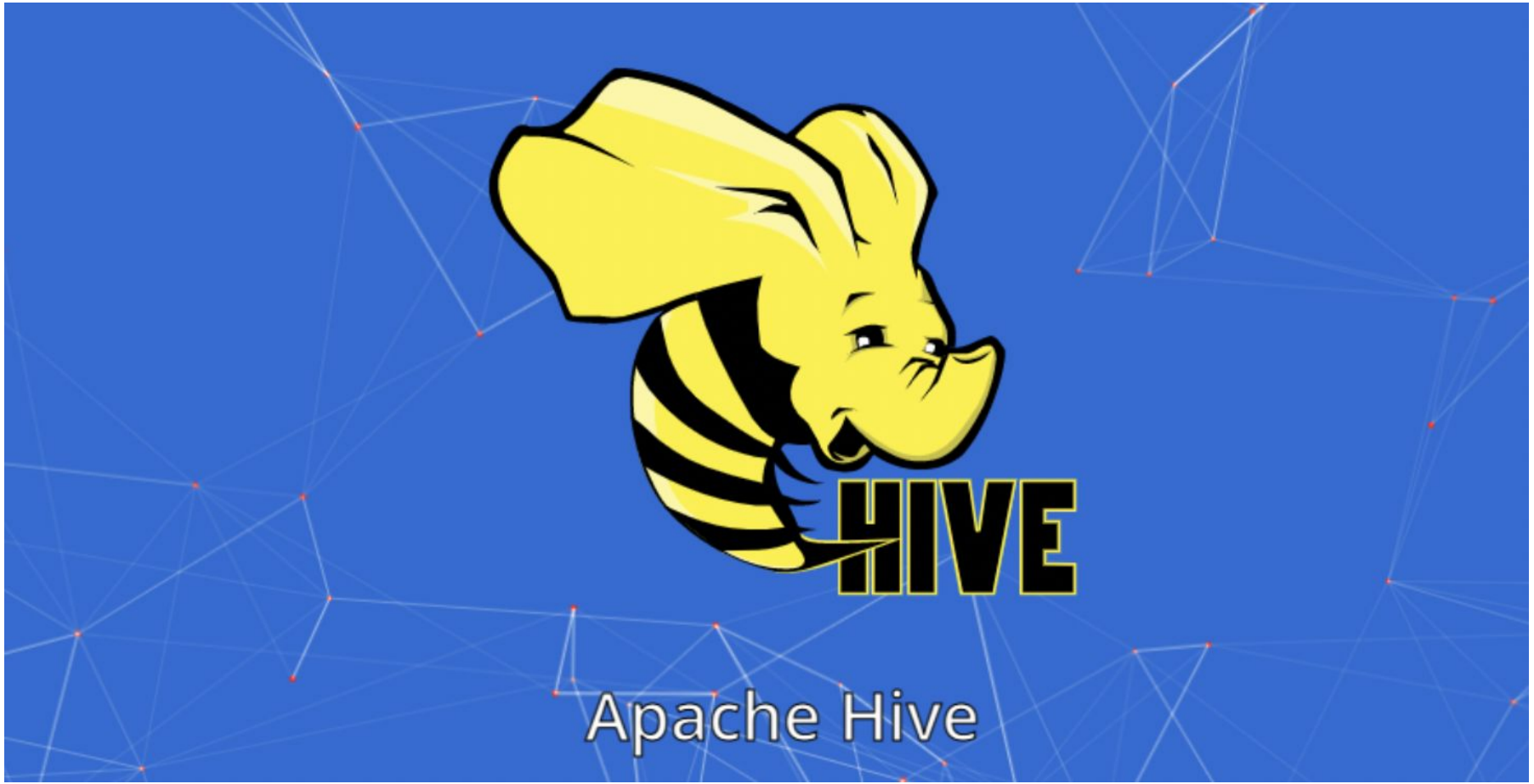
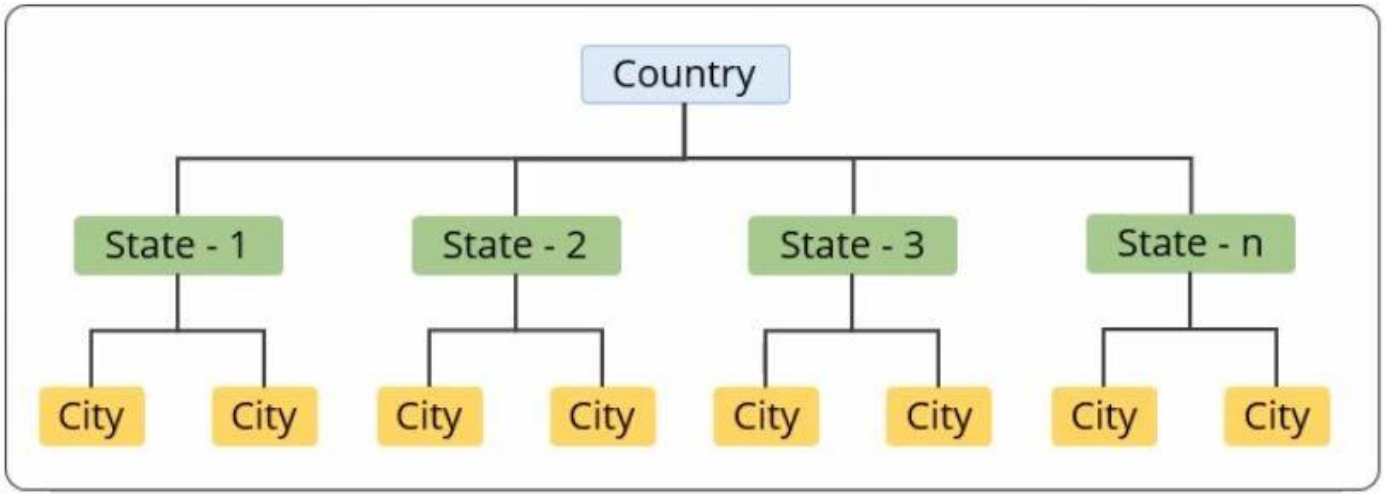

