Usuário: User

2 / 2

Data: Tuesday, June 20, 2023 at 1:24:27 AM - (Asia/Calcutta)

URL: https://chat.openai.com/c/b9f2c13e3-cbc-45d3-a5a0-325424f9730f

player(player_id,name,game_account_balance,location, pincode)
matches(match_id,type_of_game,location)
transactions(trans_id,player_id,bet_amount)
city(pincode,name)

write a sql query for
find the player name who has lost maximum amoung in bets

To find the player who has lost the maximum amount in bets, you can use the following SQL query:

sql

```
SELECT p.name, SUM(t.bet_amount) AS total_loss
FROM player p
JOIN transactions t ON p.player_id = t.player_id
WHERE t.bet_amount < 0
GROUP BY p.player_id, p.name
ORDER BY total_loss ASC
LIMIT 1;
```

This query joins the "player" and "transactions" tables on the "player_id" column. It calculates the total loss by summing the "bet_amount" for each player. The "WHERE" clause filters out positive bet amounts, considering only losses. The results are grouped by player and ordered in ascending order of total loss. Finally, the "LIMIT 1" clause ensures that only the player with the maximum loss is returned.

Please note that this query assumes a negative "bet_amount" indicates a loss. Adjust the

"WHERE" clause if the opposite is true in your schema.

P 1:

write oracle query for above

Certainly! Here's the equivalent Oracle SQL query for finding the player who has lost the maximum amount in bets:

sql

```
SELECT p.name, SUM(t.bet_amount) AS total_loss
FROM player p
JOIN transactions t ON p.player_id = t.player_id
WHERE t.bet_amount < 0
GROUP BY p.player_id, p.name
ORDER BY total_loss ASC
FETCH FIRST 1 ROWS ONLY;
```

In Oracle, the "LIMIT" clause is replaced with "FETCH FIRST <n> ROWS ONLY" , where "<n>" represents the number of rows you want to retrieve. In this case, we only want to fetch the first row, which represents the player with the maximum loss.

Again, please verify that the condition "t.bet_amount < 0" correctly identifies loss amounts in your specific schema. Adjust the query if necessary.

P 2:

find city name with maximum avedrage bet amount

To find the city name with the maximum average bet amount, you can use the following Oracle SQL query:

sql

```
SELECT c.name, AVG(t.bet_amount) AS average_bet_amount
FROM city c
JOIN player p ON c.pincode = p.location_pincode
JOIN transactions t ON p.player_id = t.player_id
GROUP BY c.name
ORDER BY average_bet_amount DESC
FETCH FIRST 1 ROWS ONLY;
```

This query joins the "city" , "player" , and "transactions" tables based on the corresponding columns. It calculates the average bet amount using the "AVG" function. The results are grouped by city name and ordered in descending order of average bet amount. Finally, the "FETCH FIRST 1 ROWS ONLY" clause ensures that only the city with the maximum average bet amount is returned.

Please note that you need to adjust the table and column names if they differ in your schema.

P 3:

find the type of game having minimum number of bets

To find the type of game with the minimum number of bets, you can use the following Oracle SQL query:

sql

```
SELECT m.type_of_game, COUNT(t.trans_id) AS total_bets
FROM matches m
LEFT JOIN transactions t ON m.match_id = t.match_id
GROUP BY m.type_of_game
ORDER BY total_bets ASC
FETCH FIRST 1 ROWS ONLY;
```

This query joins the "matches" and "transactions" tables based on the "match_id" column. It counts the number of bets using the "COUNT" function. The results are grouped by the type of game and ordered in ascending order of total bets. Finally, the "FETCH FIRST 1 ROWS ONLY" clause ensures that only the type of game with the minimum number of bets is returned.

Please make sure to adjust the table and column names according to your schema if they differ.

P 4:

why did we use left join

We used a LEFT JOIN in the query because we want to include all types of games from the "matches" table, regardless of whether there are corresponding bets in the "transactions" table.

By using a LEFT JOIN, we ensure that all records from the left table ("matches") are included in the result set, even if there are no matching records in the right table ("transactions"). This allows us to count the number of bets for each type of game, including games with zero bets.

If we were to use an INNER JOIN instead, only the types of games that have at least one corresponding bet would be included in the result set. By using a LEFT JOIN, we can identify the type of game with the minimum number of bets, including those with zero bets.

