www.8weeksqlchallenge.com

SQL CaseStudy Week#3

FOODIE-FI

AVO GOOD TIME

Satya Ranjan Ray

Introduction

Subscription based businesses are super popular and **Danny** realized that there was a large gap in the market - he wanted to create a **new streaming service** that only had food related content - something like Netflix but with only **cooking shows!**

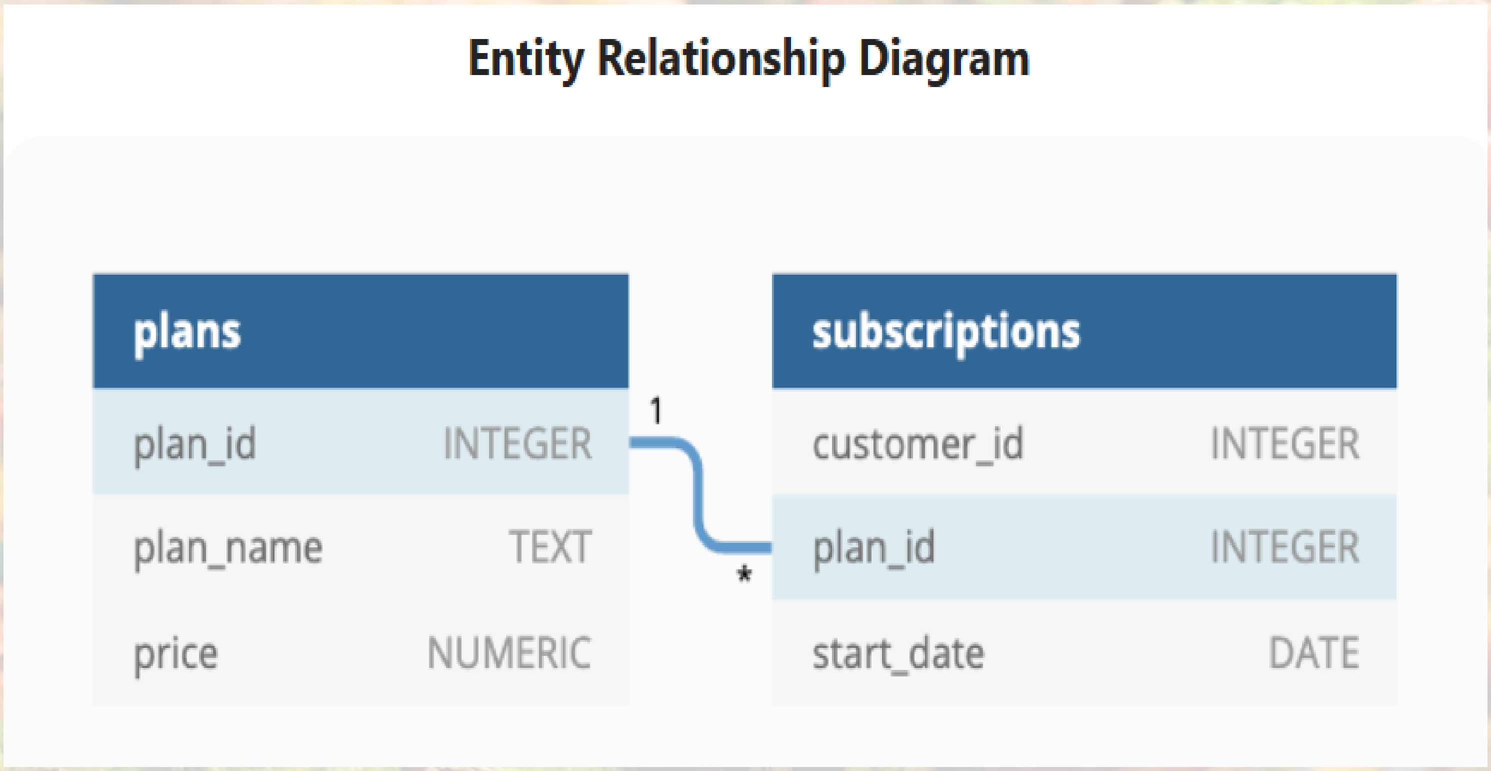
Danny finds a few smart friends to launch his new startup **Foodie-Fi** in **2020** and started selling monthly and annual subscriptions, giving their customers unlimited on-demand access to exclusive food videos from around the world!

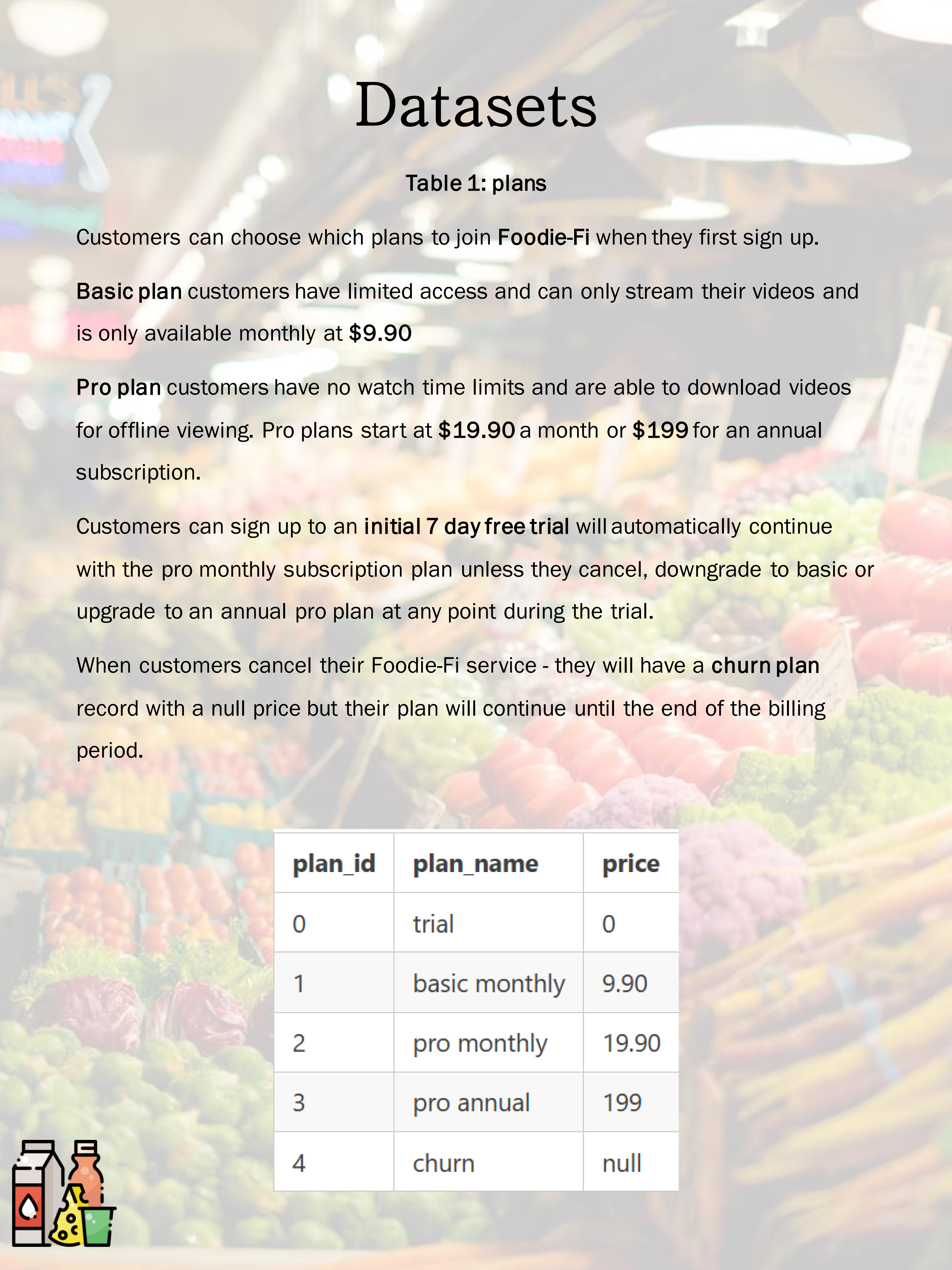
Danny created **Foodie-Fi** with a data driven mindset and wanted to ensure all future investment decisions and new features were decided using data. This case study focuses on using subscription style digital data to answer important business questions.

Problem Statement

Danny has shared the data design for **Foodie-Fi** and also short descriptions on each of the database tables - our case study focuses on only 2 tables but there will be a challenge to create a new table for the Foodie-Fi team.

All datasets exist within the **foodie_fi** database schema - be sure to include this reference within your **SQL** scripts as you start exploring the data and answering the case study questions.





Datasets

Table 1: plans

Customers can choose which plans to join **Foodie-Fi** when they first sign up.

Basic plan customers have limited access and can only stream their videos and is only available monthly at **$9.90**

Pro plan customers have no watch time limits and are able to download videos for offline viewing. Pro plans start at **$19.90** a month or **$199** for an annual subscription.

Customers can sign up to an **initial 7 day free trial** will automatically continue with the pro monthly subscription plan unless they cancel, downgrade to basic or upgrade to an annual pro plan at any point during the trial.

When customers cancel their Foodie-Fi service - they will have a **churn plan** record with a null price but their plan will continue until the end of the billing period.

plan_id	plan_name	price
0	trial	0
1	basic monthly	9.90
2	pro monthly	19.90
3	pro annual	199
4	churn	null

Table 2: subscriptions

Customer subscriptions show the exact date where their specific plan_id starts.

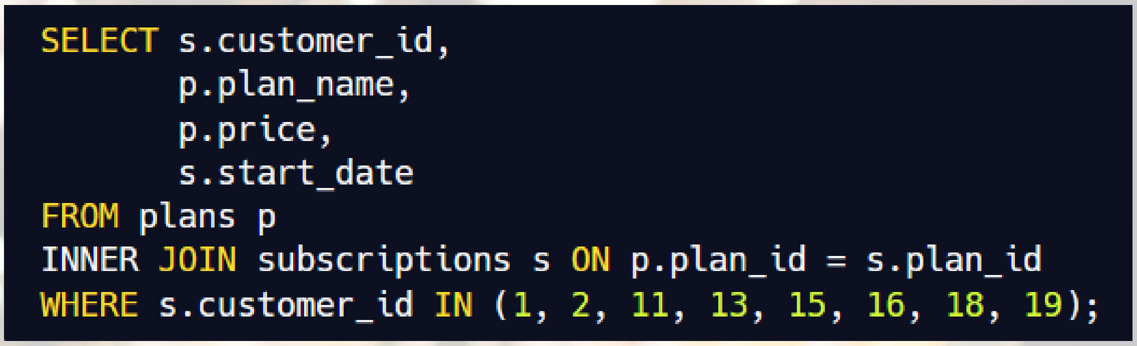
If customers downgrade from a pro plan or cancel their subscription - the higher plan will remain in place until the period is over - the start_date in the subscriptions table will reflect the date that the actual plan changes.

