Filtering and Sorting Results

Section Recap

In this section, you learned that the WHERE clause allows you to filter the results of your queries, and it is composed of the WHERE keyword and a condition that needs to evaluate to true for the rows that are going to be returned by the query.

This condition can actually be composed of several sub-conditions, but at the end of the day, all the sub-conditions are reduced to one single condition, which determines if the row has to be returned.

The order in which the sub-conditions are evaluated is determined by Oracle based on the order in which they appear and the operators' precedence, which is a topic that will be covered later in the course, but if you want to force a specific order of evaluation of the conditions, you can use parentheses to do it. The conditions that are nested at the deepest levels of parentheses are evaluated first.

The order of evaluation is very important because as you saw in one example, a simple pair of parentheses can make the query produce different results.

You also learned that even though you can write dates as if they were strings, it is better to use date literals, because they work correctly on any system, regardless of the local date format settings. Date literals are written using the word DATE, and a string that represents the desired date in the format 'YYYY-MM-DD'.

Here is an example of how you can filter rows comparing dates correctly, using date literals:

```
SELECT *
FROM products
WHERE expiration_date < DATE '2016-06-01';
```

You can also create dates from strings by using a conversion function, but that is something that will be covered in a future lesson.

Another method you have to filter the results of queries is by using the DISTINCT keyword, which filters or removes duplicate rows, but remember: for a row to be removed, the whole row must be equal to some other row.

Another very important topic that was covered in this section is the ORDER BY clause, which allows you to order the results of your queries.

By default Oracle orders in ascending order, but if you want to order in descending order, you just have to add the DESC keyword after the name of the column you are ordering by.

Also, you learned that you can order by more than one column, for which you just have to specify the desired columns in the order by clause, separated by commas, and you can use a different type of ordering for each column.

It is VERY important that you remember that if you need or want the results of your query in a certain order, you MUST add an order by clause, even if you appear to get the results already in that order without adding this clause. If you fail to do that, the order is not guaranteed, and your query could start producing results in a different order at any moment.

Instead of the name of the columns, you can also use column positions in the order by clause, so if you order by 1, for example, it means that you want to order by the first column returned by the query. This is very handy, but can sometimes make SQL code less readable.

The ORDER BY clause is very flexible, and you saw that you can actually use very complex expressions there, to do things like conditional ordering, and any other thing your creativity wants you to do.

Here is a small example:

```
SELECT id, name, department_id, salary
FROM employee
ORDER BY
        CASE department_id
                WHEN 1
                        THEN salary
                ELSE id
        END;
```

In this case the ORDER BY clause includes a CASE expression, which is something you have not learned yet, so, by now just remember that you use complex expressions in the ORDER BY clause, without trying to completely understand the above example.

And finally, you learned about NULLs, so now you know that null is the equivalent to unknown, and thus, you cannot compare it to actual values using normal comparison operators like equal

to, or greater than, because those comparisons result in an unknown logical value. The correct way to compare NULLs or to write conditions to check for NULLs is the use of the IS NULL or IS NOT NULL operators.

Keep in mind that a condition can evaluate to 3 different logical values: TRUE, FALSE, and UNKNOWN, and the only rows that will be returned are those for which the condition evaluates to TRUE.

You should also think about nulls when ordering the results of your queries. By default, when Oracle orders in ascending order, it leaves nulls at the end, and when it orders in descending order it puts nulls at the beginning, before the non-null values, but if that is not what you want, you can add the optional NULLS FIRST or NULLS LAST clauses in the ORDER BY clause of your query.

There are some functions that simplify our work with NULLs, which includes:

- NVL, which lets you define a substitute for null values.
- COALESCE, which returns the first non-null value from the list of values or expressions passed as parameters.
- NVL2, which allows you to return different values, depending on whether the first expression passed is null or not.
- NULLIF, which returns NULL if the two expressions passed as parameters are equal.
- And LNNVL, which is a little different because it receives a condition as a parameter and is also used as a condition. You can think of it as an "IS NOT TRUE" condition.

NVL, NVL2, and LNNVL are Oracle-specific, and COALESCE and NULLIF are part of the SQL standard.

Here is an example where these functions are used:

```
SELECT id,
   name,
   phone,
   NVL(phone, '* No Phone *'),
   COALESCE (salary, bonus, 0),
   NVL2(phone, 'Has Phone', 'Does Not Have Phone'),
   NULLIF(bonus, 100)
FROM employee
WHERE LNNVL(phone = '1.123.456.7890');
```


The above condition can be understood as “WHERE it is not true that the phone is equal to '1.123.456.7890'”, so, it returns rows in which the phone is different from that number, but also rows in which the phone is null.
See you in the next section!

