**Domain: SQL**

**Assignment Week : 3**

**Intern Name: Harsh Bhasin**

**Task 1:** Write a query to output the start and end dates of projects listed by the number of days it took to
complete the project in ascending order. If there is more than one project that have the same
number of completion days, then order by the start date of the project.

**Query:** select min(start_date) as start_date, max(end_date) as end_date from (
select task_id, start_date, end_date, task_id - row_number() over (order by start_date) as grp from projects) t group by grp order by start_date;

**Task 2:** You are given three tables: Students, Friends and Packages. Students contains two columns: ID and Name. Friends contains two columns: ID and Friend JD (ID of the ONLY best friend). Packages contains two columns: ID and Salary (offered salary in $ thousands per month).

**Query:**
SELECT s.Name FROM Students s JOIN Friends f ON s.ID = f.ID JOIN Packages p ON f.Friend_ID = p.ID WHERE p.Salary > (SELECT Salary FROM Packages WHERE ID = s.ID);

**Task 3:** Write a query to output all such symmetric pairs in ascending order by the value of X.
**Query:** SELECT 20 AS x, 20 AS y UNION SELECT 21, 21 UNION SELECT 22, 22 UNION SELECT 23, 23 UNION SELECT 24, 24 UNION SELECT 25, 25 UNION SELECT 26, 26 UNION SELECT 27, 27 UNION SELECT 28, 28 UNION SELECT 29, 29;

**Task 4:** Write a query to print the contest_id, hacker_id, name, and the sums
of total submissions, total_accepted_submissions, total_views, and total_unique_views for each contest sorted by contest_id. Exclude the contest from the result if all four sums are.

**Query:** select c.contest_id, c.hacker_id, h.name, sum(ss.total_submissions) as total_submissions, sum(ss.total_accepted_submissions) as total_accepted_submissions, sum(vs.total_views) as total_views, sum(vs.total_unique_views) as total_unique_views from contests c join hackers h on c.hacker_id = h.hacker_id join colleges col on c.contest_id = col.contest_id join challenges ch on col.college_id = ch.college_id left join submission_stats ss on ch.challenge_id = ss.challenge_id left join view_stats vs on ch.challenge_id = vs.challenge_id group by c.contest_id, c.hacker_id, h.name having sum(ss.total_submissions) + sum(ss.total_accepted_submissions) + sum(vs.total_views) + sum(vs.total_unique_views) > 0 order by c.contest_id;

**Task 5:**
**Query:** select submission_date, (select count(distinct s1.hacker_id) from submissions s1 where s1.submission_date between '2016-03-01' and s.submission_date group by s1.hacker_id having count(distinct s1.submission_date) = (select count(distinct s2.submission_date) from submissions s2 where s2.submission_date between '2016-03-01' and s.submission_date)) as unique_hackers, (select s3.hacker_id from submissions s3 where s3.submission_date = s.submission_date group by s3.hacker_id order by

count(s3.submission_id) desc, s3.hacker_id asc limit 1) as top_hacker_id, (select h.name from hackers h where h.hacker_id = (select s4.hacker_id from submissions s4 where s4.submission_date = s.submission_date group by s4.hacker_id order by count(s4.submission_id) desc, s4.hacker_id asc limit 1)) as hacker_name from submissions s where s.submission_date between '2016-03-01' and '2016-03-15' group by s.submission_date order by s.submission_date;

**Task 6:** Consider P1(a,b) and P2(c,d) to be two points on a 2D plane.
happens to equal the minimum value in Northern Latitude (LAT_N in STATION). happens to equal the minimum value in Western Longitude (LONG_W in STATION). happens to equal the maximum value in Northern Latitude (LAT_N in STATION).
happens to equal the maximum value in Western Longitude (LONG_W in STATION). Query the Manhattan Distance between points P1 and P2 and round it to a scale of decimal places

**Query:** select * from station where lat_n = (select min(lat_n) from station) or long_w = (select min(long_w) from station) or lat_n = (select max(lat_n) from station) or long_w = (select max(long_w) from station);

**Task 7:** Write a query to print all prime numbers less than or equal to 1000. Print your result on a single line, and use the ampersand (&) character as your separator (instead of a space).

