

Use float or decimal for accounting application dollar amount?

Asked 16 years, 3 months ago Modified 3 years, 4 months ago

Viewed 49k times



77



We are rewriting our legacy [accounting system](#) in VB.NET and SQL Server. We brought in a new team of .NET/ SQL Programmers to do the rewrite. Most of the system is already completed with the dollar amounts using floats. The legacy system language, I programmed in, did not have a float, so I probably would have used a decimal.

What is your recommendation?

Should the float or decimal data type be used for dollar amounts?

What are some of the pros and cons for either?

One *con* mentioned in our [daily scrum](#) was you have to be careful when you calculate an amount that returns a result that is over two decimal positions. It sounds like you will have to round the amount to two decimal positions.

Another *con* is all displays and printed amounts have to have a *format statement* that shows two decimal positions. I noticed a few times where this was not done

and the amounts did not look correct. (i.e. 10.2 or 10.2546)

A *pro* is the float-only approach takes up eight bytes on disk where the decimal would take up nine bytes (decimal 12,2).

sql-server

vb.net

database-design

currency

accounting

Share

Improve this question

Follow

edited Jul 27, 2021 at 16:18



Peter Mortensen

31.6k ● 22 ● 109 ● 133

asked Sep 15, 2008 at 4:55



Gerhard Weiss

9,751 ● 19 ● 68 ● 70

7 Go back and get rid of your floats. – [Loren Pechtel](#) Oct 23, 2010 at 20:26

Actual banking and settlement systems deployed today often use binary floating point with a built in scale. So 1 dollar might be represented as 100000 float double. The idea there is that the systems gain performance from native float support on the cpu, and that more math operations are available than with decimal. The con is the devs have to know what they are doing . – [Frank](#) Dec 6, 2022 at 9:51

24 Answers

Sorted by:

Highest score (default)





112

Should Float or Decimal data type be used for dollar amounts?



The answer is easy. Never floats. *NEVER!*



Floats were according to [IEEE 754](#) always binary, only the new standard [IEEE 754R](#) defined decimal formats. Many of the fractional binary parts can never equal the exact decimal representation.

Any binary number can be written as $m/2^n$ (m , n positive integers), any decimal number as $m/(2^n \cdot 5^n)$. As binaries lack the prime factor 5, all binary numbers can be exactly represented by decimals, but not vice versa.

$$0.3 = 3/(2^1 * 5^1) = 0.3$$

$$0.3 = [0.25/0.5] [0.25/0.375] [0.25/3.125] [0.2825/3.125]$$

$$1/4$$

$$1/8$$

$$1/16$$

$$1/32$$

So you end up with a number either higher or lower than the given decimal number. Always.

Why does that matter? Rounding.

Normal rounding means 0..4 down, 5..9 up. So it *does* matter if the result is either $0.049999999999 \dots$ or $0.0500000000 \dots$ You may know that it means 5 cent, but

the the computer does not know that and rounds

0.4999 ... down (wrong) and 0.5000 ... up (right).

Given that the result of floating point computations always contain small error terms, the decision is pure luck. It gets hopeless if you want decimal round-to-even handling with binary numbers.

Unconvinced? You insist that in your account system everything is perfectly ok? Assets and liabilities equal? Ok, then take each of the given formatted numbers of each entry, parse them and sum them with an independent decimal system!

Compare that with the formatted sum. Oops, there is something wrong, isn't it?

For that calculation, extreme accuracy and fidelity was required (we used Oracle's FLOAT) so we could record the "billionth's of a penny" being accrued.

It doesn't help against this error. Because all people automatically assume that the computer sums right, and practically no one checks independently.

