Stop Losing Customers!

A SQL-based approach to cohort retention

Overview

There's a lot to be said for creating a brand that connects with and engages new customers. A company's offering should have ability to demonstrate the ability to improve a customer's quality of life right from the jump. Proving inherent value upon first impression increases brand interest and awareness for newcomers and makes for a great fiscal quarter.

However, if your product or service can only entice new users into a one-time purchase, never to return, you're probably headed towards leaner times in the future. The ability to observe, monitor and fix where a customer cohort begins to lose interest is tantamount to continued brand loyalty and year over year success.

If you have a SQL database, you can use SQL to perform cohort analysis on this data to understand customer retention and loyalty. This document offers a step-by-step approach to writing the script and interpreting the results.

Ignore old customers at your own peril

Businesses should use every analytic tool available to understand the importance of retaining old customers versus acquiring new ones. Aggressive client prospecting helps drive short term profits, however the business may still suffer financially when older clients feel forgotten and churn out.

Cohort analysis using SQL is a popular toolset many skilled analysts use to understand any differences between groups of customers. In our case, an analyst would be able to monitor older and new customers to observe when product interest drops for either group which may be associated with seasonal shifts, local demographic changes or any number of factors.

For this document, we will outline step-by-step methods to script your own cohort analysis in SQL with examples and interpretations of the resulting data.

Defining customer retention

The way a company decides to define retention varies by business model, for example:

1. A mobile gaming company may measure app engagement as a retained user,
2. A subscription based service may measure months-to-churn, or
3. A retailer may measure repeat club member purchases over a fiscal year.

For our example, we will create a fake mobile gaming company that uses mobile app engagement as a central key performance indicator (KPI). We will also create two groups of people who play the game in different, but identifiable ways:

- User 1 plays the game on Monday and Tuesday and is considered a retained customer.
- User 2 played the game on Monday but not on Tuesday and is considered a lapsed user.

Retention for Monday is the number of retained users divided by the number of total users. If User 1 and User 2 were the only two users on Monday, then retention for Monday is 50%. This only tells us what the retention was between two users on a single day. How do we take this even farther with SQL?

Calculating basic user retention

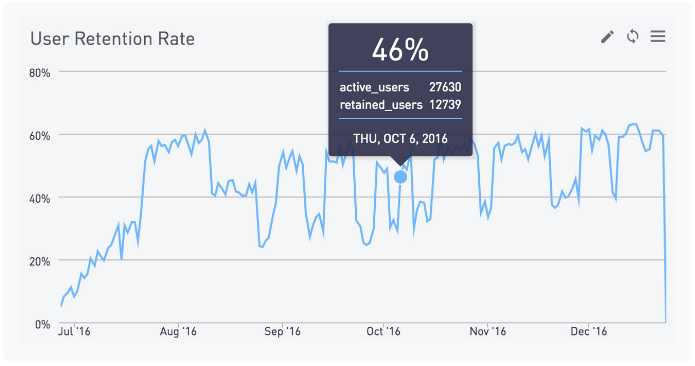
The key to calculating retention is counting users who were active at time #1, then counting how many were active at time #2. An easy way to do this in SQL is to left join your user activity table to itself like so:

```
select *
from gameplays
left join gameplays as future_gameplays on
   gameplays.user_id = future_gameplays.user_id
   and gameplays.date = future_gameplays.date -
   interval ‘1 day’
```

Now, for every row of user activity, we have—in that same row—their activity 1 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
```

In our Periscope SQL Editor we get this chart:



For extra credit, change the 1-day retention to 7-day or 30-day to capture a sense of longer-term user engagement.

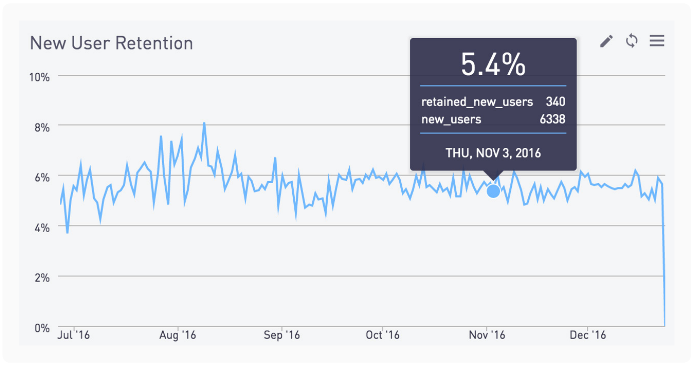
Calculating retention of new vs. existing users

Often retention is quite different for users who just signed up as compared to loyal long-term 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
```



We see that while overall retention is 46%, new user retention is only 5.4%! Now we see why it's so useful to split out new users. Improving new-user retention should clearly be a priority.

