MODERNIZING

DATA LAKES

and

DATA WAREHOUSES

WITH

& CLOUD

Introduction

Course series

1 Off data .pw

2 data pipeline in Batch — ETL / ELT / EtLT

3 Streaming (python, data flow) pi + bronze

(Course: items 1–3)

4 Share analytics (workflow, ML)

Course introduction

Agenda
- role of data eng
- dim and delta

Introduction to data eng

modern role

a data eng Build data pipeline
- challenge in the class
- bp Data problems
- data lake / lake diff
- perform with other roles
- data access policy and governance
- productionizing and monitoring
- case study example

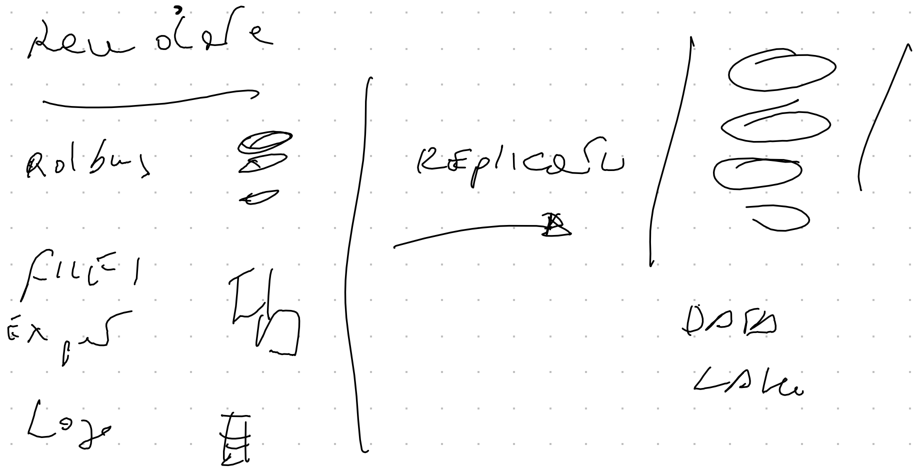
role of data eng

data eng Build Data pipeline

To have data in one place
To have data driven
business decisions

data in usable condition

raw data is not very useful



store raw data into bucket
cloud dpll

think about
- data types to handle
- ingestion throughput
- type of fmt - permissions
  access control

to make raw data accessible
for analytics

Especially on the cloud you need
ETL Extract Transform Load

using

Beam or Data flow

Streaming Load

using

pub/sub or Dataflow

By

What are challenges?

1 - access to data

2 - data accuracy and quality

3 - enough resources for Transf

4 - enough space for query data

(1) ex data scattered across multiple systems / tools / schema

commonly data is in silos with each org having it's system

(2) Build ETL to clean and transform data

=> in EDW

data in EDW is
- cleaned
- joinable
- queryable

(3) on premise = TOC

difficult — manage + provision
— scattered across time

!!

BEtter job :)

