

UNIT-4

Hadoop Related Tools (PIG & HIVE)

⇒ Introduction to PIG.

- Apache Pig is a platform for data analysis.
- It is an alternative to Map Reduce Programming.
- Pig was developed by Yahoo

→ Key features of Pig :-

- 1) It provides an Engine for executing dataflows
- 2) It provides a language called "pig Latin".
- 3) Pig Latin contains operators for many of traditional data operations such as join, filter, sort etc.
- 4) It allows users to develop their own functions (user defined functions).

→ Anatomy of PIG :-

The main components of Pig are as follows :

- 1) Data flow language (Pig Latin Script)
- 2) Interactive shell where you can type the Pig Latin statements (Grunt)
- 3) Pig interpreter and execution engine.

→ PIG on Hadoop

- pig uses both Hadoop Distributed file system and MapReduce programming.
- By default pig reads input from HDFS
- pig also supports the following.
 - 1) HDFS commands
 - 2) UNIX ^{shell} commands
 - 3) Relational operators
 - 4) positional parameters
 - 5) Common mathematical function
 - 6) custom functions
 - 7) Complex data structures.

⇒ Pig Latin overview.

* Pig Latin statements

- Pig Latin statements are basic constructs to process data using Pig
- Pig Latin statement is an operator
- An operator in Pig Latin takes relation as input and yields another relation as output.
- It includes schemas & expressions to process data.
- Pig Latin statements should end with a semi colon.

Pig Latin statements are generally ordered as follows:

- 1) LOAD → To read data from file system
- 2) Series of statements to perform transformation
- 3) DUMP or STORE → to display / store result.

* Pig Latin : Keywords

→ keywords are reserved. It cannot be used to name things.

* Pig Latin : Identifiers

→ Identifiers are named assigned to fields or other data structures.

→ It should begin with a letter and should be followed only by letters, numbers and underscores.

* Pig Latin : comments

→ Single line comments that begin with " -- ".

→ Multiline comments that begin with /* and end with */.

* Pig Latin : Case Sensitivity

→ Keywords are not case sensitive such as LOAD, STORE, GROUP, FOREACH, DUMP etc.

→ Relations and paths are case-sensitive.

→ Function names are case sensitive such as PigStorage, COUNT

* Operators in Pig Latin

Arithmetic	Comparison	NULL	Boolean
+	=	IS NULL	AND
-	!=	IS NOT NULL	OR
*	<		NOT
/	>		
%	<=		
	>=		

→ Data types in Pig

1) Simple Data types :-

→ In Pig, fields of unspecified types are considered as an array of bytes which is known as byte array.

→ NULL : It denotes a value is unknown or is non-existent.

2) Complex Data types :-

Name	Description
Int	Whole numbers
Long	Large Whole numbers
Float	Decimals
Double	Very precise decimals
Chararray	Text strings
Bytearray	Raw bytes
Datetime	Datetime
Boolean	True or False.

2) complex data types :-

Name	Description
Tuple	An ordered set of fields
Bag	A collection of tuples
Map	key, Value pair

⇒ Running PIG.

→ We can run Pig in two ways.

1) Interactive Mode:- By invoking grunt shell.

2) Batch Mode:- By writing pig Latin statements in a file and save it with .pig extension.

⇒ Execution modes of Pig

1) Local mode :- To run pig in local mode, you need to have your file in local filesystem
syntax :- pig -x local filename .

2) MapReduce Mode :- You need to have access to Hadoop Cluster to read/write file. This is default mode in pig.

Syntax :- pig filename.

⇒ Relational operators in PIG.

- | | |
|-------------|-------------|
| a) FILTER | f) ORDER BY |
| b) FOREACH | g) JOIN |
| c) GROUP | h) UNION |
| d) DISTINCT | i) SPLIT |
| e) LIMIT | j) SAMPLE |

⇒ EVAL FUNCTION IN PIG

- a) AVG
- b) MAX
- c) COUNT

The sections have been designed as follows:

Objective: What is it that we are trying to achieve here?

Input: What is the input that has been given to us to act upon?

Act: The actual statement/command to accomplish the task at hand.

