[SQL Prompt Code Analysis: Avoid using the ISNUMERIC Function (E1029)](https://www.red-gate.com/hub/product-learning/sql-prompt/sql-prompt-code-analysis-avoid-using-isnumeric-function-e1029" \o "SQL Prompt Code Analysis: Avoid using the ISNUMERIC Function (E1029))

[Phil Factor](https://www.red-gate.com/blog/author/phil-factor)

24 August 2018

FROM: <https://www.red-gate.com/hub/product-learning/sql-prompt/sql-prompt-code-analysis-avoid-using-isnumeric-function-e1029>

This is a guest post from Phil Factor. Phil Factor (real name withheld to protect the guilty), aka Database Mole, has 30 years of experience with database-intensive applications.  
  
Despite having once been shouted at by a furious Bill Gates at an exhibition in the early 1980s, he has remained resolutely anonymous throughout his career.  
  
He is a regular contributor to [Simple Talk](http://www.simple-talk.com/) and [SQLServerCentral](http://www.sqlservercentral.com/).

Avoid using the IsNumeric() function, because it can often lead to data type conversion errors, when importing data. SQL Prompt Code Analysis rules include an Execution rule, [E1029](https://documentation.red-gate.com/codeanalysis/execution-rules/ei029), which will alert you to use of this function, in your T-SQL. If you’re working on SQL Server 2012 or later, it’s much better to use the Try\_Convert()or Try\_Cast() function instead. On earlier SQL Server versions, the only way to avoid it is by using LIKE expressions.

The stated purpose of IsNumeric() is to “determine whether an expression is a valid numeric type“. It accepts int, numeric, float and money numbers, and will tell you whether the string or number you provided can be cast or converted into a number. The trouble is that it won’t tell you what sort of number; it will tell you that something like ',' or '$-.' or '-' or '0E0' can be converted to a number but leaves you to guess what sort.

Data type conversions when using IsNumeric()

Let’s take an obvious use for IsNumeric(). You are importing values from a log file that has a column that is supposed to be a floating point number. You want to be sure that each input row can be inserted into your import table, which has a float column. If not, you want the row to be held in a separate input table, for investigation. The CSV file is parsed, and your input includes the string ‘+’. You check with IsNumeric(), it passes the check and so you insert the row. Bang!

|  |  |
| --- | --- |
| 1 | 'Error converting data type varchar to float.' |

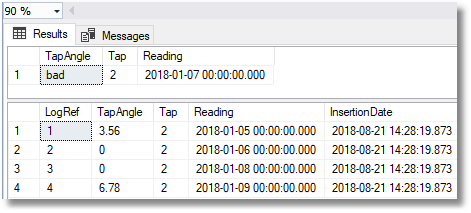
You’ve got an exception. The ‘+’ string can be cast into money (0.00), an int (0) but not a float.

Let’s flesh this out a bit so we can try things out. We’ll start by successfully filtering out a bad row, so that we can deal with afterwards, by manual intervention.

