# Hive Decimal Precision/Scale Support

Xuefu Zhang

## 1. Introduction

HIVE-2693 introduced DECIMAL data type into Hive, which encapsulates Java BigDecimal type as the underlying data type. While this solves the accuracy problems associated with double and float, whose precisions are not good enough in certain applications, such as financial transactions, and lays important ground work for future enhancement, the feature is incomplete and non-standard in many ways, which prevents its wide adoption to a certain degree.

HIVE-3976 is to address the main issue with Decimal data type, the lack of capability of specifying precision and scale for a decimal column. The default precision of 38 and variable scale for any decimal column not only lacks a flexibility of specifying different precision and scale, but also can incur performance penalty in some scenarios. Moreover, having a variable scale is not desirable in many cases. For instance, applications (particularly native applications) such as SAS which need to pre-allocate memory require fixed types to do so efficiently.

In a sense, Hive current decimal data type is more of a java way rather than of DB way of handling precision-sensitive calculations.

HIVE-3976 tries to correct that, improving the current decimal data type to be aligned more to that of a Database. On the surface, it allows user to specify precision and scale parameters on the data type, but in essence, we are redefining the feature in a lot of vague areas to make it more usable in real life. In doing so, we may need to sacrifice backward compatibility. We assume that's acceptable, as decimal data type is still at its early stages with limited adoption and more importantly we are “standardizing” the feature to avoid bigger problems down the road.

This document tries to cover the overall functionality of decimal data type with precision and scale as type parameters. We try to follow standard as close as possible, and follow the implementation from other DB (such as mysql) when no standard is available.

Whenever backward compatibility is broken, we make a special note in this document.

## 2. Functionality

### *2.1 Syntactical and Semantical Changes*

We will enhance the hive grammar, such that for a decimal column, user can specify none, precision, or precision plus scale. Therefore, the following are all valid:

```
create table DEC_TABLE( c decimal );
```

```
create table DEC_TABLE( c decimal(10) );
create table DEC_TABLE( c decimal(5, 2) );
```

When decimal type is given without precision, the default precision is 10 (implementation specific, we follow mysql). When scale is missing from the type, the default is 0 (SQL standard). This is also in line with mysql.

Thus, the following types are equivalent:

```
decimal = decimal(10, 0)
decimal(5) = decimal(5, 0)
```

Backward Compatibility: currently decimal type has no parameters, which actually means a precision of 38 and a variable scale. Such behavior is changed.

The maximum precision and scale are implementation-specific. For Hive, we keep the current maximum precision 38. The maximum scale is also set to 38. This gives precision's range 1-38 and scale's range 0-38. Scale must be less than or equal to precision.

The maximum value of 38 for `precision` means that calculations on `decimal` values are accurate up to 38 digits.

For `decimal(`*p*`,`*s*`)`, the SQL standard requires a precision of at least *p* digits but permits more. In Hive, `decimal(`*p*`,`*s*`)` has a precision of exactly *p* digits for simplicity.

When Hive is unable to determine the precision and scale of a decimal type, such as in case of non-generic UDF that has an evaluate() method that returns decimal, a precision/scale of (38, 18) is assumed. However, user is encouraged to develop generic UDF instead, for which exact precision/scale may be specified.

Backward Compatibility: currently when Hive assumes a precision of 38 and a variable scale in such cases. With this change, a precision of 38 and a scale of 18 is assumed.

## *2.2 Arithmetic Rules*

SQL standard specified certain rules regarding the scale of the result of an operator with at least one of the operant as decimal. Specifically, for +/-, the result scale is max(s1, s2), and for multiplication, s1+s2. The scale of the result of a division is implementation specific. The precision for all operators is implementation specific.

