Tree-structured—hierarchical—data is all around us. In addition to a real tree you might have in your back yard or in the nearby park that has a trunk and branches, you may attend a basketball tournament where the bracket is a time-based version of a tree structure, with only one team ending up at the root at the end of the tournament. At your job, the management structure is, by nature, a tree structure with the boss or owner at the top, and a couple or many levels of management before reaching the hourly workers at the bottom of the tree.

Entire database management systems are built around hierarchical data. One of IBM Corporation’s most popular products, Information Management System (IMS), arrived in the late 1960s and is optimized for hierarchical data and transaction speed. Although it is still one of IBM’s best-selling products, its implementation makes it difficult to access the data in a variety of ways without additional coding. XML is a relatively new way of representing tree-structured data that offers standardized data interchange, but is challenging if you try to traverse an XML document in an alternative order. In the late 1970s, Oracle and its relational database architecture began to address the inflexibility of the hierarchical model and to provide support for hierarchical traversal of one or more tables using built-in features.

Figure 13-1 shows two hierarchical trees; they could be in different tables or in the same table. Oracle handles more than one tree within the same table without any problem, as you will see in one of the recipes in this chapter. Each node’s key within a tree is typically a primary or alternate key. A given node can have zero, one, or many children; a given child node can have only one parent. The concept of a level is often used when traversing a tree; it specifies how far a node is from the root node of a tree, with level 1 being the level of the root node(s). A row at each level can be one of two types depending on where in the hierarchy they are. Here are the definitions we’ll use throughout the chapter:

**Root**: the highest row in a hierarchical structure

**Child**: any non-root row

**Parent**: any row that has children rows

**Leaf**: a row without any children rows
In Figure 13-1, nodes 1 and 3 are root nodes, 19 and 27 (and many others at the bottom of each tree) are leaf nodes, and all nodes except the leaf nodes are parent nodes. All nodes other than the root nodes are child nodes. A node cannot have three of the four roles, or all four, and a table with one row is the only case where a node can be a root node and a leaf node at the same time.

In this chapter, we’ll cover most of the ways to access hierarchical data in a single table or across several tables. The key clauses you’ll see in a hierarchical query are CONNECT BY and START WITH. The CONNECT BY clause specifies how you want to link rows to their predecessors and successors; this clause may also contain other filters, as you will see in one of the recipes. Typically, you will also have a START WITH clause in your query, but it is not required. You use START WITH to specify where in the hierarchy you want to begin retrieving rows. This is especially useful if you have more than one hierarchy in your table, as in the example in Figure 13-1. You also might use START WITH to exclude the top node of the hierarchy, for example.

In many situations, your hierarchical data might not be consistent. For example, your database might inadvertently indicate that the employee Smith reports to King, the employee Jones reports to Smith, and the employee King reports to Jones. This is clearly not a valid business condition, and Oracle will automatically detect these conditions and return an error message; alternatively, you can use NOCYCLE with CONNECT BY _ISCYCLE to easily identify rows that have a successor that is also a predecessor somewhere in the hierarchy.

Other pseudo-columns make your hierarchical queries more useful for reporting. The LEVEL pseudo-column returns the tree depth of the current row. In Figure 13-1, nodes 1 and 3 would return a LEVEL of 1, nodes 5 and 15 will return a LEVEL of 2, and so forth. Qualifying a column with CONNECT_BY_ROOT returns the column’s value at the top of the hierarchy, regardless of the level number of the current row in the query results.