How does BigQuery store data?

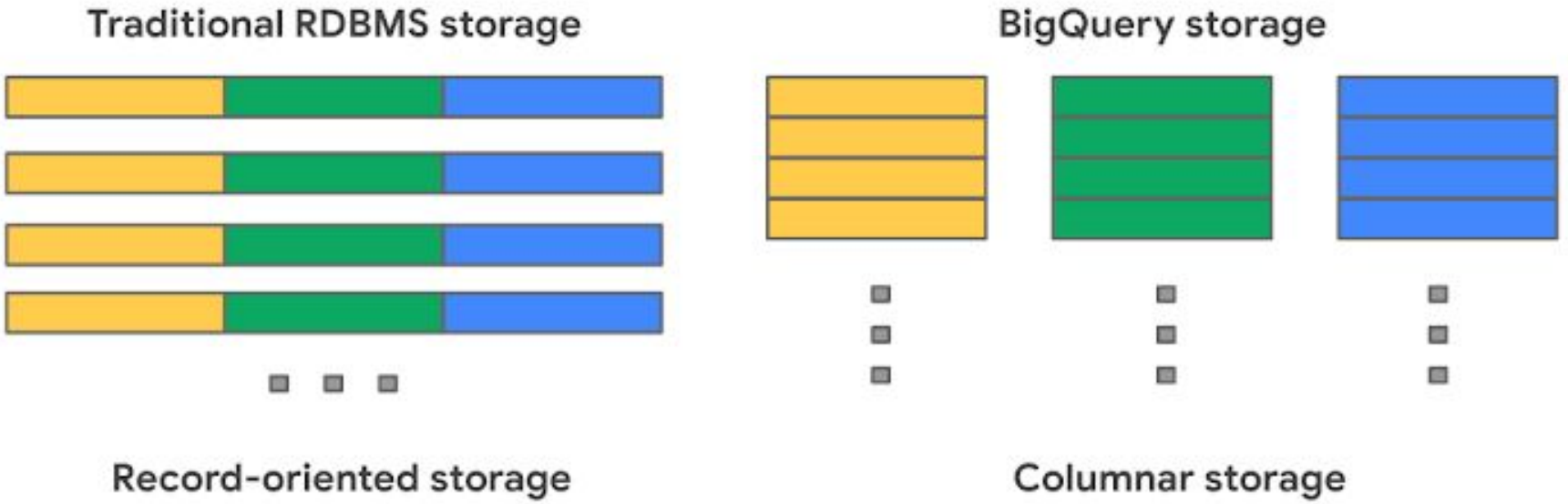
Columnar Storage

- BigQuery offers f**ully managed storage**:
  - You don’t have to provision servers.
  - Sizing is done automatically and you only pay for what you use.

- BigQuery is designed for large scale data analytics, data is stored in **columnar** format.
- Traditional relational databases, like Postgres and MySQL, store data row-by-row in record-oriented storage.

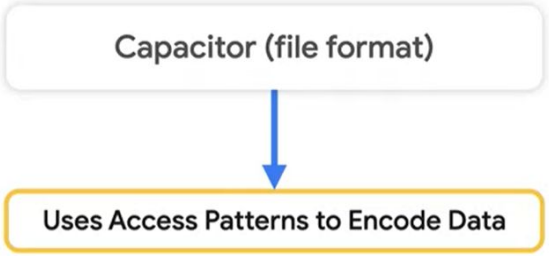
- This makes them great for transactional updates and OLTP (Online Transaction Processing) use cases because they only need to open up a single row to read or write data.
- When you want to perform an aggregation like a sum of an entire column, you would need to read the entire table into memory.

Columnar Storage



- BigQuery uses columnar storage where each column is stored in a separate file block.
- This makes BigQuery an ideal solution for OLAP (Online Analytical Processing) use cases.
- When you want to perform aggregations you only need to read the column that you are aggregating over.

Capacitor File Format

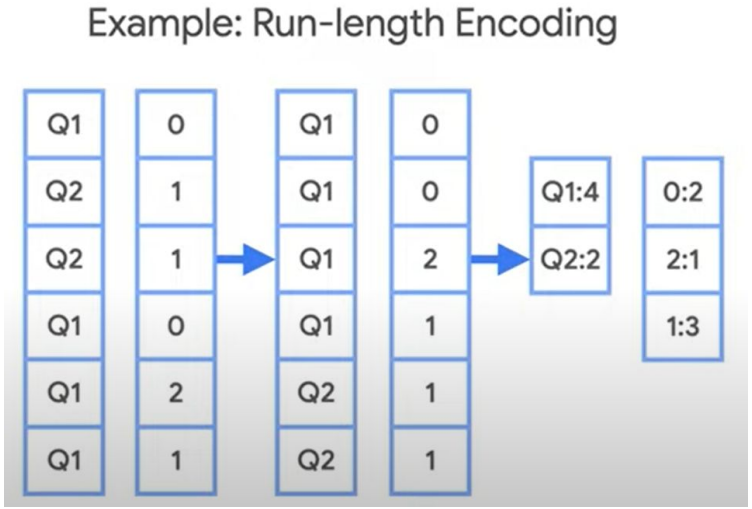


BigQuery stores data in a proprietary columnar format called Capacitor.

