Minimum and Maximum Values by Data Type

Max Ganz II @ Redshift Research Project

5th March 2023

Abstract

The official Redshift documentation indicates minimum and maximum values for data types. These values contain factual errors, typographic errors, and errors of omission (some data types have no values given). The actual minimum and maximum values are presented. During investigation, it also became clear that the mechanisms used to connect to Redshift - `psycopg2`, `psql`, `ODBC`, etc - all seems to be performing significant data processing, and at times are getting it wrong, leading to behaviour such as the silent modification of inserted values and incorrect values being presented from `SELECT`. Finally, there appears to be a bug in Redshift's handling of very small and very large `float8` values, such that rather than `INSERT` failing, the values are inserted but are converted to -/+ infinity.

Contents

Introduction

The official documentation provides minimum and maximum values by data type, but there are factual errors, one or two typographic errors, and, finally, two errors of omission, where `float4` and `float8` do not have values listed.

This white paper investigates the minimum and maximum values for each data type.

One important matter which became apparent during investigation is that the various mechanisms used to connect to Redshift (`psql`, `psycopg2`, `ODBC`, etc) generally seem to do their own data processing, including value range validation, and at times get it wrong, with consequences such as throwing improper errors, restricting minimum and maximum values, or, worse, even silently changing them.

Test Method

The test script does not explore the minimum and maximum values to find them; rather, I know from prior experimentation what those values are, and I demonstrate this by issuing the minimum and maximum values, and then the next smaller or larger value, respectively, and show the error which occurs.

The script then simply issues a series of `INSERT` statements, tracking which succeed and which fail, and the results demonstrate then the minimum and maximum valid values, and also any unusual, remarkable or improper behaviour.

Results

See Appendix A for the Python `pprint` dump of the results dictionary.

Test duration was 296 seconds (includes server bring-up and shut-down time).

dc2.large, 2 nodes (1.0.30840)

Documented Minimums

Data Type	Minimum
bool	FALSE / ‘f’ / ‘false’ / ‘n’ / ‘no’ / ‘0’
date	4713 BC
float4	Not given
float8	Not given
int2	-32768
int4	-2147483648
int8	-9223372036854775808
numeric(19,0)	-9223372036854775808
numeric(38,0)	-99999999999999999999999999999999999999
time	00:00:00
timetz	00:00:00+1459
timestamp	4713 BC
timestamptz	4713 BC

Documented Maximums

Data Type	Maximum
bool	TRUE / ‘t’ / ‘true’ / ‘y’ / ‘yes’ / ‘1’
date	294276 AD
float4	Not given
float8	Not given
int2	32767
int4	2147483647
int8	9223372036854775807
numeric(19,0)	9223372036854775807
numeric(38,0)	99999999999999999999999999999999999999
time	24:00:00
timetz	00:00:00+1459
timestamp	294276 AD
timestamptz	294276 AD

Actual Minimums

Data Type	Minimum
bool	See Discussion
date	4714-11-24 BC
float4	-340282346638528878701170114963097780224
float8	Too long; see below
int2	-32768
int4	-2147483648
int8	-9223372036854775808
numeric(19,0)	-9223372036854775808
numeric(38,0)	-99999999999999999999999999999999999999
time	00:00:00.000000
timetz	00:00:00.000000-1559

Data Type	Minimum
timestamp	4714-11-24 00:00:00.000000 BC
timestamptz	4714-11-24 00:00:00.000000+0000 BC

`Minimum `float8` :`
`-17976931348623158079372897140530341507993413271003782693617377898044496829276475094664901797758720709633028641669288791094655554785194040263065748867150582068190890200070838367627385484581771153176447573027006985557136695962284291481986083493647529271907416844436551070434271155969950809304288017790417449779`

Actual Maximums

Data Type	Maximum
bool	See Discussion
date	5874897-12-31 AD
float4	340282346638528878701170114963097780224
float8	Too long; see below
int2	32767
int4	2147483647
int8	9223372036854775807
numeric(19,0)	9223372036854775807
numeric(38,0)	99999999999999999999999999999999999999
time	23:59:59.999999
timetz	23:59:59.999999+1559
timestamp	294276-12-31 23:59:59.999999 AD
timestamptz	294277-01-09 04:00:54.775807+0000 AD

`Maximum `float8` :`
`17976931348623158079372897140530341507993413271003782693617377898044496829276475094664901797758720709633028641669288791094655554785194040263065748867150582068190890200070838367627385484581771153176447573027006985557136695962284291481986083493647529271907416844436551070434271155969950809304288017790417449779`

General Proofs

bool

Description	Value Inserted	Insert Result	Value Selected
Docs False #1	FALSE	Success	False
Docs False #2	‘f’	Success	False
Docs False #3	‘n’	Success	False
Docs False #4	‘no’	Success	False
Docs False #5	0	Success	False
Docs True #1	TRUE	Success	True
Docs True #2	‘t’	Success	True

