

Churn Rates with Codeflix

Learn SQL from scratch

Patrick Fitzgerald

09.18.2018

Table of Contents

1. Getting Familiar with the company

- How many months has the company been operating and how many months do we have enough information to calculate churn for?
- What segment of users exist?

2. What is the overall churn trend since the company started?

3. Compare the churn rates between user segments

- Which segment of users should the company focus on expanding?

1. Getting familiar with the company

1.1 First we take a look at the data, how many different segments do we see?

For this I just did a simple “Select (*) all columns”

From the “subscriptions” table

And I limited my results to 100 rows

From this we can see there are two user segments

```
test.sqlite ↗  
  
1 SELECT *  
2 FROM subscriptions  
3 LIMIT 100;  
4  
5
```

Query Results			
id	subscription_start	subscription_end	segment
1	2016-12-01	2017-02-01	87
2	2016-12-01	2017-01-24	87
3	2016-12-01	2017-03-07	87
4	2016-12-01	2017-02-12	87
5	2016-12-01	2017-03-09	87
6	2016-12-01	2017-01-19	87
7	2016-12-01	2017-02-03	87
8	2016-12-01	2017-03-02	87
9	2016-12-01	2017-02-17	87
10	2016-12-01	2017-01-01	87
11	2016-12-01	2017-01-17	87
12	2016-12-01	2017-02-07	87
13	2016-12-01	∅	30
14	2016-12-01	2017-03-07	30
15	2016-12-01	2017-02-22	30
16	2016-12-01	∅	30
17	2016-12-01	∅	30
18	2016-12-02	2017-01-29	87
19	2016-12-02	2017-01-13	87
20	2016-12-02	2017-01-15	87
21	2016-12-02	2017-01-15	87
22	2016-12-02	2017-01-24	87
23	2016-12-02	2017-01-14	87
24	2016-12-02	2017-01-19	87

1.2 Next we will determine the range of months of data provided. How many months will we be able to calculate churn for?

Here we just did a simple “SELECT MIN” and “SELECT MAX” of the subscription_start column to see that the data starts on 2016-12-01 and the last subscription_start ends on 2017-03-30. We can see that the company have been in business for 4 months.

Since all subscriptions must last for at least a month this means we will have no subscription_end in December, which means we will be able to calculate churn for 3 months, Jan, Feb and March.

```
test.sqlite

1  SELECT MIN(subscription_start),
2      MAX(subscription_start)
3  FROM subscriptions;
4
5
```

Query Results	
MIN(subscription_start)	MAX(subscription_start)
2016-12-01	2017-03-30
Database Schema	
subscriptions 2000 rows	
id	INTEGER
subscription_start	TEXT
subscription_end	TEXT
segment	INTEGER

2. What is the overall churn trend since the company started?

2.1 First Step, we create a temporary table of months

So our first step here is just to create a table of the beginning and ending days of each month so we have something to use as a marker in our case statements later.

Query Results	
first_day	last_day
2017-01-01	2017-01-31
2017-02-01	2017-02-28
2017-03-01	2017-03-31

```
1  WITH months AS(  
2  SELECT  
3    '2017-01-01' AS first_day,  
4    '2017-01-31' AS last_day  
5  UNION  
6  SELECT  
7    '2017-02-01' AS first_day,  
8    '2017-02-28' AS last_day  
9  UNION  
10 SELECT  
11    '2017-03-01' AS first_day,  
12    '2017-03-31' AS last_day  
13 )  
14 SELECT *  
15 FROM months;
```

2.2 Create a cross join table from table and months

In this step we create another temporary table which merges the subscriptions table and the newly created months table. We use a cross join which basically merges every row with one table with every row of the other table. As you can see in the query results below we now have a first_day and last_day for each month for every id.

Query Results					
id	subscription_start	subscription_end	segment	first_day	last_day
1	2016-12-01	2017-02-01	87	2017-01-01	2017-01-31
1	2016-12-01	2017-02-01	87	2017-02-01	2017-02-28
1	2016-12-01	2017-02-01	87	2017-03-01	2017-03-31
2	2016-12-01	2017-01-24	87	2017-01-01	2017-01-31
2	2016-12-01	2017-01-24	87	2017-02-01	2017-02-28
2	2016-12-01	2017-01-24	87	2017-03-01	2017-03-31
3	2016-12-01	2017-03-07	87	2017-01-01	2017-01-31
3	2016-12-01	2017-03-07	87	2017-02-01	2017-02-28

```
test.sqlite
1  WITH months AS
2  (SELECT
3    '2017-01-01' AS first_day,
4    '2017-01-31' AS last_day
5  UNION
6  SELECT
7    '2017-02-01' AS first_day,
8    '2017-02-28' AS last_day
9  UNION
10 SELECT
11    '2017-03-01' AS first_day,
12    '2017-03-31' AS last_day
13 ),
14 cross_join AS
15 (SELECT *
16  FROM subscriptions
17  CROSS JOIN months)
18 SELECT *
19  FROM cross_join
20  LIMIT 100;
21
```


2.3 Creating a status table

Here we start our first step in creating our status table. We use a CASE statement to create two new columns. `is_active_87` gives us a 1 if the user was active at the start of that month and they are part of segment 87 and a 0 otherwise. Similarly, `is_active_30` gives us a 1 if the user was active at the start of that month and they are part of segment 30 and a 0 otherwise.

