Basic Data types in Hive

We will see the Basic data types in Hive.

Numeric Types

**TINYINT**  
1 byte signed integer. Values range -128 to 127

**SMALLINT**  
2 byte signed integer. Values range -32,768 to 32,767

**INT**  
4 byte signed integer. Values range -2,147,483,648 to 2,147,483,647

**BIGINT**  
8 byte signed integer. Values range -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807

**FLOAT**  
4 byte single precision floating point number

**DOUBLE**  
8 byte double precision floating point number

**DECIMAL**  
Decimal is used to represent arbitrary precision decimal numbers. We can provide the precision and scale for the Decimal numbers.  
Precision- Total number of digits.  
Scale – Number of digits in the fractional part i.e digits after ‘.’.

String Types

**STRING**A variable length character string.

**VARCHAR**  
A variable length character string for which we need to specify the maximum number of characters allowed(1-65355). If string value have more characters than specified, then the string value is truncated. If the string value have less characters then only those characters are stored.

**CHAR**  
A fixed length character string(1-255). If string value have more characters than specified, then the string value is truncated. If the string value have less characters then the string is padded with spaces.

Date Types

**Date**  
Date can store year/month/day component of a Date. Date do not store time components. Format YYYY-MM-DD.

**Timestamp**  
Timestamp stores the timestamp values in the format ‘YYYY-MM-DD HH:MM:SS.fffffffff’.

Others

**Boolean**Boolean can have literal values TRUE or FALSE. Mixed case is allowed whicle using boolean literals.

We can cast TINYINT, SMALLINT, INT, BIGINT, FLOAT, DOUBLE, or DECIMAL to BOOLEAN by using CAST() function. A value of 0 is considered as false, and any non-zero value is considered as true.

We cannot cast a STRING value to BOOLEAN. But we can cast a BOOLEAN value to STRING, returning ‘1’ for true and ‘0’ for false.

Complex data type in Hive: Array

Array – a complex data type in Hive which can store an ordered collection of similar elements accessible using 0 based index.

Create Table

While creating a table with Array data type, we need to specify the ‘COLLECTION ITEMS TERMINATED BY’ character. This character will be used to specify different elements in an array. We will create a table containing an array<int> data type.

CREATE TABLE Array\_test(

id int,

all\_nums array<int>

) COMMENT 'This is a table stored as textfile'

ROW FORMAT DELIMITED

FIELDS TERMINATED BY ','

COLLECTION ITEMS TERMINATED BY '$'

STORED AS TEXTFILE;

Load Data

We have a file ‘/home/hadoop/array\_test‘ on our HDFS.  
The file contains 4 rows and each row contains an id and an array. Contents of the file:-

1,326362$3443$23432$875665$3443$43534$234$342

2,123$323$546$546$5476

3,435$345$678$122$98987

4,234$7234$65242$6272

Load the data into table:-

LOAD DATA INPATH '/home/hadoop/array\_test' overwrite into table array\_test;

Verify data

Now let us execute some queries on the array data type.

#Select the id and all\_nums array for all the rows.

0: jdbc:hive2://localhost:10000> select id, all\_nums from array\_test;

+-----+------------------------------------------------+--+

| id  |                    all\_nums                    |

+-----+------------------------------------------------+--+

| 1   | [326362,3443,23432,875665,3443,43534,234,342]  |

| 2   | [123,323,546,546,5476]                         |

| 3   | [435,345,678,122,98987]                        |

| 4   | [234,7234,65242,6272]                          |

+-----+------------------------------------------------+--+

#Select the id and 2nd array element from all the rows. the index is 0 based

0: jdbc:hive2://localhost:10000> select id, all\_nums[1] as num from array\_test;

+-----+-------+--+

| id  |  num  |

+-----+-------+--+

| 1   | 3443  |

| 2   | 323   |

| 3   | 345   |

| 4   | 7234  |

+-----+-------+--+

#Select the id and 5th array element from all the rows. NULL is returned in case no

#element is present at the given index.

0: jdbc:hive2://localhost:10000> select id, all\_nums[4] as num from array\_test;

+-----+--------+--+

| id  |  num   |

+-----+--------+--+

| 1   | 3443   |

| 2   | 5476   |

| 3   | 98987  |

| 4   | NULL   |

+-----+--------+--+

Complex data type in Hive: Struct

Struct – a complex data type in Hive which can store a set of fields of different data types. The elements of a struct are accessed using dot notation.

Create Table

While creating a table with Struct data type, we need to specify the ‘COLLECTION ITEMS TERMINATED BY’ character. This character will be used to specify different elements in an Struct. We will create a table containing a Struct data type.