Outcome: The result/output as a consequence of executing the statement.

10.11 RELATIONAL OPERATORS

10.11.1 FILTER

FILTER operator is used to select tuples from a relation based on specified conditions.

Objective: Find the tuples of those student where the GPA is greater than 4.0.

Input:

Student (rollno:int, name:chararray, gpa:float)

Act:

```
A = load '/pigdemo/student.tsv' as (rollno:int, name:chararray, gpa:float);
```

```
B = filter A by gpa > 4.0;
```

```
DUMP B;
```

Output:

```
(1003,Smith,4.5)
(1004,Scott,4.2)
[root@volga1nx010 pigdemos]#
```

10.11.2 FOREACH

Use **FOREACH** when you want to do data transformation based on columns of data.

Objective: Display the name of all students in uppercase.

Input:

Student (rollno:int, name:chararray, gpa:float)

Act:

```
A = load '/pigdemo/student.tsv' as (rollno:int, name:chararray, gpa:float);
```

```
B = foreach A generate UPPER (name);
```

```
DUMP B;
```

Output:

```
(JOHN)
(JACK)
(SMITH)
(SCOTT)
(DONALD)
[root@volga1nx010 pigdemos]#
```

10.11.3 GROUP

GROUP operator is used to group data.

Objective: Group tuples of students based on their GPA.

Input:

Student (rollno:int, name:chararray, gpa:float)

Act:

```
A = load '/pigdemo/student.tsv' as (rollno:int, name:chararray, gpa:float);
```

```
B = GROUP A BY gpa;
```

```
DUMP B;
```

Output:

```
(3.0, { (1001, John, 3.0), (1001, John, 3.0) })
(3.5, { (1005, Joshi, 3.5), (1005, Joshi, 3.5) })
(4.0, { (1008, James, 4.0), (1002, Jack, 4.0) })
(4.2, { (1007, David, 4.2), (1004, Scott, 4.2) })
(4.5, { (1006, Alex, 4.5), (1003, Smith, 4.5) })
[root@volgailnx010 pigdemos]#
```

10.11.4 DISTINCT

DISTINCT operator is used to remove duplicate tuples. In Pig, *DISTINCT* operator works on the entire tuple and NOT on individual fields.

Objective: To remove duplicate tuples of students.

Input:

Student (rollno:int, name:chararray, gpa:float)

Input:

1001	John	3.0
1002	Jack	4.0
1003	Smith	4.5
1004	Scott	4.2
1005	Joshi	3.5
1006	Alex	4.5
1007	David	4.2
1008	James	4.0
1001	John	3.0
1005	Joshi	3.5

Act:

A = load '/pigdemo/student.tsv' as (rollno:int, name:chararray, gpa:float);

B = DISTINCT A;

DUMP B;

Output:

```
(1001,John,3.0)
(1002,Jack,4.0)
(1003,Smith,4.5)
(1004,Scott,4.2)
(1005,Joshi,3.5)
(1006,Alex,4.5)
(1007,David,4.2)
(1008,James,4.0)
[root@volgai ~]#
```

10.11.5 LIMIT

LIMIT operator is used to limit the number of output tuples.

Objective: Display the first 3 tuples from the "student" relation.

Input:

Student (rollno:int, name:chararray, gpa:float)

Act:

A = load '/pigdemo/student.tsv' as (rollno:int, name:chararray, gpa:float);

B = LIMIT A 3;

DUMP B;

Output:

```
(1001,John,3.0)
(1002,Jack,4.0)
(1003,Smith,4.5)
[root@volgai ~]#
```

10.11.6 ORDER BY

ORDER BY is used to sort a relation based on specific value.

Objective: Display the names of the students in Ascending Order.

Input:

Student (rollno:int, name:chararray, gpa:float)

Act:

```
A = load '/pigdemo/student.tsv' as (rollno:int, name:chararray, gpa:float);
B = ORDER A BY name;
DUMP B;
```

Output:

```
(1006,Alex,4.5)
(1007,David,4.2)
(1002,Jack,4.0)
(1008,James,4.0)
(1001,John,3.0)
(1001,John,3.0)
(1005,Joshi,3.5)
(1005,Joshi,3.5)
(1004,Scott,4.2)
(1003,Smith,4.5)
[root@volga1nx010 pigdemos]#
```

10.11.7 JOIN

It is used to join two or more relations based on values in the common field. It always performs inner Join.

