Codecademy
Learn SQL from Scratch

Capstone: Churn Rates

Reid Smiegiel I 2018.07.01

Table of Contents

1. Get Familiar with Codeflix

- **How many months has the company been operating?**

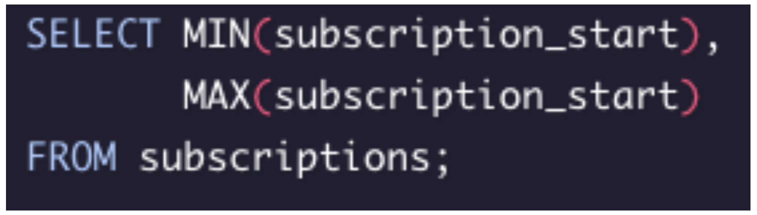
- Codeflix has been operating for 4 months: December 2016 - March 2017.

- **Which months do you have enough information to calculate a churn rate?**

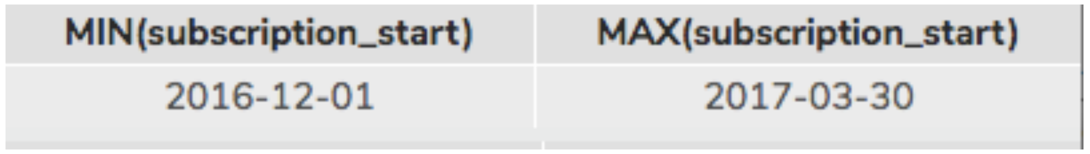
- Churn rate is the percent of subscribers who cancel their subscriptions within a given period of time. In this case, we are calculating churn on a monthly basis.

- There is enough data to calculate the churn rate for 3 months: January, February, March.

- There is no churn rate for December 2016 because subscribers are not able to terminate their subscriptions during that first month.



```
SELECT MIN(subscription_start),
       MAX(subscription_start)
FROM subscriptions;
```

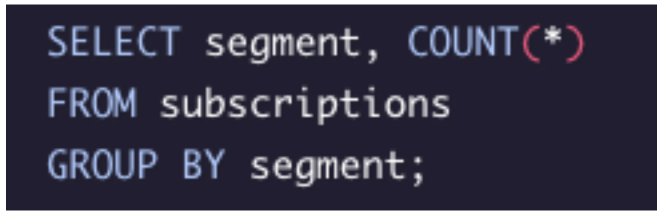


MIN(subscription_start)	MAX(subscription_start)
2016-12-01	2017-03-30

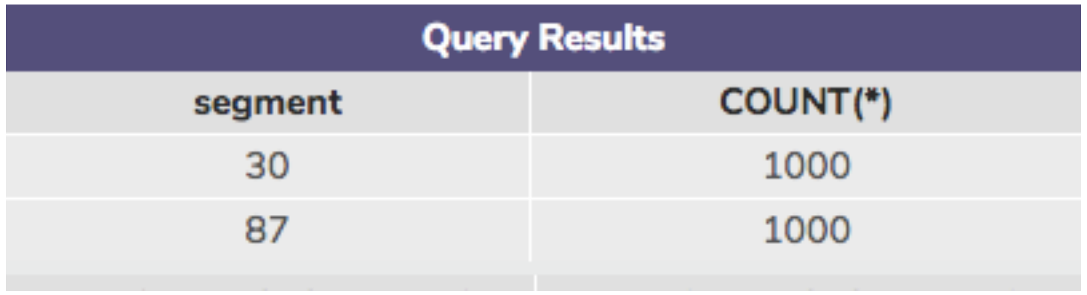
- I used the max and min functions in relation to Codeflix's user subscription start dates to identify how many months the company has been operating.

1.1 Get Familiar with Codeflix (Cont'd)

- **Which segments of users exist?**
- Segments are distinct groups of subscribers that share one or more characteristics.
- Codeflix has two user segments: 87 & 30



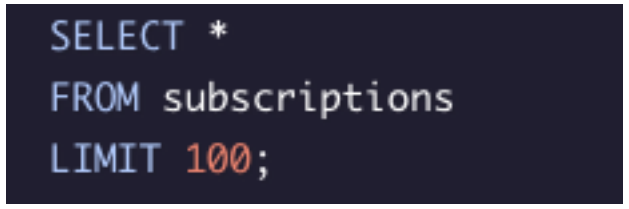
```
SELECT segment, COUNT(*)
FROM subscriptions
GROUP BY segment;
```