When customers upgrade their account from a basic plan to a pro or annual pro plan - the higher plan will take effect straightaway. When customers churn - they will keep their access until the end of their current billing period but the start_date will be technically the day they decided to cancel their service.

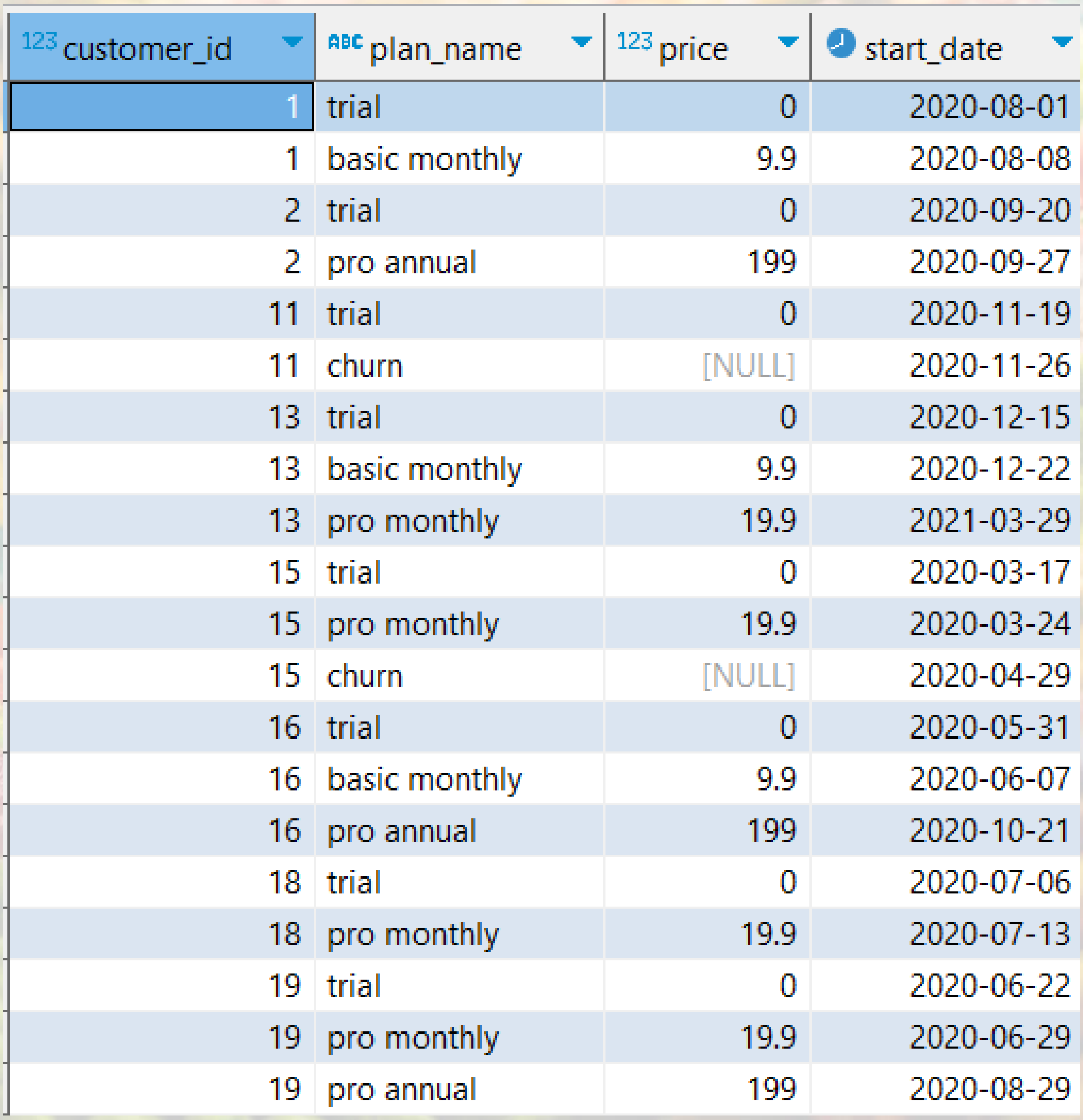
customer_id	plan_id	start_date
1	0	2020-08-01
1	1	2020-08-08
2	0	2020-09-20
2	3	2020-09-27
11	0	2020-11-19
11	4	2020-11-26
13	0	2020-12-15
13	1	2020-12-22
13	2	2021-03-29
15	0	2020-03-17
15	2	2020-03-24
15	4	2020-04-29
16	0	2020-05-31
16	1	2020-06-07
16	3	2020-10-21
18	0	2020-07-06
18	2	2020-07-13
19	0	2020-06-22
19	2	2020-06-29
19	3	2020-08-29

Customer Journey

Based off the 8 sample customers provided in the sample from the subscriptions table, write a brief description about each customer's onboarding journey.



```sql
SELECT s.customer_id,
       p.plan_name,
       p.price,
       s.start_date
FROM plans p
INNER JOIN subscriptions s ON p.plan_id = s.plan_id
WHERE s.customer_id IN (1, 2, 11, 13, 15, 16, 18, 19);
```



customer_id	plan_name	price	start_date
1	trial	0	2020-08-01
1	basic monthly	9.9	2020-08-08
2	trial	0	2020-09-20
2	pro annual	199	2020-09-27
11	trial	0	2020-11-19
11	churn	[NULL]	2020-11-26
13	trial	0	2020-12-15
13	basic monthly	9.9	2020-12-22
13	pro monthly	19.9	2021-03-29
15	trial	0	2020-03-17
15	pro monthly	19.9	2020-03-24
15	churn	[NULL]	2020-04-29
16	trial	0	2020-05-31
16	basic monthly	9.9	2020-06-07
16	pro annual	199	2020-10-21
18	trial	0	2020-07-06
18	pro monthly	19.9	2020-07-13
19	trial	0	2020-06-22
19	pro monthly	19.9	2020-06-29
19	pro annual	199	2020-08-29

- Customer 1 signed up for a free trial of Foodie-Fi on August 1, 2000 and later subscribed to the Basic Monthly Plan on August 8, 2020 for a fee of $9.90
- Customer 2 signed up for a free trial of Foodie-Fi on September 20, 2020 and later subscribed to the Pro Annual Plan on September 27, 2020 for a fee of $199
- Customer 11 signed up for a free trial of Foodie-Fi on November 19, 2020 and cancelled the service on November 26, 2020, right after the end of the 7-day free trial
- Customer 13 started a free trial of Foodie-Fi on December 15, 2020. After the trial period ended, they subscribed to the Basic Monthly Plan on December 22, 2020 for a fee of $9.90 Later, on March 29, 2021, they upgraded to the Pro Monthly Plan for a price of $19.90
- Customer 15 started a free trial of Foodie-Fi on March 17, 2020. After the trial period ended, they subscribed to the Pro Monthly Plan on March 24, 2020 for a fee of $19.90 However, they cancelled the service on April 29, 2020, after using it for a month
- Customer 16 started a free trial of Foodie-Fi on May 31, 2020. After the trial period ended, they subscribed to the Basic Monthly Plan on June 7, 2020 for a fee of $9.90 Later, on October 21, 2020, they upgraded to the Pro Annual Plan for a price of $199.00
- Customer 18 started a free trial of Foodie-Fi on July 6, 2020. After the trial period ended, they subscribed to the Pro Monthly Plan on July 13, 2020 for a fee of $19.90
- Customer 19 started a free trial of Foodie-Fi on June 22, 2020. After the trial period ended, they subscribed to the Pro Monthly Plan on June 29, 2020 for a fee of $19.90. Later, on August 29, 2020, they upgraded to the Pro Annual Plan for a price of $199.00

Data Analysis Questions

Q1: How many customers has Foodie-Fi ever had?

```
SELECT COUNT(DISTINCT customer_id) AS unique_customers
FROM subscriptions;
```

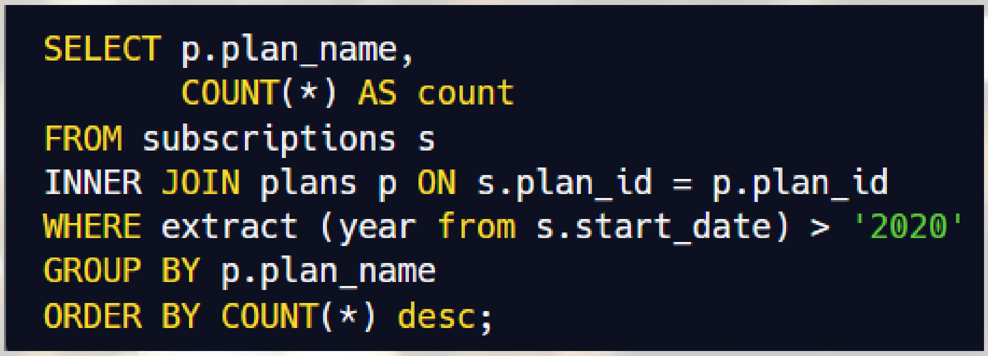
unique_customers
1,000

Q2: What is the monthly distribution of trial plan start_date values for our dataset

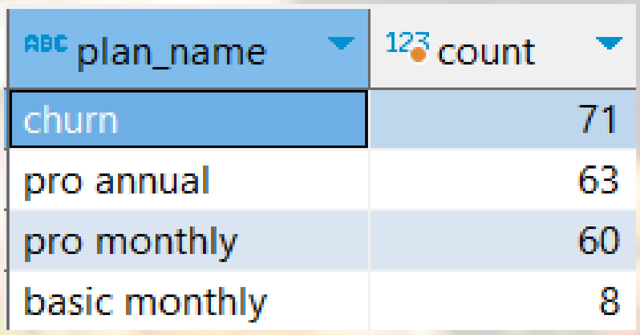
```
SELECT EXTRACT(month from s.start_date) AS month,
       TO_CHAR(s.start_date, 'Month') AS month_name,
       COUNT(*) AS trial_count
FROM subscriptions s
INNER JOIN plans p ON s.plan_id = p.plan_id
WHERE p.plan_name = 'trial'
GROUP BY month,month_name
ORDER BY month
```

month	month_name	trial_count
1	January	88
2	February	68
3	March	94
4	April	81
5	May	88
6	June	79
7	July	89
8	August	88
9	September	87
10	October	79
11	November	75
12	December	84

Q3: What plan start_date values occur after the year 2020 for our dataset?

