codecademy

User Churn Project

Codeflix

Analyze Data with SQL
User Churn Rate

Alex Ricciardi

Date: 04/04/2020

CodeFlix

Codeflix, a streaming video startup, is interested in measuring their user churn rate.
Four months into launching Codeflix, management asks you to look into subscription churn rates. It's early on in the business and people are excited to know how the company is doing.

The marketing department is particularly interested in how the churn compares between two segments of users. They provide you with a dataset containing subscription data for users who were acquired through two distinct channels.
In this project, you'll be helping them answer these questions about their churn:

1. Get familiar with the company.

   How many months has the company been operating? Which months do you have enough information to calculate a churn rate?
   What segments of users exist?

2. What is the overall churn trend since the company started?

3. Compare the churn rates between user segments.

   Which segment of users should the company focus on expanding?

Codeflix Project Database Schema:

The dataset provided to you contains one SQL table, subscriptions. Within the table, there are 4 columns:

- id - the subscription id
- subscription_start - the start date of the subscription
- subscription_end - the end date of the subscription
- segment - this identifies which segment the subscription owner belongs to

Codeflix requires a minimum subscription length of 31 days, so a user can never start and end their subscription in the same month.

Database Schema		
subscriptions		2000 rows
id	INTEGER	
subscription_start	TEXT	
subscription_end	TEXT	
segment	INTEGER	

Project Tasks:

- **Get familiar with the data**

  1. Take a look at the first 100 rows of data in the subscriptions table.

  2. Determine the range of months of data provided. Calculate churn rate for each segment

- **Calculate churn rate for each segment**

  3.You'll be calculating the churn rate for both segments (87 and 30) over the first 3 months of 2017.

  4. Create a temporary table, cross_join, from subscriptions and your months. Be sure to SELECT every column.

  5. Create a temporary table, status, from the cross_join table you created.

  6. Add an is_canceled_87 and an is_canceled_30 column to the status temporary table.

  7. Create a status_aggregate temporary table that is a SUM of the active and canceled subscriptions for each segment, for each month.

  8.Calculate the churn rates for the two segments over the three-month period. Which segment has a lower churn rate?

- **Bonus**

  9.How would you modify this code to support a large number of segments?

1. Project Task: Take a look

1.1 Description of the task

Take a look at the first 100 rows of data in the subscriptions table.

How many different segments do you see?

1.2 How many different segments

The subscription table possess two different segments that I can see under the column ‘segment’:

Segment 87

Segment 30

Query code

```
SELECT *
FROM subscriptions
LIMIT 100;

 -- How many different segments do you see?

SELECT DISTINCT segment
FROM subscriptions;
```

Query output from the first query code

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

Query output from the second query code

segment
87
30

2. Project Task: Range of months

2.1 Description of the task

Determine the range of months of data provided.

Which months will you be able to calculate churn for?

2.2 Determine the range of months of data provided

- The Range of months provided is:

From	December,	2016
To	March,	2017

- Which months will you be able to calculate churn for?

Codeflix requires a minimum subscription length of 31 days, so a user can never start and end their subscription in the same month.
The month of December churn rate can not be calculated, since there no cancelations in December, December 2016 is the first month in the data inputted in the table.

We calculate the churn rate for the months of:

January,	2017
February,	2017
March,	2017

$$monthly\ churn\ rate = \frac{cancelations\ for\ the\ month}{Total\ subscriptions}$$

Query code

```
SELECT
    MIN(subscription_start) AS 'range_from_date',
    MAX(subscription_end) AS 'range_to_date'
FROM subscriptions;
```

Query output

range_from_date	range_to_date
2016-12-01	2017-03-31

3. Project Task:
Create a temporary table months

3.1 Description of the task

You'll be calculating the churn rate for both segments (87 and 30) over the first 3 months of 2017.

(you can't calculate it for December, since there are no subscription_end values yet).

To get started, create a temporary table of months.

3.2 Create a temporary table of months.

The temporary table of months was created using the UNION operator.

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

Query output

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

4. Project Task:
Create a temporary table cross_join

4.1 Description of the task

Create a temporary table, cross_join, from subscriptions and your months.

Be sure to SELECT every column.

4.2 Create a temporary table of months.

The temporary cross_join table was created using the CROSS JOIN clause.

