Google

Introduction to Google BigQuery

Ido Flatow

Cloud Solutions Architect, Google Cloud, EMEA

BigQuery Introduction

What is BigQuery?

- BigQuery is Google Cloud Platform's data warehouse solution to perform high speed, scalable and interactive analysis on data.
- It sits under the Big Data product category for Google Cloud Platform and is useful for storing petabytes of data as well as performing analysis on that data.
- It is built on the principle that double the amount of data queried should not take double the time to return results.
- BigQuery data can also be used in other tools like Google's Data Studio, Data Lab and others.

Google BigQuery

BigQuery: 100% serverless data warehouse

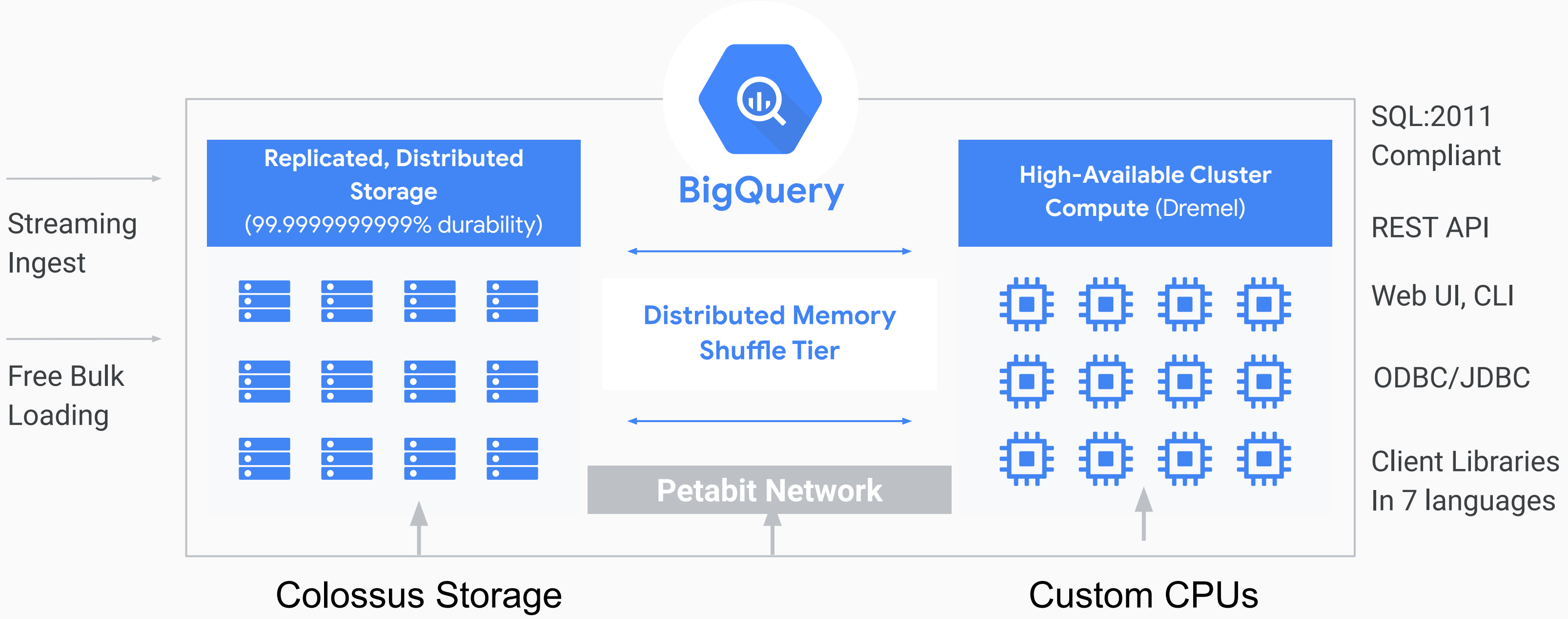
- Google Cloud’s Enterprise Data Warehouse for Analytics
- Petabyte-Scale and Fast Convenience of Standard SQL
- Encrypted, Durable and Highly Available
- Fully Managed and Serverless

OLAP vs OLTP: Which fits my use case?

	OLTP	OLAP
	OnLine Transaction Processing	OnLine Analytical Processing
Data Source	Operational	Historical
Focus	Updating/Retrieve	Reporting
Queries	Simple	Complex
Query Latency	Low	High
Google Cloud Platform Products	Cloud SQL, Cloud Datastore, Cloud Spanner, BigTable	BigQuery

BigQuery | Architecture

Decoupled storage and compute for maximum flexibility



Gmail, YouTube, AdWords,
Machine Learning, Search

BigQuery Structure

Project

1 Main Containers in BigQuery are called Projects, they store billing information and list of users with access.

Data Set

2 A Project has datasets that allow us to organize and control the access to the tables that are contained in them.

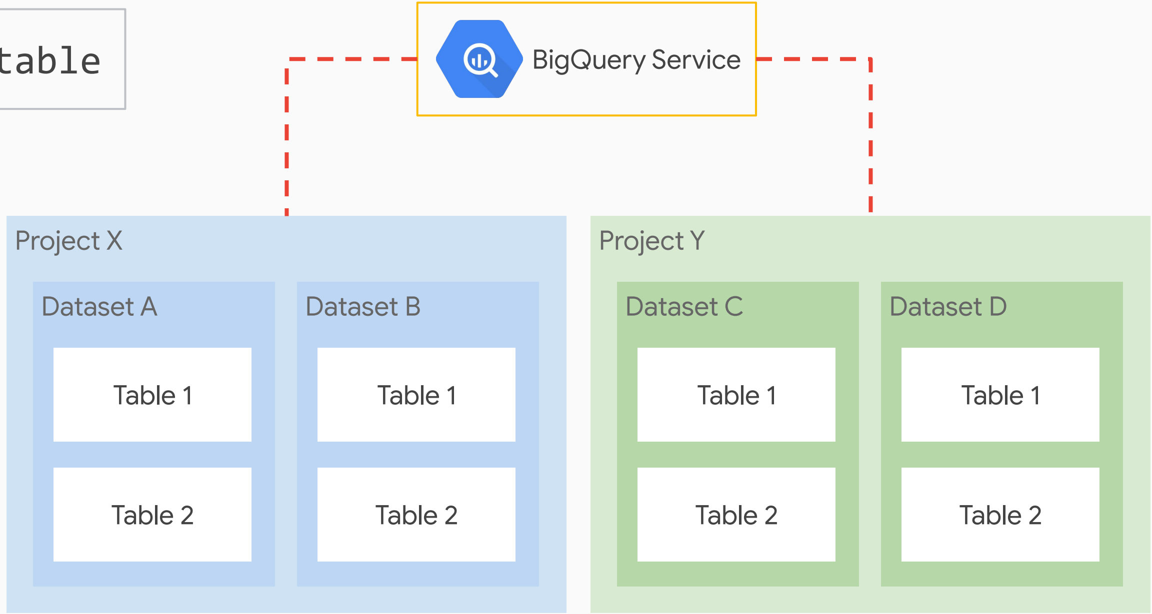
Tables

3 Each dataset has tables that contain the data following a schema that describes the field names, types and other useful information that we can query.

BigQuery Structure

BigQuery organizes data tables into units called datasets

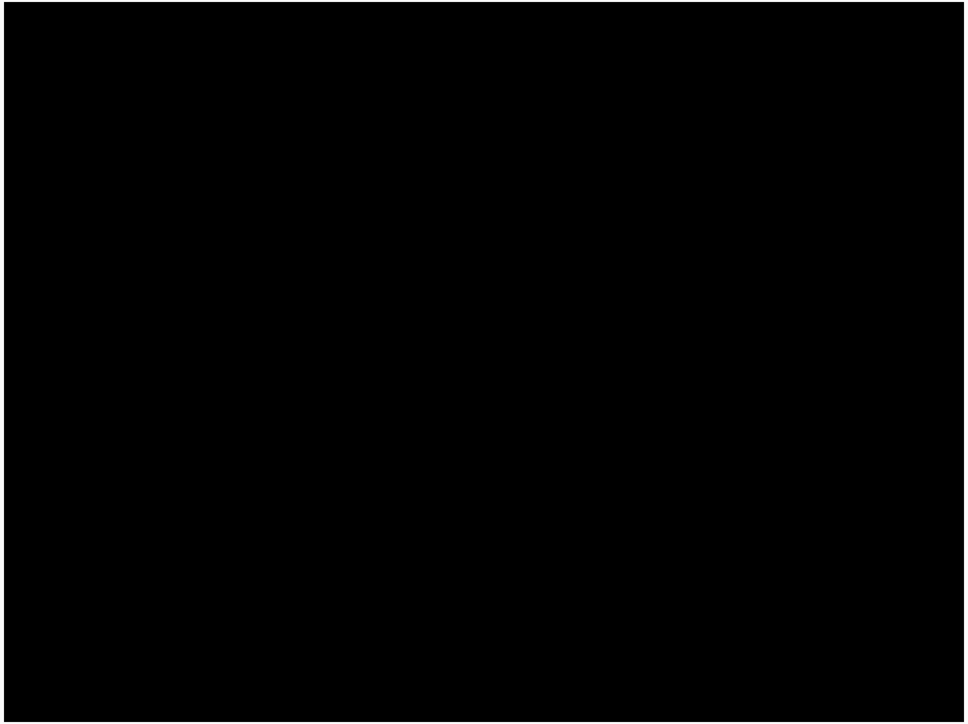
```
project.dataset.table
```



What are some reasons to structure your information into:

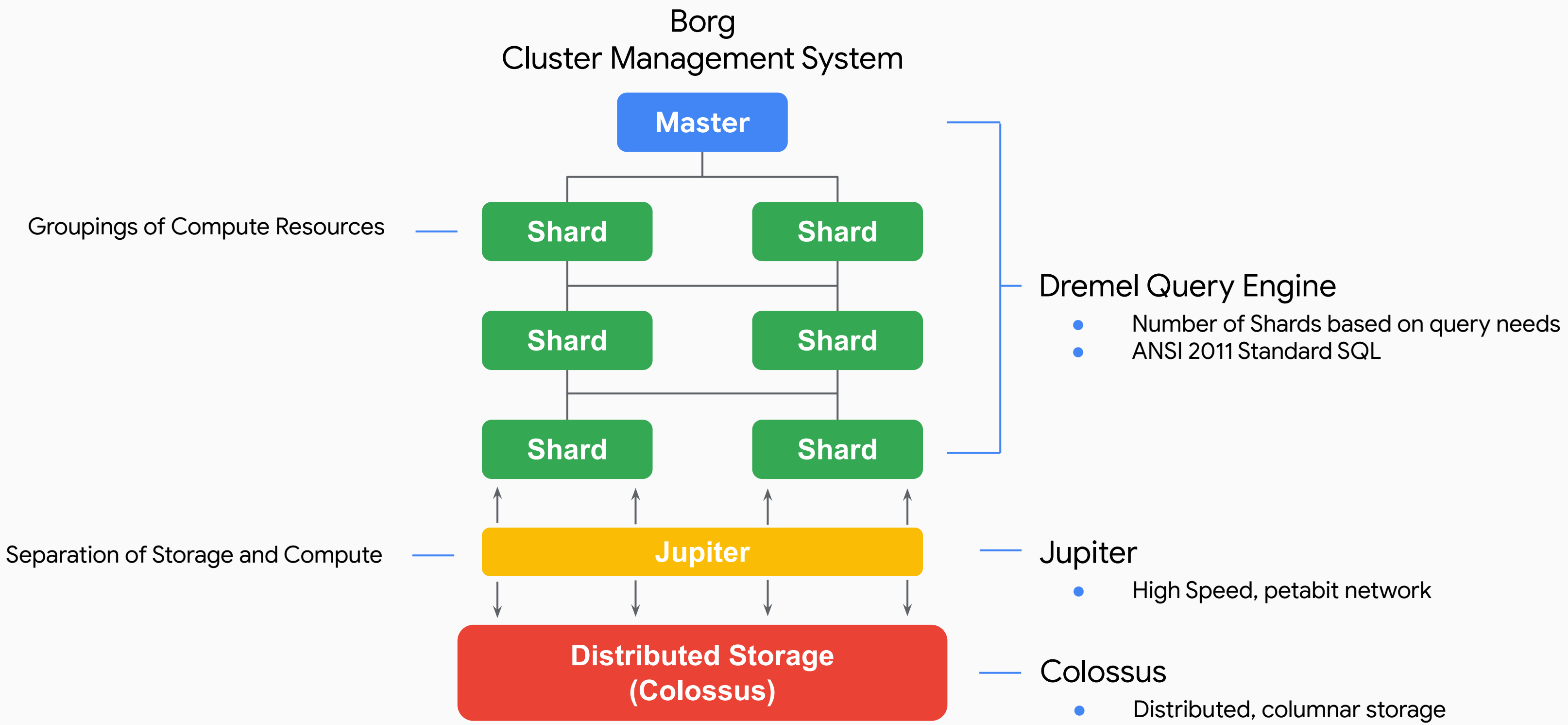
- Datasets?
- Projects?
- Tables?

