Data Engineering Good to know Concepts

1) OLTP Vs OLAP
2) Database Vs Data Warehouse Vs Datalake
3) ETL Vs ELT
4) Data Warehouse - Types, Schema types, Fact table, Measures types (Operational, Enterprise, Star schema, Snowflake)
5) ER Vs Dimensional Modelling
6) SLOWLY changing Dimension (SCD)
7) Normalization Vs Denormalization
8) Basic Principles - ACID, BASE & CAP Theorem
9) Data Engineering Basic Flow - Hadoop, AWS, GCP, Azure
10) HDFS Vs Cloud Datalake
11) Object Storage Vs Block storage
12) Linux & HDFS Basic commands
13) HDFS - Mapreduce Overview - Architecture, Process, Internals
14) SQoop Basics - Import, Incremental load, Full load, Export
15) Hive Basics - Architecture, types of tables, Load Patterns, Basic Optimization
16) Spark Overview

OLTP

Relational databases

Application oriented

Tables are normalized

Operational current data

Used for business tasks

Mostly in GB data size

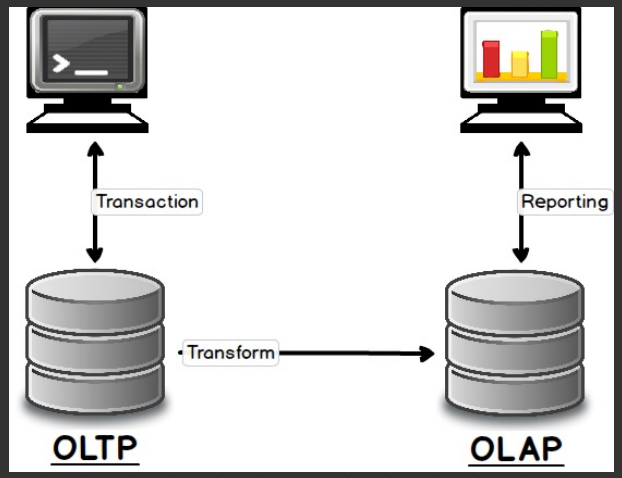
Both Read and Write

Mostly structured/ Batch usage

Reveals a snapshot of current business task

Query response time is less (ms)

Online Transaction Processing



OLAP

Relational databases or datawarehouses

Subject Oriented

Denormalized

Historical data from various databases

Used for Data Mining, Analytics, Decision Making

Terabytes / Petabytes of data

Mostly only Read data

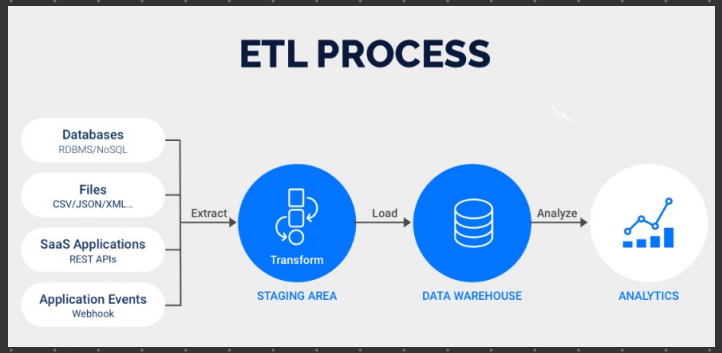
Adhoc Query usage from Business users

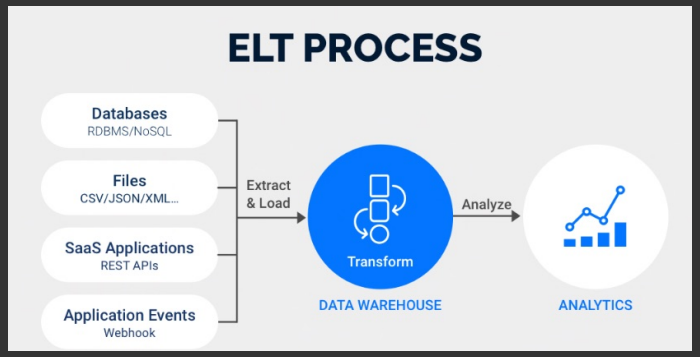
Provides multi-dimensional view of different bus. tasks

Query response time is more

Online Analytical Processing

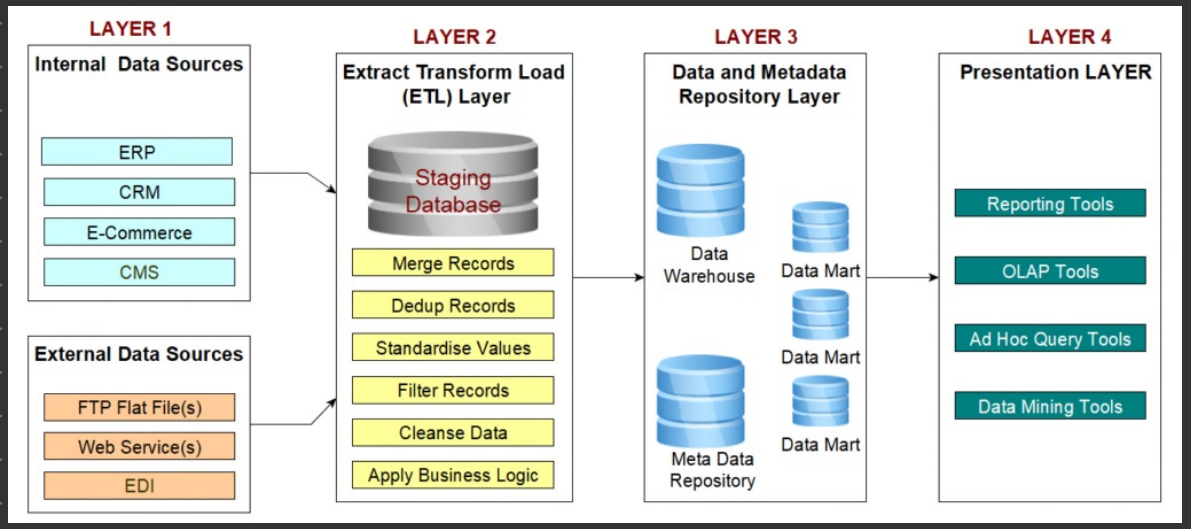
	Database	Data Warehouse	Datalake
Processing Method	OLTP	OLAP	OLAP
Type of data	Structured/ Semi Structured	Structured / Semi structured	Structured / Semi Structured / Unstructured
Infer of schema	Schema on write	Schema on write	Schema on read
Input Processing	ETL	ETL	ELT
Data View	Operational current data	Current and Historical data	current and historical data
Cost	High	High	Low
Technique	Capture data	Analyze data	Analyze data
Usage	Designed using ER modelling	Designed using Dimensional modelling	
Query-type	Simple	Complex	Complex
Examples	Relational DB - Oracle MySQl	Teradata	HDFS, AWS S3, GCS
Users	Application Developers	Business Analyst	Data scientists





