Subscription Style Analysis with SQL

1. Project Background

Foodie-Fi is a subscription based businesses - streaming service that only had food related content, something like Netflix but with only cooking shows.

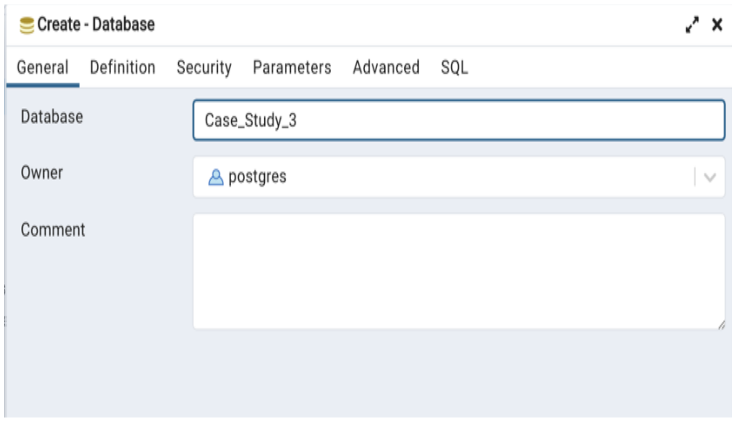
Danny as CEO and founder, realised that there was a large gap in the market. He finds a few smart friends to launch his new start-up Foodie-Fi in 2020 and started selling monthly and annual subscriptions, giving their customers unlimited on-demand access to exclusive food videos from around the world.

In this project, you will extract and analyse performance by focusing on using subscription-style digital data (only 2 tables) to answer subscriber behaviour and tell the story of how you have been able to generate that growth.

2. Data Preparation

2.1 Create database, schema and table

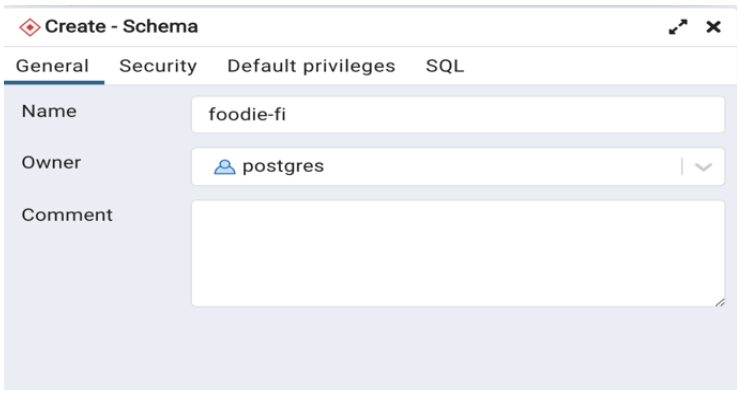
For this case study, I used PostgreSQL then I create the database, schema and table by using menu options after right clicking on database as shown in following image:



Database: Case_Study_3

Owner: postgres

Comment:

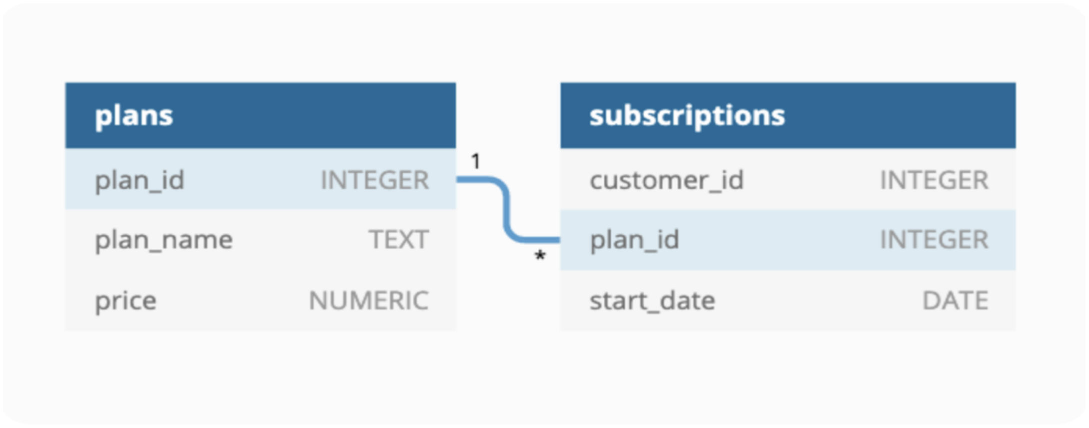


Name: foodie-fi

Owner: postgres

Comment:

2.2 Entity Relationship Diagram (ERD)



2.3 Tables

Once the database and schema are created, then the two tables are created according to the following SQL query :

a. Plans

Records consist of plan id, plan name and price. There are 5 rows in this table.

As we can see, customers can choose which plans to join Foodie-Fi when they first sign up.

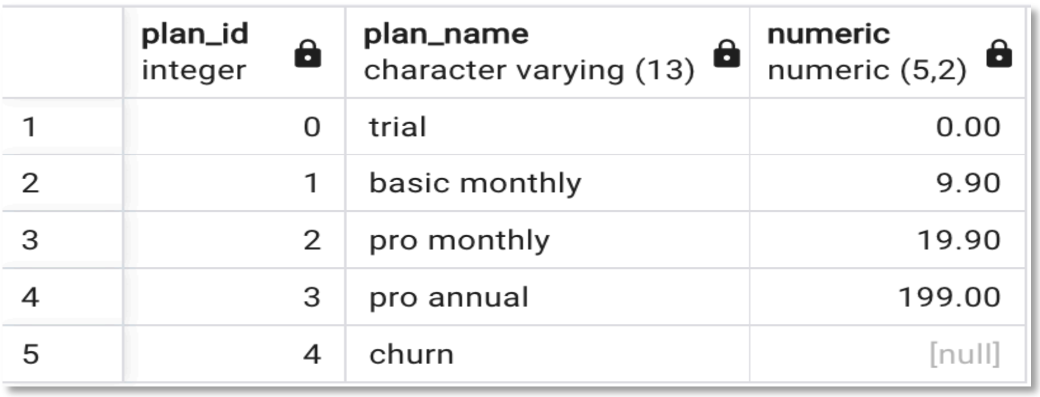
Basic plan customers have limited access and can only stream their videos and is only available monthly at $9.90.