HIVE
Apache Hive

Partitioning in Hive

GROW DATA SKILLS

Apache Hive organizes tables into partitions. Partitioning is a way of dividing a table into related parts based on the values of particular columns like date, city, and department.

Each table in the hive can have one or more partition keys to identify a particular partition. Using partition it is easy to do queries on slices of the data.



Why is Partitioning Important?

- ***Speeds Up Data Query:*** Partitioning reduces data search space for queries, speeding up data retrieval.
- ***Reduces I/O Operations:*** Only relevant data partitions are scanned, reducing unnecessary I/O operations.
- ***Improves Query Performance:*** By limiting data read, partitioning boosts query performance.
- ***Saves Resources:*** Querying only relevant partitions uses fewer computational resources.
- ***Manages Large Data Sets:*** Helps handle large datasets by dividing them into smaller, manageable parts.
- ***Filter Data Efficiently:*** Speeds up queries that commonly filter by certain columns.
- ***Enables Scalability:*** As data grows, new partitions can be added without degradation in performance.
- ***Data Management and Archiving:*** Makes it easier to archive or delete data based on time or other attributes.

How to create Partitioned table

```
CREATE TABLE zipcodes
(
        RecordNumber int,
        Country string,
        City string,
        Zipcode int
)
PARTITIONED BY(state string)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ',';
```

Types of Hive Partitioning

1. Static Partitioning

- Insert input data **files individually** into a partition table is Static Partition.
- Usually when loading files **(big files)** into Hive tables static partitions are preferred.
- Static Partition **saves your time** in loading data compared to dynamic partition.
- You "**statically**" add a partition in the table and move the file into the partition of the table.
- We can **alter** the partition in the static partition.
- Static partition is the default partition strategy in hive
- Static partition is in **Strict Mode.**
- You should use where clause to use limit in the static partition.
- You can perform Static partition on Hive **Manage** table or **External** table.