P 5:

find city names from which no citizen bets done so far

To find the city names from which no citizen has placed any bets so far, you can use the following Oracle SQL query:

sql

```
SELECT c.name
FROM city c
LEFT JOIN player p ON c.pincode = p.location_pincode
LEFT JOIN transactions t ON p.player_id = t.player_id
WHERE t.trans_id IS NULL
```

```
GROUP BY c.name;
```

This query uses a combination of LEFT JOIN and IS NULL condition to identify the cities where no bets have been placed by citizens. The LEFT JOINs are used to include all cities and player records, regardless of whether there are matching transactions. The WHERE clause filters out the rows where "trans_id" is NULL, indicating no corresponding transaction exists. The result is grouped by city names, ensuring each city appears only once.

Please adjust the table and column names according to your schema if they differ.

P 6:

LEFT JOIN player p ON c.pincode = p.location_pincode why do we write this

We use the "LEFT JOIN" statement "player p ON c.pincode = p.location_pincode" in the query to connect the "city" table with the "player" table based on the matching "pincode" and "location_pincode" columns.

By performing a "LEFT JOIN" , we ensure that all cities from the "city" table are included in the result set, regardless of whether there are matching records in the "player" table. This allows us to capture all cities, even if there are no corresponding citizen records.

Including this join condition enables us to identify cities from which no citizen has placed any bets so far. By checking if there are any corresponding records in the "player" and "transactions" tables using the "WHERE t.trans_id IS NULL" condition, we can determine the cities with no associated bets.

P 7:

2 / 2

emp(eid,ename,salary,did,eid_of_manager)

dept(did,dname)

projects(pid,eid)

find number of managers from aiml dept doing projects

To find the number of managers from the AIML department who are working on projects, you can use the following SQL query:

sql

```
SELECT COUNT(DISTINCT e.eid_of_manager) AS num_managers
FROM emp e
JOIN dept d ON e.did = d.did
JOIN projects p ON e.eid = p.eid
WHERE d.dname = 'AIML';
```

This query joins the "emp" , "dept" , and "projects" tables based on the corresponding columns. It counts the number of distinct "eid_of_manager" values, which represents the managers. The "WHERE" clause filters the records to only consider employees from the AIML department. The result will give you the count of managers from the AIML department who are working on projects.

Please make sure to adjust the table and column names according to your schema if they differ.

P 8:

why we use double time join

Apologies for the confusion caused by the double join in the previous query. To correctly find the number of managers from the AIML department working on projects, we need to perform a single join between the "emp" and "dept" tables and then filter the result to include only managers from the AIML department.

Here's the updated query:

sql

```
SELECT COUNT(*) AS num_managers
FROM emp e
JOIN dept d ON e.did = d.did
WHERE d.dname = 'AIML'
AND e.eid = e.eid_of_manager
AND EXISTS (
    SELECT 1
    FROM projects p
    WHERE p.eid = e.eid
);
```

This query joins the "emp" and "dept" tables based on the "did" column. It then applies the following conditions:

* "d.dname = 'AIML'" : Filters the result to include only employees from the AIML department.
* "e.eid = e.eid_of_manager" : Ensures that the employee is a manager.
* "EXISTS (SELECT 1 FROM projects p WHERE p.eid = e.eid)" : Checks if the manager is working on any projects.

The "COUNT(*)" function is used to count the number of managers who meet these conditions.

Please adjust the table and column names based on your schema if needed.

P 9:

find employ namees from cse whose sal greater than max avg salary of all depts

To find the names of employees from the CSE department whose salary is greater than the maximum average salary across all departments, you can use the following SQL query:

sql

```
SELECT e.ename
FROM emp e
JOIN dept d ON e.did = d.did
WHERE d.dname = 'CSE'
AND e.salary > (
    SELECT MAX(avg_salary)
```

```
    FROM (
        SELECT AVG(salary) AS avg_salary
        FROM emp
        GROUP BY did
    )
);
```

This query joins the "emp" and "dept" tables based on the "did" column. It then applies the following conditions:

* "d.dname = 'CSE'" : Filters the result to include only employees from the CSE department.
* "e.salary > (SELECT MAX(avg_salary) FROM (SELECT AVG(salary) AS avg_salary FROM emp GROUP BY did))" : Compares the employee's salary with the maximum average salary across all departments.

