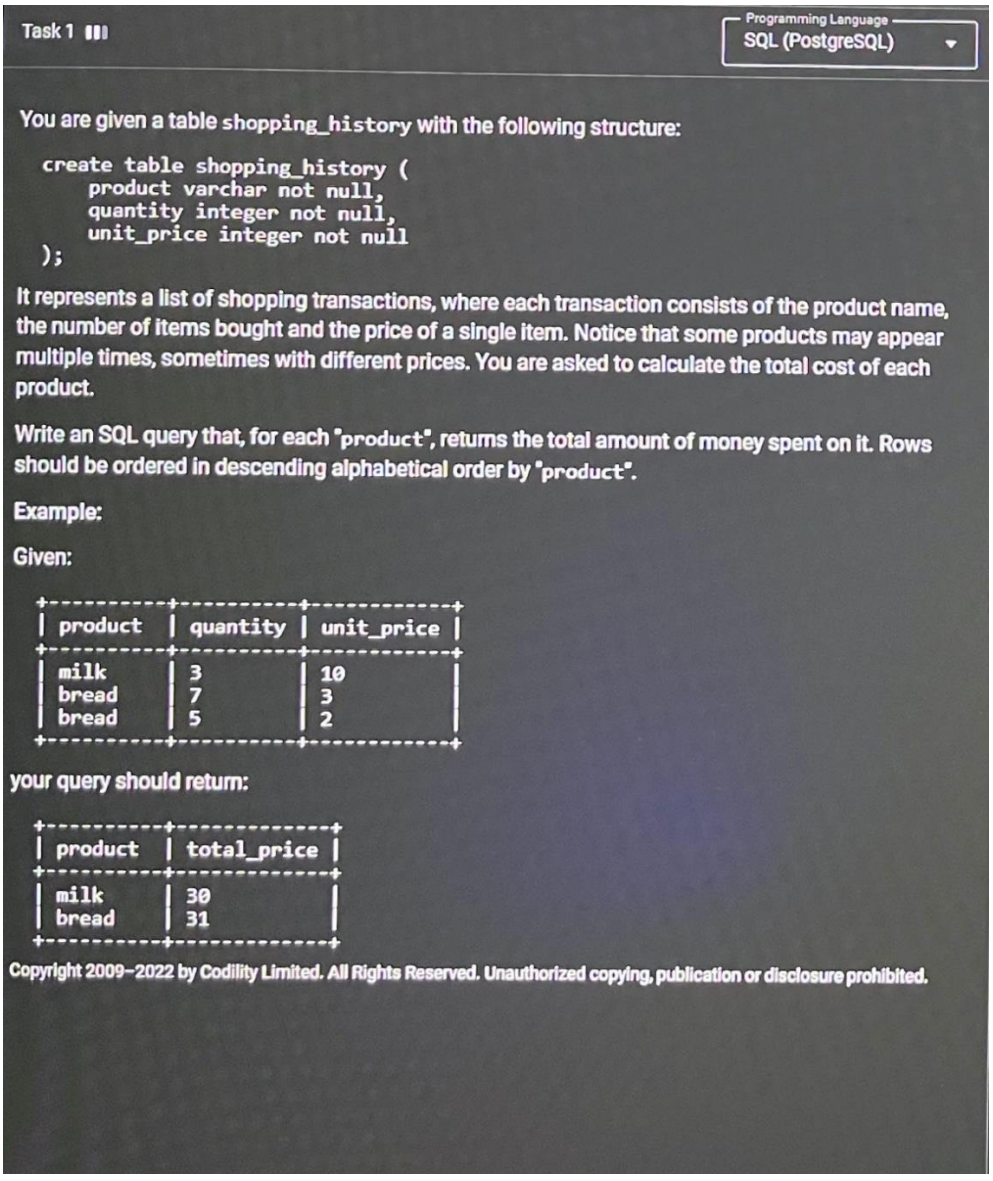
ASSIGNMENT 1

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:



Task 1

Programming Language: SQL (PostgreSQL)

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 first create warehouse and database

```
Step 1. Create warehouse and use it

CREATE WAREHOUSE MySQL_Assignments;
USE MySQL_Assignments;

Step 2. Create database and use it

CREATE DATABASE MySQL;
USE MySQL;

Step 3. Create a table named shopping_history

-- create a table named as shopping_history
CREATE TABLE shopping_history(
product varchar not null,
quantity integer not null,
unit_price integer not null
);

Step 4. Insert values into shopping_history table

-- insert values into shopping_history
INSERT INTO shopping_history values('milk',3,10);
INSERT INTO shopping_history values('bread',7,3);
INSERT INTO shopping_history values('bread',5,2);

SELECT * FROM shopping_history;
```