Objective: To join two relations namely, "student" and "department" based on the values contained in the "rollno" column.

Input:

```
Student (rollno:int,name:chararray,gpa:float)
Department(rollno:int,deptno:int,deptname:chararray)
```

Act:

```
A = load '/pigdemo/student.tsv' as (rollno:int, name:chararray, gpa:float);
B = load '/pigdemo/department.tsv' as (rollno:int, deptno:int,deptname:chararray);
C = JOIN A BY rollno, B BY rollno;
DUMP C;
DUMP B;
```

Output:

```
(1001,John,3.0,1001,101,B.E.)
(1001,John,3.0,1001,101,B.E.)
(1002,Jack,4.0,1002,102,B.Tech)
(1003,Smith,4.5,1003,103,M.Tech)
(1004,Scott,4.2,1004,104,MCA)
(1005,Joshi,3.5,1005,105,MBA)
(1005,Joshi,3.5,1005,105,MBA)
(1006,Alex,4.5,1006,101,B.E.)
(1007,David,4.2,1007,104,MCA)
(1008,James,4.0,1008,102,B.Tech)
[root@volga1nx010 pigdemos]#
```

10.11.8 UNION

It is used to merge the contents of two relations.

Objective: To merge the contents of two relations "student" and "department".

Input:

Student (rollno:int,name:chararray,gpa:float)
Department(rollno:int,deptno:int,deptname:chararray)

Act:

```
A = load '/pigdemo/student.tsv' as (rollno, name, gp);
B = load '/pigdemo/department.tsv' as (rollno, deptno,deptname);
C = UNION A,B;
STORE C INTO '/pigdemo/uniondemo';
DUMP B;
```

Output:

"Store" is used to save the output to a specified path. The output is stored in two files: part-m-00000 contains "student" content and part-m-00001 contains "department" content.

Name	Type	Size	Replication	Block Size	Modification Time	Permission	Owner	Group
SUCCESS	file	0 B	3	128 MB	2015-02-24 17:23	rw-r--r--	root	supergroup
part-m-00000	file	146 B	3	128 MB	2015-02-24 17:23	rw-r--r--	root	supergroup
part-m-00001	file	114 B	3	128 MB	2015-02-24 17:23	rw-r--r--	root	supergroup

File: /pigdemo/uniondemo/part-m-00000

Goto: /pigdemo/uniondemo

[Go back to dir listing](#)

[Advanced view](#) [Download options](#)

```
1001 John 3.0
1002 Jack 4.0
1003 Smith 4.5
1004 Scott 4.2
1005 Joshi 3.5
1006 Alex 4.5
1007 David 4.2
1008 James 4.0
1009 John 3.0
1009 Joshi 3.5
```

File: /pigdemo/uniondemo/part-m-00001

Goto: /pigdemo/uniondemo

[Go back to dir listing](#)

[Advanced view](#) [Download options](#)

```
1001 101 B.E.
1002 102 B.Tech
1003 103 M.Tech
1004 104 MCA
1005 105 MBA
1006 101 B.E.
1007 104 MCA
1008 102 B.Tech
```

10.11.9 SPLIT

It is used to partition a relation into two or more relations.

Objective: To partition a relation based on the GPAs acquired by the students.

- GPA = 4.0, place it into relation X.
- GPA is < 4.0, place it into relation Y.

Input:

Student (rollno:int, name:chararray, gpa:float)

Act:

```
A = load '/pigdemo/student.tsv' as (rollno:int, name:chararray, gpa:float);
SPLIT A INTO X IF gpa == 4.0, Y IF gpa <= 4.0;
DUMP X;
```

Output: Relation X

```
(1002,Jack,4.0)
(1008,James,4.0)
[root@volgalmx010 pigdemos]#
```

Output: Relation Y

```
(1001,John,3.0)
(1002,Jack,4.0)
(1005,Joshi,3.5)
(1008,James,4.0)
(1001,John,3.0)
(1005,Joshi,3.5)
[root@volgalmx010 pigdemos]#
```

10.11.10 SAMPLE

It is used to select random sample of data based on the specified sample size.

