Google Cloud

Building a Data Warehouse

Agenda

The modern data warehouse

Intro to BigQuery

- Getting Started
- Loading Data
- Lab: Loading Data with Console and CLI

Exploring Schemas

- Schema Design
- Nested and Repeated Fields
- Lab: ARRAYs and STRUCTs
- Optimizing with Partitioning and Clustering

Preview: Transforming Batch and Streaming Data

A data warehouse should consolidate data from many sources

The data in a warehouse should have quality, consistency, and accuracy

A data warehouse should be optimized for simplicity of access and high-speed query performance

Google Cloud

A modern data warehouse

- Gigabytes to petabytes

A modern data warehouse

- Gigabytes to petabytes
- Serverless and no-ops, including ad hoc queries

A modern data warehouse

- Gigabytes to petabytes
- Serverless and no-ops, including ad hoc queries
- Ecosystem of visualization and reporting tools

A modern data warehouse

- Gigabytes to petabytes
- Serverless and no-ops, including ad hoc queries
- Ecosystem of visualization and reporting tools
- Ecosystem of ETL and data processing tools

A modern data warehouse

- Gigabytes to petabytes
- Serverless and no-ops, including ad hoc queries
- Ecosystem of visualization and reporting tools
- Ecosystem of ETL and data processing tools
- Up-to-the-minute data

A modern data warehouse

- Gigabytes to petabytes
- Serverless and no-ops, including ad hoc queries
- Ecosystem of visualization and reporting tools
- Ecosystem of ETL and data processing tools
- Up-to-the-minute data
- Machine learning

A modern data warehouse

- Gigabytes to petabytes
- Serverless and no-ops, including ad hoc queries
- Ecosystem of visualization and reporting tools
- Ecosystem of ETL and data processing tools
- Up-to-the-minute data
- Machine learning
- Security and collaboration

Agenda

The modern data warehouse

Intro to BigQuery

- Getting Started
- Loading Data
- Lab: Loading Data with Console and CLI

Exploring Schemas

- Schema Design
- Nested and Repeated Fields
- Lab: ARRAYs and STRUCTs
- Optimizing with Partitioning and Clustering

Preview: Transforming Batch and Streaming Data

BigQuery has many capabilities that make it an ideal data warehouse

BigQuery

- Interactive SQL queries over large datasets (petabytes) in seconds
- Serverless and no-ops, including ad hoc queries
- Ecosystem of visualization and reporting tools
- Ecosystem of ETL and data processing tools
- Up-to-the-minute data
- Machine Learning
- Security and collaboration

BigQuery has many capabilities that make it an ideal data warehouse

- Interactive SQL queries over large datasets (petabytes) in seconds
- Serverless and no-ops, including ad hoc queries
- Ecosystem of visualization and reporting tools
- Ecosystem of ETL and data processing tools
- Up-to-the-minute data
- Machine Learning
- Security and collaboration

BigQuery has many capabilities that make it an ideal data warehouse

- Interactive SQL queries over large datasets (petabytes) in seconds
- Serverless and no-ops, including ad hoc queries
- Ecosystem of visualization and reporting tools
- Ecosystem of ETL and data processing tools
- Up-to-the-minute data
- Machine Learning
- Security and collaboration

BigQuery has many capabilities that make it an ideal data warehouse

- Interactive SQL queries over large datasets (petabytes) in seconds
- Serverless and no-ops, including ad hoc queries
- Ecosystem of visualization and reporting tools
- Ecosystem of ETL and data processing tools
- Up-to-the-minute data
- Machine Learning
- Security and collaboration

BigQuery has many capabilities that make it an ideal data warehouse

- Interactive SQL queries over large datasets (petabytes) in seconds
- Serverless and no-ops, including ad hoc queries
- Ecosystem of visualization and reporting tools
- Ecosystem of ETL and data processing tools
- Up-to-the-minute data
- Machine Learning
- Security and collaboration

BigQuery has many capabilities that make it an ideal data warehouse

- Interactive SQL queries over large datasets (petabytes) in seconds
- Serverless and no-ops, including ad hoc queries
- Ecosystem of visualization and reporting tools
- Ecosystem of ETL and data processing tools
- Up-to-the-minute data
- Machine Learning
- Security and collaboration

BigQuery has many capabilities that make it an ideal data warehouse

BigQuery

- Interactive SQL queries over large datasets (petabytes) in seconds
- Serverless and no-ops, including ad hoc queries
- Ecosystem of visualization and reporting tools
- Ecosystem of ETL and data processing tools
- Up-to-the-minute data
- Machine Learning
- Security and collaboration

BigQuery has many capabilities that make it an ideal data warehouse

- Interactive SQL queries over large datasets (petabytes) in seconds
- Serverless and no-ops, including ad hoc queries
- Ecosystem of visualization and reporting tools
- Ecosystem of ETL and data processing tools
- Up-to-the-minute data
- Machine Learning
- Security and collaboration

Demo

Query TB+ of data in seconds

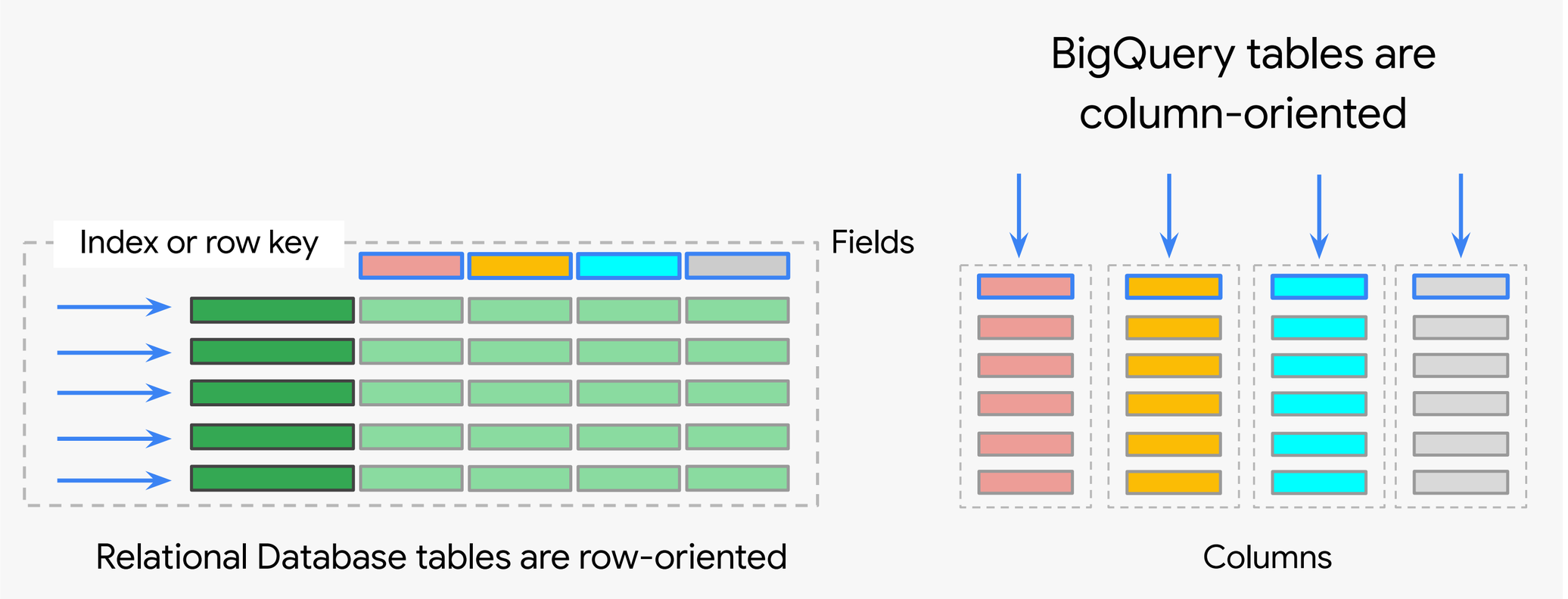
BigQuery is serverless. You don't need to worry about:

- X Data aging
- X Backups
- X Storage management
- X Fault recovery
- X Query engine optimization
- X Hardware
- X Updates

Free up real people-hours by not having to worry about common tasks

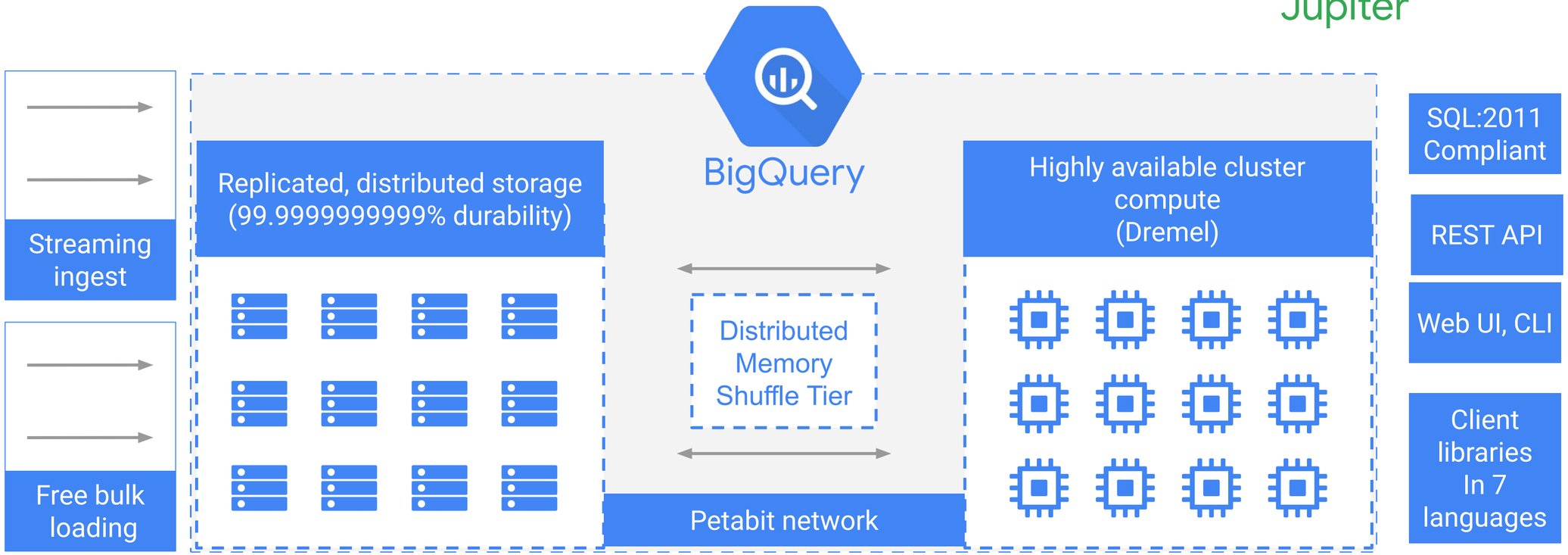
Google Cloud

What makes BigQuery fast?

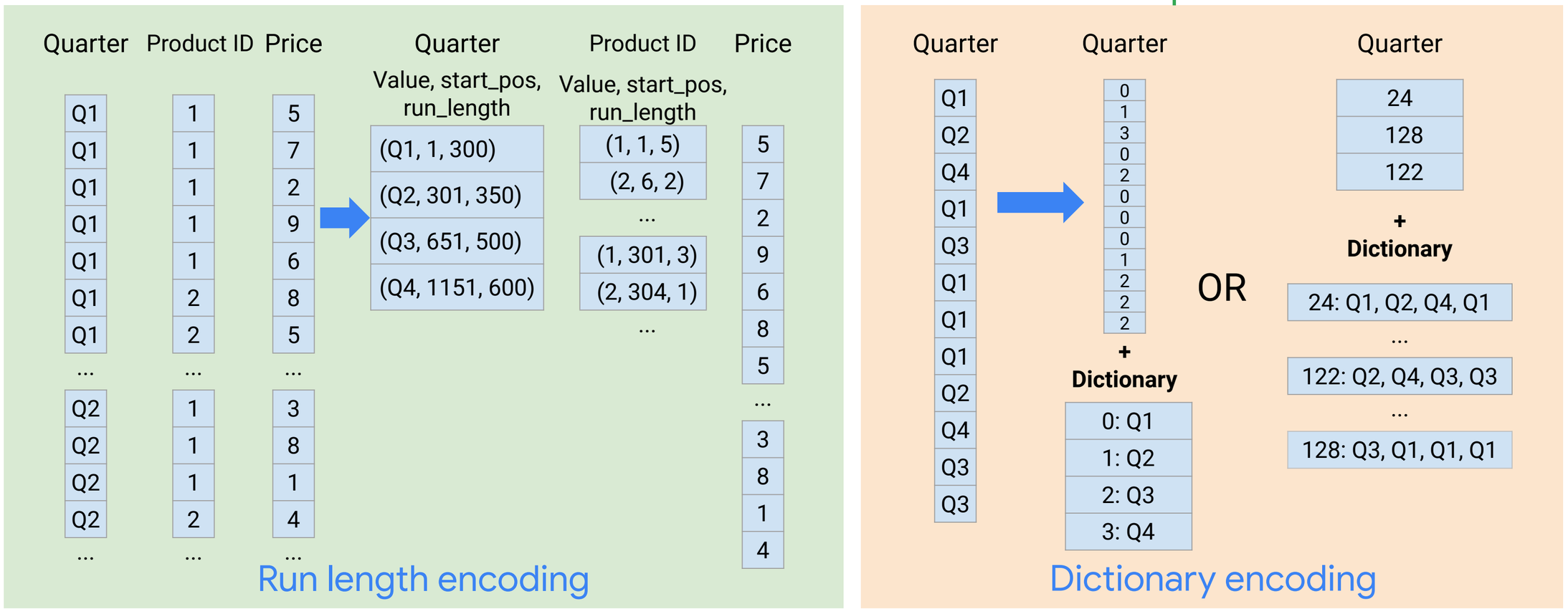


The data is physically stored in a redundant way separate from the compute cluster

Colossus
Jupiter

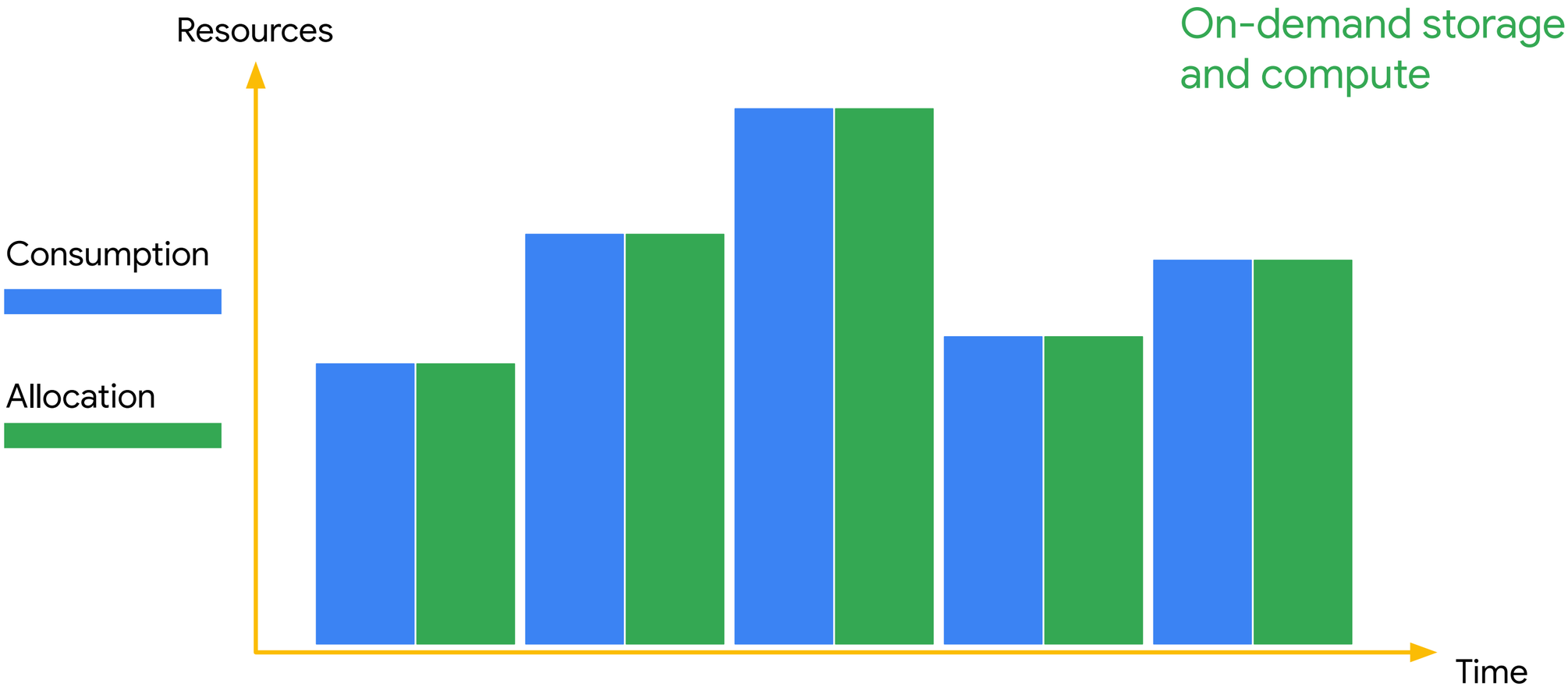


The data are also run length-encoded and dictionary-encoded

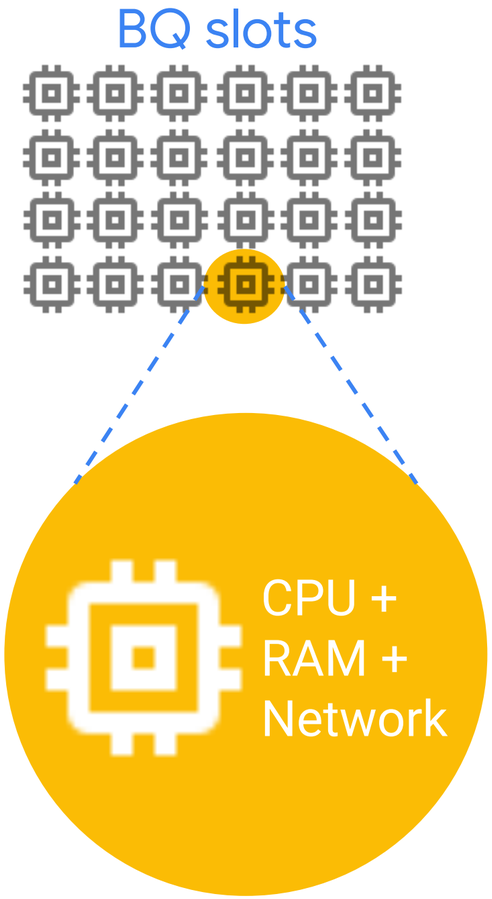


Examples taken from VLDB 2009 tutorial on Column Oriented Database Systems

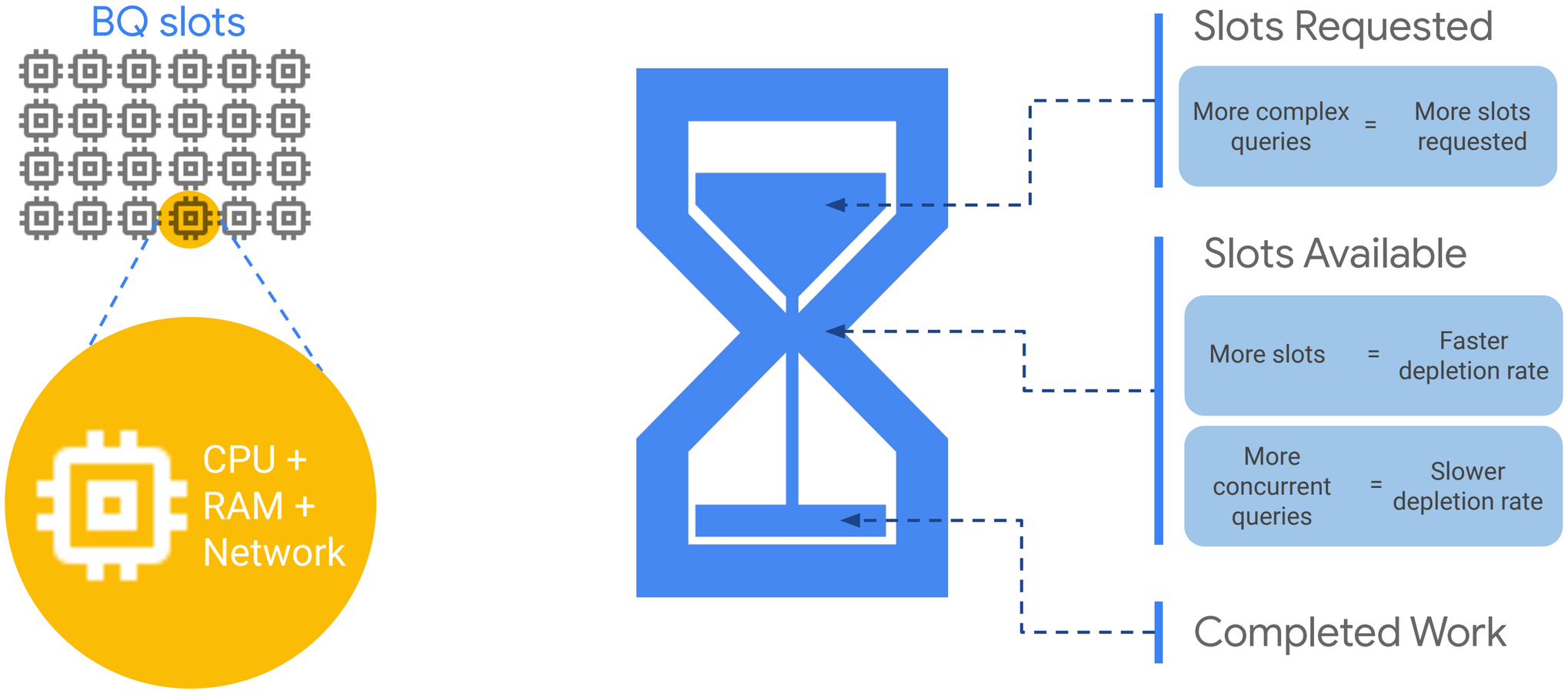
You don’t need to provision resources before using BigQuery



A BigQuery slot is a combination of CPU, memory, and networking resources



The actual number of slots allotted to a query depends on query complexity and project quota