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.

```
CREATE TABLE plans (
  plan_id INTEGER,
  plan_name VARCHAR(13),
  price DECIMAL(5,2)
);

INSERT INTO plans
  (plan_id, plan_name, price)
VALUES
  ('0', 'trial', '0'),
  ('1', 'basic monthly', '9.90'),
  ('2', 'pro monthly', '19.90'),
  ('3', 'pro annual', '199'),
  ('4', 'churn', null);
```



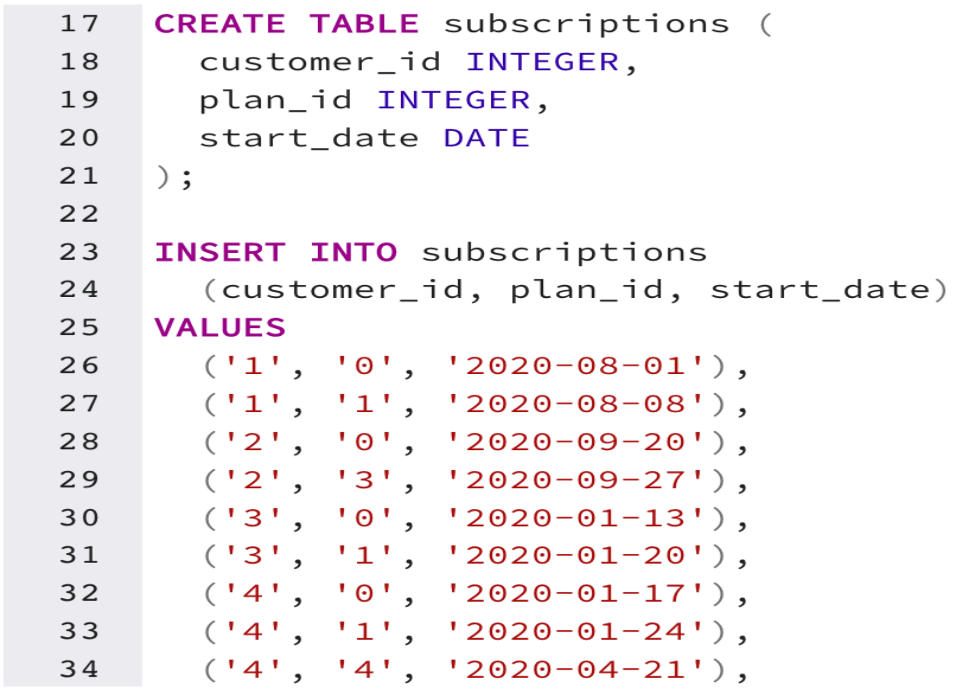
	plan_id integer	plan_name character varying (13)	numeric numeric (5,2)
1	0	trial	0.00
2	1	basic monthly	9.90
3	2	pro monthly	19.90
4	3	pro annual	199.00
5	4	churn	[null]

b. Subscriptions

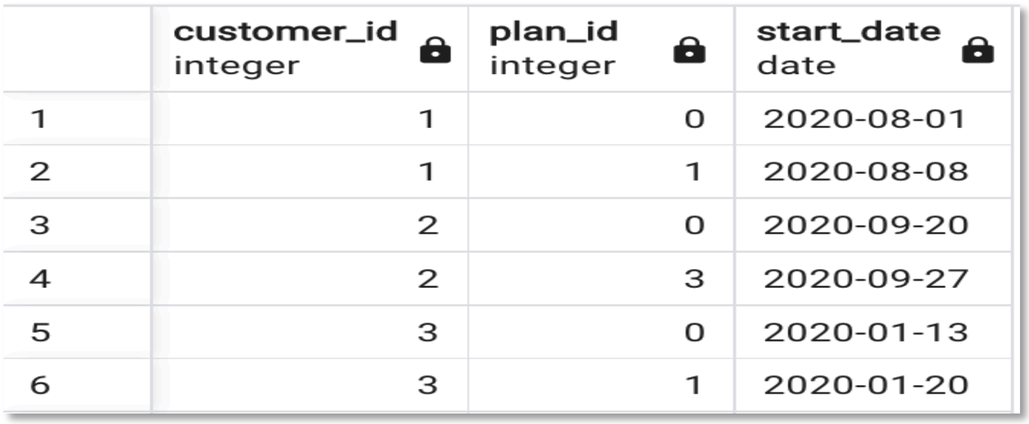
Records consist of customer id, plan id and the exact date where specific plan id starts. There are 2650 rows in this table.
Customer subscriptions shows If customers downgrade from a pro plan or cancel their subscription - the higher plan will remain in place until the period is over - the start date 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.



```sql
CREATE TABLE subscriptions (
  customer_id INTEGER,
  plan_id INTEGER,
  start_date DATE
);

INSERT INTO subscriptions
  (customer_id, plan_id, start_date)
VALUES
  ('1', '0', '2020-08-01'),
  ('1', '1', '2020-08-08'),
  ('2', '0', '2020-09-20'),
  ('2', '3', '2020-09-27'),
  ('3', '0', '2020-01-13'),
  ('3', '1', '2020-01-20'),
  ('4', '0', '2020-01-17'),
  ('4', '1', '2020-01-24'),
  ('4', '4', '2020-04-21'),
```



	customer_id integer	plan_id integer	start_date date
1	1	0	2020-08-01
2	1	1	2020-08-08
3	2	0	2020-09-20
4	2	3	2020-09-27
5	3	0	2020-01-13
6	3	1	2020-01-20

3. Project Goal

This project is split into an initial data understanding question before diving straight into data analysis questions before finishing with 1 single extension challenge.

3.1 Customer Journey

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

3.2 Objectives

- Subscriber behaviour

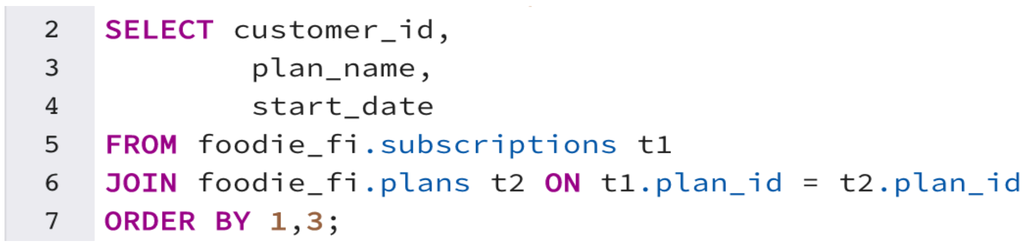
3.3 Problems Question

1. How many customers has Foodie-Fi ever had?
2. What is the monthly distribution of trial plan start_date values for our dataset - use the start of the month as the group by value?
3. What plan start_date values occur after the year 2020 for our dataset? Show the breakdown by count of events for each plan_name?
4. What is the customer count and percentage of customers who have churned rounded to 1 decimal place?
5. How many customers have churned straight after their initial free trial - what percentage is this rounded to the nearest whole number?
6. What is the number and percentage of customer plans after their initial free trial?
7. What is the customer count and percentage breakdown of all 5 plan_name values at 2020-12-31?
8. How many customers have upgraded to an annual plan in 2020?
9. How many days on average does it take for a customer to an annual plan from the day they join Foodie-Fi?
10. Can you further breakdown this average value into 30 day periods (i.e. 0-30 days, 31-60 days etc)
11. How many customers downgraded from a pro monthly to a basic monthly plan in 2020?