Query output

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

Query 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
  FROM subscriptions
),
cross_join AS (
  SELECT *
  FROM subscriptions
  CROSS JOIN months
)
SELECT *
FROM cross_join
LIMIT 10;
```

5. Project Task:
Create a temporary table status

5.1 Description of the task

Create a temporary table, status, from the cross_join table you created.

This table should contain:

- id selected from cross_join
- month as an alias of first_day
- is_active_87
  created using a CASE WHEN to find any users from segment 87 who existed prior to the beginning of the month.
  This is 1 if true and 0 otherwise.
- is_active_30
  created using a CASE WHEN to find any users from segment 30 who existed prior to the beginning of the month.
  This is 1 if true and 0 otherwise.

5.2 Create a temporary table status

Adding the expression

'AND segment = 87'

in the 'CASE' statement, outputting if the subscriptions are active, true = 1 or false = 0, during each individual months, will output 1 for is_active_87, for the subscriptions who are active and are only part of the segment 87, if not it will out 0 for is_active_87 and is_active_30.

Adding the expression

'AND segment = 30'

in the 'CASE' statement, outputting if the subscriptions are active, true = 1 or false = 0, during each individual months, will output 1 for is_active_30, for the subscriptions who are active and are only part of the segment 30, if not it will out 0 for is_active_30 and is_active_87.

Query output

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
4	2017-02-01	1	0
4	2017-03-01	0	0
5	2017-01-01	1	0
5	2017-02-01	1	0
5	2017-03-01	1	0
6	2017-01-01	1	0
6	2017-02-01	0	0
6	2017-03-01	0	0
7	2017-01-01	1	0
7	2017-02-01	1	0
7	2017-03-01	0	0
8	2017-01-01	1	0
8	2017-02-01	1	0

Query 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
  FROM subscriptions
),
cross_join AS (
  SELECT *
  FROM subscriptions
  CROSS JOIN months
),
status AS (
  SELECT
    id,
    first_day AS month,
    -- Checking if the customer's subscription is active during each individual months
    CASE -- segment 87
      WHEN (subscription_start < first_day)
        AND segment = 87
        AND (subscription_end > first_day
          OR subscription_end IS NULL) THEN 1
      ELSE 0
    END AS is_active_87,
    CASE -- segment 30
      WHEN (subscription_start < first_day)
        AND segment = 30
        AND (subscription_end > first_day
          OR subscription_end IS NULL) THEN 1
      ELSE 0
    END AS is_active_30
  FROM cross_join
)
SELECT *
FROM status
LIMIT 100;
```

6. Project Task:
Add an is_canceled_87 and an is_canceled_30

6.1 Description of the task

Add an is_canceled_87 and an is_canceled_30 column to the status temporary table.

This should be 1 if the subscription is canceled during the month and 0 otherwise.

6.2 Create a temporary table status

Adding the expression

‘AND segment = 87’

in the ‘CASE’ statement, outputting if the subscriptions are canceled, true = 1 or false = 0, during each individual months, will output 1 for is_ canceled _87, for the subscriptions who are canceled and are only part of the segment 87, if not it will out 0 for is_ canceled _87 and is_ canceled _30.

Adding the expression

‘AND segment = 30’

in the ‘CASE’ statement, outputting if the subscriptions are canceled, true = 1 or false = 0, during each individual months, will output 1 for is_ canceled _30, for the subscriptions who are canceled and are only part of the segment 30, if not it will out 0 for is_ canceled _30 and is_canceled_87.

Query output

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

Query code

The snippet only shows the status table query code

```
status AS (
  SELECT
    id,
    first_day AS month,
    -- Checking if the customer's subscription is active during each individual months
    CASE -- segment 87 active
      WHEN (subscription_start < first_day)
        AND segment = 87
        AND (subscription_end > first_day
          OR subscription_end IS NULL) THEN 1
        ELSE 0
      END AS is_active_87,
      CASE -- segment 30 active
        WHEN (subscription_start < first_day)
          AND segment = 30
          AND (subscription_end > first_day
            OR subscription_end IS NULL) THEN 1
        ELSE 0
      END AS is_active_30,
      -- Checking if the customer's subscription is canceled during each individual months
      CASE -- segment 87 canceled
        WHEN (subscription_end BETWEEN first_day AND last_day)
          AND segment = 87 THEN 1
        ELSE 0
      END AS is_canceled_87,
      CASE -- segment 30 canceled
        WHEN (subscription_end BETWEEN first_day AND last_day)
          AND segment = 30 THEN 1
        ELSE 0
      END AS is_canceled_30
  FROM cross_join
)
------------------------ Query
SELECT *
FROM status
LIMIT 10;
```

7. Project Task: Create a status_aggregate temporary table

7.1 Description of the task

Create a status_aggregate temporary table that is a SUM of the active and canceled subscriptions for each segment, for each month.

