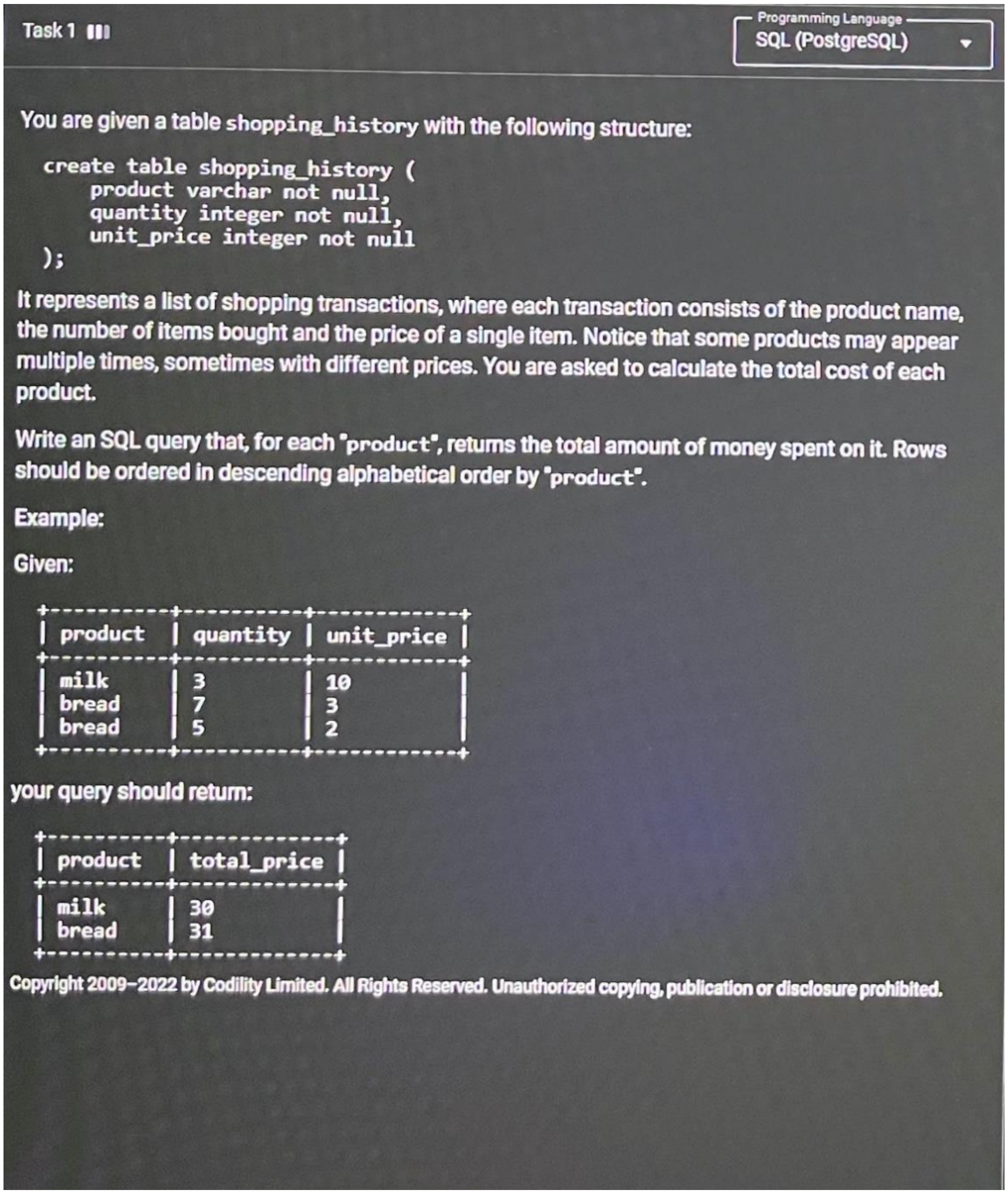
ASSIGNMENT 1

Create the following table structure in SNOWFLAKE by creating your own warehouse. Insert some 10 rows using INSERT command (check task 3 and same way insert for all task tables) in the table by trying different values for all the columns and then check using SELECT *

Once data is loaded, performed the below task

Task 1



You are given a table shopping_history with the following structure:

```
create table shopping_history (
    product varchar not null,
    quantity integer not null,
    unit_price integer not null
);
```

It represents a list of shopping transactions, where each transaction consists of the product name, the number of items bought and the price of a single item. Notice that some products may appear multiple times, sometimes with different prices. You are asked to calculate the total cost of each product.

Write an SQL query that, for each "product", returns the total amount of money spent on it. Rows should be ordered in descending alphabetical order by "product".

Example:

Given:

```
+----------+----------+------------+
| product  | quantity | unit_price |
+----------+----------+------------+
| milk     | 3        | 10         |
| bread    | 7        | 3          |
| bread    | 5        | 2          |
+----------+----------+------------+
```

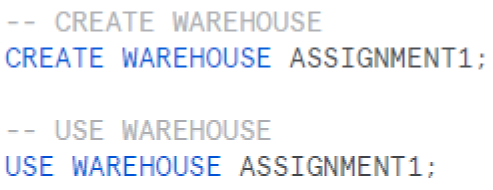
your query should return:

```
+----------+-------------+
| product  | total_price |
+----------+-------------+
| milk     | 30          |
| bread    | 31          |
+----------+-------------+
```

Answer:

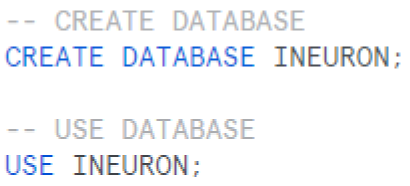
After login into snowflake, we will

Step 1. Create or select an existing warehouse and use it



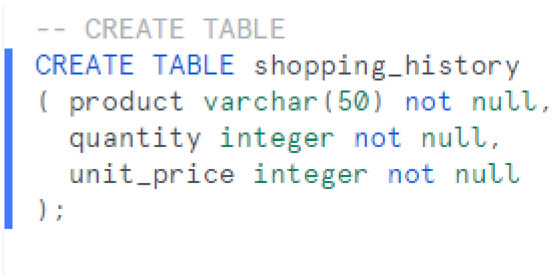
```
-- CREATE WAREHOUSE
CREATE WAREHOUSE ASSIGNMENT1;

-- USE WAREHOUSE
USE WAREHOUSE ASSIGNMENT1;
```

Step 2. Create or select a database and use it



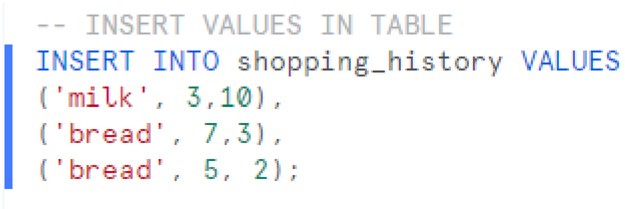
```
-- CREATE DATABASE
CREATE DATABASE INEURON;

-- USE DATABASE
USE INEURON;
```

Step 3. Create a Table named shopping_history



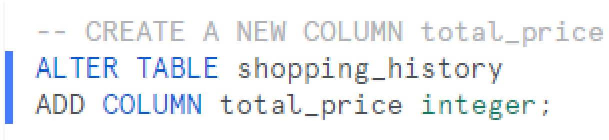
```
-- CREATE TABLE
CREATE TABLE shopping_history
( product varchar(50) not null,
  quantity integer not null,
  unit_price integer not null
);
```

Step 4. Insert Values in the table



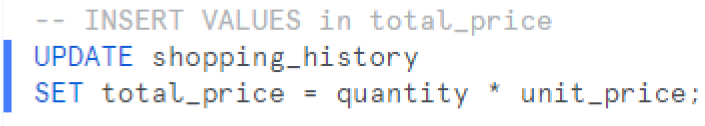
```
-- INSERT VALUES IN TABLE
INSERT INTO shopping_history VALUES
('milk', 3,10),
('bread', 7,3),
('bread', 5, 2);
```

