

SQL Assessment

Qn.1) What will be the result of the query below?

```
SELECT * FROM runners WHERE id NOT IN (SELECT winner_id FROM races)
```

Explain your answer and also provide an alternative version of this query that will avoid the issue that it exposes.

```
sql> SELECT * FROM runners;
```

id	name
1	John Doe
2	Jane Doe
3	Alice Jones
4	Bobby Louis
5	Lisa Romero

```
sql> SELECT * FROM races;
```

id	event	winner_id
1	100 meter dash	2
2	500 meter dash	3
3	cross-country	2
4	triathlon	NULL

Ans) The issue will arise with the query `SELECT * FROM runners WHERE id NOT IN (SELECT winner_id FROM races)` due to the presence of NULL values in the `winner_id` column of the `races` table. When comparing with NULL values, the result is UNKNOWN, and rows with NULL values in `winner_id` will not be excluded from the result set.

Alternative Query:

To address this issue and ensure accurate results, you can rewrite the query using a LEFT JOIN and explicitly checking for NULL values. Here's an alternative version of the query:

```
sql
```

```
SELECT r.*
```

```
FROM runners r
```

```
LEFT JOIN races ra ON r.id = ra.winner_id
```

```
WHERE ra.winner_id IS NULL OR ra.winner_id = '';
```

Explanation of Alternative Query:

This query uses a LEFT JOIN between the `runners` and `races` tables on the `id` and `winner_id`.

By checking for both NULL values and empty strings specifically in the `winner_id` column from the `races` table, it ensures that all runners who have not won any races are included in the result set.

Qn.2)

```
create table test_a(id numeric);

create table test_b(id numeric);

insert into test_a(id) values
(10),
(20),
(30),
(40),
(50);

insert into test_b(id) values
(10),
(30),
(50);
```

Write a query to fetch values in table test_a that are and not in test_b without using the NOT keyword.

Ans) SELECT a.id
FROM test_a a
LEFT JOIN test_b b ON a.id = b.id
WHERE b.id IS NULL;

Qn.3)

```
SELECT * FROM users;
```

user_id	username
1	John Doe
2	Jane Don
3	Alice Jones
4	Lisa Romero

```
SELECT * FROM training_details;
```

user_training_id	user_id	training_id	training_date
1	1	1	"2015-08-02"
2	2	1	"2015-08-03"
3	3	2	"2015-08-02"
4	4	2	"2015-08-04"
5	2	2	"2015-08-03"
6	1	1	"2015-08-02"
7	3	2	"2015-08-04"
8	4	3	"2015-08-03"
9	1	4	"2015-08-03"
10	3	1	"2015-08-02"
11	4	2	"2015-08-04"
12	3	2	"2015-08-02"
13	1	1	"2015-08-02"
14	4	3	"2015-08-03"

Write a query to to get the list of users who took the a training lesson more than once in the same day, grouped by user and training lesson, each ordered from the most recent lesson date to oldest date.

Ans)

SELECT

```
u.user_id,
td.training_lesson,
td.lesson_date,
COUNT(*) AS lesson_count
```

FROM

```
users u
```

```
JOIN training_details td ON u.user_id = td.user_id
```

GROUP BY

```
u.user_id,
td.training_lesson,
td.lesson_date
```

HAVING

```
COUNT(*) > 1
```

ORDER BY

```
u.user_id,
td.training_lesson,
td.lesson_date DESC;
```

Qn.4)

Emp_Id	Emp_name	Salary	Manager_Id
10	Anil	50000	18
11	Vikas	75000	16
12	Nisha	40000	18
13	Nidhi	60000	17
14	Priya	80000	18
15	Mohit	45000	18
16	Rajesh	90000	–
17	Raman	55000	16
18	Santosh	65000	17

Write a query to generate below output:

Manager_Id	Manager	Average_Salary_Under_Manager
16	Rajesh	65000
17	Raman	62500
18	Santosh	53750

Ans)

```
SELECT E2.EMP_ID, E2.EMP_NAME, AVG(E1.SALARY)
FROM MANAGER_EMP E1
INNER JOIN MANAGER_EMP E2
ON E1.MANAGER_ID = E2.EMP_ID
GROUP BY E2.EMP_ID, E2.EMP_NAME
```