Codeflix

Learn SQL From Scratch - Calculating Churn Rates

Melissa Krouse

Table of Contents

1. Getting Familiar with Codeflix
2. Overall Churn Trend
3. Comparing Churn Rates between User Segments

Getting Familiar with Codeflix

How Many Months has Codeflix been Operating?

Codeflix has been operating for four months.

With such a new company, management is extremely curious to know how Codeflix is doing, particularly in churn rates.

Which Months Have Enough Information to Calculate a Churn Rate?

Here I utilized an aggregate function (MAX, MIN) to determine the earliest and latest subscription dates from the subscriptions table. Although the first start date is December 1, 2016, Codeflix requires a minimum length of 31 days so a user can never start and end their subscription in the same month. A user must be before the beginning of the month to be considered active for that month. Because they started within the month of December, I cannot calculate the churn rate for December.

--Query

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

Result:

First Start Date: December 1, 2016
Last Start Date: March 30, 2017

Churn Rate Months: January, February, and March

What Segments of Users Exist?

Here I wanted to determine which segments exist, or which segment the subscription owner belongs to. With this query, I was able to select all from the subscriptions table, and limit it to solely view the first 100. The following table appeared with 100 lines of data:

id	subscription_start	subscription_end	segment

In navigating the results, I found two different segments: Segment 87 and Segment 30.

--Query

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

Result:

Segment 87
Segment 30

Overall Churn Trend

Creating a Temporary Table: “months”

Because I am creating a temporary table, and it is the first temporary table I have created in this SQL query, I have to start it with “WITH” and follow it with “months” to name it months. This months table will eventually be used as a cross join with the subscriptions table to see if the subscriptions fall within the given months. Next, I selected the first day and aliased it as first_day, continued with the last day as last_day. Then, stacked the table using UNION with the other months, selecting the first and last days as well. Finalized it by selecting all from months.

```
--Query
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
) SELECT * FROM months;
```

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

The resulting table gives the first day and last day of each month that the churn rate can be calculated for

Creating a Temporary Table: “cross_join”

Because this is the second temporary table, I no longer have to start the Query with “WITH”, and can solely add to the already existing Query from the previous slide (starting where “‘2017-03-31’ as last_day),” ends). I create a new temporary table named “cross_join” which includes everything (SELECT *) from the subscriptions table AND the months table that I had just created. For the purpose of this presentation, I limited the rows showing to 3 (as that shows each possible combo for that given user; if I were to allow all rows to show, each user would have 3 rows (the potential first and last months))

```
--Query

cross_join AS
(SELECT * FROM
subscriptions
  CROSS JOIN months
) SELECT * FROM cross_join
LIMIT 3;
```

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

Creating a Temporary Table: “status”

I continue by adding to the existing Query from the previous slide (starting where “‘CROSS JOIN months),” ends). Here, I create a new table named “status”. I utilize CASE WHEN statements to determine if users were active for the month AKA subscribed before the beginning of the month (subscription_start < first_day) AND subscription end is later than the first day or has not yet been canceled (subscription_end > first_day OR subscription_end IS NULL) AND belongs to Segment 87 (segment = 87). If ALL were true, then it would return true, represented by a 1 (THEN 1), otherwise a 0 (ELSE 0). I end the case statement by naming it “is_active_87” (END AS is_active_87). You will then be able to count how many 1’s resulted, so you can determine how many users were active in that month (and resultantly the churn rate). I repeated these steps for Segment 30. For the purpose of this presentation, I limited the rows to the first 10, as they appear on the next slide.

```
--Query
status AS
(SELECT
 id,
 first_day AS month,
 CASE
  WHEN (subscription_start < first_day) AND
 (subscription_end > first_day OR subscription_end IS NULL) AND
(segment = 87) THEN 1
  ELSE 0
 END AS is_active_87,
 CASE
  WHEN (subscription_start < first_day) AND
 (subscription_end > first_day OR subscription_end IS NULL) AND
(segment = 30) THEN 1
  ELSE 0
 END AS is_active_30
 FROM cross_join
) SELECT * FROM status LIMIT 10;
```

Temporary Table: “status”

id	month	is_active_87	is_active_30
1	2017-01-01	1	0
1	2017-02-01	0	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

I have initial concern with Segment 30, as they have no 1’s yet (less active users to be incorporated in the churn rate). However, this is limited to solely the first 10 rows, so no decisions can be made yet.

