



# Codeflix

Churn Rate: Q1 2017

# Table of Contents

1. Get familiar with Codeflix
2. What is the overall churn rate by month?
3. Compare the churn rates between segments

# 1. Getting Familiar with the Data

We've been given the schema for this database, but it's a good idea to take a look at the table we'll be working with.

- You can begin by selecting all columns in the table, then limiting the search to the first 100 rows to return. You may even need to limit to 42 or so, depending on other factors.
- In this table, the id column appears to be the primary key
- Subscriptions start as early as December 1, 2016, and some continue past the available range of data. It doesn't look like any users canceled in December.
- There appear to be two different segments, 30 & 87, but you may want to run another query to confirm (this table has 2000 rows!!)

```
SELECT *
FROM subscriptions
LIMIT 42;

SELECT DISTINCT segments
FROM subscriptions;

SELECT subscription_end
FROM subscriptions
WHERE subscription_end IS NOT NULL
ORDER BY 1
LIMIT 1;
```

Query Results	
subscription_end	
2017-01-01	

Query Results	
segment	
87	
30	

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	2017-01-18	87
25	2016-12-02	2017-02-24	87
26	2016-12-02	2017-01-18	87
27	2016-12-02	2017-01-11	87
28	2016-12-02	2017-03-30	30
29	2016-12-02	2017-02-11	30
30	2016-12-02	2017-01-20	30
31	2016-12-02	∅	30
32	2016-12-02	2017-01-11	30
33	2016-12-02	∅	30
34	2016-12-02	2017-02-06	30
35	2016-12-03	2017-02-17	87
36	2016-12-03	2017-03-06	87
37	2016-12-03	2017-03-08	87
38	2016-12-03	2017-02-28	87
39	2016-12-03	∅	30
40	2016-12-03	∅	30
41	2016-12-03	∅	30
42	2016-12-03	2017-03-29	30

## 2.1 Setting the Stage

The database we are working with doesn't have any markers for when months begin or end... Sort of an important factor to determine monthly churn

- We'll start by creating another table that defines 3 one-month periods.
- We won't ever see this new table, but it would look something like the one below

Query Results

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

```
WITH months AS (  
  SELECT  
    '2017-01-01' AS first_day,  
    '2017-01-31' AS last_day  
  UNION  
  SELECT  
    '2017-02-01' AS first_day,  
    '2017-02-28' AS last_day  
  UNION  
  SELECT  
    '2017-03-01' AS first_day,  
    '2017-03-31' AS last_day  
) ,
```

## 2.2 Lights

Now we are going to join these two tables together to create a third table we can reference to start assessing user status by month

- The new table 'cross\_join' will start with all the columns from the subscriptions table,
- We can join the months table to the right of the subscriptions table to complete our new table

```
cross_join AS (  
  SELECT *  
  FROM subscriptions  
  CROSS JOIN months  
) ,
```

Query Results

id	subscription_start	subscription_end	first_day	last_day
1	2017-03-18	∅	2017-01-01	2017-01-31
1	2017-03-18	∅	2017-02-01	2017-02-28
1	2017-03-18	∅	2017-03-01	2017-03-31
2	2017-03-05	∅	2017-01-01	2017-01-31
2	2017-03-05	∅	2017-02-01	2017-02-28

## 2.3 Camera

Things are really starting to get interesting here. We're making columns that didn't exist before!

- The new table 'status' will generate the data to calculate how many users joined/canceled each month,
- This new table harkens back to the beginnings of computing...simple 1's and 0's

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	0	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


```
status AS (  
  SELECT  
    id,  
    segment,  
    first_day AS month,  
    CASE  
      WHEN (segment = 87)  
        AND  
(subscription_start < first_day)  
        AND (  
          subscription_end > first_day  
          OR subscription_end IS NULL  
        ) THEN 1  
      ELSE 0  
    END AS is_active_87,  
    CASE  
      WHEN (segment = 30)  
        AND  
(subscription_start < first_day)  
        AND (  
          subscription_end > first_day  
          OR subscription_end IS NULL  
        ) THEN 1  
      ELSE 0  
    END AS is_active_30,  
    CASE  
      WHEN (segment = 87) AND (subscription_end BETWEEN first_day AND last_day)  
        THEN 1  
      ELSE 0  
    END AS is_canceled_87,  
    CASE  
      WHEN (segment = 30) AND (subscription_end BETWEEN first_day AND last_day) THEN  
1  
      ELSE 0  
    END AS is_canceled_30  
  FROM cross_join  
)
```

## 2.4 Action

And execute! Look at how this new table takes all the data from the subscriptions table and aggregates it in much more digestible size

- For each month we can see how many users joined and how many canceled
- We can also break it down by user segment.
- Look at how many fewer users from segment 30 canceled!

```
status_aggregate AS (  
  SELECT  
    month,  
    SUM(is_active_87) AS sum_active_87,  
      SUM(is_active_30) AS sum_active_30,  
    SUM(is_canceled_87) AS sum_canceled_87,  
      SUM(is_canceled_30) AS sum_canceled_30  
  FROM status  
  GROUP BY 1
```



Query Results				
month	sum_active_87	sum_active_30	sum_canceled_87	sum_canceled_30
2017-01-01	278	291	70	22
2017-02-01	462	518	148	38
2017-03-01	531	716	258	84

### 3. Editing

My favorite part of the process, and one too often overlooked.

- We can compare the data from our previous table against each other to analyze the rate of change (or churn) for different user segments over time
- Canceled/active = churn
- You can choose how precise you want the result to be, here we round to 3 decimal places
- This is the only table that will actually show up given the calculation we've written
- **Based on these data, we should really think about expanding user segment 30**

Query Results		
month	churn_rate_87	churn_rate_30
2017-01-01	0.252	0.076
2017-02-01	0.32	0.073
2017-03-01	0.486	0.117

```
SELECT
  month,
  ROUND(1.0 * sum_canceled_87 / sum_active_87, 3) AS
churn_rate_87,
  ROUND(1.0 * sum_canceled_30 / sum_active_30, 3) AS
churn_rate_30
FROM status_aggregate;
```