Introduction to Encodings, and the Raw Encoding

Max Ganz II @ Redshift Observatory

24th October 2021

Abstract

The official documentation is significantly out of date with regard to which encodings support which data types. The white paper presents a programmatic enumeration of encodings and which encodings support which data types, and determines how many values of each data type can be stored in a single block, which reveals that the `boolean` type is 1 bit per value, and that columns which are `NULL` (as opposed to `NOT NULL`) consume an additional 1 bit of store per value, except for `varchar`, which consumes an additional 1 byte of store per value.

Contents

Introduction

Redshift is a column-store relational database, which means that each column in a table is stored independently.

It is often the case that the data in a single column has similar characteristics - for example, it might be all names, or ages, or might be integer values within a given range; in other words, data which is far from randomly distributed across the value range for the data type of the column.

This provides an opportunity for unusually effective data compression, as well as an opportunity to blunder terribly, as Redshift offers a range of data compression methods (known in the Redshift documentation as “encodings”), most of which work spectacularly well with and only with data which expresses the suitable characteristics for that data compression method (and spectacularly badly with data which lacks those suitable characteristics).

It is then necessary to understand the type of data characteristics suitable for each of the data compression methods offered by Redshift, as well of course as the properties, behaviours and limitations of the data compression methods, so the good choices can be made when selecting data compression for columns.

This document is one in a series, each of which examines one data compression method offered by Redshift, which here investigates the `raw` encoding.

(I normally do not work, but currently I am on a full-time contract, so if I spent the time necessary to produce a single document with all encodings, you would hear nothing from me for many weeks. Once all the encodings have been documented, a single document, “Encodings”, will be released.)

Test Method

First, the encodings available to Redshift are enumerated by using the function `format_encoding( int4 )`. This function takes an argument between 0 and 255 (despite taking an `int4`) which is the ID of an encoding, and returns a string which is the name of the encoding, or the string “unknown”, if the ID has no matching encoding.

Second, we then enumerate which data types can use which encodings, by obtaining a complete list of data types from `pg_type`, and making a table for every combination of data type and encoding, and noting which combinations are permitted and which are not.

(For `char` and `varchar`, we use a small selection of variants, where we independently vary both the DDL lengths and actual length of the strings. Note that `geometry`, `hllsketch` and `super` are not investigated, because I’ve not yet learned about them, so I have only superficial knowledge of how they work, not enough to investigate them here. When I do, I’ll update this document.)

Third, we then find how many rows of each raw encoded data type fit into a single block.

Results

The results are given here for ease of reference, but they are primarily presented, piece by piece along with explanation, in the Discussion.

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

The script used to generated these results in designed to readers to use, and is available here.

Test duration, excluding server bring-up and shut-down, was 1641 seconds.

dc2.large, 2 nodes (1.0.32574)

Enumerated Encodings

ID	Name
0	none
1	bytedict
2	delta
3	lzo
4	runlength
5	delta32k
7	text255
11	globaldict256
12	globaldict64k
13	globaldict4B
15	mostly8
16	mostly16
17	mostly32
18	text32k
19	zstd
20	az64
128	none
131	lzo
133	delta32k
147	zstd
148	az64

Valid Encodings for Data Types

Data Type	Encodings
bool	raw, runlength, zstd
bpchar	bytedict, lzo, raw, runlength, zstd
char	bytedict, lzo, raw, runlength, zstd
date	az64, bytedict, delta, delta32k, lzo, raw, runlength, zstd
float4	bytedict, raw, runlength, zstd
float8	bytedict, raw, runlength, zstd
geometry	raw
hllsketch	raw
int2	az64, bytedict, delta, lzo, mostly8, raw, runlength, zstd
int4	az64, bytedict, delta, delta32k, lzo, mostly16, mostly8, raw, runlength, zstd
int8	az64, bytedict, delta, delta32k, lzo, mostly16, mostly32, mostly8, raw, runlength, zstd
numeric	az64, bytedict, delta, delta32k, lzo, mostly16, mostly32, mostly8, raw, runlength, zstd
super	lzo, raw, zstd
text	bytedict, lzo, raw, runlength, text32k, zstd
time	az64, bytedict, delta, delta32k, lzo, raw, runlength, zstd
timestamp	az64, bytedict, delta, delta32k, lzo, raw, runlength, zstd
timestamptz	az64, bytedict, delta, delta32k, lzo, raw, runlength, zstd
timetz	az64, bytedict, delta, delta32k, lzo, raw, runlength, zstd
varchar	bytedict, lzo, raw, runlength, text255, text32k, zstd

