Note!

This Powerpoint goes into fairly granular detail about the SQL code (for eg. why certain commands are used, and when). I did this deliberately to force myself to write out my thought process in a step-by-step manner and is a way for me to learn the ins-and-outs of SQL at a much deeper level.

codecademy

Codeflix Churn Rates Analysis

Richmond Wong

5 June 2018 Cohort

Table of Contents

Section 1
Get Familiar with Codeflix

What segments of users exist?

- Two segments of users exist: *30* and *87*

SQL CODE:

```
SELECT segment
FROM subscriptions
GROUP BY segment;
```

segment
30
87

Questions about the range of months

How many months has the company been operating?

The data available to us runs from 1 Dec 2016 to 31 March 2017, for a total of 4 full months. This doesn't mean the company has been operating for 4 months only, just that the data set available to us is limited to these dates.

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

On the other hand, there is no cancellation data for the month of Dec 2016 – the earliest *subscription_end* data is 1 Jan 2017.

Churn rate is calculated by:

The number of cancellations during the month in question

DIVIDED BY

The total number of accounts at the beginning of the month carried over from the previous month AND not taking into account any new accounts (even those activated on the first day of the month in question)

Therefore, since we don't have any cancellation data for Dec 2016, we are only able to calculate the Churn Rates for January to March 2017.

SQL CODE:

```
SELECT min(subscription_start),
min(subscription_end),
max(subscription_end)
FROM subscriptions;
```

min(subscription_start)	min(subscription_end)	max(subscription_end)
2016-12-01	2017-01-01	2017-03-31

Section 2
Calculating Churn Rate for Each Segment from Jan to Mar 2017

Step 1 - Create a Temporary Table, “months”

- To calculate the Churn Rates for each of the first three months in 2017, we start off by creating a table (aliased as “months”) that displays Jan, Feb and Mar 2017.

- We can do so by creating a two-column table displaying the first and last days of each month.

- We use UNION to sequentially stack the three months on top of each other so that the table would look like the one on the right.

SQL CODE:

```
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)
```

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

Step 2 – Create “cross_join” temporary table

- Step 2 is to CROSS JOIN our “months” table with our original *subscriptions* table.
- CROSS JOINing these two tables allows us to take the *first_day* and *last_day* rows for Jan, Feb and Mar and join it to every single user id. This is why you see each user id repeated three times (the total number of rows in a table created from a CROSS JOIN can be found by multiplying the number of rows of the first table by the number of rows in the second table)
- Since we SELECTed for all columns in *subscription*, we also join information on *segment, subscription_start and subscription_end*.

SQL CODE:

```
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 subscriptions
CROSS JOIN months;
```

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
2	2016-12-01	2017-01-24	87	2017-01-01	2017-01-31
2	2016-12-01	2017-01-24	87	2017-02-01	2017-02-28
2	2016-12-01	2017-01-24	87	2017-03-01	2017-03-31
3	2016-12-01	2017-03-07	87	2017-01-01	2017-01-31
3	2016-12-01	2017-03-07	87	2017-02-01	2017-02-28
3	2016-12-01	2017-03-07	87	2017-03-01	2017-03-31
4	2016-12-01	2017-02-12	87	2017-01-01	2017-01-31
4	2016-12-01	2017-02-12	87	2017-02-01	2017-02-28
4	2016-12-01	2017-02-12	87	2017-03-01	2017-03-31

Step 3 – Create “status” temporary table

- Step 3 is to create a “status” temporary table.
- The code inside the “status” alias takes the previous CROSS JOINed table and SELECTs the *user_id*, *first_day* (which represents the whole months of Jan, Feb and Mar) and several CASE statements.
- The CASE statements take the *subscription_start* and *subscription_end* dates and depending on what criteria is fulfilled will output a 1 or 0 (representing true or false respectively) to the four *is_active* and *is_canceled* columns.
- This will allow us to see a user’s active/inactive status on a month by month basis from the beginning of Jan to the end of Mar.

id	first_day	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

SQL CODE:

```
/* months temporary table */
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),

/* cross_join temporary table */
cross_join AS (SELECT *
FROM subscriptions
CROSS JOIN months),

/* status temporary table */
status as (SELECT id, first_day,

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)
```

Step 4 – Create “status_aggregate” temporary table

- Step 4 involves summing all the 1’s and 0’s in the four *is_active* and *is_canceled* columns.
- The code inside the *status_aggregate* alias would produce a table like the one below.

first_day	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

SQL CODE:

```
/* months temporary table */
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),

/* cross_join temporary table */
cross_join AS (SELECT *
FROM subscriptions
CROSS JOIN months),

/* status temporary table */
status as (SELECT id, first_day,

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),

/* status_aggregate temporary table */
status_aggregate AS (SELECT first_day, 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 first_day)
```

Step 5 – Churn Calculation

- From the chart we see that Churn Rates are:

Segment *87*'s are 25% (Jan 2017), 32% (Feb 2017), 49% (Mar 2017)

Segment *30*'s are: 8% (Jan 2017), 7% (Feb 2017), 12% (Mar 2017)

- Segment *30* has a Lower churn Rate.

first_day	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

SQL CODE:

```
/* months temporary table */
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),

/* cross join temporary table */
cross_join AS (SELECT *
FROM subscriptions
CROSS JOIN months),

/* status temporary table */
status as (SELECT id, first_day,

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),

/* status aggregate temporary table */
status_aggregate AS (SELECT first_day, 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 first_day)

/* Churn calculation */
SELECT first_day,
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;
```