SQL SCRIPTS

AWS REDSHIFT: A COMPREHENSIVE GUIDE

Demo: Creating Cluster

```
-- Find total sales on a given calendar date.
SELECT sum(qtysold)
FROM   sales, date
WHERE  sales.dateid = date.dateid
AND    caldate = '2008-01-06';
```

```
-- Find top 10 buyers by quantity.
SELECT firstname, lastname, total_quantity
FROM   (SELECT buyerid, sum(qtysold) total_quantity
        FROM  sales
        GROUP BY buyerid
        ORDER BY total_quantity desc limit 10) Q, users
WHERE Q.buyerid = userid
ORDER BY Q.total_quantity desc;
```

Demo: Uploading External File

```
--create schema
create schema ext_file_demo
```

```
--create table: sales
create table ext_file_demo.sales(
      salesid integer not null,
      listid integer not null distkey,
      sellerid integer not null,
      buyerid integer not null,
      eventid integer not null,
      dateid smallint not null sortkey,
      qtysold smallint not null,
      pricepaid decimal(8,2),
      commission decimal(8,2),
      saletime timestamp);
```

```
--load data from external file
COPY ext_file_demo.sales from
's3://test-redshift-demo/demo-2/sales_tab.txt'
credentials
'aws_iam_role=arn:aws:iam::$account_id:role/test-redshift-role-s3'
delimiter '\t'
timeformat 'MM/DD/YYYY HH:MI:SS'
region 'us-west-2';
```

```
--validation query: get total count of sales records
SELECT count(*) from ext_file_demo.sales;
```

```
--validation query: Find total sales on a given calendar date
SELECT sum(qtysold)
FROM   ext_file_demo.sales s, date
WHERE  s.dateid = date.dateid
AND    caldate = '2008-01-05';
```

Demo: Accessing Redshift Externally

```
--create user
CREATE USER test_user_1 WITH PASSWORD 'test_USER_1' VALID UNTIL
'2022-01-01';
```

```
-- assign permissions to user
GRANT ALL PRIVILEGES ON ALL TABLES IN SCHEMA public TO test_user_1;
```

Demo: Loading & Unloading Data

```
--create schema
create schema loading_data_demo
```

```
--setting search path
SET search_path = loading_data_demo,public;
```

```
--create table: sales
create table sales(
      salesid integer not null,
      listid integer not null distkey,
      sellerid integer not null,
      buyerid integer not null,
      eventid integer not null,
      dateid smallint not null sortkey,
      qtysold smallint not null,
      pricepaid decimal(8,2),
      commission decimal(8,2),
      saletime timestamp);
```

```
--load data from s3
copy sales from 's3://test-redshift-demo/demo-4/sales_'
credentials
'aws_iam_role=arn:aws:iam::$account_id:role/test-redshift-role-s3'
delimiter '\t'
timeformat 'MM/DD/YYYY HH:MI:SS'
region 'us-west-2';
```

```
-- create table sales_update with source as sales

create table sales_update as
select * from sales;
```

```
-- change every fifth seller, so we have updates in sales_update table
update sales_update
set qtysold = qtysold*2,
pricepaid = pricepaid*0.8,
commission = commission*1.1
where saletime > '2008-11-30'
and mod(sellerid, 5) = 0;
```

```
-- Add some new rows so we have sample insert data. The query creates a
duplicate of every fourth seller id

insert into sales_update
select (salesid + 172456) as salesid, listid, sellerid, buyerid, eventid,
dateid, qtysold, pricepaid, commission, getdate() as saletime
from sales
where saletime > '2008-11-30'
and mod(sellerid, 4) = 0;
```

```
-- Create a staging table and populate it with updated rows from
SALES_UPDATE

create temp table stagesales as
select * from sales_update
where sales_update.saletime > '2008-11-30'
and sales_update.salesid = (select sales.salesid from sales
where sales.salesid = sales_update.salesid
and sales.listid = sales_update.listid
and (sales_update.qtysold != sales.qtysold
or sales_update.pricepaid != sales.pricepaid));
```

```
-- insert the new records into the stage table

insert into stagesales
select
su.salesid,su.listid,su.sellerid,su.buyerid,su.eventid,su.dateid,su.qtysold
,su.pricepaid,su.commission,su.saletime
from sales_update su left join sales on su.salesid = sales.salesid
where sales.salesid is null;
```

```
-- Start a new transaction
begin transaction;


delete from sales
using stagesales
where sales.salesid = stagesales.salesid
and sales.listid = stagesales.listid
and sales.saletime > '2008-11-30';

-- Insert all the rows from the staging table into the target table
insert into sales
select * from stagesales;


-- End transaction and commit
end transaction;
```

```
-- Drop the staging table
drop table stagesales;
```

```
-- validating results

select count(*) from sales

select count(*) from sales_update

select count(*) from sales_update
where sales_update.saletime > '2008-11-30'
and sales_update.salesid = (select sales.salesid from sales
where sales.salesid = sales_update.salesid
and sales.listid = sales_update.listid
and (sales_update.qtysold != sales.qtysold
or sales_update.pricepaid != sales.pricepaid));
```

```
-- unload
unload ('select * from sales')
```

```
to 's3://test-redshift-demo/demo-4/unload/sales_'
iam_role 'arn:aws:iam::$account_id:role/test-redshift-role-s3';
```