Step 5. We would require a total_price column so we will add a new column in table



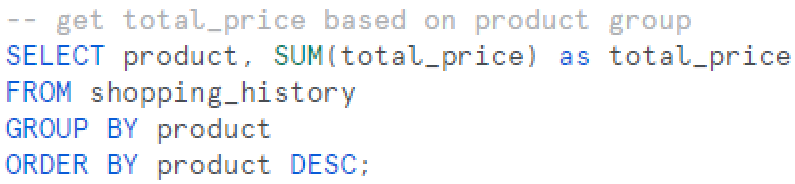
```
-- CREATE A NEW COLUMN total_price
ALTER TABLE shopping_history
ADD COLUMN total_price integer;
```

Step 6. Insert values in total_price



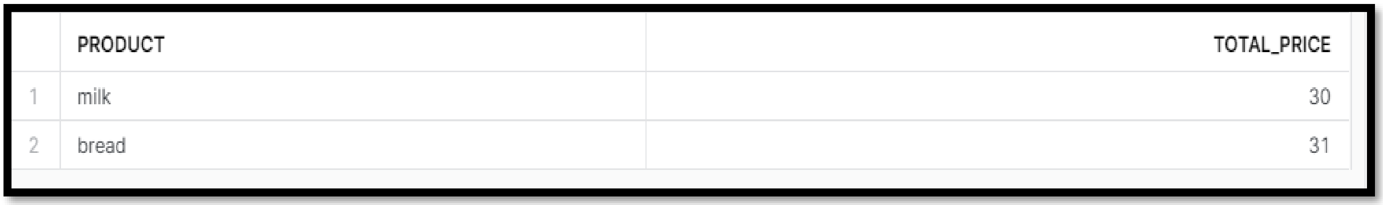
```
-- INSERT VALUES in total_price
UPDATE shopping_history
SET total_price = quantity * unit_price;
```

Step 7. The result that we need is the total_price based on product group



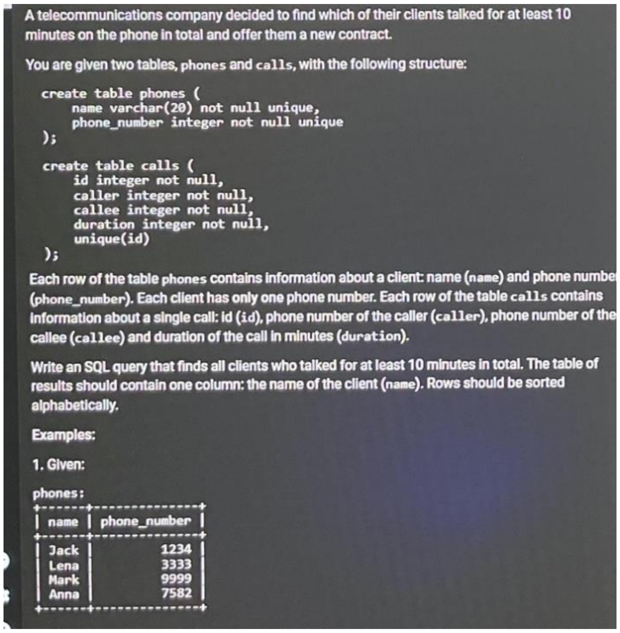
```
-- get total_price based on product group
SELECT product, SUM(total_price) as total_price
FROM shopping_history
GROUP BY product
ORDER BY product DESC;
```

The output which we get is the desired result



	PRODUCT	TOTAL_PRICE
1	milk	30
2	bread	31

Task 2:



A telecommunications company decided to find which of their clients talked for at least 10 minutes on the phone in total and offer them a new contract.

You are given two tables, phones and calls, with the following structure:

```
  create table phones (
      name varchar(20) not null unique,
      phone_number integer not null unique
  );

  create table calls (
      id integer not null,
      caller integer not null,
      callee integer not null,
      duration integer not null,
      unique(id)
  );
```

Each row of the table phones contains information about a client: name (name) and phone numbe (phone_number). Each client has only one phone number. Each row of the table calls contains information about a single call: id (id), phone number of the caller (caller), phone number of the callee (callee) and duration of the call in minutes (duration).

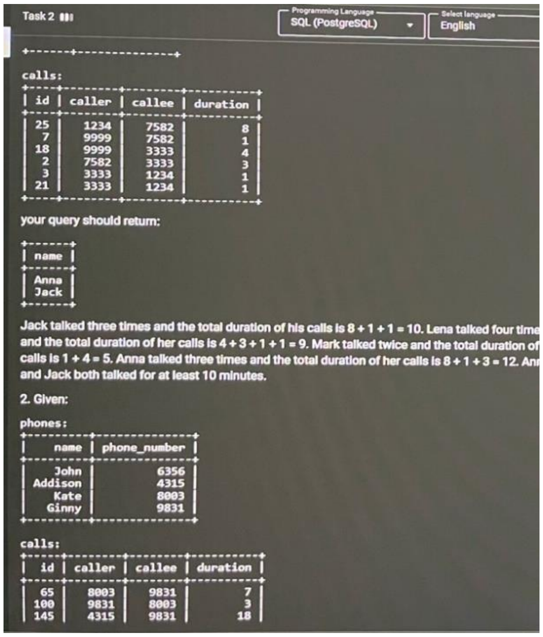
Write an SQL query that finds all clients who talked for at least 10 minutes in total. The table of results should contain one column: the name of the client (name). Rows should be sorted alphabetically.

Examples:

1. Given:

phones:

```
+------+--------------+
| name | phone_number |
+------+--------------+
| Jack |         1234 |
| Lena |         3333 |
| Mark |         9999 |
| Anna |         7582 |
+------+--------------+
```



Task 2

Programming Language: SQL (PostgreSQL)

Select language: English

```
+------+--------------+
```

calls:

```
+----+--------+--------+----------+
| id | caller | callee | duration |
+----+--------+--------+----------+
| 25 |   1234 |   7582 |        8 |
|  7 |   9999 |   7582 |        1 |
| 18 |   9999 |   3333 |        4 |
|  2 |   7582 |   3333 |        3 |
|  3 |   3333 |   1234 |        1 |
| 21 |   3333 |   1234 |        1 |
+----+--------+--------+----------+
```

your query should return:

```
+------+
| name |
+------+
| Anna |
| Jack |
+------+
```

Jack talked three times and the total duration of his calls is 8 + 1 + 1 = 10. Lena talked four time and the total duration of her calls is 4 + 3 + 1 + 1 = 9. Mark talked twice and the total duration of calls is 1 + 4 = 5. Anna talked three times and the total duration of her calls is 8 + 1 + 3 = 12. Ann and Jack both talked for at least 10 minutes.

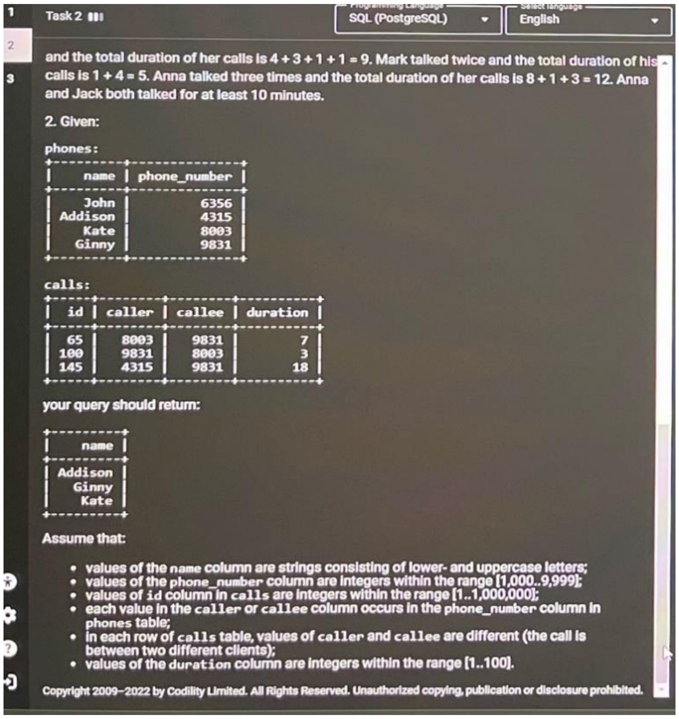
2. Given:

phones:

```
+---------+--------------+
|    name | phone_number |
+---------+--------------+
|    John |         6356 |
| Addison |         4315 |
|    Kate |         8003 |
|   Ginny |         9831 |
+---------+--------------+
```

calls:

```
+-----+--------+--------+----------+
|  id | caller | callee | duration |
+-----+--------+--------+----------+
|  65 |   8003 |   9831 |        7 |
| 100 |   9831 |   8003 |        3 |
| 145 |   4315 |   9831 |       18 |
```