To look at returning-user retention, 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:

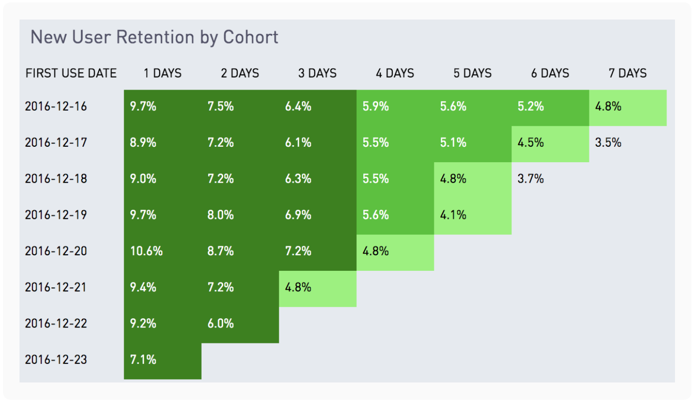
```
select
 gameplays.date as date,
 count(distinct gameplays.user_id) as new_users,
 count(distinct future_activity.user_id) as
retained_users,
 count(distinct future_activity.user_id) /
  count(distinct gameplays.user_id)::float as
retention
from gameplays

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, existing-user retention is higher than the overall average: 68% vs. 46%.

Calculating retention in cohorts

It can be very helpful to compare the retention of users who joined on day A with those who joined on day B. This lets us see if our product changes are improving our retention rate. Ideally we'd end up with a chart that shows the diminishing number of returning customers within 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 new users, then identifies their first use date 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 will then group this table by our user's 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:

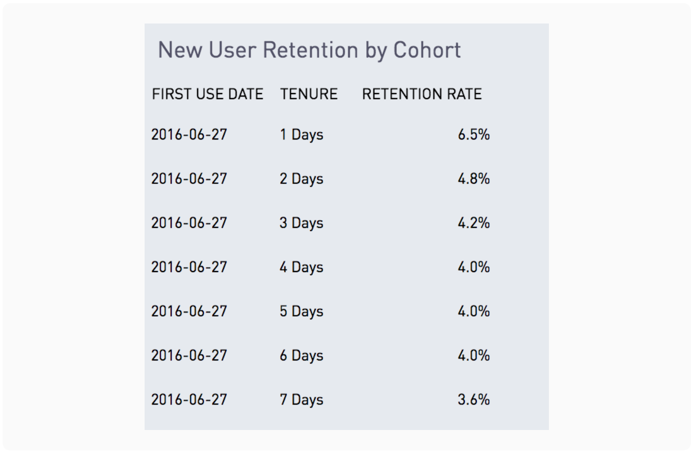
```
, cohort_active_user_count as (
select
 first_use_date
 , sum(users) as first_users
from new_user_activity
group by 1)
```

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, to get multiple days of retention in one chart.

Without further ado:

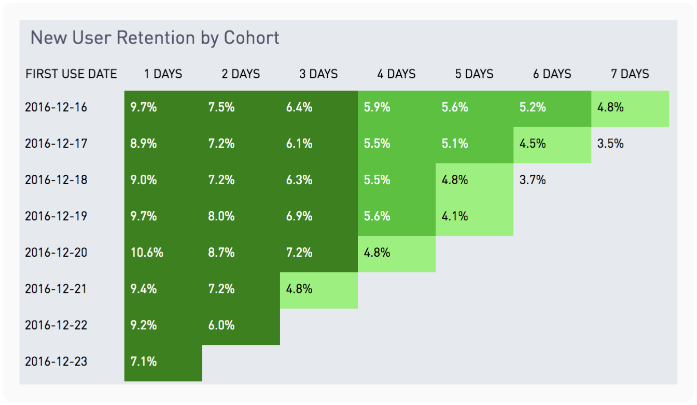
```
select
   new_user_tenure.first_use_date
   , all_numbers.tenure || ' Days' as tenure
   , sum(new_user_tenure.users) / cohort_active_
   user_count.first_users::float as retention_rate
   from
      all_numbers
      left join new_user_tenure on
         all_numbers.tenure <= new_user_tenure.
         tenure
      left join cohort_active_user_count on
            new_user_tenure.first_use_date = cohort_
            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
```



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%

With Periscope, we can automatically pivot the result 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%						

Interpretation of cohort analysis

For cohort analysis, we are 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 which might have contributed to a drop in retention.

Customer retention with SQL for better analyses

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, Periscope Data has created an analytics platform designed to reduce the time spent in clumsy workflows and produce immediate, up-to-date results.

Try a free trial of Periscope and see just how quickly you can find results using your own data with our SQL editor that features multi-SQL language support, searchable table histories, revision history, and autocomplete to help you go from query to answer in seconds.