# SQL Assignment: 1

**Note : For all below questions, use below command to create a new DB and then create sample tables as per questions inside it. Don’t use any existing DB.**

**create database assignment1;**

**use assignment1;**

**Qn 1:**

Table: Person

```
+-------------+---------+
| Column Name | Type    |
+-------------+---------+
| personId    | int     |
| lastName    | varchar |
| firstName   | varchar |
+-------------+---------+
```

personId is the primary key column for this table.
This table contains information about the ID of some persons and their first and last names.

Table: Address

```
+-------------+---------+
| Column Name | Type    |
+-------------+---------+
| addressId   | int     |
| personId    | int     |
| city        | varchar |
| state       | varchar |
+-------------+---------+
```

addressId is the primary key column for this table.
Each row of this table contains information about the city and state of one person with ID = PersonId.

Write an SQL query to report the first name, last name, city, and state of each person in the Person table. If the address of a personId is not present in the Address table, report null instead.

Return the result table in **any order**.

The query result format is in the following example.

**Example 1:**

**Input:**
Person table:

```
+----------+----------+-----------+
| personId | lastName | firstName |
+----------+----------+-----------+
| 1        | Wang     | Allen     |
| 2        | Alice    | Bob       |
+----------+----------+-----------+
```

Address table:

```
+-----------+----------+---------------+------------+
| addressId | personId | city          | state      |
+-----------+----------+---------------+------------+
| 1         | 2        | New York City | New York   |
| 2         | 3        | Leetcode      | California |
+-----------+----------+---------------+------------+
```

**Output:**

```
+-----------+----------+---------------+----------+
| firstName | lastName | city          | state    |
+-----------+----------+---------------+----------+
| Allen     | Wang     | Null          | Null     |
| Bob       | Alice    | New York City | New York |
+-----------+----------+---------------+----------+
```

**Explanation:**
There is no address in the address table for the personId = 1 so we return null in their city and state.
addressId = 1 contains information about the address of personId = 2.

**Schema to be used:**

```
Create table If Not Exists Person (personId int, firstName varchar(255), lastName
varchar(255));
Create table If Not Exists Address (addressId int, personId int, city varchar(255),
state varchar(255));
insert into Person (personId, lastName, firstName) values ('1', 'Wang', 'Allen');
insert into Person (personId, lastName, firstName) values ('2', 'Alice', 'Bob');
insert into Address (addressId, personId, city, state) values ('1', '2', 'New York
City', 'New York');
insert into Address (addressId, personId, city, state) values ('2', '3', 'Leetcode',
'California');
```

---

Qn 2:

**Schema to be used:**

```
Create table If Not Exists Employee (id int, name varchar(255), salary int,
managerId int)
Truncate table Employee
insert into Employee (id, name, salary, managerId) values ('1', 'Joe', '70000', '3')
insert into Employee (id, name, salary, managerId) values ('2', 'Henry', '80000', '4')
insert into Employee (id, name, salary, managerId) values ('3', 'Sam', '60000',
'None')
insert into Employee (id, name, salary, managerId) values ('4', 'Max', '90000',
'None')
```

Table: Employee

```
+-------------+---------+
| Column Name | Type    |
+-------------+---------+
| id          | int     |
| name        | varchar |
| salary      | int     |
| managerId   | int     |
+-------------+---------+
```
id is the primary key column for this table.
Each row of this table indicates the ID of an employee, their name, salary, and the ID of their manager.

Write an SQL query to find the employees who earn more than their managers.

Return the result table in **any order**.

The query result format is in the following example.

