AQUA

Max Ganz II @ Redshift Research Project

21st September 2022 (updated 19th May 2024)

Abstract

AQUA is a solution to the problem that min-max culling (aka the Zone Map) is wholly ineffective when searching for strings with wildcard patterns, forcing full column scans, and as such all `scan` steps and only all `scan` steps which use the LIKE and/or SIMILAR TO operators are routed to AQUA, as these are the two operators offering search in strings with wildcards. AQUA is implemented as a multi-node distributed set of independent data processors, separate from RMS and from Redshift, and as such the basic design is the same as Redshift Spectrum, with each processor scanning a portion of the data in a table and returning results to the Redshift cluster, and appears to implement most if not all of the SQL functionality possible with this design, which is to say, that functionality which can be performed by examining a single row at a time in a single table, and by maintaining aggregate information from those rows; so there exists functionality which AQUA cannot implement, such as `distinct`. When AQUA is presented with a `scan` step with work AQUA cannot perform, necessarily some, most or all of the rows in the table must be returned to the Redshift cluster, such that that work can there be performed. As such, queries must be written correctly for AQUA, just as with Spectrum. AWS do not provide the information to do this, and so guesswork and experimentation must be used. When a `scan` step is routed to AQUA, AQUA downloads to itself, not to the Redshift cluster local SSD cache, the table being scanned from RMS. With the 12,988 block test table, on an idle two node `ra3.xlplus` cluster, this took 41 seconds. After this, AQUA performs more quickly than the cluster, with the time taken with test queries used for the part of the work routed to AQUA going from about 9.4 seconds to about 1.7 seconds. It is necessary then to issue a number of LIKE or SIMILAR TO queries on a table to reclaim the large initial cost of the first query, otherwise AQUA will actually *reduce* performance. There are no findings about how long must pass before a table needs to be re-read by AQUA. In the course of investigating AQUA, it was discovered that RMS is a block-based store, not a table-based store; Redshift brings into local SSD cache only those blocks which are needed to service queries. With an `ra3.xlplus` node, bandwidth to RMS is about 160mb/second, and bandwidth to the local SSD cache is about 720mb/second.

Contents

Introduction

The Hype Lives!

I feel I need to begin with the marketing for AQUA, because it took on a life of its own.

AWS issued a press release announcing general availability in April 2021;

> AQUA provides a new distributed and hardware accelerated cache that brings compute to the storage layer for Amazon Redshift and delivers up to 10x faster query performance than other enterprise cloud data warehouses.

Grammatically not the clearest, but seems to say using AQUA gives you up to 10x performance over other data warehouses (leaving open the question of what you get without AQUA =-)

However, a longer AWS tech blog post on the same day states (my italics);

> AQUA pushes the computation needed to handle *reduction and aggregation queries* closer to the data. This reduces network traffic, offloads work from the CPUs in the RA3 nodes, and allows AQUA to improve the performance of those queries by up to 10x, at no extra cost and without any code changes.

Ah. So it's 10x only for "reduction and aggregation" queries, whatever they are, and that's 10x over whatever Redshift has without AQUA, rather 10x faster than everyone else (and when AWS say "for free", bear in mind because *everything* costs money - FPGA designers do not work for nothing, nor do chip foundries - and for AWS all money in the end comes from end-users).

Now, it you look closely and read right to the end of that long tech blog post, with its many screengrabs and SQL and performance measures and so on, all showing how amazing AQUA is, at the very end in a single bullet point buried in a list of bullet points, there's this;

> AQUA is designed to deliver up to 10X performance on queries that perform large scans, aggregates, and filtering with LIKE and SIMILAR_TO predicates. Over time we expect to add support for additional queries.

Yup. Forget the blurb about large scans and so on, it's malformed grammar - AQUA is used when and only when there is a LIKE or a SIMILAR TO and

that's it.

(As an aside, I looked around, and could find no indication of any additional functionality in the years since this blog post was made.)

Meanwhile, over at TechCrunch, Forbes, Blocks & Files, and plenty of others, we find…

> AWS speeds up Redshift queries 10x with AQUA
>
> How AQUA For Amazon Redshift Performs Its Queries Up To 10X Faster Than Before
>
> Amazon Web Services is bringing compute closer to its Redshift cloud data warehouse storage to accelerate query processing by 10x.

The only 10x I can see here is the gap between marketing, the press, and reality.

Now, on the face of it, this is all profoundly underwhelming but I'm of the view - as will become clear - that AQUA is actually rather smart and addresses a particular narrow but significant scalability issue. I also think if AWS didn't spend all their time emitting hyperbole, and instead went for truth and facts, they would be in a position to explain what *is* important about this, whereas instead we have enough hot air to power an ecologically friendly mass-transit system.

AQUA Becomes Mandatory

On or about the 10th September 2022 AWS emailed end-users running RA3 clusters and who had used AQUA in the past (there was no public announcement) with the following message;

> We want to inform you that Amazon Redshift will now automatically determine when to apply performance acceleration techniques that leverage AQUA technology. Thus, the ability to enable and disable AQUA will be removed from the console starting the week of September 12, 2022.

That grandiose first statement made me think AQUA was being made mandatory.

This prompted me to drop what I was doing and spend what turned out to be the following eleven days, all day, every day, investigating while I still could (to investigate, you need to be able to compare with to without, which means being able to disable AQUA).

In fact, as time passed, it became clear *all* AWS have done is… …remove the option from the Redshift console.

From Python using `boto3`, the AQUA configuration options are unchanged, and I would imagine this is true for everything, except the Redshift console.

Additionally, I suspect "automatically determine" in fact means "will always be used if `LIKE` or `SIMILAR TO` are in use", e.g. exactly as it always was. It's just now you can't turn it off, if you start the cluster via the console.

As of 18th May 2024, I happened purely by chance to look at the `create_cluster` command for Redshift in boto3.

> AquaConfigurationStatus (string) – This parameter is retired. It does not set the AQUA configuration status. Amazon Redshift automatically determines whether to use AQUA (Advanced Query Accelerator).

To my knowledge there has been no announcement of this change. It looks like `AQUA` really is now mandatory - or rather, it's up to Redshift whether it is in use or not.

On my clusters, where I do set this parameter, I see the following;

```
"AquaConfiguration":
{
  "AquaStatus": "disabled",
  "AquaConfigurationStatus": "auto"
}
```

I've also seen this on a client's cluster. It looks to say that Redshift can enable or disabled `AQUA` as it sees fit, and it is currently disabled.

This is a fundamental problem. To put it simply : it is no longer completely safe to use `LIKE` or `SIMILAR TO` on Redshift.

The problem is that `AQUA` makes a copy of the tables in use by a query which runs on `AQUA`, and so for a query running on `AQUA` the first run is very slow (tables being copied), the excess time taken being gradually recovered over the course of further runs of that query or queries which are using the same tables.

In other words, if you run queries on `AQUA` just once or twice or a few times, `AQUA` can perfectly well hammer your system, and by a *lot*, not speed it up.

Note that I've no information on how long `AQUA` keeps its copy of a table on Redshift, or what happens when that table changes.

So, now, we have in the heart of your system an undocumented black box controlled by a third party which can sabotage your system performance and it can be enabled or disabled (or rapidly toggle, who knows?) at any time for reasons of which you know nothing, where you have no control over whether `AQUA` is enabled or disabled, and you cannot disable it if it is a problem.

The devs are lunatics - actually, I think they're academics, and wholly lack real-world experience.

There was a incident a few years back with the Redshift version string. It's been like this since day one, so ten years plus;

```
PostgreSQL 8.0.2 on i686-pc-linux-gnu, compiled by GCC gcc (GCC) 3.4.2 20041017 (Red Hat 3
```

Then one fine day back in late 2021, without warning, the devs changed it, to this;

```
dev=# select version();
                           version
-------------------------------------------------------------
```

```
 Redshift 1.0.32574 on Amazon Linux, compiled by gcc-7.3.0
(1 row)
```

It's much nicer (turned out the original version string has been lying about the GCC version for about a year, and I imagine the OS for much longer).

Only problem, as you can imagine, is that the original format string as you can see is unstructured and software around the world uses a thousand different ways to pull out the version string, and so changing the version string format broke a *lot* of systems - it broke SQLAlchemy. It broke AWS's own JDBC driver (at which point it instantly became clear the RS test suite, which has long been considered weak and questionable, could not be including even a single connection to RS over AWS's own JDBC driver).

Now you and I would know without a microsecond of thought this would happen.

I heard (directly - not via the usual channels) from someone on the Support team who said "we were going to introduce an option to use the old format of version string in the next release".

These are the devs responsible for the black box which is `AQUA` sitting in the heart of your business critical Redshift cluster.

Until this point, you could turn off everything which the devs had done which was a fundamental problem - this is the first time when it is no longer so. I knew this had to come sooner or later; the most recent close call was with auto-MV, but you can disable that in the cluster configuration. With `AQUA`, you could only work around the risk by not using `LIKE` or `SIMILAR TO` any more - and that's central `SQL` functionality.

What is AQUA?

Prior to such results as will come from this investigation, all there is to go on are the hints wrapped in marketing that AWS publish in the guise of documentation and in that material this sentence, for whatever it's worth, is the best I've found describing the implementation.

> AQUA is a cost-effective addition to Amazon Redshift–managed storage [...] (that) uses acceleration based on a field-programmable gate array (FPGA) to push as much computation as possible into the storage layer.

That sounds to me like FPGAs in the servers which are RMS, and they perform work on the blocks they read from RMS.

There's then another page, which describes, but do not actually specify, additional functionality - that when a LIKE or SIMILAR TO is present, and so AQUA is invoked, it *will* handle other WHERE clauses, and it will handle GROUP BY (although there's also a long list of limitations; there's a lot of LIKE or SIMILAR TO which will not be handled by AQUA.)

Finally, there's a YouTube video published by AWS describing AQUA. I questioned and thought improper and misleading about 90% of the statements made, which is pretty much my view of all material emitted by AWS.

So, that above is all the information I can find about AQUA and so all I know to write here.

However, I have an idea of what AQUA is in fact all about, and why it was made, and to my eye, it's a smart move.

The issue is the Zone Map.

So there's a method in computing called "min-max culling". This is implemented in Redshift as the "Zone Map". Every block on disk has stored its minimum and maximum value and so when you come to scan a column for a particular value or range of values, you can immediately know which blocks that value or those values *cannot* be in, because the value or values are all smaller than the minimum, and/or larger than the maximum, for each block.

If I have a block storing integers, and the minimum value in that block is 120 and the maximum value is 450, and I want to find those rows with the value 85, I *know* no such rows can be in that block.

This method, along with sorting, when and only when correctly operated, are how sorted relational databases provide timely SQL on Big Data.

The problem with LIKE and SIMILAR TO is that they defeat the Zone Map; it can no longer be used. You have to scan the whole column - and that's death on toast when you have your archetypal Big Data petabyte of data.

Consider; imagine we have a column of strings - let's say, names for all the people in Ukraine.

Redshift stores the first eight bytes of each string as it's value for min-max culling.

If we search for 'Victoria', then the Zone Map allows us to ignore all blocks which cannot possibly contain that string.

But what if we search for 'LIKE %Victoria%'?

Well, now we have a problem.

The Zone Map cannot help us because of the wildcards; the part of the string we might match to could be anywhere in the string, and so we can no longer eliminate blocks using min-max culling; knowing the minimum and maximum strings in a block does *not* tell you in any given string in a block contains the substring 'Victoria'.

This means we must now scan the whole column, and that, with Big Data, is game over.

If LIKE (and SIMILAR TO, which has the same problem) can be made to scale, then a fundamental limitation has been lifted, and that of course will unlock entirely new use cases.

However, all of this is only my guess, and so I must say then that AQUA is not a defined product. I do not know what it is, of what it is for, from AWS documentation. It's capabilities, performance, behaviour, strengths and weaknesses are almost wholly unknown. As such, if you are using AQUA, unless

you have access to information I do not, you are simply taking a gamble, where the benefits, downsides and risks are all almost unknown.

Test Method

There are two separate sets of tests.

The first set consists of general experiments - queries - to get an idea of the nature of AQUA.

The second set consists of a methodical exploration of performance and behaviour with and without AQUA, using and not using LIKE, and when the tables used are in, or not in, the local SSD cache.

The second set of tests has three variables to control;

1. is AQUA enabled or disabled
2. is LIKE or SIMILAR TO being used, or not used
3. is the table in local cache, or only in RMS

Controlling AQUA is easy; it's part of cluster configuration.

Controlling LIKE / SIMILAR TO is easy, as we control the test query being issued.

Controlling whether a table is in local SSD cache or RMS is not so easy. There are no options to control the local SSD cache, so we have to operate the cluster in such a way that we can ensure a table is no longer in cache.

The `ra3.xlplus` nodes have 954,367 megabytes of local store, and I always run two node clusters (single node clusters are an aberration - do not use them, ever), so there are 1,908,734 megabytes of disk, and it could be all of it is used for cache. I need to write data to this cache until it is full, and then write additional data so that the *first* data written begins to be flushed from the cache (assuming LRU).

The basic approach is to create and populates tables until the disk size plus 25% has been written, and then use the very first tables written in the test query. Those first tables should by then no longer be in the cache.

The tables are identical, with the following DDL;

```
create table table_%d
(
  column_1  int2        not null encode raw distkey,
  column_2  char(1024)  not null encode raw
)
diststyle key;
```

The goal is to have two values in the table, which one each slice alternate value row by row, with an equal number of rows on each slice. The two values are 1024 characters of 'a' and 1024 characters of 'b'.

I could find no way to achieve this using self-join, which is the normal method I use to create test data, because although you can write a query which in theory would produce this output, SQL specifies no ordering to rows and Redshift is free to swap the left and right sides of the self-join, which it does in fact, and so rather than alternating rows, you end up with very long sets of contiguous identical values.

In the end, I had to use a separate Python script to generate the data, as four files (one per slice), and load that using COPY.

The values to use in `column_1`, to control which slice a row goes to, can be empirically determined by executing a loop which begins at 0 and increments, and writing a single row using the loop value for `column_1`, then checking `stv_blocklist` to see which slice the row ended up on. (You cannot use the `slice_num()` function for this - now days, this indicates which slice processed the row, not which slice the row is stored on; the docs have not been updated and are incorrect.)

These values are known for a two-node cluster and are constant, being 0, 1, 2, and 5.

These values are used by the `data_generation.py` script, which iterates over them, opening a file, writing 3,263,442 rows (1,631,721 each of 'a's and 'b's), and closing the file, resulting in four 3.2 GB `CSV` files and so a total of 13,053,768 rows in the table. The number of rows is that which was produced by the earlier attempt at using self-joins; the table size that earlier attempted used worked well, so I continued to use that number of rows.

I compress these files with `bzip2`, which brings them down to about 8kb each.

I tried `gzip`, which gives about 7.5mb, and `zstd`, which gives about 300kb, but both `gzip` and `zstd` fail to load with COPY, because Redshift has problems with decompressing very highly compressed files (not enough space allocated for output); however, and unexpectedly, `bzip2` worked.