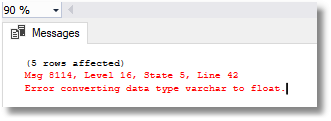
|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50 | -- We pretend that this is the table we want to import into    DECLARE @OurPretendImportTable TABLE      (      LogRef INT IDENTITY,      TapAngle FLOAT NOT NULL,      Tap INT,      Reading DATETIME NOT NULL,      InsertionDate DATETIME NOT NULL DEFAULT GetDate()      );    -- We pretend that this is the import values from the log file    DECLARE @OurRawData TABLE      (      TapAngle VARCHAR(10) NOT NULL,      Tap VARCHAR(3) NOT NULL,      Reading DATETIME NOT NULL      );    -- We pretend that this contain the bad values from the log file    DECLARE @OurBadRawData TABLE      (      TapAngle VARCHAR(10) NOT NULL,      Tap VARCHAR(3) NOT NULL,      Reading DATETIME NOT NULL      );    --Stock the OurRawData table with strings from the pretend log    INSERT INTO @OurRawData (Tapangle,tap,reading)      SELECT Tapangle,tap,reading FROM (VALUES        ('3.56','2','1/5/2018'),        ('0E0','2','1/6/2018'),        ('bad','2','1/7/2018'),        ('-0D0','2','1/8/2018'),        ('6.78','2','1/9/2018')            )importValues(Tapangle,tap,reading)    --we check to see if the values are numeric; if not, into OurBadRawData they go    IF EXISTS (SELECT \*  FROM @OurRawData      WHERE IsNumeric(TapAngle)=0)      INSERT INTO @OurBadRawData (Tapangle,tap,reading)        SELECT Tapangle,tap,reading           FROM @OurRawData            WHERE IsNumeric(TapAngle)=0    INSERT INTO @OurPretendImportTable (Tapangle,tap,reading)    SELECT Tapangle,tap,reading         FROM @OurRawData        WHERE IsNumeric(TapAngle)=1    /\*    (5 rows affected)    (1 row affected)    (4 rows affected)    \*/    SELECT \* FROM @OurBadRawData    SELECT \* FROM @OurPretendImportTable |

Listing 1

Execute this, and you’ll find that the routine has successfully filtered out the bad value:



Now all we do is to change the word ‘bad’ in the VALUES list into ‘+’ and what happens?



You’ve been caught out. The ‘+’ symbol can be cast to money or an int, but not a float.

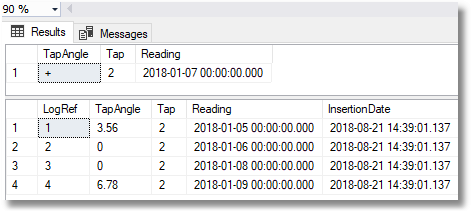
Replace IsNumeric with Try\_Convert

The answer is to replace IsNumeric('+') with the more recent Try\_Convert(). In Listing 1, leave our previously-problematic ‘+‘ symbol in place, and change the two final expressions that check to see if the values are numeric, as follows:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | --we check to see if the values are numeric; if not, into OurBadRawData they go    IF EXISTS (SELECT \*  FROM @OurRawData      WHERE Try\_Convert(FLOAT,TapAngle) IS null)      INSERT INTO @OurBadRawData (Tapangle,tap,reading)        SELECT Tapangle,tap,reading           FROM @OurRawData            WHERE Try\_Convert(FLOAT,TapAngle) IS null    INSERT INTO @OurPretendImportTable (Tapangle,tap,reading)    SELECT Tapangle,tap,reading           FROM @OurRawData        WHERE Try\_Convert(FLOAT,TapAngle) IS NOT NULL |

Listing 2

It all works fine, and we are back with a more robust input routine:



Flushed with success, you can then change the routine to check the dates and the tap number too.

