Hive interview Questions

Q1 What is the definition of Hive? What is the present version of Hive?

Ans:- Hive is a distributed, fault-tolerant data warehouse system that enables analytics at a massive scale.

The present version of hive is- 16 November 2022: release 4.0.0-alpha-2 available

- This release works with Hadoop 3.3.1

At present Apache hive version **4.0 is available now.**

Q2. Is Hive suitable to be used for OLTP systems? Why?

Ans:- Hive doesn't support OLTP. Hive supports Online Analytical Processing (OLAP), but not Online Transaction Processing (OLTP) because hive does not provide insert and update at row level.

Q3. How is HIVE different from RDBMS? Does hive support ACID

transactions. If not then give the proper reason.

Ans:- RDBMS:
RDBMS stands for Relational Database Management System. RDBMS is a such type of database management system which is specifically designed for relational databases. RDBMS is a subset of DBMS. A relational database refers to a database that stores data in a structured format using rows and columns and that structured form is known as table. There are some certain rules defined in RDBMS and that are known as Codd's rule.
Hive**:**
Hive is a data warehouse software system that provides data query and analysis. Hive gives an interface like SQL to query data stored in various databases and file systems that integrate with Hadoop. Hive helps with querying and managing large datasets real fast. It is an ETL tool for Hadoop ecosystem.

RDBMS	Hive
It is used to maintain database.	It is used to maintain data warehouse.
It uses SQL (Structured Query Language).	It uses HQL (Hive Query Language).
Schema is fixed in RDBMS.	Schema varies in it.
Normalized data is stored.	Normalized and de-normalized both type of data is stored.
Tables in rdms are sparse.	Table in hive are dense.
It doesn't support partitioning.	It supports automation partition.
No partition method is used.	Sharding method is used for partition.

Below are some of the limitations of using Hive ACID transactions.

- To support ACID, Hive tables should be created with **TRANSACTIONAL** table property.
- Currently, Hive supports ACID transactions on tables that store **ORC** file format.
- Enable ACID support by setting transaction manager to **DbTxnManager**
- Transaction tables cannot be accessed from the non-ACID Transaction Manager (**DummyTxnManager**) session.
- **External tables** cannot be created to support ACID since the changes on external tables are beyond Hive control.
- **LOAD** is not supported on ACID transactional Tables. hence use **INSERT INTO**.
- On the Transactional session, all operations are auto commit as **BEGIN**, **COMMIT**, and **ROLLBACK** are not yet supported.

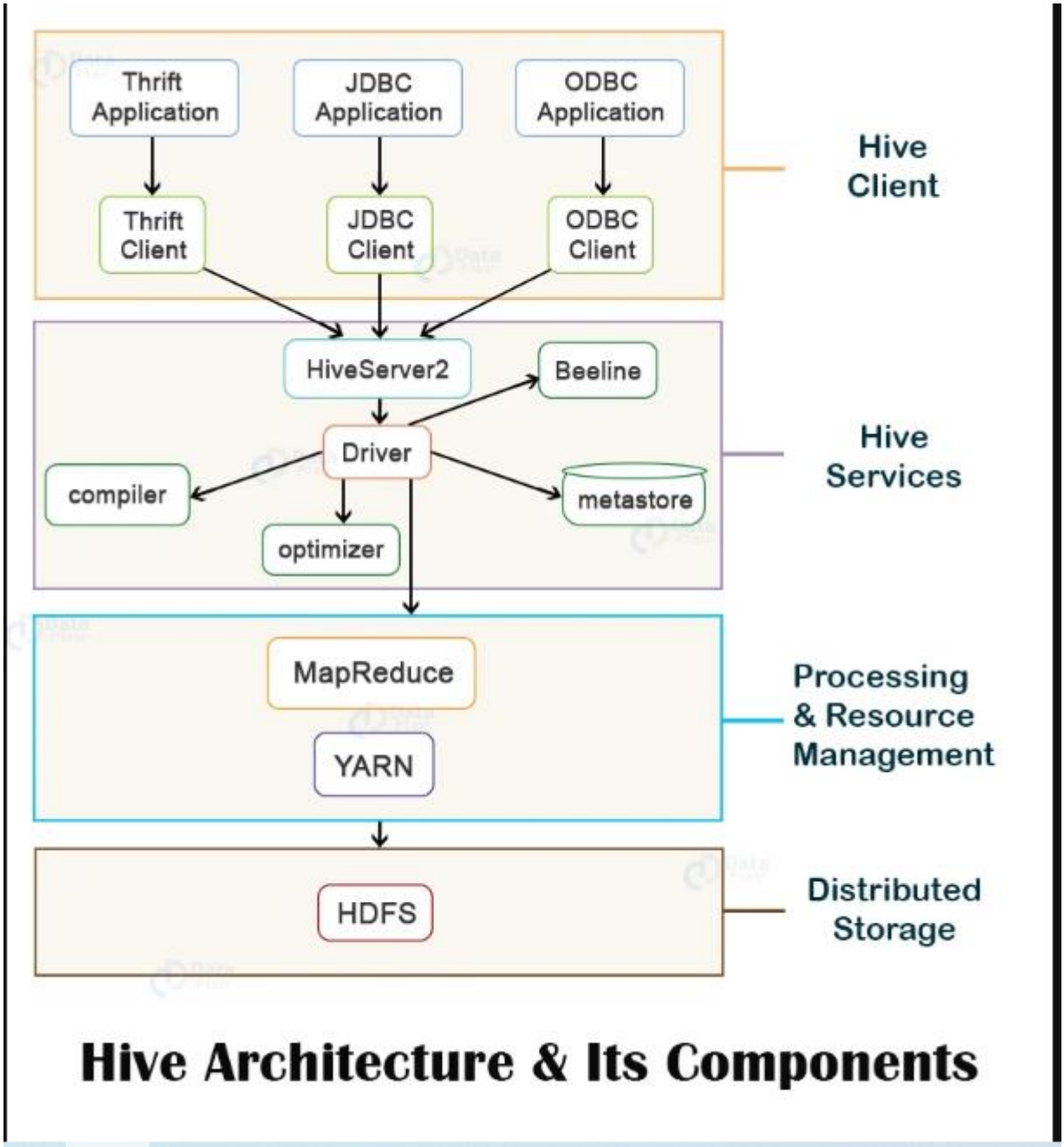
Hive supports ACID (Atomicity, Consistency, Isolation, and Durability) transactions. ACID transactions are provided at row levels. Following are the options Hive uses to support ACID transactions:

- Insert

- Delete
- Update

Q4. Explain the hive architecture and the different components of a Hive architecture?

Ans:-



The above figure shows the architecture of Apache Hive and its major components. The major components of Apache Hive are:

1. **Hive Client**
2. **Hive Services**
3. **Processing and Resource Management**
4. **Distributed Storage**

[ps2id id='Hive-Client' target="/]Hive Client

Hive supports applications written in any language like Python, Java, C++, Ruby, etc. using JDBC, ODBC, and Thrift drivers, for performing queries on the Hive. Hence, one can easily write a hive client application in any language of its own choice.

Hive clients are categorized into three types:

1. Thrift Clients

The Hive server is based on Apache Thrift so that it can serve the request from a thrift client.

2. JDBC client

Hive allows for the Java applications to connect to it using the JDBC driver. JDBC driver uses Thrift to communicate with the Hive Server.

3. ODBC client

Hive ODBC driver allows applications based on the ODBC protocol to connect to Hive. Similar to the JDBC driver, the ODBC driver uses Thrift to communicate with the Hive Server.

[ps2id id='Hive-Services' target="/]Hive Service

To perform all queries, Hive provides various services like the Hive server2, Beeline, etc. The various services offered by Hive are:

1. Beeline

The Beeline is a command shell supported by HiveServer2, where the user can submit its queries and command to the system. It is a **JDBC** client that is based on **SQLLINE CLI** (pure Java-console-based utility for connecting with relational databases and executing SQL queries).

2. Hive Server 2

HiveServer2 is the successor of HiveServer1. HiveServer2 enables clients to execute queries against the Hive. It allows multiple clients to submit requests to Hive and retrieve the final results. It is basically designed to provide the best support for open API clients like JDBC and ODBC.

Note: Hive server1, also called a Thrift server, is built on Apache Thrift protocol to handle the cross-platform communication with Hive. It allows different client applications to submit requests to Hive and retrieve the final results.
It does not handle concurrent requests from more than one client due to which it was replaced by HiveServer2.

3. Hive Driver

The Hive driver receives the **HiveQL** statements submitted by the user through the command shell. It creates the session handles for the query and sends the query to the compiler.

4. Hive Compiler

Hive compiler parses the query. It performs semantic analysis and type-checking on the different query blocks and query expressions by using the metadata stored in metastore and generates an execution plan.

The execution plan created by the compiler is the **DAG(Directed Acyclic Graph)**, where each stage is a map/reduce job, operation on HDFS, a metadata operation.

5. Optimizer

Optimizer performs the transformation operations on the execution plan and splits the task to improve efficiency and scalability.

6. Execution Engine

Execution engine, after the compilation and optimization steps, executes the execution plan created by the compiler in order of their dependencies using Hadoop.

7. Metastore

Metastore is a central repository that stores the metadata information about the structure of tables and partitions, including column and column type information.

It also stores information of serializer and deserializer, required for the read/write operation, and HDFS files where data is stored. This metastore is generally a relational database.

Metastore provides a Thrift interface for querying and manipulating Hive metadata.

We can configure metastore in any of the two modes:

- **Remote:** In remote mode, metastore is a Thrift service and is useful for non-Java applications.
- **Embedded:** In embedded mode, the client can directly interact with the metastore using JDBC.

8. HCatalog

HCatalog is the table and storage management layer for Hadoop. It enables users with different data processing tools such as Pig, MapReduce, etc. to easily read and write data on the grid.

It is built on the top of Hive metastore and exposes the tabular data of Hive metastore to other data processing tools.

9. WebHCat

WebHCat is the REST API for HCatalog. It is an HTTP interface to perform Hive metadata operations. It provides a service to the user for running Hadoop MapReduce (or YARN), Pig, Hive jobs.

[ps2id id=’Processing-and-Resource-Management’ target=”/]Processing Framework and Resource Management

Hive internally uses a **MapReduce** framework as a defacto engine for executing the queries.

MapReduce is a software framework for writing those applications that process a massive amount of data in parallel on the large clusters of commodity hardware. MapReduce job works by splitting data into chunks, which are processed by map-reduce tasks.

[ps2id id='Distributed-Storage' target="/]Distributed Storage

Hive is built on top of Hadoop, so it uses the underlying Hadoop Distributed File System for the distributed storage.

Q 5. Mention what Hive query processor does? And Mention what are the components of a Hive query processor?

Ans:- Hive query processor convert graph of MapReduce jobs with the execution time framework. So that the jobs can be executed in the order of dependencies.

Following are the components of a Hive Query Processor:

- Parse and Semantic Analysis (ql/parse)
- Metadata Layer (ql/metadata)
- Type Interfaces (ql/typeinfo)
- Sessions (ql/session)
- Map/Reduce Execution Engine (ql/exec)
- Plan Components (ql/plan)
- Hive Function Framework (ql/udf)
- Tools (ql/tools)
- Optimizer (ql/optimizer)

Q6. What are the three different modes in which we can operate Hive?

Ans:- Hive can be used in three different modes:

- **Local or Standalone Mode:** Hive runs as a single process on a single machine in Local Mode. This mode is appropriate for testing and development because it allows experimenting with Hive without the need for a cluster or additional resources.
- **Pseudo-distributed Mode:** In pseudo-distributed mode, Hive is run on a single machine, but the machine is configured to simulate a distributed environment by running multiple processes, such as NameNode, DataNode, ResourceManager, and NodeManager, on the same machine. This configuration allows Hive to take advantage of the parallel processing capabilities of Hadoop, making it useful for testing and development.
- **Fully-distributed Mode:** Distributed mode refers to the mode in which Hive is run on a cluster of machines. The data is stored in a distributed file system like HDFS, and the processing is done across multiple nodes in the cluster. Hive queries are translated into jobs that can be executed in parallel across the cluster, allowing for high performance and scalability. The three modes MapReduce, Tez, and Spark all fall under this type of distributed mode.
  - **MapReduce Mode:** In MapReduce Mode, Hive runs on a Hadoop cluster and processes queries using the Hadoop MapReduce framework. This mode is appropriate for production deployments and large-scale data processing because it allows Hive to leverage Hadoop's distributed processing capabilities.
  - **Tez Mode**: In Tez mode, Hive runs on a Hadoop cluster using the Tez execution engine, which is designed to provide high-performance data processing. Hive translates the queries into Tez tasks, which are then executed on the cluster. This mode is suitable for processing large datasets that require high-performance data processing and can benefit from the optimized execution plan provided by Tez.
  - **Spark mode:** Hive runs on a Spark cluster and processes queries using the Spark execution engine in Spark Mode. This mode is appropriate for deployments that use Spark as their primary processing framework because it enables Hive to benefit from Spark's advanced processing capabilities and performance optimizations.

Q7. Features and Limitations of Hive.

Ans:- # Features of Apache Hive

The Various key-features of Apache Hive are:

1. Open-source: Apache Hive is an open-source tool. We can use it free of cost.
2. Query large datasets: Hive can query and manage huge datasets stored in Hadoop Distributed File System.

3. Multiple-users: Multiple users can query the data using Hive Query Language simultaneously.
4. Backward compatible: Apache Hive perfectly fits the low level interface requirement of Apache Hadoop.
5. Partitioning and Bucketing: Apache Hive supports partitioning and bucketing of data at the table level to improve performance.
6. File-formats: Hive provides support for various file formats such as textFile, ORC, Avro Files, SequenceFile, Parquet, RCFile, LZO Compression etc.
7. Hive Query Language: Hive uses Hive Query Language which is similar to SQL. We do not require any knowledge of programming languages to work with Hive. Only the knowledge of basic SQL query is enough to work with Hive.
8. Built-In function: Hive provides various Built-In functions.
9. User-Defined Functions: It also provides support for User-Defined Functions for the tasks like data cleansing and filtering. We can define UDFs according to our requirements
10. External Table: Apache Hive supports external tables. This allows us to process data without actually storing data in HDFS.
11. Fast: Hive is a fast, scalable, extensible tool and uses familiar concepts.
12. Warehouse: Apache Hive is a distributed data warehouse tool.
13. Table Structure: Table structure in Hive is similar to table structure in RDBMS.
14. ETL support: Hive supports ETL operations. Hive is an effective ETL tool.
15. Storage: Hive allows us to access files stored in HDFS and other similar data storage systems such as HBase.
16. OLAP: Hive is designed for OLAP (Online Analytical Processing).
17. Client application: Hive can support client applications written in PHP, Python, Java, C++ and Ruby.
18. Rule Based Optimizer: Hive has a rule based optimizer for optimizing logical plans.
19. Ad-hoc queries: Hive allows us to run Ad-hoc queries which are the loosely typed command or query whose value depends on some variable for the data analysis.
20. Data Visualization: Hive can be used for Data Visualization. Integrating Hive with Apache Tez will provide the real time processing capabilities.

Limitations of Apache Hive

Some of the limitations of Apache Hive are:

1. Hive is not designed for the OLTP (Online transaction processing). We can use it for OLAP.

2. It does not offer real-time queries.

3. It provides limited subquery support.

4. Latency of Hive is generally very high.

Q8. How to create a Database in HIVE?

Ans:- Hive is a database technology that can define databases and tables to analyze structured data. The theme for structured data analysis is to store the data in a tabular manner, and pass queries to analyze it. This chapter explains how to create Hive database. Hive contains a default database named **default**.

Create Database Statement

Create Database is a statement used to create a database in Hive. A database in Hive is a **namespace** or a collection of tables. The **syntax** for this statement is as follows:

```
CREATE DATABASE|SCHEMA [IF NOT EXISTS] <database name>
```

Here, IF NOT EXISTS is an optional clause, which notifies the user that a database with the same name already exists. We can use SCHEMA in place of DATABASE in this command. The following query is executed to create a database named **userdb**:

```
hive> CREATE DATABASE [IF NOT EXISTS] userdb;
```

or

```
hive> CREATE SCHEMA userdb;
```

The following query is used to verify a databases list:

```
hive> SHOW DATABASES;
default
userdb
```

or

hive>create database <database name>;

hive>use <database name>;

hive> show databases;

default

<database name>

Q9. How to create a table in HIVE?

Ans:- Create Table Statement

Create Table is a statement used to create a table in Hive. The syntax and example are as follows:

Syntax

```
CREATE [TEMPORARY] [EXTERNAL] TABLE [IF NOT EXISTS] [db_name.] table_name

[(col_name data_type [COMMENT col_comment], ...)]
[COMMENT table_comment]
[ROW FORMAT row_format]
[STORED AS file_format]
```

Example

Let us assume you need to create a table named **employee** using **CREATE TABLE** statement. The following table lists the fields and their data types in employee table:

Sr.No	Field Name	Data Type
1	Eid	int
2	Name	String
3	Salary	Float
4	Designation	String

The following data is a Comment, Row formatted fields such as Field terminator, Lines terminator, and Stored File type.

```
COMMENT ‘Employee details’
FIELDS TERMINATED BY ‘\t’
LINES TERMINATED BY ‘\n’
STORED IN TEXT FILE
```

The following query creates a table named **employee** using the above data.

```
hive> CREATE TABLE IF NOT EXISTS employee ( eid int, name String,
salary String, destination String)
COMMENT ‘Employee details’
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ‘\t’
LINES TERMINATED BY ‘\n’
STORED AS TEXTFILE;
```

If you add the option IF NOT EXISTS, Hive ignores the statement in case the table already exists.

On successful creation of table, you get to see the following response:

```
OK
Time taken: 5.905 seconds
hive>
```

Syntax To Create Table in Hive

```
CREATE TABLE [IF NOT EXISTS] <table-name> (

<column-name>    <data-type>,

<column-name>    <data-type> COMMENT 'Your Comment',

<column-name>    <data-type>,

.

.

.

<column-name>    <data-type>

)

COMMENT 'Add if you want'

LOCATION 'Location On HDFS'

ROW FORMAT DELIMITED

FIELDS TERMINATED BY ',';
```

Note:

1. We can add a comment to the table as well as to each individual column.

2. ROW FORMAT DELIMITED shows that whenever a new line is encountered the new record entry will start.

3. FIELDS TERMINATED BY ',' shows that we are using ',' delimiter to separate each column.

4. We can also override the default database location with the *LOCATION* option.

So let's create the table *student_data* in our *student_detail* database with the help of the command shown below.

```
CREATE TABLE IF NOT EXISTS student_data(

Student_Name STRING COMMENT 'This col. Store the name of student',

Student_Rollno INT COMMENT 'This col. Stores the rollno of student',

Student_Marks FLOAT)

ROW FORMAT DELIMITED

FIELDS TERMINATED BY ',';
```

We have successfully created the **table student_data** in our **student_detail database** with 3 different fields Student_Name, Student_Rollno, Student_Marks as STRING, INT, FLOAT respectively.

```
hive> CREATE TABLE IF NOT EXISTS student_data (
    > Student_Name STRING COMMENT 'This col. Store the name of st
    > Student_Rollno INT COMMENT 'This col. Stores the rollno of
    > Student_Marks FLOAT)
    > ROW FORMAT DELIMITED
    > FIELDS TERMINATED BY ',';
OK
Time taken: 1.358 seconds
hive>
```

Q10.What do you mean by describe and describe extended and describe formatted with respect to database and table

Ans:-

DESCRIBE:-

1.DESCRIBE DATABASE- DESCRIBE DATABASE shows/displays the name of the database, its comment and the root location on the file system.

Describe database [Extended|Formatted] db_name;

2.

Describe Table/View/Column -

DESCRIBE shows/displays the list of columns (including partition columns) for the given table. EXTENDED, FORMATTED keyword is optional. If the EXTENDED is specified, it show/displays all the metadata for the specified table. If the FORMATTED is specified, it show/displays the metadata in a tabular format.

DESCRIBE of View and Column is also same to get respective details specified. For a view, DESCRIBE EXTENDED or FORMATTED can be used to retrieve the view's definition. Two formats are available to describe table/view/column syntax depending the database is specified or not.

Syntax – without DB specified:

```
DESCRIBE [EXTENDED|FORMATTED]
  table_name[.col_name ( [.field_name] | [.'$elem$']
  | [.'$key$'] | [.'$value$'] )* ];
```

Syntax – with DB specified:

```
DESCRIBE [EXTENDED|FORMATTED]
  [db_name.]table_name[ col_name ( [.field_name] |
   [.'$elem$'] | [.'$key$'] | [.'$value$'] )* ];
```

Syntax – Column names:

```
DESCRIBE FORMATTED [db_name.]table_name column_name;
DESCRIBE FORMATTED [db_name.]table_name column_name
 PARTITION (partition_spec);
```

Describe extended - This will show table columns, data types, and other details of the table. Other details will be displayed in single line.

Describe formatted - This will show table columns, data types, and other details of the table. Other details will be displayed into multiple lines.

Q11.How to skip header rows from a table in Hive?

Ans:- skip header rows from a table in Hive means the first line in the files behind the tables will be skipped.

We use tblproperties ("skip.header.line.count"="1"); for skipping header rows from a table.

For Example

```
create external table employee (id int, name string)

lines terminated by '\n'

location '/user/hirw/employees'

tblproperties ("skip.header.line.count"="1");
```

Q12.What is a hive operator? What are the different types of hive operators?

Ans:- **Hive operator** -Apache Hive provides various Built-in operators for data operations to be implemented on the tables present inside Apache Hive warehouse. Hive operators are used for mathematical operations on operands. It returns specific value as per the logic applied.

There are four types of operators in Hive:

- Relational Operators
- Arithmetic Operators
- Logical Operators
- Complex Operators

Relational Operators

These operators are used to compare two operands. The following table describes the relational operators available in Hive:

Operator	Operand	Description
A = B	all primitive types	TRUE if expression A is equivalent to expression B otherwise FALSE.

A != B	all primitive types	TRUE if expression A is not equivalent to expression B otherwise FALSE.
A < B	all primitive types	TRUE if expression A is less than expression B otherwise FALSE.
A <= B	all primitive types	TRUE if expression A is less than or equal to expression B otherwise FALSE.
A > B	all primitive types	TRUE if expression A is greater than expression B otherwise FALSE.
A >= B	all primitive types	TRUE if expression A is greater than or equal to expression B otherwise FALSE.
A IS NULL	all types	TRUE if expression A evaluates to NULL otherwise FALSE.
A IS NOT NULL	all types	FALSE if expression A evaluates to NULL otherwise TRUE.
A LIKE B	Strings	TRUE if string pattern A matches to B otherwise FALSE.
A RLIKE B	Strings	NULL if A or B is NULL, TRUE if any substring of A matches the Java regular expression B , otherwise FALSE.
A REGEXP B	Strings	Same as RLIKE.

Example

Let us assume the **employee** table is composed of fields named Id, Name, Salary, Designation, and Dept as shown below. Generate a query to retrieve the employee details whose Id is 1205.

```
+-----+--------------+--------+---------------------------+------+
| Id  | Name         | Salary | Designation               | Dept |
+-----+--------------+-----------------------------------+------+
```

```
|1201 | Gopal       | 45000  | Technical manager        | TP   |
|1202 | Manisha     | 45000  | Proofreader              | PR   |
|1203 | Masthanvali | 40000  | Technical writer         | TP   |
|1204 | Krian       | 40000  | Hr Admin                 | HR   |
|1205 | Kranthi     | 30000  | Op Admin                 | Admin|
+-----+-------------+--------+--------------------------+------+
```

The following query is executed to retrieve the employee details using the above table:

```
hive> SELECT * FROM employee WHERE Id=1205;
```

On successful execution of query, you get to see the following response:

```
+-----+-----------+-----------+---------------------------------+
| ID  | Name      | Salary    | Designation             | Dept  |
+-----+-----------+-----------+---------------------------------+
|1205 | Kranthi   | 30000     | Op Admin                | Admin |
+-----+-----------+-----------+---------------------------------+
```

The following query is executed to retrieve the employee details whose salary is more than or equal to Rs 40000.

```
hive> SELECT * FROM employee WHERE Salary>=40000;
```

On successful execution of query, you get to see the following response:

```
+-----+------------+--------+--------------------------+------+
| ID  | Name       | Salary | Designation              | Dept |
+-----+------------+--------+--------------------------+------+
|1201 | Gopal      | 45000  | Technical manager        | TP   |
|1202 | Manisha    | 45000  | Proofreader              | PR   |
|1203 | Masthanvali| 40000  | Technical writer         | TP   |
|1204 | Krian      | 40000  | Hr Admin                 | HR   |
+-----+------------+--------+--------------------------+------+
```

Arithmetic Operators

These operators support various common arithmetic operations on the operands. All of them return number types. The following table describes the arithmetic operators available in Hive:

Operators	Operand	Description
A + B	all number types	Gives the result of adding A and B.
A - B	all number types	Gives the result of subtracting B from A.

A * B	all number types	Gives the result of multiplying A and B.
A / B	all number types	Gives the result of dividing B from A.
A % B	all number types	Gives the reminder resulting from dividing A by B.
A & B	all number types	Gives the result of bitwise AND of A and B.
A | B	all number types	Gives the result of bitwise OR of A and B.
A ^ B	all number types	Gives the result of bitwise XOR of A and B.
~A	all number types	Gives the result of bitwise NOT of A.

Example

The following query adds two numbers, 20 and 30.

```
hive> SELECT 20+30 ADD FROM temp;
```

On successful execution of the query, you get to see the following response:

```
+--------+
|  ADD   |
+--------+
|  50    |
+--------+
```

Logical Operators

The operators are logical expressions. All of them return either TRUE or FALSE.

Operators	Operands	Description
A AND B	Boolean	TRUE if both A and B are TRUE, otherwise FALSE.
A && B	Boolean	Same as A AND B.
A OR B	Boolean	TRUE if either A or B or both are TRUE, otherwise FALSE.

A || B	Boolean	Same as A OR B.
NOT A	Boolean	TRUE if A is FALSE, otherwise FALSE.
!A	Boolean	Same as NOT A.

Example

The following query is used to retrieve employee details whose Department is TP and Salary is more than Rs 40000.

```
hive> SELECT * FROM employee WHERE Salary>40000 && Dept=TP;
```

On successful execution of the query, you get to see the following response:

```
+------+--------------+-------------+-------------------+--------+
| ID   | Name         | Salary      | Designation       | Dept   |
+------+--------------+-------------+-------------------+--------+
|1201  | Gopal        | 45000       | Technical manager | TP     |
+------+--------------+-------------+-------------------+--------+
```

Complex Operators

These operators provide an expression to access the elements of Complex Types.

Operator	Operand	Description
A[n]	A is an Array and n is an int	It returns the nth element in the array A. The first element has index 0.
M[key]	M is a Map<K, V> and key has type K	It returns the value corresponding to the key in the map.
S.x	S is a struct	It returns the x field of S.

Q13.Explain about the Hive Built-In Functions

Ans:- The functions look quite similar to SQL functions, except for their usage.

Built-In Functions

Hive supports the following built-in functions:

Return Type	Signature	Description
BIGINT	round(double a)	It returns the rounded BIGINT value of the double.
BIGINT	floor(double a)	It returns the maximum BIGINT value that is equal or less than the double.
BIGINT	ceil(double a)	It returns the minimum BIGINT value that is equal or greater than the double.
double	rand(), rand(int seed)	It returns a random number that changes from row to row.
string	concat(string A, string B,...)	It returns the string resulting from concatenating B after A.
string	substr(string A, int start)	It returns the substring of A starting from start position till the end of string A.
string	substr(string A, int start, int length)	It returns the substring of A starting from start position with the given length.
string	upper(string A)	It returns the string resulting from converting all characters of A to upper case.
string	ucase(string A)	Same as above.
string	lower(string A)	It returns the string resulting from converting all characters of B to lower case.
string	lcase(string A)	Same as above.

string	trim(string A)	It returns the string resulting from trimming spaces from both ends of A.
string	ltrim(string A)	It returns the string resulting from trimming spaces from the beginning (left hand side) of A.
string	rtrim(string A)	rtrim(string A) It returns the string resulting from trimming spaces from the end (right hand side) of A.
string	regexp_replace(string A, string B, string C)	It returns the string resulting from replacing all substrings in B that match the Java regular expression syntax with C.
int	size(Map<K.V>)	It returns the number of elements in the map type.
int	size(Array<T>)	It returns the number of elements in the array type.
value of <type>	cast(<expr> as <type>)	It converts the results of the expression expr to <type> e.g. cast('1' as BIGINT) converts the string '1' to it integral representation. A NULL is returned if the conversion does not succeed.
string	from_unixtime(int unixtime)	convert the number of seconds from Unix epoch (1970-01-01 00:00:00 UTC) to a string representing the timestamp of that moment in the current system time zone in the format of "1970-01-01 00:00:00"
string	to_date(string timestamp)	It returns the date part of a timestamp string: to_date("1970-01-01 00:00:00") = "1970-01-01"

int	year(string date)	It returns the year part of a date or a timestamp string: year("1970-01-01 00:00:00") = 1970, year("1970-01-01") = 1970
int	month(string date)	It returns the month part of a date or a timestamp string: month("1970-11-01 00:00:00") = 11, month("1970-11-01") = 11
int	day(string date)	It returns the day part of a date or a timestamp string: day("1970-11-01 00:00:00") = 1, day("1970-11-01") = 1
string	get_json_object(string json_string, string path)	It extracts json object from a json string based on json path specified, and returns json string of the extracted json object. It returns NULL if the input json string is invalid.

Example

The following queries demonstrate some built-in functions:

round() function

```
hive> SELECT round(2.6) from temp;
```

On successful execution of query, you get to see the following response:

```
3.0
```

floor() function

```
hive> SELECT floor(2.6) from temp;
```

On successful execution of the query, you get to see the following response:

```
2.0
```

ceil() function

```
hive> SELECT ceil(2.6) from temp;
```

On successful execution of the query, you get to see the following response:

```
3.0
```

Aggregate Functions

Hive supports the following built-in **aggregate functions**. The usage of these functions is as same as the SQL aggregate functions.

Return Type	Signature	Description
BIGINT	count(*), count(expr),	count(*) - Returns the total number of retrieved rows.
DOUBLE	sum(col), sum(DISTINCT col)	It returns the sum of the elements in the group or the sum of the distinct values of the column in the group.
DOUBLE	avg(col), avg(DISTINCT col)	It returns the average of the elements in the group or the average of the distinct values of the column in the group.
DOUBLE	min(col)	It returns the minimum value of the column in the group.
DOUBLE	max(col)	It returns the maximum value of the column in the group.

Q14. Write hive DDL and DML commands.

Ans:-

Introduction to Hive DDL commands

Hive DDL commands are the statements used for defining and changing the structure of a table or database in Hive. It is used to build or modify the tables and other objects in the database.

The several types of Hive DDL commands are:

1. CREATE
2. SHOW

3. DESCRIBE
4. USE
5. DROP
6. ALTER
7. TRUNCATE

Table-1 Hive DDL commands

DDL Command	Use With
CREATE	Database, Table
SHOW	Databases, Tables, Table Properties, Partitions, Functions, Index
DESCRIBE	Database, Table, view
USE	Database
DROP	Database, Table
ALTER	Database, Table
TRUNCATE	Table

Before moving forward, note that the Hive commands are **case-insensitive**. CREATE DATABASE is the same as create database.

So now, let us go through each of the commands deeply. Let's start with the DDL commands on Databases in Hive.

DDL Commands On Databases in Hive

1. CREATE DATABASE in Hive

The **CREATE DATABASE** statement is used to create a database in the Hive. The DATABASE and SCHEMA are interchangeable. We can use either DATABASE or SCHEMA.

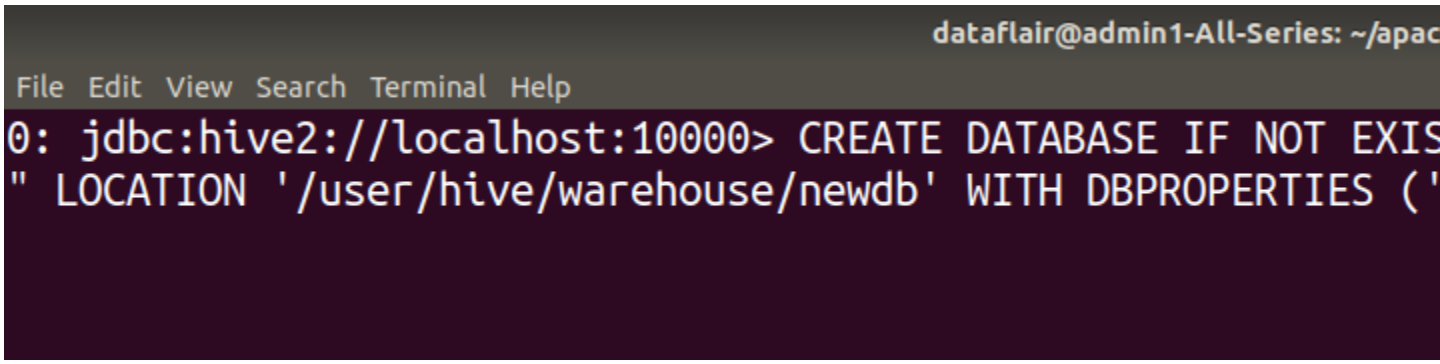
Syntax:

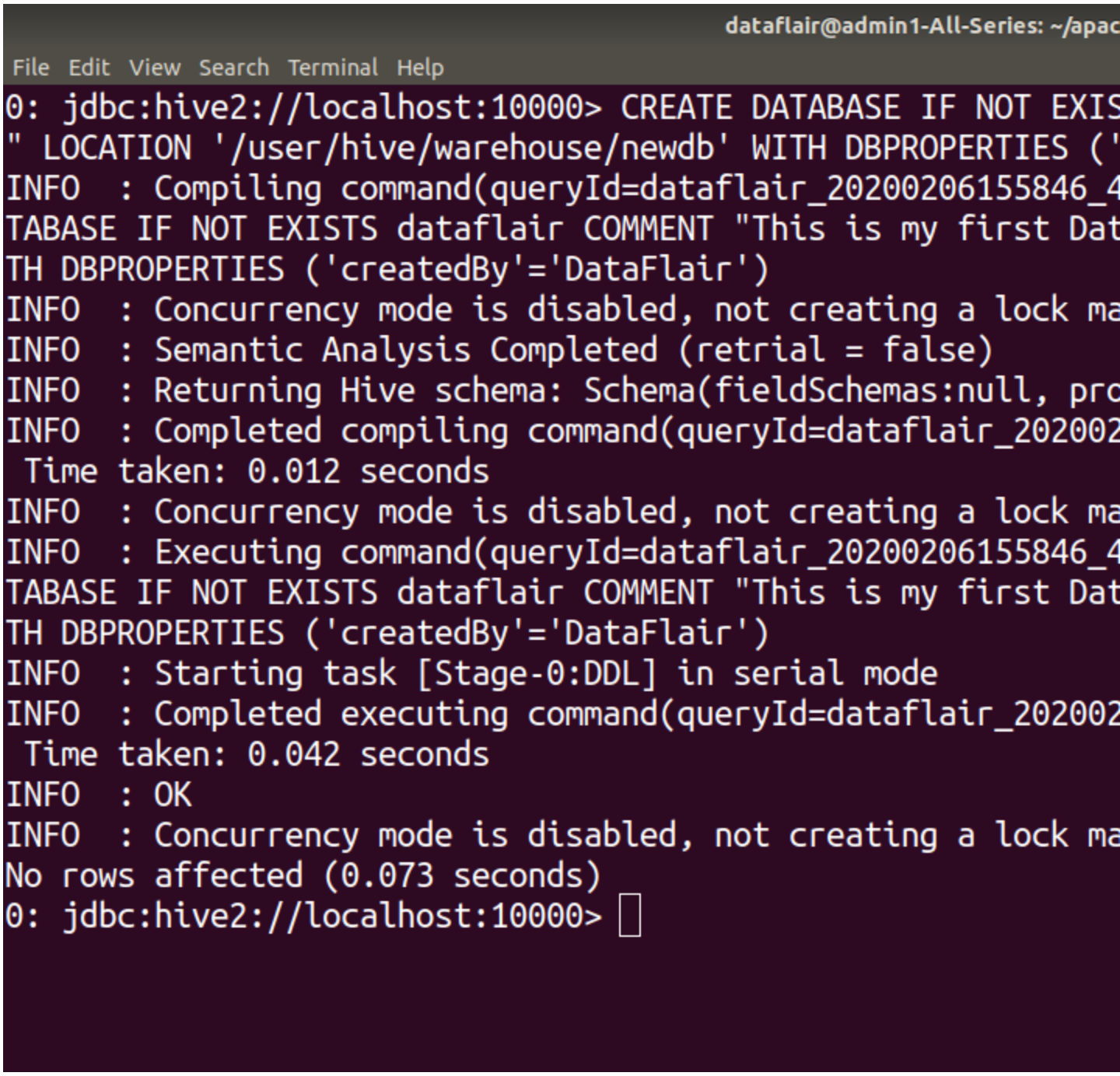
```
CREATE (DATABASE|SCHEMA) [IF NOT EXISTS] database_name

[COMMENT database_comment]

[LOCATION hdfs_path]
```

[WITH DBPROPERTIES (property_name=property_value, ...)];

DDL CREATE DATABASE Example:

Here in this example, we are creating a database ‘dataflair’.