Agenda

The modern data warehouse

Intro to BigQuery

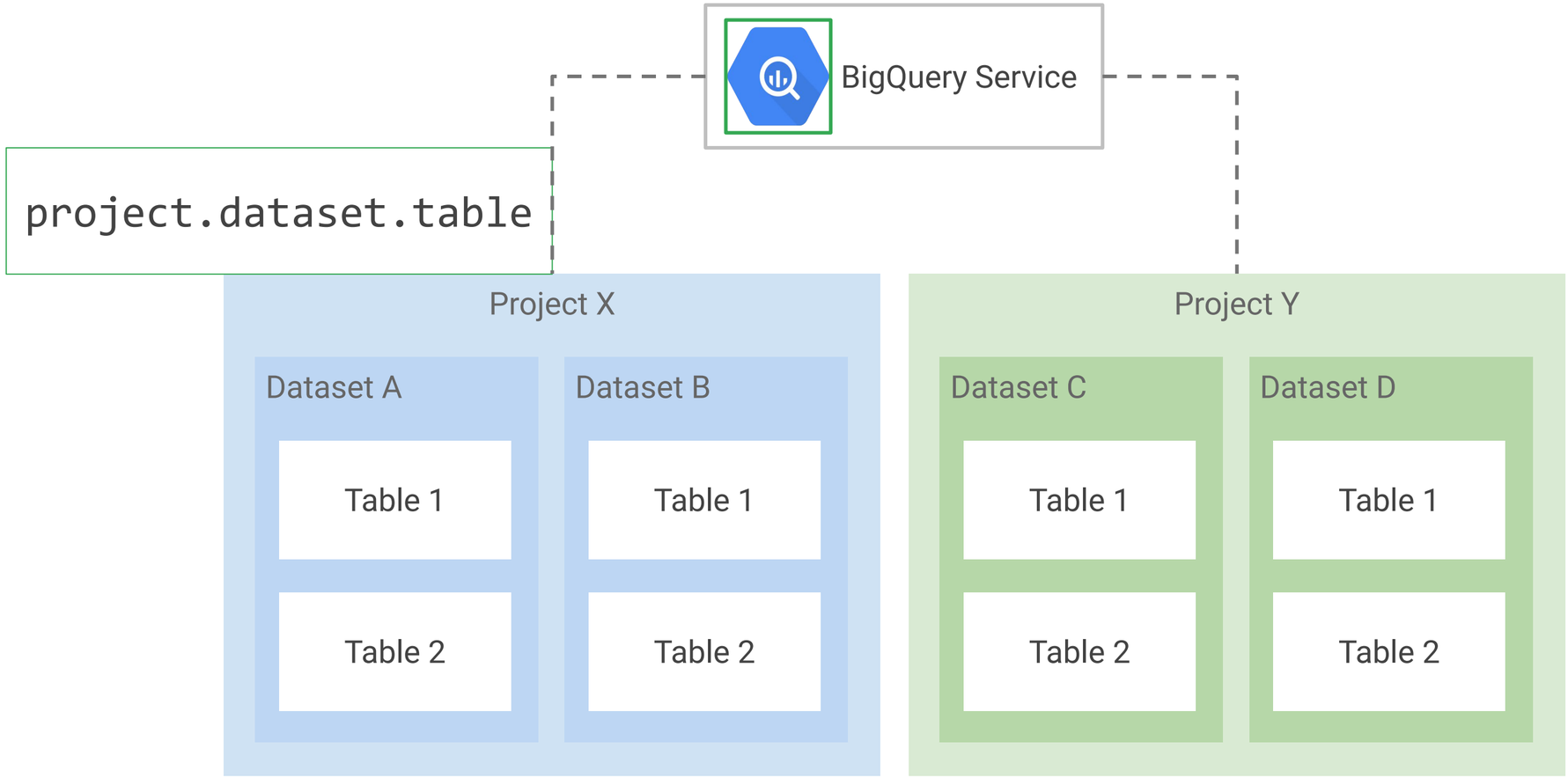
- Getting Started
- Loading Data
- Lab: Loading Data with Console and CLI

Exploring Schemas

- Schema Design
- Nested and Repeated Fields
- Lab: ARRAYs and STRUCTs
- Optimizing with Partitioning and Clustering

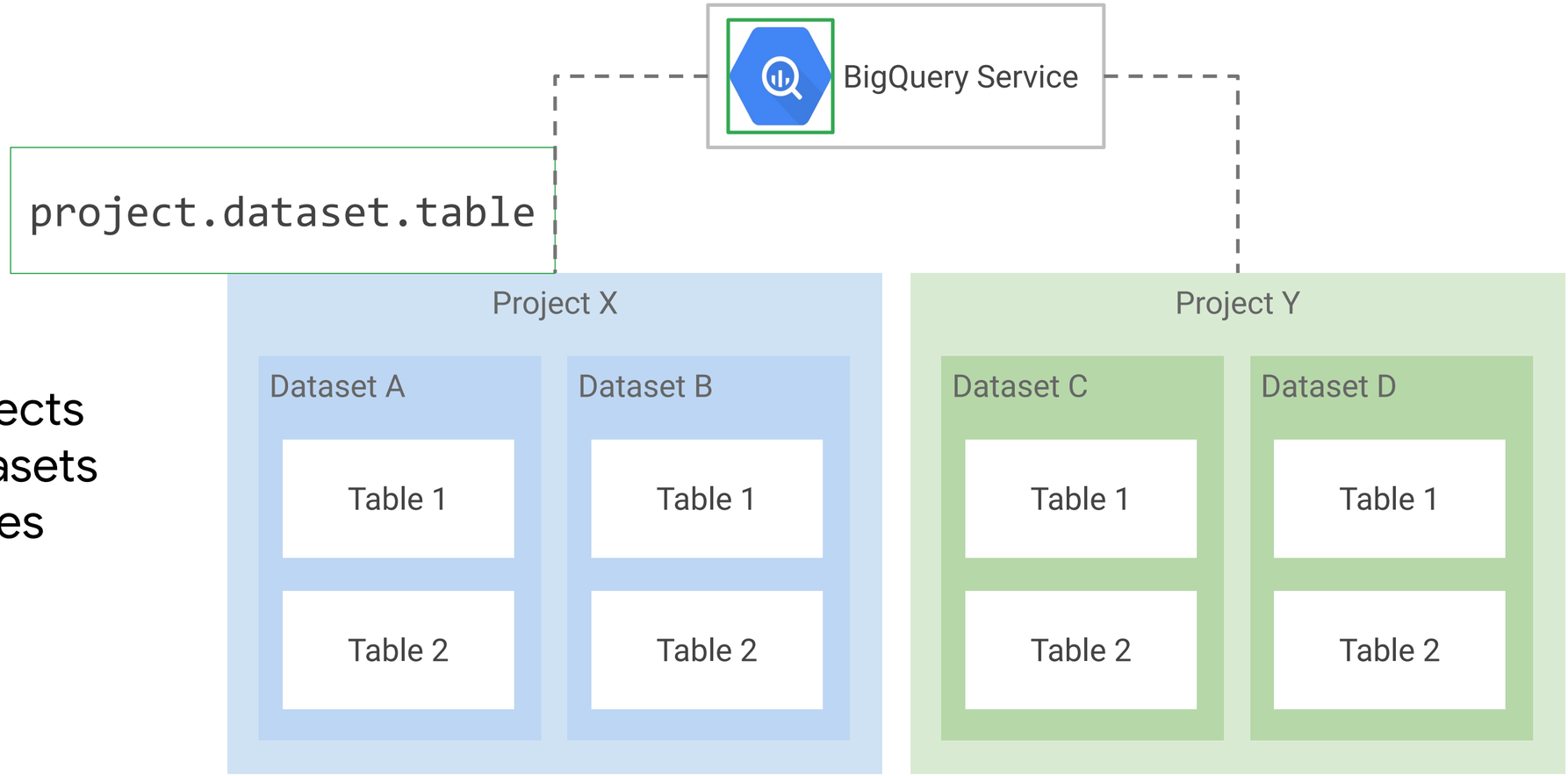
Preview: Transforming Batch and Streaming Data

BigQuery organizes data tables into units called datasets

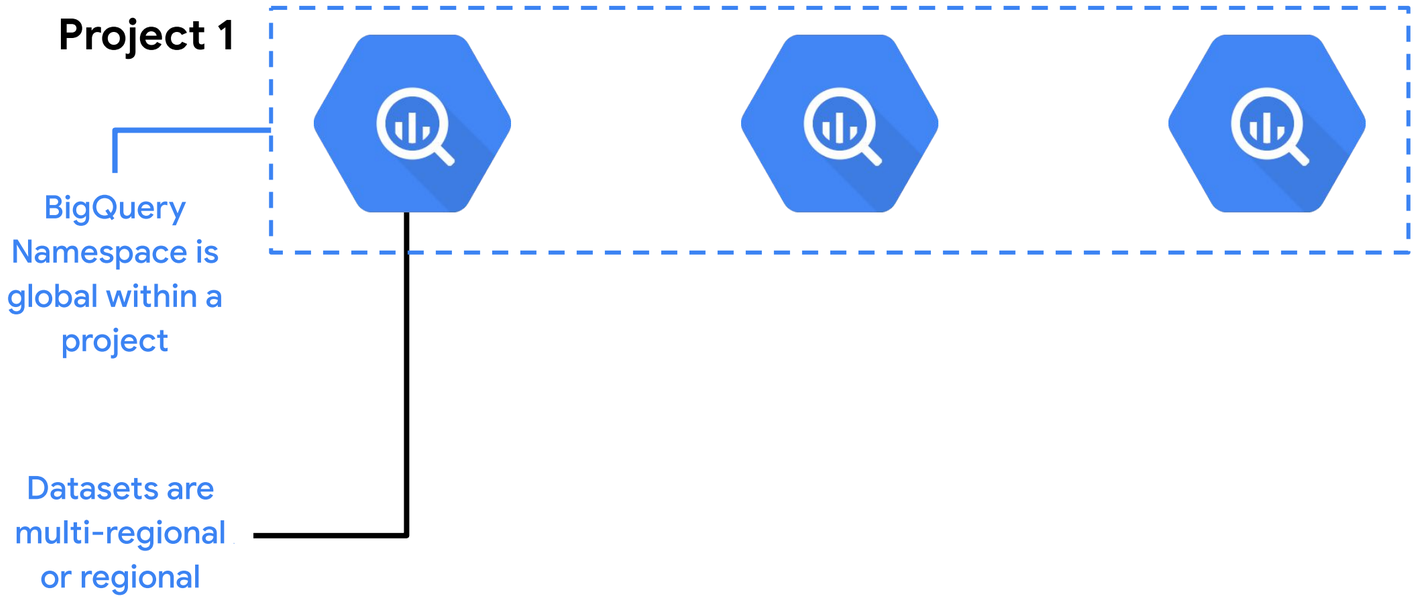


What are some reasons to structure your information?

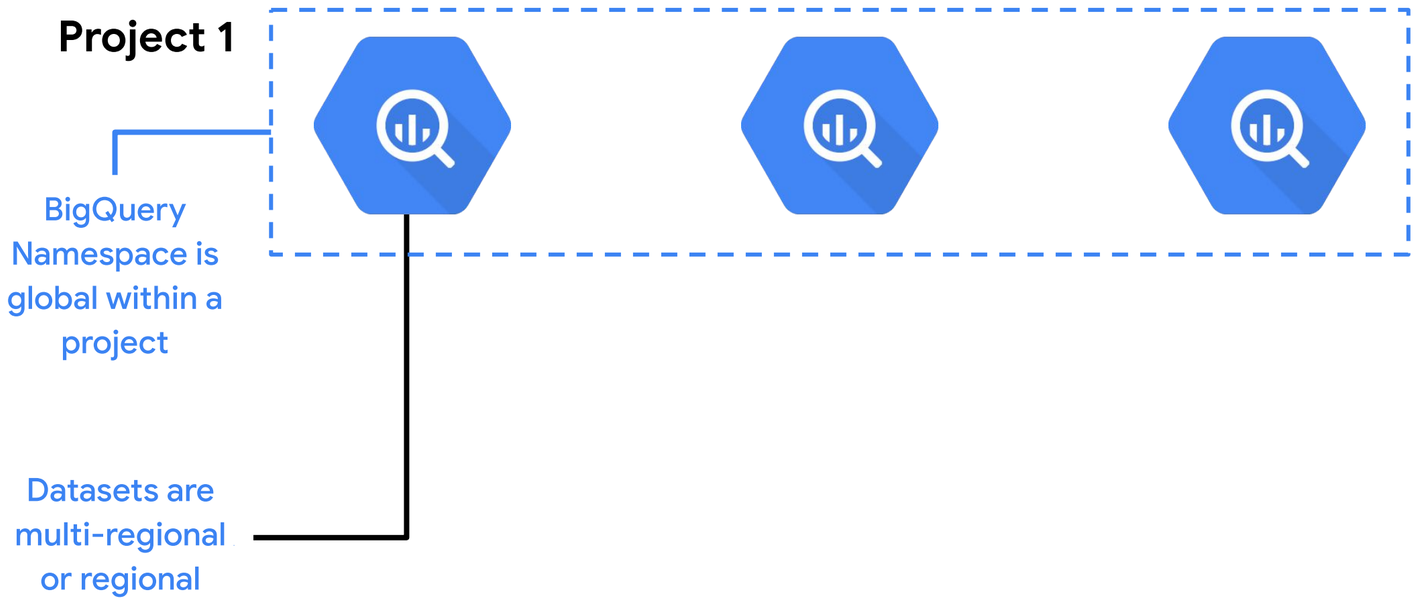
- Projects
- Datasets
- Tables



BigQuery datasets belong to a project



Access control to run a query is via Cloud IAM



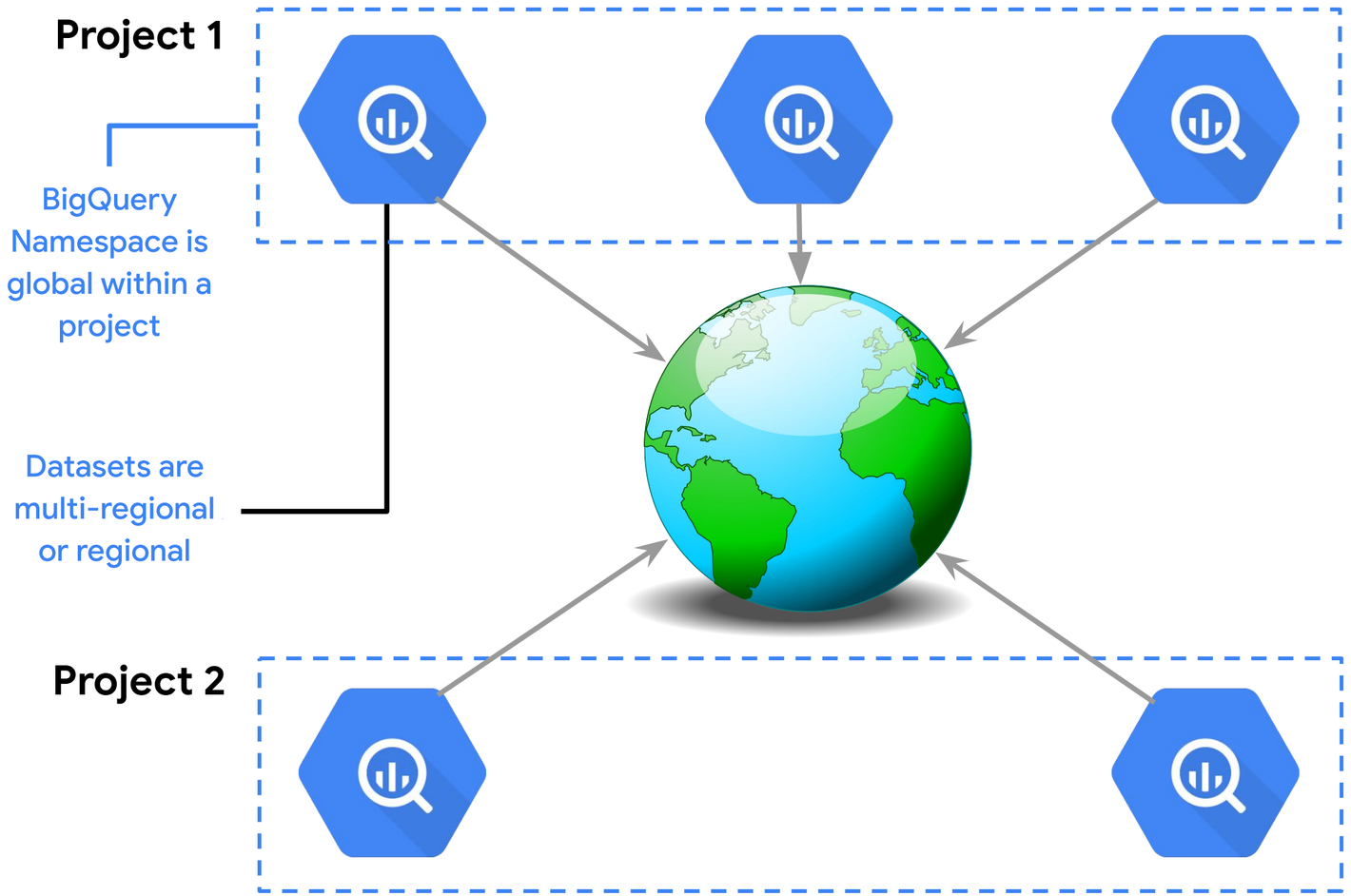
Access control is through Cloud IAM

Organization
Project
Dataset

Access control is to the Dataset, not to the Table or Column

A user has to run jobs to query datasets

BigQuery datasets can be regional or multi-regional



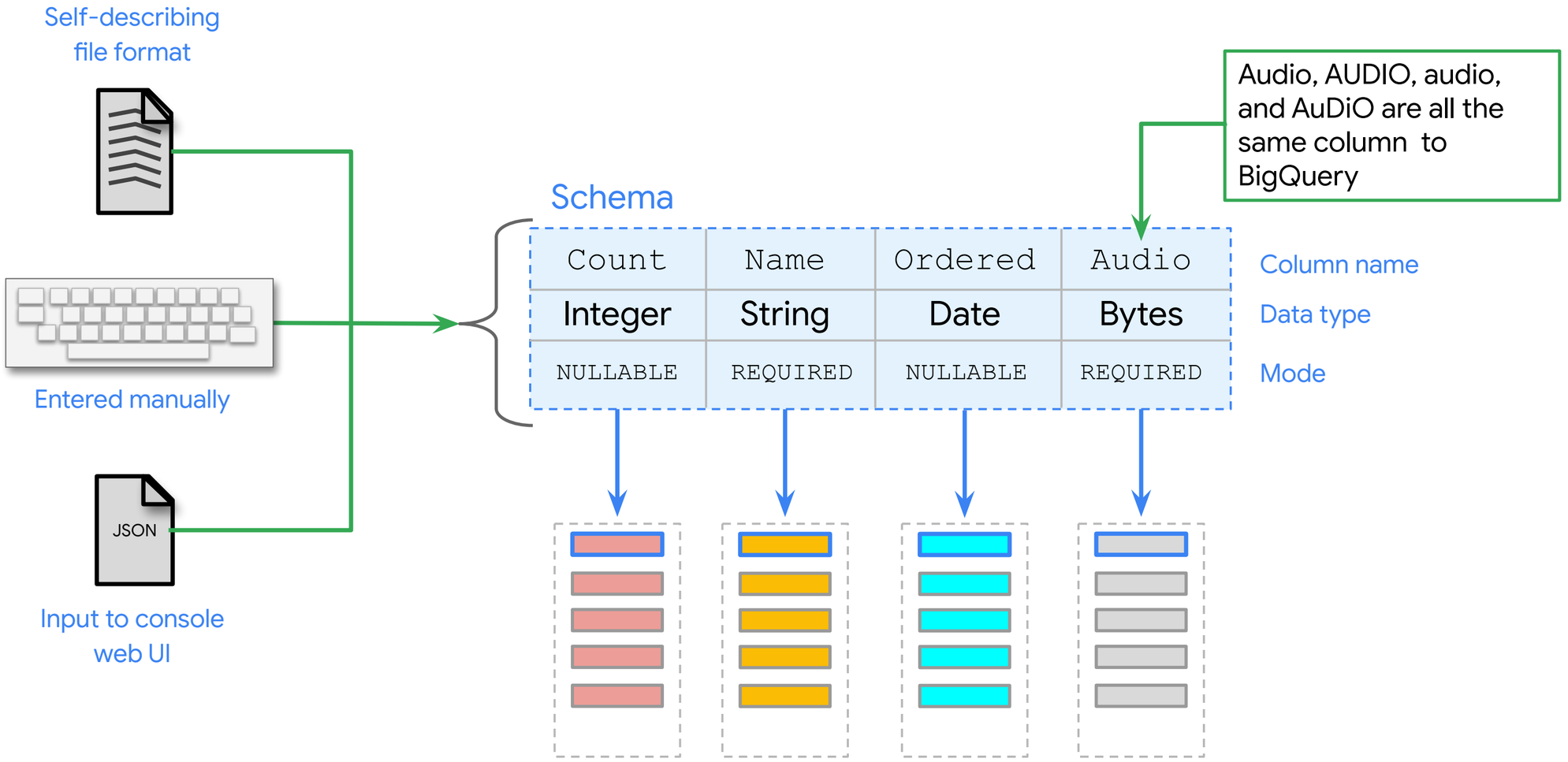
Access control is through Cloud IAM

Organization
Project
Dataset

Access control is to the Dataset, not to the Table or Column

A user has to run jobs to query datasets

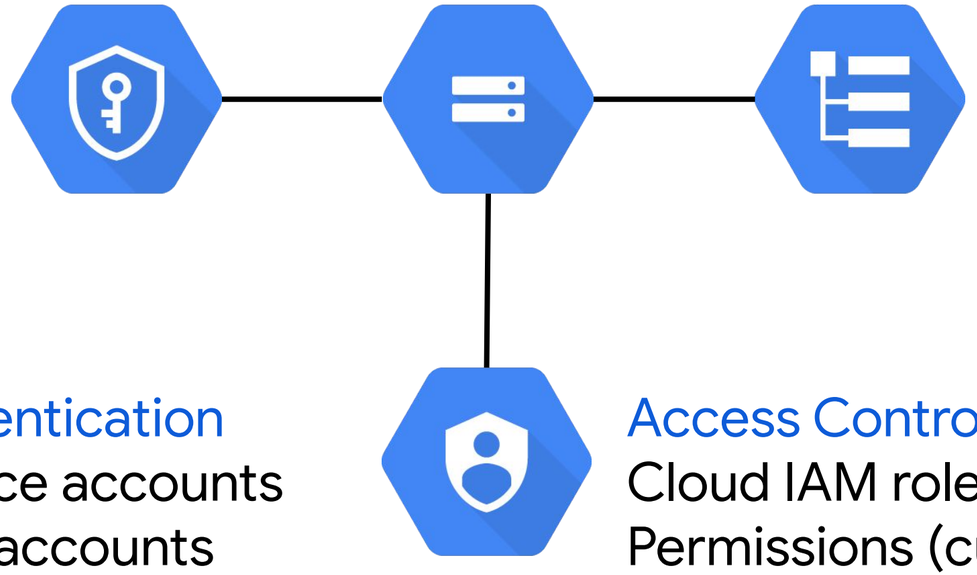
The table schema provides structure to the data