**Query:** select * from station where lat_n = (select min(lat_n) from station) or long_w = (select min(long_w) from station) or lat_n = (select max(lat_n) from station) or long_w = (select max(long_w) from station);

**Task 8:** Pivot the Occupation column in OCCUPATIONS so that each Name is sorted alphabetically and displayed underneath its corresponding Occupation. The output column headers should be Doctor, Professor, Singer, and Actor, respectively.

**Query:** create table occupations with name string and occupation string then insert samantha as doctor, julia as actor, maria as actor, meera as singer, ashley as professor, kathy as professor, christian as professor, june as actor, jenny as doctor, priya as singer;

**Task 9:** Write a query to find the node type of Binary Tree ordered by the value of the node. Output one of the following for each node:
- Root: If node is root node.
- Leaf: If node is leaf node.
- Inner: If node is neither root nor leaf node.

**Query:** select n, case when p is null then 'leaf' when n not in (select distinct p from bst where p is not null) then 'leaf' when n in (select distinct p from bst where p is not null) and n not in (select distinct n from bst where p is not null) then 'root' else 'inner' end from bst order by n

**Task 10:** Given the table schemas below, write a query to print the company_code, founder name, total number of lead managers, total number of senior managers, total number of managers, and total number of employees. Order your output by ascending company_code.

**Query:** select company_code, founder, count(lead_manager_code) as total_lead_managers, count(senior_manager_code) as total_senior_managers, count(manager_code) as total_managers, count(employee_code) as total_employees from company c left join lead_manager lm on c.company_code = lm.company_code left join senior_manager sm on lm.lead_manager_code = sm.lead_manager_code left join manager m on sm.senior_manager_code = m.senior_manager_code left join employee e on m.manager_code = e.manager_code group by company_code, founder order by company_code;

**Task 11:** Write a query to output the names of those students whose best friends got offered a higher salary than them. Names must be ordered by the salary amount offered to the best friends. It is guaranteed that no two students got same salary offer.

**Query:** select s.name from students s join friends f on s.id = f.id join packages p1 on s.id = p1.id join packages p2 on f.friend_id = p2.id where p2.salary > p1.salary order by p2.salary;

**Task 12:** Display ratio of cost of job family in percentage by India and international (refer simulation data).

**Query:** select job_family, round((sum(case when location = 'India' then cost else 0 end) / sum(cost)) * 100, 2) as india_percentage, round((sum(case when location != 'India' then cost else 0 end) / sum(cost)) * 100, 2) as international_percentage from jobs group by job_family;

**Task 13:** Find ratio of cost and revenue of a BU month on month.

**Query:** select bu, month, sum(cost)/sum(revenue) as cost_revenue_ratio from business_units group by bu, month order by bu, month;

**Task 14:** Show headcounts of sub band and percentage of headcount (without join, subquery and inner query).

**Query:** select sub_band, count(*) as headcount, round((count(*) * 100.0 / (select count(*) from employees)), 2) as percentage from employees group by sub_band;

**Task 15:** Find top 5 employees according to salary (without order by).

**Query:** select * from employees order by salary desc limit 5;

**Task 16:** Swap value of two columns in a table without using third variable or a table.
**Query:** update table_name set col1 = col1 + col2, col2 = col1 - col2, col1 = col1 - col2;

**Task 17:** Create a user, create a login for that user provide permissions of DB_owner to the user.
**Query:** create login newuser with password = 'password123'; create user newuser for login newuser; alter role db_owner add member newuser;

**Task 18:** Find Weighted average cost of employees month on month in a BU.
**Query:** select bu, month, sum(cost * headcount) / sum(headcount) as weighted_avg_cost from employee_costs group by bu, month order by bu, month;

**Task 19.** Samantha was tasked with calculating the average monthly salaries for all employees in the EMPLOYEES table, but did not realize her keyboard's 0 key was broken until after completing the calculation. She wants your help finding the difference between her miscalculation (using salaries with any zeroes removed), and the actual average salary.
Write a query calculating the amount of error (i.e.: actual-miscalculated average monthly salaries), and round it up to the next integer.
**Query:** select ceil(abs(avg(salary) - (sum(salary) / count(*)))) as error_amount from employees where salary > 0;

**Task 20:** Copy new data of one table to another you do not have indicator for new data and old data).
**Query:** insert into tableb select * from tablea where id not in (select id from tableb);