Introduction to Google BigQuery

Nir Carasso

Customer Engineer , Google Cloud, EMEA

Agenda

- Data Analytics Markets & Macro trends
- Intro to BigQuery
- BigQuery Engine | Schema | Storage
- BigQuery Performance Optimization

Data Analytics

Goodbye 2022

Market Pressures...

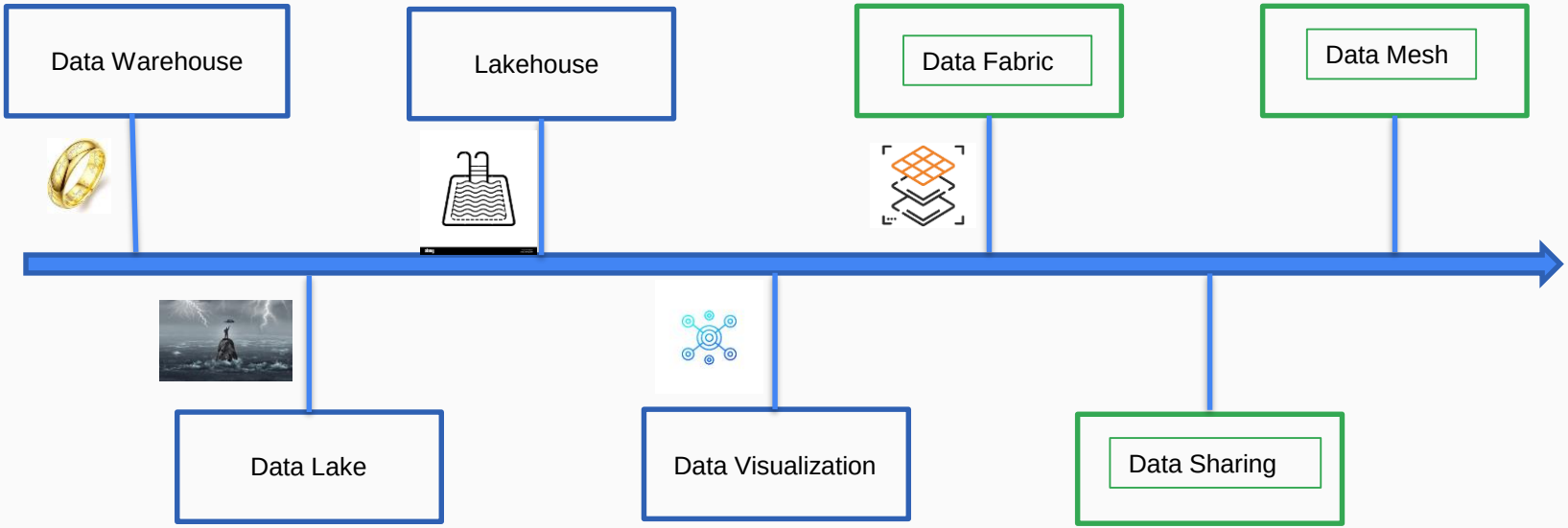
- **Slowing** economy -> reduce IT spend, except cloud computing
- Higher tech stack **complexity** while talent is hard to find
- Global **regulations** are increasing data-> security | privacy | governance

Macro Trends

- User Experience
- Managed Services
- Open Standards
- Decentralization (Data Products)
- Semantic /Metric Layer
- Data Ops/Observability | Fin Ops

Google BigQuery

Data delivery models



Market & Trends

- Budget
- Simple & Managed
- Metadata management (regulation)

Data models evolution

- Semantic /Metric Layer
- Decentralization
- Open Standards

Google BigQuery

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

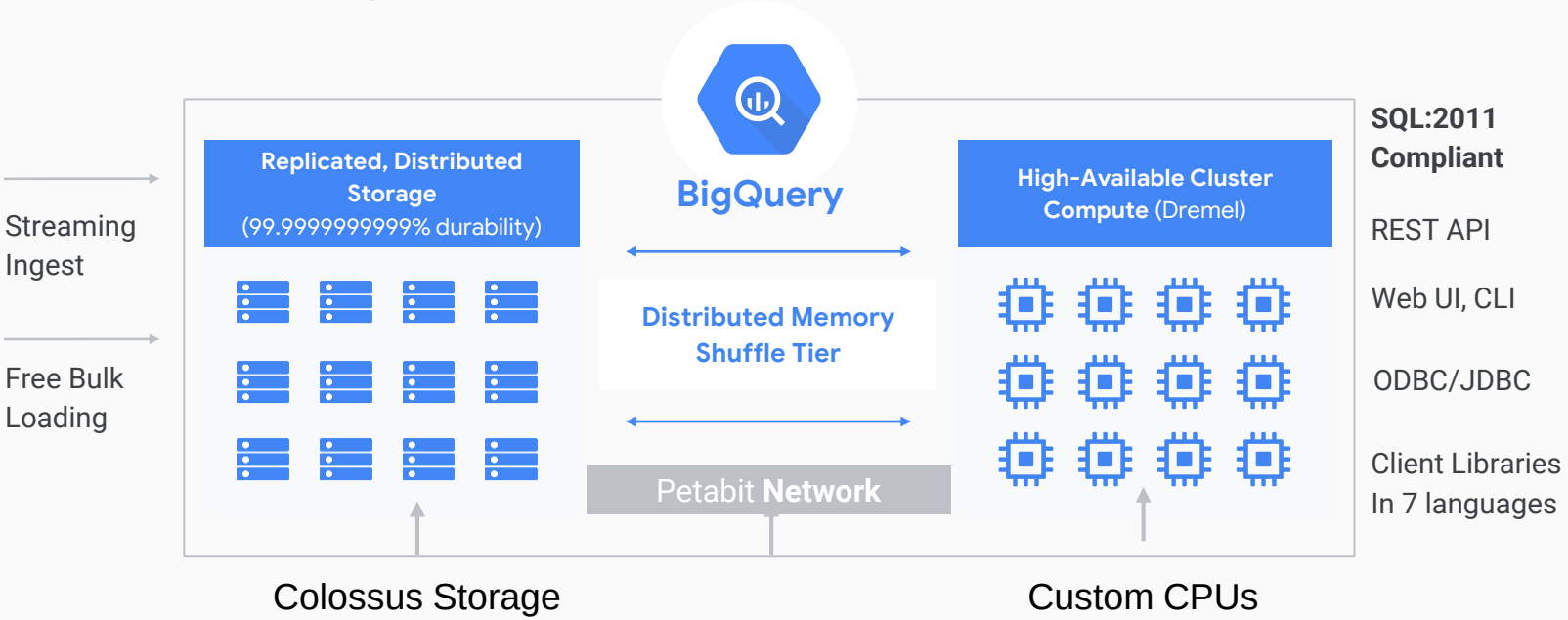
- Google Cloud's **Enterprise** Data Warehouse for Analytics
- **Petabyte-Scale** and Fast Convenience of Standard SQL
- **Encrypted**, Durable and Highly Available
- **Fully Managed and Serverless**

OLAP vs OLTP: Which fits my use case?