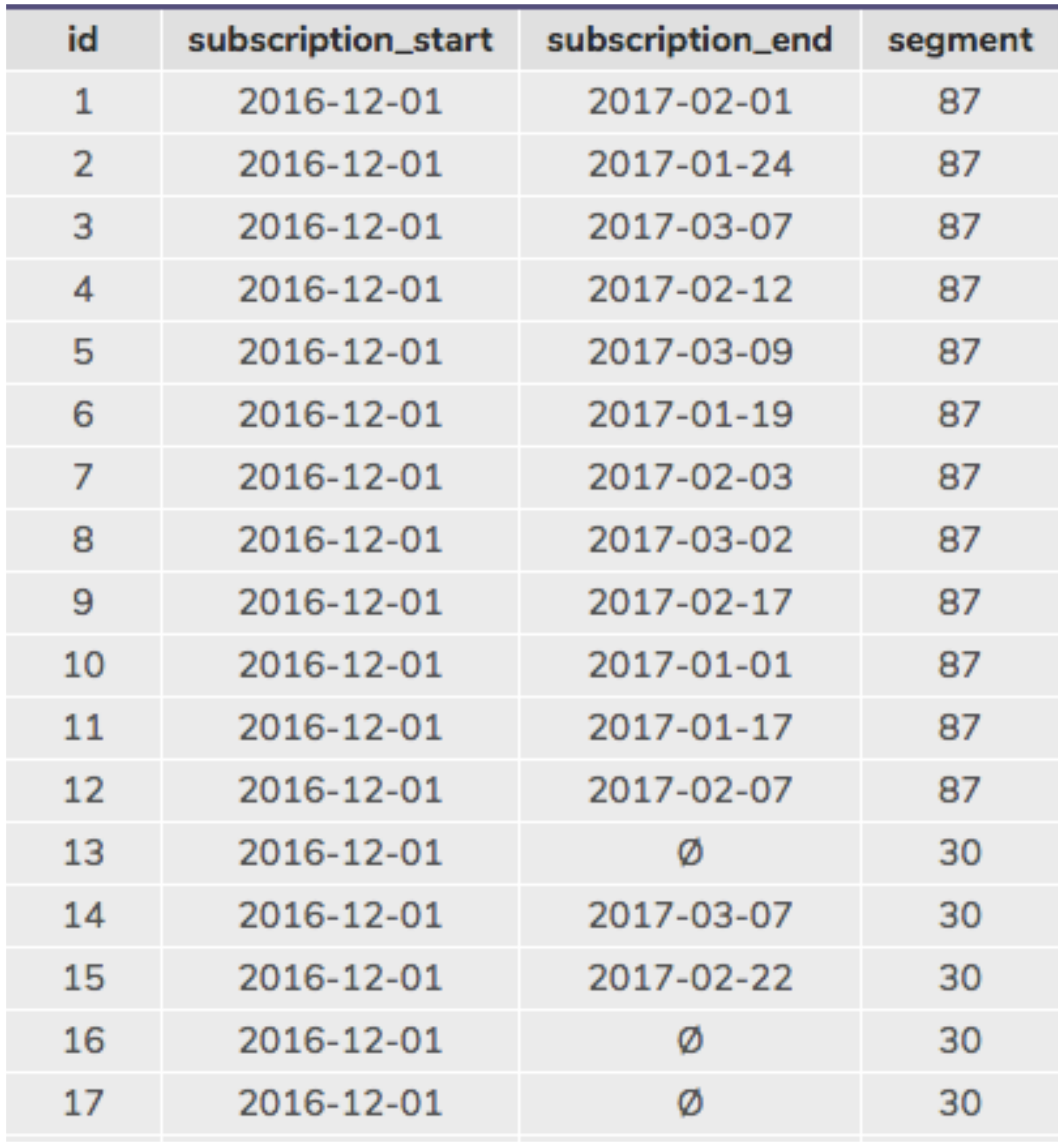
Query Results

segment	COUNT(*)
30	1000
87	1000

- I used the GROUP BY function to group the records in the table by segment, which showed me that there are two different user segments (above).
- I used the SELECT * function to get familiar with all of Codeflix's user data in the subscription table (right).



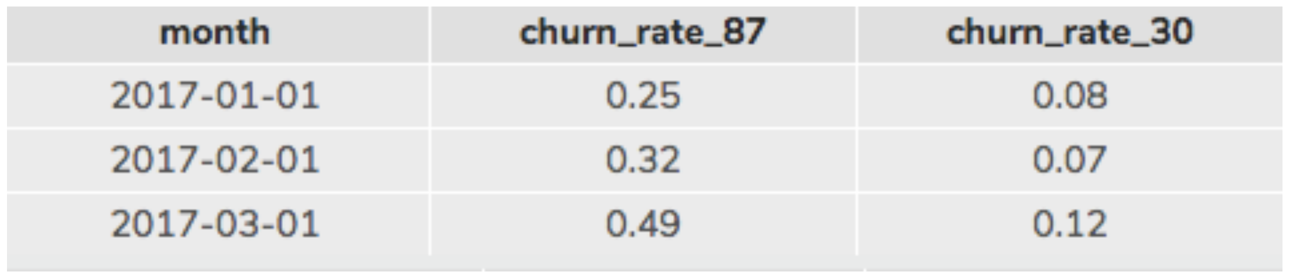
```
SELECT *
FROM subscriptions
LIMIT 100;
```