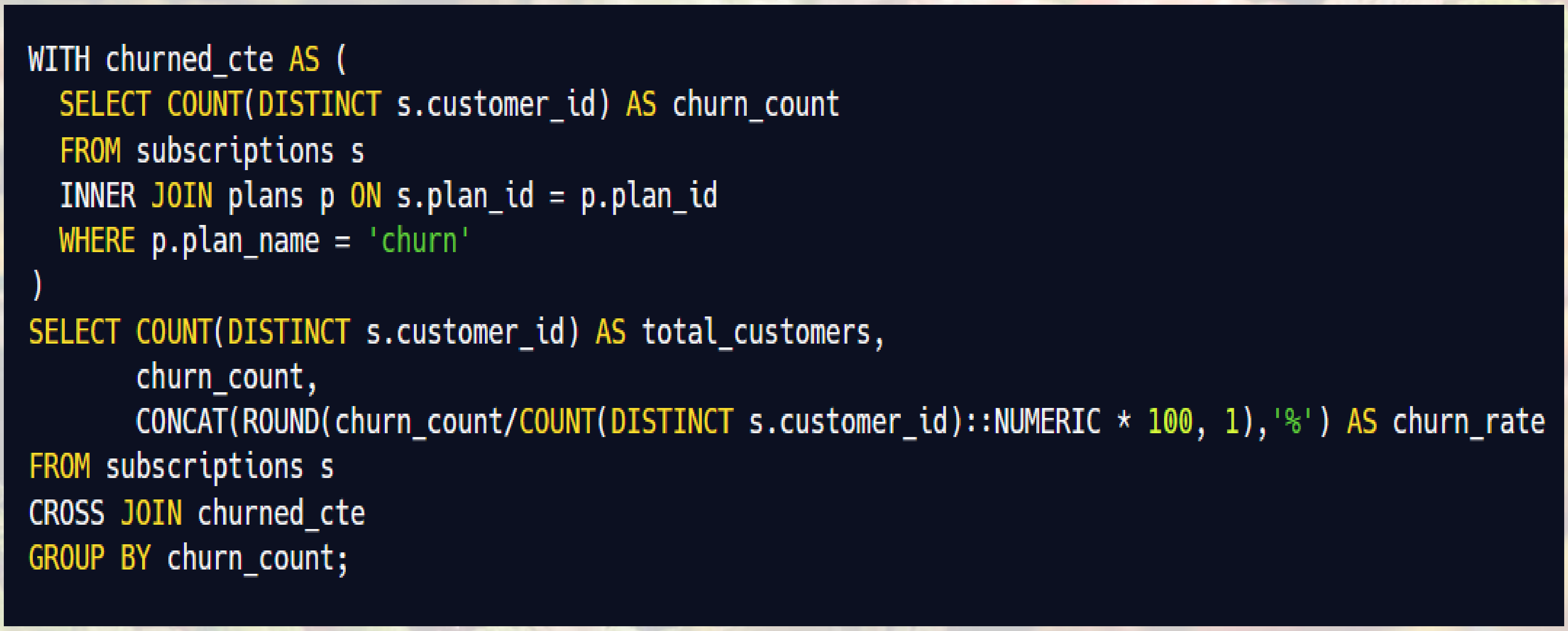


```
SELECT p.plan_name,
       COUNT(*) AS count
FROM subscriptions s
INNER JOIN plans p ON s.plan_id = p.plan_id
WHERE extract (year from s.start_date) > '2020'
GROUP BY p.plan_name
ORDER BY COUNT(*) desc;
```

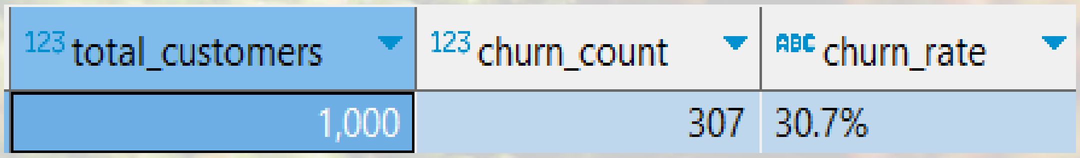


plan_name	count
churn	71
pro annual	63
pro monthly	60
basic monthly	8

Q4: What is the customer count and percentage of customers who have churned rounded to 1 decimal place?



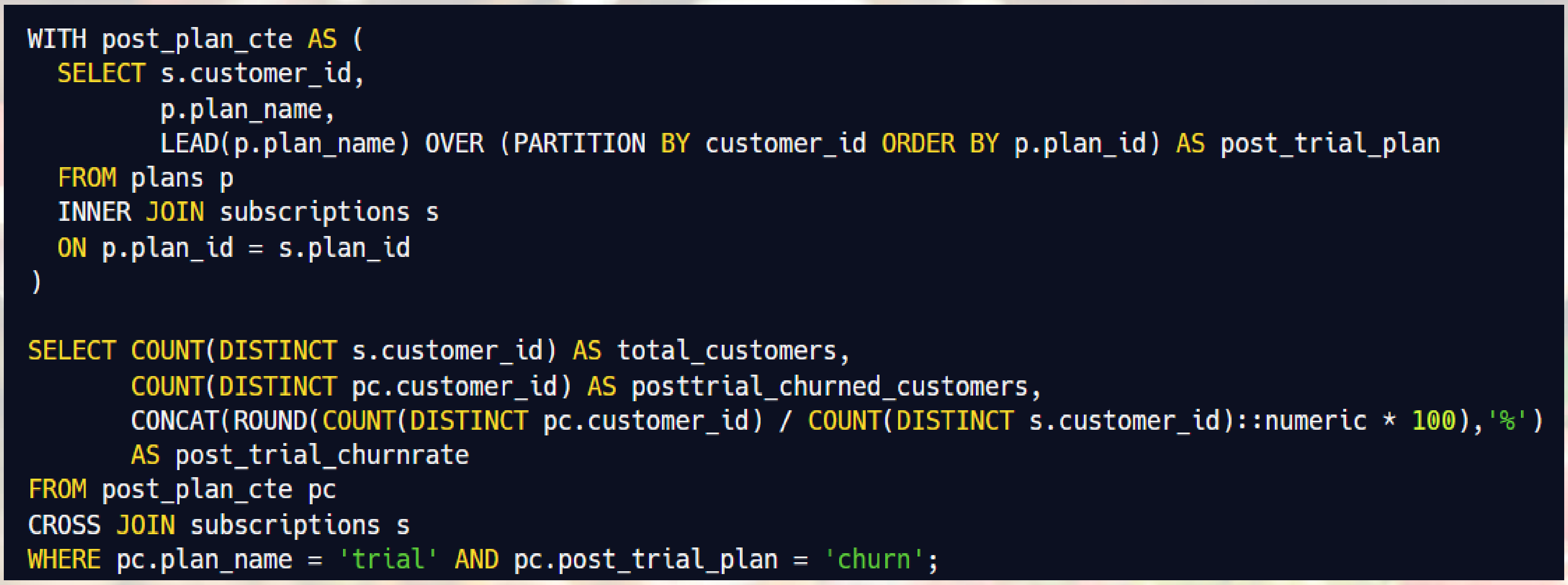
```
WITH churned_cte AS (
  SELECT COUNT(DISTINCT s.customer_id) AS churn_count
  FROM subscriptions s
  INNER JOIN plans p ON s.plan_id = p.plan_id
  WHERE p.plan_name = 'churn'
)
SELECT COUNT(DISTINCT s.customer_id) AS total_customers,
       churn_count,
       CONCAT(ROUND(churn_count/COUNT(DISTINCT s.customer_id)::NUMERIC * 100, 1),'%') AS churn_rate
FROM subscriptions s
CROSS JOIN churned_cte
GROUP BY churn_count;
```



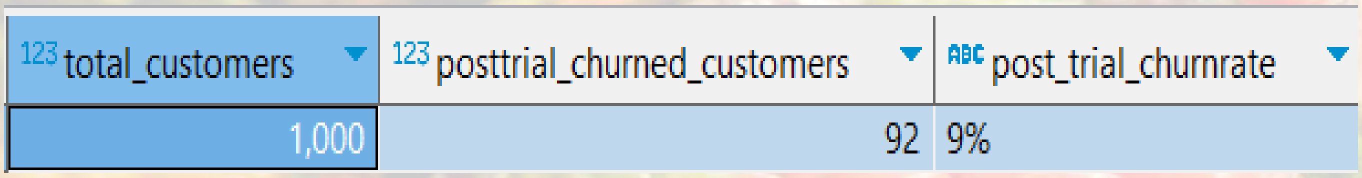
total_customers	churn_count	churn_rate
1,000	307	30.7%

Q5: How many customers have churned straight after their initial free trial

What percentage is this rounded to the nearest whole number?

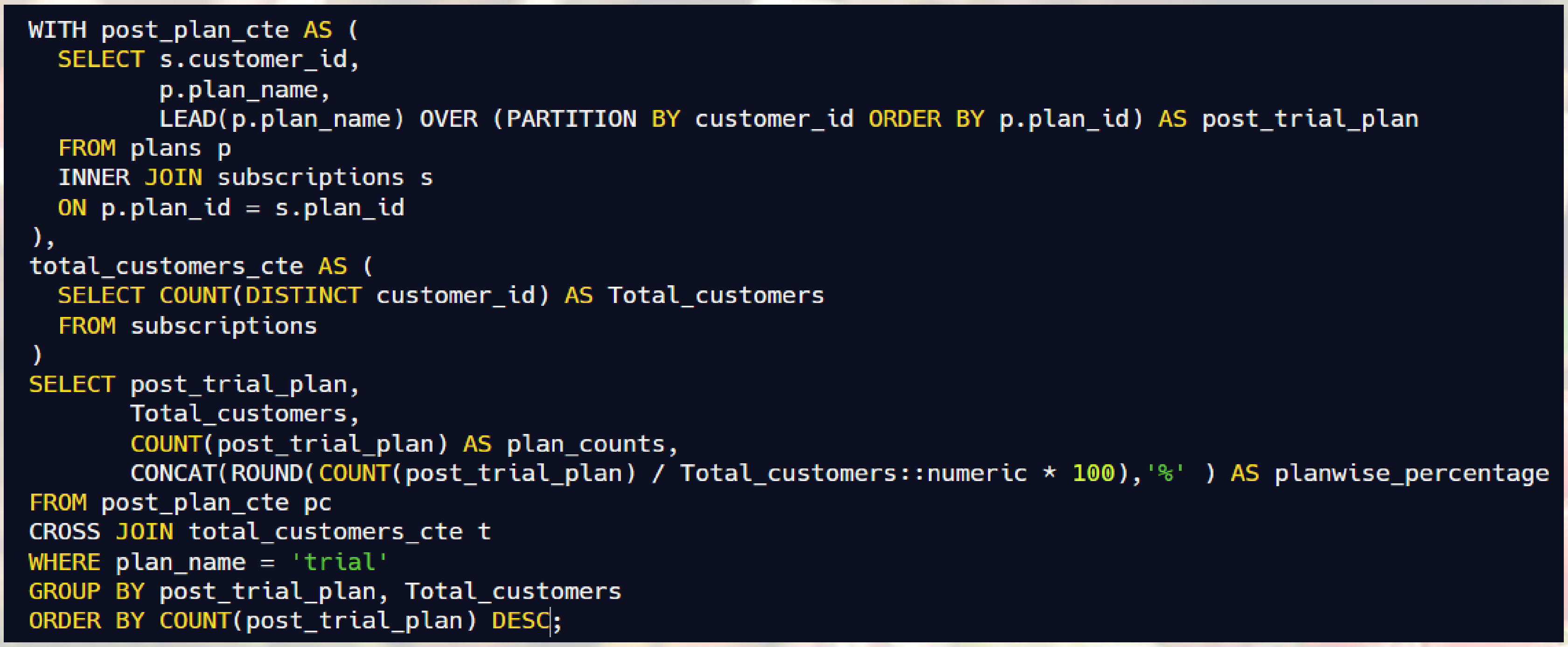


```sql
WITH post_plan_cte AS (
  SELECT s.customer_id,
         p.plan_name,
         LEAD(p.plan_name) OVER (PARTITION BY customer_id ORDER BY p.plan_id) AS post_trial_plan
  FROM plans p
  INNER JOIN subscriptions s
  ON p.plan_id = s.plan_id
)

SELECT COUNT(DISTINCT s.customer_id) AS total_customers,
       COUNT(DISTINCT pc.customer_id) AS posttrial_churned_customers,
       CONCAT(ROUND(COUNT(DISTINCT pc.customer_id) / COUNT(DISTINCT s.customer_id)::numeric * 100),'%')
       AS post_trial_churnrate
FROM post_plan_cte pc
CROSS JOIN subscriptions s
WHERE pc.plan_name = 'trial' AND pc.post_trial_plan = 'churn';
```