BigQuery Interface Walkthrough

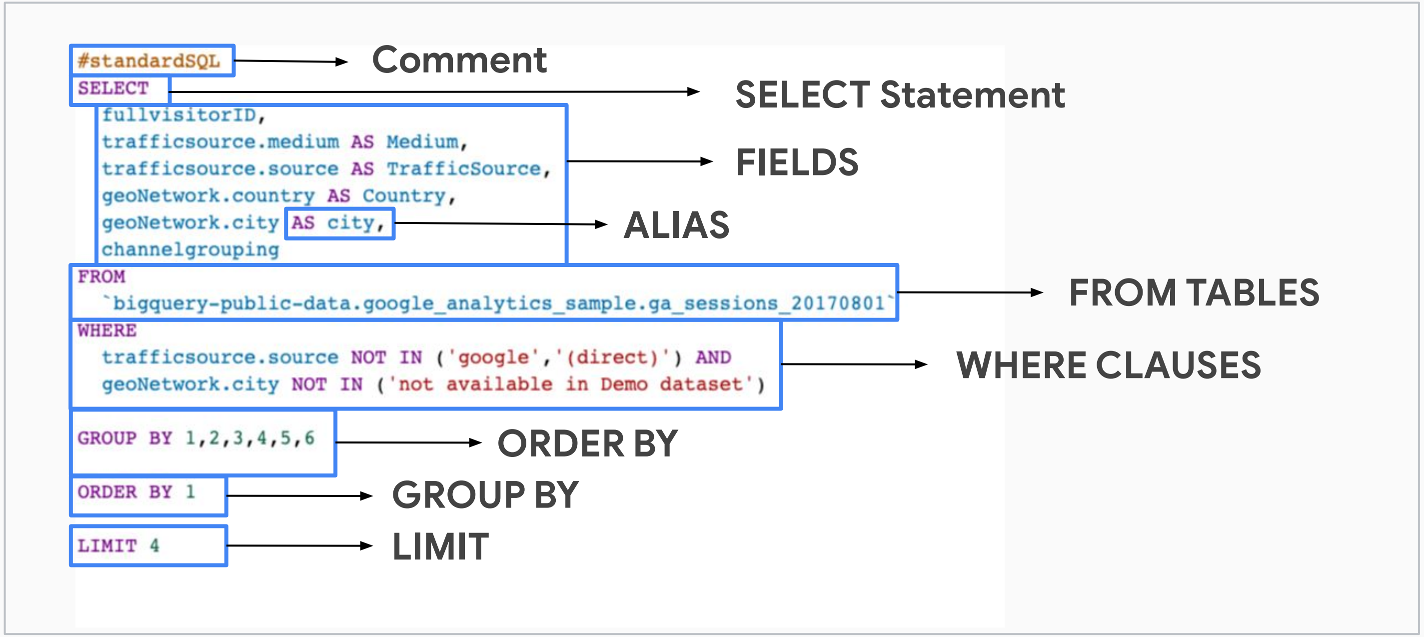


BigQuery Query Engine

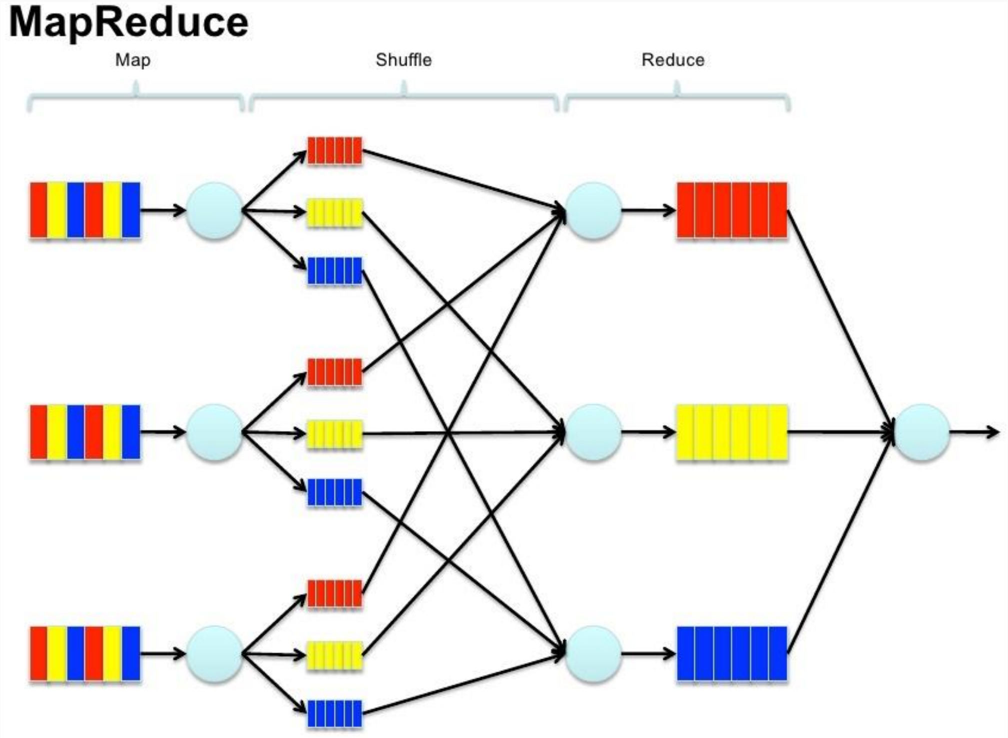
The Power of BigQuery



Query Syntax



MapReduce

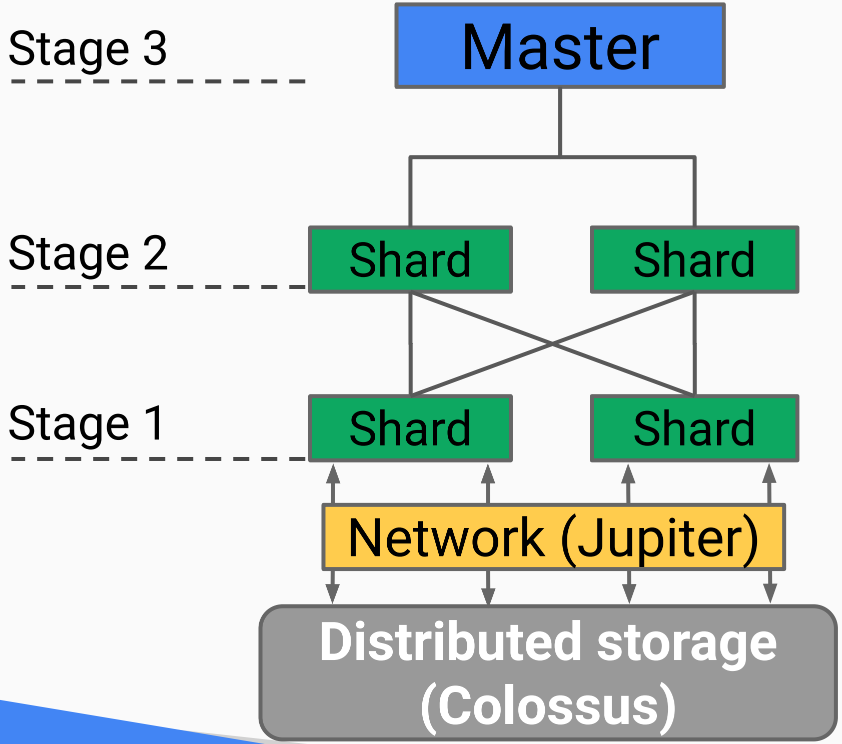


Query Processing Example

Count of babies by state, year

```
#StandardSQL
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
```

Query Processing Example



10 rows (output)

WRITE: count_babies, state, year
LIMIT: 10
READ: count_babies, state, year FROM stage 2

90 rows output (after Stage 2)

WRITE: count_babies, state, year
AGGREGATE: SUM_OF_COUNTS(count_babies)
READ: count_babies, state, year FROM stage 1

~48K rows output (after Stage 1)

WRITE: count_babies, state, year BY HASH(0, state)
AGGREGATE: count(*) AS count_babies, GROUP BY state, year
READ: state, year WHERE year >= 1980 AND year < 1990

