SQL Assignments -2

Note: use same database from assignment 1.

Qn 1:

Schema to be used:

```
Create table If Not Exists Logs (id int, num int);
insert into Logs (id, num) values ('1', '1');
insert into Logs (id, num) values ('2', '1');
insert into Logs (id, num) values ('3', '1');
insert into Logs (id, num) values ('4', '2');
insert into Logs (id, num) values ('5', '1');
insert into Logs (id, num) values ('6', '2');
insert into Logs (id, num) values ('7', '2');
```

```
+-------------+---------+
| Column Name | Type    |
+-------------+---------+
| id          | int     |
| num         | varchar |
+-------------+---------+
```

id is the primary key for this table.
id is an autoincrement column.

Write an SQL query to find all numbers that appear at least three times consecutively.

Return the result table in any order.

The query result format is in the following example.

Example 1:

Input:

Logs table:

```
+----+-----+
| id | num |
+----+-----+
| 1  | 1   |
| 2  | 1   |
```

```
| 3  | 1   |
| 4  | 2   |
| 5  | 1   |
| 6  | 2   |
| 7  | 2   |
+----+-----+
```

Output:

```
+-----------------+
| ConsecutiveNums |
+-----------------+
| 1               |
+-----------------+
```

Explanation: 1 is the only number that appears consecutively for at least three times.

Qn 2:

Schema to be used:

```
Create table If Not Exists Candidate (id int, name varchar(255));
Create table If Not Exists Vote (id int, candidateId int);
insert into Candidate (id, name) values ('1', 'A');
insert into Candidate (id, name) values ('2', 'B');
insert into Candidate (id, name) values ('3', 'C');
insert into Candidate (id, name) values ('4', 'D');
insert into Candidate (id, name) values ('5', 'E');
insert into Vote (id, candidateId) values ('1', '2');
insert into Vote (id, candidateId) values ('2', '4');
insert into Vote (id, candidateId) values ('3', '3');
insert into Vote (id, candidateId) values ('4', '2');
insert into Vote (id, candidateId) values ('5', '5');
```

```
+-------------+----------+
| Column Name | Type     |
+-------------+----------+
| id          | int      |
| name        | varchar  |
+-------------+----------+
```

id is the primary key column for this table.

Each row of this table contains information about the id and the name of a candidate.

Table: Vote

+-------------+------+
| Column Name | Type |
+-------------+------+
| id | int |
| candidateId | int |
+-------------+------+
id is an auto-increment primary key.
candidateId is a foreign key to id from the Candidate table.
Each row of this table determines the candidate who got the ith vote in the elections.

Write an SQL query to report the name of the winning candidate (i.e., the candidate who got the largest number of votes).

The test cases are generated so that exactly one candidate wins the elections.

The query result format is in the following example.

Example 1:

Input:
Candidate table:
+----+------+
| id | name |
+----+------+
1	A
2	B
3	C
4	D
5	E
+----+------+

Vote table:

```
+----+-------------+
| id | candidateId |
+----+-------------+
| 1  | 2           |
| 2  | 4           |
| 3  | 3           |
| 4  | 2           |
| 5  | 5           |
+----+-------------+
```

Output:

```
+------+
| name |
+------+
| B    |
+------+
```

Explanation:
Candidate B has 2 votes. Candidates C, D, and E have 1 vote each.
The winner is candidate B.

Qn 3:

Schema to be used:

```
Create table If Not Exists Student (student_id int,student_name varchar(45), gender varchar(6), dept_id int);
Create table If Not Exists Department (dept_id int, dept_name varchar(255));
insert into Student (student_id, student_name, gender, dept_id) values ('1', 'Jack', 'M', '1');
insert into Student (student_id, student_name, gender, dept_id) values ('2', 'Jane', 'F', '1');
insert into Student (student_id, student_name, gender, dept_id) values ('3', 'Mark', 'M', '2');

insert into Department (dept_id, dept_name) values ('1', 'Engineering');
insert into Department (dept_id, dept_name) values ('2', 'Science');
insert into Department (dept_id, dept_name) values ('3', 'Law');
```

able: Student

```
+--------------+---------+
| Column Name  | Type    |
+--------------+---------+
| student_id   | int     |
| student_name | varchar |
| gender       | varchar |
```

| dept_id | int |
+-------------+---------+
student_id is the primary key column for this table.
dept_id is a foreign key to dept_id in the Department tables.
Each row of this table indicates the name of a student, their gender, and the id of their department.

Table: Department
+-------------+---------+
| Column Name | Type |
+-------------+---------+
| dept_id | int |
| dept_name | varchar |
+-------------+---------+
dept_id is the primary key column for this table.
Each row of this table contains the id and the name of a department.

