Hive & SQL
Session 3

BUCKETING VS PARTITIONING

•Partitioning helps in elimination of data, if used in WHERE clause, where as bucketing helps in organizing data in each partition into multiple files, so that the same set of data is always written in same bucket.

•Bucketing helps a lot in joining of columns.

•Hive Bucket is nothing but another technique of decomposing data or decreasing the data into more manageable parts or equal parts.

•Partitioning a table stores data in sub-directories categorized by table location, which allows Hive to exclude unnecessary data from queries without reading all the data every time a new query is made.

•Hive does support Dynamic Partitioning (DP) where column values are only known at EXECUTION TIME. To enable Dynamic Partitioning :

SET hive.exec.dynamic.partition =true;

•Another situation we want to protect against dynamic partition insert is that the user may accidentally specify all partitions to be dynamic partitions without specifying one static partition, while the original intention is to just overwrite the sub-partitions of one root partition.

SET hive.exec.dynamic.partition.mode =strict;

To enable bucketing:

SET hive.enforce.bucketing =true;

ORC FORMAT

ORC stands for Optimized Row Columnar which means it can store data in an optimized way than the other file formats. ORC reduces the size of the original data up to 75%(eg: 100GB file will become 25GB). As a result the speed of data processing also increases. ORC shows better performance than Text, Sequence and RC file formats.
An ORC file contains rows data in groups called as Stripes along with a file footer. ORC format improves the performance when Hive is processing the data.

An ORC file contains groups of row data called **stripes**, along with auxiliary information in a **file footer**. At the end of the file a **postscript** holds compression parameters and the size of the compressed footer.

The default stripe size is 250 MB. Large stripe sizes enable large, efficient reads from HDFS.

The file footer contains a list of stripes in the file, the number of rows per stripe, and each column's data type. It also contains column-level aggregates count, min, max, and sum.

This diagram illustrates the ORC file structure:

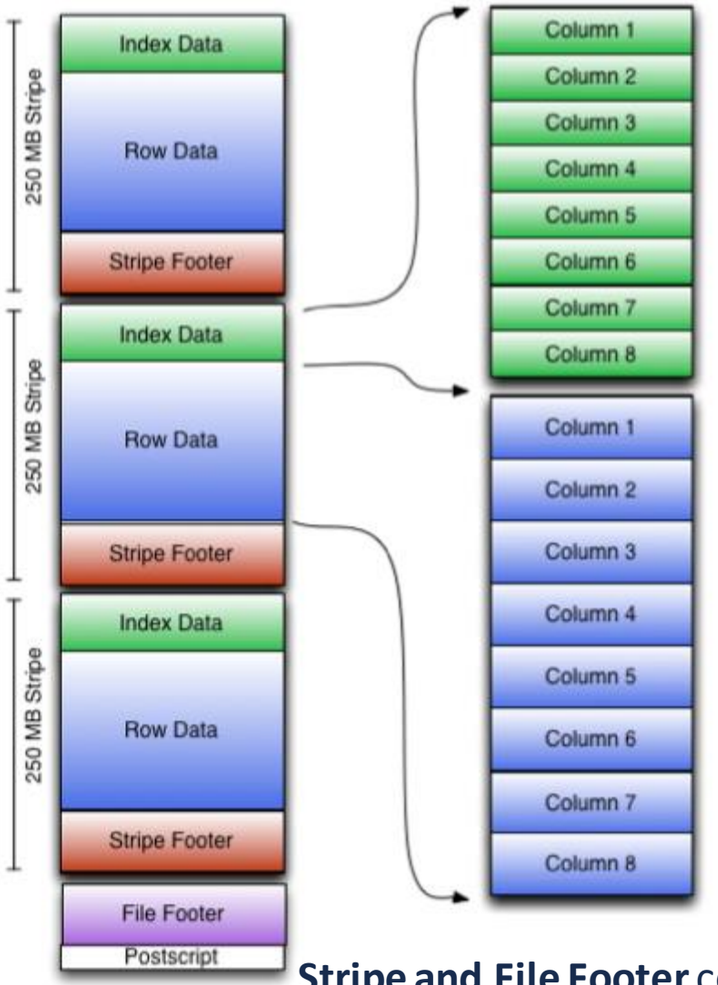
PARQUET FORMAT

Parquet is an open source file format available to any project in the Hadoop ecosystem. Apache Parquet is designed for efficient as well as performant flat columnar storage format of data compared to row based files like CSV or TSV files.

Columnar storage like Apache Parquet is designed to bring efficiency compared to row-based files like CSV. When querying, columnar storage you can skip over the non-relevant data very quickly. As a result, aggregation queries are less time consuming compared to row-oriented databases. This way of storage has translated into hardware savings and minimized latency for accessing data.