Intro to BQ

(BQ) ~ petabyte
managed
EDWH

Data mart = tables are organized into datasets

Data lake = def external schema as federated query

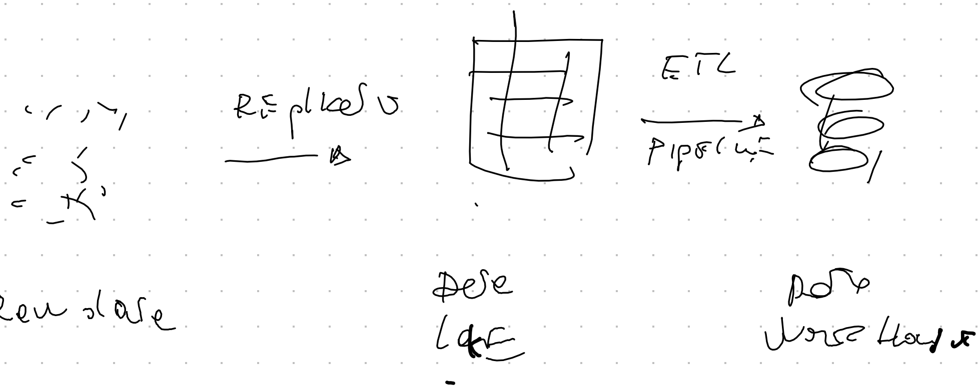
Tables and view = ANSI SQL

Grants = IAM for permissions

In BQ you don't need to provide !!
on-demand storage and compute

query has slots — e.g. CPU + RAM
resources

Data lakes and DW

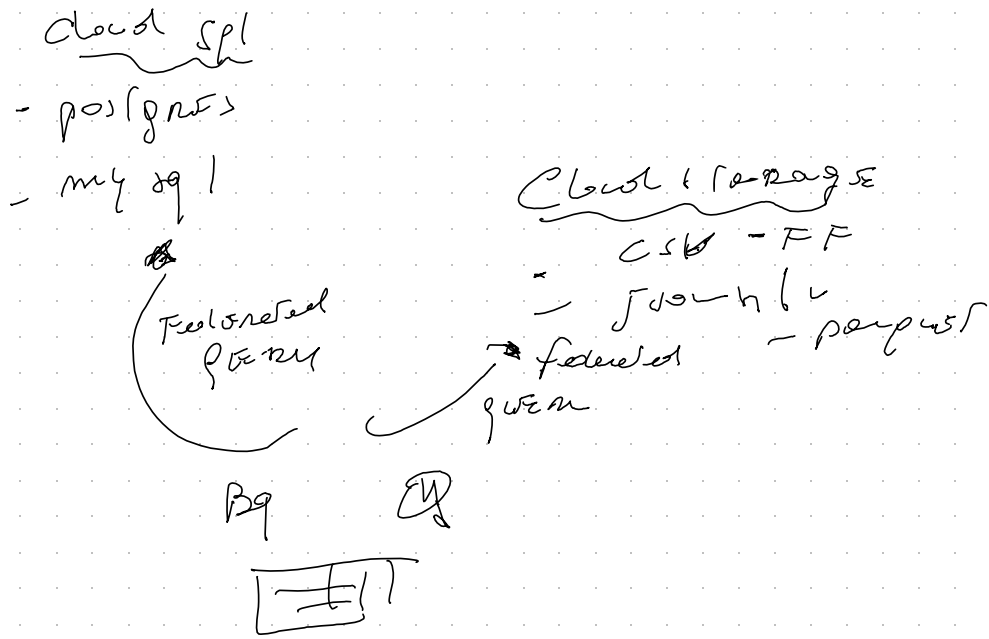


questions when building DW

- Batch and/or streaming sink for pipelines
- scale to meet needs
  - to store data
  - for query performance
- how data is
  - organized
  - cataloged
  - access controlled

What is a modern DWH

you can use BP only on federated query engine



11 from bq cloud db vs dw

cloud sql is fully managed RDMS
- automatic encryption
- 64 TB capacity
- 60 K R/W ops
- auto scale and auto bkp

cloud sql is optimized for OLTP

Cloud SQL

- scales to TB
- for backend app
- Record based storage

BQ

- scales to PB
- For analytics
- For ETL
- column-based storage

ETL partners with others tools

- NLP = uses BQ ML

Data Analysis or BI analysis

Data Eng

&

BI Engineers (no admins)

Cloud monitoring to track queries and performance

data access and governance

we need a data governance model

- who should access
- how PII are handled

use on

- data catalog → data discovery
- DLP → PII masking

production-ready pipelines

- ensure data pipeline health
  data cleanliness
- keep uptime
- handle errors

Apache Airflow

vs

cloud composer

How we bp to do analysis

- carry out queries
- combine and do analysis

- Intro
  list of NOAA public data

- Setup
  Open bp in cloud console

- Task 1
  use project -> citibike_trips

1 JOINED
  min(start_station_name) as start_name
  max(end_station_name) as end_name,
  APPROX_QUANTILES(tripduration, 10)
    [OFFSET(5)] as typical_duration
  from `bigquery-public-data.new_york_citibike.citibike_trips`
  WHERE
    start_station_id != end_station_id
  GROUP BY
    start_station_id, end_station_id

- task 2

```
select
  wx.date,
  wx.value / 10.0 as prcp
from
  `bigquery-public-data.ghcn_d.ghcnd_2015`
    as wx
where
  id = 'USW...' and qflag is NULL
order by
  wx.date
```

