S35: SQL SUBQUERIES

WHAT IS SQL SUBQUERY?

- In **SQL**, a subquery also known as an **inner query** or **nested query**, is a query embedded within another **SQL** query.
- It is a **SELECT** statement that is nested inside another **SELECT, INSERT, UPDATE**, or **DELETE** statement.
- The subquery is executed first, and its result is then used as a parameter or condition for the outer query.
- **Order of execution**
  Inner query **>** Inner query generate the result pass through outer query **>** Outer query
- **What is the scope of inner query?**
  Until Outer query completes.

TYPES OF SUBQUERIES

- Subqueries in **SQL** can be categorized **based on** the **results they return** and **how they work.**
- Here is a detailed breakdown of subqueries according to these criteria:

BASED ON - THE RESULTS THEY RETURN

- Here, subqueries can be categorized into three types: Scalar subqueries, Row subqueries, and Table subqueries

SCALAR SUBQUERY

- Returns a single value (single row and single column)

ROW SUBQUERY

- Returns a single row with multiple columns.

TABLE SUBQUERY

- Returns multiple rows and multiple columns.

BASED ON - WORKING

- Here, subqueries can be categorized into three types: Scalar subqueries, Row subqueries, and Table subqueries

INDEPENDENT (NON-CORRELATED) SUBQUERY

- Independent of the outer query and executes once.

CORRELATED SUBQUERY

- Depends on the outer query and executes once for each row processed by the outer query.

WHERE CAN SUBQUERIES BE USED?

- Subqueries can be used in the following contexts within **SQL** statements:

INSERT STATEMENT

- **INSERT INTO ... SELECT:**
  - ♣ Subqueries can be used to insert data into a table based on the results of another query.
  - ♣ **Example: INSERT INTO target_table (columns) SELECT columns FROM source_table WHERE condition;**

SELECT STATEMENT

- Subqueries can be used in the **SELECT list**, **FROM clause**, **WHERE clause**, and **HAVING clause**.
- **SELECT List:**
  - ♣ To include a computed column based on the results of a subquery.
  - ♣ **Example: SELECT column, (SELECT Subqueries can be used in the SELECT list, FROM clause, WHERE clause, and HAVING clause.ubquery) AS alias FROM table;**
- **FROM Clause:**
  - ♣ To use a subquery as a derived table or inline view.
  - ♣ **Example: SELECT columns FROM (SELECT subquery) AS alias;**
- **WHERE Clause:**
  - ♣ To filter rows based on the results of a subquery.

- ♣ **Example: SELECT columns FROM table WHERE column = (SELECT subquery);**
- **HAVING Clause:**
  - ♣ To filter groups based on the results of a subquery.
  - ♣ **Example: SELECT columns FROM table GROUP BY column HAVING aggregate_function > (SELECT subquery);**

UPDATE STATEMENT

- Subqueries can be used to update records in a table based on the results of another query.
- **SET Clause:** To update column values based on the results of a subquery.
- **WHERE Clause:** To specify which rows to update based on the results of a subquery.
  - ♣ **Example: UPDATE table SET column = value WHERE column = (SELECT subquery);**

DELETE STATEMENT

- Subqueries can be used to delete records from a table based on the results of another query.
- **WHERE Clause:** To specify which rows to delete based on the results of a subquery.
  - ♣ **Example: DELETE FROM table WHERE column = (SELECT subquery);**

INDEPENDENT SUBQUERY – SCALAR SUBQUERY EXAMPLE

```
# Independent Subquery - Scalar Subquery

-- Ex1. Find the movie with highest profit (vs order by)
SELECT * FROM movies
WHERE (gross - budget) = (
        SELECT MAX(gross - budget) AS max_profit FROM movies
);

-- Ex2. Find how many movies avg a rating > the avg of all the movie rating (Find the count of above average movies)
SELECT COUNT(*) FROM movies
WHERE score > (
        SELECT AVG(score) FROM movies
);

-- Ex3. Find the highest rated movie of 2000
SELECT * FROM movies
WHERE year = 2000 AND score = (
        SELECT MAX(score) FROM movies
        WHERE year = 2000
);

-- Ex4. Find the highest rated among all movies whose number of voters as > the data avg votes
SELECT * FROM movies
WHERE score = (
        SELECT MAX(score) FROM movies
        WHERE votes > (
                SELECT AVG(votes) FROM movies
        )
);
```