Sources	Relational database, Flat files	Relational databases, flat files, Images, Videos
Transformation	Before loading	After loading
Destination	Datawarehouse, Datalakes	Data warehouse, Datalakes
Location/Infrastructure	On-premise, cloud	Mostly cloud
Data sizes	Mostly in Terabytes	Prefered of terabytes of data or more
Data types	Mostly structured / Semi structured	Structured, Semi structured & Unstructured
Storage Requirement	less, since only required data is loaded	Higher, since entire raw data will be loaded
Scalability	Mostly on-premise data centre is low	High due to auto scaling
Compliance	Masking / Removing sensitive data before loading	Risk of exposing data since data is loaded
Data Transformation	Data transformation are compute intensive	Data transformations are less complex

Data Warehouse Basics



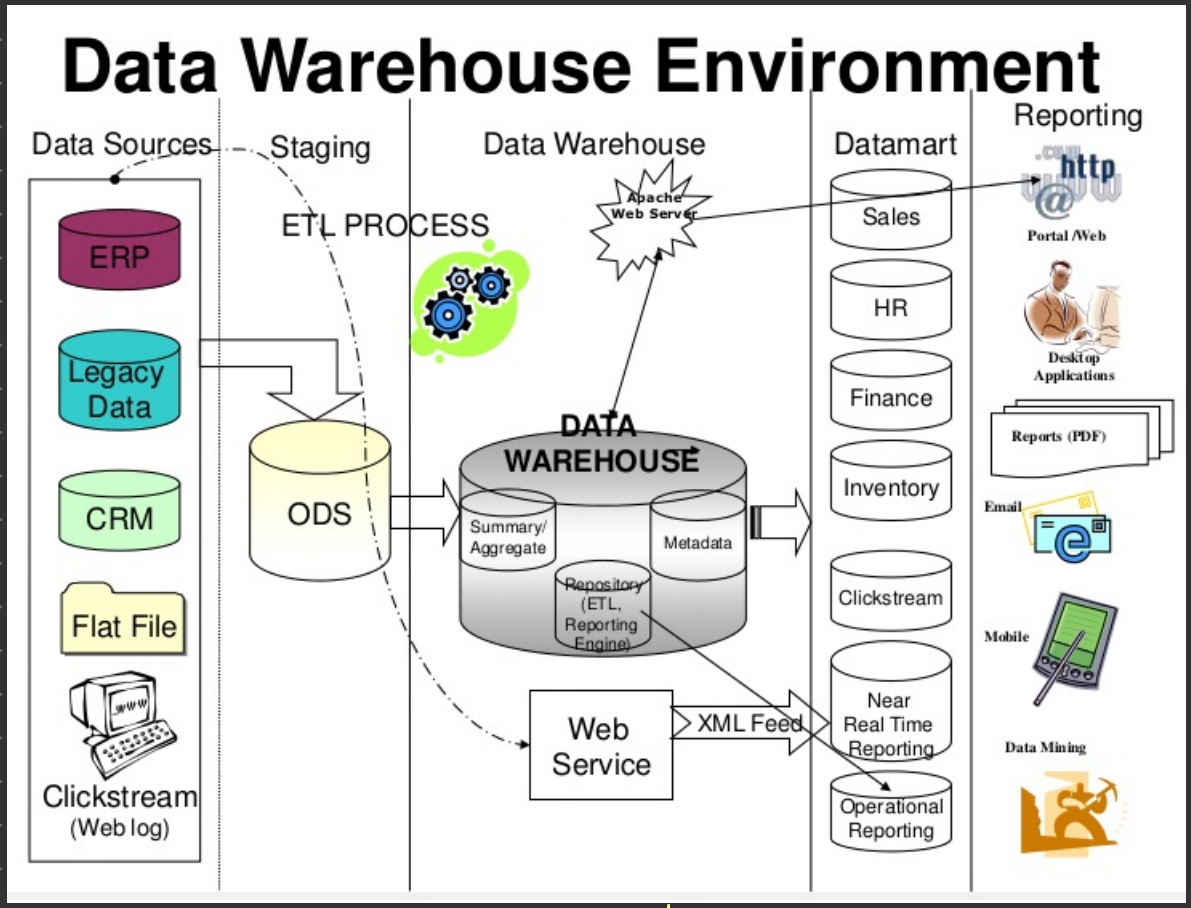
1) Data warehouse works as a central repository where information arrives from one (or) more data sources.

2) Mostly data flows into a data warehouse from transactional system and other relational databases.

3) Data may be structured mostly and semi structured.

4) Data is processed, transformed and ingested so that users can access processed data in data warehouse through BI tools, SQL clients and spreadsheets.

5) Types of Data warehouse – Operational Data store
Enterprise Data Warehouse
Datamarts



ODS Enterprise DWH Data marts

Operational Data Store:

Data is refreshed in rare real time and used for routine business activity

Ex: storing records of the employees.

Enterprise Data Warehouse

Centralized data warehouse and provides decision support services across Enterprise

Offers unified approach for organizing and representing data.

Data Marts

Datamart is subset of data warehouse

specially designed for particular line of business eg: Sales, finance etc

Dimensions Vs Measures:

Measures are the facts, the number of event.

Ex: Total profit, Total sale

Dimensions are the details that explains your fact

Ex: Time, Location, Name, Product

Types of Measures:

Additive Measures:

Numeric value in fact table that is more flexible and for each dimension you can sum up. Ex: total sales of company

Semi Additive Measures:

Semi additive facts are facts that can be summed up for some of the dimensions but not for others: Ex. No. of items in warehouse for each day can sum up but not for entire year.

Non Additive Measures:

Cannot be added to any of the dimensions.

Types of Dimensions:

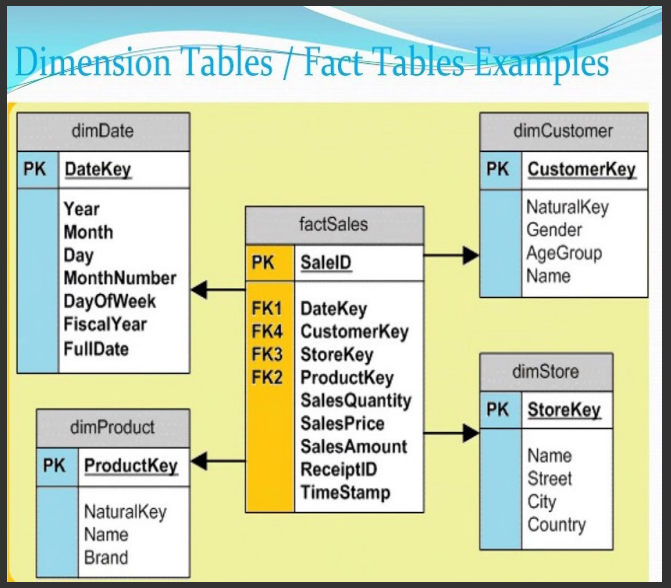
Most important SCD - slowly changing dimension

Basic components of data warehouse schemas:

Fact Table:

Fact table aggregates, metrics, measurements or facts about business process

Fact table store primary keys of dimension table as foreign key



Dimension Table:

Dimension tables are non-denormalized tables used to store data attributes or dimensions.