Syntax to load data in Static Partitioned Table:

```
LOAD DATA INPATH '/hdfs/path/to/datafile' INTO TABLE employees PARTITION (year='2023');
```

OR

```
INSERT OVERWRITE TABLE employees PARTITION (year='2023') SELECT name, age FROM emp_data
WHERE year = '2023';
```

2. Dynamic Partitioning

- **Single insert** to partition table is known as a dynamic partition.
- Usually, dynamic partition loads the data from the **non-partitioned table**.
- Dynamic Partition **takes more time** in loading data compared to static partition.
- When you have large data stored in a table then the Dynamic partition is suitable.
- If you want to partition a number of columns but you don't know how many columns then also dynamic partition is suitable.
- Dynamic partition there is no required where clause to use limit.
- We **can't perform alter** on the Dynamic partition.
- You can perform dynamic partition on hive **external** table and **managed** table.
- If you want to use the Dynamic partition in the hive then the mode is in non-strict mode.
    - SET hive.exec.dynamic.partition = true;
    - SET hive.exec.dynamic.partition.mode = nonstrict;

Syntax to load data in Dynamic Partitioned Table:

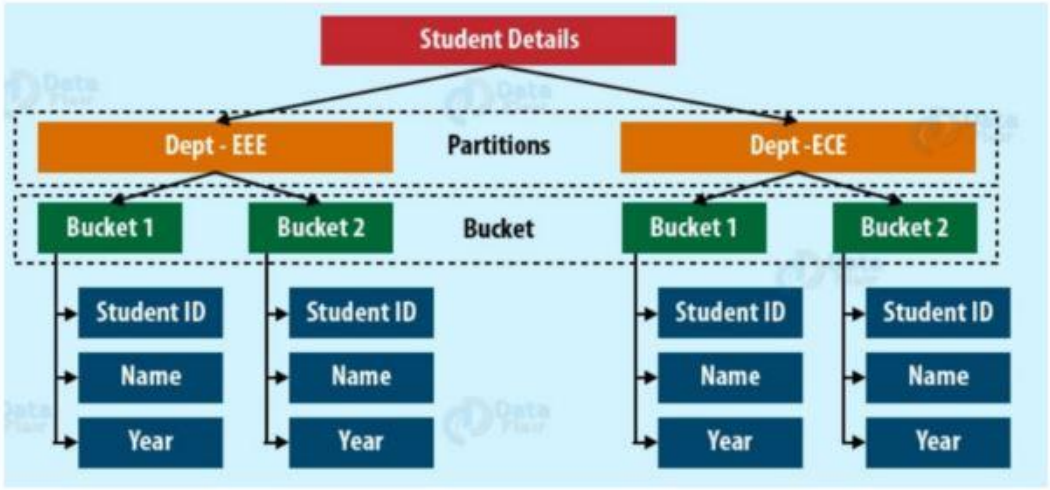
INSERT OVERWRITE TABLE employees PARTITION (year) SELECT name, age, year FROM emp_data;

Bucketing in Hive

You've seen that partitioning gives results by segregating HIVE table data into multiple files only when there is a limited number of partitions, what if partitioning the tables results in a **large number of partitions**. This is where the concept of **bucketing** comes in.

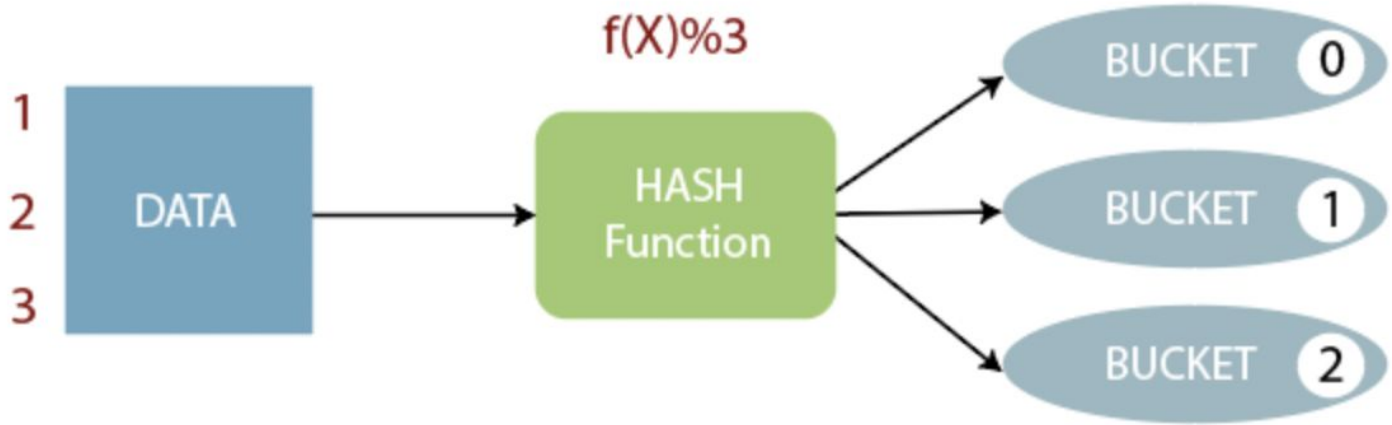
When a column has a high cardinality, we can't perform partitioning on it. A very high number of partitions will generate **too many Hadoop files** which would increase the load on the node. That's because the node will have to keep the metadata of every partition, and that would affect the performance of that node

In simple words, You can use bucketing if you need to run queries on columns that have huge data, which makes it difficult to create partitions.