Description	Value Inserted	Insert Result	Value Selected
Docs True #3	‘y’	Success	True
Docs True #4	‘yes’	Success	True
Docs True #5	1	Success	True
False Valid #1	0	Success	False
False Valid #2	00	Success	False
False Valid #3	‘false’	Success	False
False Valid #4	‘fals’	Success	False
False Valid #5	‘fal’	Success	False
False Valid #6	‘fa’	Success	False
False Valid #7	‘fAlSe’	Success	False
True Valid #1	1	Success	True
True Valid #2	01	Success	True
True Valid #3	-1	Success	True
True Valid #4	2	Success	True
True Valid #5	10000	Success	True
True Valid #6	‘true’	Success	True
True Valid #7	‘tru’	Success	True
True Valid #8	‘tr’	Success	True
True Valid #9	‘tRuE’	Success	True
True Valid #10	‘ye’	Success	True
Invalid #1	‘oink’	Failure	
Invalid #2	‘truelove’	Failure	
Invalid #3	‘terribletwins’	Failure	
Invalid #4	‘noo’	Failure	
Invalid #5	5.5	Failure	
Invalid #6	‘00’	Failure	
Invalid #7	‘01’	Failure	

date

Description	Value Inserted	Insert Result
Min Invalid	4714-11-23 BC	Failure
Minimum	4714-11-24 BC	Success
Maximum	5874897-12-31 AD	Success
Max Invalid	5874898-01-01 AD	Failure

float4

Description	Value Inserted	Insert Result
Min Invalid	-340282346638528878701170114963097780225	Failure
Minimum	-340282346638528878701170114963097780224	Success
Theoretical Min	-340282346638528859811704183484516925440	Success
Theoretical Max	340282346638528859811704183484516925440	Success
Maximum	340282346638528878701170114963097780224	Success
Max Invalid	340282346638528878701170114963097780225	Failure

Description	Value Inserted	Insert Result
-Infinity	'-Infinity'	Success
+Infinity	'+Infinity'	Success
NaN	'NaN'	Success

float8

Description	Value Inserted	Insert Result
Min Invalid	Too long; see below	Failure
Min -Infinity	Too long; see below	Success
Max -Infinity	Too long; see below	Success
Minimum	Too long; see below	Success
Theoretical Min	Too long; see below	Success
Theoretical Max	Too long; see below	Success
Maximum	Too long; see below	Success
Min +Infinity	Too long; see below	Success
Max +Infinity	Too long; see below	Success
Max Invalid	Too long; see below	Failure
-Infinity	'-Infinity'	Success
+Infinity	'+Infinity'	Success
NaN	'NaN'	Success

```
Min Invalid `float8` :
-17976931348623159077293051907890247336179769789423065727343008115773267580550096313270847732240753602112011387987139335765878976881441662249284743063947412437776789342486548527630221960124609411945308295208500576883815068234246288147391311054082723716335051068458629823994724593847971630483535632962422413721637
Min -Infinity `float8` :
-17976931348623159077293051907890247336179769789423065727343008115773267580550096313270847732240753602112011387987139335765878976881441662249284743063947412437776789342486548527630221960124609411945308295208500576883815068234246288147391311054082723716335051068458629823994724593847971630483535632962422413721537
Max -Infinity `float8` :
-17976931348623158079372897140530341507993413271003782693617377898044496829276475094664901797758720709633028641669288791094655554785194040263065748867150582068190890200070838367627385484581771153176447573027006985557136695962284291481986083493647529271907416844436551070434271155969950809304288017790417449779
Minimum `float8` :
-17976931348623158079372897140530341507993413271003782693617377898044496829276475094664901797758720709633028641669288791094655554785194040263065748867150582068190890200070838367627385484581771153176447573027006985557136695962284291481986083493647529271907416844436551070434271155969950809304288017790417449779
```

```
Theoretical Min `float8` :
-17976931348623157081452742373170435679807056752584499659891747
680315726078002853876058955863276687817154045895351438246423432
132688946418276846754670353751698604991057655128207624549009038
932894407586850845513394230458323690322294816580855933212334827
479782620414472316873817718091929988125040402618412485836800
```

```
Theoretical Max `float8` :
17976931348623157081452742373170435679807056752584499659891747680
315726078002853876058955863276687817154045895351438246423432132
688946418276846754670353751698604991057655128207624549009038932
894407586850845513394230458323690322294816580855933212334827479
782620414472316873817718091929988125040402618412485836800
```

```
Maximum `float8` :
17976931348623158079372897140530341507993413271003782693617377898
044496829276475094664901797758720709633028641669288791094655547
851940402630657488671505820681908902000708383676273854845817711
531764475730270069855571366959622842914819860834936475292719074
168444365510704342711559699508093042880177904174497791
```

```
Min +Infinity `float8` :
17976931348623158079372897140530341507993413271003782693617377898
044496829276475094664901797758720709633028641669288791094655547
851940402630657488671505820681908902000708383676273854845817711
531764475730270069855571366959622842914819860834936475292719074
168444365510704342711559699508093042880177904174497792
```

```
Max +Infinity `float8` :
17976931348623159077293051907890247336179769789423065727343008115
773267580550096313270847732240753602112011387987139335765878976
881441662249284743063947412437776789342486548527630221960124609
411945308295208500576883815068234246288147391311054082723716335
051068458629823994724593847971630483535632962422413721 5
```

```
Max Invalid `float8` :
17976931348623159077293051907890247336179769789423065727343008115
773267580550096313270847732240753602112011387987139335765878976
881441662249284743063947412437776789342486548527630221960124609
411945308295208500576883815068234246288147391311054082723716335
051068458629823994724593847971630483535632962422413721 6
```

int2

Description	Value Inserted	Insert Result
Min Invalid	-32769	Failure
Minimum	-32768	Success
Maximum	32767	Success
Max Invalid	32768	Failure

int4

Description	Value Inserted	Insert Result
Min Invalid	-2147483649	Failure
Minimum	-2147483648	Success
Maximum	2147483647	Success
Max Invalid	2147483648	Failure

int8

Description	Value Inserted	Insert Result
Min Invalid	-9223372036854775809	Failure
Minimum	-9223372036854775808	Success
Maximum	9223372036854775807	Success
Max Invalid	9223372036854775808	Failure

numeric(19,0)

