codecademy

Capstone: Churn Rates

Learn SQL from Scratch

Natalia Morawska

Table of Contents

1. Get familiar with Codeflix
2. Calculate churn rate for each segment
3. Plan for a higher number of segments

Get familiar with Codeflix

Getting familiar with the data

Capstone started collecting data in December 2016, and has a total set until March 2017.

There are two different segments of customers – 87 and 30.

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

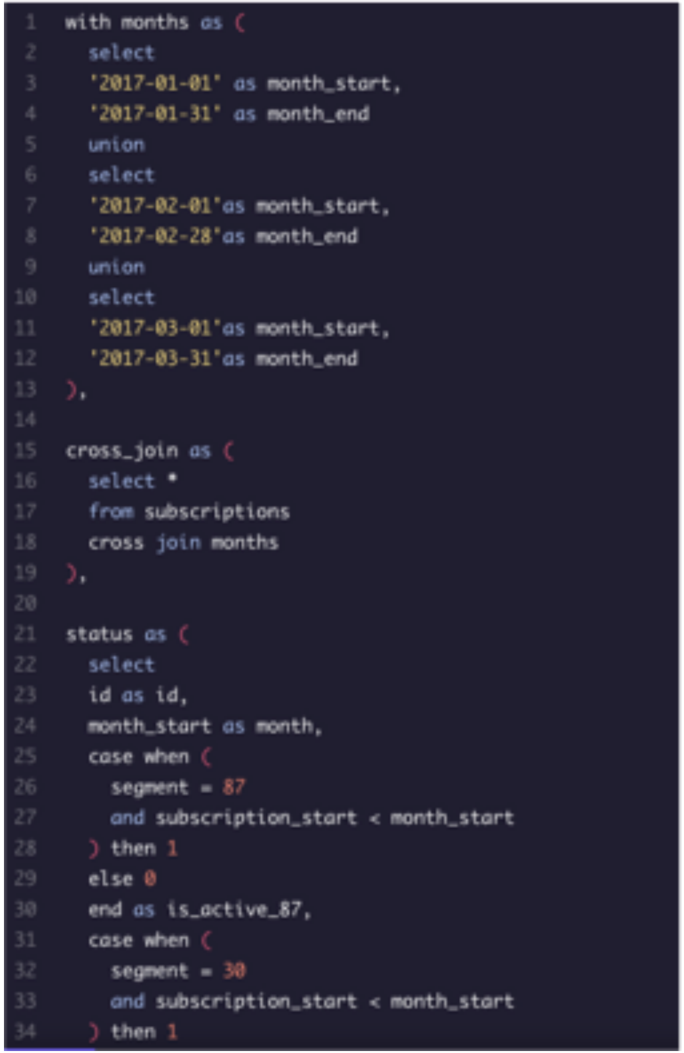
```
SELECT *
FROM subscriptions
LIMIT 100;

----------------------


select min(subscription_start) as start_date,
max(subscription_start) as end_date
from subscriptions;
```

Calculate churn rate for each segment

Temporary tables

In order to analyze the churn rate, three temporary tables were created:

- Months – with start and end dates of Jan-March 2017
- Cross_join – join of the table above and the subscriptions table
- Status – with information if the subscription was active and/or canceled in the specific month