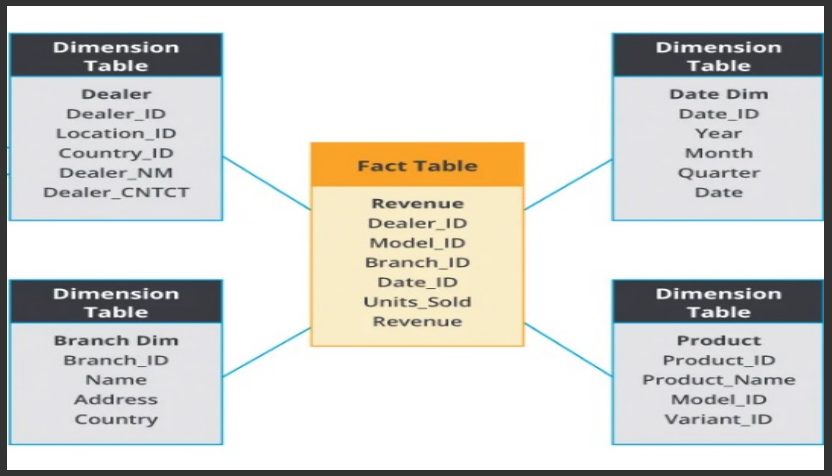
Primary key of dimension table is stored as foreign key in fact table.

Types of schemas:

Star Schema

Snowflake schema

Galaxy schema

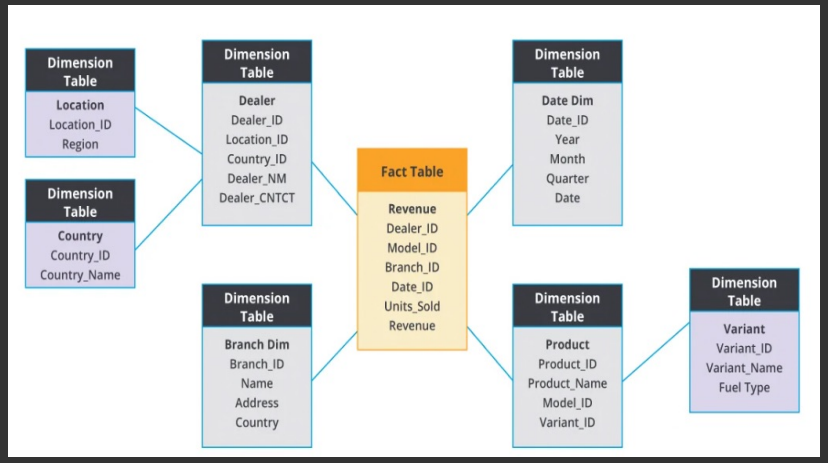


Star Schema:

Star DWH schema create denormalized database that enable quick quering responses.

Primary key in dimension table is joined to fact table by foreign key.

Dimension tables are not to be connected directly

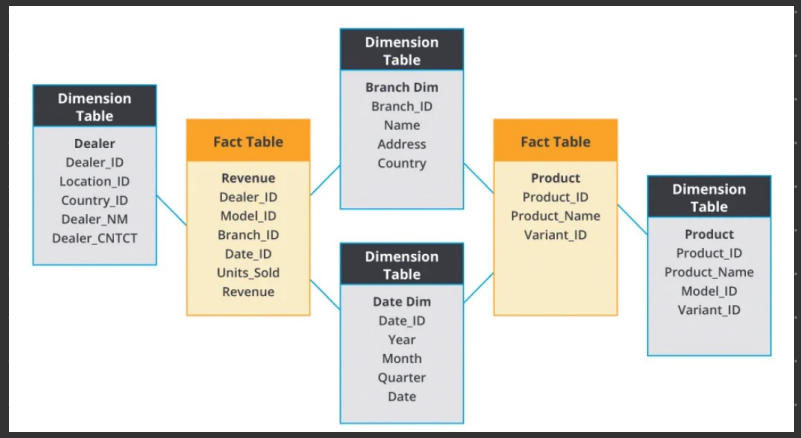


Snowflake Schema

Snowflake schema - dimension tables joined to other dimension table.

Snowflake schema are to have one fact table only

Snowflake schema offers easier way to implement dimension



Galaxy Schema:

Galaxy schema is multidimensional acting as strong design consideration for complex systems

Galaxy schema reduces redundancy as a result of normalization

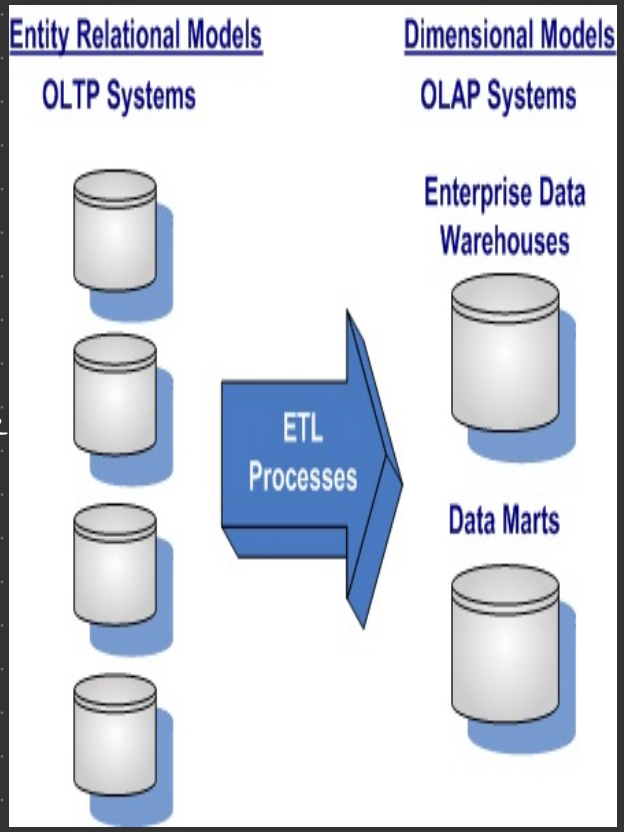
known for high data quality leads to effective reporting.

Types of fact tables

Characteristics	Transaction Grain Fact table	Periodic Snapshot Grain fact table	Accumulating Snapshot fact table
Time Period	Point in time	Regular predictable intervals	In determinate time span
Grain	One row / transaction event	One row / period	One row / span life
Fact table loads	Insert	Insert	Insert and Update
Fact row updates	No	No	Whenever required
Date dimension	Transaction date	End of period date	Multiple dates
Facts	Transaction Activity	Performance for Predefined time interval	Performance over finite lifetime
Examples			

Entity Relationship Modelling Vs Dimensional Modelling

1) Transaction Oriented
2) Entities and Relationships
3) Few levels of granularity
4) Real time information
5) Eliminates redundancy
6) High transaction volumes using few records at time
7) Highly volatile data
8) Physical and logical Model
9) Normalization is suggested
10) OLTP Application
11) Application used for buying products from e-commerce websites.