Write an SQL query to report the respective department name and number of students majoring in each department for all departments in the Department table (even ones with no current students).
Return the result table ordered by student_number in descending order. In case of a tie, order them by dept_name alphabetically.
The query result format is in the following example.

Example 1:
Input:
Student table:
+------------+--------------+--------+---------+
| student_id | student_name | gender | dept_id |
+------------+--------------+--------+---------+
1	Jack	M	1
2	Jane	F	1
3	Mark	M	2
+------------+--------------+--------+---------+
Department table:
+---------+-------------+
| dept_id | dept_name |
+---------+-------------+
| 1 | Engineering |
| 2 | Science |

| 3 | Law |
+---------+-------------+

Output:

+-------------+----------------+
| dept_name | student_number |
+-------------+----------------+
Engineering	2
Science	1
Law	0
+-------------+----------------+

Qn 4:

Schema to be used :

Create Table If Not Exists Insurance (pid int, tiv_2015 float, tiv_2016 float, lat float, lon float);
insert into Insurance (pid, tiv_2015, tiv_2016, lat, lon) values ('1', '10', '5', '10', '10');
insert into Insurance (pid, tiv_2015, tiv_2016, lat, lon) values ('2', '20', '20', '20', '20');
insert into Insurance (pid, tiv_2015, tiv_2016, lat, lon) values ('3', '10', '30', '20', '20');
insert into Insurance (pid, tiv_2015, tiv_2016, lat, lon) values ('4', '10', '40', '40', '40');

Table: Insurance

+-------------+-------+
| Column Name | Type |
+-------------+-------+
pid	int
tiv_2015	float
tiv_2016	float
lat	float
lon	float
+-------------+-------+

pid is the primary key column for this table.
Each row of this table contains information about one policy where:
pid is the policyholder's policy ID.
tiv_2015 is the total investment value in 2015 and tiv_2016 is the total investment value in 2016.
lat is the latitude of the policy holder's city.
lon is the longitude of the policy holder's city.

Write an SQL query to report the sum of all total investment values in 2016 tiv_2016, for all policyholders who:

- have the same tiv_2015 value as one or more other policyholders, and
- are not located in the same city like any other policyholder (i.e., the (lat, lon) attribute pairs must be unique).

Round tiv_2016 to two decimal places.
The query result format is in the following example.

Example 1:
Input:
Insurance table:

```
+-----+----------+----------+-----+-----+
| pid | tiv_2015 | tiv_2016 | lat | lon |
+-----+----------+----------+-----+-----+
| 1   | 10       | 5        | 10  | 10  |
| 2   | 20       | 20       | 20  | 20  |
| 3   | 10       | 30       | 20  | 20  |
| 4   | 10       | 40       | 40  | 40  |
+-----+----------+----------+-----+-----+
```

Output:

```
+----------+
| tiv_2016 |
+----------+
| 45.00    |
+----------+
```

Explanation:
The first record in the table, like the last record, meets both of the two criteria. The tiv_2015 value 10 is the same as the third and fourth records, and its location is unique.

The second record does not meet any of the two criteria. Its tiv_2015 is not like any other policyholders and its location is the same as the third record, which makes the third record fail, too.
So, the result is the sum of tiv_2016 of the first and last record, which is 45.

Qn 5:
Schema to be used:

```
Create Table If Not Exists Point2D (x int not null, y int not null);
insert into Point2D (x, y) values ('-1', '-1');
```

```
insert into Point2D (x, y) values ('0', '0');
insert into Point2D (x, y) values ('-1', '-2');
```

Table: Point2D

```
+-------------+------+
| Column Name | Type |
+-------------+------+
| x           | int  |
| y           | int  |
+-------------+------+
```

(x, y) is the primary key column for this table.
Each row of this table indicates the position of a point on the X-Y plane.

The distance between two points $p_1(x_1, y_1)$ and $p_2(x_2, y_2)$ is $\text{sqrt}((x_2 - x_1)^2 + (y_2 - y_1)^2)$.
Write an SQL query to report the shortest distance between any two points from the Point2D table. Round the distance to **two decimal points**.
The query result format is in the following example.

Example 1:
Input:
Point2D table:

```
+----+----+
| x  | y  |
+----+----+
| -1 | -1 |
| 0  | 0  |
| -1 | -2 |
+----+----+
```

Output:

```
+----------+
| shortest |
+----------+
| 1.00     |
+----------+
```

Explanation: The shortest distance is 1.00 from point (-1, -1) to (-1, 2).