Description	Value Inserted	Insert Result
Min Invalid	-9223372036854775809	Failure
Minimum	-9223372036854775808	Success
Maximum	9223372036854775807	Success
Max Invalid	9223372036854775808	Failure

numeric(38,0)

Description	Value Inserted	Insert Result
Min Invalid	-100000000000000000000000000000000000	Failure
Minimum	-99999999999999999999999999999999999999	Success
Maximum	99999999999999999999999999999999999999	Success
Max Invalid	100000000000000000000000000000000000000	Failure

time

Description	Value Inserted	Insert Result
Minimum	00:00:00.000000	Success
Maximum	23:59:59.999999	Success

timetz

Description	Value Inserted	Insert Result
Min Invalid	00:00:00.000000-1600	Failure
Minimum	00:00:00.000000-1559	Success
Maximum	23:59:59.999999+1559	Success

Description	Value Inserted	Insert Result
Max Invalid	23:59:59.999999+1600	Failure

timestamp

Description	Value Inserted	Insert Result
Min Invalid	4714-11-23 23:59:59.999999 BC	Failure
Minimum	4714-11-24 00:00:00.000000 BC	Success
Maximum	294276-12-31 23:59:59.999999 AD	Success
Max Invalid	394277-01-01 00:00:00.000000 AD	Failure
Rounding #1	2020-01-01 00:00:00.9999994 AD	Success
Rounding #2	2020-01-01 00:00:00.9999995 AD	Failure

timestamptz

Description	Value Inserted	Insert Result
Min Invalid	4714-11-23 23:59:59.999999+0000 BC	Failure
Minimum	4714-11-24 00:00:00.000000+0000 BC	Success
Maximum	294277-01-09 04:00:54.775807+0000 AD	Success
Max Invalid	294277-01-09 04:00:54.775808+0000 AD	Failure
Underflow #1	4714-11-24 00:01:00.000000+0001 BC	Success
Underflow #2	4714-11-24 00:01:00.000000+0002 BC	Failure
Overflow #1	294277-01-09 03:59:54.775807-0001 AD	Success
Overflow #2	294277-01-09 03:59:54.775807-0002 AD	Failure
Rounding #1	2020-01-01 00:00:00.9999994 AD	Success
Rounding #2	2020-01-01 00:00:00.9999995 AD	Failure

Discussion

bool

The Redshift docs specify six values which mean `false` and six values which mean `true`. The docs are correct in that the values documented do indeed behave as described; but upon examination we find other sets of values - including a very strange set - producing valid booleans.

First, *all* integers, other than zero, produce a `true` boolean, with `0` producing `false`. (The docs actually do not list integers as being valid booleans, rather, the single character strings `'0'` and `'1'` are listed as valid booleans.)

Second, and this is the very strange set, of the strings which are documented as producing valid `boolean` values, it turns out that *any* substring, starting at the beginning of the word, produces a valid `boolean`.

So for example, the string `'false'` produces a valid `false` boolean; and thus so also do the strings `'fals'`, `'fal'`, `'fa'` and `'f'`. (Although `'f'` is documented as a valid value).

This behaviour is case-insensitive, so for example, `'fAlS'` produces a valid `boolean`.

The following strings are documented as producing `boolean` values, and all exhibit this behaviour;

1. `'false'`
2. `'true'`
3. `'yes'`
4. `'no'`

This, honestly, is so bizarre it's almost surreal. It violates the principle of strong data typing (especially since it's undocumented), but it's practically impossible to imagine this being implemented by accident - it must be someone thought this was a good idea.

There are two more strings which produce valid booleans, `'0'` and `'1'`, but they are both one character long. Note in this case the string is not being converted to an integer; the strings `'00'` and `'01'` for example do not produce valid booleans (they fail to insert, the same as strings in general).

date

The documentation indicates the minimum value for `date` is `4713 BC`, the maximum `294276 AD`. No month or day are given.

It turns out the leader node and worker nodes are using different representation of dates.

The leader node is using the AD/BC standard, where there is no year 0, and dates are written with a positive number for years, but a `BC` indicator for years prior to `1 AD`), e.g. `4000-01-01 BC`.

The worker nodes are using the ISO 8601 standard, where there *is* a year 0, and dates are written where years can be a negative number and there is no `BC` indicator, e.g. `-3999-01-01`.

Where AD/BC has no year 0, but ISO 8601 has a year 0, the year specifed by `4000-01-01 BC` is also the year specified by `-3999-01-01`.

Redshift worker nodes accept only AD/BC notation, but emit only ISO 8601, and as such cannot accept their own output for BC dates.

The leader node accepts and emits AD/BC notation.

The leader node has an actual minimum date of `4713-01-01 BC` and a maximum date of `5874897-12-31 AD`.

