LanguageManual Types

Hive Data Types

Overview

This lists all supported data types in Hive. See Type System in the Tutorial for additional information.

For data types supported by HCatalog, see:

- HCatLoader Data Types
- HCatStorer Data Types
- HCatRecord Data Types

Numeric Types

- `TINYINT` (1-byte signed integer, from `-128` to `127`)
- `SMALLINT` (2-byte signed integer, from `-32,768` to `32,767`)
- `INT`/`INTEGER (4-byte signed integer, from -2,147,483,648 to 2,147,483,647)`

- `BIGINT` (8-byte signed integer, from `-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)
- `DOUBLE PRECISION` (alias for DOUBLE, only available starting with Hive 2.2.0)

- `DECIMAL`
  - Introduced in Hive 0.11.0 with a precision of 38 digits
  - Hive 0.13.0 introduced user-definable precision and scale
- `NUMERIC` (same as `DECIMAL`, starting with Hive 3.0.0)

Date/Time Types

- `TIMESTAMP` (Note: Only available starting with Hive 0.8.0)
- `DATE` (Note: Only available starting with Hive 0.12.0)
- `INTERVAL` (Note: Only available starting with Hive 1.2.0)

String Types

- `STRING`
- `VARCHAR` (Note: Only available starting with Hive 0.12.0)
- `CHAR` (Note: Only available starting with Hive 0.13.0)

Misc Types

- `BOOLEAN`
- `BINARY` (Note: Only available starting with Hive 0.8.0)

Complex Types

- arrays: `ARRAY<data_type>` (Note: negative values and non-constant expressions are allowed as of Hive 0.14.)
- maps: `MAP<primitive_type, data_type>` (Note: negative values and non-constant expressions are allowed as of Hive 0.14.)
- structs: `STRUCT<col_name : data_type [COMMENT col_comment], ...>`
- union: `UNIONTYPE<data_type, data_type, ...>` (Note: Only available starting with Hive 0.7.0.)

Column Types

Integral Types (`TINYINT`, `SMALLINT`, `INT/INTEGER`, `BIGINT`)

Integral literals are assumed to be `INT` by default, unless the number exceeds the range of `INT` in which case it is interpreted as a BIGINT, or if one of the following postfixes is present on the number.

Type	Postfix	Example
TINYINT	Y	100Y
SMALLINT	S	100S
BIGINT	L	100L

Version

`INTEGER` is introduced as a synonym for `INT` in Hive 2.2.0 (HIVE-14950).

Strings

String literals can be expressed with either single quotes (') or double quotes ("). Hive uses C-style escaping within the strings.

Varchar

Varchar types are created with a length specifier (between 1 and 65535), which defines the maximum number of characters allowed in the character string. If a string value being converted/assigned to a varchar value exceeds the length specifier, the string is silently truncated. Character length is determined by the number of code points contained by the character string.

Like string, trailing whitespace is significant in varchar and will affect comparison results.

Limitations

Non-generic UDFs cannot directly use varchar type as input arguments or return values. String UDFs can be created instead, and the varchar values will be converted to strings and passed to the UDF. To use varchar arguments directly or to return varchar values, create a GenericUDF.
There may be other contexts which do not support varchar, if they rely on reflection-based methods for retrieving type information. This includes some SerDe implementations.

Version

Varchar datatype was introduced in Hive 0.12.0 (HIVE-4844).

Char

Char types are similar to Varchar but they are fixed-length meaning that values shorter than the specified length value are padded with spaces but trailing spaces are not important during comparisons. The maximum length is fixed at 255.

```
CREATE TABLE foo (bar CHAR(10))
```

Version

Char datatype was introduced in Hive 0.13.0 (HIVE-5191).

Timestamps

Supports traditional UNIX timestamp with optional nanosecond precision.

Supported conversions:

- Integer numeric types: Interpreted as UNIX timestamp in seconds
- Floating point numeric types: Interpreted as UNIX timestamp in seconds with decimal precision
- Strings: JDBC compliant java.sql.Timestamp format "`YYYY-MM-DD HH:MM:SS.fffffffff`" (9 decimal place precision)

Timestamps are interpreted to be timezoneless and stored as an offset from the UNIX epoch. Convenience UDFs for conversion to and from timezones are provided (`to_utc_timestamp`, `from_utc_timestamp`).
All existing datetime UDFs (month, day, year, hour, etc.) work with the `TIMESTAMP` data type.

Timestamps in text files have to use the format `yyyy-mm-dd hh:mm:ss[.f...]`. If they are in another format, declare them as the appropriate type (INT, FLOAT, STRING, etc.) and use a UDF to convert them to timestamps.