4. Analysis and Visualization

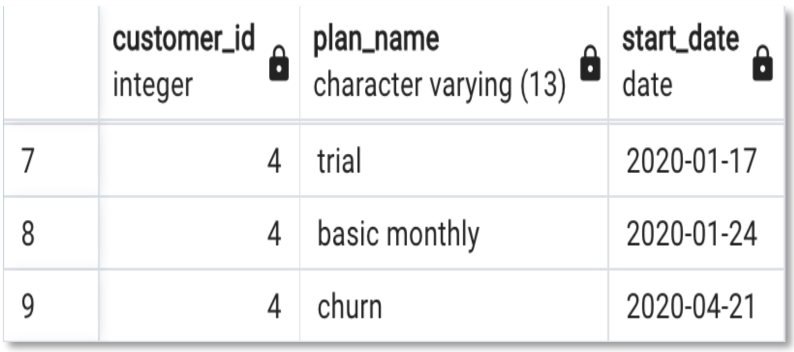
4.1. Customer Journey

1. Write a brief description about each customer's onboarding journey.



```sql
SELECT customer_id,
        plan_name,
        start_date
FROM foodie_fi.subscriptions t1
JOIN foodie_fi.plans t2 ON t1.plan_id = t2.plan_id
ORDER BY 1,3;
```

Result :

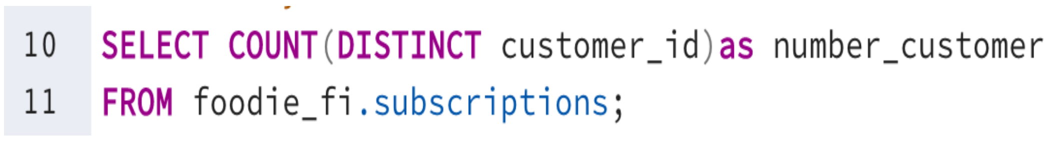


	customer_id integer	plan_name character varying (13)	start_date date
7	4	trial	2020-01-17
8	4	basic monthly	2020-01-24
9	4	churn	2020-04-21

As you can see, the customer number 1 started the trial plan in 2020–01-17 and upgraded the plan to basic monthly a week later, until the customer number 1 churned the plan in 2020–04–21. That's how you read the customer's journey.

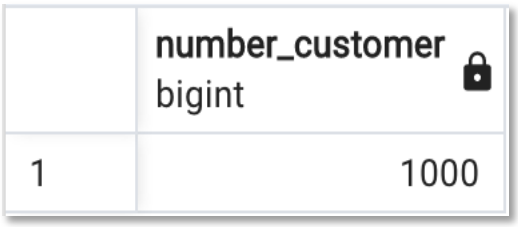
4.2. Data Analysis Questions

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



```sql
SELECT COUNT(DISTINCT customer_id)as number_customer
FROM foodie_fi.subscriptions;
```

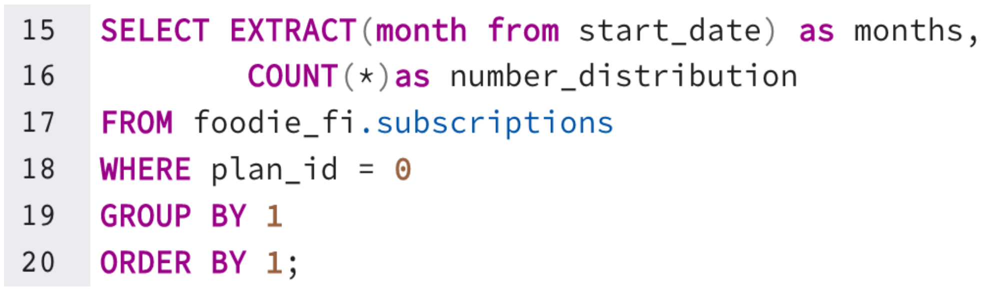
Result :



	number_customer bigint
1	1000

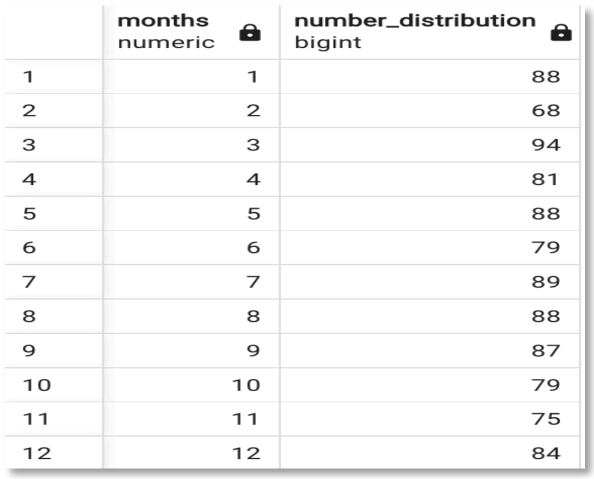
Foodie-Fi has 1000 subscribers.

Q2. What is the monthly distribution of trial plan start_date values for our dataset. Use the start of the month as the group by value?



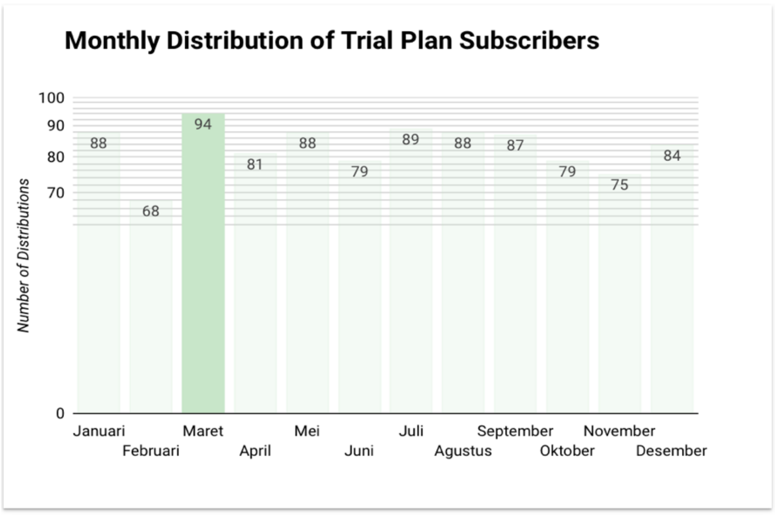
```sql
SELECT EXTRACT(month from start_date) as months,
            COUNT(*)as number_distribution
FROM foodie_fi.subscriptions
WHERE plan_id = 0
GROUP BY 1
ORDER BY 1;
```

Result :



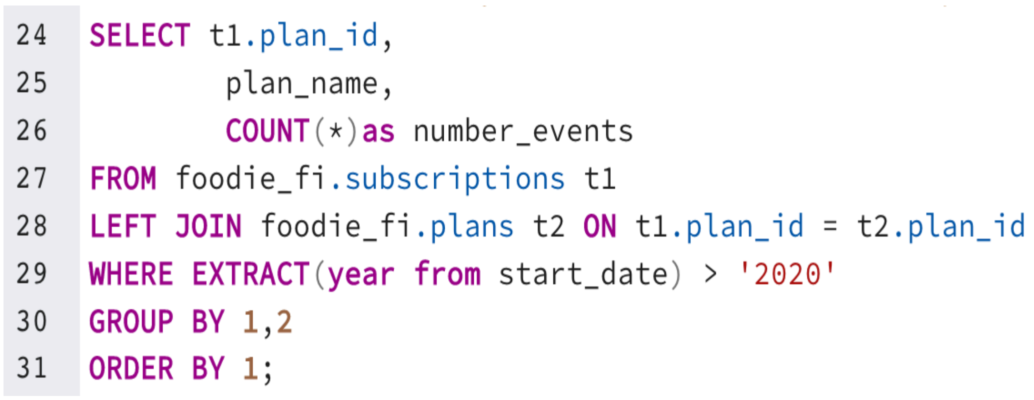
	months numeric	number_distribution bigint
1	1	88
2	2	68
3	3	94
4	4	81
5	5	88
6	6	79
7	7	89
8	8	88
9	9	87
10	10	79
11	11	75
12	12	84