Replacing IsNumeric with LIKE expressions

You haven’t got SQL Server 2012 or later? That’s a shame. You’ll need to build some bulkier LIKE expressions instead, in order to replace IsNumeric() with code that tells you more precisely whether the string can be cast as a float. To avoid adding all that complexity to our code, we’ll opt to hide it in our import table as a calculated column. For anyone stuck on SQL Server 2005, I’ll put the calculated column in a temporary table, rather than a table variable.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62 | -- We pretend that this is the table we want to import into    DROP TABLE IF EXISTS #OurRawData    DECLARE @OurPretendImportTable TABLE      (      LogRef INT IDENTITY,      TapAngle FLOAT NOT NULL,      Tap INT,      Reading DATETIME NOT NULL,      InsertionDate DATETIME NOT NULL DEFAULT GetDate()      );    -- We pretend that this is the import values from the log file    CREATE TABLE #OurRawData      (      TapAngle VARCHAR(10) NOT NULL,      Tap VARCHAR(3) NOT NULL,      Reading DATETIME NOT NULL,      TapAngleValid AS CASE              WHEN Stuff(LTrim(TapAngle),1,1,'') NOT LIKE '%[^-+.ED0123456789]%' --is it a float?                AND Left(LTrim(TapAngle),1) LIKE '[-.+0123456789]'                   AND TapAngle LIKE '%[0123456789][ED][-+0123456789]%'                   AND Right(TapAngle ,1) LIKE N'[0123456789]'                 THEN 'float'             WHEN Stuff(LTrim(TapAngle),1,1,'') NOT LIKE '%[^.0123456789]%' --is it numeric                  AND Left(LTrim(TapAngle),1) LIKE '[-.+0123456789]'                  AND TapAngle LIKE '%.%' AND TapAngle NOT LIKE '%.%.%'                  AND TapAngle LIKE '%[0123456789]%'                 THEN 'float' ELSE NULL  END,      );    -- We pretend that this contains the bad values from the log file    DECLARE @OurBadRawData TABLE      (      TapAngle VARCHAR(10) NOT NULL,      Tap VARCHAR(3) NOT NULL,      Reading DATETIME NOT NULL      );    --so we stock our raw data table with strings from the pretend log    INSERT INTO #OurRawData (Tapangle,tap,reading)      SELECT Tapangle,tap,reading FROM (VALUES        ('3.56','2','1/5/2018'),        ('0E0','2','1/6/2018'),        ('+','2','1/7/2018'),        ('-0D0','2','1/8/2018'),        ('6.78','2','1/9/2018')            )importValues(Tapangle,tap,reading)    --we check to see if the values are numeric    IF EXISTS (SELECT \*  FROM #OurRawData      WHERE TapAngleValid IS null)      INSERT INTO @OurBadRawData (Tapangle,tap,reading)        SELECT Tapangle,tap,reading             FROM #OurRawData            WHERE TapAngleValid IS null    INSERT INTO @OurPretendImportTable (Tapangle,tap,reading)    SELECT Tapangle,tap,reading            FROM #OurRawData        WHERE TapAngleValid IS NOT NULL    /\*    (5 rows affected)    (1 row affected)    (4 rows affected)    \*/    SELECT \* FROM @OurBadRawData    SELECT \* FROM @OurPretendImportTable |

Listing 3

You don’t need to figure out these LIKE expressions because they are all in my blog ‘[Is It Time To Stop Using IsNumeric()](https://www.red-gate.com/simple-talk/blogs/time-stop-using-isnumeric/)‘. There are also more explanation of the quirkiness of IsNumeric here: ‘[IsNumeric() Broken? Only up to a point.](https://www.red-gate.com/simple-talk/blogs/isnumeric-broken-only-up-to-a-point/)‘

Conclusion

The IsNumeric() function was never much use, because you only rarely want to ask ‘can this string be cast to any numeric type?‘ You almost always want to know whether it can be cast specifically to money, int, numeric, float and so on. It is much better to use Try\_Convert() or Try\_Cast (the only difference is the way parameters are passed). Try\_Parse() is more useful for parsing dates in other language formats.

If you understand the quirks of IsNumeric()and can accept them, then by all means use it. The worst of its quirks is that it doesn’t know about Unicode currency symbols, such as the UK Pound sign ‘₤’ and the Euro, ‘€’, so it says that they can’t be cast as money. Obviously, if you only want to know whether a string or numeric value can be cast as one or more of various the numeric types, and you aren’t worried about the fact that it doesn’t recognize most of the world’s currencies as money, then use IsNumeric(). However, for the majority of cases, Try\_Convert() is far handier and reliable.

If you are stuck with an old version of SQL Server, then your best choice is to use the LIKE expressions that I’ve shown you in my blog ‘[Is It Time To Stop Using IsNumeric()](https://www.red-gate.com/simple-talk/blogs/time-stop-using-isnumeric/)‘.