The subquery "(SELECT AVG(salary) AS avg_salary FROM emp GROUP BY did)" calculates the average salary per department, and the outer query selects the maximum value from those averages. This is then used as the threshold for the salary comparison.

The query returns the names of employees from the CSE department whose salary is higher than the maximum average salary of all departments.

Please adjust the table and column names based on your schema if needed.

P 10:

write subquery for above

Certainly! Here's the updated query using a subquery to find the maximum average salary across all departments:

sql

```
SELECT e.ename
FROM emp e
JOIN dept d ON e.did = d.did
WHERE d.dname = 'CSE'
AND e.salary > (
    SELECT MAX(avg_salary)
    FROM (
        SELECT AVG(salary) AS avg_salary
        FROM emp
        GROUP BY did
    ) subquery
);
```

In this query, the subquery "(SELECT AVG(salary) AS avg_salary FROM emp GROUP BY did)" calculates the average salary per department. The outer query then selects the maximum value from those averages.

The subquery is aliased as "subquery" , and we use "subquery" in the main query to compare

the employee's salary with the maximum average salary.

Please adjust the table and column names based on your schema if needed.

P 11:

2 / 2

artists(artist_id,name,location_pincode)

skills(skills_id,skill_name)

artists_skills(artist_id,skill_id,skill_name,exp_in_years)

movies(movie_name,artist_id,language,genre)

find the artists who are both direction and producer for action gentre

To find the artists who are both directors and producers for the action genre in movies, you can use the following SQL query:

sql

```
SELECT a.artist_id, a.name
FROM artists a
JOIN artists_skills as1 ON a.artist_id = as1.artist_id
JOIN artists_skills as2 ON a.artist_id = as2.artist_id
JOIN movies m ON a.artist_id = m.artist_id
WHERE as1.skill_name = 'director'
AND as2.skill_name = 'producer'
AND m.genre = 'action';
```

This query joins the "artists" , "artists_skills" , and "movies" tables based on the corresponding columns. It applies the following conditions:

* "as1.skill_name = 'director'" : Filters the result to include only artists with the skill of a director.
* "as2.skill_name = 'producer'" : Filters the result to include only artists with the skill of a producer.
* "m.genre = 'action'" : Filters the result to include only movies in the action genre.

By joining the "artists_skills" table twice, once for the director skill and once for the producer skill, we ensure that the artist must have both skills to be included in the result set.

The query returns the artist IDs and names of those who are both directors and producers for the action genre in movies.

Please adjust the table and column names based on your schema if needed.

P 12:

find music directors from hyderabad city who have at least 5 years of experience as singer in telugu movies

To find the music directors from Hyderabad city who have at least 5 years of experience as singers in Telugu movies, you can use the following SQL query:

sql

```
SELECT DISTINCT ad.artist_id, ad.name
FROM artists ad
JOIN artists_skills ads ON ad.artist_id = ads.artist_id
JOIN artists_skills asm ON ad.artist_id = asm.artist_id
JOIN movies m ON ad.artist_id = m.artist_id
JOIN skills s ON ads.skill_id = s.skill_id
WHERE ad.location_pincode = 'Hyderabad'
AND asm.skill_name = 'singer'
AND m.language = 'Telugu'
AND asm.exp_in_years >= 5
AND s.skill_name = 'music director';
```

This query joins the "artists", "artists_skills", "movies", and "skills" tables based on the corresponding columns. It applies the following conditions:

* "ad.location_pincode = 'Hyderabad'" : Filters the result to include only artists from Hyderabad city.
* "asm.skill_name = 'singer'" : Filters the result to include only artists with the skill of a singer.
* "m.language = 'Telugu'" : Filters the result to include only Telugu movies.
* "asm.exp_in_years >= 5" : Filters the result to include only artists with at least 5 years of experience as a singer.
* "s.skill_name = 'music director'" : Filters the result to include only artists with the skill of a music director.

By joining the "artists_skills" table twice, once for the singer skill and once for the music director skill, we ensure that the artist must have both skills to be included in the result set.

The query returns the artist IDs and names of music directors from Hyderabad city who have at least 5 years of experience as singers in Telugu movies.