Insight :



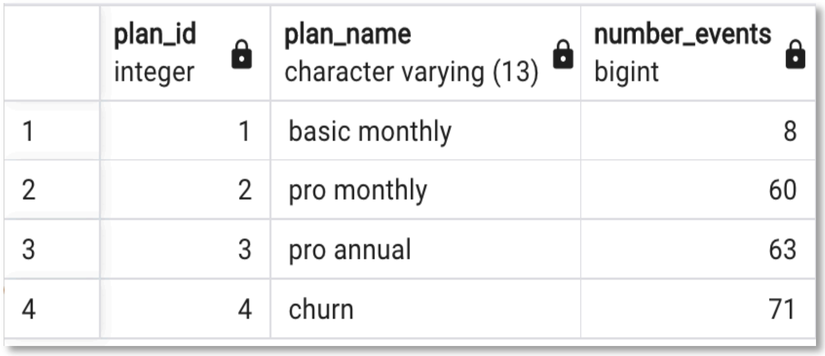
The distribution of the number of subscribers reached its highest figure in March

Q3. What plan start_date values occur after the year 2020 for our dataset? Show the breakdown by count of events for each plan_name?



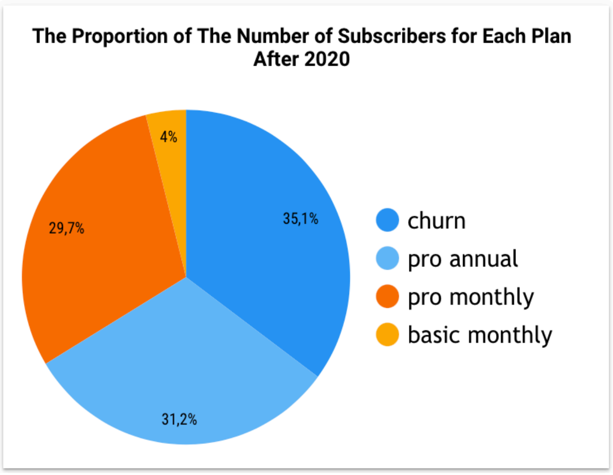
```
SELECT t1.plan_id,
        plan_name,
        COUNT(*)as number_events
FROM foodie_fi.subscriptions t1
LEFT JOIN foodie_fi.plans t2 ON t1.plan_id = t2.plan_id
WHERE EXTRACT(year from start_date) > '2020'
GROUP BY 1,2
ORDER BY 1;
```

Result :



	plan_id integer	plan_name character varying (13)	number_events bigint
1	1	basic monthly	8
2	2	pro monthly	60
3	3	pro annual	63
4	4	churn	71

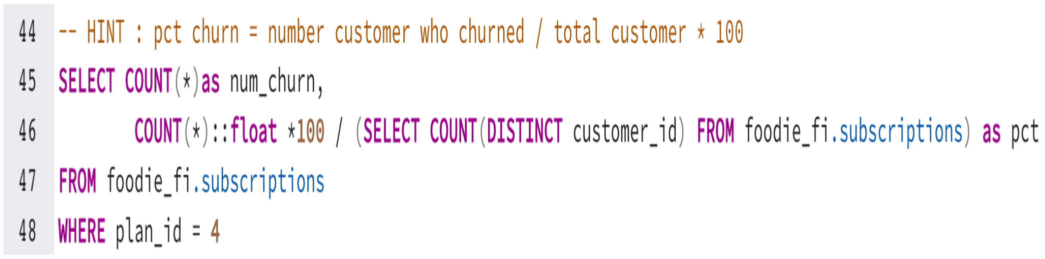
Insight :



The number of subscribers who churned the plan was the biggest one after the year 2020, with 71 subscribers.

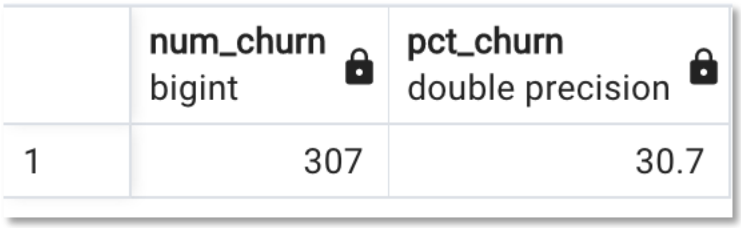
Q4. What is the customer count and percentage of customers who have churned rounded to 1 decimal place?

To find the percentage we can calculate the number of rows multiplied by 100 divided by the total number of customers.



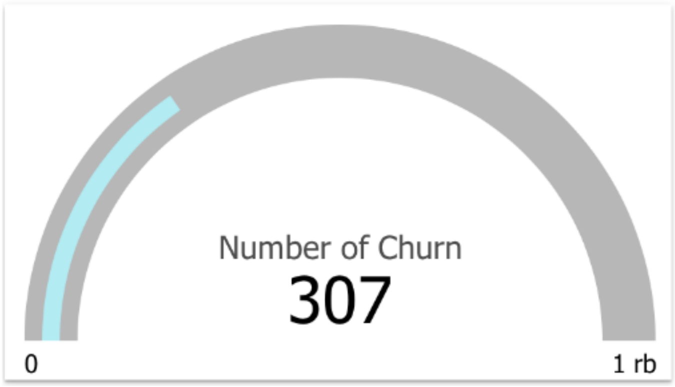
```
-- HINT : pct churn = number customer who churned / total customer * 100
SELECT COUNT(*)as num_churn,
       COUNT(*)::float *100 / (SELECT COUNT(DISTINCT customer_id) FROM foodie_fi.subscriptions) as pct
FROM foodie_fi.subscriptions
WHERE plan_id = 4
```

Result :



	num_churn bigint	pct_churn double precision
1	307	30.7

Insight :

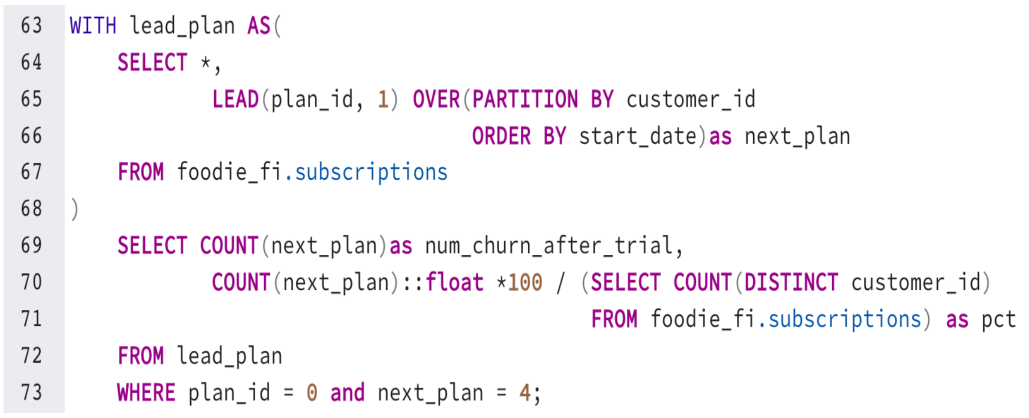


There are 307 customers who have churned, which 30.7% of customers who have churned the plans.