The worker nodes have an actual minimum date of `4714-11-24 BC` (which in ISO 8601 is `-4713-11-24`) and a maximum date of `5874897-12-31 AD`.

So it looks like for the minimum year the documentation is confused about, or doesn't know about, the two different representations in use, and so has the correct year in AD/BC notation for the leader node, but the incorrect year for the worker nodes.

The documentation also does not mention the peculiar month and day values for the worker node minimum date.

For the maximum year, the documentation is simply wrong.

In fact the maximum year given is the maximum year for the `timestamp` type (and this is the correct maximum year for that type); I wonder if this was a copy and paste error. The docs are I think not checked by technical staff, so all such errors go undetected.

There is of course no mention that two different date representation systems are in use.

Finally, note all BC dates can be inserted, but not selected, with both `psycopg2` and the Amazon Redshift ODBC driver. I think what's happening is that both are expecting as is the case with Postgres AD/BC format output, but they're getting ISO 8601, and they both reject it when parsing.

float4

The documentation does not provide a minimum or maximum value for `float4`.

This data type under the hood is an IEEE-754 32-bit single precision floating point number, and so is composed of a sign bit, an exponent, and a mantissa. An exponent which is all `1` is used for special values (infinity and "Not a Number") and so the largest exponent which represents a number is all `1` except for the least significant bit. The mantissa is simply used as-is, no special values, so the largest mantissa is all `1`.

This mathematically gives us the largest value as `(2^127) * (2 - (1 / (2^23)))`, the negative version of this value being the minimum, the positive version the maximum (as `754` type floats have a sign bit).

Now, the integer value we get from this is `340282346638528859811704183484516925440`, and when we insert this, and then select the value we inserted, we get `-3.4028235e+38` (this is with `set extra_float_digits to 2;`)

However, I think due to the inherent inaccuracy of floating point values, the actual largest number you can insert without insert failing is `340282346638528878701170114963097780224`.

To my pleasant surprise, it turns out Redshift can handle in `INSERT` the full range of `float4` values actually written out in full as an integer, like so;

```
insert into table_1 ( column_1 ) values ( (340282346638528878701170114963097780224)::float
);
```

float8

The documentation does not provide a minimum or maximum value for `float8`.

There were complications involved in figuring out the minimum and maximum values for `float8`.

To explain, I first need to explain the nature of a `float8`, and so the expected minimum and maximum values, so I can then by contrast explain what was actually found.

This data type under the hood is an IEEE-754 32-bit single precision floating point number, and so is composed of a sign bit, an exponent, and a mantissa. An exponent which is all `1` is used for special values (infinity and "Not a Number") and so the largest exponent which represents a number is all `1` except for the least significant bit. The mantissa is simply used as-is, no special values, so the largest mantissa is all `1`.

This mathematically gives us the largest value as `2^1023 * (1 + (1 - (1/2^52)))`, the negative version of this value being the minimum, the positive version the maximum (as `754` type floats have a sign bit).

The integer value we get from this is `17976931348623157081452742373170435679807056752584499659891747`
which is in theory the largest number we can store (and the negative version of this number is the smallest number we can store).

Now, the first issue we find is that the larger number we can actually get away with storing is in fact `17976931348623158079372897140530341507993413271003782693617377898044968282`

I think this is permitted because of the inherent inaccuracy of floating point numbers; I guess - but I have not looked and proved it - that this number in fact ends up being the theoretical maximum and so it is allowed.

As such, we would then expect if we add one to this actual maximum, insert would fail.

This brings us to the second issue; `INSERT` does not fail. What happens now is that when you select the value you inserted, to examine it, you get back `Infinity`.

That's definitely wrong.

You see, if you want to insert positive or negative infinity, you do so by inserting the strings `'+Infinity'` or `'-Infinity'`. Any actual *number*, which is out of range, has to be rejected by insert - just as inserting 100,000 into a signed two-byte integer has to be rejected.

We find then that the range of numbers which leads to `Infinity` being selected continues all the way up to and including `17976931348623159077293051907890247336179769789423065727343`

It is only when we add one to this number that we finally get `INSERT` to fail.

(All of this is mirrored with negative numbers, as would be expected - the only difference being the sign bit is now set.)

So that's one of the problems.

The second problem I think is from the Python module `psycopg2`.

So, we have these four numbers as described above - the theoretical maximum, the actual maximum, the beginning of the improper positive infinity range, the end of the positive infinity range (the first insert which fails).

After an insert, selecting the value inserted, using `psycopg2`, has the number come back as `'inf'`, which is positive infinity.

That's wrong, for sure, because at the very least the theoretical maximum definitely isn't positive infinity; the exponent doesn't have all bits set to `1`.

However, if the value being selected is cast to `varchar`, then the first two values (theoretical maximum and actual maximum) both become actual numbers - they stop being returned as `'inf'`. The other two numbers (beginning and end of positive infinity) both still come back as `'Infinity'`.

This behaviour is not found with `psql`, issuing insert and select by hand; here we always get what we get when casting to `varchar` when using `psycopg2`.

So where does this leave us?