- task 3

```
with bicycle_rentals AS (
  select
    count(starttime) as num_trips,
    extract(DATE from starttime) as
                            trip_date
  from `bigquery-public.new_york_bike.trips`
  group by trip_date
)
```

```
rainy_days as (
  select
    date,
    (max(precip) > 5) as rainy
  from
    (# sql table 2)
  group by date
)

SELECT
  round(avg(bk.num_trips)) as
        num_trips,
  wx.rainy
from bicycle_rentals as bk
  join rainy_days as wx
  on wx.date = bk.trip_date
order by wx.rainy
```

BUILDING a DATA LAKE

Introduction

1 - what is a DL
2 - data storage options on GCP
3 - build DL using buckets
4 - secure access to blobs
5 - store all data types
6 - cloud SQL

Introduction to DL

DL: a scalable and secure and durable place to import - store - process - analyze any type of volume of info

- standard - multi - region
- cold - nearline
- cold - coldline - ML/AI

Ex construction site

1. Raw material delivered to construction site ⟷ raw data ingested

2. material need to be cut and assemble ⟷ pipelines

   ingest data sinks EDW

3. Building floor and roof ⟷ tables, etc models, reports

4. Supervisor direct and coordinate ⟷ workflow orchestration

Data Storage and ETL Options on GCP

about storage we know

- cloud storage
- cloud sql
- cloud spanner

- Firestore
+ - cloud Bigtable

Keep in mind:

- where data is now
- how big it is
- where data has to go in datasink
- How much to transfer in ΔT

Consider

ETL = data input in a format compatible to load in datasink

input AVRO into bq

ELT = we need to clean or transform the data

load from source into cloud and then apply T logic

use sql to write sql to transform input data and write it into new table

ETL = we apply T before to load

T greatly reduce the data

binary proprietary input formats
T it before to load into target

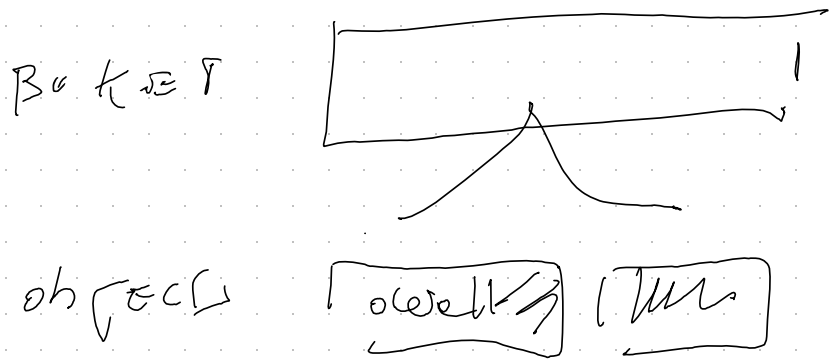
usually dataflow pipeline before write into bq

How to build a model using CS

CS signal characteristics
- permittance
- durability
- strong consistency
- availability (global or simple zone)
- high throughput

It's a object store
with low filtering structure

a Bucket contains objects



bucket is globally unique
on-created with region

low access changes eventually
data REGION

multi region Buckets
↳ replicate across regions

single region Bucket
↳ replicate across zones

metadata is stored along the obj

object

data | obj metadata

/
compression
encryption
lifecycle mgt

storage class
- standard
- Nearline storage
- Cold line storage
- Archive storage