Q5. How many customers have churned straight after their initial free trial - what percentage is this rounded to the nearest whole number?

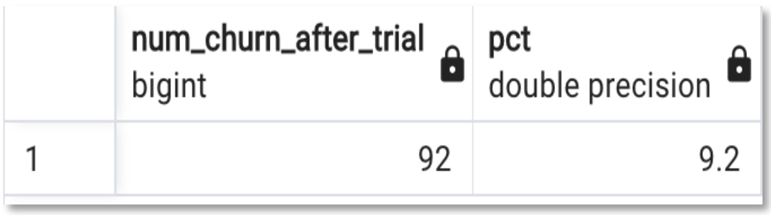
We can use CTE to create a temporary result that can be referred later on. LEAD clause is used to look forward a number of rows and access data of that row from the current row.

Then we named the result from LEAD clause as lead_plan so we can recall it outside the CTE set. Filter the plan_id = 0 and next_plan = 4 because we want to find the percentage who have churned after trial.



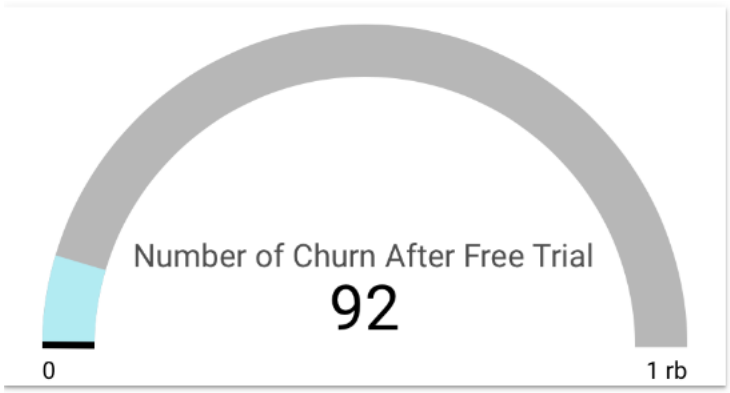
```
WITH lead_plan AS(
    SELECT *,
            LEAD(plan_id, 1) OVER(PARTITION BY customer_id
                                  ORDER BY start_date)as next_plan
    FROM foodie_fi.subscriptions
)
    SELECT COUNT(next_plan)as num_churn_after_trial,
            COUNT(next_plan)::float *100 / (SELECT COUNT(DISTINCT customer_id)
                                            FROM foodie_fi.subscriptions) as pct
    FROM lead_plan
    WHERE plan_id = 0 and next_plan = 4;
```

Result :



	num_churn_after_trial bigint	pct double precision
1	92	9.2

Insight :



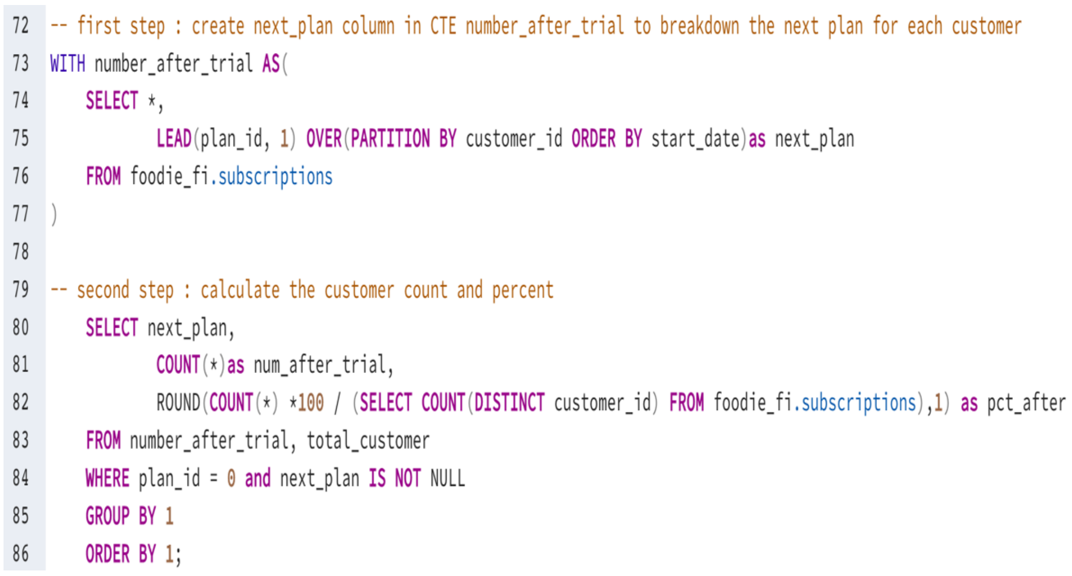
There are 92 customers who have churned straight after their initial free trial, which 9% of the customer base.

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

To breakdown all plan after their initial free trial, there are few steps to do:

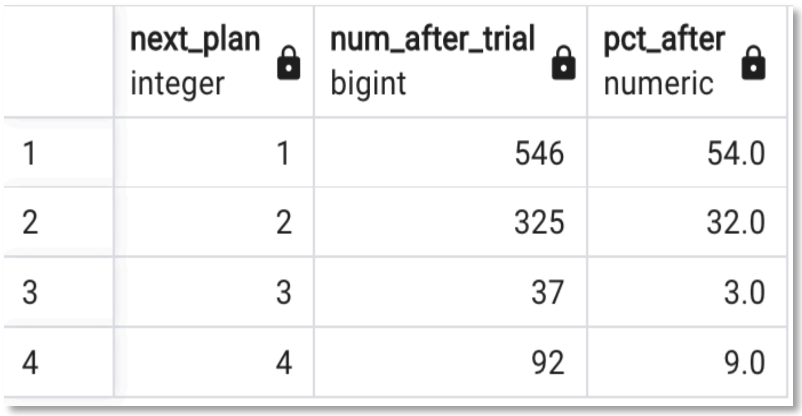
We can use CTE to create a temporary result that can be referred later on. **LEAD** clause is used **to return the plan_id of the current plan and the next plan of each customer.**

Then we named the result from LEAD clause as lead_plan so we can recall it outside the CTE set. Filter the plan_id = 0 and next_plan IS NOT NULL because we want to find the percentage after trial plan.