Security, encryption, and auditing for BigQuery

Key Management

Customer-managed encryption keys



Cloud Audit Logs

Admin activity
System event
Data access

Authentication

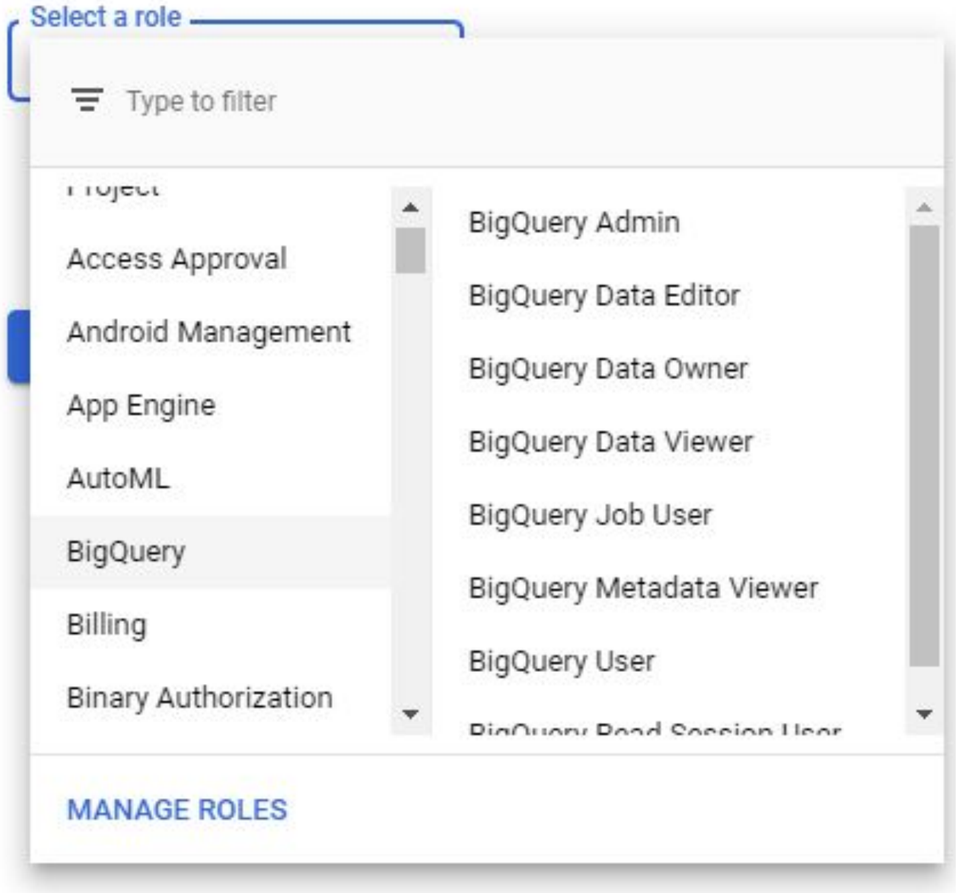
Service accounts
User accounts

Access Control

Cloud IAM roles
Permissions (custom)

Project and Dataset

BigQuery provides predefined roles for controlling access to resources



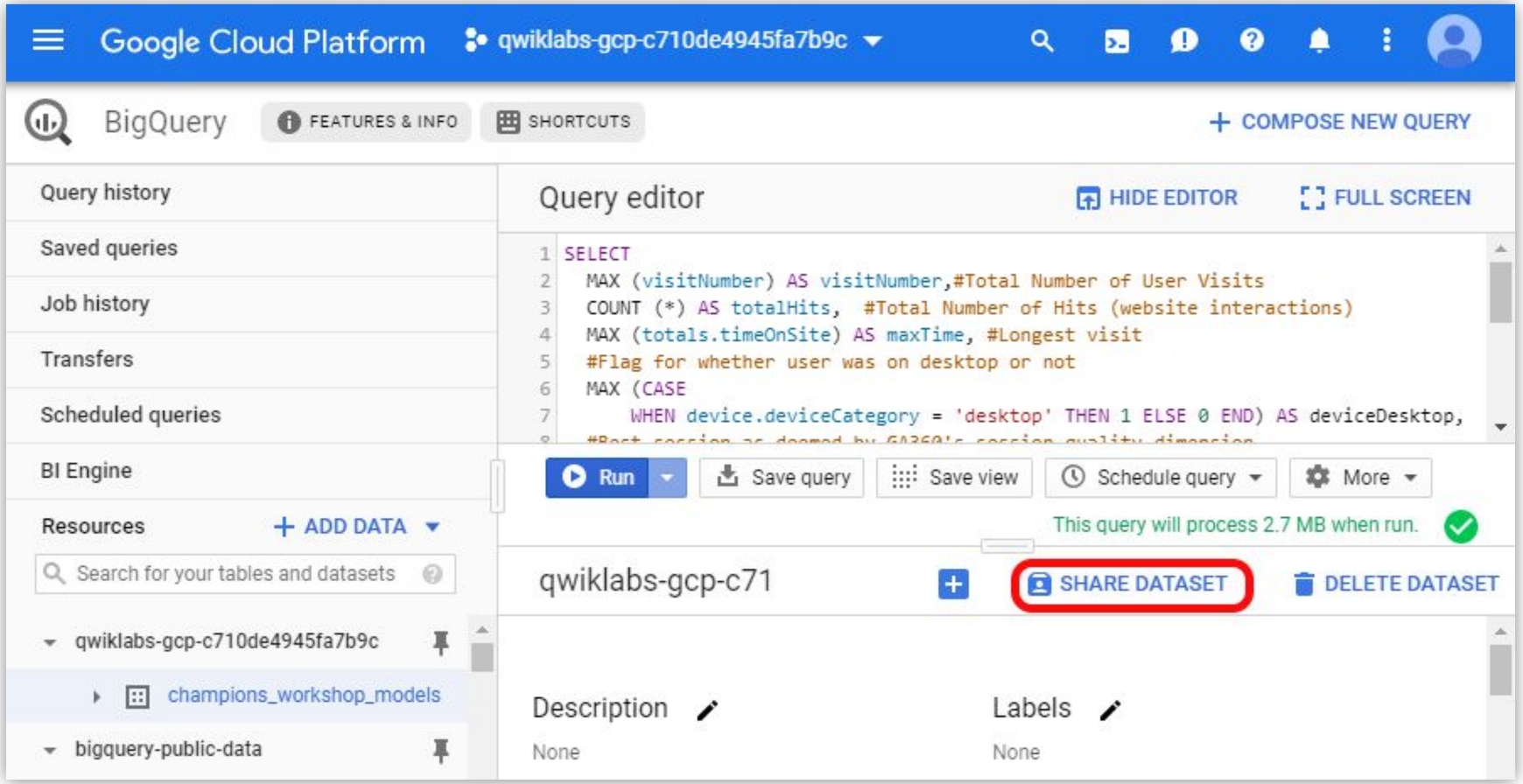
Grants

Cloud IAM grants permission to perform specific actions

IAM applies to datasets

Use authorized views to restrict at a finer-grained level

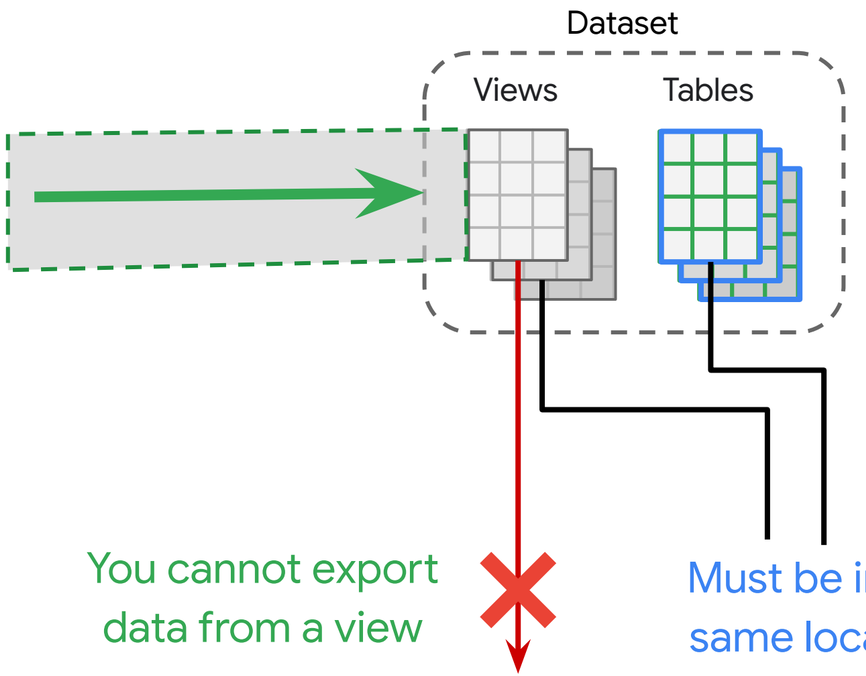
It’s easy to share access to datasets with other analysts



Google Cloud

Views add another degree of access control

A *view* is a virtual table defined by a SQL query



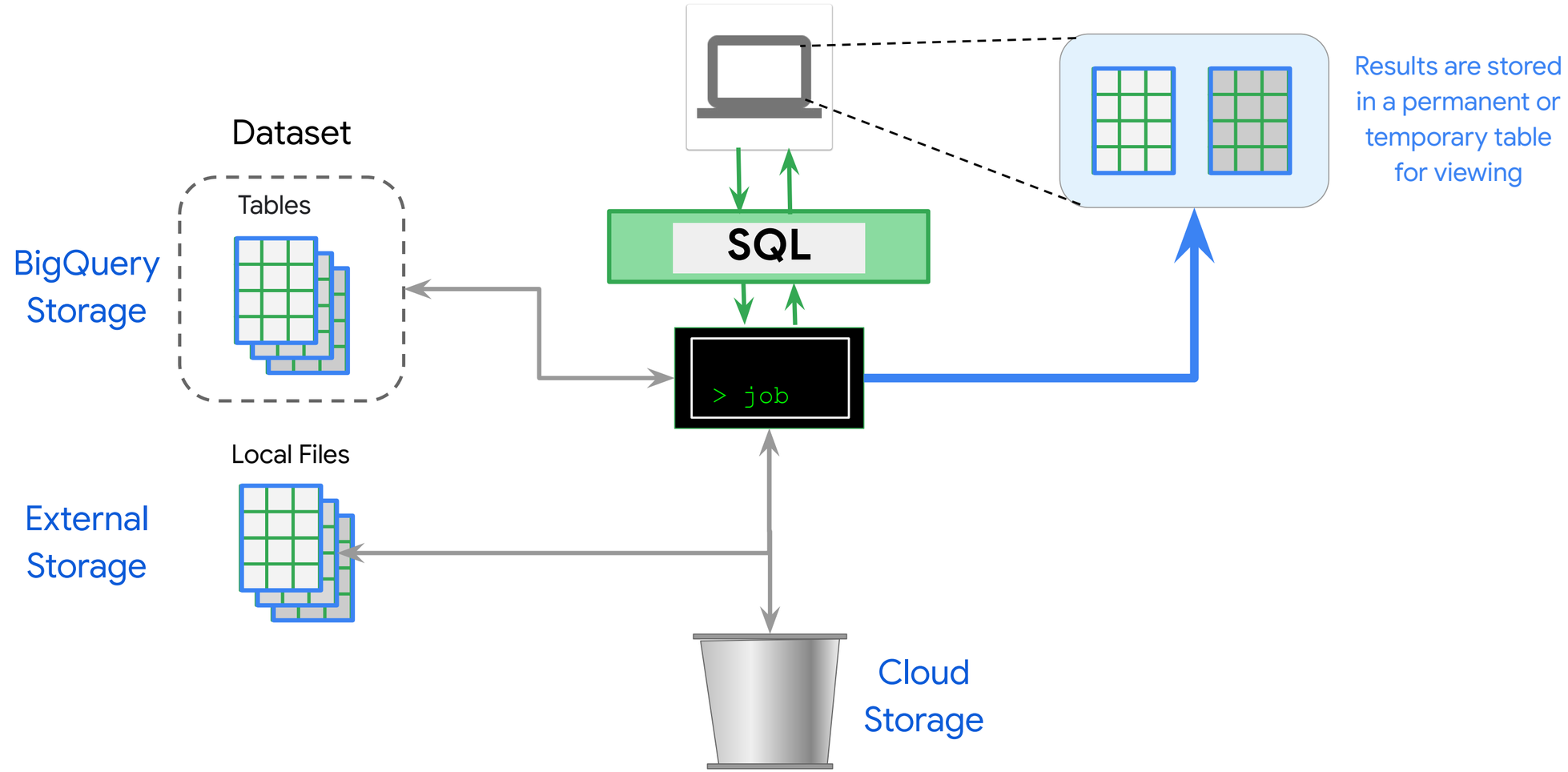
An authorized view allows you to share data externally without sharing the underlying table

```
CREATE VIEW dsB.myview AS
SELECT
name, number
FROM dsB.mytable
WHERE year >1950
```

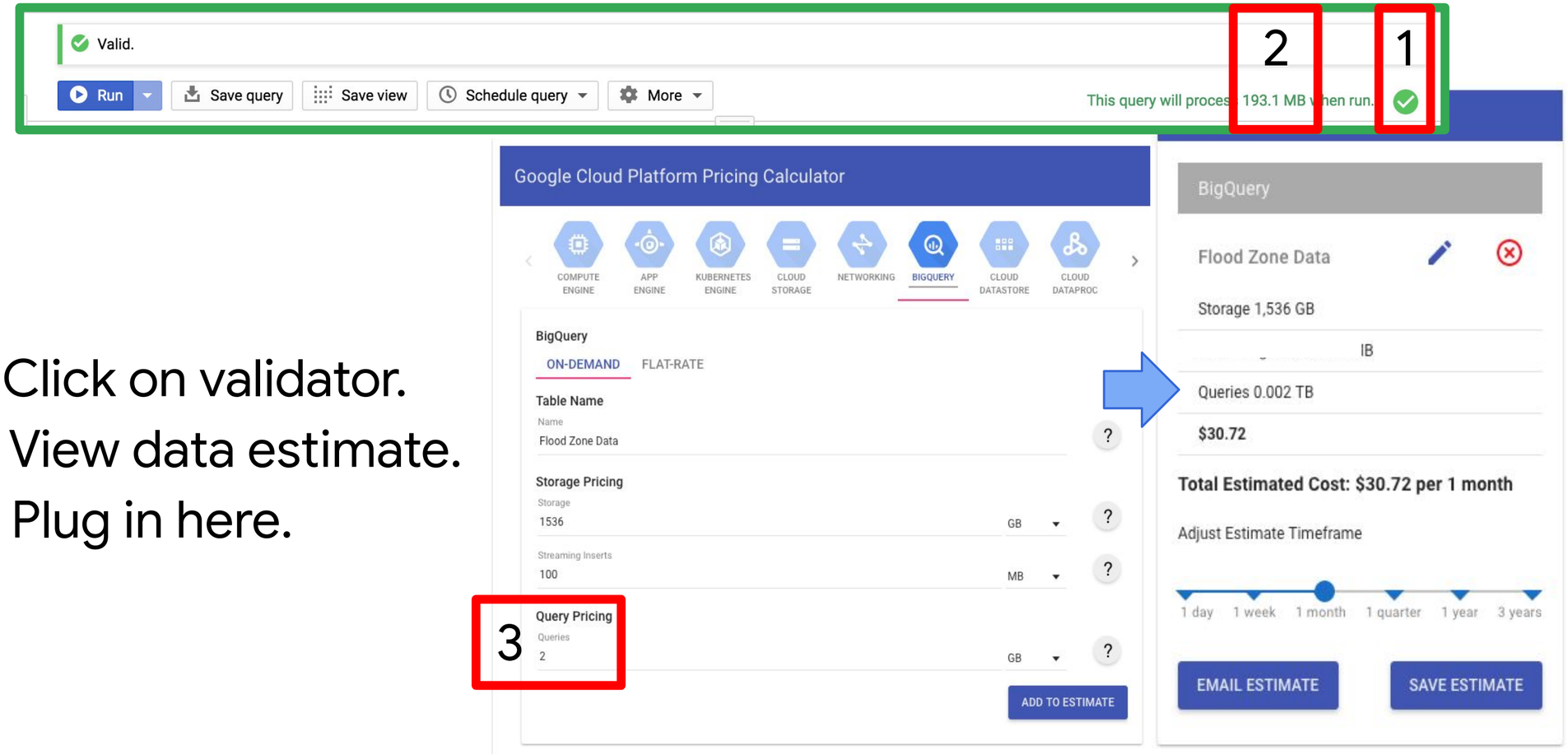
You cannot export data from a view

Must be in the same location

The life of a BigQuery SQL query



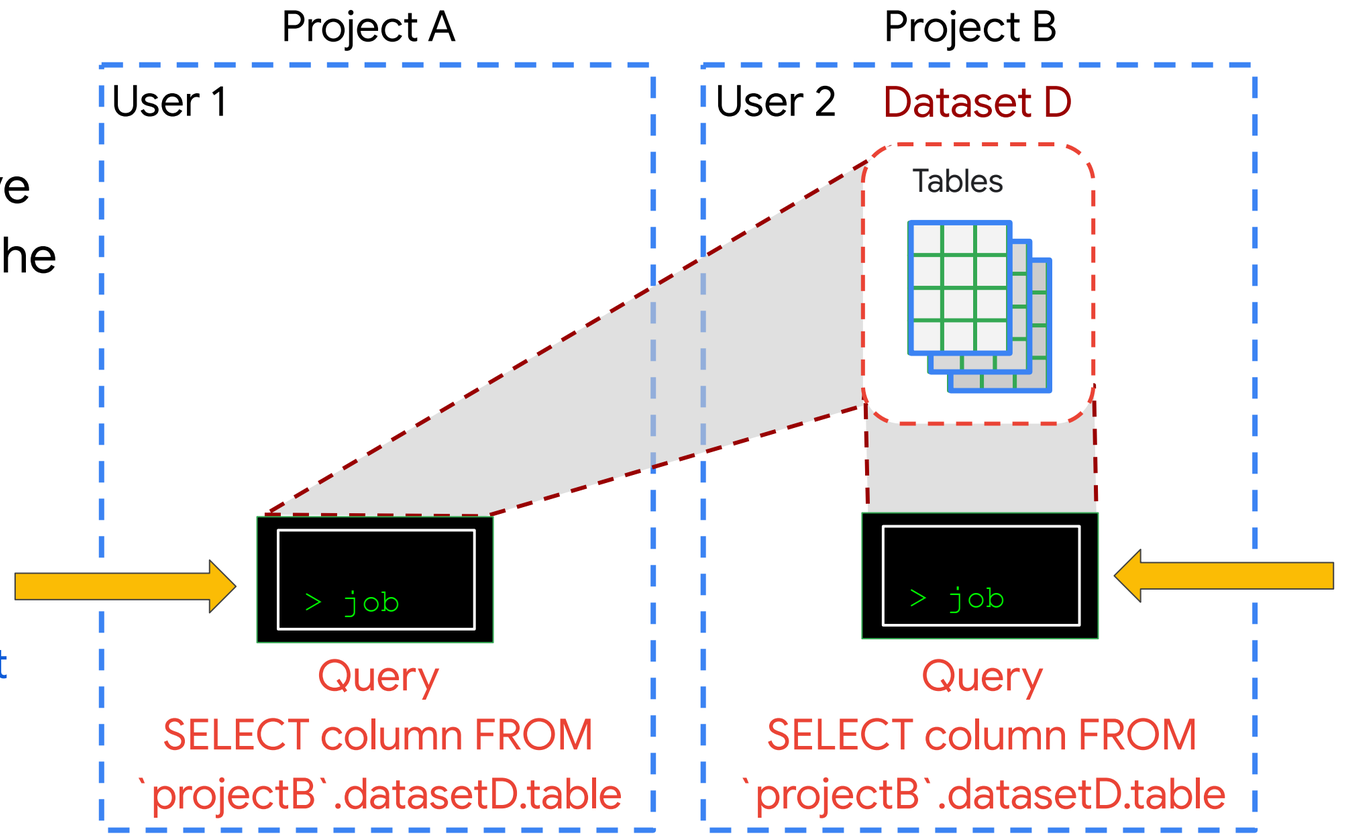
Use query validator with pricing calculator for estimates



1) Click on validator.
2) View data estimate.
3) Plug in here.

You can separate cost of storage and cost of queries

NOTE: The cost of a query is always assigned to the active project from where the query is executed



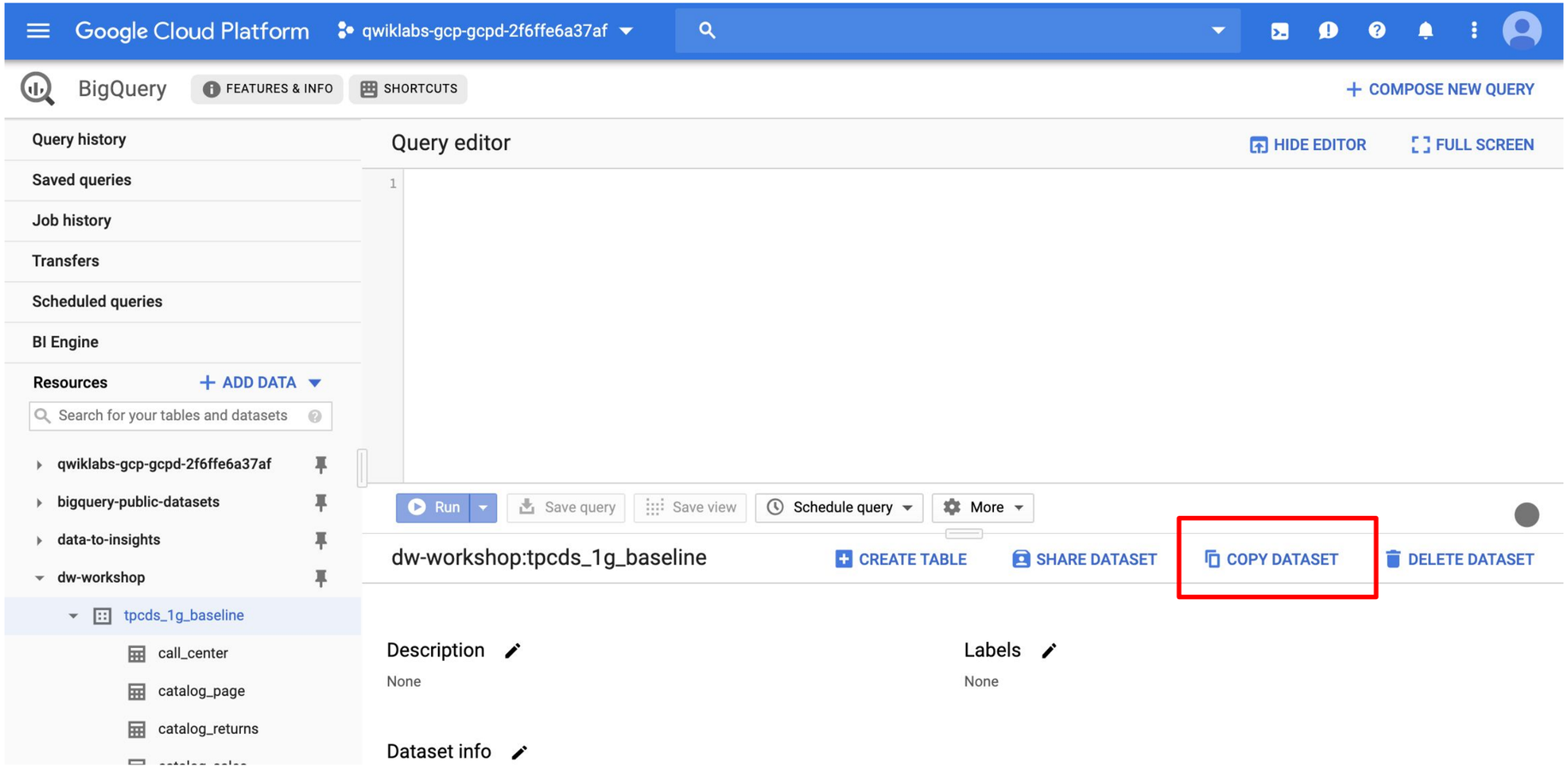
Project A pays for User 1's queries against data stored in Project B, Dataset D (assuming data access is shared)

Project B pays for storage and queries by User 2 executed from Project B.