1) 30 days once a month
4) 90 days once a quarter
4) 365 days

Secure cloud storage

control access to obj

IAM

1

std in gcp
for all
product

ACL

2

apply at bucket
apply to obj

finer

IAM bucket roles =

- Reader
- Writer
- Owner
- set ACL policy

delete/create buckets is a proj
level role

bob read/wr bucket roles
access to files w/ acl

Encryption is also via

GMEK

DEK + Key encryption key
Rotated on schedule

by Google

CMEK

by customer managed

CSEK

by customer supplied

+ client-side encryption
if extra-protection

You can set lock policy so the
bucket cannot be changed

Short overview of data

Transactional system are 80% WRITE,
20% READ

Analytical systems are populated from transactional system

If you require low latency in order of milliseconds

↓

cloud BigT

cloud sql vs relational database

cloud sql
- fully managed (RDBMS) db services

- sql server
- mysql
- postgresql

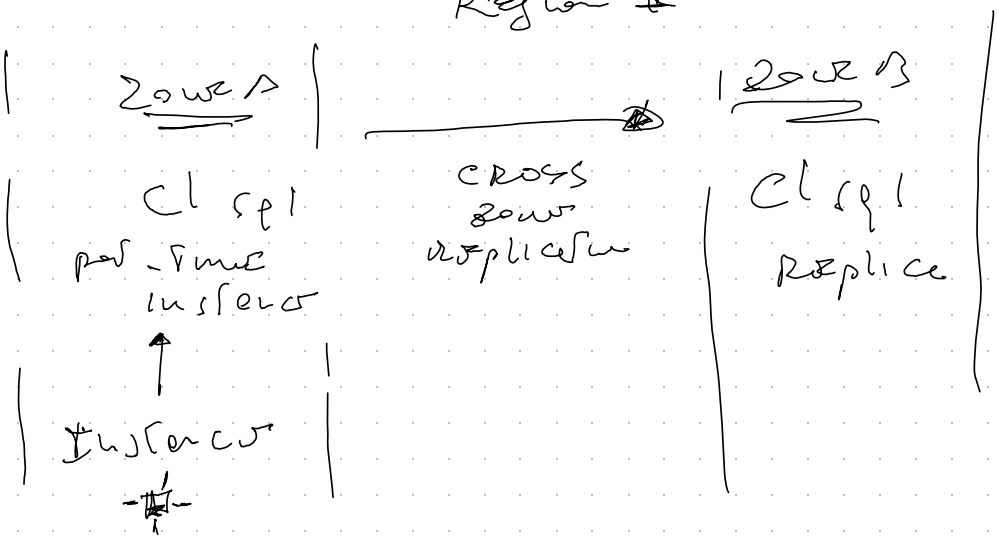
Cloud Spanner (R+W)

