

# **Lab Manual for Database Management System**

## **Lab No. 6** **Sub Queries**

## **LAB 6: SUB QUERIES**

### **1. INTRODUCTION:**

A subquery, also known as a nested query or inner query, is a powerful SQL construct that allows for the embedding of one SQL query within another. It serves as a means to retrieve data conditionally or to perform operations based on the results of another query. Subqueries are enclosed within parentheses and can be used in various SQL clauses, including SELECT, FROM, WHERE, and HAVING. The primary purpose of a subquery is to simplify complex queries by breaking them down into more manageable components. By incorporating subqueries, SQL developers can create dynamic and adaptable queries, making it possible to perform operations such as filtering, sorting, and aggregating data based on the outcome of a nested query.

### **2. OBJECTIVE:**

After completing this lab the student should be able to:

- a. Employ subqueries for efficient data retrieval from multiple tables.*
- b. Combine information from two tables into a single result set using subqueries.*
- c. Gain proficiency in utilizing self joins within SQL Server for intricate data relationships.*

### **3. THEORY**

A “sub query” is a normal SQL query that is nested inside another query. The outer query is called as **main query** and inner query is called as **subquery**. They are created using parentheses when you have a SELECT statement that serves as the basis for either part of the data or the condition in another query. “Sub queries” are generally used to fill one of a couple of needs:

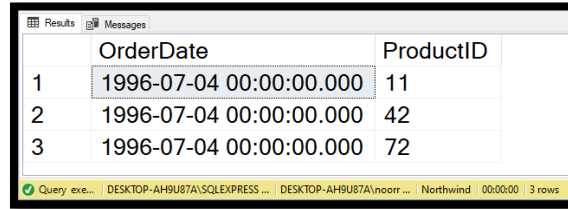
- Break a query up into a series of logical steps
- Provide a listing to be the target of a WHERE clause together with [IN|EXISTS|ANY|ALL]
- To provide a lookup driven by each individual record in a parent query

#### **Example**

1. We wanted to know the ProductIDs of every item sold on the first day any product was purchased from the system. If you already know the first day that an order was placed in the system, then it’s no problem; the query would look something like this:

```
SELECT DISTINCT o.OrderDate, od.ProductID
FROM Orders o
JOIN [Order Details] od
ON o.OrderID = od.OrderID
WHERE OrderDate = '7/4/1996' --This is first OrderDate in the system
```

This yields us the correct results:



	OrderDate	ProductID
1	1996-07-04 00:00:00.000	11
2	1996-07-04 00:00:00.000	42
3	1996-07-04 00:00:00.000	72

Using “Sub Query” it will be done as follows:

```
SELECT DISTINCT o.OrderDate, od.ProductID FROM Orders o
JOIN [Order Details] od
ON o.OrderID = od.OrderID
WHERE o.OrderDate = (SELECT MIN(OrderDate) FROM Orders)
```

### Categories of Sub Queries

Subqueries can be categorized based on their usage and the placement within a SQL statement. Here are the common types of subqueries:

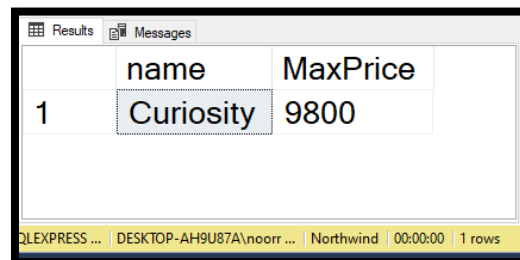
#### a. Scalar Sub Query:

A scalar subquery is used to retrieve a single value and is often employed in situations where an expression or a single value is expected. It is commonly used with aggregate functions.

Let take an example;

```
SELECT name ,
(SELECT MAX(price) FROM Paintings) [MaxPrice]
from Paintings
where price = (SELECT MAX(price) FROM Paintings);
```

In this example, we retrieve product names along with the highest unit price from the Products table using a scalar subquery. It shows the output as:

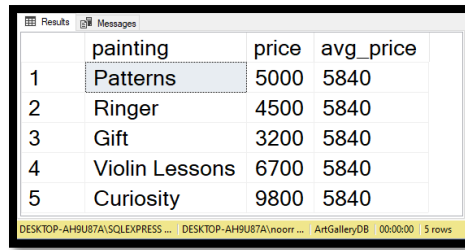


	name	MaxPrice
1	Curiosity	9800

Let's see another example.

```
SELECT name AS painting, price,
(SELECT AVG(price) FROM paintings) AS avg_price
FROM paintings;
```

