ALL DAYS COMPILATION

1. Write a query to find **all** the details of the **movie** that has the **third-highest** revenue.

➔ Use offset when needed…

```
select * from movies
order by revenue desc
limit 1 offset 2;
```

OR

```
SELECT * from movies
order by revenue desc
limit 2,1
```

2. Display all the details of the employees who did **not work** at any job in the **past**.

➔ Referring the same table to establishing a condition…

```
SELECT *
FROM employees
where employee_id not in
    (
        SELECT employee_id
        from job_history
    )
```

3. Write a query to display the employee details who report to **Adam**

➔ Just like above…

```
select * from employees
where manager_id in
    (
        select employee_id
        from employees
        where first_name like '%Adam%'
    )
```

4. Based on the employee's salary, divide the employees into three different classes.

➔ Classic use of CASE…as you can see no commas or brackets just WHEN-THEN…END

```
select employee_id, salary,
case
when salary > 20000 then 'Class A'
when salary <=20000 and salary >=10000 then 'Class B'
when salary < 10000 then 'Class C'
end as Salary_bin from employees;
```

5. Display the details of the employees who joined the company before their managers joined the company.

➔ Self-join to establish a condition

```
SELECT DISTINCT e.employee_id, e.first_name, e.last_name
FROM employees e
INNER JOIN employees m
ON e.manager_id = m.employee_id
AND e.hire_date < m.hire_date;
```

6. Write a query to find the shortest distance between any two points from the **points** table. Round the distance to two decimal points

➔ Cross join so that every row is compared with every other & sort the distant using the classic distance formula - $\sqrt{((x2 - x1)^2 + (y2 - y1)^2)}$

X	Y
1	1
2	3
0	1

```
select round(sqrt(pow((p1.x - p2.x),2)+ pow((p1.y - p2.y),2)),2) as shortest
from points p1, points p2
where not (p1.x = p2.x and p1.y = p2.y)
order by shortest
limit 1;
```

7. Write a query to find all the details of those employees who earn the **third-highest** salary

➔ Use Offset with sub-query…

```
select * from employees
where salary = (
    select distinct salary from employees
    order by salary desc
    limit 2, 1
)
```

8. Display the employee's full name (first name and last name separated by space) as 'full name' of all those employees whose salary is greater than 40% of their department's total salary.

➔ Just like above…

```
SELECT concat(first_name,' ',last_name) as full_name
from employees as e
where salary > (
   SELECT sum(salary) * 0.4
   from employees as e1
   where e.department_id = e1.department_id
   group by department_id
)
```

9. Display the 'full name' (first and last name separated by space) of a manager who manages 4 or more employees.

➔ Get the count by correct aggregation…

```
select concat(first_name, ' ', last_name) as 'full_name'
from employees where employee_id in
(select manager_id from employees
group by manager_id having count(*) >= 4);
```

10. Given a table of candidates and their skills, you're asked to find the candidates best suited for an open Data Science job. We want to find candidates who are proficient in '**Python**', '**Tableau**', and '**MySQL**'.

    Write a query to list the candidates who possess all three required skills for the job. Sort the output by **candidate_id** in ascending order.

```
select candidate_id from
(select candidate_id, count(*) as skills from candidates
where skill in ('Python','Tableau','MySQL')
group by candidate_id) a
where skills = 3
order by candidate_id;
```

11. Write a query to find the **customer_id** and **customer_name** of customers who bought products "**Bread**", and "**Milk**" but did not buy the product "**Eggs**" since we want to recommend them to purchase this product.

```
select c.customer_id, c.customer_name
from orders o
join customers c
on c.customer_id = o.customer_id
group by c.customer_id, c.customer_name
having sum(o.product_name="Bread") > 0 and
sum(o.product_name="Milk") > 0 and
sum(o.product_name="Eggs") = 0
order by customer_id;
```

12. Display all the details of those departments where the salary of any employee in that department is **at least 9000**.