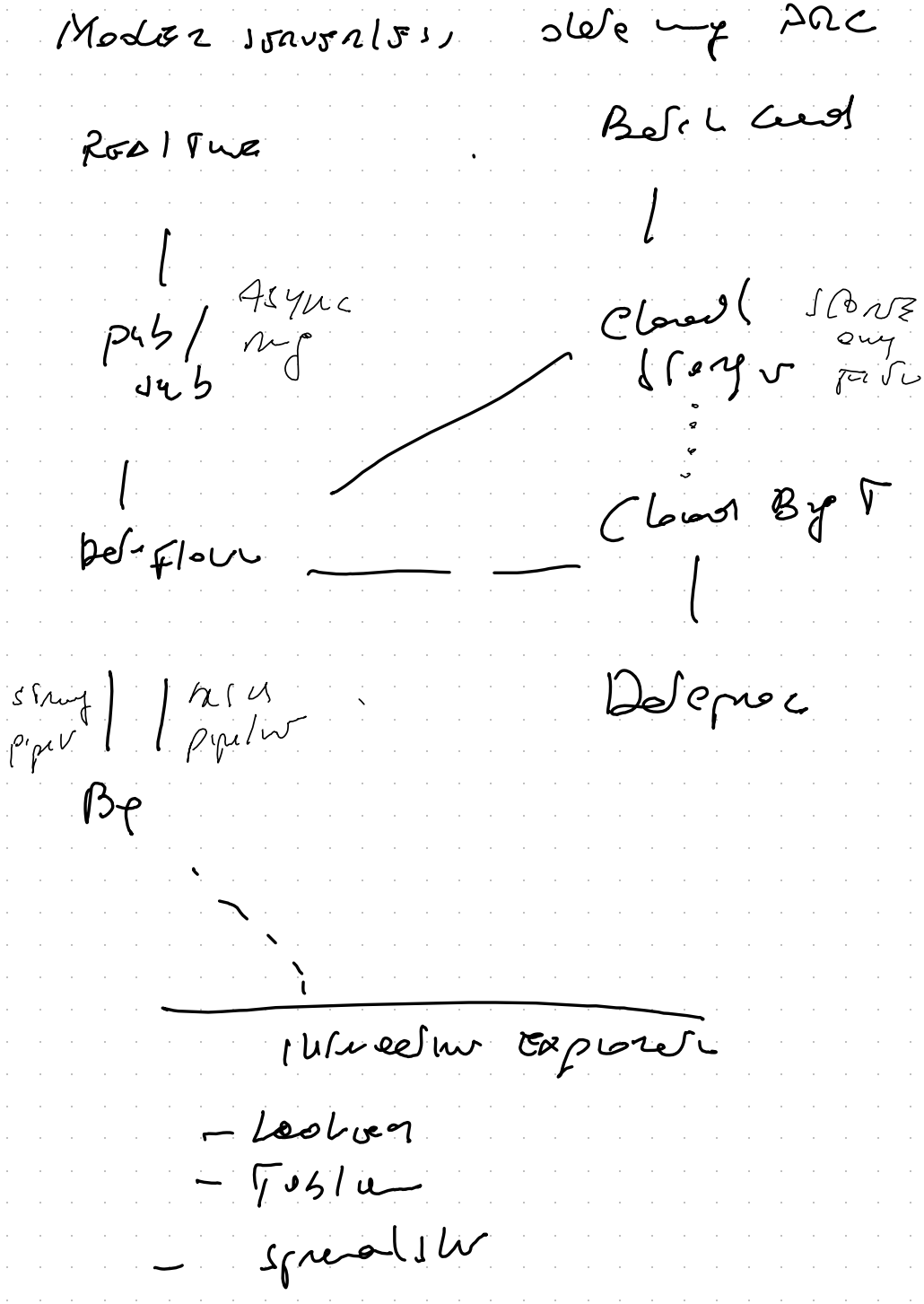
- 64 processing units
- 100 GB SSD

Cloud Bigtable (R)

- Replicas

> If you need HDFS (R+W)
Consider Cloud Dataproc





II Lab Load Taxi data into gcloud sql

- create cloud sql instance
- create web db
- import
- check data integrity

- Task 1

0 export PROJECT_ID=$(gcloud info --format='value(config.project)')

1 export BUCKET=${PROJECT_ID}-ml

- Task 2

0 gcloud sql instances create taxi --tier=db-n1-std-1 --activation_policy=ALWAYS

1 gcloud sql users set-password root

2 # ip address

export ADDRESS=$(wget -qO ——)

3 gcloud ______ patch taxi --authorized networks

• Task 3

→ gcloud storage cp gs:// ______ .csv

1. mysql --host=$:MYSQLIP --user=root

2) Load DATA ______

BUILDING A DATA WAREHOUSE

Introduction

- 1 modern data warehouse
- 2 intro to bq
- 3 load data in bq
- 4 Kimball schema
- 5 schema only
- 6 Normal Nested fields

The modern data warehouse

- Scale from GB to PB
- Serverless (no-ops)
- Natural eco-system
- ETL eco-system
- up to minute data
- ML available with no data mov
- Security and collaboration

Intro to bq

cost similar to cloud storage
so no indexes

GIS and ML
streaming data insert via api

ANSI SQL 2011

no vacuum process like redshift
no idx to be deleted

bq tables are column oriented

we have storage engine
and
compute engine

no need to provision of resources

slots = unit of computation ← cpu / memory / network / resources

avg 2000 slots for users

get started with bq

data / tables organized into datasets

"project . dataset . table"

datasets = analytical model
project level scope to where [illegible] occurs,
only billing ⟷ project

you can a query in your own proj
even using data data in another
project

to run queries ⟶ IAM role to
submit a job

AC is at a dataset or table or
view or column level

bq datasets can be — regional
— multi-regional

each table has a defined schema

email address can be used
or
payment bank/card accounts

Admin & system events are all logged

table expiration is system event

four predefined roles
- bq admin
- bq data editor
- bq data owner — create routine, datasets, + auth views
- bq data viewer
- bq job user
- bq metadata user

you can share
- authorized views
- row level permission

CREATE ROW ACCESS POLICY

sales_filter ON [table]

GRANT TO ("group:sales-apac@example.com")

FILTER USING

(Region = "APAC")

Region

APAC

US

Authorized View →

User can access view

but not

underlying Table / data

Diff materialized view → base query

& data loaded automatically

Load data into BQ

Use for EL / ELT / ETL

you can batch load data into BQ

CSV - JSON - avro, parquet, orc

gzip compressed, firestore export

Load job - creates target table
if not exist

Schema is inferred on load job

BQ has limits on the # of files
and size you can load every day

BQ API is used mostly by
dataproc / dataflow

BQ has connectors

BQ maintains 7 day history

Create snapshot

@ CREATE OR REPLACE table t1
AS
SELECT * FROM `______`
FOR SYSTEM TIME AS OF
TIMESTAMP_SUB(CURRENT_TS, INTERVAL 24H)