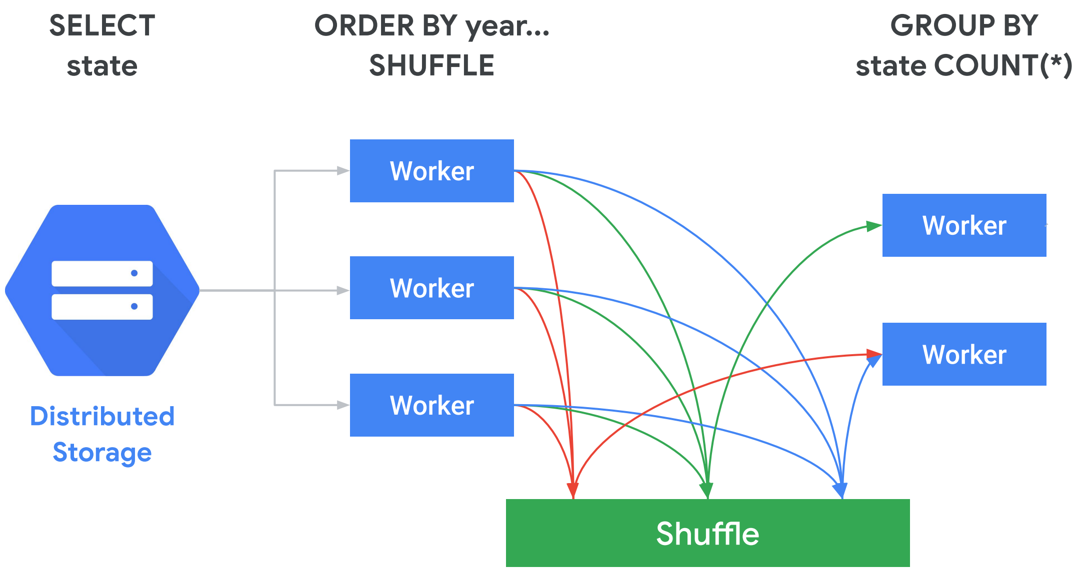
~137M rows in

BigQuery remote memory shuffle

Faster performance for complex queries

Join and aggregate more data

Better scalability



Google Cloud

Let’s do a (bigger) query …

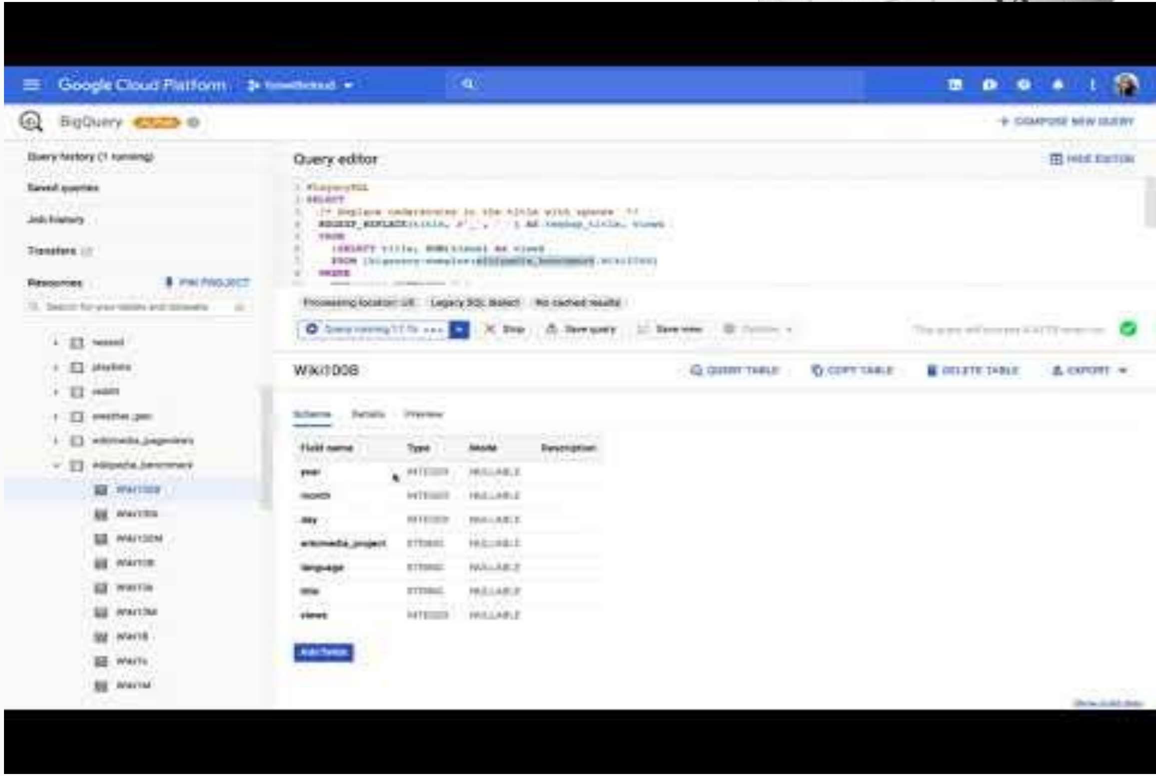
```
#standardsql
SELECT
  /* Replace underscores in the title with spaces  */
  REGEXP_REPLACE(title, r'_', ' ') AS regexp_title, views
  FROM
    (SELECT title, SUM(views) as views
   FROM `bigquery-samples.wikipedia_benchmarkWiki`
  WHERE
    NOT title like '%:%'
    AND wikimedia_project='wp'
    AND language='en'
    /* Match titles that start with 'G', */
    /* end with 'e', and contain two 'o's  */
    AND REGEXP_CONTAINS(title, r'^G.*o.*o.*e$')
  GROUP BY
    title
  ORDER BY
    views DESC
  LIMIT 100)
```

Find most viewed wiki articles that contain regex:

^G.*o.*o.*e$

Let’s do a (bigger) query ...

```
#standardsql
SELECT
  /* Replace underscores in the title with spaces  */
  REGEXP_REPLACE(title, r'_', ' ') AS regexp_title,
views
  FROM
    (SELECT title, SUM(views) as views
    FROM `bigquery-samples.wikipedia_benchmark. Wiki`
  WHERE
    NOT title like '%:%'
    AND wikimedia_project='wp'
    AND language='en'
    /* Match titles that start with 'G', */
    /* end with 'e', and contain two 'o's  */
    AND REGEXP_CONTAINS(title, r'^G.*o.*o.*e$')
  GROUP BY
    title
  ORDER BY
    views DESC
  LIMIT 100)
```



Serial vs Parallel

- **BigQuery Execution Time: 38 seconds**
  - 4TB of data read
  - 100 billion regular expressions run
  - 276 GB shuffled (read post-filter)
- **Serial execution times for each task**
  - 11.6 hrs to read 4TB from disk (@ 100MBps)
  - 27 hrs to run 100b regexps (@1 µsec each)
  - 37 minutes to shuffle 278 GB across the network (@ 1Gbps)

Serial execution takes almost 40 hrs.

Some BigQuery Stats

10.5 Trillion	Largest query (rows)
2.1 petabytes	Largest query (data size)
62 petabytes	Largest storage customer
4.5 million rows/sec	Peak ingestion rate

Simple query execution - explain plan


```
SELECT COUNT(*) FROM
wikipedia_benchmark.Wiki1B
WHERE title LIKE "G%o%o"
```

Simple query execution - more data

	Stage timing				Rows	
	Wait	Read	Compute	Write	Input	Output
Stage 1					10,677,046,566	2,136
Stage 2					2,136	1

```
SELECT COUNT(*) FROM
wikipedia_benchmark.Wiki10B
WHERE title LIKE "G%o%o"
```

Stage 1					106,770,465,660	21,355
Stage 2					21,355	1

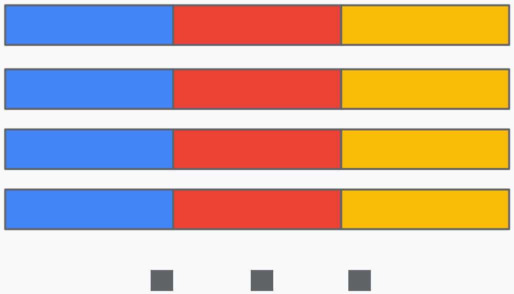
```
SELECT COUNT(*) FROM
wikipedia_benchmark.Wiki100B
WHERE title LIKE "G%o%o"
```

BigQuery Storage

BigQuery Structure

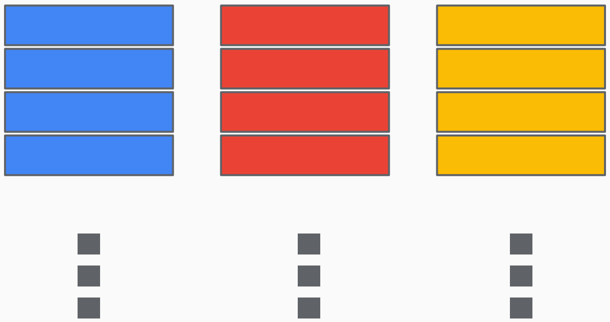
BigQuery Storage is columnar

Relational Database



Record Oriented Storage
Supports transactional updates

BigQuery Storage



Each column is seperate, compressed, encrypted file, replicated three times. No indexes, keys or partitions required; for immutable massive datasets.

Row based Storage

UID	First Name	Last Name	Age	City
42398714298	Jessie	Walters	20	New York
349872349873	Ashuk	Patel	34	Chicago
3498734871	Lisa	LaBlenc	54	Austin
34598792358	Greg	Smith	28	Boston

pur_id	UID	Product	Price
3459872980	349872349873	Google Home	100.01
38479782	349872349873	Pixel 4	550.5
8937492	349872349873	Pixel 4 case	20.23
349872735	34598792358	Google Home	100.01

- Read less data faster
- Skip unused columns
- Column compression > Row Compression
- Supports vectorized columnar processing

BigQuery Capacitor Files

UID	First Name	Last Name	Age	City	Purchases
349872349873	Ashuk	Patel	34	Chicago	[{ "pur_id": 3459872980, "UID": 349872349873, "Product": "Google Home", "Price": 100.01 }, { "pur_id": 38479782, "UID": 349872349873, "Product": "Pixel 4", "Price": 550.5 }, { "pur_id": 8937492, "UID": 349872349873, "Product": "Pixel 4 case", "Price": 20.23 }]
42398714298	Jessie	Walters	20	New York	[{ ""pur_id"": 3459872980, ""UID"": 349872349873, ""Product"": ""Google Home"", ""Price"": 100.01 }]
3498734871	Lisa	LaBlenc	54	Austin	[{...}]
34598792358	Greg	Smith	28	Boston	{...}]

Storage Engine: Capacitor

```
SELECT play_count FROM songs WHERE name LIKE "%Sun%";
```

Data

0	xc*
1	c8!
1	8ec
0	7h!
2	a7c
1	c-%

Emit

{7833}

Dictionary

0	Hey Jude
1	My Michelle
2	Here Comes the Sun

Filter

LIKE "%Sun%"
LIKE "%Sun%"
LIKE "%Sun%"

Lookup

F
F
T

Dictionary

```
name LIKE “%Sun%”;
```

Dictionary

```
name LIKE "%Sun%";
```

	Dictionary		Filter	Result
0	Hey Jude	→	LIKE "%Sun%"	F
1	My Michelle		LIKE "%Sun%"	F
2	Here Comes the Sun		LIKE "%Sun%"	T

Dictionary

```
name LIKE "%Sun%";
```

	Dictionary		Filter	Result
0	Hey Jude	→	LIKE "%Sun%"	F
1	My Michelle	→	LIKE "%Sun%"	F
2	Here Comes the Sun	→	LIKE "%Sun%"	T

```
SELECT COUNT(*) GROUP BY name
```

Dictionary

```
name LIKE "%Sun%";
```

	Dictionary	Filter	Result
0	Hey Jude	LIKE "%Sun%"	F
1	My Michelle	LIKE "%Sun%"	F
2	Here Comes the Sun	LIKE "%Sun%"	T

```
SELECT COUNT(*) GROUP BY name
```

	Dictionary	Group By	Aggregation
0	Hey Jude	0	15
1	My Michelle	1	9
2	Here Comes the Sun	2	31

RLE

```
REGEXP_EXTRACT(Quarter, “(\d)+”)
```

Original	RLE	Result
Q1	(3,Q1)	(3,1)
Q1	(4,Q2)	(4,2)
Q1		
Q2		
Q2		
Q2		
Q2		

RLE

Original

Q2	WA	Bread
Q1	OR	Eggs
Q2	WA	Milk
Q1	OR	Bread
Q2	CA	Eggs
Q1	WA	Bread
Q2	CA	Milk

0 RLE runs

RLE

Original

Q2	WA	Bread
Q1	OR	Eggs
Q2	WA	Milk
Q1	OR	Bread
Q2	CA	Eggs
Q1	WA	Bread
Q2	CA	Milk

0 RLE runs

Ordered

Q1	OR	Bread
Q1	OR	Eggs
Q1	WA	Bread
Q2	CA	Eggs
Q2	CA	Milk
Q2	WA	Bread
Q2	WA	Milk

5 RLE runs

RLE

Original

Q2	WA	Bread
Q1	OR	Eggs
Q2	WA	Milk
Q1	OR	Bread
Q2	CA	Eggs
Q1	WA	Bread
Q2	CA	Milk

0 RLE runs

Ordered

Q1	OR	Bread
Q1	OR	Eggs
Q1	WA	Bread
Q2	CA	Eggs
Q2	CA	Milk
Q2	WA	Bread
Q2	WA	Milk

5 RLE runs

Optimal

Q1	OR	Eggs
Q1	OR	Bread
Q1	WA	Bread
Q2	WA	Bread
Q2	WA	Milk
Q2	CA	Milk
Q2	CA	Eggs

7 RLE runs

BigQuery
Partitions & Clustering

Table 1

Partitions	join_date	uid	name_last	name_first	title	gender
1	10/12/2018	1164581708	GREGGS	RHONDA	MD.	F
	10/12/2018	1366612186	KHOKASIAN	NAYRI	CRNA	F
	10/12/2018	1396897104	RIDGLEY	PHILLIP	CRNA	M
	10/12/2018	1447245733	ABRENICA	EVA	CRNA	F
	10/12/2018	1821060963	LEMPERT	MARK	M.D.	M
2	10/13/2018	1326011719	ANDERSON	JOHN	CRNA	M
	10/13/2018	1437188547	MANDABACH	MARK	MD	M
	10/13/2018	1699946673	HOROWITZ	DEBORAH	CRNA	F
	10/13/2018	1902853989	CAMPBELL	STEPHEN	MD	M