Please adjust the table and column names based on your schema if needed.

P 13:

find all the artists who have commonly worked for kgf rrr and pushpa

To find all the artists who have worked in common for the movies "KGF", "RRR", and "Pushpa," you can use the following SQL query:

sql

```
SELECT a.artist_id, a.name
FROM artists a
JOIN movies m1 ON a.artist_id = m1.artist_id
JOIN movies m2 ON a.artist_id = m2.artist_id
JOIN movies m3 ON a.artist_id = m3.artist_id
WHERE m1.movie_name = 'KGF'
AND m2.movie_name = 'RRR'
AND m3.movie_name = 'Pushpa';
```

This query joins the "artists" table with the "movies" table multiple times based on the

"artist_id" column. It applies the following conditions:

* "m1.movie_name = 'KGF'" : Filters the result to include only artists who have worked in the movie "KGF."
* "m2.movie_name = 'RRR'" : Filters the result to include only artists who have worked in the movie "RRR."
* "m3.movie_name = 'Pushpa'" : Filters the result to include only artists who have worked in the movie "Pushpa."

By joining the "movies" table multiple times, we ensure that only the artists who have worked in all three specified movies are included in the result set.

The query returns the artist IDs and names of the artists who have commonly worked in the movies "KGF," "RRR," and "Pushpa."

Please adjust the table and column names based on your schema if needed.

P 14:

I apologize for the confusion caused by my previous response. Since the table structure provided doesn't have explicit information about which artists have worked on which movies, it is not possible to directly retrieve the artists who have worked in common for "KGF," "RRR," and "Pushpa" using the given tables.

To accomplish this, you would need to have a table that specifically records the relationship between artists and movies they have worked on. Without such a table, it is not possible to retrieve the desired information. If you have additional tables or information available, please

provide more details, and I'll be happy to assist you further.

P 15:

2 / 2

emp(eid,ename,salary,did,eid_of_manager)

dept(did,dname)

projects(pid,eid)

find department names from which none of the employess ar doing atleat one job

To find the department names from which none of the employees are assigned to at least one job/project, you can use the following SQL query:

sql

```
SELECT d.dname
FROM dept d
LEFT JOIN emp e ON d.did = e.did
LEFT JOIN projects p ON e.eid = p.eid
WHERE p.pid IS NULL
GROUP BY d.dname;
```

This query uses a combination of "LEFT JOIN" and "GROUP BY" to identify the departments that have no corresponding records in the "projects" table, indicating that none of the employees in those departments are assigned to any job.

The "LEFT JOIN" ensures that all departments from the "dept" table are included in the result set, regardless of whether there are matching records in the "emp" or "projects" tables. The "WHERE p.pid IS NULL" condition filters out the rows where there are no corresponding records in the "projects" table, meaning no jobs are assigned to employees in those departments.

The "GROUP BY d.dname" clause groups the result by department name, ensuring each department appears only once in the output.

The query returns the department names from which none of the employees are assigned to at least one job/project.

Please adjust the table and column names based on your schema if needed.

P 16:

find employ names who are managers to themselves

To find employee names who are managers to themselves, you can use the following SQL query:

sql

```
SELECT e.ename
FROM emp e
WHERE e.eid = e.eid_of_manager;
```

This query selects the employee names from the "emp" table where the "eid" (employee ID) is equal to the "eid_of_manager" (employee ID of the manager). By comparing these two columns, we can identify employees who are managers to themselves.

The query returns the names of employees who are managers to themselves.

Please adjust the table and column names based on your schema if needed.

P 17:

find employee namee who are not manager to anybody

To find employee names who are not managers to anybody, you can use the following SQL query:

sql

```
SELECT e.ename
FROM emp e
WHERE e.eid NOT IN (SELECT eid_of_manager FROM emp WHERE
eid_of_manager IS NOT NULL);
```

This query uses a subquery to identify the "eid_of_manager" values of employees who are

managers. The subquery "(SELECT eid_of_manager FROM emp WHERE eid_of_manager IS NOT NULL)" retrieves all non-null values of "eid_of_manager" . The main query selects the employee names from the "emp" table where the "eid" (employee ID) is not present in the subquery result, indicating that they are not managers to anybody.

The query returns the names of employees who are not managers to anybody.