	OLTP	OLAP
	OnLine Transaction Processing	OnLine **Analytical** Processing
Data nature	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

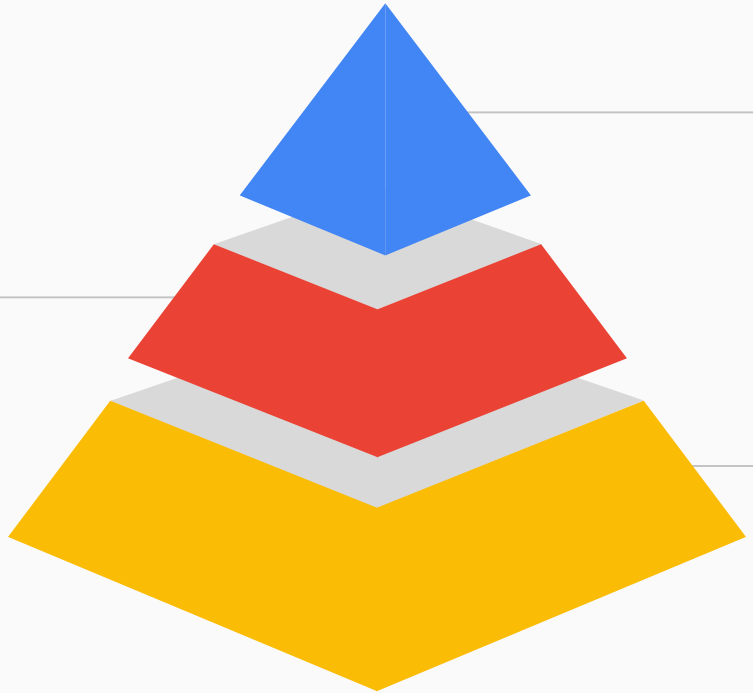
BigQuery | Architecture

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

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

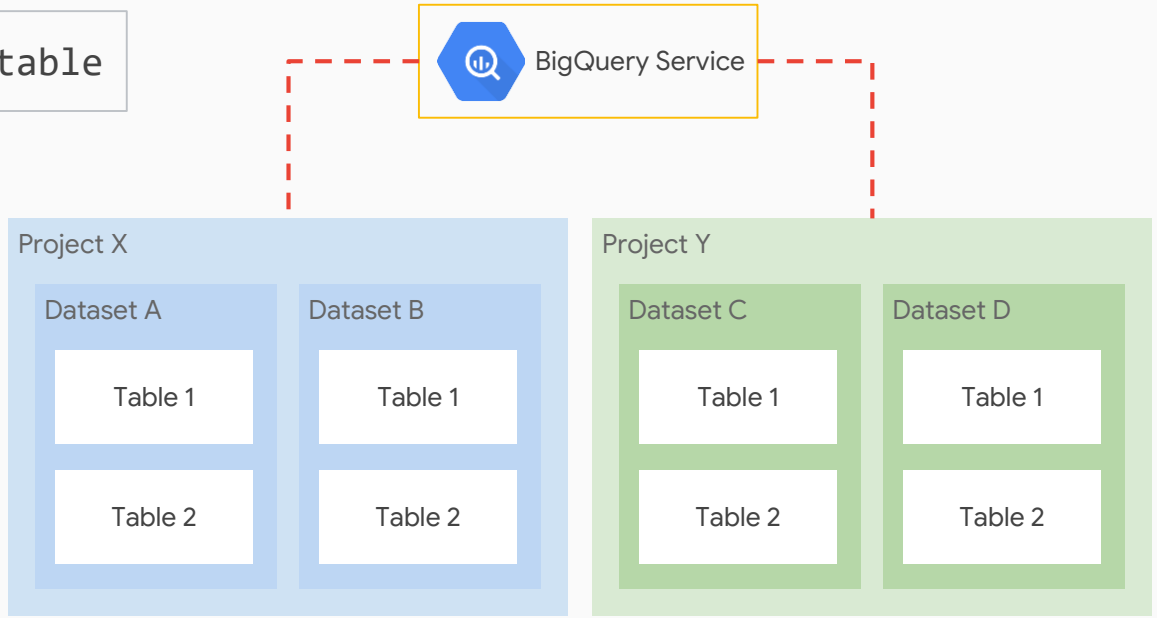
Tables

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
```



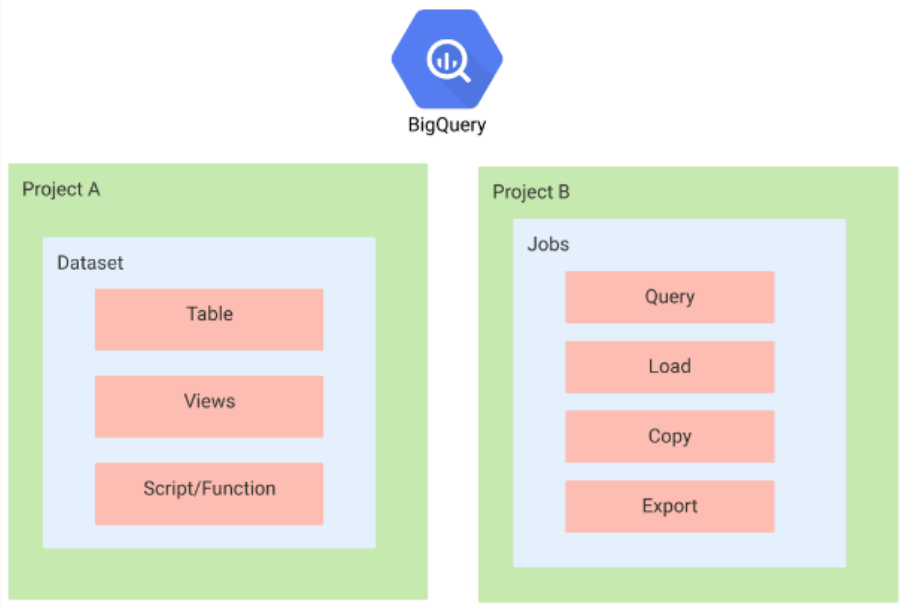
What are some reasons to structure your information into:

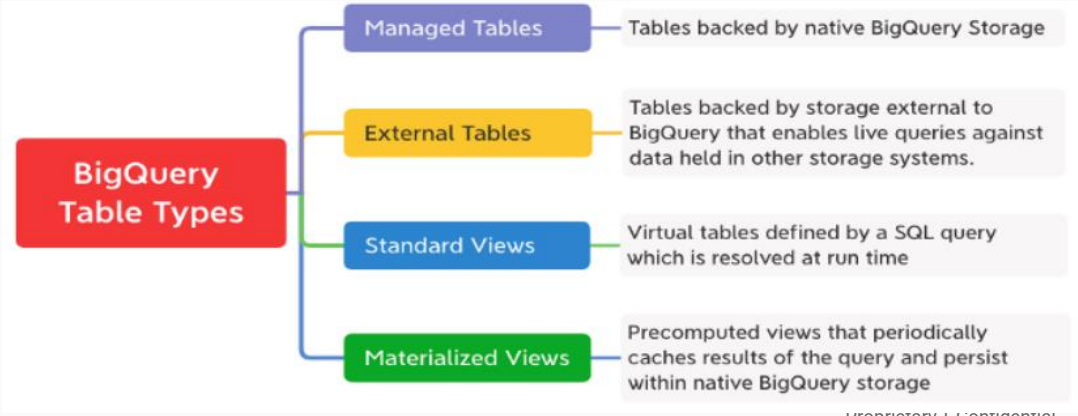
- Datasets?
- Projects?
- Tables?

BigQuery Resource Model

Tables
•Collections of columns and rows stored in managed storage. These could be natively managed or federated.
•Defined by a schema with strongly-typed columns of values
