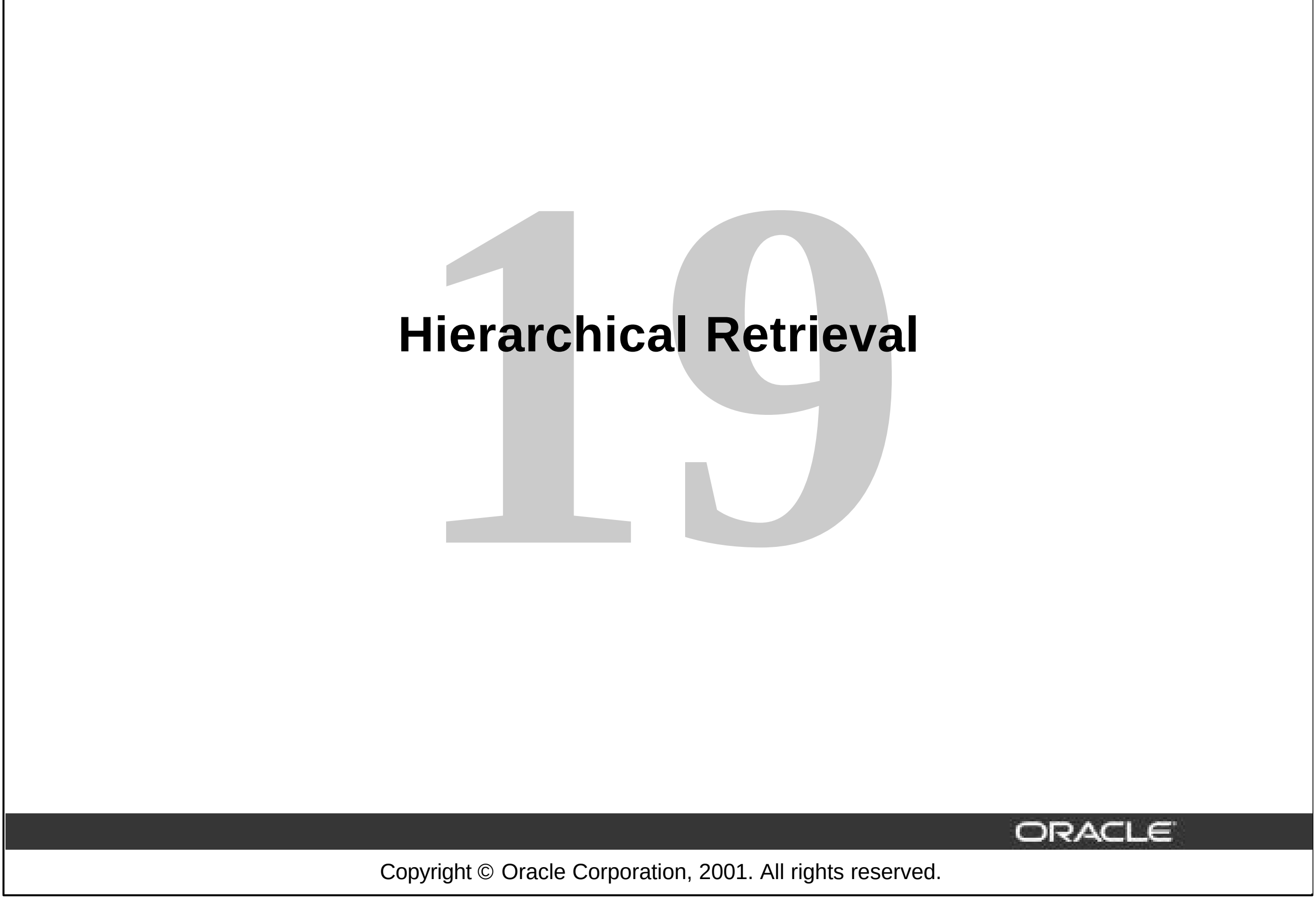