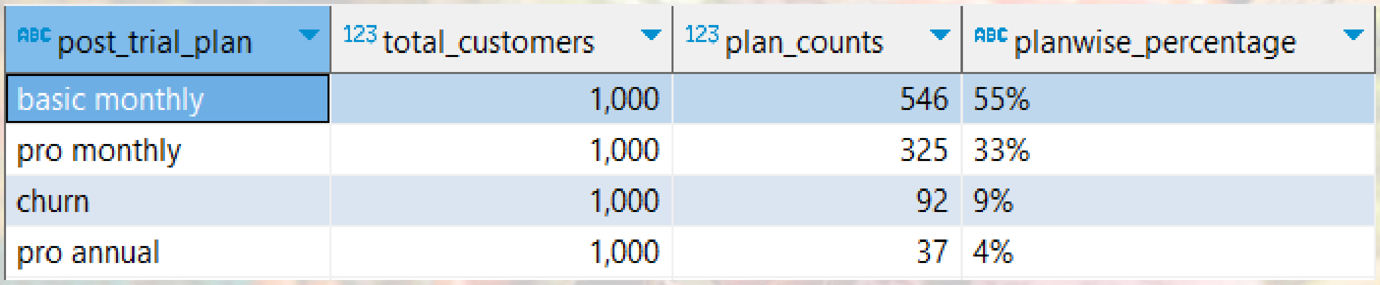


123 total_customers	123 posttrial_churned_customers	ABC post_trial_churnrate
1,000	92	9%

Q6: What is the number and percentage of customer plans after their initial free trial?

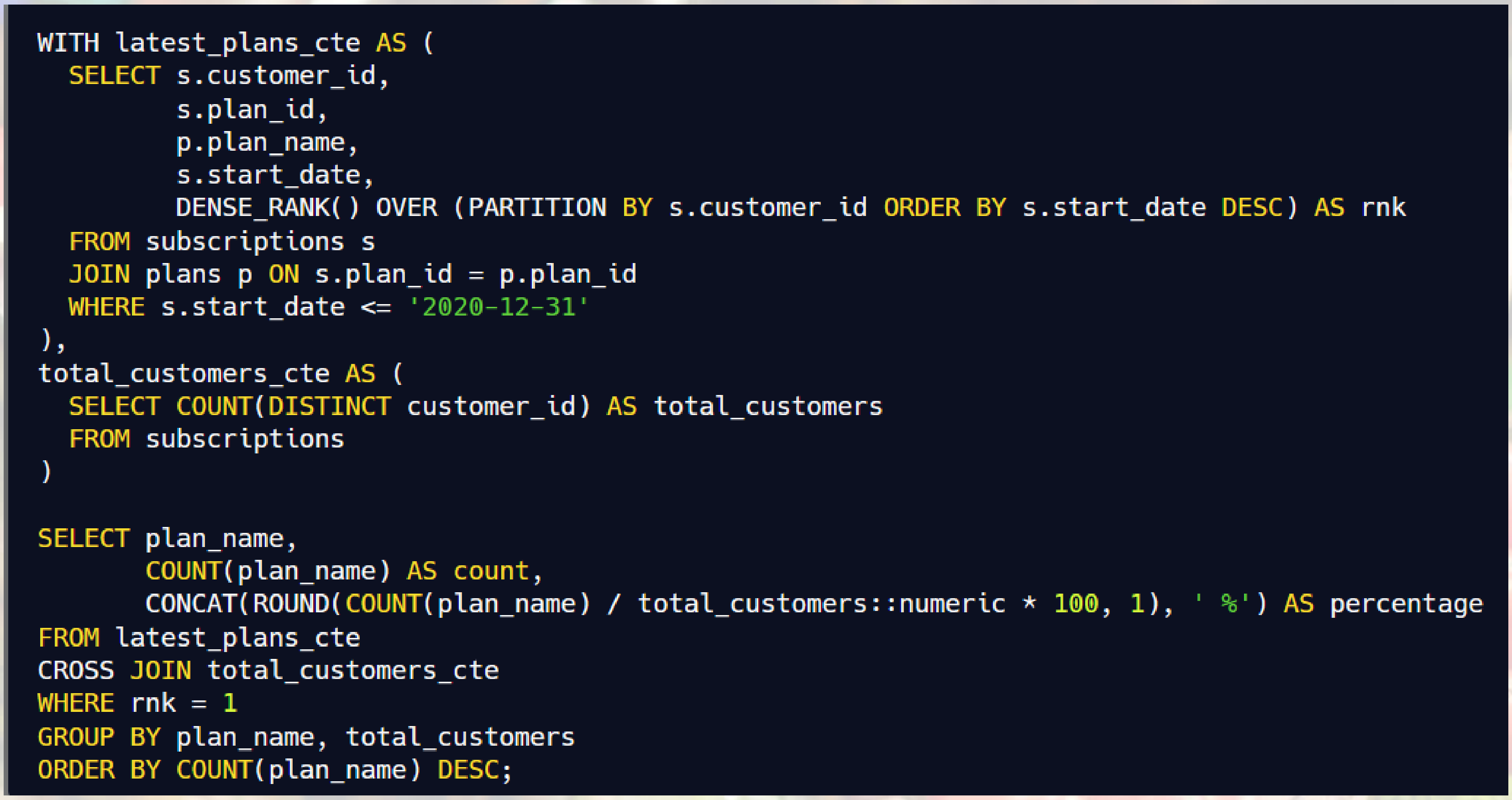


```sql
WITH post_plan_cte AS (
  SELECT s.customer_id,
         p.plan_name,
         LEAD(p.plan_name) OVER (PARTITION BY customer_id ORDER BY p.plan_id) AS post_trial_plan
  FROM plans p
  INNER JOIN subscriptions s
  ON p.plan_id = s.plan_id
),
total_customers_cte AS (
  SELECT COUNT(DISTINCT customer_id) AS Total_customers
  FROM subscriptions
)
SELECT post_trial_plan,
       Total_customers,
       COUNT(post_trial_plan) AS plan_counts,
       CONCAT(ROUND(COUNT(post_trial_plan) / Total_customers::numeric * 100),'%' ) AS planwise_percentage
FROM post_plan_cte pc
CROSS JOIN total_customers_cte t
WHERE plan_name = 'trial'
GROUP BY post_trial_plan, Total_customers
ORDER BY COUNT(post_trial_plan) DESC;
```

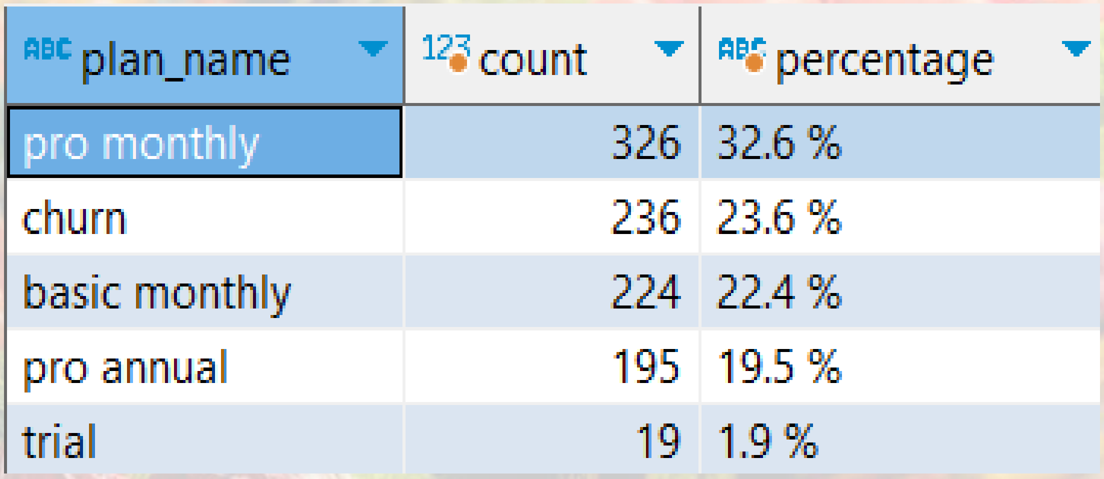


post_trial_plan	total_customers	plan_counts	planwise_percentage
basic monthly	1,000	546	55%
pro monthly	1,000	325	33%
churn	1,000	92	9%
pro annual	1,000	37	4%

Q7: What is the customer count and percentage breakdown of all 5 plan_name values at 2020-12-31?