•Allow access control at Table level , Column level and Row Level.

Views
•Virtual tables defined by a SQL query
•Allow access control at View level



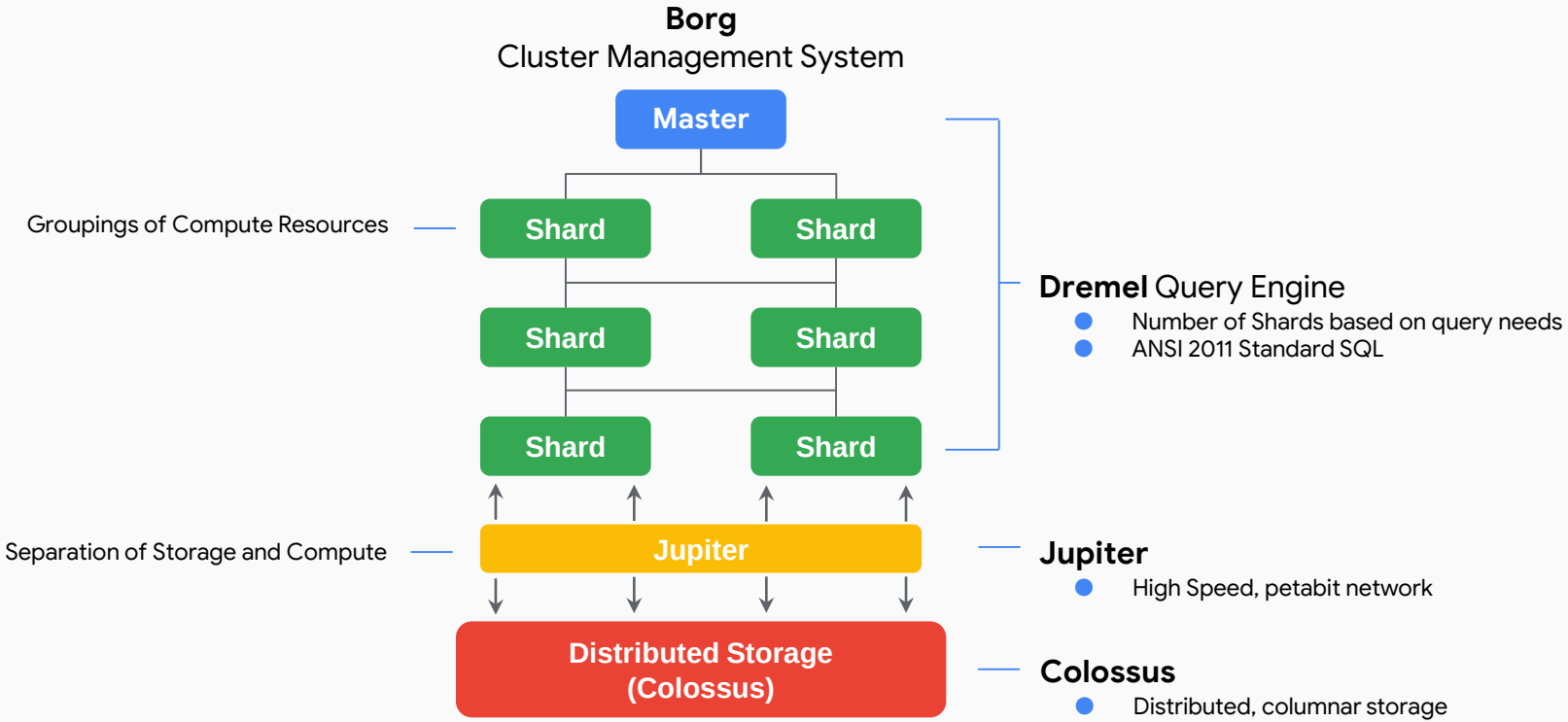


BigQuery Interface Walkthrough (video)

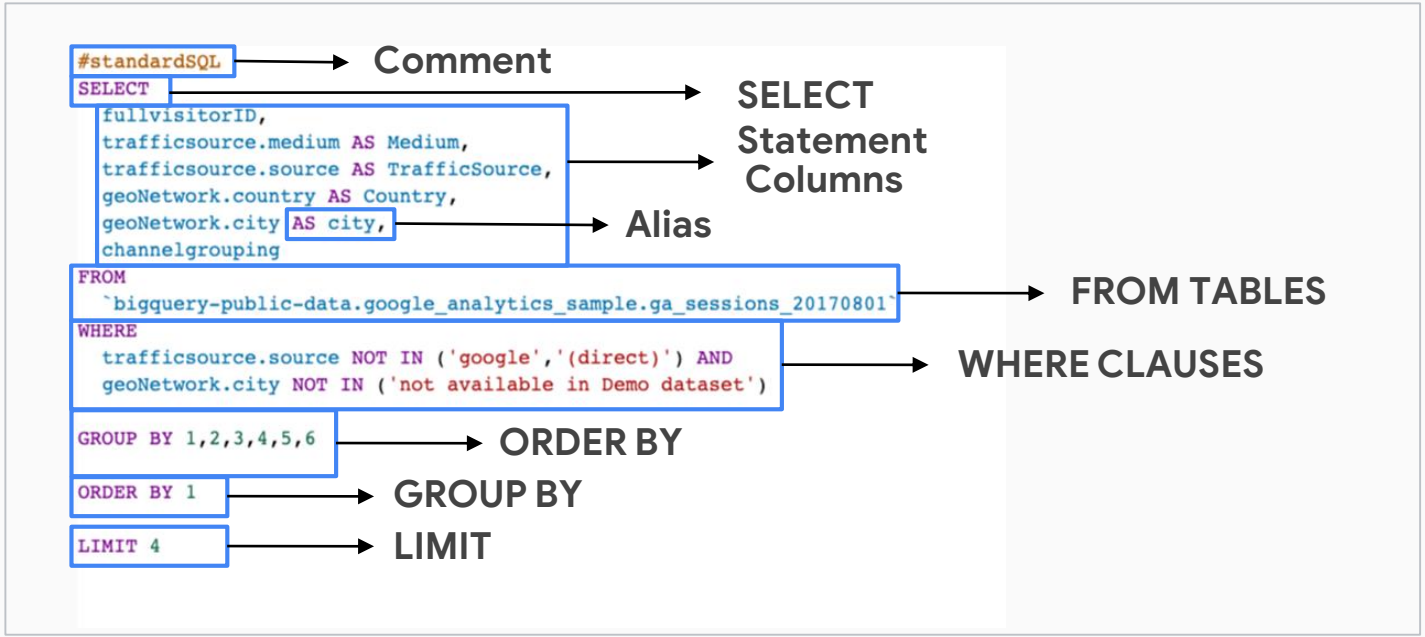
- Project | dataset | tables
- Search bar
- Query history (personal, project)
- Save queries (personal, project)
- Job history (personal, project, cloud dataflow)
- Transfer and Schedule queries
- BI Engine

BigQuery Query Engine

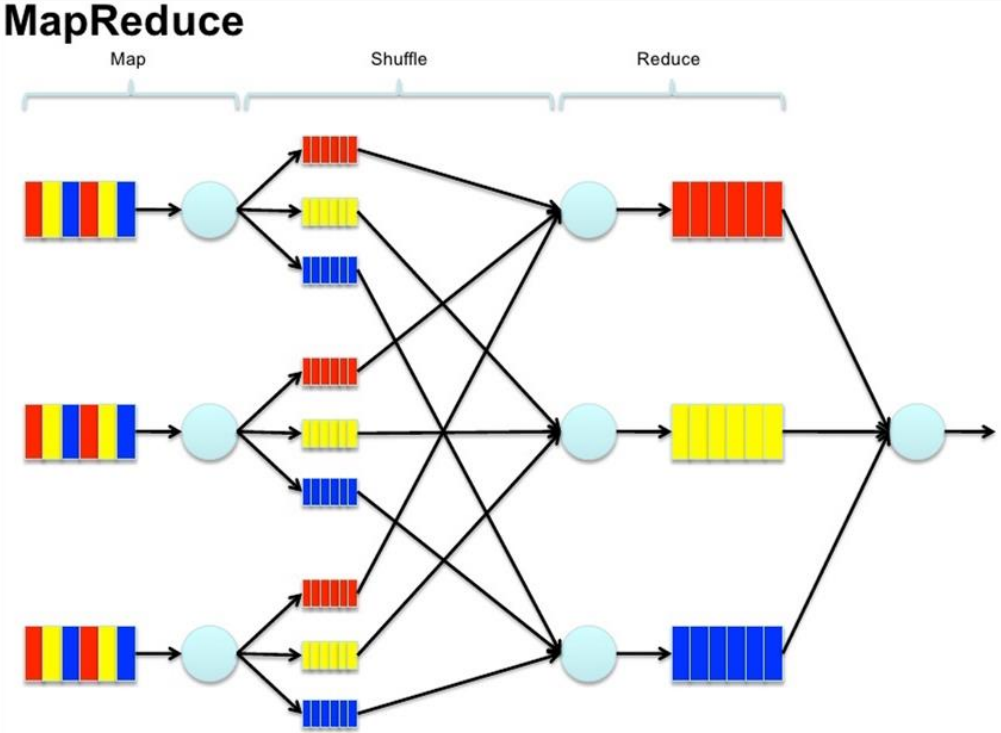
The Power of BigQuery



Query Syntax



MapReduce

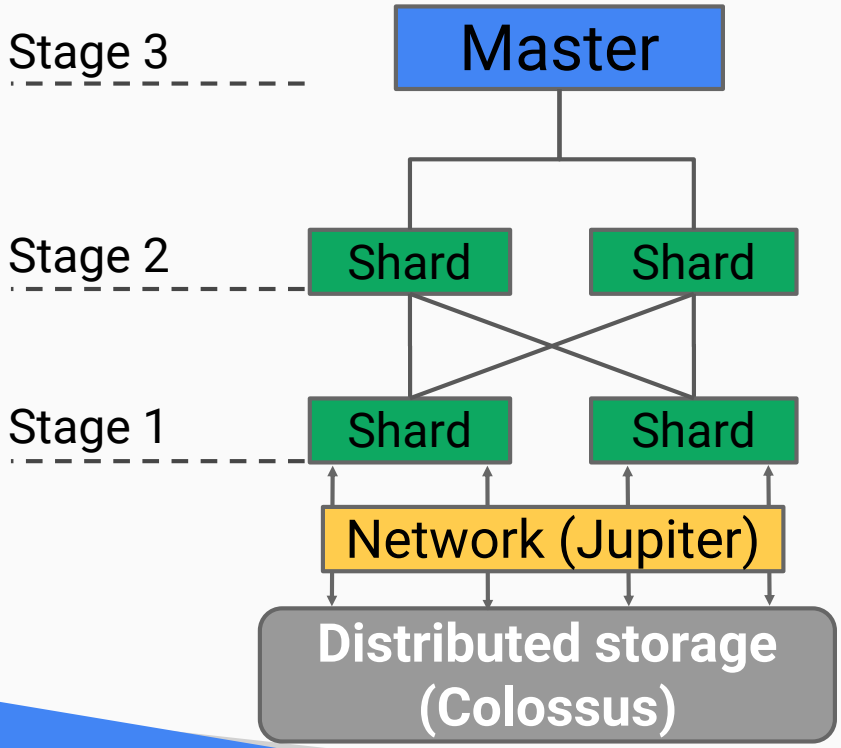


Query Processing Example

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

60 rows output (after Stage 2)

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

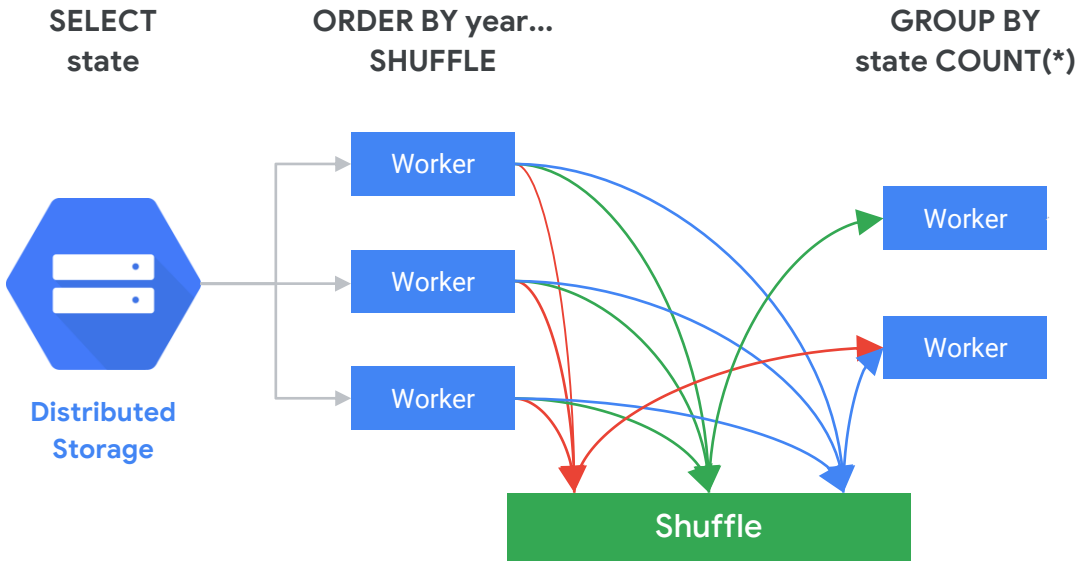