➔ Classic grouping by department…**having** is use to establish condition for aggregate functions

```
select  * from departments
where department_id in
(select department_id from employees
group by department_id
having min(salary) >= 9000);
```

13. Show the details of the employees who have the 5th highest salary in each **job category**.

```
select employee_id,first_name, job_id from (
select employee_id,first_name, job_id,
dense_rank() over(partition by job_id order by
salary desc) 'salary_rank' from employees)t where salary_rank = 5;
```

14. Write a Query to find the **first day of the first job** of every employee and return it as 'first_day_job'.

➔ First_value & last_value over a partition can help u in these cases…

```
select distinct first_name,
first_value(start_date) over(partition by jhist.employee_id
order by start_date) as 'first_day_job'
from job_history jhist
join employees emp
on jhist.employee_id=emp.employee_id
order by first_name;
```

15. Find the **quartile** of each record based on the **salary** of the employee save as 'Quartile'.

➔ Quartile is nothing but dividing salary into 4 buckets…NTILE () function is used here

```
select employee_id, first_name, department_id, job_id, salary,
ntile(4) over(order by salary) 'Quartile'
from employees;
```

16. Each row in the table contains the visit_date and visit_id to the mall with the number of people during the visit. No two rows will have the same visit_date
Write a query to display the records with **three or more** rows with consecutive id's, and the number of people is greater than or equal to 100 for each.

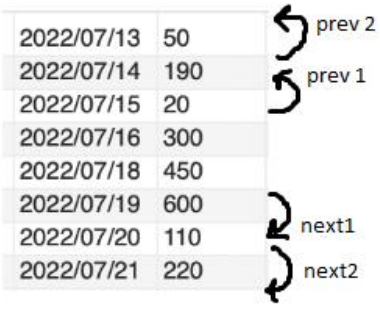
Sample Input:

Table: mall

id	visit_date	people
1	2022/07/13	50
2	2022/07/14	190
3	2022/07/15	20
4	2022/07/16	300
5	2022/07/18	450
6	2022/07/19	600
7	2022/07/20	110
8	2022/07/21	220

Sample Output:

id	visit_date	people
4	2022/07/16	300
5	2022/07/18	450
6	2022/07/19	600
7	2022/07/20	110
8	2022/07/21	220



```sql
select id, visit_date, people
from
(select id, visit_date, people,
lead(people) over (order by id asc) as next1,
lead(people,2) over (order by id asc) as next2,
lag(people)over (order by id asc) as prev1,
lag(people,2)over (order by id asc) as prev2
from mall
) as mall_ppl

where (people >= 100 and next1 >= 100 and next2 >= 100) or
(people >= 100 and prev1 >= 100 and prev2 >= 100)
order by visit_date;
```

17. The winning streak of a player is calculated as the number of consecutive wins uninterrupted by draws or losses. Write a query to count the **longest winning streak** for **each** player and save the new column as 'longest_streak'.

➔ Think harder on this question ***

Table: matches

player_id	match_day	result
1	2022/07/13	Win
1	2022/07/14	Win
1	2022/07/16	Win
1	2022/07/18	Draw
1	2022/07/20	Win
2	2022/07/18	Lose
2	2022/07/19	Lose
3	2022/07/18	Win
3	2022/07/21	Win
3	2022/07/22	Lose
3	2022/07/23	Lose

Sample Output:

player_id	longest_streak
1	3
2	0
3	2

```sql
select m1.player_id, ifnull(max(cnt), 0) as longest_streak
from matches m1
left join(
select player_id, (p_rnk - rnk) as diff, count(p_rnk - rnk) as cnt
from(
select *,
rank() over(partition by player_id, result order by match_day) as rnk,
rank() over(partition by player_id order by match_day) as p_rnk
from matches m2) t1
where result = 'Win'
group by player_id, diff)t2
on m1.player_id = t2.player_id
group by m1.player_id
order by m1.player_id;
```

18. Write a query to calculate the total number of comments received for each user in the **30 or less days** before **2020-02-10** and save the column as 'comments_count'

```
select user_id, sum(number_of_comments) as 'comments_count'
from fb_comments
where created_at between DATE_SUB('2020-02-10', INTERVAL 30 DAY) and '2020-02-10'
group by user_id
order by user_id;
```

19. Display the details of the employees who had worked **less than a year**.

```
select jh.employee_id, concat(first_name,' ',last_name) 'full_name', job_title
from employees emp
join job_history  jh  on  jh.employee_id = emp.employee_id
join jobs job on  jh.job_id = job.job_id
where (datediff(end_date,start_date) /365) < 1
order by employee_id, job_title;
```

20. Display the **year** from the hire_date as 'Year' and count the number of employees who joined in that year and save it as 'Employees_count'

➔ Extracting **year**

```
select year(hire_date) 'Year',
count(employee_id) 'Employees_count'
from employees
group by Year
order by Employees_count desc;
```

21. Calculate the net salary for the employees and save the column as '**Net salary**' and display the details of those employees whose net salary is greater than **15000**(Use the CTE method)