Please adjust the table and column names based on your schema if needed.

P 18:

User
find city names from which no citizen bets done so far using count

To find city names from which no citizen has placed any bets so far using the "COUNT" function, you can use the following SQL query:

sql

```
SELECT c.name
FROM city c
LEFT JOIN player p ON c.pincode = p.location_pincode
LEFT JOIN transactions t ON p.player_id = t.player_id
GROUP BY c.name
HAVING COUNT(t.trans_id) = 0;
```

This query joins the "city" table with the "player" and "transactions" tables based on the corresponding columns. By performing a "LEFT JOIN" , all cities from the "city" table are included in the result set, regardless of whether there are matching records in the other tables.

The "GROUP BY c.name" clause groups the result by city name, allowing us to count the number of transactions for each city.

The "HAVING COUNT(t.trans_id) = 0" condition filters out the cities where the count of transactions is 0, meaning no bets have been placed by citizens from those cities.

The query returns the names of cities from which no citizen has placed any bets so far.

Please adjust the table and column names based on your schema if needed.

P 19:

find departments whose avg salary is less than 50k

To find departments whose average salary is less than 50,000, you can use the following SQL query:

sql

```
SELECT d.dname
FROM dept d
JOIN emp e ON d.did = e.did
GROUP BY d.did, d.dname
```

```
HAVING AVG(e.salary) < 50000;
```

This query joins the "dept" and "emp" tables based on the department ID ("did") column. It then applies the following steps:

1. "GROUP BY d.did, d.dname" : Groups the result by department ID and department name, to calculate the average salary per department.
2. "HAVING AVG(e.salary) < 50000" : Filters the result to include only departments where the average salary ("AVG(e.salary)") is less than 50,000.

The query returns the names of departments whose average salary is less than 50,000.

Please adjust the table and column names based on your schema if needed.

P 20:

find number of employes in each department

To find the number of employees in each department, you can use the following SQL query:

sql

```
SELECT d.dname, COUNT(e.eid) AS employee_count
FROM dept d
LEFT JOIN emp e ON d.did = e.did
```

```
GROUP BY d.dname;
```

This query joins the "dept" and "emp" tables based on the department ID ("did") column using a "LEFT JOIN" to include departments even if they have no employees.

The "COUNT(e.eid)" function counts the number of non-null "eid" values (employee IDs) in the "emp" table for each department.

The "GROUP BY d.dname" clause groups the result by department name ("dname"), ensuring each department appears only once in the output.

The query returns the department names along with the number of employees in each department.

Please adjust the table and column names based on your schema if needed.

P 21:

find department wise average salary

To find the department-wise average salary, you can use the following SQL query:

sql

```
SELECT d.dname, AVG(e.salary) AS average_salary
```

```
FROM dept d
JOIN emp e ON d.did = e.did
GROUP BY d.dname;
```

This query joins the "dept" and "emp" tables based on the department ID ("did") column. It then applies the following steps:

1. "GROUP BY d.dname" : Groups the result by department name ("dname"), allowing us to calculate the average salary per department.
2. "AVG(e.salary)" : Calculates the average salary for each department by taking the average ("AVG") of the "salary" column in the "emp" table.

The query returns the department names along with the average salary for each department.

Please adjust the table and column names based on your schema if needed.

P 22:

Subqueries – Use Two tables (emp and dept – common column is did- note dname is in dept only)
(i). Find emp names who are working for CSE and AIML

To find the employee names who are working for both "CSE" and "AIML" departments using two tables ("emp" and "dept") with the common column "did" , you can use the following SQL query:

sql

```
SELECT e.ename
FROM emp e
WHERE e.did IN (SELECT d.did FROM dept d WHERE d.dname = 'CSE')
AND e.did IN (SELECT d.did FROM dept d WHERE d.dname = 'AIML');
```

This query uses subqueries to retrieve the "did" values of the "CSE" and "AIML" departments separately from the "dept" table. The main query selects the employee names ("ename") from the "emp" table where the "did" value exists in both subqueries.

The subqueries "(SELECT d.did FROM dept d WHERE d.dname = 'CSE')" and "(SELECT d.did FROM dept d WHERE d.dname = 'AIML')" retrieve the "did" values of the respective departments.