Share Improve this answer

Follow

edited Jul 27, 2021 at 16:20



Peter Mortensen

31.6k ● 22 ● 109 ● 133

answered Sep 15, 2008 at 20:35



TSK

1,144 ● 1 ● 7 ● 3

But do make sure to use at least 4 decimal places in the decimal field if you want to do calculations on it especially division. – [HLGEM](#) Feb 9, 2009 at 16:49

1 And make sure that you know that (by default) \$0.045 rounds to \$0.04 and \$0.055 rounds to \$0.06 – [Keith](#) Feb 10, 2009 at 15:28

8 For those unsure by what Keith means, Decimal types use a different kind of rounding. It seems to be commonly called "bankers' rounding" but Wikipedia has a number of alternative names: round half to even, unbiased rounding, convergent rounding, statistician's rounding, Dutch rounding, Gaussian rounding, or bankers' rounding (en.wikipedia.org/wiki/...). – [patridge](#) Feb 19, 2010 at 17:39

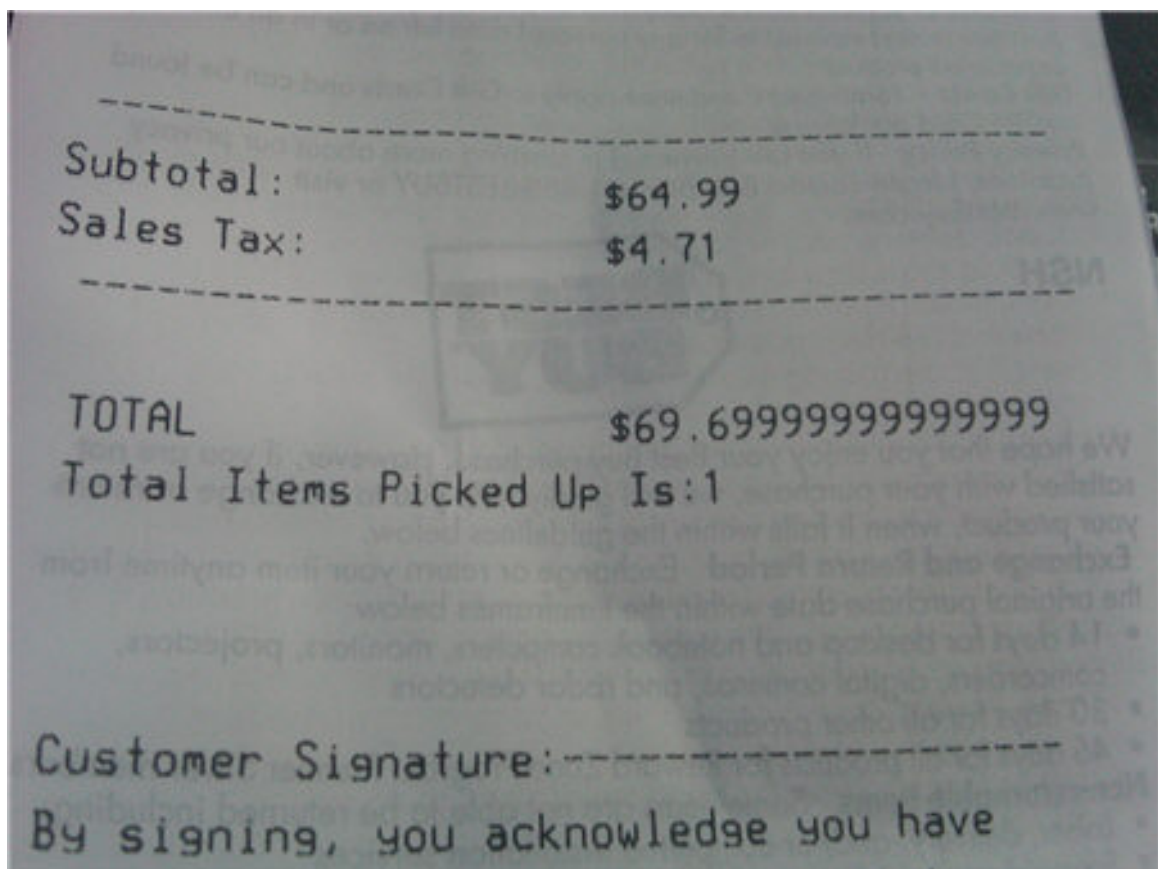
2 Another thing to keep in mind is that Decimal.Round and String.Format give different results:
Decimal.Round(0.045M,2) = 0.04 and String.Format("{0:0.00}",0.045M) = 0.05 – [Jason Massey](#) Aug 27, 2012 at 21:36 ✎