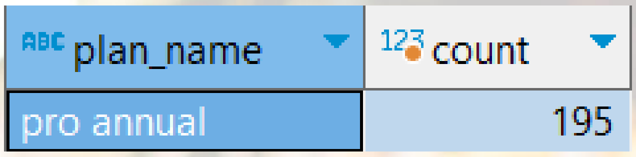
```
WITH latest_plans_cte AS (
  SELECT s.customer_id,
         s.plan_id,
         p.plan_name,
         s.start_date,
         DENSE_RANK() OVER (PARTITION BY s.customer_id ORDER BY s.start_date DESC) AS rnk
  FROM subscriptions s
  JOIN plans p ON s.plan_id = p.plan_id
  WHERE s.start_date <= '2020-12-31'
),
total_customers_cte AS (
  SELECT COUNT(DISTINCT customer_id) AS total_customers
  FROM subscriptions
)

SELECT plan_name,
       COUNT(plan_name) AS count,
       CONCAT(ROUND(COUNT(plan_name) / total_customers::numeric * 100, 1), ' %') AS percentage
FROM latest_plans_cte
CROSS JOIN total_customers_cte
WHERE rnk = 1
GROUP BY plan_name, total_customers
ORDER BY COUNT(plan_name) DESC;
```



plan_name	count	percentage
pro monthly	326	32.6 %
churn	236	23.6 %
basic monthly	224	22.4 %
pro annual	195	19.5 %
trial	19	1.9 %

Q8: How many customers have upgraded to an annual plan in 2020?

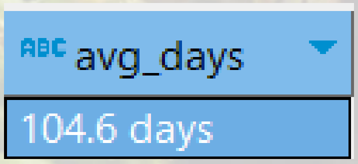
```
SELECT p.plan_name,
       COUNT(*) AS count
FROM plans p
INNER JOIN subscriptions s ON p.plan_id = s.plan_id
WHERE EXTRACT(YEAR FROM s.start_date) = '2020' AND p.plan_name = 'pro annual'
GROUP BY p.plan_name;
```



plan_name	count
pro annual	195

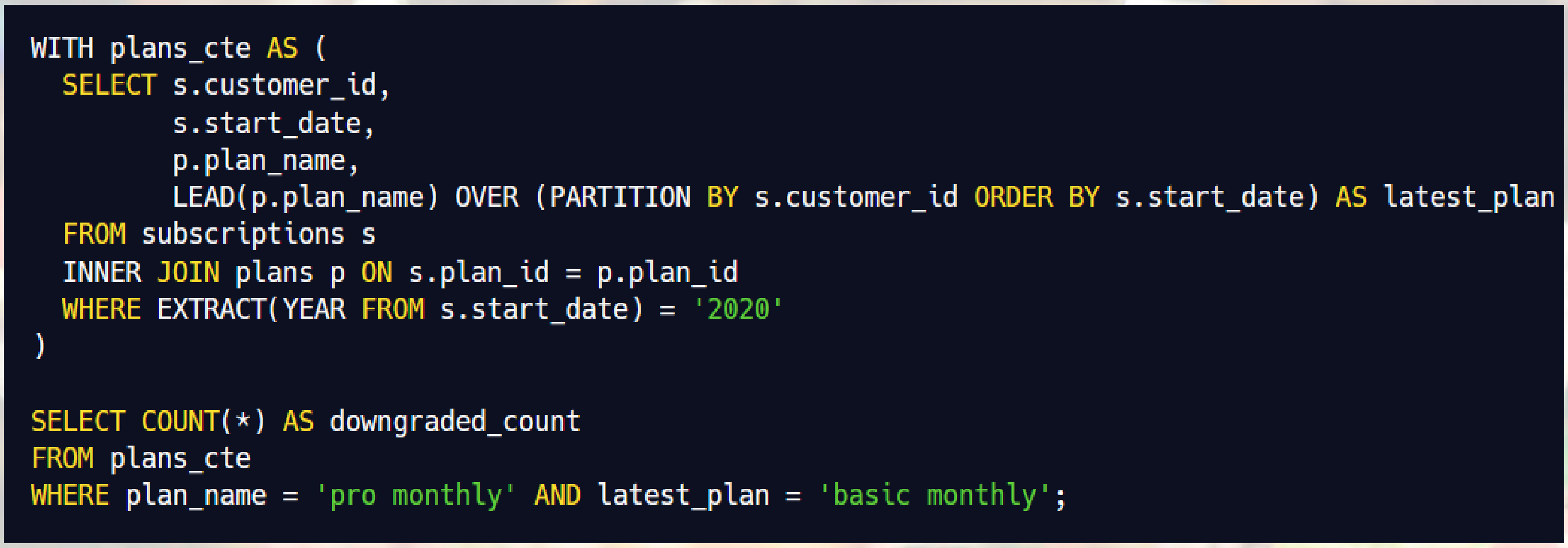
Q9: How many days on average does it take for a customer to an annual plan from the day they join Foodie-Fi?

```
WITH annual_plan_cte AS (
  SELECT s.customer_id
  FROM plans p
  INNER JOIN subscriptions s ON p.plan_id = s.plan_id
  WHERE p.plan_name = 'pro annual'
),
upgrade_date_cte AS (
  SELECT ap.customer_id,
         p.plan_name,
         s.start_date AS joining_date,
         LEAD(start_date) OVER (PARTITION BY s.customer_id ORDER BY
s.start_date) AS updated_date
  FROM annual_plan_cte ap
  INNER JOIN subscriptions s ON ap.customer_id = s.customer_id
  INNER JOIN plans p ON s.plan_id = p.plan_id
  WHERE p.plan_name IN ('trial', 'pro annual')
)

SELECT CONCAT(ROUND(AVG(updated_date - joining_date), 1), ' days') AS avg_days
FROM upgrade_date_cte;
```

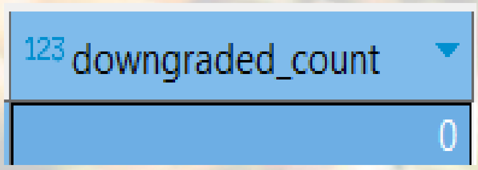


avg_days
104.6 days

Q10: How many customers downgraded from a pro monthly to a basic monthly plan in 2020?



```
WITH plans_cte AS (
  SELECT s.customer_id,
         s.start_date,
         p.plan_name,
         LEAD(p.plan_name) OVER (PARTITION BY s.customer_id ORDER BY s.start_date) AS latest_plan
  FROM subscriptions s
  INNER JOIN plans p ON s.plan_id = p.plan_id
  WHERE EXTRACT(YEAR FROM s.start_date) = '2020'
)

SELECT COUNT(*) AS downgraded_count
FROM plans_cte
WHERE plan_name = 'pro monthly' AND latest_plan = 'basic monthly';
```



downgraded_count
0

Insights

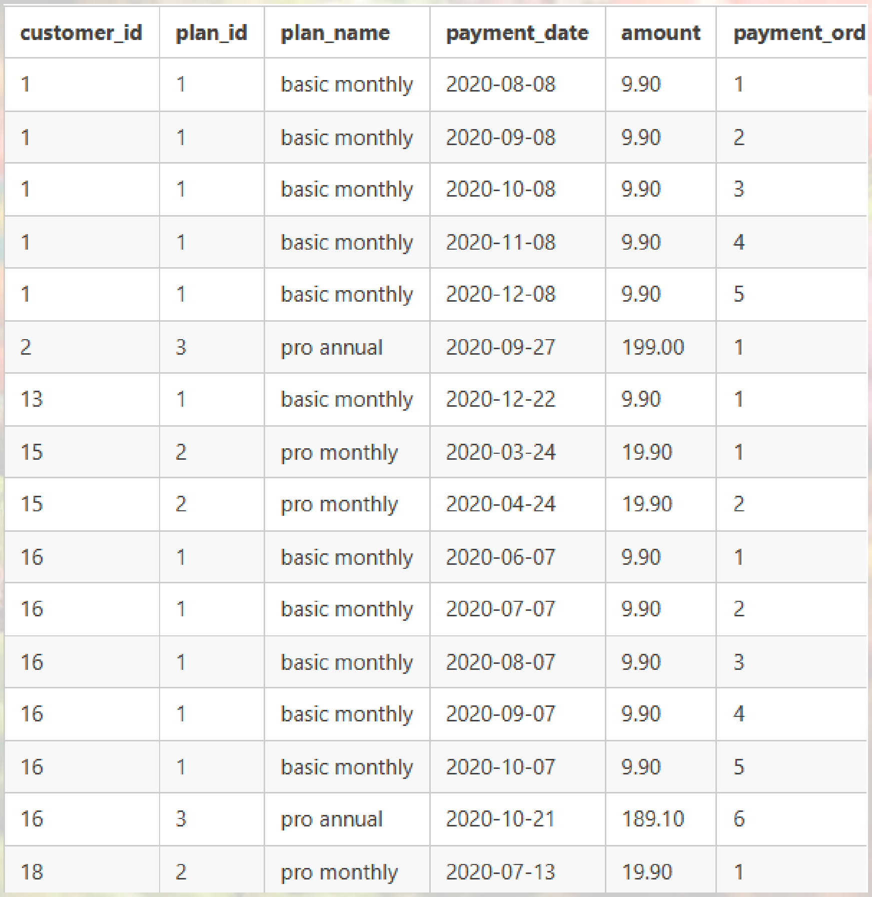
- Foodie Fi had the most free trial sign-ups in March.
- Regarding the customer churn rate, 30% of the total 1000 customers have churned among which 9% of the customers churned right after their free trial.
- 55% of Foodie Fi's free trial users decided to continue their subscription with the basic monthly plan.
- By the end of 2020, most Foodie Fi customers were on the pro monthly plan
- It took an average of 104 days for Foodie Fi customers to upgrade to an annual plan

Challenge Payment Question

The Foodie-Fi team wants you to create a new payments table for the year 2020 that includes amounts paid by each customer in the subscriptions table with the following requirements:

- Monthly payments always occur on the same day of month as the original start_date of any monthly paid plan
- upgrades from basic to monthly or pro plans are reduced by the current paid amount in that month and start immediately
- upgrades from pro monthly to pro annual are paid at the end of the current billing period and also starts at the end of the month period
- once a customer churns they will no longer make payments