~35K rows output (after Stage 1)

WRITE: count_babies, state, year BY HASH(state, year)
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



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

BigQuery: Interoperability

BQ Storage API

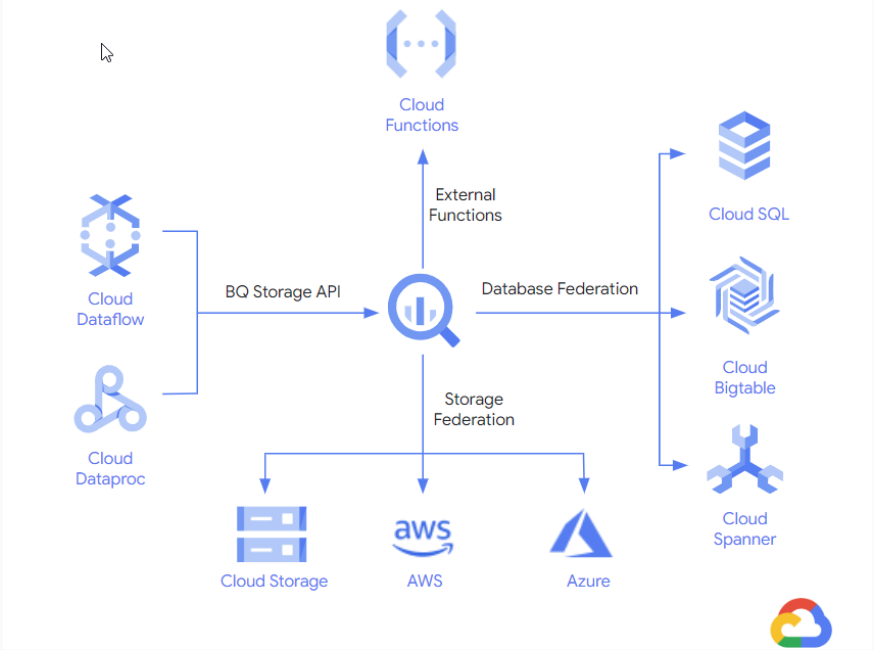
High-performance BigQuery Storage API for Dataflow and Dataproc, break down the Data Warehouse storage wall.

External Storage Federation

Easily integrate with data lakes and open formats on GCP and on other clouds with BigQuery Omni

External Database Federation

Query your Cloud SQL and Cloud Spanner instances directly from BigQuery, without moving data around.

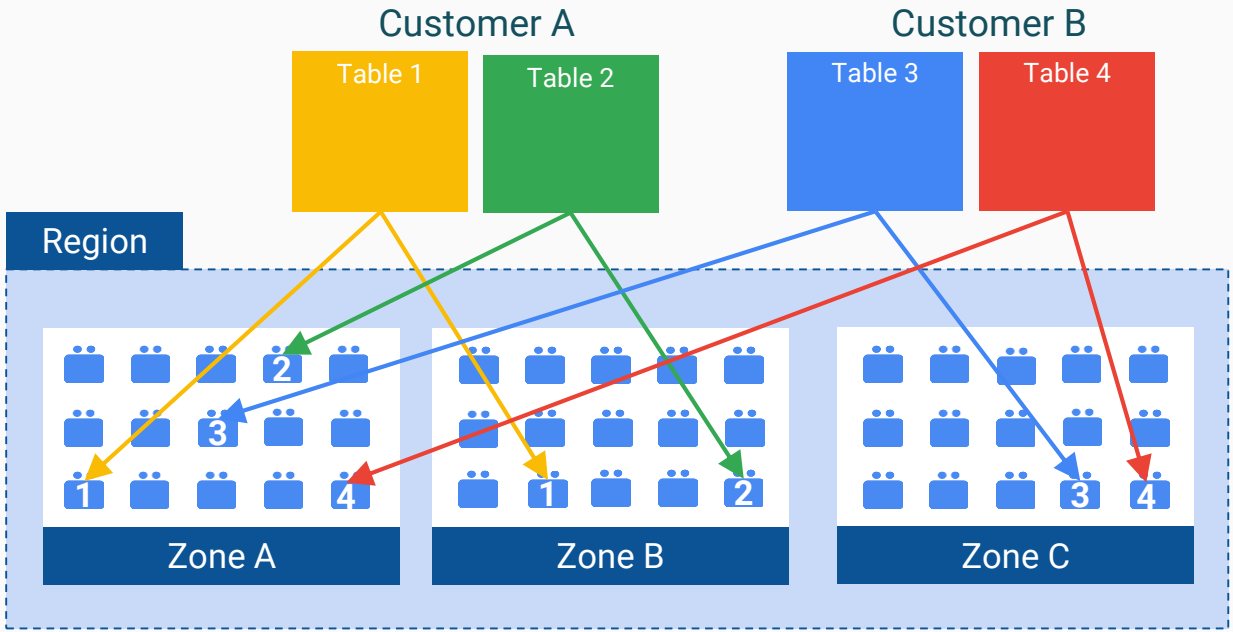


BigQuery Storage

BigQuery | Managed storage

Durable and persistent storage with automatic backup

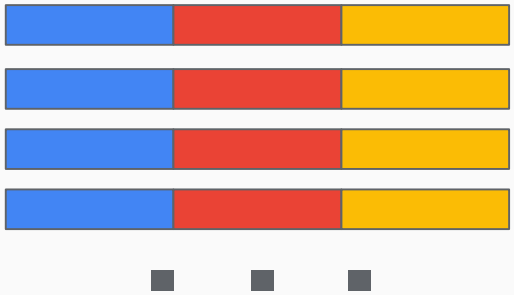