This means that the values of each field, or column, are stored separately so the overhead of reading the file is proportional to the number of fields you actually read.

This doesn't necessarily mean that each column is in its own file, it just means that each column is stored in a file block, which is actually compressed independently for increased optimization.

Capacitor builds an approximation model that takes in relevant factors like the type of data (e.g. a really long string vs. an integer) and usage of the data (e.g. some columns are more likely to be used as filters in WHERE clauses) in order to reshuffle rows and encode columns.



Capacitor File Format

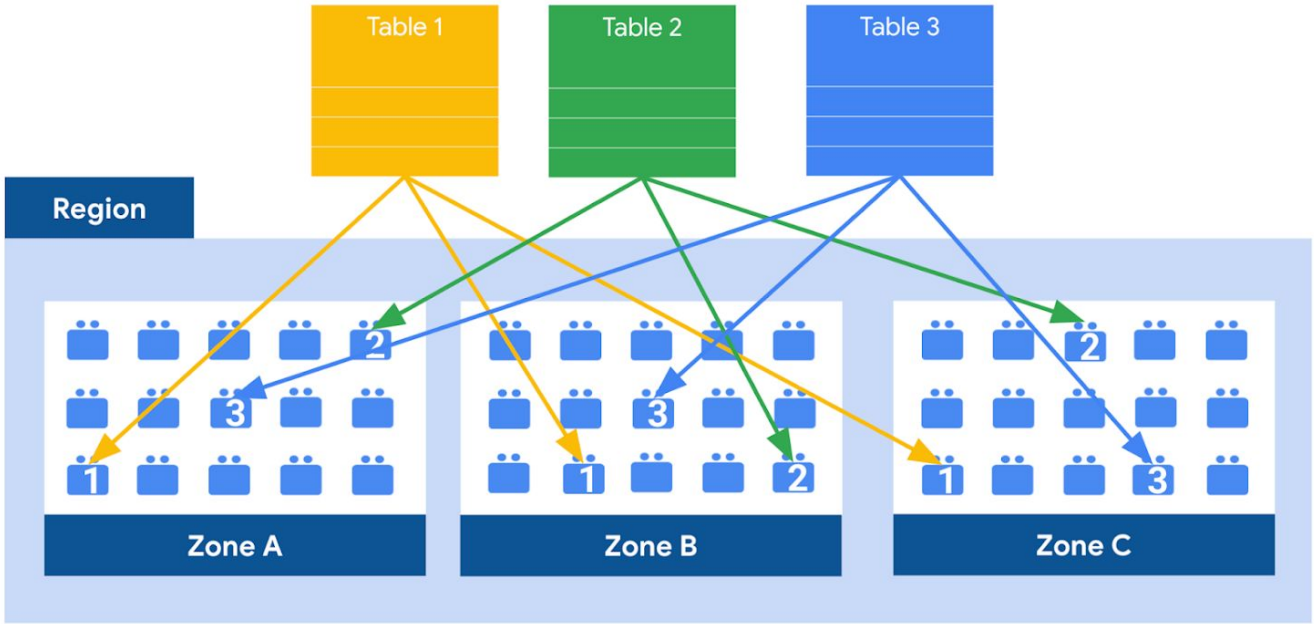
UID	First Name	Last Name	Age	City	Purchases

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	{...}]