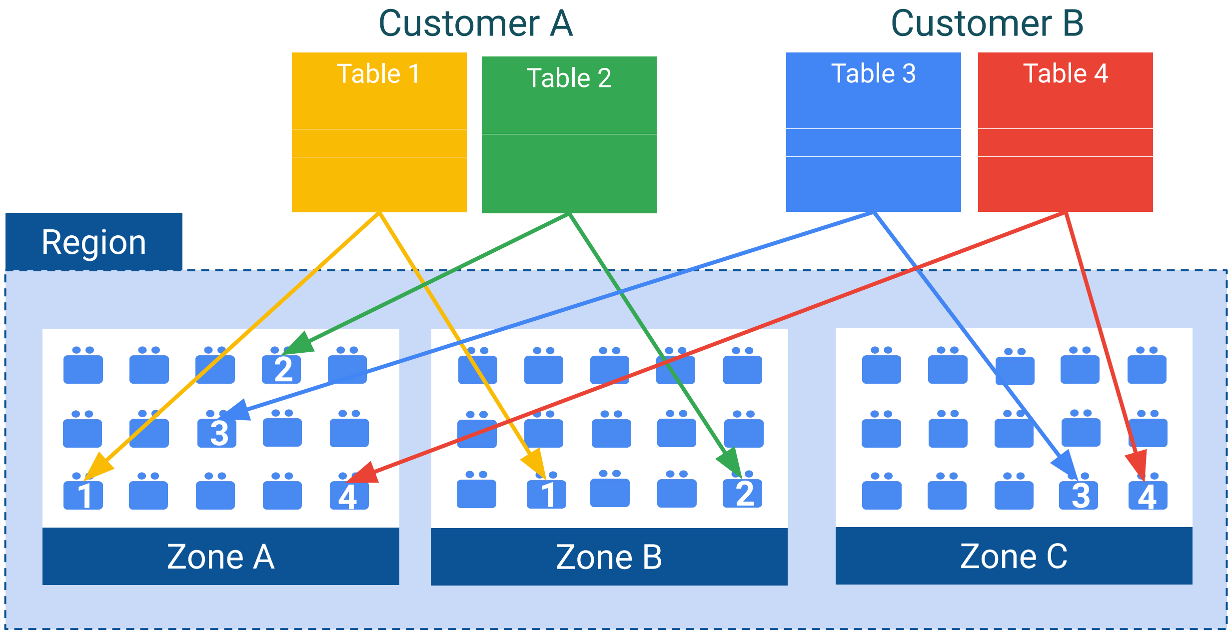
Table 2

Partitions	join_date	uid	address	city	state
1	10/12/2018	1164581708	200 HAWTHORNE LANE	CHARLOTTE	NC
	10/12/2018	1366612186	50 STANIFORD STREET	BOSTON	MA
	10/12/2018	1366612186	105 BONNIE LOCH CT	BOSTON	MA
	10/12/2018	1366612186	310 E. 14TH STREET	NEW YORK	NY
	10/12/2018	1396897104	171 ASHLEY AVE	CHARLESTON	SC
	10/12/2018	1447245733	138 HAVERHILL ST	ANDOVER	MA
	10/12/2018	1821060963	3600 JOSEPH SIEWICK DRIVE	FAIRFAX	VA
2	10/13/2018		721 MADISON ST	HUNTSVILLE	AL
	10/13/2018		619 19TH STREET SOUTH	BIRMINGHAM	AL
	10/13/2018		105 BONNIE LOCH CT	ORLANDO	FL
	10/13/2018		125 DOUGHTY ST	CHARLESTON	SC

BigQuery | Managed storage

Durable and persistent storage with automatic backup

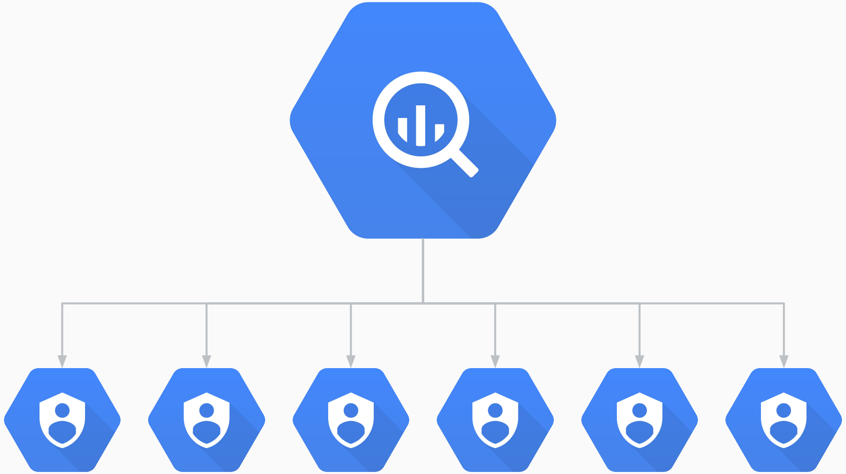
- Tables are stored in optimized columnar format
- Each table is encrypted on disk
- Storage is durable & each table is replicated across datacenters



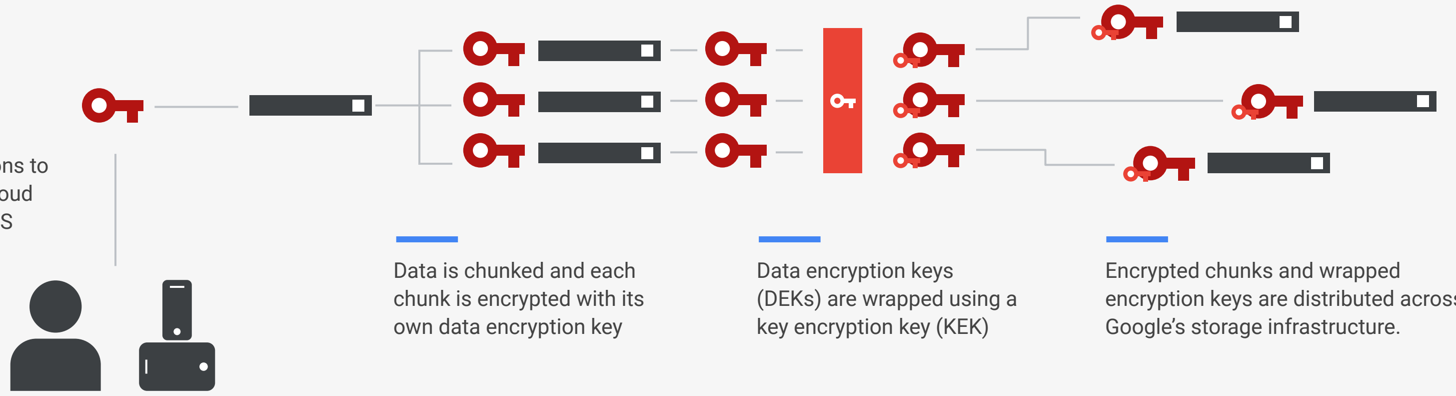
Google Cloud

BigQuery handles reliability automatically so you don't have to

1. No virtual machines to manage and maintain BigQuery's availability
2. Automatic replication (minimum of 2 times) in multiple regions/zones at any time
3. Auto failover incases of zonal outages
4. Maintains **99.99**% uptime SLAs to meet your business objectives
5. Table changes in the last 7 days are maintained, allowing for time travel
6. Data is encrypted in transit and at rest by default



Encryption by default in transit and at rest



Connections to Google Cloud require TLS

Data is chunked and each chunk is encrypted with its own data encryption key

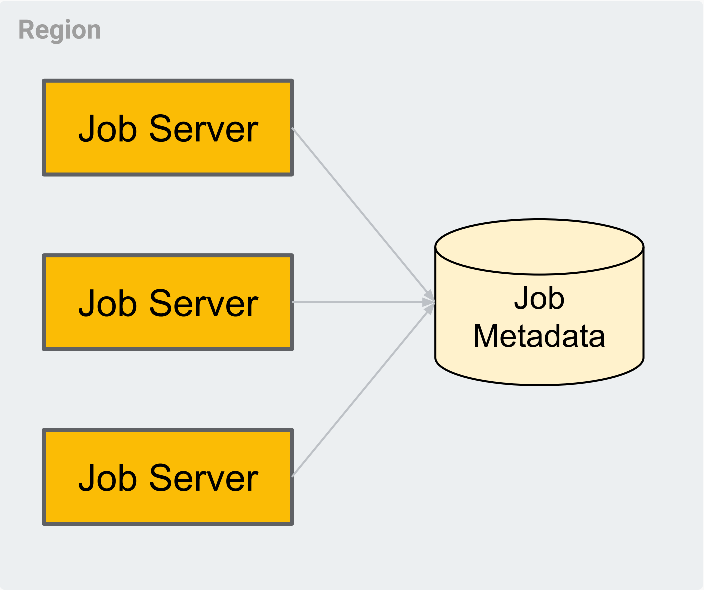
Data encryption keys (DEKs) are wrapped using a key encryption key (KEK)

Encrypted chunks and wrapped encryption keys are distributed across Google's storage infrastructure.

BigQuery Scheduling

Query Overview | Job Queueing

- Jobs start in the **PENDING** state.
    - Can transition to either **RUNNING** or **DONE** (due to timeout).
    - Most jobs immediately enter the **RUNNING** state.
- Jobs defer their **RUNNING** transition when:
    - **BATCH** priority: always defer at least 1 minute, longer if awaiting quota or the individual server is nearing capacity.
    - **INTERACTIVE** priority: never.
- The Job server will periodically re-evaluate the deferment, with exponential backoff.



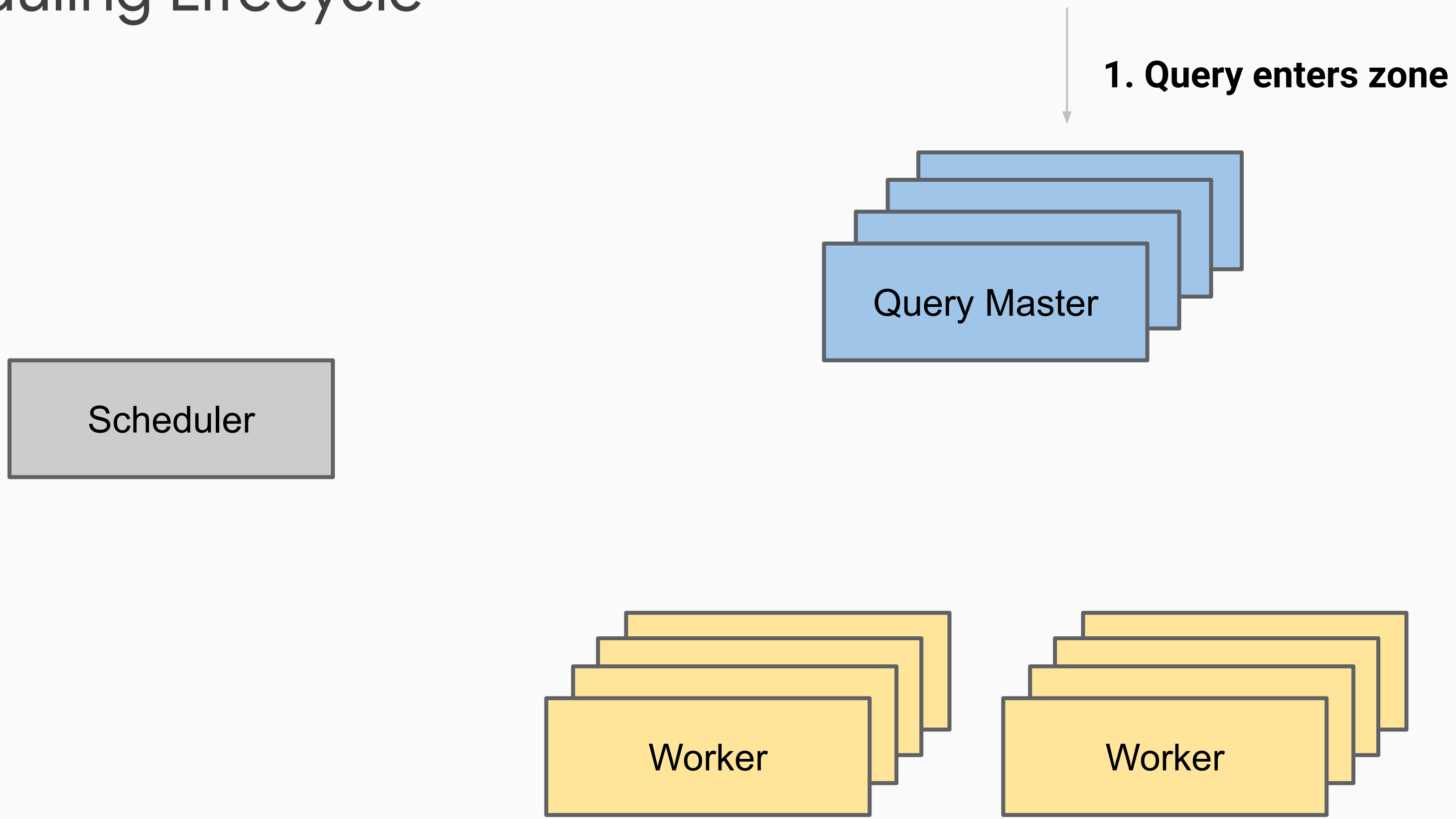
What is a slot?

