Advanced SQL Concepts

Improve your SQL skills to take full advantage of Query

This document assumes you are already familiar with the topics in the "Introduction to SQL" help document.

For more in-depth, technical coverage of the topics introduced in this section, please refer to the official Redshift documentation, which provides explanation and examples for all supported commands and functions.

public.people

The examples in this document use a hypothetical table, '`public.people`'. Assume this table contains one row per person and contains the following data about each individual:

Column Name	Column Description	Data Type	Example Value
`id`	Unique person identifier	integer	`12345`
`first_name`	Person's first name	varchar	`FOZZIE`
`last_name`	Person's last name	varchar	`BEAR`
`email`	Person's email address	varchar	`wockawocka@muppets.com`
`state`	Person's state of residence	char(2)	`CA`
`signup_date`	Date the person first signed up for the organization's newsletter	date	`2008-11-04`
`is_active`	Contains 1 if the person is currently subscribed any newsletter, 0 if not	integer	`1`
`months_subscribed`	How many months the person has been subscribed	integer	`80`

© 2016 Civis Analytics, Inc., All Rights Reserved

Column Types

Every column in Redshift is assigned a data type. These are the supported types, as listed in the Redshift documentation:

Data Type	Aliases	Description
`smallint`	`int2`	Signed two-byte integer
`integer`	`int, int4`	Signed four-byte integer
`bigint`	`int8`	Signed eight-byte integer
`decimal(a,b)`	`numeric`	Exact numeric of selectable precision (a digits before decimal, b digits after)
`real`	`float4`	Single precision floating-point number
`double`	`float8, float`	Double precision floating-point number
`boolean`	`bool`	Logical Boolean (true/false)
`char(n)`	`character`	Fixed-length (n) character string
`varchar(n)`	`character varying, text`	Variable-length character string with a user-defined limit (n)
`date`		Calendar date (year, month, day)
`timestamp`	`timestamp without timezone`	Date and time (without time zone)

Working with dates

Storing dates with the date column type enables a host of useful functions.

System date and time

You can retrieve the current date without a timestamp using **current_date**, or the current date with a timestamp (in GMT) using **getdate()**. To find all rows corresponding to users who signed up before today , for example:

```
select *
from public.people
where signup_date < current_date
```

© 2016 Civis Analytics, Inc., All Rights Reserved

Date parts

To extract a date part, use **extract**. This is useful for aggregating data by month or year, for example. The following query provides a breakdown of signups per month since October 2014:

```
select extract(year from signup_date) as signup_year,
    extract(month from signup_date) as signup_month,
    count(*) as num_signups
from public.people
where signup_date >= '2014-10-01'
group by 1,2
order by 1,2
```

Date interval functions

You can use the **dateadd** function to determine the number of days, weeks, months, quarters, etc. since a given date. In the following query, we count how many users signed up in the past three months:

```
select count(*)
from public.people
where signup_date <= dateadd(month, 3, current_date)
```

To get dates within a given interval—before or after—you can use **datediff**. The following query returns the average tenure, in days, for current newsletter subscribers:

```
select avg(datediff(day, signup_date, current_date)) as avg_tenure
from public.people
where signup_date < current_date
```

TO_DATE

If your raw dates are saved in column of type varchar instead of date, you can use the **to_date** function, specifying the raw date column and the raw date format string, to convert them to the default date style (‘`YYYY-MM-DD`’) and enable date functionality. Assume in the following example you have a varchar column called ‘`end_date`’ with dates such as ‘`1/12/15`’:

```
select *
from schema.table
where to_date(end_date, 'MM/DD/YY') > '2014-01-01'
```

© 2016 Civis Analytics, Inc., All Rights Reserved

The preceding query returns records with an end_date after January 1, 2014.

The to_date function can deal with a variety of raw date formats. The format string for a column with a string date, such as 'January 12, 2015', for example, is 'Month DD, YYYY'. A full datetime format strings reference is available in the official Redshift documentation.

Getting Data from Multiple Tables (JOIN)

Your data may be spread across more than one table. You can use **join** to connect two or more tables based on columns they have in common.