This photo answers:

47





This is another situation: [man from Northampton got a letter stating his home would be seized if he didn't pay up zero dollars and zero cents!](#)

ments have not been made. The total amount now

	\$109.60
	\$1.64
	\$0.00
	\$0.00
	<u>(\$111.24)</u>
DUE:	\$0.00

before February 4, 2011 BAC Home Loans Service

Share Improve this answer

edited Feb 20, 2015 at 6:08

Follow

answered Oct 23, 2010 at 0:04



Nakilon

35k ● 16 ● 111 ● 146

3 This made me laugh. Way to go, Best Buy.
– [LittleBobbyTables - Au Revoir](#) May 26, 2011 at 14:36

19 I got a bill for \$0.01 from a phone company every month for year. So I paid them \$0.02 online, then got a bill for -\$0.01 for six months, then it stopped. – [Neil McGuigan](#) Jul 4, 2012 at 5:18

Well, there will be plenty of maintenance jobs to clean up this mess. – [Peter Mortensen](#) Jul 29, 2021 at 22:55



23



First you should read [What Every Computer Scientist Should Know About Floating Point Arithmetic](#). Then you should really consider using some type of [fixed point / arbitrary-precision number](#) package (e.g., Java BigDecimal or Python decimal module). Otherwise, you'll be in for a world of hurt. Then figure out if using the native SQL decimal type is enough.

Floats and doubles exist(ed) to expose the fast [x87 floating-point coprocessor](#) that is now pretty much obsolete. Don't use them if you care about the accuracy of the computations and/or don't fully compensate for their limitations.

Share Improve this answer

Follow

edited Jul 29, 2021 at 22:12



Peter Mortensen

31.6k ● 22 ● 109 ● 133

answered Sep 15, 2008 at 8:04



Rich Schuler

42k ● 6 ● 74 ● 59

-
- 2 While learning more about floating point is useful, using the decimal type in C# is akin to using a fixed point / arbitrary-precision number package as you suggest, built-in to the language. See msdn.microsoft.com/en-us/library/system.decimal.aspx for an explanation of how decimal stores precise powers of 10 with decimals instead of powers of 2 for the decimal component (it's basically an int with a decimal placement component). – Chris Moschini Jun 3, 2012 at 2:25
-
- 2 "to expose the fast x87 fp that is now pretty much obsolete", that's simply not true floating point numbers are still one of the most used datatypes on computers, e.g. simulations, games, signals processing... – markmn1 Sep 9, 2015 at 7:26
-



10



Just as an additional warning, SQL Server and the .NET framework use a different default algorithm for rounding. Make sure you check out the MidPointRounding parameter in Math.Round(). .NET framework uses [bankers' rounding](#) by default and SQL Server uses Symmetric Algorithmic Rounding. Check out the Wikipedia article [here](#).

Share Improve this answer

Follow

edited Jul 29, 2021 at 22:31



Peter Mortensen

31.6k ● 22 ● 109 ● 133

answered Sep 15, 2008 at 13:21



Darrel Miller

What name does "Symmetric Algorithmic Rounding" have in the Wikipedia article? Or is it not covered there? What is "Symmetric Algorithmic Rounding"? Can you add a reference? – [Peter Mortensen](#) Jul 29, 2021 at 22:34



7



Ask your accountants! They will frown upon you for using float. Like [David Singer said](#), use float *only* if you don't care for accuracy. Although I would always be against it when it comes to money.



In accounting software is *not* acceptable a float. Use decimal with four decimal points.

Share Improve this answer

Follow

edited Jul 29, 2021 at 22:07



[Peter Mortensen](#)

31.6k ● 22 ● 109 ● 133

answered Sep 15, 2008 at 5:22



Ricardo C



6



A bit of background here....

No number system can handle all real numbers accurately. All have their limitations, and this includes both the standard IEEE floating point and signed decimal. The IEEE floating point is more accurate per bit used, but that doesn't matter here.