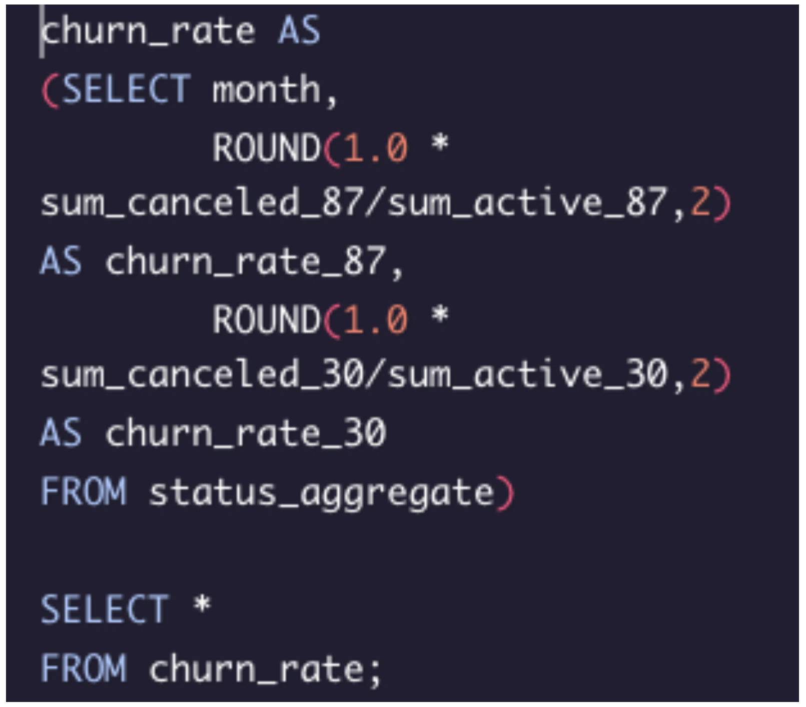
id	subscription_start	subscription_end	segment
1	2016-12-01	2017-02-01	87
2	2016-12-01	2017-01-24	87
3	2016-12-01	2017-03-07	87
4	2016-12-01	2017-02-12	87
5	2016-12-01	2017-03-09	87
6	2016-12-01	2017-01-19	87
7	2016-12-01	2017-02-03	87
8	2016-12-01	2017-03-02	87
9	2016-12-01	2017-02-17	87
10	2016-12-01	2017-01-01	87
11	2016-12-01	2017-01-17	87
12	2016-12-01	2017-02-07	87
13	2016-12-01	Ø	30
14	2016-12-01	2017-03-07	30
15	2016-12-01	2017-02-22	30
16	2016-12-01	Ø	30
17	2016-12-01	Ø	30

2. What is the Overall Churn Trend?

- **Overall, the monthly churn rate is increasing each month for subscribers of Codeflix.**
- For both segments, the churn rate is higher each subsequent month than in the previous month.
- **This means a larger percentage of users are canceling their memberships each subsequent month, across both segments.**
- The one *slight exception* is that the churn rate for segment 30 is 1% lower in February 2017 than in January 2017.
- As you can see to the right, I calculated churn by taking the total number of cancelations within a particular month and segment and dividing that sum by the total number of active users at the beginning of that month and in that segment.



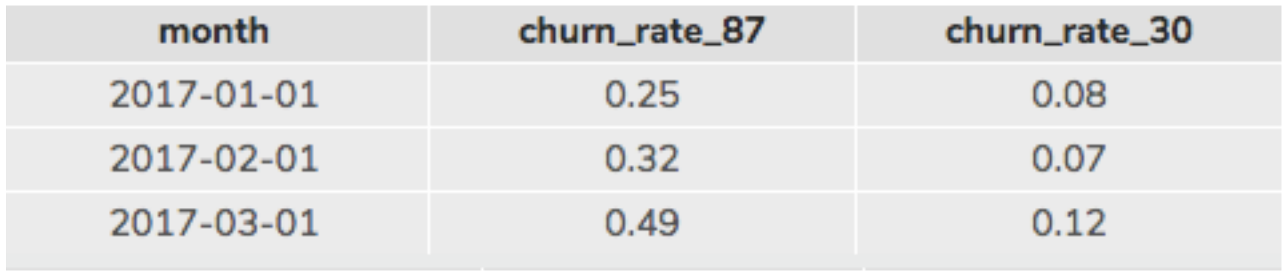
month	churn_rate_87	churn_rate_30
2017-01-01	0.25	0.08
2017-02-01	0.32	0.07
2017-03-01	0.49	0.12