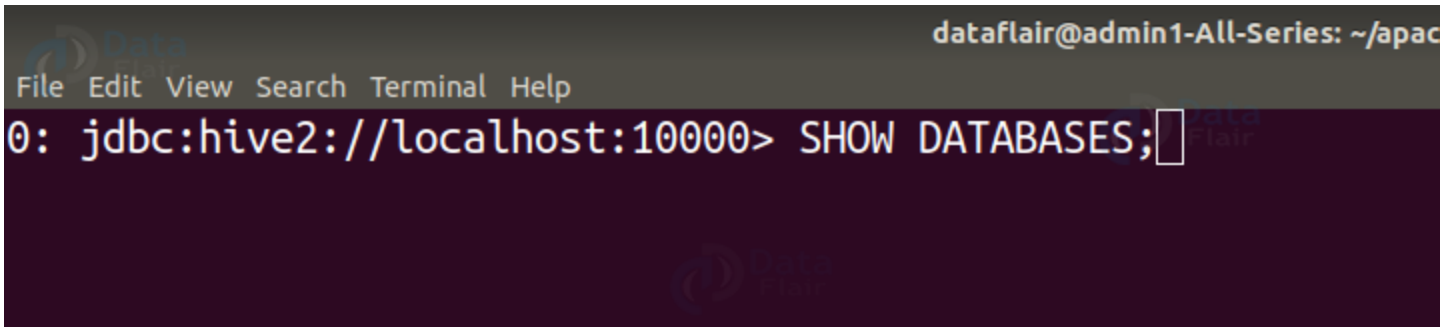


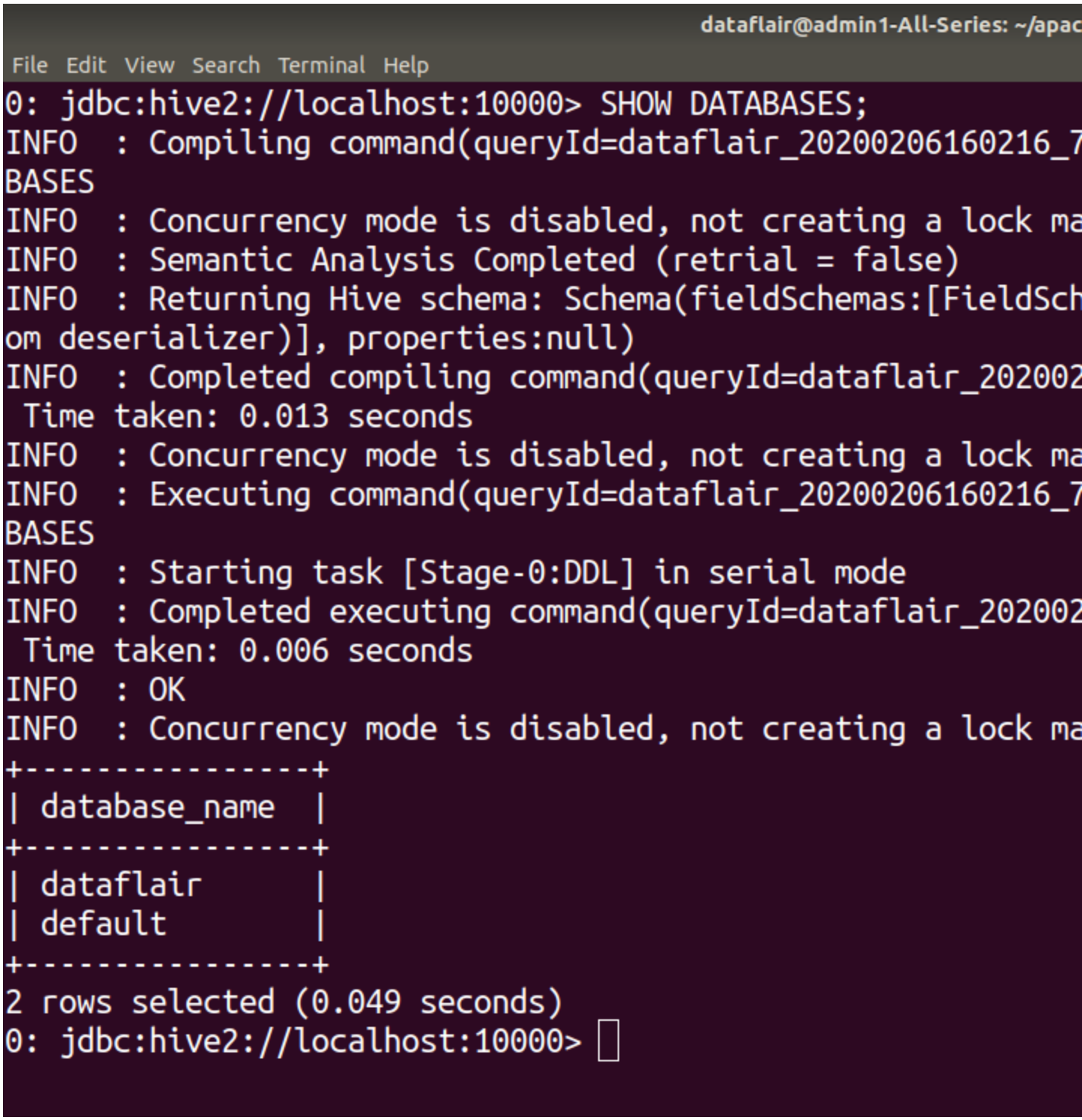
2. SHOW DATABASE in Hive

The **SHOW DATABASES** statement lists all the databases present in the Hive.
Syntax:

```
SHOW (DATABASES|SCHEMAS);
```

DDL SHOW DATABASES Example:


dataflair@admin1-All-Series: ~/apac
File Edit View Search Terminal Help
0: jdbc:hive2://localhost:10000> SHOW DATABASES;



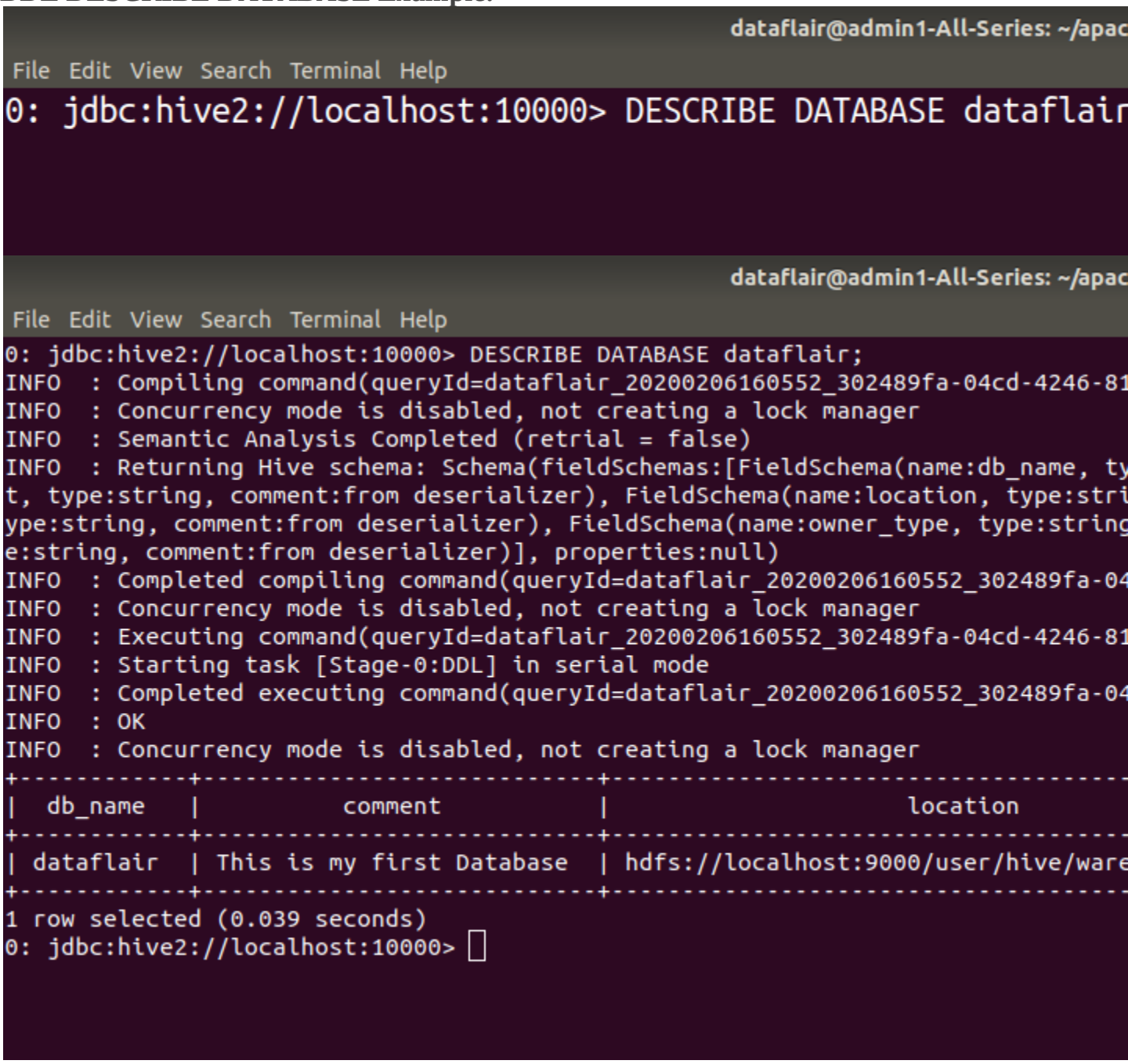
3. DESCRIBE DATABASE in Hive

The **DESCRIBE DATABASE** statement in Hive shows the name of Database in Hive, its comment (if set), and its location on the file system.
The **EXTENDED** can be used to get the database properties.

Syntax:

DESCRIBE DATABASE/SCHEMA [EXTENDED] db_name;

DDL DESCRIBE DATABASE Example:



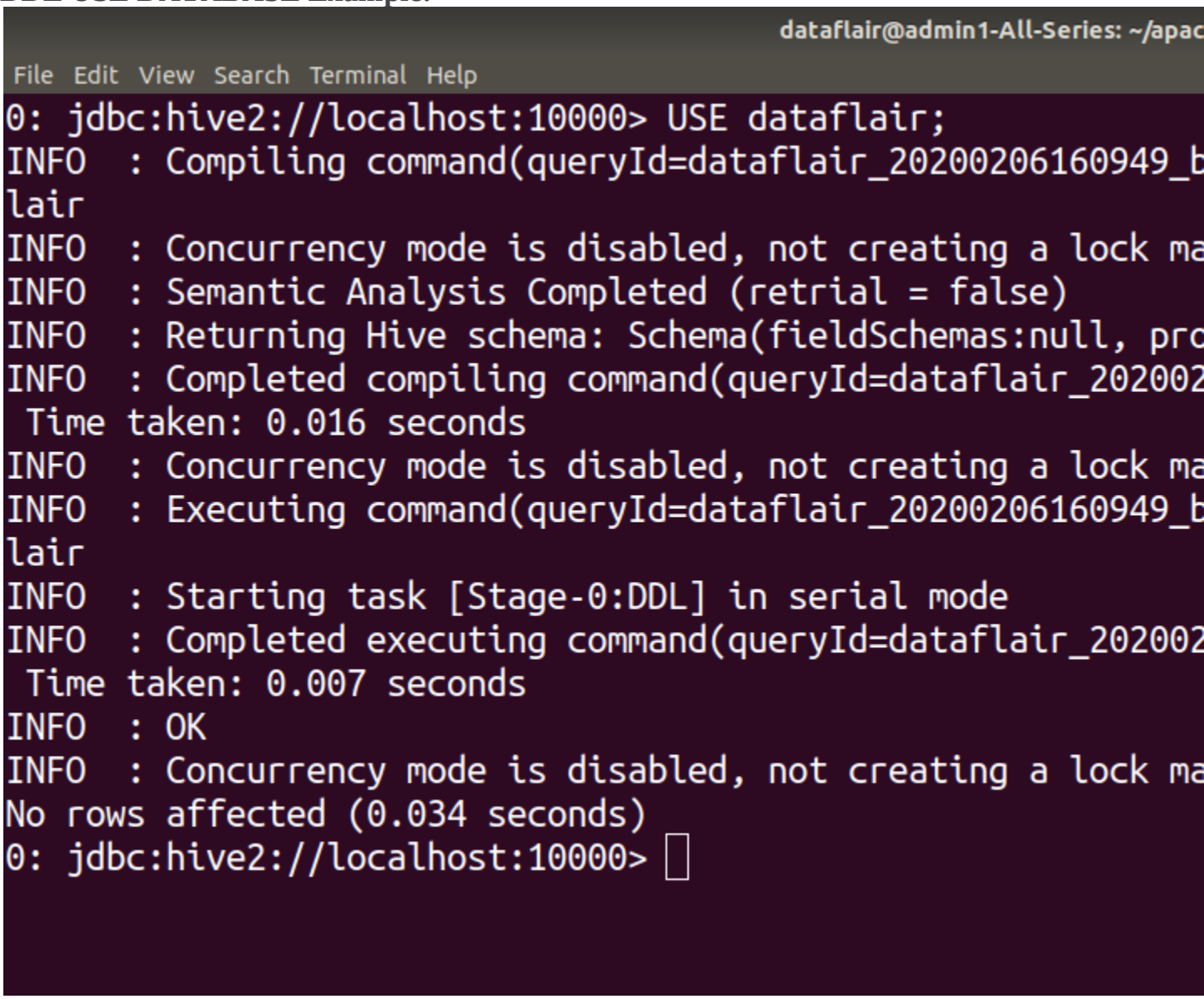
4. USE DATABASE in Hive

The **USE** statement in Hive is used to select the specific database for a session on which all subsequent HiveQL statements would be executed.

Syntax:

USE database_name;

DDL USE DATABASE Example:



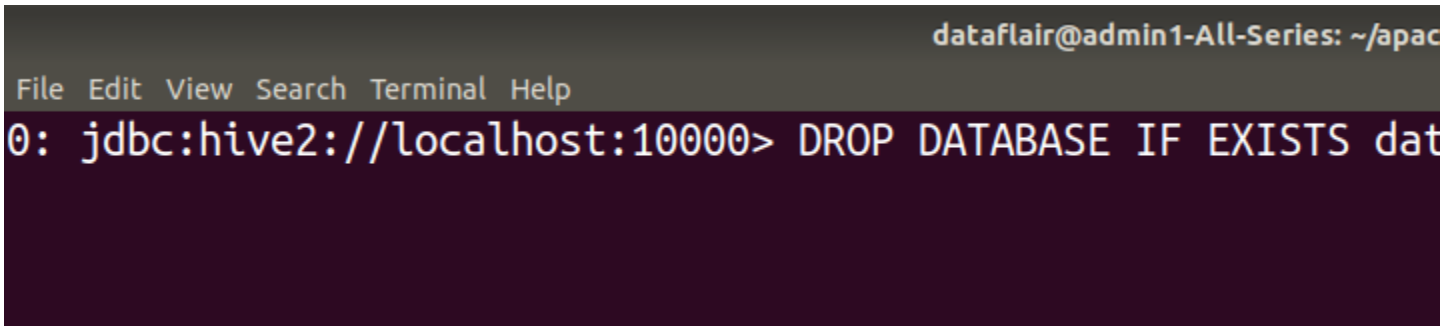
5. DROP DATABASE in Hive

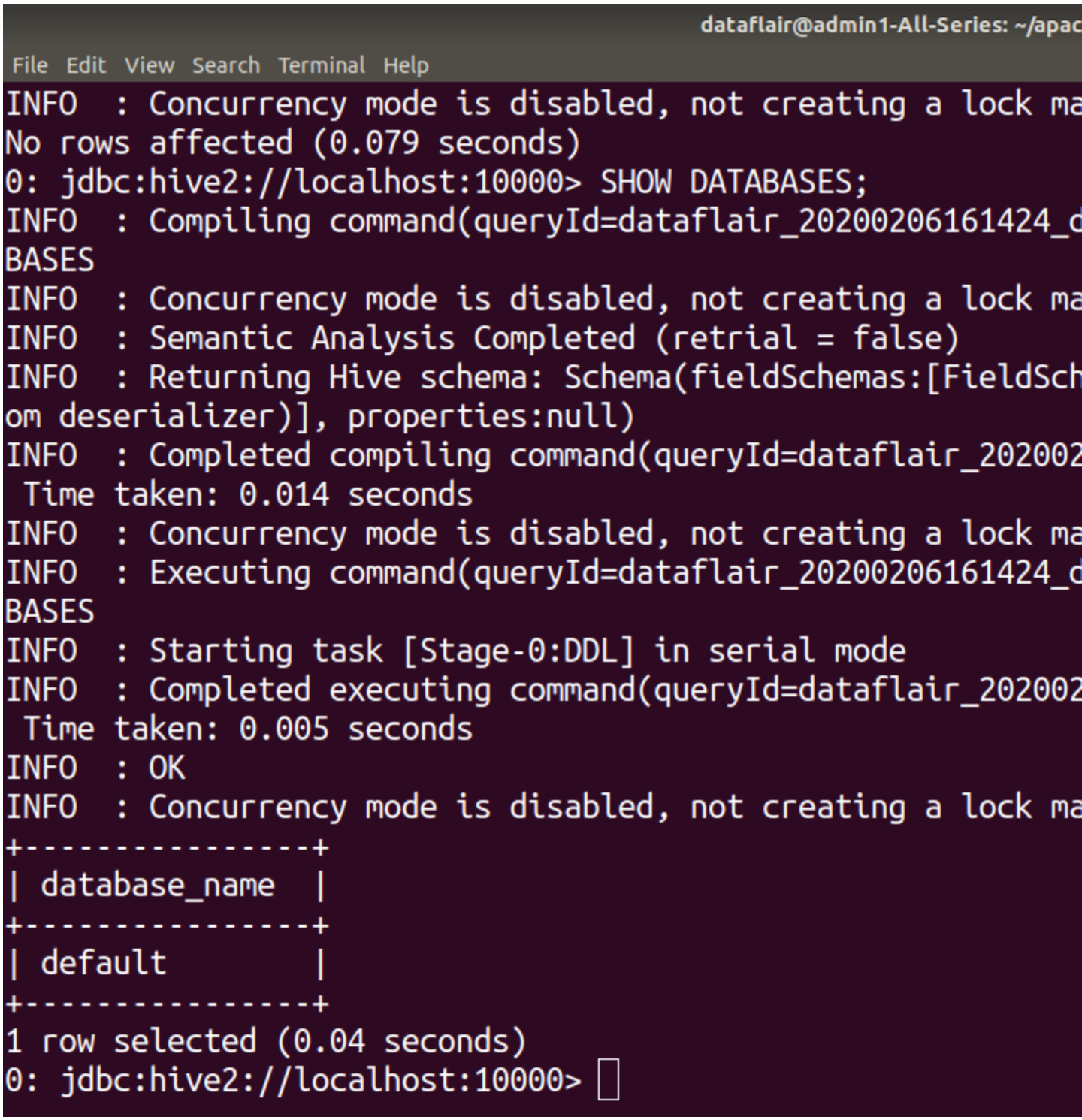
The **DROP DATABASE** statement in Hive is used to Drop (delete) the database.
The default behavior is RESTRICT which means that the database is dropped only when it is empty. To drop the database with tables, we can use CASCADE.

Syntax:

DROP (DATABASE|SCHEMA) [**IF** EXISTS] database_name [RESTRICT|CASCADE];

DDL DROP DATABASE Example:
Here in this example, we are dropping a database ‘dataflair’ using the DROP statement.





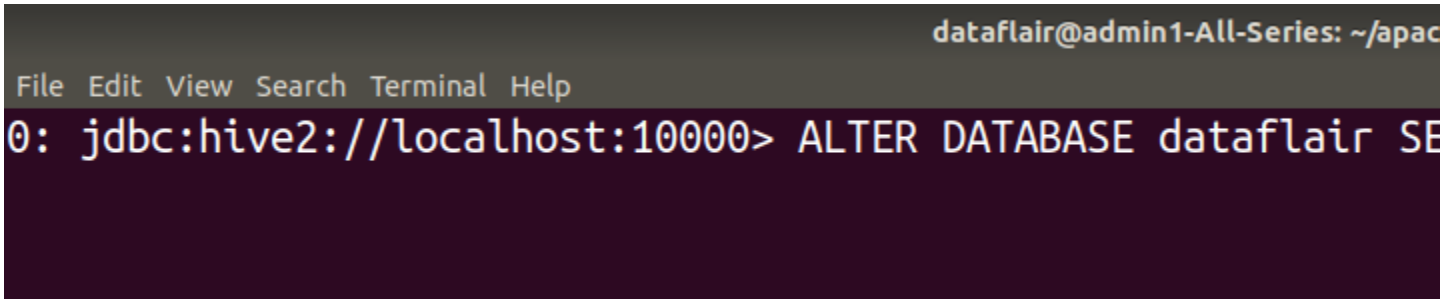
6. *ALTER DATABASE in Hive*

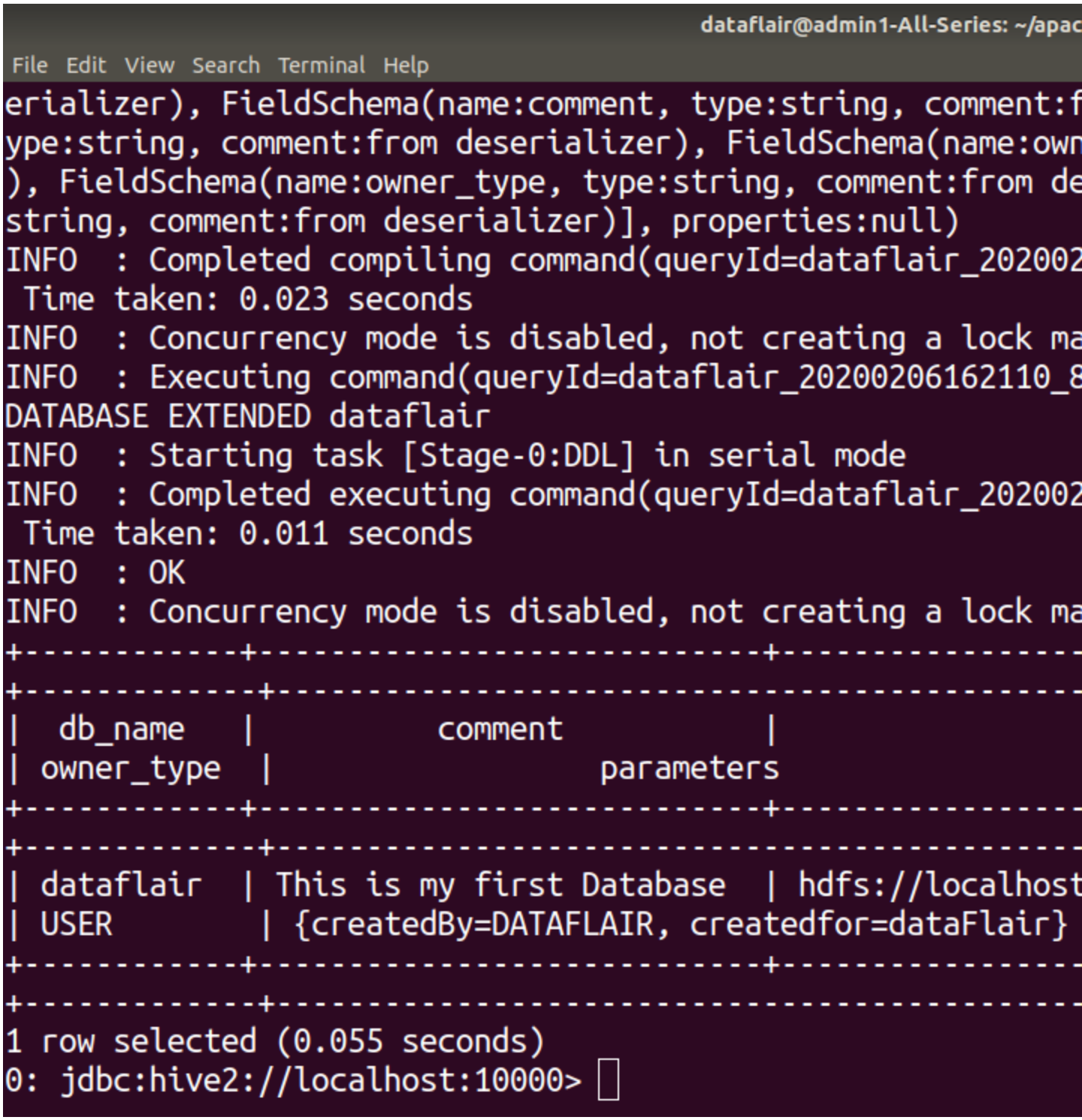
The **ALTER DATABASE** statement in Hive is used to change the metadata associated with the database in Hive.

Syntax for changing Database Properties:

ALTER (DATABASE|SCHEMA) database_name SET DBPROPERTIES (property_name=property_value, ...);

DDL ALTER DATABASE properties Example:
In this example, we are setting the database properties of the 'dataflair' database after its creation by using the ALTER command.



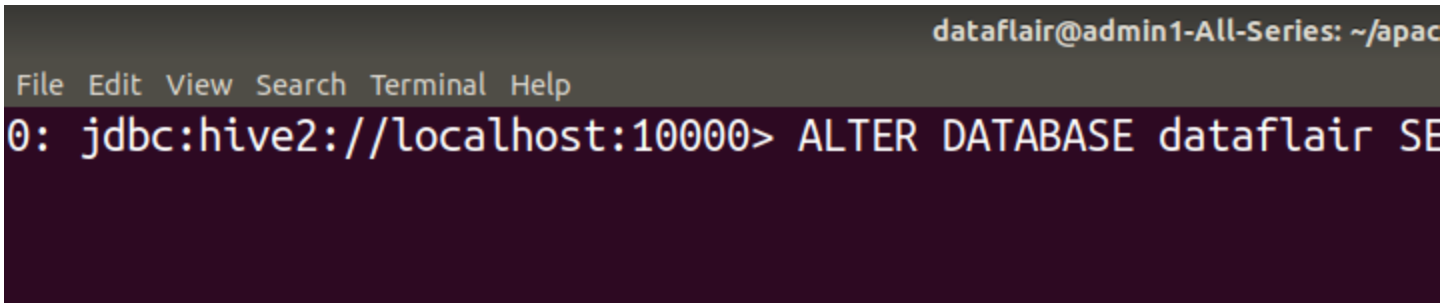


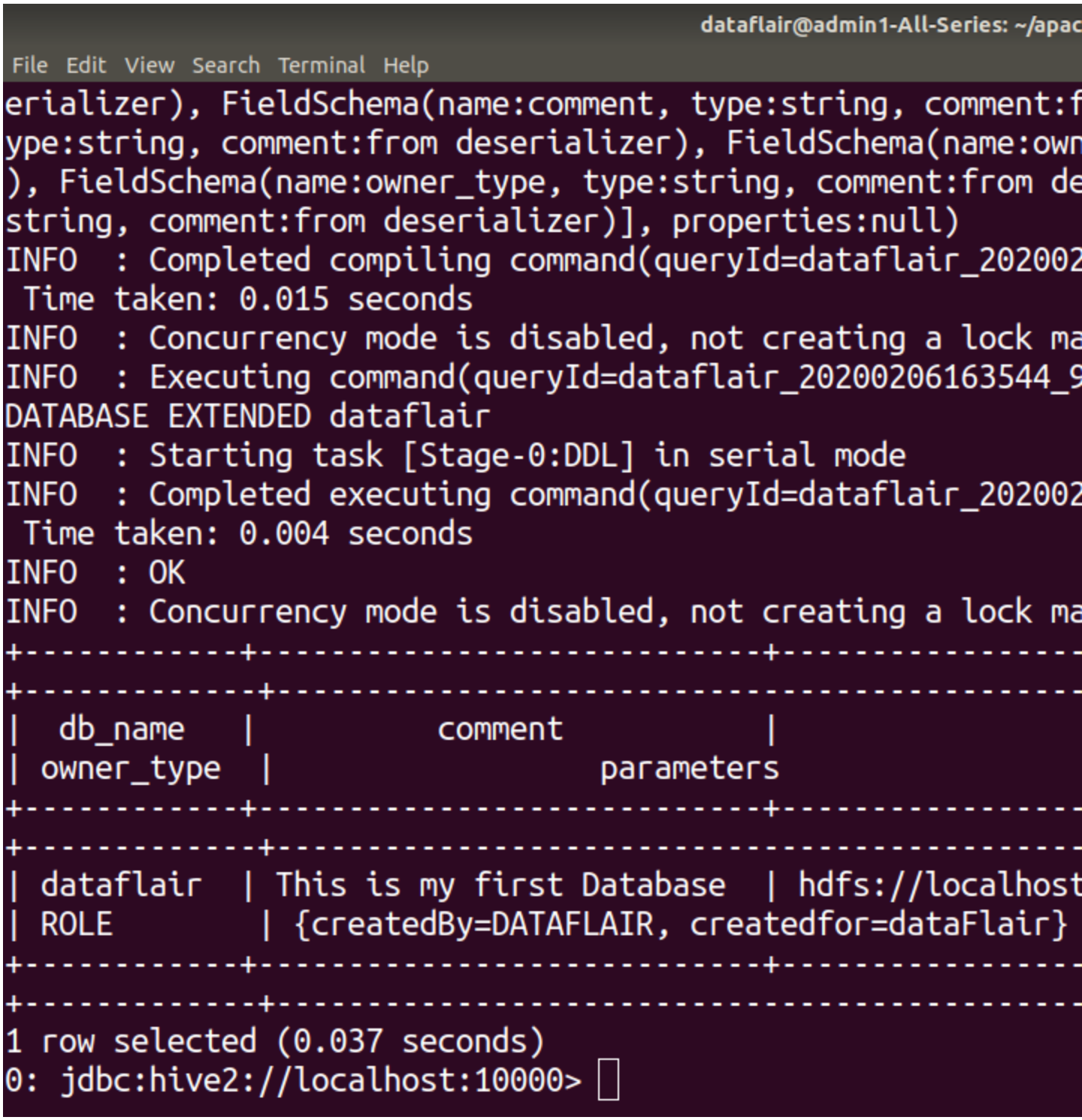
Syntax for changing Database owner:

ALTER (DATABASE|SCHEMA) database_name SET OWNER [USER|ROLE] user_or_role;

DDL ALTER DATABASE owner Example:

In this example, we are changing the owner role of the 'dataflair' database using the ALTER statement.


dataflair@admin1-All-Series: ~/apac
File Edit View Search Terminal Help
0: jdbc:hive2://localhost:10000> ALTER DATABASE dataflair SE



Syntax for changing Database Location:

ALTER (DATABASE|SCHEMA) database_name SET LOCATION hdfs_path;

Note: The ALTER DATABASE ... SET LOCATION statement does not move the database current directory contents to the newly specified location.

This statement does not change the locations associated with any tables or partitions under the specified database. Instead, it changes the default parent-directory, where new tables will be added for this database.

No other metadata associated with the database can be changed.

DDL Commands on Tables in Hive

1. CREATE TABLE

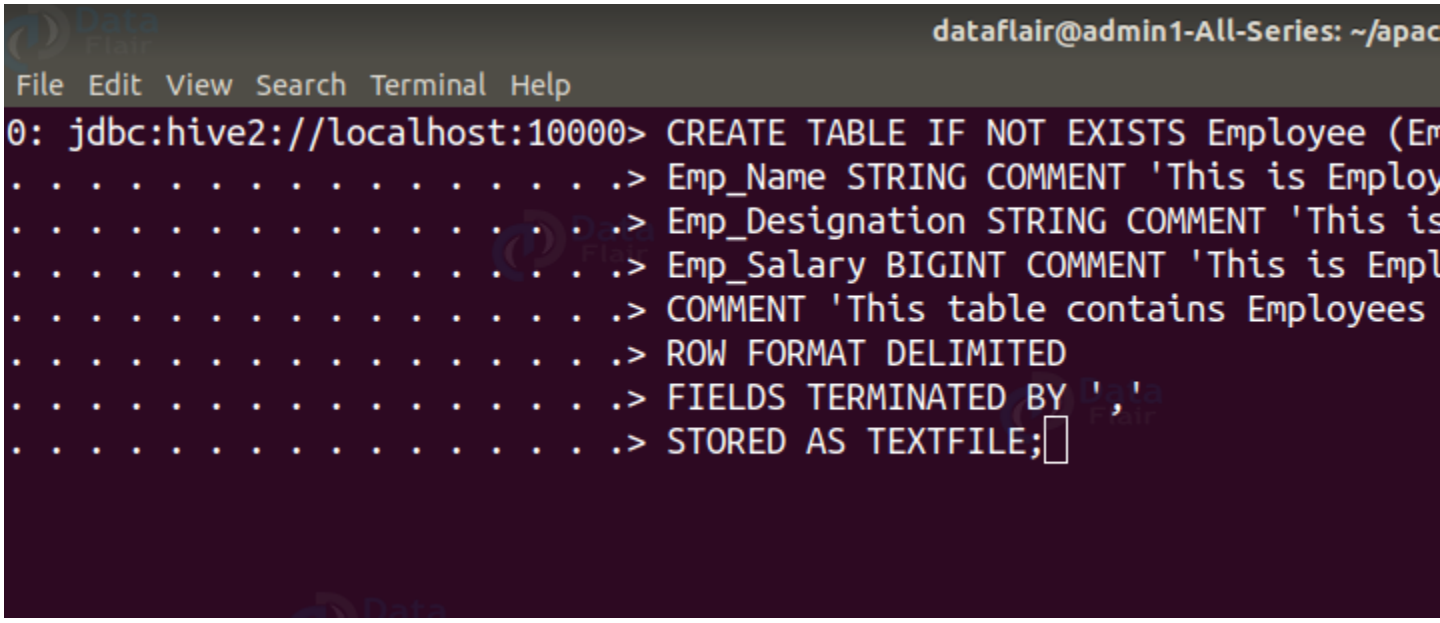
The **CREATE TABLE** statement in Hive is used to create a table with the given name. If a table or view already exists with the same name, then the error is thrown. We can use **IF NOT EXISTS** to skip the error.