Financial numbers are based on centuries of paper-and-pen practice, with associated conventions. They are reasonably accurate, but, more importantly, they're reproducible. Two accountants working with various numbers and rates should come up with the same number. Any room for discrepancy is room for fraud.

Therefore, for financial calculations, the right answer is whatever gives the same answer as a CPA who's good at arithmetic. This is decimal arithmetic, not IEEE floating point.

Share Improve this answer

answered Feb 9, 2009 at 16:42

Follow



[David Thornley](#)

57k ● 9 ● 95 ● 158

I feel this answer makes more sense in general. I read several similar questions and answers, talking about accuracy, rounding etc. However, I still feel weird about those answers and something is missing. The word “reproducible” seems to be the key here – [Lujun Weng](#) Feb 7, 2022 at 13:07





6



Floating points have unexpected irrational numbers.

For instance you can't store $1/3$ as a decimal, it would be 0.3333333333... (and so on)

Floats are actually stored as a binary value and a power of 2 exponent.

So 1.5 is stored as 3×2 to the -1 (or $3/2$)

Using these base-2 exponents create some odd irrational numbers, for instance:

Convert 1.1 to a float and then convert it back again, your result will be something like: 1.09999999999989

This is because the binary representation of 1.1 is actually $154811237190861 \times 2^{-47}$, more than a double can handle.

More about this issue on [my blog](#), but basically, for storage, you're better off with decimals.

On Microsoft SQL server you have the `money` data type - this is usually best for financial storage. It is accurate to 4 decimal positions.

For calculations you have more of a problem - the inaccuracy is a tiny fraction, but put it into a power function and it quickly becomes significant.

However decimals aren't very good for any sort of maths - there's no native support for decimal powers, for instance.

answered Sep 15, 2008 at 9:59



Keith

155k ● 82 ● 306 ● 446

-
- 5 "irrational" isn't the word you're looking for. $1/3$ is still rational, but it doesn't have a finite binary representation...
– [Brian Postow](#) Feb 9, 2009 at 17:03

Yeah, I know - I'm just not sure what else to call it: a number that can't be represented is a bit too wordy. – [Keith](#) Feb 10, 2009 at 15:25

They are approximations, but then numbers that could be represented could be approximated too. An actual irrational number is one that cannot be represented by any integer fraction, regardless of base. These are numbers that can be represented in base 10, but can't in base 2. – [Keith](#) May 18, 2009 at 11:54

A number with a non-terminating decimal representation – that *is* too wordy! – [Kenny Evitt](#) May 3, 2011 at 14:07

-
- 1 Perhaps you could say that floating point numbers typically store unexpected, and irrelevant, fractional values.
– [Kenny Evitt](#) May 3, 2011 at 14:08
-



5

I'd recommend using 64-bit integers that store the whole thing in cents.



Follow



Peter Mortensen

31.6k ● 22 ● 109 ● 133



answered Nov 3, 2008 at 1:08



Joshua

43.1k ● 9 ● 78 ● 148

With the obvious caveat that partial-cent values (ie. \$0.015) cannot be represented at all. A reasonable limitation for most apps. – [rocketmonkeys](#) Dec 6, 2010 at 21:59

Simple solution: Store it in thousands of cents.. I store the stuff in millionths of the currency in question.. – [Marenz](#) Aug 18, 2011 at 11:58

- 1 Check your overflow. Millionths of cents overflows at just over 20 billion dollars. Thousandths of cents at 20 trillion (which may or may not be acceptable), while cents is 20 quadrillion (which I deem safe). – [Joshua](#) Aug 18, 2011 at 15:05
-

@Marenz: At any given stage of calculation, it should often be possible to define a minimum-sized unit upon which the calculation will be performed, and have no round-off errors of any magnitude occur at any points other than when things are explicitly rounded. If one buys five thousand of something at 3 for \$1, the total price should typically be \$1666.67 (5000/3, rounded to the penny), rather than \$1666.66667 (5000/3, rounded to 1/1000 penny) the or \$1666.65 (0.33333 times 5000). – [supercat](#) Jun 4, 2012 at 22:03

Cents? No [pennies](#), then? – [Peter Mortensen](#) Jul 29, 2021 at 22:41



Use SQL Server's **decimal** type.