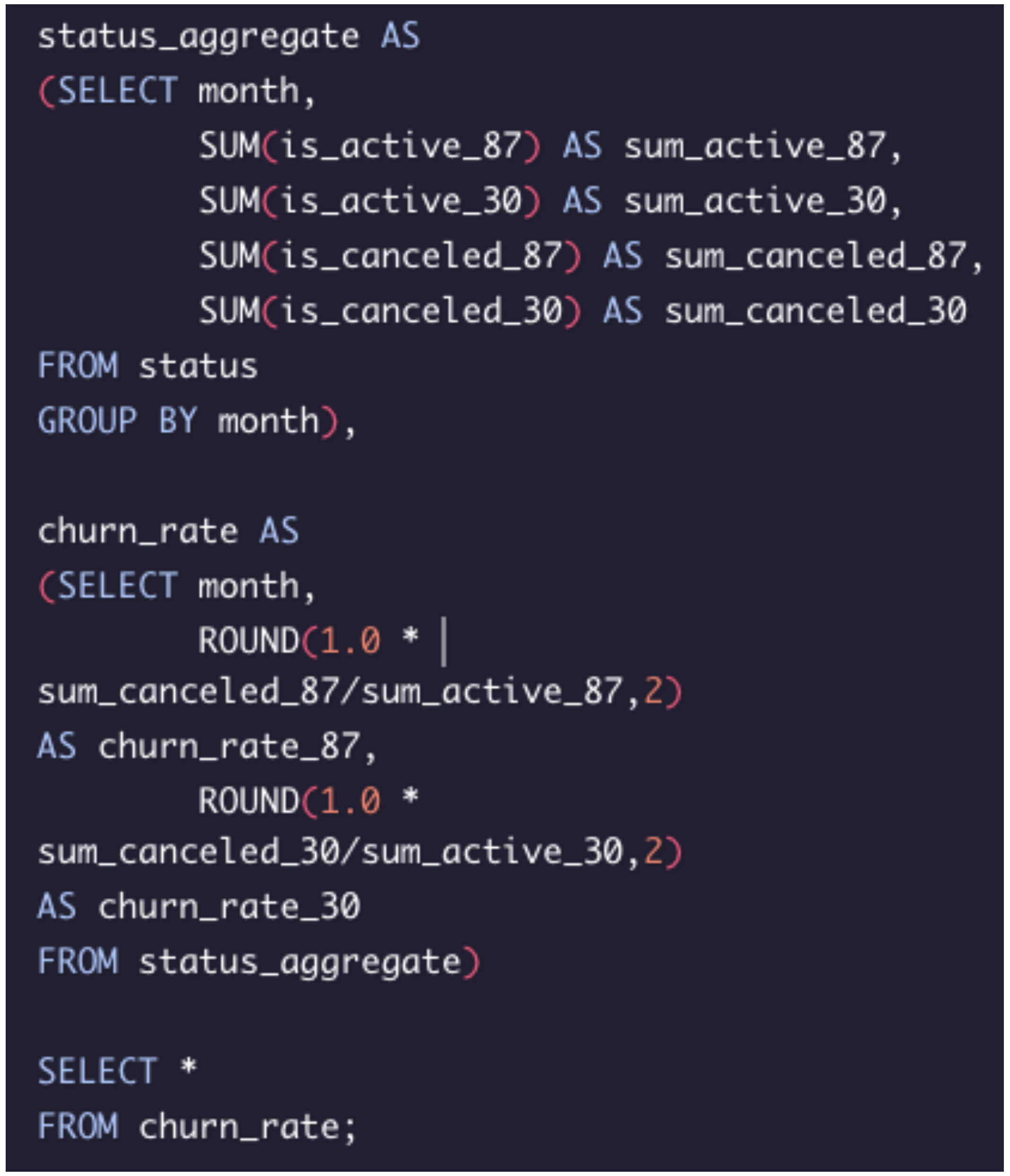
```
churn_rate AS
(SELECT month,
        ROUND(1.0 *
sum_canceled_87/sum_active_87,2)
AS churn_rate_87,
        ROUND(1.0 *
sum_canceled_30/sum_active_30,2)
AS churn_rate_30
FROM status_aggregate)

SELECT *
FROM churn_rate;
```

3. Compare the Churn Rates Between User Segments

- The churn rate for segment 87 is consistently higher than the churn rate for segment 30.
- The churn rate for segment 87 also increases at a faster pace than the churn rate for segment 30.
- The highest churn rate for segment 30 is still lower than the lowest churn rate for segment 87.
- Unlike segment 87, the churn rate for segment 30 decreases slightly from January to February before increasing from February to March.
- **Codeflix is losing a larger percent of users in segment 87 than in segment 30 every month.**
- **I recommend that Codeflix focus on expanding segment 30, which has the lower churn rate of the two groups. The consistently lower churn rate suggests that the users in segment 30 are more likely to be long-term customers.**



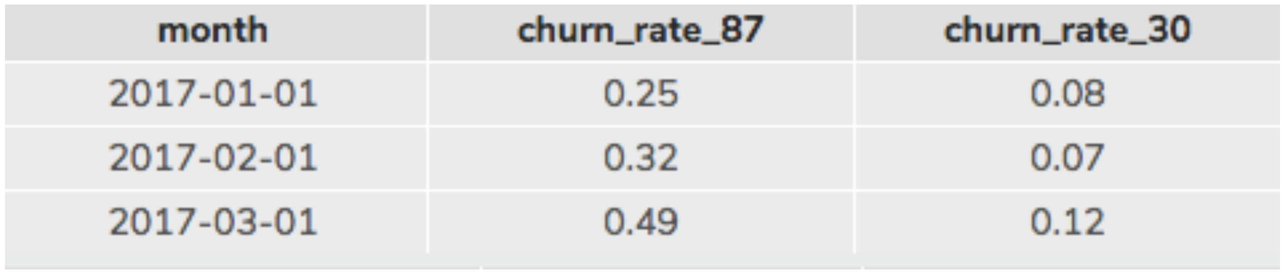
month	churn_rate_87	churn_rate_30
2017-01-01	0.25	0.08
2017-02-01	0.32	0.07
2017-03-01	0.49	0.12