Control costs with user and/or project quotas

Google Cloud

With the BigQuery Data Transfer Service, you can copy large datasets from different projects to yours in seconds



Google Cloud

Agenda

The modern data warehouse

Intro to BigQuery

- Getting Started
- Loading Data
- Lab: Loading Data with Console and CLI

Exploring Schemas

- Schema Design
- Nested and Repeated Fields
- Lab: ARRAYs and STRUCTs
- Optimizing with Partitioning and Clustering

Preview: Transforming Batch and Streaming Data

The method you use to load data depends on how much transformation is needed

EL	ELT	ETL
		
Extract and Load	Extract, Load, and Transform	Extract, Transform, and Load

Google Cloud

If the data is usable in its original form, just load it

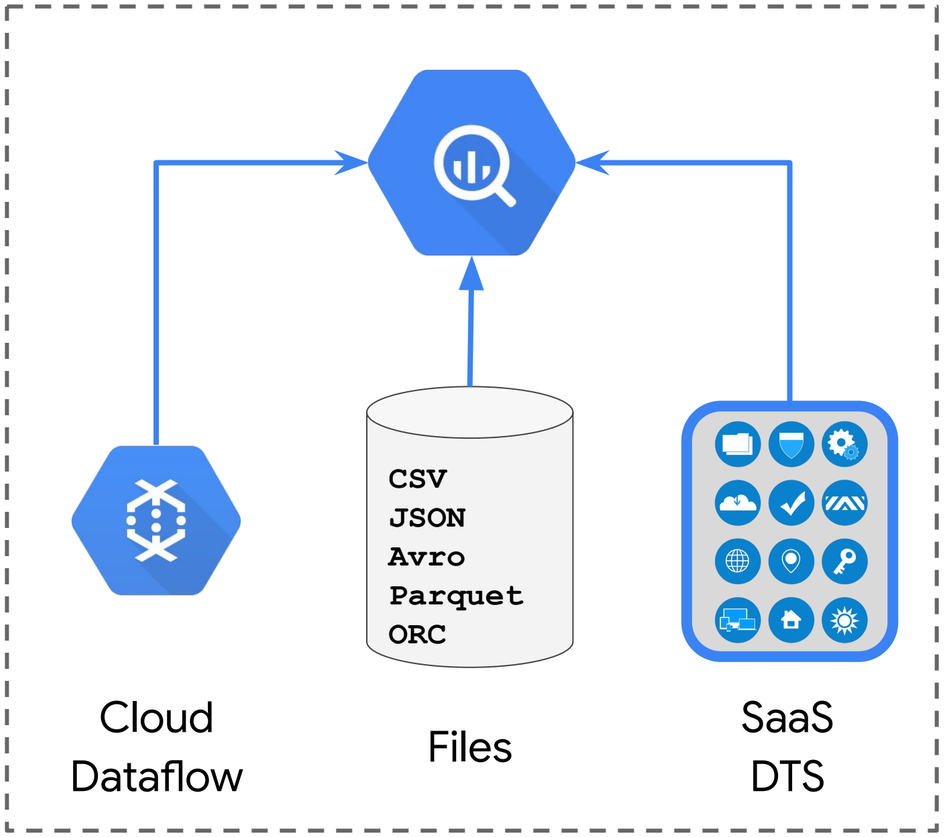
BigQuery Data Transfer Service (DTS)

BigQuery

Batch load supports different file formats

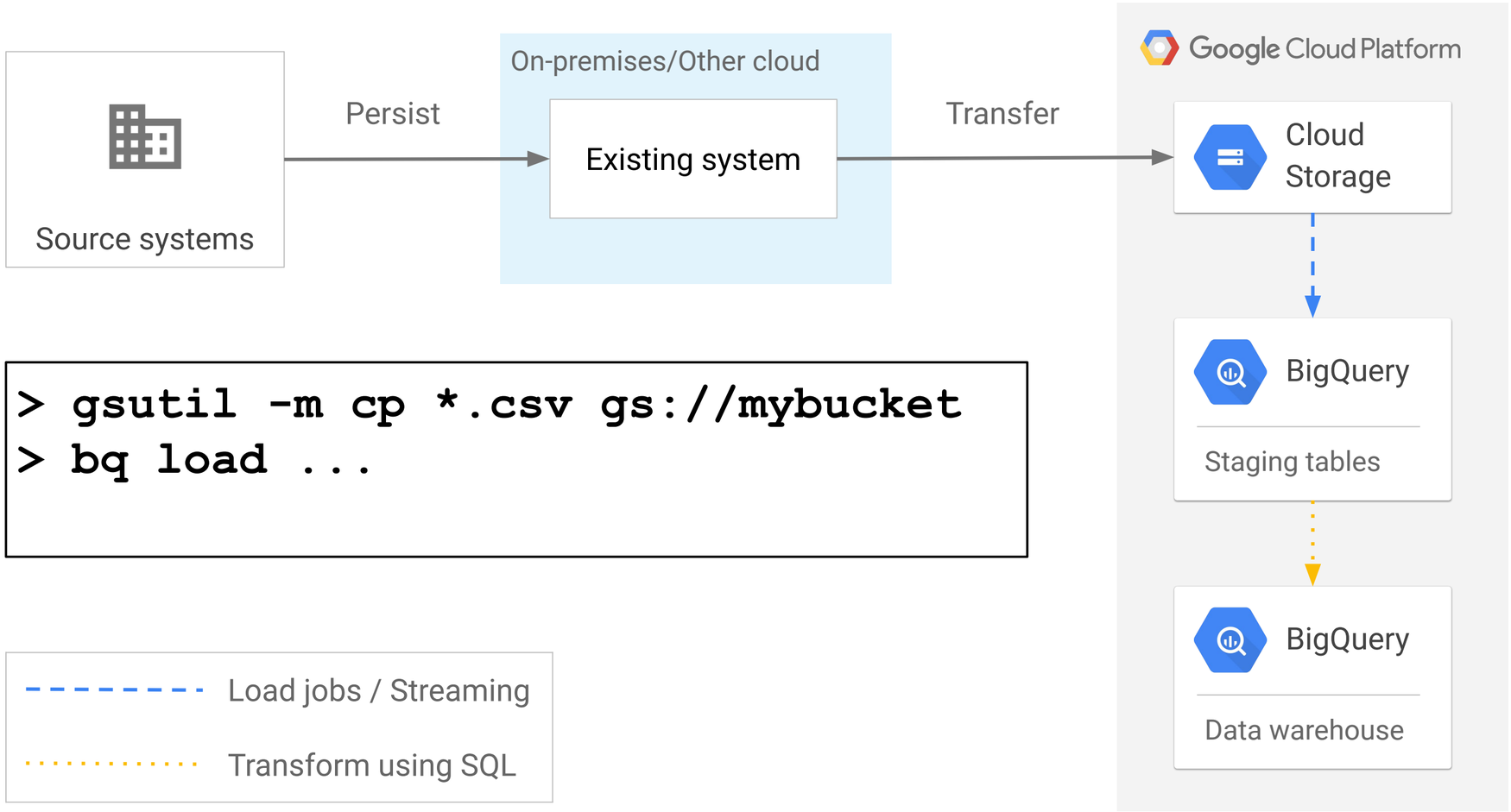
- CSV
- NEWLINE_DELIMITED_JSON
- AVRO
- DATASTORE_BACKUP
- PARQUET
- ORC

Most common is loading data into BigQuery tables (batch, periodic)

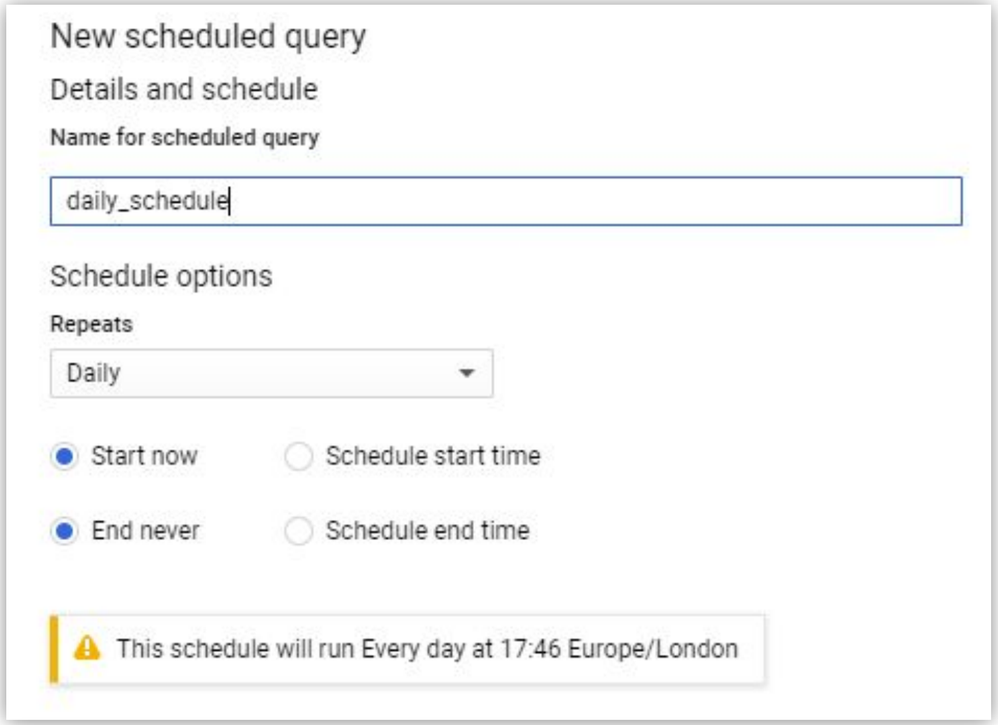


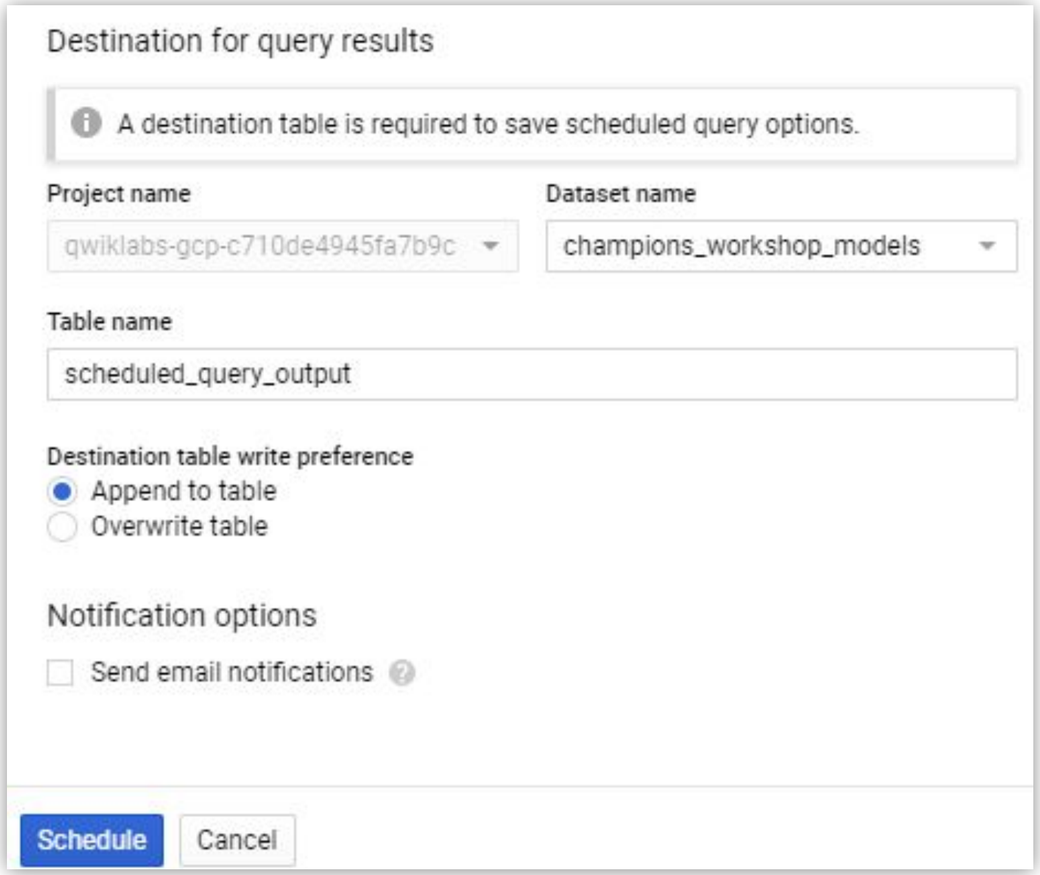
Loading data into BigQuery tables (batch, periodic) offers the best performance

Loading data through Cloud Storage



Automate the execution of queries based on a schedule





Google Cloud

BigQuery addresses backup and disaster recovery at the service level (time travel)

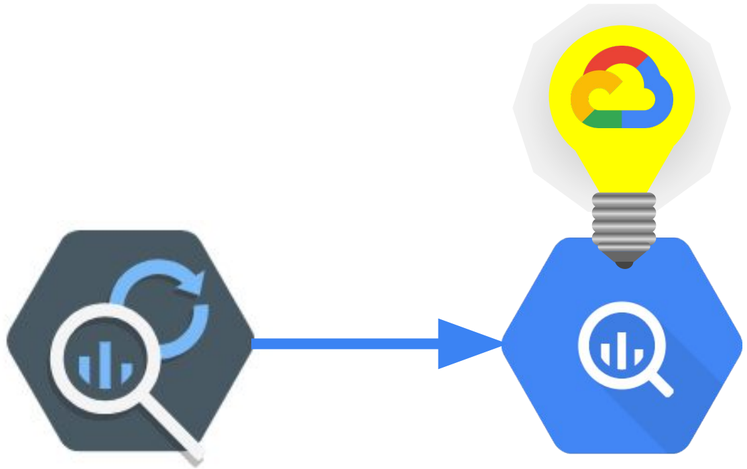
```
CREATE OR REPLACE TABLE ch10eu.restored_cycle_stations AS
SELECT
*
FROM `bigquery-public-data`.london_bicycles.cycle_stations
FOR SYSTEM_TIME AS OF
TIMESTAMP_SUB(CURRENT_TIMESTAMP(), INTERVAL 24 HOUR)
```

Query a point-in-time snapshot (up to 7 days)
Use it to create a backup

```
NOW=$(date +%s)
SNAPSHOT=$(echo "($NOW - 120)*1000" | bc)
bq --location=EU cp \
ch10eu.restored_cycle_stations@$SNAPSHOT \
ch10eu.restored_table
```

Restore a 2-min old copy
(Deleted tables are flushed after 2 days or if recreated with same name)

BigQuery Data Transfer Service helps you build and manage your data warehouse



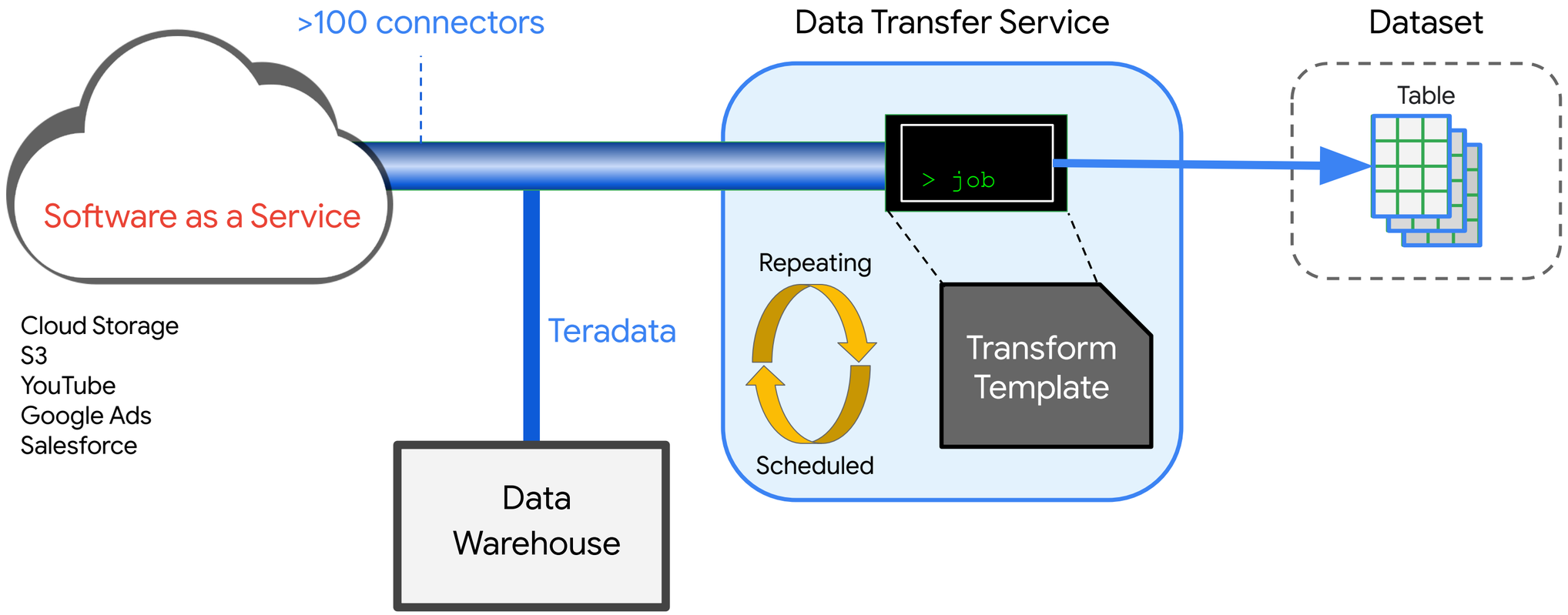
BigQuery Data Transfer Service (DTS)

Cloud Data Warehouse (DW)

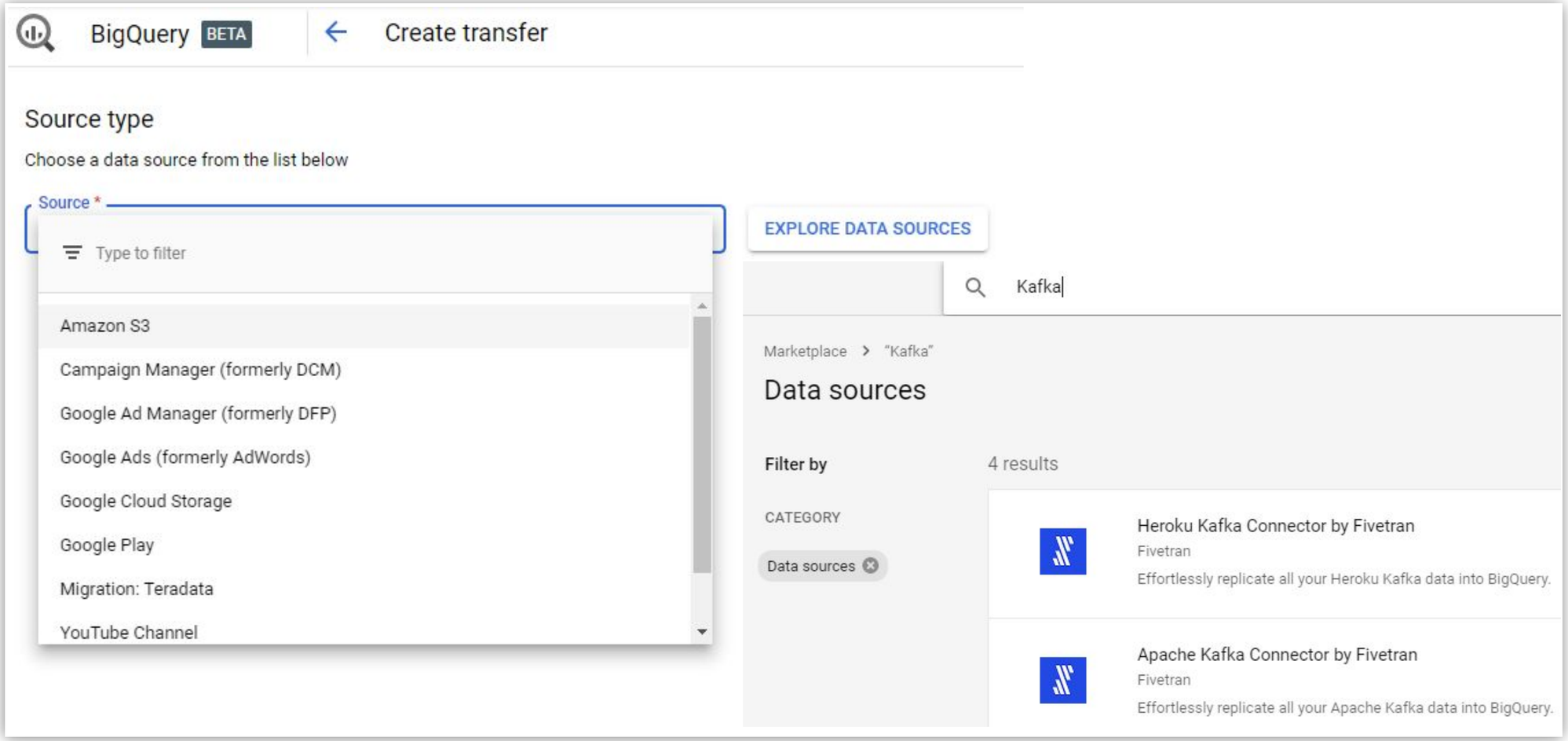
EL

- Managed service
- Automatic transfers
- Scheduled
- Data staging
- Data processing
- Data backfills

Data Transfer Service provides SaaS connectors

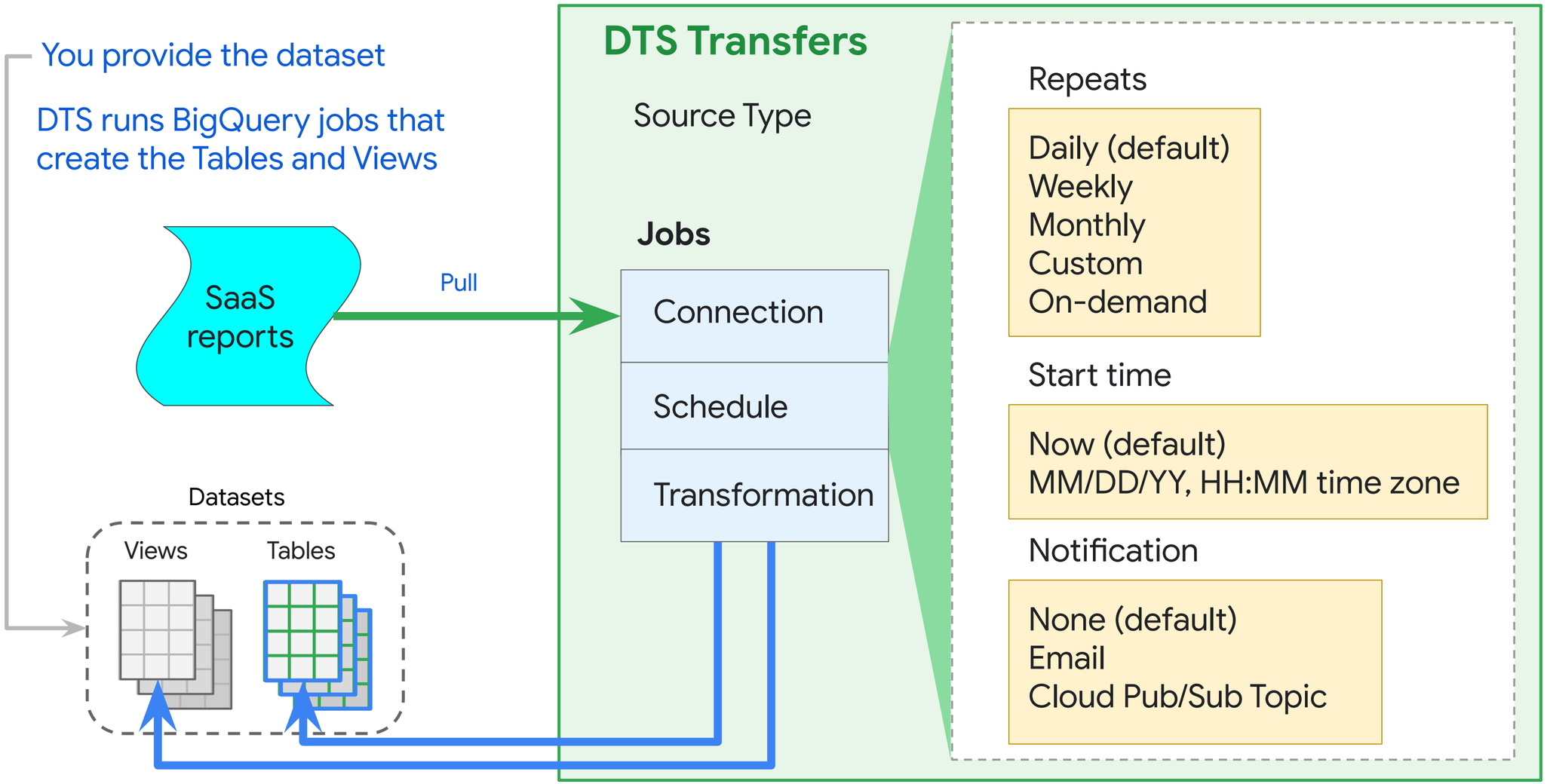


BigQuery DTS supports 100+ SaaS applications



Google Cloud

How DTS transfers work



If the data requires simple transformations, such as scaling, maybe it can be handled in SQL

