




“Information is the oil of the 21st century, and analytics is the combustion engine.”

–Peter Sondergaard, Gartner Research



## CASE STUDY: RETAIL LOG PROCESSING

### *About the Company*

**TENTOTEN** is a Retail Store which has a chain of hypermarkets in India. They have 250+ stores across 95 cities and towns. About 45,000+ people are working in **TENTOTEN**. **TENTOTEN** deals in a wide range of products including fashion apparels, food products, books, furniture, etc. Around 1500+ customers visit and/or purchase products every day from each of these stores.

### *Problem Scenario*

The approximate size of **TENTOTEN** log datasets is 12 TB. Information about the various stores is stored in the form of semi-structured data. Traditional Business Intelligence (BI) tools are good when data is present in pre-defined schema and datasets are just several hundreds of gigabytes. But the **TENTOTEN** dataset is mostly log dataset, which does not conform to any particular schema. Querying such large dataset is difficult and immensely time consuming.

The challenges are:

1. Moving the log dataset to HDFS (Hadoop Distributed File System).
2. Performing analysis on HDFS data.

Hadoop MapReduce can be used to resolve these issues. However we will still have to deal with the below constraints:

1. Writing complex MapReduce jobs in Java can be tedious and error prone.
2. Joining across large datasets is quite tricky.

# What it is?

- Hive is a Data Warehousing tool that sits on top of Hadoop.
- It is an open-source distributed data warehousing database which operates on Hadoop Distributed File System.
- Hive was built for querying and analyzing big data.
- The data is stored in the form of tables (just like RDBMS).

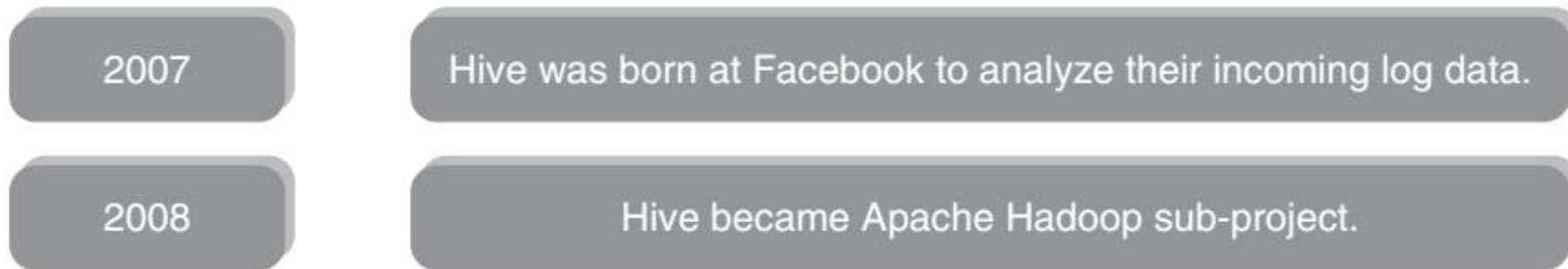
What it is?

---

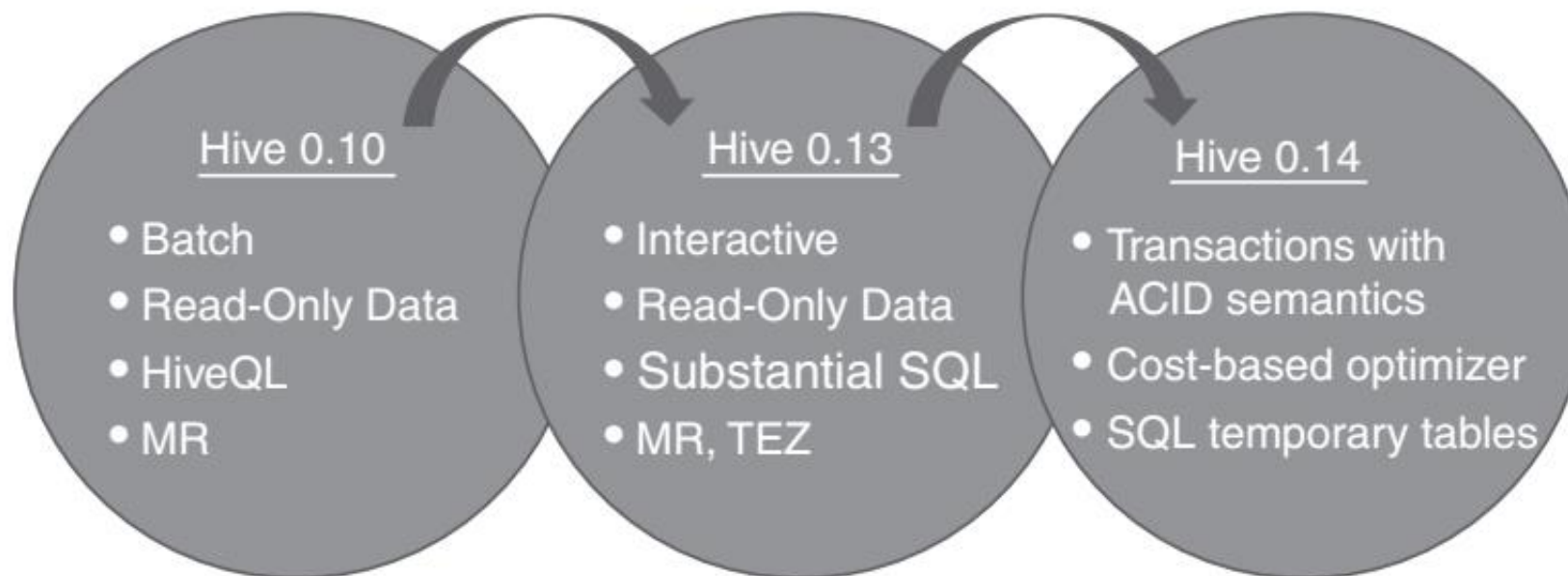
Data operations can be performed using a SQL interface called [HiveQL](#).

