

Lab 3: Introduction to Hive - Detailed Report

Course: ISIT312/ISIT912 Big Data Management

Semester: Spring 2023

Student: [Your Name]

Date: October 17, 2025

Executive Summary

This laboratory exercise introduced Apache Hive, a data warehouse infrastructure built on top of Hadoop. The lab covered fundamental operations including starting Hive services, creating and managing databases, working with internal and external tables, handling complex data types (arrays, maps, structs), and understanding how Hive maps relational tables to HDFS files.

Part 1: Starting Hive Services

1.1 Verifying Hive Installation

```
bash
echo $HIVE_HOME
echo $HIVE_CONF_DIR
```

These commands confirmed the Hive installation directory and configuration folder location.

1.2 Starting Metastore Service

Terminal 1:

```
bash
$HIVE_HOME/bin/hive --service metastore
```

The metastore service started successfully, managing metadata for Hive tables.

1.3 Starting HiveServer2

Terminal 2:

```
bash
$HIVE_HOME/bin/hiveserver2
```

HiveServer2 started successfully, providing the JDBC/ODBC interface for client connections.

1.4 Connecting via Beeline

Terminal 3:

```
bash  
$HIVE_HOME/bin/beeline
```

Connection command:

```
sql  
!connect jdbc:hive2://localhost:10000
```

Pressed Enter for both username and password prompts. Connection established successfully with prompt:

```
0: jdbc:hive2://localhost:10000>
```

Part 2: Database Operations

2.1 Viewing Existing Databases

```
sql  
show databases;
```

Output:

```
+-----+  
| database_name |  
+-----+  
| default      |  
| isit312      |  
| lab          |  
+-----+
```

2.2 Creating a New Database

```
sql  
create database tpch;
```

Verified creation:

```
sql  
show databases;
```

Output:

```
+-----+  
| database_name |  
+-----+  
| default      |  
| isit312      |  
| lab          |  
| tpchr        |  
+-----+
```

2.3 Database Location in HDFS

```
bash  
\$SHADOOP\_HOME/bin/hadoop fs -ls /user/hive/warehouse
```

Result: Database created as folder [/user/hive/warehouse/tpchr.db](#)

2.4 Describing Database

```
sql  
describe database tpchr;
```

Output:

```
+-----+-----+-----+-----+-----+-----+  
| db_name | comment |           location           | owner_name | owner_type | parameters |  
+-----+-----+-----+-----+-----+-----+  
| tpchr  |       | hdfs://localhost:8020/user/hive/warehouse/tpchr.db | bigdata   | USER      |          |  
+-----+-----+-----+-----+-----+-----+
```

2.5 Switching Databases

```
sql  
use isit312;  
select current_database();
```

Output:

```
+-----+
| _c0 |
+-----+
| isit312 |
+-----+
```

Part 3: Working with Internal Tables

3.1 Creating Internal Table in isit312 Database

Table: orders

```
sql
describe orders;
```

Output:

```
+-----+-----+-----+
| col_name | data_type | comment |
+-----+-----+-----+
| part     | char(7)    |      |
| customer | varchar(30) |      |
| amount   | decimal(8,2) |      |
| oyear    | decimal(4,0) |      |
| omonth  | decimal(2,0) |      |
| oday     | decimal(2,0) |      |
+-----+-----+-----+
```

3.2 Inserting Data via INSERT Statement

```
sql
INSERT INTO orders VALUES ('P00001', 'John Smith', 1500.50, 2023, 10, 15);
```

Execution time: 18.915 seconds

Note: INSERT operations are slow in Hive as they trigger MapReduce jobs.

3.3 Verifying Data

```
sql
```

```
SELECT * FROM orders;
```

Output:

orders.part	orders.customer	orders.amount	orders.oyear	orders.omonth	orders.oday	
P00001	John Smith	1500.50	2023	10	15	
bolt	James	200.00	2016	1	1	
bolt	Peter	100.00	2017	1	30	
bolt	Bob	300.00	2018	5	23	
screw	James	20.00	2017	5	11	
screw	Alice	55.00	2018	1	1	
nut	Alice	23.00	2018	3	16	
washer	James	45.00	2016	4	24	
washer	Peter	100.00	2016	5	12	
bolt	James	200.00	2018	1	5	
bolt	Peter	100.00	2018	1	5	
	NULL	NULL	NULL	NULL	NULL	

3.4 HDFS File Structure

```
bash
```

```
$SHADOOP_HOME/bin/hadoop fs -ls /user/hive/warehouse/isit312.db/orders
```

Output:

```
Found 2 items
```

```
-rwxr-xr-x 1 bigdata supergroup 38 2025-10-17 20:42 /user/hive/warehouse/isit312.db/orders/000000_0
-rwxr-xr-x 1 bigdata supergroup 260 2025-05-15 11:52 /user/hive/warehouse/isit312.db/orders/orders.txt
```

Key Observation: The new INSERT created file `000000_0`, while existing data was in `orders.txt`.