Bucketing in Hive

- The concept of bucketing is based on the hashing technique.
- modules of current column value and the number of required buckets is calculated (let say, F(x) % 3).
- Based on the resulted value, the data stored into the corresponding bucket.
- The Records with the same bucketed column stored in the same bucket
- This function requires you to use the Clustered By clause to divide a table into buckets.



Syntax To Create Buckets

```
CREATE TABLE employees_bucketed (
    employee_id INT,
    first_name STRING,
    last_name STRING,
    salary FLOAT,
    department_id INT)
CLUSTERED BY (employee_id) INTO 4 BUCKETS
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ','
STORED AS TEXTFILE;
```

Difference between Partitioning and Bucketing

GROW DATA SKILLS

Features	Partitioning	Bucketing
Purpose	To segregate and organize data based on column values.	To distribute data evenly across a fixed number of files (buckets).
Storage	Each partition is stored as a separate subdirectory in HDFS.	All buckets are stored together in the same directory.
Use Case	When column values are limited and frequently queried on.	When dealing with high cardinality columns or skewed data.
Query Efficiency	Queries are faster as irrelevant partitions can be skipped.	Allows efficient data sampling and optimized join operations.
Data Management	Can manage data efficiently, e.g., archiving or deleting data is easy.	Handles large number of unique values better by evenly distributing data.
Syntax	`` `PARTITIONED BY (column_name)` `` in the `` `CREATE TABLE` `` statement.	`` `CLUSTERED BY (column_name) INTO num_buckets BUCKETS` `` in the `` `CREATE TABLE` `` statement.

Partitioning and Bucketing Together

```
CREATE TABLE employees (
    employee_id INT,
    first_name STRING,
    last_name STRING,
    salary FLOAT)
PARTITIONED BY (department_id INT)
CLUSTERED BY (employee_id) INTO 4 BUCKETS
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ','
STORED AS TEXTFILE;
```

Partitioning and bucketing are not dependent on each other, but they can be used together to improve query performance and data management. They serve different purposes and can coexist to complement each other.

Using partitioning and bucketing together allows Hive to prune data at the partition level and further organize data within a partition by distributing it into buckets.

Benefits Of Partitioning

- **Filtering**: If queries often filter data based on a certain column, partitioning on that column can significantly reduce the amount of data read, thus improving performance. For example, if a table is partitioned by date and queries frequently request data from a specific date, partitioning can speed up these queries.

- **Aggregation**: If you're aggregating data based on the partition column, partitioning can optimize these operations by reducing the amount of data Hive needs to read.

- **Data Management**: Partitioning helps manage and organize data better. It can be useful for removing or archiving data efficiently. For example, you can quickly drop a partition to delete data for a specific date.

Benefits Of Bucketing

- **Sampling**: Bucketing allows efficient sampling of data. Since each bucket essentially represents a sample of data, you can quickly get a sample by querying a single bucket.

- **Join Operations**: Bucketing can be used to perform more efficient map-side joins when joining on the bucketed column. If two tables are bucketed on the join columns and are of similar size, Hive can perform a bucketed map join, which is much faster than a regular join.

- **Handling Skew**: If data is skewed (i.e., some values appear very frequently), bucketing can distribute the data more evenly across files, improving query performance.

Map Side Join (Broadcast Join) in Hive

GROW DATA SKILLS

A Map Side Join (also known as Map Join) is an optimized version of the join operation in Hive, where one table is small enough to fit into memory. This smaller table (also known as the dimension table) is loaded into memory, and the larger table (also known as the fact table) is read line by line. Because the join operation occurs at the map phase and doesn't need a reduce phase, it's much faster than a traditional join operation.