Do not use *money* or *float*.

5



money uses four decimal places and is faster than using decimal, **but** suffers from some obvious and some not so obvious problems with rounding ([see this connect issue](#)).



Share Improve this answer

edited Jul 30, 2021 at 2:11



Follow

answered Nov 3, 2008 at 1:00



Mitch Wheat

300k ● 44 ● 477 ● 550

See @David Thornley's answer. It *may* be that the **money** type most closely reproduces accounting conventions, however (in)approximate they are. – [Roy Tinker](#) Feb 7, 2012 at 1:15



The only reason to use Float for money is if you don't care about accurate answers.

4



Share Improve this answer

answered Sep 15, 2008 at 5:00

Follow



David Singer

1,182 ● 1 ● 10 ● 11



4

Floats are not exact representations, precision issues are possible, for example when adding very large and very small values. That's why decimal types are recommended



for currency, even though the precision issue may be sufficiently rare.



To clarify, the decimal 12,2 type will store those 14 digits exactly, whereas the float will not as it uses a binary representation internally. For example, 0.01 cannot be represented exactly by a floating point number - the closest representation is actually 0.0099999998

Share Improve this answer

edited Sep 15, 2008 at 5:10

Follow

answered Sep 15, 2008 at 4:58



Niall

5,121 ● 1 ● 22 ● 12

Decimals are not exact either, unless they are infinite precision. – [1800 INFORMATION](#) Sep 15, 2008 at 5:00

0.1 can be stored exactly in a Decimal field. Decimals are not exact *for every number*, but are exact for *most* (some?) common monetary amounts. Sometimes. – [rocketmonkeys](#) Dec 6, 2010 at 21:59



4

For a banking system I helped develop, I was responsible for the "interest accrual" part of the system. Each day, my code calculated how much interest had been accrued (earnt) on the balance that day.



For that calculation, extreme accuracy and fidelity was required (we used Oracle's FLOAT) so we could record



the "billionth's of a penny" being accrued.

When it came to "capitalising" the interest (ie. paying the interest back into your account) the amount was rounded to the penny. The data type for the account balances was two decimal places. (In fact it was more complicated as it was a multi-currency system that could work in many decimal places - but we always rounded to the "penny" of that currency). Yes - there were "fractions" of loss and gain, but when the computer's figures were actualised (money paid out or paid in) it was always REAL money values.

This satisfied the accountants, auditors and testers.

So, check with your customers. They will tell you their banking/accounting rules and practices.

Share Improve this answer

answered Sep 15, 2008 at 10:42

Follow




Guy

9,826 ● 7 ● 39 ● 43

-
- 1 Billionths of a penny is 0.01×10^{-9} - there is absolutely no reason to use Oracle's FLOAT here for "extreme accuracy and fidelity", since it is a floating-point representation, which is an approximate number rather than an exact number. TSQL's DECIMAL(38,18) would be more accurate. Without you explaining how you handled multi-currency, I am skeptical you are error-free. If the testers were converting from the Euro to the Zimbabwe dollar, they might see a real rounding problem. – [John Zabroski](#) Aug 15, 2014 at 20:11

Just to clarify, I used floats for the interest accrual process. Decimals were used for the actual transactions (when the

accrued interest was paid out). At the time the system was single currency. If I had my time again, I probably would have not used floats. :) – [Guy](#) Aug 18, 2014 at 10:16 

[Bankers' rounding?](#) – [Peter Mortensen](#) Jul 29, 2021 at 22:20



3



Even better than using decimals is using just plain old integers (or maybe some kind of bigint). This way you always have the highest accuracy possible, but the precision can be specified. For example the number `100` could mean `1.00`, which is formatted like this:



```
int cents = num % 100;
int dollars = (num - cents) / 100;
printf("%d.%02d", dollars, cents);
```

If you like to have more precision, you can change the 100 to a bigger value, like: 10^n , where n is the number of decimals.