Task 2

Programming Language: SQL (PostgreSQL)

Select language: English

and the total duration of her calls is 4 + 3 + 1 + 1 = 9. Mark talked twice and the total duration of his calls is 1 + 4 = 5. Anna talked three times and the total duration of her calls is 8 + 1 + 3 = 12. Anna and Jack both talked for at least 10 minutes.

2. Given:

phones:

```
+---------+--------------+
|    name | phone_number |
+---------+--------------+
|    John |         6356 |
| Addison |         4315 |
|    Kate |         8003 |
|   Ginny |         9831 |
+---------+--------------+
```

calls:

```
+-----+--------+--------+----------+
|  id | caller | callee | duration |
+-----+--------+--------+----------+
|  65 |   8003 |   9831 |        7 |
| 100 |   9831 |   8003 |        3 |
| 145 |   4315 |   9831 |       18 |
+-----+--------+--------+----------+
```

your query should return:

```
+---------+
|    name |
+---------+
| Addison |
|   Ginny |
|    Kate |
+---------+
```

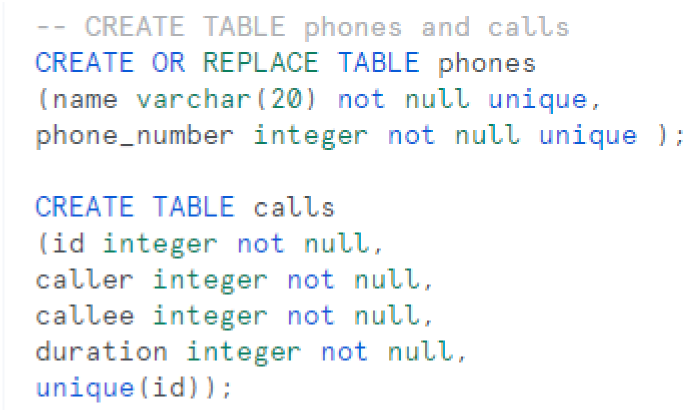
Assume that:

- values of the name column are strings consisting of lower- and uppercase letters;
- values of the phone_number column are integers within the range [1,000..9,999];
- values of id column in calls are integers within the range [1..1,000,000];
- each value in the caller or callee column occurs in the phone_number column in phones table;
- in each row of calls table, values of caller and callee are different (the call is between two different clients);
- values of the duration column are integers within the range [1..100].

Answer:

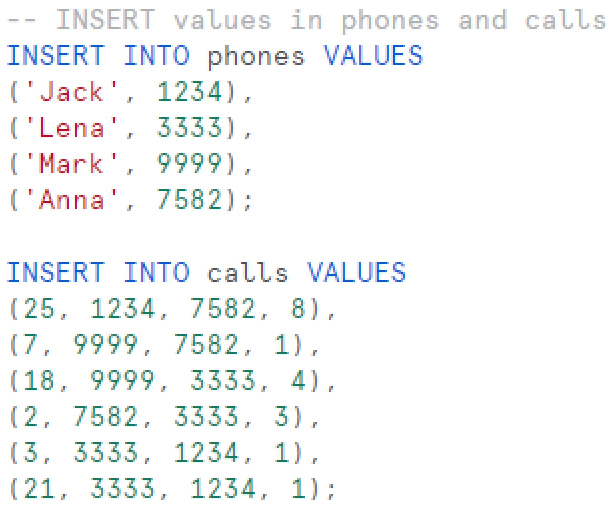
In this task, we need to get the total duration which is >=10 for the calls which are made. We will perform following steps:

Step 1. Create both phones and calls table



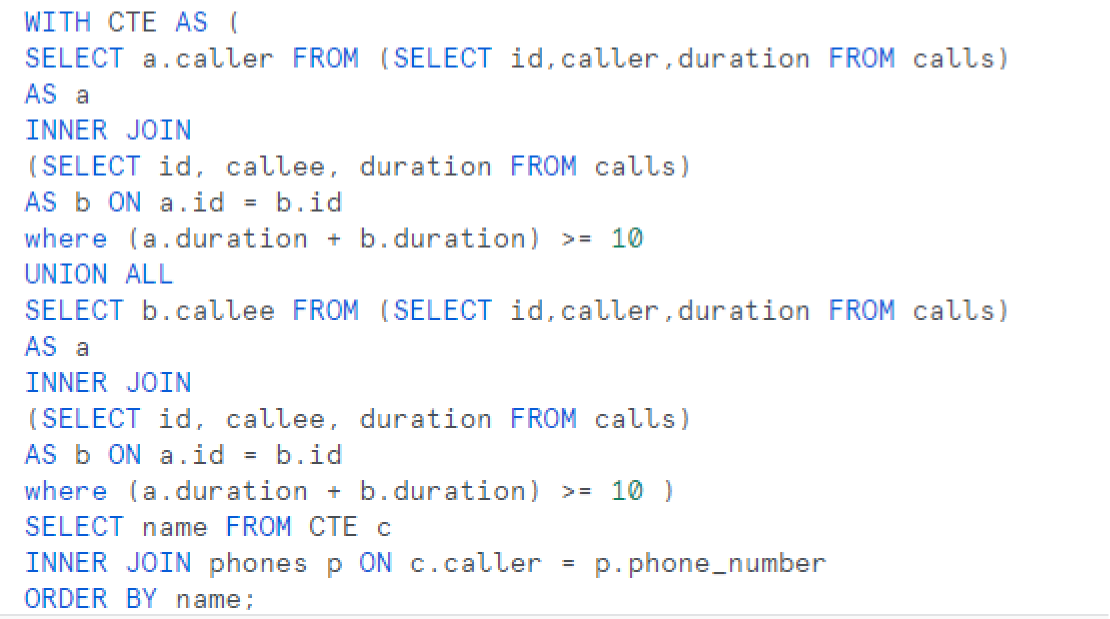
```
-- CREATE TABLE phones and calls
CREATE OR REPLACE TABLE phones
(name varchar(20) not null unique,
phone_number integer not null unique );

CREATE TABLE calls
(id integer not null,
caller integer not null,
callee integer not null,
duration integer not null,
unique(id));
```

Step 2. Insert values in phones and calls table



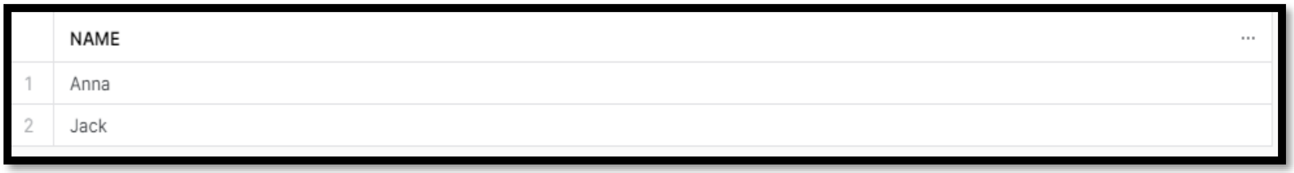
```
-- INSERT values in phones and calls
INSERT INTO phones VALUES
('Jack', 1234),
('Lena', 3333),
('Mark', 9999),
('Anna', 7582);

INSERT INTO calls VALUES
(25, 1234, 7582, 8),
(7, 9999, 7582, 1),
(18, 9999, 3333, 4),
(2, 7582, 3333, 3),
(3, 3333, 1234, 1),
(21, 3333, 1234, 1);
```