Colossus Distributed File System

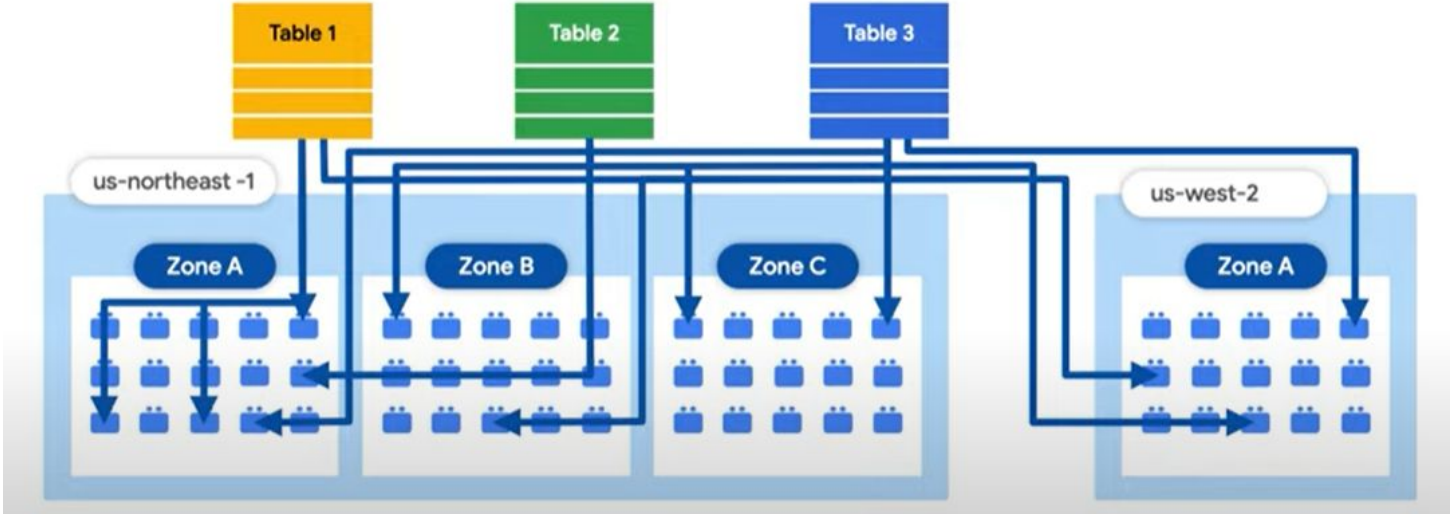


BigQuery's persistence layer is provided by Google's distributed file system, Colossus, where data is automatically compressed, encrypted, replicated, and distributed.

Colossus Distributed File System

Colossus also ensures durability by using something called erasure encoding - which breaks data into fragments and saves redundant pieces across a set of different disks.

However, to ensure the data is both durable and available, the data is also replicated to another availability zone within the same region that was designated when you created your dataset.

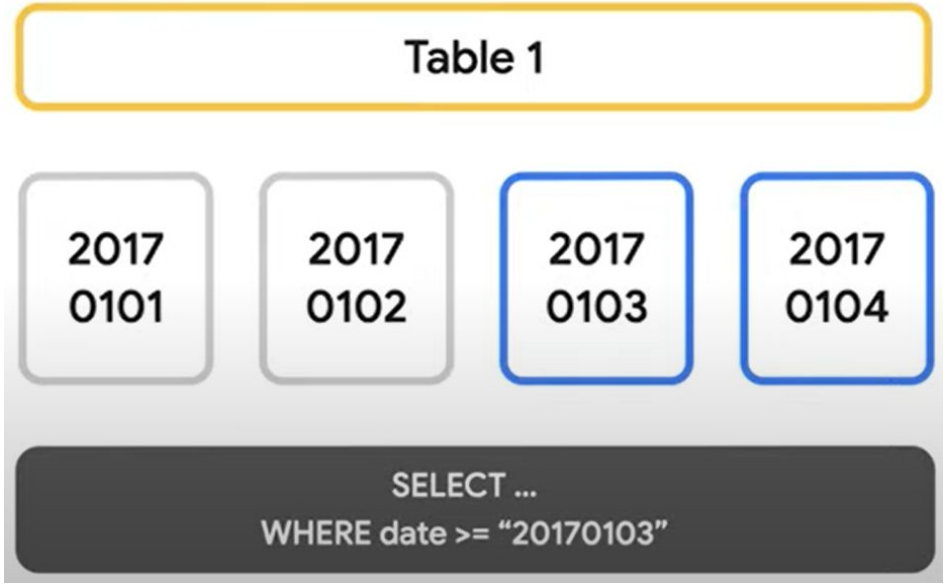


Storage Optimization

Partitioning

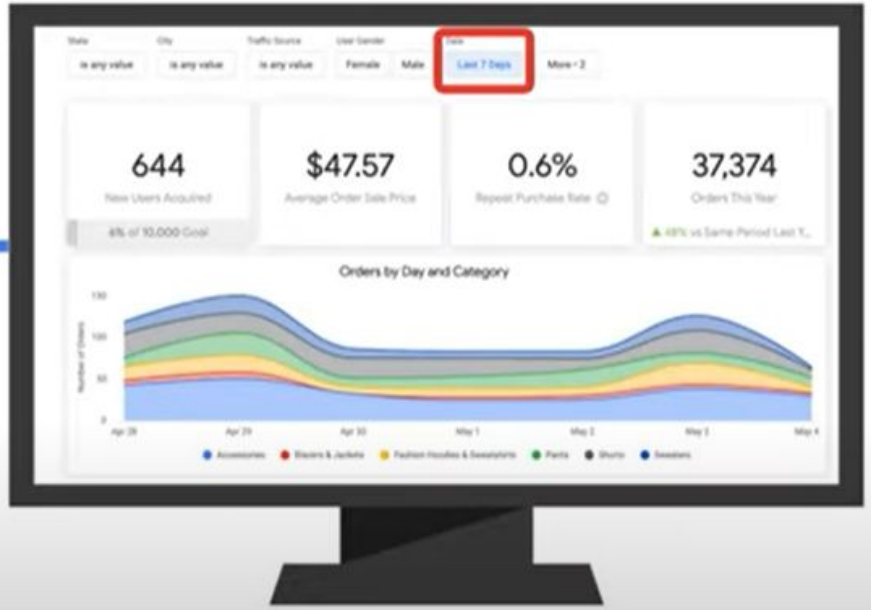
- Ingestion Time
- Date / Timestamp Column
- Integer Column