BQ transfer data service does automatic data transfer

BQ Transfer service provides

- connections (100+)
- transformation templates
- scheduling

∘ gcloud storage cp *.csv
    gs://mybucket

! bq load ———

bq supports standard DML

INSERT / UPDATE /
    DELETE / MERGE

! bq is not OLTP

Want to create complex expressions
in another language

! create temp function multiply (x, y)
RETURNS FLOAT64
LANGUAGE JS AS ... return x * y; ...

Loading data - bq : L3

- loading data. in bq

then - ______ using ui /console

- using cli

tools

```
bq load --source_format=CSV \
        --autodetect \
        --noreplace \
        dataset1.table A
        gs://cloud-xy/data/123.csv
```

explore schemas

information_schema — TABLES
                   — COLUMNS

--TABLES--

Schema design

By data u

customer	order id	date	items
Doug	123	2021/01	Potato, Tomato, 1kg onion
Alex	124	—	potatoes, 10kg Cipolle

Normalized data

orders

A1	123	2021/01
A2	124	—

order_items

123	Potato	—
123	Tomato	—
123	Onion	1kg
124	Poule	—
124	Cipolle	10kg

customer

Doug	A1
Alex	A2

access data by row
by column

Java JDBC

diff way to operate data

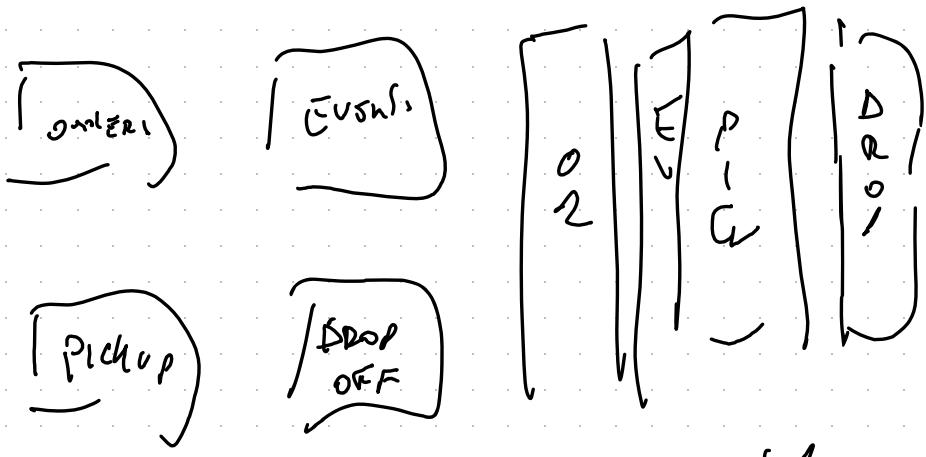
usually you DENORMALIZE data before putting into BQ

BQ supports columns with
nested and
repeated data

order_id	order_date	ord.product	ord.quantity
123	20/10	pear	✓
	,	carrot	1
		apple	10
124	✓	lemon	
		orange	2

in this way BQ can organize the data in smart way / colocated
still
using a flat/denormalized structure

Normalized and separated fields



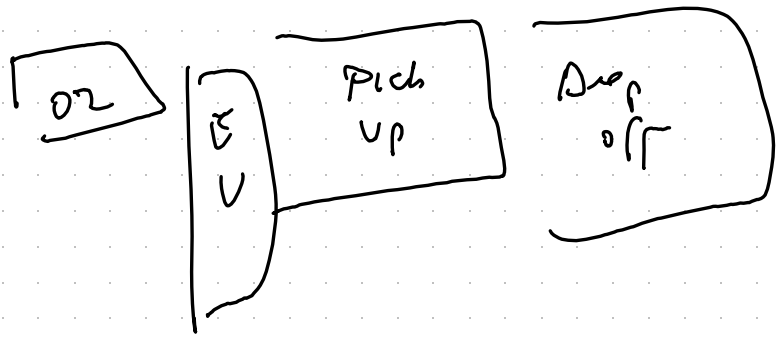
Many tables

Easily joins

Joins

Slower

Volume



Not used and // diff frequency

separated

working with JSON data

- loading semi-structured data
- creating and querying of arrays / structs
- querying nested / repeated fields

Task.1

create dataset -> fruit-store

Task.2

Row	Fruit	Price
1	raspberry	jelly
2	blackberry	jelly
3	cherry	fresh.
4	apple	jelly

repeated fields!