On the table level, alternative timestamp formats can be supported by providing the format to the SerDe property "timestamp.formats" (as of release 1.2.0 with HIVE-9298). For example, `yyyy-MM-dd'T'HH:mm:ss.SSS,yyyy-MM-dd'T'HH:mm:ss.`

Version

Timestamps were introduced in Hive 0.8.0 (HIVE-2272).

Dates

`DATE` values describe a particular year/month/day, in the form `YYYY--MM--DD`. For example, DATE '2013--01--01'. Date types do not have a time of day component. The range of values supported for the Date type is 0000--01--01 to 9999--12--31, dependent on support by the primitive Java Date type.

Version

Dates were introduced in Hive 0.12.0 (HIVE-4055).

Casting Dates

Date types can only be converted to/from Date, Timestamp, or String types.

Valid casts to /from Date type	Result
cast(date as date)	Same date value
cast(timestamp as date)	The year/month/day of the timestamp is determined, based on the local timezone, and returned as a date value.
cast(string as date)	If the string is in the form 'YYYY-MM-DD', then a date value corresponding to that year/month/day is returned. If the string value does not match this formate, then NULL is returned.
cast(date as timestamp)	A timestamp value is generated corresponding to midnight of the year/month/day of the date value, based on the local timezone.
cast(date as string)	The year/month/day represented by the Date is formatted as a string in the form 'YYYY-MM-DD'.

Intervals

Supported Interval Description	Example	Meaning	Since
Intervals of time units: SECOND / MINUTE / DAY / MONTH / YEAR	INTERVAL '1' DAY	an interval of 1 day(s)	Hive 1.2.0 (HIVE-9792).
Year to month intervals, format: SY-M S: optional sign (+/-) Y: year count M: month count	INTERVAL '1-2' YEAR TO MONTH	shorthand for: INTERVAL '1' YEAR + INTERVAL '2' MONTH	Hive 1.2.0 (HIVE-9792).
Day to second intervals, format: SD H:M:S.nnnnnn S: optional sign (+/-) D: day countH: hours M: minutes S: seconds nnnnnn: optional nanotime	INTERVAL '1 2:3:4.000005' DAY	shorthand for: INTERVAL '1' DAY+ INTERVAL '2' HOUR + INTERVAL '3' MINUTE + INTERVAL '4' SECOND + INTERVAL '5' NANO	Hive 1.2.0 (HIVE-9792).
Support for intervals with constant numbers	INTERVAL 1 DAY	aids query readability / portability	Hive 2.2.0 (HIVE-13557).

Support for intervals with expressions: this may involve other functions/columns. The expression must return with a number (which is not floating-point) or with a string.	INTERVAL (1+dt) DAY	enables dynamic intervals	Hive 2.2.0 (HIVE-13557).
Optional usage of interval keyword the usage of the INTERVAL keyword is mandatory for intervals with expressions (ex: INTERVAL (1+dt) SECOND)	1 DAY '1-2' YEAR TO MONTH	INTERVAL 1 DAY INTERVAL '1-2' YEARS TO MONTH	Hive 2.2.0 (HIVE-13557).
Add timeunit aliases to aid portability / readability: SECONDS / MINUTES / HOURS / DAYS / WEEKS / MONTHS / YEARS	2 SECONDS	2 SECOND	Hive 2.2.0 (HIVE-13557).

Decimals

Version

Decimal datatype was introduced in Hive 0.11.0 (HIVE-2693) and revised in Hive 0.13.0 (HIVE-3976).

`NUMERIC` is the same as `DECIMAL` as of Hive 3.0.0 (HIVE-16764).

The `DECIMAL` type in Hive is based on Java's BigDecimal which is used for representing immutable arbitrary precision decimal numbers in Java. All regular number operations (e.g. +, -, *, /) and relevant UDFs (e.g. Floor, Ceil, Round, and many more) handle decimal types. You can cast to/from decimal types like you would do with other numeric types. The persistence format of the decimal type supports both scientific and non-scientific notation. Therefore, regardless of whether your dataset contains data like 4.004E+3 (scientific notation) or 4004 (non-scientific notation) or a combination of both, `DECIMAL` can be used for it.

- Hive 0.11 and 0.12 have the precision of the `DECIMAL` type fixed and limited to 38 digits.
- As of Hive 0.13 users can specify scale and precision when creating tables with the `DECIMAL` datatype using a `DECIMAL(precision, scale)` syntax. If scale is not specified, it defaults to 0 (no fractional digits). If no precision is specified, it defaults to 10.

```
CREATE TABLE foo (
  a DECIMAL, -- Defaults to decimal(10,0)
  b DECIMAL(9, 7)
)
```

For usage, see Floating Point Types in the Literals section below.