**Example 1:**

**Input:**
Employee table:
```
+----+-------+--------+-----------+
| id | name  | salary | managerId |
+----+-------+--------+-----------+
| 1  | Joe   | 70000  | 3         |
| 2  | Henry | 80000  | 4         |
| 3  | Sam   | 60000  | Null      |
| 4  | Max   | 90000  | Null      |
+----+-------+--------+-----------+
```
**Output:**
```
+----------+
| Employee |
+----------+
| Joe      |
+----------+
```
**Explanation:** Joe is the only employee who earns more than his manager

Qn 3:

Schema to be used:

```
Create table If Not Exists Cinema (seat_id int primary key auto_increment, free
bool);

insert into Cinema (seat_id, free) values ('1', '1');
insert into Cinema (seat_id, free) values ('2', '0');
insert into Cinema (seat_id, free) values ('3', '1');
insert into Cinema (seat_id, free) values ('4', '1');
insert into Cinema (seat_id, free) values ('5', '1');
```

Table: Cinema

```
+-------------+------+
| Column Name | Type |
+-------------+------+
| seat_id     | int  |
| free        | bool |
+-------------+------+
```

seat_id is an auto-increment primary key column for this table.
Each row of this table indicates whether the ith seat is free or not. 1 means free while 0 means occupied.

Write an SQL query to report all the consecutive available seats in the cinema.

Return the result table **ordered** by seat_id **in ascending order**.

The query result format is in the following example.

**Example 1:**

**Input:**
Cinema table:

```
+---------+------+
| seat_id | free |
+---------+------+
| 1       | 1    |
| 2       | 0    |
| 3       | 1    |
| 4       | 1    |
| 5       | 1    |
+---------+------+
```
**Output:**
```
+---------+
| seat_id |
+---------+
| 3       |
| 4       |
| 5       |
+---------+
```

Qn 4 :

Schema to be used:

```
Create table If Not Exists Department (id int, revenue int, month varchar(5));
insert into Department (id, revenue, month) values ('1', '8000', 'Jan');
insert into Department (id, revenue, month) values ('2', '9000', 'Jan');
insert into Department (id, revenue, month) values ('3', '10000', 'Feb')
insert into Department (id, revenue, month) values ('1', '7000', 'Feb');
insert into Department (id, revenue, month) values ('1', '6000', 'Mar');
```

Table: Department

```
+-------------+---------+
| Column Name | Type    |
+-------------+---------+
| id          | int     |
| revenue     | int     |
| month       | varchar |
+-------------+---------+
```
(id, month) is the primary key of this table.
The table has information about the revenue of each department per month.
The month has values in
["Jan","Feb","Mar","Apr","May","Jun","Jul","Aug","Sep","Oct","Nov","Dec"].

Write an SQL query to reformat the table such that there is a department id column and a revenue column **for each month**.

Return the result table in **any order**.

The query result format is in the following example.

**Example 1:**

**Input:**
Department table:
```
+------+---------+-------+
| id   | revenue | month |
+------+---------+-------+
| 1    | 8000    | Jan   |
| 2    | 9000    | Jan   |
| 3    | 10000   | Feb   |
| 1    | 7000    | Feb   |
| 1    | 6000    | Mar   |
+------+---------+-------+
```
**Output:**
```
+------+-------------+-------------+-------------+-----+-------------+
| id   | Jan_Revenue | Feb_Revenue | Mar_Revenue | ... | Dec_Revenue |
+------+-------------+-------------+-------------+-----+-------------+
| 1    | 8000        | 7000        | 6000        | ... | null        |
| 2    | 9000        | null        | null        | ... | null        |
| 3    | null        | 10000       | null        | ... | null        |
+------+-------------+-------------+-------------+-----+-------------+
```
**Explanation:** The revenue from Apr to Dec is null.
Note that the result table has 13 columns (1 for the department id + 12 for the months).

Qn 5 :

Schema to be used:

```
Create table If Not Exists Activity (player_id int, device_id int, event_date date, games_played int);
insert into Activity (player_id, device_id, event_date, games_played) values ('1', '2', '2016-03-01', '5');
insert into Activity (player_id, device_id, event_date, games_played) values ('1', '2', '2016-05-02', '6');
insert into Activity (player_id, device_id, event_date, games_played) values ('2', '3', '2017-06-25', '1');
insert into Activity (player_id, device_id, event_date, games_played) values ('3', '1', '2016-03-02', '0');
insert into Activity (player_id, device_id, event_date, games_played) values ('3', '4', '2018-07-03', '5');
```

Table: Activity

```
+--------------+---------+
| Column Name  | Type    |
+--------------+---------+
| player_id    | int     |
| device_id    | int     |
| event_date   | date    |
| games_played | int     |
+--------------+---------+
```

(player_id, event_date) is the primary key of this table.
This table shows the activity of players of some games.
Each row is a record of a player who logged in and played a number of games (possibly 0) before logging out on someday using some device.