Valid Data Types for Encodings

Encoding	Data Types
az64	date, int2, int4, int8, numeric, time, timestamp, timestamptz, timetz
bytedict	bpchar, char, date, float4, float8, int2, int4, int8, numeric, text, time, timestamp, timestamptz, timetz, varchar

Encoding	Data Types
delta	date, int2, int4, int8, numeric, time, timestamp, timestamptz, timetz
delta32k	date, int4, int8, numeric, time, timestamp, timestamptz, timetz
globaldict256	
globaldict4B	
globaldict64k	
lzo	bpchar, char, date, int2, int4, int8, numeric, super, text, time, timestamp, timestamptz, timetz, varchar
mostly16	int4, int8, numeric
mostly32	int8, numeric
mostly8	int2, int4, int8, numeric
raw	bool, bpchar, char, date, float4, float8, geometry, hllsketch, int2, int4, int8, numeric, super, text, time, timestamp, timestamptz, timetz, varchar
runlength	bool, bpchar, char, date, float4, float8, int2, int4, int8, numeric, text, time, timestamp, timestamptz, timetz, varchar
text255	varchar
text32k	text, varchar
zstd	bool, bpchar, char, date, float4, float8, int2, int4, int8, numeric, super, text, time, timestamp, timestamptz, timetz, varchar

raw Encoding

Data Type	Values/Block (NN)	Values/Block (N)	Diff	Notes
boolean	8,387,697	4,193,849	4,193,848	
char(0001)	1,048,463	931,967	116,496	one character string
char(0008)	131,051	129,035	2,016	one character string
char(0008)	131,051	129,035	2,016	full length string
char(0064)	16,375	16,343	32	one character string
char(0064)	16,375	16,343	32	full length string
char(4096)	248	248	0	one character string
char(4096)	248	248	0	full length string
date	262,085	254,143	7,942	
float4	262,085	254,143	7,942	
float8	130,994	128,978	2,016	
int2	524,219	493,382	30,837	
int4	262,085	254,143	7,942	
int8	130,994	128,978	2,016	
time	130,994	128,978	2,016	

Data Type	Values/Block (NN)	Values/Block (N)	Diff	Notes
timestamp	130,994	128,978	2,016	
timestamptz	130,994	128,978	2,016	
timetz	130,994	128,978	2,016	
numeric(1,0)	130,994	128,978	2,016	
numeric(19,0)	130,994	128,978	2,016	
numeric(20,0)	65,401	64,894	507	
numeric(38,0)	65,401	64,894	507	
varchar(00001)	209,694	174,745	34,949	one character string
varchar(00008)	209,694	174,745	34,949	one character string
varchar(00008)	87,372	80,651	6,721	full length string
varchar(00064)	209,694	174,745	34,949	one character string
varchar(00064)	15,418	15,195	223	full length string
varchar(04096)	209,694	174,745	34,949	one character string
varchar(04096)	255	255	0	full length string
varchar(65535)	209,694	174,745	34,949	one character string
varchar(65535)	15	15	0	full length string

Data Type	Bits/Value (NN)	Bits/Value (N)	Unused Bytes (NN)	Unused Bytes (N)	Notes
boolean	1	2	113	113	
char(0001)	8	9	113	113	one character string
char(0008)	64	65	168	166	one character string
char(0008)	64	65	168	166	full length string
char(0064)	512	513	576	581	one character string
char(0064)	512	513	576	581	full length string
date	32	33	236	236	
float4	32	33	236	236	
float8	64	65	624	629	
int2	16	17	138	139	
int4	32	33	236	236	
int8	64	65	624	629	
time	64	65	624	629	
timestamp	64	65	624	629	
timestamptz	64	65	624	629	
timetz	64	65	624	629	
numeric(1,0)	64	65	624	629	
numeric(19,0)	64	65	624	629	