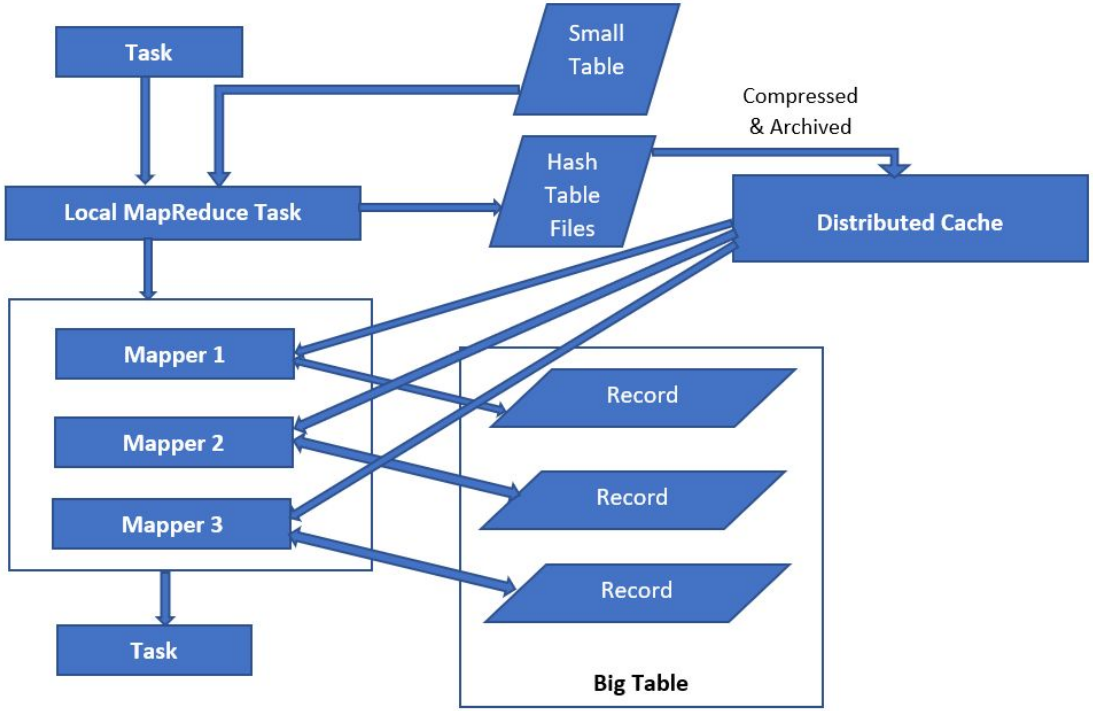
To use a Map Join, the following properties need to be set:

- **hive.auto.convert.join:** This property should be set to true. It allows Hive to automatically convert a common join into a Map Join based on the sizes of the tables.

- **hive.mapjoin.smalltable.filesize:** This property sets the maximum size for the small table that can be loaded into memory. If the size of the table exceeds the value set by this property, Hive won't perform a Map Join. The default value is 25000000 bytes (approximately 25MB).

  ```
  SET hive.auto.convert.join=true;
  SET hive.mapjoin.smalltable.filesize=50000000;  // setting limit to 50MB
  ```

Also, keep in mind that if your join operation includes more than two tables, the table in the last position of the FROM clause is considered the large table (fact table), and the other tables are considered small tables (dimension tables). This matters because Hive attempts to perform the Map Join operation using the last table as the fact table.



Bucket Map Join in Hive

GROW DATA SKILLS

A Bucket Map Join is an optimization of a Map Join in Hive where both tables are bucketed on the join columns. Instead of loading the entire small table into memory as done in a standard Map Join, the Bucket Map Join only needs to load the relevant bucket from the small table into memory, reducing memory usage and potentially allowing larger tables to be used in a Map Join.

To perform a Bucket Map Join, the following conditions need to be satisfied:

- Both tables should be bucketed on the join column.
- The number of buckets in the large table should be a multiple of the number of buckets in the small table.

