. The number of workers processing data at each stage is dynamically allocated according to the needs of the query.
- The workers pass the data to the master in the final stage. The master does the final aggregation, sorting, limiting, and returning the data to the client.

# Benefits of New Architecture

## Dynamic execution

Automatic sizing and query planning.

No more EACH keyword

Helps with JOIN, GROUP BY, PARTITION, and large query results

## Poseidon

5 times faster to get data into BigQuery

New load file formats (AVRO) (10x faster than JSON imports)

## Consistent scheduling

Fair scheduling between customers

Each customer can use up to **2000** parallel workers

Predictable performance

### Notes:

*Dynamic Execution:* There is no need to use the EACH keyword. With dynamic query planning, BigQuery calculates how to run (size) your query.

*Poseidon:* The data ingestion mechanism for BigQuery is now 5x faster. BigQuery also supports the AVRO storage format which is up to 10x faster.

*Consistent scheduling:* Consistent scheduling ensures your queries do not interfere with queries running for other customers (by default you get 2000 workers). If you need more than 2000 workers, contact your sales representative.

# Query Plan Explanation (1 of 2)

- BigQuery provides diagnostic information on completed query's execution plan
  - Similar to EXPLAIN statement available in some other query engines
  - Can often use this information to improve query performance
- After a query runs, click Explanation to view metadata
- Information on previous slide is from Explanation

# Query Plan Explanation (2 of 2)

- Example using the previous natality query:



- Additional details:

<https://cloud.google.com/bigquery/query-plan-explanation>

# Agenda

1 BigQuery Organization

2 BigQuery Storage

3 BigQuery Architecture

4 Interacting with BigQuery

5 Quiz & Lab

# BigQuery Web User Interface

- Browser-based graphical user interface  
<https://bigquery.cloud.google.com/>
- Upload and import data
- Build and execute queries and view results
- Switch projects, create, list, maintain datasets and tables
- View schemas, and metadata
- Download, save, and export data
- View, edit, and re-execute jobs from history

# Web UI Components (1 of 3)

Table name is represented as follows:  
Current Project: `<dataset>.<table name>`  
Different Project: `<project>:<dataset>.<table>`  
Example: `publicdata:samples.wikipedia`

Write a new query  
Project name  
Dataset

The screenshot displays the Google BigQuery web interface. On the left, a sidebar shows the 'Big Query Training Project' with a tree view containing datasets like 'flight\_data', 'reference', and 'zip\_codes'. The 'zip\_codes' dataset is selected. The main area is titled 'New Query' and contains a query editor with the text: `1 SELECT FROM [reference.zip_codes] LIMIT 1000`. Below the editor are buttons for 'RUN QUERY', 'Save Query', 'Save View', 'Format Query', and 'Show Options'. At the bottom, the 'Table Details: zip\_codes' section shows a 'Schema' table with columns 'zip' and 'type', both of type 'STRING' and 'NULLABLE'.

# Web UI Components (2 of 3)

The screenshot displays the BigQuery web interface. At the top, a 'New Query' section contains a SQL editor with the query: `1 SELECT FROM [reference.zip_codes] LIMIT 1000`. Below the editor are buttons for 'RUN QUERY', 'Save Query', 'Save View', 'Format Query', and 'Show Options'. The 'Table Details: zip\_codes' section shows a schema table with columns 'zip', 'STRING', 'NULLABLE', and 'Describe this field...'. The 'Show Options' panel is open, showing settings for 'Destination Table', 'Write Preference', 'Results Size', 'Results Schema', 'Query Caching', 'Query Priority', 'UDF Source URIs', 'Maximum Billing Tier', and 'SQL Version'. The 'SQL Version' option is checked for 'Use Legacy SQL'.

- Send results to new or existing tables
- Allow for large result sets
- Turn data caching on or off
- Change query priority
- Edit User Defined Functions (UDFs)

## Notes:

Destination table - write the query result to the selected table.

Result size - the maximum query response size is 128M compressed. When "Allow Large Results" is enabled, there is no limit to the result size.

Query caching - use cached result if available. You aren't charged for cached queries.

Query priority - running query in batch (delayed) or interactive mode

UDF source URIs – use to reference an external User-Defined Function (UDF), written in Javascript. Similar to the "Map" function in a MapReduce: it takes a single row as input and produces zero or more rows as output. The output can potentially have a different schema than the input.

SQL Version - use BigQuery SQL (when turned off, uses Standard SQL)

# Web UI Components (3 of 3)

New Query ?

```
1 SELECT state, year, COUNT(*) AS count_babies
2 FROM [bigquery-public-data:samples.natality]
3 WHERE year >= 1980 and year < 1990
4 GROUP BY state, year
5 ORDER BY 3 desc
6 LIMIT 10
```

Estimated size before running the query and validation of query syntax

Valid: This query will process 1.48 GB when run.

RUN QUERY

Save Query

Save View

Format Query

Show Options

Query complete (2.8s elapsed, 1.48 GB processed) ✓

Results Explanation

Download as CSV

Download as JSON

Save as Table

Save to Google Sheets

Row	state	year	count_babies
1	CA	1989	570365
2	CA	1988	533429

Saving/sharing result in CSV, JSON, table, or Sheets

## BigQuery CLI (1 of 2)

- Python-based tool to access BigQuery
- Best option for creating command scripts
- Provides more options and features than Web UI
- Allows for asynchronous submission of commands
- Provides an interactive mode
- Use on client machine or Compute Engine instance by installing Google Cloud SDK
- Also accessible via Cloud Shell

# BigQuery CLI (2 of 2)

- Use from shell script or from most languages
  - `subprocess.call()`
  - `Runtime.getRuntime().exec()`
  - `system()`, and so on
- bq manages authentication, authorization

USAGE: `bq.py [--global_flags] <command> [--command_flags] [args]`

Any of the following commands:

`cp, extract, head, help, init, insert, load, ls, mk, query, rm, shell, show, update, version, wait`

**cp** Copies one table to another.

Examples:

`bq cp dataset.old_table dataset2.new_table`

**extract** Perform an extract operation of `source_table` into `destination_uris`.

Usage:

`extract <source_table> <destination_uris>`

Examples:

`bq extract ds.summary gs://mybucket/summary.csv`

Arguments:

`source_table`: Source table to extract.