Data Type	Bits/Value (NN)	Bits/Value (N)	Unused Bytes (NN)	Unused Bytes (N)	Notes
numeric(20,0)	128	129	2160	2160	
numeric(38,0)	128	129	2160	2160	
varchar(00001)	40	48	106	106	one character string
varchar(00008)	40	48	106	106	one character string
varchar(00008)	96	104	112	113	full length string
varchar(00064)	40	48	106	106	one character string
varchar(00064)	544	552	152	121	full length string
varchar(04096)	40	48	106	106	one character string
varchar(65535)	40	48	106	106	one character string

Data Type	Bits/Value (NN)	Bits/Value (N)	Unused Bytes (NN)	Unused Bytes (N)	Notes
char(4096)	32768	32769	32768	32737	one character string
char(4096)	32768	32769	32768	32737	full length string
varchar(04096)	32768	32769	4096	4064	full length string
varchar(65535)	524312	524320	65491	65476	full length string

Discussion

To begin with, we enumerate all the encodings Redshift knows about.

There's a function, `format_encoding( int4 )`, which takes a single `int4` argument which is an encoding ID (which range from 0 to 255 - outside this range and you get an error), and returns a string which is the name of the encoding, or "unknown" if there is no encoding for the given ID.

Setting aside all "unknown"s, we find the following;

ID	Name
0	none
1	bytedict
2	delta
3	lzo
4	runlength
5	delta32k
7	text255
11	globaldict256
12	globaldict64k
13	globaldict4B
15	mostly8
16	mostly16
17	mostly32
18	text32k
19	zstd
20	az64
128	none
131	lzo
133	delta32k
147	zstd
148	az64

There are a couple of items of note;

1. `none` means `raw`.
2. Some encodings have more than one ID.
3. There's a set of three `globaldict` encodings which are not mentioned in the documentation.

Next, let's check to see which data types can use which encodings.

(We can only specify encodings by their names, so we can't try to use the different IDs of encodings with multiple IDs. Also note normally I always use Redshift internal names, so say `int8`, which is the name you find in the system tables, rather than `bigint`, which is an alias, but with `raw` encoding the internal name is `none` but you can't use that with `CREATE TABLE` - it only understands `raw`.)

Valid Encodings for Data Types

Data Type	Encodings
bool	raw, runlength, zstd
bpchar	bytedict, lzo, raw, runlength, zstd
char	bytedict, lzo, raw, runlength, zstd
date	az64, bytedict, delta, delta32k, lzo, raw, runlength, zstd
float4	bytedict, raw, runlength, zstd
float8	bytedict, raw, runlength, zstd
geometry	raw
hllsketch	raw
int2	az64, bytedict, delta, lzo, mostly8, raw, runlength, zstd
int4	az64, bytedict, delta, delta32k, lzo, mostly16, mostly8, raw, runlength, zstd
int8	az64, bytedict, delta, delta32k, lzo, mostly16, mostly32, mostly8, raw, runlength, zstd
numeric	az64, bytedict, delta, delta32k, lzo, mostly16, mostly32, mostly8, raw, runlength, zstd
super	lzo, raw, zstd
text	bytedict, lzo, raw, runlength, text32k, zstd
time	az64, bytedict, delta, delta32k, lzo, raw, runlength, zstd
timestamp	az64, bytedict, delta, delta32k, lzo, raw, runlength, zstd
timestamptz	az64, bytedict, delta, delta32k, lzo, raw, runlength, zstd
timetz	az64, bytedict, delta, delta32k, lzo, raw, runlength, zstd
varchar	bytedict, lzo, raw, runlength, text255, text32k, zstd

Valid Data Types for Encodings

Encoding	Data Types
az64	date, int2, int4, int8, numeric, time, timestamp, timestamptz, timetz
bytedict	bpchar, char, date, float4, float8, int2, int4, int8, numeric, text, time, timestamp, timestamptz, timetz, varchar
delta	date, int2, int4, int8, numeric, time, timestamp, timestamptz, timetz
delta32k	date, int4, int8, numeric, time, timestamp, timestamptz, timetz
globaldict256	
globaldict4B	
globaldict64k	
lzo	bpchar, char, date, int2, int4, int8, numeric, super, text, time, timestamp, timestamptz, timetz, varchar
mostly16	int4, int8, numeric
mostly32	int8, numeric
mostly8	int2, int4, int8, numeric
raw	bool, bpchar, char, date, float4, float8, geometry, hllsketch, int2, int4, int8, numeric, super, text, time, timestamp, timestamptz, timetz, varchar
runlength	bool, bpchar, char, date, float4, float8, int2, int4, int8, numeric, text, time, timestamp, timestamptz, timetz, varchar
text255	varchar
text32k	text, varchar
zstd	bool, bpchar, char, date, float4, float8, int2, int4, int8, numeric, super, text, time, timestamp, timestamptz, timetz, varchar

The docs page for which encodings support which data types is here. It seems evidently hand-maintained, as it is out of date.

