# Apache Hive

Transactional vs Analytical system

## Transactional

=> day to day transactions
=> inserts, updates, deletes
=> example ATM / placing an order on amazon
=> RDBMS - Databases (Monolythic Systems)
=> mysql, oracle

## Analytical system

=> Historical data
=> sales campaign, why sales are less this month.
=> you talk mostly abour reads, analyse large batches of data.
=> DWH - teradata
=> Distributed systems are best for for this.

Hive is a Datawarehouse

Hive has something called as a metastore - Metadata (Schema)

HDFS / S3 / ADLS gen2 / GCS - Datalake - DATA

The Apache hive is a distributed, fault tolerant data warehouse that enables analytics at a massive scale and facilitates querying petabytes of data residing in distributed storage using SQL like syntax (HQL)

HQL queries will be converted to mapreduce jobs / tez

abstracting the complexity from the user.

A hive table has 2 parts

1. Actual Data - HDFS / Datalake
2. Metadata (Schema) - Metastore DB

your metadata could also have been stored in HDFS

why metadata is kept in a database?

not keeping inside hdfs
but keeping inside mysql or any other database

if you keep your metadata in hdfs then you

=>cannot perform updates
=>if something is stored in hdfs then

datalakes generally offers high throughput but cannot offer low latency.

RDBMS
======

=> schema on write

we first create a table
we insert the data in the table
while writing the data it will validate things

Hive
======

=> Schema on read

1. we already have some data in HDFS in the form of files
2. we create the table on top of this data so that we can see a tabular view.

Hive Metastore (HMS) - stores the table schema / Metadata

HiveServer2 - is a service that enable the clients to execute queries against hive

JDBC client - Beeline (to interact with HiveServer2)

```
hive

beeline
!connect jdbc:hive2://m02.itversity.com:10000/;auth=noSasl

create database trendytech_101 ;

set hive.metastore.warehouse.dir=/user/itv005857/warehouse

set hive.server2.logging.operation.level=NONE;
```

```
use trendytech_101;

CREATE TABLE IF NOT EXISTS demo_table_1 (
  id INT,
  name STRING,
  age INT
);


INSERT INTO demo_table_1 VALUES
  (1, 'John', 25),
  (2, 'Jane', 30),
  (3, 'Bob', 22);
```

a directory is created for your database

/user/<username>/warehouse/<databasename.db>/

======================

Types of tables in Hive
========================

2 types of tables

1. Managed table - hive manages both data + metadata
2. External table - hive manages only the metadata

Data + Metadata

Managed table - when we drop the table both the data and metadata gets deleted

External table - when we drop the table only the metadata is deleted. Data is still kept intact.

```
set hive.metastore.warehouse.dir;
set hive.metastore.warehouse.dir=/user/itv005857/warehouse;

CREATE TABLE IF NOT EXISTS orders_managed (
  order_id integer,
  order_date string,
  customer_id integer,
  order_status string
)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ','
STORED AS TEXTFILE;

LOAD DATA INPATH '/user/itv005857/hive_datasets/orders.csv' INTO TABLE
orders_managed;

insert into orders_managed values (11111111,"2013-07-25
00:00:00.0",4530,"COMPLETE");

update orders_managed set order_status = "CLOSED" where order_id =
11111111;
```

normally by default updates and deletes are not supported.

Data should be in ORC format
the table should be bucketed
transaction = true (property should be set)

hive supports 3 engines

=> mr
=> spark
=> tez

```
insert into orders_managed values (11111112,"2013-07-25
00:00:00.0",4530,"COMPLETE");
```

================

```
CREATE TABLE IF NOT EXISTS orders_managed (
  order_id integer,
  order_date string,
  customer_id integer,
  order_status string
)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ','
STORED AS TEXTFILE;

LOAD DATA LOCAL INPATH '/home/itv005857/hive_datasets/orders.csv'
INTO TABLE orders_managed;
```

```
describe formatted orders_managed;


CREATE EXTERNAL TABLE IF NOT EXISTS orders_external(
  order_id integer,
  order_date string,
  customer_id integer,
  order_status string
)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ','
STORED AS TEXTFILE
LOCATION '/user/itv005857/hive_datasets/orders'

hadoop fs -mkdir /user/itv005857/hive_datasets/orders

hadoop fs -put hive_datasets/orders.csv /user/itv005857/hive_datasets/orders

insert into orders_external values (11111111,"2013-07-25
00:00:00.0",4530,"COMPLETE");

CREATE EXTERNAL TABLE IF NOT EXISTS orders_external(
  order_id integer,
  order_date int,
  customer_id integer,
  order_status string
)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ','
STORED AS TEXTFILE
LOCATION '/user/itv005857/hive_datasets/orders'
```

