Transposing Rows into Columns and Columns into Rows

Section Recap

This was ANOTHER very important section because transposing columns into rows or rows into columns is considered an advanced topic by many people, so learning how to do it correctly using the right features can really set you apart.

We saw that we can transpose rows into columns by using CASE expressions, which is the traditional way in which this was done in older versions of the database (in even older versions, you would have to use DECODE instead).

For example, this query returns the count of employees that where hired in 2014 or 2015, in different columns:

```
SELECT department_id,
  COUNT(
    CASE to_char (hire_date, 'yyyy')
      WHEN '2014' THEN 1
    END
  ) AS "2014",
  COUNT(
    CASE to_char(hire_date, 'yyyy')
      WHEN '2015' THEN 1
    END
  ) AS "2015"
FROM employee
GROUP BY department_id
ORDER BY department_id;
```

As you see, we need a case expression for each column, in this case, 2014 and 2015. We used COUNT in this example, but depending on the requirement you could do another type of operation.

But we saw then, that this kind of operations can be done much more easily by using the PIVOT feature, which was introduced in Oracle 11g.

The equivalent query using PIVOT would be like this:

```
SELECT *
FROM
(
   SELECT department_id,
      TO_CHAR(hire_date, 'YYYY') AS year
   FROM employee
)
PIVOT (
   COUNT(*) AS employees_hired
   FOR YEAR IN (2014 AS year2014, 2015 AS year2015)
)
ORDER BY department_id;
```

The key points to remember here are:

You need an aggregate function. In this case, I used COUNT, but depending on what you want to do, it could be MAX, MIN, AVG, etc…

Sometimes you don't want to aggregate anything, because you are working with a single row for each value. In those cases, you can use some function like MAX or MIN just to comply with the requirement of using an aggregate function, but because there is only one row for each value, the results would be correct.

Then after the FOR keyword, you need a column, whose rows you want to convert into columns.

In this case, I used 'YEAR', which is actually not a column from the employee table. It is not possible to use functions there, and that is why I needed to use the TO_CHAR function to extract the year from the date in a subquery, which as you might remember, is of the type of subqueries that are called INLINE VIEWS.

Then, inside parentheses, you define the columns you want. In the above example, I used values from the YEAR column, so, when a row has 2014 in the YEAR column, it will count it and will put the results in a new column, and the same will be done for 2015.

A small caveat is that those values must be pre-defined at design time. Or in other words, you can't use a subquery here to provide the values at runtime.

Also, remember that you can use aliases for the aggregate function and for the columns in the IN clause.

And you can have more than one aggregate function, like this:

```
SELECT *
FROM
  (
    SELECT department_id,
      TO_CHAR(hire_date, 'YYYY') AS YEAR,
      salary
    FROM employee
  )
PIVOT (
  COUNT(*) as employees_hired,
  SUM(salary)
  FOR YEAR IN (2014 as year2014, 2015 as year2015)
)
ORDER BY department_id;
```

And you can also have combinations of values here, like in this example:

```
SELECT *
FROM
  (
  SELECT department_id,
    TO_CHAR(hire_date, 'YYYY') AS hire_year,
    TO_CHAR(birthdate, 'YYYY') AS birth_year
  FROM employee
  )
PIVOT (
  COUNT(*)
  FOR (birth_year,hire_year) IN ((1997,2014), (1995,2015))
)
ORDER BY department_id;
```

Here, the first column returns the count for employees who were born in 1997 AND hired in 2014, and the second one for 1995 and 2015 respectively.

And we also looked at the UNPIVOT feature, which allows us to transpose columns into rows very easily, like in this example:

```
SELECT *
FROM department
UNPIVOT (
  data_value
  FOR category IN (monthly_budget, last_employee_id)
);
```


Here monthly budget and last employee id are different columns of the department table, but with UNPIVOT, we are putting the values from these columns into a single column but in different rows. The name of that column was defined as "data_value", but it can be anything.

And this also creates another column in which we have the description of where the values are coming from, and the name of such a column was defined in this example as "category", but it can also be anything.

And we can add the optional INCLUDE NULLS after the UNPIVOT keyword, to tell Oracle to generate rows even when the data in the original column is NULL.

We also saw that you can define more than one column, and that you can have combinations of columns in the IN clause as well, as in this example:

```
WITH aggregates AS --This just generates testing data
(
  SELECT department_id,
    COUNT(*) AS total_employees,
    SUM(salary) AS total_salaries,
    COUNT(
      CASE
        WHEN salary < 3000 THEN 1
      END
    ) AS employees_3000,
    SUM(
      CASE
        WHEN salary < 3000 THEN salary
      END
    ) AS salaries_3000
  FROM employee
  GROUP BY department_id
) --End of testing data. The real UNPIVOT example starts in the next line.
SELECT *
FROM aggregates
UNPIVOT (
  (employees, salaries)
  FOR type IN (
    (total_employees,total_salaries) as 'ALL EMPLOYEES',
    (employees_3000,salaries_3000) as 'EMPLOYEES WHO EARN LESS THAN 3000'
  )
)
ORDER BY department_id, type;
```

And finally, we saw that we can include PIVOT and UNPIVOT in the same query, as in this example:

```
WITH aggregates AS --This just generates testing data
(
  SELECT department_id,
    COUNT(*) AS total_employees,
    SUM(salary) AS total_salaries,
    COUNT(
      CASE
        WHEN salary < 3000 THEN 1
      END
    ) AS employees_3000,
    SUM(
      CASE
        WHEN salary < 3000 THEN salary
      END
    ) AS salaries_3000
  FROM employee
  GROUP BY department_id
) --End of testing data. The real UNPIVOT example starts in the next line.
SELECT *
FROM aggregates
UNPIVOT (
  (employees, salaries)
  FOR type IN (
    (total_employees,total_salaries) AS 'ALL EMPLOYEES'
    , (employees_3000,salaries_3000) AS 'EMPLOYEES WHO EARN LESS THAN 3000'
  )
)
PIVOT (
  MAX(salaries) AS sal,
  MAX(employees) AS emp
  FOR department_id IN (1 AS dept1,2 AS dept2,3 AS dept3,4 AS dept4)
);
```

In this case, Oracle will perform the UNPIVOT first, because that is what appears first in the query, and then will apply PIVOT to the results of the UNPIVOT operation, which is awesome!

These features are very powerful, and there are many people who don't know them, so, if you learn them well, you will have a great chance of standing out, so, if you feel they are complex, don't hesitate to watch the lessons again, as many times as needed, until you are able to complete the practice challenges confidently.

See you in the next section!