3.5 Viewing File Content

```
bash
```

```
$SHADOOP_HOME/bin/hadoop fs -cat /user/hive/warehouse/isit312.db/orders/000000*
```

Output:

Part 4: Loading Data into Internal Tables

4.1 Creating names Table (First Attempt - Incorrect)

sql

```
create table names(
    first_name VARCHAR(30),
    last_name  VARCHAR(30),
    age        DECIMAL(3));
```

Data file (names.tbl):

```
James,Bond,35
Harry,Potter,16
Robin,Hood,120
```

sql

```
load data local inpath '/home/bigdata/Desktop/names.tbl' into table names;
select * from names;
```

Output (Incorrect):

```
+-----+-----+-----+
| names.first_name | names.last_name | names.age |
+-----+-----+-----+
| James,Bond,35   | NULL          | NULL      |
| Harry,Potter,16 | NULL          | NULL      |
| Robin,Hood,120  | NULL          | NULL      |
+-----+-----+-----+
```

Problem: Table created without specifying field delimiter.

4.2 Recreating names Table (Correct)

sql

```
DROP TABLE names;
```

```
CREATE TABLE names(
  first_name VARCHAR(30),
  last_name VARCHAR(30),
  age      DECIMAL(3))
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ','
STORED AS TEXTFILE;
```

sql

```
load data local inpath '/home/bigdata/Desktop/names.tbl' into table names;
select * from names;
```

Output (Correct):

names.first_name	names.last_name	names.age
James	Bond	35
Harry	Potter	16
Robin	Hood	120

Lesson Learned: Must specify **(ROW FORMAT DELIMITED FIELDS TERMINATED BY)** when loading delimited data files.

Part 5: External Tables

5.1 Preparing Data in HDFS

Terminal window:

```
bash
```

```
$SHADOOP_HOME/bin/hadoop fs -mkdir /user/bigdata/a-new-hdfs-folder
$SHADOOP_HOME/bin/hadoop fs -put /home/bigdata/Desktop/names.tbl /user/bigdata/a-new-hdfs-folder
$SHADOOP_HOME/bin/hadoop fs -ls /user/bigdata/a-new-hdfs-folder
```

5.2 Creating External Table

sql

```
create external table enames(
    first_name VARCHAR(30),
    last_name VARCHAR(30),
    age      DECIMAL(3))
row format delimited fields terminated by ','
stored as textfile location '/user/bigdata/a-new-hdfs-folder';
```

5.3 Querying External Table

```
sql
select * from enames;
```

Output:

```
+-----+-----+-----+
| enames.first_name | enames.last_name | enames.age |
+-----+-----+-----+
| James          | Bond           | 35       |
| Harry          | Potter          | 16       |
| Robin          | Hood           | 120      |
+-----+-----+-----+
```

5.4 Comparing Internal vs External Tables

Verification:

```
bash
$SHADOOP_HOME/bin/hadoop fs -ls /user/hive/warehouse/
```

Result: Only internal tables (not `enames`) appear in warehouse folder.

5.5 Dropping External Table

```
sql
drop table enames;
```

Verification:

```
bash
$SHADOOP_HOME/bin/hadoop fs -ls /user/bigdata/a-new-hdfs-folder
$SHADOOP_HOME/bin/hadoop fs -cat /user/bigdata/a-new-hdfs-folder/names.tbl
```

Key Finding: External table metadata deleted, but HDFS data file remains intact.

5.6 Dropping Internal Table

```
sql
```

```
drop table names;
```

Verification:

```
bash
```

```
$SHADOOP_HOME/bin/hadoop fs -ls /user/hive/warehouse/
```

Key Finding: Both metadata and HDFS data deleted for internal tables.

Part 6: Complex Data Types

6.1 Array Data Type

6.1.1 Creating friend Table

```
sql
```

```
use default;
```

```
create table friend(
  name varchar(30),
  friends array<string> )
row format delimited
fields terminated by '|'
collection items terminated by ','
stored as textfile;
```

6.1.2 Data File (friend.tbl)

```
James|Kate,John
John|
Kate|
Harry|James
```

6.1.3 Loading Data

```
sql
```

```
load data local inpath '/home/bigdata/Desktop/friend.tbl' into table friend;
```

6.1.4 Querying Array Data

```
sql
```

```
select * from friend;
```

Output:

```
+-----+-----+
| friend.name | friend.friends |
+-----+-----+
| James      | ["Kate","John"] |
| John       | []          |
| Kate       | []          |
| Harry      | ["James"]   |
+-----+-----+
```