I begin by creating a single table, named `table_base`, and load these files using COPY, like so;

```
copy              table_base ( column_1, column_2 )
from              's3://nnn/'
access_key_id     'nnn'
secret_access_key 'nnn'
bzip2
compupdate        off
csv
maxerror          0
statupdate        on;
```

I then iterate, creating a new test table each time, and using an `INSERT` from `SELECT` to populate the new table, like so;

```
insert into
  table_%d ( column_1, column_2 )
select
  column_1,
  column_2
from
  table_base;
```

So to run the tests in cache, we start a new cluster and create one table per test query. One table is created per test query to avoid caching effects induced by other test queries.

To run the tests out of RMS, 184 tables are created, which takes about two hours.

This all has to be done twice, once for a cluster with AQUA disabled, then again for a cluster with AQUA enabled.

The test queries are;

1. `select count(*) from table_0 where column_2 = 'aaa...';`

   This match the first value, 1024 characters of 'a' (I've not listed them all here :-), reads every block, and returns 50% of rows.

2. `select count(*) from table_1 where column_2 like '%';`

   This reads every block, and returns 100% of rows.

3. `select count(*) from table_1 where column_2 like '%a%';`

   This reads every block, and returns 50% of rows.

4. `select count(*) from table_1 where column_2 like '%b%';`

   This reads every block, and returns 50% of rows.

5. `select count(*) from table_1 where column_2 like '%c%';`

   This reads every block, and returns 0% of rows.

6. `select count(*) from table_2 where column_2 similar to '%a%';`

   This reads every block, and returns 50% of rows, but is using SIMILAR TO.

The first query, for equality, produces exactly the same output as the third query, which uses LIKE, and the sixth query, which uses SIMILAR TO. These queries are directly exactly comparable, varying only by the operator used.

The first query uses the equality operator, and so does not invoke AQUA. All the other queries, as they use LIKE and SIMILAR TO, do invoke AQUA (at least, when AQUA is enabled on the cluster).

Where local SSD caching is directly involved in the behaviour being tested, it is not meaningful to run the same query the usual five times, discarding the fastest and slowest queries, as we would *expect* behaviour to be fundamentally different between each run of a test query.

Instead, what's done is that each query is run three times, so we can see how query behaviour evolves due to caching.

After a cluster is created, `boto3` is used to obtain information about the cluster, which is dumped into the results, so we can definitively know its AQUA status.

Returning to the first set of tests, the general experiments, these load the data from S3 using COPY, as described above, and then create a single test table from that data. This single table is used for all of the first set of tests.

Finally, note;

1. Automated materialized views are disabled (*always* disable this).
2. Result cache is disabled.
3. Automatic rewriting of queries for materialized views with AQUA, is disabled.

That's it for the test method.

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 cluster was a two-node `ra3.xlplus` in `us-east-1`.

General Experiments

The cluster here has AQUA enabled. The REDSHIFT / AQUA column indicates whether the query was routed to AQUA or ran on the cluster.

```
1. NO LIKE, NO SIMILAR TO           : REDSHIFT : 6526884 rows : 6735744288 bytes
2. LIKE                             :     AQUA :      28 rows :        224 bytes
3. LIKE, COMPARE                    :     AQUA :      21 rows :        168 bytes
4. LIKE, COMPARE, MAX               :     AQUA :      21 rows :        210 bytes
5. NO LIKE, NO SIMILAR TO, GROUP BY : REDSHIFT : 6526884 rows : 6748798056 bytes
6. LIKE, GROUP BY                   :     AQUA :      28 rows :        280 bytes
7. LIKE, DISTINCT                   :     AQUA : 6526884 rows : 6683529216 bytes
```

Main Test

1. NO AQUA / AQUA indicates whether the cluster has AQUA enabled or disabled.
2. CACHE / RMS indicates whether the table being tested is already in the local SSD cache, or is only in RMS.
3. EQUALITY / LIKE % / LIKE A / LIKE B / LIKE C / SIMILAR TO indicates which comparison operator was used in the test query.
4. The test query is repeated three times, to see how behaviour changes due to caching. The `R0/R1/R2` column indicates the repetition number.
5. The results, in brackets, are the times taken in seconds by each slice to execute the ***first segment*** only, which is a `scan`, two `project`, and an `aggregate`.

   A segment is a process and is composed of steps and rows stream through the steps, so any given step may be the bottleneck, and the end time (and

so the duration) is the time when the process (which is to say, segment) finishes. There is no information available in the system tables about when a step finishes, and as a concept that doesn't quite make sense anyway, as the slowest step in a segment will slow all the other steps.

```
NO AQUA : CACHE : EQUALITY   : R0 : ( 0:00:09.207733 / 0:00:09.204580 / 0:00:09.013693 / 0:00:08.998986 )
NO AQUA : CACHE : EQUALITY   : R1 : ( 0:00:08.993426 / 0:00:08.988677 / 0:00:09.344057 / 0:00:09.348855 )
NO AQUA : CACHE : EQUALITY   : R2 : ( 0:00:09.250344 / 0:00:09.248743 / 0:00:09.550706 / 0:00:09.555538 )

NO AQUA : CACHE : LIKE %     : R0 : ( 0:00:09.269197 / 0:00:09.262832 / 0:00:09.546041 / 0:00:09.547510 )
NO AQUA : CACHE : LIKE %     : R1 : ( 0:00:09.249538 / 0:00:09.244753 / 0:00:09.559937 / 0:00:09.565394 )
NO AQUA : CACHE : LIKE %     : R2 : ( 0:00:09.249663 / 0:00:09.244878 / 0:00:09.575789 / 0:00:09.578883 )

NO AQUA : CACHE : LIKE A     : R0 : ( 0:00:09.267869 / 0:00:09.269394 / 0:00:09.453487 / 0:00:09.451897 )
NO AQUA : CACHE : LIKE A     : R1 : ( 0:00:09.250844 / 0:00:09.249226 / 0:00:09.404371 / 0:00:09.409178 )
NO AQUA : CACHE : LIKE A     : R2 : ( 0:00:09.249227 / 0:00:09.250828 / 0:00:09.335284 / 0:00:09.349675 )

NO AQUA : CACHE : LIKE B     : R0 : ( 0:00:09.265181 / 0:00:09.271362 / 0:00:09.455281 / 0:00:09.448595 )
NO AQUA : CACHE : LIKE B     : R1 : ( 0:00:09.310843 / 0:00:09.316331 / 0:00:09.608579 / 0:00:09.597943 )
NO AQUA : CACHE : LIKE B     : R2 : ( 0:00:09.250475 / 0:00:09.245907 / 0:00:09.600638 / 0:00:09.597784 )

NO AQUA : CACHE : LIKE C     : R0 : ( 0:00:09.262156 / 0:00:09.268511 / 0:00:09.579287 / 0:00:09.589439 )
NO AQUA : CACHE : LIKE C     : R1 : ( 0:00:09.251058 / 0:00:09.249509 / 0:00:09.408094 / 0:00:09.409781 )
NO AQUA : CACHE : LIKE C     : R2 : ( 0:00:09.251943 / 0:00:09.247216 / 0:00:09.381876 / 0:00:09.385503 )

NO AQUA : CACHE : SIMILAR TO : R0 : ( 0:00:09.298151 / 0:00:09.192925 / 0:00:09.338577 / 0:00:09.522980 )
NO AQUA : CACHE : SIMILAR TO : R1 : ( 0:00:09.242530 / 0:00:09.259425 / 0:00:09.474999 / 0:00:09.333545 )
NO AQUA : CACHE : SIMILAR TO : R2 : ( 0:00:09.270805 / 0:00:09.216178 / 0:00:09.489395 / 0:00:09.316543 )

NO AQUA : RMS   : EQUALITY   : R0 : ( 0:00:49.576080 / 0:00:48.813398 / 0:00:53.204578 / 0:00:53.571728 )
NO AQUA : RMS   : EQUALITY   : R1 : ( 0:00:09.224023 / 0:00:09.420448 / 0:00:09.201846 / 0:00:08.697027 )
NO AQUA : RMS   : EQUALITY   : R2 : ( 0:00:09.261867 / 0:00:09.265024 / 0:00:09.348863 / 0:00:09.345673 )

NO AQUA : RMS   : LIKE %     : R0 : ( 0:00:51.763577 / 0:00:51.676877 / 0:01:01.149151 / 0:01:01.664853 )
NO AQUA : RMS   : LIKE %     : R1 : ( 0:00:08.608361 / 0:00:08.603606 / 0:00:08.657271 / 0:00:08.652435 )
NO AQUA : RMS   : LIKE %     : R2 : ( 0:00:09.249899 / 0:00:09.244920 / 0:00:09.349570 / 0:00:09.343210 )

NO AQUA : RMS   : LIKE A     : R0 : ( 0:01:08.177246 / 0:01:08.683746 / 0:00:48.815787 / 0:00:50.480384 )
NO AQUA : RMS   : LIKE A     : R1 : ( 0:00:08.525502 / 0:00:08.530090 / 0:00:08.568479 / 0:00:08.573761 )
NO AQUA : RMS   : LIKE A     : R2 : ( 0:00:09.247934 / 0:00:09.250880 / 0:00:09.349531 / 0:00:09.347938 )

NO AQUA : RMS   : LIKE B     : R0 : ( 0:00:51.321767 / 0:00:53.027248 / 0:00:52.187921 / 0:00:52.088932 )
NO AQUA : RMS   : LIKE B     : R1 : ( 0:00:08.710603 / 0:00:08.705800 / 0:00:09.350113 / 0:00:09.346366 )
NO AQUA : RMS   : LIKE B     : R2 : ( 0:00:08.797242 / 0:00:08.802712 / 0:00:09.354782 / 0:00:09.349460 )

NO AQUA : RMS   : LIKE C     : R0 : ( 0:00:50.910098 / 0:00:50.761279 / 0:00:49.429202 / 0:00:49.508892 )
NO AQUA : RMS   : LIKE C     : R1 : ( 0:00:08.354100 / 0:00:08.339638 / 0:00:08.398365 / 0:00:08.226090 )
NO AQUA : RMS   : LIKE C     : R2 : ( 0:00:09.250590 / 0:00:09.252165 / 0:00:09.351336 / 0:00:09.346647 )

NO AQUA : RMS   : SIMILAR TO : R0 : ( 0:00:55.580887 / 0:00:56.749457 / 0:00:52.061169 / 0:00:53.048903 )
NO AQUA : RMS   : SIMILAR TO : R1 : ( 0:00:09.057150 / 0:00:09.046908 / 0:00:09.121725 / 0:00:09.127153 )
NO AQUA : RMS   : SIMILAR TO : R2 : ( 0:00:09.265822 / 0:00:09.259137 / 0:00:09.366165 / 0:00:09.371360 )

   AQUA : CACHE : EQUALITY   : R0 : ( 0:00:09.204733 / 0:00:09.176037 / 0:00:08.823521 / 0:00:08.822097 )
   AQUA : CACHE : EQUALITY   : R1 : ( 0:00:09.248911 / 0:00:09.250495 / 0:00:08.578060 / 0:00:08.576426 )
   AQUA : CACHE : EQUALITY   : R2 : ( 0:00:09.247384 / 0:00:09.250562 / 0:00:09.352312 / 0:00:09.350722 )

   AQUA : CACHE : LIKE %     : R0 : ( 0:00:43.327411 / 0:00:43.899995 / 0:00:41.643454 / 0:00:42.943825 )
   AQUA : CACHE : LIKE %     : R1 : ( 0:00:01.744283 / 0:00:01.791337 / 0:00:03.399622 / 0:00:01.745078 )
   AQUA : CACHE : LIKE %     : R2 : ( 0:00:01.731125 / 0:00:04.634349 / 0:00:01.807203 / 0:00:01.766072 )

   AQUA : CACHE : LIKE A     : R0 : ( 0:00:41.195449 / 0:00:41.796387 / 0:00:43.170872 / 0:00:41.442968 )
   AQUA : CACHE : LIKE A     : R1 : ( 0:00:01.559614 / 0:00:02.288364 / 0:00:01.580123 / 0:00:01.581609 )
   AQUA : CACHE : LIKE A     : R2 : ( 0:00:01.664650 / 0:00:01.589867 / 0:00:01.605557 / 0:00:01.574602 )

   AQUA : CACHE : LIKE B     : R0 : ( 0:00:43.572248 / 0:00:42.263507 / 0:00:42.688915 / 0:00:46.140976 )
   AQUA : CACHE : LIKE B     : R1 : ( 0:00:01.851338 / 0:00:01.867054 / 0:00:01.858184 / 0:00:01.770956 )
   AQUA : CACHE : LIKE B     : R2 : ( 0:00:01.623430 / 0:00:01.566698 / 0:00:01.577199 / 0:00:01.511778 )

   AQUA : CACHE : LIKE C     : R0 : ( 0:00:42.210546 / 0:00:43.961976 / 0:00:42.622863 / 0:00:46.603875 )
   AQUA : CACHE : LIKE C     : R1 : ( 0:00:00.901284 / 0:00:00.917168 / 0:00:00.850170 / 0:00:00.905184 )
   AQUA : CACHE : LIKE C     : R2 : ( 0:00:01.002802 / 0:00:00.972192 / 0:00:00.913016 / 0:00:01.145805 )
```

```
AQUA : CACHE : SIMILAR TO : R0 : ( 0:00:46.094824 / 0:00:42.369193 / 0:00:48.642382 / 0:00:41.628065 )
AQUA : CACHE : SIMILAR TO : R1 : ( 0:00:01.521027 / 0:00:01.583562 / 0:00:01.595717 / 0:00:01.567324 )
AQUA : CACHE : SIMILAR TO : R2 : ( 0:00:01.490310 / 0:00:01.542446 / 0:00:01.602099 / 0:00:01.591313 )

AQUA : RMS   : EQUALITY   : R0 : ( 0:00:50.714166 / 0:00:49.495406 / 0:00:49.868700 / 0:00:49.279289 )
AQUA : RMS   : EQUALITY   : R1 : ( 0:00:08.931096 / 0:00:08.995204 / 0:00:08.328496 / 0:00:08.772002 )
AQUA : RMS   : EQUALITY   : R2 : ( 0:00:09.272983 / 0:00:09.278882 / 0:00:09.349943 / 0:00:09.346755 )

AQUA : RMS   : LIKE %     : R0 : ( 0:00:45.996555 / 0:00:48.870548 / 0:00:45.798664 / 0:00:46.332533 )
AQUA : RMS   : LIKE %     : R1 : ( 0:00:01.889942 / 0:00:01.829568 / 0:00:01.786090 / 0:00:03.406696 )
AQUA : RMS   : LIKE %     : R2 : ( 0:00:01.773647 / 0:00:01.848038 / 0:00:01.817244 / 0:00:01.824604 )

AQUA : RMS   : LIKE A     : R0 : ( 0:00:46.308171 / 0:01:44.695472 / 0:00:46.558651 / 0:00:48.327062 )
AQUA : RMS   : LIKE A     : R1 : ( 0:00:01.611775 / 0:00:01.605035 / 0:00:01.549466 / 0:00:01.516289 )
AQUA : RMS   : LIKE A     : R2 : ( 0:00:01.580145 / 0:00:01.640644 / 0:00:01.537107 / 0:00:01.512776 )

AQUA : RMS   : LIKE B     : R0 : ( 0:00:46.227704 / 0:00:46.398593 / 0:00:48.542814 / 0:00:49.854854 )
AQUA : RMS   : LIKE B     : R1 : ( 0:00:01.596248 / 0:00:01.557935 / 0:00:01.602313 / 0:00:01.626484 )
AQUA : RMS   : LIKE B     : R2 : ( 0:00:01.594181 / 0:00:01.576170 / 0:00:02.305109 / 0:00:01.566581 )

AQUA : RMS   : LIKE C     : R0 : ( 0:00:51.031678 / 0:00:47.104351 / 0:00:47.005895 / 0:00:52.106889 )
AQUA : RMS   : LIKE C     : R1 : ( 0:00:00.887638 / 0:00:01.020990 / 0:00:00.814183 / 0:00:00.874858 )
AQUA : RMS   : LIKE C     : R2 : ( 0:00:00.923204 / 0:00:00.924688 / 0:00:00.892344 / 0:00:00.909735 )

AQUA : RMS   : SIMILAR TO : R0 : ( 0:00:49.881206 / 0:01:32.406387 / 0:00:46.325607 / 0:00:46.805514 )
AQUA : RMS   : SIMILAR TO : R1 : ( 0:00:01.543379 / 0:00:01.624610 / 0:00:01.553433 / 0:00:01.534623 )
AQUA : RMS   : SIMILAR TO : R2 : ( 0:00:01.608424 / 0:00:01.597847 / 0:00:01.615707 / 0:00:01.618182 )
```