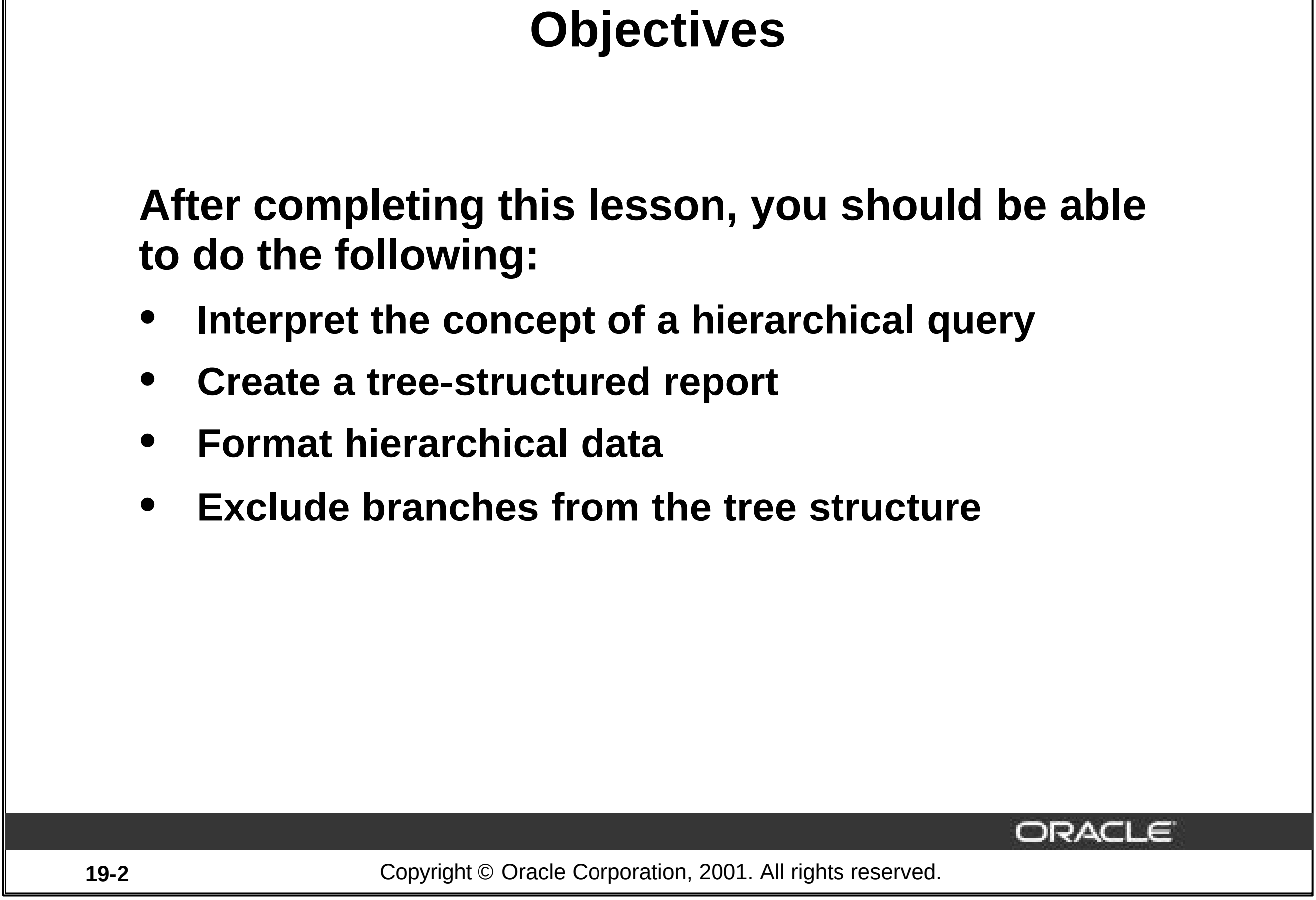