Syntax:

CREATE TABLE [**IF** NOT EXISTS] [db_name.] table_name [(col_name data_type [COMMENT col_comment], ... [COMMENT col_comment])] [COMMENT table_comment] [ROW FORMAT row_format] [STORED AS file_format] [LOCATION hdfs_path];

DDL CREATE TABLE Example:

In this table, we are creating a table 'Employee' in the 'dataflair' database.



ROW FORMAT DELIMITED means we are telling the Hive that when it finds a new line character, that means a new record.

FIELDS TERMINATED BY ',' tells Hive what delimiter we are using in our files to separate each column.

STORED AS TEXTFILE is to tell Hive what type of file to expect.

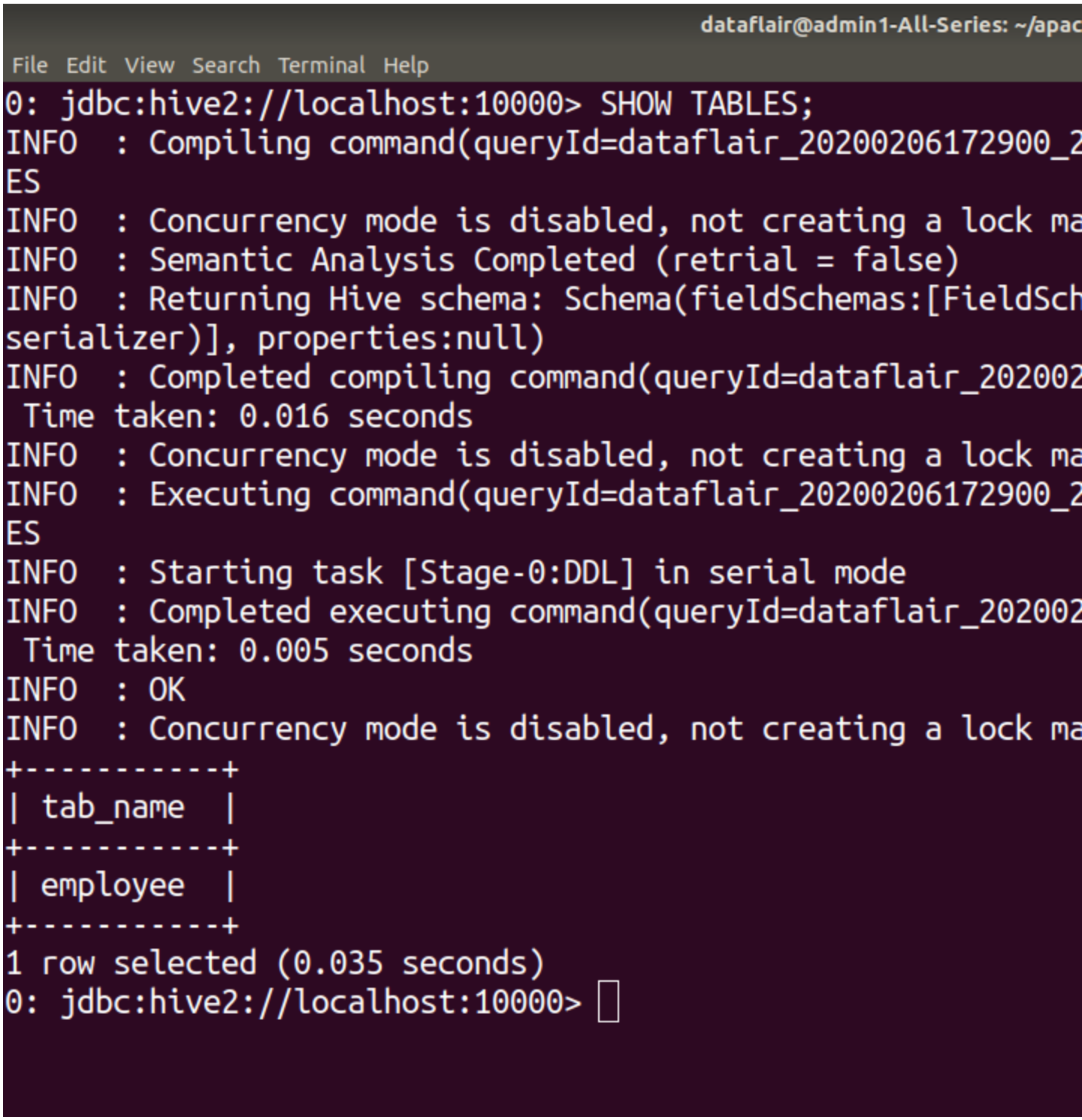
2. SHOW TABLES in Hive

The **SHOW TABLES** statement in Hive lists all the base tables and **views** in the current database.

Syntax:

```
SHOW TABLES [IN database_name];
```

DDL SHOW TABLES Example:



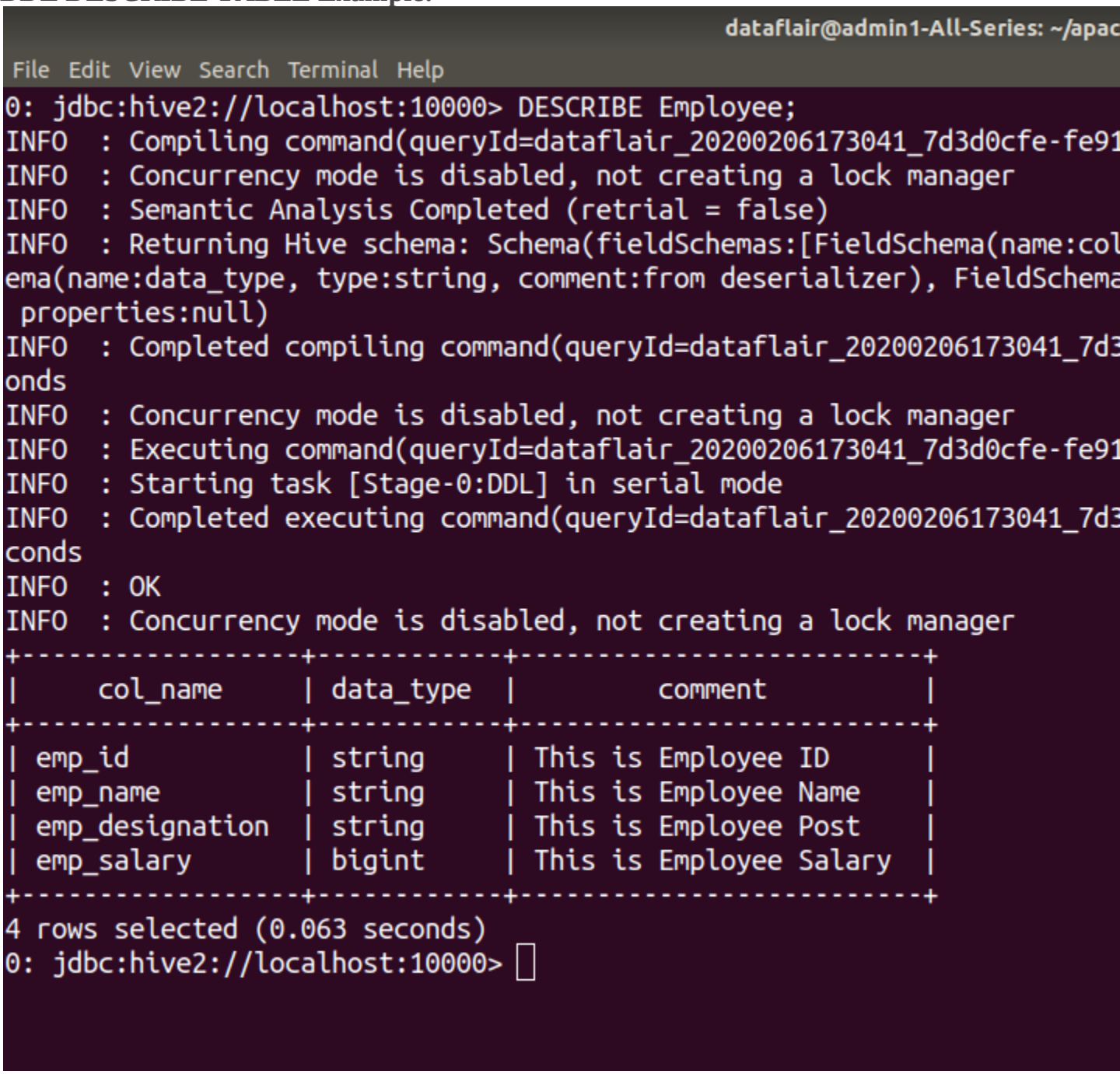
3. DESCRIBE TABLE in Hive

The **DESCRIBE** statement in Hive shows the lists of columns for the specified table.

Syntax:

DESCRIBE [EXTENDED|FORMATTED] [db_name.] table_name[.col_name ([.field_name])];

DDL DESCRIBE TABLE Example:



4. DROP TABLE in Hive

The **DROP TABLE** statement in Hive deletes the data for a particular table and remove all metadata associated with it from Hive metastore.

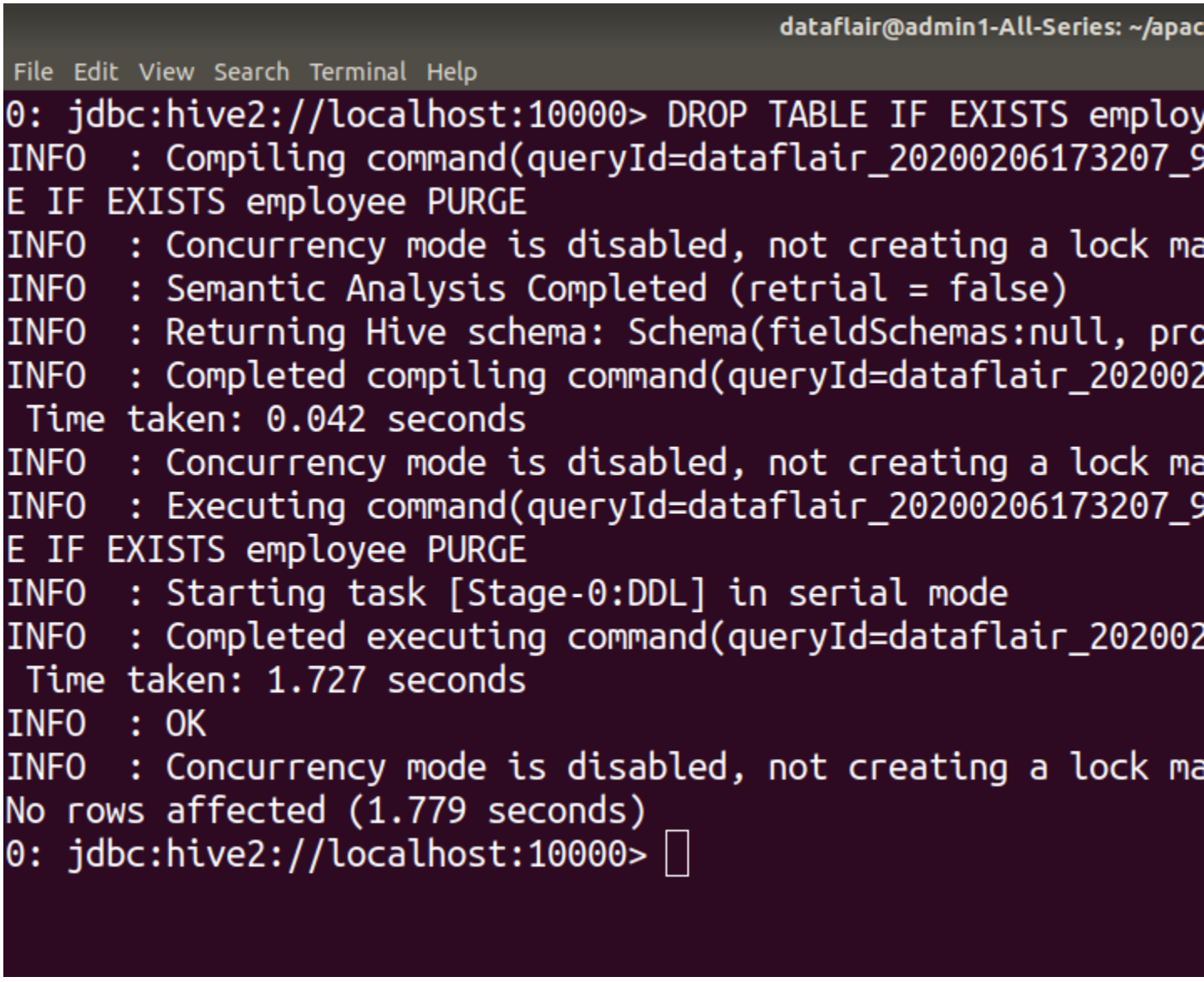
If **PURGE** is not specified then the data is actually moved to the .Trash/current directory. If **PURGE** is specified, then data is lost completely.

Syntax:

```
DROP TABLE [IF EXISTS] table_name [PURGE];
```

DDL DROP TABLE Example:

In the below example, we are deleting the 'employee' table.



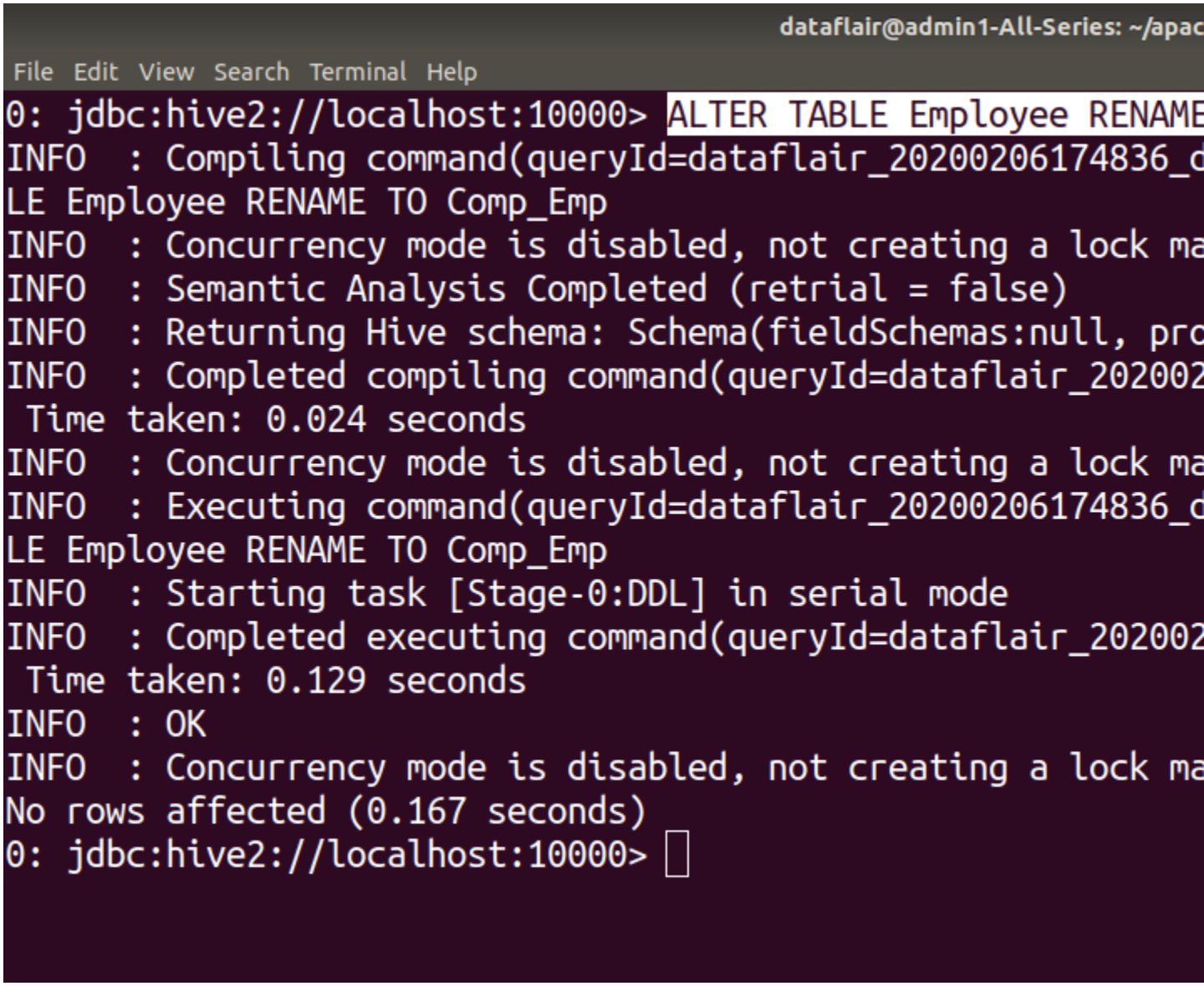
5. ALTER TABLE in Hive

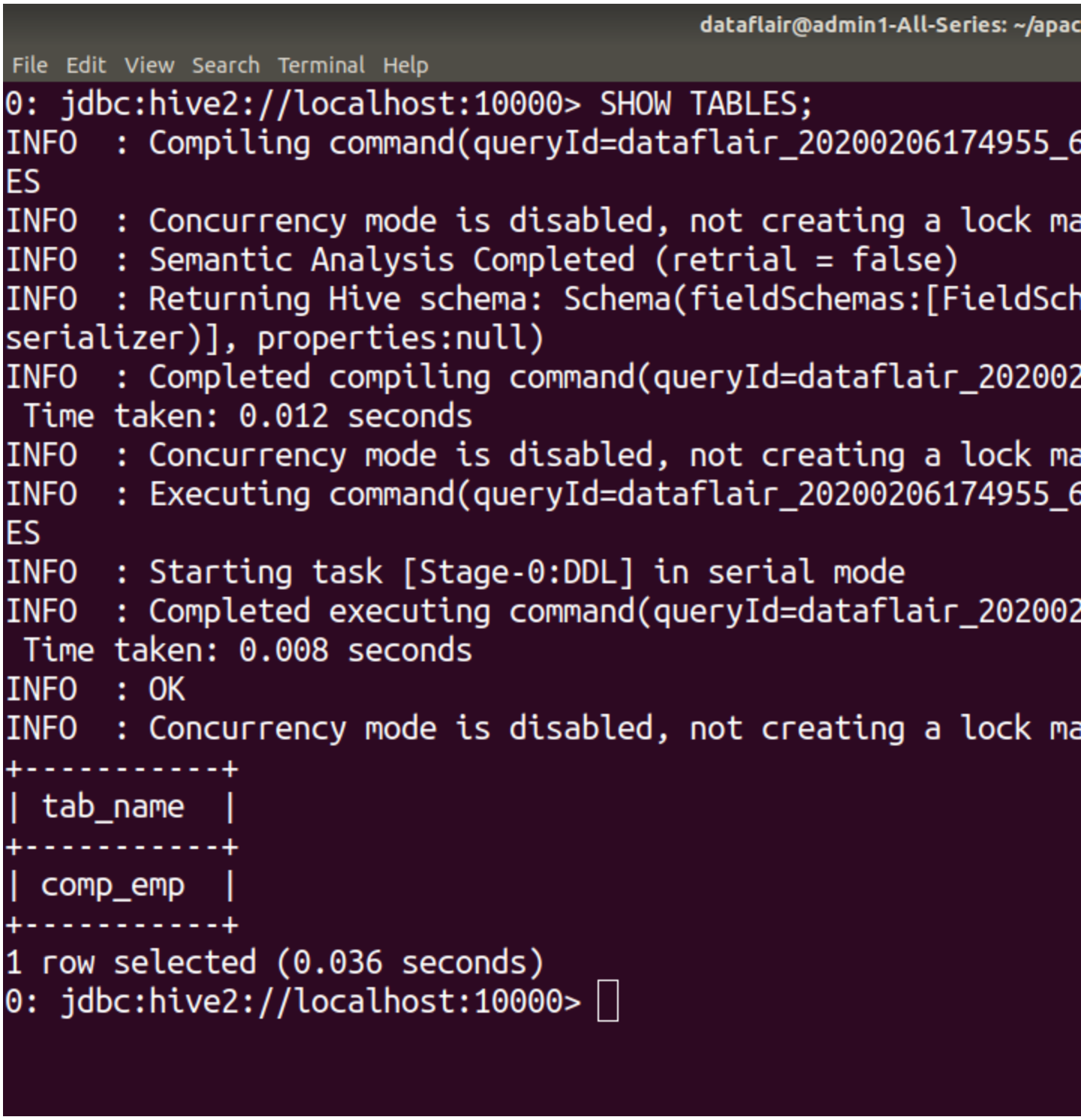
The **ALTER TABLE** statement in Hive enables you to change the structure of an existing table. Using the ALTER TABLE statement we can rename the table, add columns to the table, change the table properties, etc.

Syntax to Rename a table:

ALTER TABLE table_name RENAME TO new_table_name;

DDL ALTER TABLE name Example:
In this example, we are trying to rename the 'Employee' table to 'Com_Emp' using the ALTER statement.



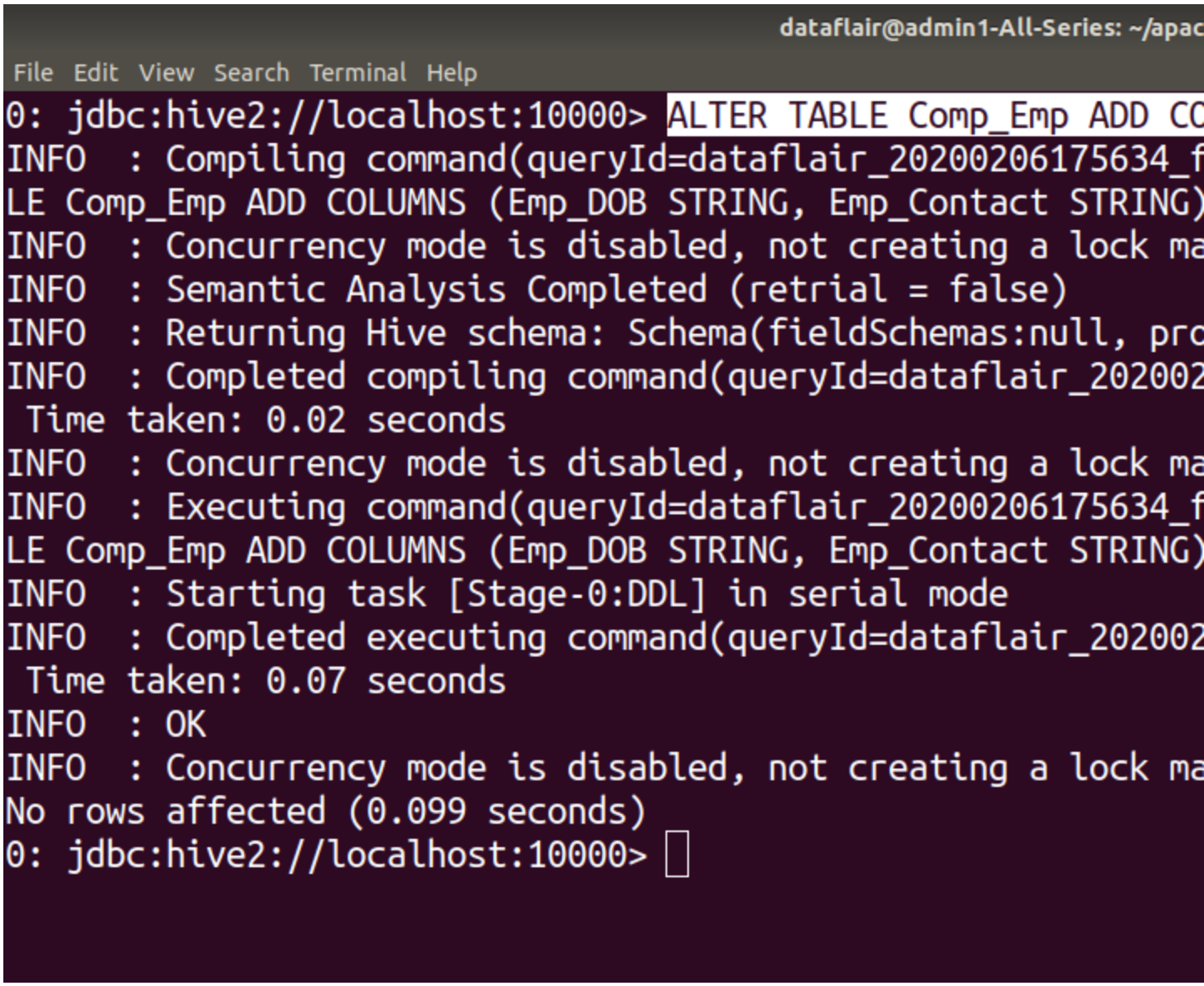


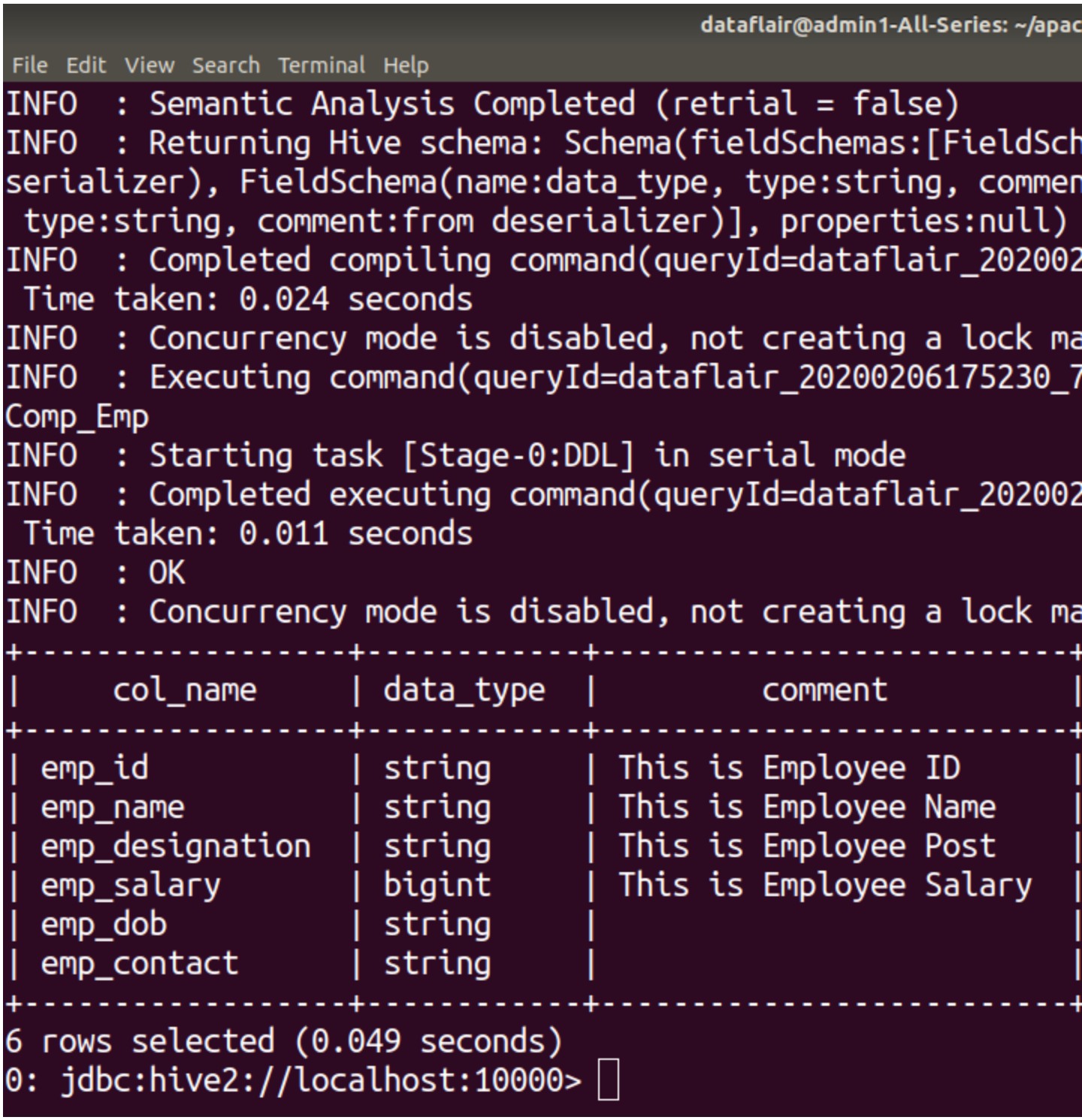
Syntax to Add columns to a table:

ALTER TABLE table_name ADD COLUMNS (column1, column2) ;

DDL ALTER TABLE columns Example:

In this example, we are adding two columns 'Emp_DOB' and 'Emp_Contact' in the 'Comp_Emp' table using the ALTER command.


dataflair@admin1-All-Series: ~/apac
File Edit View Search Terminal Help
0: jdbc:hive2://localhost:10000> ALTER TABLE Comp_Emp ADD CO
INFO : Compiling command(queryId=dataflair_20200206175634_f
LE Comp_Emp ADD COLUMNS (Emp_DOB STRING, Emp_Contact STRING)
INFO : Concurrency mode is disabled, not creating a lock ma
INFO : Semantic Analysis Completed (retrial = false)
INFO : Returning Hive schema: Schema(fieldSchemas:null, pro
INFO : Completed compiling command(queryId=dataflair_20200
Time taken: 0.02 seconds
INFO : Concurrency mode is disabled, not creating a lock ma
INFO : Executing command(queryId=dataflair_20200206175634_f
LE Comp_Emp ADD COLUMNS (Emp_DOB STRING, Emp_Contact STRING)
INFO : Starting task [Stage-0:DDL] in serial mode
INFO : Completed executing command(queryId=dataflair_20200
Time taken: 0.07 seconds
INFO : OK
INFO : Concurrency mode is disabled, not creating a lock ma
No rows affected (0.099 seconds)
0: jdbc:hive2://localhost:10000>

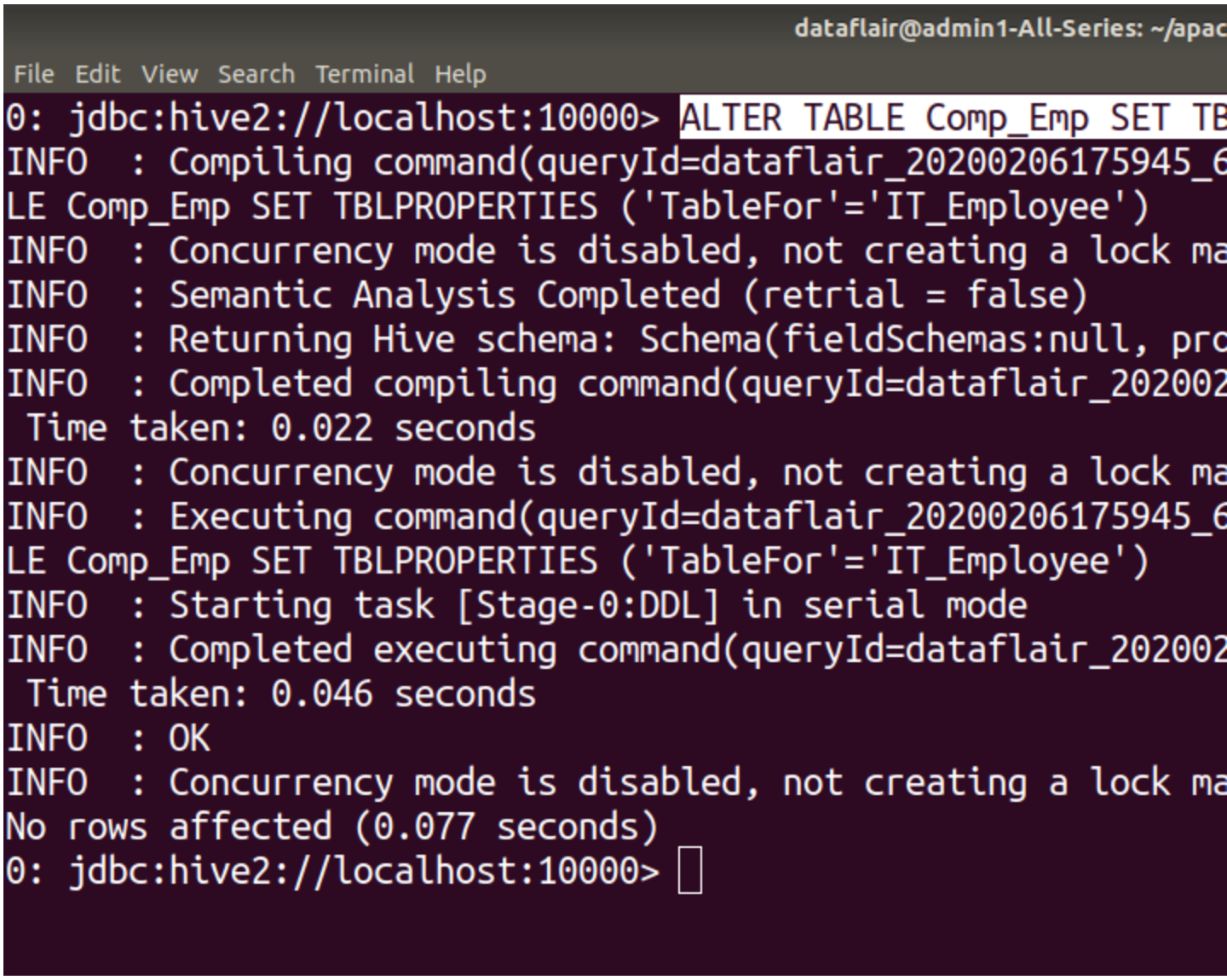


Syntax to set table properties:

ALTER TABLE table_name SET TBLPROPERTIES (‘property_key’=’property_new_value’);

DDL ALTER TABLE properties Example:

In this example, we are setting the table properties after table creation by using ALTER command.



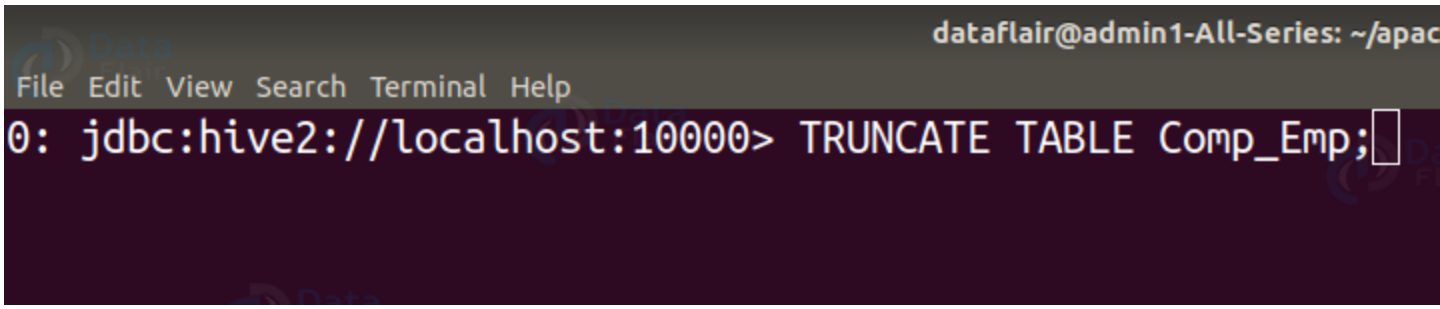
6. TRUNCATE TABLE

TRUNCATE TABLE statement in Hive removes all the rows from the table or partition.

Syntax:

TRUNCATE TABLE table_name;

DDL TRUNCATE TABLE Example:



Introduction to Hive DML commands

Hive DML (Data Manipulation Language) commands are used to insert, update, retrieve, and delete data from the Hive table once the table and database schema has been defined using Hive DDL commands.

The various Hive DML commands are:

1. LOAD
2. SELECT
3. INSERT
4. DELETE
5. UPDATE
6. EXPORT
7. IMPORT

Let us now learn each DML command individually.