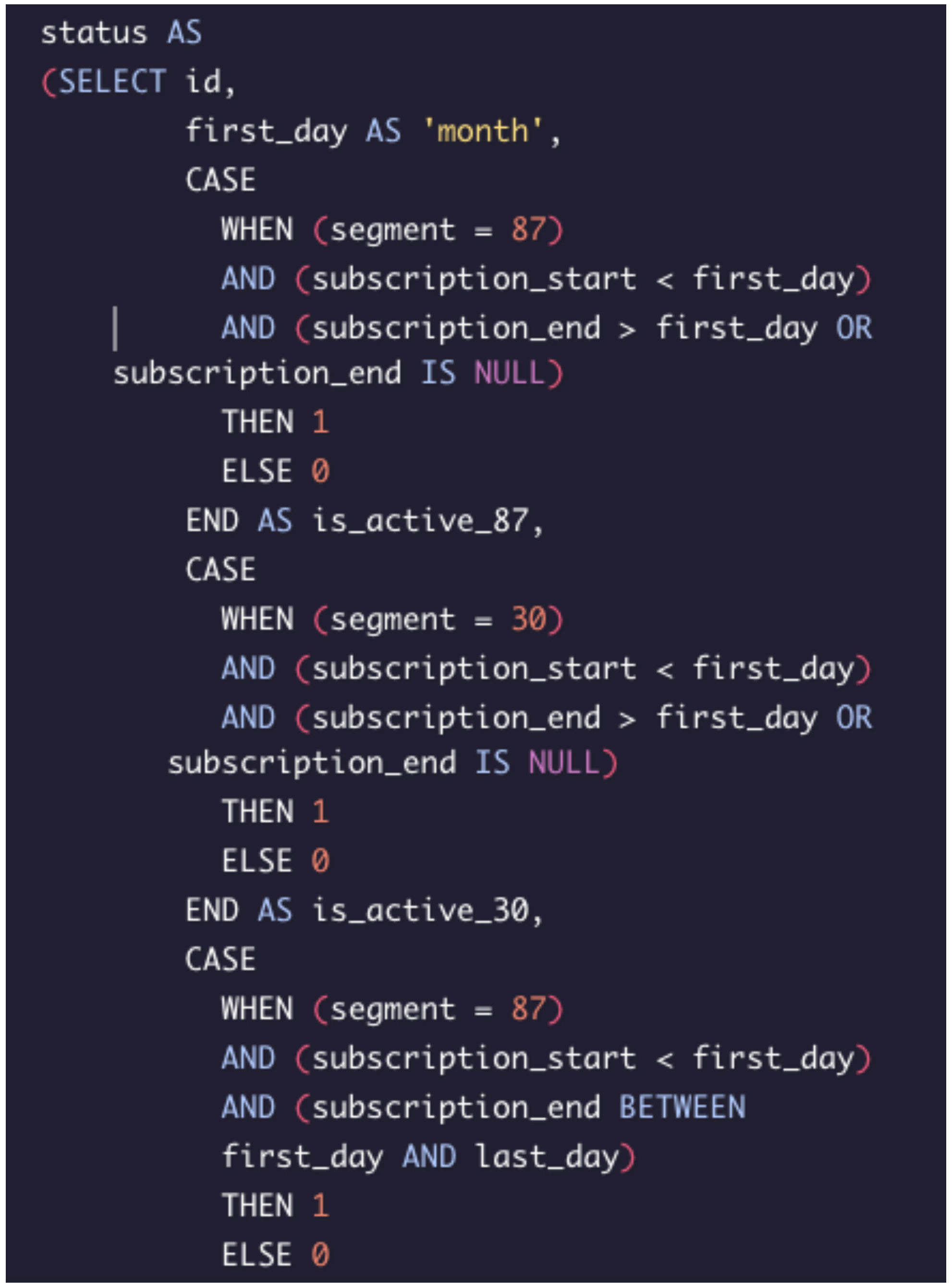
```
status_aggregate AS
(SELECT month,
        SUM(is_active_87) AS sum_active_87,
        SUM(is_active_30) AS sum_active_30,
        SUM(is_canceled_87) AS sum_canceled_87,
        SUM(is_canceled_30) AS sum_canceled_30
FROM status
GROUP BY month),

churn_rate AS
(SELECT month,
        ROUND(1.0 * |
sum_canceled_87/sum_active_87,2)
AS churn_rate_87,
        ROUND(1.0 *
sum_canceled_30/sum_active_30,2)
AS churn_rate_30
FROM status_aggregate)

SELECT *
FROM churn_rate;
```