Compression ratio is 60-70%.

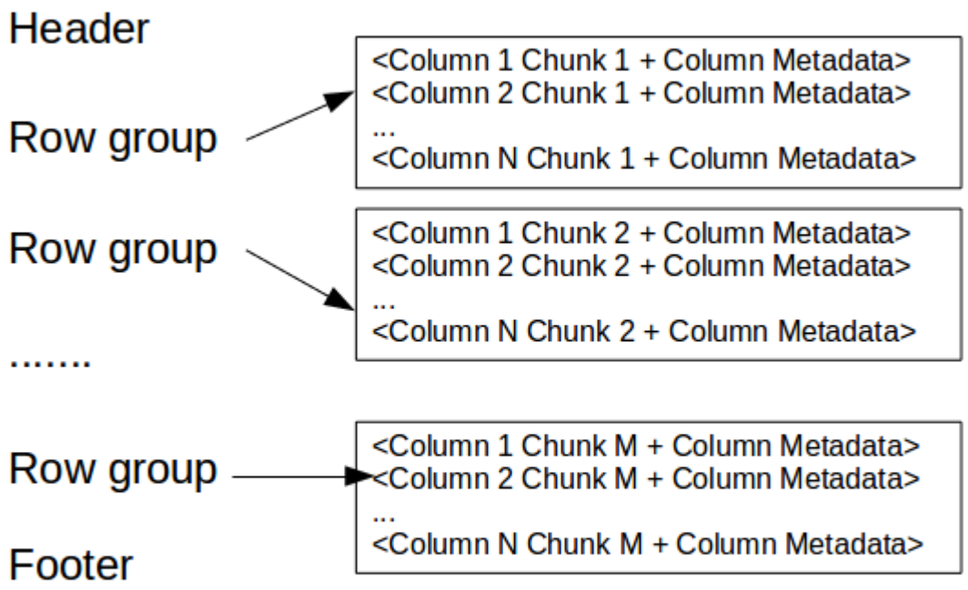
ORC



Index data includes min and max values for each column and the row positions within each column

Raw Data
the row values

PARQUET



Stripe and File Footer contains the list of stripes(Files footer) in the file, the number of rows per stripe, and each column's data type. It also contains column-level aggregates count, min, max, and sum at respective levels.

AVRO FORMAT

-.avro

-Compression ratio 50-55%

-Schema is stored separately in avsc (avro schema) file

-Schema evolution advantage – Even if one column data doesn't appear automatically hive will default it to some value.

-The default value is specified in the avsc file.

Name , address, phone

1,blr,99168

2,Noida, 9999

3,nocityfound,9777

JOINS AND TYPES

Type	Approach	Pros	Cons
Shuffle Join	Join keys are shuffled using map/reduce and joins performed join side.	Works regardless of data size or layout.	Most resource-intensive and slowest join type.
Broadcast Join	Small tables are loaded into memory in all nodes, mapper scans through the large table and joins.	Very fast, single scan through largest table.	All but one table must be small enough to fit in RAM.
Sort-Merge-Bucket Join	Mappers take advantage of co-location of keys to do efficient joins.	Very fast for tables of any size.	Data must be sorted and bucketed ahead of time.

Shuffle Joins – the default

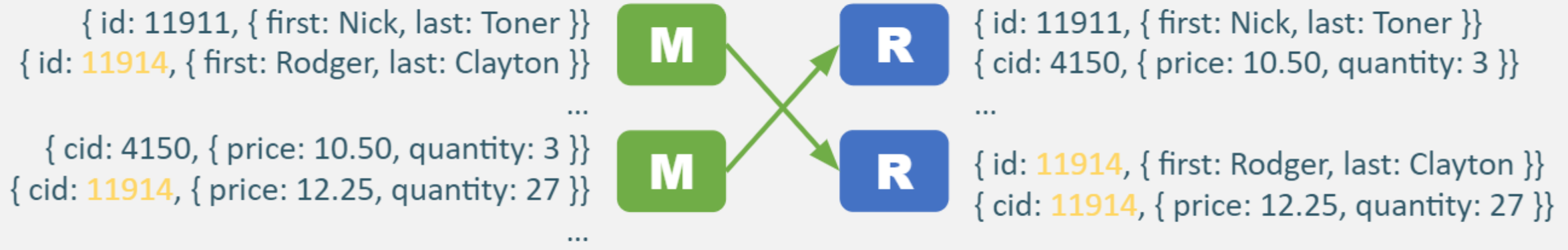
customer

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
3491	5.99	5
2934	39.99	22
11914	40.50	10