The main query then filters the employee names by checking if their "did" value is present in both subqueries using the "IN" operator.

The query returns the employee names who are working for both the "CSE" and "AIML" departments.

Please adjust the table and column names based on your schema if needed.

P 23:

use subqueries for above

Certainly! Here's an alternative version of the query using subqueries to find employee names who are working for both "CSE" and "AIML" departments:

sql

```
SELECT e.ename
FROM emp e
WHERE e.did IN (SELECT did FROM dept WHERE dname = 'CSE')
AND e.did IN (SELECT did FROM dept WHERE dname = 'AIML');
```

This query uses two subqueries within the main query's "WHERE" clause. Each subquery retrieves the "did" values of the respective departments from the "dept" table.

The main query then checks if the employee's "did" value exists in both subqueries using the "IN" operator.

The query returns the employee names who are working for both the "CSE" and "AIML" departments.

Please adjust the table and column names based on your schema if needed.

P 24:

. Find emp names who are having salary greater than avg salary of CSE Dept

To find the employee names who have a salary greater than the average salary of the "CSE" department, you can use the following SQL query with subqueries:

sql

```
SELECT e.ename
FROM emp e
WHERE e.salary > (SELECT AVG(e2.salary) FROM emp e2 JOIN dept d ON
e2.did = d.did WHERE d.dname = 'CSE');
```

This query uses a subquery to calculate the average salary of the "CSE" department. The main query selects the employee names ("ename") from the "emp" table where their salary is greater than the average salary of the "CSE" department.

The subquery "(SELECT AVG(e2.salary) FROM emp e2 JOIN dept d ON e2.did = d.did WHERE d.dname = 'CSE')" retrieves the average salary ("AVG(e2.salary)") of employees who belong to the "CSE" department. It joins the "emp" and "dept" tables based on the department ID ("did") to ensure we only consider employees from the "CSE" department.

The main query then compares the employee's salary ("e.salary") with the average salary obtained from the subquery to filter out the employees whose salary is greater.

The query returns the employee names who have a salary greater than the average salary of the

"CSE" department.

Please adjust the table and column names based on your schema if needed.

P 25:

Find the name of employee who is having maximum salary in AIML Dept.

To find the name of the employee who has the maximum salary in the "AIML" department, you can use the following SQL query:

sql

```
SELECT e.ename
FROM emp e
JOIN dept d ON e.did = d.did
WHERE d.dname = 'AIML'
ORDER BY e.salary DESC
FETCH FIRST 1 ROW ONLY;
```

This query joins the "emp" and "dept" tables based on the department ID ("did") to ensure we only consider employees from the "AIML" department. It then applies the following steps:

1. "WHERE d.dname = 'AIML'" : Filters the result to include only employees from the "AIML" department.

2. "ORDER BY e.salary DESC" : Orders the result by salary ("e.salary") in descending order, so that the employee with the maximum salary appears first.
3. "FETCH FIRST 1 ROW ONLY" : Retrieves only the first row of the result, which corresponds to the employee with the maximum salary.

The query returns the name of the employee who has the maximum salary in the "AIML" department.

Please adjust the table and column names based on your schema if needed.

P 26:

use subqueries method

Certainly! Here's an alternative version of the query using subqueries to find the name of the employee who has the maximum salary in the "AIML" department:

sql

```
SELECT e.ename
FROM emp e
WHERE e.salary = (SELECT MAX(e2.salary) FROM emp e2 JOIN dept d ON
e2.did = d.did WHERE d.dname = 'AIML');
```

This query uses a subquery to find the maximum salary ("MAX(e2.salary)") of employees who

belong to the "AIML" department. The subquery joins the "emp" and "dept" tables based on the department ID ("did") to ensure we only consider employees from the "AIML" department.

The main query then selects the employee name ("e.ename") from the "emp" table where their salary ("e.salary") is equal to the maximum salary obtained from the subquery.

The query returns the name of the employee who has the maximum salary in the "AIML" department.

Please adjust the table and column names based on your schema if needed.

P 27:

```
SELECT ename
FROM emp
WHERE sal > (
SELECT avg(sal)
FROM emp
where dno = (select did from dept where dname='CSE'))
```

; is this correct for the above question

The query you provided is checking if an employee's salary is greater than the average salary of the employees in the "CSE" department. However, the original question asks for finding the employee with the maximum salary in the "AIML" department.

