Hive

Apache Hive is a data WH system that is built on top of hadoop.

→ data analysts
→ SQL developers

spark sql

Ease working on data
no Java or MR code

Facebook developed Hive ≈ 2010

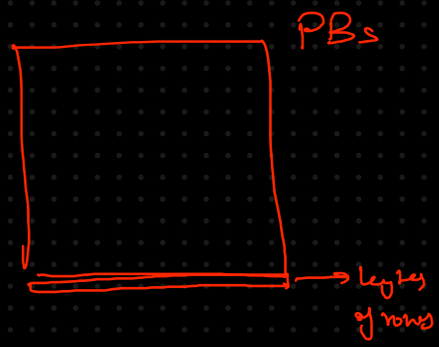
SQL like interface

HQL

PBs of data

Library =

librarian with a ipad / computer



How Hive makes data processing easier:

Without hive	With Hive
Java based MR code to process data	SQL like queries
Programmers	SQL users Analyst

Few Common Ques/misconceptions about Hive

Misconception

1. Hive is a database like SQL / Postgres SQL ?

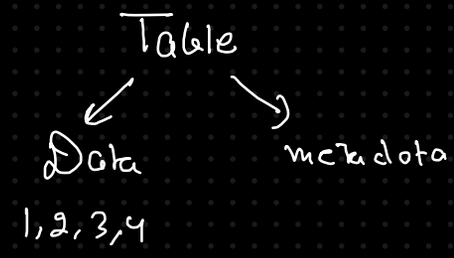
Hive is not a database but a data warehouse tool build on Hadoop.

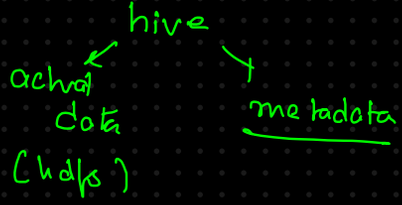
an interface to Query large datasets stored in hdfs

Feature	Traditional RDBMS (MySQL, PostgreSQL)	Apache Hive
Storage	Stores structured data in tables on disk	Uses **HDFS** for distributed storage
Query Execution	Uses in-memory processing	Uses **MapReduce, Tez, or Spark**
Transaction Type	OLTP (Row-level operations)	OLAP (Batch processing)

Feature	Traditional Databases (MySQL, PostgreSQL)	Apache Hive
Storage	Stores data in structured tables on disks	Stores data in **HDFS** (distributed storage)
Query Processing	Executes queries in-memory	Uses **MapReduce, Tez, or Spark** for distributed processing
Schema	Schema-on-Write (Data must fit schema)	Schema-on-Read (Can query raw files)
Transaction Type	OLTP (Transactional systems)	OLAP (Batch processing, analytics)
Speed	Fast for small queries, real-time updates	Optimized for large-scale batch processing

Hive is a metastore



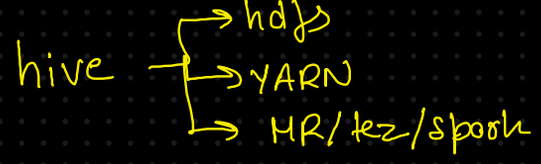


In big data world, any tech which supports data Querying not necessarily a dB

2. Hive is a replacement for Hadoop

built on top of hadoop & depends on hdfs, YARN

Query engine




```
Hive Session ID = 61b9b5ae-7609-498c-a06d-4fa12313a968
hive> set hive.execution.engine
    > ;
hive.execution.engine=tez
hive> set hive.execution.engine = mr
    > ;
Hive-on-MR is deprecated in Hive 2 and may not be available in the future versions. Consider using a different execution eng
ine (i.e. spark, tez) or using Hive 1.X releases.
hive>
```

Why I teach things in depth ?