19

Hierarchical Retrieval

ORACLE

Schedule:	Timing	Topic
	30 minutes	Lecture
	20 minutes	Practice
	50 minutes	Total



Objectives

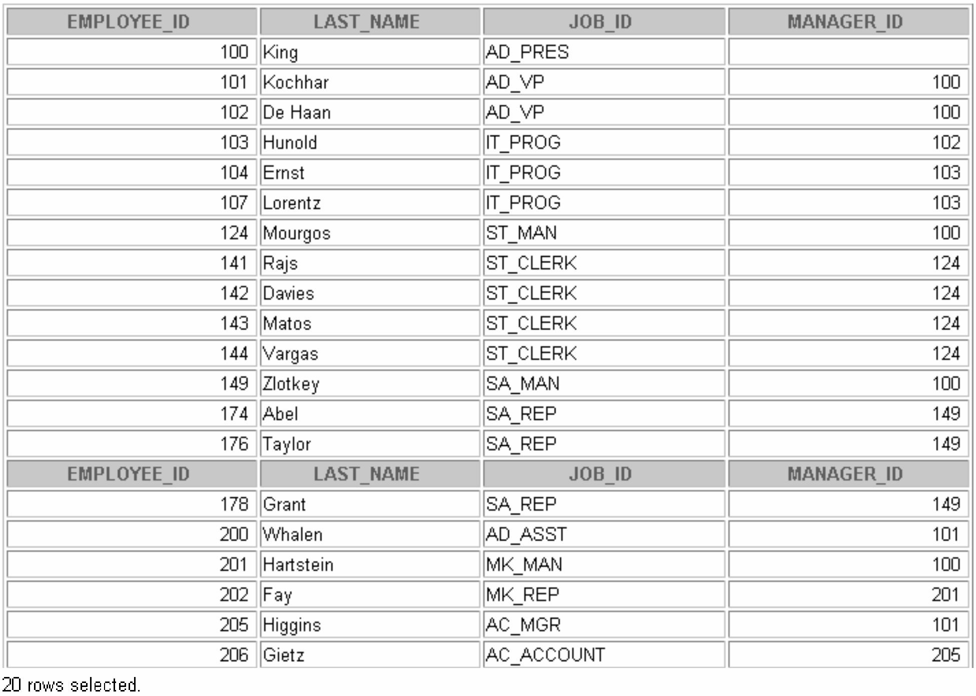
After completing this lesson, you should be able to do the following:

- Interpret the concept of a hierarchical query
- Create a tree-structured report
- Format hierarchical data
- Exclude branches from the tree structure

Lesson Aim

In this lesson, you learn how to use hierarchical queries to create tree-structured reports.

Sample Data from the EMPLOYEES Table



EMPLOYEE_ID	LAST_NAME	JOB_ID	MANAGER_ID
100	King	AD_PRES	
101	Kochhar	AD_VP	100
102	De Haan	AD_VP	100
103	Hunold	IT_PROG	102
104	Ernst	IT_PROG	103
107	Lorentz	IT_PROG	103
124	Mourgos	ST_MAN	100
141	Rajs	ST_CLERK	124
142	Davies	ST_CLERK	124
143	Matos	ST_CLERK	124
144	Vargas	ST_CLERK	124
149	Zlotkey	SA_MAN	100
174	Abel	SA_REP	149
176	Taylor	SA_REP	149
178	Grant	SA_REP	149
200	Whalen	AD_ASST	101
201	Hartstein	MK_MAN	100
202	Fay	MK_REP	201
205	Higgins	AC_MGR	101
206	Gietz	AC_ACCOUNT	205

20 rows selected.