Subject Oriented
Facts & Dimensional tables
Multiple levels of granularity
Historical Information
Introduces redundancy
Low transaction volumes using many records at a time
Non Volatile data
Physical Model
De-Normalization is suggested
OLAP application
Applications to analyze buying patterns of customer of various cities over 5 years.

Slow Changing Dimensions types

When organising a datawarehouse, we relate fact records to a specific dimension record with its related attributes.

What if information in the dimension changes?

1) Do you now associate all fact records with new value?
2) Do you ignore the change to keep historical accuracy?

For this scenario, there are several different types of SCD

Example: Customer Name Jack shifts from India to UK.

Current:

ID	Customer	Country
1	Jack	India

Mostly Used Type 2

Type: 0

Fixed dimension

No changes allowed

Type: 1

No history

Update record directly

1	Jack	UK

Type: 2 Row Versioning

Track changes as version records with current flag and active dates

Key	ID	Customer	Country	Flag	Effective Date	Expiry Date
1	1	Jack	India	0	1/1/2021	1/1/2022
2	1	Jack	UK	1	1/1/2022	

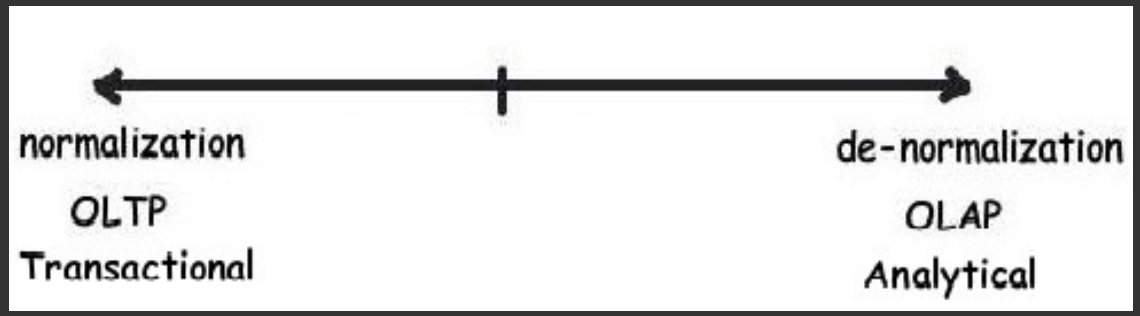
Type: 3 Previous Value Column

1	Jack	India	UK

Type: 4 History table

1	1	Jack	India	1/1/2021
2	1	Jack	UK	1/1/2022

Normalization and Denormalization



Basic	Normalization is the process of Creating a set of schema to store non-redundant & consistent data.	Denormalization is the process of Combining the data so that it can be queried fastly
Purpose	Data redundancy & inconsistency is reduced	Redundancy is added for quick execution of queries
Used in	OLTP	OLAP
Data Integrity	Maintained	Not maintained
Number of tables	Increases due to less redundancy	Decreases
Disk Space	Optimized Usage	Wastage

BASIC PRINCIPLES – ACID, BASE & CAP Theorem

ACID — Atomicity, Consistency, Isolation and Durability — Ex: MySQL, Oracle

BASE — Basically Available, Soft state, Eventually Consistent — Ex: Dynamo DB, Hbase, Mongodb, Redis

CAP — Consistent, Availability & Partition Tolerance — Ex: Both on combination

CA: RDBMS, Oracle

CP: Mongodb, Hbase

AP: DynamodB, S3

ACID compliant database used in RDBMS that focuses on consisteney.

BASE used in many NOSQL database that focus on availability.

CAP is basically a continuum along which BASE & ACID are on opposite ends.

CAP is Consisteney, Availability & Partition tolerance. Basically you can pick 2 of those but not all 3.

ACID focus on consisteney and availability.

BASE focus on Partition tolerance & availability and throws Consisteney out of window.

ACID Consisteney & CAP consisteney are different. → Eventual consisteney in all nodes

↳ Consistent view of data for all DB

Data Engineering Basic Pipeline With Datalake

Ingestion → Storage → Process → Serving / Presentation

	Ingestion	Storage	Process	Serving / Presentation	
				Relational	NOSQL
On-Premise (Hadoop)	SQOOP	HDFS	MAPREDUCE / SPARK	HIVE	HBASE
AWS	AWS Glue	Amazon S3	Athena / Redshift	RDS	DynamoDB
Azure	ADF (Azure Data factory)	ADLS Gen2	Databricks / Synapse	Azure SQL	Cosmos DB
GCP	Dataflow	Google Cloud Storage	Big Query	Cloud SQL	Cloud Big table

HDFS Vs Cloud datalake

Distributed file system that store data as blocks

HDFS is not persistent

Storage is tightly coupled with compute

Data in HDFS from one cluster cannot be accessed from another cluster.

Drawbacks of HDFS / on Premise
[No Agility, No Geo Distribution, More Opex]

Object-based storage that stores data as objects.
Ex: Amazon S3

Amazon S3 is persistent

Storage is decoupled from compute.
Ex: Different Services -
Storage S3
Compute EC2
storage cost only for data stored in S3.

Any number of cluster can access the same data.

Advantages of Cloud service Provider
[Agility, Min Opex, Geo Distribution] — AWS, GCP, Azure
S3, GCS, ADLS Gen 2

Block Storage Vs Object Storage

	Block Storage	Object Storage
Data storage	Fixed-sized blocks store portions of data in hierarchical system Ex: HDFS	Unique, identifiable and distinct units called objects store data in a flat file system Ex: Amazon S3
Cost	More expensive	More cost effective
Scalability	Limited scalability	Unlimited scalability
Performance	Best of transactional data and database storage.	Suitable for high volumes of unstructured data. Performs best with large files.
Location	A centralized system that stores data on premise (or) in private cloud.	A centralized (or) geographically dispersed system that stores data on-premise, private, hybrid (or) public cloud.

Linux Commands

pwd - present working directory

touch - creates an empty file

vi - edit (or) create a file

mkdir - make new directory

rmdir - removes an empty directory

rm - removes files

rm -R - removes a dir recursively

cp - copy files / directories

mv - move/rename files and directories

cat file - View file contents

grep - search

cat file1 >> file2 - concatenate file1 with file2

cat fileA fileB > fileAB - Merge fileA fileB

head - show first 10 lines

tail - show last 10 lines

du -s -h ⇒ Disk usage

cd - change specific directory

cd ~ - Go to home directory

cd / - Go to root directory

cd . - Current directory

cd .. - Go to one level up

cd - - Go to previous present [accessed] directory

ls - list files & dir

ls -a list hidden files

ls -l - long listing format

ls -t sort by last modified

ls -r reverse order by dictionary

ls -S Sort by file size

ls -R recursive list

CHMOD

r read 4

w write 2

x Execute 1

CHMOD -R [Recursively]

Ex: CHMOD 777 file1

4+2+1 rwx Owner

4+2+1 rwx Group

4+2+1 rwx others

HDFS Commands

Prefix the linux commands as below on a distributed system.

```
hadoop fs -
hdfs dfs -
```

