SQL 08 : SUB-QUERY & VIEWS

20/05/24

AGENDA

1. Sub-Query - why & what?
2. WHERE
3. IN
4. FROM
5. ALL, ANY
6. CO-RELATED
7. EXISTS
8. VIEWS.
9. Doubts.

SUB-QUERY

→ Select stmt within another stmt.
→ All CRUD operations.
→ Break down a complex problem
  - into simple sub problems
  - add them back to get the complete solution.
→ Intuitive - simpler way to write queries.

```
SELECT
    .  .  .
        (SELECT
            .  .  .  SELECT.
```

Q.1 Print name & psp of all students whose psp > psp of stu_id = 5.

table: students

Soln:

```
SELECT  S1.name, S1.psp
FROM    students S1
JOIN    students S2
ON      S2.sid = 5          S2.psp
AND     S1.psp > S2.psp;
```

Solution 2 ALGORITHM :-

1. find the psp of stu-id = 5 → psp-id-5.
2. find all the students whose psp > psp-id-5

1. SELECT psp FROM STUDENTS WHERE sid = 5; → psp-id-5

2. SELECT name, psp FROM STUDENTS WHERE psp > (psp-id-5)

1 + 2

```
SELECT name, psp
FROM STUDENTS
WHERE psp > (SELECT psp FROM STUDENTS
             WHERE sid = 5);
```

Q.2 Print the name and psp of all students whose psp > max psp of bid = 2.

Soln ALGORITHM

1. Find the max psp of bid = 2 → max-psp-b2
2. find all students whose psp > max-psp-b2.

Combine

```
SELECT name, psp
FROM STUDENTS
WHERE psp > ( SELECT MAX(PSP)
              FROM STUDENTS
              WHERE B_ID = 2 );
```

PROS -> Easy / Intuitive to understand.

CONS -> Performance tradeoff.

Pseudo Code

```
students = []
ans = []

for s1 in students:
    max_psp_b2 = 0
    for s2 in students:
        if s2.bid == 2:
            max_psp_b2 = max(max_psp_b2, s2.psp)

    if s1.psp > max_psp_b2:
        ans.add(s1)

for each row in ans:
    print(row['name'], row['psp'])
```

TC = O(N^2)

row	col
1	1
1	m
m	1
m	m

WHERE
row.
col.
table, sub-table

Q3

Table: Students

id	name	is_stud	is_ta
1	Raushan	1	1
2	Sumit	1	0
3	Rago	1	0
4	Asar	1	0
5	Sumit	0	1

Raushan
Sumit.

[Raushan, Sumit]

Print the name of all students whose name is same as TA ; need NOT be same person

JOINS

Soln1

```
SELECT   S.name
FROM     Students  S
JOIN     students  T
ON       S.name = T.name
  AND    S.is_stud = 1
  AND    T.is_TA = 1 ;
```

Soln2

ALGORITHM

1. Get name of all TAs → ta_list.
2. Print all students whose name is in - ta_list.

①
```
SELECT name
From students
WHERE is_ta = 1;
```
→ ta list.

②
```
SELECT name
From Students
WHERE is_Stud = 1
AND name IN ( SELECT name
              From students
              WHERE is_ta = 1);
```

Qp Print the name of all students whose PSP is NOT less than the least PSP of ANY batch.

Bid	Min(PSP)
1	80
2	50
3	60
4	70

A, PSP = 82 ✓

B, PSP = 62 ✗

PSP > (max of min PSP of every batch)

PSP > 80

ALGORITHM

① Find the min psp of each batch → min_psps
② Find the max of min_psps → max_of_min_psps.
③ Find all students whose psp > max_of_min_psps.

① SELECT MIN(PSP)
FROM STUDENTS
GROUP BY B_ID ; — min_psps.

② SELECT MAX(PSP)
FROM min_psps ; → max_of_min_psps

③ SELECT name
FROM STUDENTS
WHERE PSP > max_of_min_psps
Atc(PSP)

Combine ① & ② & ③ :-

```
SELECT name
FROM STUDENTS
WHERE PSP > ( SELECT MAX(MIN_PSPS)
              FROM
                  ( SELECT MIN(PSP) as MIN_PSPS
                    FROM STUDENTS
                    GROUP BY B_ID) min_psp_list
            );
```

NOTE: MANDATORY TO GIVE AN ALIAS TO SUB-QUERY AFTER "FROM"

eg If psp = 65 ✗ False psp = 85 ✓

① PSP > ALL (80, 50, 70, 60) — AND

psp = 65 ✓ TRUE

② PSP > ANY (80, 50, 70, 60) — OR.

PSP IN (80, 50, 70, 60)

Soln
```
SELECT name
FROM STUDENTS
WHERE PSP > ALL( SELECT MIN(PSP)
                 FROM STUDENTS
                 GROUP BY B-ID );
```

Ques Print name of all students whose PSP > Avg PSP of their own batch.