BigQuery

Modify table data with standard DML statements

INSERT, UPDATE, DELETE, MERGE records into tables

```
UPDATE table_A
SET
  y = table_B.y,
  z = table_B.z + 1
FROM table_B
WHERE table_A.x = table_B.x
  AND table_A.y IS NULL;
```

```
INSERT INTO table VALUES (1,2,3), (4,5,6), (7,8,9);
```

```
DELETE FROM table WHERE TRUE;
```

Create new tables from data with SQL DDL

```
SELECT
   *
FROM
   movielens.movies_raw
WHERE
   movieId < 5;
```

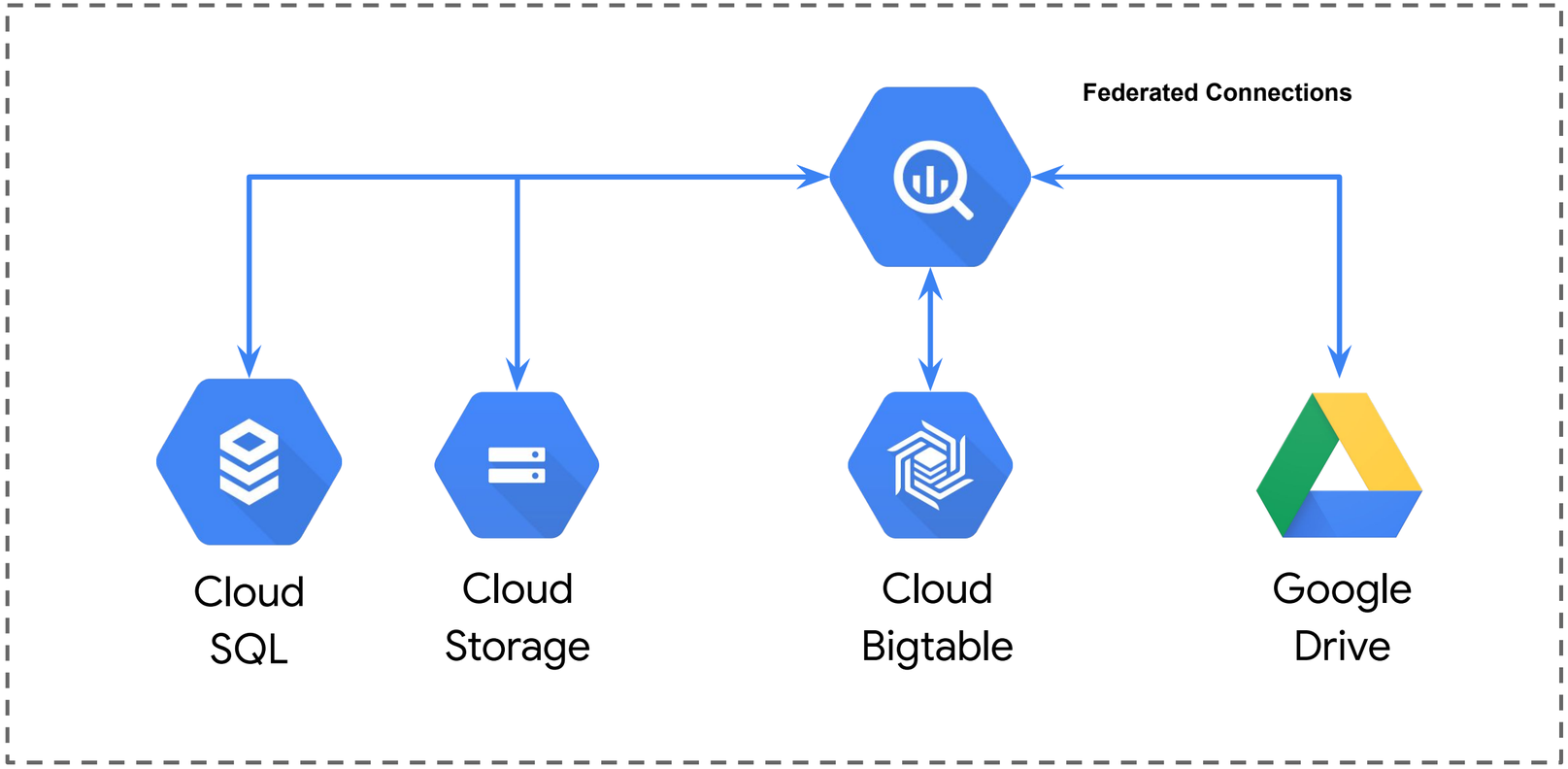
	movieId	title	genres
0	3	Grumpier Old Men (1995)	Comedy|Romance
1	4	Waiting to Exhale (1995)	Comedy|Drama|Romance
2	2	Jumanji (1995)	Adventure|Children|Fantasy
3	1	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy

```
CREATE OR REPLACE TABLE
movielens.movies AS
SELECT
   * REPLACE(SPLIT(genres, "|")
AS genres)
FROM
   Movielens.movies_raw;
-- Execute multiple statements.
SELECT * FROM
movielens.movies;
```

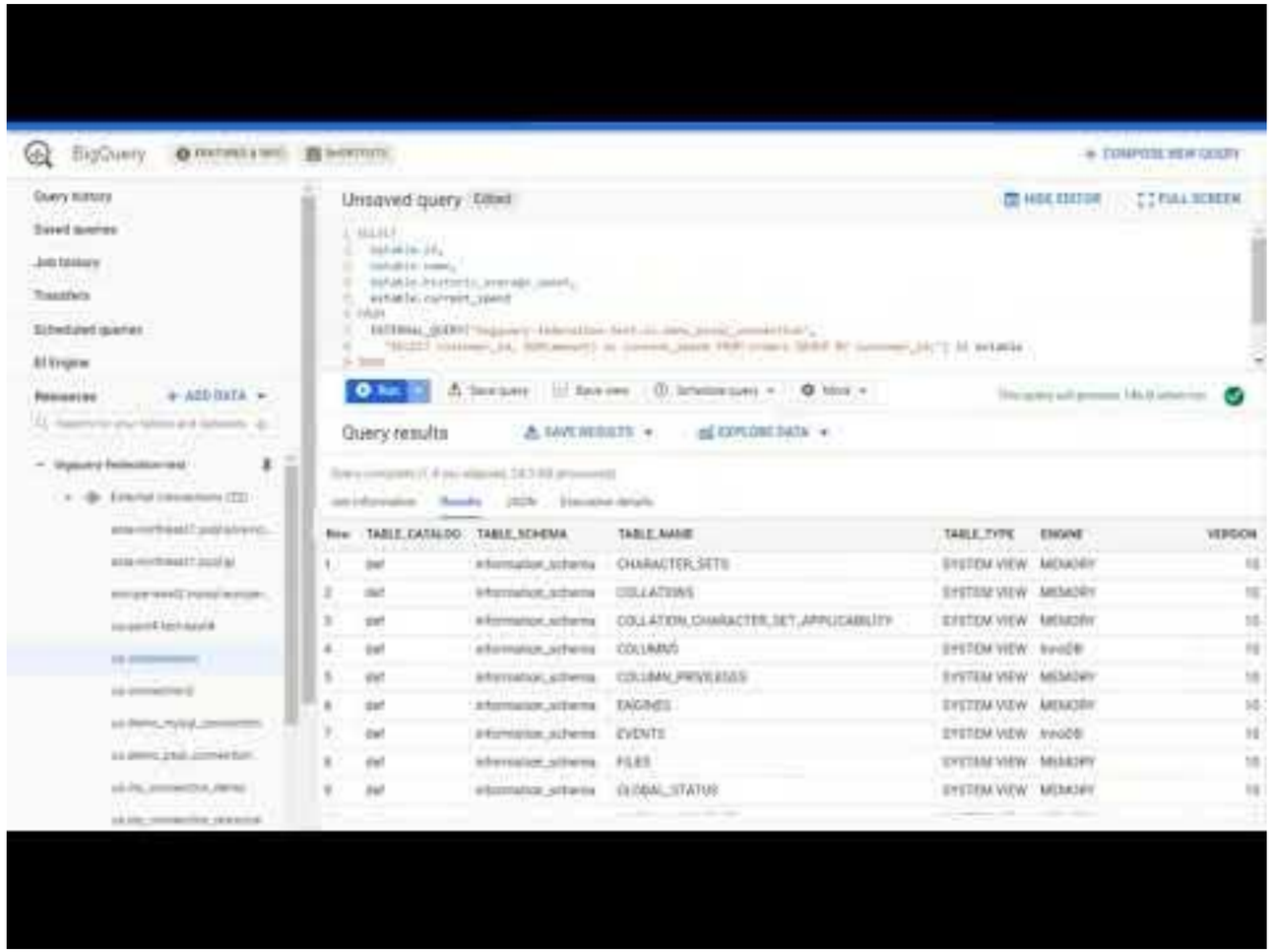
	movieId	title	genres
0	4	Waiting to Exhale (1995)	[Comedy, Drama, Romance]
1	3	Grumpier Old Men (1995)	[Comedy, Romance]
2	2	Jumanji (1995)	[Adventure, Children, Fantasy]
3	1	Toy Story (1995)	[Adventure, Animation, Children, Comedy, Fantasy]

Question: What's the difference between CREATE OR REPLACE TABLE and CREATE TABLE IF NOT EXISTS ? When would you use each?

Some data can be used in place without importing into BigQuery tables using external data sources

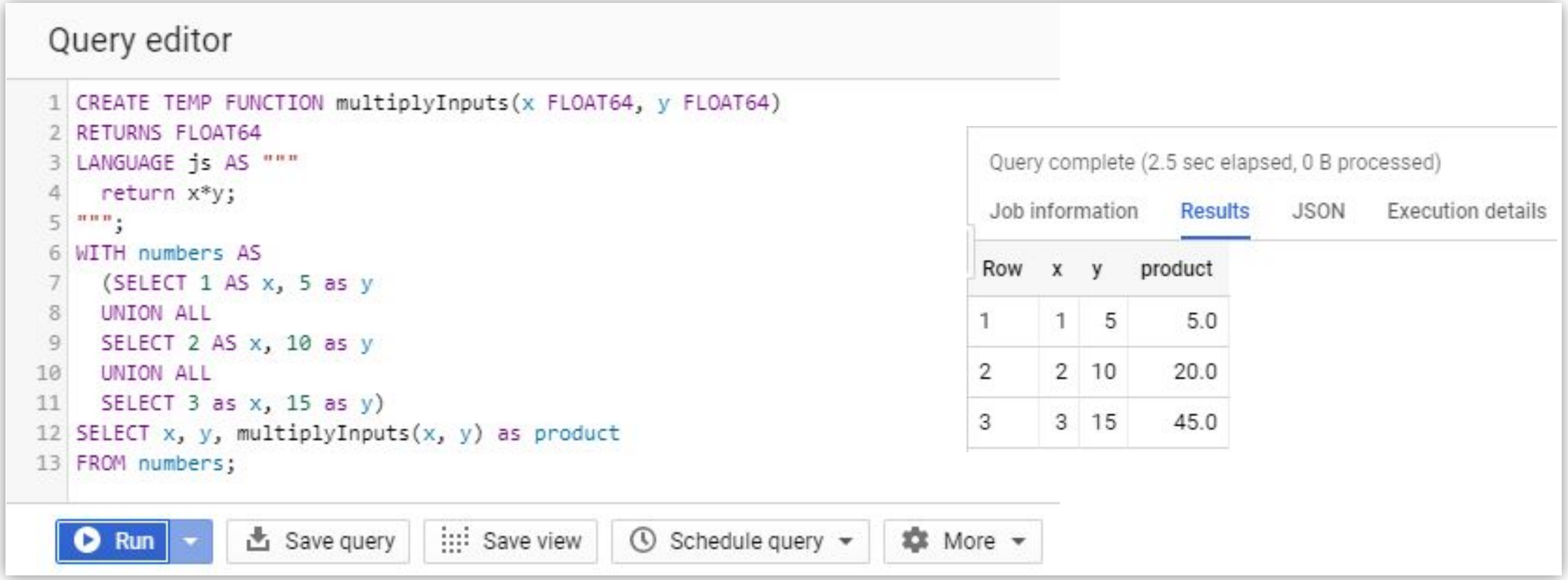


Video Demo: Querying Cloud SQL directly from BigQuery

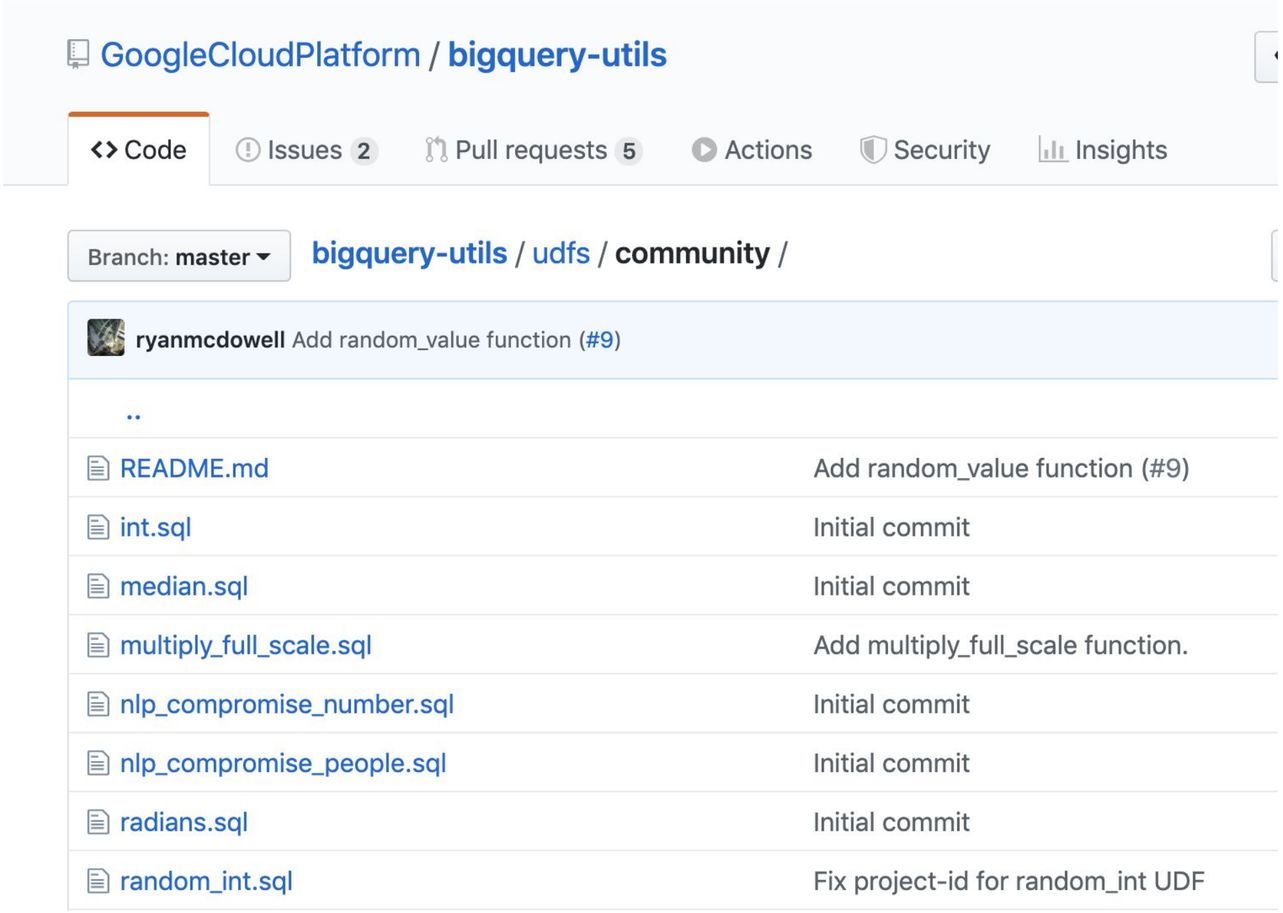


Google Cloud

Custom transformations? BigQuery supports user-defined functions in SQL, JavaScript, and scripting



You can persist and share your UDF objects with other team members or publically



The BigQuery team has a public GitHub repo for common User Defined Functions https://github.com/GoogleCloudPlatform/bigquery-utils/tree/master/udfs/community

Agenda

The modern data warehouse

Intro to BigQuery

- Getting Started
- Loading Data
- Lab: Loading Data with Console and CLI

Exploring Schemas

- Schema Design
- Nested and Repeated Fields
- Lab: ARRAYs and STRUCTs
- Optimizing with Partitioning and Clustering

Preview: Transforming Batch and Streaming Data

Lab

Loading data into BigQuery

Objectives

- Loading data into BigQuery from various sources
- Loading data into BigQuery using the CLI and Console
- Using DDL to create tables

Agenda

The modern data warehouse

Intro to BigQuery

- Getting Started
- Loading Data
- Lab: Loading Data with Console and CLI

Exploring Schemas

- Schema Design
- Nested and Repeated Fields
- Lab: ARRAYs and STRUCTs
- Optimizing with Partitioning and Clustering

Preview: Transforming Batch and Streaming Data

Designing schemas that scale is a core job of data engineers -- let's explore BigQuery Public Dataset schemas

Public datasets include flights, taxi cab logs, weather recordings, and many more

https://cloud.google.com/bigquery/public-data/

Demo

Exploring BigQuery Public Datasets with SQL using INFORMATION_SCHEMA

Agenda

The modern data warehouse

Intro to BigQuery

- Getting Started
- Loading Data
- Lab: Loading Data with Console and CLI

Exploring Schemas

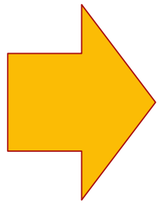
- Schema Design
- Nested and Repeated Fields
- Lab: ARRAYs and STRUCTs
- Optimizing with Partitioning and Clustering

Preview: Transforming Batch and Streaming Data

Transactional databases often use normal form

Original data

Customer	OrderID	Date	Items	
Doug	1600p	8/20/19		
			Product	**Quantity**
			Caulk	3 boxes
			Soffit	34 meters
			Sealant	2 liters
Tom	221b	10/29/19		
			Product	**Quantity**
			Sealant	1 liter
			Soffit	17 meters
			Caulk	4 tubes



Normalized data

Orders

Date	OrderID
8/20/2019	1600p
10/29/2018	221b

Order_Items

OrderID	Product	Quantity
1600p	Caulk	3 boxes
221b	Sealant	1 liter
1600p	Soffit	34 meters
221b	Soffit	17 meters
221b	Caulk	4 tubes
1600p	Sealant	2 liters

Data warehouses often denormalize

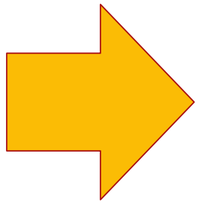
Normalized data

Orders

Date	OrderID
08/20/2019	1600p
10/29/2018	221b

Order_Items

OrderID	Product	Quantity
1600p	Caulk	3 boxes
221b	Sealant	1 liter
1600p	Soffit	34 meters
221b	Soffit	17 meters
221b	Caulk	4 tubes
1600p	Sealant	2 liters



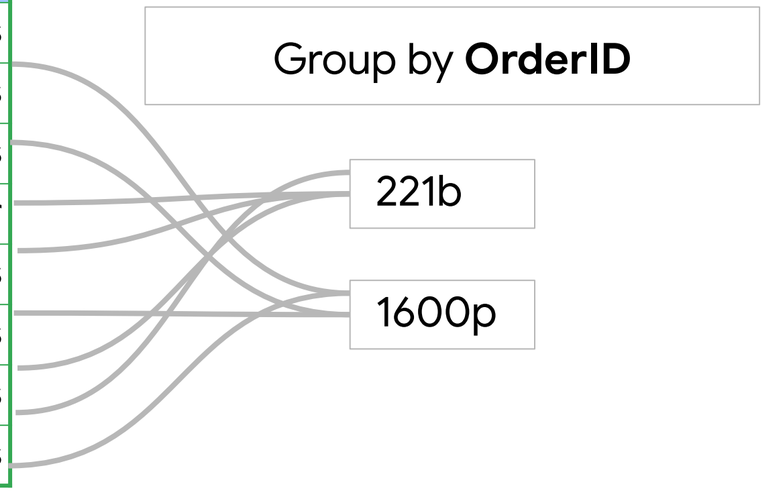
Denormalized flattened data

Customer	OrderID	Date	Product	Quantity
Doug	1600p	08/20/2019	Siding	3 boxes
Doug	1600p	08/20/2019	Caulk	12 tubes
Tom	221b	10/29/2019	Soffit	17 meters
Tom	221b	10/29/2019	Sealant	1 liter
Doug	1600p	08/20/2019	Soffit	34 meters
Tom	221b	10/29/2019	Siding	2 boxes
Tom	221b	10/29/2019	Caulk	4 tubes
Doug	1600p	08/20/2019	Sealant	2 liters

Grouping on a 1-to-many field in flattened data can cause shuffling of data over the network

Denormalized flattened table

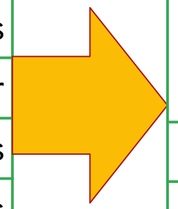
Customer	OrderID	Date	Product	Quantity
Doug	1600p	08/20/2019	Siding	3 boxes
Doug	1600p	08/20/2019	Caulk	12 tubes
Tom	221b	10/29/2019	Soffit	17 meters
Tom	221b	10/29/2019	Sealant	1 liter
Doug	1600p	09/09/2018	Soffit	34 meters
Tom	221b	10/29/2019	Siding	2 boxes
Tom	221b	10/29/2019	Caulk	4 tubes
Doug	1600p	08/20/2018	Sealant	2 liters



Nested and repeated columns improve the efficiency of BigQuery with relational source data