Query Results			
id	month	is_active_87	is_active_30
1	2017-01-01	1	0
1	2017-02-01	1	0
1	2017-03-01	0	0
2	2017-01-01	1	0
2	2017-02-01	0	0
2	2017-03-01	0	0
3	2017-01-01	1	0
3	2017-02-01	1	0
3	2017-03-01	1	0
4	2017-01-01	1	0
4	2017-02-01	1	0
4	2017-03-01	0	0

```
18 status AS
19 (SELECT id, first_day AS month,
20 CASE
21     WHEN (subscription_start < first_day)
22         AND (segment = 87)
23         AND (
24             subscription_end >= first_day
25             OR subscription_end IS NULL
26         ) THEN 1
27     ELSE 0
28 END as is_active_87,
29 CASE
30     WHEN (subscription_start < first_day)
31         AND (segment = 30)
32         AND (
33             subscription_end >= first_day
34             OR subscription_end IS NULL
35         ) THEN 1
36     ELSE 0
37 END as is_active_30
38 FROM cross_join)
39 SELECT *
40 FROM status
41 LIMIT 100;
```

2.4 Adding the is_canceled columns

In this segment we use additional CASE statements to create additional columns to create values for users of each segment that have canceled within that month.

Query Results					
id	month	is_active_87	is_active_30	is_canceled_87	is_canceled_30
1	2017-01-01	1	0	0	0
1	2017-02-01	1	0	1	0
1	2017-03-01	0	0	0	0
2	2017-01-01	1	0	1	0
2	2017-02-01	0	0	0	0
2	2017-03-01	0	0	0	0
3	2017-01-01	1	0	0	0
3	2017-02-01	1	0	0	0
3	2017-03-01	1	0	1	0
4	2017-01-01	1	0	0	0
4	2017-02-01	1	0	1	0
4	2017-03-01	0	0	0	0

```
38 CASE
39     WHEN subscription_end
40         BETWEEN first_day AND last_day
41         AND segment = 87
42         THEN 1
43     ELSE 0
44 END as is_canceled_87,
45 CASE
46     WHEN subscription_end
47         BETWEEN first_day AND last_day
48         AND segment = 30
49         THEN 1
50     ELSE 0
51 END as is_canceled_30
52 FROM cross_join)
```

2.5 Create a status_aggregate table

Finally we create another temporary table that gives us the sum of all the 1's in each of the columns. We create the status_aggregate table by using the SUM command on each of the columns from our status table, then group those sums by month.

```
51  END as is_canceled_30
52  FROM cross_join),
53  status_aggregate AS
54  (SELECT month,
55       SUM(is_active_87) as sum_active_87,
56       SUM(is_canceled_87) as sum_canceled_87,
57       SUM(is_active_30) as sum_active_30,
58       SUM(is_canceled_30) as sum_canceled_30
59  FROM status
60  GROUP BY month)
61  SELECT *
62  FROM status_aggregate;
```

Query Results				
month	sum_active_87	sum_canceled_87	sum_active_30	sum_canceled_30
2017-01-01	279	70	291	22
2017-02-01	467	148	518	38
2017-03-01	541	258	718	84

3. Compare the churn rate between user segments

3.1 Churn comparisons between user segments

Here we see the final results:

	User Segment 87	User Segment 30
January	25%	7.5%
February	31.6%	7.3%
March	47.6%	11.6%

Query Results		
month	churn_rate_87	churn_rate_30
2017-01-01	0.25089605734767	0.0756013745704467
2017-02-01	0.316916488222698	0.0733590733590734
2017-03-01	0.476894639556377	0.116991643454039

```
61 SELECT
62     month,
63     1.0 * sum_canceled_87/sum_active_87 AS churn_rate_87,
64     1.0 * sum_canceled_30/sum_active_30 AS churn_rate_30
65 FROM status_aggregate;
```

Conclusion: Here we can see that the churn rate for user segment 87 is much higher, showing that up to 50 percent of users have canceled during March . The company should clearly focus on expanding the growth of user segment 30.

Bonus

How would we modify this code to support a large number of segments?

Here we add the segment column to our status table. Then we add it to the status_aggregate table. Then just add it to the final table and group by segment as well as month.

Query Results		
month	segment	churn_rate
2017-01-01	30	0.0756013745704467
2017-01-01	87	0.25089605734767
2017-02-01	30	0.0733590733590734
2017-02-01	87	0.316916488222698
2017-03-01	30	0.116991643454039
2017-03-01	87	0.476894639556377

```
19 (SELECT id, segment, first_day as month,
20 CASE
21     WHEN (subscription_start < first_day)
22         AND (
23             subscription_end >= first_day
24             OR subscription_end IS NULL
25         ) THEN 1
26     ELSE 0
27 END as is_active,
28 CASE
29     WHEN subscription_end BETWEEN first_day AND last_day THEN 1
30     ELSE 0
31 END as is_canceled
32 FROM cross_join),
33 status_aggregate AS
34 (SELECT month, segment,
35     SUM(is_active) as active,
36     SUM(is_canceled) as canceled
37 FROM status
38 GROUP BY month, segment)
39 SELECT
40     month, segment,
41     1.0 * canceled/active AS churn_rate
42 FROM status_aggregate;
```