# Assignment 2 – Partitioning and Clustering in BigQuery

## Introduction

In BigQuery, partitioning and clustering are optimization techniques that organize data for faster access and reduced cost. Partitioning divides the table into segments based on a column, while clustering arranges data within partitions based on column values.

## Partitioning

Partitioning divides a table into segments (partitions) based on a column.  
  
Types of Partitioning:  
• Ingestion-time: Uses \_PARTITIONTIME (insert timestamp)  
• Column-based: Based on DATE, TIMESTAMP, or INTEGER column  
• Integer-range: Custom numeric ranges  
  
Benefits:  
• Data pruning – queries scan only relevant partitions  
• Reduced query costs when filtering by partition column  
• Ideal for time-series or log data  
  
Example:  
CREATE TABLE sales\_partitioned  
PARTITION BY DATE(order\_date) AS  
SELECT \* FROM raw\_sales;

## Clustering

Clustering organizes data within a table (or a partition) by column values.  
  
Benefits:  
• Boosts performance on filtered queries (e.g., WHERE, JOIN)  
• Reduces scanned data  
• Supports up to 4 clustering columns  
  
Example:  
CREATE TABLE sales\_clustered  
CLUSTER BY customer\_id, product\_id AS  
SELECT \* FROM raw\_sales;

## Partitioning vs Clustering

| Feature | Partitioning | Clustering |  
|--------------|-------------------------------|-----------------------------------|  
| Granularity | Coarse (entire partitions) | Fine-grained (within partitions) |  
| Filter Req. | Must filter on partition column| Benefits without strict filtering |  
| Column Types | DATE, TIMESTAMP, INTEGER | STRING, INT, DATE, etc. |  
| Max Columns | 1 partition column | Up to 4 clustering columns |  
| Storage | Physical partitioning | Logical ordering |  
| Best Use | Time-based or range queries | High-cardinality filter columns |

## Step 1: Create Dataset

CREATE SCHEMA IF NOT EXISTS demo\_analytics;

## Step 2: Create Partitioned and Clustered Table

CREATE TABLE demo\_analytics.sales\_partitioned\_clustered (  
 order\_id STRING,  
 customer\_id STRING,  
 product\_id STRING,  
 order\_amount FLOAT64,  
 order\_date DATE  
)  
PARTITION BY order\_date  
CLUSTER BY customer\_id;

## Step 3: Insert Sample Records

INSERT INTO demo\_analytics.sales\_partitioned\_clustered (  
 order\_id, customer\_id, product\_id, order\_amount, order\_date  
)  
VALUES  
 ('O001', 'C001', 'P001', 100.0, '2024-09-01'),  
 ('O002', 'C002', 'P002', 200.0, '2024-09-01'),  
 ('O003', 'C001', 'P003', 150.0, '2024-09-02'),  
 ('O004', 'C003', 'P001', 300.0, '2024-09-02'),  
 ('O005', 'C002', 'P004', 120.0, '2024-09-03'),  
 ('O006', 'C004', 'P002', 180.0, '2024-09-04'),  
 ('O007', 'C001', 'P005', 220.0, '2024-09-05');

## Step 4: Sample Query (Using Partitioning + Clustering)

SELECT \*  
FROM demo\_analytics.sales\_partitioned\_clustered  
WHERE order\_date = '2024-09-02'  
 AND customer\_id = 'C001';

## Query Benefits

• Filters by order\_date → partition pruning  
• Filters by customer\_id → benefits from clustering  
This results in lower query cost and improved performance.