Denormalized flattened table

Customer	OrderID	Date	Product	Quantity
Doug	1600p	08/20/2019	Siding	3 boxes
Doug	1600p	08/20/2019	Caulk	12 tubes
Tom	221b	10/29/2019	Soffit	17 meters
Tom	221b	10/29/2019	Sealant	1 liter
Doug	1600p	8/20/2019	Soffit	34 meters
Tom	221b	10/29/2019	Siding	2 boxes
Tom	221b	10/29/2019	Caulk	4 tubes
Doug	1600p	08/20/2019	Sealant	2 liters



Denormalized with nested and repeated data

Order.ID	Order.Date	Order.Product	Order.Quantity
1600p	08/20/2019	Siding	3 boxes
		Caulk	12 tubes
		Soffit	34 meters
		Sealant	2 liters
221b	10/29/2019	Soffit	17 meters
		Sealant	1 liter
		Siding	2 boxes
		Caulk	4 tubes

Agenda

The modern data warehouse

Intro to BigQuery

- Getting Started
- Loading Data
- Lab: Loading Data with Console and CLI

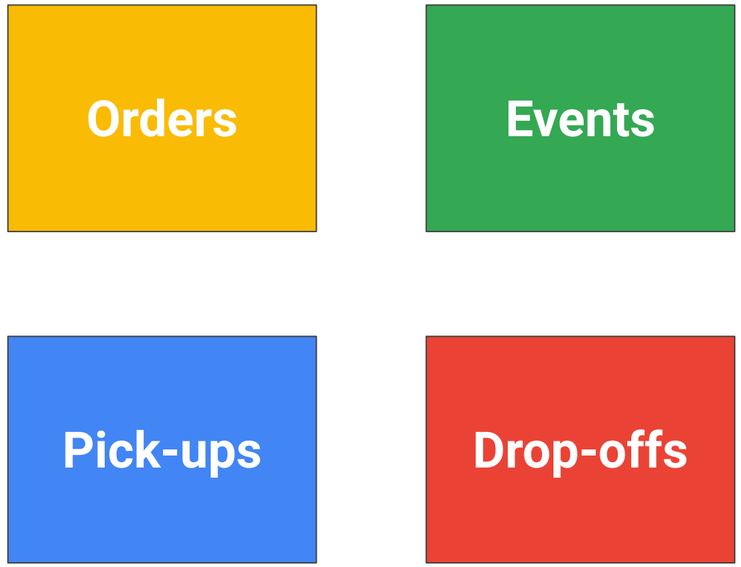
Exploring Schemas

- Schema Design
- Nested and Repeated Fields
- Lab: ARRAYs and STRUCTs
- Optimizing with Partitioning and Clustering

Preview: Transforming Batch and Streaming Data

GO-JEK is a ride booking service in Indonesia running on GCP

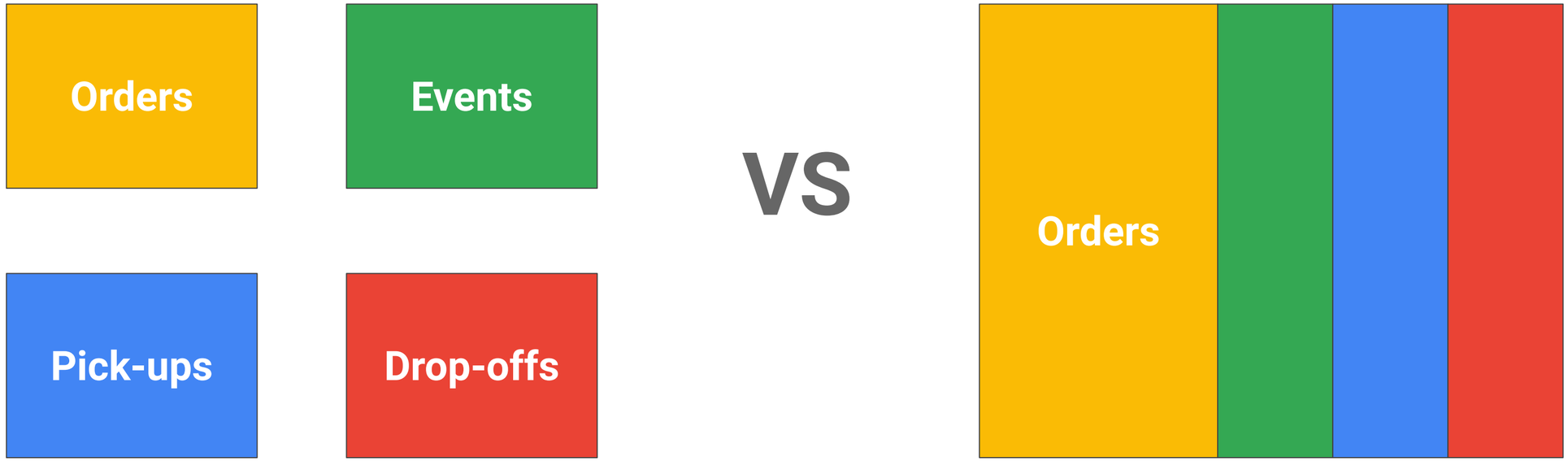
GO-JEK has 13+PB of data queried each month



- Each ride is stored as an order
- Each ride has a **single** pickup and drop-off
- Each ride can have **one-to-many** events:
  - Ride confirmed
  - Driver en route
  - Pick-up
  - Drop-off

How do you structure your data warehouse for scale?
Four separate and large tables that we join together?

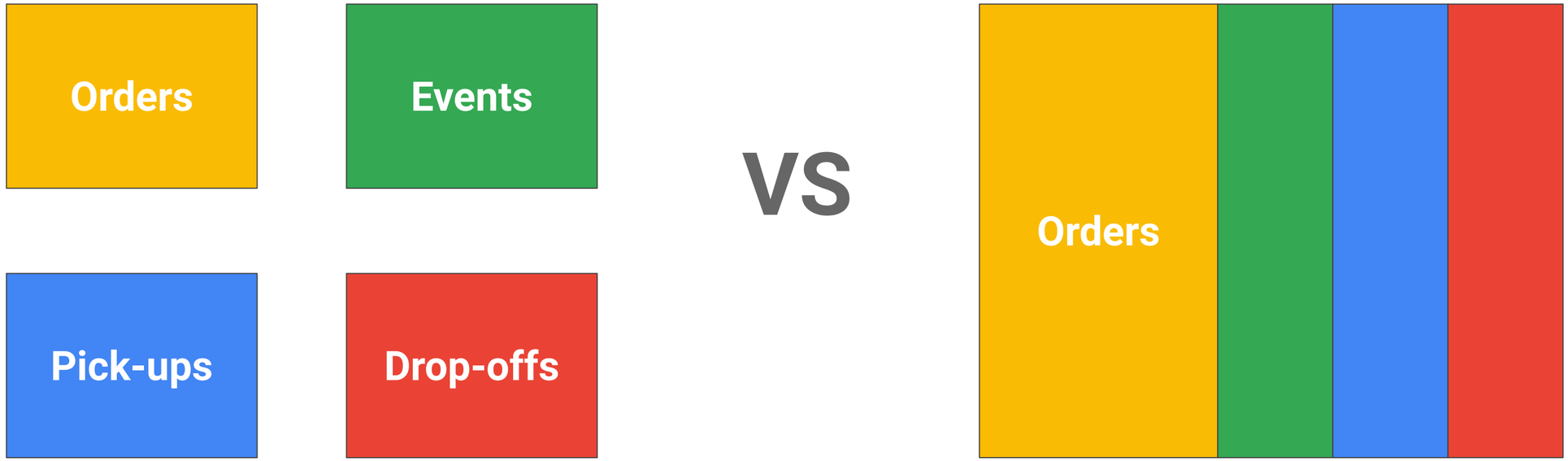
Reporting Approach: Should we normalize or denormalize?



Many tables

One big table

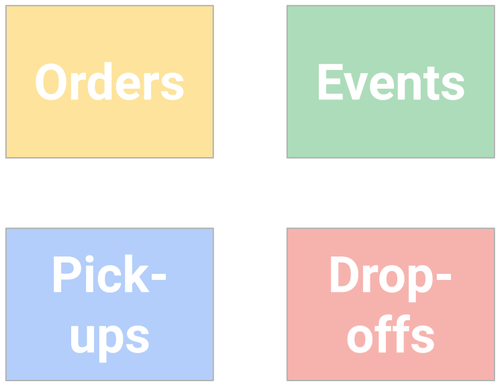
Reporting Approach: Should we normalize or denormalize?



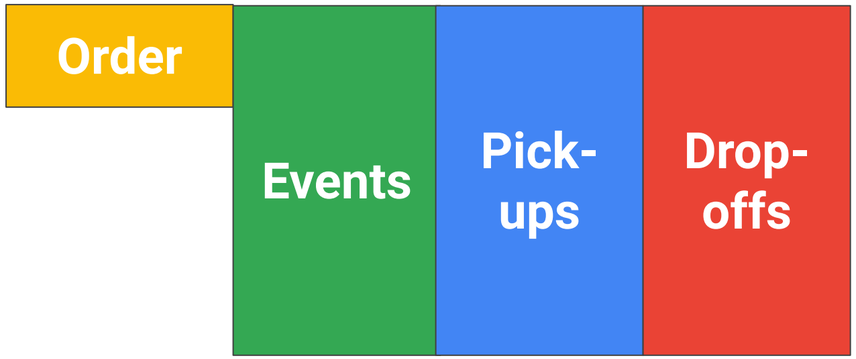
JOINs are costly

Data is repeated

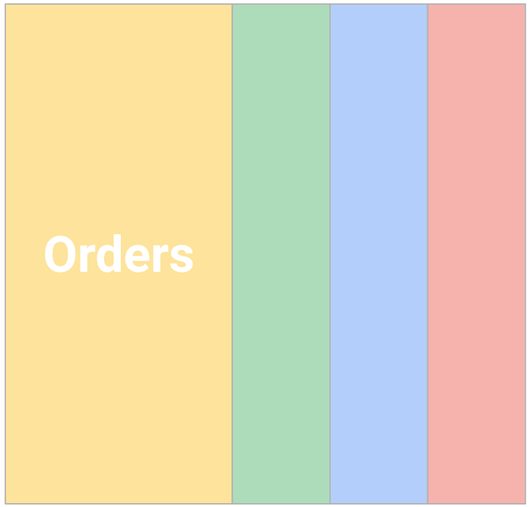
Nested and Repeated Fields allow you to have multiple levels of data granularity



JOINs are costly



Data is nested and repeated



Data is repeated

Store complex data with nested fields (ARRAYS)

Row	order_id	service_type	payment_method	event.status	event.time	pickup.latitude	pickup.longitude	destination.latitude	destination.longitude	total_distance_km	pricing_type
151	FD-5117	GO_FOOD	GOPAY	CREATED	2018-12-31 04:44:02.545210 UTC	-7.75105	110.410561	-7.7430367	110.4046433	1.56	regular
				COMPLETED	2018-12-31 05:06:27.897769 UTC						
				PICKED_UP	2018-12-31 04:48:25.945331 UTC						
				DRIVER_FOUND	2018-12-31 04:44:06.869010 UTC						
152	FD-6834	GO_FOOD	CASH	PICKED_UP	2018-12-31 12:49:52.518880 UTC	1.121272	104.049739	1.1368655	104.03322	4.84	surge
				DRIVER_FOUND	2018-12-31 12:40:14.214843 UTC						
				COMPLETED	2018-12-31 13:04:00.291780 UTC						
				CREATED	2018-12-31 12:40:13.431094 UTC						
153	FD-6293	GO_FOOD	PARTIAL_PAYMENT	PICKED_UP	2018-12-31 04:33:11.856445 UTC	-7.9657554	112.6247491	-7.9384084	112.6227862	4.68	regular
				COMPLETED	2018-12-31 04:56:05.885521 UTC						
				CREATED	2018-12-31 04:16:24.356539 UTC						
				DRIVER_FOUND	2018-12-31 04:16:25.643766 UTC						
154	FD-7817	GO_FOOD	CASH	COMPLETED	2018-12-31 09:14:44.897136 UTC	-6.353915	106.247312	-6.368896	106.25787	3.51	regular
				PICKED_UP	2018-12-31 09:01:11.471274 UTC						
				CREATED	2018-12-31 08:40:31.821796 UTC						
				DRIVER_FOUND	2018-12-31 08:40:32.910319 UTC						

Table JSON

First < Prev Rows 151 - 154 of 1137 Next > Last

Google Cloud

Report on all data in once place with STRUCTS

Row	order_id	service_type	payment_method	event.status	event.time	pickup.latitude	pickup.longitude	destination.latitude	destination.longitude	total_distance_km	pricing_type
151	FD-5117	GO_FOOD	GOPAY	CREATED	2018-12-31 04:44:02.545210 UTC	-7.75105	110.410561	-7.7430367	110.4046433	1.56	regular
				COMPLETED	2018-12-31 05:06:27.897769 UTC						
				PICKED_UP	2018-12-31 04:48:25.945331 UTC						
				DRIVER_FOUND	2018-12-31 04:44:06.869010 UTC						
152	FD-6834	GO_FOOD	CASH	PICKED_UP	2018-12-31 12:49:52.518880 UTC	1.121272	104.049739	1.1368655	104.03322	4.84	surge
				DRIVER_FOUND	2018-12-31 12:40:14.214843 UTC						
				COMPLETED	2018-12-31 13:04:00.291780 UTC						
				CREATED	2018-12-31 12:40:13.431094 UTC						
153	FD-6293	GO_FOOD	PARTIAL_PAYMENT	PICKED_UP	2018-12-31 04:33:11.856445 UTC	-7.9657554	112.6247491	-7.9384084	112.6227862	4.68	regular
				COMPLETED	2018-12-31 04:56:05.885521 UTC						
				CREATED	2018-12-31 04:16:24.356539 UTC						
				DRIVER_FOUND	2018-12-31 04:16:25.643766 UTC						
154	FD-7817	GO_FOOD	CASH	COMPLETED	2018-12-31 09:14:44.897136 UTC	-6.353915	106.247312	-6.368896	106.25787	3.51	regular
				PICKED_UP	2018-12-31 09:01:11.471274 UTC						
				CREATED	2018-12-31 08:40:31.821796 UTC						
				DRIVER_FOUND	2018-12-31 08:40:32.910319 UTC						

Table JSON

First < Prev Rows 151 - 154 of 1137 Next > Last

Google Cloud

Nested ARRAY fields and STRUCT fields allow for differing data granularity in the same table

Row	order_id	service_type	payment_method	event.status	event.time	pickup.latitude	pickup.longitude	destination.latitude	destination.longitude	total_distance_km	pricing_type
151	FD-5117	GO_FOOD	GOPAY	CREATED	2018-12-31 04:44:02.545210 UTC	-7.75105	110.410561	-7.7430367	110.4046433	1.56	regular
				COMPLETED	2018-12-31 05:06:27.897769 UTC						
				PICKED_UP	2018-12-31 04:48:25.945331 UTC						
				DRIVER_FOUND	2018-12-31 04:44:06.869010 UTC						
152	FD-6834	GO_FOOD	CASH	PICKED_UP	2018-12-31 12:49:52.518880 UTC	1.121272	104.049739	1.1368655	104.03322	4.84	surge
				DRIVER_FOUND	2018-12-31 12:40:14.214843 UTC						
				COMPLETED	2018-12-31 13:04:00.291780 UTC						
				CREATED	2018-12-31 12:40:13.431094 UTC						
153	FD-6293	GO_FOOD	PARTIAL_PAYMENT	PICKED_UP	2018-12-31 04:33:11.856445 UTC	-7.9657554	112.6247491	-7.9384084	112.6227862	4.68	regular
				COMPLETED	2018-12-31 04:56:05.885521 UTC						
				CREATED	2018-12-31 04:16:24.356539 UTC						
				DRIVER_FOUND	2018-12-31 04:16:25.643766 UTC						
154	FD-7817	GO_FOOD	CASH	COMPLETED	2018-12-31 09:14:44.897136 UTC	-6.353915	106.247312	-6.368896	106.25787	3.51	regular
				PICKED_UP	2018-12-31 09:01:11.471274 UTC						
				CREATED	2018-12-31 08:40:31.821796 UTC						
				DRIVER_FOUND	2018-12-31 08:40:32.910319 UTC						

Table JSON

First < Prev Rows 151 - 154 of 1137 Next > Last

Google Cloud

Your turn

- Practice reading the new schema
- Spot the STRUCTS
- Type RECORD = STRUCTS

booking

Schema | Details | Preview

Field name	Type	Mode
order_id	STRING	NULLABLE
service_type	STRING	NULLABLE
payment_method	STRING	NULLABLE
event	RECORD	REPEATED
event. **status**	STRING	NULLABLE
event. **time**	TIMESTAMP	NULLABLE
pickup	RECORD	NULLABLE
pickup. **latitude**	FLOAT	NULLABLE
pickup. **longitude**	FLOAT	NULLABLE
destination	RECORD	NULLABLE
destination. **latitude**	FLOAT	NULLABLE
destination. **longitude**	FLOAT	NULLABLE
total_distance_km	FLOAT	NULLABLE
pricing_type	STRING	NULLABLE
duration	RECORD	NULLABLE
duration. **booking_to_dispatch**	FLOAT	NULLABLE
duration. **booking_to_pickup**	FLOAT	NULLABLE

Practice reading the new schema

- Practice reading the new schema
- Spot the STRUCTS
- Type RECORD = STRUCTS

booking

Schema | Details | Preview

Field name	Type	Mode
order_id	STRING	NULLABLE
service_type	STRING	NULLABLE
payment_method	STRING	NULLABLE
event	RECORD	REPEATED
event. **status**	STRING	NULLABLE
event. **time**	TIMESTAMP	NULLABLE
pickup	RECORD	NULLABLE
pickup. **latitude**	FLOAT	NULLABLE
pickup. **longitude**	FLOAT	NULLABLE
destination	RECORD	NULLABLE
destination. **latitude**	FLOAT	NULLABLE
destination. **longitude**	FLOAT	NULLABLE
total_distance_km	FLOAT	NULLABLE
pricing_type	STRING	NULLABLE
duration	RECORD	NULLABLE
duration. **booking_to_dispatch**	FLOAT	NULLABLE
duration. **booking_to_pickup**	FLOAT	NULLABLE

Events

Pick-ups

Destination

Duration

Your turn

- Practice reading the new schema
- Spot the ARRAYS
- Hint: Look at Mode

booking

Schema | Details | Preview

Field name	Type	Mode
order_id	STRING	NULLABLE
service_type	STRING	NULLABLE
payment_method	STRING	NULLABLE
event	RECORD	REPEATED
event. **status**	STRING	NULLABLE
event. **time**	TIMESTAMP	NULLABLE
pickup	RECORD	NULLABLE

Practice reading the new schema

- Practice reading the new schema
- Spot the ARRAYS
- REPEATED = ARRAY

booking

Schema | Details | Preview

Field name	Type	Mode
order_id	STRING	NULLABLE
service_type	STRING	NULLABLE
payment_method	STRING	NULLABLE
event	RECORD	REPEATED
event. **status**	STRING	NULLABLE
event. **time**	TIMESTAMP	NULLABLE
pickup	RECORD	NULLABLE

Events

Row	order_id	service_type	payment_method	event.status	event.time
151	FD-5117	GO_FOOD	GOPAY	CREATED	2018-12-31 04:44:02.545210 UTC
				COMPLETED	2018-12-31 05:06:27.897769 UTC

Status and Time in an ARRAY of Event STRUCTs

Recap

- STRUCTS (RECORD)
- ARRAYS (REPEATED)
- ARRAYS can be part of regular fields or STRUCTS
- A single table can have many STRUCTS

booking

Schema | Details | Preview

Field name	Type	Mode
order_id	STRING	NULLABLE
service_type	STRING	NULLABLE
payment_method	STRING	NULLABLE
event	RECORD	REPEATED
event. **status**	STRING	NULLABLE
event. **time**	TIMESTAMP	NULLABLE
pickup	RECORD	NULLABLE
pickup. **latitude**	FLOAT	NULLABLE
pickup. **longitude**	FLOAT	NULLABLE
destination	RECORD	NULLABLE
destination. **latitude**	FLOAT	NULLABLE
destination. **longitude**	FLOAT	NULLABLE
total_distance_km	FLOAT	NULLABLE
pricing_type	STRING	NULLABLE
duration	RECORD	NULLABLE
duration. **booking_to_dispatch**	FLOAT	NULLABLE
duration. **booking_to_pickup**	FLOAT	NULLABLE

BigQuery stores repeated, nested fields in a columnar format

r_1

```
DocId: 10
Links
  Forward: 20
  Forward: 40
  Forward: 60
Name
  Language
    Code: 'en-us'
    Country: 'us'
  Language
    Code: 'en'
  Url: 'http://A'
Name
  Url: 'http://B'
Name
  Language
    Code: 'en-gb'
    Country: 'gb'
```

```
message Document {
  required int64 DocId;
  optional group Links {
    repeated int64 Backward;
    repeated int64 Forward; }
  repeated group Name {
    repeated group Language {
      required string Code;
      optional string Country; }
    optional string Url; }}
```

r_2

```
DocId: 20
Links
  Backward: 10
  Backward: 30
  Forward:  80
Name
  Url: 'http://C'
```

DocID

value	r	d
10	0	0
20	0	0

Name.Url

value	r	d
http://A	0	2
http://B	1	2
NULL	1	1
http://C	0	2

Links.Forward

value	r	d
20	0	2
40	1	2
60	1	2
80	0	2

Links.Backward

value	r	d
NULL	0	1
10	0	2
30	1	2

Name.Language.Code

value	r	d
en-us	0	2
en	2	2
NULL	1	1
en-gb	1	2
NULL	0	1

Name.Language.Country

value	r	d
us	0	3
NULL	2	2
NULL	1	1
gb	1	3
NULL	0	1

Dremel

Demo

Nested and repeated fields in BigQuery

General guidelines to design the optimal schema for BigQuery

Instead of joins, take advantage of nested and repeated fields in denormalized tables.

General guidelines to design the optimal schema for BigQuery

Instead of joins, take advantage of nested and repeated fields in denormalized tables.

Keep a dimension table smaller than 10 gigabytes normalized, unless the table rarely goes through `UPDATE` and `DELETE` operations.

General guidelines to design the optimal schema for BigQuery

Instead of joins, take advantage of nested and repeated fields in denormalized tables.

Keep a dimension table smaller than 10 gigabytes normalized, unless the table rarely goes through `UPDATE` and `DELETE` operations.

Denormalize a dimension table larger than 10 gigabytes, unless data manipulation or costs outweigh benefits of optimal queries.