A unit of **compute** within BigQuery:

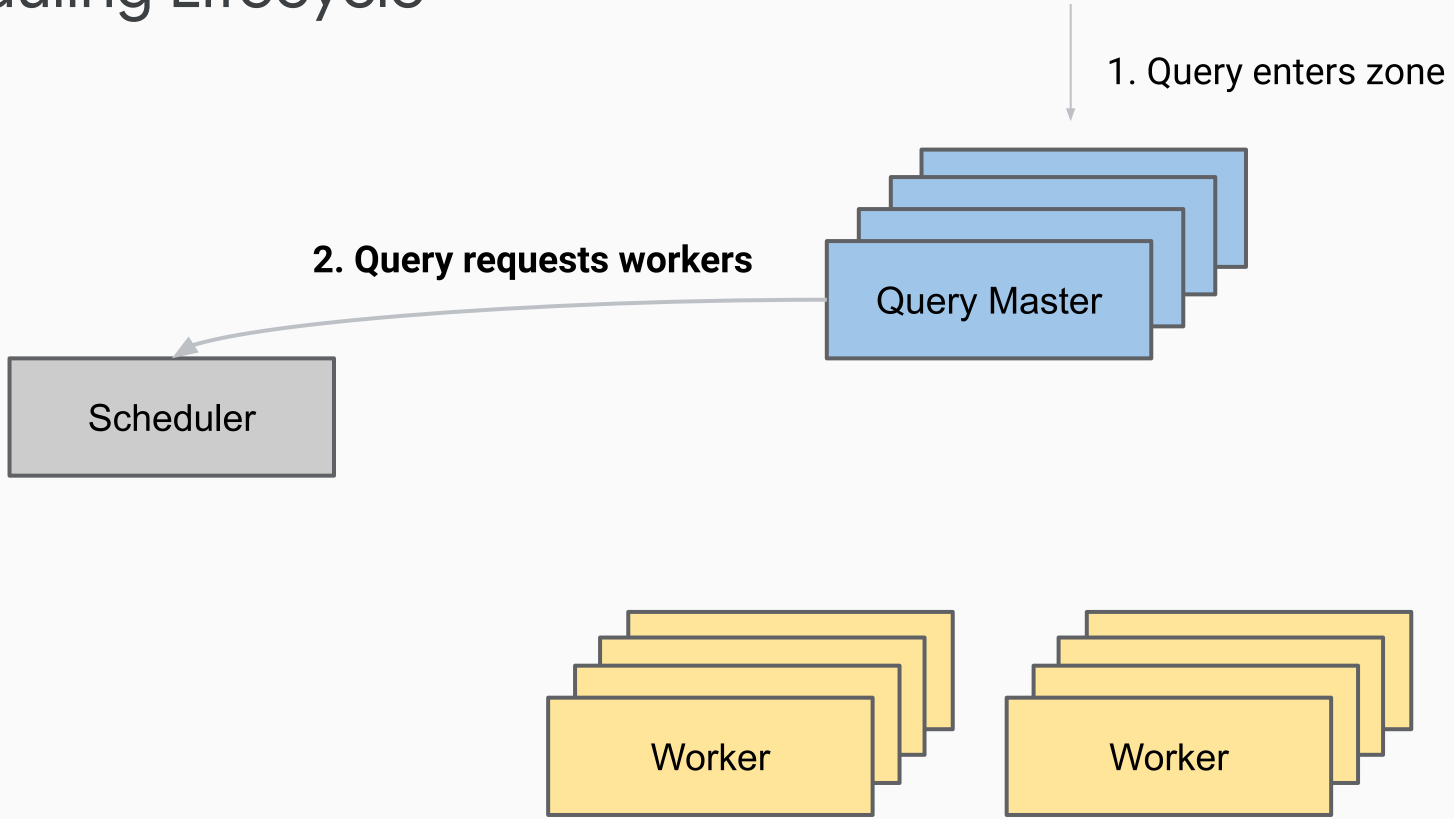
- Encapsulates CPU, memory, disk
- In reality, a slice of a core (~0.5 CPU and ~0.5 GB of RAM)
- Dynamically sized based on query demands

Query → *consists of* → **Stages** → *consists of* → **Tasks** → *executes on* → **Slots**

