PERISCOPE DATA PRESENTS

A SQL-Based Approach to Cohort Retention and Analysis

Why Retention

Some brands are great at connecting with new audiences. They might have a killer marketing campaign, demonstrably fulfill a gap in the marketplace, or just have that certain "it factor." Being able to get new customers on board can lead to fantastic short-term financial gains.

However, none of that matters if you can't keep your audience. If people download your app and never open it, sign up for your newsletter and never go back to your site, or make one purchase in your store and never return, all of your marketing efforts were for naught.

To sustain a successful business, it's critical to understand what's working or not working with your marketing efforts, your content, and your product(s). The ability to observe, monitor, and fix where a customer cohort begins to lose interest is tantamount to continued brand loyalty, revenue growth, and year-over-year success.

The Value of Cohorts

Cohort analysis segments users into groups based on time, and then examines them against metrics such as retention and churn. Performing cohort analysis using SQL is a technique many skilled analysts use to understand differences between groups of customers. In this example we'll evaluate established and new customers to observe when product interest drops for either group, changes which may be associated with seasonal shifts, demographics, marketing, or any other number of factors.

Maintaining and analyzing cohorts can improve your marketing, refine your user experience, and ensure your products and services are meeting the needs of your most valuable audiences.

In this guide, we'll outline step-by-step methods to script your own cohort analysis in SQL with examples and interpretations of the resulting data.

How Do We Define Retention?

The way a company decides to define retention varies by business model. For example:

- A mobile app company may measure app engagement – such as daily active users (DAU)—when gauging retention.
- A subscription-based service may measure months-to-churn.
- A retailer may measure repeat club member purchases over a fiscal year.

For our example, let's pretend we're a mobile gaming company that looks at mobile app engagement—DAU, in our case—as a central key performance indicator (KPI). We have two groups of people who play the game in different, but identifiable ways:

User One plays the game on Monday and Tuesday. They're a **retained user**.

- User Two played the game Monday, but not on Tuesday. They're considered a **lapsed user**.
- Retention for Monday is the number of retained users divided by the number of total users. If User One was the only user on Tuesday, then retention for Tuesday is 50%. How do we take this even further with SQL?

Calculating Basic User Retention

The key to calculating retention is counting users who were active at one time period, then counting how many were active at another. An easy way to do this in SQL is to left join your user activity table to itself.

ID	USER ID	PLATFORM	DATE
41	16	web	2016-06-27 00:00:00

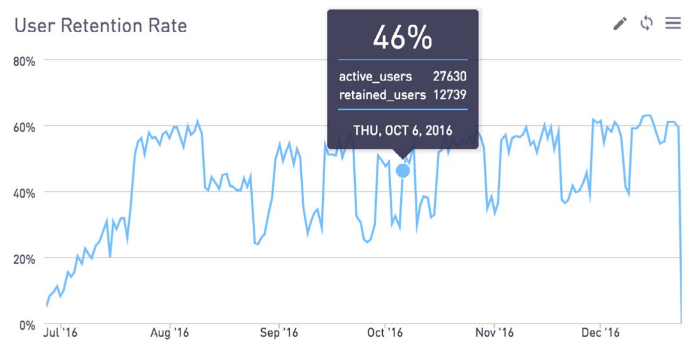
We can then do our self-join like so:

```
select *
from activity
left join activity as future_activity
  on activity.user_id = future_activity.user_id
  and activity.date = future_activity.date - interval
'1 day'
```

Now, for every row of user activity, we have—in that same row—their activity one day in the future. This gives us an ideal table for calculating retention with some simple counts:

```
select
  gameplays.date
  , count(distinct gameplays.user_id) as active_users
  , count(distinct future_gameplays.user_id)
      as retained_users
  , count(distinct future_gameplays.user_id) /
      count(distinct gameplays.user_id)::float
      as retention
from gameplays
left join gameplays as future_activity
  on gameplays.user_id = future_activity.user_id
  and gameplays.date = future_activity.date - interval
  '1 day'
group by 1
```

Entering this into our Periscope Data SQL Editor will give us this chart:



You can also change the retention interval to seven days or 30 days to get a sense of longer-term user engagement.

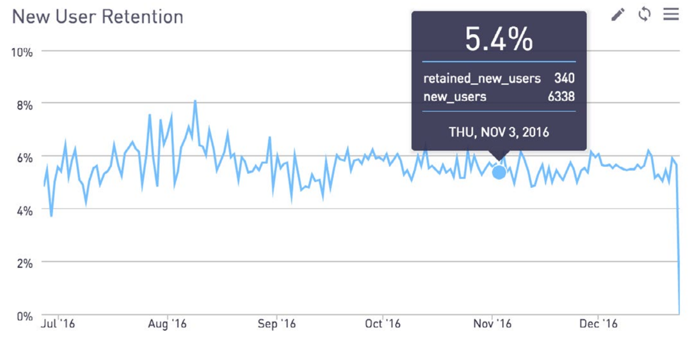
Calculating Retention of New Versus Existing Users