Objective: To depict the use of *SAMPLE*.

Input:

Student (rollno:int, name:chararray, gpa:float)

Act:

```
A = load '/pigdemo/student.tsv' as (rollno:int, name:chararray, gpa:float);
B = SAMPLE A 0.01;
DUMP B;
```

10.12 EVAL FUNCTION

10.12.1 AVG

AVG is used to compute the average of numeric values in a single column bag.

Objective: To calculate the average marks for each student.

Input:

Student (studname:chararray,marks:int)

Act:

```
A = load '/pigdemo/student.csv' USING PigStorage(',') as (studname:chararray,marks:int);
B = GROUP A BY studname;
C = FOREACH B GENERATE A.studname, AVG(A.marks);
DUMP C;
```

Output:

```
{(Jack),(Jack),(Jack),(Jack)},39.75
{((John),(John),(John))},39.0
```

Note: You need to use PigStorage function if you wish to manipulate files other than .tsv.

10.12.2 MAX

MAX is used to compute the maximum of numeric values in a single column bag.

Objective: To calculate the maximum marks for each student.

Input:

Student (studname:chararray,marks:int)

Act:

```
A = load '/pigdemo/student.csv' USING PigStorage(',') as (studname:chararray, marks:int);
B = GROUP A BY studname;
C = FOREACH B GENERATE A.studname, MAX(A.marks);
DUMP C;
```

Output:

```
{(Jack),(Jack),(Jack),(Jack)},46
{((John),(John),(John))},45
```

Note: Similarly, you can try the MIN and the SUM functions as well.

10.12.3 COUNT

COUNT is used to count the number of elements in a bag.

Objective: To count the number of tuples in a bag.

Input:

Student (studname:chararray,marks:int)

Act:

```
A = load '/pigdemo/student.csv' USING PigStorage(',') as (studname:chararray, marks:int);
B = GROUP A BY studname;
C = FOREACH B GENERATE A.studname,COUNT(A);
DUMP C;
```

Output:

```
{((Jack),(Jack),(Jack),(Jack))},4
{((John),(John),(John))},4
```

Note: The default file format of Pig is .tsv file. Use PigStorage() to manipulate files other than .tsv file.

⇒ PIG versus HIVE

PIG

HIVE

- | | |
|---|--|
| * Used by programmers and researchers | * Used by the analyst |
| * Used for programming | * Used for Reporting |
| * Procedural dataflow language (PIGLATIN) | * Hive Query (HQL) language (like SQL) |
| * Semi structured data. | * For Structured data. |
| * Works on client side of a cluster | * Works on Server side of a cluster. |
| * Supports Avro file format | * Doesnot support Avro file format |
| * Developed at Yahoo | * Developed at Facebook |
| * It has web interface | * No webinterface. |

Introduction to HIVE

- Hive is a data Warehousing tool that sits on top of Hadoop. Hive is used to process structured data in Hadoop.
- The three main tasks by Apache Hive are:
 - *) Summarization
 - *) Querying
 - *) Analysis
- Hive provides HQL (Hive Query Language) or Hive QL which is similar to SQL.
- It is designed to support OLAP (online Analytical Processing).

- Hive features :-
 - It is similar to SQL
 - HQL is easy to code.
 - Supports structs, lists & maps datatypes
 - Supports SQL filters, group by & order-by clause
 - Custom Types, custom functions can be defined
- Hive Data Units :-
 - 1) Databases :- Namespace for tables
 - 2) Tables :- Set of records
 - 3) Partitions :- Logical separation of data based on classification of given information as per specific attributes
 - 4) Buckets (clusters) :- Determines bucket in which record should be placed.

Let us take an example to understand partitioning and bucketing.

PICTURE THIS...