---

Hive brings in SQL capability on top of Hadoop, making it a horizontally scalable database and a great choice for DWH environments.



**Figure 9.2** History of Hive.



# Journey from Facebook to Apache Hadoop :

- Hive (which later became Apache) was initially developed by Facebook when they found their data growing exponentially from GBs to TBs in a matter of days.
- Facebook loaded their data into RDBMS databases using Python. Performance and scalability quickly became issues for them, since RDBMS databases can only scale vertically.

# Journey from Facebook to Apache Hadoop :

- Facebook needed a database that could scale horizontally and handle really large volumes of data.
- Hadoop was already popular by then; shortly afterward, Hive, which was built on top of Hadoop, came along.
- **Note:** Hive is similar to an RDBMS database, but it is not a complete RDBMS.

Remember :

Hive is not RDBMS.

It is not designed to support OLTP (Online Transaction Processing).

It is not designed for real-time queries.

It is not designed to support row-level updates.



# Features of Hive:

It is similar to SQL.

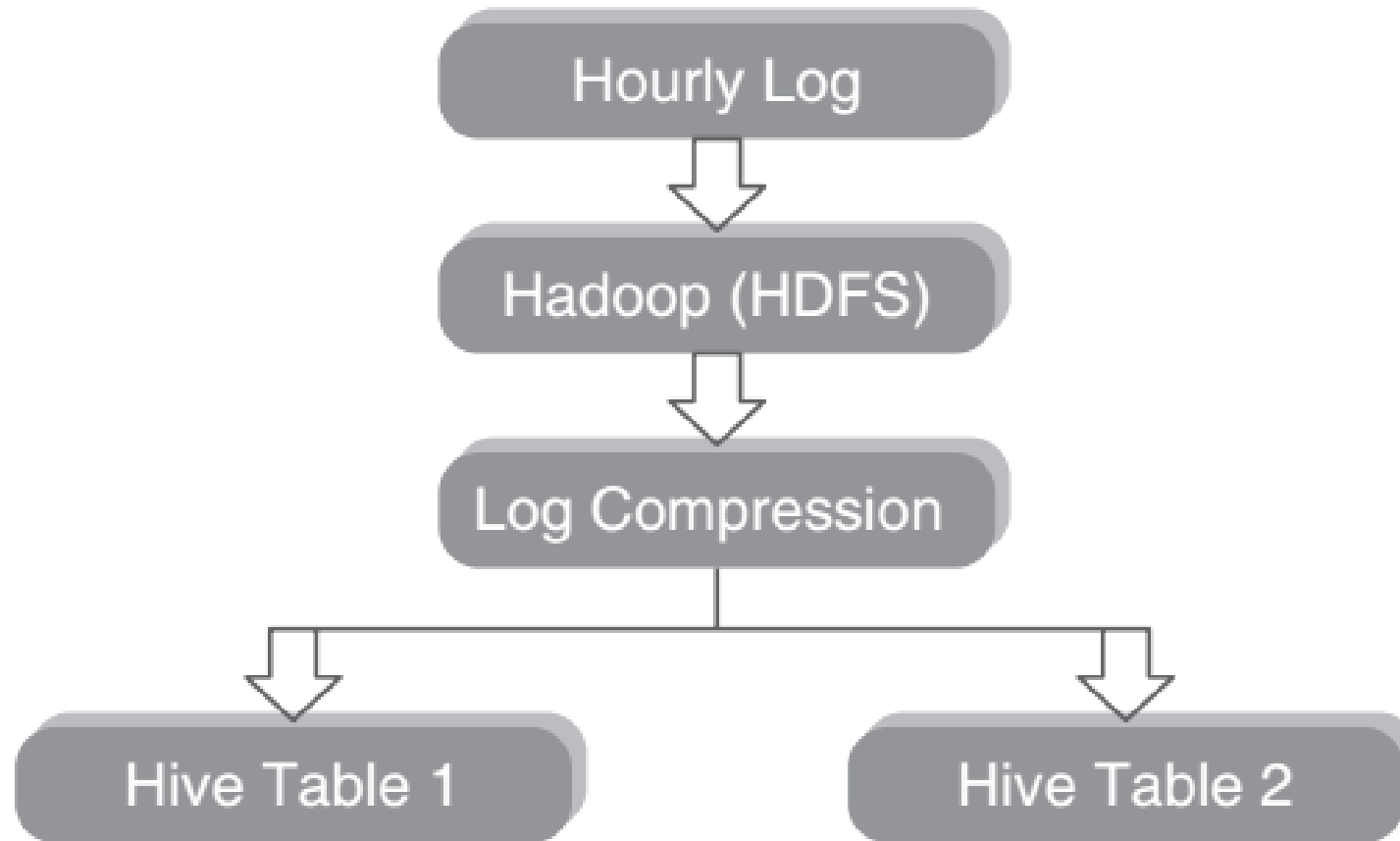
HQL is easy to code.