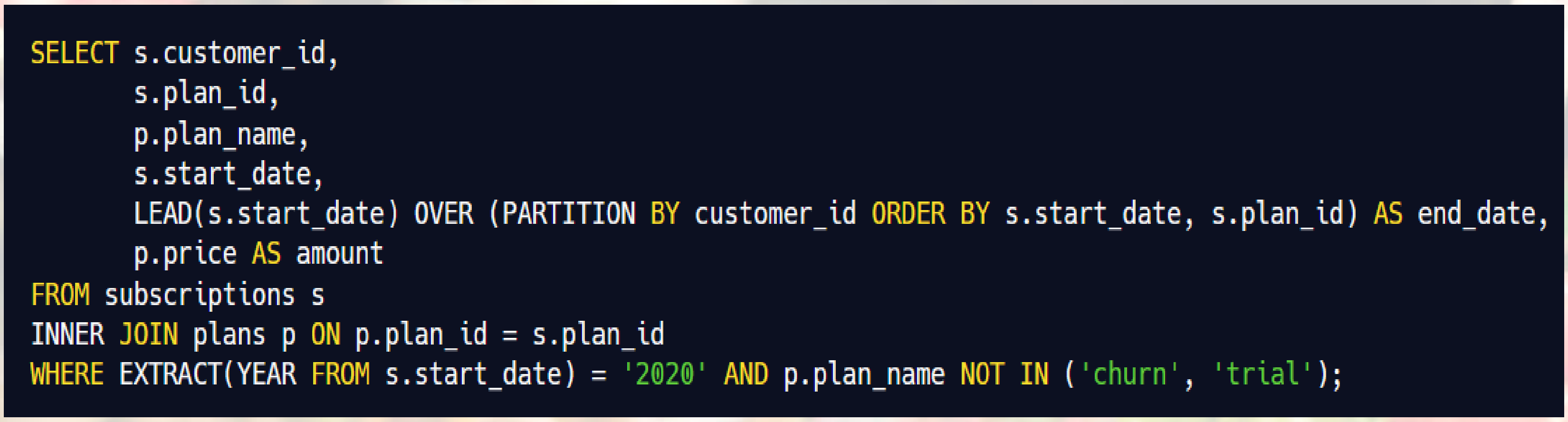
Example output



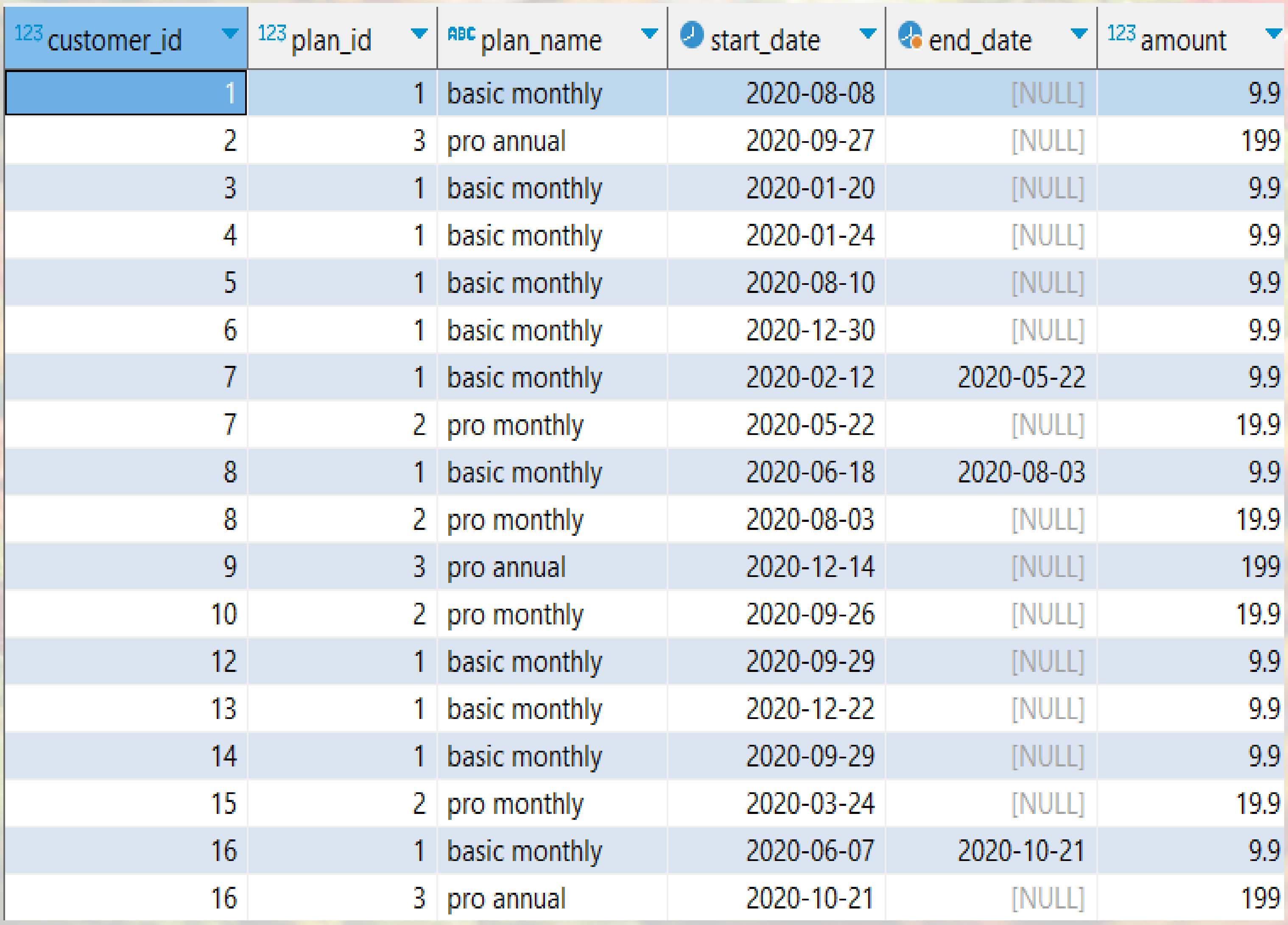
customer_id	plan_id	plan_name	payment_date	amount	payment_ord
1	1	basic monthly	2020-08-08	9.90	1
1	1	basic monthly	2020-09-08	9.90	2
1	1	basic monthly	2020-10-08	9.90	3
1	1	basic monthly	2020-11-08	9.90	4
1	1	basic monthly	2020-12-08	9.90	5
2	3	pro annual	2020-09-27	199.00	1
13	1	basic monthly	2020-12-22	9.90	1
15	2	pro monthly	2020-03-24	19.90	1
15	2	pro monthly	2020-04-24	19.90	2
16	1	basic monthly	2020-06-07	9.90	1
16	1	basic monthly	2020-07-07	9.90	2
16	1	basic monthly	2020-08-07	9.90	3
16	1	basic monthly	2020-09-07	9.90	4
16	1	basic monthly	2020-10-07	9.90	5
16	3	pro annual	2020-10-21	189.10	6
18	2	pro monthly	2020-07-13	19.90	1

To solve this challenge, we would do some steps

STEP 1: I filtered the subscriptions to only include those made in the year **2020.** I used the lead window function to calculate the end date for the next plan whenever it was available. I also excluded any plans labeled as **'trial'** or **'churn'** since no payments were made for these.

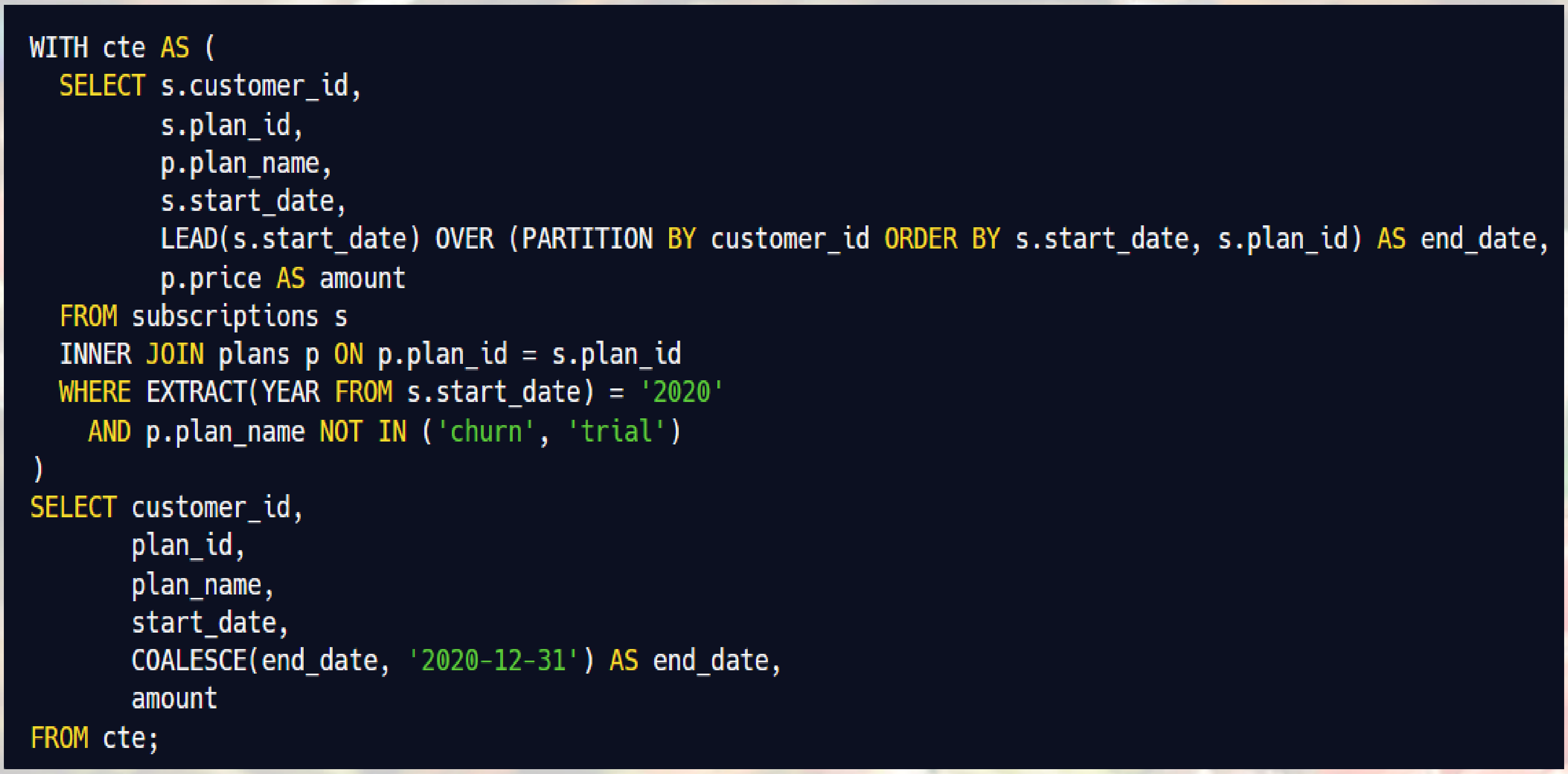


```sql
SELECT s.customer_id,
       s.plan_id,
       p.plan_name,
       s.start_date,
       LEAD(s.start_date) OVER (PARTITION BY customer_id ORDER BY s.start_date, s.plan_id) AS end_date,
       p.price AS amount
FROM subscriptions s
INNER JOIN plans p ON p.plan_id = s.plan_id
WHERE EXTRACT(YEAR FROM s.start_date) = '2020' AND p.plan_name NOT IN ('churn', 'trial');
```

