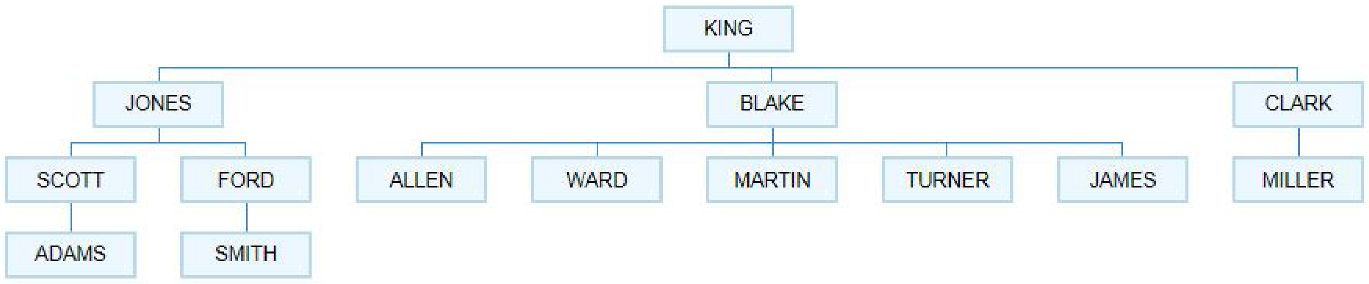
Hierarchical Queries

Section Recap

The first thing you learned in this section is that hierarchical data is the kind of data in which one row can be related to another row by means of a parent-child relationship, for example, in an employees table, where employees are related to their managers.

This kind of data can be represented as a tree, in which the children of a node appear below that node, like in this image, which corresponds to the emp table (which doesn't exist in your schema, and was used only for my demonstrations):



Here, KING is the root of the tree, so he is the first level in the hierarchy. Then, JONES, BLAKE, and CLARK are the children of KING, and they are at the second level, ... and so on.

A node that doesn't have any children, is called a "leaf".

A long time ago, you would usually need to use SELF JOINs to solve hierarchical data problems, but now we can use the HIERARCHICAL clause, which was designed specifically for this kind of problem.

This clause is composed of 2 main parts: The "START WITH" subclause and the "CONNECT BY" subclause.

This was our first example, from the emp table:

```
SELECT
   empno,
   ename,
   job,
   PRIOR ename AS manager
FROM emp
START WITH mgr IS NULL
   CONNECT BY mgr = PRIOR empno
ORDER BY empno;
```

START WITH is used to define where the hierarchy starts, so in this case, it will start with the rows that have NULL in the manager column, because that is the condition I defined.

If I don't include this sub-clause, then Oracle will build one hierarchy tree starting at each of the rows in the table, so in this case, I would have 14 trees, which means that I would get a lot more rows in the result set.

And CONNECT BY allows me to define how the rows are related to each other. In this case, the condition says that what a row has in the manager column must be equal to the empno column of the row in the previous level.

So, the row of ADAMS has the employee id of SCOTT in its manager column, as you can infer from the tree image shown before.

And how do I have access to data from the previous level?

That's done is by means of the PRIOR operator, so in the above example, PRIOR empno is the empno of the previous level, and PRIOR ename is the ename of the previous level.

You also learned that if your data is incorrect because there are cycles in the hierarchy, which happens when there is a row that is a child and a parent of some other row, at the same time, a hierarchical query would return an error, but you can make it not throw an error by adding the optional NOCYCLE parameter in the CONNECT BY clause.

We also saw that there are some pseudo columns that are available in hierarchical queries.

One of them is "level", which tells you at which level of the hierarchy the row is, and the other one is CONNECT_BY_ISLEAF, which tells you whether the row is a leaf or not.

There is also a CONNECT_BY_ISCYCLE pseudo column which is only valid when you add the NOCYCLE parameter, and tells you if the row has a child which is also its ancestor.

What other features are available?

There is a CONNECT BY ROOT operator that is pretty similar to the PRIOR operator, but instead of providing access to the previous level, it gives us access to the root of the hierarchy tree to which the current row belongs.

Another very useful thing we have available is the SYS_CONNECT_BY_PATH function, which returns the path from the root of the tree to the current row.

This function takes 2 parameters: The first one is the column you want to use to build the path, and the second one is the separator you want to use between nodes in the path. This function is usually VERY useful when debugging hierarchical queries.

Here the previous example modified to include some of the additional pseudo-columns, operators and functions we covered:

```
SELECT
   empno,
   ename,
   job,
   PRIOR ename AS manager,
   level,
   connect_by_isleaf,
   connect_by_root ename,
   sys_connect_by_path(ename,'->')
FROM emp
START WITH mgr IS NULL
   CONNECT BY mgr = PRIOR empno
ORDER BY empno;
```

Near the end of the section you learned that the order in which things happen in a hierarchical query is this:

· If there are joins, they are done first.
· Then the hierarchy is built according to the START WITH and CONNECT BY clauses.
· Then the filtering conditions in the WHERE clause are evaluated
· And finally, the order by clause is applied, if it is present.

The fact that the WHERE conditions are evaluated AFTER building the hierarchy is particularly important, and that's what allows us to use hierarchical pseudo columns in the WHERE clause filtering conditions.

You also learned that you can traverse the hierarchy from the leaves upwards to the root, by switching the position of the PRIOR operator in the CONNECT BY condition, like in this example:

```
SELECT
  empno,
  ename,
  sys_connect_by_path(ename, ' / ') AS path,
  level
FROM emp
START WITH ename = 'SMITH'
  CONNECT BY empno = prior mgr;
```

In this case, instead of applying the PRIOR operator to the empno, I'm applying it to the manager, so I'm essentially switching the order of the levels, and thus, traversing the tree upwards to the root.

Can you write a similar query with your employees table?

Prove it!

And at the very end of the section, we saw that we can add the SIBLINGS keyword to the ORDER BY clause, if we want to apply an order to nodes at the same level, without breaking the hierarchical order of the data, in which the children of a row always come after their parent, like in this example:

```
SELECT
  empno,
  ename,
  sys_connect_by_path(ename, ' / ') AS path,
  level
FROM emp
CONNECT BY prior empno = mgr
  START WITH mgr is null
ORDER SIBLINGS BY ename;
```

Here, the children of the same parent, which are at the same level, or in other words, SIBLINGS,

will be ordered by the ename column, but the hierarchical order will be preserved.
And that's it!!

You now understand and know how to work with hierarchical data, and this knowledge can really set you apart!

Awesome!

I will see you in the next section!