Topic 7: Hive - SQL on Hadoop

1. What is Hive?

Core Definition

- **Software system** providing **tabular view** of data stored in HDFS
- **SQL-like methods** for manipulating HDFS data
- Started at **Facebook in 2010**
- High-level interface to HDFS

Key Concept

- **HQL (Hive Query Language)**: Implements SQL-92 standard (almost)
- **Frees analysts** from Java MapReduce programming (mostly)
- **HQL statements** → Parsed by Hive client → Translated to **MapReduce jobs** → Processed by Hadoop

Hive vs Pig

- **Hive**: SQL-like abstractions (declarative)
- **Pig**: Data flow language (procedural)

2. Hive Architecture

Prerequisites

- **Hadoop and HDFS** must be running
- Cannot operate independently

Core Components

1. Metastore

Definition: Relational database storing metadata

Contents:

- **Mappings**: Tables → HDFS directory locations
- **Input/Output formats**: CSV, TextFile, ORC, Parquet, etc.
- **SerDes**: Serialization/Deserialization functions
- **Table schemas**: Column names, types, properties

Implementation:

- **Default**: Derby (embedded relational database)
- **Production**: MySQL, PostgreSQL (recommended)
- **Read/written** by Hive client

Purpose:

- Extract records and fields from files
- Map logical tables to physical storage

2. Hive Client

- Accepts and parses HQL commands
- Connects to metastore
- Generates MapReduce jobs
- Returns results to user or saves to HDFS

Data Organization

Top level view:

- **Databases**: Collection of related tables
- **Tables**: Structured data view
- **Partitions**: Subdivisions of tables (optional)
- **Buckets**: Further subdivisions (optional)

3. Hive Interfaces

1. Command Line Interface (CLI)

- **Commands**: `hive` or `beeline` (newer)
- Accepts HQL statements
- Interactive or batch mode

2. JDBC/ODBC Connectors

Compatible tools:

- Beeline (CLI)
- Oracle SQL Developer (GUI)
- Talend Open Studio (ETL tools)
- Jasper Reports, QlikView (BI tools)
- Microsoft Excel 2013 (analysis)
- Tableau (visualization)

3. Storage Handler

- Integrates with **HBase**
- Allows HQL queries on HBase tables

4. HUE (Hadoop User Experience)

- **Unified web interface** to HDFS and Hive
- Interactive query environment
- Visual query builder

5. HCatalog

- **Metadata management system**
- Works with Hadoop, Pig, Hive, and MapReduce
- Unified metadata layer

4. HQL Components

1. Data Definition Language (DDL)

Purpose: Creating, deleting, altering schema objects

Objects:

- Databases
- Tables
- Views
- Partitions
- Buckets

2. Data Selection and Scope Language

Purpose: Querying and filtering data

Operations:

- SELECT queries
- JOIN operations
- WHERE clauses
- Data range limiting

3. Data Manipulation Language (DML)

Purpose: Moving and transforming data

Operations:

- LOAD DATA
- INSERT
- UPDATE (limited)
- Sorting
- Distributing

4. Data Aggregation and Sampling Language

Purpose: Analyzing data

Operations:

- GROUP BY
- Aggregate functions (SUM, COUNT, AVG, etc.)
- HAVING clauses
- Sampling techniques

5. Hive Data Types

Primitive Data Types

Type	Size	Example
TINYINT	1 byte	10Y
SMALLINT	2 bytes	10S
INT	4 bytes	10
BIGINT	8 bytes	10L
FLOAT	4 bytes	0.1234567
DOUBLE	8 bytes	0.1234567891234
DECIMAL(m,n)	variable	3.14
BINARY	n bytes	1011001
BOOLEAN	1 byte	TRUE
STRING	up to 2GB	'Abcdef'
CHAR	up to 255 bytes	'Hello'
VARCHAR	variable	'Hive'
DATE	-	'2017-05-03'
TIMESTAMP	-	'2017-05-03 15:10:00.345'

Complex Data Types

ARRAY

```
-- Definition
column_name ARRAY<string>

-- Example
['Hadoop', 'Pig', 'Hive']

-- Access
bigdata[1]  -- Returns 'Pig' (0-indexed)
```

MAP

```
-- Definition
column_name MAP<int, string>

-- Example
```

```
{'k1':'Hadoop', 'k2':'Pig'}

-- Access
bigdata['k2']  -- Returns 'Pig'
```

