It is like the concept of subquery exists because we cannot create variables in SQL

This is because SQL is a nonprocedural programming language.

You cannot make variables and loops

How can I know if it is a subquery

* ‘Jo aisa ho’ jab aayega jab use karna hai

SELECT MAX(score) FROM session\_35.movies;

SELECT \* FROM session\_35.movies

WHERE score = 9.3

* We can convert the above code to subquery like so

SELECT \* FROM session\_35.movies

WHERE score = (SELECT MAX(score) FROM session\_35.movies)

SELECT MAX(score) FROM session\_35.movies

Is called as inner query

SELECT \* FROM session\_35.movies

WHERE score = ()

Is called as outer query

**Independent Subquery -Scalar Subquery**

Scalar subquery will give only a single value

1. Find the movie with highest profit(vs order by)

USE session\_35;

SELECT \* FROM session\_35.movies

WHERE (gross - budget) = (SELECT MAX(gross - budget) FROM movies);

We can also use Order by and do this question

SELECT \* FROM movies

ORDER BY (gross - budget) DESC LIMIT 1

But on a big dataset. 2nd query will be faster

Because Order by uses indexing.

If we don’t consider indexing and are looking on the basis of time complexity, then the sub query will be faster

Sub Query = O(n) + O(n) = O(2n) = O(n)

ORDER BY = O(nlogn)

1. Find how many movies have a rating > the avg of all the movie ratings(Find the count of above average movies)

SELECT COUNT(\*) FROM movies

WHERE score > (SELECT AVG(score)

FROM movies)

1. Find the highest rated movie of 2000

SELECT \* FROM movies

WHERE year = 2000 and score = (SELECT MAX(score)

FROM movies

WHERE year = 2000)

1. Find the highest rated movie among all movies whose number of votes are > the dataset avg votes

SELECT \* FROM movies

WHERE score = (SELECT MAX(score)

FROM movies

WHERE votes > (SELECT AVG(votes)

FROM movies))

**Independent Subquery -Row Subquery(One Col Multi Rows)**

1. Find all users who never ordered

SELECT \* FROM users

WHERE user\_id NOT IN (SELECT DISTINCT(user\_id) FROM orders)

1. Find all the movies made by top 3 directors(in terms of total gross income)

SELECT \* FROM movies

WHERE director IN (SELECT director

FROM movies

GROUP BY director

ORDER BY SUM(gross) DESC LIMIT 3)

Above code has not worked because the version does not support LIMIT yet

We have tried different way: By common table expression

WITH top\_directors AS (SELECT director

FROM movies

GROUP BY director

ORDER BY SUM(gross) DESC LIMIT 3)

SELECT \* FROM movies

WHERE director IN (SELECT \* FROM top\_directors)

When we user WITH, we make a temporary table and then we can use that inside our main SELECT

1. 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)

AND votes > 25000

**Independent Subquery -Table Subquery(Multi Col Multi Row)**

As you have noticed in Scalar Subquery, we were using = or >, < in WHERE,

While in Row Subquery and Table Subquery we will use IN or NOT IN instead of = or >, <

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)

1. Find the highest rated movie of each genre votes cutoff of 25000

SELECT \* FROM movies

WHERE (genre, score) IN (SELECT genre, MAX(score)

FROM movies

WHERE votes > 25000

GROUP BY genre)

AND votes > 25000

1. Find the highest grossing movies of top 5 actor/director combo in terms of total gross income

WITH top\_duos AS (

SELECT star, director, MAX(gross)

FROM movies

GROUP BY star, director

ORDER BY SUM(gross) LIMIT 5

)

SELECT \* FROM movies

WHERE (star, director, gross) IN (SELECT \* FROM top\_duos)

**Correlated Subquery**

<https://infytq.onwingspan.com/web/en/app/toc/lex_auth_0127673005629194241_shared/overview>

1. Find all the movies that have a rating higher than the average rating of movies in the same genre.[Animation]

SELECT \* FROM movies m1

WHERE score > (SELECT AVG(score) FROM movies m2 WHERE m2.genre = m1.genre)

It is something like inner sub query is using the value from outside

1. Find the favorite food of each customer.

Logic – First we calculate how much has each customer ordered a particular food item

WITH fav\_food AS (

SELECT t2.user\_id, name, f\_name, COUNT(\*) AS 'frequency' FROM users t1

JOIN orders t2 ON t1.user\_id = t2.user\_id

JOIN order\_details t3 ON t2.order\_id = t3.order\_id

JOIN food t4 ON t3.f\_id = t4.f\_id

GROUP BY t2.user\_id,t3.f\_id

)

SELECT \* FROM fav\_food f1

WHERE frequency = (SELECT MAX(frequency)

FROM fav\_food f2

WHERE f2.user\_id = f1.user\_id)

**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

FROM movies

1. Display all movie names ,genre, score and avg(score) of genre

-> Why this is inefficient?

Because in a way we are traversing the complete movies 2 time for 2 selects. Like we are selecting each row and traversing through the whole movies using SELECT

SELECT name, genre, score,

(SELECT AVG(score) FROM movies m2 WHERE m2.genre = m1.genre)

FROM movies m1

**Usage with FROM**

1. Display average rating of all the restaurants

SELECT r\_name, avg\_rating

FROM (SELECT r\_id, AVG(restaurant\_rating) AS 'avg\_rating'

FROM orders

GROUP BY r\_id) t1 JOIN restaurants t2

ON t1.r\_id = t2.r\_id

**Usage with HAVING**

1. Find genres having avg score > avg score of all the movies

SELECT genre, AVG(score) FROM movies

GROUP BY genre

HAVING AVG(score) > (SELECT AVG(score) FROM movies)

**Subquery In INSERT**

Populate an already created loyal\_customers table with records of only those customers who have ordered food more than 3 times.

INSERT INTO loyal\_users (user\_id, name)

SELECT t1.user\_id, t2.name

FROM orders t1

JOIN users t2 ON t1.user\_id = t2.user\_id

GROUP BY t1.user\_id, t2.name

HAVING COUNT(\*) > 3

**Subquery in UPDATE**

Populate the money col of loyal\_cutomer table using the orders table. Provide a 10% app money to all customers based on their order value.

UPDATE loyal\_users

SET money = (

SELECT SUM(amount) \* 0.1

FROM orders

WHERE orders.user\_id = loyal\_users.user\_id

)

**Subquery in DELETE**

Delete all the customers record who have never ordered.

DELETE FROM users

WHERE user\_id IN (SELECT user\_id FROM users

WHERE user\_id NOT IN (SELECT DISTINCT(user\_id) FROM orders))