↳ Study nice or have doubts lifetime

end-to-end

MR ⟶ 3hrs

MR
Spark
Tez

3. Hive Queries run like normal SQL Queries DB

1st ⇒ abstraction of MR, Tez, Spark

~AI

select * from table ⟶ MR code

2nd distributed

4. Hive can perform Row level transactions like MySQL

⇒ update/delete

Hive as a technology
is designed for batch
Processing

Bazooka to kill an ant

→ it will not modify individual records, but create new partitions

5. Hive is always slower than Spark

Hive on MR ≃ slower

Hive on Spark/Tez ⇒ faster than MR

Hive is better for structured storage and
complex Queries

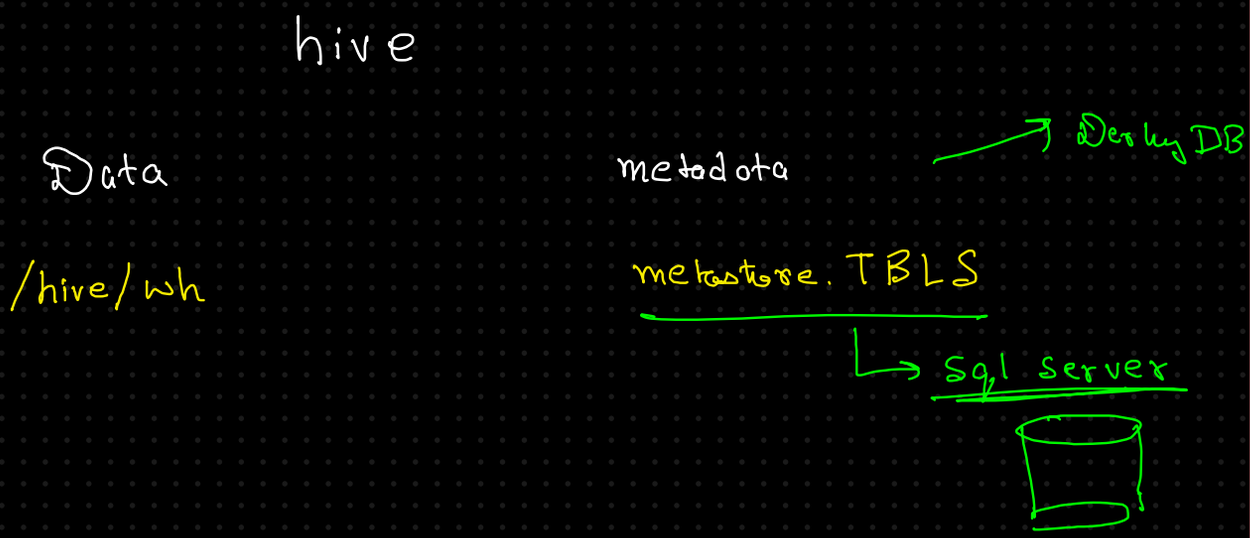
Use Case	Use Hive	Use Spark
Large-scale batch processing	✅ Yes	❌ No
Real-time analytics	❌ No	✅ Yes
ETL jobs on structured data	✅ Yes	❌ No
Streaming data processing	❌ No	✅ Yes

Hive only works with HDFS

s3, ADLS, GCS, hdfs

Feature	Hive Advantage
SQL-Like Interface	Allows non-programmers to work with big data.
Scalability	Handles **petabytes** of data.
Optimized Execution	Uses **MapReduce, Tez, or Spark** for distributed processing.
Storage Flexibility	Works with **HDFS, S3, Azure Blob, GCS**.
Schema Flexibility	Schema-on-read enables analyzing raw files.

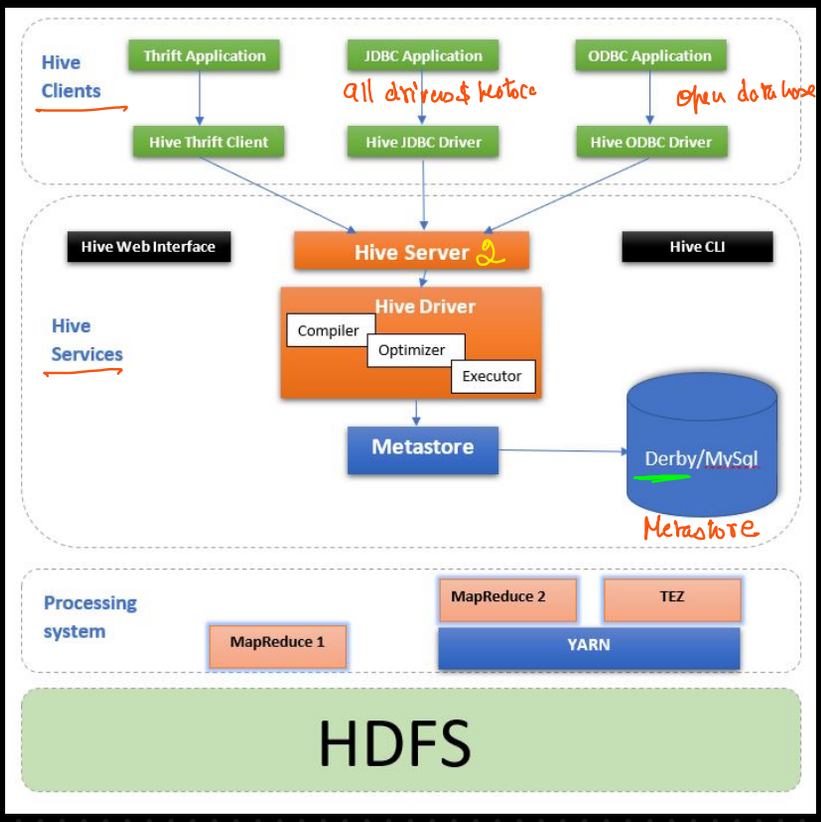
Feature	Hive CLI	Beeline (Recommended)
Connection Type	Directly connects to Hive	Uses JDBC for **remote** execution
Security	No authentication	Supports authentication (LDAP, Kerberos)
Multiple Connections	Single session only	Supports multiple concurrent sessions
Performance	More resource-heavy	Optimized for **query performance**
Recommended?	❌ Deprecated	✅ Yes, for production use



Hive Architecture

Hive is built on top of Hadoop & provides a SQL like interface for querying and managing the large datasets

Be ontime instead of udemy us this course please focus on yourself.