[ps2id id='LOAD-command' target="/]1. LOAD Command

The **LOAD** statement in Hive is used to move data files into the locations corresponding to Hive tables.

- If a **LOCAL** keyword is specified, then the LOAD command will look for the file path in the local filesystem.
- If the LOCAL keyword is not specified, then the Hive will need the absolute URI of the file.
- In case the keyword OVERWRITE is specified, then the contents of the target table/partition will be deleted and replaced by the files referred by filepath.

- If the OVERWRITE keyword is not specified, then the files referred by filepath will be appended to the table.

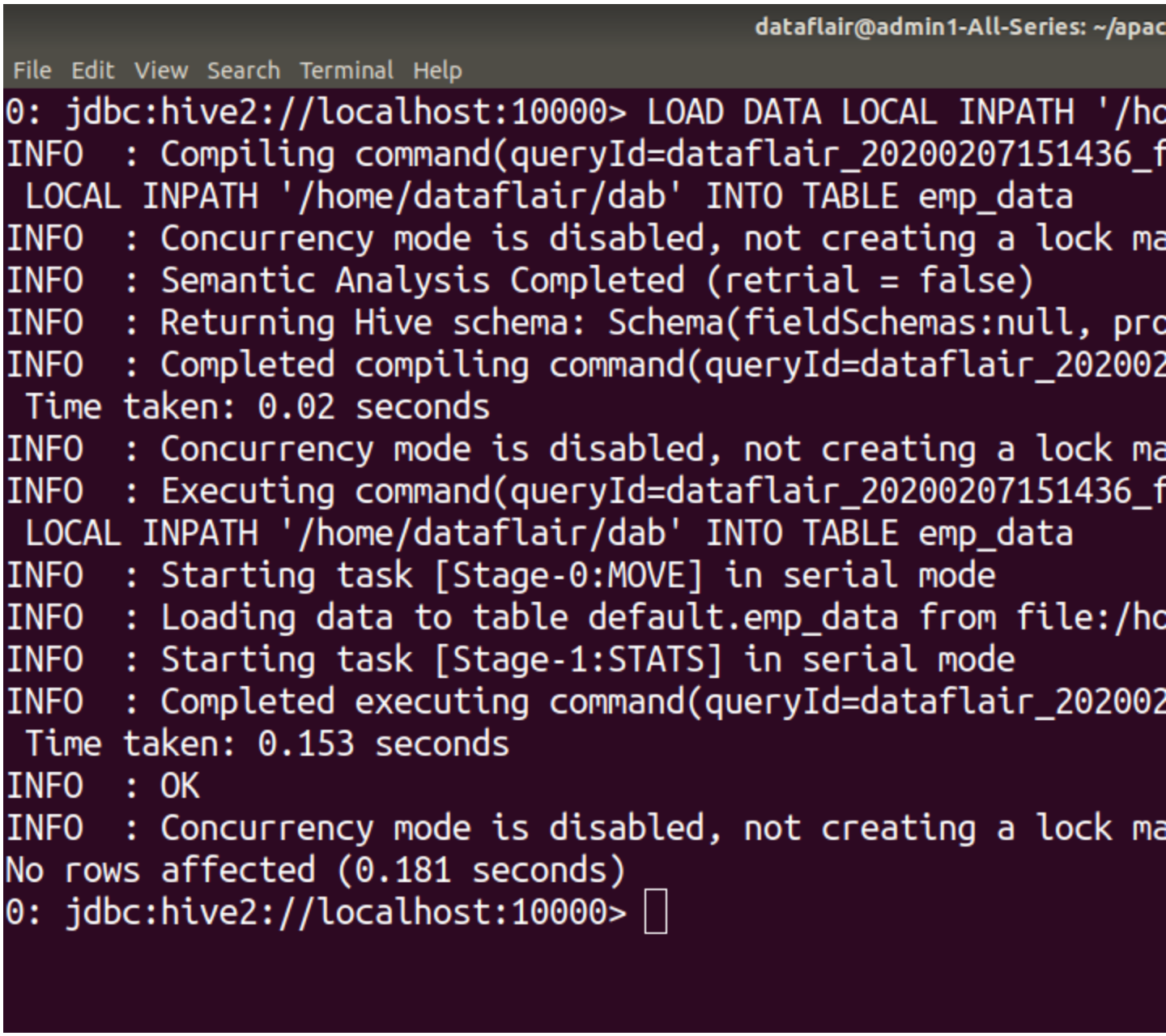
Want to practice these Hive commands yourself? Follow the **Hive-3.1.2 installation guide** to install the latest Hive version on your system.

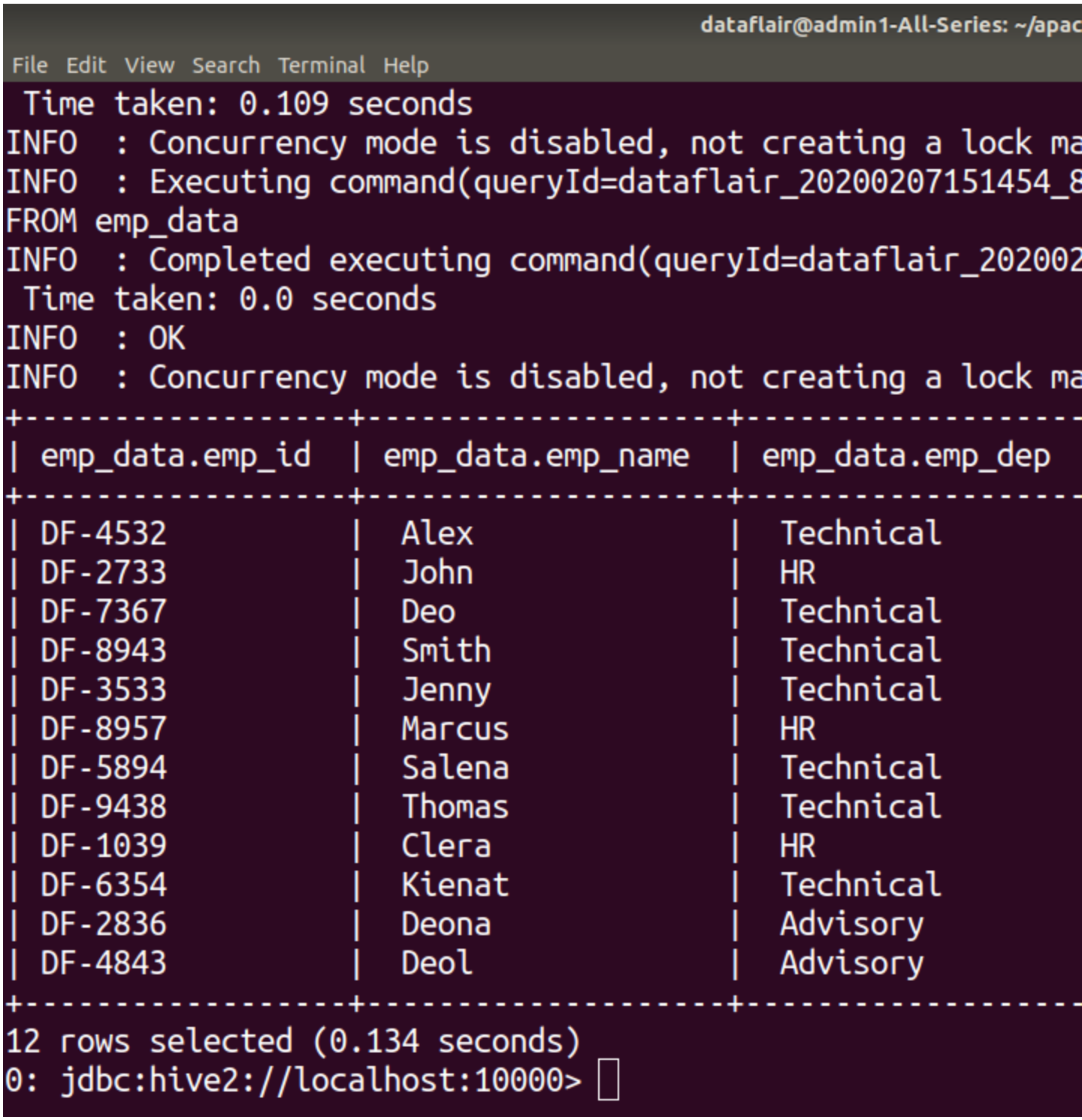
Syntax:

```
LOAD DATA [LOCAL] INPATH 'filepath' [OVERWRITE] INTO TABLE tablename [PARTITION
(partcol1=val1, partcol2=val2 ...)];
```

Example:

Here we are trying to load data from the ‘dab’ file in the local filesystem to the ‘emp_data’ table.


dataflair@admin1-All-Series: ~/apac
File Edit View Search Terminal Help
0: jdbc:hive2://localhost:10000> LOAD DATA LOCAL INPATH '/ho
INFO : Compiling command(queryId=dataflair_20200207151436_f
LOCAL INPATH '/home/dataflair/dab' INTO TABLE emp_data
INFO : Concurrency mode is disabled, not creating a lock ma
INFO : Semantic Analysis Completed (retrial = false)
INFO : Returning Hive schema: Schema(fieldSchemas:null, pro
INFO : Completed compiling command(queryId=dataflair_20200
Time taken: 0.02 seconds
INFO : Concurrency mode is disabled, not creating a lock ma
INFO : Executing command(queryId=dataflair_20200207151436_f
LOCAL INPATH '/home/dataflair/dab' INTO TABLE emp_data
INFO : Starting task [Stage-0:MOVE] in serial mode
INFO : Loading data to table default.emp_data from file:/ho
INFO : Starting task [Stage-1:STATS] in serial mode
INFO : Completed executing command(queryId=dataflair_20200
Time taken: 0.153 seconds
INFO : OK
INFO : Concurrency mode is disabled, not creating a lock ma
No rows affected (0.181 seconds)
0: jdbc:hive2://localhost:10000>



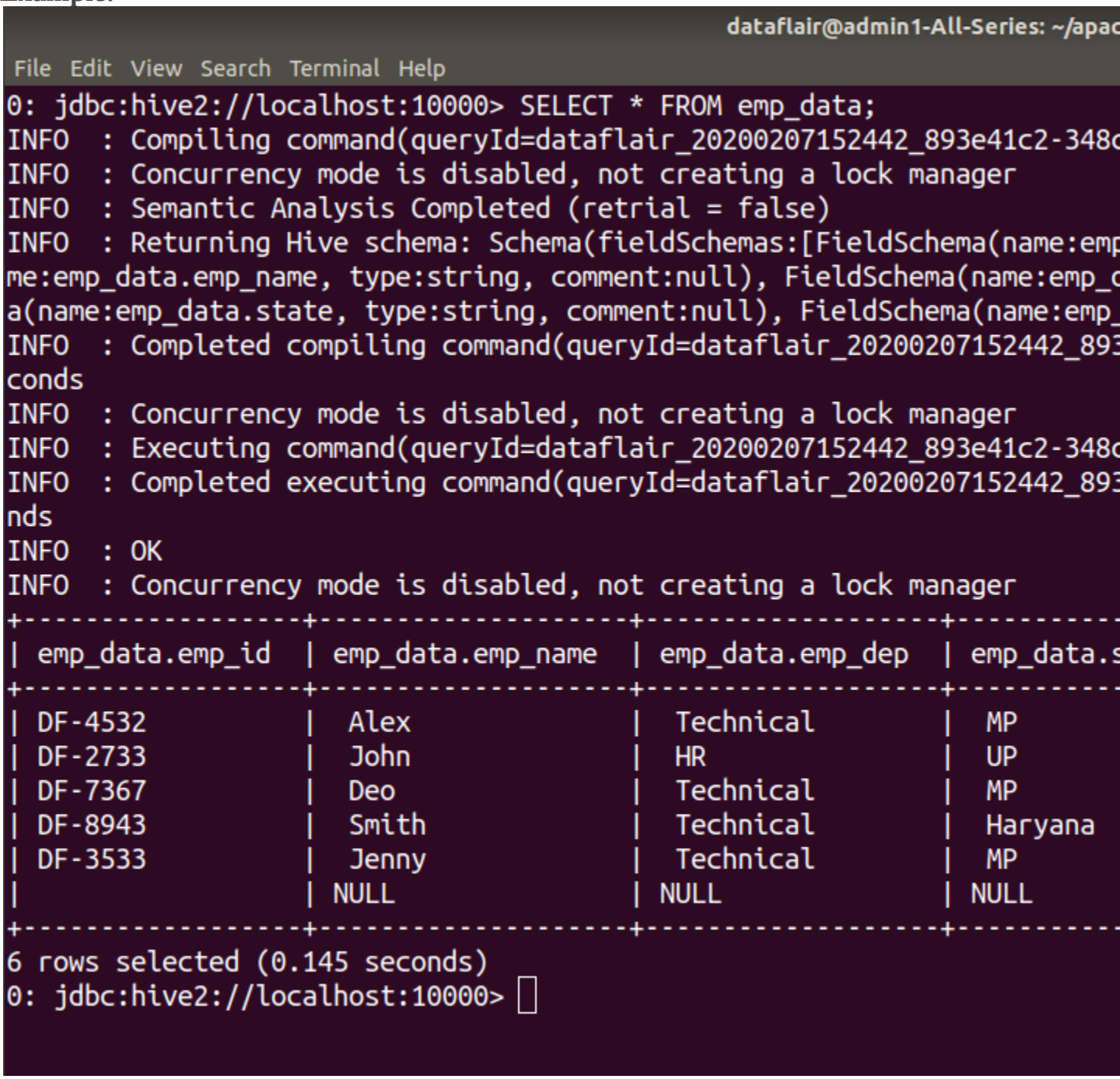
[ps2id id='SELECT-command' target="/]2. SELECT COMMAND

The SELECT statement in Hive is similar to the SELECT statement in SQL used for retrieving data from the database.

Syntax:

SELECT col1,col2 FROM tablename;

Example:



[ps2id id='INSERT-command' target="/]3. INSERT Command

The **INSERT** command in Hive loads the data into a Hive table. We can do insert to both the Hive table or partition.

a. INSERT INTO

The **INSERT INTO** statement appends the data into existing data in the table or partition. INSERT INTO statement works from Hive version 0.8.

Syntax:

INSERT INTO TABLE tablename1 [PARTITION (partcol1=val1, partcol2=val2 ...)] select_statement1 FROM from_statement;

Example:

Here in this example, we are trying to insert the data of 'emp_data' table created above into the table 'example'.

Q15. Explain about SORT BY, ORDER BY, DISTRIBUTE BY and

CLUSTER BY in Hive.

Ans:- Sort By, Order By, Distribute By, and Cluster By in Hive

The **SORT BY** and **ORDER BY** clauses are used to define the order of the output data. However, **DISTRIBUTE BY** and **CLUSTER BY** clauses are used to distribute the data to multiple reducers based on the key columns. We can use Sort by or Order by or Distribute by or Cluster by clauses in a hive SELECT query to get the output data in the desired order.

We will use the below sample table data as an input dataset to understand the difference between these four hive clauses (SORT BY, ORDER BY, DSITRIBUTE BY, and CLUSTER BY).

SalesYear	Amount
2017	100
2018	200
2019	300

SalesYear	Amount
2020	400
2017	500
2018	600
2019	700
2020	800

Sample input dataset – Table tbl_Sales

In the above dataset, table **tbl_Sales** contains two columns:

1. **SalesYear**– Year of the sale as an Integer value in YYYY format.
2. **Amount** – Total Sales Amount for the given year.

Now, let's start using the Sort by, Order by, Distribute by, and Cluster by clauses on the above table data to understand the difference and similarity among these clauses.

SORT BY

The SORT by clause **sorts the data per reducer**. As a result, if we have N number of reducers, **we will have N number of sorted files** in the output. These files can have **overlapping data ranges**. Also, the output data is not globally sorted because the hive sorts the rows before feeding them to reducers based on the key columns used in the SORT BY clause. The syntax of the SORT BY clause is as below:

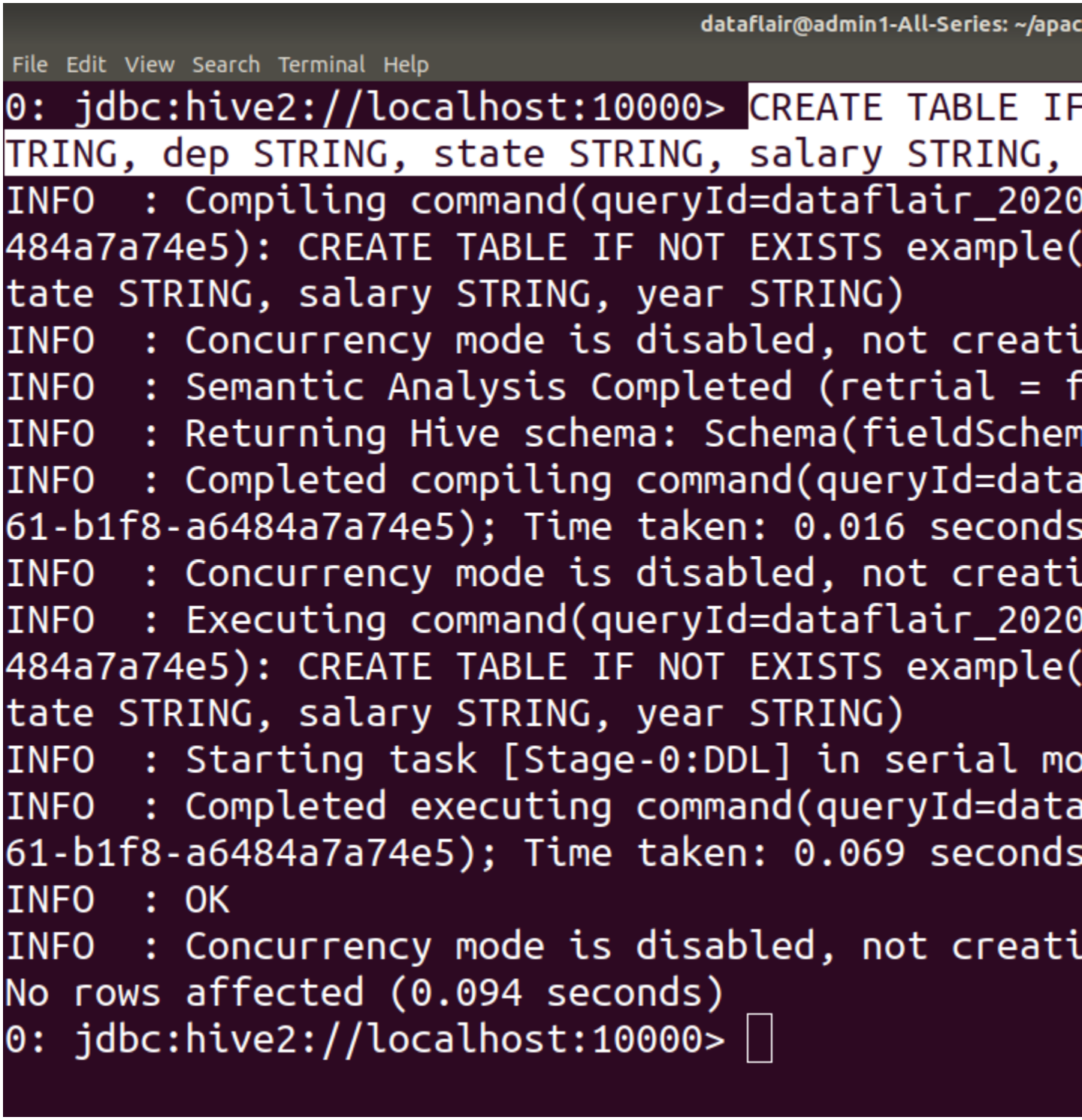
SELECT Col1, Col2,……ColN FROM TableName SORT BY Col1 <ASC | DESC>, Col2 <ASC | DESC>, …. ColN <ASC | DESC>

?

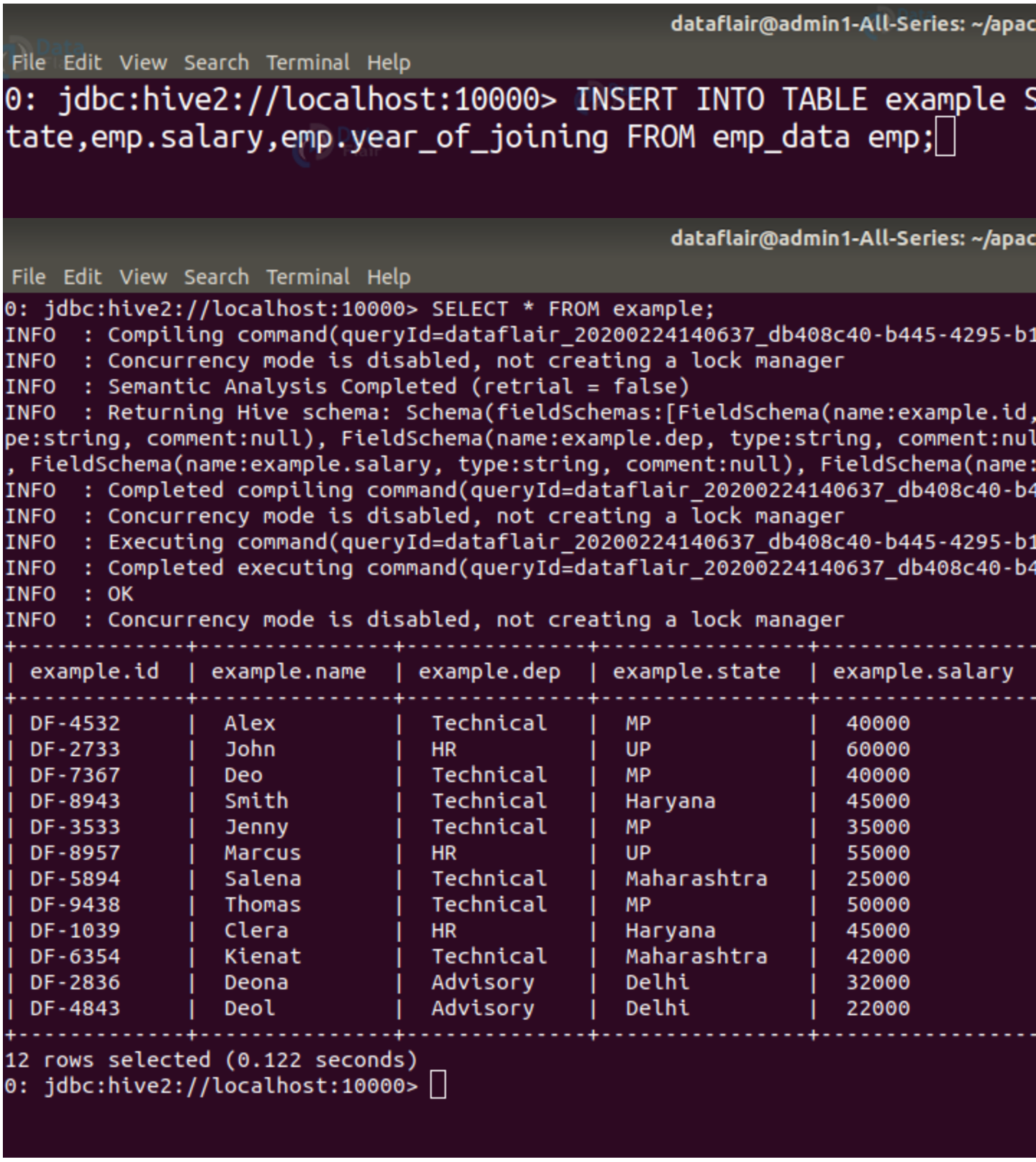
```
1  SELECT SalesYear, Amount
2  FROM tbl_Sales
3  SORT BY SalesYear;
```

Now, suppose we have two reducers to execute this query. So, in the final output, we will get two locally sorted files based on the **SalesYear** column values from each of the reducers. That means the files are sorted per reducer basis only, and not globally. **Note that the final output data is not globally sorted and may have overlapping data ranges.**

The output of the above query with 2 reducers will look like this:



INSERT statement to load data into table “example”.


dataflair@admin1-All-Series: ~/apac
File Edit View Search Terminal Help
0: jdbc:hive2://localhost:10000> INSERT INTO TABLE example S
tate,emp.salary,emp.year_of_joining FROM emp_data emp;
dataflair@admin1-All-Series: ~/apac
File Edit View Search Terminal Help
0: jdbc:hive2://localhost:10000> SELECT * FROM example;
INFO : Compiling command(queryId=dataflair_20200224140637_db408c40-b445-4295-b1
INFO : Concurrency mode is disabled, not creating a lock manager
INFO : Semantic Analysis Completed (retrial = false)
INFO : Returning Hive schema: Schema(fieldSchemas:[FieldSchema(name:example.id,
pe:string, comment:null), FieldSchema(name:example.dep, type:string, comment:nul
, FieldSchema(name:example.salary, type:string, comment:null), FieldSchema(name:
INFO : Completed compiling command(queryId=dataflair_20200224140637_db408c40-b4
INFO : Concurrency mode is disabled, not creating a lock manager
INFO : Executing command(queryId=dataflair_20200224140637_db408c40-b445-4295-b1
INFO : Completed executing command(queryId=dataflair_20200224140637_db408c40-b4
INFO : OK
INFO : Concurrency mode is disabled, not creating a lock manager
+-------------+---------------+-------------+----------------+-----------------
| example.id | example.name | example.dep | example.state | example.salary
+-------------+---------------+-------------+----------------+-----------------
| DF-4532 | Alex | Technical | MP | 40000
| DF-2733 | John | HR | UP | 60000
| DF-7367 | Deo | Technical | MP | 40000
| DF-8943 | Smith | Technical | Haryana | 45000
| DF-3533 | Jenny | Technical | MP | 35000
| DF-8957 | Marcus | HR | UP | 55000
| DF-5894 | Salena | Technical | Maharashtra | 25000
| DF-9438 | Thomas | Technical | MP | 50000
| DF-1039 | Clera | HR | Haryana | 45000
| DF-6354 | Kienat | Technical | Maharashtra | 42000
| DF-2836 | Deona | Advisory | Delhi | 32000
| DF-4843 | Deol | Advisory | Delhi | 22000
+-------------+---------------+-------------+----------------+-----------------
12 rows selected (0.122 seconds)
0: jdbc:hive2://localhost:10000>

b. INSERT OVERWRITE

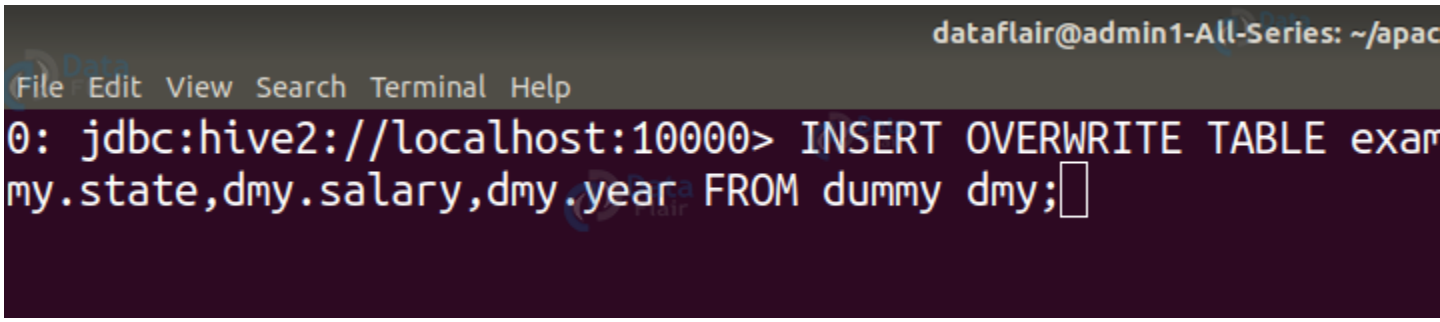
The **INSERT OVERWRITE** table overwrites the existing data in the table or partition.

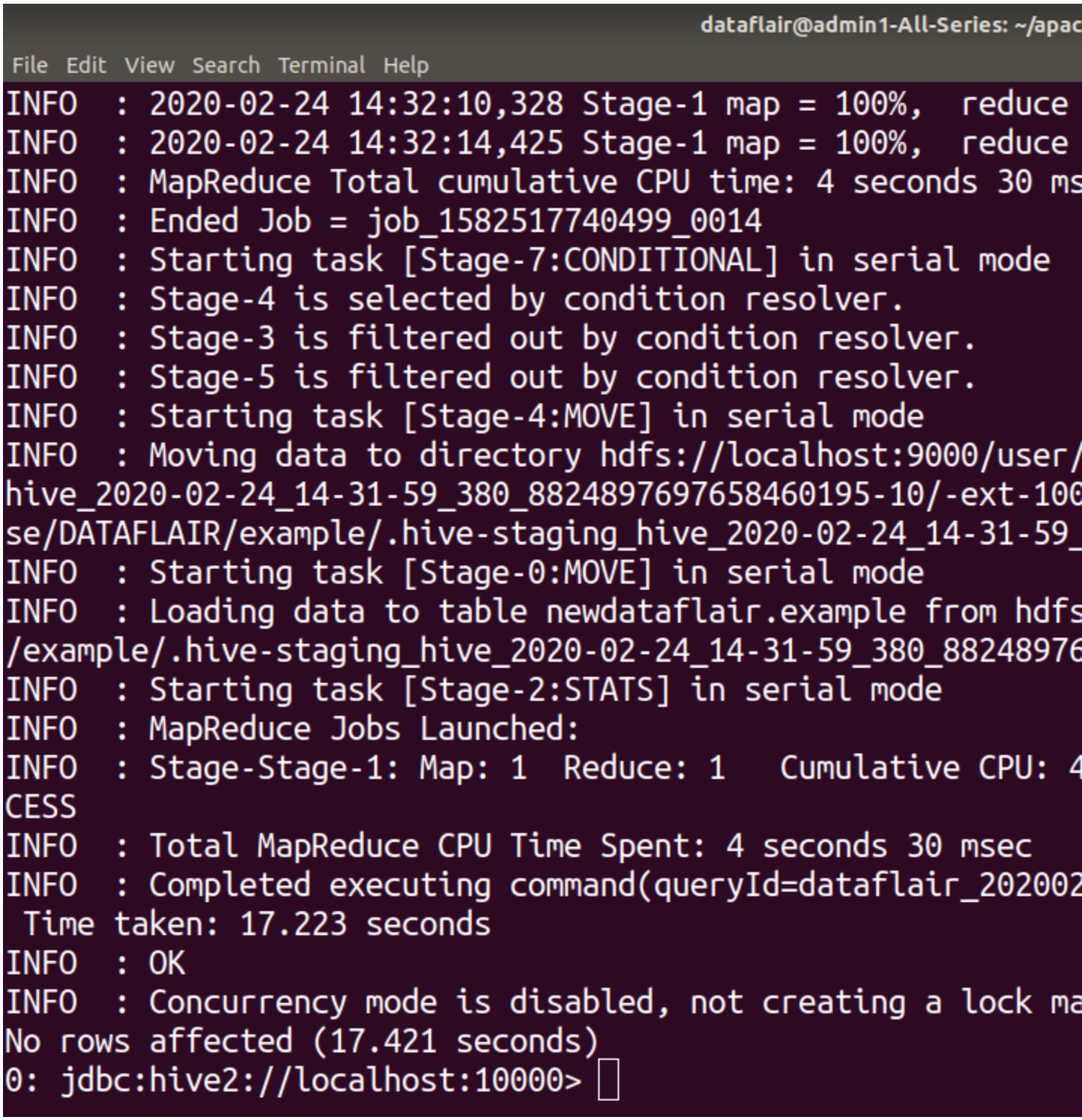
Syntax:

INSERT OVERWRITE TABLE tablename1 [PARTITION (partcol1=val1, ..) [**IF** NOT EXISTS]] select_statement FROM from_statement;

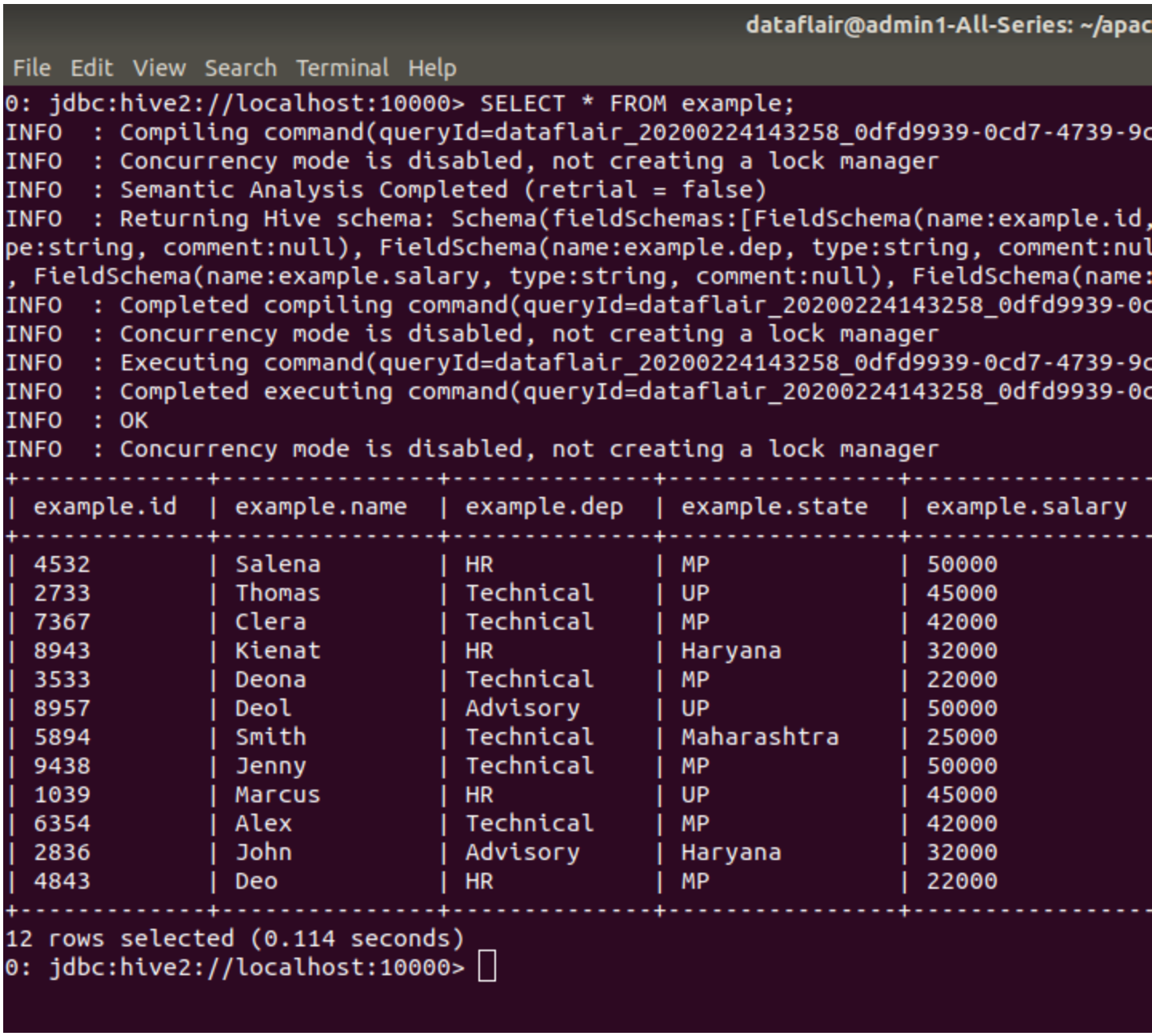
Example:

Here we are overwriting the existing data of the table ‘example’ with the data of table ‘dummy’ using INSERT OVERWRITE statement.