Step 3. Get the callers name who has spoken on call for at least 10 min. Here we are using CTE and Inner Join to get the desired result.



```
WITH CTE AS (
SELECT a.caller FROM (SELECT id,caller,duration FROM calls)
AS a
INNER JOIN
(SELECT id, callee, duration FROM calls)
AS b ON a.id = b.id
where (a.duration + b.duration) >= 10
UNION ALL
SELECT b.callee FROM (SELECT id,caller,duration FROM calls)
AS a
INNER JOIN
(SELECT id, callee, duration FROM calls)
AS b ON a.id = b.id
where (a.duration + b.duration) >= 10 )
SELECT name FROM CTE c
INNER JOIN phones p ON c.caller = p.phone_number
ORDER BY name;
```

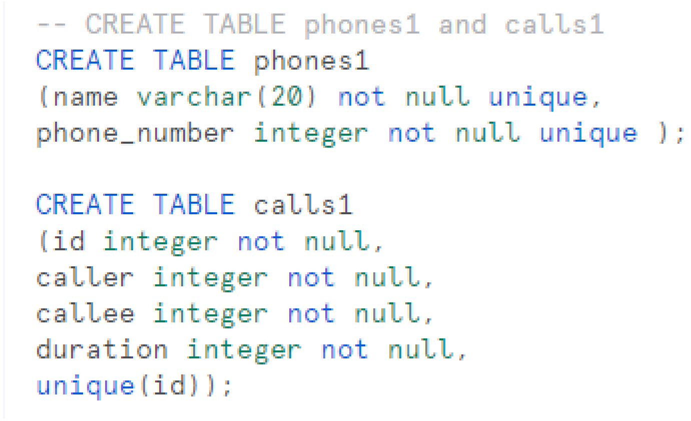
Hence, we get the two callers names who have spoken for at least 10 min



	NAME
1	Anna
2	Jack

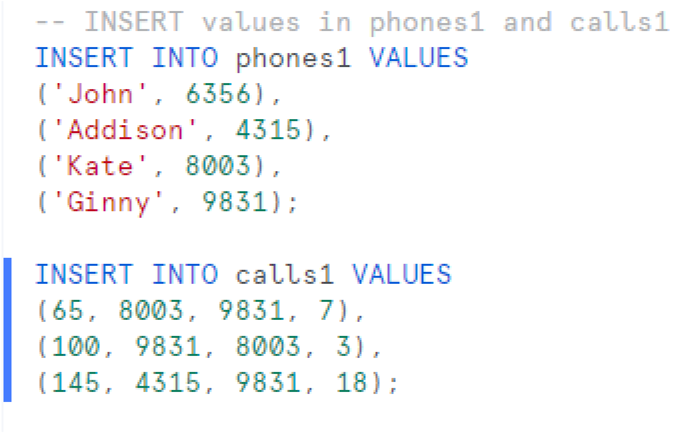
The second part of Task 2 is also related to what we have done above. Similarly let's follow the following steps:

Step 1. Create both phones1 and calls1 table



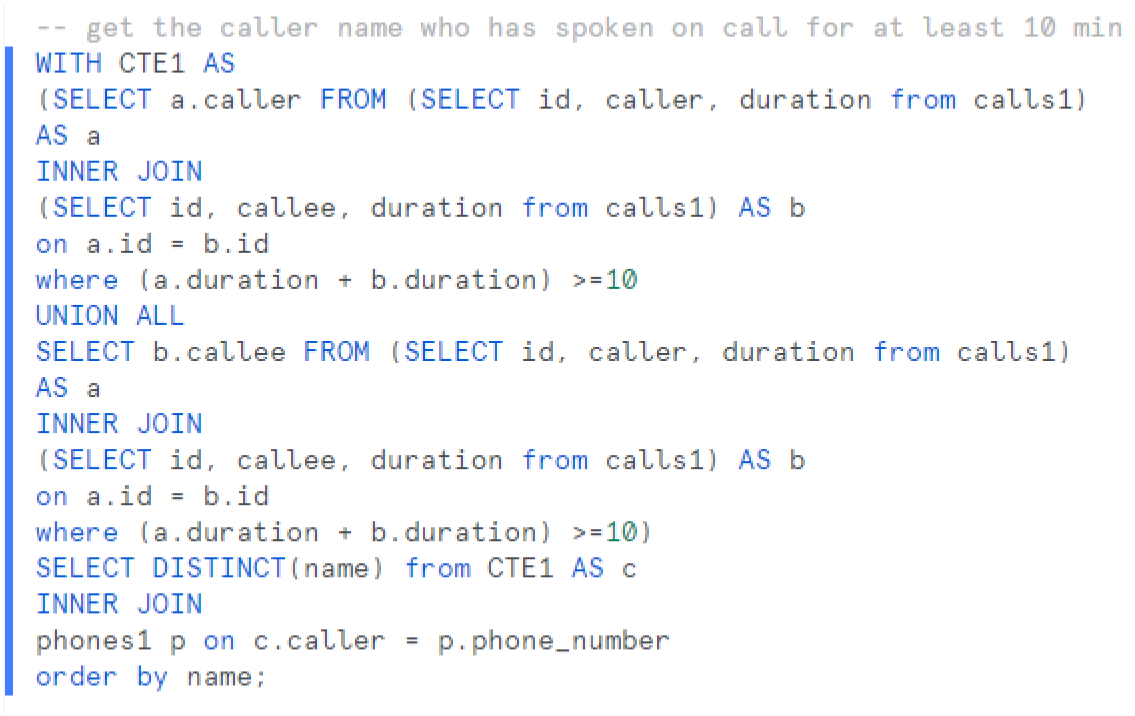
```
-- CREATE TABLE phones1 and calls1
CREATE TABLE phones1
(name varchar(20) not null unique,
phone_number integer not null unique );

CREATE TABLE calls1
(id integer not null,
caller integer not null,
callee integer not null,
duration integer not null,
unique(id));
```

Step 2. Insert values in phones1 and calls1 table



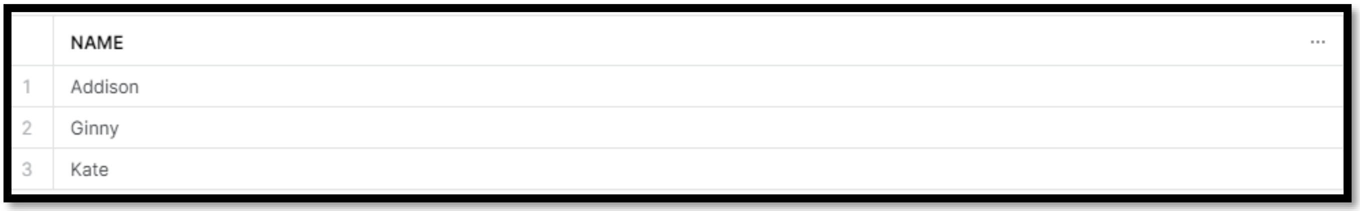
```
-- INSERT values in phones1 and calls1
INSERT INTO phones1 VALUES
('John', 6356),
('Addison', 4315),
('Kate', 8003),
('Ginny', 9831);

INSERT INTO calls1 VALUES
(65, 8003, 9831, 7),
(100, 9831, 8003, 3),
(145, 4315, 9831, 18);
```

Step 3. Get the callers name who has spoken on call for at least 10 min. Here we are using CTE and Inner Join to get the desired result.