Row	fruit (array)	Price
1	[raspberry, blackberry, apple]	jelly
2	[cherry]	fresh

array only type

Struct
New versio fruit_items
total

Fork ?

ARRAY_AGG()

SELECT
fullvisitorId,
date,
ARRAY_AGG(product_name) as pn
ARRAY_AGG(page_title) as pt
FROM `ecommerce.all_sessions`
group By fullVisitorId, date

Recap
ARRAY_LENGTH(column)
ARRAY_AGG(DISTINCT <field>)
ARRAY_AGG(<field> ORDER BY <..>)
ARRAY_AGG(<field>) LIMIT N)

Could access a field product on
a value with sql < array >

! before we can query repeated
you must first break array -> rows

hits.page.pageTitle

select distinct
using
h_unnest.page.pageTitle

from `ga-sessions`
unnest (hits) as h_unnest

Tables

Look at structs

- 1+ fields
- same or diff types
- other structs

select _._
totals.*, device.*._

Task 6.

```
SELECT STRUCT("Rudisha" as name
  23.: res split)
AS runner

SELECT STRUCT('Rudisha' as name,
[23.4, 25.2, 26.1, 26.1] as splits)
AS runner
```

schema as text:

```
[
  { name, STRING, NULLABLE },
  { runner, RECORD, REPEATED
  { fields: [
      { name
        STRING
        NULLABLE }
      { splits
        FLOAT
        REPEATED }
  ]
]
}
```

Tables schema to be created

Field name	Type	Mode
RACE	STRING	NULLABLE
* PARTICIPANTS	RECORD	REPEATED
name	STRING	NULL
splits	FLOAT	REPEATED

INSERT * FROM racing.race_results

row	race	participants.name	participants.splits
1	800m	Rudisha	23.4 26.3 26.1
		Makwala	24.5 "
		Murphy	23.1 24.2 0.0

SELECT race, participants.name

FROM racing.race_results

CROSS JOIN

participants # used as table inside the table

race_results

.

participants # all full name

or alias

SELECT race, participants.name

FROM racing.race_results AS r

,

r.participants

Join 1.

use explode

SELECT count(p.name) as pc

FROM racing.race_results AS r

,

UNNEST(r.participants) AS p

Task 3.

```
select
    p.name
    AVG(split_time) as avg_st
from racing.race_results as r
    unnest(r.performances) as p
    unnest(p.splits) as split_time
group by p.name
order by avg_st desc
```

optimizing with partition and clustering

reduce cost with partition

c1 c2 c3 event date c4

↓

data is split in separate chunks

querying them will be READ

3 way partition

- ingestion time -

  bq query --destination_table d.t

  --time_partitioning_type=DAY

- on col of type datetime, date, timestamp

  bq mk --table --schema

  a:STRING

  b:TIMESTAMP

  --time_partitioning_field b

- integer type column

  --range_partitioning = (customer_id, 0, 100, 10)

improve cost and performance

```
SELECT
  Acol,
FROM
  mypowerBI.table1
WHERE
  _PARTITIONTIME >
    TIMESTAMP_SUB
      (TIMESTAMP(
        '2022-07-01',
        INTERVAL 5 DAY)
```

before on LEFT most the partition field

bq query --require_partition_filter

clustering on field helps too

```
CREATE TABLE mydb.myclustered
  ( c1 NUM,
    userId STRING,
    c2 STRING,
    eventDate TIMESTAMP
  )
PARTITION BY DATE(eventDate)
CLUSTER BY userId
OPTIONS
  (partition_expiration_days=3)

AS
  SELECT * FROM mydb.mytable
```

BQ does auto re-clustering for you

use clustering → your data is partitioned on dates/time
→ w/ agg/filter on commonly fields