1. It looks like the code to return `float8` in `psycopg2` is bugged (I've not investigated to find out how small the value has to become for the `float8` code to return the value correctly).
2. It looks like the code in Redshift to handle `float8` above the actual maximum is bugged

int2

This is a simple signed two-byte integer, and the documented minimum and maximum values are correct.

int4

This is a simple signed four-byte integer, and the documented minimum and maximum values are correct.

int8

This is a simple signed eight-byte integer, and the documented minimum and maximum values are correct.

numeric(19,0)

Up to and including 19 scale, `numeric` is under the hood a simple signed eight-byte integer. The minimum and maximum values of a signed eight-byte integer are less than that supported by scale 19, and so the minimum and maximum values for scale 19 are actually that of the signed eight-byte integer. This is documented.

numeric(38,0)

At 20 scale and above, `numeric` is under the hood a simple signed sixteen-byte integer. The minimum and maximum values of a signed sixteen-byte integer are quite a bit larger than that supported by scale 38, and so the minimum and maximum values for scale 38 (a 38 digit base 10 number) are the limits. This is not documented, but it is obvious.

time

The documented minimum is `00:00:00`, which is correct, but the documented maximum is `24:00:00`, which is not; the author obviously was looking to indicate the end of the day, but the time for that is actually `23:59:59.999999`.

timetz

So, there are three errors here.

1. As with `time`, the minimum value of `00:00:00` is correct but the maximum value is incorrect, but now it's incorrect in a different way =-) rather than being `24:00:00`, it's now `00:00:00`, which is like saying the range is from say `10:00:00` to `10:00:00`, which is a range of zero. Again, the correct final time is `23:59:59.999999`.
2. The second error is typographic; both the minimum and maximum times indicate a '+' for the timezone difference. This needs to be a `-` for the minimum time.
3. The third error is the maximum amount of timezone difference. The documentation has this as `1459`, which is to say, fourteen hours and fifty-nine minutes. In fact, it is `1559`.

This is probably a good time (see what I did there :-) to mention that I have, through finding over the years in the docs a range of blatant fundamental factual errors, come absolutely to the view that no one technical proof reads the documentation.

My guess is someone explains the matter in hand to the author, who is not technical, he does his best to write what he understands, and no one checks the result - and in all of this is the constant effort by all involved to obfuscate everything which is not a strength.

timestamp

As with `date`, the minimum value is actually `4714-11-24 00:00:00.000000` rather than `4713 BC`. The maximum value is correctly documented.

Timestamps in Redshift have six fractional digits, but you can actually specify more; they will be rounded off to six digits - but this rounding works *only* for the fractional part of the second. If the rounding would cause the timestamp to increment the number of seconds, the `INSERT` will fail.

So you can enter, say, `2020-01-01 00:00:00.9999994 AD` and this will be okay, and will be rounded to `2020-01-01 00:00:00.999999 AD`, but if you enter `2020-01-01 00:00:00.9999995 AD`, which would require rounding up and so changing the number of seconds, the `INSERT` fails.

timestamptz

Both the minimum and maximum values are incorrect, but not by much, although the maximum is incorrect in a very peculiar way.

The minimum value is only wrong in the way the other date and timestamp data types are wrong; the documented value is `4713 BC` but is actually `4714-11-24 00:00:00.000000+0000 BC`.

The maximum value is documented as `294276 AD`, but is in fact the very strange value `294277-01-09 04:00:54.775807+0000 AD`.

Even one microsecond more produces an invalid value. Why is it the maximum `timestamptz` involves 775807 microseconds? I feel like Isidor Rabi.

It is not possible to insert a timestamp which is outside of the minimum or maximum timestamp but, which once the timezone difference is applied, will fall within the minimum and maximum range. The timestamp, regardless of its timezone difference, must always be within the minimum and maximum range.

Finally, rounding occurs as with `timestamp`.

Conclusions

The `boolean` data type has unexpected behaviour with regard to the four strings, `false`, `true`, `yes` and `no`, which are valid booleans; any case-insensitive substring, starting from the beginning of the string, is also a valid boolean.

The `date` type on the leader node uses AD/BC notation, where there is no year 0, but on the worker node uses ISO 8601 notation, where there is a year 0.

The differences in the handling of year 0 mean the documented minimum value for `date`, which is given in AD/BC format as `4713 BC`, is correct for the leader node, but incorrect for the worker nodes, where the minimum date is `4714 BC` (which converts in ISO 8601 format to the year `-4713`).

Additionally, the documentation gives only the year of the minimum date, which rather implies the 1st January. This is correct for the leader node, but incorrect for the worker nodes, where the minimum date, oddly, is `4714-11-24 BC`.

Note the leader node accepts and emits AD/BC notation. Worker nodes accept AD/BC notation but emit ISO 8601 notation; worker nodes cannot accept what they emit.

The documented maximum value for `date`, which is `294276 AD`, is wrong, for both the leader and worker nodes. The actual value for both leader and worker nodes is `5874897-12-31 AD`.

With floating point types, there is a theoretical maximum (as defined by the IEEE-754 standard) and an actual maximum, where I believe (but have not confirmed) the inherent inaccuracy of floating point values causes the actual maximum to become the theoretical maximum and so be a valid number.

The `float8` data type appears to be bugged, in that the value range above the actual maximum, up to a value some way above that, produces `+Infinity`, when it should cause `INSERT` to throw an error (the same being true of course for the minimum, except the value there being `-Infinity`).

