Cloud Spanner has a variety of column data types but does not have a data type for arbitrary precision numbers (https://en.wikipedia.org/wiki/Arbitrary-precision_arithmetic). When you need to store arbitrary precision numbers in Cloud Spanner, we recommend that you store them as strings.

In many databases, arbitrary precision numbers are expressed as `NUMERIC(p,s)`, where `p` is the decimal precision (total number of digits), and `s` is the scale (number of digits after the decimal point). For example, `NUMERIC(8,3)` can represent -99999.999 to +99999.999.

Cloud Spanner does not have a data type for arbitrary precision numbers. Cloud Spanner has two numeric types: `INT64` and `FLOAT64`. Neither of these types has the ideal precision for financial, scientific, or engineering calculations, where a precision of 30 digits or more is commonly required. `FLOAT64` provides 15 digits of precision, with a scale of -294 to +294, and `INT64` provides 18 digits of precision, with a scale of zero. For more details on how floating point precision is calculated, see Double-precision floating-point format (https://en.wikipedia.org/wiki/Double-precision_floating-point_format).

Arbitrary precision numeric types normally store values by scaling them and storing them as integers. For example, for a numeric type with a precision of 5 and scale of 2, the value 0.3 is scaled by a factor of 100 (10^2) and stored as `00030`. Storing values this way avoids decimal fraction errors and loss of precision.

In contrast, Cloud Spanner stores `FLOAT64` values as binary numbers. The IEEE 64-bit floating point (https://ieeexplore.ieee.org/document/4610935) binary representation that Cloud Spanner uses cannot precisely represent decimal fractions. This loss of precision introduces rounding errors for some decimal fractions.

For example, when you store the decimal value 0.2 using the `FLOAT64` data type, the binary representation converts back to a decimal value of 0.20000000000000001 (to 18 digits of precision). Similarly (1.4 * 165) converts back to 230.999999999999971 and (0.1 + 0.2) converts back to 0.30000000000000004. This is why 64-bit floats are described as only having 15 digits of precision.

When you need to store an arbitrary precision number in a Cloud Spanner database, and you need more precision than `FLOAT64` provides, we recommend that you store the value as its decimal representation in a `STRING` column. For example, the number `123.4` is stored as the string `"123.4"`.

With this approach, your application must perform a lossless conversion between the application-internal representation of the number and the `STRING` column value for database reads and writes.

Most arbitrary precision libraries have built-in methods to perform this lossless conversion. In Java, for example, you can use the `BigDecimal.toPlainString()` (https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/math/BigDecimal.html#toPlainString()) method and the `BigDecimal(String)` (https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/math/BigDecimal.html#%3Cinit%3E(java.lang.String)) constructor.

Storing the number as a string has the advantage that the value is stored with exact precision (up to the `STRING` column length limit), and the value remains human-readable.

To perform *exact* aggregations and calculations on string representations of arbitrary precision numbers, your application must perform these calculations. You cannot use SQL aggregate functions.

For example, to perform the equivalent of a SQL `SUM(value)` over a range of rows, the application must query the string values for the rows, then convert and sum them internally in the app.

You can use SQL queries to perform *approximate* aggregate calculations by casting the values to `FLOAT64`:

Similarly, you can sort by numeric value or limit values by range with casting:

These calculations are approximate to the limits of the `FLOAT64` data type.

There are other ways to store arbitrary precision numbers in Cloud Spanner. If storing arbitrary precision numbers as strings does not work for your application, consider the following alternatives:

To store arbitrary precision numbers, you can pre-scale the values before writing, so that numbers are always stored as integers, and re-scale the values after reading. Your application stores a fixed scale factor, and the precision is limited to the 18 digits provided by the `INT64` data type.

Take, for example, a number that needs to be be stored with an accuracy of 5 decimal places. The application converts the value to an integer by multiplying it by 100,000 (shifting the decimal point 5 places to the right), so the value 12.54321 is stored as `1254321`.

In monetary terms, this approach is like storing dollar values as multiples of milli-cents, similar to storing time units as milliseconds.

The application determines the fixed scaling factor. If you change the scaling factor, you must convert all of the previously scaled values in your database.

This approach stores values that are human-readable (assuming you know the scaling factor). Also, you can use SQL queries to perform calculations directly on values stored in the database, as long as the result is scaled correctly and does not overflow.

You can also store arbitrary precision numbers in Cloud Spanner using two elements:

- The unscaled integer value stored in a byte array.
- An integer that specifies the scaling factor.

First your application converts the arbitrary precision decimal into an unscaled integer value. For example, the application converts `12.54321` to `1254321`. The scale for this example is `5`.

Then the application converts the unscaled integer value into a byte array using a standard portable binary representation (for example, big-endian two's complement (https://en.wikipedia.org/wiki/Two%27s_complement)).

The database then stores the byte array (`BYTES`) and integer scale (`INT64`) in two separate columns, and converts them back on read.

In Java, you can use `BigDecimal` (https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/math/BigDecimal.html) and `BigInteger` (https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/math/BigInteger.html) to perform these calculations:

You can read back to a Java `BigDecimal` using the following code:

This approach stores values with arbitrary precision and a portable representation, but the values are not human-readable in the database, and all calculations must be performed by the application.

Another option is to serialize the arbitrary precision decimal values to byte arrays using the application's internal representation, then store them directly in the database.

The stored database values are not human-readable, and the application needs to perform all calculations.

This approach has portability issues. If you try to read the values with a programming language or library different from the one that originally wrote it, it might not work. Reading the values back might not work because different arbitrary precision libraries can have different serialized representations for byte arrays.

- Read about other data types (/spanner/docs/data-types#numeric-types) available for Cloud Spanner.
- Learn how to correctly set up a Cloud Spanner schema design and data model (/spanner/docs/schema-and-data-model).
- Learn about optimizing your schema design for Cloud Spanner (/spanner/docs/whitepapers/optimizing-schema-design).