AWS REDSHIFT

AWS Redshift Cheat Sheet

10th May 2020 Omid

Basics…

get list of databases

```
select oid as database_id,
       datname as database_name,
       datallowconn as allow_connect
from pg_database
order by oid;
```

connect to db

```
SELECT * FROM PG_TABLE_DEF WHERE  schemaname ='public'
```

Cheat sheet for basic SQL operations on Redshift.

Create Schema

```
create  SCHEMA  test_schema
```

Create table

```
create table test_schema.users(
        userid integer not null distkey sortkey,
        username char(8),
        firstname varchar(30),
        lastname varchar(30),
        city varchar(30),
        state char(2),
```

```
        email varchar(100),
        phone char(14),
```

CTAS

```
create table event_backup as select * from event;
```

CTAS with distkey and sory key

```
create table myTable2
distkey (col1)
sortkey (col1,col3)
as
select *
from MyTable;
```

Insert into table from S3

```
COPY test_schema.users FROM 's3://ariel-s3-
buket/tickitdb/allusers_pipe.txt' iam_role
'arn:aws:iam::527228915290:role/RedshiftAccessS3' delimiter '|'
region 'us-east-1';
```

Case – If

```
select venue city,
case venuecity
when 'New York City'
then 'Big Apple' else 'other'
end
from venue
order by venueid desc;
```

Casting

```
select cast(pricepaid as integer)
from sales where salesid=100;
```

Redshit create table with emphasis on performance

some good reads…. before i summerize it for you.

https://docs.aws.amazon.com/redshift/latest/dg/r_CREATE_TABLE_NEW.html

https://docs.aws.amazon.com/redshift/latest/dg/r_CREATE_TABLE_examples.html

https://www.flydata.com/blog/amazon-redshift-distkey-and-sortkey/

Redshift Distribution Key – Choosing Best Distribution Style

use distsytle or distkey with sortkey – create table demystified

DISTSTYLE { AUTO | EVEN | KEY | ALL }

Keyword that defines the data distribution style for the whole table. Amazon Redshift distributes the rows of a table to the compute nodes according to the distribution style specified for the table. The default is AUTO. The distribution style that you select for tables affects the overall performance of your database.

```
create table venue(
venueid smallint not null,
venuename varchar(100),
venuecity varchar(30))
diststyle all;
```

DISTKEY (*column_name*)

Keyword that specifies that the column is the distribution key for the table. **Only one column in a table can be the distribution key**. You can use the DISTKEY keyword after a column name or as part of the table definition by using the DISTKEY (*column_name*) syntax. Either method has the same effect. For more information, see the DISTSTYLE parameter later in this topic. Notice this can cause skews in your cluster storage.

create example with distkey

```
create table sales(
salesid integer not null,
listid integer not null,
sellerid integer not null,
buyerid integer not null,
eventid integer not null encode mostly16,
dateid smallint not null,
qtysold smallint not null encode mostly8,
pricepaid decimal(8,2) encode delta32k,
commission decimal(8,2) encode delta32k,
saletime timestamp,
primary key(salesid),
foreign key(listid) references listing(listid),
foreign key(sellerid) references users(userid),
foreign key(buyerid) references users(userid),
foreign key(dateid) references date(dateid))
```

```
distkey(listid)
compound sortkey(listid,sellerid);
```

[{ COMPOUND | INTERLEAVED }] **SORTKEY** (*column_name* [,...])

Keyword that specifies that the column is the sort key for the table. When data is loaded into the table, the data is sorted by one or more columns that are designated as sort keys. You can use the SORTKEY keyword after a column name to specify a single-column sort key, or you can specify one or more columns as sort key columns for the table by using the SORTKEY (*column_name* [, ...]) syntax. Only compound sort keys are created with this syntax.

If you don't specify any sort keys, the table isn't sorted. You can define a maximum of 400 SORTKEY columns per table.