Sample Data from the EMPLOYEES Table

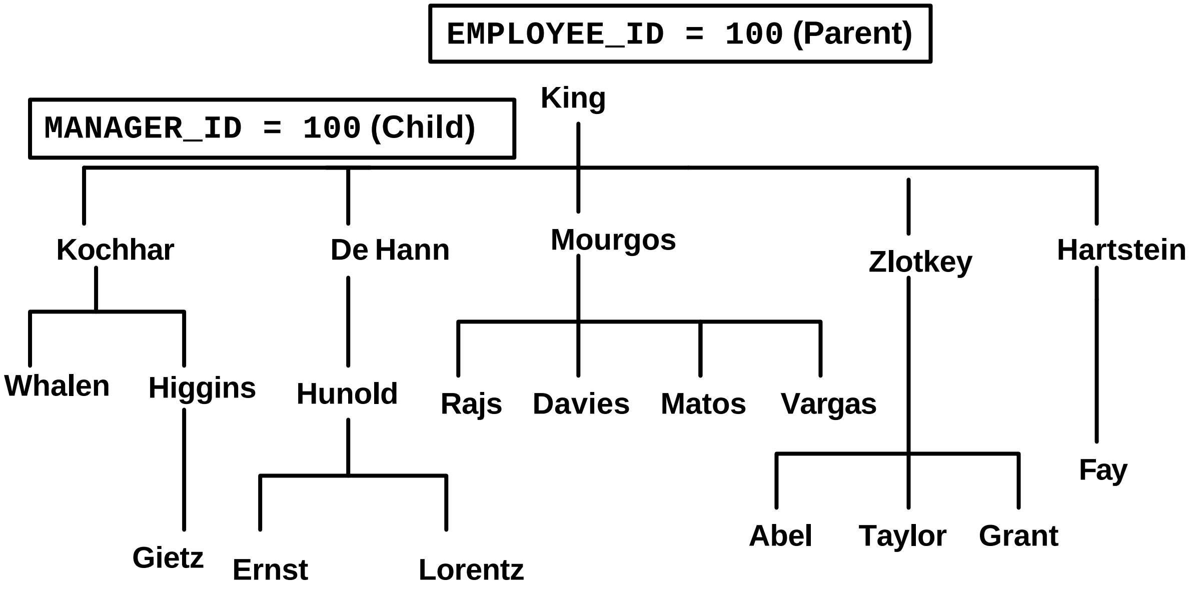
Using hierarchical queries, you can retrieve data based on a natural hierarchical relationship between rows in a table. A relational database does not store records in a hierarchical way. However, where a hierarchical relationship exists between the rows of a single table, a process called *tree walking* enables the hierarchy to be constructed. A hierarchical query is a method of reporting, in order, the branches of a tree.

Imagine a family tree with the eldest members of the family found close to the base or trunk of the tree and the youngest members representing branches of the tree. Branches can have their own branches, and so on.

A hierarchical query is possible when a relationship exists between rows in a table. For example, in the slide, you see that employees with the job IDs of AD_VP, ST_MAN, SA_MAN, and MK_MAN report directly to the president of the company. We know this because the MANAGER_ID column of these records contain the employee ID 100, which belongs to the president (AD_PRES).

Note: Hierarchical trees are used in various fields such as human genealogy (family trees), livestock (breeding purposes), corporate management (management hierarchies), manufacturing (product assembly), evolutionary research (species development), and scientific research.

Natural Tree Structure



`EMPLOYEE_ID = 100` (Parent)

`MANAGER_ID = 100` (Child)

King

Kochhar

De Hann

Mourgos

Zlotkey

Hartstein

Whalen

Higgins

Hunold

Rajs

Davies

Matos

Vargas

Fay

Abel

Taylor

Grant

Gietz

Ernst

Lorentz

Natural Tree Structure

The `EMPLOYEES` table has a tree structure representing the management reporting line. The hierarchy can be created by looking at the relationship between equivalent values in the `EMPLOYEE_ID` and `MANAGER_ID` columns. This relationship can be exploited by joining the table to itself. The `MANAGER_ID` column contains the employee number of the employee's manager.