By using the SELECT statement we can verify whether the existing data of the table ‘example’ is overwritten by the data of table ‘dummy’ or not.



c. INSERT .. VALUES

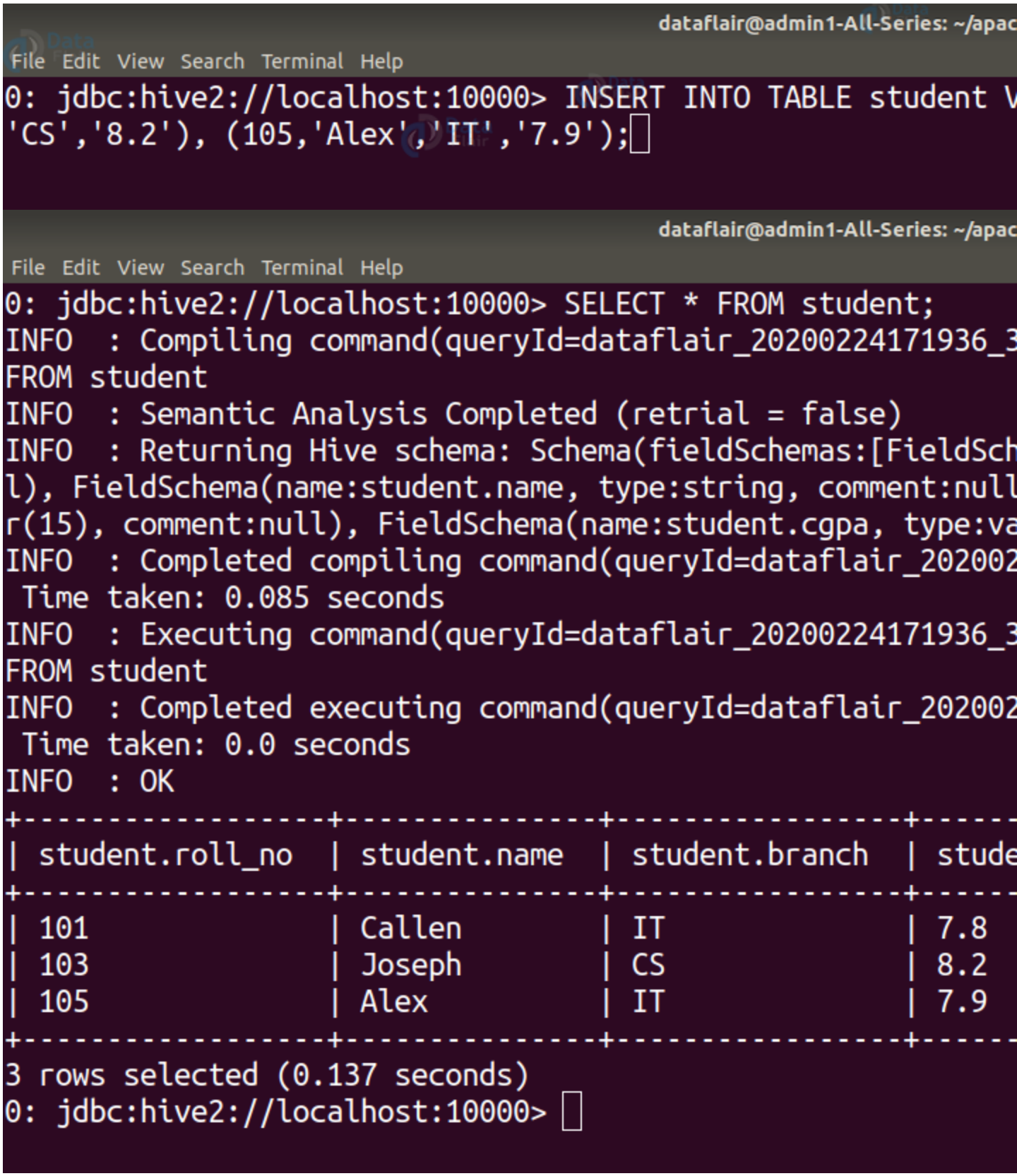
INSERT ..VALUES statement in Hive inserts data into the table directly from SQL. It is available from Hive 0.14.

Syntax:

INSERT INTO TABLE tablename [PARTITION (partcol1[=val1], partcol2[=val2] ...)] VALUES values_row [, values_row ...];

Example:

Inserting data into the ‘student’ table using INSERT ..VALUES statement.


dataflair@admin1-All-Series: ~/apac
File Edit View Search Terminal Help
0: jdbc:hive2://localhost:10000> INSERT INTO TABLE student V
'CS','8.2'), (105,'Alex','IT','7.9');
dataflair@admin1-All-Series: ~/apac
File Edit View Search Terminal Help
0: jdbc:hive2://localhost:10000> SELECT * FROM student;
INFO : Compiling command(queryId=dataflair_20200224171936_3
FROM student
INFO : Semantic Analysis Completed (retrial = false)
INFO : Returning Hive schema: Schema(fieldSchemas:[FieldSch
l), FieldSchema(name:student.name, type:string, comment:null
r(15), comment:null), FieldSchema(name:student.cgpa, type:va
INFO : Completed compiling command(queryId=dataflair_202002
Time taken: 0.085 seconds
INFO : Executing command(queryId=dataflair_20200224171936_3
FROM student
INFO : Completed executing command(queryId=dataflair_202002
Time taken: 0.0 seconds
INFO : OK
| student.roll_no | student.name | student.branch | stude
| 101 | Callen | IT | 7.8
| 103 | Joseph | CS | 8.2
| 105 | Alex | IT | 7.9
3 rows selected (0.137 seconds)
0: jdbc:hive2://localhost:10000>

[ps2id id=’DELETE-command’ target=”/]4. DELETE command

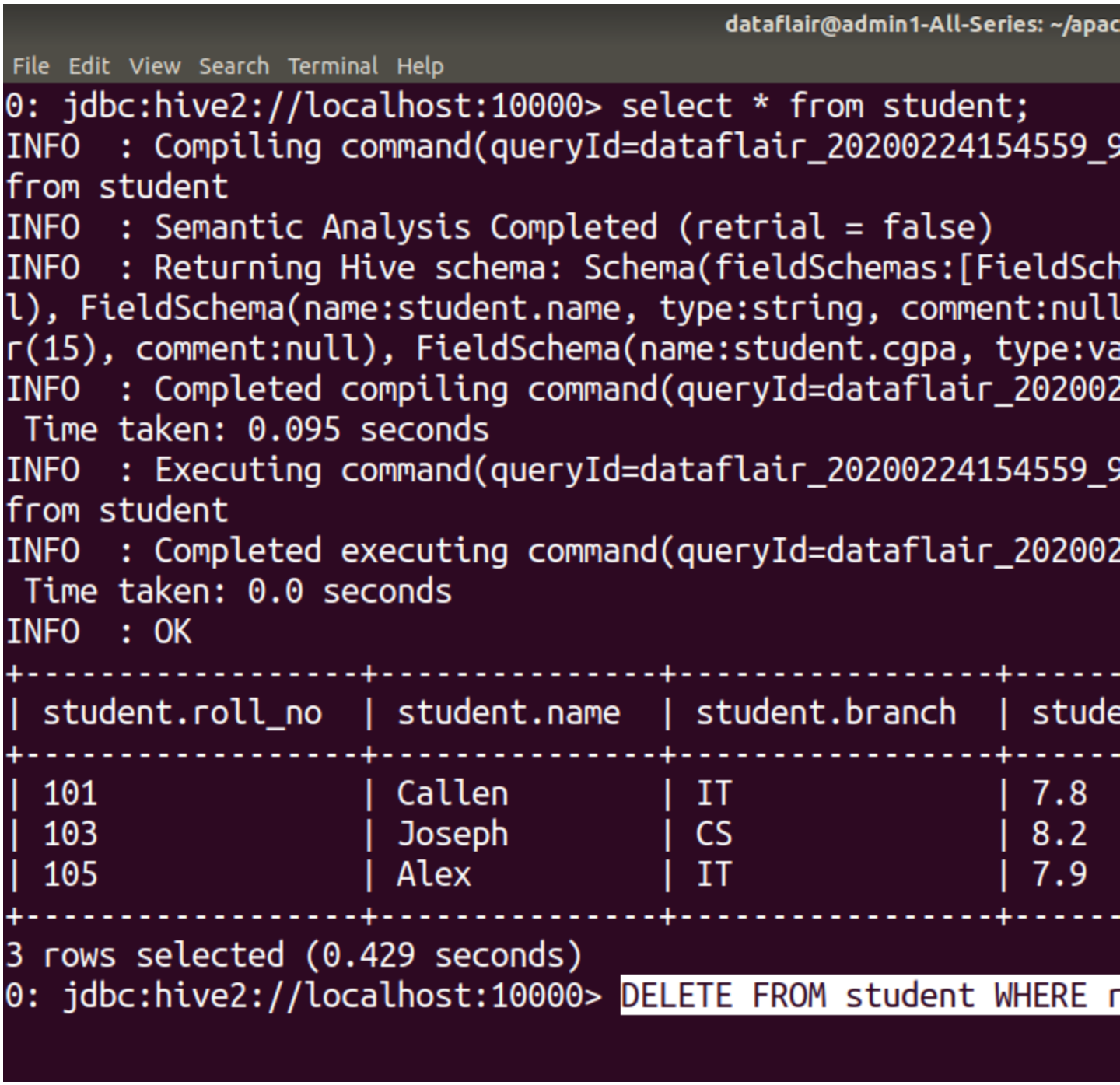
The DELETE statement in Hive deletes the table data. If the WHERE clause is specified, then it deletes the rows that satisfy the condition in where clause.

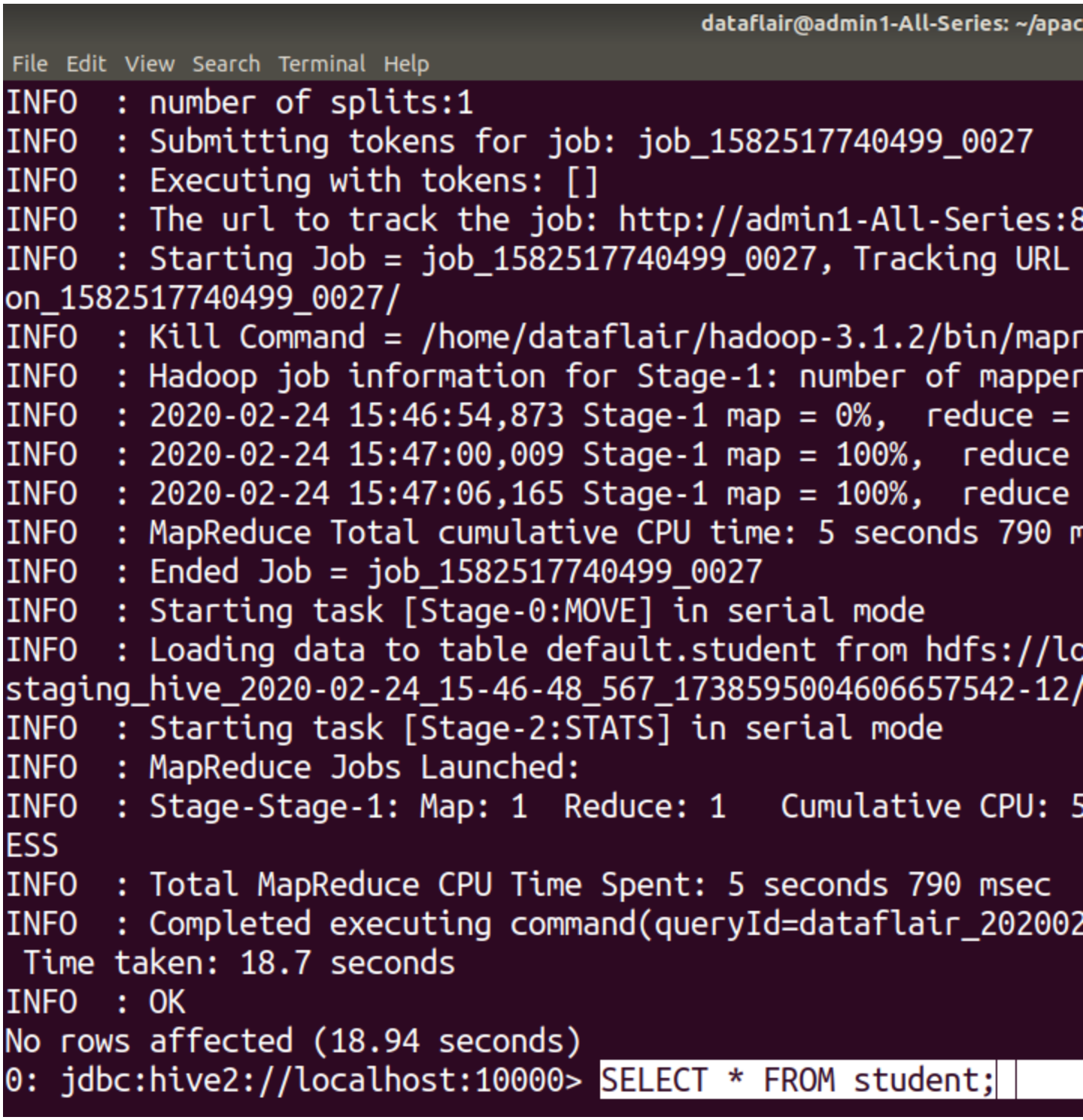
The DELETE statement can only be used on the hive tables that support ACID.

Syntax:

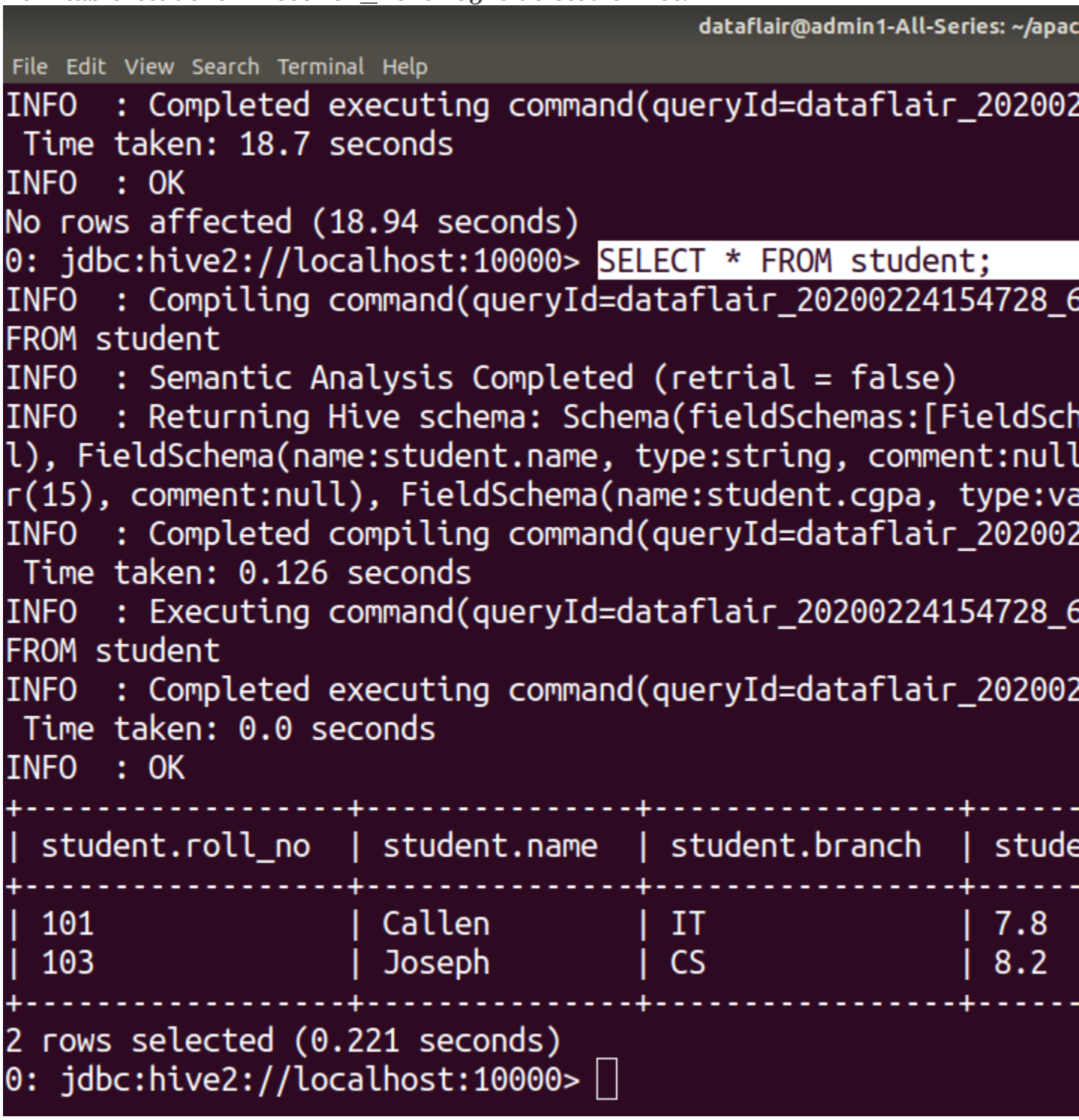
DELETE FROM tablename [WHERE expression];

Example:
In the below example, we are deleting the data of the student from table ‘student’ whose roll_no is 105.


dataflair@admin1-All-Series: ~/apac
File Edit View Search Terminal Help
0: jdbc:hive2://localhost:10000> select * from student;
INFO : Compiling command(queryId=dataflair_20200224154559_9
from student
INFO : Semantic Analysis Completed (retrial = false)
INFO : Returning Hive schema: Schema(fieldSchemas:[FieldSch
l), FieldSchema(name:student.name, type:string, comment:null
r(15), comment:null), FieldSchema(name:student.cgpa, type:va
INFO : Completed compiling command(queryId=dataflair_202002
Time taken: 0.095 seconds
INFO : Executing command(queryId=dataflair_20200224154559_9
from student
INFO : Completed executing command(queryId=dataflair_202002
Time taken: 0.0 seconds
INFO : OK
| student.roll_no | student.name | student.branch | stude
| 101 | Callen | IT | 7.8
| 103 | Joseph | CS | 8.2
| 105 | Alex | IT | 7.9
3 rows selected (0.429 seconds)
0: jdbc:hive2://localhost:10000> DELETE FROM student WHERE r


dataflair@admin1-All-Series: ~/apac
File Edit View Search Terminal Help
INFO : number of splits:1
INFO : Submitting tokens for job: job_1582517740499_0027
INFO : Executing with tokens: []
INFO : The url to track the job: http://admin1-All-Series:8
INFO : Starting Job = job_1582517740499_0027, Tracking URL
on_1582517740499_0027/
INFO : Kill Command = /home/dataflair/hadoop-3.1.2/bin/mapr
INFO : Hadoop job information for Stage-1: number of mapper
INFO : 2020-02-24 15:46:54,873 Stage-1 map = 0%, reduce =
INFO : 2020-02-24 15:47:00,009 Stage-1 map = 100%, reduce
INFO : 2020-02-24 15:47:06,165 Stage-1 map = 100%, reduce
INFO : MapReduce Total cumulative CPU time: 5 seconds 790 m
INFO : Ended Job = job_1582517740499_0027
INFO : Starting task [Stage-0:MOVE] in serial mode
INFO : Loading data to table default.student from hdfs://lo
staging_hive_2020-02-24_15-46-48_567_17385950046066575 42-12/
INFO : Starting task [Stage-2:STATS] in serial mode
INFO : MapReduce Jobs Launched:
INFO : Stage-Stage-1: Map: 1 Reduce: 1 Cumulative CPU: 5
ESS
INFO : Total MapReduce CPU Time Spent: 5 seconds 790 msec
INFO : Completed executing command(queryId=dataflair_202002
Time taken: 18.7 seconds
INFO : OK
No rows affected (18.94 seconds)
0: jdbc:hive2://localhost:10000> SELECT * FROM student;

By using the SELECT statement we can verify whether the data of the student from table ‘student’ whose roll_no is 105 is deleted or not.



[ps2id id=’UPDATE-command’ target=”/]5. UPDATE Command

The update can be performed on the hive tables that support ACID.

The UPDATE statement in Hive deletes the table data. If the WHERE clause is specified, then it updates the column of the rows that satisfy the condition in WHERE clause.

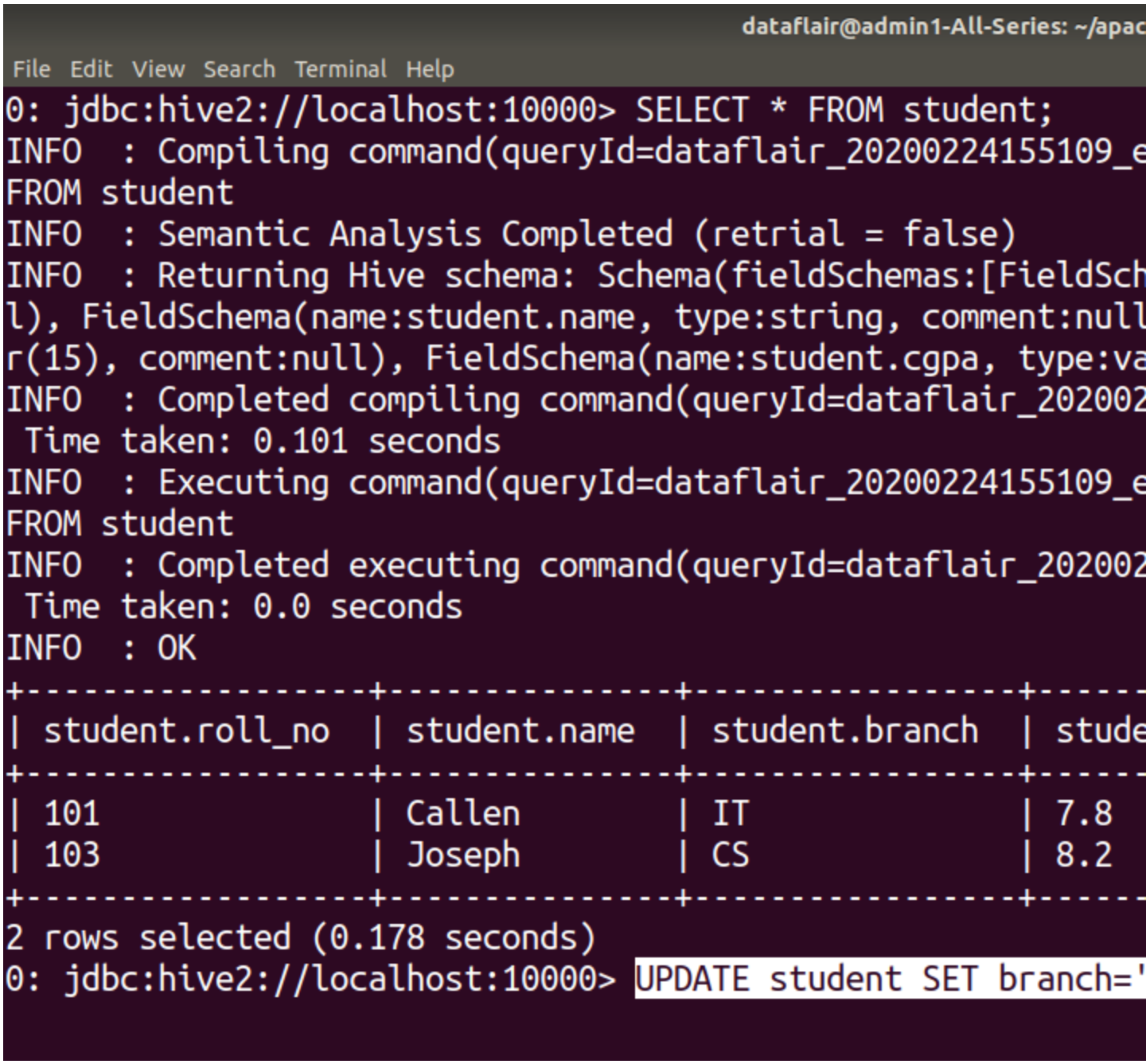
Partitioning and **Bucketing** columns cannot be updated.

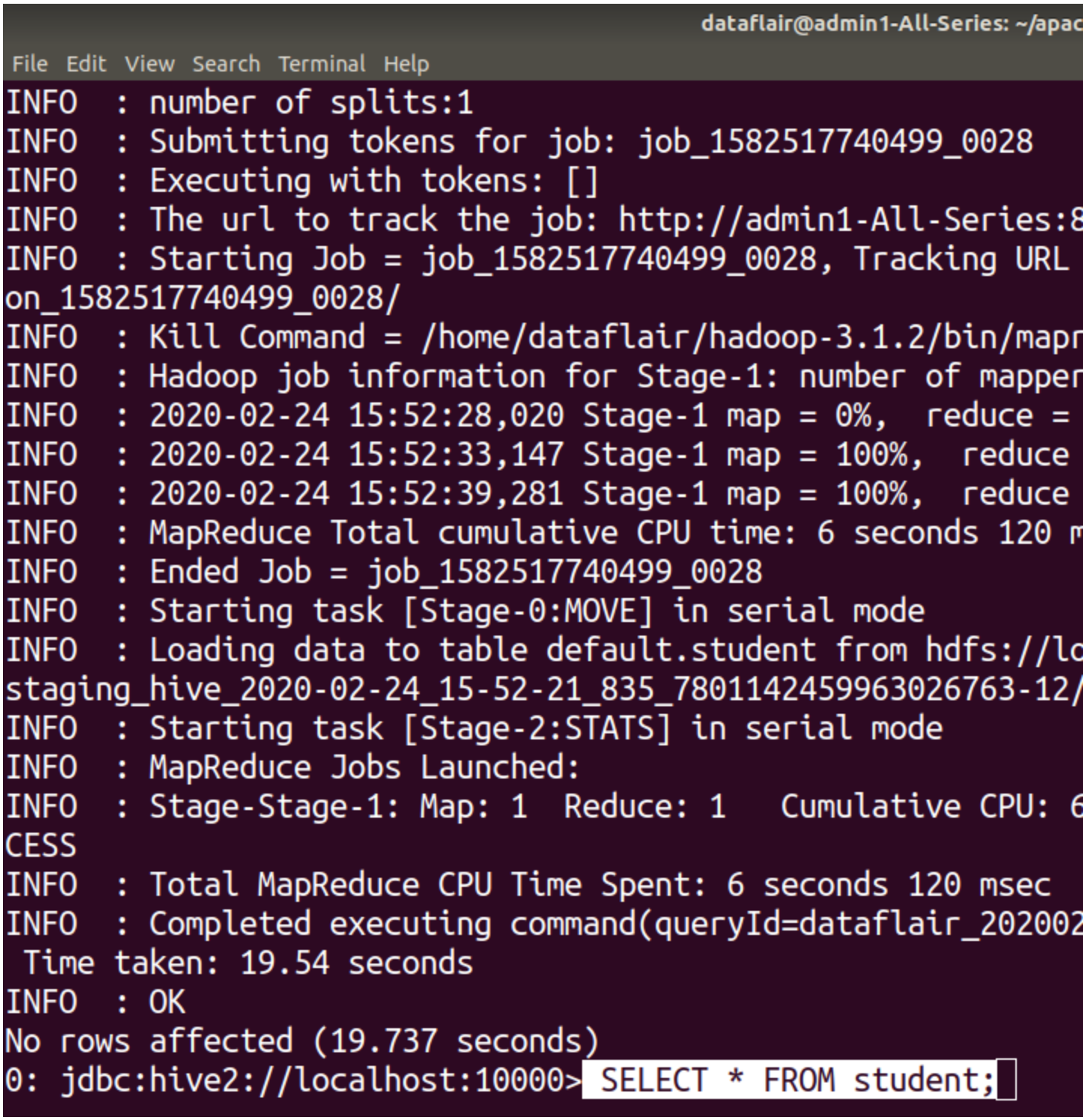
Syntax:

```
UPDATE tablename SET column = value [, column = value ...] [WHERE expression];
```

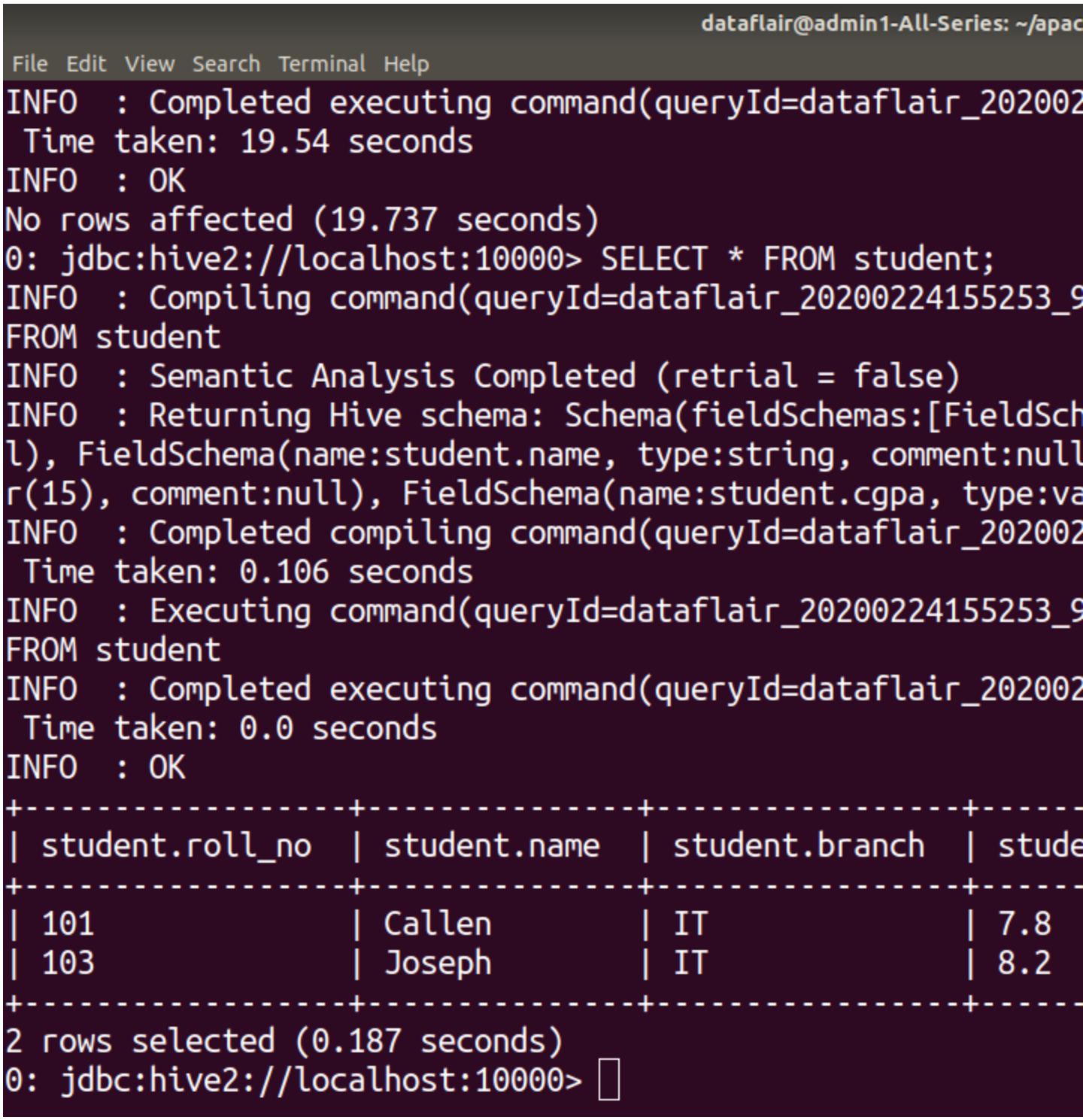
Example:

In this example, we are updating the branch of the student whose roll_no is 103 in the 'student' table using an UPDATE statement.


dataflair@admin1-All-Series: ~/apac
File Edit View Search Terminal Help
0: jdbc:hive2://localhost:10000> SELECT * FROM student;
INFO : Compiling command(queryId=dataflair_20200224155109_e
FROM student
INFO : Semantic Analysis Completed (retrial = false)
INFO : Returning Hive schema: Schema(fieldSchemas:[FieldSch
l), FieldSchema(name:student.name, type:string, comment:null
r(15), comment:null), FieldSchema(name:student.cgpa, type:va
INFO : Completed compiling command(queryId=dataflair_202002
Time taken: 0.101 seconds
INFO : Executing command(queryId=dataflair_20200224155109_e
FROM student
INFO : Completed executing command(queryId=dataflair_202002
Time taken: 0.0 seconds
INFO : OK
| student.roll_no | student.name | student.branch | stude
| 101 | Callen | IT | 7.8
| 103 | Joseph | CS | 8.2
2 rows selected (0.178 seconds)
0: jdbc:hive2://localhost:10000> UPDATE student SET branch='



By using the SELECT statement we can verify whether the branch of the student whose roll_no is 103 in the ‘student’ table is updated or not.