Decimal Literals

Integral literals larger than BIGINT must be handled with Decimal(38,0). The Postfix BD is required. Example:

```
select CAST(18446744073709001000BD AS DECIMAL(38,0)) from my_table limit 1;
```

Decimal Type Incompatibilities between Hive 0.12.0 and 0.13.0

With the changes in the Decimal data type in Hive 0.13.0, the pre-Hive 0.13.0 columns (of type "decimal") will be treated as being of type decimal (10,0). What this means is that existing data being read from these tables will be treated as 10-digit integer values, and data being written to these tables will be converted to 10-digit integer values before being written. To avoid these issues, Hive users on 0.12 or earlier with tables containing Decimal columns will be required to migrate their tables, after upgrading to Hive 0.13.0 or later.

Upgrading Pre-Hive 0.13.0 Decimal Columns

If the user was on Hive 0.12.0 or earlier and created tables with decimal columns, they should perform the following steps on these tables **after** upgrading to Hive 0.13.0 or later.

1. Determine what precision/scale you would like to set for the decimal column in the table.
2. For each decimal column in the table, update the column definition to the desired precision/scale using the ALTER TABLE command:

   ```
   ALTER TABLE foo CHANGE COLUMN dec_column_name dec_column_name DECIMAL(38,18);
   ```

   If the table is not a partitioned table, then you are done. If the table has partitions, then go on to step 3.
3. If the table is a partitioned table, then find the list of partitions for the table:

   ```
   SHOW PARTITIONS foo;

   ds=2008-04-08/hr=11
   ds=2008-04-08/hr=12
   ...
   ```

4. Each existing partition in the table must also have its DECIMAL column changed to add the desired precision/scale.

   This can be done with a single ALTER TABLE CHANGE COLUMN by using dynamic partitioning (available for ALTER TABLE CHANGE COLUMN in Hive 0.14 or later, with HIVE-8411):

   ```
   SET hive.exec.dynamic.partition = true;

   -- hive.exec.dynamic.partition needs to be set to true to enable dynamic partitioning with ALTER
   PARTITION
   -- This will alter all existing partitions of the table - be sure you know what you are doing!
   ALTER TABLE foo PARTITION (ds, hr) CHANGE COLUMN dec_column_name dec_column_name DECIMAL(38,18);
   ```

   Alternatively, this can be done one partition at a time using ALTER TABLE CHANGE COLUMN, by specifying one partition per statement (This is available in Hive 0.14 or later, with HIVE-7971.):

   ```
   ALTER TABLE foo PARTITION (ds='2008-04-08', hr=11) CHANGE COLUMN dec_column_name dec_column_name DECIMAL
   (38,18);
   ALTER TABLE foo PARTITION (ds='2008-04-08', hr=12) CHANGE COLUMN dec_column_name dec_column_name DECIMAL
   (38,18);
   ...
   ```

The Decimal datatype is discussed further in Floating Point Types below.

Union Types

UNIONTYPE support is incomplete

The UNIONTYPE datatype was introduced in Hive 0.7.0 (HIVE-537), but full support for this type in Hive remains incomplete. Queries that reference UNIONTYPE fields in JOIN (HIVE-2508), WHERE, and GROUP BY clauses will fail, and Hive does not define syntax to extract the tag or value fields of a UNIONTYPE. This means that UNIONTYPEs are effectively pass-through-only.

Union types can at any one point hold exactly one of their specified data types. You can create an instance of this type using the `create_union` UDF:

```
CREATE TABLE union_test(foo UNIONTYPE<int, double, array<string>, struct<a:int,b:string>>);
SELECT foo FROM union_test;

{0:1}
{1:2.0}
{2:["three","four"]}
{3:{"a":5,"b":"five"}}
{2:["six","seven"]}
{3:{"a":8,"b":"eight"}}
{0:9}
{1:10.0}
```

The first part in the deserialized union is the *tag* which lets us know which part of the union is being used. In this example `0` means the first data_type from the definition which is an `int` and so on.

To create a union you have to provide this tag to the `create_union` UDF:

```
SELECT create_union(0, key), create_union(if(key<100, 0, 1), 2.0, value), create_union(1, "a", struct(2, "b"))
FROM src LIMIT 2;

{0:"238"}        {1:"val_238"}        {1:{"col1":2,"col2":"b"}}
{0:"86"}        {0:2.0}        {1:{"col1":2,"col2":"b"}}
```

Literals

Floating Point Types

Floating point literals are assumed to be DOUBLE. Scientific notation is not yet supported.

Decimal Types

Version

Decimal datatype was introduced in Hive 0.11.0 (HIVE-2693). See Decimal Datatype above.

`NUMERIC` is the same as `DECIMAL` as of Hive 3.0.0 (HIVE-16764).

Decimal literals provide precise values and greater range for floating point numbers than the DOUBLE type. Decimal data types store exact representations of numeric values, while DOUBLE data types store very close approximations of numeric values.