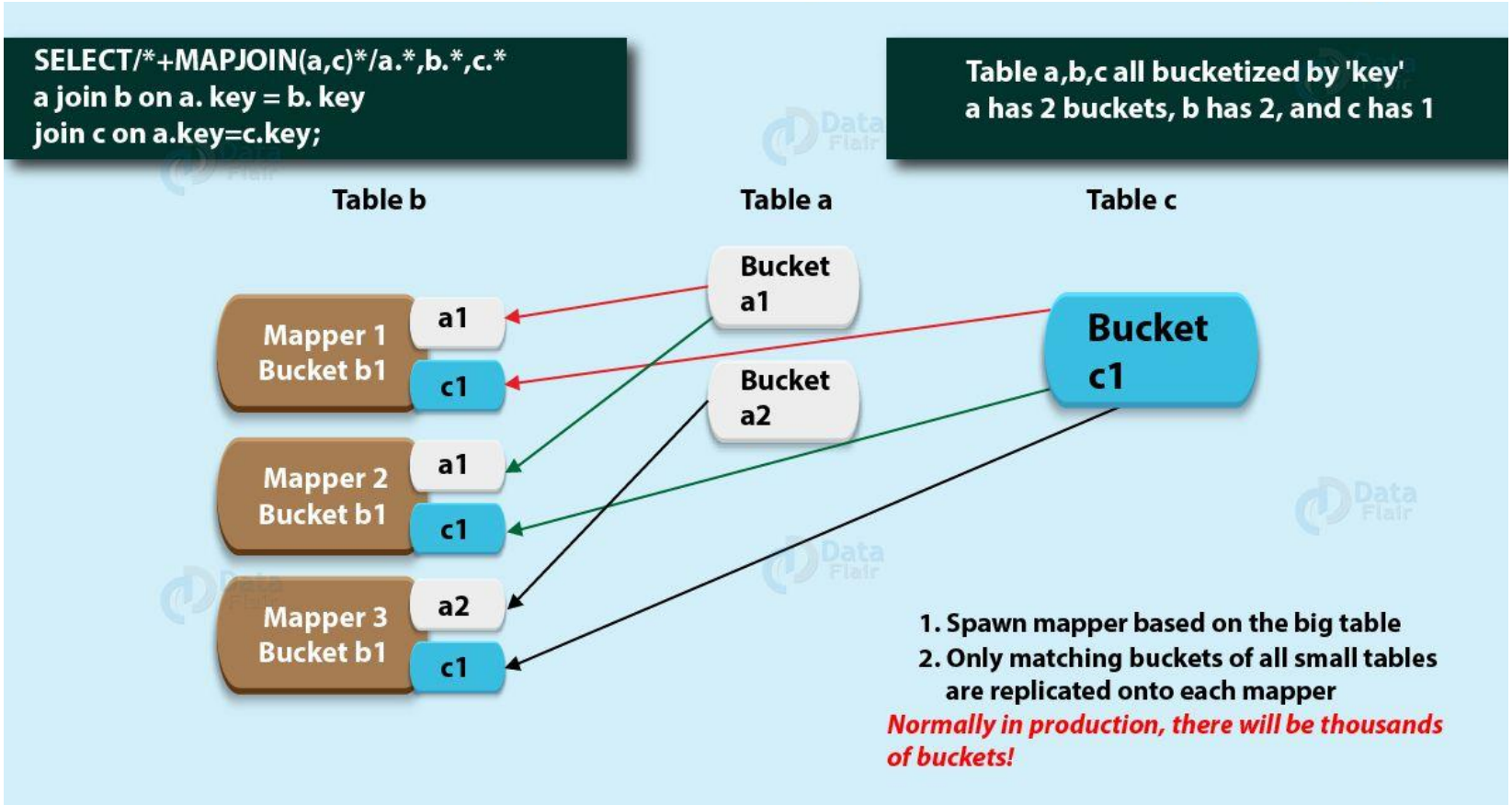
To enable Bucket Map Joins, the following properties need to be set:

- **hive.auto.convert.join**: This property should be set to true. It allows Hive to automatically convert a common join into a Map Join based on the sizes of the tables.

- **hive.optimize.bucketmapjoin**: This property should be set to true to allow Hive to convert common joins into Bucket Map Joins when possible.

  ```
  SET hive.auto.convert.join=true;
  SET hive.optimize.bucketmapjoin=true;
  ```

Bucket Map Join in Hive



Sorted Merge Bucket Join in Hive

GROW DATA SKILLS

A Sorted Merge Bucket (SMB) Join in Hive is an optimization for bucketed tables where not only are the tables bucketed on the join column, but also sorted on the join column. This is similar to the bucket map join, but SMB join does not require one table to fit into memory, making it more scalable.