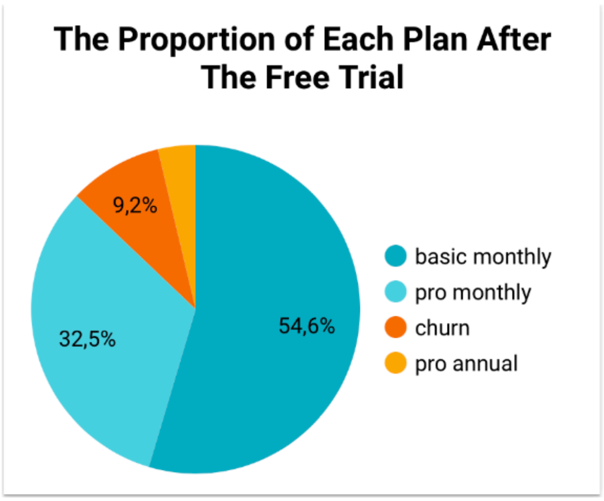
```sql
-- first step : create next_plan column in CTE number_after_trial to breakdown the next plan for each customer
WITH number_after_trial AS(
    SELECT *,
            LEAD(plan_id, 1) OVER(PARTITION BY customer_id ORDER BY start_date)as next_plan
    FROM foodie_fi.subscriptions
)

-- second step : calculate the customer count and percent
    SELECT next_plan,
            COUNT(*)as num_after_trial,
            ROUND(COUNT(*) *100 / (SELECT COUNT(DISTINCT customer_id) FROM foodie_fi.subscriptions),1) as pct_after
    FROM number_after_trial, total_customer
    WHERE plan_id = 0 and next_plan IS NOT NULL
    GROUP BY 1
    ORDER BY 1;
```

Result :



	next_plan integer	num_after_trial bigint	pct_after numeric
1	1	546	54.0
2	2	325	32.0
3	3	37	3.0
4	4	92	9.0

Insight :

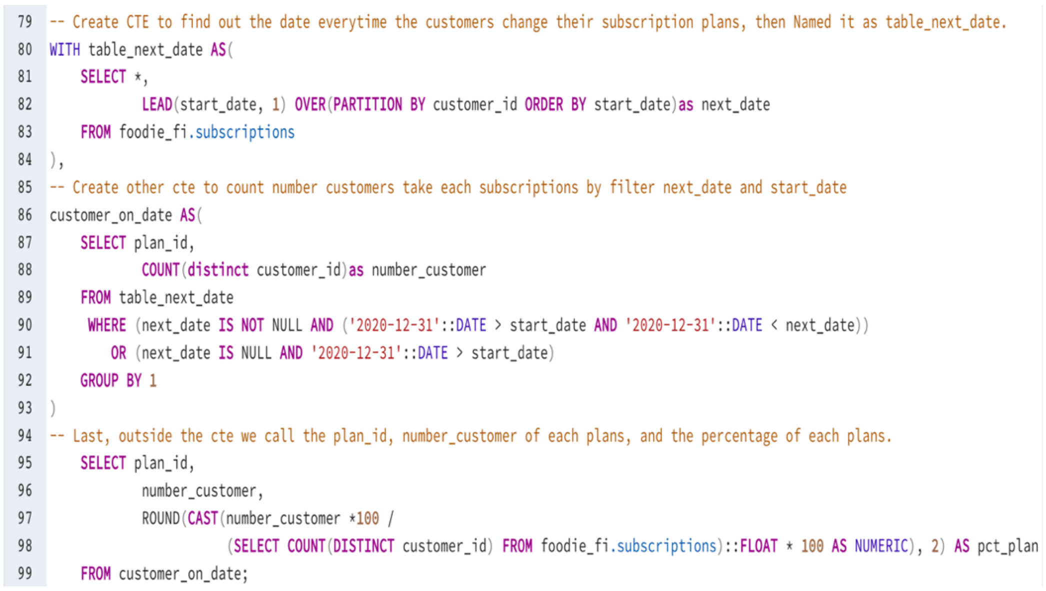


- 54.5% of customers choose basic monthly after their initial trial.
- 32.5% of customers choose pro monthly after their initial trial.
- 3.7% of customers choose pro annual after their initial trial.
- 9.2% of customers choose churn after their initial trial.

Q7. What is the customer count and percentage breakdown of all 5 plan name values at 2020-12-31?

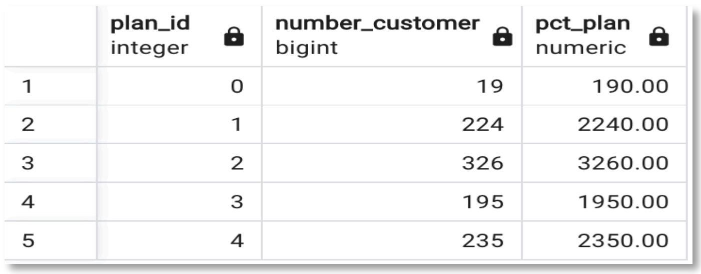
To **breakdown all 5 plan name at the exact date which is 2020–12–31**, there are few steps to do:

- Create CTE to find out the date every time the customers change their subscription plans, Or **LEAD** clause is used **to return the current date and next date of each customer.** then named it as 'table next date'.
- Create other CTE to count number of customers take each plan.
- Last, outside the CTE we call the plan id, number of customers of each plans, and the percentage of each plans



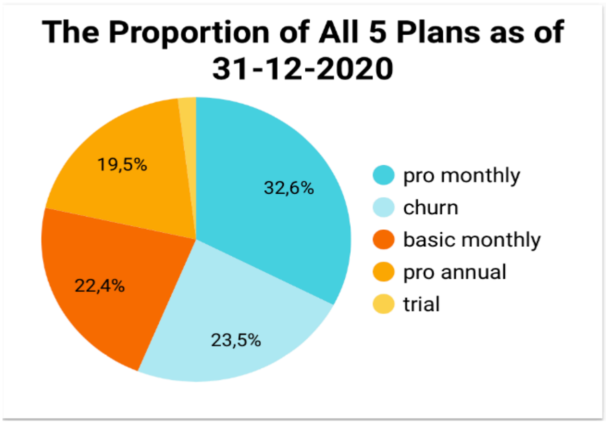
```sql
-- Create CTE to find out the date everytime the customers change their subscription plans, then Named it as table_next_date.
WITH table_next_date AS(
    SELECT *,
            LEAD(start_date, 1) OVER(PARTITION BY customer_id ORDER BY start_date)as next_date
    FROM foodie_fi.subscriptions
),
-- Create other cte to count number customers take each subscriptions by filter next_date and start_date
customer_on_date AS(
    SELECT plan_id,
            COUNT(distinct customer_id)as number_customer
    FROM table_next_date
     WHERE (next_date IS NOT NULL AND ('2020-12-31'::DATE > start_date AND '2020-12-31'::DATE < next_date))
        OR (next_date IS NULL AND '2020-12-31'::DATE > start_date)
    GROUP BY 1
)
-- Last, outside the cte we call the plan_id, number_customer of each plans, and the percentage of each plans.
    SELECT plan_id,
            number_customer,
            ROUND(CAST(number_customer *100 /
                        (SELECT COUNT(DISTINCT customer_id) FROM foodie_fi.subscriptions)::FLOAT * 100 AS NUMERIC), 2) AS pct_plan
    FROM customer_on_date;
```

Result :