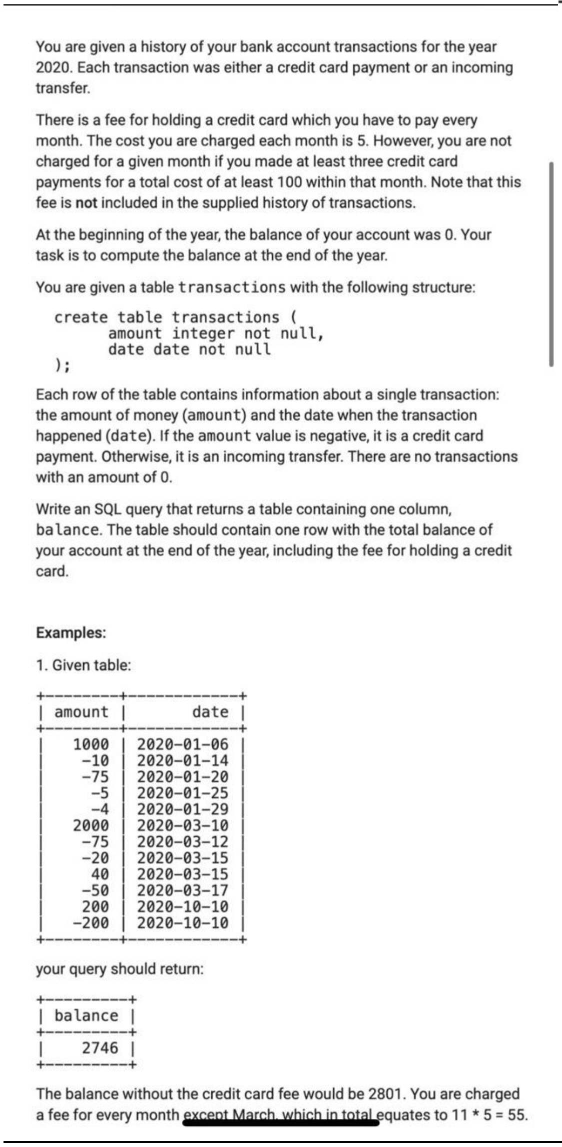
```
-- get the caller name who has spoken on call for at least 10 min
WITH CTE1 AS
(SELECT a.caller FROM (SELECT id, caller, duration from calls1)
AS a
INNER JOIN
(SELECT id, callee, duration from calls1) AS b
on a.id = b.id
where (a.duration + b.duration) >=10
UNION ALL
SELECT b.callee FROM (SELECT id, caller, duration from calls1)
AS a
INNER JOIN
(SELECT id, callee, duration from calls1) AS b
on a.id = b.id
where (a.duration + b.duration) >=10)
SELECT DISTINCT(name) from CTE1 AS c
INNER JOIN
phones1 p on c.caller = p.phone_number
order by name;
```

Hence, we get the three distinct callers names who have spoken for at least 10 min



	NAME
1	Addison
2	Ginny
3	Kate

Task 3: Output display is just one column balance



You are given a history of your bank account transactions for the year 2020. Each transaction was either a credit card payment or an incoming transfer.

There is a fee for holding a credit card which you have to pay every month. The cost you are charged each month is 5. However, you are not charged for a given month if you made at least three credit card payments for a total cost of at least 100 within that month. Note that this fee is **not** included in the supplied history of transactions.

At the beginning of the year, the balance of your account was 0. Your task is to compute the balance at the end of the year.

You are given a table `transactions` with the following structure:

```
create table transactions (
      amount integer not null,
      date date not null
);
```

Each row of the table contains information about a single transaction: the amount of money (`amount`) and the date when the transaction happened (`date`). If the `amount` value is negative, it is a credit card payment. Otherwise, it is an incoming transfer. There are no transactions with an amount of 0.

Write an SQL query that returns a table containing one column, `balance`. The table should contain one row with the total balance of your account at the end of the year, including the fee for holding a credit card.

Examples:

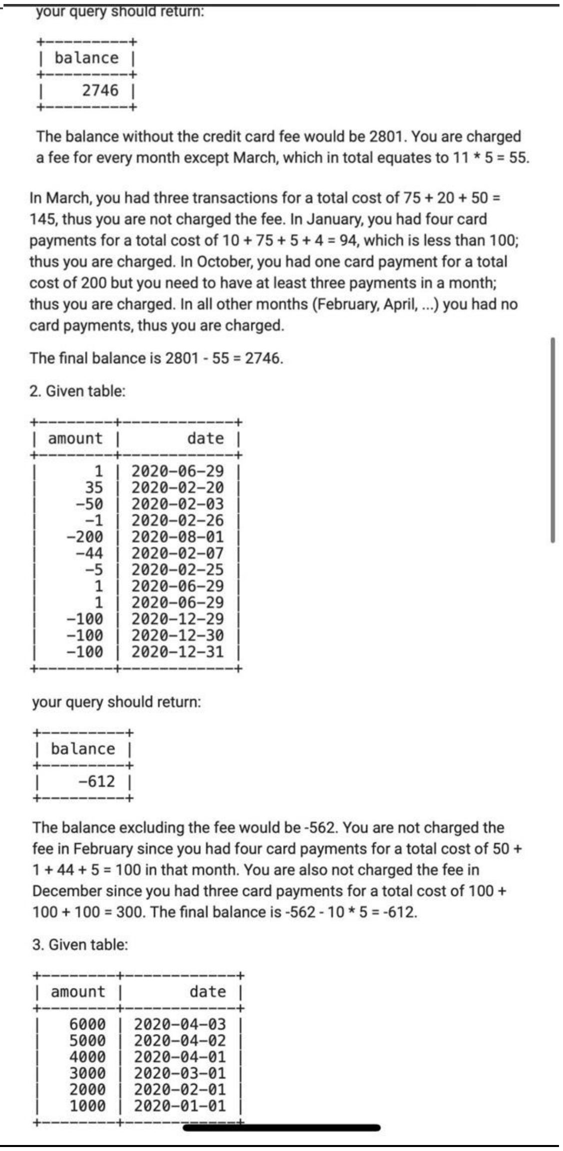
1. Given table:

```
+--------+------------+
| amount |       date |
+--------+------------+
|   1000 | 2020-01-06 |
|    -10 | 2020-01-14 |
|    -75 | 2020-01-20 |
|     -5 | 2020-01-25 |
|     -4 | 2020-01-29 |
|   2000 | 2020-03-10 |
|    -75 | 2020-03-12 |
|    -20 | 2020-03-15 |
|     40 | 2020-03-15 |
|    -50 | 2020-03-17 |
|    200 | 2020-10-10 |
|   -200 | 2020-10-10 |
+--------+------------+
```

your query should return:

```
+---------+
| balance |
+---------+
|    2746 |
+---------+
```

The balance without the credit card fee would be 2801. You are charged a fee for every month except March, which in total equates to 11 * 5 = 55.



your query should return:

```
+---------+
| balance |
+---------+
|    2746 |
+---------+
```

The balance without the credit card fee would be 2801. You are charged a fee for every month except March, which in total equates to 11 * 5 = 55.

In March, you had three transactions for a total cost of 75 + 20 + 50 = 145, thus you are not charged the fee. In January, you had four card payments for a total cost of 10 + 75 + 5 + 4 = 94, which is less than 100; thus you are charged. In October, you had one card payment for a total cost of 200 but you need to have at least three payments in a month; thus you are charged. In all other months (February, April, ...) you had no card payments, thus you are charged.

The final balance is 2801 - 55 = 2746.