====================================

## Hive Optimizations
====================

1. Table structure level - partitioning and bucketing

2. query level optimization - Join optimizations

partitioning vs bucketing
============================

orders table

select * from orders where order_status = ?

you can partition your data based on order_status

/user/<username>/warehouse/trendytech_101.db/orders/order_status=CLOSED
/user/<username>/warehouse/trendytech_101.db/orders/order_status=COMPLETE
.
.
.
.

select * from orders where order_id = ?
fixed number of buckets - 4,8,12

```
1,2013-07-25 00:00:00.0,11599,CLOSED
2,2013-07-25 00:00:00.0,256,PENDING_PAYMENT
3,2013-07-25 00:00:00.0,12111,COMPLETE
4,2013-07-25 00:00:00.0,8827,CLOSED
5,2013-07-25 00:00:00.0,11318,COMPLETE
6,2013-07-25 00:00:00.0,7130,COMPLETE
7,2013-07-25 00:00:00.0,4530,COMPLETE
8,2013-07-25 00:00:00.0,2911,PROCESSING
```

4 buckets

b0 - 4,8,12,16
b1 - 1,5,9,13
b2 - 2,6,10,14
b3 - 3,7,11,15

select * from orders where order_id = 11

```
CREATE TABLE IF NOT EXISTS orders_p(
  order_id integer,
  order_date string,
  customer_id integer
) PARTITIONED BY (order_status string)
ROW FORMAT DELIMITED
```

```
FIELDS TERMINATED BY ','
STORED AS TEXTFILE;

insert into orders_p partition(order_status) select order_id, order_date,
customer_id, order_status from orders_managed;

insert into orders_p values (11111113,"2013-07-25
00:00:00.0",4530,"COMPLETE");

select * from orders_p where order_status ="COMPLETE" and order_id =
11111113;

explain extended select * from orders_p where order_status ="COMPLETE"
and order_id = 11111113;

explain extended select * from orders_p where order_id = 11111113;
```

====================

Bucketing
==========

```
CREATE TABLE IF NOT EXISTS orders_b (
  order_id integer,
  order_date string,
  customer_id integer,
  order_status string
)
clustered by(order_id) into 4 buckets
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ','
STORED AS TEXTFILE;

insert into orders_b select order_id, order_date, customer_id, order_status
from orders_managed;

select * from orders_b where order_id = 1234;

explain extended select * from orders_b where order_id = 1234;

insert into orders_b values (111114,"2013-07-25
00:00:00.0",4530,"COMPLETE");
```

2 levels of partitioning

partitioning + bucketing

order_status + order_id

select * from orders where order_status = "CLOSED" and order_id = 101

bucketing + partitioning

==================================

Hive Optimizations

1. Table structure level - partitioning and bucketing

2. Query level optimization - Join optimizations

=> try reducing number of joins if possible.

join is a costly operation and involves shuffling.

broadcast join in spark

1. Map side join (when one of the table is small enough to fit in memory)

This small table can be broadcasted

view on each machine - complete small table / part of large table

left table is small & right table is big

inner join - possible
left outer join - not possible
right outer join - possible
full outer join - not possible

when left table is big and right table is small

inner join - possible
left outer join - possible
right outer join - not possible
full outer join - not possible

there is only a map phase, no reduce activity is involved..

2. bucket map join - lets say both the table are big.

is there a way to optimize join of 2 large tables...

constraints
============

1. both the tables should be bucketed on join column

2. number of buckets in one table should be an integral multiple of number of buckets in other table.

|   |   |
|---|---|
| 2 | 2,4,6,8 |
| 3 | 3,6,9,12 |

in map side join we load the complete smaller table in memory..

in bucket map join we load just one bucket in memory.

1. Map side join - one small and one large table

2. Bucket map join - 2 large table
    -both the tables should be bucketed on join columns
    -number of buckets in integral multiple

3. SMB (sort merge bucket join) - 2 large table
    - both the tables should be bucketed on join columns
    - number of buckets in both tables should be exactly same
    - both the tables should be sorted on join columns

| table1 | table2 |
|---|---|
| 4 buckets | 4 buckets |
| | |
| b1 | b1 |
| b2 | b2 |
| b3 | b3 |
| b4 | b4 |

```
CREATE EXTERNAL TABLE IF NOT EXISTS customers_external (
  customer_id INT,
  customer_fname STRING,
  customer_lname STRING,
  username STRING,
  password STRING,
  address STRING,
  city STRING,
  state STRING,
  pincode INT
)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ','
STORED AS TEXTFILE
LOCATION '/user/itv005857/hive_datasets/customers';
```

```
/user/itv005857/hive_datasets/customers
/user/itv005857/hive_datasets/orders
```

```
CREATE EXTERNAL TABLE IF NOT EXISTS orders_external (
  order_id integer,
  order_date string,
  customer_id integer,
  order_status string
)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ','
STORED AS TEXTFILE
LOCATION '/user/itv005857/hive_datasets/orders';
```

```
explain extended select o.*,c.* from orders_external o join customers_external
c on o.customer_id = c.customer_id limit 5;
```