Beeline

! jdbc : 10000

Client Server

Front Backend

res flows request

Hive Clients : Allows user to connect/interact with Hive via commands or application

JDBC, ODBC, Thrift

Java application

Apache Thrift

Hive Server: - Acts as a middle layer between client & execution engine

manages client request, Sends them to execution & returns the result

Hive server 2 successor of 1

Driver : Control Query execution from start to end

Parsing Query
Communicate with Compiler, optimizer & execution engine

Compiler parses and Translate HiveQL → execution plan

Query → tasks

Optimizers Improves execution efficiency

Applies technique like predicate pushdown, join reordering

Execution engine run the execution plan using Hadoop framework

MR, Tez, Spark
⇓
default

Metastore Store the metadata about databases, columns tables, partn schema

Essential for Hive to keep track of data

Select id, mobile num from customers

you haven't showed us on how to connect to the metastore this time as well to see the content inside Metastore

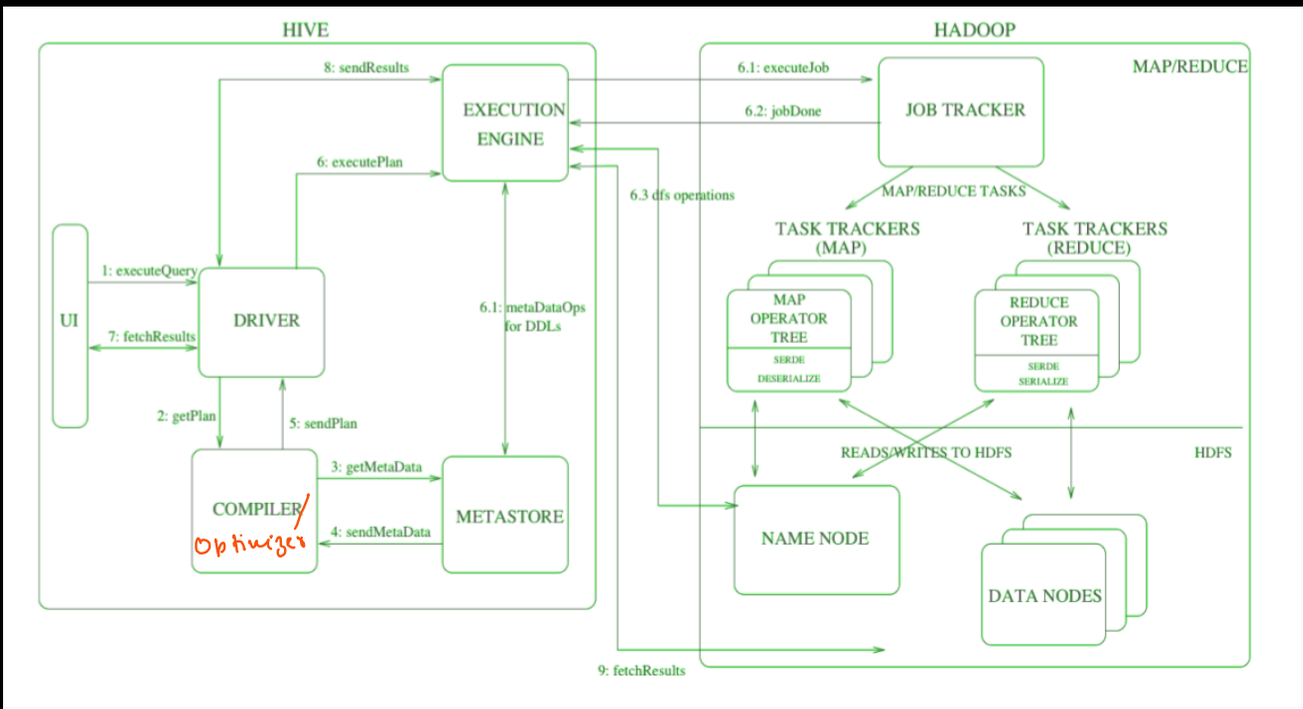
⇐ Revise Revise Revise

Storage Layer (hdfs) actual location where data resides

External → hdfs

managed → hive not on hdfs

Metastore → Remote → metastore outside and useful for non-java application
→ Embedded



1. User submits a Query via CLI, JDBC, ODBC, Thrift
2. Driver recieves the Query & forwards it to Compiler
3. Compiler checks for the syntax & retrieve metadata from metastore
4. Optimizer optimizes the Query by restructuring, predicate pushdown etc.
5. The execution engine determine the processing framework
6. Tasks are assigned to YARN or JT
7. HDFS data is read and processed via MR, Tez, Spark
8. Results are written back to hdfs & returned to user