Assume we have a table '`public.subscriptions`' with the following layout:

Column Name	Column Description	Data Type	Example Value
`subscription_id`	Unique subscription identifier	integer	`52758`
`person_id`	Person identifier (same as '`id`' column in `public.people`)	varchar	`12345`
`supporter_list_flag`	Contains 1 if the person is subscribed to the supporter email list, 0 if not	integer	`1`
`volunteer_list_flag`	Contains 1 if the person is subscribed to the volunteer email list, 0 if not	integer	`1`
`donor_list_flag`	Contains 1 if the person is subscribed to the donor email list, 0 if not	integer	`0`

Since public.subscriptions has a '`person_id`' column that corresponds to the '`id`' column in public.people, we can use a join to link the two tables.

We can then select columns and/or aggregates from either table. In this case, we get a count of active users by state and a count of how many people in each state are subscribed to the donor list. Since identically-named columns may appear in both tables, you must specify which table's version of the column to use by preceding the column name with the table name and a period:

```
select state, count(distinct people.id) as num_people,
    sum(subscriptions.donor_list_flag) as num_donors
from public.people
join public.subscriptions
    on subscriptions.person_id = people.id
where people.is_active = 1
group by 1
order by 1;
```

© 2016 Civis Analytics, Inc., All Rights Reserved

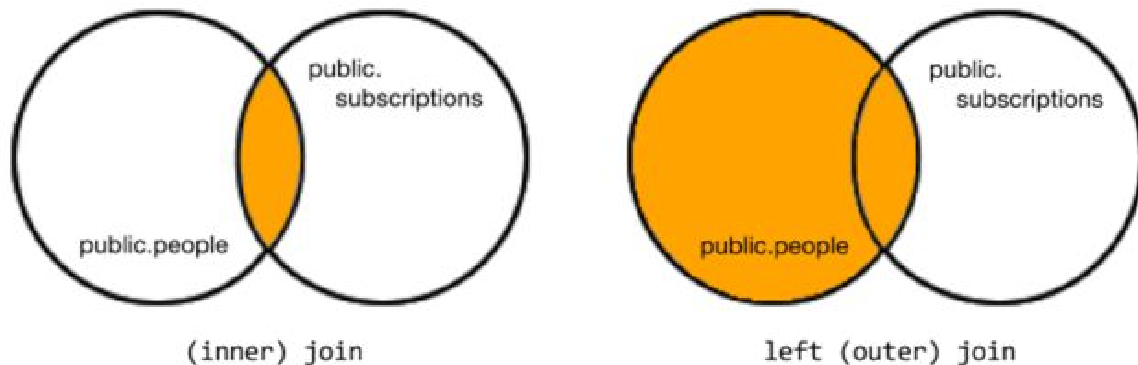
Join types

When using **join**—also referred to as **inner join**—you will only consider records for which the join key is present in both tables. In other words, if person is not subscribed to any email lists, and therefore has no corresponding records in public.subscriptions, they are not included in the 'num_people' count in the above query.

To select all rows regardless of whether they have corresponding records in the joined table, you can use a **left join**—also referred to as a **left outer join**:

```
select state, count(distinct people.id) as num_people,
    sum(subscriptions.donor_list_flag) as num_donors
from public.people
left join public.subscriptions
    on subscriptions.person_id = people.id
where people.is_active = 1
group by 1
order by 1;
```

The diagram below illustrates the difference between these two join types:



© 2016 Civis Analytics, Inc., All Rights Reserved

CASE Statements

You can use **case** to organize data using if-then-else logic, defining one or more logical condition in each **when** clause. This query buckets months_subscribed into three groups, assuming all values are valid:

```
select case when months_subscribed < 6 then 'Less than 6 months'
            when score_band between 6 and 12 then '6-12 months'
            when score_band > 12 then 'More than 1 year'
            else 'Invalid value'
            end as months_subscribed_bucket,
    count(*)
from public.people
group by 1;
```

NTILE and subqueries

You can use the **ntile** function to group data in *n* buckets, such as quartiles and deciles.

This query buckets months_subscribed into four equally-sized groups and returns the average subscription length for each group. Note that it uses a **subquery**. You can treat any query like a table, and therefore select from and/or join to it, by wrapping it in parentheses:

```
select subscription_quartile, avg(months_subscribed), count(*)
from (
    select id, months_subscribed, ntile(4) over(order by months_subscribed asc) as
subscription_quartile
    from public.people
    where months_subscribed is not null
    ) person_by_quartile
group by 1
order by 1;
```

© 2016 Civis Analytics, Inc., All Rights Reserved