1. All encodings which can support `time` and `timetz` (`az64`, `bytedict`, ‘delta’, `delta32k`, `lzo`, `runlength`, `zstd`) are missing support for those two data types.
2. ‘delta’ and `delta32k` support for `timestamptz` is missing.
3. ‘lzo’ and ‘zstd’ support for ‘super’ is missing.

I may be wrong, but it seems obvious to me any serious documentation for a continually evolving software product must at least in part to be automatically generated if it is to avoid becoming increasingly inaccurate over time.

It also would seem if there’s any ongoing checking of the docs, it is ineffective, since we see here the most simple, basic and fundamental information is inaccurate.

Having then set the scene, both as you will see for the quality of the documentation, as well as for encodings, let us turn to each encoding in turn, and see for each what we can find out.

We turn now to the `raw` encoding.

Being what it is, there's nothing to say about how `raw` encodes - but by being raw, by doing nothing, it allows us to examine other properties of storing rows in blocks. In particular, a critical question turns out to be how many rows of each data type fit into a single block.

This turns out to be an excellent question, because we find that first, `NULL` or `NOT NULL` matters, and, secondly, it's not just a case of there being as many rows as will fit in one megabyte; there's some unused space, and it seems to be the longer the data type, the more unused space there is.

Here we see the number of values stored per block when `NOT NULL` is set "(NN)", the number when `NULL` is set "(N)" and the difference between the two.

Data Type	Values/Block (NN)	Values/Block (N)	Diff	Notes
boolean	8,387,697	4,193,849	4,193,848	
char(0001)	1,048,463	931,967	116,496	one character string
char(0008)	131,051	129,035	2,016	one character string
char(0008)	131,051	129,035	2,016	full length string
char(0064)	16,375	16,343	32	one character string
char(0064)	16,375	16,343	32	full length string
char(4096)	248	248	0	one character string
char(4096)	248	248	0	full length string
date	262,085	254,143	7,942	
float4	262,085	254,143	7,942	
float8	130,994	128,978	2,016	
int2	524,219	493,382	30,837	
int4	262,085	254,143	7,942	
int8	130,994	128,978	2,016	
time	130,994	128,978	2,016	
timestamp	130,994	128,978	2,016	
timestamptz	130,994	128,978	2,016	
timetz	130,994	128,978	2,016	
numeric(1,0)	130,994	128,978	2,016	
numeric(19,0)	130,994	128,978	2,016	
numeric(20,0)	65,401	64,894	507	
numeric(38,0)	65,401	64,894	507	
varchar(00001)	209,694	174,745	34,949	one character string
varchar(00008)	209,694	174,745	34,949	one character string
varchar(00008)	87,372	80,651	6,721	full length string
varchar(00064)	209,694	174,745	34,949	one character string
varchar(00064)	15,418	15,195	223	full length string
varchar(04096)	209,694	174,745	34,949	one character string
varchar(04096)	255	255	0	full length string

Data Type	Values/Block (NN)	Values/Block (N)	Diff	Notes
varchar(65535)	209,694	174,745	34,949	one character string
varchar(65535)	15	15	0	full length string

The first and most startling observation is the huge number of values stored by `boolean`, which must be 1 bit per value to be storing 8.3m values in a one megabyte block. We also note that setting `NULL` (as opposed to `NOT NULL`) roughly halves the number of values - clearly, a 1 bit flag per value is used to indicate whether a value is `NULL` or not. More on this below.

The official documentation for `boolean`, found here, states `boolean` is 1 byte per value, and, what's more, that it is 1 byte whether true, false or `NULL`, which is not just wrong, but also misleads readers as to how `NULL` is handled.

Moving on to `char`, we can see that the maximum length of the char in the DDL determines the store required; the actual length of the string is not relevant. When we get to `char(4096)`, there are only 248 values in a block; each when `NULL` requires one more bit to store, but 248 bits is small enough that it makes no difference to the number of values which can be stored.