- Tables are stored in optimized columnar format
- Each table is encrypted on disk
- Storage is durable & each table is replicated across datacenters



BigQuery Structure

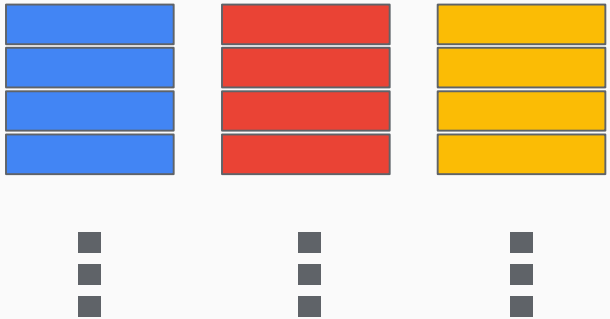
BigQuery Storage is columnar

Relational Database



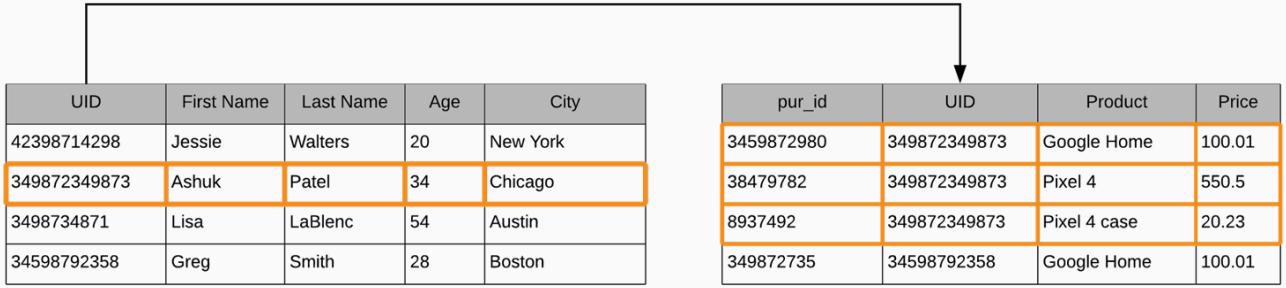
Record Oriented Storage
Supports transactional updates

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

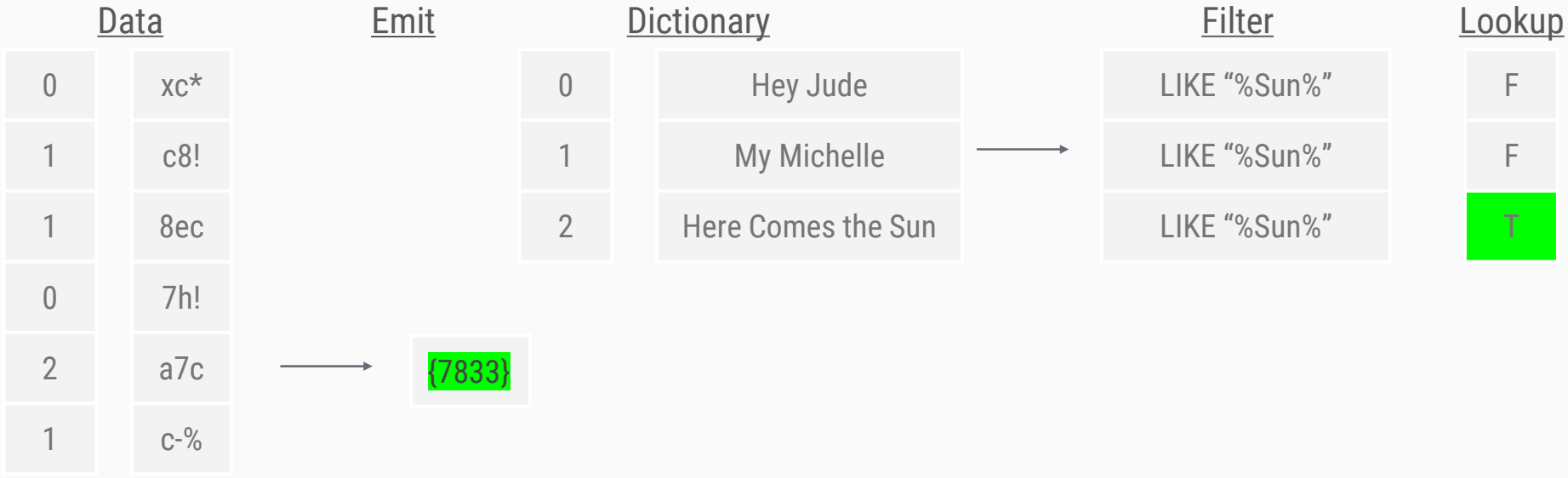
349872349873	Ashuk	Patel	34	Chicago	[
					{
					"pur_id": 3459872980,
					"UID": 349872349873,
					"Product": "Google Home",
					"Price": 100.01
					},
					{
					"pur_id": 38479782,
					"UID": 349872349873,
					"Product": "Pixel 4",
					"Price": 550.5
					},
					{
					"pur_id": 8937492,
					"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%";
```