The Python module `psycopg2` also appears to be bugged in its handling of `float8`, in that when selecting values at and above the theoretical maximum,

it always returns `'inf'` (for `Infinity`), but if the value being selected is cast to `varchar`, it returns for the theoretical and actual maximum the actual and correct floating point value.

The Amazon Redshift ODBC driver seems to be bugged, being unable to handle negative dates (`BC`). The driver was not investigated beyond this finding, except to note it seems very large (the library file is fifty-nine megabytes in size; I'd expect one or two megabytes).

In general the mechanisms used to connect to Redshift seem to need to be under genuine and active suspicion and the results found, especially when working near the extremes of value ranges, may be a result of the connection mechanism going awry rather than being results actually from Redshift.

There are minor but confusing documentation errors for the `time` and `timetz` types, and a somewhat more significant error, in that the maximum timezone difference is documented as `1459` (fourteen hours, fifty-nine minutes) but in fact is actually `1559`.

With `timestamp` and `timestamptz`, timestamps with more than the normal six fractional digits can be specified, and the fractional part will be rounded, but the rounding is not permitted to modify the seconds column; if it would, the `INSERT` fails (e.g. `.9999994` is fine, but `.9999995` will fail to insert).

The documented maximum value for `timestamptz` is `294276 AD`, but is in fact the completely bizarre and unexpected `294277-01-09 04:00:54.775807+0000 AD`. Who ordered this?

With `timestamptz`, the timestamp entered, regardless of the timezone difference, must always be within the minimum and maximum values. A timestamp which is outside of those values, but which would after the timezone difference is applied come to be within them, will be rejected by `INSERT`.

Unexpected Findings

When you investigate Redshift, there are *always* unexpected findings.

1. One general finding was that the various mechanisms through which you communicate with Redshift - `psycopg2`, `ODBC`, `psql`, etc - is that many of these mechanisms perform their own value range checking and at times get it wrong.

2. The Amazon Redshift ODBC driver appears to be unable to handle BC dates.

   Inserting and then selecting the date `2000-01-01 AD' works fine, but then we find the date`2000-01-01 BC' can be inserted, and can be selected, but when iterating over the results from the select produces the error;

   ```
   pyodbc.DataError: ('22007', "[22007] [Amazon][Support]
   (40481) Invalid date format for '-1999-01-01'. The
   format should be [-]YYYY-MM-DD. (40481) (SQLGetData)")
   ```

Just in case you're thinking the insert format is wrong, and should be `-2000-01-01`, inserting that gives the error;

```
pyodbc.DataError: ('22009', '[22009] [Amazon][Amazon
Redshift] (30) Error occurred while trying to execute
a query: [SQLState 22009] ERROR:  time zone displacement
out of range: "-2000-01-01" (30) (SQLExecDirectW)')
```

It looks like Redshift itself is fine, but the Amazon Redshift ODBC driver is incorrectly handling BC dates and falling over.

I've not tried any other data types; this white paper is about Redshift, not about connection mechanisms.

3. The Amazon Redshift ODBC driver is *fifty-nine megabytes* in size. For what it's doing, this is staggeringly enormous. What's going on?

Revision History

v1

- Initial release.

v2

- Updated `date` section in Discussion, modified Conclusions accordingly.

v3

- Changed to Redshift Research Project (AWS have a copyright on "Amazon Redshift").