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

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

Partitioning tables changes how Hive structures the data storage. Hive will create subdirectories reflecting the partitioning structure like

`.../customers/country=ABC`

Although partitioning helps in enhancing performance and is recommended, having too many partitions may prove detrimental for few queries.

Bucketing is another technique of managing large datasets. If we partition the dataset based on `customer_ID`, we would end up with far too many partitions. Instead, if we bucket the customer table and use `customer_id` as the bucketing column, the value of this column will be hashed by a user-defined number

into buckets. Records with the same `customer_id` will always be placed in the same bucket. Assuming we have far more `customer_ids` than the number of buckets, each bucket will house many `customer_ids`. While creating the table you can specify the number of buckets that you would like your data to be distributed in using the syntax "`CLUSTERED BY (customer_id) INTO XX BUCKETS`"; here XX is the number of buckets.

When to Use Partitioning/Bucketing?

Bucketing works well when the field has high cardinality (cardinality is the number of values a column or field can have) and data is evenly distributed among buckets. Partitioning works best when the cardinality of the partitioning field is not too high. Partitioning can be done on multiple fields with an order (Year/Month/Day) whereas bucketing can be done on only one field.

Figure 9.5 shows how these data units are arranged in a Hive Cluster. Figure 9.6 describes the semblance of Hive structure with database.

A database contains several tables. Each table is constituted of rows and columns. In Hive, tables are stored as a folder and partition tables are stored as a sub-directory. Bucketed tables are stored as a file.

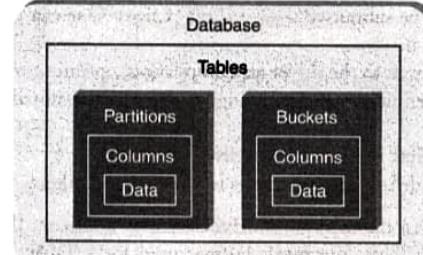


Figure 9.5 Data units as arranged in a Hive.

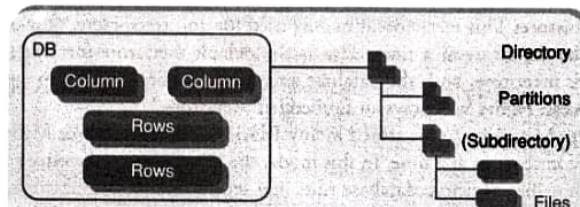


Figure 9.6 Semblance of Hive structure with database.

9.2 HIVE ARCHITECTURE

Hive Architecture is depicted in Figure 9.7. The various parts are as follows:

1. **Hive Command-Line Interface (Hive CLI):** The most commonly used interface to interact with Hive.
2. **Hive Web Interface:** It is a simple Graphic User Interface to interact with Hive and to execute query.
3. **Hive Server:** This is an optional server. This can be used to submit Hive Jobs from a remote client.

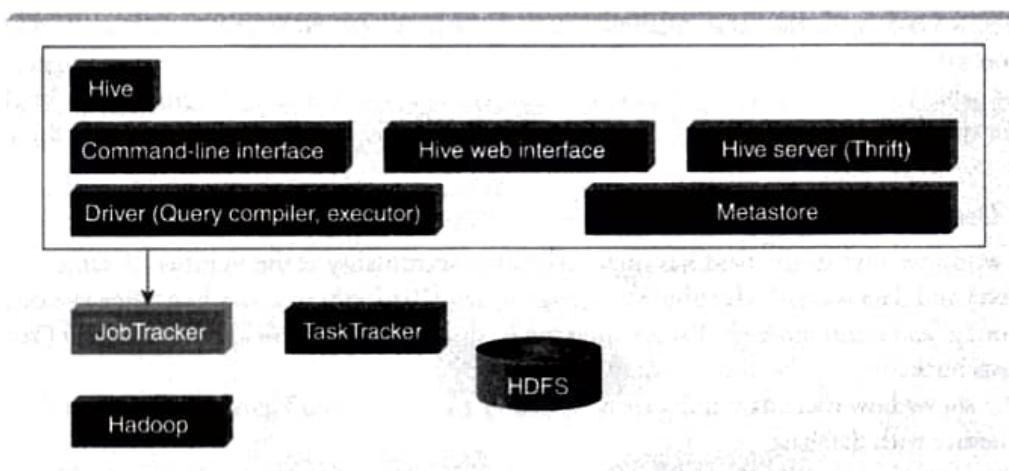


Figure 9.7 Hive architecture.