6.1.5 Accessing Array Elements

```
sql
```

```
select name, friends[0], friends[1], friends[2]
from friend;
```

Output:

```
+-----+-----+-----+
| name | c1  | c2  | c3  |
+-----+-----+-----+
| James | Kate | John | NULL |
| John  | NULL | NULL | NULL |
| Kate  | NULL | NULL | NULL |
| Harry | James | NULL | NULL |
+-----+-----+-----+
```

Key Observations:

- Array indexing starts at 0
- Empty arrays represented as `[]`
- Non-existent indices return NULL

6.2 Struct Data Type

6.2.1 Creating employee Table

```
sql

create table employee(
    number decimal(7),
    address struct<city:string,street:string,house:int,flat:int> )
row format delimited
fields terminated by '|'
collection items terminated by ','
stored as textfile;
```

6.2.2 Data File (employee.tbl)

```
007|London,Victoria St.,7,77
123|Dapto,Station St.,1,0
```

6.2.3 Loading Data

```
sql

load data local inpath '/home/bigdata/Desktop/employee.tbl' into table employee;
```

Note: The session ended before completing struct queries, but the table was created successfully.

Part 7: Cross-Database Operations

7.1 Creating Table in Specific Database

```
sql

create table tpchr.hello(message varchar(50));
insert into tpchr.hello values( 'Hello James !');
```

Execution time: 12.15 seconds

7.2 Verifying Table Location

```
sql
```

```
use tpchr;
select current_database();
show tables;
```

Output:

```
+-----+
| _c0 |
+-----+
| tpchr |
+-----+

+-----+
| tab_name |
+-----+
| hello   |
+-----+
```

7.3 HDFS Verification

```
bash
```

```
$SHADOOP_HOME/bin/hadoop fs -ls /user/hive/warehouse/tpchr.db/
```

Output:

```
Found 1 item
drwxr-xr-x - bigdata supergroup 0 2025-10-17 21:11 /user/hive/warehouse/tpchr.db/hello
```

Key Findings and Observations

1. Performance Characteristics

- **INSERT statements:** Very slow (12-19 seconds per row) due to MapReduce overhead
- **LOAD DATA:** Much faster for bulk loading, recommended approach for Hive

2. Internal vs External Tables

Aspect	Internal Table	External Table
Data location	/user/hive/warehouse/<db>/<table>	User-specified location
DROP behavior	Deletes both metadata and data	Deletes only metadata

Aspect	Internal Table	External Table
Use case	Hive-managed data	Shared data with other tools

3. Complex Data Types

- **Arrays:** Support indexing with `[n]` syntax, useful for multi-valued attributes
- **Structs:** Support nested structures accessed with dot notation
- **Maps:** Support key-value pairs (not fully tested in this session)

4. Delimiter Specification

Critical to specify delimiters when creating tables:

- `FIELDS TERMINATED BY` - separates columns
- `COLLECTION ITEMS TERMINATED BY` - separates array/map elements
- `MAP KEYS TERMINATED BY` - separates keys from values in maps

5. Database Management

- Databases implemented as folders in HDFS with `.db` suffix
 - Multiple databases can contain tables with identical names
 - `USE` command switches active database context
-

Challenges Encountered

1. **Initial delimiter mistake:** First `names` table created without delimiter specification, resulting in incorrect parsing
 2. **Session management:** Multiple beeline connections shown in logs, requiring careful tracking of active database context
 3. **Temporary tables:** A temporary table `values_tmp_table_1` appeared after INSERT operations
-

Conclusions

This laboratory successfully demonstrated:

1. **Hive Architecture:** Understanding of Metastore and HiveServer2 components
2. **Table Types:** Practical differences between internal and external tables
3. **Data Loading:** Two methods (INSERT vs LOAD DATA) with significant performance differences

4. Complex Types: Ability to work with arrays and structs for denormalized data

5. HDFS Integration: How Hive abstracts HDFS files as relational tables

Best Practices Identified:

- Use LOAD DATA for bulk inserts
 - Always specify delimiters in table definitions
 - Use external tables for shared data sources
 - Prefer complex types over multiple normalized tables for nested data
-

Appendix: Complete Command Reference

Database Commands

```
sql  
  
show databases;  
create database <name>;  
describe database <name>;  
use <database>;  
drop database <name>;  
select current_database();
```

Table Commands

```
sql  
  
show tables;  
describe <table>;  
create table <name>(...);  
create external table <name>(...) location '<path>';  
drop table <name>;
```

Data Manipulation

```
sql  
  
insert into <table> values (...);  
load data local inpath '<file>' into table <table>;  
select * from <table>;
```

HDFS Commands

```
bash
```

```
$SHADOOP_HOME/bin/hadoop fs -ls <path>
$SHADOOP_HOME/bin/hadoop fs -cat <file>
$SHADOOP_HOME/bin/hadoop fs -mkdir <path>
$SHADOOP_HOME/bin/hadoop fs -put <local> <hdfs>
```

End of Report