Discussion

General Experiments

```
1. NO LIKE, NO SIMILAR TO           : REDSHIFT : 6526884 rows : 6735744288 bytes
2. LIKE                             :     AQUA :      28 rows :        224 bytes
3. LIKE, COMPARE                    :     AQUA :      21 rows :        168 bytes
4. LIKE, COMPARE, MAX               :     AQUA :      21 rows :        210 bytes
5. NO LIKE, NO SIMILAR TO, GROUP BY : REDSHIFT : 6526884 rows : 6748798056 bytes
6. LIKE, GROUP BY                   :     AQUA :      28 rows :        280 bytes
7. LIKE, DISTINCT                   :     AQUA : 6526884 rows : 6683529216 bytes
```

1. NO LIKE, NO SIMILAR TO

   ```
   select count(*) from table_0 where column_2 = 'aaa...';
   ```

   Query does not use LIKE, or SIMILAR TO, and so is expected not to use AQUA, and this is what is found. The string being searched for is 1024 'a' characters, which is one of the two values in the table, and is 50% of the rows. The `scan` step as we see here returns in half the rows in the table (the total row count in the table is 13,053,768).

2. LIKE

   ```
   select count(*) from table_0 where column_2 like '%a%';
   ```

   Now we replace the equality operator with LIKE, and by this we expect AQUA to be invoked, and it is; the `scan` type became 35, the AQUA scan. We can see that the amount of data returned from AQUA is very small; it is returning the *results* of the first segment to the Redshift cluster.

3. LIKE, COMPARE

   ```
   select count(*) from table_0 where column_2 like '%a%' and
   column_1 >= 1;
   ```

   We repeat the query, but adding in a simple extra comparison clause on another column. We can see from very small number of rows returned from the scan step that AQUA is still invoked, and is handling the extra clause.

4. LIKE, COMPARE, MAX

   ```
   select count(*), max(column_1) from table_0 where column_2
   like '%a%' and column_1 >= 1;
   ```

Here, a `max()` is added, and once again, AQUA is handling it.

5. NO LIKE, NO SIMILAR TO, GROUP BY

   ```
   select count(*) from table_0 where column_2 = '%s' group by
   column_1;
   ```

   Here we remove the LIKE, and add a `GROUP BY`. No LIKE, no AQUA.

6. LIKE, GROUP BY

   ```
   select count(*) from table_0 where column_2 like '%a%' group
   by column_1;
   ```

   Now we replace the equality operator with a LIKE, AQUA is invoked, and is handling the `GROUP BY`.

7. LIKE, DISTINCT

   ```
   select distinct column_2 from table_0 where column_2 like
   '%a%';
   ```

   Finally, this query introduces `distinct`. Where a LIKE is issued, we get AQUA, but now we see AQUA returning all the matching rows (half the table), because AQUA nodes run in parallel and have no organizing master node, so they cannot handle a `distinct`; they must return the data to Redshift.

So, what do we make of this?

For the very limited set of queries issued, the criteria for invoking AQUA seems straightforward; if the query uses a LIKE or a SIMILAR TO, the given `scan` step goes to AQUA. Given the obfuscations and marketing AWS come out with, all the stuff about "Redshift intelligently choosing when to use AQUA" makes me very strongly suspect that LIKE and SIMILAR TO simply always invoke AQUA.

We see AQUA is handling comparisons and `group by` and `max()`. SQL has an enormous range of functionality so I'm not in a position to quickly write a generalized SQL test to see what AQUA can or cannot do.

However, we see `distinct` is not handled, and causes AQUA to return all matching rows to the Redshift cluster.

To my eye then AQUA - in line with such descriptions as AWS have emitted - is a multi-node distributed set of independent processors, each of which processes a part of a given set of data, and there is no mechanism for those processors to communicate between themselves.

In other words, the same design as Redshift Spectrum; the idea is you carefully craft your queries so that the multi-node distributed system can perform scanning and aggregation work and return tiny results to the cluster.

The problem is if you issue the wrong query, the multi-node distributed system *cannot* handle it, and must, perforce, return some, most or even all of the table back to the Redshift cluster.

AWS do not document the capabilities of these systems, AQUA or Spectrum, so you cannot know in advance what can or cannot be done; you can only find by trying it, which is a ridiculous; every developer in the world has better things to do that reproduce the same work trying to guess what these systems can do.

In fact, where AWS have obscured the nature of both systems from users (last I heard the line being pedalled by TAMs is "Spectrum is the same as Redshift but cheaper"), what normally happens is users have no clue at all about these issues and no clue why Redshift is performing badly (and these days, that means they now eventually move to Snowflake).

With Spectrum, where Spectrum is intended for data volumes larger than a cluster can handle, getting the query wrong can perfectly well means the cluster grinds to a halt, until it eventually completely fills local disk and then kills the query.

For AQUA, I would say where the tables being used are expected to be normal Redshift tables (even if RMS means those tables can now be very large), probably the consequences are going to be more manageable.

Moreover, LIKE and SIMILAR TO are in my experience very rare and almost unheard of, respectively. In 25 years of writing SQL, I have *never* used SIMILAR TO. This should, overall, act to greatly minimize any adverse consequences, as I suspect AQUA in the very large majority of cases simply is not in use.

Main Test

So, I will progress down the results, comparing NO AQUA with AQUA.

To begin with, we have CACHE and EQUALITY.

```
NO AQUA : CACHE : EQUALITY   : R0 : ( 0:00:09.207733 / 0:00:09.204580 / 0:00:09.013693 / 0:00:08.998986 )
NO AQUA : CACHE : EQUALITY   : R1 : ( 0:00:08.993426 / 0:00:08.988677 / 0:00:09.344057 / 0:00:09.348855 )
NO AQUA : CACHE : EQUALITY   : R2 : ( 0:00:09.250344 / 0:00:09.248743 / 0:00:09.550706 / 0:00:09.555538 )

   AQUA : CACHE : EQUALITY   : R0 : ( 0:00:09.204733 / 0:00:09.176037 / 0:00:08.823521 / 0:00:08.822097 )
   AQUA : CACHE : EQUALITY   : R1 : ( 0:00:09.248911 / 0:00:09.250495 / 0:00:08.578060 / 0:00:08.576426 )
   AQUA : CACHE : EQUALITY   : R2 : ( 0:00:09.247384 / 0:00:09.250562 / 0:00:09.352312 / 0:00:09.350722 )
```

As we are using EQUALITY, AQUA should make no difference, and we see it does not.

The table used in the query is 12,988 blocks, and processing that in 9 seconds means processing about 1,440 megabytes per second, which would mean about 720 megabytes per second per node.

Next, we have CACHE and LIKE % (the results for all the LIKE queries identical, so I do not discuss them separately).

```
NO AQUA : CACHE : LIKE %     : R0 : ( 0:00:09.269197 / 0:00:09.262832 / 0:00:09.546041 / 0:00:09.547510 )
NO AQUA : CACHE : LIKE %     : R1 : ( 0:00:09.249538 / 0:00:09.244753 / 0:00:09.559937 / 0:00:09.565394 )
NO AQUA : CACHE : LIKE %     : R2 : ( 0:00:09.249663 / 0:00:09.244878 / 0:00:09.575789 / 0:00:09.578883 )

   AQUA : CACHE : LIKE %     : R0 : ( 0:00:43.327411 / 0:00:43.899995 / 0:00:41.643454 / 0:00:42.943825 )
   AQUA : CACHE : LIKE %     : R1 : ( 0:00:01.744283 / 0:00:01.791337 / 0:00:03.399622 / 0:00:01.745078 )
   AQUA : CACHE : LIKE %     : R2 : ( 0:00:01.731125 / 0:00:04.634349 / 0:00:01.807203 / 0:00:01.766072 )
```

Where LIKE is being used, AQUA should be invoked, and we will see if makes a difference, and *pow* it really does - the first segment of the first AQUA query

took a jaw-dropping 43 seconds.

This behaviour is *not* expected, is critically significant, and AWS have made no mention of it at all.

Mmm. I have to say, in my experience, Redshift features are like this. AWS, at least with regard to Redshift, which is the only area I know about, emits only marketing; relentlessly positive, obfuscates all weaknesses, and says nothing about implementation or usage. Only strengths are discussed. You simply cannot trust AWS, at least for Redshift; information is *always* hidden, and critical information at that. Keep that in mind, always, when receiving any information from AWS or its staff.

Moving on, to R1 and R2, we then see something else of note - once that slow first query has completed, AQUA then has the first segment running in about 1.7 seconds. NO AQUA is taking about 9.4 seconds, across the board, no variations.

We see then AQUA *is* faster, and by quite a bit - about 6x - but there's a huge great big stonking latency on the first query. I did some impromptu testing, and I found once the initial slow query has passed on a given table, all the LIKE queries were accelerated.

If we look in the raw results at the disk in use before and after the slow first AQUA query, we see disk use *has not changed*, remaining at 91,016 blocks.

It then is *not* the case that the table is being downloaded to the local SSD cache; for we do see in the RMS tests, when a table is not in cache and it is accessed, the disk used has increased by the expected amount, so the measure of disk use is working.

AWS *do* (for whatever it's worth) describe AQUA as a "caching layer". To my eye that's not quite right (because AQUA *cannot* fulfil a query until it loads the table, where a cache normally is a layer over a separate, independent system), but it's close enough not to worry about; when you query AQUA, if it doesn't have your table, it has to fetch it, and then it can fulfil your query. So it's slow and then fast, which is what you expect from a cache.

How and where this cached copy is being stored, I have no idea, nor how large the cache is, or how quickly tables are retired.

Finally, note that the NO AQUA EQUALITY and all the LIKE queries run at the same speed. This is unexpected - LIKE should be more expensive - but it may be the data and LIKE pattern I've chosen are such that the work is identical, where LIKE can in fact be made much more expensive with more complex patterns.

Next, CACHE and SIMILAR TO.

```
NO AQUA : CACHE : SIMILAR TO : R0 : ( 0:00:09.298151 / 0:00:09.192925 / 0:00:09.338577 / 0:00:09.522980 )
NO AQUA : CACHE : SIMILAR TO : R1 : ( 0:00:09.242530 / 0:00:09.259425 / 0:00:09.474999 / 0:00:09.333545 )
NO AQUA : CACHE : SIMILAR TO : R2 : ( 0:00:09.270805 / 0:00:09.216178 / 0:00:09.489395 / 0:00:09.316543 )

   AQUA : CACHE : SIMILAR TO : R0 : ( 0:00:46.094824 / 0:00:42.369193 / 0:00:48.642382 / 0:00:41.628065 )
   AQUA : CACHE : SIMILAR TO : R1 : ( 0:00:01.521027 / 0:00:01.583562 / 0:00:01.595717 / 0:00:01.567324 )
   AQUA : CACHE : SIMILAR TO : R2 : ( 0:00:01.490310 / 0:00:01.542446 / 0:00:01.602099 / 0:00:01.591313 )
```

The results are identical to LIKE, as expected. I included this here just to show that SIMILAR TO is indeed also being handled by AQUA.

Next, we're on to the RMS tests - where the table is *not* in the cache. This is really interesting stuff, because we will finally have some idea of how much latency is introduced by using RMS rather than local SSD cache.

We begin with RMS and EQUALITY, and here I also include the result (given above already) for CACHE and EQUALITY, so we can compare directly to the latency for RMS.

```
NO AQUA : CACHE : EQUALITY   : R0 : ( 0:00:09.207733 / 0:00:09.204580 / 0:00:09.013693 / 0:00:08.998986 )
NO AQUA : CACHE : EQUALITY   : R1 : ( 0:00:08.993426 / 0:00:08.988677 / 0:00:09.344057 / 0:00:09.348855 )
NO AQUA : CACHE : EQUALITY   : R2 : ( 0:00:09.250344 / 0:00:09.248743 / 0:00:09.550706 / 0:00:09.555538 )

NO AQUA : RMS   : EQUALITY   : R0 : ( 0:00:49.576080 / 0:00:48.813398 / 0:00:53.204578 / 0:00:53.571728 )
NO AQUA : RMS   : EQUALITY   : R1 : ( 0:00:09.224023 / 0:00:09.420448 / 0:00:09.201846 / 0:00:08.697027 )
NO AQUA : RMS   : EQUALITY   : R2 : ( 0:00:09.261867 / 0:00:09.265024 / 0:00:09.348863 / 0:00:09.345673 )

   AQUA : CACHE : EQUALITY   : R0 : ( 0:00:09.204733 / 0:00:09.176037 / 0:00:08.823521 / 0:00:08.822097 )
   AQUA : CACHE : EQUALITY   : R1 : ( 0:00:09.248911 / 0:00:09.250495 / 0:00:08.578060 / 0:00:08.576426 )
   AQUA : CACHE : EQUALITY   : R2 : ( 0:00:09.247384 / 0:00:09.250562 / 0:00:09.352312 / 0:00:09.350722 )

   AQUA : RMS   : EQUALITY   : R0 : ( 0:00:50.714166 / 0:00:49.495406 / 0:00:49.868700 / 0:00:49.279289 )
   AQUA : RMS   : EQUALITY   : R1 : ( 0:00:08.931096 / 0:00:08.995204 / 0:00:08.328496 / 0:00:08.772002 )
   AQUA : RMS   : EQUALITY   : R2 : ( 0:00:09.272983 / 0:00:09.278882 / 0:00:09.349943 / 0:00:09.346755 )
```

Where the test table is not in cache, it is expected it must be brought down from RMS, and we see in both R0 cases the first segment is taking about 50 seconds.