The above SQL query displays the average price of all paintings besides their original price. The result looks like this:



	painting	price	avg_price
1	Patterns	5000	5840
2	Ringer	4500	5840
3	Gift	3200	5840
4	Violin Lessons	6700	5840
5	Curiosity	9800	5840

The subquery returns the average price of 5840, which is a single value. It gets added to each row of the table. (**Note:** The inner query is independent of the outer query and gives a meaningful result by running on its own.)

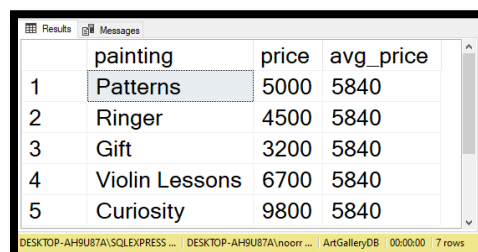
#### b. Single Row Sub Query:

Subqueries that return a single row as an output to their parent query are called single-row subqueries. Single-row subqueries are used in a SQL SELECT statement with HAVING clause, WHERE clause, or a FROM clause and a comparison operator. Single-row subqueries can be used with the WHERE clause in the SELECT statement to filter the results of the outer query.

Let's see an example of it.

```
SELECT * FROM sales_agents
WHERE agency_fee > (SELECT AVG(agency_fee) FROM sales_agents);
```

The above SQL query shows the records of the sales agents whose agency fee is greater than the average of all the fees. The subquery present in the statement calculates the average agency fee, which is 2728. The parent statement then uses the returned value to filter out the information of those sales agents with higher-than-average agency fees. The result looks like the following table:



	painting	price	avg_price
1	Patterns	5000	5840
2	Ringer	4500	5840
3	Gift	3200	5840
4	Violin Lessons	6700	5840
5	Curiosity	9800	5840

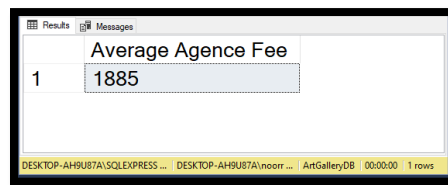
**c. Multiple Row Sub Query:**

Subqueries that return multiple rows as an output to their parent query are called multiple-row subqueries. Multiple row subqueries can be used in a SQL SELECT statement with a HAVING clause, WHERE clause, a FROM clause, and a logical operator (ALL, IN, NOT IN, and ANY).

**Let's explore it using an example.**

```
SELECT AVG(agency_fee) as [Average Agence Fee] FROM sales_agents
WHERE id NOT IN (SELECT id FROM managers);
```

The above SQL Query calculates the average agency fee of all the sales agents who are not managers. The subquery returns the list of IDs of all managers. Then the outer query filters the table to find the records of the sales agents who are not managers and calculates the average agency fee of those agents. It returns a single average value of the agency fee. The output looks like the following:



	Average Agence Fee
1	1885

**d. Multiple Column Subqueries**

Subqueries that return multiple columns as an output to their parent query are called multiple-column subqueries.

**Let's see it through an example.**

```
SELECT id, name, price FROM paintings p1
WHERE EXISTS (
    SELECT 1 FROM paintings p2 WHERE p2.name = p1.name
    AND p2.price = (SELECT MIN(price) FROM paintings));
```

The above SQL Query retrieves the painting with the lowest price. The inner subquery returns the record of the painting with the lowest price. The outer query compares all the records using the IN operator and returns the one with the minimum price.

It gives the following output:



	id	name	price
1	3	Gift	3200

### e. Correlated Subqueries

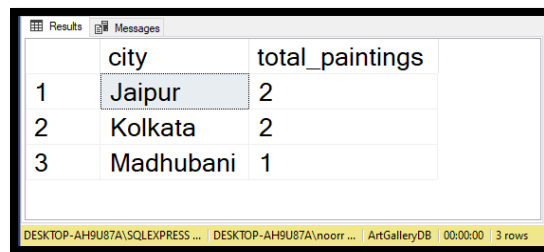
Subqueries that return multiple columns as output depending on the information obtained from the parent query are called correlated subqueries. The interdependence of the inner and outer query makes it complicated to understand. Correlated subqueries can be used in SELECT statements using WHERE and FROM clauses.