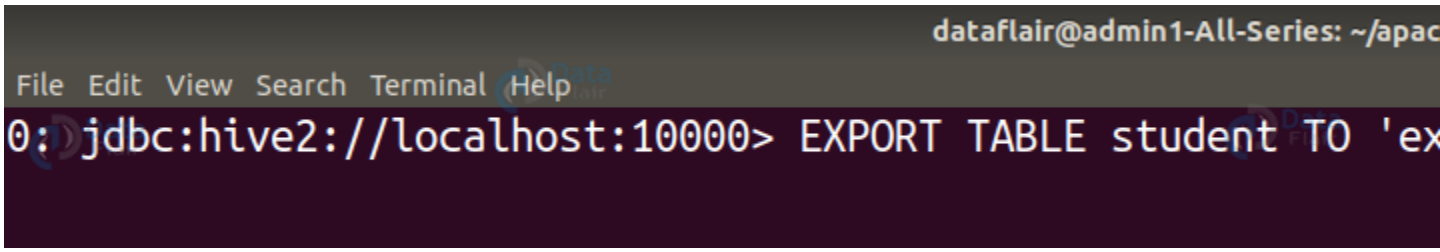
[ps2id id='EXPORT-command' target="/]6. EXPORT Command

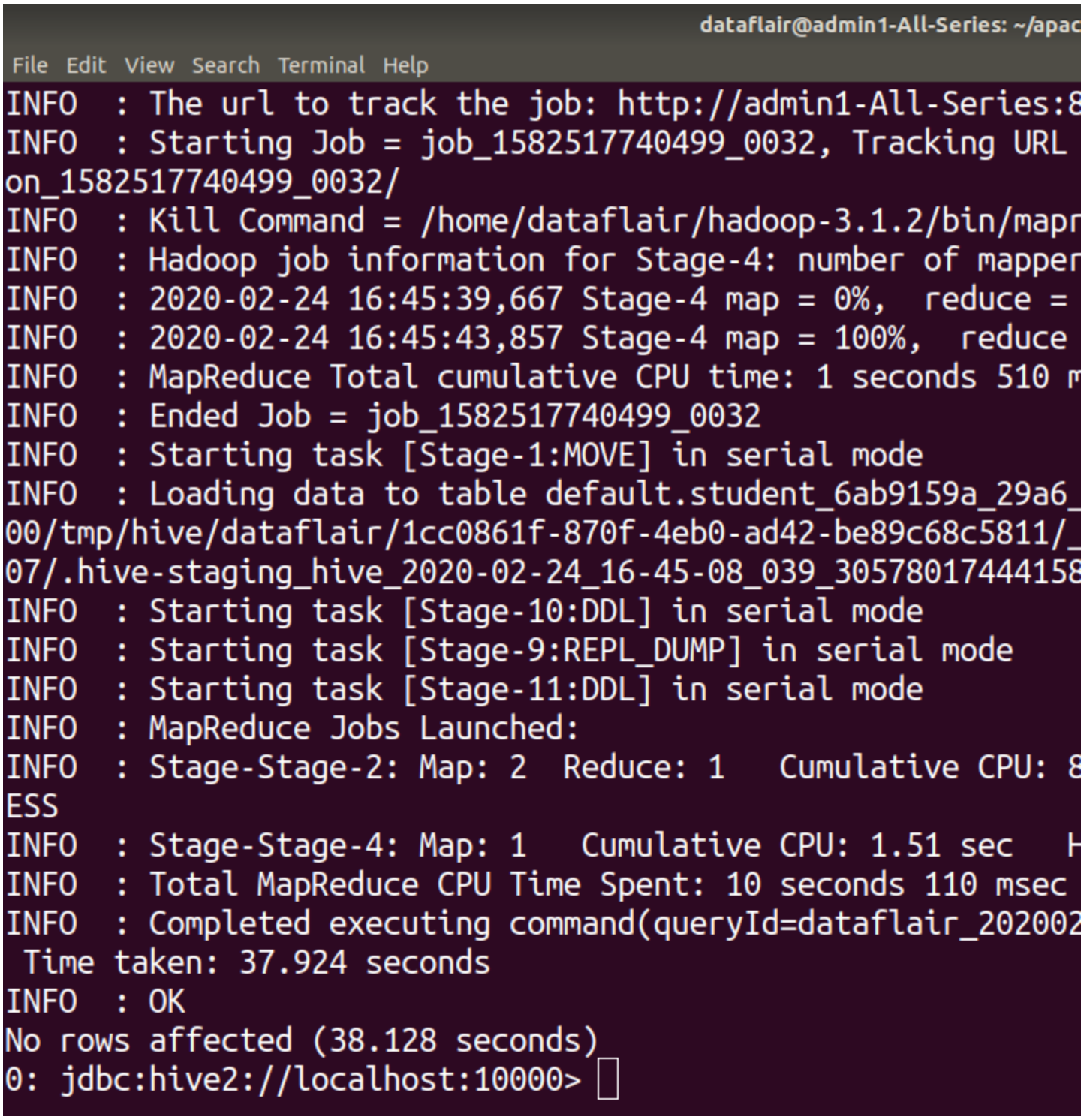
The Hive **EXPORT** statement exports the table or **partition** data along with the metadata to the specified output location in the HDFS.
Metadata is exported in a **_metadata** file, and data is exported in a subdirectory **'data.'**

Syntax:

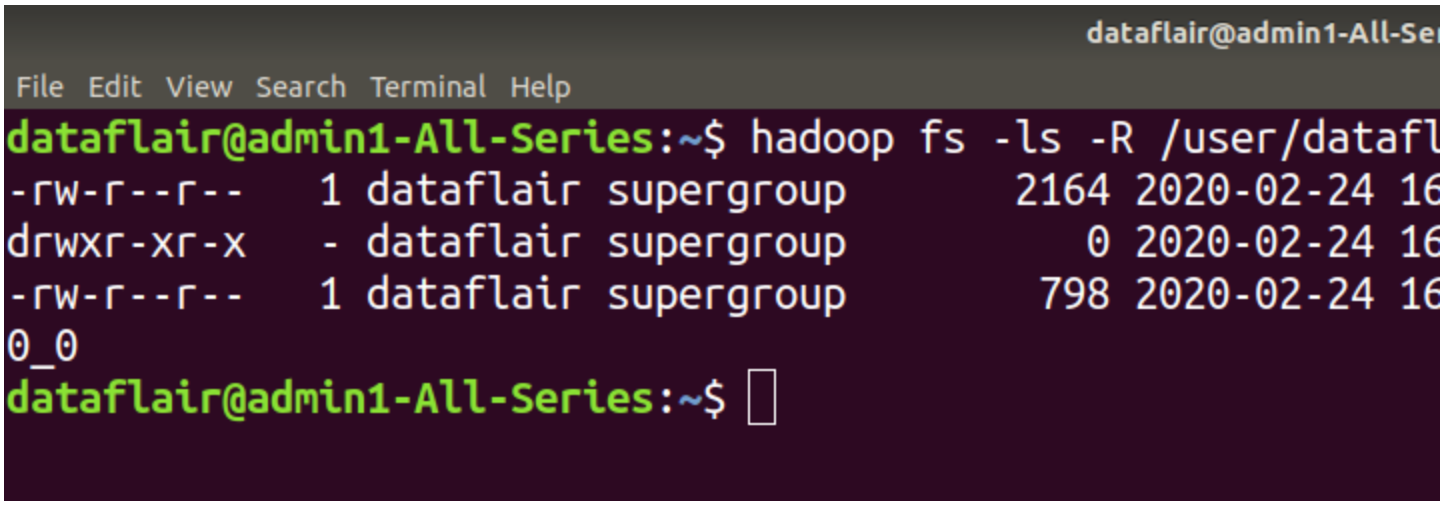
```
EXPORT TABLE tablename [PARTITION (part_column="value"[, ...])]

TO 'export_target_path' [ FOR replication('eventid') ];
```

Example:
Here in this example, we are exporting the student table to the HDFS directory "export_from_hive".





The table successfully exported. You can check for the _metadata file and data sub-directory using **ls** command.



[ps2id id='IMPORT-command' target="/]7. IMPORT Command

The Hive IMPORT command imports the data from a specified location to a new table or already existing table.

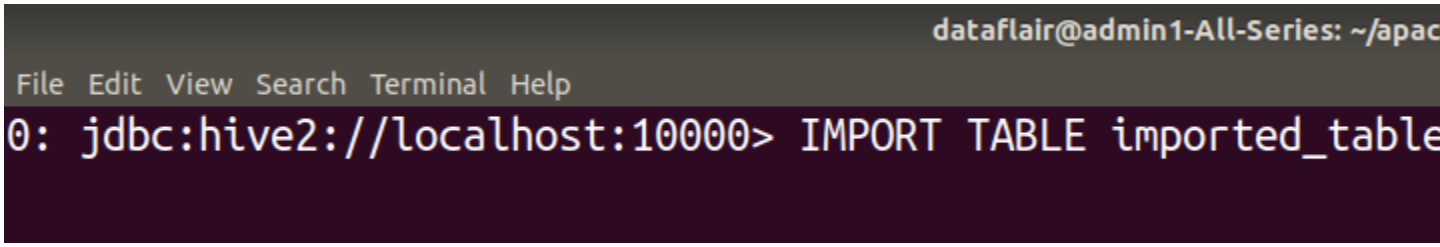
Syntax:

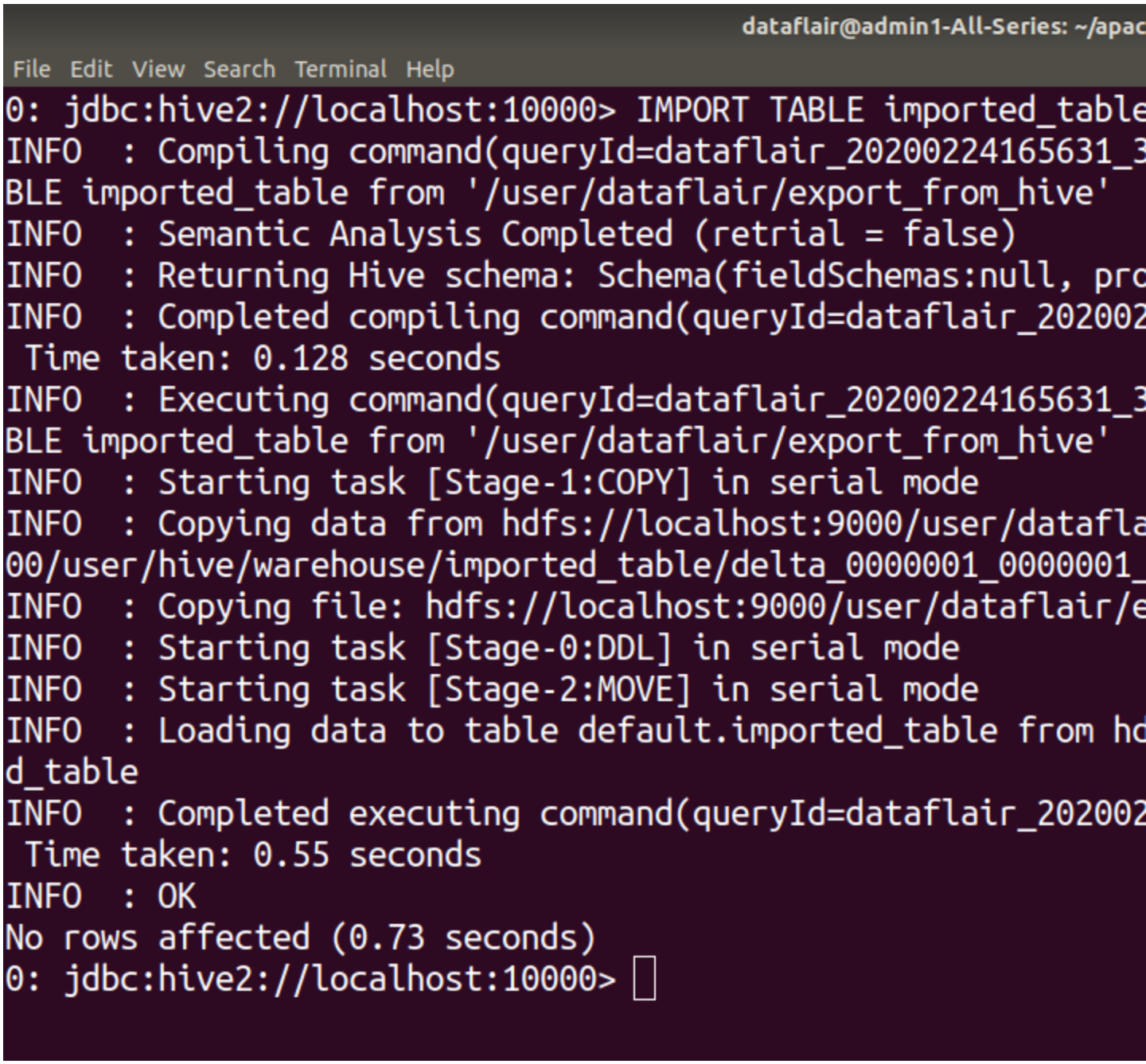
IMPORT [[EXTERNAL] TABLE new_or_original_tablename [PARTITION (part_column="value"[, ...])]]

FROM 'source_path' [LOCATION 'import_target_path'];

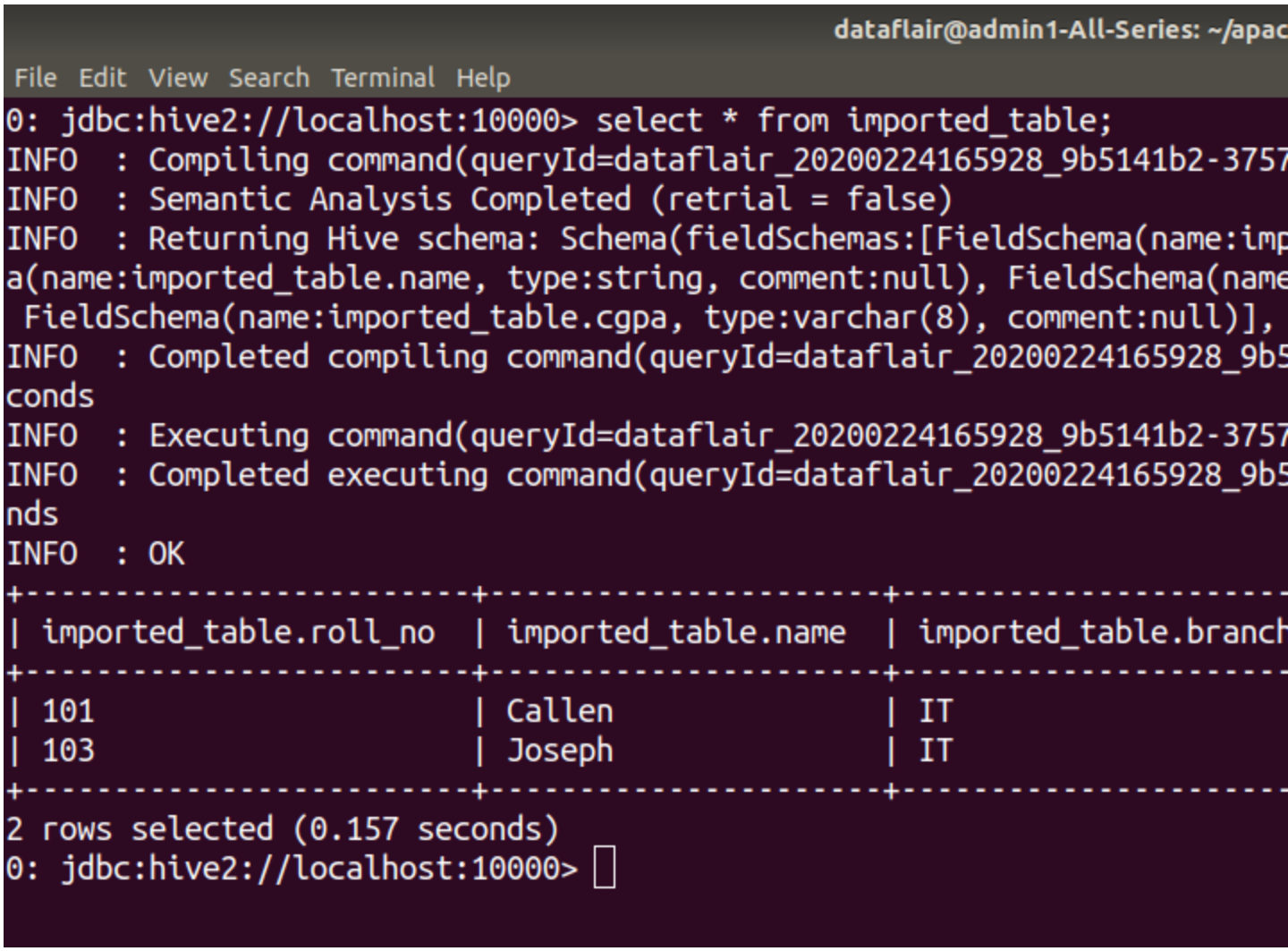
Example:

Here in this example, we are importing the data exported in the above example into a new Hive table 'imported_table'.

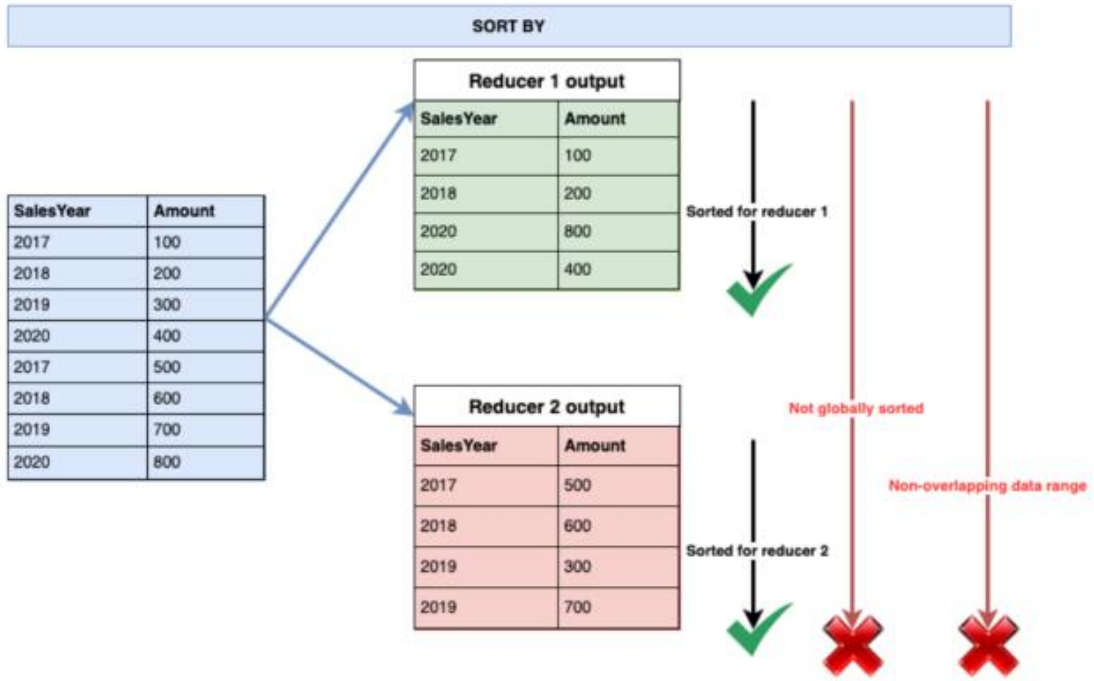




Verifying whether the data is imported or not using hive SELECT statement.


dataflair@admin1-All-Series: ~/apac
File Edit View Search Terminal Help
0: jdbc:hive2://localhost:10000> select * from imported_table;
INFO : Compiling command(queryId=dataflair_20200224165928_9b5141b2-3757
INFO : Semantic Analysis Completed (retrial = false)
INFO : Returning Hive schema: Schema(fieldSchemas:[FieldSchema(name:imp
a(name:imported_table.name, type:string, comment:null), FieldSchema(name
FieldSchema(name:imported_table.cgpa, type:varchar(8), comment:null)],
INFO : Completed compiling command(queryId=dataflair_20200224165928_9b5
conds
INFO : Executing command(queryId=dataflair_20200224165928_9b5141b2-3757
INFO : Completed executing command(queryId=dataflair_20200224165928_9b5
nds
INFO : OK
| imported_table.roll_no | imported_table.name | imported_table.branch
| 101 | Callen | IT
| 103 | Joseph | IT
2 rows selected (0.157 seconds)
0: jdbc:hive2://localhost:10000>

The output of the above query with 2 reducers will look like this:



How SORT BY works in hive

ORDER BY

ORDER BY clause **orders the data globally**. Because it ensures the global ordering of the data, all the **data need to be passed from a single reducer only**. As a result, the order by clause **outputs one single file only**. Bringing all the data on one single reducer can become a performance killer, especially if our output dataset is significantly large. So, we should always avoid the ORDER BY clause in the hive queries. However, if we need to enforce a global ordering of the data, and the output dataset is not that big, we can use this hive clause to order the final dataset globally.

The syntax of the ORDER BY clause in hive is as below:

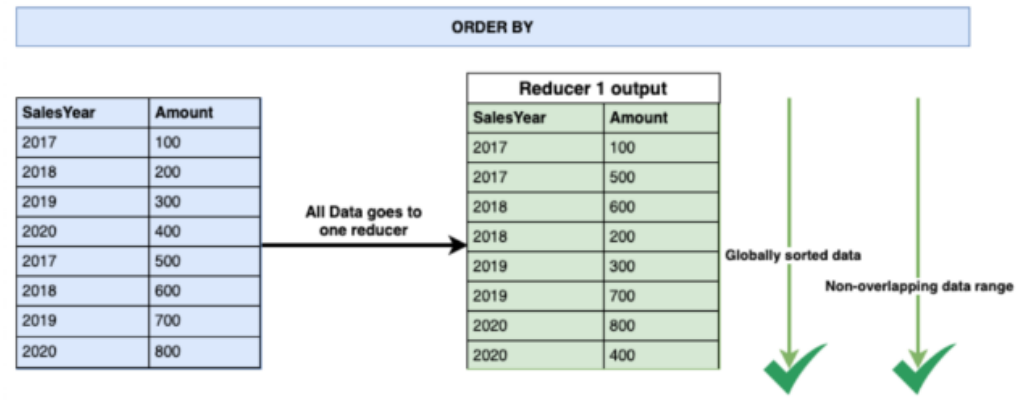
SELECT Col1, Col2,.....ColN FROM TableName ORDER BY Col1 <ASC | DESC>, Col2 <ASC | DESC>, ... ColN <ASC | DESC>

?

```
1    SELECT SalesYear, Amount
```

```
2    FROM tbl_Sales
3    SORT BY SalesYear;
```

The output of the above query will look like this:



How ORDER BY works in hive

DISTRIBUTE BY

DISTRIBUTE BY clause is used to **distribute the input rows among reducers**. It ensures that all rows for the same key columns are going to the same reducer. So, if we need to partition the data on some key column, we can use the DISTRIBUTE BY clause in the hive queries. However, the DISTRIBUTE BY clause **does not sort the data either at the reducer level or globally**. Also, the same key values might not be placed next to each other in the output dataset.

As a result, the DISTRIBUTE BY clause may **output N number of unsorted files** where N is the number of reducers used in the query processing. But, the output files do not contain overlapping data ranges.

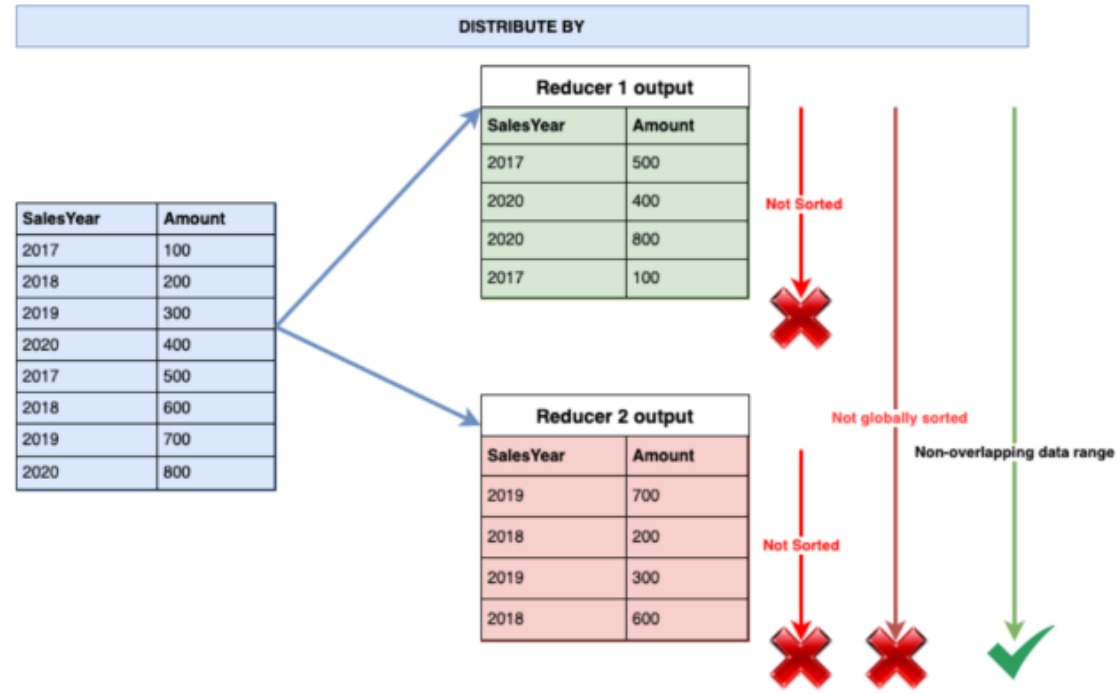
The syntax of the DISTRIBUTE BY clause in hive is as below:

SELECT Col1, Col2,.....ColN FROM TableName DISTRIBUTE BY Col1, Col2, ... ColN

?

```
SELECT SalesYear, Amount
FROM tbl_Sales
DISTRIBUTE BY SalesYear;
```

The output of the above query is as below:



How DISTRIBUTE BY works in hive

CLUSTER BY

CLUSTER BY clause **is a combination of DISTRIBUTE BY and SORT BY clauses together**. That means the output of the CLUSTER BY clause is equivalent to the output of **DISTRIBUTE BY + SORT BY** clauses. The CLUSTER BY clause distributes the data based on the key column and then sorts the output data by putting the **same key column values adjacent to each other**. So, the output of the CLUSTER BY clause is **sorted at the reducer level**. As a result, we can get **N number of sorted output files** where N is the number of reducers used in the query processing. Also, the CLUSTER by clause ensures that we are getting **non-overlapping data ranges** into the final outputs. However, if the query is processed by only one reducer the output will be equivalent to the output of the ORDER BY clause.

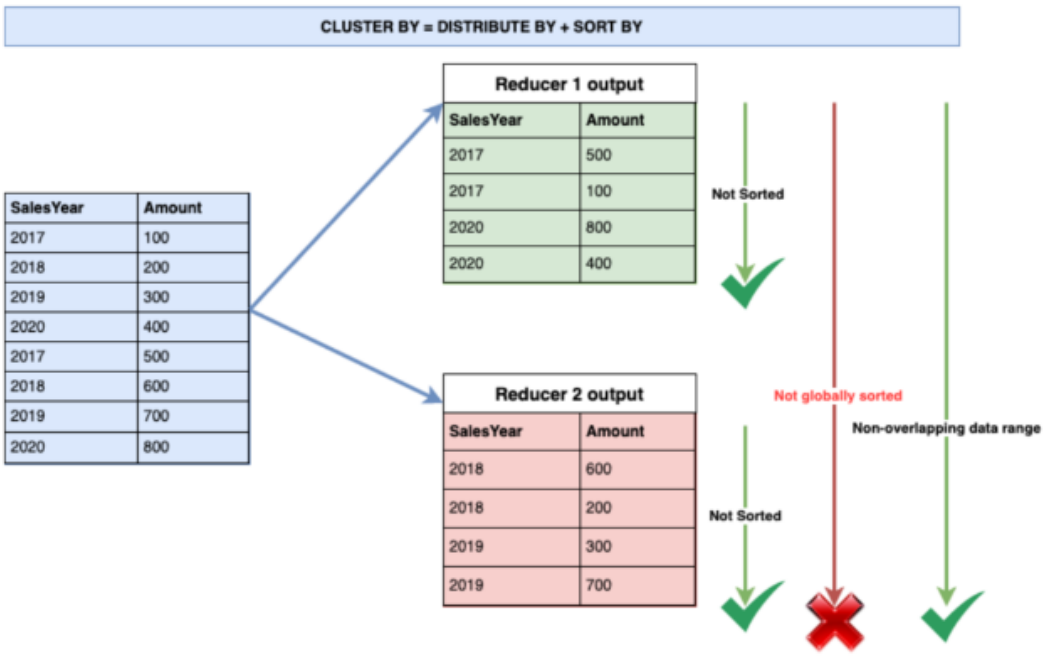
The syntax of the CLUSTER BY clause is as below:

SELECT Col1, Col2,.....ColN FROM TableName CLUSTER BY Col1, Col2, ... ColN

?

```
1    SELECT SalesYear, Amount
2    FROM tbl_Sales
3    CLUSTER BY SalesYear;
```

The output of the above query look like this:



How CLUSTER BY works in hive

Feature-wise comparison of Sort By, Order By, Distribute By, and Cluster By in hive queries

Comparison chart				
	SORT BY	ORDER BY	DISTRIBUTE BY	CLUSTER BY
Supports global sorting?	✗	✓	✗	✗
Sorts data at reducer level?	✓	✓	✗	✓
Non-overlapping data range?	✗	✓	✓	✓
Number of files in the output	One file per reducer	Only one file	One file per reducer	One file per reducer

Comparison of SORT BY vs ORDER BY vs DISTRIBUTE BY vs CLUSTER BY

Q16.Difference between "Internal Table" and "External Table" and Mention when to choose “Internal Table” and “External Table” in Hive?

Ans:- ***Internal or Managed Tables:***

The tables that are created with the Hadoop Hive’s context, is very much similar to tables that are created on any of the RDBMS systems. Each of the tables that get created is associated with a directory configured within the `${HIVE_HOME}/conf/hive-site.xml` in the Hadoop HDFS cluster.

By default on a Linux machine, it is this path **/user/hive/warehouse** in HDFS. If there is a **/user/hive/warehouse/match** created by Hive in HDFS for a match table. All the data for the table is recorded in the same folder as mentioned above and hence such tables are called **INTERNAL** or **MANAGED** tables.

When the data resides in the internal tables, then Hive takes the full responsibility of maintaining the life-cycle of the data and the table in itself. Hence it is evident that the data is removed the moment when the internal tables are dropped.

External Tables:

If there is data that is already existing in the HDFS cluster of Hadoop then an external Hive table is created to describe the data. These tables are called External tables, because they are going to be residing in the path specified by the LOCATION properties instead of the default warehouse directory (as described in the above paragraph).

When the data is stored in the external tables and when it is dropped, the metadata table is deleted but then the data is kept as is. This means that Hive evidently ignores the data that is present residing in the path specified by LOCATION property and is left untouched forever. If you want to delete such data, then use the command to achieve the same:

```
hadoop fs –rmr ‘tablename’
```

Difference between Hive Internal and External Table

Let us now see the difference between both Hive tables. The major differences in the internal and external tables in Hive are:

1. LOAD semantics

The Load semantics varies in both the tables. Let us see the difference in load semantics between the internal table and the external table.

a. Internal Table

When we load data into an internal table, then Hive moves data into the warehouse directory.

For example:
Here in this example, we are creating a table ‘internaldemo’. When we load data into the ‘internaldemo’ table, then Hive moves the data to the warehouse directory.

Now, loading data into the internal table created above.

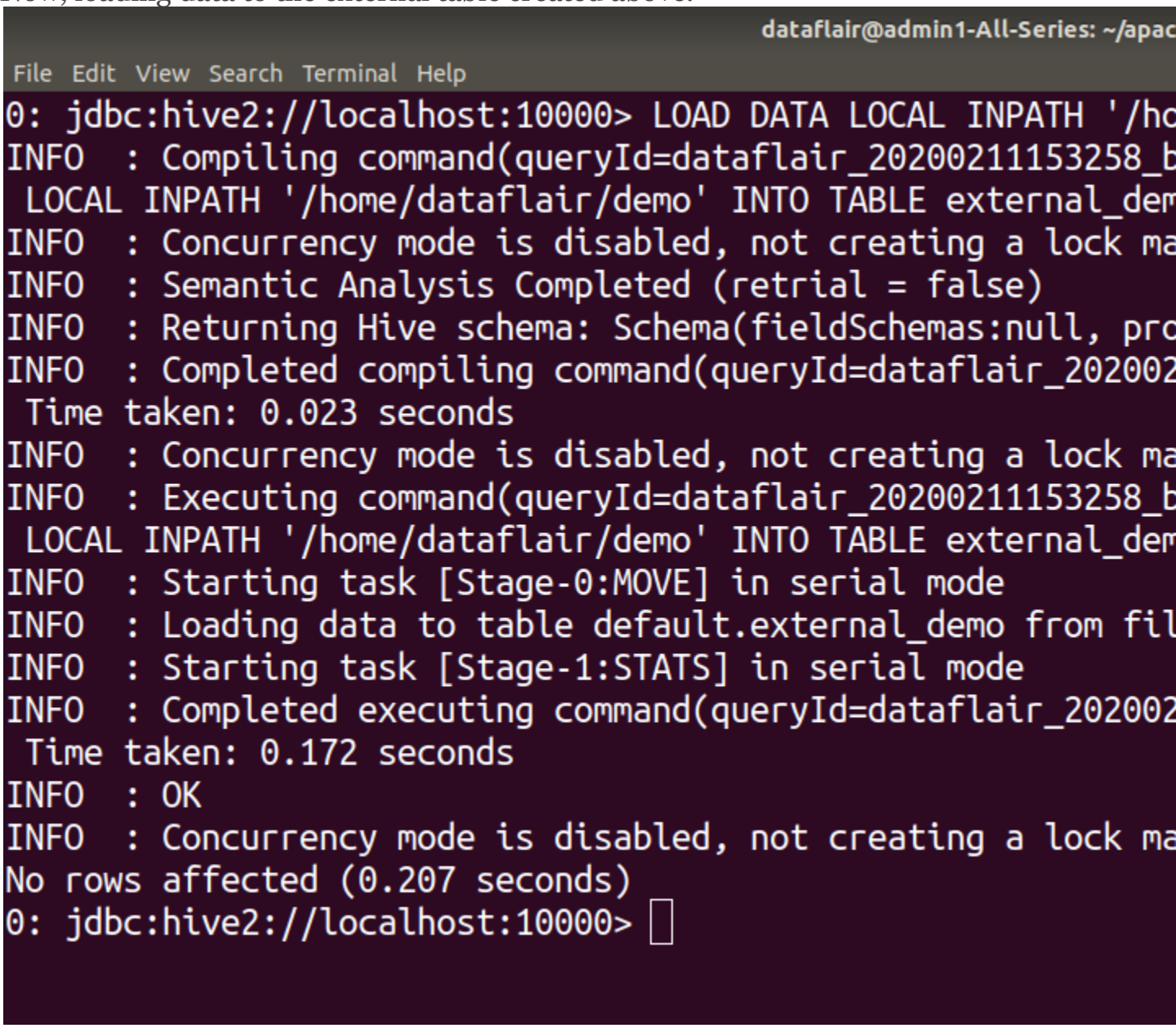
On describing the table, we see that the table data is moved to the Hive warehouse directory.

b. External Table

With the EXTERNAL keyword, Hive knows that it is not managing the table data, so it does not move data to its warehouse directory. Hive does not even check whether the external location at the time it is defined exists or not.

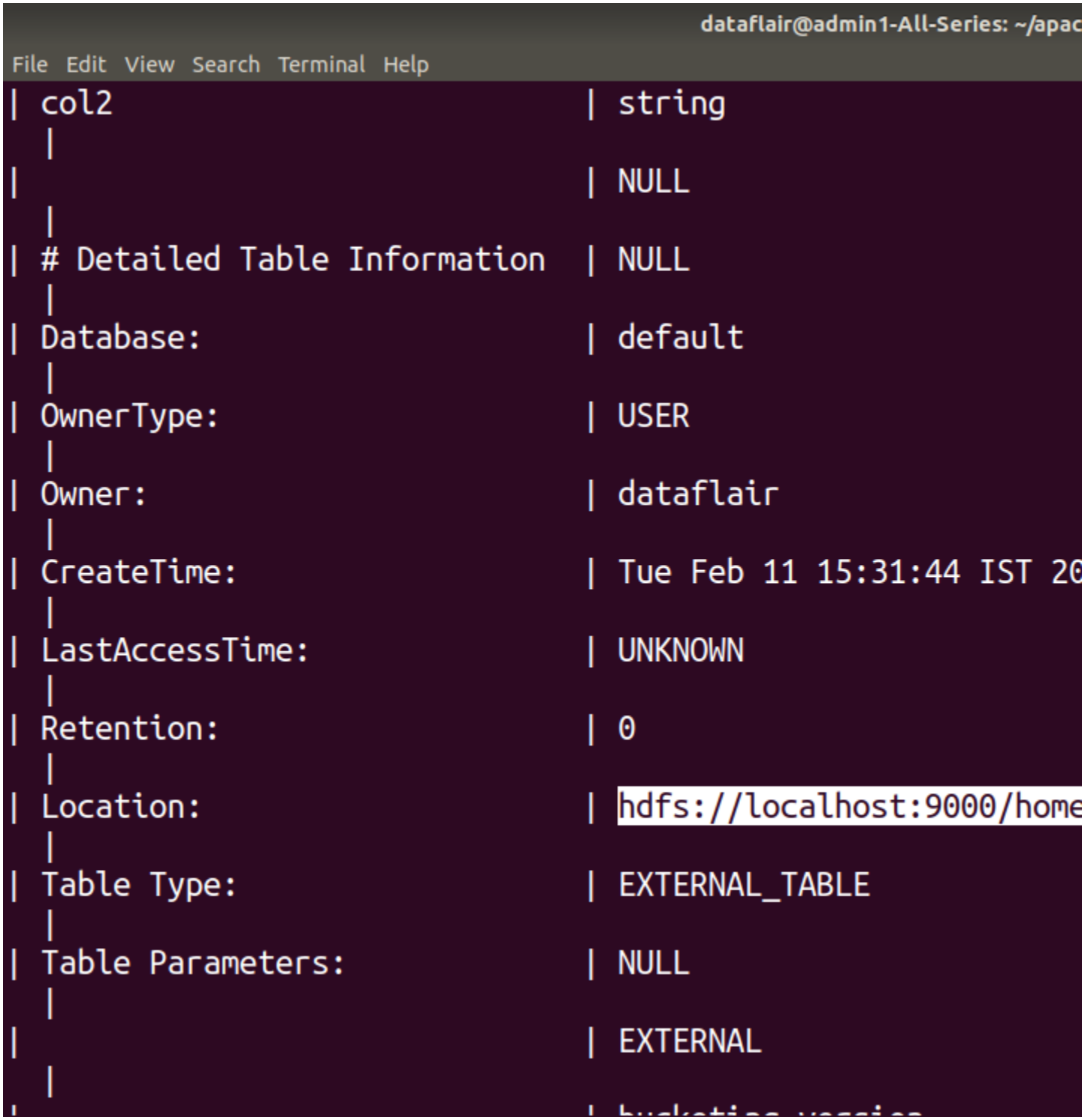
For example:
In this example, we are creating an external table, ‘external_demo’ on the specified location that is ‘/home/dataflair/’. On loading the data into the external table, the Hive does not move table data to its warehouse directory.