Ex: `hadoop fs -ls`

`hadoop fs -mkdir /user/data`

`hadoop fs -rm -R /user/`

Copy files/directories from local to HDFS:

```
hadoop fs -put <local file path> <hdfs file path>
```

(-copyFromLocal)

Copy files/directories from HDFS to local:

```
hadoop fs -get <hdfs file path> <local file path>
```

(copyToLocal)

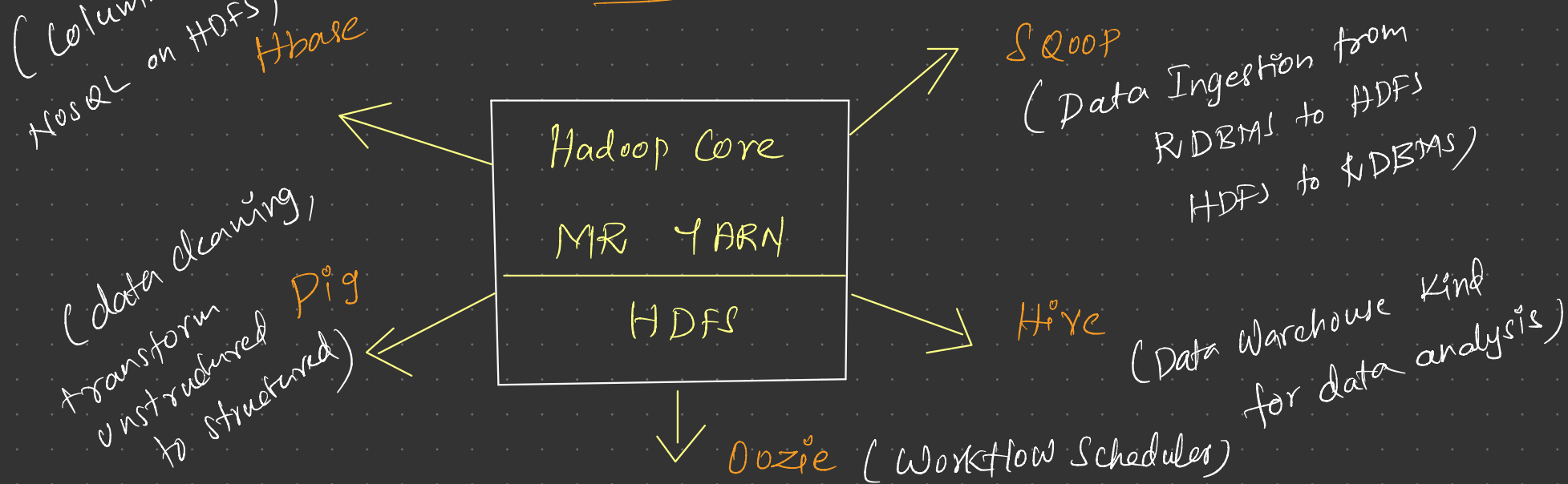
Hadoop

(For distributed storage & Processing)

Versions

1.0	– HDFS, Mapreduce Jobtracker	Single Name node (Single point of failure)
2.0	– HDFS, Mapreduce, Yarn	One or more Name Nodes (No single point of failure)
3.0	– HDFS, Mapreduce, Yarn	

Hadoop Ecosystem



HDFS Architecture

Ex: File A size - 500 MB

Hadoop 3.0 → Blocksize ↓ 128 MB

500/128 ⇒ 4 Blocks

B1, B2, B3, B4

DN - Commodity machines

NN - Non Comodity machines

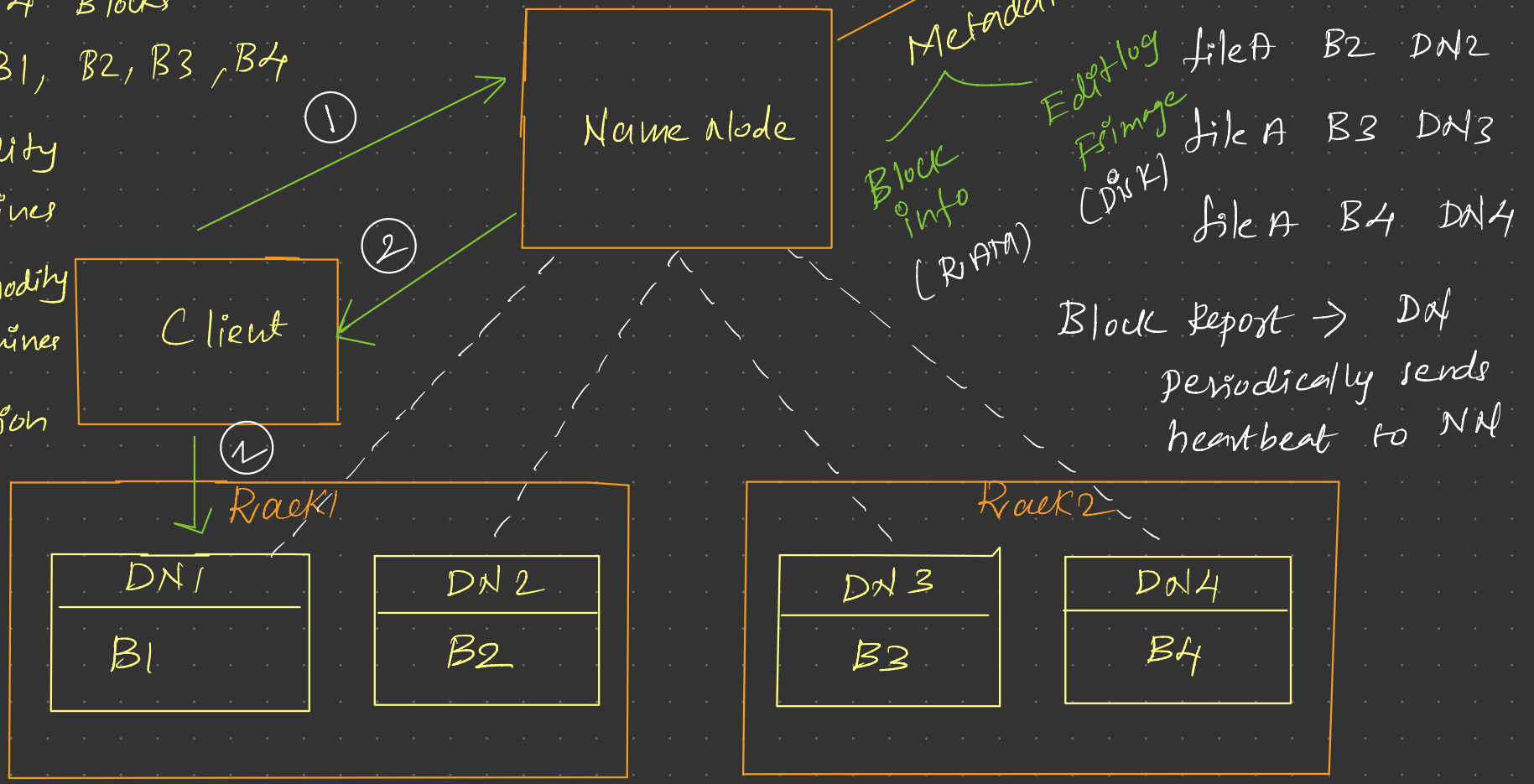
Block Replication

Default = 3

Parity block in hadoop = 3

Name Node failure → Secondary Namenode (Backup NN) / Name Node Federation

Data Node failure → Block Replication



Metadata

file A	B1	DN1
file A	B2	DN2
file A	B3	DN3
file A	B4	DN4

Block Report → DN periodically sends heartbeat to NN

Client → Request → Namenode → Namenode sends block info/ Metadata info

→ Client reads data from blocks of data node based on metadata

Basic Concepts in HDFS

Rack Awareness Policy:

Data Replication — High Availability
Fault tolerance

Not more than 1 replica in one data node

Not more than 2 replica in same rack

Name Node failure Management

fsimage — Snapshots of in-memory files at given time

editlogs — all new changes after above snapshot taken

Data locality:

Application code is moved to the data node where data resides and computation/process happens on the same node [same block]

Check pointing:

secondary name node does check pointing

Block Report:

Data Node periodically sends signal (Heart Beat) to Name Node

every 3 secs → fault tolerance

Blocks: Breaking down file into multiple parts as per block size

Failure Management:

Data Node — Name Node will create one more copy to maintain replication factor when any data node is down

Name Node — Secondary NN merge both fsimage & edit logs to keep fsimage updated

Map Reduce

Mapreduce is a computing paradigm for processing data that resides on many computers.

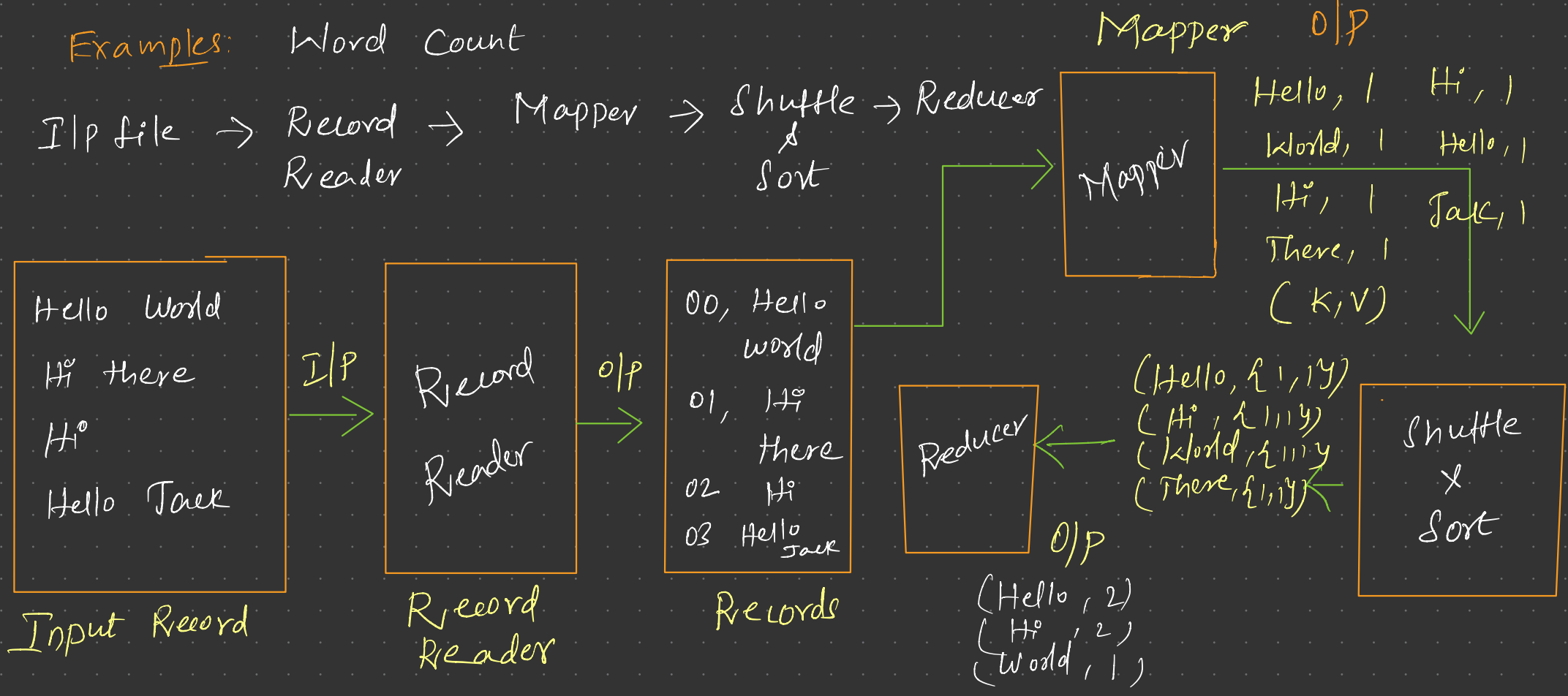
Two stages
- Map
- Reduce

Reducer can be zero, if there is not any aggregation/action.
Ex: for filter [Record fetch] only mappers required

Both Map & Reduce works only on Key, Value pairs (K, V)

Examples: Word Count

I/p file → Record Reader → Mapper → Shuffle & Sort → Reducer



Number of mappers = No of Blocks

Ex: 1GB; 1024/128; 8 Blocks; 8 Mappers; [II+3 cluster 3 mappers] 8 tasks

Map Reduce Concepts

Mapper + Combiner + Reducer — Code logic

Record Reader – Role of record reader is to convert each input line into (Key, Value) pair suitable for reading by Mapper.

Combiner – To be done on mapper side. Improves Parallelism & reduces data transfer.

Shuffling – The movement of data from mapper machine to reducer machine is called shuffling.

Both sorting & shuffling will happen on reducer machine. They are internal operations.

Default number of reducers → 1

(We can also customize partitioning logic)

When to make the number of reducers to zero?

There can be few jobs which do not require aggregation.

Ex: filter

Shuffle and sort only happen when we have one (or) more reducer.

When we have more than one reducer! Map → Partition → Shuffle → Sort → reducer

Concept of partition comes into play to tell which (K, V) pair goes to which reducer (Hash function)

Sqoop Basics

SQOOP is a tool designed to transfer data between Hadoop and relational databases.

SQOOP Import:

Transfers data from relational databases such as Oracle, MySQl to Hadoop (HDFS)

Each row in a table is treated as a record in HDFS.

Records can be stored as different formats such as Text file, Sequence file, Avro & Parquet file format.

SQOOP Export:

Transfers set of files from HDFS to Relational databases.

Files are read and parsed into a set of records according to user specified delimiters.