How internally a map side join works
=====================================

before the mapreduce is execution, there is a local process which is executed.

this local task will take the small table and will create a hashmap

once the hashmap is created it is put on hdfs

from hdfs it is broadcasted to all the nodes

after broadcasting it is now present on local disk of all the nodes. This is called distributed cache.

hashtable is loaded in memory for each of the node.

mapreduce job starts...

Bucket map join
================

=> both the tables bucketed on join columns
=> number of buckets - integral multiple.

```
CREATE TABLE IF NOT EXISTS customers_demo_b (
  customer_id INT,
  customer_fname STRING,
  customer_lname STRING,
  username STRING,
  password STRING,
  address STRING,
  city STRING,
  state STRING,
  pincode INT
)
clustered by(customer_id) into 4 buckets
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ','
STORED AS TEXTFILE;

insert into customers_demo_b select * from customers_external;


CREATE TABLE IF NOT EXISTS orders_demo_b (
  order_id integer,
  order_date string,
  customer_id integer,
  order_status string
)
clustered by(customer_id) into 8 buckets
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ','
STORED AS TEXTFILE;


insert into orders_demo_b select * from orders_external;

set hive.enforce.bucketing=true;
set hive.optimize.bucketmapjoin=true;
```

```
select o.*,c.* from orders_demo_b o join customers_demo_b c on
o.customer_id = c.customer_id limit 5;
```

3. SMB (sort merge bucket join)

=> the number of buckets should be exactly the same
=> both the tables should be sorted on join column

```
CREATE TABLE IF NOT EXISTS customers_demo_b1 (
  customer_id INT,
  customer_fname STRING,
  customer_lname STRING,
  username STRING,
  password STRING,
  address STRING,
  city STRING,
  state STRING,
  pincode INT
)
clustered by(customer_id) sorted by (customer_id asc)  into 4 buckets
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ','
STORED AS TEXTFILE;

insert into customers_demo_b1 select * from customers_external;


CREATE TABLE IF NOT EXISTS orders_demo_b1 (
  order_id integer,
  order_date string,
  customer_id integer,
  order_status string
)
clustered by(customer_id) sorted by (customer_id asc) into 4 buckets
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ','
STORED AS TEXTFILE;

insert into orders_demo_b1 select * from orders_external;
```

```
explain extended select o.*,c.* from orders_demo_b1 o join
customers_demo_b1 c on o.customer_id = c.customer_id limit 5;

set
hive.input.format=org.apache.hadoop.hive.ql.io.BucketizedHiveInputFormat;
set hive.auto.convert.sortmerge.join=true;
set hive.auto.convert.sortmerge.join.noconditionaltask=true;
set hive.optimize.bucketmapjoin=true;
set hive.optimize.bucketmapjoin.sortedmerge=true;
set hive.enforce.bucketing=true;
set hive.enforce.sorting=true;
set hive.auto.convert.join=true;
```

=================

Transactional Tables in Hive
==============================

Databases

ACID properties

Atomicity
Consistency
Isolation
Durability

ACID transactions ensure the highest possible data reliability and integrity.

the data should never fall in an inconsistent state.

Multi Insert in a single statement. if this operation fails, the partial writes/inserts should not be visible to the user.

A user can read a table and simultaneously, another user can add rows to that table.

================

why do we want to perform transactions in hive (ACID)

Inserts/updates/deletes

slowly changing dimensions - In a typical star schema datawarehouse

fact - orders

dimensions - customers (inserts/updates/deletes)

For example-
A retiler will open new stores, this needs to be added to the stores table.

A user may be contractually required to remove their customers data upon termination of their relationship.

Basic design of HDFS

HDFS ideally does not support in place changes in file.

In order to provide these features on top of HDFS hive follows the standard approach used in other data warehousing tools.