2. Given table:

```
+--------+------------+
| amount |       date |
+--------+------------+
|      1 | 2020-06-29 |
|     35 | 2020-02-20 |
|    -50 | 2020-02-03 |
|     -1 | 2020-02-26 |
|   -200 | 2020-08-01 |
|    -44 | 2020-02-07 |
|     -5 | 2020-02-25 |
|      1 | 2020-06-29 |
|      1 | 2020-06-29 |
|   -100 | 2020-12-29 |
|   -100 | 2020-12-30 |
|   -100 | 2020-12-31 |
+--------+------------+
```

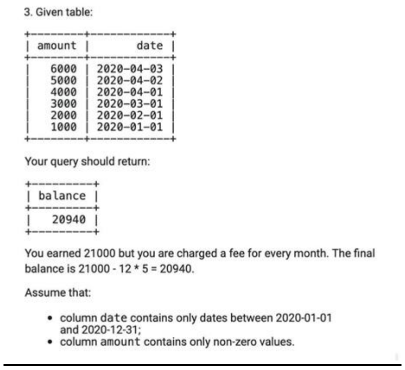
your query should return:

```
+---------+
| balance |
+---------+
|    -612 |
+---------+
```

The balance excluding the fee would be -562. You are not charged the fee in February since you had four card payments for a total cost of 50 + 1 + 44 + 5 = 100 in that month. You are also not charged the fee in December since you had three card payments for a total cost of 100 + 100 + 100 = 300. The final balance is -562 - 10 * 5 = -612.

3. Given table:

```
+--------+------------+
| amount |       date |
+--------+------------+
|   6000 | 2020-04-03 |
|   5000 | 2020-04-02 |
|   4000 | 2020-04-01 |
|   3000 | 2020-03-01 |
|   2000 | 2020-02-01 |
|   1000 | 2020-01-01 |
+--------+------------+
```



3. Given table:

```
+--------+------------+
| amount |       date |
+--------+------------+
|   6000 | 2020-04-03 |
|   5000 | 2020-04-02 |
|   4000 | 2020-04-01 |
|   3000 | 2020-03-01 |
|   2000 | 2020-02-01 |
|   1000 | 2020-01-01 |
+--------+------------+
```

Your query should return:

```
+---------+
| balance |
+---------+
|   20940 |
+---------+
```

You earned 21000 but you are charged a fee for every month. The final balance is 21000 - 12 * 5 = 20940.

Assume that:

- column `date` contains only dates between 2020-01-01 and 2020-12-31;
- column `amount` contains only non-zero values.

Answer:

Note: 1.Fee for holding a credit card which you have to pay every month is 5.

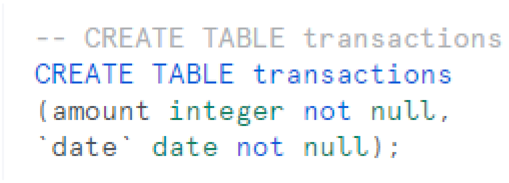
Note: 2. You are not charged for a given month if you made at least 3 credit card payments for a total cost of at least 100 within that month.

Note: 3. The fees is not included in the supplied history of transactions

Note: 4. At the beginning of the year the balance of the account was 0

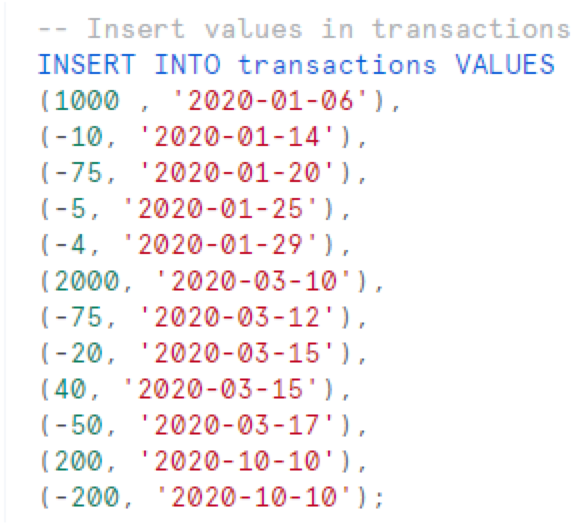
Task is to compute the balance at the end of the year

Step 1. Create table named transactions



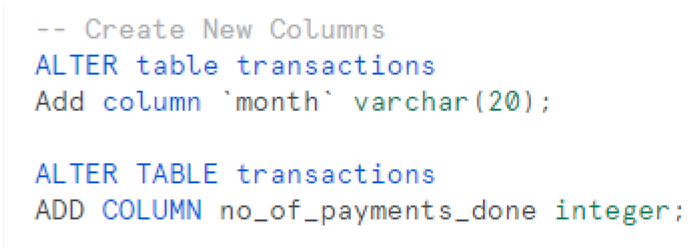
```
-- CREATE TABLE transactions
CREATE TABLE transactions
(amount integer not null,
`date` date not null);
```

Step 2. Insert values in transactions table



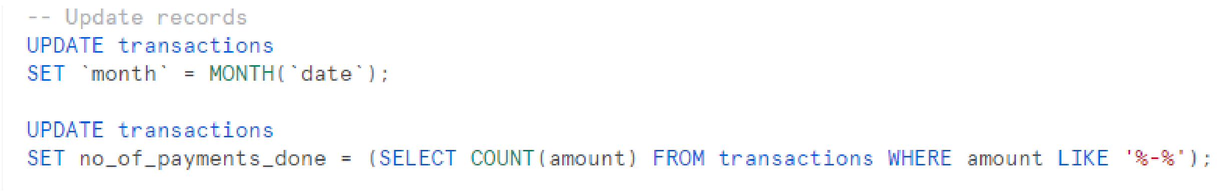
```
-- Insert values in transactions
INSERT INTO transactions VALUES
(1000 , '2020-01-06'),
(-10, '2020-01-14'),
(-75, '2020-01-20'),
(-5, '2020-01-25'),
(-4, '2020-01-29'),
(2000, '2020-03-10'),
(-75, '2020-03-12'),
(-20, '2020-03-15'),
(40, '2020-03-15'),
(-50, '2020-03-17'),
(200, '2020-10-10'),
(-200, '2020-10-10');
```

Step 3. Create New Columns



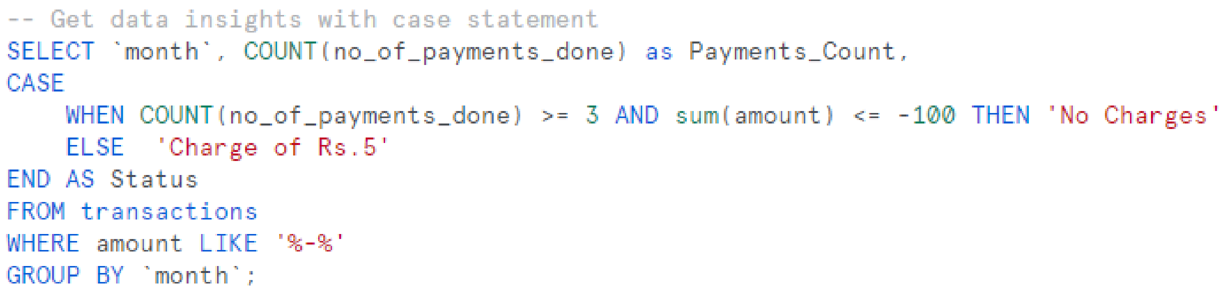
```
-- Create New Columns
ALTER table transactions
Add column `month` varchar(20);

ALTER TABLE transactions
ADD COLUMN no_of_payments_done integer;
```

Step 4. Update records in new columns



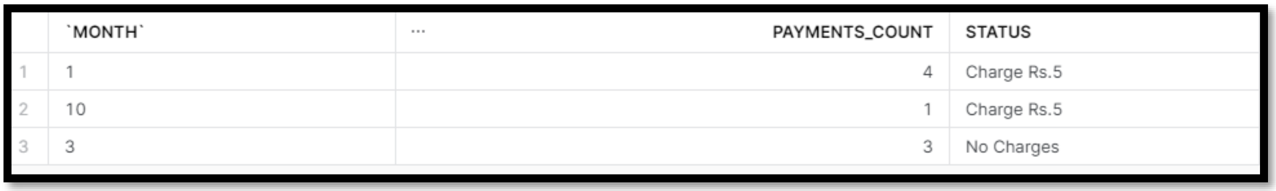
```
-- Update records
UPDATE transactions
SET `month` = MONTH(`date`);

UPDATE transactions
SET no_of_payments_done = (SELECT COUNT(amount) FROM transactions WHERE amount LIKE '%-%');
```