4.1 Bonus: How Would You Modify the Code to Support a Large Number of Segments?

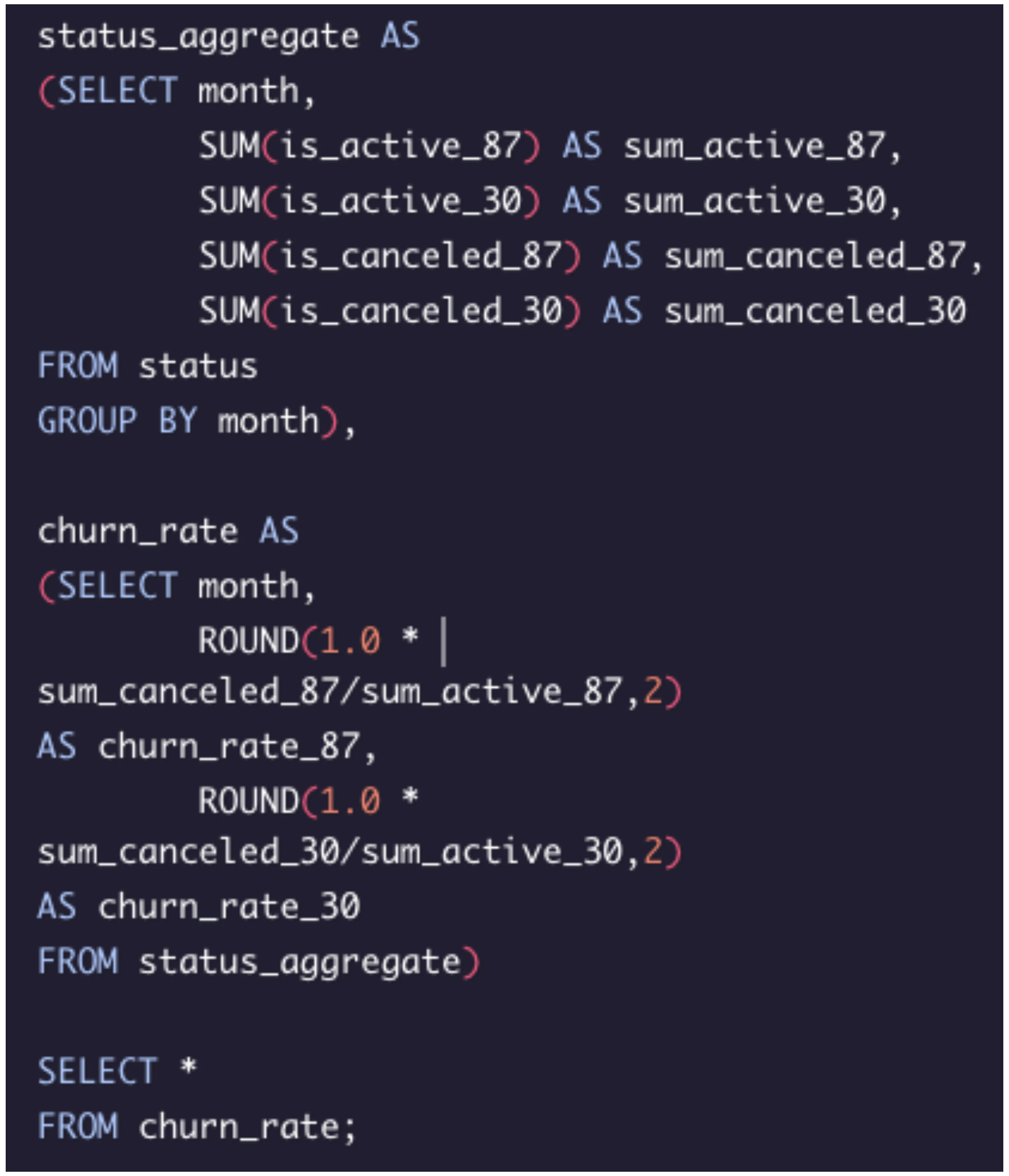
- **Since Codeflix only had two segments, I hard coded the segment numbers into the queries to return the churn rate table to the right.**



month	churn_rate_87	churn_rate_30
2017-01-01	0.25	0.08
2017-02-01	0.32	0.07
2017-03-01	0.49	0.12