Looking at the disk use before and after the first RMS query, we see disk use increases from 1,427,117 to 1,439,879 blocks, and 1,427,117 to 1,439,794 blocks, respectively. This makes the increase 12,762 and 12,677 blocks, respectively.

The test tables look like this;

```
 schema_name | table_name | column_name | data_type  | encoding | info | sblks | ublks | tblks
-------------+------------+-------------+------------+----------+------+-------+-------+-------
 public      | table_0    | column_1    | int2       | raw      |  .dn |     0 |    28 |    28
 public      | table_0    | column_2    | char(1024) | raw      |  . n |     0 | 12852 | 12852
 public      | table_0    | deletexid   | int8       | raw      |  . n |     0 |     4 |     4
 public      | table_0    | insertxid   | int8       | raw      |  . n |     0 |     4 |     4
 public      | table_0    | oid         | oid        | raw      |  . n |     0 |   100 |   100
(5 rows)
```

In both cases, this is a little bit less than the 12,852 blocks of `column_2`, for no obvious reason, but we can see what's going on.

Normally, that first segment takes about 9 seconds, so we're looking at 41 seconds of RMS latency on a 12,988 block table.

We finally have a number for the overhead of RMS!

That's about 320 megabytes per second, which is 160 megabytes per second per node.

That compares to the 720 megabytes per node for exact same operation using the local SSD cache.

That makes local SSD cache 4.5x faster than RMS.

However, this is all for a two-node cluster running a single query.

With a larger cluster, more nodes should mean more bandwidth, and larger node types probably have more network I/O as well, but it is reasonable to think this bandwidth is contended, so if we had ten concurrent queries all pulling from

RMS, we would with our current cluster see about 32 megabytes per seconds, and the delay for our test query would become a slightly eye-watering 410 seconds.

That latency is absolutely fine if you're using Redshift as its design choices mandate - for Big Data queries, where the data is vast, and so even with the staggering efficiency provided by correctly operated sorting, queries still takes some time - but is death on toast if Redshift is, as it always is, being used incorrectly, as if it were an unsorted, row-store database, maybe with BI tools sitting on the top too, like a miniature Hobbit atop the Cake of Doom.

One other interesting detail about RMS is the total amount of disk used while the 184 test tables are being made. This is not part of the focus of this white paper, so I've not included it in the results proper, but examine this logging which was produced as the tables were made, which shows the time taken in seconds and the total disk in use after the table is created, in blocks;

```
table_0... 0:00:53.084546 / 26061
table_1... 0:00:29.236125 / 39042
table_2... 0:00:28.739220 / 52034
[snip]
table_69... 0:00:31.040292 / 922507
table_70... 0:00:29.178472 / 935502
table_71... 0:00:29.279066 / 948496
[snip]
table_144... 0:00:31.125531 / 1505138
table_145... 0:00:29.740355 / 1518143
table_146... 0:00:29.864787 / 1336161
```

The amount of disk used increases by the size of the new tables, until we get to a bit more than 1,500,000 blocks, when Redshift then dumps about 200,000 blocks to RMS, freeing up space. This pattern repeats.

The total disk available in the two-node test cluster, according to `stv_partitions`, is 1,908,734 blocks.

Either about half a terabyte is being kept available, or the amount of disk being used for cache is significantly less than the total disk available.

Next, RMS and LIKE % (the results for LIKE %, LIKE A and LIKE B are identical, so I do not discuss them separately).

```
NO AQUA : RMS   : LIKE %     : R0 : ( 0:00:51.763577 / 0:00:51.676877 / 0:01:01.149151 / 0:01:01.664853 )
NO AQUA : RMS   : LIKE %     : R1 : ( 0:00:08.608361 / 0:00:08.603606 / 0:00:08.657271 / 0:00:08.652435 )
NO AQUA : RMS   : LIKE %     : R2 : ( 0:00:09.249899 / 0:00:09.244920 / 0:00:09.349570 / 0:00:09.343210 )

   AQUA : RMS   : LIKE %     : R0 : ( 0:00:45.996555 / 0:00:48.870548 / 0:00:45.798664 / 0:00:46.332533 )
   AQUA : RMS   : LIKE %     : R1 : ( 0:00:01.889942 / 0:00:01.829568 / 0:00:01.786090 / 0:00:03.406696 )
   AQUA : RMS   : LIKE %     : R2 : ( 0:00:01.773647 / 0:00:01.848038 / 0:00:01.817244 / 0:00:01.824604 )
```

Interesting stuff.

With NO AQUA, we see with the first query, with the long delay, disk use went from 1,440,006 blocks to 1,452,862 blocks, an increase of 12,856.

With AQUA, we see with the first query, with the long delay, disk use went from 1,439,981 to 1,439,981 - which is to say, *did not change*

I note the following;

1. The difference in NO AQUA and AQUA performance for the second and third queries is about 7.7 seconds.

2. Looking at LIKE %, LIKE A and LIKE B, that the difference in the time taken for the slow first query is usually about 6 seconds (barring special cases, as we see here with the over one minute time of slices 2 and 3, which will be discussed shortly).
3. In the NO AQUA `scan` step, a total of 13,053,768 rows were emitted, amounting to 13,471,488,576 bytes.
4. In the AQUA `scan` step, a total of 7 rows were emitted, amounting to 56 bytes.

To my eye, RMS is being read in both first queries, and the acceleration provided by AQUA for processing the data is accounting for most of the difference in the duration of the slow first query.

Finally, RMS and LIKE C.

```
NO AQUA : RMS   : LIKE C     : R0 : ( 0:00:50.910098 / 0:00:50.761279 / 0:00:49.429202 / 0:00:49.508892 )
NO AQUA : RMS   : LIKE C     : R1 : ( 0:00:08.354100 / 0:00:08.339638 / 0:00:08.398365 / 0:00:08.226090 )
NO AQUA : RMS   : LIKE C     : R2 : ( 0:00:09.250590 / 0:00:09.252165 / 0:00:09.351336 / 0:00:09.346647 )

   AQUA : RMS   : LIKE C     : R0 : ( 0:00:51.031678 / 0:00:47.104351 / 0:00:47.005895 / 0:00:52.106889 )
   AQUA : RMS   : LIKE C     : R1 : ( 0:00:00.887638 / 0:00:01.020990 / 0:00:00.814183 / 0:00:00.874858 )
   AQUA : RMS   : LIKE C     : R2 : ( 0:00:00.923204 / 0:00:00.924688 / 0:00:00.892344 / 0:00:00.909735 )
```

Curious. LIKE C matches no records. NO AQUA scans the table as usual, but AQUA is now only taking about 0.9 seconds, down from about 1.7 seconds. This would likely reveal something about how AQUA works internally, but I have no idea what :-)

Performance Outliers

In the set of queries published here, 72 queries in total, we have one outlier;

```
   AQUA : RMS   : SIMILAR TO : R0 : ( 0:00:49.881206 / 0:01:32.406387 / 0:00:46.325607 / 0:00:46.805514 )
```

In the test run prior to this, there were two;

```
   AQUA : CACHE : LIKE C     : R0 : ( 0:00:44.990372 / 0:05:21.654734 / 0:00:44.782741 / 0:01:33.518136 )
   AQUA : RMS   : SIMILAR TO : R0 : ( 0:00:45.453043 / 0:02:32.272677 / 0:00:46.179583 / 0:00:47.968454 )
```

That's a *five minute* scan time on slice 1, which ought to have been about 45 seconds, and remember, this is a completely idle cluster.

Now, in the prior test run the test data was not as I had intended - alternating rows of values - but rather was basically a few long contiguous sequences of each value, so of the 12,988 blocks, very nearly almost all were one value only, and there were a large number of contiguous blocks with the same value. The total number of rows of each value remained equal.

This to my eye should not in any way lead to outlier performances.

There is not enough information here to say much about this, whether the problem is in AQUA, RMS, or the cluster, but considering a typical busy cluster runs hundreds of thousands of queries per day, three seriously slow queries out of two lots of 76 queries is a significant issue, and raises questions about implementation and viability in production systems.

Step Plans

I have coined the term "step plan" for the listing of all steps taken by a query. It is a complete and canonical listing all of work done by a query, and can only be produced once a query has completed (as this information is present in the system tables only after the query has completed). It's a bit like EXPLAIN, only accurate and not misleading.

See Appendix B provides a step-plan for NO AQUA and for AQUA for the main tests. All of the test queries produce the same step plan.

The main difference between AQUA and NO AQUA is seen in the first `scan` step, which is step 0 in segment 0.

First, we see in the `Notes` column for AQUA, a scan using the AQUA system, but for NO AQUA we have a normal scan.

Second, we see the number of rows and bytes returned by the `scan` is completely different. The normal scan is returning all rows, the AQUA scan is returning only seven rows to each slice, where each row can reasonably be assumed to be eight bytes, an `int8`, and representing the output of the `count(*)` on an AQUA processor. That there are seven rows might indicate seven AQUA processors were used.

Now, what's interesting about this is that the rest of the step plan is unchanged.

Even though AQUA performed the aggregation, we still see the `aggregate` step in segment 0. In the NO AQUA scan, this step is performing the `count(*)` operation.

To my eye, this is a blunder. It is not possible to know what AQUA is doing inside its `scan` step; it is a black box. My ability to obtain information about what's going on with queries and in the cluster has been degraded. Redshift has profound issues with its system tables, and decisions like this are what have led to the current state of affairs.

(I must note also that with Redshift Serverless, the system tables have been eviscerated and barely exist. It looks like AWS would like us to put our hands over our eyes and trust them, and their marketing, completely. I utterly discourage use of Serverless. No database which almost fully obscures its internal working from its users can be trusted; and to my eye, this hiding of information from users, and the hyperbole with which AWS market Redshift, are one and the same.)

Thoughts

AQUA has a high initial overhead, loading a table from RMS, and then offers useful reductions in scan times, so a certain number of benefiting queries must be issued for the sum of the improvement to outweigh the initial overhead.

As such, AQUA is a not a magic bullet : it does not unconditionally solve scalability for searching strings with wildcards. It offers instead a trade-off, where there is a high initial cost in return for improved performance.

For example, with one of the test tables, it takes AQUA 41 seconds to make its copy. Having done so, AQUA is then about 7.3 seconds faster than the Redshift cluster (assuming the table is in the local SSD cache). About six or so queries must be issued to break even.

If only a single or a few such queries are run before the table expires from the AQUA cache, the system will be slower with AQUA, not faster.

I am then of the view that the decision by the devs to enable AQUA by default is a blunder, as AQUA can perfectly well *reduce* performance. Light LIKE users will be actually harmed. Heavy LIKE users are going to benefit. Use must be decided on a case-by-case basis, by well-informed end-users, not globally by AWS.

To work around it, as AWS in their infinite wisdom have removed the option from the Redshift Console, you must no longer start clusters from the Console, but use any other interface, such as the `AWS CLI` or `boto3` via Python and so on.

Finally, I note on-line two AQUA performance studies, both by AWS engineers. The first is the tech blog post made on the day AQUA went GA, the second is on a different site, getofo.net. It looks to me like both studies have pre-cached their tables before running their tests and have *not* disclosed this information the reader and, I think, given these are AWS staff engineers, they knew and therefore deliberately withheld this information.

RMS

Finally, a serendipitous discovery about RMS.

As noted, in the penultimate test run the data test was not alternating rows, but almost entirely blocks with only a single value.

The first query for NO AQUA, RMS and EQUALITY produced the following result;

```
NO AQUA : RMS   : EQUALITY   : R0 : ( 0:00:22.401345 / 0:00:22.573264 / 0:00:25.289982 / 0:00:24.755262 )
```

Note the 23.5 second duration, rather than the expected 50 seconds.

Examining the change in disk use from this query, the cluster went from 1,414,019 blocks to 1,420,475 blocks - a change of 6,456 blocks! in other words, the cluster had downloaded from RMS *half* the table only. This explains the reduced first segment time.

Redshift maintains in memory the Zone Map, and it seems evident that this is scanned, and then the blocks and only the blocks which could contain the rows of interest are brought down from RMS.

This means RMS is a block-based store, not a table-based stored, and the local SSD cache contains only the blocks which are actually being used.

In the AWS documentation and videos and so on, `S3` and RMS are at times used synonymously. I'm not sure if this is really true; however, if RMS (whatever it is) behaves like `S3`, then a file can be read from any point, and for any number

of bytes - so reading blocks is fine - but a file can only be written *in whole*. You cannot modify a file, only replace it.

It is unlikely RMS is writing a file per block; that would be a staggeringly vast number of files. Probably a file is a certain size, such as 128mb, or a gigabyte.

A critical unanswered question is how large the files are, as this is central to the performance of RMS, as it dictates how much data must be collected and flushed when space must be reclaimed from the local SSD cache.

We saw in the results when the cache reached about 1.5m blocks, it dropped by about 200,000 blocks. This is the only data point. It might be that 200mb is thee size of the files used by RMS, but it could perfectly well be otherwise, where RMS is in this case emitting many smaller files, having decided it wants to reclaim 200mb of space.

Conclusions

AQUA appears to be a multi-node distributed set of independent processors, each of which processes a part of the data in a table, but where there is no mechanism for those processors to communicate between themselves; in other words, the same design as Redshift Spectrum.

It is then that AQUA can implement that functionality an independent processor can perform; which is to say, the processor scans its own portion of the data, performing any work which can be performed by examining a single row at a time in a single table, and by maintaining aggregate information from those rows; so for example, GROUP BY or say MAX() works and works well, returning aggregated and therefore typically very small output, but DISTINCT cannot work well, and returns all matching rows to the cluster.

Of that which can be implemented, what has been implemented is unknown. AWS do not document this information, for AQUA or indeed for Spectrum. This makes writing queries correctly for AQUA, as with Spectrum, on the face of it, problematic.

AQUA has no access to, and does not use, the local SSD cache in the Redshift cluster; nor is it embedded in RMS and processing blocks as they are read.

Rather, when AQUA handles a query, AQUA must download the table being scanned from RMS to itself (presumably to its own SSDs). This involves a considerable delay. For the test tables, which are 12,988 blocks, with a two-node completely idle `ra3.xlplus` cluster, this takes about 41 seconds.

Although I have not specifically tested it, I think it must then be that this read by AQUA from RMS does *not* load the table into the cluster local SSD cache. If this must then occur (perhaps one query with LIKE is issued, and then one without), the load from RMS must then occur again, only this time to the local SSD cache in the cluster.

The Redshift cluster, without AQUA, takes about 9.3 seconds to run a LIKE query on a test table in the local SSD cache.

Once AQUA has downloaded the table from RMS, it takes about 1.7 seconds to run the same query.

Once AQUA has downloaded a table, all LIKE and SIMILAR TO queries on that table appear to run faster.

For AQUA to improve performance, enough queries must be handed before the

table needs to be downloaded again that the improvement in performance for each query occurs enough times to outweigh the large initial second overhead.

If this is not the case, if only one or a few queries are issued, AQUA will reduce performance, not improve it.

There are no findings in this investigation about when that initial delay is repeated.

AQUA is invoked when and only when a query uses a LIKE or SIMILAR TO operator, which is a narrow use case, and it seems, for the limited testing done, *always* to be invoked. I suspect this is probably the case, as if more intelligence were being used, DISTINCT would not be being issued to AQUA.