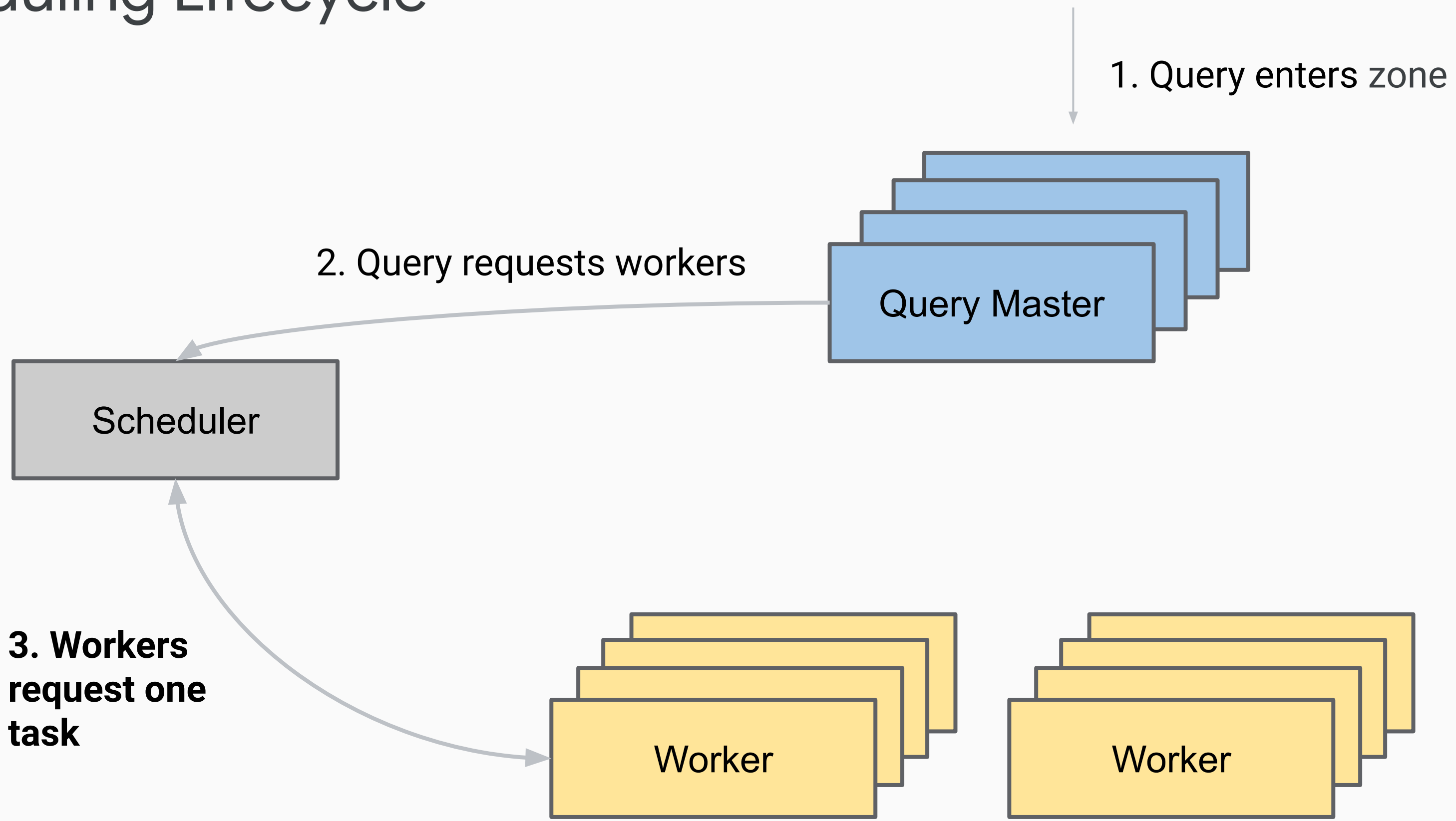
Scheduling Lifecycle



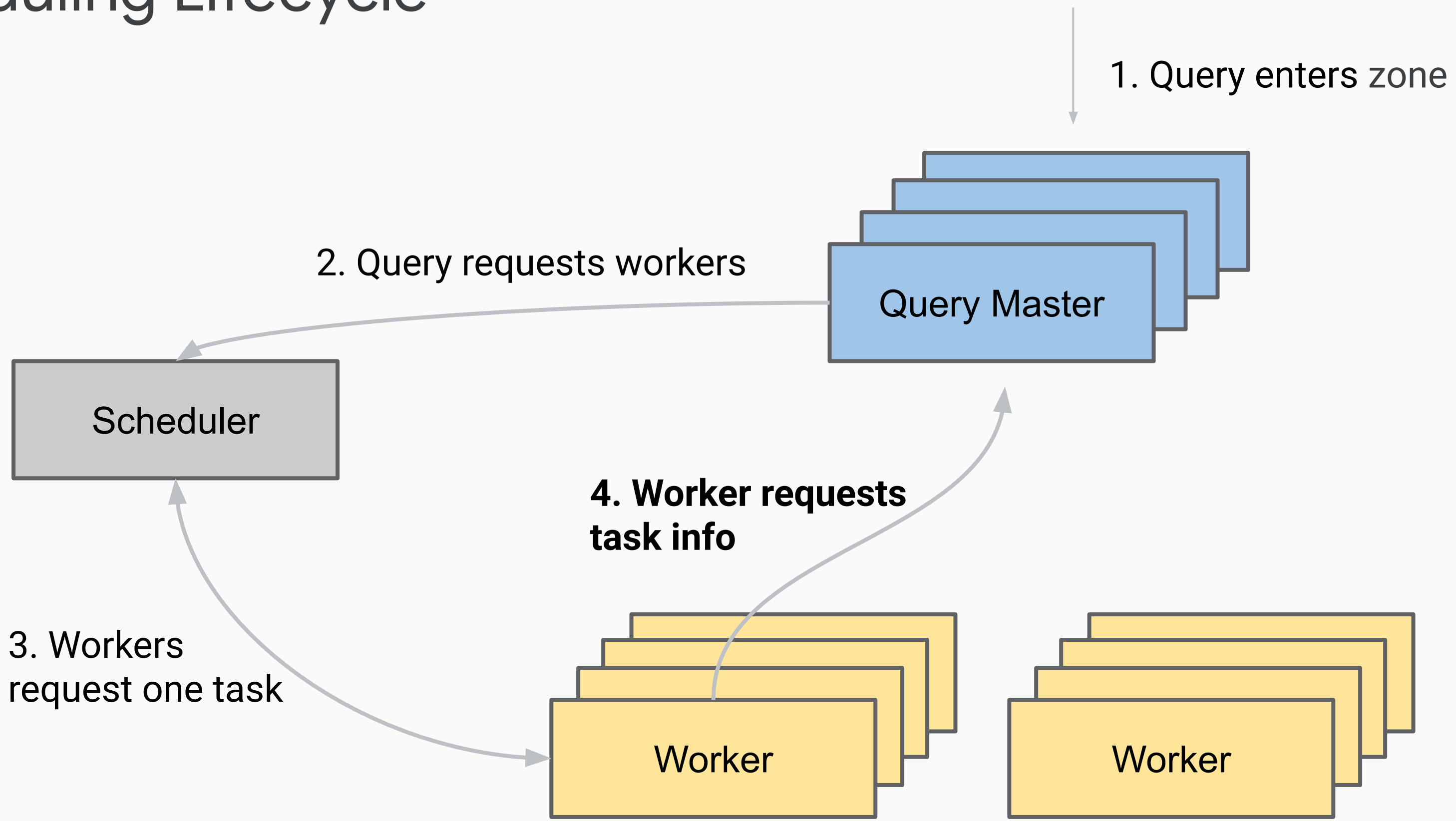
Scheduling Lifecycle



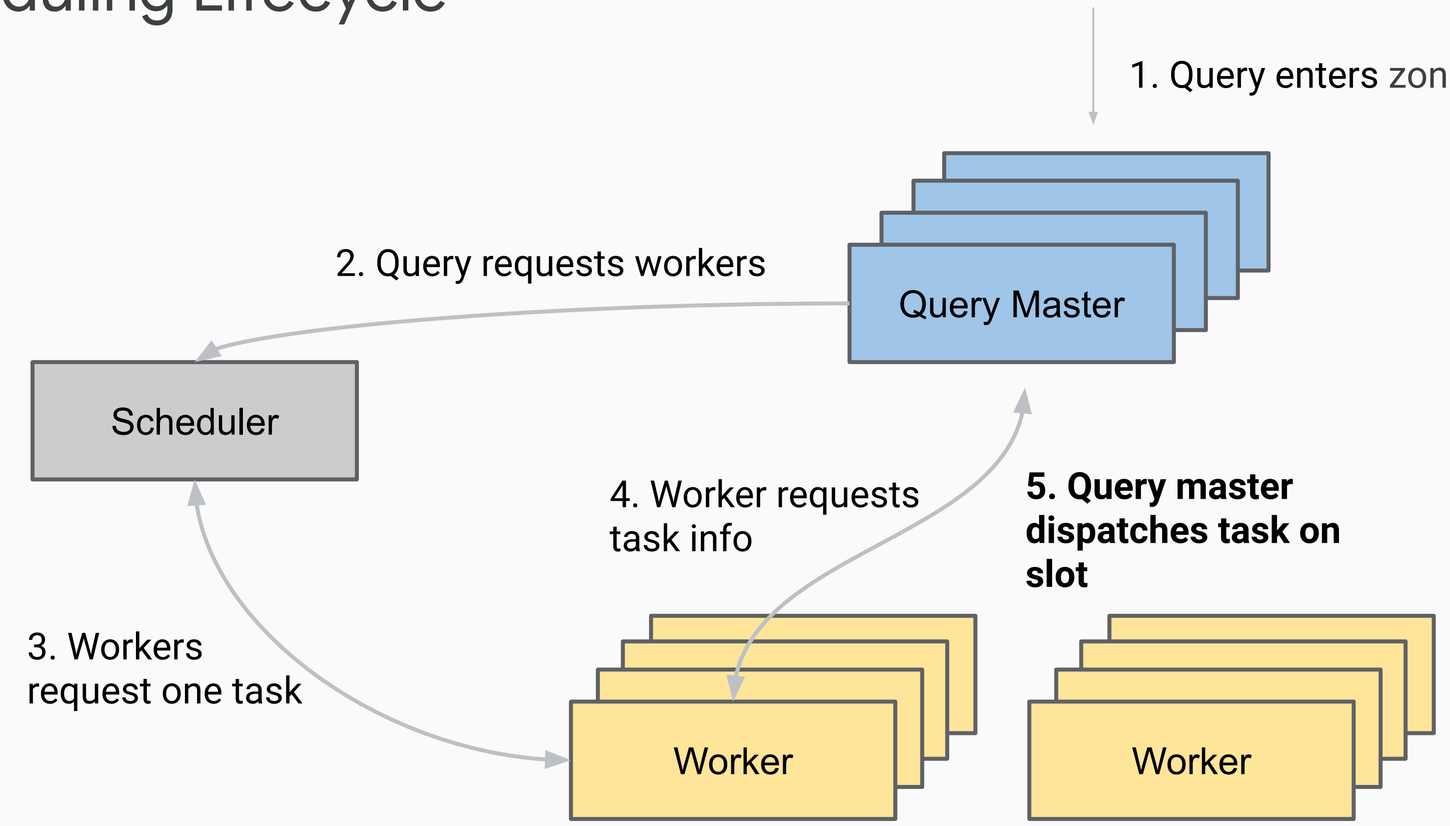
Scheduling Lifecycle



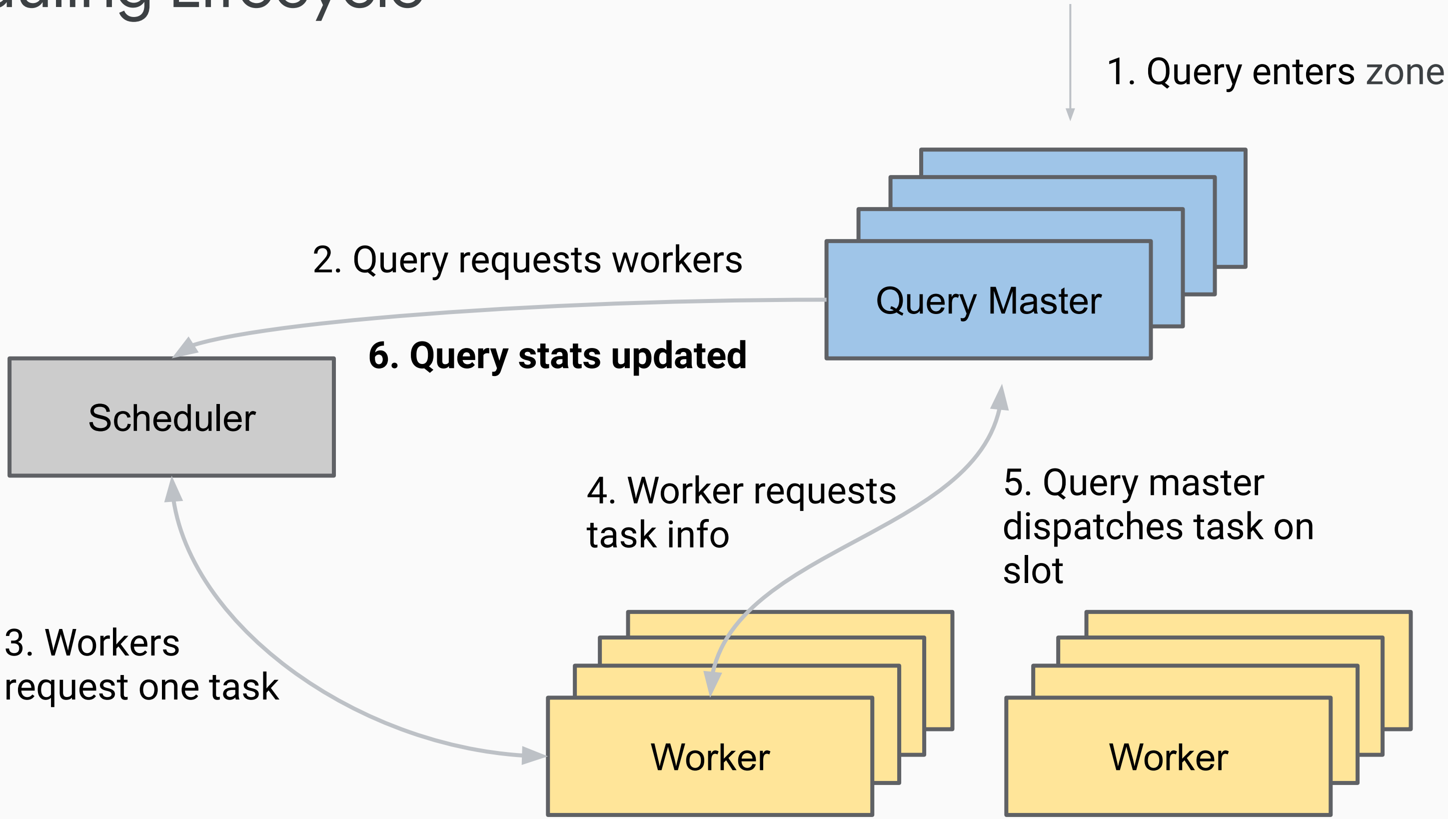
Scheduling Lifecycle



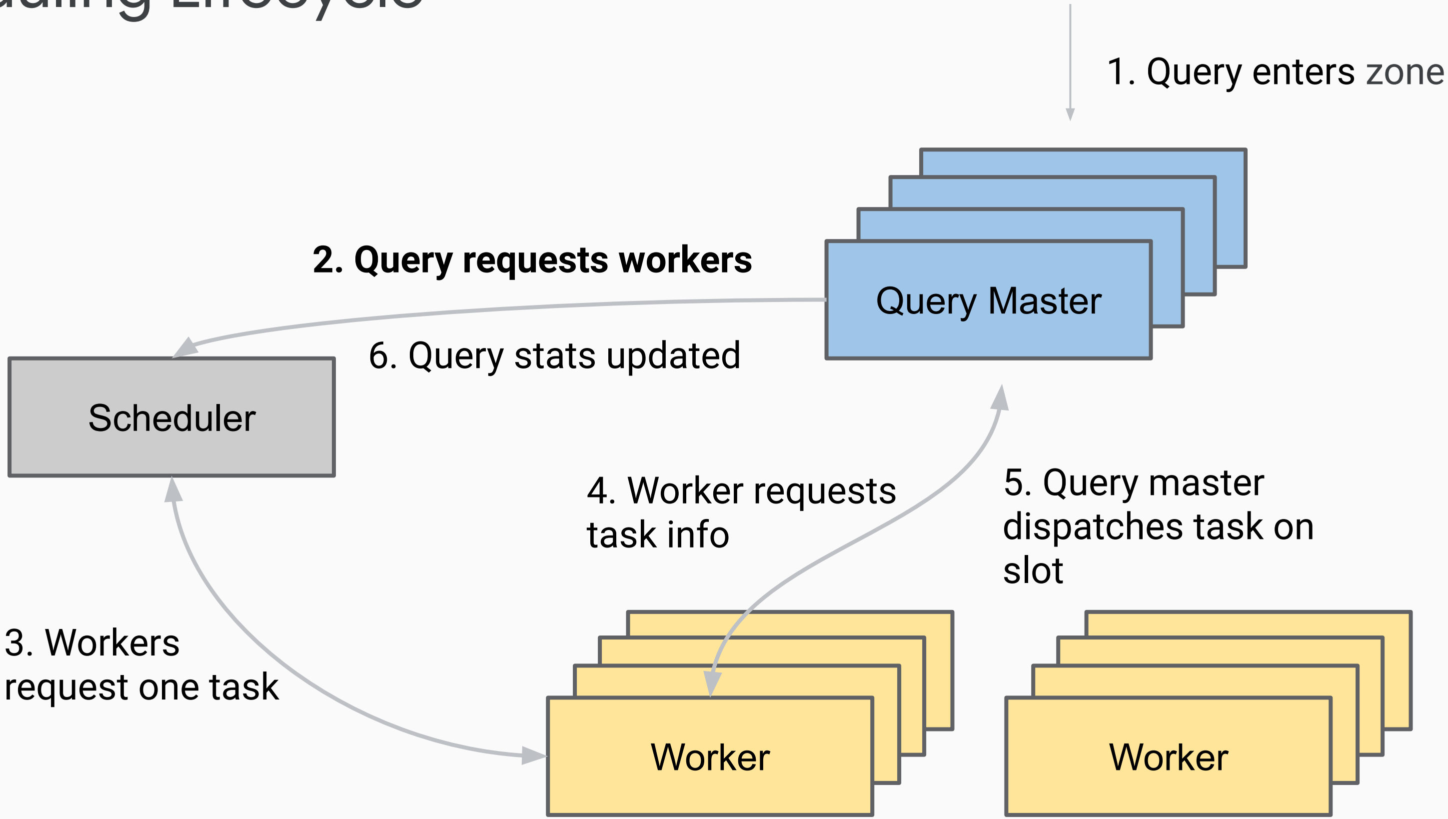
Scheduling Lifecycle



Scheduling Lifecycle



Scheduling Lifecycle



BigQuery Schema

Some things to keep in mind

- BigQuery does not use or support indexes, only partitioning and clustering
- For each query, BigQuery executes a full-partition column scan
- Filtering on a clustered column may greatly reduce the amount of data scanned
- BigQuery performance and query costs are based on the amount of data scanned
- Storage is cheap!
- BigQuery supports nested and repeated columns

Schema design in a nutshell

- Denormalization is **NOT** a requirement but can speed up slow analytical queries by reducing the amount of data to shuffle
- When join time become excessively long, you want to use nested repeated fields
- Optimize to solve actual problems, not expected ones (performance gets better over time)

Table performance / cost features

Partitioning

Filtering storage before query execution begins to reduce costs. Reduces a full table scan to the partitions specified. Single column, lower cardinality (e.g. thousands of partitions).

