Shared publicly

Discuss: Follow ANSI SQL on table insertion

Author: Gengliang Wang

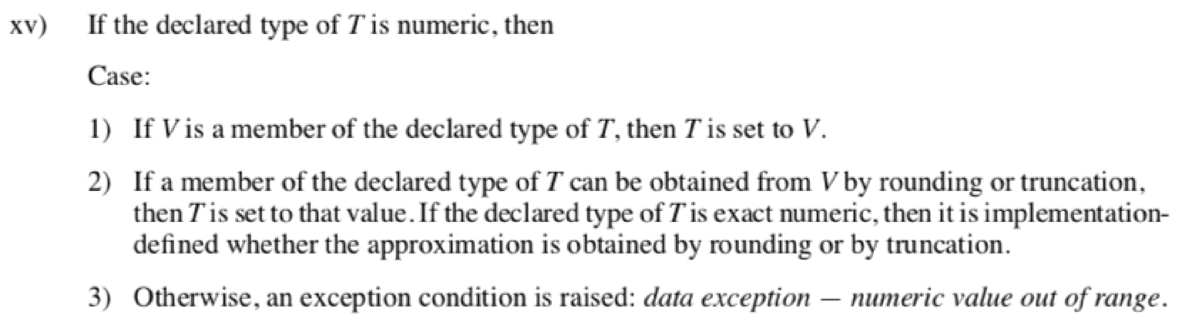
Background and Motivation

In Spark version 2.4 and earlier, when inserting into a table, Spark will cast the data type of input query to the data type of target table by coercion. This can be super confusing, e.g. users make a mistake and write string values to an int column.

In data source V2, by default, only UpCast is allowed when inserting data into a table. E.g. int -> long and int -> string are allowed, while decimal -> double or long -> int are not allowed. You can read Cast.canUpCast for more details. This prevents all possible data truncation or precision loss in table insertion.

However, the rules of UpCast was originally created for Dataset encoder. They are quite strict and there is no existing popular DBMS has the same behavior. E.g. UpCast disallows writing Long type to Int type, while MySQL/PostgreSQL/Oracle allows it (there will be runtime exception on if the value is out-of-range for the target type).

As I look up the ANSI SQL, in section 9.2 "Store assignment"



xv) If the declared type of *T* is numeric, then

Case:

1) If *V* is a member of the declared type of *T*, then *T* is set to *V*.

2) If a member of the declared type of *T* can be obtained from *V* by rounding or truncation, then *T* is set to that value. If the declared type of *T* is exact numeric, then it is implementation-defined whether the approximation is obtained by rounding or by truncation.

3) Otherwise, an exception condition is raised: *data exception — numeric value out of range*.

So rounding and truncation are allowed in table insertion as per ANSI SQL. Using UpCast for store assignment is breaking change. **It is possible that it would break existing queries after 3.0 releases.**

There is also one minor issue when with UpCast table insertion, in the following query:

```
create table t (d double);
insert into t values (1.0);
```

The value 1.0 is considered as a decimal in SQL parser, so that there is no loss of precision when cast to float/double/decimal columns.

But casting decimal to double is not strictly safe. So, if we only allow safe casting, the query above is not supported. We can work around by allowing such conversion for literals. Still, the internal inconsistency might confuse users:

- Should we allow inserting double column in the query "INSERT INTO t SELECT float_col + 1.1"?
- The same insert with a decimal column as input will fail even when a decimal literal would succeed
- Similar insert queries with "literal" inputs can be constructed through layers of indirection via views, inline views, CTEs, etc.

Goals

Propose a new table insertion behavior, so that

1. Unreasonable type coercions in data source V1 should be disallowed, e.g. string -> int, array -> string.
2. It is explainable when users find their applications broken after upgrading to 3.0.

Target Personas

- **Developers**
- **Data engineers**

Proposal

This proposal is to follow ANSI SQL store assignment rules in table insertion, instead of using UpCast.

In details, we can refer to the assignment casting rules of PostgreSQL (summarised by [Takeshi Yamamuro](#)). The rules are consistent with the ANSI SQL.
https://docs.google.com/spreadsheets/d/1I4pl1TIe8u0G4sGSc7OCItySfvK0pOTkU4IPw5Z0Qck/edit?usp=sharing

I have already created a PR for this: https://github.com/apache/spark/pull/25239
Two significant differences from Up-Cast (also allowed in PostgreSQL/Mysql/Oracle):

1. **Any numeric type can be assigned to another numeric type.**
2. **TimestampType can be assigned DateType.**

On casting failure, NULL result will be returned. It is by design since we don't want a long-running job aborted by some casting failure. But there are scenarios that users want to make sure all the data conversion are not out-of-range, like the way in

MySQL/PostgreSQL/Oracle. As a follow-up, we can have an optional mode: throw runtime exceptions on casting failures during table insertion. See https://issues.apache.org/jira/browse/SPARK-28512 for more details.