Where AQUA and Spectrum are the same fundamental design, they are operated in the same way; queries which use AQUA or Spectrum must be carefully crafted to ensure the queries issue work to AQUA or Spectrum which those systems can perform, with only minimal aggregated results being returned the Redshift cluster.

If this is not done, if the queries issued are inappropriate, both AQUA and Spectrum necessarily must return some, most or even all of the tables involved to the Redshift cluster.

With Spectrum, where the tables are intended to be larger than the cluster can handle, this typically will grind the cluster to a halt for a considerable period of time, until the cluster finally fills all available disk and then kills the query.

With AQUA, the symptoms should be more tolerable, as the tables in use typically are going to be normal Redshift tables, of a size the cluster can handle without actually grinding to a halt.

Out of the final two runs of 76 test queries, where query times were closely examined, 3 queries were found to have had exceptionally long run times - in one case, over *five minutes*, compared to the usual time of about 45 seconds.

No investigation has been performed, so it is not clear if the issue is AQUA, RMS, or the cluster. Nevertheless, such delays are not expected, or so frequently, on a completely idle cluster running the simplest of queries.

RMS

A number of serendipitous discoveries were made about RMS.

1. The total disk on a two node `ra3.xlplus` is 1,908,734 blocks. During the work to over-fill the cache, it was found the total disk used would peak at about 1,500,000 blocks and then drop to about 1,300,000 blocks.
2. RMS is block-based storage, rather than table-based storage. Redshift, presumably by using information in the Zone Map, brings down from RMS only those blocks which are necessary to fulfil a query; a table then can be partially in RMS and partially in the local SSD cache.
3. With `ra3.xlplus` nodes, the bandwidth to RMS is about 160mb/sec per node.

4. With `ra3.xlplus` nodes, the bandwidth to the local SSD cache is about 720mb/sec per node.

Revision History

v1

- Initial release.

v2

- Reworded some criticism of AWS, to be moderately more diplomatic.

v3

- Minor improvements to abstract.

v4

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

v5

- Added "About the Author". made site name in title a link, and made each chapter start a new page.

v6

- `AQUA` really is now mandatory. Introduction updated.

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.

General Experiments

```
{'us-east-1': {'ra3.xlplus': {'LIKE': [[0,
                                        datetime.timedelta(seconds=1, microseconds=508120),
                                        7,
                                        56,
                                        35],
                                       [1,
                                        datetime.timedelta(seconds=1, microseconds=630037),
                                        7,
                                        56,
                                        35],
                                       [2,
                                        datetime.timedelta(seconds=3, microseconds=197689),
                                        7,
                                        56,
                                        35],
                                       [3,
                                        datetime.timedelta(seconds=1, microseconds=562073),
                                        7,
                                        56,
                                        35]],
                                'LIKE, COMPARE': [[0,
                                                   datetime.timedelta(seconds=1, microseconds=602977),
                                                   7,
                                                   56,
                                                   35],
                                                  [1,
                                                   datetime.timedelta(seconds=1, microseconds=679212),
                                                   7,
                                                   56,
                                                   35],
                                                  [2,
                                                   datetime.timedelta(seconds=1, microseconds=695224),
                                                   7,
                                                   56,
                                                   35],
                                                  [3,
                                                   datetime.timedelta(microseconds=71109),
                                                   0,
                                                   0,
                                                   35]],
                                'LIKE, COMPARE, MAX': [[0,
                                                        datetime.timedelta(seconds=1, microseconds=595360),
                                                        7,
                                                        70,
                                                        35],
                                                       [1,
                                                        datetime.timedelta(seconds=1, microseconds=631216),
                                                        7,
                                                        70,
                                                        35],
                                                       [2,
                                                        datetime.timedelta(seconds=1, microseconds=735196),
                                                        7,
                                                        70,
                                                        35],
                                                       [3,
                                                        datetime.timedelta(microseconds=90364),
                                                        0,
                                                        0,
```

```
                     35]],
'LIKE, DISTINCT': [[0,
                    datetime.timedelta(seconds=2, microseconds=898145),
                    1631721,
                    1670882304,
                    35],
                   [1,
                    datetime.timedelta(seconds=2, microseconds=843230),
                    1631721,
                    1670882304,
                    35],
                   [2,
                    datetime.timedelta(seconds=2, microseconds=944288),
                    1631721,
                    1670882304,
                    35],
                   [3,
                    datetime.timedelta(seconds=2, microseconds=886223),
                    1631721,
                    1670882304,
                    35]],
'LIKE, GROUP BY': [[0,
                    datetime.timedelta(seconds=1, microseconds=723577),
                    7,
                    70,
                    35],
                   [1,
                    datetime.timedelta(seconds=1, microseconds=719165),
                    7,
                    70,
                    35],
                   [2,
                    datetime.timedelta(seconds=3, microseconds=301341),
                    7,
                    70,
                    35],
                   [3,
                    datetime.timedelta(seconds=1, microseconds=626814),
                    7,
                    70,
                    35]],
'NO LIKE, NO SIMILAR TO': [[0,
                            datetime.timedelta(seconds=6, microseconds=54411),
                            1631721,
                            1683936072,
                            2],
                           [1,
                            datetime.timedelta(seconds=6, microseconds=104342),
                            1631721,
                            1683936072,
                            2],
                           [2,
                            datetime.timedelta(seconds=6, microseconds=82442),
                            1631721,
                            1683936072,
                            2],
                           [3,
                            datetime.timedelta(seconds=6, microseconds=85886),
                            1631721,
                            1683936072,
                            2]],
'NO LIKE, NO SIMILAR TO, GROUP BY': [[0,
                                      datetime.timedelta(seconds=6, microseconds=744856),
                                      1631721,
                                      1687199514,
                                      2],
                                     [1,
                                      datetime.timedelta(seconds=6, microseconds=885083),
                                      1631721,
                                      1687199514,
                                      2],
                                     [2,
                                      datetime.timedelta(seconds=6, microseconds=715905),
                                      1631721,
                                      1687199514,
                                      2],
                                     [3,
                                      datetime.timedelta(seconds=6, microseconds=720547),
                                      1631721,
                                      1687199514,
                                      2]]}}}
```

Main Test

The dictionary structure is;

`[region][node_type_name]['aqua'][aqua_enabled_flag][rms_flag][compare_type]`

aqua_enabled_flag	meaning
True	AQUA enabled
False	AQUA disabled

rms_flag	meaning
True	table is in RMS only
False	table is in RMS and local cache

The `compare_type` can be `equality`, `like_%`, `like_a`, `like_b`, `like_c`, or `similar to`.

Of these, `equality` will not invoke AQUA. The others all will (assuming it is enabled in the cluster).

The results are;

`[ [slice, duration, rows, bytes, type], disk_before, disk_after ]`

The initial list is taken from `stl_scan`, and is for the first scan step of the query, which is the relevant scan step.

Datum	Meaning
slice	slice number
duration	time taken to execute the first segment on this slice
rows	number of rows processed by the `scan` step
bytes	number of byets processed by the `scan` step
type	type of scan peformed; 35 means an AQUA scan, 2 is a normal non-AQUA scan
disk_before	total disk spaces used, from `stv_partitions`, immediately before the query
disk_after	total disk spaces used, from `stv_partitions`, immediately after the query