Partitioning

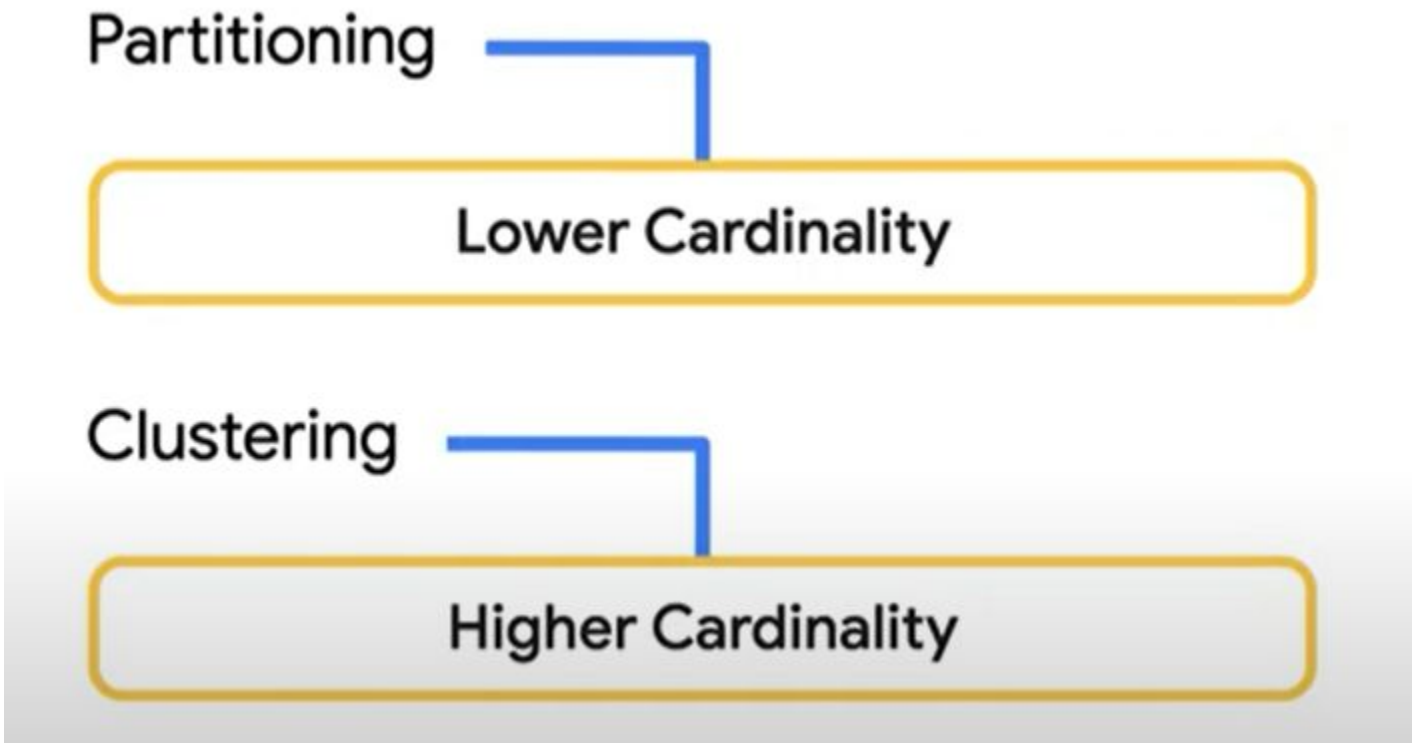


Storage Optimization

```
CREATE TABLE `project-id.my_dataset.order_items`
 (
   ORDER_ID INT64,
          ...
 )
PARTITION BY DATE(created_at)
```

