Operators

Section Recap

In this section, we talked about operators, and the first group we covered was the "comparison operators".

Some people call these operators as "conditions", and I think even the official documentation uses that term, but I usually think of a condition as the expressions being compared plus the comparison operator that makes the actual comparison.

In this group we find operators like equal to, not equal to, greater than, etcetera, but also have a couple of operators that many people forget about, and these are ANY and ALL, which are used in conjunction with the other operators in this group, and complement their functionality to create what is called "group comparison conditions".

Both, ANY and ALL must be preceded by some other comparison operator and followed by a list of one or more values or expressions, which includes the possibility of using a subquery, which will be covered in a future lesson.

We also covered these SQL operators:

- LIKE, which is used for pattern matching conditions, and allows you to define your patterns using the percentage sign, which matches 0 or more characters, and the underscore, which matches exactly one character.
- IN, which is used for membership checking conditions, so that you can check if a value or expression is one of the values in a fixed list or the list of values returned by a subquery.
- And BETWEEN, which simplifies creating range conditions.

And finally, we talked about logical operators, which are used to connect two conditions to create a bigger one.

The simple rule for AND is that it returns true only if both of the sub-conditions it connects are true.

And OR returns true if any of the sub-conditions it connects are true.

NOT, on the other hand, is used to negate or invert the logical value of a condition.

You learned that these logical operators don't have the same precedence, so if you have several conditions with NOTs, ANDs, and ORs and there are no parentheses, they are not evaluated in order from left to right. Instead, the order of evaluation is determined by the precedence of the operators, which is: NOT has the highest precedence, then it comes AND, and lastly OR, so, if you need to define a specific order of evaluation of the conditions, you MUST use parentheses.

In this section, we also explored a feature that is not part of the SQL language itself, but is present in some client tools, like SQL Developer and SQL*Plus, which allows you to include variables in your statements, which are later substituted with values entered by the user when the query is executed, as a means for writing re-usable statements. This substitution is performed by the client tool, before sending the command to the database.

To include a variable in a statement, you have to use the ampersand (&) followed by the name of the variable. And if you want the variable to get stored, so that SQL Developer doesn't ask for a value if the same variable appears again, you just use two ampersands (&&) followed by the variable name.

If you want to erase the value of a previously stored variable, you have to use the UNDEFINE command, which is a separate command from the SQL statement where you defined the variable.

But remember, UNDEFINE is not an SQL command.

Here is an example in which the different operators covered in this section are used, and then a call to the UNDEFINED command to erase the value saved for the "sal" variable:

```
SELECT *
FROM employee
WHERE department_id IN (3,4)
  AND (
    salary BETWEEN 3000 AND 5000
    OR salary = ANY (1000,5000)
  )
  AND NOT Name LIKE '%N'
  AND birthdate BETWEEN DATE '1970-01-01' AND DATE '1990-12-31'
  AND
  (
    salary = &&sal
    OR salary = &sal/2
  );
```

```
UNDEFINE sal;
```

Ok, and that was it.

I'll see you in the next section.