Write an SQL query to report the **first login date** for each player.

Return the result table in **any order**.

The query result format is in the following example.

**Example 1:**

**Input:**
Activity table:
```
+-----------+-----------+------------+--------------+
| player_id | device_id | event_date | games_played |
+-----------+-----------+------------+--------------+
| 1         | 2         | 2016-03-01 | 5            |
| 1         | 2         | 2016-05-02 | 6            |
| 2         | 3         | 2017-06-25 | 1            |
| 3         | 1         | 2016-03-02 | 0            |
| 3         | 4         | 2018-07-03 | 5            |
+-----------+-----------+------------+--------------+
```
**Output:**
```
+-----------+-------------+
| player_id | first_login |
+-----------+-------------+
| 1         | 2016-03-01  |
| 2         | 2017-06-25  |
| 3         | 2016-03-02  |
+-----------+-------------+
```

Qn 6:

**Schema to be used:**

```
Create table If Not Exists Employee (empId int, name varchar(255), supervisor int,
salary int);
Create table If Not Exists Bonus (empId int, bonus int);
insert into Employee (empId, name, supervisor, salary) values ('3', 'Brad', 'None',
'4000');
insert into Employee (empId, name, supervisor, salary) values ('1', 'John', '3',
'1000');
insert into Employee (empId, name, supervisor, salary) values ('2', 'Dan', '3',
'2000');
insert into Employee (empId, name, supervisor, salary) values ('4', 'Thomas', '3',
'4000');
insert into Bonus (empId, bonus) values ('2', '500');
insert into Bonus (empId, bonus) values ('4', '2000');
```

Table: Employee

```
+-------------+---------+
| Column Name | Type    |
+-------------+---------+
| empId       | int     |
| name        | varchar |
| supervisor  | int     |
| salary      | int     |
+-------------+---------+
```

empId is the primary key column for this table.
Each row of this table indicates the name and the ID of an employee in addition to their salary and the id of their manager.

Table: Bonus

```
+-------------+------+
| Column Name | Type |
+-------------+------+
| empId       | int  |
| bonus       | int  |
+-------------+------+
```

empId is the primary key column for this table.
empId is a foreign key to empId from the Employee table.
Each row of this table contains the id of an employee and their respective bonus.

Write an SQL query to report the name and bonus amount of each employee with a bonus **less than** 1000.

Return the result table in **any order**.

The query result format is in the following example.

**Example 1:**

**Input:**

Employee table:

```
+-------+--------+------------+--------+
| empId | name   | supervisor | salary |
+-------+--------+------------+--------+
| 3     | Brad   | null       | 4000   |
| 1     | John   | 3          | 1000   |
| 2     | Dan    | 3          | 2000   |
| 4     | Thomas | 3          | 4000   |
+-------+--------+------------+--------+
```

Bonus table:

```
+-------+-------+
| empId | bonus |
+-------+-------+
| 2     | 500   |
| 4     | 2000  |
+-------+-------+
```

**Output:**

```
+------+-------+
| name | bonus |
+------+-------+
| Brad | null  |
| John | null  |

| Dan  | 500   |

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

Qn 7 :

Schema to be used:

```
Create table If Not Exists cinema (id int, movie varchar(255), description varchar(255), rating float(2, 1));
insert into cinema (id, movie, description, rating) values ('1', 'War', 'great 3D', '8.9');
insert into cinema (id, movie, description, rating) values ('2', 'Science', 'fiction', '8.5');
insert into cinema (id, movie, description, rating) values ('3', 'irish', 'boring', '6.2');
insert into cinema (id, movie, description, rating) values ('4', 'Ice song', 'Fantacy', '8.6');
insert into cinema (id, movie, description, rating) values ('5', 'House card', 'Interesting', '9.1')
```

Table: Cinema