The `numeric` type is worth a mention, in that precision 1 to 19 gives an 8 byte value, precision 20 to 38 gives a 16 byte value. This is why I select the precisions 1, 19, 20 and 38, to demonstrate the transition.

Finally, coming to `varchar`, we find that this data type requires 1 byte, rather than 1 bit, to indicate `NULL`.

So, now we now directly from `STV_BLOCKLIST` how many values are in a block, both for `NOT NULL` and `NULL`. We can then divide the size of the block by the number of values, to see how many bits are being used per value.

There is in fact always some unused space, but the number of bits must be an integer, so if we end up with say 32.2 bits being used per value, then the number of bits must be 32, and we can compute the amount of unused space from the fractional part of the number.

Note here we have *bits* per value, but *bytes* of unused space.

Data Type	Bits/Value (NN)	Bits/Value (N)	Unused Bytes (NN)	Unused Bytes (N)	Notes
boolean	1	2	113	113	
char(0001)	8	9	113	113	one character string
char(0008)	64	65	168	166	one character string
char(0008)	64	65	168	166	full length string

Data Type	Bits/Value (NN)	Bits/Value (N)	Unused Bytes (NN)	Unused Bytes (N)	Notes
char(0064)	512	513	576	581	one character string
char(0064)	512	513	576	581	full length string
date	32	33	236	236	
float4	32	33	236	236	
float8	64	65	624	629	
int2	16	17	138	139	
int4	32	33	236	236	
int8	64	65	624	629	
time	64	65	624	629	
timestamp	64	65	624	629	
timestamptz	64	65	624	629	
timetz	64	65	624	629	
numeric(1,0)	64	65	624	629	
numeric(19,0)	64	65	624	629	
numeric(20,0)	128	129	2160	2160	
numeric(38,0)	128	129	2160	2160	
varchar(00001)	40	48	106	106	one character string
varchar(00008)	40	48	106	106	one character string
varchar(00008)	96	104	112	113	full length string
varchar(00064)	40	48	106	106	one character string
varchar(00064)	544	552	152	121	full length string
varchar(04096)	40	48	106	106	one character string
varchar(65535)	40	48	106	106	one character string

So, quite a few matters to note;

1. `boolean` is 1 bit per value
2. `char` always uses the maximum length specified in the DDL
3. all data types, except `varchar`, use one additional bit per value if the column is `NULL` (as opposed to `NOT NULL`), which will matter for small data types once you get into Big Data

4. `varchar` stores only the actual length of the string, plus a four byte header (which presumably indicates length)
5. `varchar` uses 1 byte per value if the column is `NULL`
6. As mentioned in a previous white paper, `numeric` is 8 bytes up to precision 19, then becomes 16 bytes. The actual value stored makes no difference; it is and only is the DDL which determines the data type length.

So it is then that a `varchar(1) NULL` is 48 bits in length, carrying 8 bits of data. Don't do that - if you don't need UTF-8, use a `char(1) NULL`, which is 9 bits per value.

Now, we computed the unused space by dividing the size of the block by the number of values, to see how many bits are being used per value. However, if the number of values stored in one block is the same for both `NULL` and `NOT NULL` (as happens with the long `char` and `varchar` data types), this approach partially fails, in that it ends up thinking the amount of unused space is the same in both cases - we already known, from what we've seen above, that this is not so. All that's actually happening is the overheads of handling `NULL` are so small, given the very small number of values, that they do not change the number of values which can be stored in one block.

In these special cases, the four of them below, I have in the script manually specified the number of bits per value, based on the knowledge from the table above, and *then* computed the unused space for `NULL` and `NOT NULL`.

Data Type	Bits/Value (NN)	Bits/Value (N)	Unused Bytes (NN)	Unused Bytes (N)	Notes
char(4096)	32768	32769	32768	32737	one character string
char(4096)	32768	32769	32768	32737	full length string
varchar(04096)	32768	32769	4096	4064	full length string
varchar(65535)	524312	524320	65491	65476	full length string

What's interesting here is that the amount of unused space is large. For `varchar` it makes sense - the space remaining is non-trivial, but it's always smaller than the amount needed for one more value to be stored - but for `char`, it doesn't make sense. A `char(4096)` has 32,768 unused bytes in each block. There are 248 values being stored, another 8 values could be stored (7 if `NULL`). What gives?