Often, retention is quite different for users who just signed up as compared to loyal, long-time users. To calculate new-user retention, simply join in your users table and only look at activity rows that occurred on the user's join date.

```
select
  users.date as date
  , count(distinct gameplays.user_id) as new_users,
  , count(distinct future_activity.user_id) as ,
  retained_users,
  , count(distinct future_activity.user_id) /
      count(distinct gameplays.user_id)::float
      as retention
from gameplays

-- Limits gameplays to activity from new users
join users on
  gameplays.user_id = users.id
  and users.date = gameplays.date

left join gameplays as future_activity
  on gameplays.user_id = future_activity.user_id
  and gameplays.date = future_activity.date
- interval '1 day'
group by 1
```



Now we can see that while overall retention is 46%, new-user retention is only 5.4%! This is why it's helpful to split out user segments—we now understand that improving new-user retention should clearly be a priority. In the case of our mobile game, it could mean that either our features aren't appealing enough to keep new users engaged, or that our acquisition efforts are targeting the wrong type of user.

Now let's check out returning-user retention. To calculate this, simply change:

```
users.date = gameplays.date
to:
users.date != gameplays.date
```

This effectively excludes activity from users who joined that day. Our query now looks like:

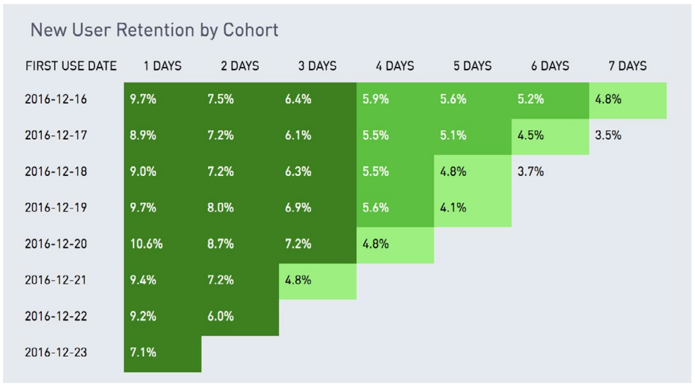
```
select
  gameplays.date as date
  , count(distinct gameplays.user_id) as new_users
  , count(distinct future_activity.user_id) as
  retained_users
  , count(distinct future_activity.user_id) /
      count(distinct gameplays.user_id)::float
      as retention
from gameplays

join users on
  gameplays.user_id = users.id
  and users.date != gameplays.date
left join gameplays as future_activity on
  gameplays.user_id = future_activity.user_id and
  gameplays.date = future_activity.date - interval '1
  day'
group by 1
```

As expected, returning-user retention is higher than the overall average: 68% versus 46%.

Calculating Retention in Cohorts

It can be illuminating to compare the retention of users who joined on **Day A** with those who joined on **Day B**. Or comparing the retention of users who joined as a result of different acquisition channels. This data gives us helpful insight into the impact that product or marketing changes have on improving retention. Ideally we'd end up with a chart that shows the diminishing number of returning users in a cohort over time, like this:


New User Retention by Cohort

FIRST USE DATE	1 DAYS	2 DAYS	3 DAYS	4 DAYS	5 DAYS	6 DAYS	7 DAYS
2016-12-16	9.7%	7.5%	6.4%	5.9%	5.6%	5.2%	4.8%
2016-12-17	8.9%	7.2%	6.1%	5.5%	5.1%	4.5%	3.5%
2016-12-18	9.0%	7.2%	6.3%	5.5%	4.8%	3.7%	
2016-12-19	9.7%	8.0%	6.9%	5.6%	4.1%		
2016-12-20	10.6%	8.7%	7.2%	4.8%			
2016-12-21	9.4%	7.2%	4.8%				
2016-12-22	9.2%	6.0%					
2016-12-23	7.1%						

We'll start by defining a few handy subqueries to simplify the problem. `New_user_activity` restricts user activity to—you guessed it—new users, then identifies their date of first use and the length of their tenure.

```
with new_user_activity as (
  select
    user_id
    , min(date(created_at)) as first_use_date
    , max(date(created_at)) - min(date(created_at)) as
      tenure
  from gameplays
     group by 1)
```

We'll then group this table by first use date and tenure.

```
, new_user_tenure as (
  select
    first_use_date
    , tenure
    , count(distinct user_id) as users
  from new_user_activity
  group by 1, 2)
```

`cohort_active_user_count` calculates the total number of active users—the denominator in our retention calculation—in each daily cohort.