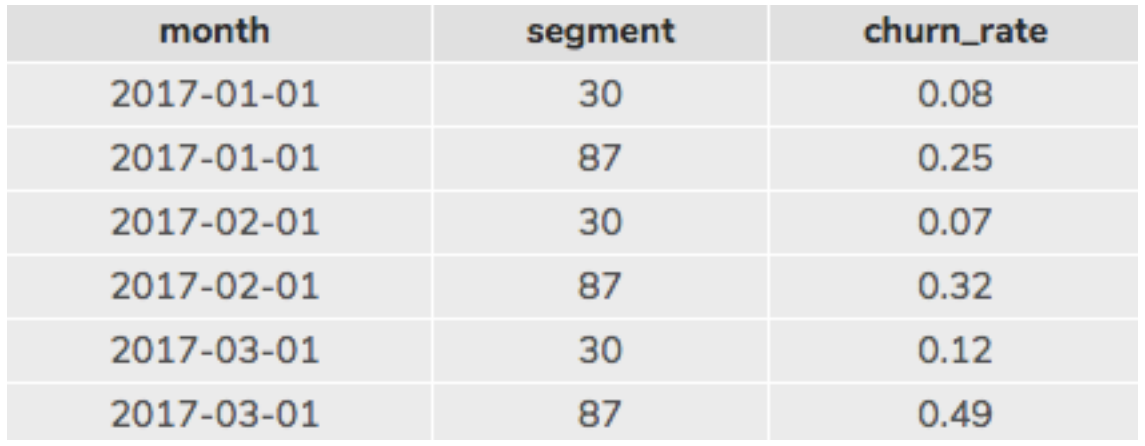
```
status AS
(SELECT id,
        first_day AS 'month',
        CASE
          WHEN (segment = 87)
          AND (subscription_start < first_day)
          AND (subscription_end > first_day OR
    subscription_end IS NULL)
          THEN 1
          ELSE 0
        END AS is_active_87,
        CASE
          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_start < first_day)
          AND (subscription_end BETWEEN
          first_day AND last_day)
          THEN 1
          ELSE 0
```



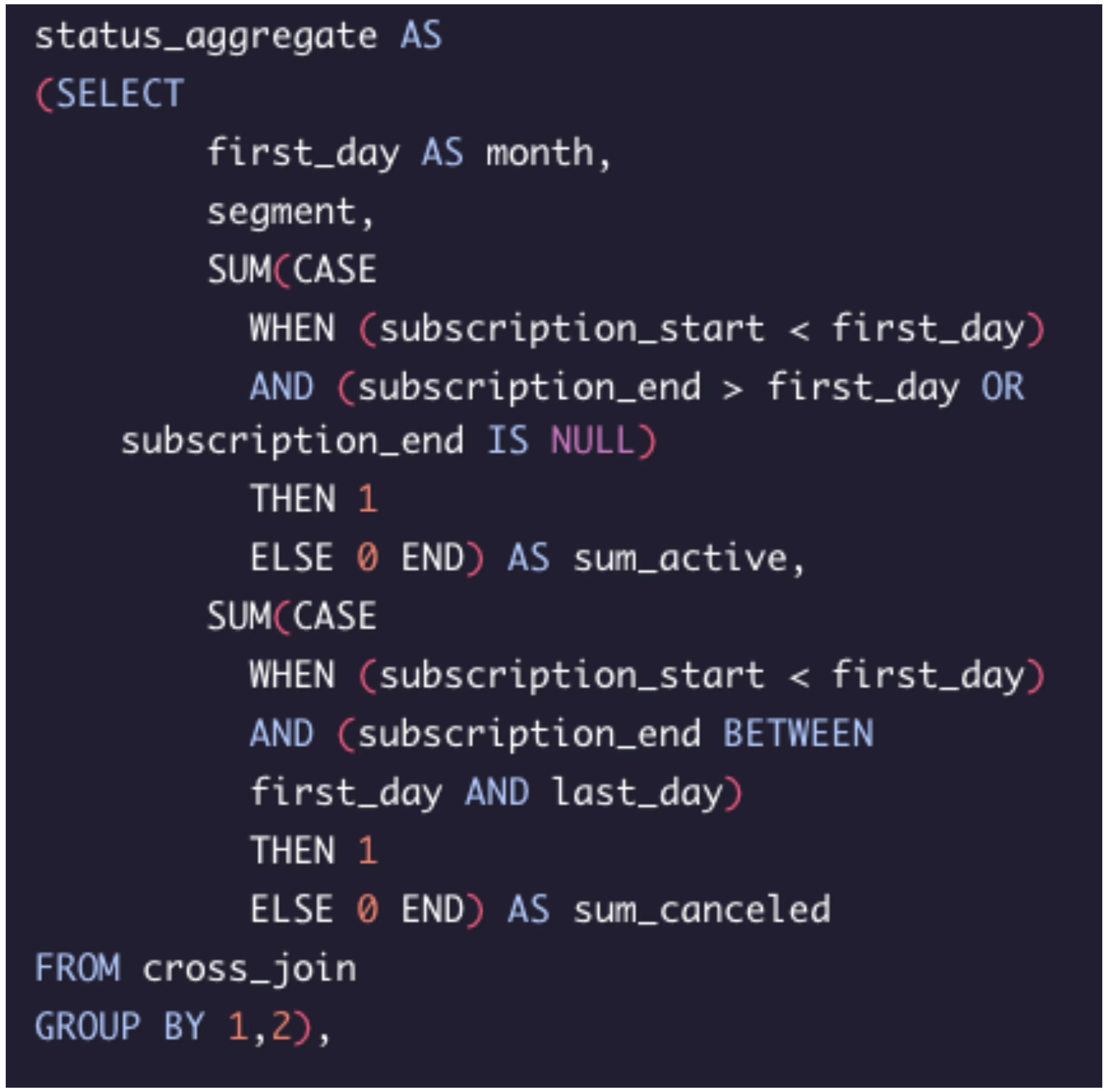
```
status_aggregate AS
(SELECT month,
        SUM(is_active_87) AS sum_active_87,
        SUM(is_active_30) AS sum_active_30,
        SUM(is_canceled_87) AS sum_canceled_87,
        SUM(is_canceled_30) AS sum_canceled_30
FROM status
GROUP BY month),

churn_rate AS
(SELECT month,
        ROUND(1.0 *
sum_canceled_87/sum_active_87,2)
AS churn_rate_87,
        ROUND(1.0 *
sum_canceled_30/sum_active_30,2)
AS churn_rate_30
FROM status_aggregate)

SELECT *
FROM churn_rate;
```