Decimal types are needed for use cases in which the (very close) approximation of a DOUBLE is insufficient, such as financial applications, equality and inequality checks, and rounding operations. They are also needed for use cases that deal with numbers outside the DOUBLE range (approximately -10^{308} to 10^{308}) or very close to zero (-10^{-308} to 10^{-308}). For a general discussion of the limits of the DOUBLE type, see the Wikipedia article Double-precision floating-point format.

The precision of a Decimal type is limited to 38 digits in Hive. See HIVE-4271 and HIVE-4320 for comments about the reasons for choosing this limit.

Using Decimal Types

You can create a table in Hive that uses the Decimal type with the following syntax:

```
create table decimal_1 (t decimal);
```

The table `decimal_1` is a table having one field of type decimal which is basically a Decimal value.

You can read and write values in such a table using either the LazySimpleSerDe or the LazyBinarySerDe. For example:

```
alter table decimal_1 set serde 'org.apache.hadoop.hive.serde2.lazy.LazySimpleSerDe';
```

or:

```
alter table decimal_1 set serde 'org.apache.hadoop.hive.serde2.lazy.LazyBinarySerDe';
```

You can use a cast to convert a Decimal value to any other primitive type such as a BOOLEAN. For example:

```
select cast(t as boolean) from decimal_2;
```

Mathematical UDFs

Decimal also supports many arithmetic operators, mathematical UDFs and UDAFs with the same syntax as used in the case of DOUBLE.

Basic mathematical operations that can use decimal types include:

- Positive
- Negative
- Addition
- Subtraction
- Multiplication
- Division
- Average (avg)
- Sum
- Count
- Modulus (pmod)
- Sign – Hive 0.13.0 and later
- Exp – Hive 0.13.0 and later
- Ln – Hive 0.13.0 and later
- Log2 – Hive 0.13.0 and later
- Log10 – Hive 0.13.0 and later
- Log(*base*) – Hive 0.13.0 and later
- Sqrt – Hive 0.13.0 and later
- Sin – Hive 0.13.0 and later
- Asin – Hive 0.13.0 and later
- Cos – Hive 0.13.0 and later
- Acos – Hive 0.13.0 and later
- Tan – Hive 0.13.0 and later
- Atan – Hive 0.13.0 and later
- Radians – Hive 0.13.0 and later
- Degrees – Hive 0.13.0 and later

These rounding functions can also take decimal types:

- Floor

- Ceiling
- Round

Power(decimal, n) only supports positive integer values for the exponent n.

Casting Decimal Values

Casting is supported between decimal values and any other primitive type such as integer, double, boolean, and so on.

Testing Decimal Types

Two new tests have been added as part of the TestCliDriver framework within Hive. They are decimal_1.q and decimal_2.q. Other tests such as udf7.q cover the gamut of UDFs mentioned above.

More tests need to be added that demonstrate failure or when certain types of casts are prevented (for example, casting to date). There is some ambiguity in the round function because the rounding of Decimal does not work exactly as the SQL standard, and therefore it has been omitted in the current work.

For general information about running Hive tests, see How to Contribute to Apache Hive and Hive Developer FAQ.

Handling of NULL Values

Missing values are represented by the special value NULL. To import data with NULL fields, check documentation of the SerDe used by the table. (The default Text Format uses LazySimpleSerDe which interprets the string \N as NULL when importing.)

Change Types

When hive.metastore.disallow.incompatible.col.type.changes is set to false, the types of columns in Metastore can be changed from any type to any other type. After such a type change, if the data can be shown correctly with the new type, the data will be displayed. Otherwise, the data will be displayed as NULL.

Allowed Implicit Conversions

	void	boolean	tinyint	smallint	int	bigint	float	double	decimal	string	varchar	timestamp	date	binary
void to	true	true	true	true	true	true	true	true	true	true	true	true	true	true
boolean to	false	true	false	false	false	false	false	false	false	false	false	false	false	false
tinyint to	false	false	true	true	true	true	true	true	true	true	true	false	false	false
smallint to	false	false	false	true	true	true	true	true	true	true	true	false	false	false
int to	false	false	false	false	true	true	true	true	true	true	true	false	false	false
bigint to	false	false	false	false	false	true	true	true	true	true	true	false	false	false
float to	false	false	false	false	false	false	true	true	true	true	true	false	false	false
double to	false	false	false	false	false	false	false	true	true	true	true	false	false	false
decimal to	false	false	false	false	false	false	false	false	true	true	true	false	false	false
string to	false	false	false	false	false	false	false	true	true	true	true	false	false	false
varchar to	false	false	false	false	false	false	false	true	true	true	true	false	false	false
timestamp to	false	false	false	false	false	false	false	false	false	true	true	true	false	false
date to	false	false	false	false	false	false	false	false	false	true	true	false	true	false
binary to	false	false	false	false	false	false	false	false	false	false	false	false	false	true