codecademy

Calculating Churn Rates

Learn SQL from Scratch

1. Get familiar with the data

1. Get familiar with the data

Table ‘subscriptions’ contains four different columns:

- User id (‘id’)
- Start date of subscription (‘subscription_start’)
- End date of subscription (‘subscription_end’)
- Segment the subscription owner belongs to (‘segment’)

The table contains data from December 2016 to March 2017.

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

```
SELECT *
FROM subscriptions
LIMIT 100;


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

2. Calculate churn rate for each segment

2. Temporary table ‘months’

A temporary table for each month has been created that shows the first and last day of a month in each column for the data set.

A WITH statement to used to work with the table moving forward.

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

```
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;
```

2. Temporary table ‘cross_join’

A temporary table ‘cross_join’ has been created that cross-joins the temporary table months and the table subscriptions.
The cross join captures the cartesian product of the tables months and subscriptions.

A WITH statement to used to work with the table moving forward.

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

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

2. Temporary table ‘status’

A temporary table ‘status’ has been created that finds any users who existed prior to the beginning of the month for both user segments ‘87’ and ‘30’ in the subscriptions table.

A WITH statement to used to work with the table moving forward.

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

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

2. Temporary table ‘status’

The columns ‘is_canceled_87’ and ‘is_canceled_30’ have been added to the table status to illustrate canceled subscriptions during a month for both segments 87 and 30.

A WITH statement to used to work with the table moving forward.

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

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

3. Results

3. Churn rates for segment 87 and segment 30

After the definition of the temporary tables months, cross_join and status, the churn rates for both segments 87 and 30 can be calculated.

It can be observed that the churn rate for segment 87 is consistently higher than for segment 30 for all months in the data table. This means that relatively more subscribers of segment 87 cancel the subscription in any given month than subscribers of segment 30.

month	churn_rate_87	churn_rate_30
2017-01-01	0.252	0.076
2017-02-01	0.320	0.073
2017-03-01	0.486	0.117

```
WITH ... (
 ...
status_aggregate AS
(SELECT
  month,
  SUM(is_active_87) as active_87,
  SUM(is_canceled_87) as canceled_87,
  SUM(is_active_30) as active_30,
  SUM(is_canceled_30) as canceled_30
FROM status
GROUP BY month)
Select month, 1.0 *canceled_87/active_87 as
churn_rate_87, 1.0 *canceled_30/active_30 as
churn_rate_30
From status_aggregate;
```

2. Interpretation

As the churn rates for segment 87 are much higher than for segment 30, the company should focus on expanding marketing activities for segment 87 subscribers to bring down cancelation rates.

In case more than just two segments would exist, the SQL code could be adjusted by looping over all existing segments to make the code more efficient.