4.2 Bonus: How Would You Modify the Code to Support a Large Number of Segments?

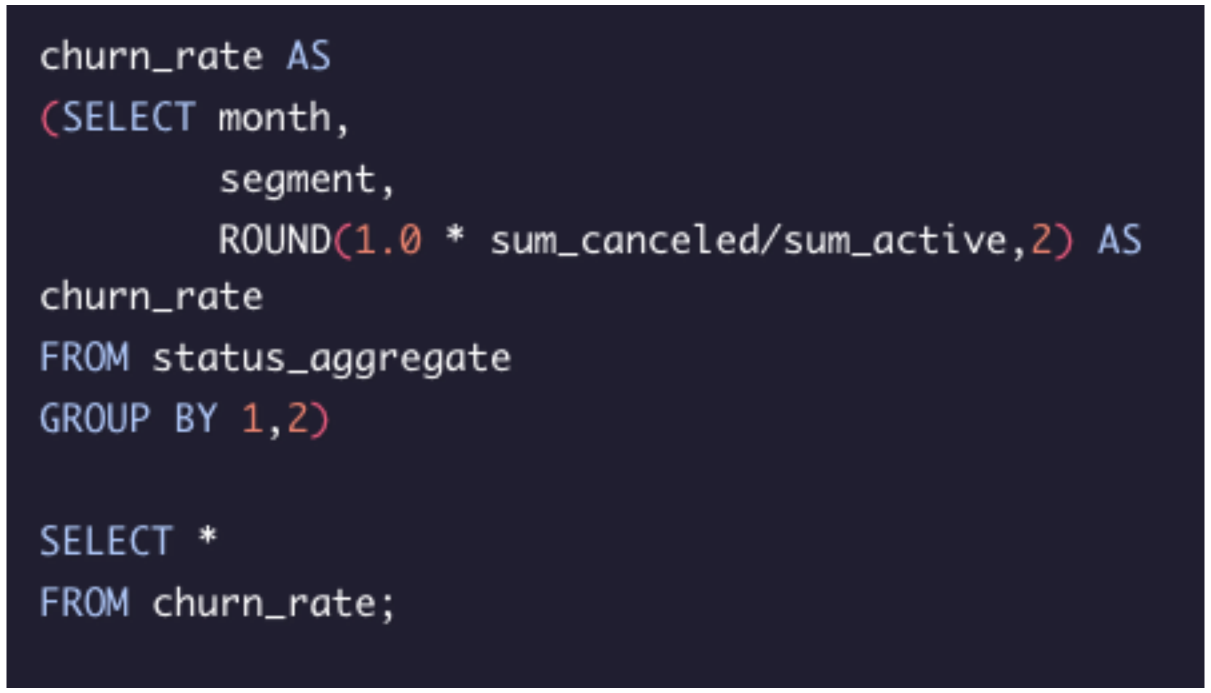
- **However, if there was a large number of segments, I would not hard code the segment numbers into the queries and create a column for each segment number…**
- **Instead, I would create one column called segment and group by month and then by segment number.**



month	segment	churn_rate
2017-01-01	30	0.08
2017-01-01	87	0.25
2017-02-01	30	0.07
2017-02-01	87	0.32
2017-03-01	30	0.12
2017-03-01	87	0.49



```
status_aggregate AS
(SELECT
        first_day AS month,
        segment,
        SUM(CASE
          WHEN (subscription_start < first_day)
          AND (subscription_end > first_day OR
    subscription_end IS NULL)
          THEN 1
          ELSE 0 END) AS sum_active,
        SUM(CASE
          WHEN (subscription_start < first_day)
          AND (subscription_end BETWEEN
          first_day AND last_day)
          THEN 1
          ELSE 0 END) AS sum_canceled
FROM cross_join
GROUP BY 1,2),
```



```
churn_rate AS
(SELECT month,
        segment,
        ROUND(1.0 * sum_canceled/sum_active,2) AS
churn_rate
FROM status_aggregate
GROUP BY 1,2)

SELECT *
FROM churn_rate;
```