Qn 6:

Schema to be used:

```
Create table If Not Exists Seat (id int, student varchar(255));
insert into Seat (id, student) values ('1', 'Abbot');
insert into Seat (id, student) values ('2', 'Doris');
insert into Seat (id, student) values ('3', 'Emerson');
insert into Seat (id, student) values ('4', 'Green');
insert into Seat (id, student) values ('5', 'Jeames');
```

Table: Seat

```
+-------------+---------+
| Column Name | Type    |
+-------------+---------+
| id          | int     |
| student     | varchar |
+-------------+---------+
```

id is the primary key column for this table.
Each row of this table indicates the name and the ID of a student.
id is a continuous increment.

Write an SQL query to swap the seat id of every two consecutive students. If the number of students is odd, the id of the last student is not swapped.
Return the result table ordered by id **in ascending order**.
The query result format is in the following example.

Example 1:
Input:
Seat table:

```
+----+---------+
| id | student |
+----+---------+
| 1  | Abbot   |
| 2  | Doris   |
| 3  | Emerson |
| 4  | Green   |
| 5  | Jeames  |
+----+---------+
```

Output:

```
+----+---------+
| id | student |
+----+---------+
| 1  | Doris   |
| 2  | Abbot   |
| 3  | Green   |
| 4  | Emerson |
| 5  | Jeames  |
+----+---------+
```

Explanation:
Note that if the number of students is odd, there is no need to change the last one's seat.

Qn 7 :

Schema to be used :

```
Create table If Not Exists Tree (id int, p_id int);
insert into Tree (id, p_id) values ('1', '0'');
insert into Tree (id, p_id) values ('2', '1');
insert into Tree (id, p_id) values ('3', '1');
insert into Tree (id, p_id) values ('4', '2');
insert into Tree (id, p_id) values ('5', '2');
```

Table: Tree

```
+-------------+------+
| Column Name | Type |
+-------------+------+
| id          | int  |
| p_id        | int  |
+-------------+------+
```

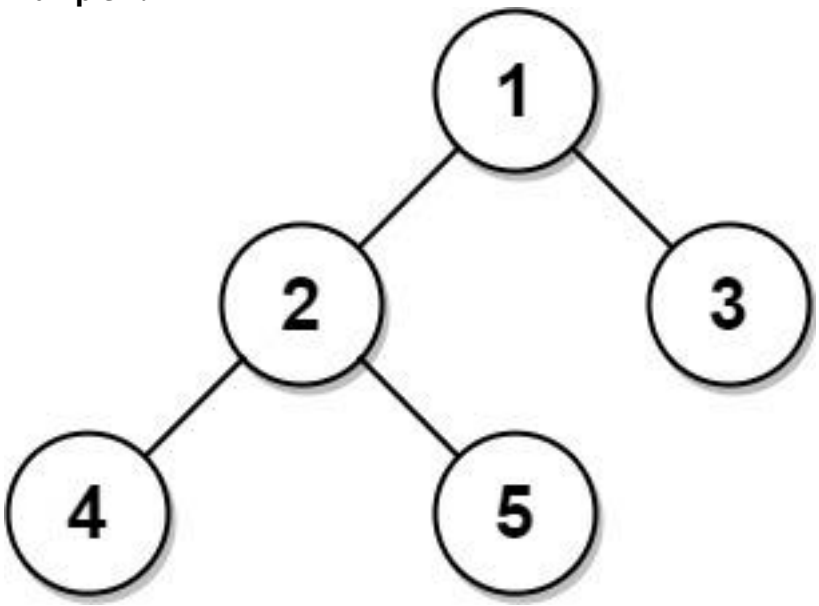
id is the primary key column for this table.
Each row of this table contains information about the id of a node and the id of its parent node in a tree.
The given structure is always a valid tree.

Each node in the tree can be one of three types:

- **"Leaf"**: if the node is a leaf node.
- **"Root"**: if the node is the root of the tree.
- **"Inner"**: If the node is neither a leaf node nor a root node.

Write an SQL query to report the type of each node in the tree.
Return the result table in **any order**.
The query result format is in the following example.

Example 1:



Input:

Tree table:

id	p_id
1	null
2	1
3	1
4	2
5	2

Output:

id	type
1	Root
2	Inner
3	Leaf
4	Leaf
5	Leaf

Explanation:
Node 1 is the root node because its parent node is null and it has child nodes 2 and 3.
Node 2 is an inner node because it has parent node 1 and child node 4 and 5.
Nodes 3, 4, and 5 are leaf nodes because they have parent nodes and they do not have child nodes.

Example 2:

Input:
Tree table:

id	p_id
1	null

Output:

id	type
1	Root

Explanation: If there is only one node on the tree, you only need to output its root attributes.