Capacitor - Columnar storage

- Allows skipping of unused files
- Provides fast lookup of selected columns
- Provides better compression
- Builds an approximation model to select compression techniques like RLE, dictionary encoding, Bit-Vector encoding, Frame of Reference encoding, etc. to minimize the impact of non-deterministic behavior of data
- Stores and regularly updates statistics about data columns

To read more: bit.ly/inside-capacitor

BigQuery Partitions & Clustering

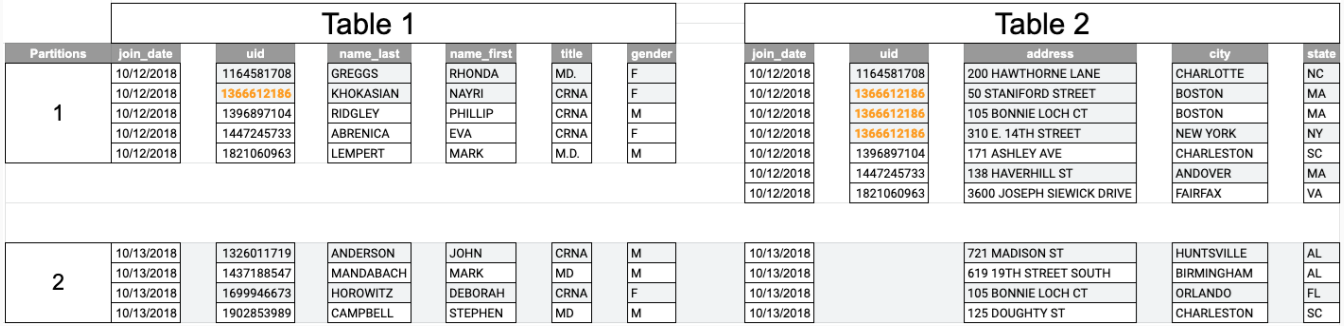
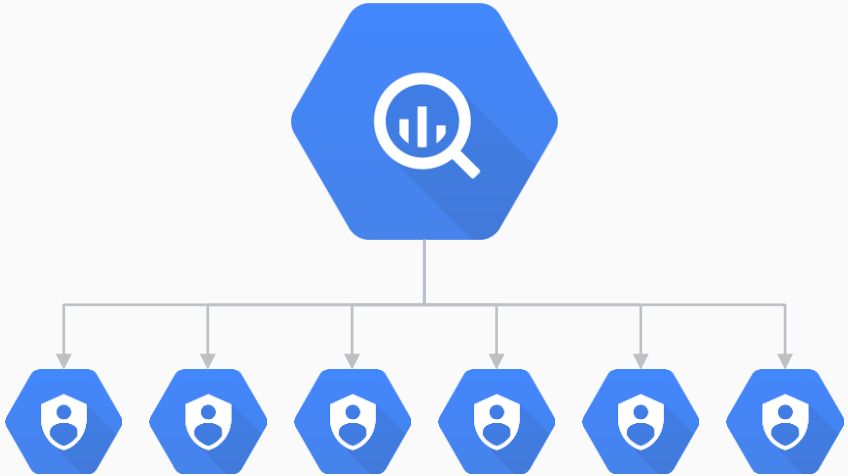