Well, I have a bit of a suspicion this - the unused space - is being done to improve `VACUUM` performance. If a user inserts only a few rows, you can maybe get away with only needing to resort the individual blocks which each take some of the new rows, because they have room to take them; it saves you needing to resort every block *after* the blocks which take new rows, which you would have to do if each block was already completely full.

Conclusions

The official documentation is out of date with regard to which encodings support which data types.

1. All encodings which can support `time` and `timetz` (`az64`, `bytedict`, `delta`, `delta32k`, `lzo`, `runlength`, `zstd`) are missing support for those two data types.
2. 'delta' and `delta32k` support for `timestamptz` is missing.
3. 'lzo' and 'zstd' support for 'super' is missing.

The `boolean` data type is 1 bit in size (the documentation states 1 byte; this is incorrect).

Setting a column to `NULL` (as opposed to `NOT NULL`) requires an additional 1 bit of store per value, except for `varchar`, which requires an additional 1 byte of store per value.

Blocks when full, in the sense that an additional value will lead to a new block being formed, have a little unused space. The amount varies by data type, and increases as the data type becomes larger (in terms of bytes per value). Typically the unused space is small, on the order of hundreds of bytes, but for long `char` and `varchar` strings (remembering that `char` always uses the full length of the DDL length, but `varchar` only uses the actual length of the string, plus a four byte length header) the unused space becomes larger, with `char(4096)` leaving 32737 bytes unused and `varchar(65535)` with a full length string leaving 65,476 bytes unused (the latter being understandable, as there is not enough room for another value).

I have a suspicion the unused space is to help with `VACUUM` performance in certain situations, but it's a guess.

Revision History

v1

- Initial release.

v2

- Added “About the Author”. made site name in title a link, and made each chapter start a new page.
- Updated links to amazonredshiftresearcproject.org to redshiftresearcproject.org.

v3

- Web-site name changed to “Redshift Observatory”.
- Updated links from redshiftresearcproject.org to redshift-observatory.ch.