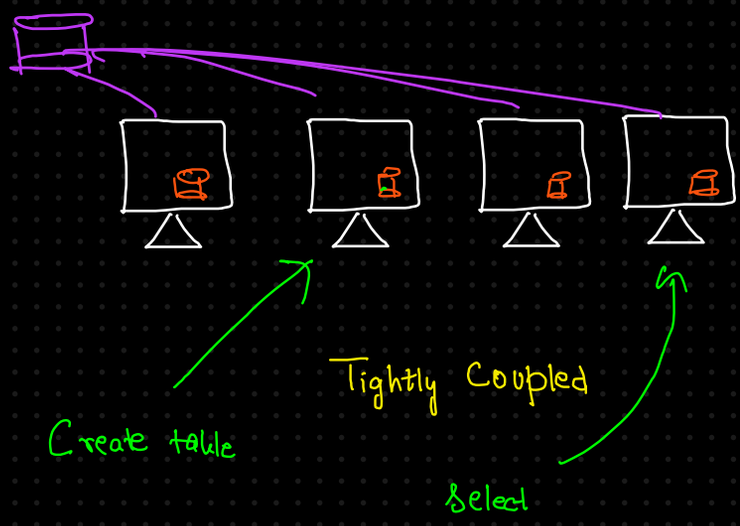
Derby DB

default dB in hive

tightly coupled

does not support Concurrent Operations

Limited Scalability



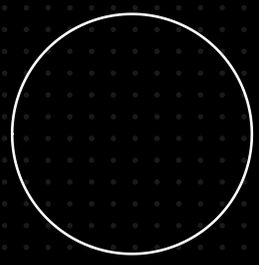
Aspect	Derby DB (Local Mode)	MySQL (Remote Mode)
What is it?	An embedded relational database (RPD) that comes bundled with Hive.	A standalone relational database server used for Hive metadata.
Mode of Operation	**Local mode (Embedded):** Stores metadata on the same machine as Hive.	**Remote mode:** Stores metadata on a dedicated MySQL server.
Multi-User Access	❌ No (Single-user only, since it locks the database for a single process).	✅ Yes (Multiple users can access Hive concurrently).
Metadata Sharing	❌ Cannot share metadata across nodes (limited to local machine).	✅ Shared and accessible across the entire Hadoop cluster.
Use Case	Suitable for local testing or single-user development environments.	Suitable for production environments with multiple users and nodes.
Scalability	❌ Poor (Not suitable for large-scale deployments).	✅ High (Handles large-scale metadata operations).
Performance	Slower for large-scale queries due to local storage.	Faster due to optimized querying and indexing.
Persistence	Metadata is stored locally and may be lost if the local system fails.	Metadata is stored centrally and is more durable.

Hive External vs Managed Table

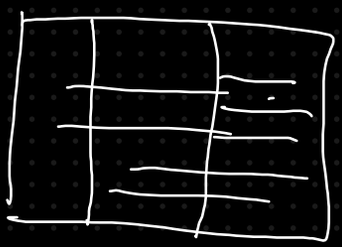
Aspect	Internal (Managed) Table	External Table
Storage Location	Data is stored in Hive's default warehouse directory (`/user/hive/warehouse/`).	Data is stored at the user-specified HDFS path.
Data Management	Hive **manages** both the table **schema and data**.	Hive **manages only the schema**, not the data.
Data Deletion	**Dropping the table deletes the data permanently** from HDFS.	**Dropping the table deletes only the schema**, but the data remains in HDFS.
Use Case	Suitable for **temporary or intermediate tables**.	Preferred when **sharing datasets** across multiple systems.
LOCATION Clause	Not required (Hive defaults to the warehouse directory).	Required to specify the **external storage path**.
Table Property	`EXTERNAL=FALSE` (Default)	`EXTERNAL=TRUE` (Set explicitly or using `EXTERNAL` keyword during creation).

Hive Partitioning

+91 –



where customer_state = 'Mumbai' and id > 1000



dynamic

hive automatically creates the partitions based on column values.

Static

manually specifying the partition values

1st March

Hive Partitioning

Static Partitioning

Dynamic Partitioning

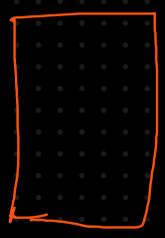
Multilevel partitioning → year → months

year → months

Bucketing in Hive

Query Optimization Techniques

Customer_id , Partitioning ?



1000000 rows

⇒ 1M folders

Worse

because we have metadata & other things to handle

1000000 → 100000

Cluster

Bucketing

Customer_id

⇓

10 buckets

⇓

hashing fn

⇓

files → for each bucket

hashmaps / dictionary

1 %

2 → 10 buckets → 1, 11, 121, 21 (files)

:

100000

Partitioning then bucketing

⇓

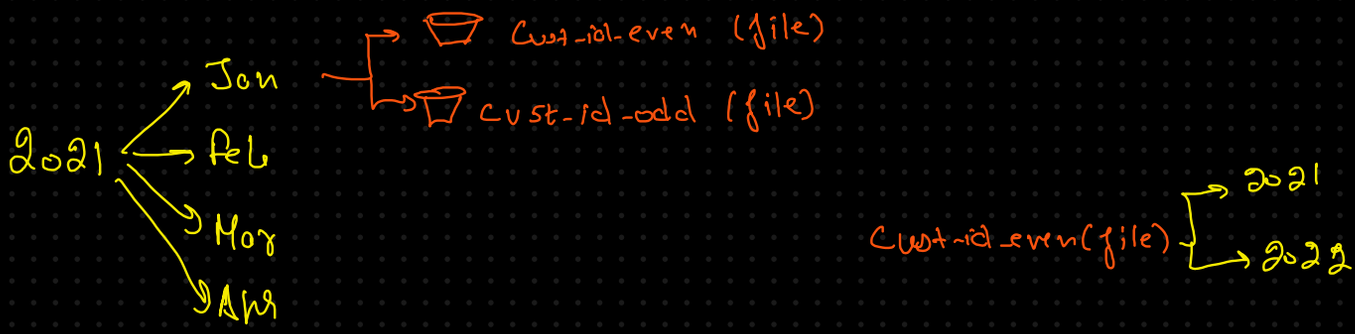
folders

1 ✓

Bucketing then partitioning

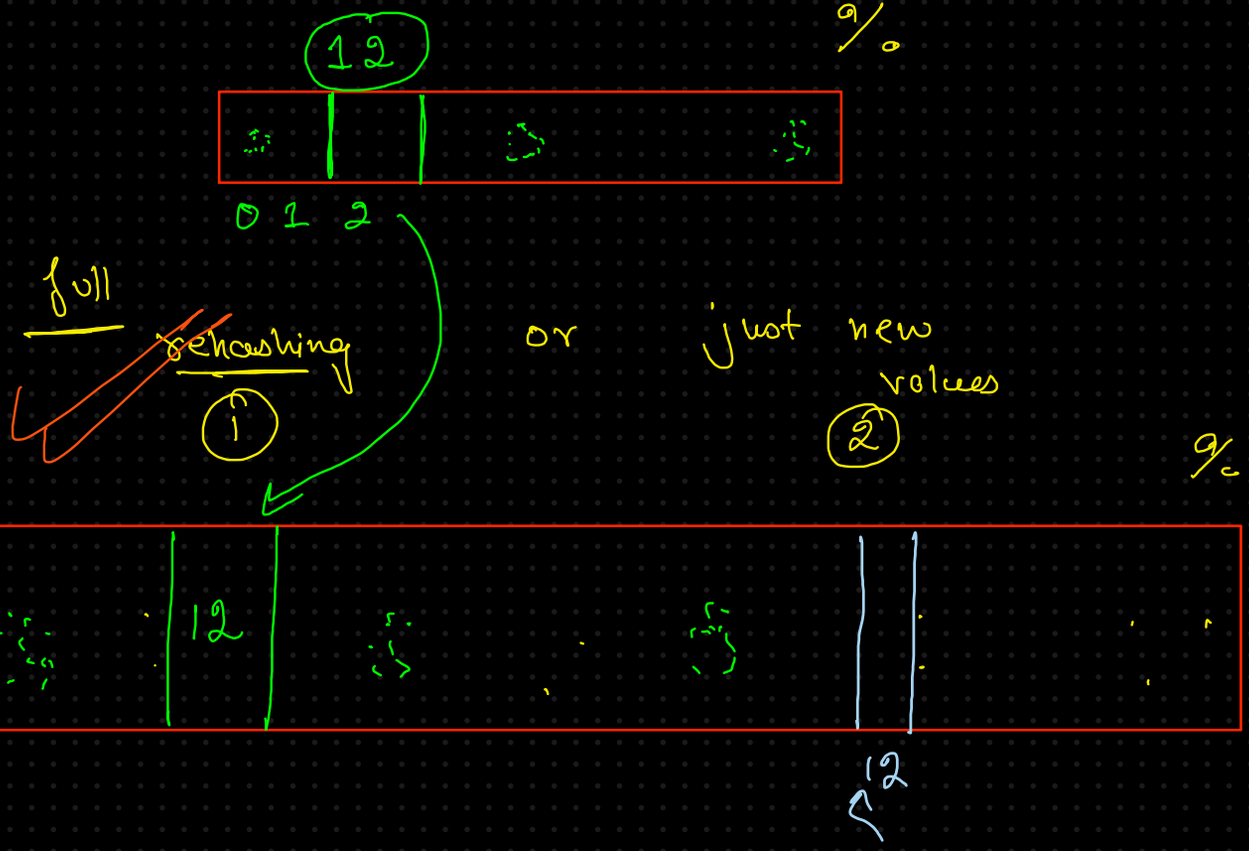
2

Not possible


2021
Jan
Feb
Mar
Apr
Cust_id_even (file)
Cust_id_odd (file)
Cust_id_even(file)
2021
2022

Bucketing vs Partitioning

Feature	Partitioning	Bucketing
Concept	Divides data into **separate folders** based on column values.	Divides data into **fixed number of files (buckets)** using hashing.
Column Type	Best for **categorical columns** with limited distinct values (e.g., `country`, `year`).	Best for **high-cardinality columns** (e.g., `customer_id`).
Data Storage	Each partition creates a **separate directory** in HDFS.	Each bucket creates a **fixed number of files** inside partitions or tables.
Query Performance	Faster queries when **filtering by partition column**.	Faster queries when **joining tables bucketed on the same column**.
Number of Partitions/Buckets	**Dynamic** (depends on unique values in the partition column).	**Fixed** (set during table creation).
Scalability	Can create **too many partitions** if unique values are high (bad for performance).	More scalable since **buckets are fixed** regardless of unique values.
Use Case	When filtering or querying data based on **a limited number of distinct values** (e.g., sales by country).	When **joining large tables efficiently** or querying **high-cardinality columns**.
Table Creation Syntax	`PARTITIONED BY (column_name)`	`CLUSTERED BY (column_name) INTO N BUCKETS`
Storage Overhead	Can create **too many small files** if partitioning on high-cardinality columns.	**Better storage balance** due to evenly distributed bucket files.
File Pruning Efficiency	Very efficient for **range queries** (`WHERE country='India'`).	Efficient for **joins and aggregations**.
Example Scenario	Sales reports by **year** (few partitions: 2021, 2022, 2023).	Customer transactions grouped by **customer_id** (hashed into buckets).



12 cust_id % 20 ⇒ 12

13 % 20 ⇒ 13

Hadoop → HDFS
→ MR
→ YARN

Spark → RDD, df, SQL
→ Caching
→ Memory mgmt
→ Optimization ← hive

Phase-1 Big data

Interview / Assignment

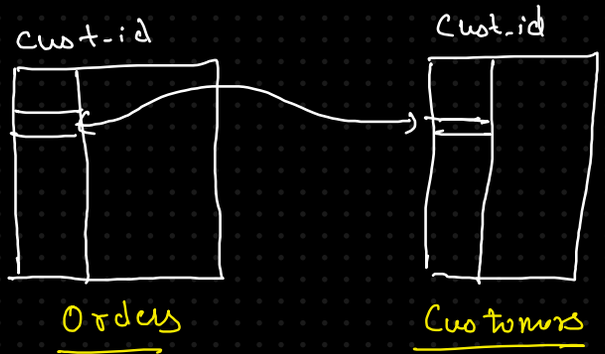
5k
→ Access to all courses on kN Academy