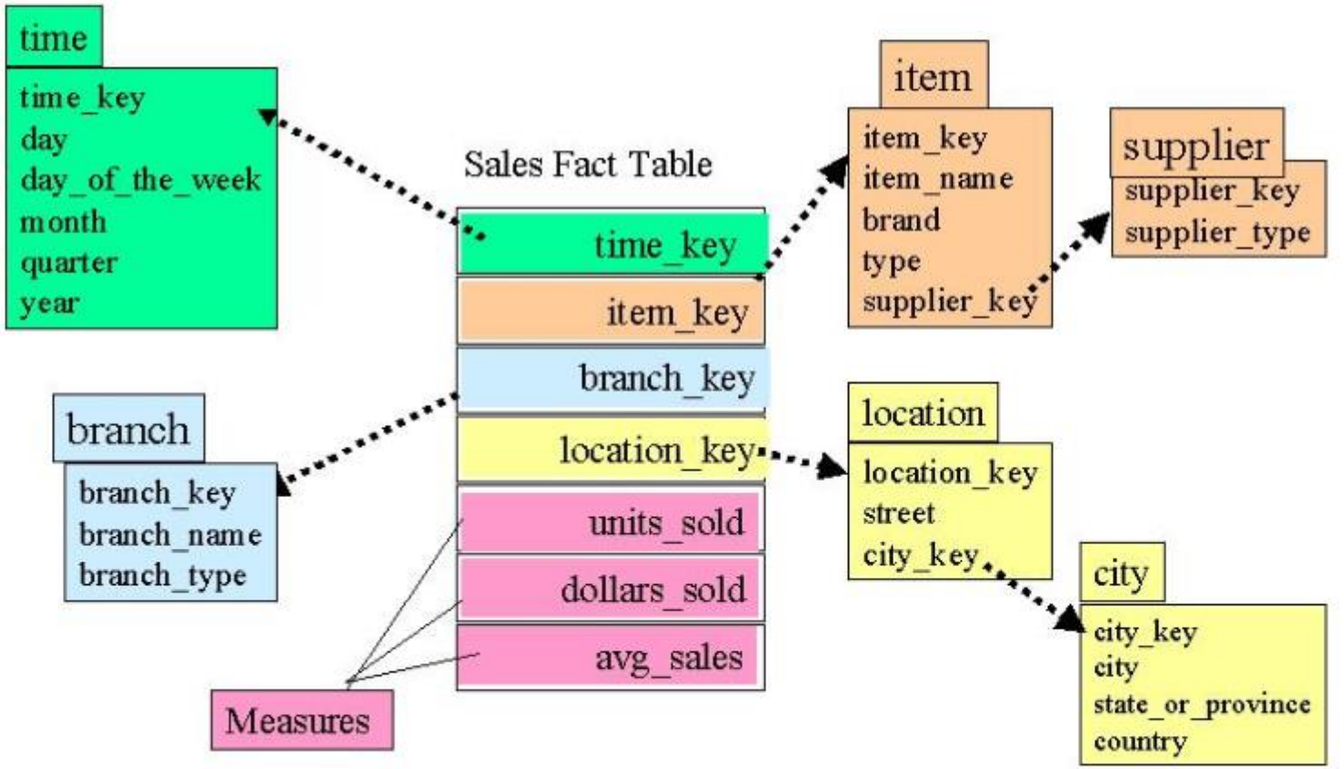
```
SELECT * FROM customer join order ON customer.id = order.cid;
```



Identical keys shuffled to the same reducer. Join done reduce-side.
Expensive from a network utilization standpoint.

Broadcast Join (aka Map-side Join)

- Star schemas (e.g. dimension tables)
- Good when table is small enough to fit in RAM



USING BROADCAST JOIN

Set hive.auto.convert.join = true

HIVE then automatically uses broadcast join, if possible

- Small tables held in memory by all nodes
- Single pass through the large table

Used for star-schema type joins common in Data warehousing use-cases

hive.auto.convert.join.noconditionaltask.size (by default 10 MB) determines data size for automatic conversion to broadcast join:

- Default 10MB is too low (check your default)
- Recommended: 256MB for 4GB container

Sort-Merge-Bucket join:
When both are too large for memory

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

```
CREATE TABLE order (cid int, price float, quantity int)
CLUSTERED BY(cid) SORTED BY(cid) INTO 32 BUCKETS;

CREATE TABLE customer (id int, first string, last string)
CLUSTERED BY(id) SORTED BY(id) INTO 32 BUCKETS;

SELECT * FROM customer join order ON customer.id = order.cid;
```