We will stick to the standard for scale. For the scale of a division, and the precision of all operators, since there is no standard, we can follow what other DB is doing. I found mysql document about this is

vague. But one thing is clear, for scale resulting from a division, the scale of the result is s1 plus a system-wide increment, which has a default 4. However, it's available in doc for MS SQL Server:

| Operation | Result precision | Result scale |
|---|---|---|
| e1 + e2 | max(s1, s2) + max(p1-s1, p2-s2) + 1 | max(s1, s2) |
| e1 - e2 | max(s1, s2) + max(p1-s1, p2-s2) + 1 | max(s1, s2) |
| e1 * e2 | p1 + p2 + 1 | s1 + s2 |
| e1 / e2 | p1 - s1 + s2 + max(6, s1 + p2 + 1) | max(6, s1 + p2 + 1) |
| e1 % e2 | min(p1-s1, p2 -s2) + max( s1,s2 ) | max(s1, s2) |

- We can add rules for other operations as we go.
- SQL standard says that if the resulting scale of a multiply operation goes beyond the maximum (38 in case of Hive), rounding is not acceptable and thus an error occurs. In this case, null value is put in place, according to the current Hive error handling (see below).

We will follow MS SQL implementation unless we are able to find out that for mysql.

## *2.3 Error Handling*

SQL standard doesn't specify the behavior in case of Out-of-Range, Overflow, and divided-by-zero. The possible choices are raising error (discarding the data), rounding or truncating, and putting null. Some DBs such as mysql have system settings controlling the behavior.

Hive should also be able to define different modes in which error are handled differently. The current mode, in which Hive sets null value for data overflow, out-of-range, or divide-by-zero, can be thought as the default mode. In the future, we can introduce other modes, and have settings available to allow user to pick.

With precision and scale introduced, “error conditions” get more interesting. For instance, assigning a decimal value 49.3241 to a column of decimal(4, 2) can be completely fine, even though precision of 49.3241 is greater than 4. This rounding should not be treated as out-or-range or overflow. Instead, it's should be completely valid and is not an error condition, regardless of the server mode. In this case, 49.3241 can be rounded to 49.32, which fits into decimal(4,2). On the other hand, if the value were 123.9, then we have to put null for the column. This concludes the following rule regarding decimal rounding rule:

A decimal value v with precision p1 and scale s1 can be rounded to fit a column with type decimal(p, s) if p1 - s1 is less than or equal to p – s. This basically says if the integer part of the decimal can be fitted to the column, then rounding by removing trailing decimal digits is valid.

The rounding method is HALF UP, which should satisfy most of in not all use case.

The rounding behavior described here is in line with mysql.

Backward Compatibility: rounding instead of setting null value is used to handle cases where the precision of a value to be cast or assigned to a column if the integer part of the value can be preserved. Previously, if the value's precision is greater than 38 (the only allowed precision for decimal type), null value is set.

Other than this, we will keep the existing behavior and null value will be placed in case of errors. Of course, this can be changed. For example, we can introduce system settings to control the behavior, providing flexibility. However, it's not essential for supporting decimal precision and scale.

### *2.4 UDFs*

HIVE-4844 has laid some ground work on propagating type parameters from a UDF's input parameters to its return type. For GenericUDFs, the returned object inspector of the UDF can encapsulate the type parameters. However, for UDFs, there is no way to propagate the type parameters because java reflection is used to determine which method to be used to evaluate the expression. For this, only default type parameters is available for the result of the UDF.

Unfortunately, all arithmetic operations in Hive are implemented using UDF. In order to propagate type parameters from the input parameters of a UDF to the returned parameter of the UDF, these “old” UDFs, such as UDFOPAdd, UDFOPDivide, and so on, needs to be replaced with GenericUDF implementation.

In addition, UDF developers will need to be aware that GenericUDFs are preferred for UDFs that process types that come with parameters (decimals, char, and varchar).

## 3. Summary

It can be seen that the main objective is to provide user more DB-styled decimal handling with flexibility by specifying a precision and and a fixed scale. While additional changes might be found during implementation, we expect to keep a lot of existing behaviors such as decimal encoding and conversion unless there is a need. If such needs arrive, the document will be updated.

HIVE-4844 has laid important ground work for type parameter handling, while this document puts the functionality in clarity.