4. **JDBC/ODBC:** Jobs can be submitted from a JDBC Client. One can write a Java code to connect to Hive and submit jobs on it.
5. **Driver:** Hive queries are sent to the driver for compilation, optimization and execution.
6. **Metastore:** Hive table definitions and mappings to the data are stored in a Metastore. A Metastore consists of the following:
 - **Metastore service:** Offers interface to the Hive.
 - **Database:** Stores data definitions, mappings to the data and others.

The metadata which is stored in the metastore includes IDs of Database, IDs of Tables, IDs of Indexes, etc., the time of creation of a Table, the Input Format used for a Table, the Output Format used for a Table, etc. The metastore is updated whenever a table is created or deleted from Hive. There are three kinds of metastore.

- 1. Embedded Metastore:** This metastore is mainly used for unit tests. Here, only one process is allowed to connect to the metastore at a time. This is the default metastore for Hive. It is Apache Derby Database. In this metastore, both the database and the metastore service run embedded in the main Hive Server process. Figure 9.8 shows an Embedded Metastore.
- 2. Local Metastore:** Metadata can be stored in any RDBMS component like MySQL. Local metastore allows multiple connections at a time. In this mode, the Hive metastore service runs in the main Hive Server process, but the metastore database runs in a separate process, and can be on a separate host. Figure 9.9 shows a Local Metastore.
- 3. Remote Metastore:** In this, the Hive driver and the metastore interface run on different JVMs (which can run on different machines as well) as in Figure 9.10. This way the database can be fire-walled from the Hive user and also database credentials are completely isolated from the users of Hive.

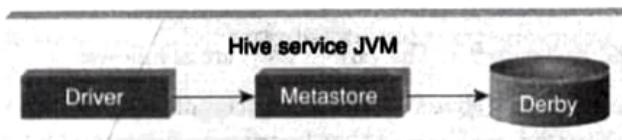


Figure 9.8 Embedded Metastore.

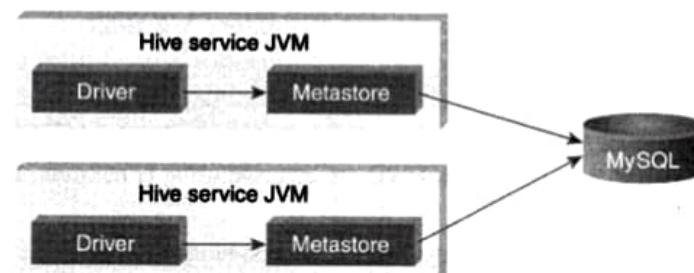


Figure 9.9 Local Metastore.

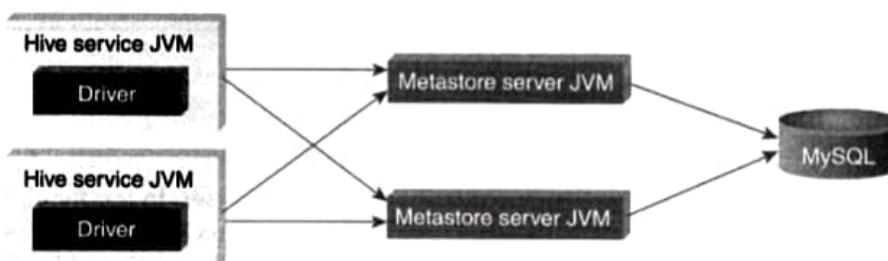


Figure 9.10 Remote Metastore.

HIVE DATA TYPES

Primitive Data Types

Numeric Data Type

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

String Types

STRING	
VARCHAR	Only available starting with Hive 0.12.0
CHAR	Only available starting with Hive 0.13.0

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

Miscellaneous Types

BOOLEAN	
BINARY	Only available starting with Hive

9.3.2 Collection Data Types

Collection Data Types

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

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

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

9.4 HIVE FILE FORMAT

The file formats in Hive specify how records are encoded in a file.