Qn 8 :

Schema to be used:

Create table If Not Exists Users (user_id int, user_name varchar(20), credit int);

Create table If Not Exists Transactions (trans_id int, paid_by int, paid_to int, amount int, transacted_on date);
insert into Users (user_id, user_name, credit) values ('1', 'Moustafa', '100');
insert into Users (user_id, user_name, credit) values ('2', 'Jonathan', '200');
insert into Users (user_id, user_name, credit) values ('3', 'Winston', '10000');
insert into Users (user_id, user_name, credit) values ('4', 'Luis', '800');

insert into Transactions (trans_id, paid_by, paid_to, amount, transacted_on) values ('1', '1', '3', '400', '2020-08-01');
insert into Transactions (trans_id, paid_by, paid_to, amount, transacted_on) values ('2', '3', '2', '500', '2020-08-02');
insert into Transactions (trans_id, paid_by, paid_to, amount, transacted_on) values ('3', '2', '1', '200', '2020-08-03');

Table: Users
+--------------+---------+
| Column Name | Type |
+--------------+---------+
user_id	int
user_name	varchar
credit	int
+--------------+---------+
user_id is the primary key for this table.
Each row of this table contains the current credit information for each user.

Table: Transactions
+---------------+---------+
| Column Name | Type |
+---------------+---------+
trans_id	int
paid_by	int
paid_to	int
amount	int
transacted_on	date
+---------------+---------+
trans_id is the primary key for this table.
Each row of this table contains information about the transaction in the bank.
User with id (paid_by) transfer money to user with id (paid_to).

Leetcode Bank (LCB) helps its coders in making virtual payments. Our bank records all transactions in the table *Transaction*, we want to find out the current balance of all users and check whether they have breached their credit limit (If their current credit is less than 0).
Write an SQL query to report.

- user_id,
- user_name,
- credit, current balance after performing transactions, and
- credit_limit_breached, check credit_limit ("Yes" or "No")

Return the result table in **any** order.
The query result format is in the following example.

Example 1:
Input:
Users table:

user_id	user_name	credit
1	Moustafa	100
2	Jonathan	200
3	Winston	10000
4	Luis	800

Transactions table:

trans_id	paid_by	paid_to	amount	transacted_on
1	1	3	400	2020-08-01
2	3	2	500	2020-08-02
3	2	1	200	2020-08-03

Output:

user_id	user_name	credit	credit_limit_breached
1	Moustafa	-100	Yes
2	Jonathan	500	No
3	Winston	9900	No
4	Luis	800	No

Explanation:
Moustafa paid $400 on "2020-08-01" and received $200 on "2020-08-03", credit (100 -400 +200) = -$100
Jonathan received $500 on "2020-08-02" and paid $200 on "2020-08-08", credit (200 +500 -200) = $500
Winston received $400 on "2020-08-01" and paid $500 on "2020-08-03", credit (10000 +400 -500) = $9990
Luis did not received any transfer, credit = $800

Qn 9 :

Schema to be used:

```
Create table If Not Exists Calls (from_id int, to_id int, duration int);
insert into Calls (from_id, to_id, duration) values ('1', '2', '59');
insert into Calls (from_id, to_id, duration) values ('2', '1', '11');
insert into Calls (from_id, to_id, duration) values ('1', '3', '20');
insert into Calls (from_id, to_id, duration) values ('3', '4', '100');
insert into Calls (from_id, to_id, duration) values ('3', '4', '200');
insert into Calls (from_id, to_id, duration) values ('3', '4', '200');
insert into Calls (from_id, to_id, duration) values ('4', '3', '499');
```

able: Calls

```
+-------------+---------+
| Column Name | Type    |
+-------------+---------+
| from_id     | int     |
| to_id       | int     |
| duration    | int     |
+-------------+---------+
```

This table does not have a primary key, it may contain duplicates.
This table contains the duration of a phone call between from_id and to_id.
from_id != to_id

Write an SQL query to report the number of calls and the total call duration between each pair of distinct persons (person1, person2) where person1 < person2.
Return the result table in **any order**.
The query result format is in the following example.

Example 1:
Input:
Calls table:

```
+---------+-------+----------+
| from_id | to_id | duration |
+---------+-------+----------+
| 1       | 2     | 59       |
| 2       | 1     | 11       |
| 1       | 3     | 20       |
| 3       | 4     | 100      |
| 3       | 4     | 200      |
| 3       | 4     | 200      |
| 4       | 3     | 499      |
+---------+-------+----------+
```

Output:

```
+---------+---------+------------+----------------+
| person1 | person2 | call_count | total_duration |
+---------+---------+------------+----------------+
| 1       | 2       | 2          | 70             |
```

| 1 | 3 | 1 | 20 |
| 3 | 4 | 4 | 999 |
+---------+---------+-----------+---------------+

Explanation:

Users 1 and 2 had 2 calls and the total duration is 70 (59 + 11).
Users 1 and 3 had 1 call and the total duration is 20.
Users 3 and 4 had 4 calls and the total duration is 999 (100 + 200 + 200 + 499).