Now, loading data to the external table created above.



On browsing the table, we can see that the Hive table data is not moved to the Hive warehouse directory. It is stored in the location specified while creating a

table.


dataflair@admin1-All-Series: ~/apac
File Edit View Search Terminal Help
col2	string
	NULL
# Detailed Table Information	NULL
Database:	default
OwnerType:	USER
Owner:	dataflair
CreateTime:	Tue Feb 11 15:31:44 IST 20
LastAccessTime:	UNKNOWN
Retention:	0
Location:	hdfs://localhost:9000/home
Table Type:	EXTERNAL_TABLE
Table Parameters:	NULL
	EXTERNAL

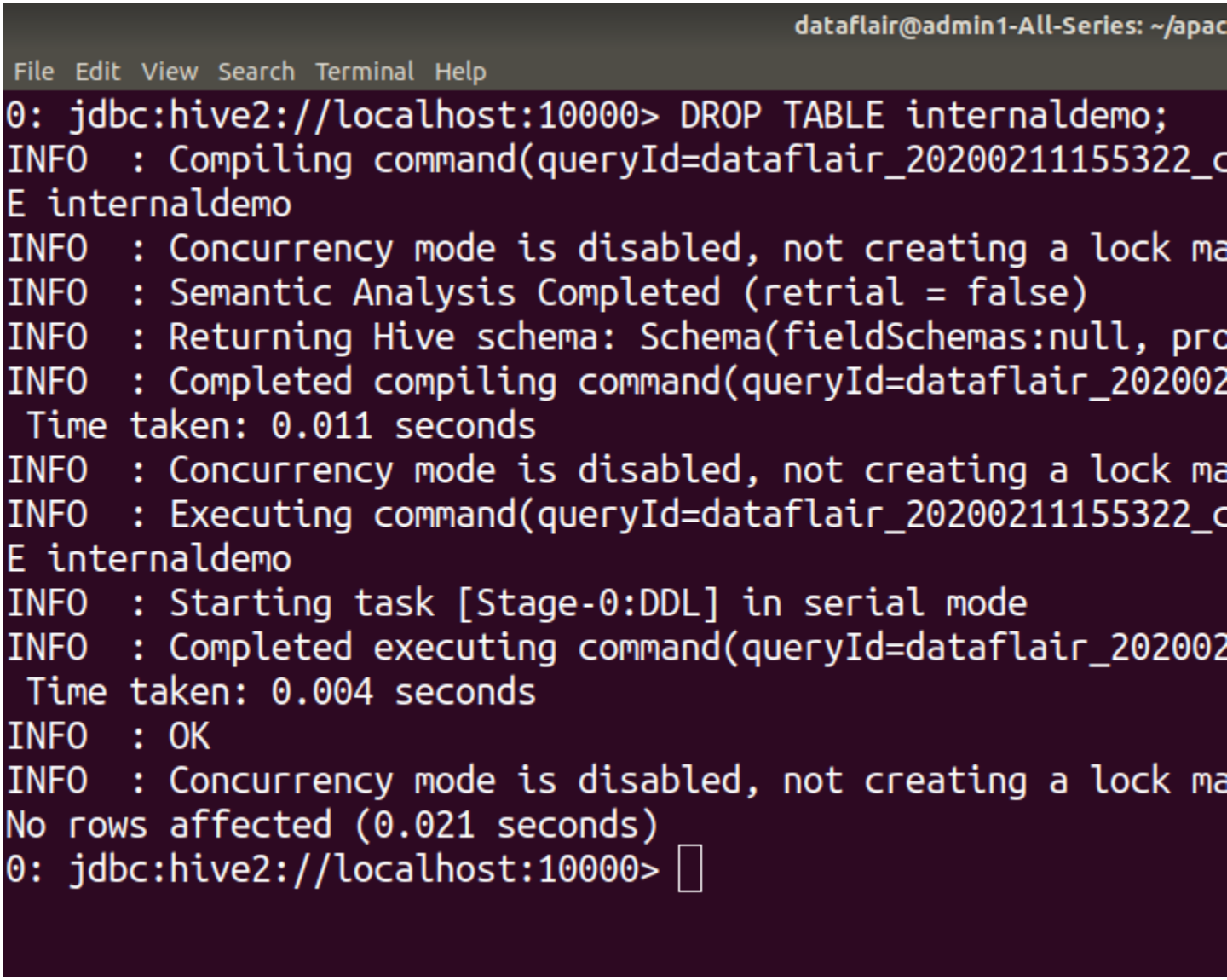
2. DROP semantics

Like load semantics, drop semantics also varies in both tables. Let us see the difference in drop semantics between the internal table and the external table.

a. Internal table

Dropping the internal table will delete the table data, as well as the metadata associated with the table.

For example:
In this example, we are dropping the managed table 'internaldemo'. You can see that it will delete both the table metadata as well as the table data.



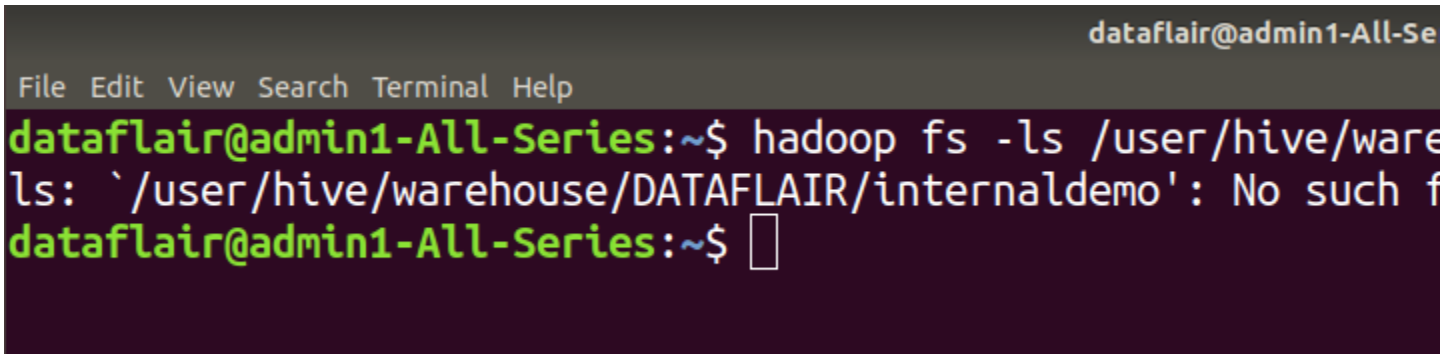
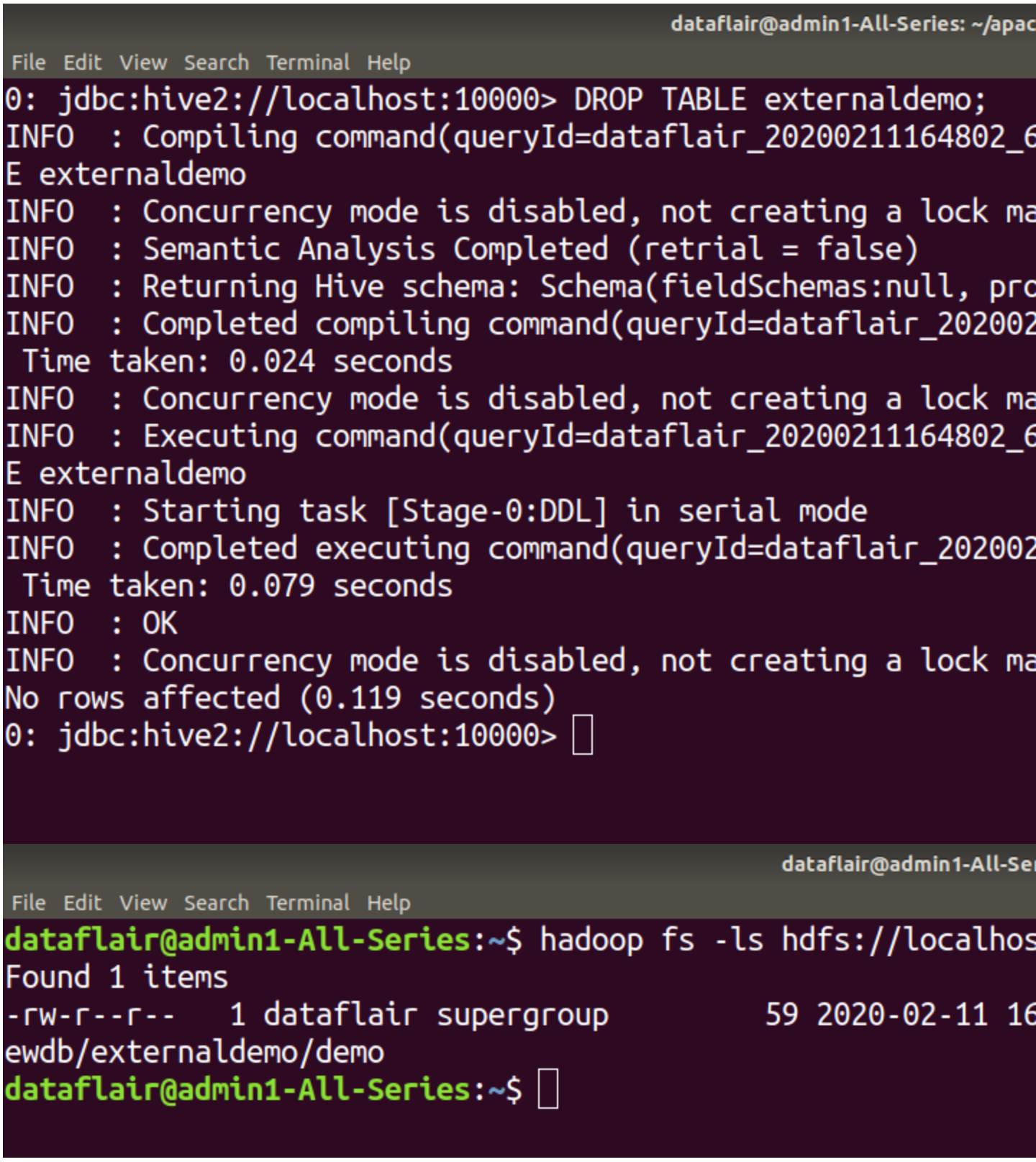


Table data also gets deleted from the HDFS.

b. External table

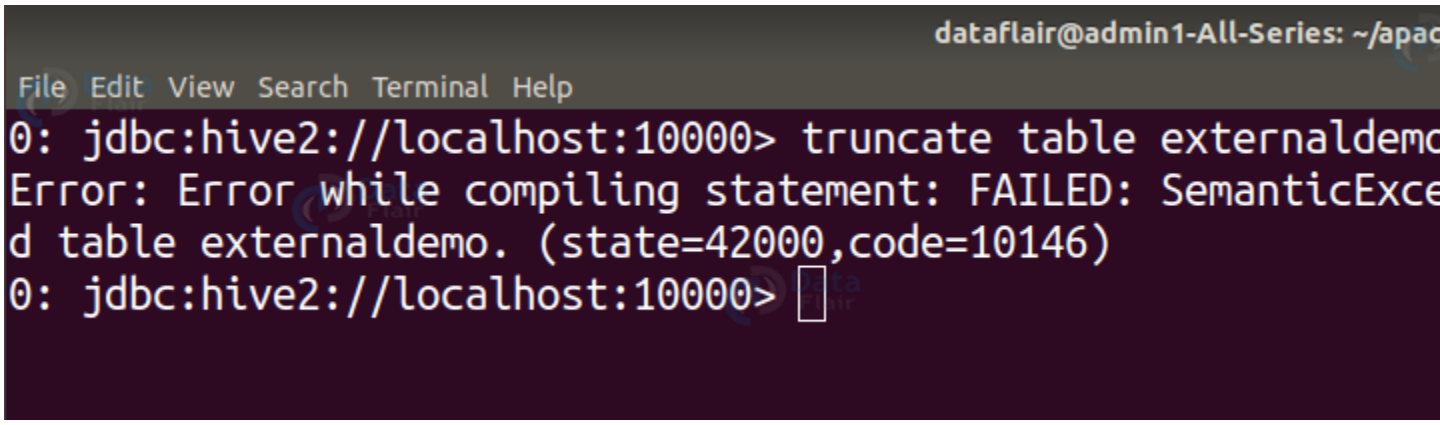
Dropping the external table will delete only table metadata. The table content remains untouched.

For example:
In this example, we can see that on deleting the external table ‘external demo’, the content of the table is still present in the HDFS location.



3. TRUNCATE support

The **TRUNCATE command** only works for the internal table.
For example:
In this example, we are trying to use the truncate command with 'externaldemo' table. We will get an error message that says 'Cannot truncate non-managed table externaldemo'.



4. ACID Support

ACID/transactional works only for the internal table. They do not work for External Table.

5. Query Result Caching

Query Result Caching that saves the results of an executed Hive query for reuse on subsequent queries only works for the internal table.

Thus there are some differences between the Hive internal table and the external table.

Let us now see when to use which Hive table.

When to use the Internal and External table?

1. Hive Internal Table

We can use the internal table in cases:

- When generating temporary tables.
- When required that Hive should manage the lifecycle of the table.

- And when we don’t want table data after deletion.

2. Hive External Table

We can use the external table in cases:

- When we are not creating the table based on the existing table.
- When required to use data outside of Hive. For example, the data files are read and processed by an existing program that does not lock the files.
- When we don’t want to delete the table data completely even after DROP.
- When the data should not be own by Hive.

Use INTERNAL tables when: The data is temporary. You want Hive to completely manage the lifecycle of the table and data.

Use EXTERNAL tables when:

1. The data is also used outside of Hive. ...
2. Data needs to remain in the underlying location even after a DROP TABLE. ...
3. You want to use a custom location such as ASV.
4. Hive should not own data and control settings, dirs, etc., you have another program or process that will do those things.

Q17.Where does the data of a Hive table get stored?

Ans:- Hive stores its database and table metadata in a metastore, which is a database or file backed store that enables easy data abstraction and discovery.

Q18.Is it possible to change the default location of a managed table?

Ans:- Yes, you can do it by using the clause – LOCATION '<hdfs_path>' we can change the default location of a managed table.

Absolutely, by using the LOCATION keyword, we can change the default location of Managed tables while creating the managed table in Hive. However, to do so, the user needs to specify the storage path of the managed table as the value to the LOCATION keyword, that will help to change the default location of a managed table.

Q19.What is a metastore in Hive? What is the default database provided by

Apache Hive for metastore?

Ans:- The Hive metastore is simply a relational database. It stores metadata related to the tables/schemas you create to easily query big data stored in HDFS. When you create a new Hive table, the information related to the schema (column names, data types) is stored in the Hive metastore relational database. Other information like input/output formats, partitions, HDFS locations are all stored in the metastore.

Databases Supported by Hive

Hive supports 5 backend databases which are as follows:

- Derby
- MySQL
- MS SQL Server
- Oracle
- Postgres

Embedded Metastore

In **Hive** by default, metastore service runs in the same JVM as the Hive service. It uses embedded **derby** database stored on the local file system in this mode. Thus both metastore service and hive service runs in the same JVM by using embedded Derby Database.

But, this mode also has limitation that, as only one embedded Derby database can access the database files on disk at any one time, so only one Hive session could be open at a time.

If we try to start the second session it produces an error when it attempts to open a connection to the metastore. So, to allow many services to connect the Metastore, it configures Derby as a network server. This mode is good for unit testing. But it is not good for the practical solutions.

ii. Local Metastore

Hive is the data-warehousing framework, so hive does not prefer single session. To overcome this limitation of Embedded Metastore, for **Local Metastore** was introduced. This mode allows us to have many Hive sessions i.e. many users can use the metastore at the same time.
We can achieve by using any JDBC compliant like MySQL which runs in a separate JVM or different machines than that of the Hive service and metastore service which are running in the same JVM.

This configuration is called as local metastore because metastore service still runs in the same process as the Hive. But it connects to a database running in a separate process, either on the same machine or on a remote machine.

Before starting Apache Hive client, add the JDBC / ODBC driver libraries to the Hive lib folder**.**
MySQL is a popular choice for the standalone metastore. In this case, the ***javax.jdo.option.ConnectionURL*** property is set to ***jdbc:mysql://host/dbname?*** **createDatabaseIfNotExist=true**,

and ***javax.jdo.option.ConnectionDriverName*** is set to ***com.mysql.jdbc.Driver.*** The JDBC driver JAR file for MySQL (Connector/J) must be on Hive's classpath, which is achieved by placing it in Hive's lib directory.

Remote Metastore

Moving further, another metastore configuration called **Remote Metastore**. In this mode, metastore runs on its own separate JVM, not in the Hive service JVM. If other processes want to communicate with the metastore server they can communicate using Thrift Network APIs.

Remote metastore uses mysql database.

Q20.Why does Hive not store metadata information in HDFS?

Ans:- Hive stores metadata information in the metastore using RDBMS instead of HDFS. The reason for choosing RDBMS is to achieve low latency as HDFS read/write operations are time consuming processes.

Q21.What is a partition in Hive? And Why do we perform partitioning in Hive?

Ans:- Partitions provide segregation of the data at the hdfs level, creating sub-directories for each partition. The partitioning in Hive means dividing the table into some parts based on

the values of a particular column like date, course, city or country. The advantage of partitioning is that since the data is stored in slices, the query response time becomes faster.

Q22.What is the difference between dynamic partitioning and static partitioning?

Ans:- There are two types of Partitioning in Apache Hive-

- Static Partitioning
- Dynamic Partitioning

Let's discuss these types of Hive Partitioning one by one-

i. Hive Static Partitioning

- Insert input data files individually into a partition table is Static Partition.
- Usually when loading files (big files) into **Hive tables** static partitions are preferred.
- Static Partition saves your time in loading data compared to dynamic partition.
- You "statically" add a partition in the table and move the file into the partition of the table.
- We can alter the partition in the static partition.
- You can get the partition column value from the filename, day of date etc without reading the whole big file.
- If you want to use the Static partition in the hive you should set property **set hive.mapred.mode = strict** This property set by default in hive-site.xml
- Static partition is in Strict Mode.

- You should use where clause to use limit in the static partition.
- You can perform Static partition on Hive Manage table or external table.

ii. Hive Dynamic Partitioning

- Single insert to partition table is known as a dynamic partition.
- Usually, dynamic partition loads the data from the non-partitioned table.
- Dynamic Partition takes more time in loading data compared to static partition.
- When you have large data stored in a table then the Dynamic partition is suitable.
- If you want to partition a number of columns but you don’t know how many columns then also dynamic partition is suitable.
- Dynamic partition there is no required where clause to use limit.
- we can’t perform alter on the Dynamic partition.
- You can perform dynamic partition on hive external table and managed table.
- If you want to use the Dynamic partition in the hive then the mode is in non-strict mode.
- Here are Hive dynamic partition properties you should allow

Q23.How do you check if a particular partition exists?

Ans:- We can run the HDFS list command to show all partition folders of a table from the Hive data warehouse

location. This option is only helpful if you have all your partitions of the table are at the same location.

Checking hive partition

If yes, you can write a small shell / python script which can call **the** above command to **check if the partition exists**.

Q24.How can you stop a partition from being queried?

Ans:- By using the ENABLE OFFLINE clause with ALTER TABLE statement.

Q25.Why do we need buckets? How Hive distributes the rows into buckets?

Ans:- Buckets are used to improve performance by eliminating table scans when dealing with a large set of data on a Hadoop file system (HDFS). Bucketing in hive is useful when dealing with large datasets that may need to be segregated into clusters for more efficient management and to be able to perform join queries with other large datasets.

Hive distributes the rows into buckets

Records get distributed in buckets based on the hash value from a defined hashing algorithm. The hash value obtained from the algorithm varies with the data type of the bucketed column. For the Int data type, the hash values are also an integer.

Q26.In Hive, how can you enable buckets?

Ans:- The command set hive. enforce. bucketing = true; allows the correct number of reducers and the cluster by column to be automatically selected based on the table. Otherwise, you would need to set the number of reducers to be the same as the number of buckets as in set mapreduce.

So we can enable buckets in Hive by using command set hive. enforce. bucketing = true;

Q27.How does bucketing help in the faster execution of queries?

Ans:- With bucketing in Hive, you can decompose a table data set into smaller parts, making them easier to handle. Bucketing allows you to group similar data types and write them to one single file, which enhances your performance while joining tables or reading data.
Bucketing in hive is the concept of breaking data down into ranges, which are known as buckets, to give extra structure to the data so it may be used for more efficient queries. The range for a bucket is determined by the hash value of one or more columns in the dataset (or Hive metastore table).

Q28.How to optimise Hive Performance? Explain in very detail.

Ans:-There are nine ways to make the most of Hive performance(to optimize hive performance).

1. **Partitioning Tables:**

Hive partitioning is an effective method to improve the query performance on larger tables. Partitioning allows you to store data in separate sub-directories under table location. It dramatically helps the queries which are queried upon the partition key(s). Although the selection of partition key is always a prudent decision, it should always be a low cardinal attribute. For example, if your data is associated with the time dimension, then the date could be a good partition key. Similarly, if data is associated with location, like a country or state, it's a good idea to have hierarchical partitions like country/state.

2. **De-normalizing data:**

   Normalization is a standard process used to model your data tables with certain rules to deal with a redundancy of data and anomalies. In simpler words, if you normalize your data sets, you end up creating multiple relational tables which can be joined at the run time to produce the results. Joins are expensive and complicated operations to perform and are common reasons for performance issues. Because of that, it's a good idea to avoid highly normalized table structures because they require to join queries to derive the desired metrics.

3. **Compress map/reduce output:**

   Compression techniques significantly reduce the intermediate data volume, which internally minimizes the amount of data transfers between mappers and reducers. All this generally occurs over the network. Compression can be applied to the mapper and reducer

output individually. Keep in mind that gzip-compressed files are not splittable. That means this should be applied with caution. Compressed file size should not be larger than a few hundred megabytes. Otherwise, it can potentially lead to an imbalanced job. Other options of compression codec could be snappy, lzo, bzip, etc.

4. **Map join:**

   Map joins are efficient if a table on the other side of a join is small enough to fit in the memory. Hive supports a parameter, **hive.auto.convert.join**, which suggests that Hive tries to map join automatically when it's set to "true." When using this parameter, be sure the auto-convert is enabled in the Hive environment.

   Additionally, it's essential to ensure the bucketing flag is set (**SET hive.enforce.bucketing=true;**) every time before writing data to the bucketed table. To leverage the bucketing in the join operation, we should **SET hive.optimize.bucketmapjoin=true**. This setting hints to Hive to do bucket level join during the map stage join. It also reduces the scan cycles to find a particular key because bucketing ensures that the key is present in a specific bucket.

5. **Input Format Selection:**

   Input formats play a critical role in Hive performance. For example, JSON, the text type of input format, is not the right choice for an extensive production system where data volume is high. These types of readable formats take a lot of space and have some parsing overhead (e.g. JSON parsing). To address these

problems, Hive comes with columnar input formats like RCFile, ORC, etc. Columnar formats allow you to reduce the read operations in analytics queries by allowing each column to be accessed individually. There are some other binary formats like **Avro**, sequence files, Thrift, and ProtoBuf, which can help in various use cases.

6. **Parallel execution:**

   Hadoop can execute MapReduce jobs in parallel, and several queries executed on Hive automatically use this parallelism. However, single, complex **Hive queries** commonly are translated to several MapReduce jobs that are executed by default sequencing. Some of a query's MapReduce stages are often not interdependent and could be executed in parallel. They then can take advantage of spare capacity on a cluster and improve cluster utilization while at the same time reducing the overall query execution time. Hive's configuration to change this behavior is merely switching a single flag **SET hive.exec.parallel=true**.

7. **Vectorization:**

   Vectorization allows Hive to process a batch of rows together instead of processing one row at a time. Each batch consists of a column vector which is usually an array of primitive types. Operations are performed on the entire column vector, which improves the instruction pipelines and cache usage. To enable vectorization, set this configuration parameter **SET hive.vectorized.execution.enabled=true**.

8. **Unit Testing:**

   Merely speaking, unit testing determines whether the smallest testable piece of your code works exactly as you expect. Unit testing gives a couple of benefits, i.e., detecting problems early, making it easier to change and refactor code, being a form of documentation that explains how code works, to name a few.

   In Hive, you can unit test UDFs, SerDes, streaming scripts, Hive queries, and more. To a large extent, it is possible to verify your whole HiveQL query's correctness by running quick local unit tests without even touching a Hadoop cluster. Because executing HiveQL query in the local mode takes literally seconds, compared to minutes, hours, or days if it runs in the Hadoop mode, it certainly saves enormous amounts of development time.

   There are several tools available that help you to test Hive queries. Some of them that you might want to look at **HiveRunner**, **Hive_test,** and **Beetest**.

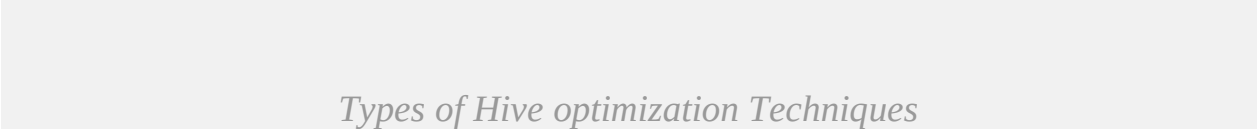
9. **Sampling:**

   Sampling allows users to take a subset of datasets and analyze it without analyzing the entire data set. If a representative sample is used, then a query can return meaningful results and finish quicker and consume fewer compute resources.Hive offers a built-in TABLESAMPLE clause that allows you to sample your tables. TABLESAMPLE can sample at various granularity levels – it can return only subsets of buckets (bucket sampling), or HDFS blocks (block sampling), or

only first N records from each input split. Alternatively, you can implement your own UDF that filters out records according to your sampling algorithm.

Types of Query Optimization Techniques in Hive

Following are the Hive optimization techniques for Hive Performance Tuning, let's discuss them one by one:



Types of Hive optimization Techniques

a. Tez-Execution Engine in Hive

Tez Execution Engine – Hive Optimization Techniques, to increase the Hive performance of our hive query by using our execution engine as Tez. On defining Tez, it is a new application framework built on **Hadoop Yarn**.
That executes complex-directed acyclic graphs of general data processing tasks. However, we can consider it to be a much more flexible and powerful successor to the map-reduce framework.

In addition, to write native YARN applications on Hadoop that bridges the spectrum of interactive and batch workloads Tez offers an API framework to developers. To be more specific, to work with petabytes of data over thousands of nodes it allows those data access applications.

b. Usage of Suitable File Format in Hive

ORCFILE File Formate – Hive Optimization Techniques, if we use appropriate file format on the basis of data. It will drastically increase our query performance.
Basically, for increasing your query performance ORC file format is best suitable. Here, ORC refers to Optimized Row Columnar. That implies we can store data in an optimized way than the other file formats.

To be more specific, ORC reduces the size of the original data up to 75%. Hence, data processing speed also increases. On comparing to Text, Sequence and RC file formats, ORC shows better performance.

Basically, it contains rows data in groups. Such as Stripes along with a file footer. Therefore, we can say when Hive is processing the data ORC format improves the performance.

c. Hive Partitioning

Hive Partition – Hive Optimization Techniques, Hive reads all the data in the directory Without partitioning. Further, it applies the query filters on it. Since all data has to be read this is a slow as well as expensive.

Also, users need to filter the data on specific column values frequently. Although, users need to understand the domain of the data on which they are doing analysis, to apply the partitioning in the Hive.

Basically, by Partitioning all the entries for the various columns of the dataset are segregated and stored in their respective partition. Hence, While we write the query to fetch the values from the table, only the required partitions of the table are queried. Thus it reduces the time taken by the query to yield the result.

d. Bucketing in Hive

Bucketing in Hive – Hive Optimization Techniques, let's suppose a scenario. At times, there is a huge dataset available. However, after partitioning on a particular field or fields, the partitioned file size doesn't match with the actual expectation and remains huge.
Still, we want to manage the partition results into different parts. Thus, to solve this issue of partitioning, Hive offers Bucketing concept. Basically, that allows the user to divide table data sets into more manageable parts.

Hence, to maintain parts that are more manageable we can use Bucketing. Through it, the user can set the size of the manageable parts or Buckets too.

e. Vectorization In Hive

Vectorization In Hive – Hive Optimization Techniques, to improve the performance of operations we use Vectorized query execution. Here operations refer to scans, aggregations, filters, and joins. It happens by performing them in batches of 1024 rows at once instead of single row each time.

However, this feature is introduced in Hive 0.13. It significantly improves query execution time, and is easily enabled with two parameters settings:

set hive.vectorized.execution = true

set hive.vectorized.execution.enabled = true

f. Cost-Based Optimization in Hive (CBO)

Cost-Based Optimization in Hive – Hive Optimization Techniques, before submitting for final execution Hive optimizes each Query's logical and physical execution plan. Although, until now these optimizations are not based on the cost of the query.

However, CBO, performs, further optimizations based on query cost in a recent addition to Hive. That results in potentially different decisions: how to order joins, which type of join to perform, the degree of parallelism and others.

To use CBO, set the following parameters at the beginning of your query:

set hive.cbo.enable=true;

set hive.compute.query.using.stats=true;

set hive.stats.fetch.column.stats=true;

set hive.stats.fetch.partition.stats=true;
Then, prepare the data for CBO by running Hive's "analyze" command to collect various statistics on the tables for which we want to use CBO.

g. Hive Indexing

Hive Index – Hive Optimization Techniques, one of the best ways is Indexing. To increase your query performance indexing will definitely help. Basically, for the original table use of indexing will create a separate called index table which acts as a reference.

As we know, there are many numbers of rows and columns, in a **Hive table**. Basically, it will take a large amount of time if we want to perform queries only on some columns without indexing.

Because queries will be executed on all the columns present in the table.

Moreover, there is no need for the query to scan all the rows in the table while we perform a query on a table that has an index, it turned out as the major advantage of using indexing. Further, it checks the index first and then goes to the particular column and performs the operation.

Hence, maintaining indexes will be easier for Hive query to look into the indexes first and then perform the needed operations within less amount of time. Well, time is the only factor that everyone focuses on, eventually.
This was all about Hive Optimization Techniques Tutorial. Hope you like our explanation of Hive Performance Tuning.

So, this was all in Hive Query Optimization Techniques. Hope you like our explanation.

Q29. What is the use of Hcatalog?

Ans:- HCatalog is a tool that allows you to access Hive metastore tables within Pig, Spark SQL, and/or custom MapReduce applications. HCatalog has a REST interface and command line client that allows you to create tables or do other operations. You then write your applications to access the tables using HCatalog libraries.

Q30. Explain about the different types of join in Hive.

Ans:- Moreover, there are several types of Hive join – HiveQL Select Joins: Hive inner join, hive left outer join, hive right outer join, and hive full outer join. We will also learn Hive Join tables in depth.

Apache Hive Join – HiveQL Select Joins Query

Basically, for combining specific fields from two tables by using values common to each one we use Hive JOIN clause.
In other words, to combine records from two or more tables in the database we use JOIN clause. However, it is more or less similar to SQL JOIN. Also, we use it to combine rows from multiple tables.
Moreover, there are some points we need to observe about Hive Join:

- In Joins, only Equality joins are allowed.
- However, in the same query more than two tables can be joined.
- Basically, to offer more control over ON Clause for which there is no match LEFT, RIGHT, FULL OUTER joins exist in order.
- Also, note that Hive Joins are not Commutative
- Whether they are LEFT or RIGHT joins in Hive, even then Joins are left-associative.

a. Apache Hive Join Syntax

Following is a syntax of Hive Join – HiveQL Select Clause.

```
join_table:
  table_reference JOIN table_factor [join_condition]
  | table_reference {LEFT|RIGHT|FULL} [OUTER] JOIN
```

table_reference
  join_condition
  | table_reference LEFT SEMI JOIN table_reference join_condition
  | table_reference CROSS JOIN table_reference [join_condition]

b. Example of Join in Hive

Example of Hive Join – HiveQL Select Clause.
However, to understand well let's suppose the following table named CUSTOMERS.

Table.1 Hive Join Example

ID	Name	Age	Address
1	Ross	32	Ahmedabad
2	Rachel	25	Delhi
3	Chandler	23	Kota
4	Monika	25	Mumbai
5	Mike	27	Bhopal
6	Phoebe	22	MP
7	Joey	24	Indore

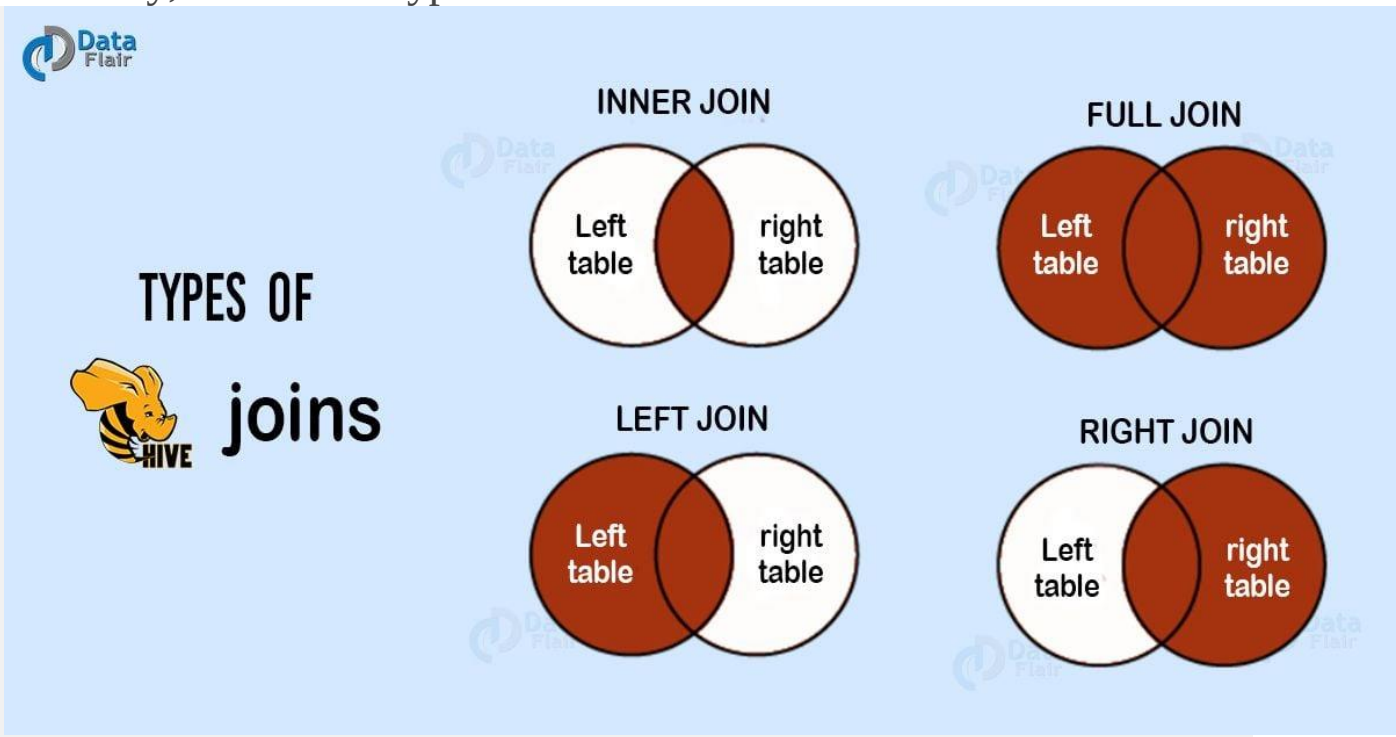
Also, suppose another table ORDERS as follows:

Table.2 – Hive Join Example

OID	Date	Customer_ID
102	2016-10-08 00:00:00	3
100	2016-10-08 00:00:00	3
101	2016-11-20 00:00:00	2

Type of Joins in Hive

Basically, there are 4 types of Hive Join. Such as:



Types of Hive Join | HiveQL Select Clause

1. Inner join in Hive
2. Left Outer Join in Hive
3. Right Outer Join in Hive
4. Full Outer Join in Hive

So, let’s discuss each Hive join in detail.

a. Inner Join

Basically, to combine and retrieve the records from multiple tables we use Hive Join clause. Moreover, in SQL JOIN is as same as OUTER JOIN. Moreover, by using the primary keys and foreign keys of the tables JOIN condition is to be raised.