INDEPENDENT SUBQUERY – ROW SUBQUERY EXAMPLES

```
# Independent Subquery - Row Subquery (one column and multiple rows)

-- Ex1. find all users who never ordered
SELECT * FROM users
WHERE user_id NOT IN (
        SELECT DISTINCT(user_id)
        FROM orders
);
```

```sql
-- Ex2. Find all the movies made by top 3 directors (in terms of total gross in income)
  SELECT DISTINCT(director) FROM movies
  WHERE director IN (
    SELECT director
    FROM (
      SELECT director FROM movies
      GROUP BY director
      ORDER BY SUM(gross) DESC
      LIMIT 3
    ) AS top_directors #derived table
  );
/*
Explanation: The subquery inside the parentheses (SELECT director FROM (SELECT director FROM movies GROUP BY director ORDER BY SUM(gross) DESC LIMIT 3) AS top_directors) selects the top 3 directors based on the sum of their gross earnings and forms a derived table top_directors. The outer query selects all movies where the director is in the list of top directors obtained from the derived table.
*/

-- Ex3. Find all movies of all those actors whose filmography's avg rating > 8.5 (take 25000 votes as cutoff)
SELECT * FROM movies
WHERE star IN (
            SELECT star FROM movies
            WHERE votes > 25000
            GROUP BY star
            HAVING AVG(score) > 8.5
);
```

INDEPENDENT SUBQUERY – TABLE SUBQUERY EXAMPLES

```sql
# Independent Subquery - Table Subquery (Multiple column and multiple rows)

-- 1. Find the most profitable movie of each year
SELECT * FROM movies
WHERE (year, gross-budget) IN (
            SELECT year , MAX(gross-budget)
            FROM movies
            GROUP BY year
);

-- 2. Find the highest rated movie of each genre vote cutoff of 25000
  SELECT * FROM movies
  WHERE (genre, score) IN (
              SELECT genre, score
              FROM (
                        SELECT genre, MAX(score) AS max_score
                        FROM movies
                        WHERE votes > 25000
                        GROUP BY genre
              ) AS temp # derived table
  ) AND votes > 25000;
/*
SELECT * FROM movies
WHERE (genre, score) IN (
    SELECT genre, MAX(score) AS max_score
    FROM movies
    WHERE votes > 25000
    GROUP BY genre
) AND votes > 25000;
-- Error Code: 2013. Lost connection to MySQL
server during query
*/

-- 3. Find the highest grossing movies of top 5 actor/ director combo in terms of total gross income
-- CTE
WITH top_duos AS (
            SELECT star, director, MAX(gross)
            FROM movies
            GROUP BY star, director
            ORDER BY SUM(gross) DESC LIMIT 5
)

SELECT * FROM movies
WHERE (star, director, gross) IN (
            SELECT * FROM top_duos
);
```

```sql
-- Ex4. Find the highest rated among all movies whose number of voters as > the data avg votes
SELECT * FROM movies
WHERE score = (
            SELECT MAX(score) FROM movies
            WHERE votes > (
                        SELECT AVG(votes) FROM movies
            )
);
```