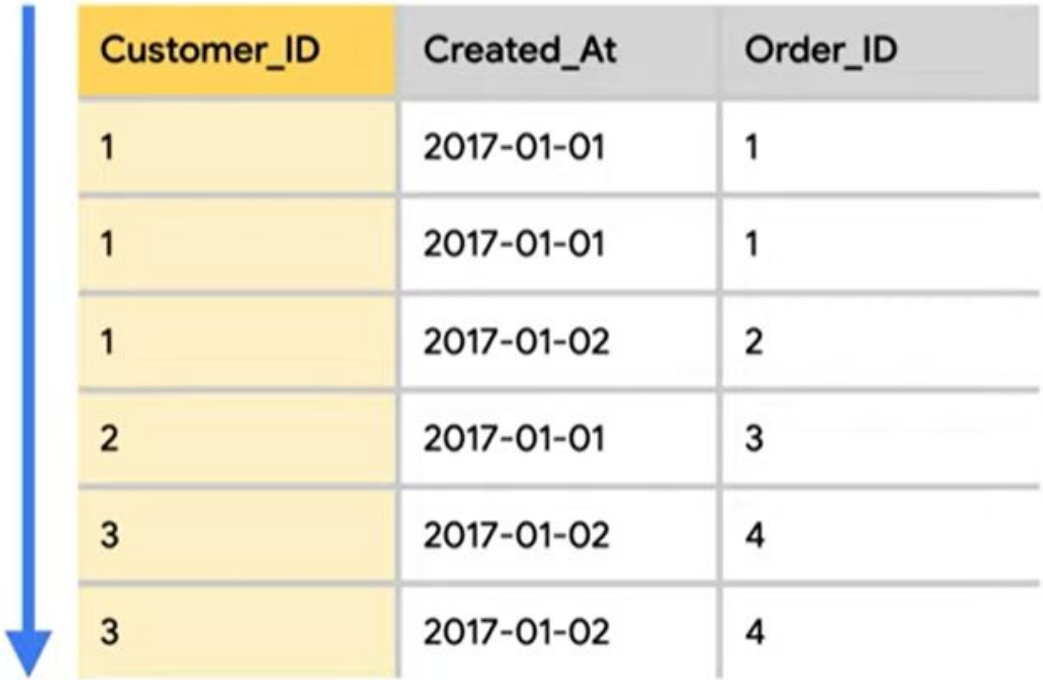


Storage Optimization



Storage Optimization

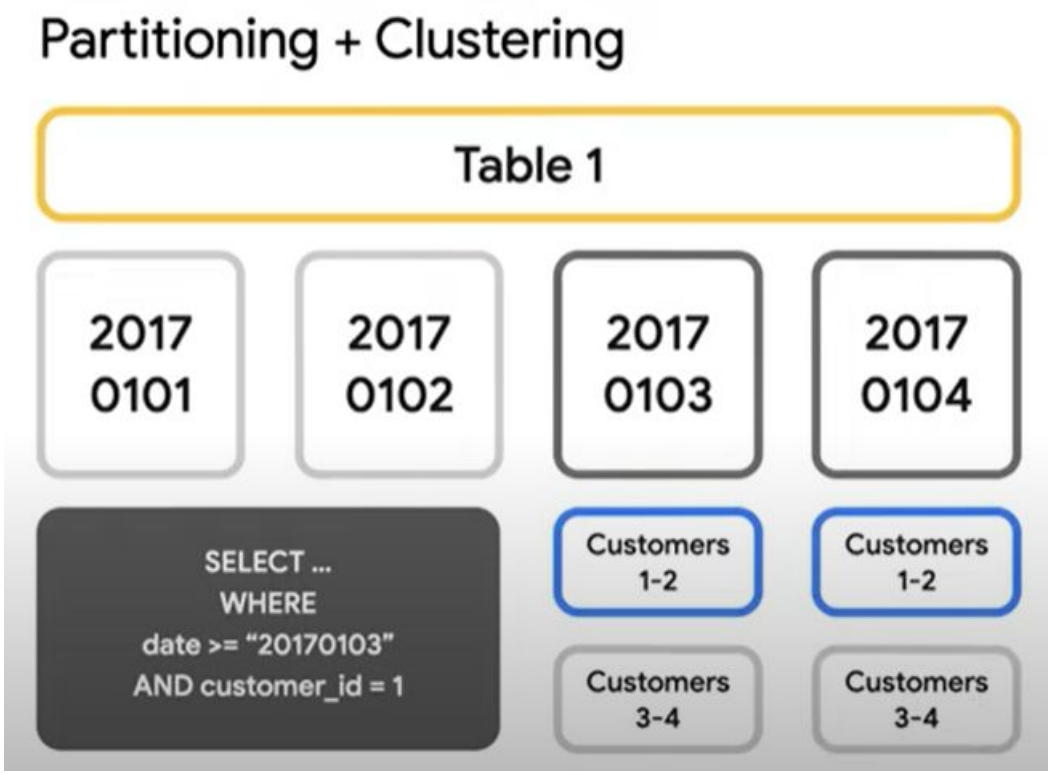
Clustering



Customer_ID	Created_At	Order_ID
1	2017-01-01	1
1	2017-01-01	1
1	2017-01-02	2
2	2017-01-01	3
3	2017-01-02	4
3	2017-01-02	4

Storage Optimization

Storage Optimization




```
CREATE TABLE `project-id.my_dataset.order_items`
 (
   ORDER_ID INT64,
            ...
 )
PARTITION BY DATE(created_at)
CLUSTER BY customer_id
```