To correct the query, you can use the following SQL query:

sql

```
SELECT e.ename
FROM emp e
WHERE e.did = (SELECT d.did FROM dept d WHERE d.dname = 'AIML')
AND e.salary = (SELECT MAX(e2.salary) FROM emp e2 WHERE e2.did =
e.did);
```

This query uses subqueries to accomplish the task.

1. The first subquery "(SELECT d.did FROM dept d WHERE d.dname = 'AIML')" retrieves the department ID ("did") of the "AIML" department.
2. The second subquery "(SELECT MAX(e2.salary) FROM emp e2 WHERE e2.did = e.did)" retrieves the maximum salary from the employees who belong to the same department as the outer query.
3. The main query selects the employee name ("e.ename") from the "emp" table where the employee's department ID matches the "AIML" department and their salary is equal to the maximum salary obtained from the subquery.

The corrected query will return the name of the employee who has the maximum salary in the "AIML" department.

Please adjust the table and column names based on your schema if needed.

P 28:

To find the employee names who have a salary greater than the average salary of the "CSE"

department, you can use the following SQL query with subqueries:

SELECT e.ename

FROM emp e

WHERE e.salary > (SELECT AVG(e2.salary) FROM emp e2 WHERE e2.did = (SELECT did

FROM dept WHERE dname = 'CSE'));

P 29:

Find dept name which is having maximum average salary

To find the department name that has the maximum average salary, you can use the following SQL query:

sql

```
SELECT d.dname
FROM dept d
JOIN emp e ON d.did = e.did
GROUP BY d.dname
HAVING AVG(e.salary) = (SELECT MAX(avg_salary) FROM (SELECT
AVG(e2.salary) AS avg_salary FROM emp e2 GROUP BY e2.did)
subquery);
```

This query calculates the average salary for each department by joining the "dept" and "emp" tables based on the department ID ("did"). It then applies the following steps:

1. "GROUP BY d.dname" : Groups the result by department name ("dname"), allowing us to calculate the average salary per department.
2. "HAVING AVG(e.salary) = (SELECT MAX(avg_salary) FROM (SELECT AVG(e2.salary) AS avg_salary FROM emp e2 GROUP BY e2.did) subquery)" : Filters the result to include only departments whose average salary is equal to the maximum average salary across all departments. The subquery "(SELECT AVG(e2.salary) AS avg_salary FROM emp e2 GROUP BY e2.did)" calculates the average salary per department, and the outer query selects the department names that have the maximum average salary.

The query returns the department name that has the maximum average salary.

Please adjust the table and column names based on your schema if needed.

P 30:

do above query using joins

To find the department name that has the maximum average salary using joins, you can use the following SQL query:

sql

```
SELECT d.dname
FROM dept d
JOIN (
```

```
    SELECT e.did, AVG(e.salary) AS avg_salary
    FROM emp e
    GROUP BY e.did
) subquery ON d.did = subquery.did
WHERE subquery.avg_salary = (
    SELECT MAX(avg_salary)
    FROM (
        SELECT AVG(e2.salary) AS avg_salary
        FROM emp e2
        GROUP BY e2.did
    ) subquery2
);
```

This query uses joins to calculate the average salary per department and find the department with the maximum average salary. Here's how it works:

1. The inner subquery "(SELECT e.did, AVG(e.salary) AS avg_salary FROM emp e GROUP BY e.did)" calculates the average salary per department by joining the "emp" table on the department ID ("did") and using the "AVG" function.
2. The outer query joins the "dept" table with the subquery using the department ID ("did") to get the department names along with their respective average salaries.
3. The WHERE clause filters the result to include only departments whose average salary is equal to the maximum average salary.
4. The subquery "(SELECT MAX(avg_salary) FROM (SELECT AVG(e2.salary) AS avg_salary FROM emp e2 GROUP BY e2.did) subquery2)" calculates the maximum average salary across all departments using the inner subquery.

The query returns the department name that has the maximum average salary.

Please adjust the table and column names based on your schema if needed.