Appendix A : Raw Data Dump

Note these results are completely unprocessed; they are a raw dump of the results, so the original, wholly unprocessed data, is available.

```
{'proofs': {'dc2.large': {2: {'bool': {'Docs False #1': ('FALSE',
                                                         'Success',
                                                         True,
                                                         False),
                                       'Docs False #2': ("'f'",
                                                         'Success',
                                                         True,
                                                         False),
                                       'Docs False #3': ("'n'",
                                                         'Success',
                                                         True,
                                                         False),
                                       'Docs False #4': ("'no'",
                                                         'Success',
                                                         True,
                                                         False),
                                       'Docs False #5': ('0',
                                                         'Success',
                                                         True,
                                                         False),
                                       'Docs True #1': ('TRUE',
                                                        'Success',
                                                        True,
                                                        True),
                                       'Docs True #2': ("'t'",
                                                        'Success',
                                                        True,
                                                        True),
                                       'Docs True #3': ("'y'",
                                                        'Success',
                                                        True,
                                                        True),
                                       'Docs True #4': ("'yes'",
                                                        'Success',
                                                        True,
                                                        True),
                                       'Docs True #5': ('1',
                                                        'Success',
                                                        True,
                                                        True),
                                       'False Valid #1': ('0',
                                                          'Success',
                                                          True,
                                                          False),
                                       'False Valid #2': ('00',
```

```
                    'Success',
                    True,
                    False),
'False Valid #3': ("'false'",
                    'Success',
                    True,
                    False),
'False Valid #4': ("'fals'",
                    'Success',
                    True,
                    False),
'False Valid #5': ("'fal'",
                    'Success',
                    True,
                    False),
'False Valid #6': ("'fa'",
                    'Success',
                    True,
                    False),
'False Valid #7': ("'fAlSe'",
                    'Success',
                    True,
                    False),
'Invalid #1': ("'oink'",
                'Failure',
                False,
                ''),
'Invalid #2': ("'truelove'",
                'Failure',
                False,
                ''),
'Invalid #3': ("'terribletwins'",
                'Failure',
                False,
                ''),
'Invalid #4': ("'noo'",
                'Failure',
                False,
                ''),
'Invalid #5': ('5.5',
                'Failure',
                False,
                ''),
'Invalid #6': ("'00'",
                'Failure',
                False,
                ''),
'Invalid #7': ("'01'",
                'Failure',
                False,
```

```
                                ''),
                    'True Valid #1': ('1',
                                      'Success',
                                      True,
                                      True),
                    'True Valid #10': ("'ye'",
                                       'Success',
                                       True,
                                       True),
                    'True Valid #2': ('01',
                                      'Success',
                                      True,
                                      True),
                    'True Valid #3': ('-1',
                                      'Success',
                                      True,
                                      True),
                    'True Valid #4': ('2',
                                      'Success',
                                      True,
                                      True),
                    'True Valid #5': ('10000',
                                      'Success',
                                      True,
                                      True),
                    'True Valid #6': ("'true'",
                                      'Success',
                                      True,
                                      True),
                    'True Valid #7': ("'tru'",
                                      'Success',
                                      True,
                                      True),
                    'True Valid #8': ("'tr'",
                                      'Success',
                                      True,
                                      True),
                    'True Valid #9': ("'tRuE'",
                                      'Success',
                                      True,
                                      True)},
 'date': {'Max Invalid': ('5874898-01-01 AD',
                          'Failure',
                          False,
                          ''),
          'Maximum': ('5874897-12-31 AD',
                      'Success',
                      True,
                      '5874897-12-31'),
          'Min Invalid': ('4714-11-23 BC',
```

```
                                'Failure',
                                False,
                                ''),
                'Minimum': ('4714-11-24 BC',
                            'Success',
                            True,
                            '4714-11-24 BC')},
    'float4': {'+Infinity': ("'+Infinity'",
                             'Success',
                             True,
                             'Infinity'),
               '-Infinity': ("'-Infinity'",
                             'Success',
                             True,
                             '-Infinity'),
               'Max Invalid': ('340282346638528878701170114963097
                               'Failure',
                               False,
                               ''),
               'Maximum': ('340282346638528878701170114963097780
                           'Success',
                           True,
                           '3.4028235e+38'),
               'Min Invalid': ('-340282346638528878701170114963
                               'Failure',
                               False,
                               ''),
               'Minimum': ('-340282346638528878701170114963097
                           'Success',
                           True,
                           '-3.4028235e+38'),
               'NaN': ("'NaN'",
                       'Success',
                       True,
                       'NaN'),
               'Theoretical Max': ('340282346638528859811704183
                                   'Success',
                                   True,
                                   '3.4028235e+38'),
               'Theoretical Min': ('-340282346638528859811704183
                                   'Success',
                                   True,
                                   '-3.4028235e+38')},
    'float8': {'+Infinity': ("'+Infinity'",
                             'Success',
                             True,
                             'Infinity'),
               '-Infinity': ("'-Infinity'",
                             'Success',
                             True,
```

```
                    '-Infinity'),
                'Max +Infinity': ('179769313486231590772930519078
                                  'Success',
                                  True,
                                  'Infinity'),
                'Max -Infinity': ('-179769313486231580793728971405
                                  'Success',
                                  True,
                                  '-Infinity'),
                'Max Invalid': ('1797693134862315907729305190789
                                'Failure',
                                False,
                                ''),
                'Maximum': ('179769313486231580793728971405303415
                            'Success',
                            True,
                            '1.7976931348623157e+308'),
                'Min +Infinity': ('179769313486231580793728971405
                                  'Success',
                                  True,
                                  'Infinity'),
                'Min -Infinity': ('-179769313486231590772930519078
                                  'Success',
                                  True,
                                  '-Infinity'),
                'Min Invalid': ('-1797693134862315907729305190789
                                'Failure',
                                False,
                                ''),
                'Minimum': ('-179769313486231580793728971405303415