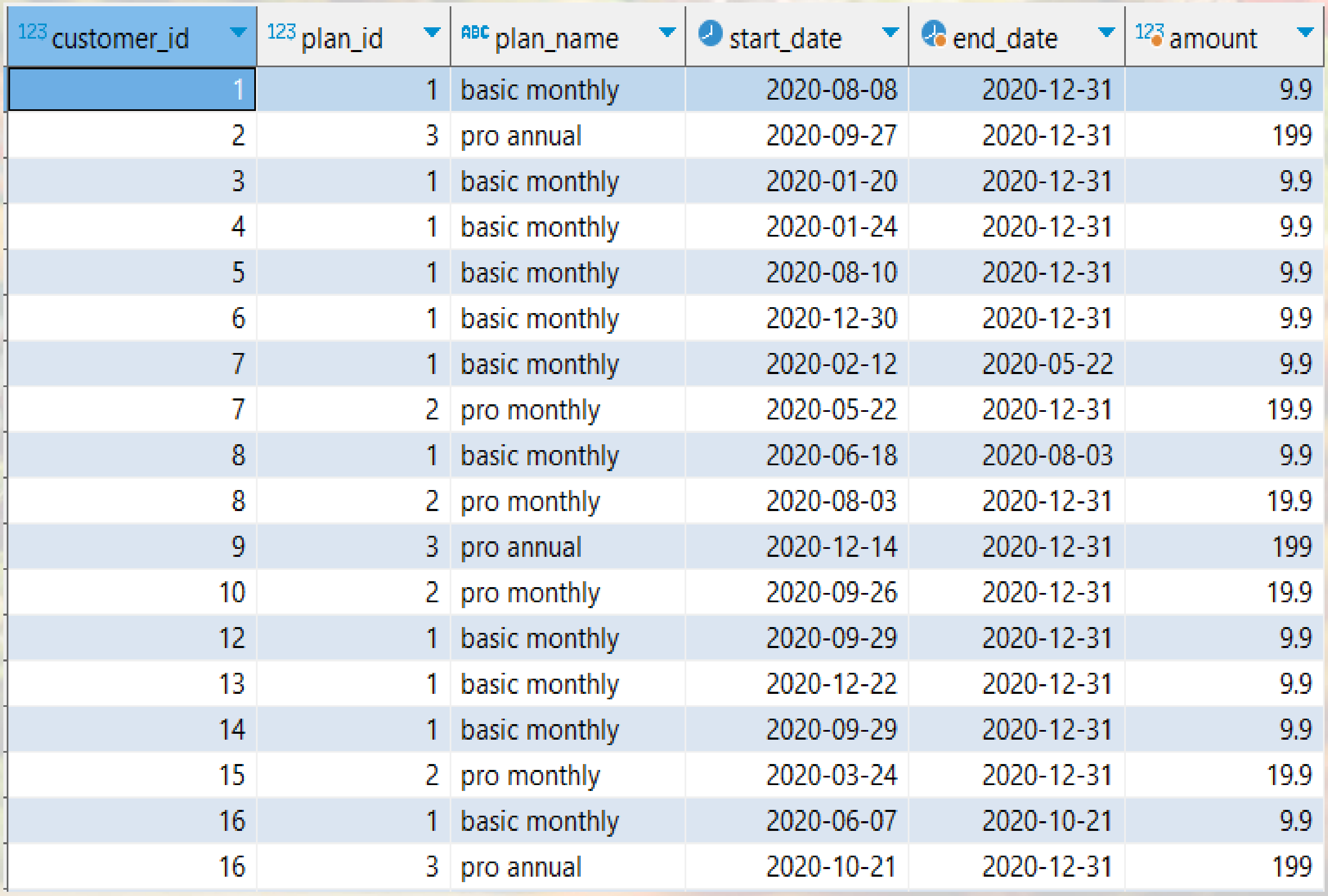


customer_id	plan_id	plan_name	start_date	end_date	amount
1	1	basic monthly	2020-08-08	[NULL]	9.9
2	3	pro annual	2020-09-27	[NULL]	199
3	1	basic monthly	2020-01-20	[NULL]	9.9
4	1	basic monthly	2020-01-24	[NULL]	9.9
5	1	basic monthly	2020-08-10	[NULL]	9.9
6	1	basic monthly	2020-12-30	[NULL]	9.9
7	1	basic monthly	2020-02-12	2020-05-22	9.9
7	2	pro monthly	2020-05-22	[NULL]	19.9
8	1	basic monthly	2020-06-18	2020-08-03	9.9
8	2	pro monthly	2020-08-03	[NULL]	19.9
9	3	pro annual	2020-12-14	[NULL]	199
10	2	pro monthly	2020-09-26	[NULL]	19.9
12	1	basic monthly	2020-09-29	[NULL]	9.9
13	1	basic monthly	2020-12-22	[NULL]	9.9
14	1	basic monthly	2020-09-29	[NULL]	9.9
15	2	pro monthly	2020-03-24	[NULL]	19.9
16	1	basic monthly	2020-06-07	2020-10-21	9.9
16	3	pro annual	2020-10-21	[NULL]	199

STEP-2: I replaced the null values in the 'end date' column with the last day of the year 2020, indicating that it was the last plan the user had for that year.

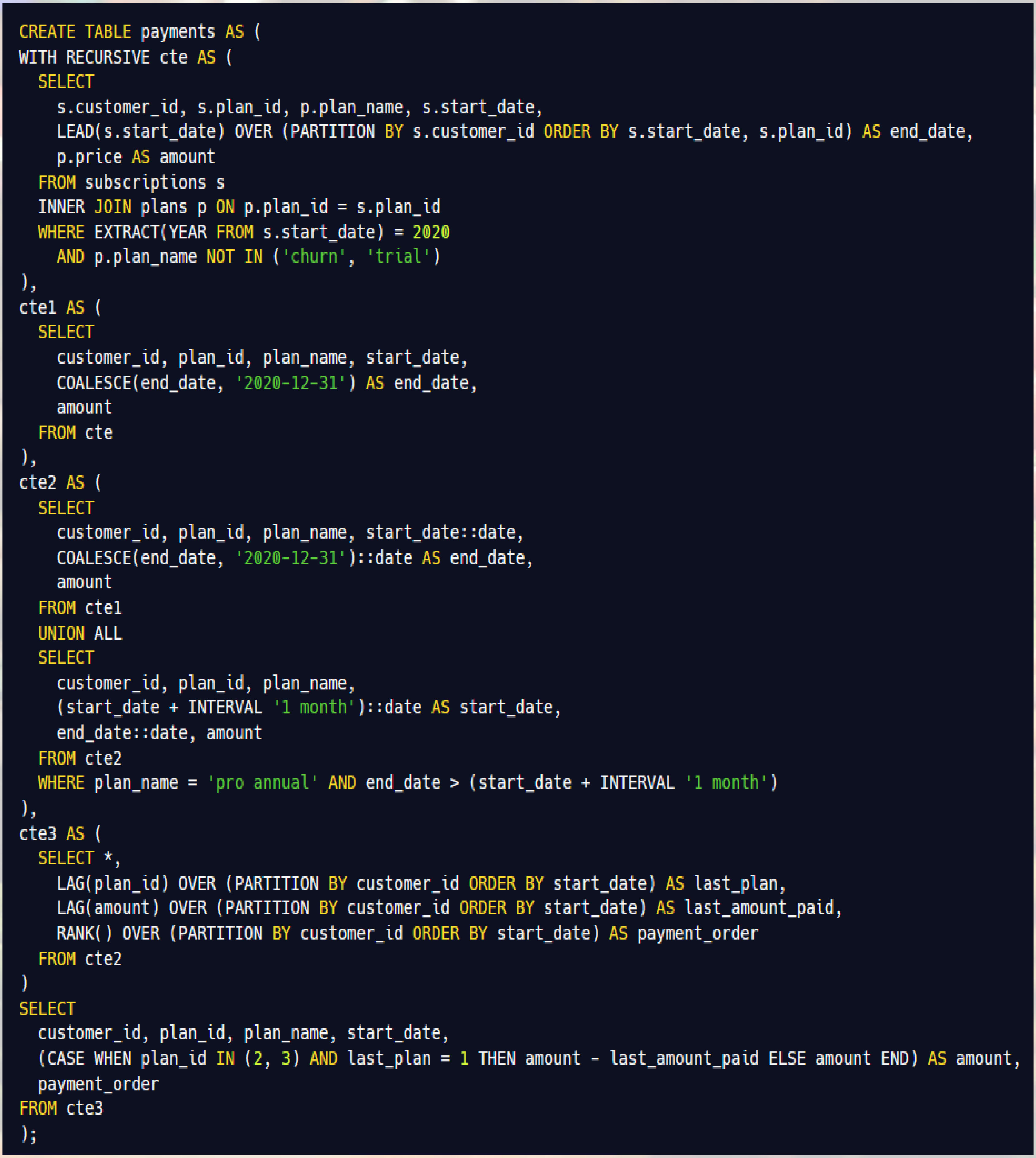


```sql
WITH cte AS (
  SELECT s.customer_id,
         s.plan_id,
         p.plan_name,
         s.start_date,
         LEAD(s.start_date) OVER (PARTITION BY customer_id ORDER BY s.start_date, s.plan_id) AS end_date,
         p.price AS amount
  FROM subscriptions s
  INNER JOIN plans p ON p.plan_id = s.plan_id
  WHERE EXTRACT(YEAR FROM s.start_date) = '2020'
    AND p.plan_name NOT IN ('churn', 'trial')
)
SELECT customer_id,
       plan_id,
       plan_name,
       start_date,
       COALESCE(end_date, '2020-12-31') AS end_date,
       amount
FROM cte;
```

