codecademy

Calculating Churn Rates

Learn SQL from Scratch

Varaksin Aleksei

Table of Contents

codecademy

About CodeFlix

CodeFlix is a subscription based video streaming service.

- With the help of querying the **min & max** subscription start and end, we can see the date of the start of business and the date of the most recent transaction.

1. Current months of operation: **12/1/2016 - 3/31/2017**
2. Foreseeable months of operation: **12/1/2016 - 5/1/2017**

- Using “GROUP BY” helps to see available segments: **30** and **87**

```
SELECT MIN(subscription_start) as 'Earliest_Start',
MAX(subscription_end) as 'Max_Sub_End',
MAX(subscription_start) as 'Max_Sub_Start',
segment

FROM subscriptions

GROUP BY 4

ORDER BY 1,2 DESC;
```

```
SELECT *
From Subscriptions
LIMIT 100;
```

Query Results

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

Query Results

Earliest_Start	Max_Sub_End	Max_Sub_Start	segment
2016-12-01	2017-03-31	2017-03-30	30
2016-12-01	2017-03-31	2017-03-30	87

codecademy

Calculating overall churn rate by month

CodeFlix management asks us to look into subscription churn rates.

- **Churn rate** is the percent of subscribers that have canceled within a certain period.
- To calculate churn rate we need to follow these rules

1. Cancellations in a given month are determined by the subscription end date, while active users are determined by both start and end date
2. By using the earlier "**SELECT * FROM"** subscriptions query, we see 4 fields exist
3. We need data table to show which users are active and who cancel subscription each month
4. We need to "**SUM**" the active and "**SUM**" the canceled users each month

$$\frac{\text{cancellations}}{\text{total subscribers}}$$

codecademy

- As we have the same volume of unique users, the churn rate calculation will weigh normally

```
SELECT segment,
COUNT(DISTINCT Id) AS 'Count'
FROM subscriptions
GROUP BY 1;
```

Query Results

Earliest_Start	Max_Sub_End	Max_Sub_Start	segment
2016-12-01	2017-03-31	2017-03-30	30
2016-12-01	2017-03-31	2017-03-30	87

- Next we will create the temporary table called “**months**”. With the help of “**UNION**” statement, we can build a table with the first and last days of each month we desire.

```
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),
```

- Now by using a “**CROSS JOIN**” statement, we append each month in table “**months**” to each row in “**subscriptions**”. This will help us see which months the user is active and where the subscription is canceled

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

- Using the “**cross_join**” table, we will create another table called “**status**”, which will help us identify who are active and who are canceled each month.

1. “**CASE WHEN**” statements can utilize multiple criteria, that, when true, will be in the first “**THEN**” statement value, and if false, will result in the “**ELSE**” statement value.

```
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
),
```

- Using table “**status**”, we will create another table “**status-aggregate**”, which will help us “**SUM**” the active and canceled users each month for each segment

```
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 1
)
```

- Now we can calculate monthly churn rate by dividing the total canceled users by the total active users for each month

```
SELECT month,
((status_aggregate.sum_canceled_87*1.0) /
(status_aggregate.sum_active_87*1.0)) AS
'Seg_87_Churn',

((status_aggregate.sum_canceled_30*1.0) /
(status_aggregate.sum_active_30*1.0)) AS
'Seg_30_Churn'
FROM status_aggregate;
```

Query Results

Earliest_Start	Max_Sub_End	Max_Sub_Start	segment
2016-12-01	2017-03-31	2017-03-30	30
2016-12-01	2017-03-31	2017-03-30	87

Compare the churn rates between segments

By using a "**SUM**" statement with the "**status-aggregate**" table, we effectively add all the cancellations for each month

```
SELECT
((SUM(status_aggregate.sum_canceled_87)*1.0) /
 (SUM(status_aggregate.sum_active_87)*1.0)) AS
 'Overall Churn 87',

((SUM(status_aggregate.sum_canceled_30)*1.0) /
 (SUM(status_aggregate.sum_active_30)*1.0)) AS
 'Overall Churn 30'
FROM status_aggregate;
```

- Overall churn rate for segment **87 = 37%**
- Overall churn rate for segment **30 = 9%**
- Monthly churn rate for segment 87 grows to a maximum of **49%**, while segment 30's maximum is **12%**
- The worst month for CodeFlix was **April 2017**, because both segments were at their highest churn rate

Query Results

Overall Churn 87	Overall Churn 30
0.374508261211644	0.0944262295081967

codecademy

Recommendations

1. CodeFlix needs to focus on expanding segment 30
2. CodeFlix should investigate February 2017’s video stream offers, because April 2017 was the worth month in the service history

codecademy

Thank you very much!