STRUCT

```
-- Definition
column_name STRUCT<a:string, b:int, c:double>

-- Example
{name:'Hadoop', age:24, salary:50000.06}

-- Access
bigdata.name  -- Returns 'Hadoop'
```

Complex Type Example

```
CREATE TABLE types(
   array_col ARRAY<string>,
   map_col MAP<int, string>,
   struct_col STRUCT<a:string, b:int, c:double>
);

INSERT INTO types
SELECT array('bolt', 'nut', 'screw'),
       map(1,'bolt', 2,'nut', 3,'screw'),
       named_struct('a','bolt', 'b',5, 'c',0.5)
FROM DUAL;
```

6. Hive Databases

Database Basics

- **Collection** of conceptually related tables
- **Implemented** as folder/directory in HDFS
- **Default database**: `/user/hive/warehouse`
- **Custom database**: `/user/hive/warehouse/<dbname>.db`

Database Operations

```
-- Create database
CREATE DATABASE tpchr;

-- Describe database
DESCRIBE DATABASE tpchr;
```

```
-- Use database (make current)
USE tpchr;

-- Drop database
DROP DATABASE tpchr;
```

Example

```
CREATE DATABASE tpchr;
-- Creates: /user/hive/warehouse/tpchr.db
```

7. Hive Tables

Internal (Managed) Tables

Characteristics:

- **Created and managed** by Hive in HDFS
- **Lifecycle** fully managed by Hive
- **Location**: Default warehouse directory
- **Deletion**: Both data and metadata removed

Create Internal Table:

```
CREATE TABLE IF NOT EXISTS intregion(
   R_REGIONKEY DECIMAL(12),
   R_NAME VARCHAR(25),
   R_COMMENT VARCHAR(152)
)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY '|'
STORED AS TEXTFILE;
```

Load Data:

```
LOAD DATA LOCAL INPATH 'region.tbl'
INTO TABLE intregion;
```

External Tables

Characteristics:

- **Data already exists** in HDFS
- **Location** specified explicitly

- **Deletion**: Only metadata removed, **data preserved**
- **Use case**: Share data across multiple systems

Create External Table:

```
CREATE EXTERNAL TABLE IF NOT EXISTS extregion(
   R_REGIONKEY DECIMAL(12),
   R_NAME VARCHAR(25),
   R_COMMENT VARCHAR(152)
)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY '|'
STORED AS TEXTFILE
LOCATION '/user/tpchr/region';
```

Load Existing HDFS File:

```
# First, put file in HDFS
hadoop fs -mkdir /user/tpchr/nation
hadoop fs -put nation.tbl /user/tpchr/nation

# Then create external table pointing to it
CREATE EXTERNAL TABLE extnation(
   N_NATIONKEY DECIMAL(12),
   N_NAME VARCHAR(25),
   N_COMMENT VARCHAR(152)
)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY '|'
STORED AS TEXTFILE
LOCATION '/user/tpchr/nation';
```

Internal vs External Tables

Aspect	Internal	External
Data Management	Hive manages	User manages
Location	Warehouse directory	User-specified
Drop Behavior	Deletes data	Preserves data
Use Case	Hive-controlled	Shared data

8. Partitions

Purpose

- **Eliminate unnecessary scans** of entire table

- **Access only required fragments** of data
- **Performance optimization**

How It Works

- Partition = **Subdirectory** in HDFS
- Based on **predefined columns**
- Only searched partitions accessed

Create Partitioned Table

```
CREATE TABLE IF NOT EXISTS part(
   P_PARTKEY DECIMAL(12),
   P_NAME VARCHAR(55),
   P_TYPE VARCHAR(25),
   P_SIZE DECIMAL(12),
   P_COMMENT VARCHAR(23)
)
PARTITIONED BY (P_BRAND VARCHAR(20))
ROW FORMAT DELIMITED
FIELDS TERMINATED BY '|'
STORED AS TEXTFILE;
```

Partition Operations

```
-- Add partition
ALTER TABLE part ADD PARTITION (P_BRAND='GoldenBolts');

-- Show partitions
SHOW PARTITIONS part;

-- Load data into partition
LOAD DATA LOCAL INPATH 'part.txt'
OVERWRITE INTO TABLE part
PARTITION (P_BRAND='GoldenBolts');
```

