Selecting Data From More Than One Table

Section Recap

In this section, you learned how to read data from more than one table in the same query, which is something you will have to do very frequently.

Queries that read data from more than one source are usually called "join queries".

The clause that allows you to read data from more than one table is the JOIN clause, which is actually part of the FROM clause.

In this clause, you have to mention the name of the table you are joining and a condition that defines how the tables are related, as in this example, where the employee and department tables are joined:

```
SELECT e.name, salary, d.name AS dept_name
FROM employee e
  JOIN department d
    ON e.department_id = d.id;
```

You also learned that when you are using or referencing a column name that exists in more than one of the tables you are querying, you need to qualify the column with the table name, otherwise Oracle will complain because it doesn't know which the one you are referring to is. You can qualify it with the table name, but it is easier to assign an alias to the table, and then use that alias to qualify the columns, as in the example above

The first type of JOIN we saw is the INNER JOIN, which is the most common type of JOIN.

An INNER JOIN returns rows that have a match between the 2 tables, so, in this example, a row will be returned if in the employee table there is a row with a department id equal to the ID column of some row in the department table.

The resulting row from the JOIN will include the columns from both tables, so, if instead of listing specific columns in the SELECT list you include the asterisk (*), the results will have all of the columns from the employee table and all of the columns from the department table, of the rows that were matched.

If there is a row in the employee table, with a department_id that doesn't exist in the department

table, or vice versa, it will not be returned, because an INNER JOIN returns only matching rows.

When you write INNER JOINs, you can include the INNER keyword in your statement, but it is optional because that is the default type of JOIN.

I showed you also, that if the columns you want to use for the JOIN condition have the same name in both tables, you can use an abbreviated form, with the USING clause, like this:

```
SELECT *
FROM employees e
  JOIN jobs j
    USING (job_id);
```

You also learned that this syntax that includes the JOIN clause was adopted by Oracle in Oracle version 9, where they adhered to the then-new ANSI standard.

Prior to that, you would put all of the tables in the FROM clause without using the JOIN keyword, separated by commas, and the join conditions would be defined in the WHERE clause, like in this example:

```
SELECT *
FROM employees e, jobs j
WHERE e.job_id = j.job_id;
```

We saw also that JOINS occur between 2 tables only, so, in a query like this one:

```
SELECT *
FROM locations l
  JOIN countries c
    ON l.country_id = c.country_id
  JOIN regions r
    ON c.region_id = r.region_id;
```

Locations is joined to countries first and then, the second JOIN clause joins the results of the first join, to regions, so each JOIN works on 2 sources exactly. The database may decide to do the joins in a different order if it believes that it is more efficient doing it that way, but it will still JOIN only 2 sources at a time.

You learned also that besides inner joins, there are other types of joins, which include CROSS joins and OUTER joins.

A Cross join returns what is called the "Cartesian product" of the 2 tables being joined. That is, every row from the first table is combined with every row from the second table.

Using the old syntax for joins, you create a Cartesian product by including the 2 or more tables in the FROM clause, separated by commas. In this case, If you don't add any JOIN condition in the WHERE clause, your query returns a Cartesian product.

In the new ANSI syntax, you create a Cartesian product by explicitly using a CROSS join this way:

```
SELECT *
FROM employees
  CROSS JOIN departments;
```

OUTER joins are the type of JOINs that returns the rows that match the JOIN condition, like inner joins, but include all of the rows from one of the tables being joined, even if there is no match in the other table. This type of JOIN will return NULLs for every column from the table in which a match was not found.

When you use a LEFT OUTER JOIN, the rows that are returned even if no matching rows exist in the other table, are the rows from the table that appears at the left side of the JOIN clause, and if you use a RIGHT OUTER JOIN, then the rows that are returned even if there is no match, are those from the table written to the RIGHT-hand side of the JOIN clause.

Left and Right outer joins work exactly in the same way, and the only difference is which of the tables being joined is the one whose rows are to be preserved.

As you might remember, the OUTER keyword is optional, so you can simply write LEFT JOIN and RIGHT JOIN. That is what I do.

Here is an example of a RIGHT OUTER JOIN:

```
SELECT *
FROM employee e
  RIGHT JOIN department d
    ON e.DEPARTMENT_ID = d.id;
```

Using the old Oracle syntax for joins, you create an outer join by using the OUTER JOIN operator, which is represented by a plus sign inside parentheses.

If you want a LEFT outer join, in your join conditions you add the outer join operator next to the columns of the table that appears TO THE RIGHT in the from clause, and vice versa.

So, this query is equivalent to the previous example:

```
SELECT *
FROM employee e, department d
WHERE e.department_id (+) = d.id;
```

See?

This is a right JOIN because the outer join operator appears next to the column from the EMPLOYEE table, which is the table that is on the left side in the FROM clause.

And besides LEFT and RIGHT outer joins, there are also FULL OUTER JOINs, which return the rows that have a match in both tables, plus the ones that exist only in the table to the left plus the ones that exist only in the table to the right.

And this kind of outer join was not possible with the old syntax. At least, not directly.

What else did you learn in this section?
Do you remember?

Well, we saw that besides join conditions that use equality comparisons, you could write conditions using GREATER THAN, SMALLER THAN, BETWEEN and other types of comparisons.

A join with these types of conditions is usually called a NON-EQUI JOIN, whereas a JOIN whose condition uses the equal sign is called an EQUI JOIN.

We also talked about SELF joins, but they are actually NOT a different type of join. The only thing that makes them special is that you join a table to itself, but other than that, there is nothing new about them. The classical example is when you join an employees table to itself, to get the details of employees and their managers.

And that was it. You now know everything you need to know about JOINS.

Remember that if you feel the need to watch any of the lectures again, you are of course free to do it, and if you need my help with anything covered in the course, please don't hesitate to let me know.

See you soon!