codecademy

Capstone Project - Churn Rates for Codeflix

Learn SQL from Scratch

By Pui Yan WU

Table of Contents

1. Get familiar with Codeflix

Very straight forward question nr.1. I entered the codes on the right to check how the data set looks like. SELECT all columns from data set ‘subscriptions’ and LIMIT the result to 100x rows.

To be more specific on how many segments are in the data set, I used GROUP BY segment and removed LIMIT. It returned 2x segments, ’87’ and ‘ 30’ .

```
SELECT *
 FROM subscriptions
 LIMIT 100;

-- or

SELECT *
FROM subscriptions
GROUP BY segment;
```

Result

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

2. Months to be used for calculating churn rate

To answer question nr.2, I used MAX(subscription_start) and MIN(subscription_start) to find out the lowest (or earliest) and highest (or latest subscription) start dates. That helped determine the range of months to be used for calculating churn rate. The results showed that we could use the period between 2016-12-01 to 2017-03-30 for calculation.

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

Results

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

3. Create a temporary table of ‘months’ and ‘cross_join’

This slide contains the answers for question nr.3 and nr.4. To create the temporary table ‘month’, I made use of SELECT and UNION commands - I set the first day of each month to be ‘first_day’, the last day of each month to be ‘last_day’ , and used UNION to join all of them together.

After that, I combined data tables ‘ subscriptions’ and ‘months’ by using the command CROSS JOIN. That combined all rows of the first table with all rows of the second table. See results on the next slide.

```
WITH
months AS
(SELECT
      '2016-12-01' AS 'first_day',
      '2016-12-31' AS 'last_day'
               UNION
     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'
    FROM subscriptions),

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

Results - months

first_day	last_day
2016-12-01	2016-12-31
2017-01-01	2017-01-31
2017-02-01	2017-02-28
2017-03-01	2017-03-31

Results – cross_join

id	subscription_start	subscription_end	segment	first_day	last_day
1	2016-12-01	2017-02-01	87	2016-12-01	2016-12-31
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	2016-12-01	2016-12-31

4. Create a temporary table of ‘status’

For questions nr.5 and nr.6, I used 4x CASE WHEN creating the following columns, ‘is_active_87’, ‘is_canceled_87’, ‘is_active_30’ and ‘is_active_30’. Though the exercise suggested to put both ‘active’ together, I preferred to put the columns with the same segment next to each other for a clearer overview.

Slightly different from the hint, I used (susbcription_end >= first_day) instead of >, in order to capture subscribers who canceled their subscriptions on th 1st day. These conditions should be adjusted based on how an active user and canceled user are defined, as well as the period of time we are looking at.

```
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 = 87
       AND ((subscription_start < first_day)
                                   AND (subscription_end BETWEEN first_day AND last_day))
               THEN 1
          ELSE 0
       END AS 'is_canceled_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 = 30
       AND ((subscription_start < first_day)
                                   AND (subscription_end BETWEEN first_day AND last_day))

                                  THEN 1
          ELSE 0
       END AS 'is_canceled_30'
                    FROM cross_join),
```

5. Which segment has a lower churn rate?

To answer questions nr. 7 and nr. 8, I created the table ' status_aggregate' using the codes shown on the right. After that, I calculated the churn rate for each segment, and casted the result to a float by multiplying by 1.0.

Based on the results, we can conclude that segment 30 has a lower churn rate for all three months, with
-- January - 0.251 (segment87) & 0.076 (segment30), difference is ~0.175
-- February - 0.317 (segment87) & 0.073 (segment30), difference is ~0.244
-- March - 0.477 (segment 87) & 0.117 (segment30), difference is ~0.36

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

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

Results

month	churn_rate_87	churn_rate_30
2016-12-01		
2017-01-01	0.25089605734767	0.0756013745704467
2017-02-01	0.316916488222698	0.0733590733590734
2017-03-01	0.476894639556377	0.116991643454039

6. Modify the codes to support a large number of segments

For the bonus question, I created an extra temporary table ‘test_status’. Although the commands are similar to those in ‘status’ , I removed the command ‘segment = 87’ and ‘segment = 30’ to avoid hard coding, and only used 2x CASE WHEN, one for all active users and one for all canceled users.

In addition, I also incorporated churn rate calculation in it and used GROUP BY ‘segment’ and ‘first_day’ to find out each segment’s churn rate for a particular month (results are checked and they are the same as those from ‘status_aggregate’). In this way, we can handle a large number of segments and will still be able to get a clear overview of the churn rate for each month of each segment.

Results

first_day	segment	churn_rate
2016-12-01	30	
2017-01-01	30	0.0756013745704467
2017-02-01	30	0.0733590733590734
2017-03-01	30	0.116991643454039
2016-12-01	87	
2017-01-01	87	0.25089605734767
2017-02-01	87	0.316916488222698
2017-03-01	87	0.476894639556377

```
test_status AS
 (SELECT first_day, segment,
1.0* SUM(CASE
          WHEN
       (subscription_start < first_day)
                              AND (subscription_end
BETWEEN first_day AND last_day)
               THEN 1
          ELSE 0
       END) /
  SUM(CASE
                            WHEN
                (subscription_start < first_day)
           AND (subscription_end >= first_day
                OR subscription_end IS NULL)
               THEN 1
            ELSE 0
       END) AS churn_rate
 FROM cross_join
 GROUP BY segment, first_day)

 SELECT *
 FROM test_status;
```