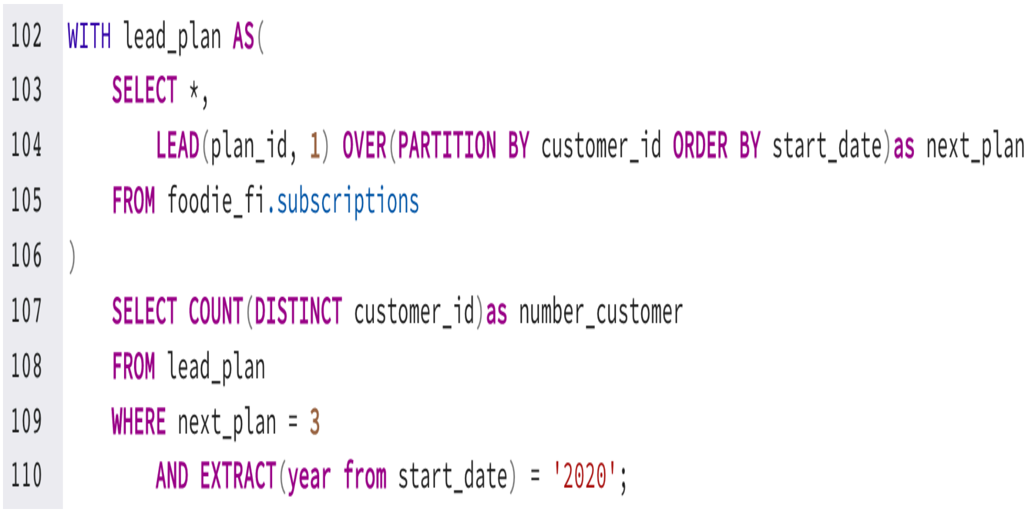
	plan_id integer	number_customer bigint	pct_plan numeric
1	0	19	190.00
2	1	224	2240.00
3	2	326	3260.00
4	3	195	1950.00
5	4	235	2350.00

Insight :



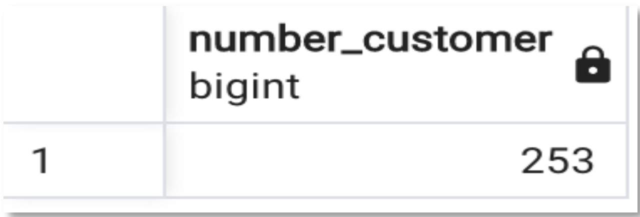
Plan_id 2 or Pro Monthly plan has the highest percentage of subscribers until 31-12-2020.

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



```sql
WITH lead_plan AS(
    SELECT *,
        LEAD(plan_id, 1) OVER(PARTITION BY customer_id ORDER BY start_date)as next_plan
    FROM foodie_fi.subscriptions
)
    SELECT COUNT(DISTINCT customer_id)as number_customer
    FROM lead_plan
    WHERE next_plan = 3
        AND EXTRACT(year from start_date) = '2020';
```

Result :



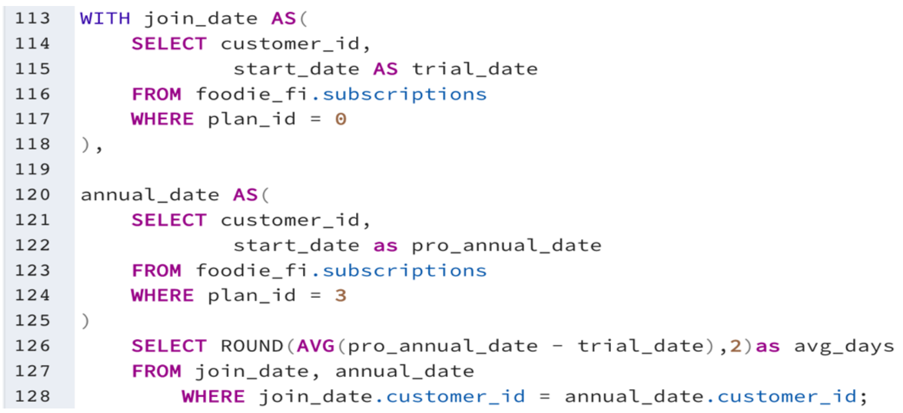
	number_customer bigint
1	253

There are 253 customers who have upgraded to annual in 2020.

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

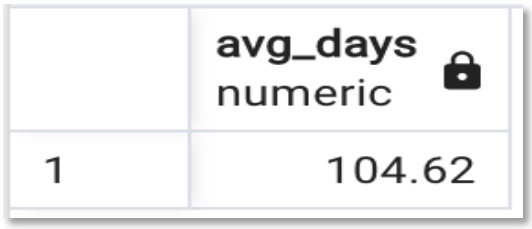
Create two CTEs to distinguish the plan_id, first cte to filter the annual plan and the second CTE is to filter the trial plan using the WHERE clause. Then, calculate the number of days between two date values.

Next, we find the average of days for how long the customers take the annual plan using AVERAGE clause.



```sql
WITH join_date AS(
    SELECT customer_id,
            start_date AS trial_date
    FROM foodie_fi.subscriptions
    WHERE plan_id = 0
),

annual_date AS(
    SELECT customer_id,
            start_date as pro_annual_date
    FROM foodie_fi.subscriptions
    WHERE plan_id = 3
)
    SELECT ROUND(AVG(pro_annual_date - trial_date),2)as avg_days
    FROM join_date, annual_date
        WHERE join_date.customer_id = annual_date.customer_id;
```

Result :



	avg_days numeric
1	104.62

On average, it takes 105 days for a customer take an annual plan from the day they joined Foodie-Fi.

Q10. Can you further breakdown this average value into 30 day periods? (i.e. 0–30 days, 31–60 days etc)

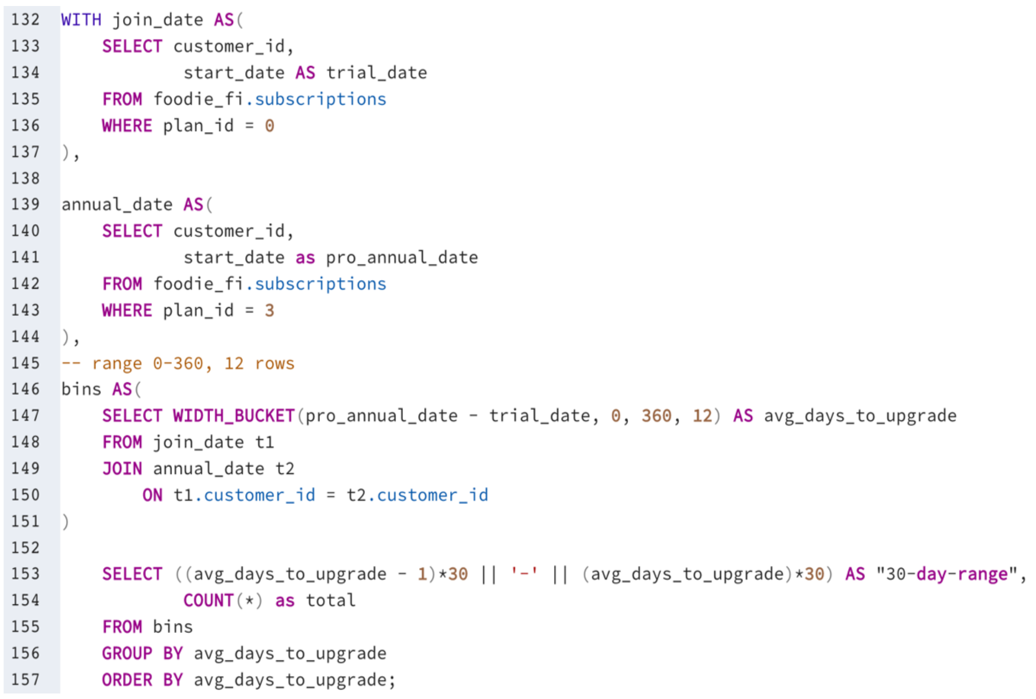
From the previous query we can add a few statements to breakdown this average value into 30 day periods? (i.e. 0–30 days, 31–60 days etc).