Step 5. Get insights from the case statement



```
-- Get data insights with case statement
SELECT `month`, COUNT(no_of_payments_done) as Payments_Count,
CASE
    WHEN COUNT(no_of_payments_done) >= 3 AND sum(amount) <= -100 THEN 'No Charges'
    ELSE  'Charge of Rs.5'
END AS Status
FROM transactions
WHERE amount LIKE '%-%'
GROUP BY `month`;
```

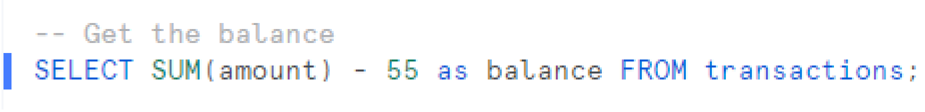
It Shows following result



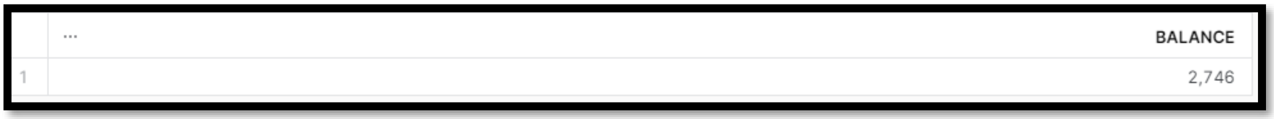
	`MONTH`	...	PAYMENTS_COUNT	STATUS
1	1		4	Charge Rs.5
2	10		1	Charge Rs.5
3	3		3	No Charges

This clearly shows that only month of march there will be no charges while other months will be chargeable i.e. 11 X 5 = 55

Step 6: Get the Balance



```
-- Get the balance
SELECT SUM(amount) - 55 as balance FROM transactions;
```

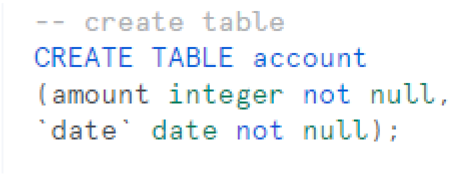


	...	BALANCE
1		2,746

Balance is 2746 after including the fees at the end of the year.

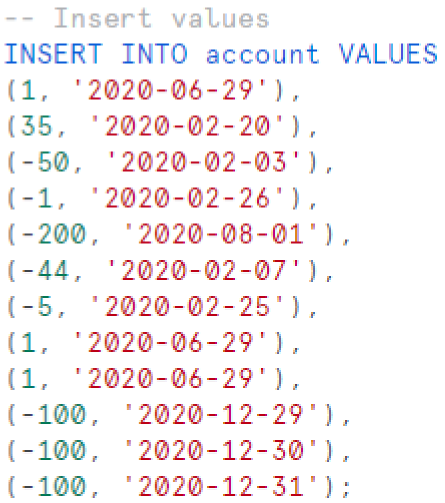
The task below is same as we have done above with a different data lets follow the steps below:

Step 1. Create table named account



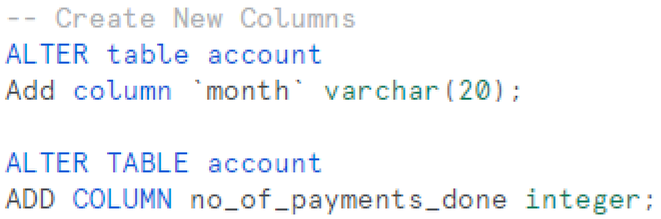
```
-- create table
CREATE TABLE account
(amount integer not null,
`date` date not null);
```

Step 2. Insert values in account table



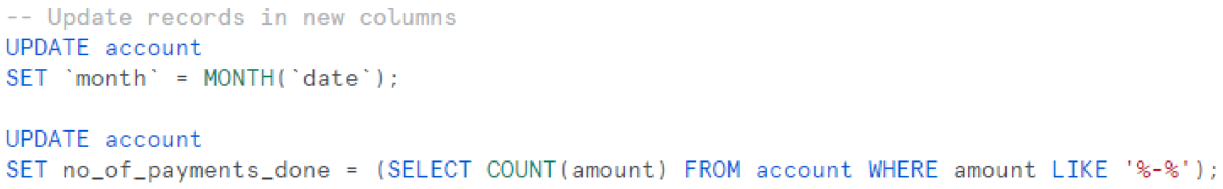
```
-- Insert values
INSERT INTO account VALUES
(1, '2020-06-29'),
(35, '2020-02-20'),
(-50, '2020-02-03'),
(-1, '2020-02-26'),
(-200, '2020-08-01'),
(-44, '2020-02-07'),
(-5, '2020-02-25'),
(1, '2020-06-29'),
(1, '2020-06-29'),
(-100, '2020-12-29'),
(-100, '2020-12-30'),
(-100, '2020-12-31');
```

Step 3. Create New Columns



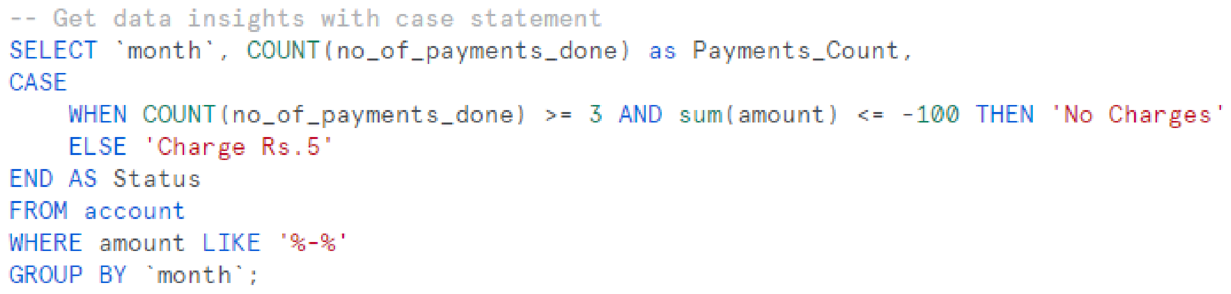
```
-- Create New Columns
ALTER table account
Add column `month` varchar(20);

ALTER TABLE account
ADD COLUMN no_of_payments_done integer;
```

Step 4: Update records in new columns



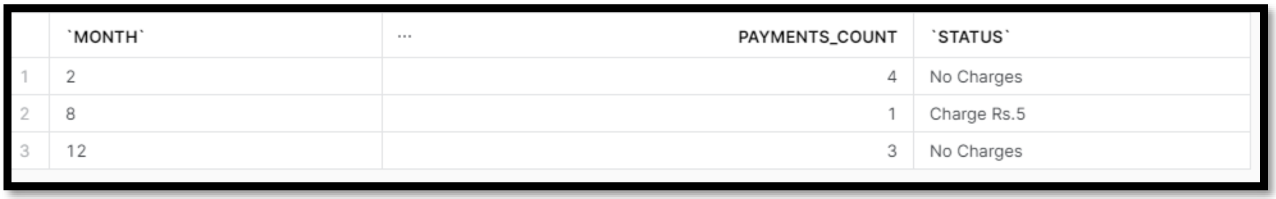
```
-- Update records in new columns
UPDATE account
SET `month` = MONTH(`date`);

UPDATE account
SET no_of_payments_done = (SELECT COUNT(amount) FROM account WHERE amount LIKE '%-%');
```

Step 5. Get insights from the case statement



```
-- Get data insights with case statement
SELECT `month`, COUNT(no_of_payments_done) as Payments_Count,
CASE
    WHEN COUNT(no_of_payments_done) >= 3 AND sum(amount) <= -100 THEN 'No Charges'
    ELSE 'Charge Rs.5'
END AS Status
FROM account
WHERE amount LIKE '%-%'
GROUP BY `month`;
```