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: {'data_type_encodings': {'bool': ['raw',
                                                               'runlength',
                                                               'zstd'],
                                                      'bpchar': ['bytedict',
                                                                 'lzo',
                                                                 'raw',
                                                                 'runlength',
                                                                 'zstd'],
                                                      'char': ['bytedict',
                                                               'lzo',
                                                               'raw',
                                                               'runlength',
                                                               'zstd'],
                                                      'date': ['az64',
                                                               'bytedict',
                                                               'delta',
                                                               'delta32k',
                                                               'lzo',
                                                               'raw',
                                                               'runlength',
                                                               'zstd'],
                                                      'float4': ['bytedict',
                                                                 'raw',
                                                                 'runlength',
                                                                 'zstd'],
                                                      'float8': ['bytedict',
                                                                 'raw',
                                                                 'runlength',
                                                                 'zstd'],
                                                      'geometry': ['raw'],
                                                      'hllsketch': ['raw'],
                                                      'int2': ['az64',
                                                               'bytedict',
         'delta',
         'lzo',
         'mostly8',
         'raw',
         'runlength',
         'zstd'],
'int4': ['az64',
         'bytedict',
         'delta',
         'delta32k',
         'lzo',
         'mostly16',
         'mostly8',
         'raw',
         'runlength',
         'zstd'],
'int8': ['az64',
         'bytedict',
         'delta',
         'delta32k',
         'lzo',
         'mostly16',
         'mostly32',
         'mostly8',
         'raw',
         'runlength',
         'zstd'],
'numeric': ['az64',
            'bytedict',
            'delta',
            'delta32k',
            'lzo',
            'mostly16',
            'mostly32',
            'mostly8',
            'raw',
            'runlength',
            'zstd'],
'super': ['lzo',
          'raw',
          'zstd'],
'text': ['bytedict',
         'lzo',
         'raw',
         'runlength',
         'text32k',
         'zstd'],
'time': ['az64',
         'bytedict',
         'delta',
                                 'delta32k',
                                 'lzo',
                                 'raw',
                                 'runlength',
                                 'zstd'],
                        'timestamp': ['az64',
                                      'bytedict',
                                      'delta',
                                      'delta32k',
                                      'lzo',
                                      'raw',
                                      'runlength',
                                      'zstd'],
                        'timestamptz': ['az64',
                                        'bytedict',
                                        'delta',
                                        'delta32k',
                                        'lzo',
                                        'raw',
                                        'runlength',
                                        'zstd'],
                        'timetz': ['az64',
                                   'bytedict',
                                   'delta',
                                   'delta32k',
                                   'lzo',
                                   'raw',
                                   'runlength',
                                   'zstd'],
                        'varchar': ['bytedict',
                                    'lzo',
                                    'raw',
                                    'runlength',
                                    'text255',
                                    'text32k',
                                    'zstd']},
'data_type_values_per_block': [('boolean',
                                8387697,
                                4193849,
                                4193848,
                                ''),
                               ('char(0001)',
                                1048463,
                                931967,
                                116496,
                                'one character '
                                'string'),
                               ('char(0008)',
                                131051,
                                129035,
 2016,
 'one character '
 'string'),
('char(0008)',
 131051,
 129035,
 2016,
 'full length '
 'string'),
('char(0064)',
 16375,
 16343,
 32,
 'one character '
 'string'),
('char(0064)',
 16375,
 16343,
 32,
 'full length '
 'string'),
('char(4096)',
 248,
 248,
 0,
 'one character '
 'string'),
('char(4096)',
 248,
 248,
 0,
 'full length '
 'string'),
('date',
 262085,
 254143,
 7942,
 ''),
('float4',
 262085,
 254143,
 7942,
 ''),
('float8',
 130994,
 128978,
 2016,
 ''),
('int2',
 524219,
 493382,
 30837,
 ''),
('int4',
 262085,
 254143,
 7942,
 ''),
('int8',
 130994,
 128978,
 2016,
 ''),
('time',
 130994,
 128978,
 2016,
 ''),
('timestamp',
 130994,
 128978,
 2016,
 ''),
('timestamptz',
 130994,
 128978,
 2016,
 ''),
('timetz',
 130994,
 128978,
 2016,
 ''),
('numeric(1,0)',
 130994,
 128978,
 2016,
 ''),
('numeric(19,0)',
 130994,
 128978,
 2016,
 ''),
('numeric(20,0)',
 65401,
 64894,
 507,
 ''),
('numeric(38,0)',
 65401,
 64894,
 507,
 ''),
('varchar(00001)',
 209694,
 174745,
 34949,
 'one character '
 'string'),
('varchar(00008)',
 209694,
 174745,
 34949,
 'one character '
 'string'),
('varchar(00008)',
 87372,
 80651,
 6721,
 'full length '
 'string'),
('varchar(00064)',
 209694,
 174745,
 34949,
 'one character '
 'string'),
('varchar(00064)',
 15418,
 15195,
 223,
 'full length '
 'string'),
('varchar(04096)',
 209694,
 174745,
 34949,
 'one character '
 'string'),
('varchar(04096)',
 255,
 255,
 0,
 'full length '
 'string'),
('varchar(65535)',
 209694,
 174745,
 34949,
 'one character '
```