Agenda

The modern data warehouse

Intro to BigQuery

- Getting Started
- Loading Data
- Lab: Loading Data with Console and CLI

Exploring Schemas

- Schema Design
- Nested and Repeated Fields
- Lab: ARRAYs and STRUCTs
- Optimizing with Partitioning and Clustering

Preview: Transforming Batch and Streaming Data

Lab

Working with JSON and Array Data in BigQuery

Objectives

- Load semi-structured JSON into BigQuery
- Create and query arrays
- Create and query structs
- Query nested and repeated fields

Agenda

The modern data warehouse

Intro to BigQuery

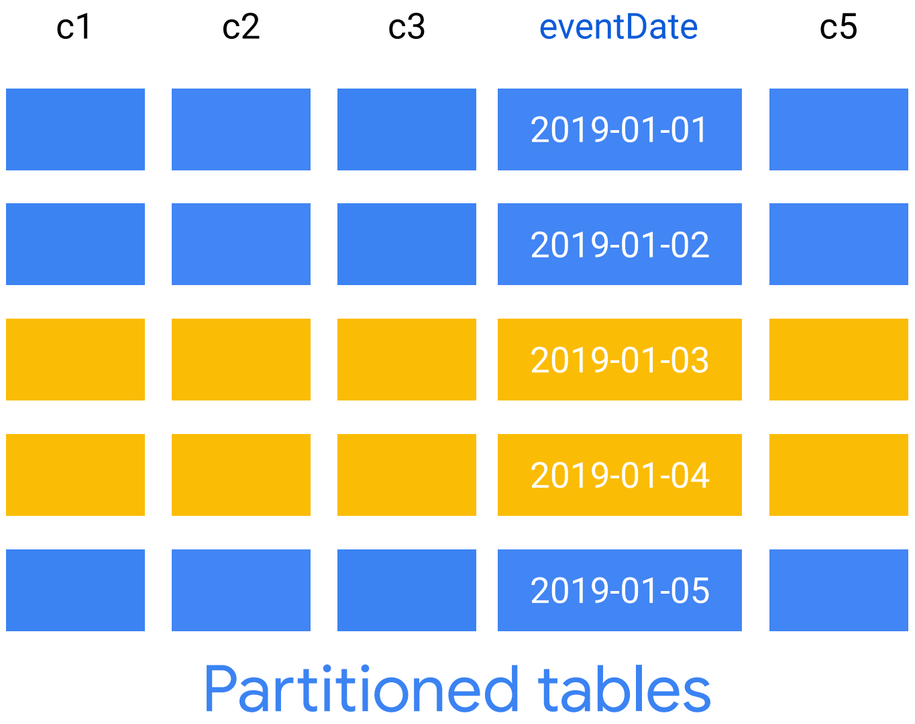
- Getting Started
- Loading Data
- Lab: Loading Data with Console and CLI

Exploring Schemas

- Schema Design
- Nested and Repeated Fields
- Lab: ARRAYs and STRUCTs
- Optimizing with Partitioning and Clustering

Preview: Transforming Batch and Streaming Data

Reduce cost and amount of data read by partitioning your tables




```
SELECT c1, c3 FROM ...
WHERE eventDate BETWEEN
“2019-01-03” AND “2019-01-04”
```

BigQuery supports three ways of partitioning tables

Ingestion time

```
bq query --destination_table mydataset.mytable
--time_partitioning_type=DAY
...
```

Any column that is of type DATE or TIMESTAMP

```
bq mk --table --schema a:STRING,tm:TIMESTAMP
--time_partitioning_field tm
```

Integer-typed column

```
bq mk --table --schema "customer_id:integer,value:integer"
--range_partitioning=customer_id,0,100,10 my_dataset.my_table
```

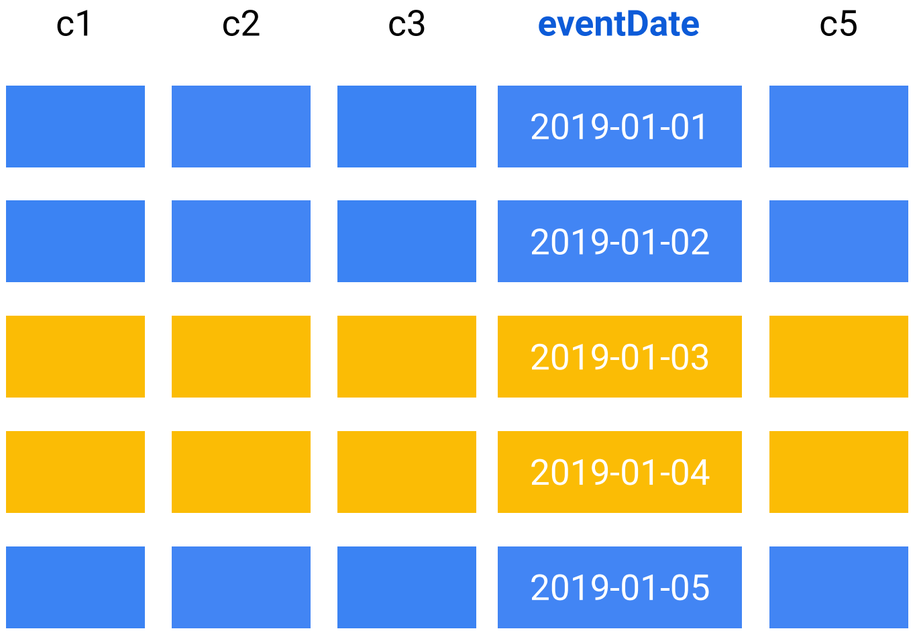
Partitioning can improve query cost and performance by reducing data being queried

```
SELECT
  field1
FROM
  mydataset.table1
WHERE
  _PARTITIONTIME > TIMESTAMP_SUB(TIMESTAMP('2016-04-15'), INTERVAL 5 DAY)
```

Isolate the partition field in the left-hand side of the query expression!

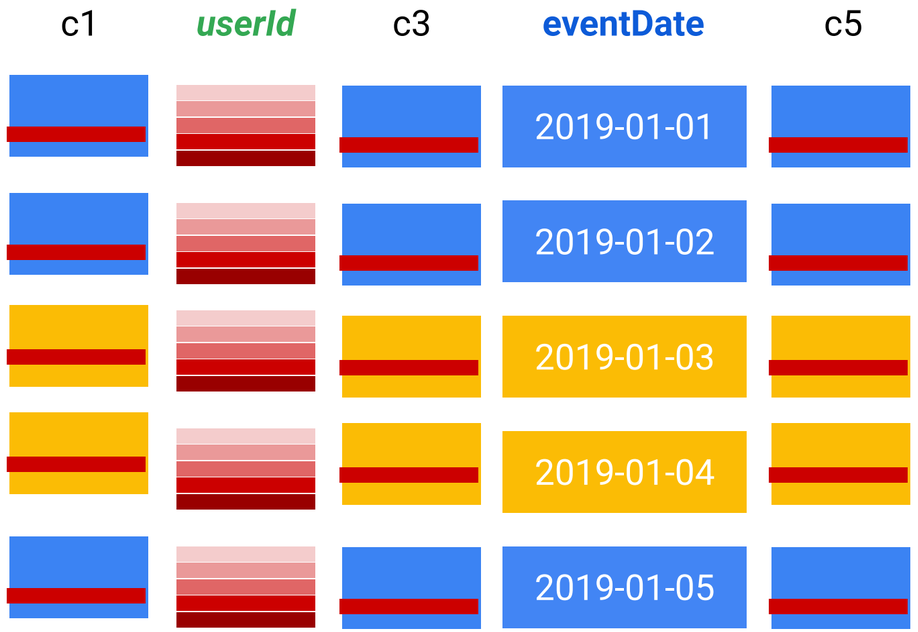
```
bq query \
--destination_table mydataset.mytable
--time_partitioning_type=DAY --require_partition_filter
...
```

BigQuery automatically sorts the data based on values in the clustering columns



SELECT c1, c3 FROM ...
WHERE **eventDate** BETWEEN “2019-01-03” AND “2019-01-04”

Partitioned tables

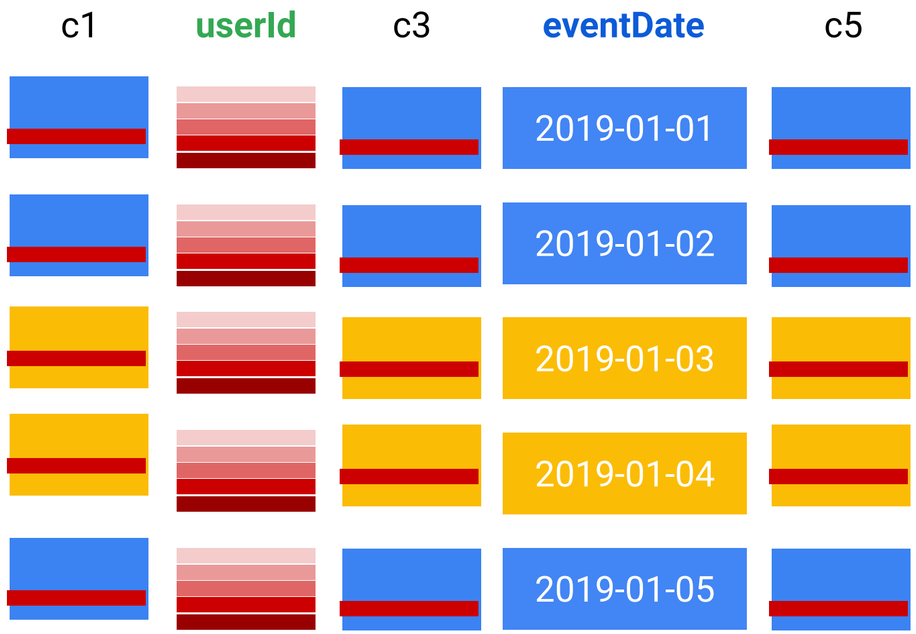


SELECT c1, c3 FROM ... WHERE ***userId*** BETWEEN 52 and 63 AND **eventDate** BETWEEN “2019-01-03” AND “2019-01-04”

Clustered tables

Google Cloud

Set up clustering at table creation time

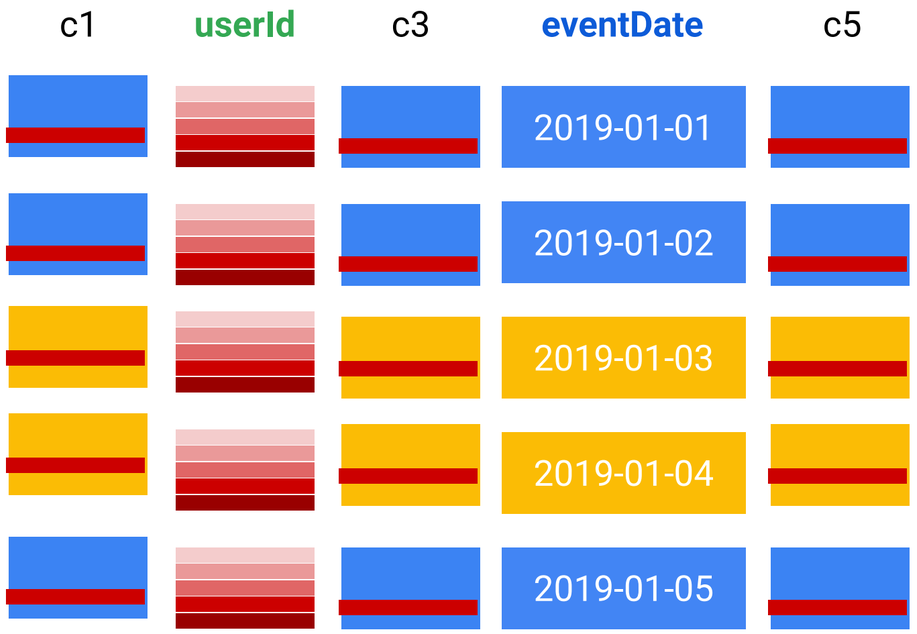



```
CREATE TABLE mydataset.myclusteredtable
 (
   c1 NUMERIC,
   userId STRING,
   c3 STRING,
   eventDate TIMESTAMP,
   C5 GEOGRAPHY
  )
 PARTITION BY DATE(eventDate)
 CLUSTER BY userId
 OPTIONS (
   partition_expiration_days=3,
   description="cluster")
 AS SELECT * FROM mydataset.myothertable
```

In streaming tables, the sorting fails over time, and so BigQuery has to recluster

```
UPDATE ds.table
SET c1 = 300
WHERE c1 = 300
AND eventDate > TIMESTAMP_SUB(CURRENT_TIMESTAMP(), INTERVAL 1 DAY)
```

Can force a recluster using DML on necessary partition



BigQuery will automatically recluster your data

Automatic re-clustering

free

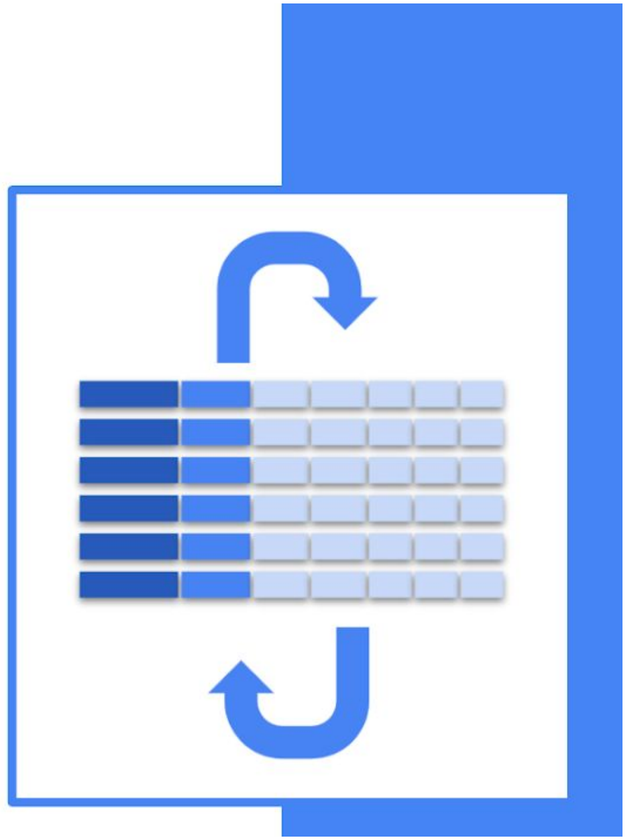
Doesn't consume your query resources

maintenance-free

Requires no setup or maintenance

autonomous

Automatically happens in the background



Organize data through managed tables

Partitioning

Filtering storage before query execution begins to reduce costs.

Reduces a full table scan to the partitions specified.

A single column results in 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).

Demo

Partitioned and Clustered Tables in BigQuery

When to use clustering

Your data is already partitioned on a `DATE` or `TIMESTAMP` or Integer Range

You commonly use filters or aggregation against particular columns in your queries.

Agenda

The modern data warehouse

Intro to BigQuery

- Getting Started
- Loading Data
- Lab: Loading Data with Console and CLI

Exploring Schemas

- Schema Design
- Nested and Repeated Fields
- Lab: ARRAYs and STRUCTs
- Optimizing with Partitioning and Clustering

Preview: Transforming Batch and Streaming Data

What if your data is not usable in its original form?

Extract, Transform,
and Load

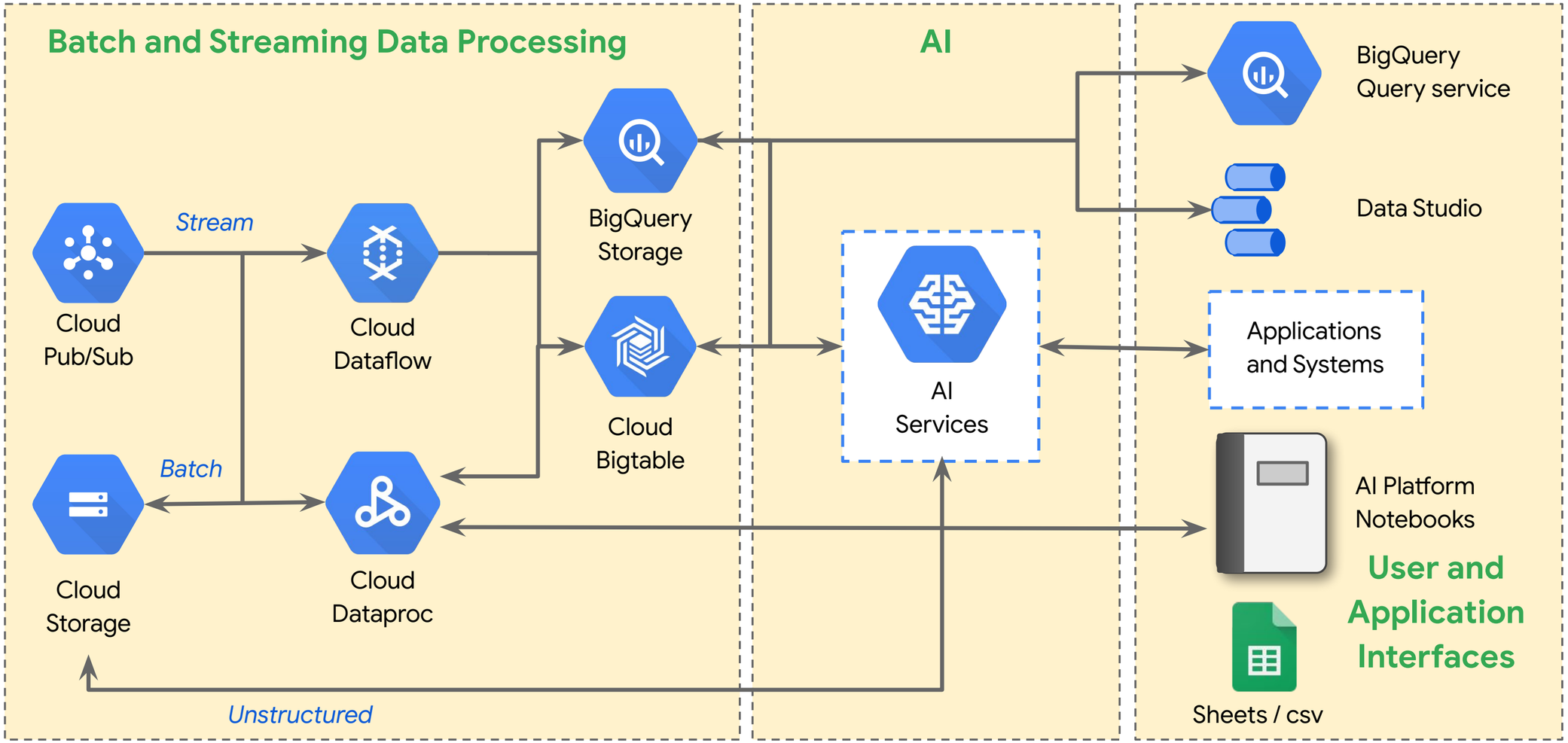
Data Processing

Cloud Dataproc

Cloud Dataflow

Data is processed in an intermediate system before it is loaded into the analytics warehouse.

A typical pipeline of GCP products that support ETL



Google Cloud

What if your data arrives continuously and endlessly?

Streaming Data Processing

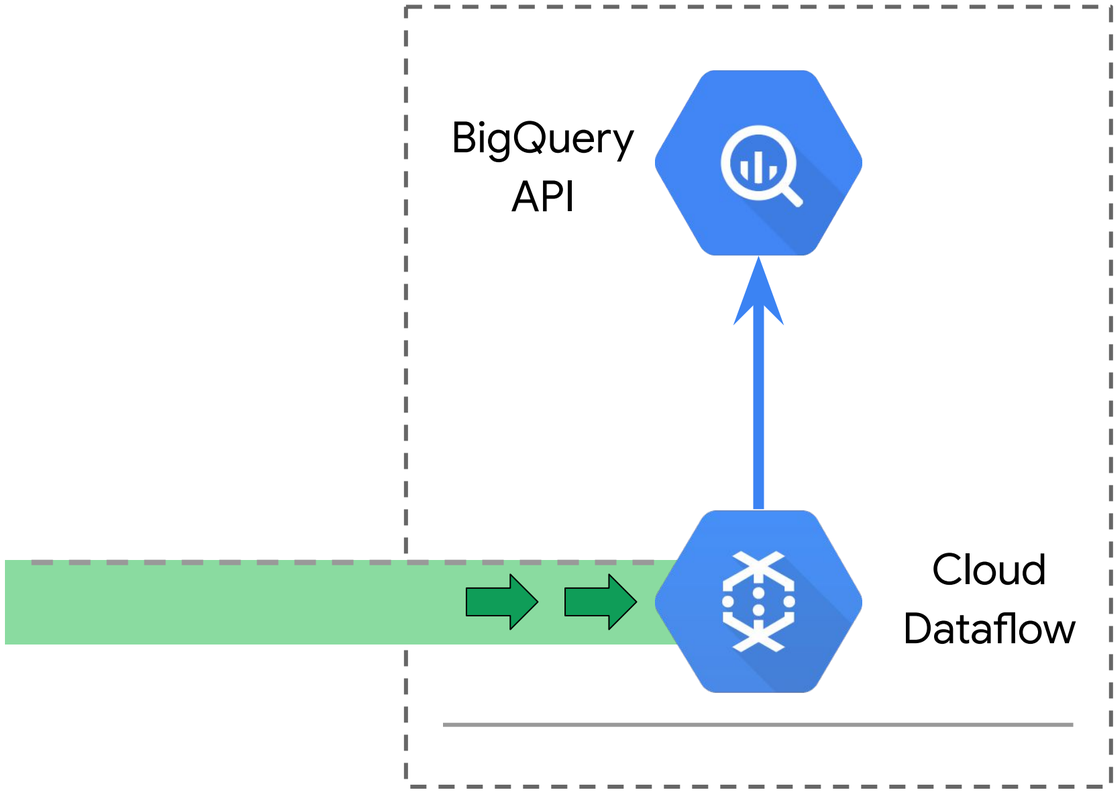
Cloud Pub/Sub

Cloud Dataflow

BigQuery

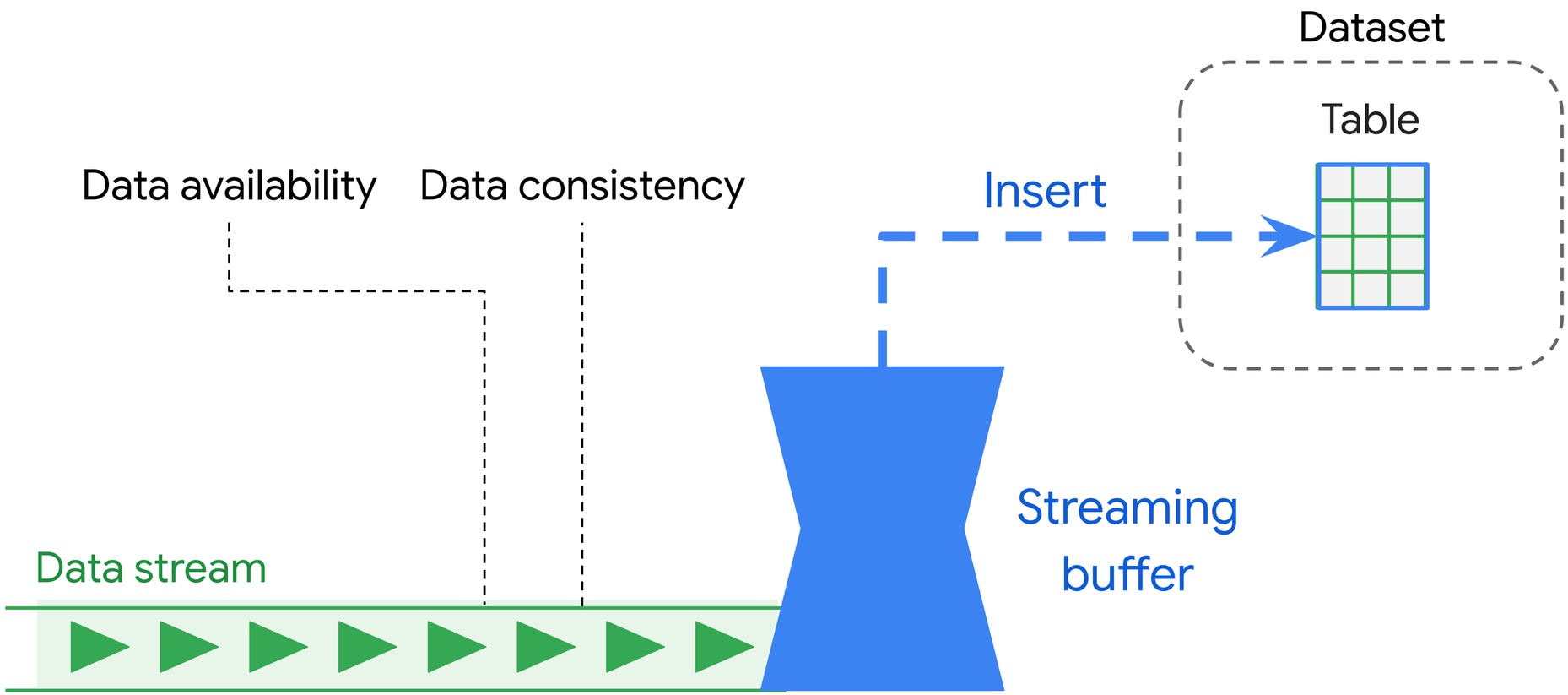
Data flows are buffered and processed rapidly in pipelines.