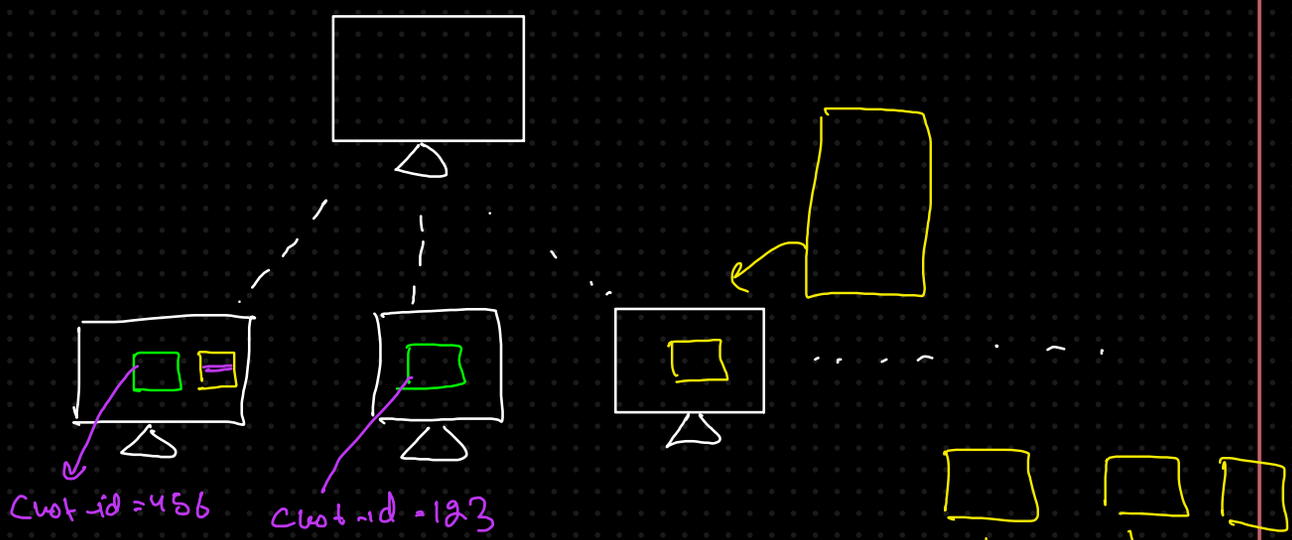
Azure
AWS

+

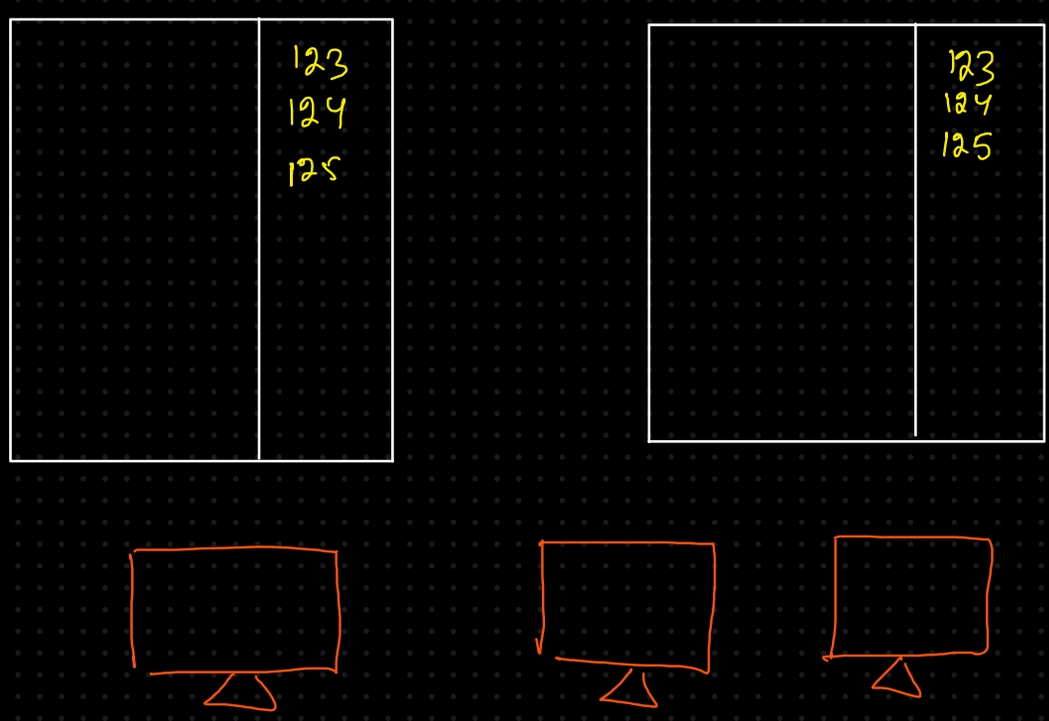
NoSQL DBs
Airflow
Kafka → Spark streaming