Demo: Materialized View

```
-- create schema
create schema mat_view_demo
```

```
-- set search path
SET search_path = mat_view_demo,public;
```

```
-- sales table contains eventids
-- event table contains eventname
-- query to get the event(eventname) and the total sales associated with it

SELECT e.eventname eventname,
 sum(s.pricepaid) total_sales
FROM event e, sales s
WHERE e.eventid = s.eventid
GROUP BY e.eventname
order by e.eventname
```

```
-- create materialized view
CREATE MATERIALIZED VIEW tickets_mv
AS (SELECT e.eventname eventname,
 sum(s.pricepaid) total_sales
FROM event e, sales s
WHERE e.eventid = s.eventid
GROUP BY e.eventname)
```

```
-- check the view data
select * from mat_view_demo.tickets_mv order by eventname
```

```
-- update the results
update sales s set pricepaid= pricepaid*2 where eventid in (select eventid
from event where eventname = '.38 Special')
```

```
-- refresh the view data
refresh materialized view mat_view_demo.tickets_mv;
```

```
-- auto refresh
CREATE MATERIALIZED VIEW tickets_mv_2 AUTO REFRESH YES
AS (SELECT e.eventname eventname,
 sum(s.pricepaid) total_sales
FROM event e, sales s
WHERE e.eventid = s.eventid
GROUP BY e.eventname)
```

Demo: Table Design - Distribution Style

```
-- create schema
create schema table_design_demo
```

```
-- setting search path
SET search_path = table_design_demo,public;
```

```
-- checking the sortkey for users
select "column", type, distkey
from pg_table_def where tablename = 'users' and schemaname='public';

select * from svv_table_info s where s.schema='public' and s.table ='users'
```

```
---checking the disk usage, distribution
select slice, col, num_values as rows, minvalue, maxvalue
from svv_diskusage
where name='users' and tbl = (select table_id from svv_table_info s where
s.schema='public' and s.table ='users')
and col=0
and num_values>0
order by slice, col;
```

```
-- state as the dist key
create table userskey distkey(state) as select * from users;

-- checking the disk usage with state as distkey
select slice, col, num_values as rows, minvalue, maxvalue
from svv_diskusage
where name='userskey' and tbl = (select table_id from svv_table_info s
where s.schema='table_design_demo' and s.table ='userskey')
and col=0
and num_values>0
```

```
-- even example
create table userseven diststyle even as
select * from users;

select slice, col, num_values as rows, minvalue, maxvalue
from svv_diskusage
where name='userseven' and tbl = (select table_id from svv_table_info s
where s.schema='table_design_demo' and s.table ='userseven')
and col=0
and num_values>0
```

```
-- all example
create table usersall diststyle all as
select * from users;

select slice, col, num_values as rows, minvalue, maxvalue
from svv_diskusage
where name='usersall' and tbl = (select table_id from svv_table_info s
where s.schema='table_design_demo' and s.table ='usersall')
and col=0
and num_values>0
```

```
-- auto example
create table usersauto as
select * from users;

select slice, col, num_values as rows, minvalue, maxvalue
from svv_diskusage
where name='usersauto' and tbl = (select table_id from svv_table_info s
where s.schema='table_design_demo' and s.table ='usersauto')
and col=0
and num_values>0

select * from svv_table_info s where s.schema='table_design_demo' and
s.table ='usersauto'
```