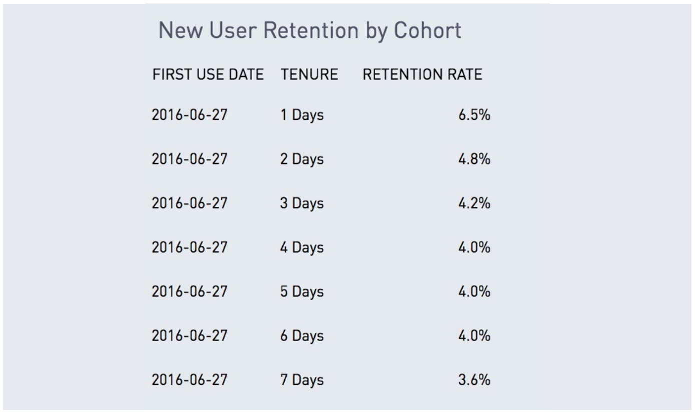
```
, cohort_active_user_count as (
select
 first_use_date
 , sum(users) as first_users
from new_user_activity
group by 1)
```

On top of that, we'll use some additional techniques:

1. In this query, we lose the simple count of active users in the cohort. Fortunately we thought of this and made our cohort_ active_user_count subquery, which we can join in and use as the denominator.
2. We will use a sequence table called all_numbers as the leftmost table. This will ensure populated data fields for all tenure lengths.
3. Notice also the range, or inequality, join. This is one of our favorite SQL tricks, and enables us to get multiple days of retention in one chart.
4. Let's take a look at the full query.

```
select
  new_user_tenure.first_use_date
  , all_numbers.tenure || ' Days' as tenure
  , sum(new_user_tenure.users) /
      cohort_active_user_count.first_users::float
      as retention_rate
  from
    all_numbers
    left join new_user_tenure
      on all_numbers.tenure <= new_user_tenure.tenure
    left join cohort_active_user_count
      on new_user_tenure.first_use_date = cohor_active_
         user_count.first_use_date
  where all_numbers.tenure != 0 and all_numbers.tenure
<= 7
  group by 1, 2, cohort_active_user_count.first_users
  order by 1,2
```

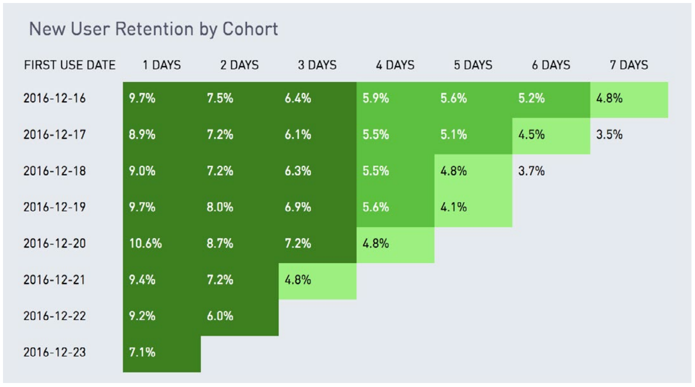
This results in the table:



New User Retention by Cohort

FIRST USE DATE	TENURE	RETENTION RATE
2016-06-27	1 Days	6.5%
2016-06-27	2 Days	4.8%
2016-06-27	3 Days	4.2%
2016-06-27	4 Days	4.0%
2016-06-27	5 Days	4.0%
2016-06-27	6 Days	4.0%
2016-06-27	7 Days	3.6%

Using Periscope Data, we can automatically pivot the result and then color by percentile.



New User Retention by Cohort

FIRST USE DATE	1 DAYS	2 DAYS	3 DAYS	4 DAYS	5 DAYS	6 DAYS	7 DAYS
2016-12-16	9.7%	7.5%	6.4%	5.9%	5.6%	5.2%	4.8%
2016-12-17	8.9%	7.2%	6.1%	5.5%	5.1%	4.5%	3.5%
2016-12-18	9.0%	7.2%	6.3%	5.5%	4.8%	3.7%	
2016-12-19	9.7%	8.0%	6.9%	5.6%	4.1%		
2016-12-20	10.6%	8.7%	7.2%	4.8%			
2016-12-21	9.4%	7.2%	4.8%				
2016-12-22	9.2%	6.0%					
2016-12-23	7.1%						

Interpreting Your Cohort Analysis

For cohort analysis, we're looking for significant differences between groups going down the chart to observe retention by start day. For example, we see an uptick in retention on 12/20 but a drop on 12/23. This would indicate to the analyst that they should investigate what occurred on those dates that might have contributed to a drop in retention.

Improving Your Customer Retention Analysis With SQL

Using the techniques outlined above, you can calculate your retention metrics and make better decisions for your business' future. These comparative metrics can mean the difference between high-yield feature development and declining user numbers following disinterest and churn.

While many of the functions outlined can be performed using a myriad of open source and spreadsheet software, the processes are very manual. Periscope Data offers an end-to-end analytics platform that reduces the time spent in clumsy workflows and produces immediate, up-to-date results.

Spend less time writing queries, and more time discovering and acting on business insights. Sign up for a free trial of Periscope Data.

About Periscope Data

Periscope Data is the analytics system of record for professional data teams. The platform enables data leaders, analysts and engineers to unlock the transformative power of data using SQL and surface actionable business insights in seconds—not hours or days. Periscope Data gives them control over the full data analysis lifecycle, from data ingestion, storage and management through analysis, visualization and sharing. The company serves nearly 900 customers including Adobe, New Relic, EY, ZipRecruiter, Tinder and Flexport, with analysts spending the equivalent of two business days per week in the platform to support data-driven decision-making. Periscope Data is headquartered in San Francisco, CA. For more information, visit www.periscopedata.com.