```
                                'string'),
                               ('varchar(65535)',
                                15,
                                15,
                                0,
                                'full length '
                                'string')],
'encodings_data_type': {'az64': ['date',
                                 'int2',
                                 'int4',
                                 'int8',
                                 'numeric',
                                 'time',
                                 'timestamp',
                                 'timestamptz',
                                 'timetz'],
                        'bytedict': ['bpchar',
                                     'char',
                                     'date',
                                     'float4',
                                     'float8',
                                     'int2',
                                     'int4',
                                     'int8',
                                     'numeric',
                                     'text',
                                     'time',
                                     'timestamp',
                                     'timestamptz',
                                     'timetz',
                                     'varchar'],
                        'delta': ['date',
                                  'int2',
                                  'int4',
                                  'int8',
                                  'numeric',
                                  'time',
                                  'timestamp',
                                  'timestamptz',
                                  'timetz'],
                        'delta32k': ['date',
                                     'int4',
                                     'int8',
                                     'numeric',
                                     'time',
                                     'timestamp',
                                     'timestamptz',
                                     'timetz'],
                        'globaldict256': [],
                        'globaldict4B': [],
'globaldict64k': [],
'lzo': ['bpchar',
        'char',
        'date',
        'int2',
        'int4',
        'int8',
        'numeric',
        'super',
        'text',
        'time',
        'timestamp',
        'timestamptz',
        'timetz',
        'varchar'],
'mostly16': ['int4',
             'int8',
             'numeric'],
'mostly32': ['int8',
             'numeric'],
'mostly8': ['int2',
            'int4',
            'int8',
            'numeric'],
'raw': ['bool',
        'bpchar',
        'char',
        'date',
        'float4',
        'float8',
        'geometry',
        'hllsketch',
        'int2',
        'int4',
        'int8',
        'numeric',
        'super',
        'text',
        'time',
        'timestamp',
        'timestamptz',
        'timetz',
        'varchar'],
'runlength': ['bool',
              'bpchar',
              'char',
              'date',
              'float4',
              'float8',
              'int2',
                                      'int4',
                                      'int8',
                                      'numeric',
                                      'text',
                                      'time',
                                      'timestamp',
                                      'timestamptz',
                                      'timetz',
                                      'varchar'],
                        'text255': ['varchar'],
                        'text32k': ['text',
                                    'varchar'],
                        'zstd': ['bool',
                                 'bpchar',
                                 'char',
                                 'date',
                                 'float4',
                                 'float8',
                                 'int2',
                                 'int4',
                                 'int8',
                                 'numeric',
                                 'super',
                                 'text',
                                 'time',
                                 'timestamp',
                                 'timestamptz',
                                 'timetz',
                                 'varchar']},
'enumerated_encodings': ['INFO:  0,none',
                         'INFO:  1,bytedict',
                         'INFO:  2,delta',
                         'INFO:  3,lzo',
                         'INFO:  4,runlength',
                         'INFO:  5,delta32k',
                         'INFO:  7,text255',
                         'INFO:  '
                         '11,globaldict256',
                         'INFO:  '
                         '12,globaldict64k',
                         'INFO:  13,globaldict4B',
                         'INFO:  15,mostly8',
                         'INFO:  16,mostly16',
                         'INFO:  17,mostly32',
                         'INFO:  18,text32k',
                         'INFO:  19,zstd',
                         'INFO:  20,az64',
                         'INFO:  128,none',
                         'INFO:  131,lzo',
                         'INFO:  133,delta32k',
                                                       'INFO:  147,zstd',
                                                       'INFO:  148,az64']}}},
 '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 1.0.32574'}}}
```

About the Author

I am a C programmer - kernel development, high performance computing, networking, data structures and so on.

I read the C. J. Date book, the classic text on relational database theory, and having learned the principles, wrote a relational database from scratch in C, which purely by chance set me up quite nicely for what came next, moving into data engineering in late 2011, when I joined as the back-end engineer two friends in their startup.

In that startup, I began using Redshift the day it came out, in 2012 (we had been trying to get into the beta programme).

We were early, heavy users for a year and a half, and I ending up having monthly one-to-one meetings with one of the Redshift team managers, where one or two features which are in Redshift today originate from suggestions made in those meetings, such as the distribution style `ALL`.

Once that was done, after a couple of years of non-Redshift data engineering work, I returned to Redshift work, and then in about mid-2018 contracted with a publisher to write a book about Redshift.

The book was largely written but it became apparent I wanted to do a lot of things which couldn't be done with a book - republish on every new Redshift release, for example - and so in the end I stepped back from the contract and developed the web-site, where I publish investigation into, and ongoing monitoring of, Redshift.

So for many years now I've been investigating Redshift sub-systems full-time, one by one, and this site and these investigations are as far as I know the and the only source of this kind of information about Redshift.

Redshift Cluster Cost Reduction Service

I provide consultancy services for Redshift - advice, design, training, getting failing systems back on their feet pronto, the usual gamut - but in particular offer a Redshift cluster cost reduction service, where the fee is and only is one month of the savings made.

Broadly speaking, to give guidance, savings are expected fall into one of two categories; either something like 20%, or something like 80%. The former is for systems where the business use case is such that Redshift cannot be operated correctly, and this outcome requires no fundamental re-engineering work, the latter is for systems where Redshift can be operated correctly, and usually requires fundamental re-engineering work (which you may or may not wish to engage in, despite the cost savings, in which case we're back to the 20%).

Details and contact information are on the web-site.