- Find out min and max values. Then we got Min 7 days and Max 346 for a customer to annual plan from the day they join Foodie-Fi

- Create the new CTE named bins and use the WIDTH_BUCKET function to returns the position of a specified operand in some specified buckets (named it as avg_days_to_upgrade in bins CTE).

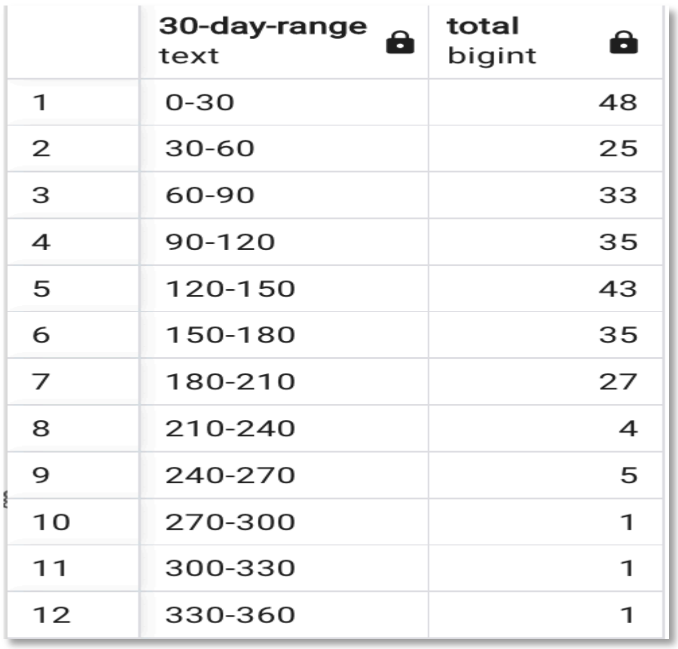
- The syntax : WIDTH_BUCKET(operand, low, high, count). We can create range start from 0 to 360 with 0 as minimum value, 360 as maximum value and 12 as number of rows

- Use || concatenate operator to create the 30 days period.



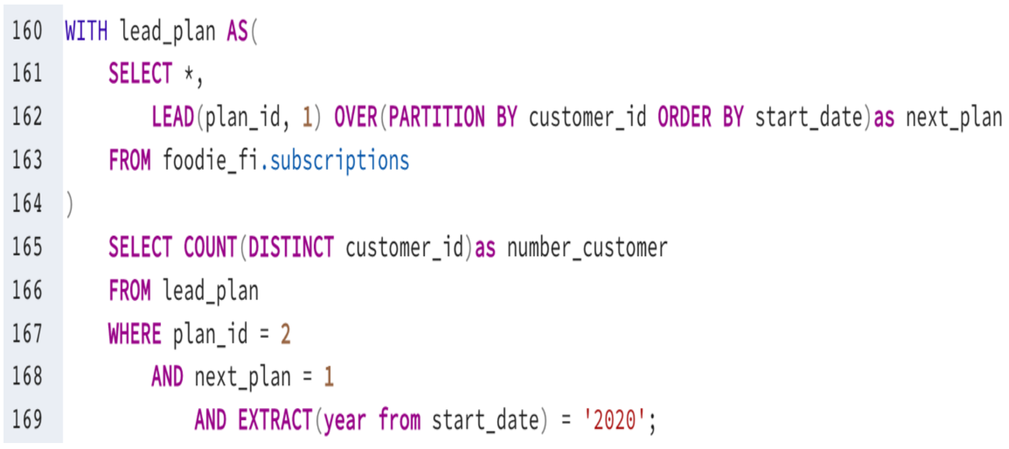
```sql
WITH join_date AS(
    SELECT customer_id,
            start_date AS trial_date
    FROM foodie_fi.subscriptions
    WHERE plan_id = 0
),

annual_date AS(
    SELECT customer_id,
            start_date as pro_annual_date
    FROM foodie_fi.subscriptions
    WHERE plan_id = 3
),
-- range 0-360, 12 rows
bins AS(
    SELECT WIDTH_BUCKET(pro_annual_date - trial_date, 0, 360, 12) AS avg_days_to_upgrade
    FROM join_date t1
    JOIN annual_date t2
        ON t1.customer_id = t2.customer_id
)

    SELECT ((avg_days_to_upgrade - 1)*30 || '-' || (avg_days_to_upgrade)*30) AS "30-day-range",
            COUNT(*) as total
    FROM bins
    GROUP BY avg_days_to_upgrade
    ORDER BY avg_days_to_upgrade;
```

Result :



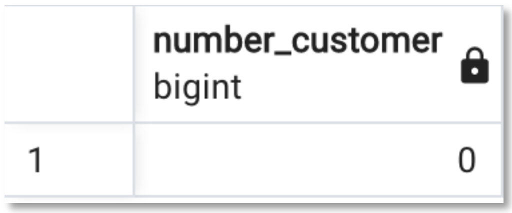
	30-day-range text	total bigint
1	0-30	48
2	30-60	25
3	60-90	33
4	90-120	35
5	120-150	43
6	150-180	35
7	180-210	27
8	210-240	4
9	240-270	5
10	270-300	1
11	300-330	1
12	330-360	1

Q11. How many customers downgraded from a pro monthly to a basic monthly plan in 2020?



```
WITH lead_plan AS(
    SELECT *,
        LEAD(plan_id, 1) OVER(PARTITION BY customer_id ORDER BY start_date)as next_plan
    FROM foodie_fi.subscriptions
)
    SELECT COUNT(DISTINCT customer_id)as number_customer
    FROM lead_plan
    WHERE plan_id = 2
        AND next_plan = 1
            AND EXTRACT(year from start_date) = '2020';
```

Result :



	number_customer bigint
1	0

5. Insight and Recommendation

5.1 Insight :

- Foodie-Fi has a total of 1000 subscribers, with customer journey from trial plan reaching the highest number of subscribers in March.
- The number of subscribers who churned the plan was the biggest one after the year 2020, with 71 subscribers.
- Of the total 30.7% of subscribers who cancelled plans, around 9.2% of the subscribers base had churned right after their initial free trial.
- The number of subscribers choosing a 'Basic Monthly' was the largest after the initial trial, with 54.5%. While the lowest out of all plan was 'Pro Annual', with 3.7%.
- Even though 'Basic Monthly' was popular overall but till 2020-12-31, the 'Pro Monthly' was the most chose one.

- There were 253 subscribers who had upgraded to annual in 2020. On average, it took customers 105 days to pick up their annual package from the time they joined Foodie-Fi.
- There was none of the subscribers downgraded from a pro monthly to a basic monthly plan in 2020.

5.2 Recommendation :

- Find out what makes a ‘Basic Monthly as their favourite plan for subscribers chose first after the initial trial and apply the same strategy in other plans.
- The highest number of churned after 2020, it can be an early diagnosis that the customer is not interested or dissatisfied with the service.
- With more than 190 customers who have upgraded to become an annual in 2020, Danny and the team can focus campaigns and budgets on loyal and potential subscribers.
- The company should utilize customer and product information for marketing strategies that will help in get loyal subscriber.