Demo: Table Design - Sort Key

```
-- check sortkey
select "column", type, sortkey
from pg_table_def where tablename = 'usersauto';
```

```
-- update the sort key
create table users_custom_sort sortkey (firstname, lastname) as
select * from users;
```

```
-- check sortkey
select "column", type, sortkey
from pg_table_def where tablename = 'users_custom_sort';
```

```
-- explain functions
explain SELECT firstname, lastname, total_quantity
FROM   (SELECT buyerid, sum(qtysold) total_quantity
        FROM  sales
        GROUP BY buyerid
        ORDER BY total_quantity desc limit 10) Q,
table_design_demo.users_custom_sort U
WHERE Q.buyerid = U.userid
ORDER BY Q.total_quantity desc;
```

Demo: Table Design - Compression

```
-- compression
select "column", type, encoding
from pg_table_def where tablename = 'usersauto' and
schemaname='table_design_demo';
```

```
-- Analyse Command
ANALYSE COMPRESSION usersauto;
```

```
-- create user table with custom compression
create table users_custom_compression(
      userid integer not null distkey sortkey,
      username char(8) encode zstd,
      firstname varchar(30) encode zstd,
      lastname varchar(30) encode zstd,
      city varchar(30),
      state char(2),
      email varchar(100),
      phone char(14),
      likesports boolean,
      liketheatre boolean,
      likeconcerts boolean,
      likejazz boolean,
      likeclassical boolean,
      likeopera boolean,
      likerock boolean,
      likevegas boolean,
      likebroadway boolean,
      likemusicals boolean);
```

```
-- inserting records from users
insert into users_custom_compression
select * from users
```

```
-- checking compression style
select "column", type, encoding
from pg_table_def where tablename = 'users_custom_compression' and
```

```
schemaname='table_design_demo';
```

```
-- analyse command
ANALYSE COMPRESSION users_custom_compression;
```

Demo: Redshift ML

```
-- create schema
create schema redshift_ml_demo

-- setting search context
SET search_path = redshift_ml_demo,public;
```

```
-- create table: customer_activity
CREATE TABLE customer_activity (
state varchar(2),
account_length int,
area_code int,
phone varchar(8),
intl_plan varchar(3),
vMail_plan varchar(3),
vMail_message int,
day_mins float,
day_calls int,
day_charge float,
total_charge float,
eve_mins float,
eve_calls int,
eve_charge float,
night_mins float,
night_calls int,
night_charge float,
intl_mins float,
intl_calls int,
intl_charge float,
cust_serv_calls int,
churn varchar(6),
record_date date);
```

```
-- load data
COPY customer_activity
FROM 's3://test-redshift-demo/demo-15/input/customer_activity.csv'
REGION 'us-west-2' IAM_ROLE 'arn:aws:iam::$account:role/test-redshift-ml'
DELIMITER ','
```

```
IGNOREHEADER 1;
```

```
-- create ML model
CREATE MODEL customer_churn_auto_model FROM (SELECT state,
             account_length,
             area_code,
             total_charge/account_length AS average_daily_spend,
             cust_serv_calls/account_length AS average_daily_cases,
             churn
      FROM customer_activity
      WHERE  record_date < '2020-01-01'
     )
TARGET churn FUNCTION ml_fn_customer_churn_auto
IAM_ROLE 'arn:aws:iam::480935361548:role/test-redshift-ml'SETTINGS (
  S3_BUCKET 'test-redshift-demo'
);
```

```
-- view model details
select schema_name, model_name, model_state from stv_ml_model_info;
```

```
-- use ML model to predict active customers
SELECT phone,
       ml_fn_customer_churn_auto(
          state,
          account_length,
          area_code,
          total_charge/account_length,
          cust_serv_calls/account_length )
          AS active FROM customer_activity WHERE record_date >
'2020-01-01';
```

```
-- using ML model: finding churners, non-churners state wise
WITH inferred AS (SELECT state,
```

```
       ml_fn_customer_churn_auto(
          state,
          account_length,
          area_code,
          total_charge/account_length,
          cust_serv_calls/account_length )::varchar(6)
          AS active FROM customer_activity
          WHERE record_date > '2020-01-01' )
SELECT state, SUM(CASE WHEN active = 'True.' THEN 1 ELSE 0 END) AS
churners,
       SUM(CASE WHEN active = 'False.' THEN 1 ELSE 0 END) AS nonchurners,
       COUNT(*) AS total_per_state
FROM inferred
GROUP BY state
ORDER BY state;
```