The resulting columns should be:

- sum_active_87
- sum_active_30
- sum_canceled_87
- sum_canceled_30

7.2 Create a status_aggregate temporary table

When taking a quick look at the query sums of the status_aggregate table, we can see that the segment 86 experienced a higher rate of cancelation than the segment 30.

Query output

month	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

Query code
The snippet only shows the satus_aggregate table query code

```
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;
```

8. Project Task: Calculate the churn rates

8.1 Description of the task

Calculate the churn rates for the two segments over the three month period.

Which segment has a lower churn rate?

8.3 Churn rates for the two segments over the three month period.

When taking a quick look at the query churn rates of the status_aggregate table.
We can see that the segment 30 has the lowest churn rate of than the segment 87.

Query code
The snippet only shows the satus_aggregate table query code

```sql
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
  month,
  -- Churn rates
  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;
```

Query output

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

8.4 Which segment has a lower churn rate?

The query of the overall churn rates of the status_aggregate table.
Clearly shows the segment 30 has the lowest churn rate of than the segment 87.

Note:
The lines code ‘month,’ and ‘GROUP BY month’
Have been removed from the temporaly status_aggregate table, to output the overall churn rates when computing the sums.

Query code
The snippet only shows the satus_aggregate table query code

```
status_aggregate AS (
  SELECT
      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
)
SELECT
  -- Churn rates
  ROUND(1.0 * sum_canceled_87 / sum_active_87, 2) AS overall_churn_rate_87,
  ROUND(1.0 * sum_canceled_30 / sum_active_30, 2) AS overall_churn_rate_30
FROM status_aggregate;
```

Query output

overall_churn_rate_87	overall_churn_rate_30
0.37	0.09

9. Project Task:
Modify code to support a larger number of segments

9.1 Description of the task

How would you modify this code to support a large number of segments?
From the churn rates query code of the two segments over the three month period.

9.2 Step-1 modify the status table.

- Added the term ‘segment’ to the table

- Modified the CASE statements
  is_active_87’ and ‘is_active_30
  is_canceled_87’ and ‘is_ canceled _30
  From a query base on the tow types of the segment data to a more general query.
  is_active
  is_canceled

- Added the statement GROUP BY with the terms
  id
  segment
  first_day
  To query the CASE statements is_active and is_canceled base on thoss three terms.

Query code
The snippet only shows the status table query code

```
status AS (
  SELECT
    id,
    first_day AS month,
    segment,
    -- Checking if the customer's subscription is active during each individual months
    CASE
      WHEN (subscription_start < first_day)
        AND (subscription_end > first_day
          OR subscription_end IS NULL) THEN 1
        ELSE 0
      END AS is_active,
      -- Checking if the customer's subscription is canceled during each individual months
      CASE
        WHEN (subscription_end BETWEEN first_day AND last_day) THEN 1
        ELSE 0
      END AS is_canceled
  FROM cross_join
  GROUP BY id, segment, first_day
)
------------------------ Query
SELECT *
FROM status
LIMIT 10;
```

Query output

id	month	segment	is_active	is_canceled
1	2017-01-01	87	1	0
1	2017-02-01	87	0	1
1	2017-03-01	87	0	0
2	2017-01-01	87	1	1
2	2017-02-01	87	0	0
2	2017-03-01	87	0	0
3	2017-01-01	87	1	0
3	2017-02-01	87	1	0
3	2017-03-01	87	1	1
4	2017-01-01	87	1	0

9.3 Step-2 modify the status_aggregate table

- Added the term ‘segment’ to the table
- Added to the statement GROUP BY month the terms
  segment
  To query no just by month but also by segment

Query output

month	segment	sum_active	sum_canceled
2017-01-01	30	291	22
2017-01-01	87	278	70
2017-02-01	30	518	38
2017-02-01	87	462	148
2017-03-01	30	716	84
2017-03-01	87	531	258

Query code
The snippet only shows the status_aggregate table query

```
status_aggregate AS (
  SELECT
      month,
      segment,
      SUM(is_active) AS sum_active,
      SUM(is_canceled) AS sum_canceled
  FROM status
  GROUP BY  month, segment
)
------------------------ Query
SELECT *
FROM status_aggregate;
```

9.4 Step-3 The churn rate query

- Added the term ‘segment’ to the table
- Modified the churn rate query expression

  churn_rate_87
  churn_rate_30

  From a query base on the tow types of the segment data to a more general query.

  month_churn_rate

- Added the statement GROUP BY with the terms

  month
  segment

  To compute the churn rate by month and segment

Query code
The snippet only shows the satus_aggregate table query code

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

Query output

month	segment	month_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