Furthermore, the below query executes JOIN the CUSTOMER and ORDER tables. Then further retrieves the records:

```
hive> SELECT c.ID, c.NAME, c.AGE, o.AMOUNT
FROM CUSTOMERS c JOIN ORDERS o
ON (c.ID = o.CUSTOMER_ID);
```

Moreover, we get to see the following response, on the successful execution of the query:

Table.3 – Inner Join in Hive

ID	Name	Age	Am
3	Chandler	23	13
3	Chandler	23	15
2	Rachel	25	15
4	Monika	25	20

b. Left Outer Join

On defining HiveQL Left Outer Join, even if there are no matches in the right table it returns all the rows from the left table.

To be more specific, even if the ON clause matches 0 (zero) records in the right table, then also this Hive JOIN still returns a row in the result. Although, it returns with NULL in each column from the right table.

In addition, it returns all the values from the left table. Also, the matched values from the right table, or NULL in case of no matching JOIN predicate.

However, the below query shows LEFT OUTER JOIN between CUSTOMER as well as ORDER tables:

```
hive> SELECT c.ID, c.NAME, o.AMOUNT, o.DATE

FROM CUSTOMERS c

LEFT OUTER JOIN ORDERS o
```

```
ON (c.ID = o.CUSTOMER_ID);
```

Moreover, we get to see the following response, on the successful execution of the HiveQL Select query:

Table.4 – Left Outer Join in Hive

ID	Name	Amount	Date
1	Ross	NULL	NULL
2	Rachel	1560	2016-11-20 00:00:00
3	Chandler	3000	2016-10-08 00:00:00
3	Chandler	1500	2016-10-08 00:00:00
4	Monika	2060	2015-05-20 00:00:00
5	Mike	NULL	NULL
6	Phoebe	NULL	NULL
7	Joey	NULL	NULL

c. Right Outer Join

Basically, even if there are no matches in the left table, HiveQL Right Outer Join returns all the rows from the right table.

To be more specific, even if the ON clause matches 0 (zero) records in the left table, then also this Hive JOIN still returns a row in the result. Although, it returns with NULL in each column from the left table

In addition, it returns all the values from the right table. Also, the matched values from the left table or NULL in case of no matching join predicate.

However, the below query shows RIGHT OUTER JOIN between the CUSTOMER as well as ORDER tables.

notranslate"> hive> SELECT c.ID, c.NAME, o.AMOUNT, o.DATE FROM CUSTOMERS c RIGHT OUTER JOIN ORDERS o ON (c.ID = o.CUSTOMER_ID);

Moreover, we get to see the following response, on the successful execution of the HiveQL Select query:

Table.5 – Right Outer Join in Hive

ID	Name	Amount	Date
3	Chandler	1300	2016-10-08 00:00:0
3	Chandler	1500	2016-10-08 00:00:0
2	Rachel	1560	2016-11-20 00:00:0
4	Monika	2060	2015-05-20 00:00:0

d. Full Outer Join

The major purpose of this HiveQL Full outer Join is it combines the records of both the left and the right outer tables which fulfills the Hive JOIN condition. Moreover, this joined table contains either all the records from both the tables or fills in NULL values for missing matches on either side.

However, the below query shows FULL OUTER JOIN between CUSTOMER as well as ORDER tables:

```
hive> SELECT c.ID, c.NAME, o.AMOUNT, o.DATE

FROM CUSTOMERS c

FULL OUTER JOIN ORDERS o

ON (c.ID = o.CUSTOMER_ID);
```

Moreover, we get to see the following response, on the successful execution of the query:

Table.6 – Full Outer Join in Hive

ID	Name	Amount	Date

1	Ross	NULL	NULL
2	Rachel	1560	2016-11-20 00:00:00
3	Chandler	3000	2016-10-08 00:00:00
3	Chandler	1500	2016-10-08 00:00:00
4	Monika	2060	2015-05-20 00:00:00
5	Mike	NULL	NULL
6	Phoebe	NULL	NULL
7	Joey	NULL	NULL
3	Chandler	3000	2016-10-08 00:00:00
3	Chandler	1500	2016-10-08 00:00:00
2	Rachel	1560	2016-11-20 00:00:00
4	Monika	2060	2015-05-20 00:00:00

This was all about HiveQL Select – Apache Hive Join Tutorial. Hope you like our explanation of Types of Joins in Hive.

Q31.Is it possible to create a Cartesian join between 2 tables, using Hive?

Ans:- Yes it is possible to create a Cartesian join between 2 tables, using Hive.

How to use a cross join in Hive

Cross join, also known as **Cartesian** product, is a way of joining multiple tables in which all the rows or tuples from one table are paired with the rows and tuples from another table. For example, if the left-hand side table has 10 rows and the right-hand side table has 13 rows then the result set after joining the two tables will be 130 rows. That means all the rows from the left-hand side table (having 10 rows) are paired with all the tables from the right-hand side table (having 13 rows).

Q32.Explain the SMB Join in Hive?

Ans:- **SMB Join**- SMB is a join performed on bucket tables that have the same sorted, bucket, and join condition columns. It reads data from both bucket tables and performs common joins (map and reduce triggered) on the bucket tables.

Q33.What is the difference between order by and sort by which one we should use?

Ans:- ORDER BY performs a total ordering of the query result set. This means that all the data is passed through a single reducer, which may take an unacceptably long time to execute for larger data sets.

SORT BY orders the data only within each reducer, thereby performing a local ordering, where each reducer's output will be sorted. You will not achieve a total ordering on the dataset. Better performance is traded for total ordering.

Assume you have a sales table in a company and it has sales entries from salesman around the globe. How do you rank each salesperson by country based on their sales volume in Hive?

Hive support several analytic functions and one of the functions is RANK() and it is designed to do this operation.

```
Hive>SELECT
rep_name, rep_country, sales_volume,
rank() over (PARTITION BY rep_country ORDER BY
sales_volume DESC) as rank
FROM
salesrep;
```

Hive supports SORT BY which sorts the data per reducer. The difference between "order by" and "sort by" is that the former guarantees total order in the output while the latter only guarantees ordering of the rows within a reducer. If there are more than one reducer, "sort by" may give partially ordered final results.

Q34.What is the usefulness of the DISTRIBUTE BY clause in Hive?

Ans:- DISTRIBUTE BY clause is used to distribute the input rows among reducers. It ensures that all rows for the same key columns are going to the same reducer. So, if we need to partition the data on some key column, we can use the DISTRIBUTE BY clause in the hive queries.

Q35. How does data transfer happen from HDFS to Hive?

Ans:- To query data in HDFS in Hive,you apply a schema to the data and then store data in ORC format.Incrementally update the imported data.

Updating imported tables involves importing incremental changes made to the original table using sqoop and then merging changes with the tables imported into Hive.

Q36.Wherever (Different Directory) I run the hive query, it creates a new metastore_db, please explain the reason for it?

Ans:- Basically, it creates the local metastore, while we run the hive in embedded mode. Also, it looks whether metastore already exist or not before creating the metastore. Hence, in configuration file hive-site.xml. Property is "javax.jdo.option.ConnectionURL" with default value "jdbc:derby:;databaseName=metastore_db;create=true" this property is defined. Hence, to change the behavior change the location to the absolute path, thus metastore will be used from that location.

37.What will happen in case you have not issued the command: 'SET hive.enforce.bucketing=true;' before bucketing a table in Hive?

Ans:- The command: 'SET hive.enforce.bucketing=true;' allows one to have the correct number of reducer while using 'CLUSTER BY' clause for bucketing a column. In case it's not done, one may find the number of files that will be generated in the table directory to be not equal to the number of buckets. As an alternative, one may also set the number of reducer equal to the number of buckets by using set mapred.reduce.task = num_bucket.

Q38.Can a table be renamed in Hive?

Ans:- You can rename the table name in the hive. You need to use the alter command. This command allows you to change the table name as shown below.

$ ALTER TABLE name RENAME TO new_name

Q39.Write a query to insert a new column(new_col INT) into a hive table at a position before an existing column (x_col)

Ans:- ALTER TABLE table_name

CHANGE COLUMN new_col

INT BEFORE x_col;

Q40.What is serde operation in HIVE?

Ans:- SerDe means Serializer and Deserializer. Hive uses SerDe and FileFormat to read and write table rows. Main use of SerDe interface is for IO operations. A SerDe allows hive to read the data from the table and write it back to the HDFS in any custom format. If we have unstructured data, then we use RegEx SerDe which will instruct hive how to handle that record.We can also write our own Custom SerDe in any format. Let us see the definition of Serializer and Deserailizer.

Deserailizer

Deserializer is conversion of string or binary data into java object when we any submit any query.

Serailizer

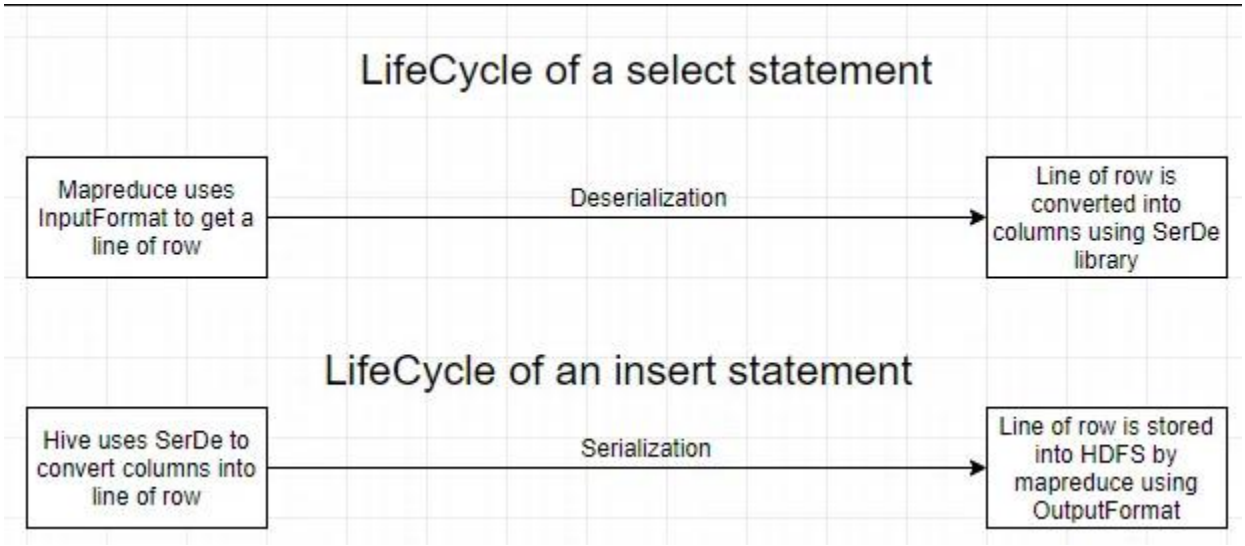
Serializer converts java object into string or binary object. It is used when writing the data such as insert- select statements.Hive currently uses these FileFormat classes to read and write HDFS files:

- TextInputFormat/HiveIgnoreKeyTextOutputFormat: These 2 classes read/write data in plain text file format.
- SequenceFileInputFormat/SequenceFileOutputFormat: These 2 classes read/write data in Hadoop SequenceFile format.

.

Q41.Explain how Hive Deserializes and serialises the data?

Ans:-



Q42.Write the name of the built-in serde in hive.

Ans:- The Hive SerDe library is in org.apache.hadoop.hive.serde2.(The Old SerDe library in org.apache.hadoop.hive.serde is deprecated.

Q43.What is the need of custom Serde?

Ans:- Despite Hive SerDe users want to write a Deserializer in most cases. It is because users just want to read their own data format instead of writing to it
By using the configuration parameter 'regex', the RegexDeserializer will deserialize the data, and possibly a list of column names (see serde2.MetadataTypedColumnsetSerDe).

Some important points about Writing Hive SerDe:

Basically, Hive SerDe, not the DDL, defines the table schema. Since some of the SerDe in Hive are implementations use the DDL for configuration. However, the SerDe can also override that.

Moreover, Column types can be arbitrarily nested arrays, maps, and structures.

However, with CASE/IF or when using complex or nested types the callback design of ObjectInspector allows lazy deserialization.

Q44.Can you write the name of a complex data type(collection data types) in Hive?

Ans:- In addition to primitive data types, Hive also supports a few complex data types:Struct, MAP, and Array.Complex data types are also known as collection data types.

Q45.Can hive queries be executed from script files? How?

Ans:- Yes.

Hive>source /path/to/file/file_with_query.hql

Q46.What are the default record and field delimiter used for hive text files?

Ans:- The default record delimiter is − \n And the filed delimiters are − \001,\002,\003

Q47.How do you list all databases in Hive whose name starts with s?

Ans:- SHOW (DATABASES|SCHEMAS) [LIKE identifier_with_wildcards];

SHOW DATABASES LIKE ‘s%’;

Q48.What is the difference between LIKE and RLIKE operators in Hive?

Ans:- LIKE is an operator similar to LIKE in SQL. We use LIKE to search for string with similar text.

RLIKE (Right-Like) is a special function in Hive where if any substring of A matches with B then it evaluates to true. It also obeys Java regular expression pattern.

Q49.How to change the column data type in Hive?

Ans:- By using this command below one can change the column data type: ALTER TABLE table_name CHANGE column_name column_name new_datatype;

Q50.How will you convert the string ’51.2’ to a float value in the particular column?

Ans:- Select cast (’51.2’ as float)

Q51.What will be the result when you cast ‘abc’ (string) as INT?

Ans:- It prints NULL

52.What does the following query do?

a. INSERT OVERWRITE TABLE employees

b. PARTITION (country, state)

c. SELECT ..., se.cnty, se.st

d. FROM staged_employees se;

Ans:- a. inserts or overwrites values in table ‘employees’

b. it is used for a windows function where partitions are required for country and state attributes.

c. selects ‘cnty’ column from ‘staged_employees’ table declared as ‘se’

Q53.Write a query where you can overwrite data in a new table from the existing table.

Ans:- create table B like A;

INSERT OVERWRITE TABLE B SELECT A.value FROM A;

Q54.What is the maximum size of a string data type supported by Hive? Explain how Hive supports binary formats.

Ans:- By default,the columns metadata for Hive does not specify a maximum data length for STRING columns.The driver has the parameter DefaultStringColumnLength, default is 255 maximum value.

Hive supports the text file format by default, and it also supports the binary format sequence files,ORC files, Avro data files, and Parquet files.Sequence file: It is a splittable,compressible,and roworiented file with a general binary format.

Q55. What File Formats and Applications Does Hive Support?

Ans:- Hive and Impala tables in HDFS can be created using text files.Sequence files,ORC files,Avro data files, and Parquet file formats.Data serialization is a way of representing data in memory as a series of bytes.Avro is an efficient data serialization framework and is widely supported throughout Hadoop and its ecosystem.

Q56.How do ORC format tables help Hive to enhance its performance?

Ans:- Using the ORC format leads to a reduction in the size of the data stored, as this file format has high compression ratios. As the data size is reduced, the time to read and write the data is also reduced.

Q57.How can Hive avoid mapreduce while processing the query?

Ans:- When you perform a “select * from <tablename>”, Hive fetches the whole data from file as a FetchTask rather than a mapreduce task which just dumps the data as it is without doing anything on it.This is similar to “hadoop dfs –text <filename>”

However, while using “select <column> from <tablename>”, Hive requires a map-reduce job since it needs to extract the ‘column’ from each row by parsing it from the file it loads.

Q58.What is view and indexing in hive?

Ans:- Views are generated based on user requirements. You can save any result set data as a view. The usage of view in Hive is same as that of the view in SQL. It is a standard RDBMS concept. We can execute all DML operations on a view.

Creating a View

You can create a view at the time of executing a SELECT statement. The syntax is as follows:

```
CREATE VIEW [IF NOT EXISTS] view_name [(column_name [COMMENT
column_comment], ...) ]
[COMMENT table_comment]
AS SELECT ...
```

Example

Let us take an example for view. Assume employee table as given below, with the fields Id, Name, Salary, Designation, and Dept. Generate a query to retrieve the employee details who earn a salary of more than Rs 30000. We store the result in a view named **emp_30000.**

```
+------+--------------+-------------+-------------------+--------+
| ID   | Name         | Salary      | Designation       | Dept   |
```

```
+------+--------------+-------------+-------------------+--------+
|1201  | Gopal        | 45000       | Technical manager | TP     |
|1202  | Manisha      | 45000       | Proofreader       | PR     |
|1203  | Masthanvali  | 40000       | Technical writer  | TP     |
|1204  | Krian        | 40000       | Hr Admin          | HR     |
|1205  | Kranthi      | 30000       | Op Admin          | Admin  |
+------+--------------+-------------+-------------------+--------+
```

The following query retrieves the employee details using the above scenario:

```
hive> CREATE VIEW emp_30000 AS
SELECT * FROM employee
WHERE salary>30000;
```

Dropping a View

Use the following syntax to drop a view:

```
DROP VIEW view_name
```

The following query drops a view named as emp_30000:

```
hive> DROP VIEW emp_30000;
```

Creating an Index

An Index is nothing but a pointer on a particular column of a table. Creating an index means creating a pointer on a particular column of a table. Its syntax is as follows:

```
CREATE INDEX index_name
ON TABLE base_table_name (col_name, ...)
AS 'index.handler.class.name'
[WITH DEFERRED REBUILD]
[IDXPROPERTIES (property_name=property_value, ...)]
[IN TABLE index_table_name]
[PARTITIONED BY (col_name, ...)]
[
   [ ROW FORMAT ...] STORED AS ...
   | STORED BY ...
]
[LOCATION hdfs_path]
```

[TBLPROPERTIES (...)]

Example

Let us take an example for index. Use the same employee table that we have used earlier with the fields Id, Name, Salary, Designation, and Dept. Create an index named index_salary on the salary column of the employee table.

The following query creates an index:

```
hive> CREATE INDEX inedx_salary ON TABLE employee(salary)
AS 'org.apache.hadoop.hive.ql.index.compact.CompactIndexHandler';
```

It is a pointer to the salary column. If the column is modified, the changes are stored using an index value.

Dropping an Index

The following syntax is used to drop an index:

```
DROP INDEX <index_name> ON <table_name>
```

The following query drops an index named index_salary:

```
hive> DROP INDEX index_salary ON employee;
```

Q59.Can the name of a view be the same as the name of a hive table?

Ans:- No, we cannot use same name for a table and view in Hive.So we have to select a unique name for a view in Hive.

Q60.What types of costs are associated in creating indexes on hive tables?

Ans:- There is a processing cost in arranging the values of the column on which index is created since Indexes occupies space.

Q61.Give the command to see the indexes on a table.

Ans:- SHOW INDEX ON table_name;

Q62. Explain the process to access subdirectories recursively in Hive queries.

Ans:- We can use following commands in Hive to recursively access sub-directories:

Hive>Set mapred.input.dir.recursive=true; hive>

Set hive.mapred.supports.subdirectories=true;

Once above options are set to true, Hive will recursively access sub-directories of a directory in MapReduce.

Q63.If you run a select * query in Hive, why doesn't it run MapReduce?

Ans:- Query you are reading every column,no filtering is required.Hence no Map phase

select *FROM TABLE –Q1 And select col1,col2 FROM TABLE –Q12

To understand the reason, first we need to know what map and reduce phases mean:

Map: Basically a filter which filters and organizes data in sorted order.For e.g. It will filter col1_name,col2_name from a row in the second query.query you are reading every column, no filtering is required. Hence no Map phase.

Reduce: Reduce is just summary operation data across the rows.for e.g. sum of a column.

In both the queries you don’t need any summary data.Hence no reducer.

Q64.What are the uses of Hive Explode?

Ans:- Explode is an essential function in Hive used to manipulate complex data types such as arrays, maps, and structs. This functions help

transform data into a format more suitable for analysis and processing. Let's look at this function and how it can be used in Hive queries.

Explode is a built-in Hive function that takes an array or a map as input and returns a new row for each element in the array or key-value pair in the map. In other words, it "explodes" the input into multiple rows. Let's look at an example:

SELECT explode(array(1, 2, 3));

This query will return three rows, each containing a single integer value:

1

2

3

Similarly, if we have a map with key-value pairs, we can use explode to split it into separate rows:

SELECT explode(map('a', 1, 'b', 2, 'c', 3));

This query will return three rows, each containing a single key-value pair:

a 1

b 2

c 3

Q65. What is the available mechanism for connecting applications when we run Hive as a server?

Ans:- There are following ways by which you can connect with the Hive Server:

1. Thrift Client: Using Thrift, we can call Hive commands from various programming languages, such as C++, PHP, Java, Python, and Ruby.

2. **JDBC Driver**: JDBC Driver enables accessing data with JDBC support, by translating calls from an application into SQL and passing the SQL queries to the Hive engine.
3. **ODBC Driver**: It implements the ODBC API standard for the Hive DBMS, enabling ODBC-compliant applications to interact seamlessly with Hive.

Q66.Can the default location of a managed table be changed in Hive?

Ans:- Yes, you can do it by using the clause – LOCATION '<hdfs_path>' we can change the default location of a managed table.Absolutely, by using the LOCATION keyword, we can change the default location of Managed tables while creating the managed table in Hive. However, to do so, the user needs to specify the storage path of the managed table as the value to the LOCATION keyword, that will help to change the default location of a managed table.

Using the LOCATION keyword, we can change the default location of Managed tables while creating the managed table in Hive.

Q67.What is the Hive ObjectInspector function?

Ans:- Hive uses ObjectInspector to analyze the internal structure of the row object and also the structure of the individual columns.

ObjectInspector provides a uniform way to access complex objects that can be stored in multiple formats in the memory, including: Instance of a Java class (Thrift or native Java).

Q68.What is UDF in Hive?

Ans:- UDF stands for User Defined Functions.

Hive is a powerful tool that allows us to provision sql queries on top of stored data for basic querying and/or analysis.And on top of an already rich set of built-in functions,it allows us to extend its functionality by writing custom functions of our own.

Types of UDFs available in Hive:

1. UDF (User Defined Function): Operates on a single row and generates a single output.

2. UDAF (User Defined Aggregate Function): Operates on a group of rows and generates a single output. Generally, used with an accompanying ‘group by’ clause.
3. UDTF (User Defined Tabular Function): Operates on a single row and generates multiple rows as an output.

Q69.Write a query to extract data from hdfs to hive.

Ans:- load data inpath ‘<hdfs location>’ into table <hive table name>;

Q70.What is TextInputFormat and SequenceFileInputFormat in hive.

Ans:- TEXTFILE format is a famous input/output format used in **Hadoop**. In Hive if we define a table as TEXTFILE it can load data of from CSV (Comma Separated Values), delimited by Tabs, Spaces, and JSON data. This means fields in each record should be separated by comma or space or tab or it may be JSON(JavaScript Object Notation) data.

By default, if we use TEXTFILE format then each line is considered as a record.

We can create a TEXTFILE format in Hive as follows:

```
create table table_name (schema of the table) row format
delimited fields terminated by ',' | stored as TEXTFILE.
```

At the end, we need to specify the type of **file format**. If we do not specify anything it will consider the file format as TEXTFILE format.

The TEXTFILE input and TEXTFILE output format are present in the Hadoop package as shown below:

```
org.apache.hadoop.mapred.TextInputFormat
```

```
org.apache.hadoop.mapred.TextOutputFormat
```

Let us see one example in Hive about how to create TEXTFILE table format, how to load data into TEXTFILE format and perform one basic select operation in Hive.

Creating TEXTFILE

```
create table olympic(athelete STRING,age INT,country
STRING,year STRING,closing STRING,sport STRING,gold
INT,silver INT,bronze INT,total INT) row format
delimited fields terminated by '\t' stored as textfile;
```

Here we are creating a table with name “olympic" and the schema of the table is as specified above. The data inside the above input file is delimited by tab space. As explained earlier the file format is specified as TEXTFILE at the end. The schema of the table created above can be checked using describe olympic;

We can load data into the created table as follows:

load data local inpath 'path of your file' into table olympic;

We have successfully loaded our input file data into our table which is of TEXTFILE format.

Now we will perform one basic SELECT operation on the data as shown below:

```
select athelete from olympic;
```

SEQUENCEFILE

We know that Hadoop's performance is drawn out when we work with a small number of files with big size rather than a large number of files with small size. If the size of a file is smaller than the typical block size in Hadoop, we consider it as a small file. Due to this, a number of metadata increases which will become an overhead to

the NameNode. To solve this problem sequence files are introduced in Hadoop. Sequence files act as a container to store the small files.

Sequence files are flat files consisting of binary key-value pairs. When Hive converts queries to MapReduce jobs, it decides on the appropriate key-value pairs to be used for a given record. Sequence files are in the binary format which can be split and the main use of these files is to club two or more smaller files and make them as a one sequence file.

In Hive we can create a sequence file by specifying STORED AS SEQUENCEFILE in the end of a CREATE TABLE statement.

There are three types of sequence files:

• Uncompressed key/value records.

• Record compressed key/value records - only 'values' are compressed here

• Block compressed key/value records - both keys and values are collected in 'blocks' separately and compressed. The size of the 'block' is configurable.

Hive has its own SEQUENCEFILE reader and SEQUENCEFILE writer libraries for reading and writing through sequence files.

In Hive we can create a sequence file format as follows:

```
create table table_name (schema of the table) row format delimited fileds terminated by ',' | stored as SEQUENCEFILE
```

Hive uses the SEQUENCEFILE input and output formats from the following packages:

```
org.apache.hadoop.mapred.SequenceFileInputFormat
```

```
org.apache.hadoop.hive.ql.io.HiveSequenceFileOutputFormat
```

Creating SEQUENCEFILE

```
create table olympic_sequencefile(athelete STRING,age INT,country STRING,year STRING,closing STRING,sport STRING,gold INT,silver INT,bronze INT,total INT) row format delimited fields terminated by '\t' stored as sequencefile
```

Here we are creating a table with name olympic_sequencefile and the schema of the table is as specified above and the data inside my input file is delimited by tab space. At the end the file format is specified as SEQUENCEFILE format. You can check the schema of your created table using:

```
describe olympic_sequencefile;
```

Now to load data into this table is somewhat different from loading into the table created using TEXTFILE format. You need to insert the data from another table because this SEQUENCEFILE format is the binary format. It compresses the data and then stores it into the table. If you want to load directly as in TEXTFILE format that is not possible because we cannot insert the compressed files into tables.

So to load the data into SEQUENCEFILE we need to use the following approach:

```
INSERT OVERWRITE TABLE olympic_sequencefile

SELECT * FROM olympic;
```

We have already created table by name olympic which is of TEXTFILE format and then we are writing the contents of the olympic table into olympic_sequencefile table.

Thus we have successfully loaded data into the SEQUENCEFILE.

Now let us perform the same basic SELECT operation which we have performed on the TEXTFILE format, on SEQUENCEFILE format also.

```
select athelete from olympic_sequencefile;
```

Q71.How can you prevent a large job from running for a long time in a hive?

Ans:- This can be achieved by setting the MapReduce jobs to execute in strict mode.

set hive.mapred.mode=strict;

The strict mode ensures that the queries on partitioned tables cannot execute without defining a WHERE clause.

Q72.When do we use explode in Hive?

Ans:- We need to use Explode in Hive to convert complex data types into desired table formats.explode UDTF basically emits all the elements in an array into multiple rows.

Q73.Can Hive process any type of data formats? Why? Explain in very detail

Ans:- HiveQL handles structured data only.

By default, Hive has derby database to store the data in it. We can configure Hive with MYSQL database.As mentioned, HiveQL can handle only structured data.Data is eventually stored in files.

Hive supports several file formats:

- TextFile
- SequenceFile
- RCFile
- Avro Files
- ORC files
- Parquet
- Custom INPUTFORMAT and OUTPUTFORMAT

The hive.default.fileformat configuration parameter determines the format to use if it is not specified in a CREATE TABLE or ALTER TABLE statement.Text file is the parameter's default value.

Hive Text File Format

Hive Text file format is a default storage format. You can use the text format to interchange the data with other client application. The text file format is very common most of the applications. Data is stored in lines, with each line being a record. Each lines are terminated by a newline character (\n).
The text format is simple plane file format. You can use the compression (*BZIP2*) on the text file to reduce the storage spaces.
Create a TEXT file by add storage option as **'STORED AS TEXTFILE'** at the end of a Hive CREATE TABLE command.
Hive Text File Format Examples
Below is the Hive CREATE TABLE command with storage format specification:

```
Create table textfile_table

(column_specs)

stored as textfile;
```

Hive Sequence File Format

Sequence files are Hadoop flat files which stores values in binary key-value pairs. The sequence files are in binary format and these files are able to split. The main advantages of using sequence file is to merge two or more files into one file.
Create a sequence file by add storage option as **'STORED AS SEQUENCEFILE'** at the end of a Hive CREATE TABLE command.

Hive Sequence File Format Example
Below is the Hive CREATE TABLE command with storage format specification:

```
Create table sequencefile_table

(column_specs)

stored as sequencefile;
```

Hive RC File Format

RCFile is row columnar file format. This is another form of Hive file format which offers high row level compression rates. If you have requirement to perform multiple rows at a time then you can use RCFile format.
The RCFile are very much similar to the sequence file format. This file format also stores the data as key-value pairs.

Create RCFile by specifying **'STORED AS RCFILE'** option at the end of a CREATE TABLE Command:
Hive RC File Format Example
Below is the Hive CREATE TABLE command with storage format specification:

```
Create table RCfile_table

(column_specs)

stored as rcfile;
```

Hive AVRO File Format

AVRO is open source project that provides data serialization and data exchange services for Hadoop. You can exchange data between Hadoop ecosystem and program written in any programming

languages. Avro is one of the popular file format in Big Data Hadoop based applications.
Create AVRO file by specifying **'STORED AS AVRO'** option at the end of a CREATE TABLE Command.

Hive AVRO File Format Example

Below is the Hive CREATE TABLE command with storage format specification:

```
Create table avro_table

(column_specs)

stored as avro;
```

Hive ORC File Format

The **ORC file** stands for Optimized Row Columnar file format. The ORC file format provides a highly efficient way to store data in Hive table. This file system was actually designed to overcome limitations of the other Hive file formats. The Use of ORC files improves performance when Hive is reading, writing, and processing data from large tables.
Create ORC file by specifying **'STORED AS ORC'** option at the end of a CREATE TABLE Command.

Hive ORC File Format Examples

Below is the Hive CREATE TABLE command with storage format specification:

```
Create table orc_table

(column_specs)

stored as orc;
```

Hive Parquet File Format

Parquet is a column-oriented binary file format. The parquet is highly efficient for the types of large-scale queries. Parquet is especially good for queries scanning particular columns within a particular table. The Parquet table uses compression Snappy, gzip; currently Snappy by default.
Create Parquet file by specifying **'STORED AS PARQUET'** option at the end of a CREATE TABLE Command.

Hive Parquet File Format Example

Below is the Hive CREATE TABLE command with storage format specification:

```
Create table parquet_table

(column_specs)

stored as parquet;
```

Q74.Whenever we run a Hive query, a new metastore_db is created. Why?

Ans:- Whenever you run the hive in embedded mode, it creates the local metastore.And before creating the metastore it looks whether metastore already exist or not.This property is defined in configuration file hive-site.xml. Property is "javax.jdo.option.ConnectionURL" with default value "jdbc:derby:;databaseName=metastore_db;create=true".So to change the behavior change the location to absolute path, so metastore will be used from that location.

Q75.Can we change the data type of a column in a hive table? Write a complete query.

Ans:- Yes By using this command below one can change the column data type:

ALTER TABLE table_name CHANGE column_name column_name new_datatype;

Q76.While loading data into a hive table using the LOAD DATA clause, how do you specify it is a hdfs file and not a local file ?

Ans:- Hive provides us the functionality to load pre-created table entities either from our local file system or from HDFS.The LOAD DATA statement is used to load data into the hive table.

Syntax:

LOAD DATA [LOCAL] INPATH “[OVERWRITE] INTO TABLE;

Note: The LOCAL Switch specifies that the data we are loading is available in our Local File System.If the LOCAL switch is not used, the hive will consider the

location as an HDFS path location.The OVERWRITE switch allows us to overwrite the table data.

Q77.What is the precedence order in Hive configuration?

Ans:- We are using a precedence hierarchy for setting properties:

1. The SET command in Hive
2. The command-line –hiveconf option
3. Hive-site.XML
4. Hive-default.xml
5. Hadoop-site.xml
6. Hadoop-default.xml

Q78.Which interface is used for accessing the Hive metastore?

Ans:- WebHCat API web interface can be used for Hive commands. It is a REST API that allows applications to make HTTP requests to access the Hive metastore (HCatalog DDL).

Q79.Is it possible to compress json in the Hive external table ?

Ans:- Yes, you need to gzip your files and put them as is (*.gz) into the table location.

Q80.What is the difference between local and remote metastores?

Ans:- A local metastore runs on the same JVM (Java Virtual Machine) in which the Hive service is running. It connects to a database running in a separate JVM that can either be on the same machine or a remote machine.

In a remote metastore, the service runs on its JVM differently from the JVM on which the Hive service runs. Other processes can communicate with the server containing the metastore using Thrift Network APIs. There can be more than one metastore server to provide more availability.

Q81.What is the purpose of archiving tables in Hive?

Ans:- Hadoop archives are used as an approach to reduce the number of files in partitions. Apache Hive has built-in support for converting files on existing partitions to a Hadoop Archive. In this manner, a partition that would have once consisted of hundreds of files can be allowed to occupy just around 3 files. This number can vary depending on the settings. The trade-off in such a case is that the queries may end up having a higher latency, caused by the additional overhead of reading from Hadoop Archive. Archiving does not compress the files, but works similar to the Linux ‘tar’ command.

Q82.What is DBPROPERTY in Hive?

Ans:- The DB properties are nothing but mentioning the details about the database created by the user. Suppose

the name of the user, the type of the database and the tables it has, the date on which the database is created etc. This makes the other user easy the recognize the database and use it according to the requirement.

Q83.Differentiate between local mode and MapReduce mode in Hive.

Ans:- Local mode is actually a local simulation of MapReduce in Hadoop's LocalJobRunner class.

MapReduce mode (also known as Hadoop mode): Pig is executed on the Hadoop cluster. In this case, the Pig Script gets converted into a series of MapReduce jobs that are then run on the Hadoop cluster.