AQUA Disabled


```
{'us-east-1': {'ra3.xlplus': {'aqua': {False: {False: {'equality': {0: [[[0,
                                                                           datetime.timedelta(seconds=9, microseconds=207733),
                                                                           1631721,
                                                                           1683936072,
                                                                           2],
                                                                          [1,
                                                                           datetime.timedelta(seconds=9, microseconds=204580),
                                                                           1631721,
                                                                           1683936072,
                                                                           2],
                                                                          [2,
                                                                           datetime.timedelta(seconds=9, microseconds=13693),
```

```
                                 1631721,
                                 1683936072,
                                 2],
                                [3,
                                 datetime.timedelta(seconds=8, microseconds=998986),
                                 1631721,
                                 1683936072,
                                 2]],
                               90997,
                               90997],
                           1: [[[0,
                                 datetime.timedelta(seconds=8, microseconds=993426),
                                 1631721,
                                 1683936072,
                                 2],
                                [1,
                                 datetime.timedelta(seconds=8, microseconds=988677),
                                 1631721,
                                 1683936072,
                                 2],
                                [2,
                                 datetime.timedelta(seconds=9, microseconds=344057),
                                 1631721,
                                 1683936072,
                                 2],
                                [3,
                                 datetime.timedelta(seconds=9, microseconds=348855),
                                 1631721,
                                 1683936072,
                                 2]],
                               90997,
                               90997],
                           2: [[[0,
                                 datetime.timedelta(seconds=9, microseconds=250344),
                                 1631721,
                                 1683936072,
                                 2],
                                [1,
                                 datetime.timedelta(seconds=9, microseconds=248743),
                                 1631721,
                                 1683936072,
                                 2],
                                [2,
                                 datetime.timedelta(seconds=9, microseconds=550706),
                                 1631721,
                                 1683936072,
                                 2],
                                [3,
                                 datetime.timedelta(seconds=9, microseconds=555538),
                                 1631721,
                                 1683936072,
                                 2]],
                               90997,
                               90997]},
'like_%': {0: [[[0,
                                 datetime.timedelta(seconds=9, microseconds=269197),
                                 3263442,
                                 3367872144,
                                 2],
                                [1,
                                 datetime.timedelta(seconds=9, microseconds=262832),
                                 3263442,
                                 3367872144,
                                 2],
                                [2,
                                 datetime.timedelta(seconds=9, microseconds=546041),
                                 3263442,
                                 3367872144,
                                 2],
                                [3,
                                 datetime.timedelta(seconds=9, microseconds=547510),
                                 3263442,
                                 3367872144,
                                 2]],
                               90997,
                               90997],
                           1: [[[0,
                                 datetime.timedelta(seconds=9, microseconds=249538),
                                 3263442,
                                 3367872144,
                                 2],
                                [1,
                                 datetime.timedelta(seconds=9, microseconds=244753),
                                 3263442,
                                 3367872144,
                                 2],
                                [2,
                                 datetime.timedelta(seconds=9, microseconds=559937),
                                 3263442,
                                 3367872144,
                                 2],
                                [3,
                                 datetime.timedelta(seconds=9, microseconds=565394),
                                 3263442,
                                 3367872144,
                                 2]],
                               90997,
                               90997],
                           2: [[[0,
```

```
                    datetime.timedelta(seconds=9, microseconds=249663),
                    3263442,
                    3367872144,
                    2],
                   [1,
                    datetime.timedelta(seconds=9, microseconds=244878),
                    3263442,
                    3367872144,
                    2],
                   [2,
                    datetime.timedelta(seconds=9, microseconds=575789),
                    3263442,
                    3367872144,
                    2],
                   [3,
                    datetime.timedelta(seconds=9, microseconds=578883),
                    3263442,
                    3367872144,
                    2]],
                  90997,
                  90997]},
 'like_a': {0: [[[0,
                    datetime.timedelta(seconds=9, microseconds=267869),
                    1631721,
                    1683936072,
                    2],
                   [1,
                    datetime.timedelta(seconds=9, microseconds=269394),
                    1631721,
                    1683936072,
                    2],
                   [2,
                    datetime.timedelta(seconds=9, microseconds=453487),
                    1631721,
                    1683936072,
                    2],
                   [3,
                    datetime.timedelta(seconds=9, microseconds=451897),
                    1631721,
                    1683936072,
                    2]],
                  90997,
                  90997],
            1: [[[0,
                    datetime.timedelta(seconds=9, microseconds=250844),
                    1631721,
                    1683936072,
                    2],
                   [1,
                    datetime.timedelta(seconds=9, microseconds=249226),
                    1631721,
                    1683936072,
                    2],
                   [2,
                    datetime.timedelta(seconds=9, microseconds=404371),
                    1631721,
                    1683936072,
                    2],
                   [3,
                    datetime.timedelta(seconds=9, microseconds=409178),
                    1631721,
                    1683936072,
                    2]],
                  90997,
                  90997],
            2: [[[0,
                    datetime.timedelta(seconds=9, microseconds=249227),
                    1631721,
                    1683936072,
                    2],
                   [1,
                    datetime.timedelta(seconds=9, microseconds=250828),
                    1631721,
                    1683936072,
                    2],
                   [2,
                    datetime.timedelta(seconds=9, microseconds=335284),
                    1631721,
                    1683936072,
                    2],
                   [3,
                    datetime.timedelta(seconds=9, microseconds=349675),
                    1631721,
                    1683936072,
                    2]],
                  90997,
                  90997]},
 'like_b': {0: [[[0,
                    datetime.timedelta(seconds=9, microseconds=265181),
                    1631721,
                    1683936072,
                    2],
                   [1,
                    datetime.timedelta(seconds=9, microseconds=271362),
                    1631721,
                    1683936072,
                    2],
                   [2,
                    datetime.timedelta(seconds=9, microseconds=455281),
```

```
                                1631721,
                                1683936072,
                                2],
                               [3,
                                datetime.timedelta(seconds=9, microseconds=448595),
                                1631721,
                                1683936072,
                                2]],
                              90997,
                              90997],
                          1: [[[0,
                                datetime.timedelta(seconds=9, microseconds=310843),
                                1631721,
                                1683936072,
                                2],
                               [1,
                                datetime.timedelta(seconds=9, microseconds=316331),
                                1631721,
                                1683936072,
                                2],
                               [2,
                                datetime.timedelta(seconds=9, microseconds=608579),
                                1631721,
                                1683936072,
                                2],
                               [3,
                                datetime.timedelta(seconds=9, microseconds=597943),
                                1631721,
                                1683936072,
                                2]],
                              90997,
                              90997],
                          2: [[[0,
                                datetime.timedelta(seconds=9, microseconds=250475),
                                1631721,
                                1683936072,
                                2],
                               [1,
                                datetime.timedelta(seconds=9, microseconds=245907),
                                1631721,
                                1683936072,
                                2],
                               [2,
                                datetime.timedelta(seconds=9, microseconds=600638),
                                1631721,
                                1683936072,
                                2],
                               [3,
                                datetime.timedelta(seconds=9, microseconds=597784),
                                1631721,
                                1683936072,
                                2]],
                              90997,
                              90997]},
'like_c': {0: [[[0,
                                datetime.timedelta(seconds=9, microseconds=262156),
                                0,
                                0,
                                2],
                               [1,
                                datetime.timedelta(seconds=9, microseconds=268511),
                                0,
                                0,
                                2],
                               [2,
                                datetime.timedelta(seconds=9, microseconds=579287),
                                0,
                                0,
                                2],
                               [3,
                                datetime.timedelta(seconds=9, microseconds=589439),
                                0,
                                0,
                                2]],
                              90997,
                              90997],
                          1: [[[0,
                                datetime.timedelta(seconds=9, microseconds=251058),
                                0,
                                0,
                                2],
                               [1,
                                datetime.timedelta(seconds=9, microseconds=249509),
                                0,
                                0,
                                2],
                               [2,
                                datetime.timedelta(seconds=9, microseconds=408094),
                                0,
                                0,
                                2],
                               [3,
                                datetime.timedelta(seconds=9, microseconds=409781),
                                0,
                                0,
                                2]],
                              90997,
                              90997],
                          2: [[[0,
```

```
                                        datetime.timedelta(seconds=9, microseconds=251943),
                                        0,
                                        0,
                                        2],
                                       [1,
                                        datetime.timedelta(seconds=9, microseconds=247216),
                                        0,
                                        0,
                                        2],
                                       [2,
                                        datetime.timedelta(seconds=9, microseconds=381876),
                                        0,
                                        0,
                                        2],
                                       [3,
                                        datetime.timedelta(seconds=9, microseconds=385503),
                                        0,
                                        0,
                                        2]],
                                      90997,
                                      90997]},
                'similar to': {0: [[[0,
                                     datetime.timedelta(seconds=9, microseconds=298151),
                                     1631721,
                                     1683936072,
                                     2],
                                    [1,
                                     datetime.timedelta(seconds=9, microseconds=192925),
                                     1631721,
                                     1683936072,
                                     2],
                                    [2,
                                     datetime.timedelta(seconds=9, microseconds=338577),
                                     1631721,
                                     1683936072,
                                     2],
                                    [3,
                                     datetime.timedelta(seconds=9, microseconds=522980),
                                     1631721,
                                     1683936072,
                                     2]],
                                   90997,
                                   90997],
                                1: [[[0,
                                     datetime.timedelta(seconds=9, microseconds=242530),
                                     1631721,
                                     1683936072,
                                     2],
                                    [1,
                                     datetime.timedelta(seconds=9, microseconds=259425),
                                     1631721,
                                     1683936072,
                                     2],
                                    [2,
                                     datetime.timedelta(seconds=9, microseconds=474999),
                                     1631721,
                                     1683936072,
                                     2],
                                    [3,
                                     datetime.timedelta(seconds=9, microseconds=333545),
                                     1631721,
                                     1683936072,
                                     2]],
                                   90997,
                                   90997],
                                2: [[[0,
                                     datetime.timedelta(seconds=9, microseconds=270805),
                                     1631721,
                                     1683936072,
                                     2],
                                    [1,
                                     datetime.timedelta(seconds=9, microseconds=216178),
                                     1631721,
                                     1683936072,
                                     2],
                                    [2,
                                     datetime.timedelta(seconds=9, microseconds=489395),
                                     1631721,
                                     1683936072,
                                     2],
                                    [3,
                                     datetime.timedelta(seconds=9, microseconds=316543),
                                     1631721,
                                     1683936072,
                                     2]],
                                   90997,
                                   90997]}},
 True: {'equality': {0: [[[0,
                           datetime.timedelta(seconds=49, microseconds=576080),
                           1631721,
                           1683936072,
                           2],
                          [1,
                           datetime.timedelta(seconds=48, microseconds=813398),
                           1631721,
                           1683936072,
                           2],
                          [2,
                           datetime.timedelta(seconds=53, microseconds=204578),
```

```
                      1631721,
                      1683936072,
                      2],
                     [3,
                      datetime.timedelta(seconds=53, microseconds=571728),
                      1631721,
                      1683936072,
                      2]],
                    1427117,
                    1439879],
                1: [[[0,
                      datetime.timedelta(seconds=9, microseconds=224023),
                      1631721,
                      1683936072,
                      2],
                     [1,
                      datetime.timedelta(seconds=9, microseconds=420448),
                      1631721,
                      1683936072,
                      2],
                     [2,
                      datetime.timedelta(seconds=9, microseconds=201846),
                      1631721,
                      1683936072,
                      2],
                     [3,
                      datetime.timedelta(seconds=8, microseconds=697027),
                      1631721,
                      1683936072,
                      2]],
                    1439879,
                    1440006],
                2: [[[0,
                      datetime.timedelta(seconds=9, microseconds=261867),
                      1631721,
                      1683936072,
                      2],
                     [1,
                      datetime.timedelta(seconds=9, microseconds=265024),
                      1631721,
                      1683936072,
                      2],
                     [2,
                      datetime.timedelta(seconds996555=9, microseconds=348863),
                      1631721,
                      1683936072,
                      2],
                     [3,
                      datetime.timedelta(seconds=9, microseconds=345673),
                      1631721,
                      1683936072,
                      2]],
                    1440006,
                    1440006]},
'like_%': {0: [[[0,
                 datetime.timedelta(seconds=51, microseconds=763577),
                 3263442,
                 3367872144,
                 2],
                [1,
                 datetime.timedelta(seconds=51, microseconds=676877),
                 3263442,
                 3367872144,
                 2],
                [2,
                 datetime.timedelta(seconds=61, microseconds=149151),
                 3263442,
                 3367872144,
                 2],
                [3,
                 datetime.timedelta(seconds=61, microseconds=664853),
                 3263442,
                 3367872144,
                 2]],
               1440006,
               1452862],
           1: [[[0,
                 datetime.timedelta(seconds=8, microseconds=608361),
                 3263442,
                 3367872144,
                 2],
                [1,
                 datetime.timedelta(seconds=8, microseconds=603606),
                 3263442,
                 3367872144,
                 2],
                [2,
                 datetime.timedelta(seconds=8, microseconds=657271),
                 3263442,
                 3367872144,
                 2],
                [3,
                 datetime.timedelta(seconds=8, microseconds=652435),
                 3263442,
                 3367872144,
                 2]],
               1452862,
               1452862],
           2: [[[0,
```

```
                    datetime.timedelta(seconds=9, microseconds=249899),
                    3263442,
                    3367872144,
                    2],
                   [1,
                    datetime.timedelta(seconds=9, microseconds=244920),
                    3263442,
                    3367872144,
                    2],
                   [2,
                    datetime.timedelta(seconds=9, microseconds=349570),
                    3263442,
                    3367872144,
                    2],
                   [3,
                    datetime.timedelta(seconds=9, microseconds=343210),
                    3263442,
                    3367872144,
                    2]],
                  1452862,
                  1452862]},
'like_a': {0: [[[0,
                   datetime.timedelta(seconds=68, microseconds=177246),
                   1631721,
                   1683936072,
                   2],
                  [1,
                   datetime.timedelta(seconds=68, microseconds=683746),
                   1631721,
                   1683936072,
                   2],
                  [2,
                   datetime.timedelta(seconds=48, microseconds=815787),
                   1631721,
                   1683936072,
                   2],
                  [3,
                   datetime.timedelta(seconds=50, microseconds=480384),
                   1631721,
                   1683936072,
                   2]],
                 1452862,
                 1465718],
           1: [[[0,
                   datetime.timedelta(seconds=8, microseconds=525502),
                   1631721,
                   1683936072,
                   2],
                  [1,
                   datetime.timedelta(seconds=8, microseconds=530090),
                   1631721,
                   1683936072,
                   2],
                  [2,
                   datetime.timedelta(seconds=8, microseconds=568479),
                   1631721,
                   1683936072,
                   2],
                  [3,
                   datetime.timedelta(seconds=8, microseconds=573761),
                   1631721,
                   1683936072,
                   2]],
                 1465718,
                 1465718],
           2: [[[0,
                   datetime.timedelta(seconds=9, microseconds=247934),
                   1631721,
                   1683936072,
                   2],
                  [1,
                   datetime.timedelta(seconds=9, microseconds=250880),
                   1631721,
                   1683936072,
                   2],
                  [2,
                   datetime.timedelta(seconds=9, microseconds=349531),
                   1631721,
                   1683936072,
                   2],
                  [3,
                   datetime.timedelta(seconds=9, microseconds=347938),
                   1631721,
                   1683936072,
                   2]],
                 1465718,
                 1465718]},
'like_b': {0: [[[0,
                   datetime.timedelta(seconds=51, microseconds=321767),
                   1631721,
                   1683936072,
                   2],
                  [1,
                   datetime.timedelta(seconds=53, microseconds=27248),
                   1631721,
                   1683936072,
                   2],
                  [2,
                   datetime.timedelta(seconds=52, microseconds=187921),
```

```
                      1631721,
                      1683936072,
                      2],
                     [3,
                      datetime.timedelta(seconds=52, microseconds=88932),
                      1631721,
                      1683936072,
                      2]],
                    1465718,
                    1478574],
                1: [[[0,
                      datetime.timedelta(seconds=8, microseconds=710603),
                      1631721,
                      1683936072,
                      2],
                     [1,
                      datetime.timedelta(seconds=8, microseconds=705800),
                      1631721,
                      1683936072,
                      2],
                     [2,
                      datetime.timedelta(seconds=9, microseconds=350113),
                      1631721,
                      1683936072,
                      2],
                     [3,
                      datetime.timedelta(seconds=9, microseconds=346366),
                      1631721,
                      1683936072,
                      2]],
                    1478574,
                    1478574],
                2: [[[0,
                      datetime.timedelta(seconds=8, microseconds=797242),
                      1631721,
                      1683936072,
                      2],
                     [1,
                      datetime.timedelta(seconds=8, microseconds=802712),
                      1631721,
                      1683936072,
                      2],
                     [2,
                      datetime.timedelta(seconds=9, microseconds=354782),
                      1631721,
                      1683936072,
                      2],
                     [3,
                      datetime.timedelta(seconds=9, microseconds=349460),
                      1631721,
                      1683936072,
                      2]],
                    1478574,
                    1478574]},
 'like_c': {0: [[[0,
                      datetime.timedelta(seconds=50, microseconds=910098),
                      0,
                      0,
                      2],
                     [1,
                      datetime.timedelta(seconds=50, microseconds=761279),
                      0,
                      0,
                      2],
                     [2,
                      datetime.timedelta(seconds=49, microseconds=429202),
                      0,
                      0,
                      2],
                     [3,
                      datetime.timedelta(seconds=49, microseconds=508892),
                      0,
                      0,
                      2]],
                    1478574,
                    1491171],
                1: [[[0,
                      datetime.timedelta(seconds=8, microseconds=354100),
                      0,
                      0,
                      2],
                     [1,
                      datetime.timedelta(seconds=8, microseconds=339638),
                      0,
                      0,
                      2],
                     [2,
                      datetime.timedelta(seconds=8, microseconds=398365),
                      0,
                      0,
                      2],
                     [3,
                      datetime.timedelta(seconds=8, microseconds=226090),
                      0,
                      0,
                      2]],
                    1491171,
                    1491441],
                2: [[[0,
```

```
                                          datetime.timedelta(seconds=9, microseconds=250590),
                                          0,
                                          0,
                                          2],
                                         [1,
                                          datetime.timedelta(seconds=9, microseconds=252165),
                                          0,
                                          0,
                                          2],
                                         [2,
                                          datetime.timedelta(seconds=9, microseconds=351336),
                                          0,
                                          0,
                                          2],
                                         [3,
                                          datetime.timedelta(seconds=9, microseconds=346647),
                                          0,
                                          0,
                                          2]],
                                        1491441,
                                        1491441]},
                    'similar to': {0: [[[0,
                                         datetime.timedelta(seconds=55, microseconds=580887),
                                         1631721,
                                         1683936072,
                                         2],
                                        [1,
                                         datetime.timedelta(seconds=56, microseconds=749457),
                                         1631721,
                                         1683936072,
                                         2],
                                        [2,
                                         datetime.timedelta(seconds=52, microseconds=61169),
                                         1631721,
                                         1683936072,
                                         2],
                                        [3,
                                         datetime.timedelta(seconds=53, microseconds=48903),
                                         1631721,
                                         1683936072,
                                         2]],
                                       1491441,
                                       1504297],
                                    1: [[[0,
                                         datetime.timedelta(seconds=9, microseconds=57150),
                                         1631721,
                                         1683936072,
                                         2],
                                        [1,
                                         datetime.timedelta(seconds=9, microseconds=46908),
                                         1631721,
                                         1683936072,
                                         2],
                                        [2,
                                         datetime.timedelta(seconds=9, microseconds=121725),
                                         1631721,
                                         1683936072,
                                         2],
                                        [3,
                                         datetime.timedelta(seconds=9, microseconds=127153),
                                         1631721,
                                         1683936072,
                                         2]],
                                       1504297,
                                       1504297],
                                    2: [[[0,
                                         datetime.timedelta(seconds=9, microseconds=265822),
                                         1631721,
                                         1683936072,
                                         2],
                                        [1,
                                         datetime.timedelta(seconds=9, microseconds=259137),
                                         1631721,
                                         1683936072,
                                         2],
                                        [2,
                                         datetime.timedelta(seconds=9, microseconds=366165),
                                         1631721,
                                         1683936072,
                                         2],
                                        [3,
                                         datetime.timedelta(seconds=9, microseconds=371360),
                                         1631721,
                                         1683936072,
                                         2]],
                                       1504297,
                                       1504297]}}}},
'cluster': {False: {'Clusters': [{'AllowVersionUpgrade': True,
                                  'AquaConfiguration': {'AquaConfigurationStatus': 'disabled',
                                                        'AquaStatus': 'disabled'},
                                  'AutomatedSnapshotRetentionPeriod': 1,
                                  'AvailabilityZone': 'us-east-1a',
                                  'AvailabilityZoneRelocationStatus': 'disabled',
                                  'ClusterAvailabilityStatus': 'Available',
                                  'ClusterCreateTime': datetime.datetime(2022, 0, 0, 0, 0, 0, 0, tzinfo=tzutc()),
                                  'ClusterIdentifier': 'nnn',
                                  'ClusterNamespaceArn': 'arn:aws:redshift:us-east-1:113054258080:namespace:nnn',
                                  'ClusterNodes': [{'NodeRole': 'LEADER',
                                                    'PrivateIPAddress': 'nnn',
```

```
                                 'PublicIPAddress': 'nnn'},
                                {'NodeRole': 'COMPUTE-0',
                                 'PrivateIPAddress': 'nnn',
                                 'PublicIPAddress': 'nnn'},
                                {'NodeRole': 'COMPUTE-1',
                                 'PrivateIPAddress': 'nnn',
                                 'PublicIPAddress': 'nnn'}],
                  'ClusterParameterGroups': [{'ParameterApplyStatus': 'in-sync',
                                              'ParameterGroupName': 'nnn'}],
                  'ClusterPublicKey': 'ssh-rsa '
                                      'nnn'
                                      'Amazon-Redshift\n',
                  'ClusterRevisionNumber': '41533',
                  'ClusterSecurityGroups': [],
                  'ClusterStatus': 'available',
                  'ClusterSubnetGroupName': 'nnn',
                  'ClusterVersion': '1.0',
                  'DBName': 'dev',
                  'DeferredMaintenanceWindows': [],
                  'ElasticResizeNumberOfNodeOptions': '[4]',
                  'Encrypted': False,
                  'Endpoint': {'Address': 'nnn.us-east-1.redshift.amazonaws.com',
                               'Port': 5439},
                  'EnhancedVpcRouting': False,
                  'IamRoles': [],
                  'MaintenanceTrackName': 'current',
                  'ManualSnapshotRetentionPeriod': 1,
                  'MasterUsername': 'admin',
                  'NextMaintenanceWindowStartTime': datetime.datetime(2022, 0, 0, 0, 0, tzinfo=tzutc()),
                  'NodeType': 'ra3.xlplus',
                  'NumberOfNodes': 2,
                  'PendingModifiedValues': {},
                  'PreferredMaintenanceWindow': 'nnn',
                  'PubliclyAccessible': True,
                  'Tags': [],
                  'VpcId': 'vpc-081763ed917bf7994',
                  'VpcSecurityGroups': [{'Status': 'active',
                                         'VpcSecurityGroupId': 'nnn'}]},
 'ResponseMetadata': {'HTTPHeaders': {'content-length': 'n',
                                      'content-type': 'text/xml',
                                      'date': 'n, '
                                              'n '
                                              'n '
                                              'n '
                                              'n '
                                              'GMT',
                                      'vary': 'accept-encoding',
                                      'x-amzn-requestid': 'nnn'},
                      'HTTPStatusCode': 200,
                      'RequestId': 'nnn',
                      'RetryAttempts': 0}}}}}}
```