***Let's understand it better using an example.***

```
SELECT city, (SELECT count(*) FROM paintings p
WHERE g.id = p.gallery_id) total_paintings
FROM galleries g;
```

The above SQL query calculates the number of paintings found in each gallery. The subquery returns a single value for the total number of paintings in the corresponding query. The outer query returns the details of each painting with the city of the art gallery it is present.

The output of the following is:

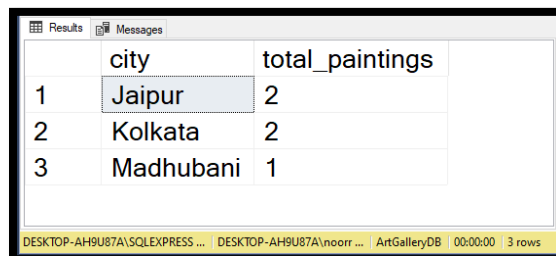


	city	total_paintings
1	Jaipur	2
2	Kolkata	2
3	Madhubani	1

You can get the same result using JOIN in SQL. Let's see how!

```
SELECT g.city, count(p.name) AS total_paintings
FROM galleries g
JOIN paintings p
ON g.id = p.gallery_id
GROUP BY g.city;
```

It gives us the same result as you can see:



	city	total_paintings
1	Jaipur	2
2	Kolkata	2
3	Madhubani	1

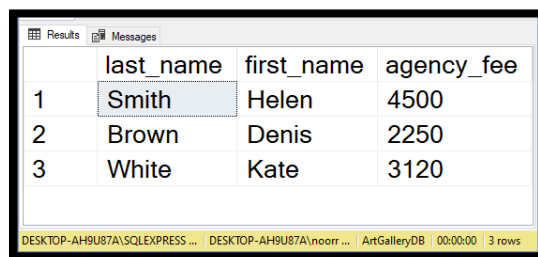
**(Note:** Generally, JOINS performs faster than subqueries, but you can use them if you find it more intuitive.)

Correlated subqueries are also used in SQL statements with WHERE clauses.

**Let's check out an example of it.**

```
SELECT last_name, first_name, agency_fee FROM sales_agents sa1
WHERE sa1.agency_fee >= (SELECT avg(agency_fee)
FROM sales_agents sa2 WHERE sa2.gallery_id = sa1.gallery_id);
```

The above SQL Query shows us the information of those sales agents whose agency fee is greater than or equal to the agency fee of their gallery. The subquery returns the average of the agency fee of the respective sales agent. The outer query returns the information of those sales agents whose agency fee is greater than or equal to the average of the galleries. The result of the above query is as follows:



	last_name	first_name	agency_fee
1	Smith	Helen	4500
2	Brown	Denis	2250
3	White	Kate	3120

#### f. Nested Subqueries

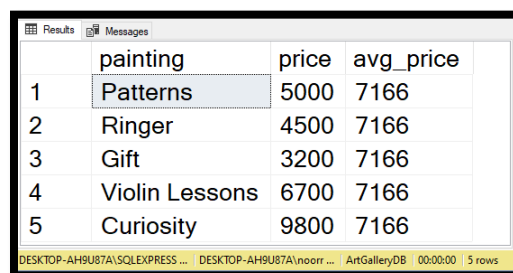
Subqueries that are inside another subquery are called nested subqueries. Subqueries are executed level by level. The innermost is executed first, and then the outer ones.

**Let's see some examples.**

```
SELECT name AS painting, price,
(SELECT AVG(price) FROM paintings WHERE price IN
(SELECT price FROM paintings WHERE price >= 5000)) AS avg_price
FROM paintings;
```

The above SQL query is to display the average price of paintings whose price is greater than 5000 than their original price. The result looks like this:

The output of the following query looks like this:



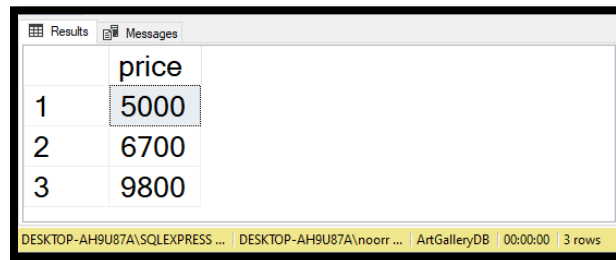
	painting	price	avg_price
1	Patterns	5000	7166
2	Ringer	4500	7166
3	Gift	3200	7166
4	Violin Lessons	6700	7166
5	Curiosity	9800	7166

The above example has three subqueries that are nested subquery, inner subquery, and outer subquery. The code in this example gets departed part by part. Let's break them down one by one to understand the results clearly.