# CLI Example

bq CLI

```
$ bq load ds.new_tbl ./info.csv
./info_schema.json
$ bq insert dataset.table /tmp/mydata.json
$ echo '{"a":1, "b":2}' | bq insert dataset.table
...
$ bq query 'select count(*) from
publicdata:samples.shakespeare'
```

# Interactive Shell Example

Interactive shell

```
$ bq shell
Welcome to BigQuery! (Type help for more
information.)
project-id> ls
datasetId
  -----
  mydataset
project-id> exit
Goodbye.
```

# API Library

- RESTful API allows for programmatic interface to BigQuery
- Supports several languages
  - Java, Python, JavaScript, Ruby, PHP
  - Google Apps Script
- Library modules handle authorization via OAuth2
  - Provide client ID and client secret
- See:

<https://developers.google.com/bigquery/client-libraries>

# Agenda

1 BigQuery Organization

2 BigQuery Storage

3 BigQuery Architecture

4 Interacting with BigQuery

5 Quiz & Lab

## Module Review

Which of the following statements are true?  
(select **2** of the available options)

- There are two interfaces you can use to access BigQuery
- Table data is stored in one zone
- A dataset belongs to a single project and contains tables
- BigQuery is highly parallel and distributed

# Lab

## Exploring BigQuery interfaces



Image by  
Cennie Zhou

# Resources

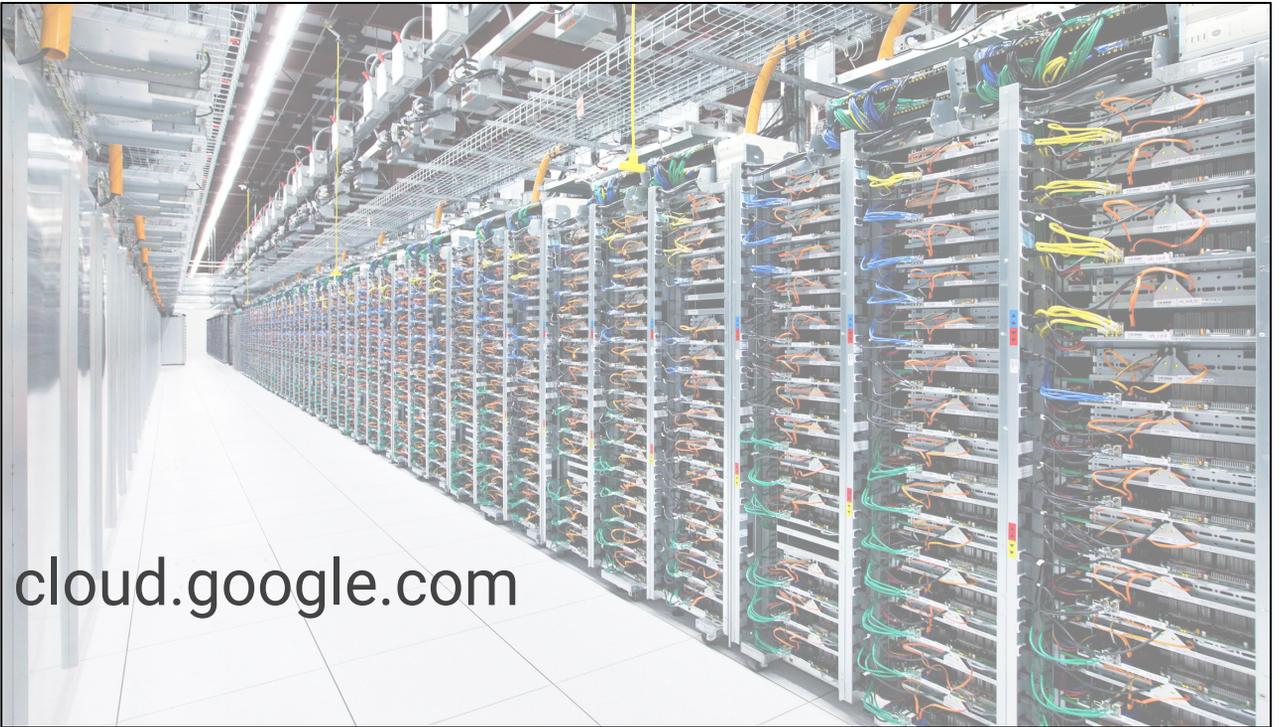
- BigQuery fundamentals  
<https://cloud.google.com/bigquery/what-is-bigquery#fundamentals>
- Google Cloud BigQuery command-line tool  
<https://developers.google.com/bigquery/bq-command-line-tool>
- Big data blog - BigQuery under the hood  
<https://cloud.google.com/blog/big-data/2016/01/bigquery-under-the-hood>

# Module Review Answers

Which of the following statements are true?

*(select 2 of the available options)*

- There are two interfaces you can use to access BigQuery
- Table data is stored in one zone
- A dataset belongs to a single project and contains tables
- BigQuery is highly parallel and distributed



[cloud.google.com](https://cloud.google.com)