Partitioning

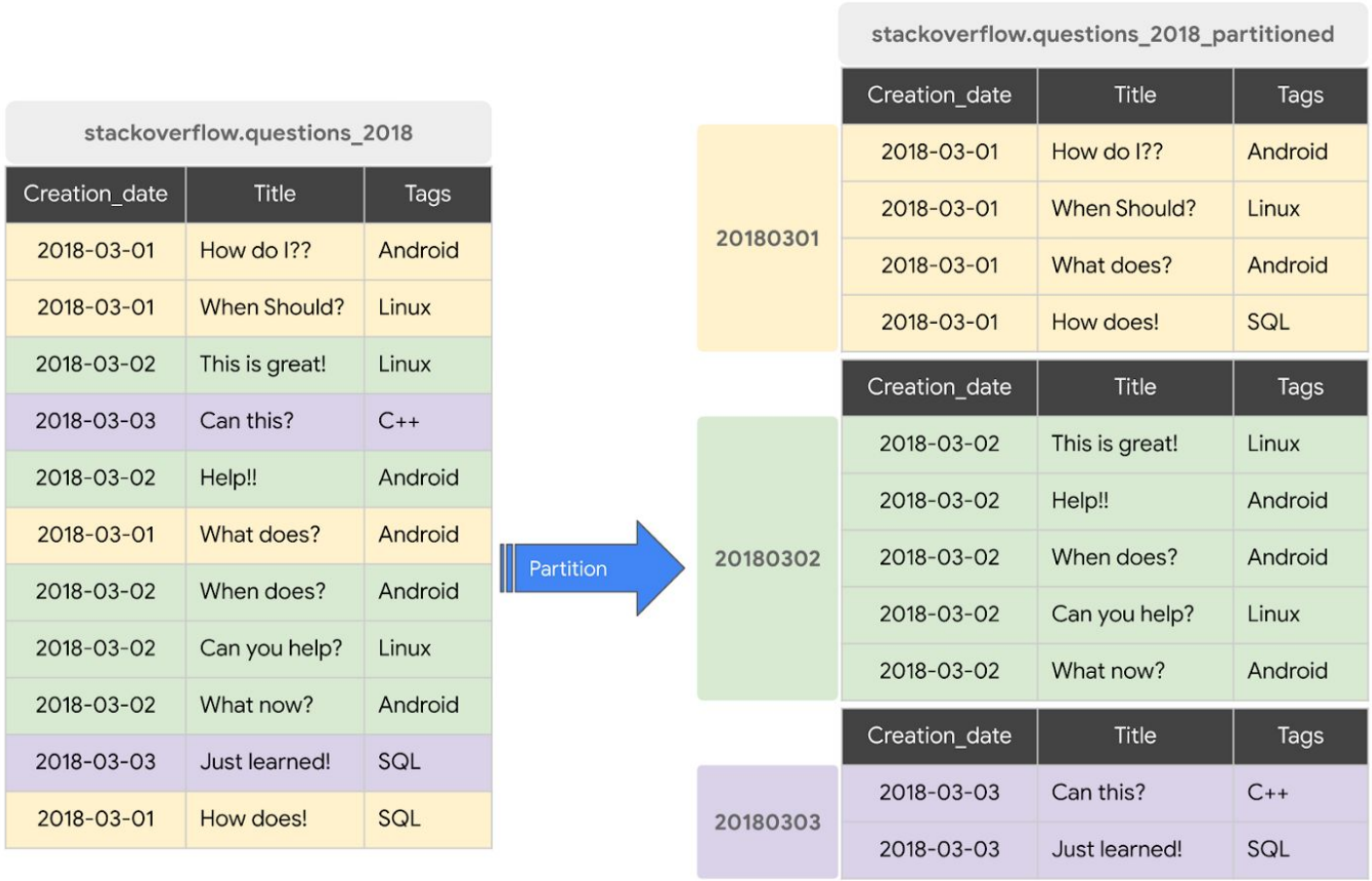
A partitioned table is a special table that is divided into segments, called partitions.

BigQuery leverages partitioning to minimize the amount of data that workers read from disk.

Queries that contain filters on the partitioning column can dramatically reduce the overall data scanned, which can yield improved performance and reduced query cost for on-demand queries.

New data written to a partitioned table is automatically delivered to the appropriate partition.

Partitioning



Partitioning

BigQuery supports the following ways to create partitioned tables:

- Ingestion time partitioned tables: daily partitions reflecting the time the data was ingested into BigQuery. This option is useful if you'll be filtering data based on when new data was added. For example, the new Google Trends Dataset is refreshed each day, you might only be interested in the latest trends.
- Time-unit column partitioned tables: BigQuery routes data to the appropriate partition based on date value in the partitioning column. You can create partitions with granularity starting from hourly partitioning. This option is useful if you'll be filtering data based on the date value in the table, for example looking at the most recent transactions by including a WHERE clause for transaction_created_date
- INTEGER range partitioned tables: Partitioned based on an integer column that can be bucketed. This option is useful if you'll be filtering data based on an integer column in the table, for example focusing on specific customers using customer_id. You can bucket the integer values to create appropriately sized partitions, like having all customers with IDs from 0-100 in the same partition.

