codecademy

Codeflix: Churn Rates

Learn SQL from Scratch

Table of Contents

1. Codeflix Overview

Subscriber Segments and Subscription Details

1.1 Codeflix Subscription Owner Segments

Based on the first 100 rows in the subscription table, there are two subscription owner segments at Codeflix

- Segment 30
- Segment 87

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

SAMPLE Output:

Query Results

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

1.2 Range of Subscription Data

Available subscription data ranges from December 2016 – March 2017. However, churn can only be calculated for the first three months of 2017.

- January 2017
- February 2017
- March 2017

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

Query Results	
MIN (subscription_start)	MAX (subscription_start)
2016-12-01	2017-03-30

2. Calculate Churn Rates

Procedures for Calculating Churn Rates by Subscriber Segment

2.1 Temporary ‘months’ Table

To prepare to calculate churn rates for January – March 2017, a temporary ‘months’ table is first created to include these months. This table defines the first and last day of each month.

Output from ‘months’ Table:

Query Results	
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;
```

2.2 Temporary ‘cross_join’ Table

Next, a temporary ‘cross_join’ table is created to join the ‘months’ and ‘subscriptions’ tables.

This table includes all fields from both tables.

SAMPLE Output from ‘cross_join’ table:

Query Results

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

```
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;
```

2.3 Temporary ‘status’ Table – Including ‘is_canceled’ Fields

A temporary ‘status’ table is also created to identify the months in which a subscription was active, by subscriber segment. This table includes the following fields:

- id
- month
- is_active_87: Indicates whether the subscription, for segment 87, was active during the month
- is_active_30: Indicates whether the subscription, for segment 30, was active during the month
- is_canceled_87: Indicates whether the subscription, for segment 87, was canceled during the month
- is_canceled_30: Indicates whether the subscription, for segment 30, as canceled during the month

SAMPLE Output from ‘status’ table:

Query Results

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

Code for Temporary ‘status’ Table (Including ‘is_canceled’ Fields)

```
status AS
(SELECT id, first_day as month,
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)
SELECT *
FROM status;
```

2.4 Temporary ‘status_aggregate’ Table

Building on the data from the ‘status’ table, a temporary ‘status_aggregate’ table is created to aggregate the total active and canceled subscriptions for each segment, by month. For each month, the following are calculated:

- sum_active_87: Total active subscriptions for segment 87
- sum_active_30: Total active subscriptions for segment 30
- sum_canceled_87: Total canceled subscriptions for segment 87
- sum_canceled_30: Total canceled subscriptions for segment 30

Output from ‘status_aggregate’ table:

Query Results				
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

Code for Temporary ‘status_aggregate’ Table

```
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;
```

2.5 Calculate Churn Rate

Finally, the churn rate for each segment is calculated, by month.

For segment, the total canceled subscriptions for the month is divided by the total active subscriptions for the month. This value should be multiplied by 1.0 to cast the result as a float.

Output for Churn Rate:

Query Results		
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

Code to calculate churn rates

```
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;
```

3. Analysis of Churn Rates

Evaluation of Churn Trends by Segment

3.1 Compare Churn Trends

While churn rates for both segments were trending up through March 2017, subscriber segment 30 consistently carries an overall lower churn rate. Codeflix should focus on expanding this segment. With lower churn rates, it will be much easier for Codeflix to grow the overall subscriber base for this segment than for segment 87.

Query Results		
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

4. Bonus Question

4.1 Bonus Question

Question: How would you modify this code to support a large number of segments?

Answer: To avoid hardcoding each segment number into the query, a separate temporary table could be created to identify distinct segments. This table would then be incorporated into the code so that each calculation would be done for every distinct segment identified within this temporary table.