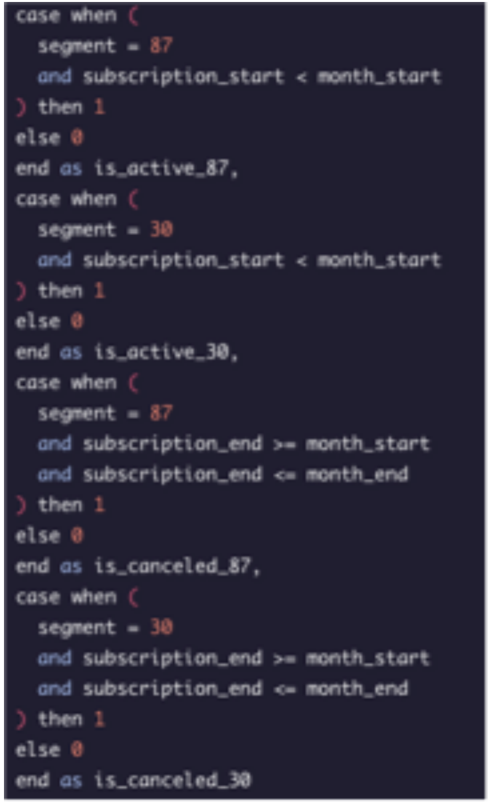
```
with months as (
  select
  '2017-01-01' as month_start,
  '2017-01-31' as month_end
  union
  select
  '2017-02-01'as month_start,
  '2017-02-28'as month_end
  union
  select
  '2017-03-01'as month_start,
  '2017-03-31'as month_end
),

cross_join as (
  select *
  from subscriptions
  cross join months
),

status as (
  select
  id as id,
  month_start as month,
  case when (
    segment = 87
    and subscription_start < month_start
  ) then 1
  else 0
  end as is_active_87,
  case when (
    segment = 30
    and subscription_start < month_start
  ) then 1
```

Is_active and Is_canceled

Two variables were created in order to calculate the churn:

- is_active – to determine if the user was active in the specific mont
- is_canceled – to determine if the user canceled in the specific month

The result is 1 or 0 (1– yes, 0 – no).



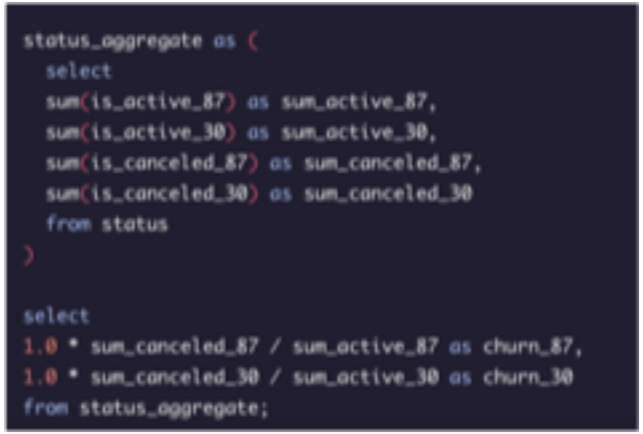
```
case when (
  segment = 87
  and subscription_start < month_start
) then 1
else 0
end as is_active_87,
case when (
  segment = 30
  and subscription_start < month_start
) then 1
else 0
end as is_active_30,
case when (
  segment = 87
  and subscription_end >= month_start
  and subscription_end <= month_end
) then 1
else 0
end as is_canceled_87,
case when (
  segment = 30
  and subscription_end >= month_start
  and subscription_end <= month_end
) then 1
else 0
end as is_canceled_30
```

Totals

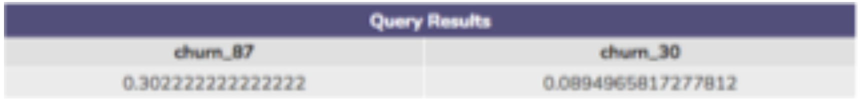
As the next step, we sum the total number of active and canceled users for each segment, and calculate the churn rates.

The results are presented below.

It is clear that the segment 87 has a much higher churn rate (30%) than segment 30 (9%), and so the efforts should be focused on lowering the churn rate in segment 87.



```
status_aggregate as (
  select
  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
)

select
1.0 * sum_canceled_87 / sum_active_87 as churn_87,
1.0 * sum_canceled_30 / sum_active_30 as churn_30
from status_aggregate;
```



Query Results

churn_87	churn_30
0.302222222222222	0.0894965817277812

Modifying the code to support large number of segments

Large number of segments

In order to support a large number of segments, I would make a temporary table listing all the segments and cross-join it with our table.

Later, I would proceed to creating the status table, and I would group by the ‘segment’ column in order to obtain a result for each segment.

My outcome would be a table like this:

segment	churn
30	0.0894965817277
87	0.3022222222222
92	0.1692547763689

Please find the code.sql file enclosed with all the steps of my analysis.

The last part is the final code.