create example with compound sortkey

```
create table sales(
salesid integer not null,
listid integer not null,
sellerid integer not null,
buyerid integer not null,
eventid integer not null encode mostly16,
dateid smallint not null,
qtysold smallint not null encode mostly8,
pricepaid decimal(8,2) encode delta32k,
commission decimal(8,2) encode delta32k,
saletime timestamp,
primary key(salesid),
foreign key(listid) references listing(listid),
foreign key(sellerid) references users(userid),
foreign key(buyerid) references users(userid),
foreign key(dateid) references date(dateid))
distkey(listid)
compound sortkey(listid,sellerid);
```

create example with interleaved sortkey

```
create table customer_interleaved (
  c_custkey             integer        not null,
  c_name                varchar(25)    not null,
  c_address             varchar(25)    not null,
  c_city                varchar(10)    not null,
  c_nation              varchar(15)    not null,
  c_region              varchar(12)    not null,
  c_phone               varchar(15)    not null,
  c_mktsegment      varchar(10)    not null)
diststyle all
interleaved sortkey (c_custkey, c_city, c_mktsegment);
```

more create examples that will impact your performance…

Before: **simple example with sortkey and distkey**

```
create table activity (
  id integer primary key,
  created_at_date date sortkey distkey,
  device varchar(30)
);
```

After: **simple example with sortkey and distkey**

```
create table activity (
  id integer primary key,
  created_at_date distkey,
  device varchar(30)
)
sortkey (created_at_date, device);
```

How to view the dist-key and sort key in table in AWS Redshift?

```
select * from   SVV_TABLE_INFO
```

Redshift Date Manipuation

```
#assuming epoch time 13 digits
date_add('ms', myEpocTimeStamp,'1970-01-01')AS session_datetime,
 # 2020-01-01 03:17:17
trunc (date_add('ms', myEpocTimeStamp,'1970-01-01')) as session_date,
# 2020-01-01
```

Redshift specific syntax

Table information like sortkeys, unsorted percentage

```
SELECT * FROM svv_table_info;
```

Table sizes in GB

```
SELECT t.name, COUNT(tbl) / 1000.0 AS gb
FROM (
  SELECT DISTINCT datname, id, name
  FROM stv_tbl_perm
  JOIN pg_database ON pg_database.oid = db_id
) AS t
JOIN stv_blocklist ON tbl = t.id
GROUP BY t.name ORDER BY gb DESC;
```

Table column metadata

```
SELECT * FROM pg_table_def
WHERE schemaname = 'public'
AND tablename = ...;
```

Vacuum progress

```
SELECT * FROM svv_vacuum_progress;
```

Find tables that need vacuum or analyze

```
SELECT "database", "schema", "table", unsorted, stats_off
FROM svv_table_info
WHERE unsorted > 20
OR stats_off > 20
```

The size in MB of each column of each table (actually the number of blocks, but blocks are 1 MB)

```
SELECT
  TRIM(name) as table_name,
  TRIM(pg_attribute.attname) AS column_name,
  COUNT(1) AS size
FROM
  svv_diskusage JOIN pg_attribute ON
    svv_diskusage.col = pg_attribute.attnum-1 AND
    svv_diskusage.tbl = pg_attribute.attrelid
GROUP BY 1, 2
ORDER BY 1, 2;
```

List users and groups

```
SELECT * FROM pg_user;
SELECT * FROM pg_group;
```

list all databases

```
SELECT * FROM pg_database;
```

List the 100 last load errors

see http://docs.aws.amazon.com/redshift/latest/dg/r_STL_LOAD_ERRORS.html

```
SELECT *
FROM stl_load_errors
ORDER BY starttime DESC
LIMIT 100;
```

Get the full SQL, plus more query details from a query ID

```
WITH query_sql AS (
  SELECT
    query,
    LISTAGG(text) WITHIN GROUP (ORDER BY sequence) AS sql
  FROM stl_querytext
  GROUP BY 1
)
SELECT
  q.query,
  userid,
  xid,
  pid,
  starttime,
  endtime,
  DATEDIFF(milliseconds, starttime, endtime)/1000.0 AS duration,
  TRIM(database) AS database,
  (CASE aborted WHEN 1 THEN TRUE ELSE FALSE END) AS aborted,
  sql
FROM
  stl_query q JOIN query_sql qs ON (q.query = qs.query)
WHERE
  q.query = ...
ORDER BY starttime;
```

Show the most recently executed DDL statements

```
SELECT
  starttime,
  xid,
  LISTAGG(text) WITHIN GROUP (ORDER BY sequence) AS sql
FROM stl_ddltext
```

```
GROUP BY 1, 2
ORDER BY 1 DESC;
```