Sqoop Import

[Full Load]

```
Sqoop job \
--create job_emp1
-- import \
--connect jdbc:mysql://localhost/pracetise
--username root \
--password-file file:///home/test/.password-file \
-- table emp \
-- columns emp_id, emp_name, emp_city \ (Select subset of columns)
-- where "emp.city in ('Chennai', 'Bangalore') \
-- split-by emp_id \ (When no primary key use split-by column) should be numeric
-- boundary-query "SELECT 1, 100000"
-- m1 (or) --num-mappers (or) --autoreset-to-one-mapper \
                                  ↳ (No primary key ignore error default to one)
-- as-sequencefile
-- fields-terminated-by '|' \
-- lines-terminated-by ';' \
-- target-dir /emp/output
          (or)
-- warehouse-dir /user/cloudera/sqoopdir — Create sub directory with name of Source table
-- warehouse-dir /queryresult 1>query.out 2>query.err
                                  ↳ Redirecting logs
-- delete-target-dir
-- Verbose → Generate logs
-- compress
-- compression-codec BZip2Codec
```

Default No of mappers ⇒ 4

Default file format ⇒ Text file format

Compression [-z ⇒ gzip]

Sqoop Incremental Import

Sqoop supports two types of incremental imports:

append — Import goes to new target loc. If target exists in HDFS, Sqoop refuses
Append import will append data to existing dataset in HDFS.

Last Modified —

With --incremental argument specific type of incremental import.

Incremental import runs in last modified mode will generate multiple datasets in HDFS. Merge will flatten datasets into one

```
sqoop job \
-- create job_emp \
-- import \
-- connect jdbc:mysql:///localhost/practise
-- username root \
-- password-file file:///home/test/.password-file \
-- table emp \
-- warehouse-dir /data \
-- incremental append \
-- check-column emp_id \
-- last-value 0
```

(or)

```
--null-non-string "-1"
--null-string "NA"
-- incremental lastmodified \
-- check-column emp_doj_date \
-- last-value \
-- merge-key emp_id
```

Append ⇒ Duplicate records in HDFS

Last Modified ⇒ Latest Records

Sqoop Import Execution Flow

How mappers divide the work:

Select 1 record and by using that it gets metadata and builds the Java file

Using Java file it builds jar file

BoundaryValsQuery based on min and max by split column.
(default is primary column)

$$\text{Split size} = \frac{\text{Max} - \text{Min}}{4 \text{ (Default is 4)}}$$

Each mapper will run select query on the source table with where condition based on the splits to read the data.

When no of mappers > 1, boundaryValsQuery will run

Can be customized using --boundary-query

We can hardcode the min and max values with an intention to remove outliers.

Sqoop Export

```
sqoop export \
  -- connect jdbc:mysql://localhost/practise \
  -- username root \
  -- password-file file:///home/test/.password.file \
  -- table emp \
  -- staging-table emp_stage \
  -- export-dir /data/emp_det.csv \
  -- fields-terminated-by ','
```

Table emp should have created before

Sqoop other commands:

Sqoop Create-hive-table — populates hive metastore with definition of table

List available sqoop jobs — sqoop job --list

Execute sqoop job — sqoop job --exec emp-job

Show last incremental parameter — sqoop job --show emp-job | grep incremental/
cd .sqoop | cat metastore.db.script | grep incremental

Delete a Sqoop Job — sqoop job --delete emp-job

Sqoop Eval — Use for query the table

HIVE

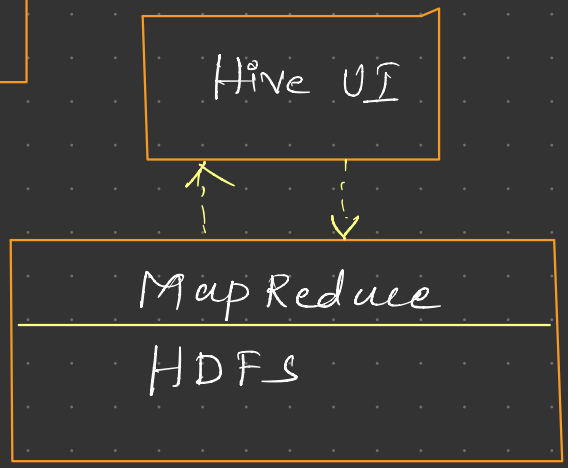
Hive is a data warehouse tool to process structed data on top of Hadoop / HDFS.

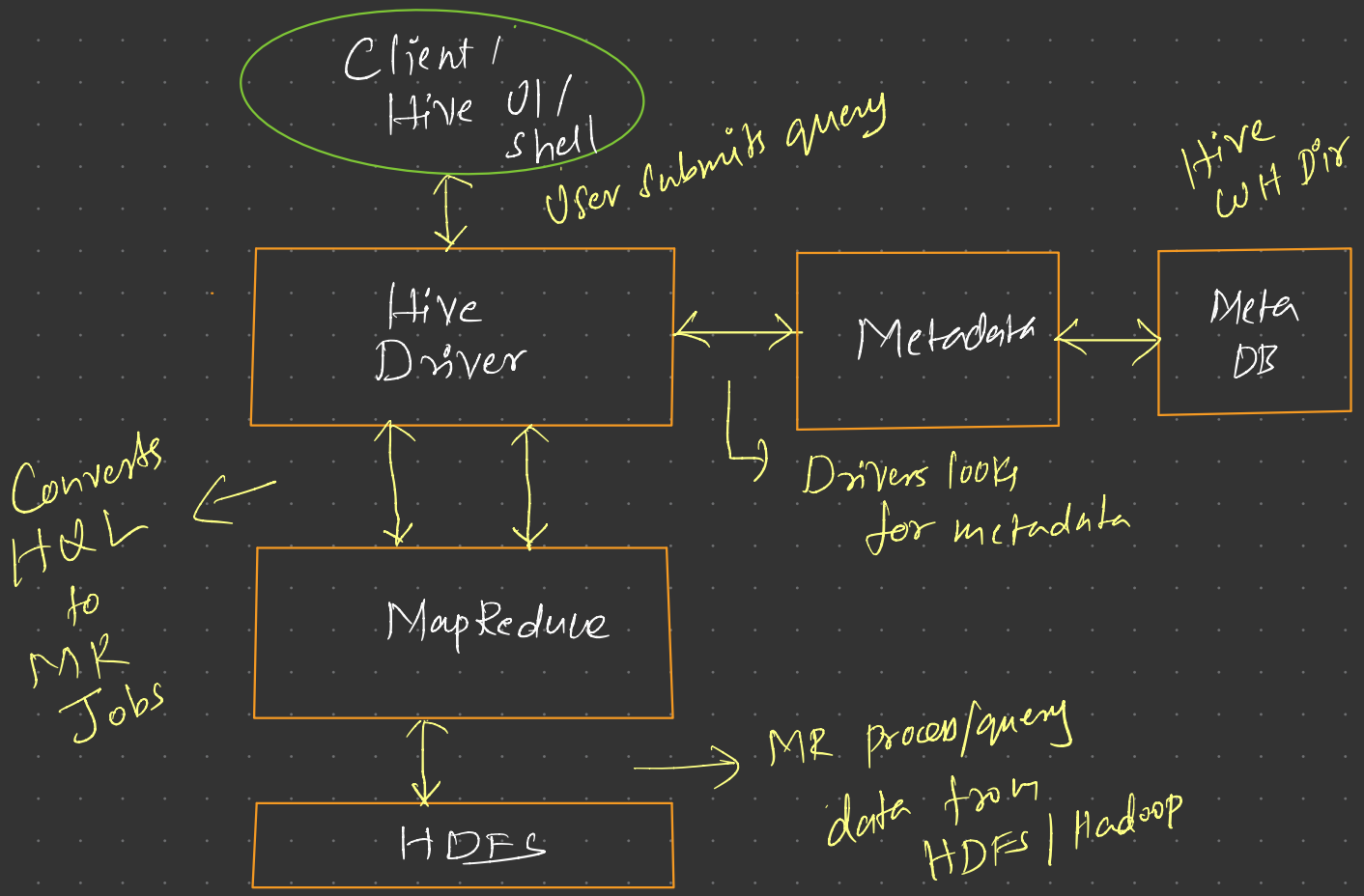
Hive provides parallelism.