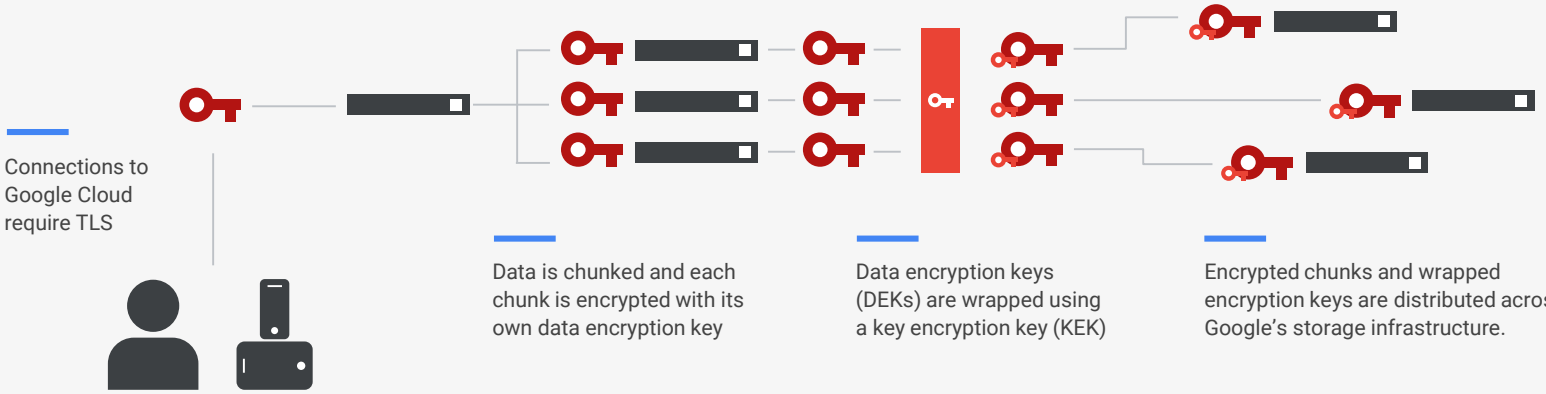
	Table 1						Table 2				
Partitions	join_date	uid	name_last	name_first	title	gender	join_date	uid	address	city	state
1	10/12/2018	1164581708	GREGGS	RHONDA	MD.	F	10/12/2018	1164581708	200 HAWTHORNE LANE	CHARLOTTE	NC
	10/12/2018	1366612186	KHOKASIAN	NAYRI	CRNA	F	10/12/2018	1366612186	50 STANIFORD STREET	BOSTON	MA
	10/12/2018	1396897104	RIDGLEY	PHILLIP	CRNA	M	10/12/2018	1366612186	105 BONNIE LOCH CT	BOSTON	MA
	10/12/2018	1447245733	ABRENICA	EVA	CRNA	F	10/12/2018	1366612186	310 E. 14TH STREET	NEW YORK	NY
	10/12/2018	1821060963	LEMPERT	MARK	M.D.	M	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	1326011719	ANDERSON	JOHN	CRNA	M	10/13/2018		721 MADISON ST	HUNTSVILLE	AL
	10/13/2018	1437188547	MANDABACH	MARK	MD	M	10/13/2018		619 19TH STREET SOUTH	BIRMINGHAM	AL
	10/13/2018	1699946673	HOROWITZ	DEBORAH	CRNA	F	10/13/2018		105 BONNIE LOCH CT	ORLANDO	FL
	10/13/2018	1902853989	CAMPBELL	STEPHEN	MD	M	10/13/2018		125 DOUGHTY ST	CHARLESTON	SC

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

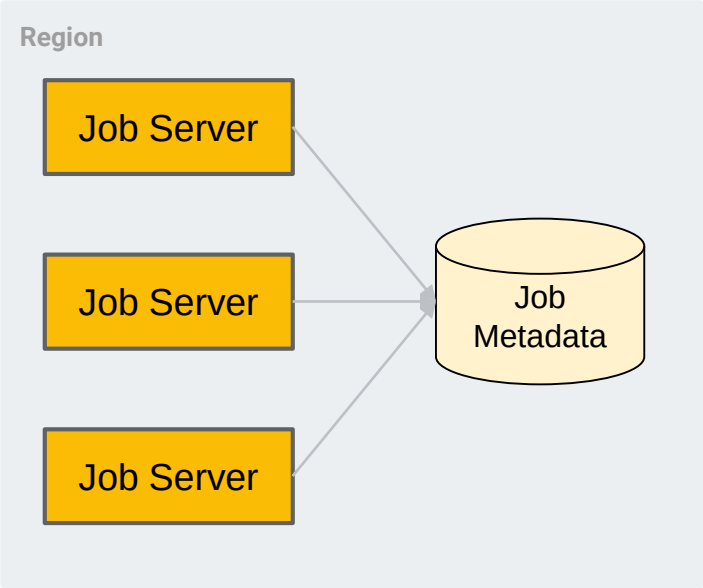
Data is chunked and each chunk is encrypted with its own data encryption key

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.

Query Overview | Job Queueing

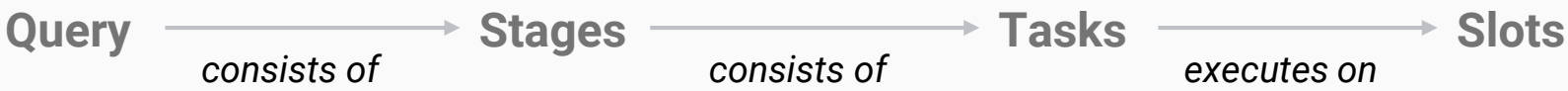
- Jobs start in the **PENDING** state.
  - Can transition to either **RUNNING** or **DONE** (due to timeout).
  - Most jobs immediately enter the **RUNNING** state.
- Jobs defer their **RUNNING** transition when:
  - **BATCH** priority: always defer at least 1 minute, longer if awaiting quota or the 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



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

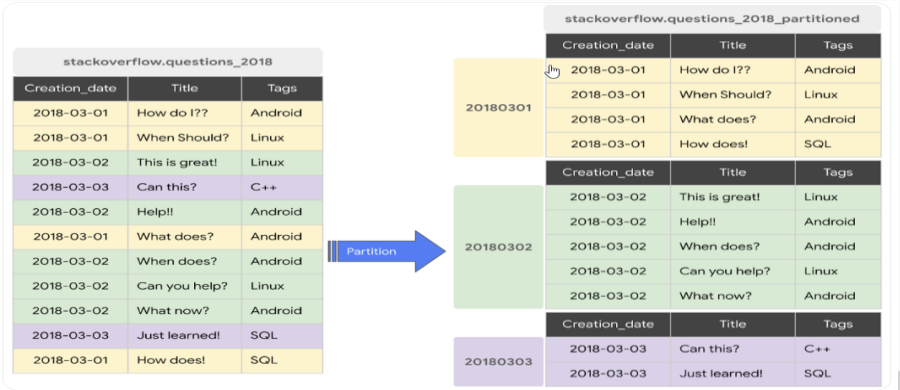
Partitioning

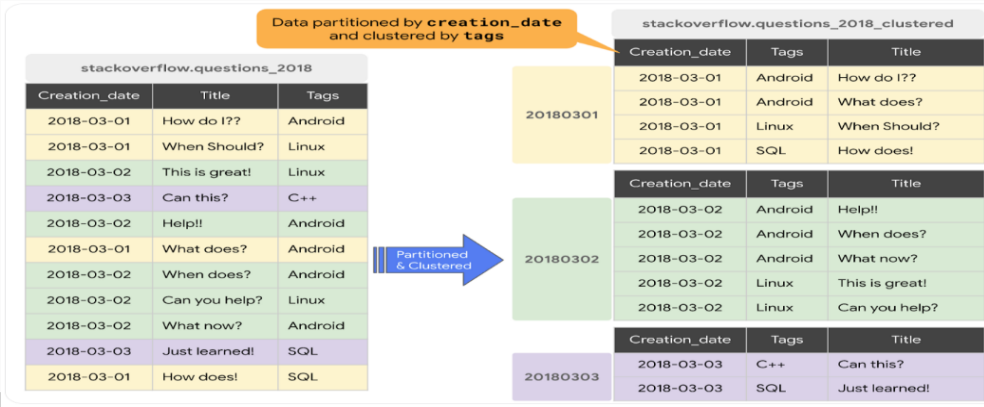
Filtering storage before query execution begins to reduce costs. Reduces a full table scan to the partitions specified. Single column, lower cardinality (e.g. thousands of partitions).