P 31:

SELECT d.dname, AVG(salary) AS average_sal
FROM dept d
JOIN employee e ON e.did=d.did
GROUP BYd.dname
ORDER BY average_sal DESC
FETCH FIRST 1 ROWS ONLY; is this correct solution

Yes, the query you provided is correct for finding the department name with the maximum average salary. However, there is a small syntax error in your query. The "GROUP BY" clause is missing a space before the "d.dname" column. Here's the corrected version:

sql

```
SELECT d.dname, AVG(e.salary) AS average_sal
FROM dept d
JOIN employee e ON e.did = d.did
GROUP BY d.dname
ORDER BY average_sal DESC
FETCH FIRST 1 ROWS ONLY;
```

This query joins the "dept" and "employee" tables based on the department ID ("did"). It calculates the average salary ("AVG(e.salary)") for each department using the "GROUP BY" clause on the department name ("d.dname"). The result is then ordered in descending order of average salary using the "ORDER BY" clause. Finally, the "FETCH FIRST 1 ROWS ONLY" clause limits the result to only the first row, which represents the department with the maximum average salary.

The corrected query will give you the department name with the highest average salary.

Please adjust the table and column names based on your schema if needed.

P 32:

Find emp names who are having salary greater than avg salary of CSE Dept.

To find the employee names who have a salary greater than the average salary of the "CSE" department, you can use the following SQL query:

sql

```
SELECT e.ename
FROM emp e
WHERE e.salary > (
    SELECT AVG(e2.salary)
    FROM emp e2
    WHERE e2.did = (SELECT did FROM dept WHERE dname = 'CSE')
```

```
);
```

This query uses subqueries to accomplish the task. Here's how it works:

1. The subquery "(SELECT did FROM dept WHERE dname = 'CSE')" retrieves the department ID of the "CSE" department.
2. The subquery "(SELECT AVG(e2.salary) FROM emp e2 WHERE e2.did = (SELECT did FROM dept WHERE dname = 'CSE'))" calculates the average salary of employees in the "CSE" department.
3. The main query selects the employee names ("e.ename") from the "emp" table where their salary ("e.salary") is greater than the average salary obtained from the subquery.

The query will return the names of the employees who have a salary greater than the average salary of the "CSE" department.

Please adjust the table and column names based on your schema if needed.

P 33:

Find the name of employee who is having maximum salary in AIML Dept.

To find the name of the employee who has the maximum salary in the "AIML" department, you can use the following SQL query:

sql

```
SELECT e.ename
FROM emp e
WHERE e.did = (SELECT d.did FROM dept d WHERE d.dname = 'AIML')
AND e.salary = (SELECT MAX(e2.salary) FROM emp e2 WHERE e2.did =
e.did);
```

This query uses subqueries to accomplish the task. Here's how it works:

1. The subquery "(SELECT d.did FROM dept d WHERE d.dname = 'AIML')" retrieves the department ID ("did") of the "AIML" department.
2. The subquery "(SELECT MAX(e2.salary) FROM emp e2 WHERE e2.did = e.did)" calculates the maximum salary from the employees who belong to the same department as the outer query.
3. The main query selects the employee name ("e.ename") from the "emp" table where the employee's department ID matches the "AIML" department and their salary is equal to the maximum salary obtained from the subquery.

The query will return the name of the employee who has the maximum salary in the "AIML" department.

Please adjust the table and column names based on your schema if needed.

P 34:

Find the name of employee who is having maximum salary in AIML Dept.

To find the name of the employee who has the maximum salary in the "AIML" department, you

can use the following SQL query:

sql

```
SELECT e.ename
FROM emp e
JOIN dept d ON e.did = d.did
WHERE d.dname = 'AIML'
ORDER BY e.salary DESC
FETCH FIRST 1 ROW ONLY;
```

This query joins the "emp" and "dept" tables based on the department ID ("did") to ensure we only consider employees from the "AIML" department. It then applies the following steps:

1. "WHERE d.dname = 'AIML'" : Filters the result to include only employees from the "AIML" department.
2. "ORDER BY e.salary DESC" : Orders the result by salary ("e.salary") in descending order, so that the employee with the maximum salary appears first.
3. "FETCH FIRST 1 ROW ONLY" : Retrieves only the first row of the result, which corresponds to the employee with the maximum salary.

The query will return the name of the employee who has the maximum salary in the "AIML" department.

Please adjust the table and column names based on your schema if needed.