Some data is real-time and streamed into BigQuery by inserting records into tables via Cloud Dataflow



Insert records into tables from streams (unbounded) using a Dataflow pipeline

You can also use the BigQuery streaming API directly



Recap

The modern data warehouse

Intro to BigQuery

- Getting Started
- Loading Data
- Lab: Loading Data with Console and CLI

Exploring Schemas

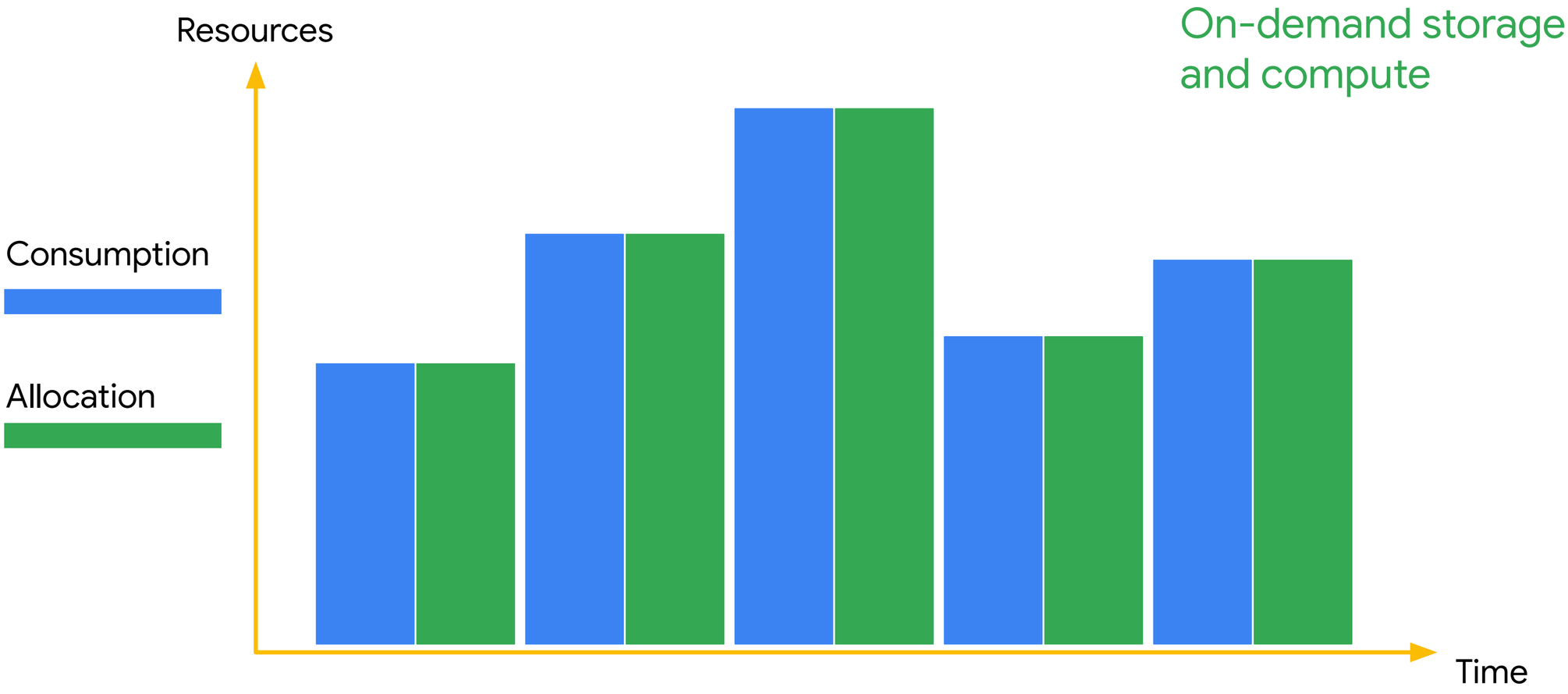
- Schema Design
- Nested and Repeated Fields
- Lab: ARRAYs and STRUCTs
- Optimizing with Partitioning and Clustering

Preview: Transforming Batch and Streaming Data

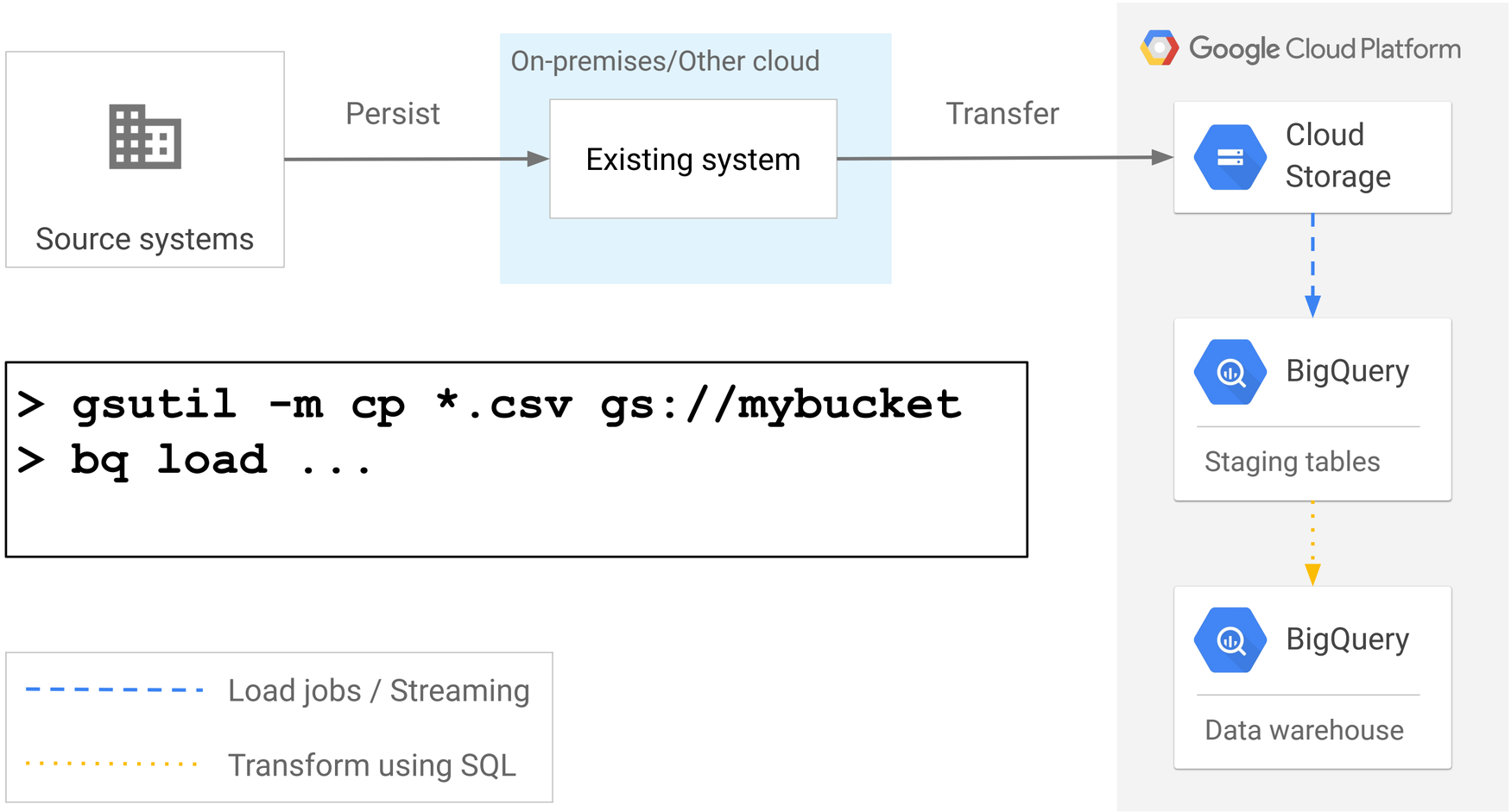
A modern data warehouse

- Gigabytes to petabytes
- Serverless and no-ops, including ad hoc queries
- Ecosystem of visualization and reporting tools
- Ecosystem of ETL and data processing tools
- Up-to-the-minute data
- Machine learning
- Security and collaboration

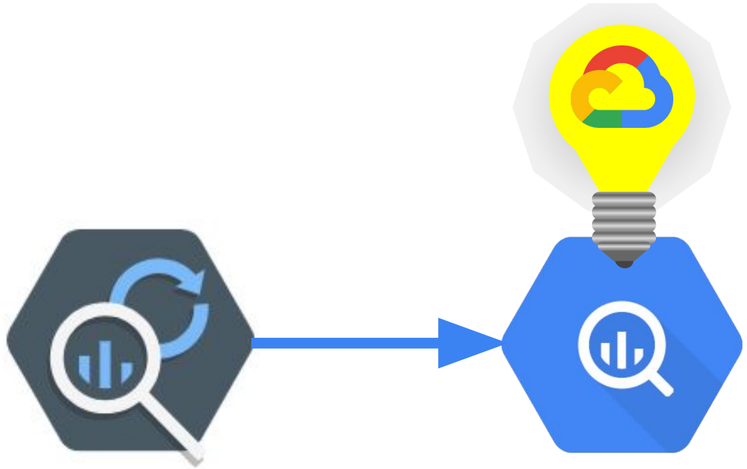
You don’t need to provision resources before using BigQuery



Loading data through Cloud Storage



BigQuery Data Transfer Service helps you build and manage your data warehouse



BigQuery Data Transfer Service (DTS)

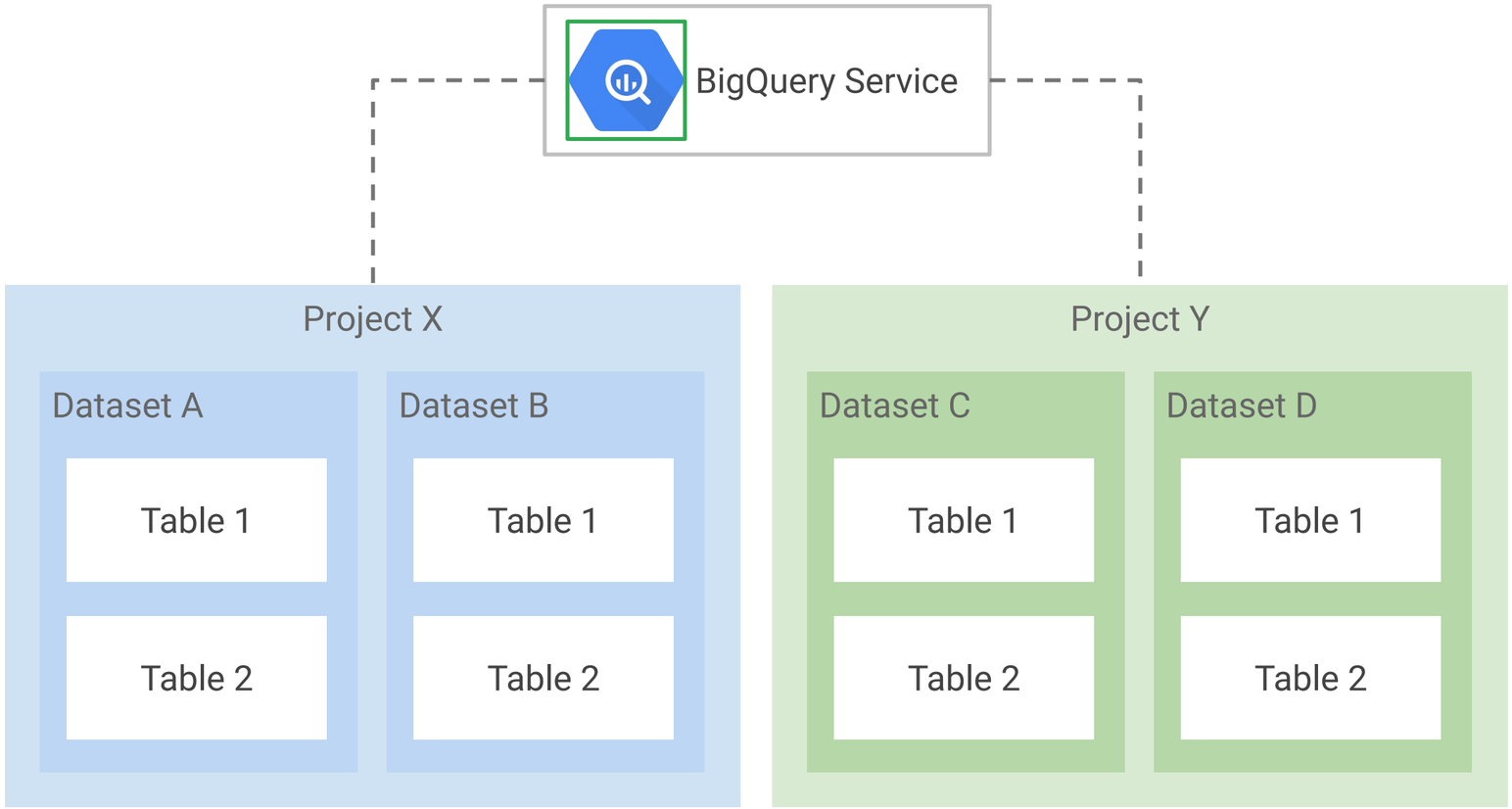
Cloud Data Warehouse (DW)

EL

- Managed service
- Automatic transfers
- Scheduled
- Data staging
- Data processing
- Data backfills

BigQuery organizes data tables into units called datasets

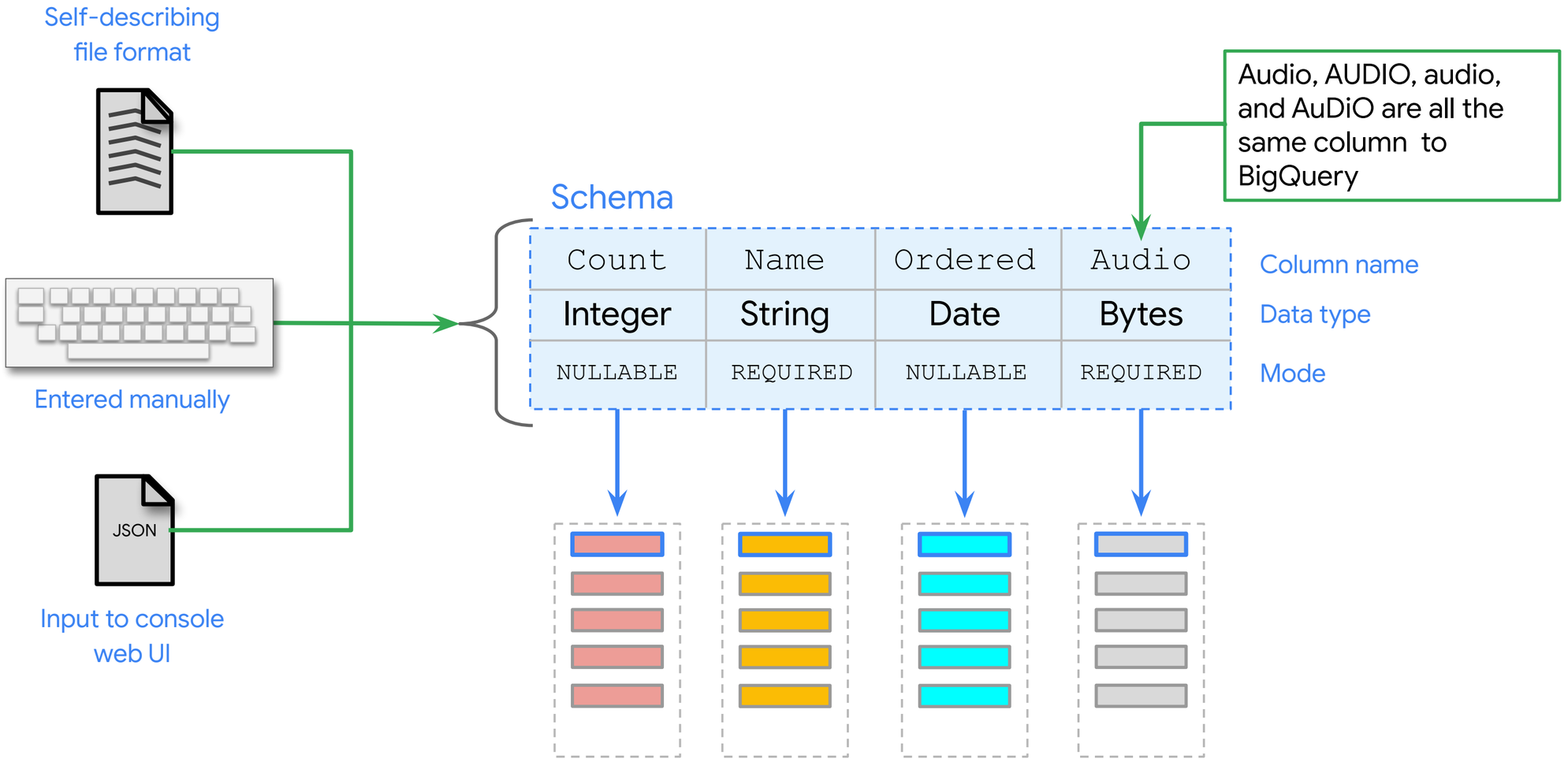
`project.dataset.table`



What are some reasons to structure your information into:

- Datasets?
- Projects?
- Tables?

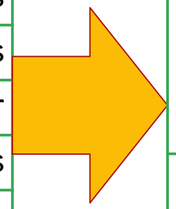
The table schema provides structure to the data



Nested and repeated columns improve the efficiency of BigQuery with relational source data

Denormalized flattened table

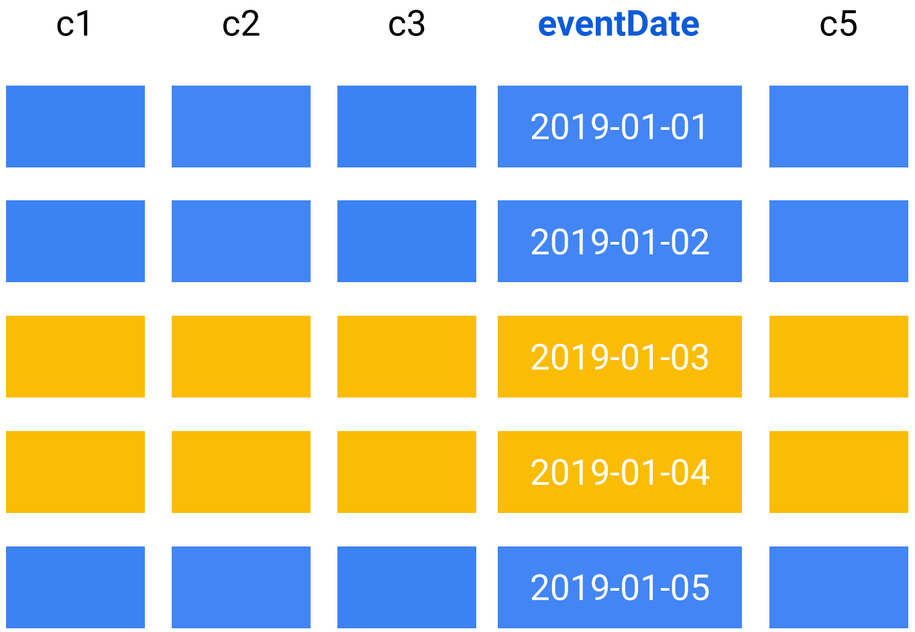
Product Name	OrderID	Date	Quantity
Siding	1600p	08/20/2019	3 boxes
Caulk	1600p	08/20/2019	12 tubes
Soffit	221b	10/29/2019	17 meters
Sealant	221b	11/05/2019	1 liter
Soffit	1600p	09/09/2018	34 meters
Siding	221b	08/20/2019	2 boxes
Caulk	221b	09/09/2019	4 tubes
Sealant	1600p	08/20/2018	2 liters



Denormalized with nested and repeated data

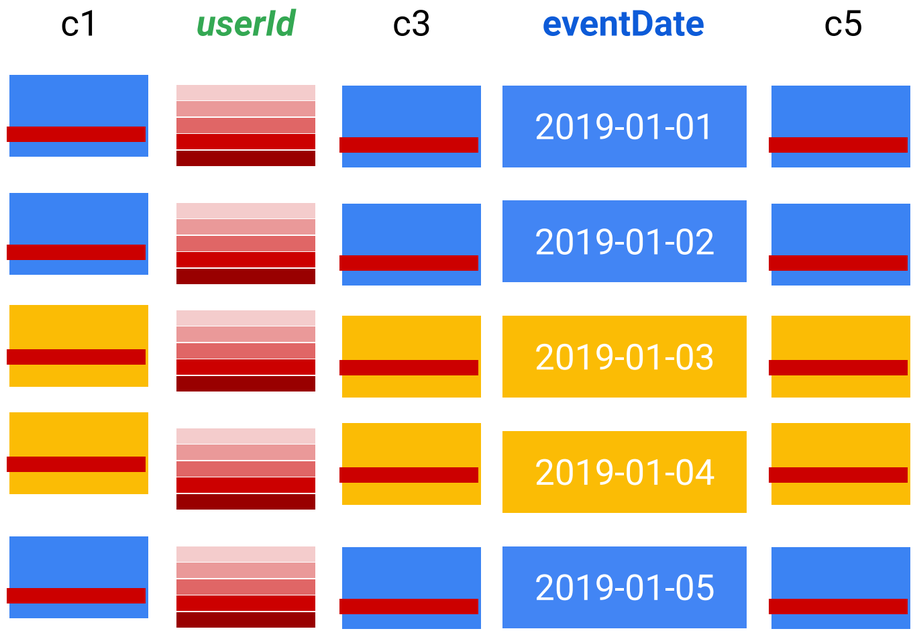
Order.ID	Order.Date	Order.Product_Name	Order.Quantity
1600p	08/20/2019	Siding	3 boxes
		Caulk	12 tubes
		Soffit	34 meters
		Sealant	2 liters
221b	10/29/2019	Soffit	17 meters
		Sealant	1 liter
		Siding	2 boxes
		Caulk	4 tubes

BigQuery automatically sorts the data based on values in the clustering columns



SELECT c1, c3 FROM ...
WHERE **eventDate** BETWEEN “2019-01-03” AND “2019-01-04”

Partitioned tables



SELECT c1, c3 FROM ... WHERE ***userId*** BETWEEN 52 and 63 AND **eventDate** BETWEEN “2019-01-03” AND “2019-01-04”

Clustered tables

Google Cloud