- **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

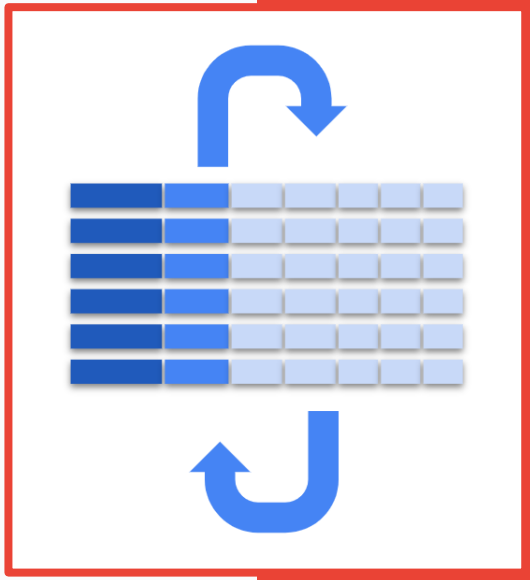
Doesn't consume your query resources

maintenance-free

Requires no setup or maintenance

autonomous

Automatically happens in the background



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

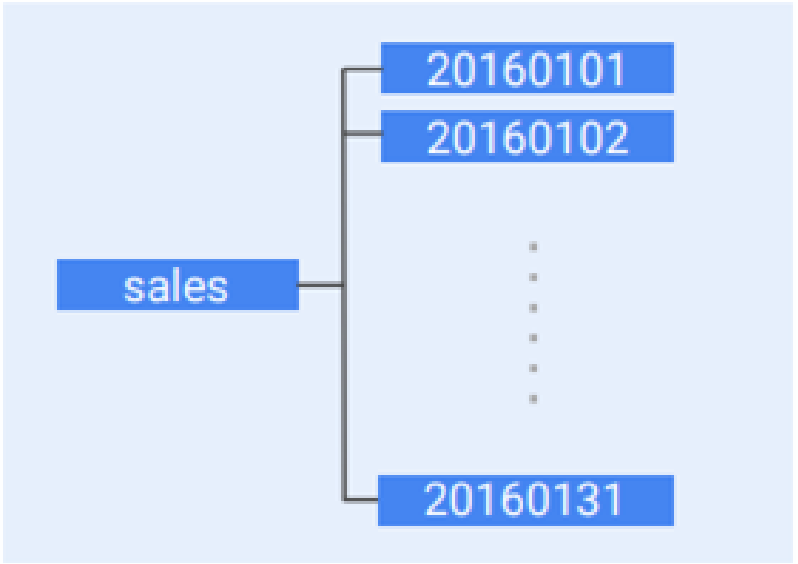
- Joins — In what order are you merging data?
- Guideline — Biggest, Smallest, Decreasing Size Thereafter
- Avoid self-join if you can, since it squares the number of rows processed

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 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

GIS

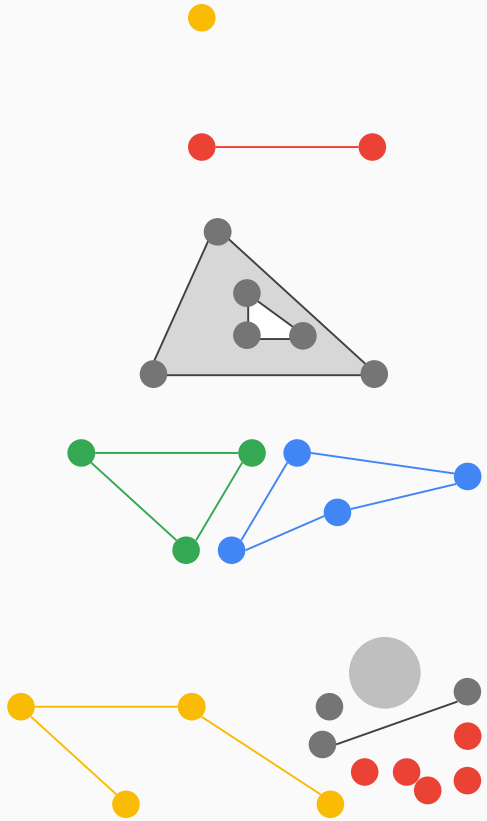
Point

Linestring

Polygon

Multi-Polygon

Collections



Formats

WKT

GeoJSON

WKB

Behind the scenes - BigQuery ML

Through SQL and within BigQuery

Leverage BigQuery's processing power to build a model

Auto-tuned learning rate

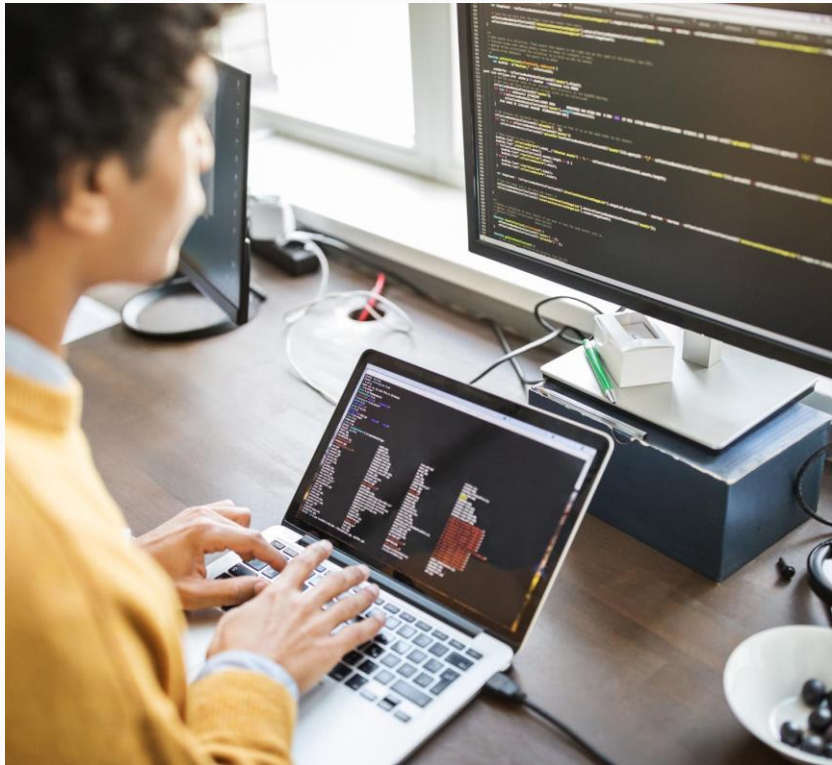
Auto-split of data into training and test

Null imputation

Standardization of numeric features

One-hot encoding of strings

Class imbalance handling



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)

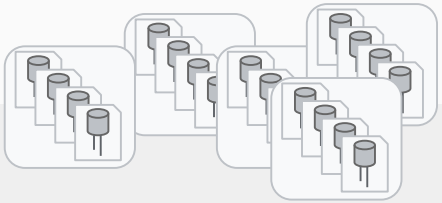
XGBoost classifier (Beta)

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)
```

Thank You!