Hive supports rich data types such as structs, lists and maps.

Hive supports SQL filters, group-by and order-by clauses.

Custom Types, Custom Functions can be defined.

# Hive Integration and Work Flow:



# Hive Data Units :

- **Databases**: The namespace for tables.
- **Tables**: Set of records that have similar schema.
- **Partitions**: Logical separations of data based on classification of given information as per specific attributes. Once hive has partitioned the data based on a specified key, it starts to assemble the records into specific folders as and when the records are inserted.
- **Buckets (or Clusters)**: Similar to partitions but uses hash function to segregate data and determines the cluster or bucket into which the record should be placed.

“XYZ Corp” has their customer base spread across 190+ countries. There are 5 million records/entities available. If it is required to fetch the entities pertaining to a particular country, in the absence of partitioning, there is no choice but to go through all of the 5 million entities. This despite the fact our

query will eventually result in few thousand entities of the particular country. However, creating partitions based on country will greatly help to alleviate the performance issue by checking the data belonging to the partition for the country in question.

# Hive Data Types :

## **Numeric Data Type**

TINYINT	1-byte signed integer
SMALLINT	2-byte signed integer
INT	4-byte signed integer
BIGINT	8-byte signed integer
FLOAT	4-byte single-precision floating-point
DOUBLE	8-byte double-precision floating-point number

.....

# Hive Data Types :

## String Types

STRING

VARCHAR

Only available starting with Hive 0.12.0

CHAR

Only available starting with Hive 0.13.0

**Strings can be expressed in either single quotes (') or double quotes (")**

# Hive Data Types :

---

## Miscellaneous Types

BOOLEAN

BINARY

Only available starting with Hive

---

---

## Collection Data Types

---

STRUCT    Similar to 'C' struct. Fields are accessed using dot notation. E.g.: struct('John', 'Doe')

MAP        A collection of key-value pairs. Fields are accessed using [] notation. E.g.: map('first', 'John', 'last', 'Doe')

ARRAY     Ordered sequence of same types. Fields are accessed using array index. E.g.: array('John', 'Doe')

---



# Hive File Format:

- **Text File:**

The supported text files are CSV and TSV. JSON or XML documents too can be specified as text file.

- **Sequential File**

Sequential files are flat files that store binary key–value pairs. It includes compression support which reduces the CPU, I/O requirements.

# Hive File Format:

- RCFile (Record Columnar File)

RCFile stores the data in Column Oriented Manner which ensures that Aggregation operation is not an expensive operation.

# RCFile (Record Columnar File)

C1	C2	C3	C4
11	12	13	14
21	22	23	24
31	32	33	34
41	42	43	44
51	52	53	54

TABLE-1

Row Group 1	Row Group 2
11, 21, 31;	41, 51;
12, 22, 32;	42, 52;
13, 23, 33;	43, 53;
14, 24, 34;	44, 54;

TABLE-3

Row Group 1			
C1	C2	C3	C4
11	12	13	14
21	22	23	24
31	32	33	34

Row Group 2			
C1	C2	C3	C4
41	42	43	44
51	52	53	54

TABLE-2