```
+----------------+----------+
| Column Name    | Type     |
+----------------+----------+
| id             | int      |
| movie          | varchar  |
| description    | varchar  |
| rating         | float    |
+----------------+----------+
```

id is the primary key for this table.
Each row contains information about the name of a movie, its genre, and its rating.
rating is a 2 decimal places float in the range [0, 10]

Write an SQL query to report the movies with an odd-numbered ID and a description that is not "boring".

Return the result table ordered by rating **in descending order**.

The query result format is in the following example.

**Example 1:**

**Input:**
Cinema table:
```
+----+------------+-------------+--------+
| id | movie      | description | rating |
+----+------------+-------------+--------+
| 1  | War        | great 3D    | 8.9    |
| 2  | Science    | fiction     | 8.5    |
| 3  | irish      | boring      | 6.2    |
| 4  | Ice song   | Fantacy     | 8.6    |
| 5  | House card | Interesting | 9.1    |
+----+------------+-------------+--------+
```
**Output:**
```
+----+------------+-------------+--------+
| id | movie      | description | rating |
+----+------------+-------------+--------+
| 5  | House card | Interesting | 9.1    |
| 1  | War        | great 3D    | 8.9    |
+----+------------+-------------+--------+
```
**Explanation:**
We have three movies with odd-numbered IDs: 1, 3, and 5. The movie with ID = 3 is boring so we do not include it in the answer.

Qn 8:

```
Create table If Not Exists Customer (id int, name varchar(25), referee_id int);
insert into Customer (id, name, referee_id) values ('1', 'Will', 'None');
insert into Customer (id, name, referee_id) values ('2', 'Jane', 'None');
insert into Customer (id, name, referee_id) values ('3', 'Alex', '2');
insert into Customer (id, name, referee_id) values ('4', 'Bill', 'None');
insert into Customer (id, name, referee_id) values ('5', 'Zack', '1');
insert into Customer (id, name, referee_id) values ('6', 'Mark', '2');
```

Table: Customer

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

id is the primary key column for this table.
Each row of this table indicates the id of a customer, their name, and the id of the customer who referred them.

Write an SQL query to report the names of the customer that are **not referred by** the customer with id = 2.

Return the result table in **any order**.

The query result format is in the following example.

**Example 1:**

**Input:**
Customer table:

```
+----+------+------------+
| id | name | referee_id |
+----+------+------------+
| 1  | Will | null       |
| 2  | Jane | null       |
| 3  | Alex | 2          |
| 4  | Bill | null       |
| 5  | Zack | 1          |
| 6  | Mark | 2          |
+----+------+------------+
```
**Output:**
```
+------+
| name |
+------+
| Will |
| Jane |
| Bill |
| Zack |
+------+
```

Qn 9:

Schema to be used:

```
Create table If Not Exists Courses (student varchar(255), class varchar(255));
insert into Courses (student, class) values ('A', 'Math');
insert into Courses (student, class) values ('B', 'English');
insert into Courses (student, class) values ('C', 'Math');
insert into Courses (student, class) values ('D', 'Biology');
insert into Courses (student, class) values ('E', 'Math');
insert into Courses (student, class) values ('F', 'Computer');
insert into Courses (student, class) values ('G', 'Math');
insert into Courses (student, class) values ('H', 'Math');
insert into Courses (student, class) values ('I', 'Math');
```

Table: Courses

```
+-------------+---------+
| Column Name | Type    |
+-------------+---------+
| student     | varchar |
| class       | varchar |
+-------------+---------+
```
(student, class) is the primary key column for this table.
Each row of this table indicates the name of a student and the class in which they are enrolled.

Write an SQL query to report all the classes that have **at least five students**.

Return the result table in **any order**.

The query result format is in the following example.

**Example 1:**

**Input:**
Courses table:
```
+---------+----------+
| student | class    |
+---------+----------+
| A       | Math     |
| B       | English  |
| C       | Math     |
| D       | Biology  |
| E       | Math     |
| F       | Computer |
| G       | Math     |
| H       | Math     |
| I       | Math     |
+---------+----------+
```
**Output:**
```
+---------+
| class   |
+---------+
| Math    |
+---------+
```
**Explanation:**
- Math has 6 students, so we include it.
- English has 1 student, so we do not include it.
- Biology has 1 student, so we do not include it.
- Computer has 1 student, so we do not include it.

Qn 10 :

Schema to be used:

```
Create table If Not Exists Employee (id int, month int, salary int);
insert into Employee (id, month, salary) values ('1', '1', '20');
insert into Employee (id, month, salary) values ('2', '1', '20');
insert into Employee (id, month, salary) values ('1', '2', '30');
insert into Employee (id, month, salary) values ('2', '2', '30');
insert into Employee (id, month, salary) values ('3', '2', '40');
insert into Employee (id, month, salary) values ('1', '3', '40');
insert into Employee (id, month, salary) values ('3', '3', '60');
insert into Employee (id, month, salary) values ('1', '4', '60');
insert into Employee (id, month, salary) values ('3', '4', '70');
insert into Employee (id, month, salary) values ('1', '7', '90');
insert into Employee (id, month, salary) values ('1', '8', '90');
```

Table: Employee

```
+-------------+------+
| Column Name | Type |
+-------------+------+
| id          | int  |
| month       | int  |
| salary      | int  |
+-------------+------+
```

(id, month) is the primary key for this table.
Each row in the table indicates the salary of an employee in one month during the year 2020.

Write an SQL query to calculate the **cumulative salary summary** for every employee in a single unified table.

The **cumulative salary summary** for an employee can be calculated as follows:

- For each month that the employee worked, **sum** up the salaries in **that month** and the **previous two months**. This is their **3-month sum** for that month. If an employee did not work for the company in previous months, their effective salary for those months is 0.

- Do **not** include the 3-month sum for the **most recent month** that the employee worked for in the summary.
- Do **not** include the 3-month sum for any month the employee **did not work**.

Return the result table ordered by id in **ascending order**. In case of a tie, order it by month in **descending order**.

The query result format is in the following example.

**Example 1:**

**Input:**
Employee table:
```
+----+-------+--------+
| id | month | salary |
+----+-------+--------+
| 1  | 1     | 20     |
| 2  | 1     | 20     |
| 1  | 2     | 30     |
| 2  | 2     | 30     |
| 3  | 2     | 40     |
| 1  | 3     | 40     |
| 3  | 3     | 60     |
| 1  | 4     | 60     |
| 3  | 4     | 70     |
| 1  | 7     | 90     |
| 1  | 8     | 90     |
+----+-------+--------+
```
**Output:**
```
+----+-------+--------+
| id | month | Salary |
+----+-------+--------+
| 1  | 7     | 90     |
| 1  | 4     | 130    |
| 1  | 3     | 90     |
| 1  | 2     | 50     |
| 1  | 1     | 20     |
| 2  | 1     | 20     |
| 3  | 3     | 100    |
| 3  | 2     | 40     |
+----+-------+--------+
```
**Explanation:**
Employee '1' has five salary records excluding their most recent month '8':
- 90 for month '7'.
- 60 for month '4'.
- 40 for month '3'.
- 30 for month '2'.
- 20 for month '1'.

So the cumulative salary summary for this employee is:
```
+----+-------+--------+
| id | month | salary |
+----+-------+--------+
| 1  | 7     | 90     |  (90 + 0 + 0)
| 1  | 4     | 130    |  (60 + 40 + 30)
| 1  | 3     | 90     |  (40 + 30 + 20)
| 1  | 2     | 50     |  (30 + 20 + 0)
| 1  | 1     | 20     |  (20 + 0 + 0)
+----+-------+--------+
```
Note that the 3-month sum for month '7' is 90 because they did not work during month '6' or month '5'.

Employee '2' only has one salary record (month '1') excluding their most recent month '2'.
```
+----+-------+--------+
| id | month | salary |
+----+-------+--------+
| 2  | 1     | 20     |  (20 + 0 + 0)
+----+-------+--------+
```

Employee '3' has two salary records excluding their most recent month '4':
- 60 for month '3'.
- 40 for month '2'.

So the cumulative salary summary for this employee is:
```
+----+-------+--------+
| id | month | salary |
+----+-------+--------+
| 3  | 3     | 100    |  (60 + 40 + 0)
| 3  | 2     | 40     |  (40 + 0 + 0)
```

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