	PRODUCT	QUANTITY	UNIT_PRICE
1	milk	3	10
2	bread	7	3
3	bread	5	2

Step 5. We would require total_price so will add new column named as total_price

```
-- add new column named as 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;
```

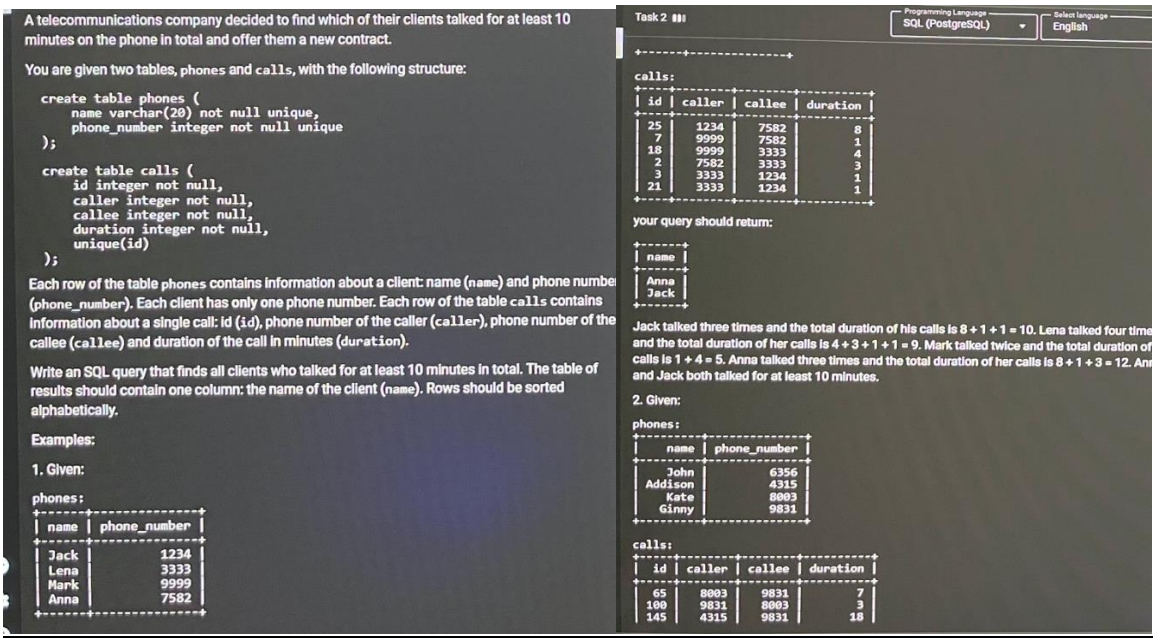
Step 7.

```
-- Get the total_price based on product group
SELECT product,sum(total_price) AS TOTAL_PRICE FROM
shopping_history
GROUP BY product;
```

The output which we get is the desired result

	PRODUCT	SUM(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 number (`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 |
+------+--------------+

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:

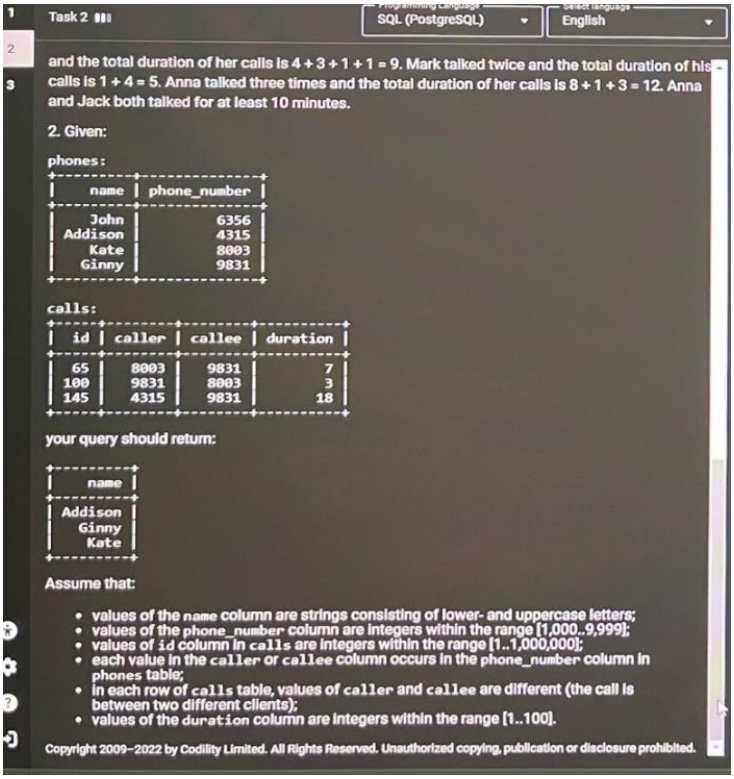
```
+------+
| name |
+------+
| Anna |
| Jack |
+------+
```