AQUA Enabled

```
{'us-east-1': {'ra3.xlplus': {'aqua': {True: {False: {'equality': {0: [[[0,
                                                                          datetime.timedelta(seconds=9, microseconds=204733),
                                                                          1631721,
                                                                          1683936072,
                                                                          2],
                                                                         [1,
                                                                          datetime.timedelta(seconds=9, microseconds=176037),
                                                                          1631721,
                                                                          1683936072,
                                                                          2],
                                                                         [2,
                                                                          datetime.timedelta(seconds=8, microseconds=823521),
                                                                          1631721,
                                                                          1683936072,
                                                                          2],
                                                                         [3,
                                                                          datetime.timedelta(seconds=8, microseconds=822097),
                                                                          1631721,
                                                                          1683936072,
                                                                          2]],
                                                                        90999,
                                                                        91016],
                                                                   1: [[[0,
                                                                          datetime.timedelta(seconds=9, microseconds=248911),
                                                                          1631721,
                                                                          1683936072,
                                                                          2],
                                                                         [1,
                                                                          datetime.timedelta(seconds=9, microseconds=250495),
                                                                          1631721,
                                                                          1683936072,
                                                                          2],
                                                                         [2,
                                                                          datetime.timedelta(seconds=8, microseconds=578060),
                                                                          1631721,
                                                                          1683936072,
                                                                          2],
                                                                         [3,
                                                                          datetime.timedelta(seconds=8, microseconds=576426),
                                                                          1631721,
                                                                          1683936072,
                                                                          2]],
```

```
                    91016,
                    91016],
                2: [[[0,
                      datetime.timedelta(seconds=9, microseconds=247384),
                      1631721,
                      1683936072,
                      2],
                     [1,
                      datetime.timedelta(seconds=9, microseconds=250562),
                      1631721,
                      1683936072,
                      2],
                     [2,
                      datetime.timedelta(seconds=9, microseconds=352312),
                      1631721,
                      1683936072,
                      2],
                     [3,
                      datetime.timedelta(seconds=9, microseconds=350722),
                      1631721,
                      1683936072,
                      2]],
                    91016,
                    91016]},
 'like_%': {0: [[[0,
                  datetime.timedelta(seconds=43, microseconds=327411),
                  7,
                  56,
                  35],
                 [1,
                  datetime.timedelta(seconds=43, microseconds=899995),
                  7,
                  56,
                  35],
                 [2,
                  datetime.timedelta(seconds=41, microseconds=643454),
                  7,
                  56,
                  35],
                 [3,
                  datetime.timedelta(seconds=42, microseconds=943825),
                  7,
                  56,
                  35]],
                91016,
                91016],
            1: [[[0,
                  datetime.timedelta(seconds=1, microseconds=744283),
                  7,
                  56,
                  35],
                 [1,
                  datetime.timedelta(seconds=1, microseconds=791337),
                  7,
                  56,
                  35],
                 [2,
                  datetime.timedelta(seconds=3, microseconds=399622),
                  7,
                  56,
                  35],
                 [3,
                  datetime.timedelta(seconds=1, microseconds=745078),
                  7,
                  56,
                  35]],
                91016,
                91016],
            2: [[[0,
                  datetime.timedelta(seconds=1, microseconds=731125),
                  7,
                  56,
                  35],
                 [1,
                  datetime.timedelta(seconds=4, microseconds=634349),
                  7,
                  56,
                  35],
                 [2,
                  datetime.timedelta(seconds=1, microseconds=807203),
                  7,
                  56,
                  35],
                 [3,
                  datetime.timedelta(seconds=1, microseconds=766072),
                  7,
                  56,
                  35]],
                91016,
                91016]},
 'like_a': {0: [[[0,
                  datetime.timedelta(seconds=41, microseconds=195449),
                  7,
                  56,
                  35],
                 [1,
                  datetime.timedelta(seconds=41, microseconds=796387),
                  7,
                  56,
```

```
                                35],
                               [2,
                                datetime.timedelta(seconds=43, microseconds=170872),
                                7,
                                56,
                                35],
                               [3,
                                datetime.timedelta(seconds=41, microseconds=442968),
                                7,
                                56,
                                35]],
                              91016,
                              91016],
                          1: [[[0,
                                datetime.timedelta(seconds=1, microseconds=559614),
                                7,
                                56,
                                35],
                               [1,
                                datetime.timedelta(seconds=2, microseconds=288364),
                                7,
                                56,
                                35],
                               [2,
                                datetime.timedelta(seconds=1, microseconds=580123),
                                7,
                                56,
                                35],
                               [3,
                                datetime.timedelta(seconds=1, microseconds=581609),
                                7,
                                56,
                                35]],
                              91016,
                              91016],
                          2: [[[0,
                                datetime.timedelta(seconds=1, microseconds=664650),
                                7,
                                56,
                                35],
                               [1,
                                datetime.timedelta(seconds=1, microseconds=589867),
                                7,
                                56,
                                35],
                               [2,
                                datetime.timedelta(seconds=1, microseconds=605557),
                                7,
                                56,
                                35],
                               [3,
                                datetime.timedelta(seconds=1, microseconds=574602),
                                7,
                                56,
                                35]],
                              91016,
                              91016]},
 'like_b': {0: [[[0,
                                datetime.timedelta(seconds=43, microseconds=572248),
                                7,
                                56,
                                35],
                               [1,
                                datetime.timedelta(seconds=42, microseconds=263507),
                                7,
                                56,
                                35],
                               [2,
                                datetime.timedelta(seconds=42, microseconds=688915),
                                7,
                                56,
                                35],
                               [3,
                                datetime.timedelta(seconds=46, microseconds=140976),
                                7,
                                56,
                                35]],
                              91016,
                              91016],
                          1: [[[0,
                                datetime.timedelta(seconds=1, microseconds=851338),
                                7,
                                56,
                                35],
                               [1,
                                datetime.timedelta(seconds=1, microseconds=867054),
                                7,
                                56,
                                35],
                               [2,
                                datetime.timedelta(seconds=1, microseconds=858184),
                                7,
                                56,
                                35],
                               [3,
                                datetime.timedelta(seconds=1, microseconds=770956),
                                7,
                                56,
                                35]],
```

```
                    91016,
                    91016],
                2: [[[0,
                      datetime.timedelta(seconds=1, microseconds=623430),
                      7,
                      56,
                      35],
                     [1,
                      datetime.timedelta(seconds=1, microseconds=566698),
                      7,
                      56,
                      35],
                     [2,
                      datetime.timedelta(seconds=1, microseconds=577199),
                      7,
                      56,
                      35],
                     [3,
                      datetime.timedelta(seconds=1, microseconds=511778),
                      7,
                      56,
                      35]],
                    91016,
                    91016]},
 'like_c': {0: [[[0,
                  datetime.timedelta(seconds=42, microseconds=210546),
                  7,
                  56,
                  35],
                 [1,
                  datetime.timedelta(seconds=43, microseconds=961976),
                  7,
                  56,
                  35],
                 [2,
                  datetime.timedelta(seconds=42, microseconds=622863),
                  7,
                  56,
                  35],
                 [3,
                  datetime.timedelta(seconds=46, microseconds=603875),
                  7,
                  56,
                  35]],
                91016,
                91016],
            1: [[[0,
                  datetime.timedelta(microseconds=901284),
                  7,
                  56,
                  35],
                 [1,
                  datetime.timedelta(microseconds=917168),
                  7,
                  56,
                  35],
                 [2,
                  datetime.timedelta(microseconds=850170),
                  7,
                  56,
                  35],
                 [3,
                  datetime.timedelta(microseconds=905184),
                  7,
                  56,
                  35]],
                91016,
                91016],
            2: [[[0,
                  datetime.timedelta(seconds=1, microseconds=2802),
                  7,
                  56,
                  35],
                 [1,
                  datetime.timedelta(microseconds=972192),
                  7,
                  56,
                  35],
                 [2,
                  datetime.timedelta(microseconds=913016),
                  7,
                  56,
                  35],
                 [3,
                  datetime.timedelta(seconds=1, microseconds=145805),
                  7,
                  56,
                  35]],
                91016,
                91016]},
 'similar to': {0: [[[0,
                      datetime.timedelta(seconds=46, microseconds=94824),
                      7,
                      56,
                      35],
                     [1,
                      datetime.timedelta(seconds=42, microseconds=369193),
                      7,
                      56,
```

```
                                        35],
                                       [2,
                                        datetime.timedelta(seconds=48, microseconds=642382),
                                        7,
                                        56,
                                        35],
                                       [3,
                                        datetime.timedelta(seconds=41, microseconds=628065),
                                        7,
                                        56,
                                        35]],
                                      91016,
                                      91016],
                                  1: [[[0,
                                        datetime.timedelta(seconds=1, microseconds=521027),
                                        7,
                                        56,
                                        35],
                                       [1,
                                        datetime.timedelta(seconds=1, microseconds=583562),
                                        7,
                                        56,
                                        35],
                                       [2,
                                        datetime.timedelta(seconds=1, microseconds=595717),
                                        7,
                                        56,
                                        35],
                                       [3,
                                        datetime.timedelta(seconds=1, microseconds=567324),
                                        7,
                                        56,
                                        35]],
                                      91016,
                                      91016],
                                  2: [[[0,
                                        datetime.timedelta(seconds=1, microseconds=490310),
                                        7,
                                        56,
                                        35],
                                       [1,
                                        datetime.timedelta(seconds=1, microseconds=542446),
                                        7,
                                        56,
                                        35],
                                       [2,
                                        datetime.timedelta(seconds=1, microseconds=602099),
                                        7,
                                        56,
                                        35],
                                       [3,
                                        datetime.timedelta(seconds=1, microseconds=591313),
                                        7,
                                        56,
                                        35]],
                                      91016,
                                      91016]}},
 True: {'equality': {0: [[[0,
                           datetime.timedelta(seconds=50, microseconds=714166),
                           1631721,
                           1683936072,
                           2],
                          [1,
                           datetime.timedelta(seconds=49, microseconds=495406),
                           1631721,
                           1683936072,
                           2],
                          [2,
                           datetime.timedelta(seconds=49, microseconds=868700),
                           1631721,
                           1683936072,
                           2],
                          [3,
                           datetime.timedelta(seconds=49, microseconds=279289),
                           1631721,
                           1683936072,
                           2]],
                         1427117,
                         1439794],
                     1: [[[0,
                           datetime.timedelta(seconds=8, microseconds=931096),
                           1631721,
                           1683936072,
                           2],
                          [1,
                           datetime.timedelta(seconds=8, microseconds=995204),
                           1631721,
                           1683936072,
                           2],
                          [2,
                           datetime.timedelta(seconds=8, microseconds=328496),
                           1631721,
                           1683936072,
                           2],
                          [3,
                           datetime.timedelta(seconds=8, microseconds=772002),
                           1631721,
                           1683936072,
                           2]],
```

```
                    1439794,
                    1439981],
                2: [[[0,
                      datetime.timedelta(seconds=9, microseconds=272983),
                      1631721,
                      1683936072,
                      2],
                     [1,
                      datetime.timedelta(seconds=9, microseconds=278882),
                      1631721,
                      1683936072,
                      2],
                     [2,
                      datetime.timedelta(seconds=9, microseconds=349943),
                      1631721,
                      1683936072,
                      2],
                     [3,
                      datetime.timedelta(seconds=9, microseconds=346755),
                      1631721,
                      1683936072,
                      2]],
                    1439981,
                    1439981]},
'like_%': {0: [[[0,
                 datetime.timedelta(seconds=45, microseconds=996555),
                 7,
                 56,
                 35],
                [1,
                 datetime.timedelta(seconds=48, microseconds=870548),
                 7,
                 56,
                 35],
                [2,
                 datetime.timedelta(seconds=45, microseconds=798664),
                 7,
                 56,
                 35],
                [3,
                 datetime.timedelta(seconds=46, microseconds=332533),
                 7,
                 56,
                 35]],
               1439981,
               1439981],
           1: [[[0,
                 datetime.timedelta(seconds=1, microseconds=889942),
                 7,
                 56,
                 35],
                [1,
                 datetime.timedelta(seconds=1, microseconds=829568),
                 7,
                 56,
                 35],
                [2,
                 datetime.timedelta(seconds=1, microseconds=786090),
                 7,
                 56,
                 35],
                [3,
                 datetime.timedelta(seconds=3, microseconds=406696),
                 7,
                 56,
                 35]],
               1439981,
               1439981],
           2: [[[0,
                 datetime.timedelta(seconds=1, microseconds=773647),
                 7,
                 56,
                 35],
                [1,
                 datetime.timedelta(seconds=1, microseconds=848038),
                 7,
                 56,
                 35],
                [2,
                 datetime.timedelta(seconds=1, microseconds=817244),
                 7,
                 56,
                 35],
                [3,
                 datetime.timedelta(seconds=1, microseconds=824604),
                 7,
                 56,
                 35]],
               1439981,
               1439981]},
'like_a': {0: [[[0,
                 datetime.timedelta(seconds=46, microseconds=308171),
                 7,
                 56,
                 35],
                [1,
                 datetime.timedelta(seconds=104, microseconds=695472),
                 7,
                 56,
```

```
                                35],
                               [2,
                                datetime.timedelta(seconds=46, microseconds=558651),
                                7,
                                56,
                                35],
                               [3,
                                datetime.timedelta(seconds=48, microseconds=327062),
                                7,
                                56,
                                35]],
                              1439981,
                              1439987],
                             1: [[[0,
                                datetime.timedelta(seconds=1, microseconds=611775),
                                7,
                                56,
                                35],
                               [1,
                                datetime.timedelta(seconds=1, microseconds=605035),
                                7,
                                56,
                                35],
                               [2,
                                datetime.timedelta(seconds=1, microseconds=549466),
                                7,
                                56,
                                35],
                               [3,
                                datetime.timedelta(seconds=1, microseconds=516289),
                                7,
                                56,
                                35]],
                              1439987,
                              1439987],
                             2: [[[0,
                                datetime.timedelta(seconds=1, microseconds=580145),
                                7,
                                56,
                                35],
                               [1,
                                datetime.timedelta(seconds=1, microseconds=640644),
                                7,
                                56,
                                35],
                               [2,
                                datetime.timedelta(seconds=1, microseconds=537107),
                                7,
                                56,
                                35],
                               [3,
                                datetime.timedelta(seconds=1, microseconds=512776),
                                7,
                                56,
                                35]],
                              1439987,
                              1439987]},
 'like_b': {0: [[[0,
                                datetime.timedelta(seconds=46, microseconds=227704),
                                7,
                                56,
                                35],
                               [1,
                                datetime.timedelta(seconds=46, microseconds=398593),
                                7,
                                56,
                                35],
                               [2,
                                datetime.timedelta(seconds=48, microseconds=542814),
                                7,
                                56,
                                35],
                               [3,
                                datetime.timedelta(seconds=49, microseconds=854854),
                                7,
                                56,
                                35]],
                              1439987,
                              1439987],
                             1: [[[0,
                                datetime.timedelta(seconds=1, microseconds=596248),
                                7,
                                56,
                                35],
                               [1,
                                datetime.timedelta(seconds=1, microseconds=557935),
                                7,
                                56,
                                35],
                               [2,
                                datetime.timedelta(seconds=1, microseconds=602313),
                                7,
                                56,
                                35],
                               [3,
                                datetime.timedelta(seconds=1, microseconds=626484),
                                7,
                                56,
                                35]],
```

```
                    1439987,
                    1439987],
                2: [[[0,
                      datetime.timedelta(seconds=1, microseconds=594181),
                      7,
                      56,
                      35],
                     [1,
                      datetime.timedelta(seconds=1, microseconds=576170),
                      7,
                      56,
                      35],
                     [2,
                      datetime.timedelta(seconds=2, microseconds=305109),
                      7,
                      56,
                      35],
                     [3,
                      datetime.timedelta(seconds=1, microseconds=566581),
                      7,
                      56,
                      35]],
                    1439987,
                    1439987]},
 'like_c': {0: [[[0,
                  datetime.timedelta(seconds=51, microseconds=31678),
                  7,
                  56,
                  35],
                 [1,
                  datetime.timedelta(seconds=47, microseconds=104351),
                  7,
                  56,
                  35],
                 [2,
                  datetime.timedelta(seconds=47, microseconds=5895),
                  7,
                  56,
                  35],
                 [3,
                  datetime.timedelta(seconds=52, microseconds=106889),
                  7,
                  56,
                  35]],
                1439987,
                1439987],
            1: [[[0,
                  datetime.timedelta(microseconds=887638),
                  7,
                  56,
                  35],
                 [1,
                  datetime.timedelta(seconds=1, microseconds=20990),
                  7,
                  56,
                  35],
                 [2,
                  datetime.timedelta(microseconds=814183),
                  7,
                  56,
                  35],
                 [3,
                  datetime.timedelta(microseconds=874858),
                  7,
                  56,
                  35]],
                1439987,
                1439987],
            2: [[[0,
                  datetime.timedelta(microseconds=923204),
                  7,
                  56,
                  35],
                 [1,
                  datetime.timedelta(microseconds=924688),
                  7,
                  56,
                  35],
                 [2,
                  datetime.timedelta(microseconds=892344),
                  7,
                  56,
                  35],
                 [3,
                  datetime.timedelta(microseconds=909735),
                  7,
                  56,
                  35]],
                1439987,
                1439987]},
 'similar to': {0: [[[0,
                      datetime.timedelta(seconds=49, microseconds=881206),
                      7,
                      56,
                      35],
                     [1,
                      datetime.timedelta(seconds=92, microseconds=406387),
                      7,
                      56,
```

```
                                                 35],
                                                [2,
                                                 datetime.timedelta(seconds=46, microseconds=325607),
                                                 7,
                                                 56,
                                                 35],
                                                [3,
                                                 datetime.timedelta(seconds=46, microseconds=805514),
                                                 7,
                                                 56,
                                                 35]],
                                               1439987,
                                               1439987],
                                          1: [[[0,
                                                 datetime.timedelta(seconds=1, microseconds=543379),
                                                 7,
                                                 56,
                                                 35],
                                                [1,
                                                 datetime.timedelta(seconds=1, microseconds=624610),
                                                 7,
                                                 56,
                                                 35],
                                                [2,
                                                 datetime.timedelta(seconds=1, microseconds=553433),
                                                 7,
                                                 56,
                                                 35],
                                                [3,
                                                 datetime.timedelta(seconds=1, microseconds=534623),
                                                 7,
                                                 56,
                                                 35]],
                                               1439987,
                                               1439987],
                                          2: [[[0,
                                                 datetime.timedelta(seconds=1, microseconds=608424),
                                                 7,
                                                 56,
                                                 35],
                                                [1,
                                                 datetime.timedelta(seconds=1, microseconds=597847),
                                                 7,
                                                 56,
                                                 35],
                                                [2,
                                                 datetime.timedelta(seconds=1, microseconds=615707),
                                                 7,
                                                 56,
                                                 35],
                                                [3,
                                                 datetime.timedelta(seconds=1, microseconds=618182),
                                                 7,
                                                 56,
                                                 35]],
                                               1439987,
                                               1439987]}}}},
'cluster': {True: {'Clusters': [{'AllowVersionUpgrade': True,
                                 'AquaConfiguration': {'AquaConfigurationStatus': 'enabled',
                                                       'AquaStatus': 'enabled'},
                                 'AutomatedSnapshotRetentionPeriod': 1,
                                 'AvailabilityZone': 'us-east-1a',
                                 'AvailabilityZoneRelocationStatus': 'disabled',
                                 'ClusterAvailabilityStatus': 'Available',
                                 'ClusterCreateTime': datetime.datetime(2022, 0, 0, 0, 0, 0, 0, tzinfo=tzutc()),
                                 'ClusterIdentifier': 'nnn',
                                 'ClusterNamespaceArn': 'nnn',
                                 'ClusterNodes': [{'NodeRole': 'LEADER',
                                                   'PrivateIPAddress': 'nnn',
                                                   'PublicIPAddress': 'nnn'},
                                                  {'NodeRole': 'COMPUTE-0',
                                                   'PrivateIPAddress': 'nnn',
                                                   'PublicIPAddress': 'nnn'},
                                                  {'NodeRole': 'COMPUTE-1',
                                                   'PrivateIPAddress': 'nnn',
                                                   'PublicIPAddress': 'nnn'}],
                                 'ClusterParameterGroups': [{'ParameterApplyStatus': 'in-sync',
                                                             'ParameterGroupName': 'nnn'}],
                                 'ClusterPublicKey': 'ssh-rsa '
                                                     'nnn'
                                                     'Amazon-Redshift\n',
                                 'ClusterRevisionNumber': '41533',
                                 'ClusterSecurityGroups': [],
                                 'ClusterStatus': 'available',
                                 'ClusterSubnetGroupName': 'nnn',
                                 'ClusterVersion': '1.0',
                                 'DBName': 'dev',
                                 'DeferredMaintenanceWindows': [],
                                 'ElasticResizeNumberOfNodeOptions': '[4]',
                                 'Encrypted': False,
                                 'Endpoint': {'Address': 'nnn',
                                              'Port': 5439},
                                 'EnhancedVpcRouting': False,
                                 'IamRoles': [],
                                 'MaintenanceTrackName': 'current',
                                 'ManualSnapshotRetentionPeriod': 1,
                                 'MasterUsername': 'admin',
                                 'NextMaintenanceWindowStartTime': datetime.datetime(2022, 0, 0, 0, 0, tzinfo=tzutc()),
                                 'NodeType': 'ra3.xlplus',
```

```
                'NumberOfNodes': 2,
                'PendingModifiedValues': {},
                'PreferredMaintenanceWindow': 'nnn',
                'PubliclyAccessible': True,
                'Tags': [],
                'VpcId': 'nnn',
                'VpcSecurityGroups': [{'Status': 'active',
                                       'VpcSecurityGroupId': 'nnn'}]}],
 'ResponseMetadata': {'HTTPHeaders': {'content-length': '4273',
                                      'content-type': 'text/xml',
                                      'date': 'n, '
                                              'n '
                                              'n '
                                              'n '
                                              'GMT',
                                      'vary': 'accept-encoding',
                                      'x-amzn-requestid': 'nnn'},
                      'HTTPStatusCode': 200,
                      'RequestId': 'nnn',
                      'RetryAttempts': 0}}}}}}
```