➔ Use of IFNULL & making CTE (**common table expression -** is a temporary named result set created from a simple SELECT statement that can be used in a subsequent SELECT statement)

```
With t as
(select employee_id,first_name, last_name, salary,
salary+(salary * ifnull(commission_pct,0)) 'Net_Salary'
from employees)

select * from t
where Net_Salary > 15000;
```

VIEW	TABLE
A database object that allows generating a logical subset of data from one or more tables	A database object or an entity that stores the data of a database
A virtual table	An actual table
View depends on the table	Table is an independent data object

View	Description
Simple View	A view based on the only a single table, which doesn't contain GROUP BY clause and any functions.
Complex View	A view based on multiple tables, which contain GROUP BY clause and functions
Inline view	A view based on a subquery in FROM Clause, that subquery creates a temporary table and simplifies the complex query.
Materialized view	A view that stores the definition as well as data. It creates replicas of data by storing it physically.

22. Write a query to reorder the entries of the **genders** table so that "**female**," "**other**," and "**male**" appear in that order in alternating rows. The table should be rearranged such that the IDs of each gender are sorted in ascending order.

user_id	gender
1	male
2	other
3	other
4	male
5	female
6	male
7	female

⇨

user_id	gender
5	female
2	other
1	male
7	female
3	other
4	male

```
select x.user_id, x.gender from
(select user_id, gender,
row_number() over (partition by gender order by user_id) as ranking,
CASE when gender = 'female' then 1
when gender = 'other' then 2
when gender = 'male' then 3 END as encode
from genders) as x
order by x.ranking, x.encode
```

23. **Transposing** the Table with substituting 0 with **NULL**

Table: Products

product_id	store1	store2	store3
0	90	105	110
1	NULL	87	85
2	NULL	30	40

Sample Output:

product_id	store	price
0	store1	90
0	store2	105
0	store3	110
1	store2	87
1	store3	85
2	store2	30
2	store3	40

```
select * from (select product_id,
if(store1 is not null,'store1',null) as store,
if(store1 is not null, store1,null) as price
from Products
union
select product_id,
if(store2 is not null,'store2',null) as store,
if(store2 is not null, store2,null) as price
from Products
union
select product_id,
if(store3 is not null,'store3',null) as store,
if(store3 is not null, store3,null) as price
from Products) as x where store is not null
```

24. Write a query to calculate the total sales amount of **each item for each year**, with the corresponding product_id, product_name, and report_year.

➔ Problem you would encounter would be to **segregate data** for each year e.g. for product 2 period start is 2019 & end is 2022 so data would be for 2019, 2020, 2021, 2022. To get this data say for 2019 we need to select that specific year data and **union** it with other years. To get this data we'll get **DATEDIFF** between least (2019-12-31, period end) and greatest (2019-01-01, period start)

Table: products

product_id	product_name
1	Phone
2	T-shirt
3	Shoes

Table: sales

product_id	period_start	period_end	average_daily_sales
1	2020/01/25	2020/12/26	100
2	2019/03/21	2021/08/22	10
3	2021/02/13	2022/09/29	1000

Sample Output:

product_id	product_name	report_year	total_amount
1	Phone	2020	34200
2	T-shirt	2019	2860
2	T-shirt	2020	3660
2	T-shirt	2021	3650
3	Shoes	2021	322000
3	Shoes	2022	365000

```
with cte as
(select * from
(select product_id, '2018' as report_year,
average_daily_sales * (datediff(least('2018-12-31', period_end), greatest('2018-01-01',
period_start))+1) as total_amount from sales
union all
select product_id, '2019' as report_year,
average_daily_sales*(datediff(least('2019-12-31', period_end), greatest('2019-01-01',
period_start))+1) as total_amount from sales
union all
select product_id, '2020' as report_year,
average_daily_sales*(datediff(least('2020-12-31', period_end), greatest('2020-01-01',
period_start))+1) as total_amount from sales
union all
select product_id, '2021' as report_year,
average_daily_sales*(datediff(least('2021-12-31', period_end), greatest('2021-01-01',
period_start))+1) as total_amount from sales
union all
select product_id, '2022' as report_year,
average_daily_sales*(datediff(least('2022-12-31', period_end), greatest('2022-01-01',
period_start))+1) as total_amount from sales) p
where total_amount>0)
select a.product_id, b.product_name, a.report_year, a.total_amount
from cte a left join products b on
a.product_id = b.product_id
order by a.product_id, a.report_year;
```