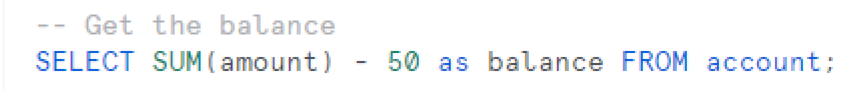
It Shows following result



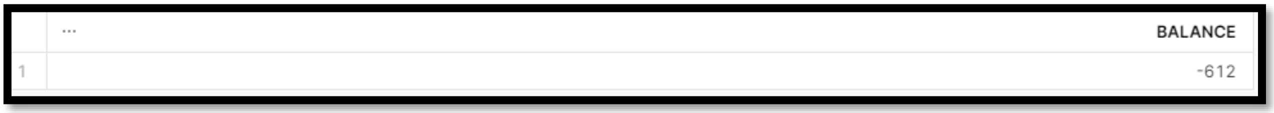
	`MONTH`	···	PAYMENTS_COUNT	`STATUS`
1	2		4	No Charges
2	8		1	Charge Rs.5
3	12		3	No Charges

This clearly shows that February and December there will be no charges while other months will be chargable i.e. 10 X 5 = 50

Step 6: Get the Balance



```
-- Get the balance
SELECT SUM(amount) - 50 as balance FROM account;
```

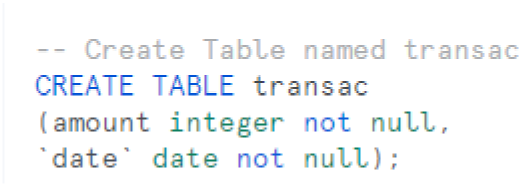


	···	BALANCE
1		-612

Balance is -612 after including the fees at the end of the year.

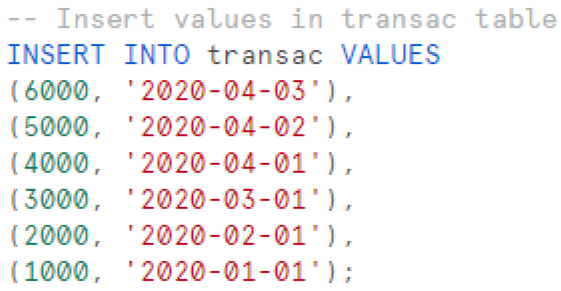
The final task is also somewhat similar but in this there are no negative values. Lets start the following steps:

Step 1. Create table transac account



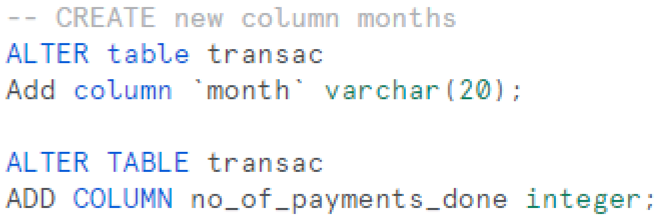
```
-- Create Table named transac
CREATE TABLE transac
(amount integer not null,
`date` date not null);
```

Step 2. Insert values in transac table



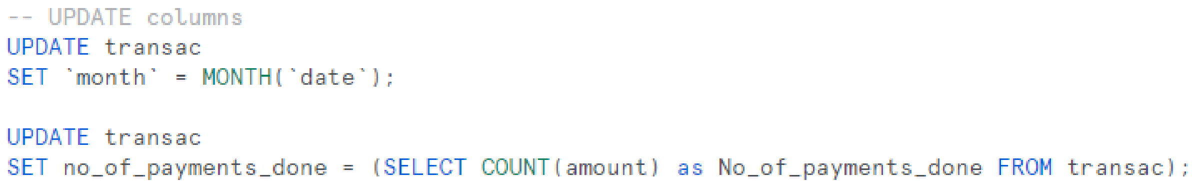
```
-- Insert values in transac table
INSERT INTO transac VALUES
(6000, '2020-04-03'),
(5000, '2020-04-02'),
(4000, '2020-04-01'),
(3000, '2020-03-01'),
(2000, '2020-02-01'),
(1000, '2020-01-01');
```

Step 3. Create New Columns



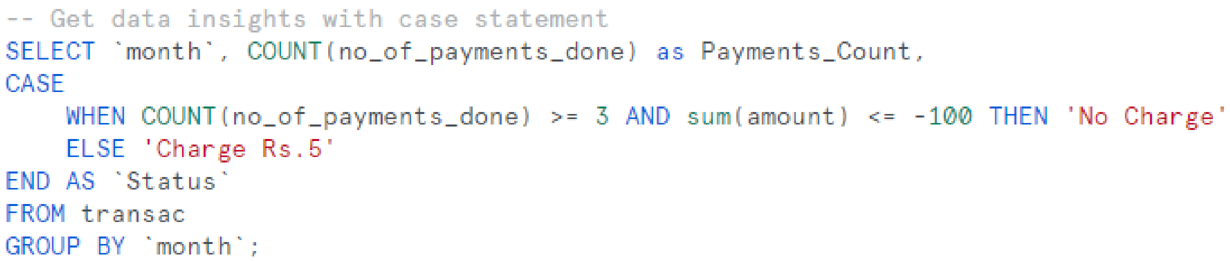
```
-- CREATE new column months
ALTER table transac
Add column `month` varchar(20);

ALTER TABLE transac
ADD COLUMN no_of_payments_done integer;
```

Step 4: Update records in new columns



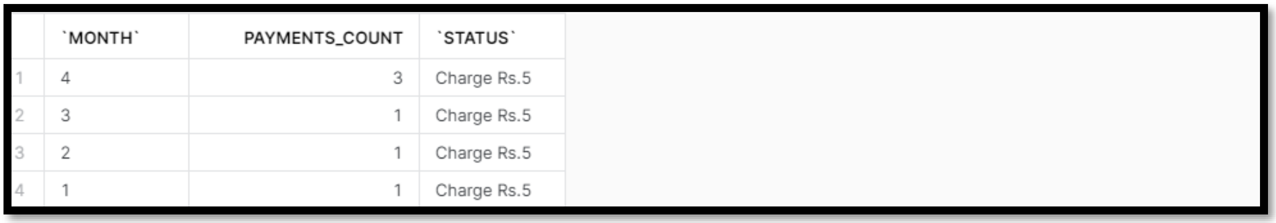
```
-- UPDATE columns
UPDATE transac
SET `month` = MONTH(`date`);

UPDATE transac
SET no_of_payments_done = (SELECT COUNT(amount) as No_of_payments_done FROM transac);
```

Step 5. Get insights from the case statement



```
-- Get data insights with case statement
SELECT `month`, COUNT(no_of_payments_done) as Payments_Count,
CASE
    WHEN COUNT(no_of_payments_done) >= 3 AND sum(amount) <= -100 THEN 'No Charge'
    ELSE 'Charge Rs.5'
END AS `Status`
FROM transac
GROUP BY `month`;
```

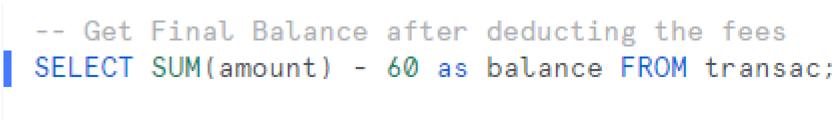
It Shows following result



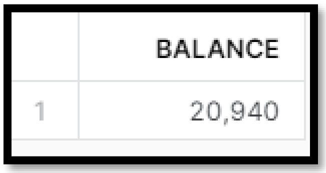
	`MONTH`	PAYMENTS_COUNT	`STATUS`
1	4	3	Charge Rs.5
2	3	1	Charge Rs.5
3	2	1	Charge Rs.5
4	1	1	Charge Rs.5

This clearly shows that all months there will be charges of Rs.5 i.e. 12 x 5 = 60.

Step 6: Get the Balance



```
-- Get Final Balance after deducting the fees
SELECT SUM(amount) - 60 as balance FROM transac;
```



	BALANCE
1	20,940

Balance is 20,940 after including the fees at the end of the year.