Appendix B : Step Plans

These step plans are for the query `select count(*) from table_2 where column_2 like '%a%';`.

NO AQUA

qid	stream	segment	step	node_id	slice_id	step_type	rows	bytes	seg_start_time	seg_duration	table_name	notes
1006	0	0	0	0	0	scan	3263442	3367872144	09:10:39.483359	4.870199s	table_2	scan data from user table
1006	0	0	0	0	1	scan	3263442	3367872144	09:10:39.48336	4.873962s	table_2	scan data from user table
1006	0	0	0	1	2	scan	3263442	3367872144	09:10:39.482804	5.161395s	table_2	scan data from user table
1006	0	0	0	1	3	scan	3263442	3367872144	09:10:39.482863	5.149253s	table_2	scan data from user table
1006	0	0	1	0	0	project	3263442		09:10:39.483359	4.870199s		
1006	0	0	1	0	1	project	3263442		09:10:39.48336	4.873962s		
1006	0	0	1	1	2	project	3263442		09:10:39.482804	5.161395s		
1006	0	0	1	1	3	project	3263442		09:10:39.482863	5.149253s		
1006	0	0	2	0	0	project	3263442		09:10:39.483359	4.870199s		
1006	0	0	2	0	1	project	3263442		09:10:39.48336	4.873962s		
1006	0	0	2	1	2	project	3263442		09:10:39.482804	5.161395s		
1006	0	0	2	1	3	project	3263442		09:10:39.482863	5.149253s		
1006	0	0	3	0	0	aggregate	1	8	09:10:39.483359	4.870199s		ungrouped, scalar aggregation in memory
1006	0	0	3	0	1	aggregate	1	8	09:10:39.48336	4.873962s		ungrouped, scalar aggregation in memory
1006	0	0	3	1	2	aggregate	1	8	09:10:39.482804	5.161395s		ungrouped, scalar aggregation in memory
1006	0	0	3	1	3	aggregate	1	8	09:10:39.482863	5.149253s		ungrouped, scalar aggregation in memory
1006	1	1	0	0	0	scan	1	8	09:10:44.646207	0.000056s		scan data from temp table
1006	1	1	0	0	1	scan	1	8	09:10:44.646208	0.000081s		scan data from temp table
1006	1	1	0	1	2	scan	1	8	09:10:44.645471	0.000069s		scan data from temp table
1006	1	1	0	1	3	scan	1	8	09:10:44.645511	0.000065s		scan data from temp table
1006	1	1	1	0	0	return	1	8	09:10:44.646207	0.000056s		
1006	1	1	1	0	1	return	1	8	09:10:44.646208	0.000081s		
1006	1	1	1	1	2	return	1	8	09:10:44.645471	0.000069s		
1006	1	1	1	1	3	return	1	8	09:10:44.645511	0.000065s		
1006	1	2	0		12817	scan	4	32	09:10:44.655443	0.000761s		scan data from network to temp table
1006	1	2	1		12817	aggregate	1	16	09:10:44.655443	0.000761s		ungrouped, scalar aggregation in memory
1006	2	3	0		12817	scan	1	16	09:10:44.657433	0.000065s		scan data from temp table
1006	2	3	1		12817	project	1		09:10:44.657433	0.000065s		
1006	2	3	2		12817	project	1		09:10:44.657433	0.000065s		
1006	2	3	3		12817	return	1	14	09:10:44.657433	0.000065s		

(30 rows)

AQUA

qid	stream	segment	step	node_id	slice_id	step_type	rows	bytes	seg_start_time	seg_duration	table_name	notes
833	0	0	0	0	0	scan	7	56	08:54:21.685284	42.673772s	table_2	scan data from an AQUA scan of RSM
833	0	0	0	0	1	scan	7	56	08:54:21.685284	40.915049s	table_2	scan data from an AQUA scan of RSM
833	0	0	0	1	2	scan	7	56	08:54:21.682799	50.529760s	table_2	scan data from an AQUA scan of RSM
833	0	0	0	1	3	scan	7	56	08:54:21.682798	40.881619s	table_2	scan data from an AQUA scan of RSM
833	0	0	1	0	0	project	7		08:54:21.685284	42.673772s		
833	0	0	1	0	1	project	7		08:54:21.685284	40.915049s		
833	0	0	1	1	2	project	7		08:54:21.682799	50.529760s		
833	0	0	1	1	3	project	7		08:54:21.682798	40.881619s		
833	0	0	2	0	0	project	7		08:54:21.685284	42.673772s		
833	0	0	2	0	1	project	7		08:54:21.685284	40.915049s		
833	0	0	2	1	2	project	7		08:54:21.682799	50.529760s		
833	0	0	2	1	3	project	7		08:54:21.682798	40.881619s		
833	0	0	3	0	0	aggregate	1	16	08:54:21.685284	42.673772s		ungrouped, scalar aggregation in memory
833	0	0	3	0	1	aggregate	1	16	08:54:21.685284	40.915049s		ungrouped, scalar aggregation in memory
833	0	0	3	1	2	aggregate	1	16	08:54:21.682799	50.529760s		ungrouped, scalar aggregation in memory
833	0	0	3	1	3	aggregate	1	16	08:54:21.682798	40.881619s		ungrouped, scalar aggregation in memory
833	1	1	0	0	0	scan	1	16	08:55:12.2209	0.000172s		scan data from temp table
833	1	1	0	0	1	scan	1	16	08:55:12.2211	0.000160s		scan data from temp table
833	1	1	0	1	2	scan	1	16	08:55:12.219693	0.000190s		scan data from temp table
833	1	1	0	1	3	scan	1	16	08:55:12.21922	0.000094s		scan data from temp table
833	1	1	1	0	0	return	1	16	08:55:12.2209	0.000172s		
833	1	1	1	0	1	return	1	16	08:55:12.2211	0.000160s		

```
 833 |      1 |       1 |    1 |       1 |        2 | return    |    1 |    16 | 08:55:12.219693 |    0.000190s |           |
 833 |      1 |       1 |    1 |       1 |        3 | return    |    1 |    16 | 08:55:12.21922  |    0.000094s |           |
 833 |      1 |       2 |    0 |         |    12811 | scan      |    4 |    64 | 08:55:12.226857 |    0.003931s |           | scan data from network to temp table
 833 |      1 |       2 |    1 |         |    12811 | aggregate |    1 |    16 | 08:55:12.226857 |    0.003931s |           | ungrouped, scalar aggregation in memory
 833 |      2 |       3 |    0 |         |    12811 | scan      |    1 |    16 | 08:55:12.28113  |    0.000199s |           | scan data from temp table
 833 |      2 |       3 |    1 |         |    12811 | project   |    1 |       | 08:55:12.28113  |    0.000199s |           |
 833 |      2 |       3 |    2 |         |    12811 | project   |    1 |       | 08:55:12.28113  |    0.000199s |           |
 833 |      2 |       3 |    3 |         |    12811 | return    |    1 |    14 | 08:55:12.28113  |    0.000199s |           |
(30 rows)
```

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.