Jack talked three times and the total duration of his calls is 8 + 1 + 1 = 10. Lena talked four times 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. 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 |
+-----+--------+--------+----------+
```



Task 2

SQL (PostgreSQL) | 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 > = 100 which are made. Following are the steps need to be perform

Step 1. Create both phones and calls table

```
-- use database MySQL
USE MySQL;


-- creating table phones
CREATE TABLE phones(
name varchar(20) NOT NULL UNIQUE,
phone_number integer NOT NULL UNIQUE
);

-- creating table calls
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

```
-- inserting values into table phones
Insert into phones values ('Jack', 1234);
Insert into phones values('Lena', 3333);
Insert into phones values('Mark', 9999);
Insert into phones values('Anna', 7582);

-- inserting values into table calls
Insert into calls values (25,1234,7582,8);
Insert into calls values (7,9999,7582,1);
Insert into calls values (18, 9999,3333, 4);
Insert into calls values (2, 7582, 3333, 3);
Insert into calls values (3, 3333, 1234, 1);
Insert into calls values (21, 3333, 1234, 1);


select * from phones;
```

	NAME	...	PHONE_NUMBER
1	Jack		1,234
2	Lena		3,333
3	Mark		9,999
4	Anna		7,582

```
select * from calls;
```

	ID	CALLER	...	CALLEE	DURATION
1	25	1,234		7,582	8
2	7	9,999		7,582	1
3	18	9,999		3,333	4
4	2	7,582		3,333	3
5	3	3,333		1,234	1
6	21	3,333		1,234	1

Step 3. Here we are using CTE, Inner Join and Union all to get the desired results. This query will help us to get the caller names who spoke more than or equal to 10 minutes.

```
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;
```

	NAME
1	Jack
2	Anna

The second part of task to is similar to task performed above. Let's check out the steps:

```
Step 1. Create and use database
-- Create database and use it
reate database task2b;
use task2b;

Step 2. Create both phones1 and calls1 table
-- creating table phone
create table phones1(
name varchar(20) not null unique,
phone_number integer not null unique
);

-- creating table calls
create table calls1(
id integer not null,
caller integer not null,
callee integer not null,
duration integer not null,
unique(id)
);
```

```
select * from phones1;
```

	NAME	...	PHONE_NUMBER
1	John		6,356
2	Addison		4,315
3	Kate		8,003
4	Ginny		9,831

```
select * from calls1;
```

	ID	···	CALLER	CALLEE	DURATION
1	65		8,003	9,831	7
2	100		9,831	8,003	3
3	145		4,315	9,831	18

Step 3. Here we are using CTE, Inner Join and Union all to get the desired results. This query will help us to get the caller names who spoke more than or equal to 10 minutes.

```
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 c
INNER JOIN phones1 p ON c.caller = p.phone_number;
```


	NAME
1	Kate
2	Addison
3	Ginny

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.

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.

You can add the following data in the table

```
CREATE TABLE transactions(
    Amount INTEGER  NOT NULL
  ,Date   DATE  NOT NULL
);
INSERT INTO transactions(Amount,Date) VALUES (1000,'2020-01-06');
INSERT INTO transactions(Amount,Date) VALUES (-10,'2020-01-14');
INSERT INTO transactions(Amount,Date) VALUES (-75,'2020-01-20');
INSERT INTO transactions(Amount,Date) VALUES (-5,'2020-01-25');
INSERT INTO transactions(Amount,Date) VALUES (-4,'2020-01-29');
INSERT INTO transactions(Amount,Date) VALUES (2000,'2020-03-10');
INSERT INTO transactions(Amount,Date) VALUES (-75,'2020-03-12');
INSERT INTO transactions(Amount,Date) VALUES (-20,'2020-03-15');
INSERT INTO transactions(Amount,Date) VALUES (40,'2020-03-15');
INSERT INTO transactions(Amount,Date) VALUES (-50,'2020-03-17');
INSERT INTO transactions(Amount,Date) VALUES (200,'2020-10-10');
INSERT INTO transactions(Amount,Date) VALUES (-200,'2020-10-10');