Partitioning

- Partitioning is a great way to optimize query performance, especially for large tables that are often filtered down during analytics.
- When deciding on the appropriate partition key, make sure to consider how everyone in your organization is leveraging the table.
- For large tables that could cause some expensive queries, you might want to require partitions to be used.
- Partitions are designed for places where there is a large amount of data and a low number of distinct values. A good rule of thumb is making sure partitions are greater than 1 GB.
- If you over partition your tables, you'll create a lot of metadata - which means that reading in lots of partitions may actually slow down your query.

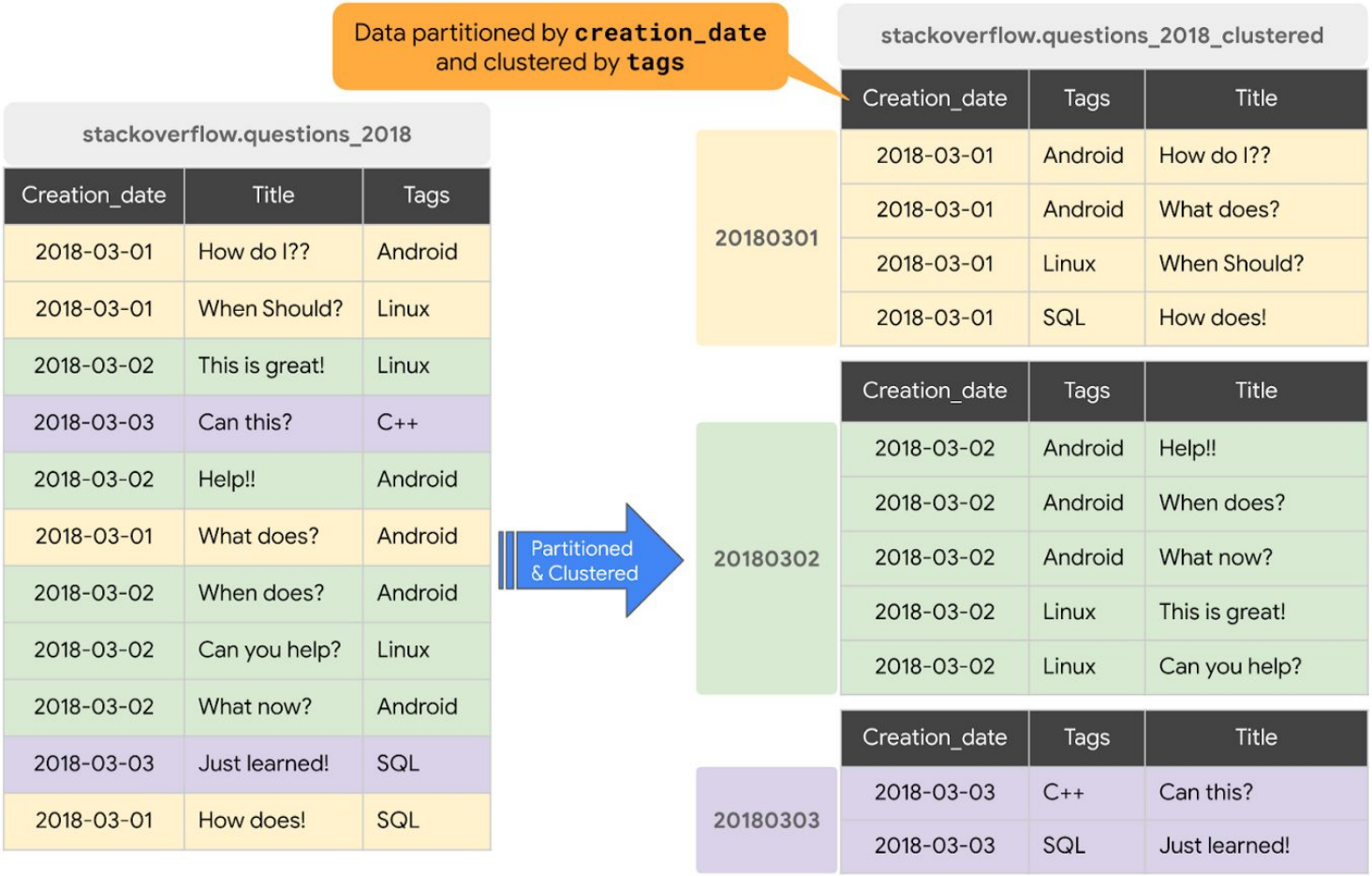
Clustering

- When a table is clustered in BigQuery, the data is automatically sorted based on the contents of one or more columns (up to 4, that you specify).

- Usually high cardinality and non-temporal columns are preferred for clustering, as opposed to partitioning which is better for fields with lower cardinality.