For a Sorted Merge Bucket Join, the following conditions need to be satisfied:

- Both tables should be bucketed and sorted on the join column.
- Both tables should have the same number of buckets.

When these conditions are satisfied, each mapper can read a bucket from each table at a time and perform the join, significantly reducing the disk I/O operations.

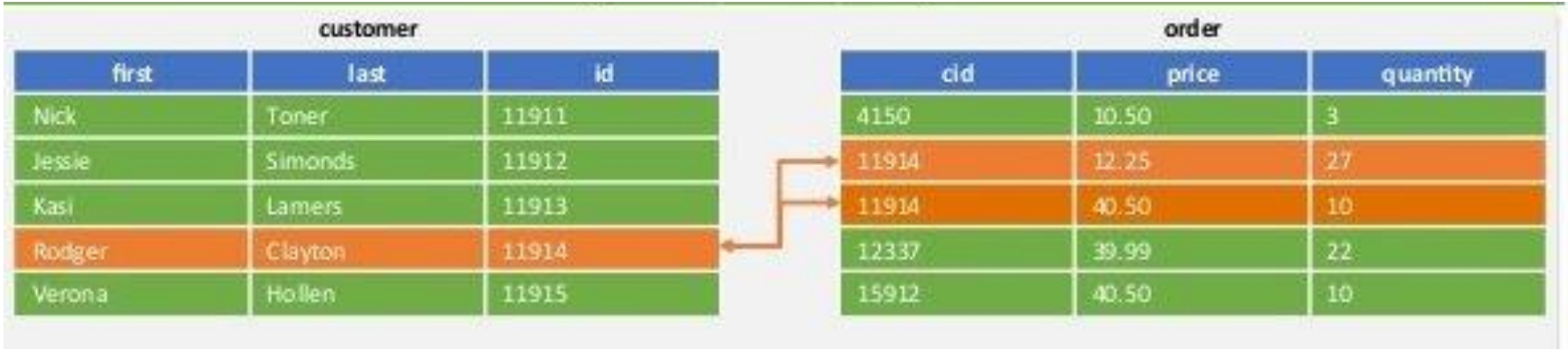
To enable SMB Joins, the following properties need to be set:

- **hive.auto.convert.sortmerge.join**: This property should be set to true. It allows Hive to automatically convert common joins into Sorted Merge Bucket Joins when possible.

- **hive.optimize.bucketmapjoin.sortedmerge**: This property should be set to true to allow Hive to perform a SMB Join.

Sorted Merge Bucket Join in Hive

```
SET hive.auto.convert.sortmerge.join=true;
SET hive.optimize.bucketmapjoin.sortedmerge=true;
```



customer

first	last	id
Nick	Toner	11911
Jessie	Simonds	11912
Kasi	Lamers	11913
Rodger	Clayton	11914
Verona	Hollen	11915

order

cid	price	quantity
4150	10.50	3
11914	12.25	27
11914	40.50	10
12337	39.99	22
15912	40.50	10

Skew Join in Hive

A Skew Join in Hive is an optimization technique for handling skewed data, where some values appear very frequently compared to others in the dataset. In a typical MapReduce job, skewed data can lead to a few reducers taking much longer to complete than others because they process a majority of the data. This can negatively impact the overall performance of the join.

A Skew Join in Hive tries to handle this problem by performing the join in two stages:

- In the first stage, Hive identifies the skewed keys and processes all the non-skewed keys.
- In the second stage, Hive processes the skewed keys. The skewed keys are partitioned into different reducers based on a hash function, thus reducing the burden on a single reducer.

To perform a Skew Join, the following conditions need to be satisfied:

- The join should be a two-table join. Currently, Hive does not support multi-table skew joins.
- There should be skew in key distribution. If the key distribution is uniform, a skew join may not provide any advantage and can be less efficient than a regular join.

Skew Join in Hive

To enable Skew Joins, the following properties need to be set:

- hive.optimize.skewjoin: This property should be set to true. It enables the skew join optimization.
- hive.skewjoin.key: This property sets the minimum number of rows for a key to be considered skewed. The default value is 10000.

```
SET hive.optimize.skewjoin=true;
SET hive.skewjoin.key=50000;  // setting limit to 50,000
```