Temporary Table: “status” With **Canceled** Subscribers

Here, I add two additional columns “is_canceled_87” and “is_canceled_30” to the already existing “status” temporary table. I do this by adding to the previous Query (starting where “END AS is_active_30,” ends). I use two additional CASE WHEN statements, to return a 1 (THEN 1) if the subscription ends between the first and last day (subscription_end BETWEEN first_day AND last_day) AND belongs to segment 87 (segment = 87). I repeat these steps for Segment 30, naming their columns utilizing END AS. For the purpose of this presentation, I limited the rows to be shown to the first 10. A new table with results appears on the following slide:

```
--Query
CASE
  WHEN (subscription_end BETWEEN first_day AND last_day)
AND (segment = 87) THEN 1
  ELSE 0
 END AS is_canceled_87,
 CASE
  WHEN (subscription_end BETWEEN first_day AND last_day)
AND (segment = 30) THEN 1
  ELSE 0
 END AS is_canceled_30
FROM cross_join
) SELECT * FROM status;
```

Temporary Table: “status” With **Canceled** Subscribers

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
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

It is evident through these results that Segment 87 is experiencing cancellations, which will appear in their churn rates.

Creating a Temporary Table: “status_aggregate”

I do this by continuing the Query from the previous slide (starting where “FROM cross_join” ends). I create a table called “status_aggregate” to sum the active users for Segment 87 (SUM(is_active_87)) and title that column (AS sum_active_87). I repeat this step for canceled 87, active 30, and canceled 30. Finally, I group it by month (GROUP BY month). I do this because I want to then be able to calculate the churn per month.

```
--Query
), 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
) SELECT * FROM status_aggregate;
```

month	sum_active_87	sum_active_30	sum_canceled_87	sum_canceled_30
2017-01-01	268	291	70	22
2017-02-01	462	518	148	38
2017-03-01	531	716	258	84

Churn Rates

The final step is to calculate the churn rate, by dividing the sum of the canceled users (per month per segment) by the sum of the active users (per month per segment), multiplied by 1.0. This is done because when dividing, the result must be cast as a float. I continue the previous query (starting where “GROUP BY month” ends), and resultantly, the churn rate is calculated per month, per segment.

```
--Query
) 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;
```

Overall Churn Trend

month	churn_rate_87	churn_rate_30
2017-01-01	0.251798561151079	0.0756013745704467
2017-02-01	0.32034632034632	0.0733590733590734
2017-03-01	0.485875706214689	0.11731843575419

The overall churn trend since the company started, for both segments, is increasing. It is clear that the churn is relatively low in both segments within January, because it is the first month of full active subscribers. However, in the month of March, churn nearly doubles for Codeflix in both segments of users. Management must look into their channels of acquisition, to lower their churn rates, or else they will face trouble in the future.

Comparing Churn Rates Between User Segments

Segment 87 and Segment 30 Churn Rates

month	churn_rate_87	churn_rate_30
2017-01-01	0.251798561151079	0.0756013745704467
2017-02-01	0.32034632034632	0.0733590733590734
2017-03-01	0.485875706214689	0.11731843575419

In comparing Segment 87 with Segment 30, each month, Segment 30 has lower churn rates, supporting that they are performing better than Segment 87. They have less people unsubscribing per given month. The marketing team should consider minor improvements to ensure their churn rate does not increase any further for Segment 30, while considering much more change and investments into Segment 87.

Modifying the Code

If I were working with a larger number of segments (not solely Segment 87 and Segment 30), I would modify the code by avoiding hardcoding. As I did with adding a column “months”, I would add another column “segments.” In doing so, I would be able to sum all “is_active” and all “is_canceled” and then GROUP BY, to narrow in to the segments of interest.