Share Improve this answer

edited Sep 16, 2008 at 10:08

Follow

answered Sep 16, 2008 at 9:58



[Peter Stuijzand](#)

5,084 ● 1 ● 25 ● 28

-
- 4 You should do this if you don't have a good fixed point type. The upside is that you get to determine where the decimal is, the downside is that you're going to screw it up. If you can get a fixed-point type, you won't have to worry about it.

– [Michael Kohne](#) Feb 9, 2009 at 16:49

That is already two magic numbers, presumably the same.

– [Peter Mortensen](#) Jul 29, 2021 at 22:35



3



Another thing you should be aware of in accounting systems is that no one should have direct access to the tables. This means all access to the accounting system must be through [stored procedures](#).



This is to prevent fraud, not just [SQL injection](#) attacks. An internal user who wants to commit fraud should not have the ability to directly change data in the database tables, ever. This is a critical internal control on your system.

Do you really want some disgruntled employee to go to the backend of your database and have it start writing them checks? Or hide that they approved an expense to an unauthorized vendor when they don't have approval authority? Only two people in your whole organization should be able to directly access data in your financial database, your database administrator (DBA) and his backup. If you have many DBAs, only two of them should have this access.

I mention this because if your programmers used float in an accounting system, likely they are completely unfamiliar with the idea of internal controls and did not consider them in their programming effort.

Share Improve this answer

Follow

edited Jul 29, 2021 at 22:50



Peter Mortensen

31.6k ● 22 ● 109 ● 133

answered Feb 9, 2009 at 16:55



HLGEM

96.4k ● 15 ● 119 ● 189



2



I had been using SQL's money type for storing monetary values. Recently, I've had to work with a number of online payment systems and have noticed that some of them use integers for storing monetary values. In my current and new projects I've started using integers and I'm pretty content with this solution.



Share Improve this answer

Follow

answered Feb 9, 2009 at 14:09



George

161 ● 1 ● 2

I am assuming you are using the ROUND verb in your procedures? – [Gerhard Weiss](#) Feb 10, 2009 at 2:41

- 1 If you mean on the SQL side then NO. I prefer the DAL to return the integer as is in the DB. It's in the Business Logic Layer that I do the transformation. $\text{int cents} = \text{value} \% 100$; $\text{int dollars} = (\text{value} - \text{cents}) / 100$; With .NET 3.5 I have an extension method for that. – [George](#) Feb 11, 2009 at 12:47

@Gerhard Weiss: It sounds like a rhetorical question. Is it?

– [Peter Mortensen](#) Jul 29, 2021 at 22:43



2



Out of the 100 fractions $n/100$, where n is a natural number such that $0 \leq n$ and $n < 100$, only four can be represented as floating point numbers. Take a look at the output of this C program:

```
#include <stdio.h>

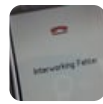
int main()
{
    printf("Mapping 100 numbers between 0 and 1 ");
    printf("to their hexadecimal exponential form (HEF ");
    printf("Most of them do not equal their HEFs. That ");
    printf("that their representations as floats ");
    printf("differ from their actual values.\n");
    double f = 0.01;
    int i;
    for (i = 0; i < 100; i++) {
        printf("%1.2f -> %a\n", f*i, f*i);
    }
    printf("Printing 128 'float-compatible' numbers ")
    printf("together with their HEFs for comparison.\n");
    f = 0x1p-7; // ==0.0071825
    for (i = 0; i < 0x80; i++) {
        printf("%1.7f -> %a\n", f*i, f*i);
    }
    return 0;
}
```

Share Improve this answer

[edited Oct 23, 2010 at 20:17](#)

Follow

[answered Oct 21, 2010 at 20:19](#)



Lars Bohl

1,011 ● 1 ● 16 ● 20

For the sake of it, I copied the above code and ran it in codepad. codepad.org/03hAQZwg This includes the output.

– Dominic K Oct 23, 2010 at 20:22 ✎



2



You can always write something like a Money type for .NET.

Take a look at this article: [A Money type for the CLR](#). The author did an excellent work in my opinion.



Share Improve this answer

edited Jul 29, 2021 at 22:36



Follow



Peter Mortensen

31.6k ● 22 ● 109 ● 133

answered Sep 21, 2008 at 6:12



Tomer Pintel

1,587 ● 1 ● 14 ● 15



1



Whatever you do, you need to be careful of rounding errors. Calculate using a greater degree of precision than you display in.