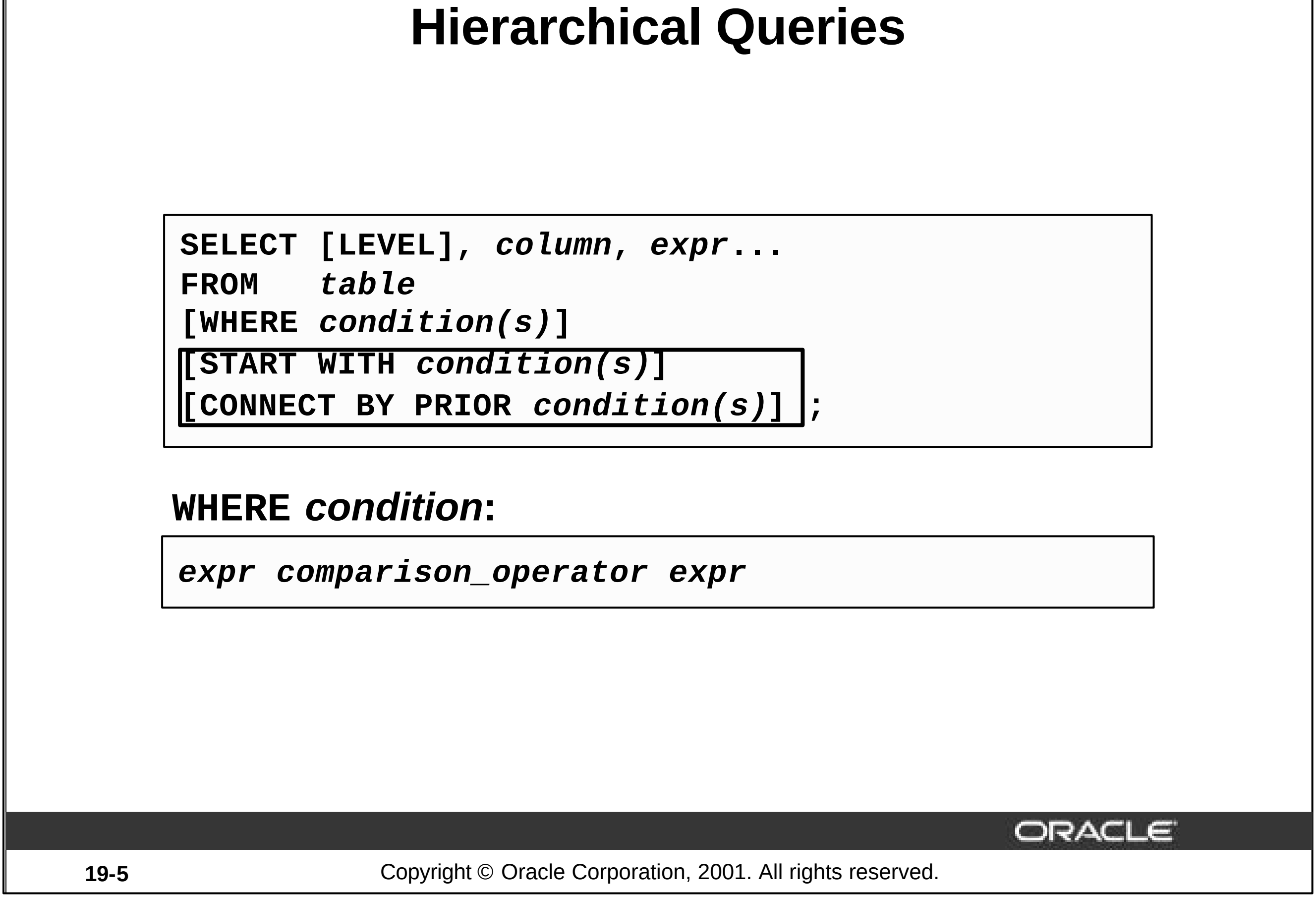
The parent-child relationship of a tree structure enables you to control:

- The direction in which the hierarchy is walked
- The starting point inside the hierarchy

Note: The slide displays an inverted tree structure of the management hierarchy of the employees in the `EMPLOYEES` table.

Instructor Note

You can use the data shown in the previous slide to explain the tree structure shown in the slide.



Hierarchical Queries

```
SELECT [LEVEL], column, expr...
FROM   table
[WHERE condition(s)]
[START WITH condition(s)]
[CONNECT BY PRIOR condition(s)] ;
```

WHERE *condition*:

```
expr comparison_operator expr
```

Keywords and Clauses

Hierarchical queries can be identified by the presence of the `CONNECT BY` and `START WITH` clauses.

In the syntax:

`SELECT` Is the standard `SELECT` clause.

`LEVEL` For each row returned by a hierarchical query, the `LEVEL` pseudocolumn returns 1 for a root row, 2 for a child of a root, and so on.

`FROM` *table* Specifies the table, view, or snapshot containing the columns. You can select from only one table.