Hive runs all process in form of MR jobs

Hive uses HQL (High Query Language)

Schema on read





HQL - Hive Query Language

Metadata - holds table definition, schema, serialization/deserialization info

Hive is [partition, URI]

For large datasets
Parallel computations
High Latency
Only Read operations
Not ACID compliant

HIVE Objects:

Database → Tables → Partitions → Buckets → Files → Records/Rows → Columns

/test /Student /year=2023/ 2 Buckets / month 1-6 file1 → Student1 → Id, name
month 7-12 file2 Student2

HIVE datatypes:

Primitive Datatypes:

Binary

Tinyint, Smallint, Int, Bigint

Boolean

Float, Double

Decimal

Timestamp, Date

Char, Varchar

String

Complex Datatypes:

Array – elements are same type. Accessed using [n] notation

Map – [Key, Value] pair. Accessed using [key] notation. Unordered collection

Struct – different data type. Accessed using DOT [.] notation

Hive Built in functions:

User defined functions [UDF] — Works on single row & output on single row
Ex: concat(), length()

User defined aggregate functions [UDAF] — Works on multiple rows & output on single row
Ex: Sum(), Avg()

User defined Table generating functions — Works on single row & output on multiple rows
Ex: explode()

Hive type of tables:

1) Managed tables — Data managed by Hive (Hive owns files & directories)
Deleting managed table deletes both table and metadata

2) External tables — Data not managed by Hive
Share underlying data across other ecosystems (Pig, Hbase)
Deletes only metadata

Loading data into Hive table:

1) Using insert command
2) Loading from file
   a) from Local file system
   b) from HDFS
3) From Hive table itself

1) Using Insert Command

```
Create table emp (
        id  bigint,
        name  string,
        address  string);

INSert into emp values
        (1, Jack, Bangalore),
        (2, Suresh, Bangalore),
        (3, Pradeep, Chennai);
```

2) Loading data from file into Hive table:

a) Local file system

```
Create table if not exists emp
(
    id      bigint,
    name    string,
    address string
)
row format delimited
fields terminated by ','
stored as textfile;

load data local inpath
'/home/local/emp.csv'
into table emp;
```

(or)

```
hadoop fs -copyfromlocal /home//local/emp.csv
                         /table_directory;
```

b) HDFS file system

```
load data inpath '/data/inbound/emp.csv'
into table emp;
```

3) Loading data from existing table into new table

```
insert into table emp_new
Select * from emp;
```

Note: emp_new to be created before

External Tables

```
Create external table if not exists product (
    id string,
    name string,
    colour array <string>
    features map <string, boolean>,
    information struct <battery : string, camera : string>
    row format delimited fields terminated by ',',
    Collection items terminated by '#'
    map keys terminated by ':'
    Stored as textfile
    location '/test/data';
```

In the same column multiple values/collections identified by '#' & key value by ':' for map

Data [csv]

Id	Name	Colour	Features	Information
1,	iphone,	white#black#rosegold,	5G:true#NFL:true	, 24 hours

Collections — map keys

Hive Partitioning & Bucketing

Partitioning

Split data into smaller subsets.

Records will be split across multiple machines in the cluster

Ex: For Monthly sales data:

/test/db/sales/year=2023/month=01 → Assume for each day 1 file → ~30 files → Limit the No. of. files from 30 ↓ 10

/month=02

/month=03

For Feb month sales data, data will be scanned only under

/test/db/sales/year=2023/month=02

Types:

Static Partitioning –

Dynamic Partitioning – Automatically create partition on data

Bucketing

Use Bucketing

Before Bucketing

/test/db/sales/year=2023/month=01/file1

/test/db/sales/year=2023/month=01/file30

After Bucketing

/test/db/sales/year=23/mon=01/file1

:

/test/db/sales/year=23/month=01/file10

Partitions:

- Unknown number of Partitions
- Based on actual column values
- Each partition stored as a directory
- Used retrieve (or) scan data in logical group

Bucketing:

- Fixed number of buckets
- Based on a hash value of column
- Each Bucket stored as files under directory
- Used for look up, joins & sampling

DISTRIBUTE BY SORT BY

Records with same column value goes to same reducer.

Ex: DISTRIBUTE BY (City) Records with same city goes to same reducer

CLUSTER BY

Distribute By does not guarantee clustering (or) sorting properties on distributed keys.

Cluster by is a shortcut for both distribute by & sort by.

HIVE OTHER BASIC COMMANDS (FUNCTIONS)

UNION

UNION ALL

DISTINCT

SUBQUERIES

IN / NOT IN

EXISTS / NOT EXISTS

EXPLODE

DISPLAY (SELECT)

ARRAY - Select id, colour[0] from product

MAP - Select id, features['battery'] from product

STRUCT - Select id, information.battery from product

RANKING FUNCTIONS

row_number()

rank()

dense_rank()

```
Show databases;
Create database emp;
Use emp;
Show tables;
describe emp_details;
describe formatted emp_details;
```

Spark Overview

General purpose in memory compute engine

Spark can replace Map Reduce in hadoop core components

HDFS + MR + YARN
↓
HDFS + SPARK + YARN

Spark is plug and play engine

Spark is faster than Map Reduce

Spark is written in Scala

Traditional Hadoop

HDFS + MR + YARN

Challenges of Map Reduce:

MR is very slow comparitive with Spark

standalone HDFS storage only

Spark

HDFS (or) AWS S3 (or) GCP GCS + Spark + YARN (or) Mesos (or) Kubernetes

Basic unit which holds the data in Spark is called RDD

(Resilient Distributed Dataset)

In spark there are two kind of operations.

- Transformations
- Actions

RDD distributed data across memory

Blocks distributed data in disk

Transformations are lazy. Means Execution happens when action is encountered before that only entries are written into DAG.

DAG is generated when spark code is executed.

Spark code can be written in

- Scala
- Python — Pyspark (Mostly used)
- Java
- R

But Spark written in Scala gives better performance.