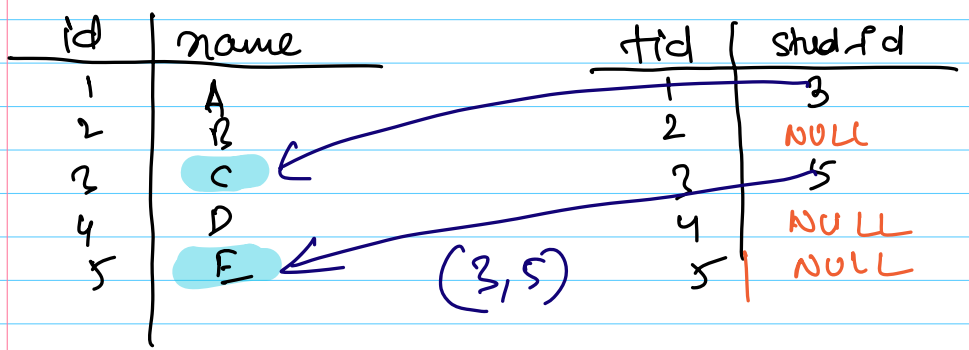
bid = 2

```
SELECT name, psp
FROM Students S
WHERE PSP > ( SELECT AVG(PSP)
              FROM STUDENTS
              WHERE bid = S.bid );
```

Co-RELATED Sub-query.

NOTE: ALIAS IS MANDATORY !

Ques6 table - students table : TA.



→ Print all students name who is also a TA

ALGORITHM

① Find all TA who is a student.
→ stud-id IS NOT NULL → table

② For each student check if its name is present in the → TA-UST.

```
SELECT name
FROM STUDENTS
WHERE STUD-ID IN ( SELECT stu-id
                   FROM TA
                   WHERE stu-id IS NOT NULL );
```

EXISTS

```
SELECT name
FROM Students S
WHERE EXISTS ( SELECT stu-id
               FROM TA
               WHERE ta.stu-id = S.stu-id );
```

NOTE: EXISTS returns TRUE - if the subquery returns > 0 rows.

→ Need not go through all rows.

VIEWS

eg Sakila DB

① film, actor, film_actor
↓ ↓ ↓
film name actor_names mapping.

films	actors
xyz	a1
xyz	a2
xyz	a3
abc	a1
abc	a2

if actor - "name"
→ name of films;

if filmname = 'xyz'
give all actor_names.

a1 → xyz, abc.
xyz → a1, a2, a3

→ filter

① Focus, customize, simplify data visualisation
② Single place to change visualisation
③ "Hide schema" → security reasons.

LINKS :

https://dev.mysql.com/doc/refman/8.0/en/subquery-optimization.html
https://dev.mysql.com/doc/refman/8.0/en/subquery-materialization.html

Materialization speeds up query execution by generating a subquery result as a temporary table, normally in memory. The first time MySQL needs the subquery result, it materializes that result into a temporary table. Any subsequent time the result is needed, MySQL refers again to the temporary table. The optimizer may index the table with a hash index to make lookups fast and inexpensive. The index contains unique values to eliminate duplicates and make the table smaller.

NOTE: In general, you cannot modify a table and select from the same table in a subquery

QUERIES

```
-- sql 08 subquery
SELECT S1.S_NAME, S1.PSP
FROM STUDENTS S1
JOIN STUDENTS S2
ON S2.S_ID=5 AND S1.PSP>S2.PSP;

-- Q1 SUB QUERY
SELECT S_NAME, PSP
FROM STUDENTS
WHERE PSP > (SELECT PSP FROM STUDENTS
                    WHERE S_ID=5);


-- Q2 ALLSTUDENTS WITH PSP > MAX PS OF B_ID = 2
SELECT S_NAME, PSP
FROM STUDENTS
WHERE PSP > (SELECT MAX(PSP) FROM STUDENTS
                    WHERE B_ID = 3);


-- Q3
-- JOINS
SELECT S.S_NAME
FROM STUDENTS S
JOIN STUDENTS T
ON S.S_NAME= T.S_NAME
AND S.IS_STUD=1
AND T.IS_TA=1;
```

```
-- SUB QUERY
SELECT S_NAME FROM STUDENTS
WHERE IS_STUD = 1
AND S_NAME IN (SELECT S_NAME FROM STUDENTS
                    WHERE IS_TA=1);



-- Q4

SELECT S_NAME, PSP
FROM STUDENTS
WHERE PSP > ( SELECT MAX(MIN_PSPS)
              FROM
                    (SELECT MIN(PSP) AS MIN_PSPS
          FROM STUDENTS
          GROUP BY B_ID) MIN_PSP_LIST
              );

-- Q5 CO-RELATED SUB QUERY
SELECT S_NAME, PSP
FROM STUDENTS S
WHERE PSP > (SELECT AVG(PSP) FROM STUDENTS
                    WHERE B_ID=S.B_ID);

-- Q6 EXISTS

SELECT S_NAME
FROM STUDENTS
WHERE S_ID IN (SELECT S_ID FROM TA WHERE S_ID IS NOT NULL);

SELECT S_NAME
FROM STUDENTS S
WHERE EXISTS (SELECT S_ID FROM TA WHERE TA.S_ID = S.S_ID);
```

```
-- VIEWS
USE SAKILA;

CREATE OR REPLACE view actor_film_name AS
SELECT
   concat(a.first_name,' ', a.last_name) AS actor_name,
   f.title AS film_name
FROM actor a
  JOIN film_actor fa
    ON fa.actor_id = a.actor_id
  JOIN film f
    ON f.film_id = fa.film_id;


SELECT FILM_NAME FROM actor_film_name
WHERE ACTOR_NAME LIKE 'JOE%';

SELECT ACTOR_NAME FROM actor_film_name
WHERE FILM_NAME LIKE 'LOVE%';

SHOW FULL TABLES WHERE table_type = 'VIEW';
```