CORRELATED SUBQUERY EXAMPLES

```sql
# Correlated subquery : Inner query depend on outer query

-- 1. Find all the movies that have rating higher than the average rating of movies in the same genre
SELECT * FROM movies AS M1
WHERE score > (
            SELECT AVG(score)
            FROM movies AS M2
            WHERE M2.genre = M1.genre
);

-- 2.Find the favourite food of each customer
USE zomato;
-- CTE
WITH fav_food AS (
            SELECT
                ZO.user_id,
                ZU.name,
                ZF.f_name,
                COUNT(ZF.f_name) AS freq_of_food
            FROM
                users AS ZU
                JOIN orders AS ZO ON ZU.user_id = ZO.user_id
                JOIN order_details AS ZOD ON ZO.order_id = ZOD.order_id
                JOIN food AS ZF ON ZOD.f_id = ZF.f_id
            GROUP BY
                ZO.user_id, ZU.name, ZF.f_name
)

SELECT *
FROM fav_food AS F1
WHERE freq_of_food = (
            SELECT MAX(freq_of_food)
            FROM fav_food AS F2
            WHERE F2.user_id = F1.user_id
);
```

USAGE WITH SELECT

```sql
# Usage with SELECT
-- 1. Get the percentage of votes for each movie compared to the total number of votes
SELECT
        name,
        (votes/(SELECT SUM(votes) FROM movies))*100 AS percentage
FROM  movies;

-- 2. Display all movie names, genre, score and avg(score) of genre
SELECT  name, genre, score,
        (
            SELECT AVG(score)
            FROM movies M2
            WHERE M2.genre = M1.genre
        ) AS average_score
FROM movies AS M1;
```

- Using correlated subqueries within a **SELECT** statement, as in the example provided, can sometimes be inefficient, especially on large datasets.
- This inefficiency arises because the subquery may need to be executed multiple times once for each row in the outer query.
- However, depending on the database system and the specific scenario, the database optimizer might handle such subqueries efficiently.

USAGE WITH FROM

```
# Usage with FROM
-- 1. Display average rating of all the restaurants
USE zomato;
SELECT r_name, average_rating
FROM (
        SELECT r_id, AVG(restaurant_rating) AS average_rating
        FROM  orders
        GROUP BY r_id
) AS T1
JOIN restaurants AS T2 ON T1.r_id = T2.r_id;
```

USAGE WITH HAVING

```
# Usage with HAVING
-- 1. Find genres having avg score greater than avg score of all the movies
SELECT genre, AVG(score) AS average_score
FROM movies
GROUP BY genre
HAVING AVG(score) > (SELECT AVG(score) FROM movies) ;
```

SUBQUERY IN INSERT

```
# Subquery in INSERT
-- Populate a already created loyal_customers table with records of only those customers who have ordered food more than 3 times
USE zomato;
-- Creating loyal_users table
CREATE TABLE loyal_users
(
        user_id INT NOT NULL PRIMARY KEY,
        name VARCHAR(255) NOT NULL,
        money INT
);

INSERT INTO loyal_users(user_id, name)
SELECT ZU.user_id, ZU.name
FROM orders AS ZO
JOIN users AS ZU ON ZO.user_id = ZU.user_id
GROUP BY ZU.user_id, ZU.name
HAVING COUNT(*) > 3;
```

SUBQUERY IN UPDATE

```
# Subquery in UPDATE
-- Populate the money col of loyal _customer table using orders table & Provide a 10% app money to all customers based on their order value

# 1st way solve:
UPDATE loyal_users AS ZLU
SET money = (
    SELECT SUM(amount) * 0.1
    FROM orders AS ZO
    WHERE ZO.user_id = ZLU.user_id
);

# 2nd way solve:
UPDATE loyal_users AS ZLU
JOIN (
    SELECT user_id, SUM(amount) * 0.1 AS total_app_money
    FROM orders
    GROUP BY user_id
) AS ZO ON ZLU.user_id = ZO.user_id
SET ZLU.money = ZO.total_app_money;
```

SUBQUERY IN DELETE

```
# Subquery in DELETE
-- DELETE all the customers record who have never ordered
USE zomato;
DELETE FROM users
WHERE user_id IN (
        SELECT user_id
        FROM
                (
                SELECT user_id
                FROM users
                WHERE user_id NOT IN (
                        SELECT DISTINCT(user_id)
                        FROM orders
                )
        ) AS most_inner_subquery
);
```