CREATE TABLE Struct\_test(

id int,

weather\_reading struct<temp:int, humidity:int, comment:string>

) COMMENT 'This is a table stored as textfile'

ROW FORMAT DELIMITED

FIELDS TERMINATED BY ','

COLLECTION ITEMS TERMINATED BY '$'

STORED AS TEXTFILE;

Load Data

We have a file ‘/home/hadoop/weather‘ on our HDFS.  
The file contains 4 rows and each row contains an id and data for Struct fields separated by ‘$’. Contents of the file:-

1,32$65$moderate

2,37$78$humid

3,43$55$hot

4,23$45$cold

Load the data into table:-

LOAD DATA INPATH '/home/hadoop/weather' overwrite into table struct\_test;

Verify data

Now let us execute some queries on the Struct data type.

#Select the id and weather\_reading struct from all the rows.

0: jdbc:hive2://localhost:10000> select id, weather\_reading from struct\_test;

+-----+-------------------------------------------------+--+

| id  |                 weather\_reading                 |

+-----+-------------------------------------------------+--+

| 1   | {"temp":32,"humidity":65,"comment":"moderate"}  |

| 2   | {"temp":37,"humidity":78,"comment":"humid"}     |

| 3   | {"temp":43,"humidity":55,"comment":"hot"}       |

| 4   | {"temp":23,"humidity":45,"comment":"cold"}      |

+-----+-------------------------------------------------+--+

#Select the id and temp field from the weather\_reading struct for all the rows

0: jdbc:hive2://localhost:10000> select id, weather\_reading.temp from struct\_test;

+-----+-------+--+

| id  | temp  |

+-----+-------+--+

| 1   | 32    |

| 2   | 37    |

| 3   | 43    |

| 4   | 23    |

+-----+-------+--+

#Select the id and humidity field from weather\_reading struct for all the rows

0: jdbc:hive2://localhost:10000> select id, weather\_reading.humidity from struct\_test;

+-----+-----------+--+

| id  | humidity  |

+-----+-----------+--+

| 1   | 65        |

| 2   | 78        |

| 3   | 55        |

| 4   | 45        |

+-----+-----------+--+

#Select the id and comment field from weather\_reading struct for all the rows

0: jdbc:hive2://localhost:10000> select id, weather\_reading.comment from struct\_test;

+-----+-----------+--+

| id  |  comment  |

+-----+-----------+--+

| 1   | moderate  |

| 2   | humid     |

| 3   | hot       |

| 4   | cold      |

+-----+-----------+--+

# Complex data type in Hive: Map

Map – a complex data type in Hive which can store Key-Value pairs. Values from a map can be accessed using the keys.

## Create Table

While creating a table with Map data type, we need to specify the –

1. ‘COLLECTION ITEMS TERMINATED BY’ character to specify different key-value pairs.
2. ‘MAP KEYS TERMINATED BY’ character to specify key and value.

We will create a table containing an Map<int, string> data type –

CREATE TABLE Map\_test(

id int,

comments\_map Map<int, string>

) COMMENT 'This is a table stored as textfile'

ROW FORMAT DELIMITED

FIELDS TERMINATED BY ','

COLLECTION ITEMS TERMINATED BY '#'

MAP KEYS TERMINATED BY '@'

STORED AS TEXTFILE;

## Load Data

We have a file ‘/home/hadoop/comments‘ on our HDFS.  
The file contains 4 rows and each row contains an id and a map containing user\_id and comment. Contents of the file:-

1,1@india is great#2@india won icc t20#3@jai hind

2,1@we are awesome#2@i like cricket

3,1@hurray we won#2@what a great match#3@watching cricket all day

4,1@hectic day#3@irctc rocks

Load the data into table:-

LOAD DATA INPATH '/home/hadoop/comments' overwrite into table map\_test;

## Verify data

Now let us execute some queries on the Map data type.

#Select id and all the comments stored in Map column.

**select id, comments\_map from map\_test;**

+-----+--------------------------------------------------------------------------+--+

| id  |                               comments\_map                               |

+-----+--------------------------------------------------------------------------+--+

| 1   | {1:"india is great",2:"india won icc t20",3:"jai hind"}                  |

| 2   | {1:"we are awesome",2:"i like cricket"}                                  |

| 3   | {1:"hurray we won",2:"what a great match",3:"watching cricket all day"}  |

| 4   | {1:"hectic day",3:"irctc rocks"}                                         |

+-----+--------------------------------------------------------------------------+--+

#Select id and all the comments by user\_id 1

**select id, comments\_map[1] as comments\_by\_user\_1 from map\_test;**

+-----+---------------------+--+

| id  | comments\_by\_user\_1  |

+-----+---------------------+--+

| 1   | india is great      |

| 2   | we are awesome      |

| 3   | hurray we won       |

| 4   | hectic day          |

+-----+---------------------+--+

#Select id and all the comments by user\_id 2

**select id, comments\_map[2] as comments\_by\_user\_2 from map\_test;**

+-----+---------------------+--+

| id  | comments\_by\_user\_2  |

+-----+---------------------+--+

| 1   | india won icc t20   |

| 2   | i like cricket      |

| 3   | what a great match  |

| 4   | NULL                |

+-----+---------------------+--+