Outer Join Queries and Set Operators

Introduction

After the previous learning unit, you should be pretty familiar with the join process and the SQL syntax of the various join methods. We will now take the join process further by showing how tables can be joined using an OUTER JOIN to retrieve matching AND non-matching rows.

Outer Join – Traditional Method

Recall earlier examples of showing which orders were placed by which customers:
FROM Customers C, Orders O WHERE C.Customer# = O.Customer#. This satisfies the request. If a customer number is not in the Orders table, then logically they do not have any orders. But if the request was to list all customers, and any order information if they have an order, then we would need an OUTER JOIN to get customers with orders (matches) and customers without orders (non-matches).

This OUTER JOIN could be done with the traditional method: FROM Customers C, Orders O WHERE C.Customer# = O.Customer#(+). Notice the use of the outer join operator (+). This tells Oracle which table will be short data: in our example, it's the Orders table that is "deficient." The + operator can only be used on one table only, not both. This can be a hindrance to using it depending on the query request. Using the JOIN keyword is more flexible.

Outer Join – JOIN Method

By using the JOIN keyword, you can specify which table the join applies to, in other words which table the non-matches should be kept from. Back to our example above, we want all customers, and order information if they have an order. The above query could be written as: FROM Customers C LEFT OUTER JOIN Orders O USING (Customer#). The LEFT keyword instructs Oracle to keep all rows (matching and non-matching) from the Customers table (first table listed is LEFT, second table listed is RIGHT). Any matching rows in the Orders table are also kept.

This query would give the same result: FROM Orders O RIGHT OUTER JOIN Customers C USING (Customer#). Notice that the Customers table is now the RIGHT table (listed second).

Say for some reason we had the situation where a customer has been deleted, but an order still exists. In theory this should not happen due to foreign key restraints, but we'll use it as an example. If we wanted a query to show all orders, and customer information if the order had one, then we reverse our query to be: FROM Orders O LEFT OUTER JOIN Customers C USING (Customer#). Again, matches would be shown, and any non-matches from the Orders table would also be shown.

If we want a query that shows all customers, the orders that belong to that customer, and the orders that have no customers, we would need to perform a FULL JOIN. A FULL JOIN instructs Oracle to join the tables and show rows that match (customers with orders), rows that have no matches on the left table (customers with no orders), and rows that have no matches on the right table (orders with no customers). The query would be: FROM Customers C FULL JOIN Orders O USING (Customer#).

One more example should drive the outer join concept home. Assume a book can be added to the database without a publisher. Assume also that a publisher can be added without any books for that publisher. You need to show all books, the publisher if it has one, and all publishers, even if they don't have a book in inventory. As you may have guessed, a FULL JOIN is required. The query would be: FROM Books B FULL JOIN Publisher P USING (PubID).

Keep in mind the WHERE clause can be used to further restrict the rows that get returned after the join takes place. Also as a side note, the keyword OUTER is optional, but is still used frequently for clarity.

NOTE: the JOIN ON syntax can also be used with Outer Joins. Example: FROM Books B FULL JOIN Publisher P ON B.PubID = P.PubId.

Set Operators

Generally, one SELECT statement is sufficient to satisfy a query request. There are times however that two or more queries need to be run to produce the desired results. Oracle provides Set Operators that allow two or more SELECT statements to be executed consecutively, then their results combined into one result set.

UNION, UNION ALL, INTERSECT, and MINUS are Set Operators and all perform a different task. These Set Operators originate from their mathematical source. UNION keeps the results from two SELECT statements (sets), but removes any rows that are duplicated. If duplicates need to be retained, use UNION ALL. INTERSECT returns only those rows that are identical from both SELECT statements. MINUS removes rows returned in the second SELECT statement from rows returned in the first SELECT statement, if they are identical. To help you remember the Set Operators, think of UNION as "Combine," INTERSECT as "Common," and MINUS as "Remove." If desired, an ORDER BY clause can be added at the very end to sort the final results.

Let's use set notation to demonstrate each of the set operators. Assume the partial results of the first SELECT statement are {1,2,3,4,5,6,7} and the partial results of the second SELECT statement are {5,6,7,8,9}. The final answer for each operation would be as follows:

Set notation expression	Final results
{1,2,3,4,5,6,7}UNION{5,6,7,8,9}	{1,2,3,4,5,6,7,8,9}
{1,2,3,4,5,6,7}UNION ALL{5,6,7,8,9}	{1,2,3,4,5,5,6,6,7,7,8,9}
{1,2,3,4,5,6,7}INTERSECT{5,6,7,8,9}	{5,6,7}
{1,2,3,4,5,6,7}MINUS{5,6,7,8,9}	{1,2,3,4}

Generally the SELECT statements involved in set operations have multiple columns being retrieved. When this is the case, keep these points in mind:

- When comparing rows for equality, all columns are used in the comparison
- Each query result set must be union-compatible: same number of columns listed in the same order on each SELECT statement, although the column names can be different
- Corresponding columns should have the same data type for the most accurate comparisons