Joins



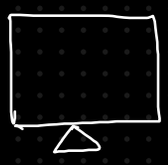


Sort and merge

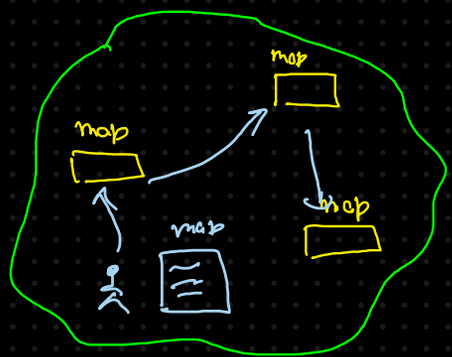


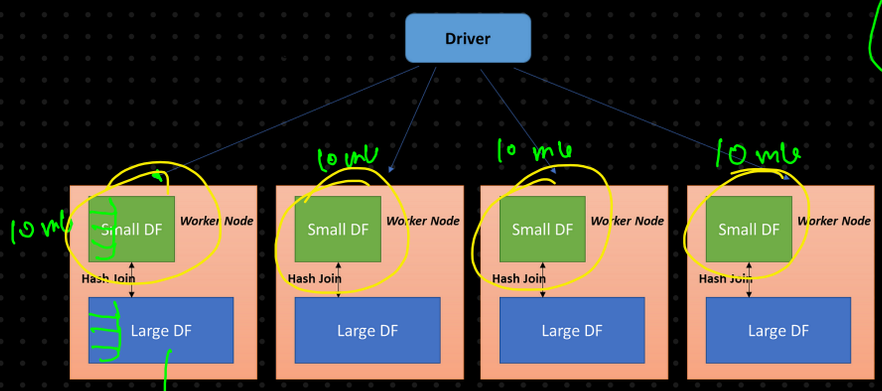
Broadcast Join (Map Side Join)