                            'Success',
                            True,
                            '-1.7976931348623157e+308'),
                'NaN': ("'NaN'",
                        'Success',
                        True,
                        'NaN'),
                'Theoretical Max': ('179769313486231570814527423
                                    'Success',
                                    True,
                                    '1.7976931348623157e+308'),
                'Theoretical Min': ('-179769313486231570814527423
                                    'Success',
                                    True,
                                    '-1.7976931348623157e+308')},
 'int2': {'Max Invalid': ('32768',
                          'Failure',
                          False,
                          ''),
          'Maximum': ('32767',
```

```
                        'Success',
                        True,
                        '32767'),
         'Min Invalid': ('-32769',
                         'Failure',
                         False,
                         ''),
         'Minimum': ('-32768',
                     'Success',
                     True,
                     '-32768')},
 'int4': {'Max Invalid': ('2147483648',
                          'Failure',
                          False,
                          ''),
          'Maximum': ('2147483647',
                      'Success',
                      True,
                      '2147483647'),
          'Min Invalid': ('-2147483649',
                          'Failure',
                          False,
                          ''),
          'Minimum': ('-2147483648',
                      'Success',
                      True,
                      '-2147483648')},
 'int8': {'Max Invalid': ('9223372036854775808',
                          'Failure',
                          False,
                          ''),
          'Maximum': ('9223372036854775807',
                      'Success',
                      True,
                      '9223372036854775807'),
          'Min Invalid': ('-9223372036854775809',
                          'Failure',
                          False,
                          ''),
          'Minimum': ('-9223372036854775808',
                      'Success',
                      True,
                      '-9223372036854775808')},
 'numeric(19,0)': {'Max Invalid': ('9223372036854775808',
                                   'Failure',
                                   False,
                                   ''),
                   'Maximum': ('9223372036854775807',
                               'Success',
                               True,
```

```
                                    '9223372036854775807'),
                        'Min Invalid': ('-9223372036854775809',
                                        'Failure',
                                        False,
                                        ''),
                        'Minimum': ('-9223372036854775808',
                                    'Success',
                                    True,
                                    '-9223372036854775808')},
'numeric(38,0)': {'Max Invalid': ('100000000000000000000000000
                                        'Failure',
                                        False,
                                        ''),
                        'Maximum': ('999999999999999999999999999
                                    'Success',
                                    True,
                                    '999999999999999999999999999
                        'Min Invalid': ('-100000000000000000000000
                                        'Failure',
                                        False,
                                        ''),
                        'Minimum': ('-999999999999999999999999999
                                    'Success',
                                    True,
                                    '-999999999999999999999999999
'time': {'Maximum': ('23:59:59.999999',
                     'Success',
                     True,
                     '23:59:59.999999'),
         'Minimum': ('00:00:00.000000',
                     'Success',
                     True,
                     '00:00:00')},
'timestamp': {'Max Invalid': ('394277-01-01 '
                              '00:00:00.000000 '
                              'AD',
                              'Failure',
                              False,
                              ''),
              'Maximum': ('294276-12-31 '
                          '23:59:59.999999 AD',
                          'Success',
                          True,
                          '294276-12-31 '
                          '23:59:59.999999'),
              'Min Invalid': ('4714-11-23 '
                              '23:59:59.999999 '
                              'BC',
                              'Failure',
                              False,
```

```
                                  ''),
                        'Minimum': ('4714-11-24 '
                                    '00:00:00.000000 BC',
                                    'Success',
                                    True,
                                    '4714-11-24 00:00:00 '
                                    'BC'),
                        'Rounding #1': ('2020-01-01 '
                                        '00:00:00.9999994 '
                                        'AD',
                                        'Success',
                                        True,
                                        '2020-01-01 '
                                        '00:00:00.999999'),
                        'Rounding #2': ('2020-01-01 '
                                        '00:00:00.9999995 '
                                        'AD',
                                        'Failure',
                                        False,
                                        '')},
 'timestamptz': {'Max Invalid': ('294277-01-09 '
                                 '04:00:54.775808+0000 '
                                 'AD',
                                 'Failure',
                                 False,
                                 ''),
                 'Maximum': ('294277-01-09 '
                             '04:00:54.775807+0000 '
                             'AD',
                             'Success',
                             True,
                             '294277-01-09 '
                             '04:00:54.775807+00'),
                 'Min Invalid': ('4714-11-23 '
                                 '23:59:59.999999+0000 '
                                 'BC',
                                 'Failure',
                                 False,
                                 ''),
                 'Minimum': ('4714-11-24 '
                             '00:00:00.000000+0000 '
                             'BC',
                             'Success',
                             True,
                             '4714-11-24 '
                             '00:00:00+00 BC'),
                 'Overflow #1': ('294277-01-09 '
                                 '03:59:54.775807-0001 '
                                 'AD',
                                 'Success',
```

```
                                        True,
                                        '294277-01-09 '
                                        '04:00:54.775807+00'),
                             'Overflow #2': ('294277-01-09 '
                                        '03:59:54.775807-0002 '
                                        'AD',
                                        'Failure',
                                        False,
                                        ''),
                             'Rounding #1': ('2020-01-01 '
                                        '00:00:00.9999994 '
                                        'AD',
                                        'Success',
                                        True,
                                        '2020-01-01 '
                                        '00:00:00.999999+00'),
                             'Rounding #2': ('2020-01-01 '
                                        '00:00:00.9999995 '
                                        'AD',
                                        'Failure',
                                        False,
                                        ''),
                             'Underflow #1': ('4714-11-24 '
                                         '00:01:00.000000+0001 '
                                         'BC',
                                         'Success',
                                         True,
                                         '4714-11-24 '
                                         '00:00:00+00 '
                                         'BC'),
                             'Underflow #2': ('4714-11-24 '
                                         '00:01:00.000000+0002 '
                                         'BC',
                                         'Failure',
                                         False,
                                         '')},
 'timetz': {'Max Invalid': ('23:59:59.999999+1600',
                            'Failure',
                            False,
                            ''),
            'Maximum': ('23:59:59.999999+1559',
                        'Success',
                        True,
                        '08:00:59.999999+00'),
            'Min Invalid': ('00:00:00.000000-1600',
                            'Failure',
                            False,
                            ''),
            'Minimum': ('00:00:00.000000-1559',
                        'Success',
```

```
                                                    True,
                                                    '15:59:00+00')}}}},
 'tests': {'dc2.large': {2: {}}},
 'versions': {'dc2.large': {2: 'PostgreSQL 8.0.2 on i686-pc-linux-gnu, '
                               'compiled by GCC gcc (GCC) 3.4.2 20041017 (Red '
                               'Hat 3.4.2-6.fc3), Redshift 1.0.30840'}}}
```