9.4.1 Text File

The default file format is text file. In this format, each record is a line in the file. In text file, different control characters are used as delimiters. The delimiters are ^A (octal 001, separates all fields), ^B (octal 002, separates the elements in the array or struct), ^C (octal 003, separates key-value pair), and \n. The term **field** is used when overriding the default delimiter. The supported text files are CSV and TSV. JSON or XML documents too can be specified as text file.

9.4.2 Sequential File

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

9.4.3 RCFFile (Record Columnar File)

RCFFile stores the data in **Column Oriented Manner** which ensures that **Aggregation** operation is not an expensive operation. For example, consider a table which contains four columns as shown in Table 9.1.

Instead of only partitioning the table horizontally like the row-oriented DBMS (row-store), RCFFile partitions this table first horizontally and then vertically to serialize the data. Based on the user-specified value, first the table is partitioned into multiple row groups horizontally. Depicted in Table 9.2, Table 9.1 is partitioned into two row groups by considering three rows as the size of each row group.

Next, in every row group RCFFile partitions the data vertically like column-store. So the table will be serialized as shown in Table 9.3.

Table 9.1 A table with four columns

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

Table 9.2 Table with two row groups

Row Group 1				Row Group 2			
C1	C2	C3	C4	C1	C2	C3	C4
11	12	13	14	41	42	43	44
21	22	23	24	51	52	53	54
31	32	33	34				

Table 9.3 Table in RCFFile Format

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

9.5 HIVE QUERY LANGUAGE (HQL)

Hive query language provides basic SQL like operations. Here are few of the tasks which HQL can do easily.

1. Create and manage tables and partitions.
2. Support various Relational, Arithmetic, and Logical Operators.
3. Evaluate functions.
4. Download the contents of a table to a local directory or result of queries to HDFS directory.

9.5.1 DDL (Data Definition Language) Statements

These statements are used to build and modify the tables and other objects in the database. The DDL commands are as follows:

1. Create/Drop/Alter Database
2. Create/Drop/Truncate Table
3. Alter Table/Partition/Column
4. Create/Drop/Alter View
5. Create/Drop/Alter Index
6. Show
7. Describe

9.5.2 DML (Data Manipulation Language) Statements

These statements are used to retrieve, store, modify, delete, and update data in database. The DML commands are as follows:

1. Loading files into table.
2. Inserting data into Hive Tables from queries.

Note: Hive 0.14 supports update, delete, and transaction operations.

9.8 USER-DEFINED FUNCTION (UDF)

In Hive, you can use custom functions by defining the User-Defined Function (UDF).

Objective: Write a Hive function to convert the values of a field to uppercase.

Act:

```
package com.example.hive.udf;
import org.apache.hadoop.hive.ql.exec.Description;
import org.apache.hadoop.hive.ql.exec.UDF;
@Description(
    name="SimpleUDFExample")
```

```
public final class MyLowerCase extends UDF {
    public String evaluate(final String word) {
        return word.toLowerCase();
    }
}
```

Note: Convert this Java Program into Jar.

```
ADD JAR /root/hivedemos/UpperCase.jar;
CREATE TEMPORARY FUNCTION touppercase AS 'com.example.hive.udf.MyUpperCase';
SELECT TOUPPERCASE(name) FROM STUDENT;
```

Outcome:

```
hive> ADD JAR /root/hivedemos/UpperCase.jar;
Added [/root/hivedemos/UpperCase.jar] to class path
Added resources: [/root/hivedemos/UpperCase.jar]
hive> CREATE TEMPORARY FUNCTION touppercase AS 'com.example.hive.udf.MyUpperCase';
OK
Time taken: 0.014 seconds
hive> ■
```

```
hive> Select touppercase (name) from STUDENT;
OK
JOHN
JACK
SMITH
SCOTT
JOSHI
ALEX
DAVID
JAMES
JOHN
JOSHI
Time taken: 0.061 seconds, Fetched: 10 row(s)
hive> ■
```