Data for the table is stored in a set of base files. New records, updates or deletes are stored in delta files. At read time the reader merges the base and delta files.

we need to set certain properties in hive to make transactional tables work.

```
SET hive.support.concurrency=true;
SET hive.txn.manager=org.apache.hadoop.hive.ql.lockmgr.DbTxnManager;
SET hive.enforce.bucketing=true;
SET hive.exec.dynamic.partition.mode=nostrict;
SET hive.compactor.initiator.on=true;
SET hive.compactor.worker.threads=1;
```

Also a few other things you should do to enable transactional tables

TBLPROPERIES('transactional'='true')
Managed tables only (not external)
ORC file format

External tables cannot be created to support ACID since the changes on external tables are beyond the control of hive.

LOAD is not supported in ACID transactional tables

Insert into

auto commit

Note- once you create a table as ACID table, you cannot convert it back to non-ACID table.

```
CREATE TABLE IF NOT EXISTS orders_trx1 (
  order_id integer,
  order_date string,
  customer_id integer,
  order_status string
)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ','
STORED AS ORC
TBLPROPERTIES ('transactional'='true');
```

```
describe formatted orders_trx1;
```

```
transactional                 true
transactional_properties      default
```

```
insert into orders_trx1 values (1,"2013-07-25 00:00:00.0",45,"COMPLETE");
```

```
update orders_trx1 SET order_status = "CLOSED" where order_id=1;
```

```
delete - (1,"2013-07-25 00:00:00.0",45,"COMPLETE");
insert - (1,"2013-07-25 00:00:00.0",45,"CLOSED");
```

```
insert into orders_trx1 values (2,"2013-07-25 00:00:00.0",47,"CLOSED");
```

```
delete from orders_trx1 where order_id = 2;
```

```
SHOW TRANSACTIONS;
```

Hive compacts ACID transaction files automatically

Automatic compaction improves query performance and reduces the metadata footprint for better performance.

Read semantics consist of snapshot isolation. Hive logically locks in the state of the warehouse when a read operation starts.

========

Inserts/updates/deletes

Insert only transactional table

To create an Insert only ACID table, me must additionally specify the transactional property as insert_only.

All file formats are supported with the insert-only transactional table

```
CREATE TABLE IF NOT EXISTS orders_trx2 (
  order_id integer,
  order_date string,
  customer_id integer,
  order_status string
)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ','
STORED AS TEXTFILE
TBLPROPERTIES ('transactional'='true',
'transactional_properties'='insert_only');
```

```
describe formatted orders_trx2
```

```
insert into orders_trx2 values (1,"2013-07-25 00:00:00.0",45,"COMPLETE");
```

```
update orders_trx2 SET order_status = "CLOSED" where order_id=1;
```

```
delete from orders_trx2 where order_id=1
```

can we convert a non ACID table to a ACID - is possible

can we convert a ACID table to a non ACID one - it is not possible

we can only convert a non-ACID managed table into a ACID table.

if you have to convert an External hive table into hive ACID table,
you must first convert it to a non-ACID managed table by running below command

lets say you already have a external table
orders_external

```
CREATE EXTERNAL TABLE IF NOT EXISTS orders_external1 (
  order_id integer,
  order_date string,
  customer_id integer,
  order_status string
)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ','
STORED AS TEXTFILE
LOCATION '/user/itv005857/hive_datasets/orders'
```

```
describe formatted orders_external1
```

```
ALTER TABLE orders_external1 set tblproperties('EXTERNAL'='FALSE')
```

A managed table can now be converted into a transaction ACID table..

```
ALTER TABLE orders_external1 SET TBLPROPERTIES
('transactional'='true','transactional_properties'='insert_only');
```

you can convert a non-ACID hive table to a full ACID TABLE only when the non-ACID table data is in orc format.

==========================

Hive Spark Integration
========================

Hive - metastore (metadata)
Data - HDFS

metastore
hive queries

=============

MSCK repair in Hive
====================

you have created a external table orders which is partitoned on order_status

/user/itv005857/warehouse/hive_datasets/orders

COMPLETE
CLOSED

we use another table to load

insert into orders select * from ........

show partitions table orders

CANCELED

```
CREATE EXTERNAL TABLE orders_p1(
  order_id integer,
  order_date string,
  customer_id integer
) PARTITIONED BY (order_status string)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ','
STORED AS TEXTFILE
LOCATION '/user/itv005857/hive_datasets/orders107';
```

```
spark.sql("insert into trendytech_107.orders_p1 select * from
trendytech_101.orders_p where order_status = 'COMPLETE' OR order_status
= 'CLOSED'");
```

order_status=CLOSED
order_status=COMPLETE

order_status=CANCELED

```
hadoop fs -cp
/user/itv005857/warehouse/trendytech_101.db/orders_p/order_status=CANCE
LED /user/itv005857/hive_datasets/orders107
```