- You’re not limited to choosing just one, you can have a single table that is both partitioned and clustered!

Clustering


Data partitioned by creation_date and clustered by tags
stackoverflow.questions_2018
Creation_date | Title | Tags
2018-03-01 | How do I?? | Android
2018-03-01 | When Should? | Linux
2018-03-02 | This is great! | Linux
2018-03-03 | Can this? | C++
2018-03-02 | Help!! | Android
2018-03-01 | What does? | Android
2018-03-02 | When does? | Android
2018-03-02 | Can you help? | Linux
2018-03-02 | What now? | Android
2018-03-03 | Just learned! | SQL
2018-03-01 | How does! | SQL
Partitioned & Clustered
stackoverflow.questions_2018_clustered
20180301
Creation_date | Tags | Title
2018-03-01 | Android | How do I??
2018-03-01 | Android | What does?
2018-03-01 | Linux | When Should?
2018-03-01 | SQL | How does!
20180302
Creation_date | Tags | Title
2018-03-02 | Android | Help!!
2018-03-02 | Android | When does?
2018-03-02 | Android | What now?
2018-03-02 | Linux | This is great!
2018-03-02 | Linux | Can you help?
20180303
Creation_date | Tags | Title
2018-03-03 | C++ | Can this?
2018-03-03 | SQL | Just learned!

Clustering

The order of clustered columns determines the sort order of the data. When new data is added to a table or a specific partition, BigQuery performs free, automatic re-clustering in the background. Specifically, clustering can improve the performance for queries:

- Containing where clauses with a clustered column: BigQuery uses the sorted blocks to eliminate scans of unnecessary data. The order of the filters in the where clause matters, so use filters that leverage clustering first

- That aggregate data based on values in a clustered column: performance is improved because the sorted blocks collocate rows with similar values

- With joins where the join key is used to cluster the table: less data is scanned, for some queries this offers a performance boost over partitioning!

Denormalizing

when performing analytical operations on normalized schemas, usually multiple tables need to be joined together. If we instead denormalize our data, so that information (like the customer address) is repeated and stored in the same table, then we can eliminate the need to have a JOIN in our query

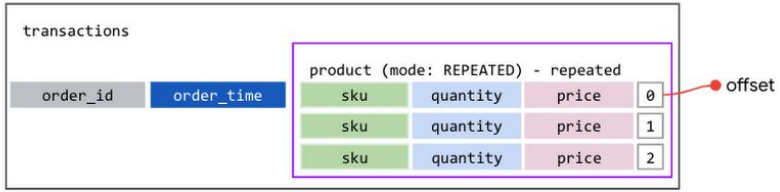
For BigQuery specifically, we can also take advantage of support for nested and repeated structures. Expressing records using STRUCTs and ARRAYs can not only provide a more natural representation of the underlying data, but in some cases it can also eliminate the need to use a GROUP BY statement. For example, using ARRAY_LENGTH instead of COUNT.

Denormalizing

Nested Fields

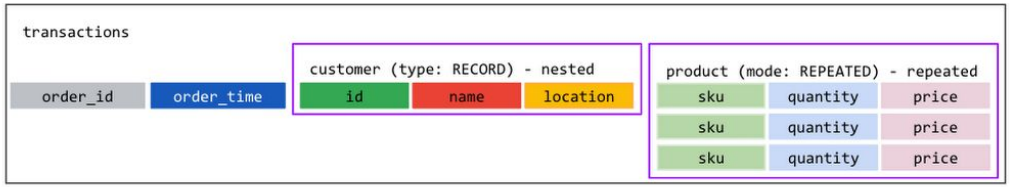
- **STRUCT/RECORD** data type contains ordered fields with a type and name.
- Use dot notation to query a nested column.
  E.g. **customer.name** refers to **name** field in **customer** column.

Repeated Fields



- **ARRAY** data type is an ordered list of zero or more elements of the same data type.
  For e.g. **product** is an **ARRAY** of **STRUCT** here.
- Use `UNNEST()` to flatten the repeated data and **OFFSET/ORDINAL** to access individual element

Nested Repeated Fields



- Combining nested and repeated fields denormalizes a 1:many relationship without joins.
- Use dot notation to query a nested column and `UNNEST()` to flatten the repeated data.

Optimizing for storage costs

- When it comes to optimizing storage costs in BigQuery, you may want to focus on removing unneeded tables and partitions.
- You can configure the default table expiration for your datasets, configure the expiration time for your tables, and configure the partition expiration for partitioned tables.
- This can be especially useful if you're creating materialized views or tables for ad-hoc workflows, or if you only need access to the most recent data.
- You can take advantage of BigQuery's long term storage.
- If you have a table that is not used for 90 consecutive days, the price of storage for that table automatically drops by 50 percent to $0.01 per GB, per month.
- This is the same cost as Cloud Storage Nearline, so it might make sense to keep older, unused data in BigQuery as opposed to exporting it to Cloud Storage.