HDFS Structure

```
/user/hive/warehouse/part/
└── p_brand=GoldenBolts/
    └── data files
```

9. Buckets

Purpose

- **Further divide data** within partitions
- **Speed up processing** through hash-based distribution
- **Improve join performance**

How It Works

- Bucket column values **hashed** into fixed number of buckets
- Each bucket = **segment of file** in HDFS
- Data distributed evenly across buckets

Create Bucketed Table

```
CREATE TABLE customer(
   C_CUSTKEY DECIMAL(12),
   C_NAME VARCHAR(25),
   C_PHONE CHAR(15),
   C_ACCTBAL DECIMAL(12,2)
)
CLUSTERED BY (C_CUSTKEY) INTO 2 BUCKETS
ROW FORMAT DELIMITED
FIELDS TERMINATED BY '|';
```

Enable Bucketing

```
SET mapreduce.job.reduces = 2;
SET hive.enforce.bucketing = true;
```

Load Bucketed Table

```
INSERT INTO customer VALUES
(1, 'Customer#000000001', '25-989-741-2988', 711.56),
(2, 'Customer#000000002', '23-768-687-3665', 121.65),
(3, 'Customer#000000003', '11-719-748-3364', 7498.12);
```

10. Views

Characteristics

- **Logical data structures** (not materialized)
- **Simplify complex queries**
- **Do not store data**
- **Schema frozen** at creation time

Create View

```
CREATE VIEW vcustomer AS
SELECT C_CUSTKEY, C_NAME, C_PHONE
FROM CUSTOMER
WHERE C_CUSTKEY < 5;
```

Important Notes

- Changes in base tables **not reflected** in view schema
- Must recreate view after schema changes
- Performance: No improvement over base query

11. Hive vs Relational DBMS

Similarities

1. **Tabular view** of data
2. **Directories and files** viewed as tables
3. **Column types** defined
4. **SQL-like** access through HQL
5. **JDBC** programming interface

Differences

Aspect	Relational DBMS	Hive
Data Model	Normalized, structured	Denormalized, schema-on-read
Updates	Fine-grained (row-level)	Coarse-grained (partition-level)
Transactions	ACID guaranteed	No transactions (pre-Hive 3)
Constraints	PK, FK, CHECK enforced	No constraint enforcement
Storage	Proprietary format	Files in HDFS
Access	Exclusive through DBMS	Direct HDFS access possible
Performance	Optimized for OLTP	Optimized for batch analytics
Latency	Milliseconds	Seconds to minutes
Data Management	DBMS controls	Load and read-only (mostly)

Key Points for Exam

Core Concepts

1. **Hive = SQL on Hadoop**
2. **HQL → MapReduce**: HQL translated to MapReduce jobs
3. **Metastore**: Relational DB storing metadata
4. **Schema-on-read**: Interpret data at query time

5. **NOT a database**: Data warehouse system

Critical Architecture

- **Metastore**: Derby (default) or MySQL (production)
- **Hive Client**: Parses HQL, generates MapReduce
- **HDFS**: Stores actual data
- **Hadoop**: Executes MapReduce jobs

Data Types to Remember

- **Primitive**: TINYINT, INT, BIGINT, FLOAT, DOUBLE, STRING, DATE, TIMESTAMP
- **Complex**: ARRAY, MAP, STRUCT

Tables

Internal (Managed):

- Hive manages lifecycle
- Data deleted on DROP

External:

- User manages data
- Data preserved on DROP

Performance Optimization

1. **Partitions**: Eliminate unnecessary scans
2. **Buckets**: Hash-based distribution
3. **File formats**: ORC, Parquet (compressed, columnar)

Important Clauses

- **PARTITIONED BY**: Create partitions
- **CLUSTERED BY**: Create buckets
- **ROW FORMAT DELIMITED**: Specify delimiters
- **STORED AS**: Specify file format
- **LOCATION**: Specify HDFS path (external tables)

Key Differences from SQL

- No transactions (traditional)
- No row-level updates (traditional)
- No constraints enforcement
- Optimized for batch, not OLTP
- Higher latency than RDBMS

When to Use Hive

- Large-scale data analysis

- Batch processing
- Data warehousing
- ETL operations
- Users know SQL but not Java

When NOT to Use Hive

- Real-time queries (use Impala, Presto)
- Transactional workloads (use HBase)
- Row-level updates (use HBase)
- Low-latency requirements