Module 10 – Subqueries

Module Objectives

List the different types of subqueries

Write single-row and multiple-row subqueries

Discuss the various issues that arise using subqueries

Suppose you want to write a query to find out which products cost more than Tofu. To solve this problem, you need two queries: one query to find out the unit price for Tofu and a second query to find out which products cost more than that amount.

You can solve this problem by combining the two queries, placing one query inside the other query.

The inner query or the subquery returns a value that is used by the outer query or the main query. Using a subquery is equivalent to performing two sequential queries and using the results of the first query as the search value in the second query.

The syntax for a subquery is as follows:

```
SELECT select_list

FROM table_list

WHERE expression comparison_operator

                    (SELECT select_list

                    FROM table_list);
```

The subquery (inner query) executes once before the main query. The result of the subquery is then used by the main query (outer query) to get the final results.

You can place a subquery in a FROM clause, a WHERE clause, or a HAVING clause.

It is important to note that the comparison operators fall into two different classes: single-row operators (>, <, < >, > =, < =, =) and multiple-row operators (IN, ANY, ALL).

You will often hear a subquery referred to as a nested SELECT, sub-SELECT, or an inner SELECT statement.

Guidelines for Subqueries

Enclose subqueries in parentheses.

Place subqueries on the right side of the comparison operator.

Do not add an ORDER BY clause to a subquery.

Use single-row operators with single-row subqueries.

Use multiple-row operators with multiple-row subqueries.

List the different types of subqueries

Single-row subquery

Queries that return only one row from the inner SELECT statement.

```
SELECT productname
FROM your_schema.products
WHERE unitprice >
        (SELECT unitprice
        FROM your_schema.products
        WHERE productname = 'Tofu');
```

```
SELECT productname
FROM your_schema.products
WHERE unitprice >
        (SELECT unitprice
        FROM your_schema.products
        WHERE productname = 'Tofu')
AND unitsinstock >
        (SELECT unitsinstock
        FROM your_schema.products
        WHERE productname = 'Pavlova');
```

A SELECT statement can be considered as a query block. This example displays the products whose unit price is higher than the unit price of Tofu and the products that have more units in stock than Pavlova.

This example consists of three query blocks: the outer query and two inner queries. The inner query blocks are executed first, producing the query results: 23.25 and 29, respectively. The outer query block is then processed and uses the values returned by the inner queries to complete its search conditions.

Since both inner queries returns single values, this SQL statement is considered a single-row subquery.

An important fact to know is that the inner queries do not have to be based upon the same table as each other or the outer query.

```
SELECT productname
FROM your_schema.products
WHERE unitprice >
        (SELECT avg(unitprice)
        FROM your_schema.products
        WHERE categoryid = 1);
```

You can use a group function in a subquery. In this example, we are displaying the product name of the products whose unit price is higher than the average unit price of all the products whose category id is 1. By using the group function in a subquery, we are able to return a single-row result, which is then used in the comparison of the outer query WHERE clause.

Just like the WHERE clause, we can also use subqueries in the HAVING clause.

```
SELECT categoryid, avg(unitprice)
FROM your_schema.products
GROUP BY categoryid
HAVING avg(unitprice)  >
        (SELECT avg(unitprice)
        FROM your_schema.products
        WHERE categoryid = 2);
```

In this example, we do just that. Here we select all the groups of data that match the inner queries results.

Multiple-row subquery

Multiple-row subqueries, are queries that return more than one row from the inner SELECT statement. You must use a multiple-row operator with a multiple-row subquery. The multiple-row operator expects one or more values returned.

```
SELECT productname
FROM your_schema.products
WHERE unitprice > ANY
        (SELECT unitprice
        FROM your_schema.products
        WHERE productname = 'Tofu');
```

IN – Equal to any member in the list

ANY – Compare value to ***each*** value returned by the subquery

ALL – Compare value to ***every*** value returned by the subquery

< ANY means less than the maximum

> ANY means more than the minimum

= ANY is equivalent to IN

< ALL means less than the minimum

> ALL means more than the maximum

NOT can be used with IN, ANY, and ALL operators.

Multiple-column subquery

Queries that return more than one column from the inner SELECT statement.

So far you have written single-row subqueries and multiple-row subqueries where only one column was compared in the WHERE clause or HAVING clause of the SELECT statement. If you want to compare two or more columns, you must write a compound WHERE clause using logical operators. Multiple-column subqueries enable you to combine duplicate WHERE conditions into a single WHERE clause.

```
SELECT orderid, productid, quantity
FROM your_schema.orderdetails
WHERE (productid, quantity) IN
        (SELECT productid, quantity
        FROM your_schema.orderdetails
        WHERE orderid = 10800)
and orderid <> 10800;
```

This is an example of a multiple-column subquery because the subquery returns more than one column. It compares the values in the PRODUCTID column and the QUANTITY column of each candidate row in the ORDERDETAILS table to the values in the PRODUCTID column and the QUANTITY column for products in order 10800.

When this SQL statement is executed, it compares the values in both the PRODUCTID and QUANTITY columns and returns those orders where the product number and quantity for that product match BOTH the productnumber and the quantity for a product in order 10800. This shows us which orders were placed for the same quantity of this product and was place on order 10800.

Discuss the various issues that arise using subqueries

One common error with subqueries is more than one row returned for a single-row subquery. If the inner query contains a GROUP BY clause, and the comparison operator is a single-row operator and error stating “Single-row subquery returns more than one row” will be returning.

In order to use an inner query with multiple values, you must use a multiple-row operator, such as IN, ANY, or ALL.

Another common error is when no rows are returned from the inner query. If the inner query returns to rows, then it is not possible for the outer query to return any rows either. The only way would be for the outer query to have multiple test conditions in the WHERE clause.

If the inner query returns multiple values, and one of those values is null, the results of the outer query will be null. The reason is that all conditions that compare a null value result in a null. Therefore, whenever null values are likely to be part of the results set of a subquery, do not use the NOT IN operator. The NOT IN operator is equivalent to != ALL.