- **Time Partitioning (Pseudocolumn)**
- **Time Partitioning (User Date/Time Column)**
- **Integer Range Partitioning**

Clustering

Storage optimization within columnar segments to improve filtering and record colocation. Clustering performance and cost savings can't be assessed before query begins. Prioritized clustering of up to 4 columns, on more diverse types (but no nested columns).

BigQuery provides Automatic re-clustering

free

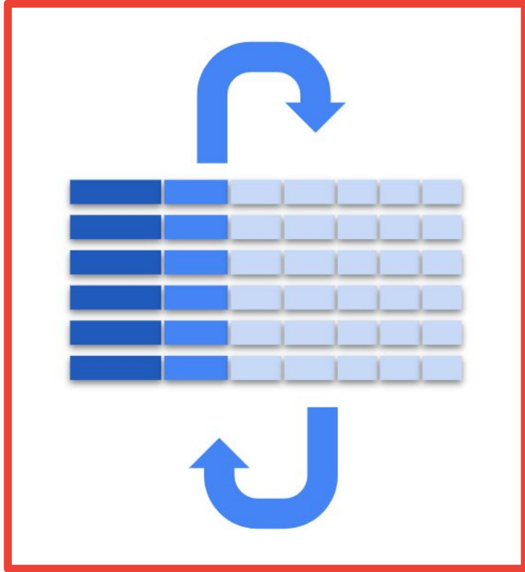
Doesn’t consume your query resources

maintenance-free

Requires no setup or maintenance

autonomous

Automatically happens in the background



Loading Data into BigQuery

Loading data

Batch ingest is free

Doesn't consume query capacity

ACID semantics

Load petabytes per day

Streaming API for real-time

Data ingestion options

Batch ingestion

Data from GCS or via HTTP POST

Multiple File Formats Supported

Snapshot-based arrival - All Data arrives at once, or not at all

Streaming ingestion

Continuous ingestion from many sources (web/mobile apps, point of sale, supply chain)

Immediate query availability from buffer

Deferred creation of managed storage

Query materialization

SELECT results yield data in the form of tables, either anonymous (cached) or named destinations

ETL/ELT Ingest + Transform via Federated Query

Data Transfer Service (DTS)

Managed ingestion of other sources (doubleclick, adwords, youtube)

Newer: Scheduled Queries, Scheduled GCS Ingestion

Options for third-party integration

Ingestion formats

Faster

Avro (Compressed)

Avro (Uncompressed)

Parquet / ORC

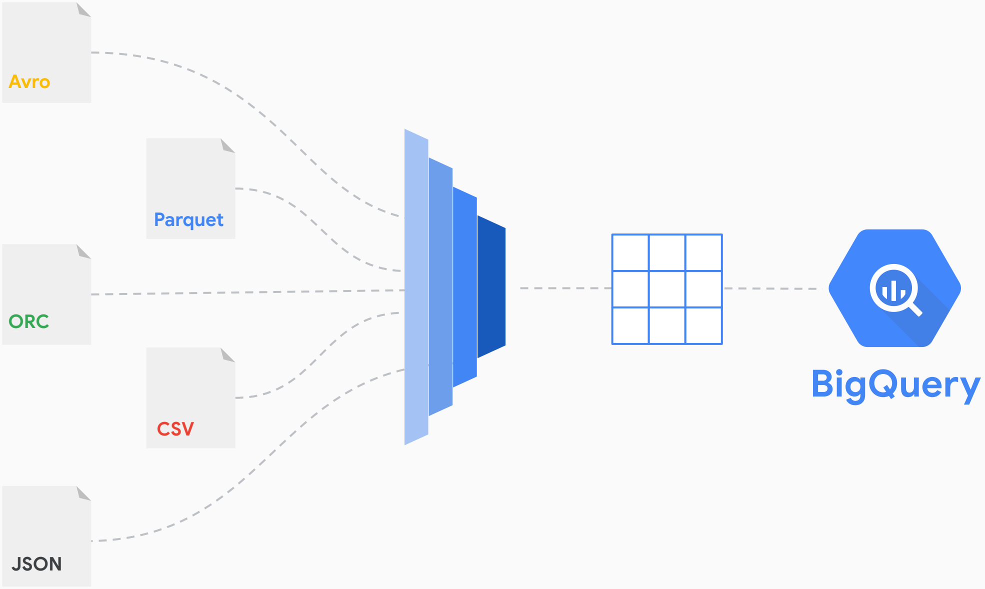
CSV

JSON

CSV (Compressed)

JSON (Compressed)

Slower



BigQuery streaming ingestion

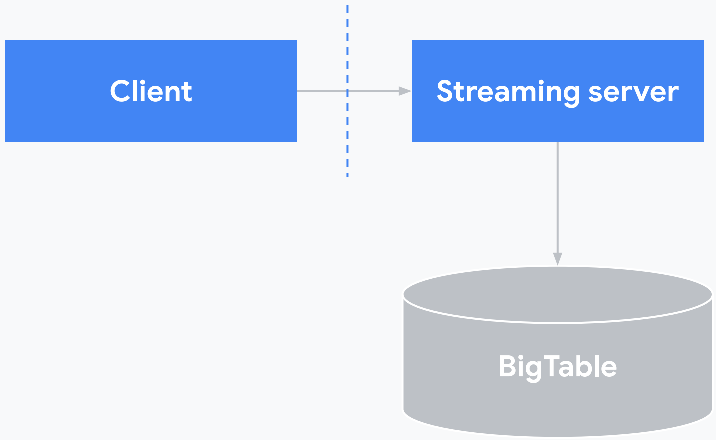
HTTPS Post individual rows or groups of rows

Up to 1M rows / second / table

Streaming inserts buffered in Bigtable

Hot tables spray across multiple zones

BigQuery streaming architecture

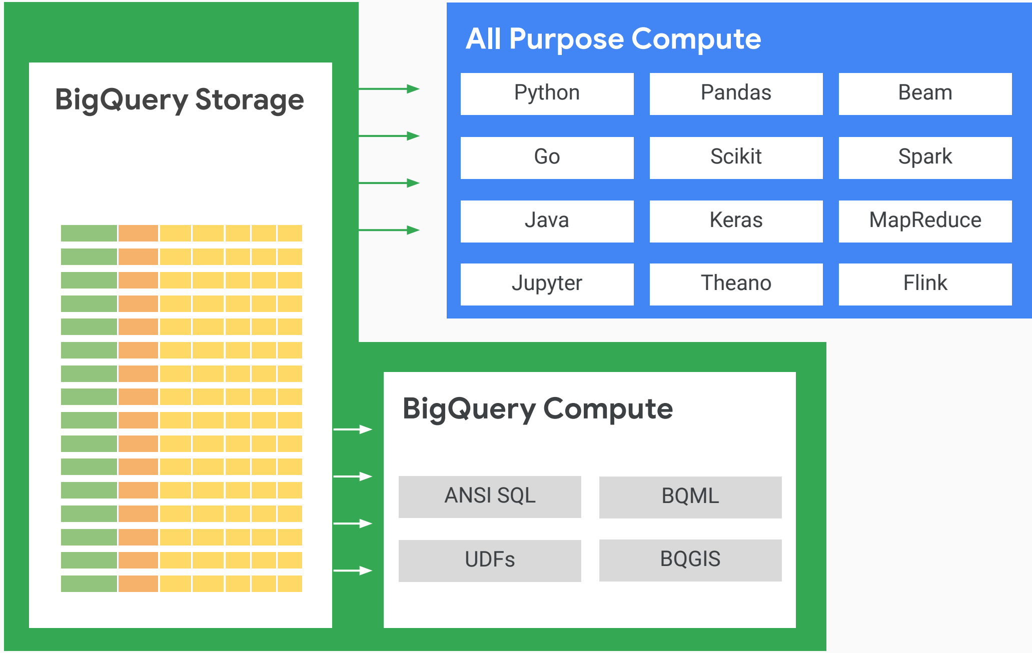


BigQuery storage API... treat Data Warehouse storage like storage!

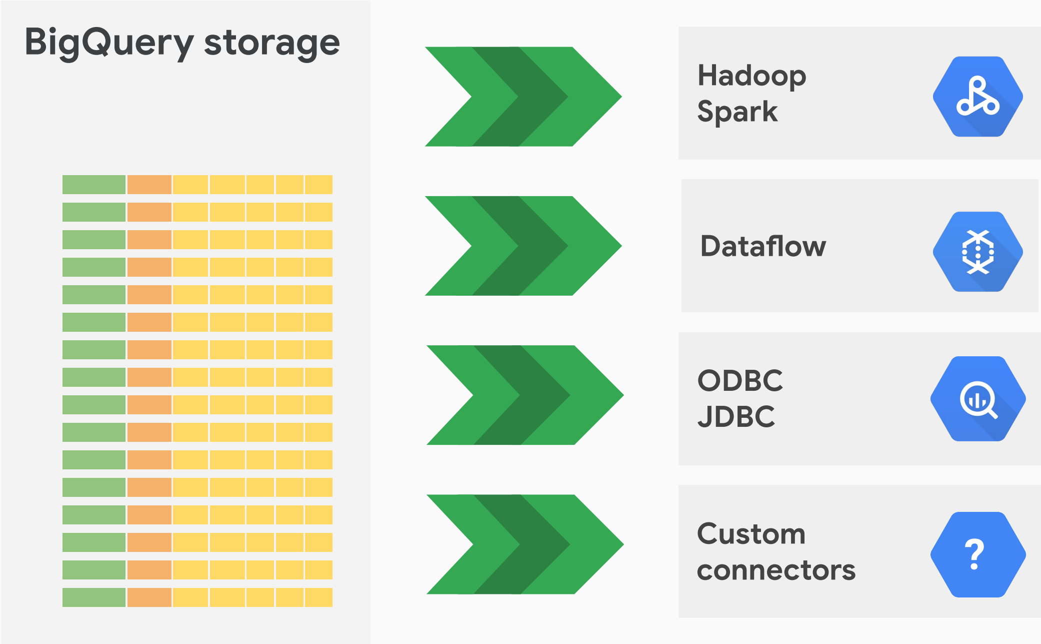
Real-time
1 sec to first byte

Fast
50Gb/sec

Flexible
Filters rows & columns



BigQuery storage API - Use cases



High-perf dataframes

Unified batch & stream ETL

Third party **tools**

Break down the storage wall

BigQuery Performance Optimization

How do you optimize queries

- ➔ Less work → Faster Query
- ➔ What is *work* for a query?
    - I/O — How many bytes did you read?
    - Shuffle — How many bytes did you pass to the next stage?
        - Grouping — How many bytes do you pass to each group?
    - Materialization — How many bytes did you write?
    - CPU work — User-defined functions (UDFs), functions