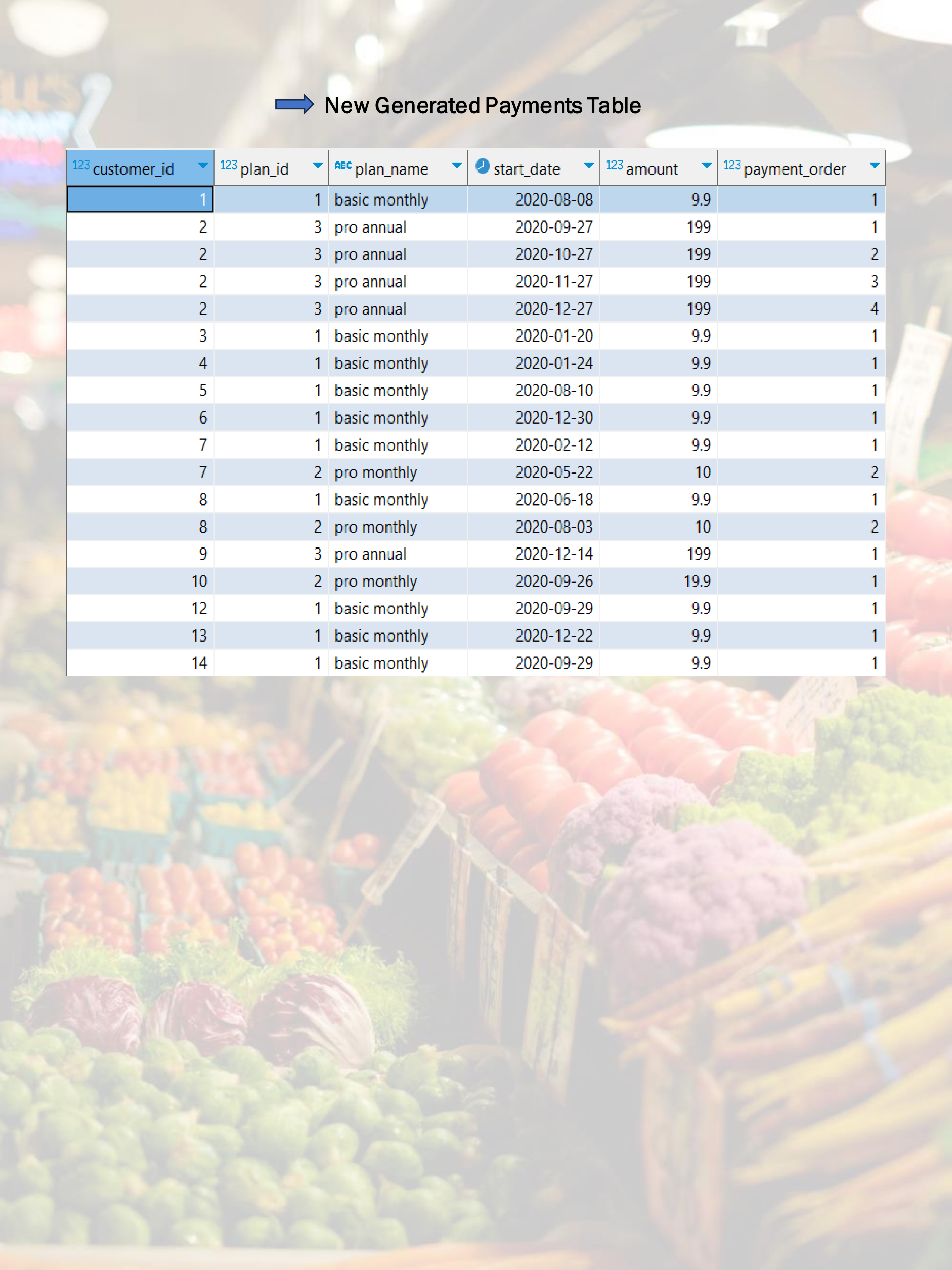


customer_id	plan_id	plan_name	start_date	end_date	amount
1	1	basic monthly	2020-08-08	2020-12-31	9.9
2	3	pro annual	2020-09-27	2020-12-31	199
3	1	basic monthly	2020-01-20	2020-12-31	9.9
4	1	basic monthly	2020-01-24	2020-12-31	9.9
5	1	basic monthly	2020-08-10	2020-12-31	9.9
6	1	basic monthly	2020-12-30	2020-12-31	9.9
7	1	basic monthly	2020-02-12	2020-05-22	9.9
7	2	pro monthly	2020-05-22	2020-12-31	19.9
8	1	basic monthly	2020-06-18	2020-08-03	9.9
8	2	pro monthly	2020-08-03	2020-12-31	19.9
9	3	pro annual	2020-12-14	2020-12-31	199
10	2	pro monthly	2020-09-26	2020-12-31	19.9
12	1	basic monthly	2020-09-29	2020-12-31	9.9
13	1	basic monthly	2020-12-22	2020-12-31	9.9
14	1	basic monthly	2020-09-29	2020-12-31	9.9
15	2	pro monthly	2020-03-24	2020-12-31	19.9
16	1	basic monthly	2020-06-07	2020-10-21	9.9
16	3	pro annual	2020-10-21	2020-12-31	199

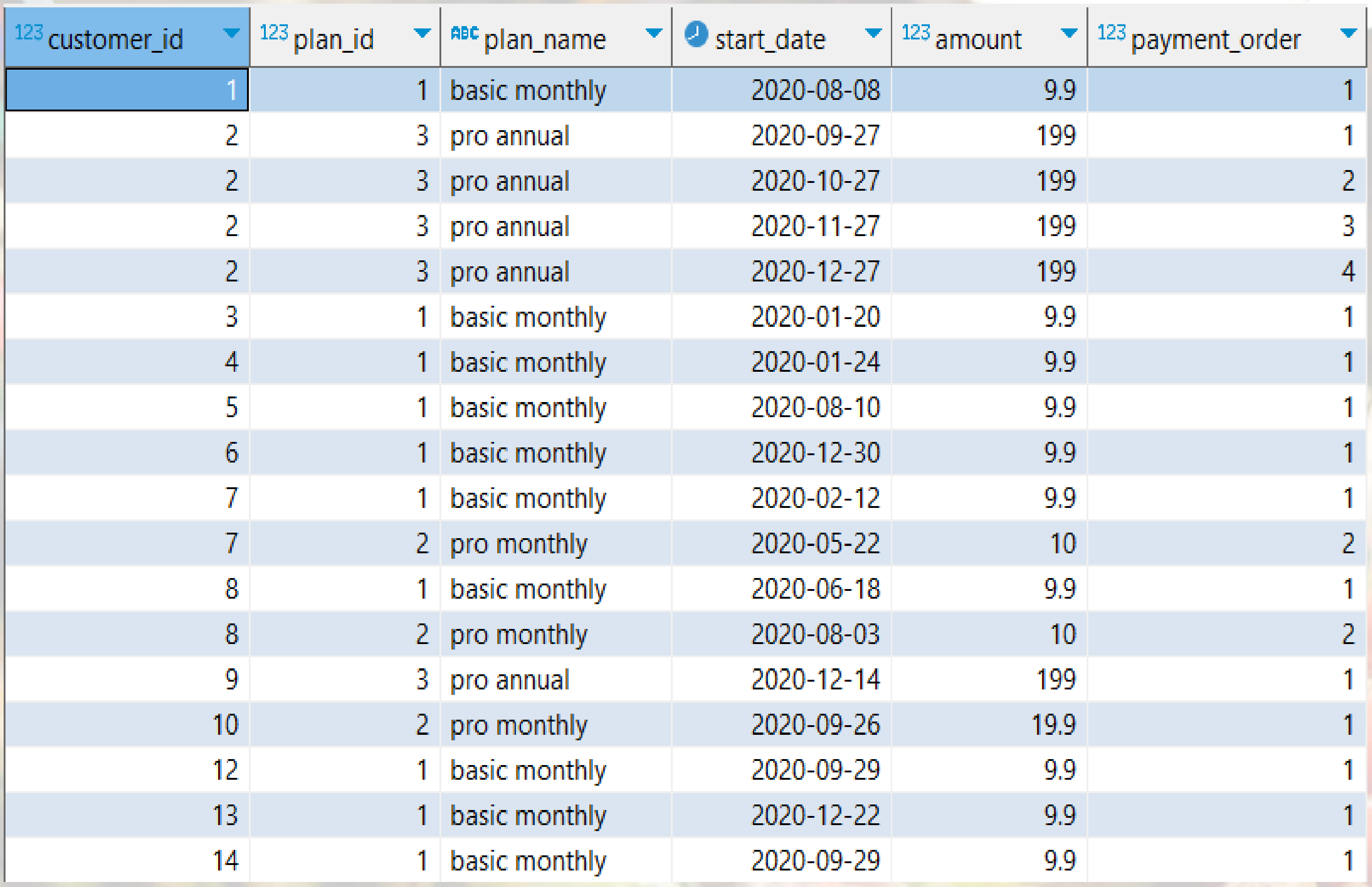
STEP 3: I created a recursive **CTE** query that generates new rows for users on monthly plans by incrementing the **start date** by a month until it reaches the end date. I deducted the money paid for the basic plan from pro plans when users upgraded from the basic plan. The results were then ranked by start date for each customer. Finally, I used these results to create a table named **'payments'**.



```
CREATE TABLE payments AS (
WITH RECURSIVE cte AS (
  SELECT
    s.customer_id, s.plan_id, p.plan_name, s.start_date,
    LEAD(s.start_date) OVER (PARTITION BY s.customer_id ORDER BY s.start_date, s.plan_id) AS end_date,
    p.price AS amount
  FROM subscriptions s
  INNER JOIN plans p ON p.plan_id = s.plan_id
  WHERE EXTRACT(YEAR FROM s.start_date) = 2020
    AND p.plan_name NOT IN ('churn', 'trial')
),
cte1 AS (
  SELECT
    customer_id, plan_id, plan_name, start_date,
    COALESCE(end_date, '2020-12-31') AS end_date,
    amount
  FROM cte
),
cte2 AS (
  SELECT
    customer_id, plan_id, plan_name, start_date::date,
    COALESCE(end_date, '2020-12-31')::date AS end_date,
    amount
  FROM cte1
  UNION ALL
  SELECT
    customer_id, plan_id, plan_name,
    (start_date + INTERVAL '1 month')::date AS start_date,
    end_date::date, amount
  FROM cte2
  WHERE plan_name = 'pro annual' AND end_date > (start_date + INTERVAL '1 month')
),
cte3 AS (
  SELECT *,
    LAG(plan_id) OVER (PARTITION BY customer_id ORDER BY start_date) AS last_plan,
    LAG(amount) OVER (PARTITION BY customer_id ORDER BY start_date) AS last_amount_paid,
    RANK() OVER (PARTITION BY customer_id ORDER BY start_date) AS payment_order
  FROM cte2
)
SELECT
  customer_id, plan_id, plan_name, start_date,
  (CASE WHEN plan_id IN (2, 3) AND last_plan = 1 THEN amount - last_amount_paid ELSE amount END) AS amount,
  payment_order
FROM cte3
);
```



New Generated Payments Table



customer_id	plan_id	plan_name	start_date	amount	payment_order
1	1	basic monthly	2020-08-08	9.9	1
2	3	pro annual	2020-09-27	199	1
2	3	pro annual	2020-10-27	199	2
2	3	pro annual	2020-11-27	199	3
2	3	pro annual	2020-12-27	199	4
3	1	basic monthly	2020-01-20	9.9	1
4	1	basic monthly	2020-01-24	9.9	1
5	1	basic monthly	2020-08-10	9.9	1
6	1	basic monthly	2020-12-30	9.9	1
7	1	basic monthly	2020-02-12	9.9	1
7	2	pro monthly	2020-05-22	10	2
8	1	basic monthly	2020-06-18	9.9	1
8	2	pro monthly	2020-08-03	10	2
9	3	pro annual	2020-12-14	199	1
10	2	pro monthly	2020-09-26	19.9	1
12	1	basic monthly	2020-09-29	9.9	1
13	1	basic monthly	2020-12-22	9.9	1
14	1	basic monthly	2020-09-29	9.9	1

THANK YOU !