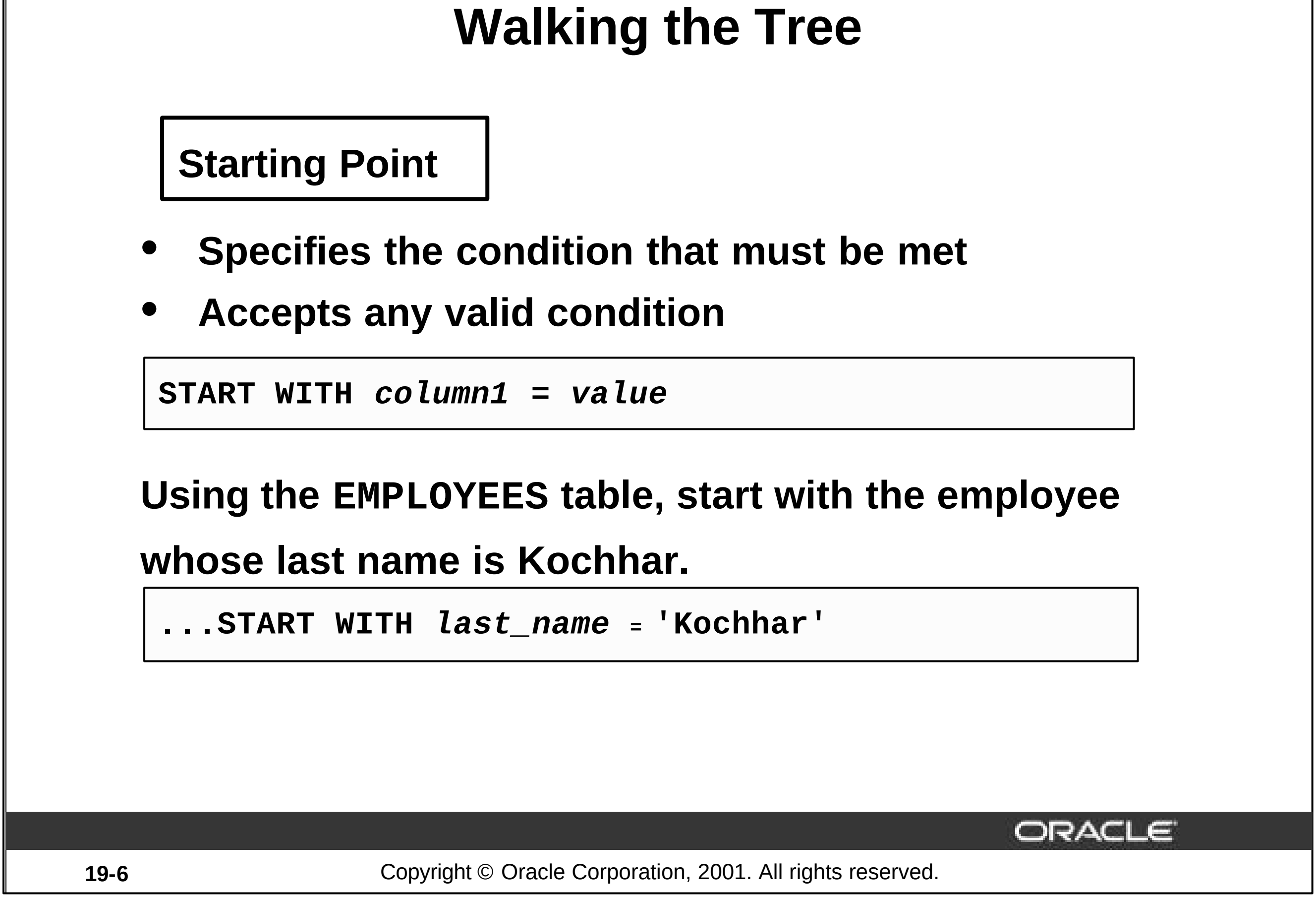
`WHERE` Restricts the rows returned by the query without affecting other rows of the hierarchy.

condition Is a comparison with expressions.

`START WITH` Specifies the root rows of the hierarchy (where to start). This clause is required for a true hierarchical query.

`CONNECT BY PRIOR` Specifies the columns in which the relationship between parent and child rows exist. This clause is required for a hierarchical query.

The `SELECT` statement cannot contain a join or query from a view that contains a join.