Don’t project unnecessary columns

- On how many columns are you operating?
- Excess columns incur wasted I/O and materialization

Don’t SELECT * unless you need every field

Filter early and often using WHERE clauses

- On how many rows (or partitions) are you operating?
- Excess rows incur “waste” similar to excess columns

Do the biggest joins first

- Joins — In what order are you merging data?
- Guideline — Biggest, Smallest, Decreasing Size Thereafter
- Avoid self-join if you can, since it squares the number of rows processed

Wildcard tables — Standard SQL (1 of 2)

- Use wildcards to query multiple tables using concise SQL statements
- Wildcard tables are a union of tables matching the wildcard expression
- Useful if your dataset contains:
  - Multiple, similarly named tables with compatible schemas
  - Sharded tables
- When you query, each row contains a special column with the wildcard match

Wildcard tables — Standard SQL (2 of 2)

- Example:

  FROM `bigquery-public-data.noaa_gsod.gsod*`

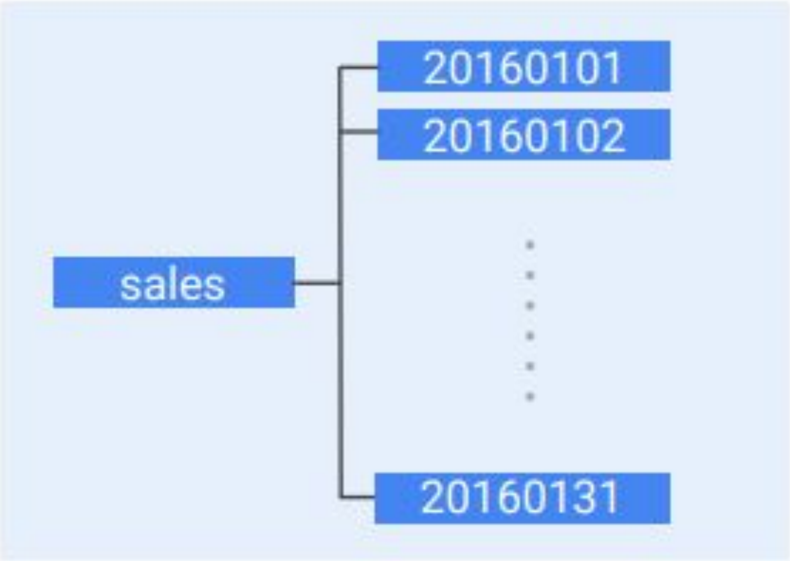
- Matches all tables in noaa_gsod that begin with string ‘gsod’
- The backtick (``) is required
- Richer prefixes perform better than shorter prefixes
  - For example: .gsod200* versus .*

Table Partitioning

- Time-partitioned tables are a cost-effective way to manage data
- Easier to write queries spanning time periods
- When you create tables with time-based partitions, BigQuery automatically loads data in correct partition
    - Declare the table as partitioned at creation time using this flag: --time_partitioning_type
    - To create partitioned table with expiration time for data, using this flag: --time_partitioning_expiration

Example - Time Partitioning

```
SELECT ...
FROM `sales`
WHERE _PARTITIONTIME
BETWEEN TIMESTAMP("20160101")
    AND TIMESTAMP("20160131")
```

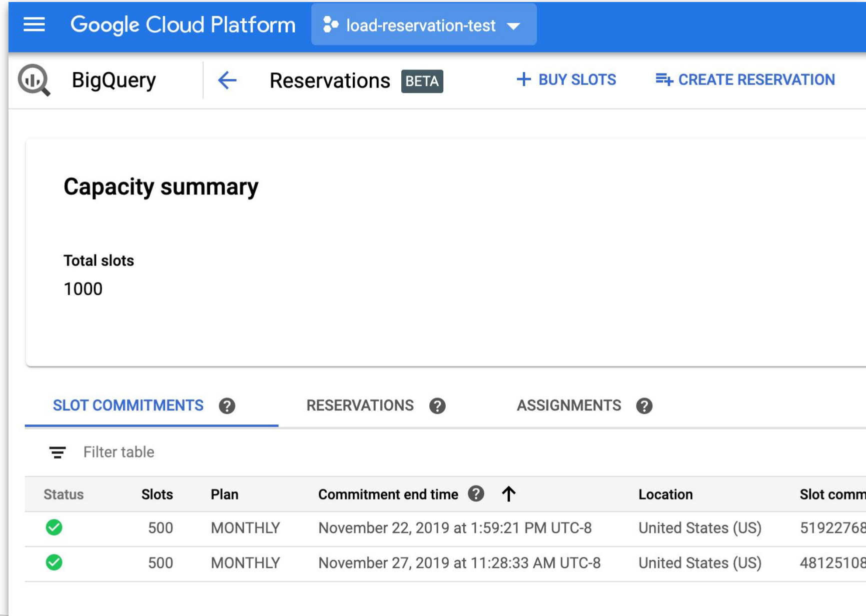


BigQuery Slots and Reservations

Enterprise-grade Workload management With Reservations

BigQuery Reservations allows customers to:

- Control flat-rate spend
- Buy slots in Web UI in seconds
- Efficiently manage workloads in BigQuery
- Automatically share any unused capacity



BigQuery Commitment Types and Use Cases

Seconds Minutes — **Hours Days** — **Months** — **Years**

Seconds / Minutes	Hours / Days	Months	Years
Flex Slots	**Flex Slots**	**Monthly Commitments**	**Annual Commitments**
Rapidly respond to business demands and Evaluate performance.	Plan for business-critical calendar events.	Ideal balance between flexibility and cost.	The most cost-effective option for steady-state workloads.

BigQuery workload management

Customers can programmatically perform workload management using Reservations:

Create and delete reservations

Move projects between reservations

Move slots between reservations

Idle slots are seamlessly and automatically shared in real-time

Example

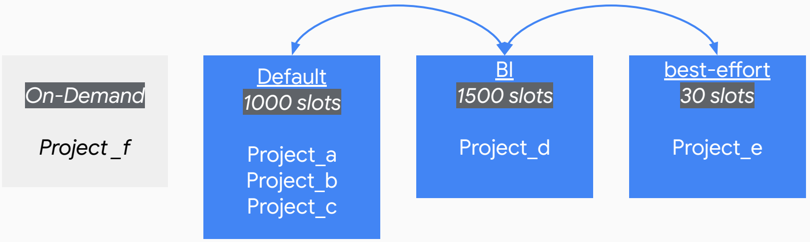
At 3am an important workload in project_d needs to run

At 3am we create a reservation
Move 1000 slots to the reservation
Move project_d into reservation

At 6am we delete the reservation
Move 1000 slots back
Move project_d back

Project_d was guaranteed 1000 slots 3am-6am

Idle slots seamlessly shared



BigQuery Specialities

Analyze GIS data in BigQuery

BigQuery GIS

Accurate spatial analyses with Geography data type over GeoJSON and WKT formats

Support for core GIS functions – measurements, transforms, constructors, etc. – using familiar SQL

Behind the scenes - BigQuery ML

Through SQL and within BigQuery

Leverage BigQuery's processing power to build a model

Auto-tuned learning rate

Auto-split of data into training and test

Null imputation

Standardization of numeric features

One-hot encoding of strings

Class imbalance handling

GIS

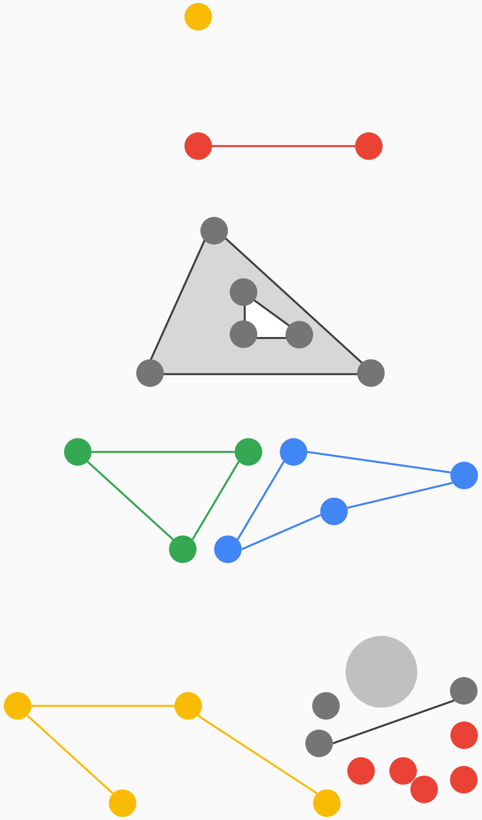
Point

Linestring

Polygon

Multi-Polygon

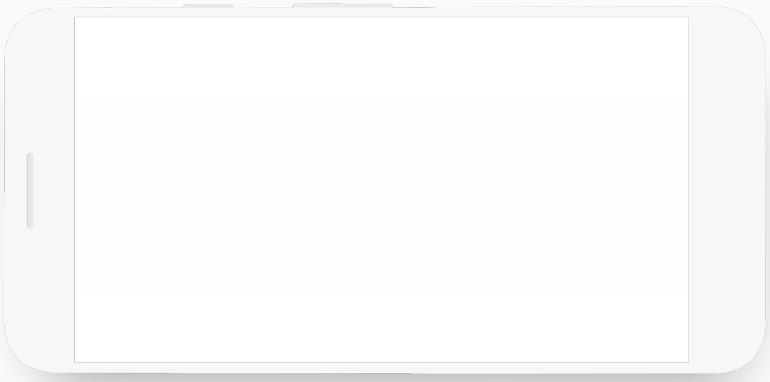
Collections



Formats

- WKT
- GeoJSON
- WKB

BigQuery ML for predictive analytics



1. **Execute** ML initiatives without moving data from BigQuery
2. **Iterate** on models in SQL in BigQuery to increase development speed
3. **Automate** common ML tasks, and hyperparameter tuning

Supported models

Classification

Logistic regression

DNN classifier (Beta)

XGBoost classifier (Beta)

Other models

k-means clustering

Recommendation: Matrix factorization (Beta)

Regression

Linear regression

DNN regressor (Beta)

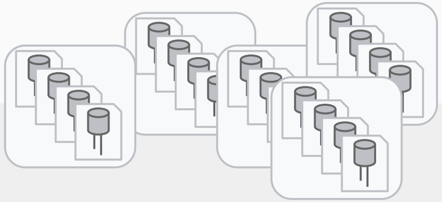
XGBoost classifier (Beta)

Model import

Import TensorFlow and XGBoost models for prediction (Beta)

Time travel

Read data from any time within the last 7 days.



```
SELECT x, y
FROM dataset.table
FOR SYSTEM_TIME AS OF
 TIMESTAMP_SUB(CURRENT_TIMESTAMP(), INTERVAL 3 DAY)
```

Higher Education Programs Summary

Higher Education Programs (https://cloud.google.com/edu)

Students | **Teaching Faculty** | **Researchers** | **IT Admin / Instructional Learning Staff**

- Training Credits (Qwiklabs and Coursera)
- GCP Credits for Teaching & Learning
- GCP Research Credits
- Career Readiness Tracks (Coursera)
- Faculty Experts Program
- Google Cloud Certifications
- Higher Ed Trainers
- Student Programming
- Instructor-Led Training
- Customer Advisory Board

Higher Education Cloud Credit Programs

Training Credits

Get **hands-on experience** using Google Cloud Platform with training for everyone from beginners through advanced users.

Students & IT enablers can apply for 200 Qwiklabs credits.

Credits for Learning

Currently students ask faculty members to apply for credits for courses and receive **$50 in GCP credits**.

Get started with our trainings to gain an understanding of GCP.

Career Readiness

Jump-start your career in cloud technology.

Tap in to in-person training, self-paced modules, and professional courses to **build skills and prepare for certification(s)**.

Thank You!