```

```
-- Task 3.1
Step 1. Create and use database
-- Create and use database
create database task2c;
use task2c;

Step 2. Create table and insert values in table transactions
-- create table transactions and insert values
create table transactions (
amount integer not null,
date date not null
);
```

```
Step 3 Add columns in transactions table and update values
-- add columns and add values in transactions table

Alter table transactions
add column `Month` varchar(20);

Update transactions
set `Month` = Month(date);

Alter table transactions
add column no_of_payment_done integer;

Update transactions
set no_of_payment_done = (Select count(amount) from transactions where amount LIKE '%-%');
```

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

select * from transactions;
```

```sql
Update transactions
set `Month` = Month(date);

Alter table transactions
add column no_of_payment_done integer;

Update transactions
set no_of_payment_done = (Select count(amount) from transactions where amount LIKE '%-%');


 Step 3 Add columns in transactions table and update values
 -- add columns and add values in transactions table

 Alter table transactions
 add column `Month` varchar(20);

 Update transactions
 set `Month` = Month(date);


Step 4 Case statement to get the require months where charges are not levied
--Case statement to get the require months where charges are not levied
Select `month`, Count(no_of_payment_done) as Payments,
CASE
 WHEN COUNT(no_of_payment_done)>=3 AND sum(amount)<=-100 THEN 'NO CHARGES'
 ELSE 'Charge RS. 5'
END AS CHARGES
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 5 Query to get the desired results
-- query to get total amount
Select sum(amount) - 55 as Balance from transactions;
```

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.

```sql
--Task 3.2
Step 1 Create and use database
-- create and use database
Create database transact_task;
Use transact_task;

Step 2 Creating and inserting values in table
-- create table name as transactions1 and insert values
create table transactions1 (
amount integer not null,
date date not null
);
```

```sql
-- Inserting values into transactions1 table
Insert into transactions1 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 Alter table and add require columns

```sql
-- alter table to add require columns
ALTER TABLE transactions1
ADD COLUMN `Month` varchar(20);

ALTER TABLE transactions1
ADD COLUMN no_of_payments_done integer;
```

Step. 4 Updating values in new columns

```sql
-- Update `Month`and no_of_payments_done table
UPDATE transactions1
SET `Month` = MONTH(DATE);

UPDATE transactions1
SET no_of_payments_done = (SELECT COUNT(amount) FROM transactions1 WHERE amount LIKE '%-%');
```

Step 4 Case statement to get the require months where charges are not levied

```sql
--Case statement to get the require months where charges are not levied
SELECT `MONTH`, COUNT(NO_OF_PAYMENTS_DONE) AS Payment_COUNT,
CASE
    WHEN COUNT(NO_OF_PAYMENTS_DONE) >=3 AND SUM(amount) <=100 THEN 'NO CHARGES'
    ELSE 'CHARGES'
END AS Status
FROM transactions1
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

```
-- shows that there were no charges levied for the month of February and December
--while other months will be charged ie. 10*5 = 50

SELECT * FROM transactions1;

Step 5 Query to get the require results
-- query to get the desired results
SELECT SUM(AMOUNT) - 50 AS BALANCE FROM transactions1;
```

	... 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. Let's start the following steps

```
-- Task 3.3
Step 1. Create and use database
Create database transact_task1;
Use transact_task1;

Step 2. Create table and insert values into table
-- creating table name as transactions_1
create table transactions_1 (
amount integer not null,
date date not null
);
```

```
Insert into transactions_1 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. Add require columns
-- add require columns

ALTER TABLE transactions_1
ADD COLUMN `MONTH` varchar(20);

ALTER TABLE transactions_1
ADD COLUMN no_of_payments_done integer;
```

```
Step 4. Update values in table
-- update values in new columns
UPDATE transactions_1
SET `MONTH` = MONTH(DATE);

UPDATE transactions_1
SET no_of_payments_done = (SELECT COUNT(amount) FROM transactions_1);

alter table transactions 1
```

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`;
```

	`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.