Query duration stats per database, user and query group; including the max, median, 99 percentile, etc.

```
WITH
durations1 AS (
  SELECT
    TRIM("database") AS db,
    TRIM(u.usename) AS "user",
    TRIM(label) AS query_group,
    DATE_TRUNC('day', starttime) AS day,
    -- total_queue_time/1000000.0 AS duration,
    -- total_exec_time/1000000.0 AS duration,
    (total_queue_time + total_exec_time)/1000000.0 AS duration
  FROM stl_query q, stl_wlm_query w, pg_user u
  WHERE q.query = w.query
    AND q.userid = u.usesysid
    AND aborted = 0
),
durations2 AS (
  SELECT
    db,
    "user",
    query_group,
    day,
    duration,
    PERCENTILE_CONT(0.50) WITHIN GROUP (ORDER BY duration) OVER
(PARTITION BY db, "user", query_group, day) AS median,
    PERCENTILE_CONT(0.75) WITHIN GROUP (ORDER BY duration) OVER
(PARTITION BY db, "user", query_group, day) AS p75,
    PERCENTILE_CONT(0.90) WITHIN GROUP (ORDER BY duration) OVER
(PARTITION BY db, "user", query_group, day) AS p90,
    PERCENTILE_CONT(0.95) WITHIN GROUP (ORDER BY duration) OVER
(PARTITION BY db, "user", query_group, day) AS p95,
    PERCENTILE_CONT(0.99) WITHIN GROUP (ORDER BY duration) OVER
(PARTITION BY db, "user", query_group, day) AS p99,
    PERCENTILE_CONT(0.999) WITHIN GROUP (ORDER BY duration) OVER
(PARTITION BY db, "user", query_group, day) AS p999
  FROM durations1
)
SELECT
  db,
  "user",
  query_group,
  day,
  MIN(duration) AS min,
  AVG(duration) AS avg,
  MAX(median) AS median,
```

```
  MAX(p75) AS p75,
  MAX(p90) AS p90,
  MAX(p95) AS p95,
  MAX(p99) AS p99,
  MAX(p999) AS p999,
  MAX(duration) AS max
FROM durations2
GROUP BY 1, 2, 3, 4
ORDER BY 1, 2, 3, 4;
```

Currently executing and recently executed queries with status, duration, database, etc.

```
SELECT
  r.pid,
  TRIM(status) AS status,
  TRIM(db_name) AS db,
  TRIM(user_name) AS "user",
  TRIM(label) AS query_group,
  r.starttime AS start_time,
  r.duration,
  r.query AS sql
FROM stv_recents r LEFT JOIN stv_inflight i ON r.pid = i.pid;
```

show remote host and port of running queries

```
SELECT
  recents.pid,
  TRIM(db_name) AS db,
  TRIM(user_name) AS "user",
  TRIM(label) AS query_group,
  recents.starttime AS start_time,
  recents.duration,
  recents.query AS sql,
  TRIM(remotehost) AS remote_host,
  TRIM(remoteport) AS remote_port
FROM stv_recents recents
LEFT JOIN stl_connection_log connections ON (recents.pid =
connections.pid)
LEFT JOIN stv_inflight inflight ON recents.pid = inflight.pid
WHERE TRIM(status) = 'Running'
AND event = 'initiating session';
```

Show user permissions

```
WITH
  users AS (
    SELECT usename AS user_name FROM pg_user
  ),
```

```
  objects AS (
    SELECT
      schemaname AS schema_name,
      'table' AS object_type,
      tablename AS object_name,
      schemaname + '.' + tablename AS full_object_name
    FROM pg_tables
    WHERE schemaname NOT IN ('pg_internal')
    UNION
    SELECT
      schemaname AS schema_name,
      'view' AS object_type,
      viewname AS object_name,
      schemaname + '.' + viewname AS full_object_name
    FROM pg_views
    WHERE schemaname NOT IN ('pg_internal')
  )
SELECT
  schema_name,
  object_name,
  object_type,
  user_name,
  HAS_TABLE_PRIVILEGE(users.user_name, full_object_name, 'select') AS
"select",
  HAS_TABLE_PRIVILEGE(users.user_name, full_object_name, 'insert') AS
"insert",
  HAS_TABLE_PRIVILEGE(users.user_name, full_object_name, 'update') AS
"update",
  HAS_TABLE_PRIVILEGE(users.user_name, full_object_name, 'delete') AS
"delete",
  HAS_TABLE_PRIVILEGE(users.user_name, full_object_name,
'references') AS "references"
FROM users, objects
ORDER BY full_object_name;
```

Credit for this blog goes to To:

Omid Vahdaty, ilan Rosen, Ariel Yoef

PREVIOUS POST
Comprehensive Python Training Program for Data Engineers in 200 KM/h

NEXT POST
Unable to connect to GCE instance via ssh after removing 0.0.0.0/0 rule from the FW.

Leave a Reply

Enter your comment here...

Search …