At first, the nested query runs as follows:

```
SELECT price FROM paintings WHERE price >= 5000;
```

This nested query retrieves the price of the paintings whose value is greater than or equal to 5000. The output of the following query is:



	price
1	5000
2	6700
3	9800

DESKTOP-AH9U87A\SQLEXPRESS ... | DESKTOP-AH9U87A\ncorr ... | ArtGalleryDB | 00:00:00 | 3 rows

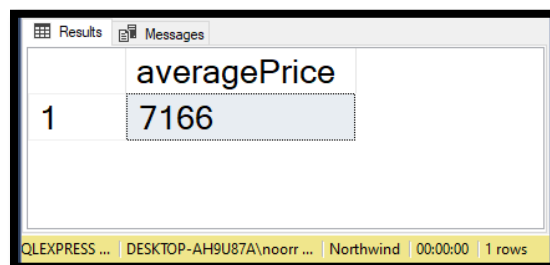
The inner subquery looked like this:

```
SELECT AVG(price) FROM paintings WHERE price IN  
(SELECT price FROM paintings WHERE price >= 5000)
```

After receiving the output from the nested subquery, it internally transforms to:

```
SELECT AVG(price) as averagePrice FROM paintings WHERE price IN(5000, 6700, 9800)
```

The subquery returns the average price of paintings of those returned by the nested subquery (i.e., 5000, 6700, 9800). The result of the subquery is a single average value of 7166.66. It gives the output as:



	averagePrice
1	7166

QLEXPRESS ... | DESKTOP-AH9U87A\ncorr ... | Northwind | 00:00:00 | 1 rows

The outer query is transformed by receiving the result from the subquery and the nested query.

```
SELECT name AS painting, price,  
        (output from the inner subquery)  
        (output from the nested subquery)) AS avg_price  
FROM paintings;
```



The query internally works as follows:

```
SELECT name AS painting, price, (7166.66) AS avg_price  
FROM paintings;
```

The outer query returns the name of paintings whose price is greater than or equal to 5000, with their original and average price beside them.

The following is the final output:

	painting	price	avg_price
1	Patterns	5000	7166.66
2	Ringer	4500	7166.66
3	Gift	3200	7166.66
4	Violin Lessons	6700	7166.66
5	Curiosity	9800	7166.66

#### 4. ACTIVITY TIME BOXING

Activity Name	Activity Time	Total Time
Instruments Allocation + Setting up Lab	10 mints	10 mints
Walk through Theory & Tasks (Lecture)	60 mints	60 mints
Implementation & Practice time	90 mints	80 mints
Evaluation Time	20 mints	20 mints
	Total Duration	180 mints

## 5. EXERCISE:

### PERFORMANCE TASKS:

#### 1. All Example Tasks.

### LAB FILE TASKS:

1. Find the company's name that placed order 10290. (Tables : Customers & Orders)
2. Create a report that shows the product name and supplier id for all products supplied by Exotic Liquids, Tokyo Traders, Ma Maison and Lyngbysild. (Tables : Products & Suppliers)
3. Create a report that shows all products by name that are in the Confections. (Tables : Products & Categories)
4. Retrieve name of product having maximum, minimum and average unit price along with their prices. (Tables : Products)
5. Create a report that shows all 5 companies by name that sell products in the Seafood category. (Tables: Suppliers, Products & Categories)
6. Write query using a "sub query" to display which Customers were served by which Employee. (Tables: Customers & Employees)
7. Write query using a "sub query" to list of all the stores that have discount records (use pubs).
8. Write query using a "sub query" to list all the authors available in Barnum's store (use pubs).
9. Write query using a "sub query" to give the customer id and amount spent of the customer who spent the most.
10. Write a query using a "sub query" to find customers who have placed orders. (Tables: Customers & Orders)