Share Improve this answer

answered Sep 15, 2008 at 4:59

Follow



1800 INFORMATION

135k ● 30 ● 163 ● 242



1



Have you considered using the money-data type to store dollar-amounts?

Regarding the con that decimal takes up one more byte, I would say don't care about it. In 1 million rows you will only use 1 more MB and storage is very cheap these days.

Share Improve this answer

Follow

edited Jul 27, 2021 at 16:21



Peter Mortensen

31.6k ● 22 ● 109 ● 133

answered Sep 15, 2008 at 4:59



Espo

41.9k ● 21 ● 136 ● 161

1 Do not use the money datatype. (It's a hangover from SyBase.) – [Mitch Wheat](#) Nov 3, 2008 at 1:00



1



You will probably want to use some form of fixed point representation for currency values. You will also want to investigate [banker's rounding](#) (also known as "round half to even"). It avoids bias that exist in the usual "round half up" method.

Share Improve this answer

Follow

edited Jul 29, 2021 at 22:22



Peter Mortensen

31.6k ● 22 ● 109 ● 133

answered Sep 15, 2008 at 13:19



user6931

11 ● 3



Always use Decimal. Float will give you inaccurate values due to rounding issues.

0



Share Improve this answer

answered Sep 15, 2008 at 12:50

Follow



Roel Vlemmings





0

Floating point numbers can *only* represent numbers that are a sum of negative multiples of the base - for binary floating point, of course, that's two.



There are only four decimal fractions representable precisely in binary floating point: 0, 0.25, 0.5 and 0.75.



Everything else is an approximation, in the same way that 0.3333... is an approximation for 1/3 in decimal arithmetic.



Floating point is a good choice for computations where the scale of the result is what is important. It's a bad choice where you're trying to be accurate to some number of decimal places.

Share Improve this answer

answered Sep 15, 2008 at 12:55

Follow



Mike Dimmick

9,792 ● 2 ● 26 ● 48



0

This is an excellent article describing [when to use float and decimal](#). Float stores an approximate value and decimal stores an exact value.



In summary, exact values like money should use decimal, and approximate values like scientific measurements should use float.



Here is an interesting example that shows that both float and decimal are capable of losing precision. When adding a number that is not an integer and then

subtracting that same number float results in losing precision while decimal does not:

```
DECLARE @Float1 float, @Float2 float, @Float3 float
SET @Float1 = 54;
SET @Float2 = 3.1;
SET @Float3 = 0 + @Float1 + @Float2;
SELECT @Float3 - @Float1 - @Float2 AS "Should be 0"
```

Should be 0

1.13797860024079E-15

When multiplying a non integer and dividing by that same number, decimals lose precision while floats do not.

```
DECLARE @Fixed1 decimal(8,4), @Fixed2 decimal(8,4), @Fixed3 decimal(8,4)
SET @Fixed1 = 54;
SET @Fixed2 = 0.03;
SET @Fixed3 = 1 * @Fixed1 / @Fixed2;
SELECT @Fixed3 / @Fixed1 * @Fixed2 AS "Should be 1";
```

Should be 1

0.99999999999999900

Share Improve this answer

answered Oct 23, 2010 at 1:58

Follow



[BrokeMyLegBiking](#)

5,988 ● 15 ● 52 ● 66



0

Your accountants will want to control how you round. Using float means that you'll be constantly rounding,



usually with a `FORMAT()` type statement, which isn't the way you want to do it (use `floor` / `ceiling` instead).



You have currency datatypes (`money`, `smallmoney`), which should be used instead of float or real. Storing decimal (12,2) will eliminate your roundings, but will also eliminate them during intermediate steps - which really isn't what you'll want at all in a financial application.

Share Improve this answer

Follow

edited Feb 18, 2020 at 15:14



Nimantha

6,438 ● 6 ● 30 ● 75

answered Sep 15, 2008 at 12:48



David T. Macknet

3,162 ● 3 ● 28 ● 36