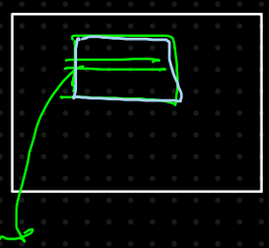
big table
$
a small table

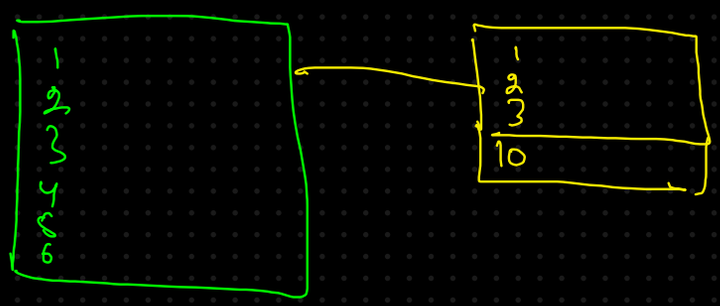


1gb table ⇒ distributed
10mb table ⇒ distributed







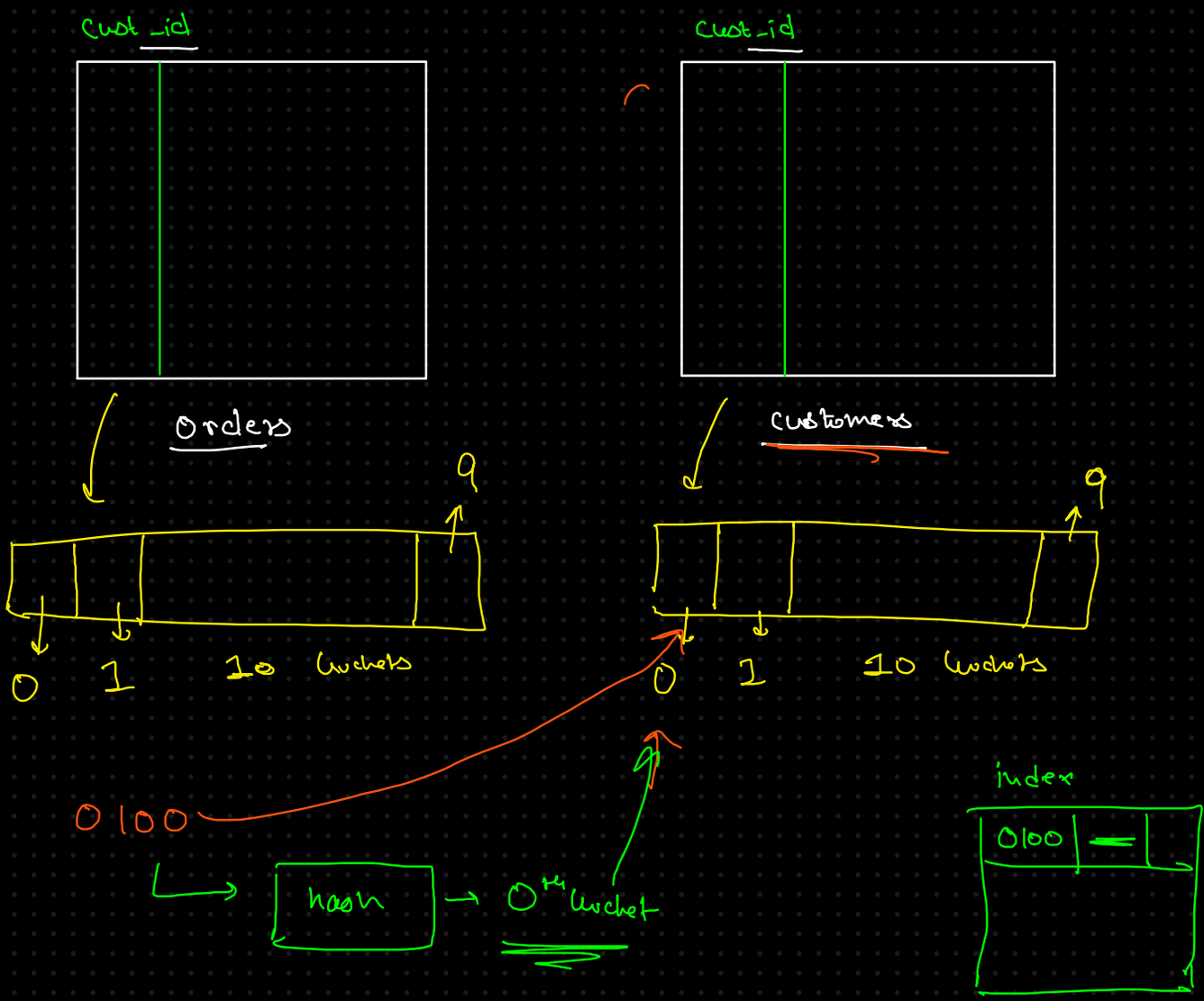


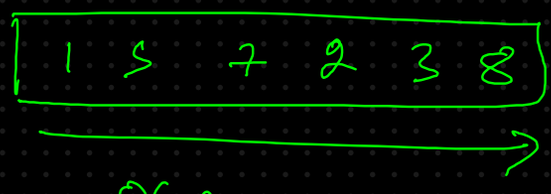
Customer → 1gb

Payment → 10mb

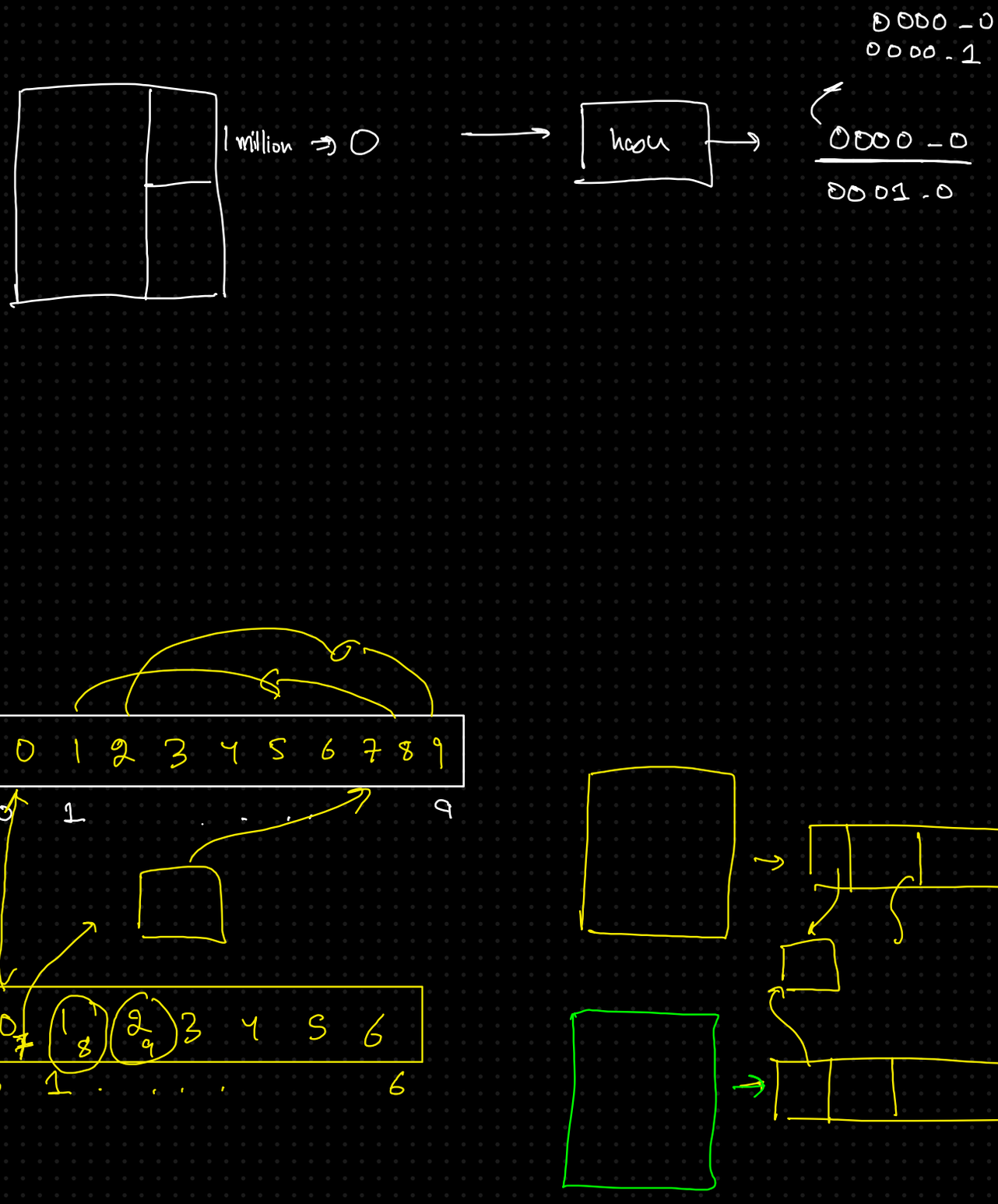
Bucket Map Join

large
large





O(n)
:
O(1)


0000_0
0000_1
1 million → 0
hash
0000_0
0001_0
0 1 2 3 4 5 6 7 8 9
0 1 9
0 1 2 3 4 5 6
0 1 6