Walking the Tree

Starting Point

- **Specifies the condition that must be met**
- **Accepts any valid condition**

```
START WITH column1 = value
```

Using the EMPLOYEES table, start with the employee whose last name is Kochhar.

```
...START WITH last_name = 'Kochhar'
```

Walking the Tree

The row or rows to be used as the root of the tree are determined by the START WITH clause. The START WITH clause can be used in conjunction with any valid condition.

Examples

Using the EMPLOYEES table, start with King, the president of the company.

```
    ... START WITH manager_id IS NULL
```

Using the EMPLOYEES table, start with employee Kochhar. A START WITH condition can contain a subquery.

```
    ... START WITH employee_id = (SELECT employee_id
                                  FROM    employees
                                  WHERE   last_name = 'Kochhar')
```

If the START WITH clause is omitted, the tree walk is started with all of the rows in the table as root rows. If a WHERE clause is used, the walk is started with all the rows that satisfy the WHERE condition. This no longer reflects a true hierarchy.

Note: The clauses CONNECT BY PRIOR and START WITH are not ANSI SQL standard.

Instructor Note

You may wish to add that multiple hierarchical outputs are generated if more than one row satisfies the START WITH condition.

Walking the Tree

```
CONNECT BY PRIOR column1 = column2
```

Walk from the top down, using the EMPLOYEES table.

```
... CONNECT BY PRIOR employee_id = manager_id
```

Direction

Top down → **Column1 = Parent Key**
Column2 = Child Key

Bottom up → **Column1 = Child Key**
Column2 = Parent Key

Walking the Tree (continued)

The direction of the query, whether it is from parent to child or from child to parent, is determined by the `CONNECT BY PRIOR` column placement. The `PRIOR` operator refers to the parent row. To find the children of a parent row, the Oracle Server evaluates the `PRIOR` expression for the parent row and the other expressions for each row in the table. Rows for which the condition is true are the children of the parent. The Oracle Server always selects children by evaluating the `CONNECT BY` condition with respect to a current parent row.

Examples

Walk from the top down using the `EMPLOYEES` table. Define a hierarchical relationship in which the `EMPLOYEE_ID` value of the parent row is equal to the `MANAGER_ID` value of the child row.

```
... CONNECT BY PRIOR employee_id = manager_id
```

Walk from the bottom up using the `EMPLOYEES` table.

```
... CONNECT BY PRIOR manager_id = employee_id
```

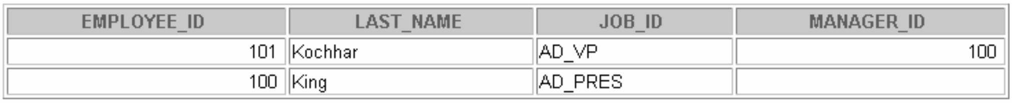
The `PRIOR` operator does not necessarily need to be coded immediately following the `CONNECT BY`. Thus, the following `CONNECT BY PRIOR` clause gives the same result as the one in the preceding example.

```
... CONNECT BY employee_id = PRIOR manager_id
```

Note: The `CONNECT BY` clause cannot contain a subquery.

Walking the Tree: From the Bottom Up

```
SELECT employee_id, last_name, job_id, manager_id
FROM   employees
START  WITH  employee_id = 101
CONNECT BY PRIOR manager_id = employee_id ;
```



EMPLOYEE_ID	LAST_NAME	JOB_ID	MANAGER_ID
101	Kochhar	AD_VP	100
100	King	AD_PRES	

Walking the Tree: From the Bottom Up

The example in the slide displays a list of managers starting with the employee whose employee ID is 101.

Example

In the following example, `EMPLOYEE_ID` values are evaluated for the parent row and `MANAGER_ID`, and `SALARY` values are evaluated for the child rows. The `PRIOR` operator applies only to the `EMPLOYEE_ID` value.

```
... CONNECT BY PRIOR employee_id = manager_id
                                  AND salary > 15000;
```

To qualify as a child row, a row must have a `MANAGER_ID` value equal to the `EMPLOYEE_ID` value of the parent row and must have a `SALARY` value greater than $15,000.

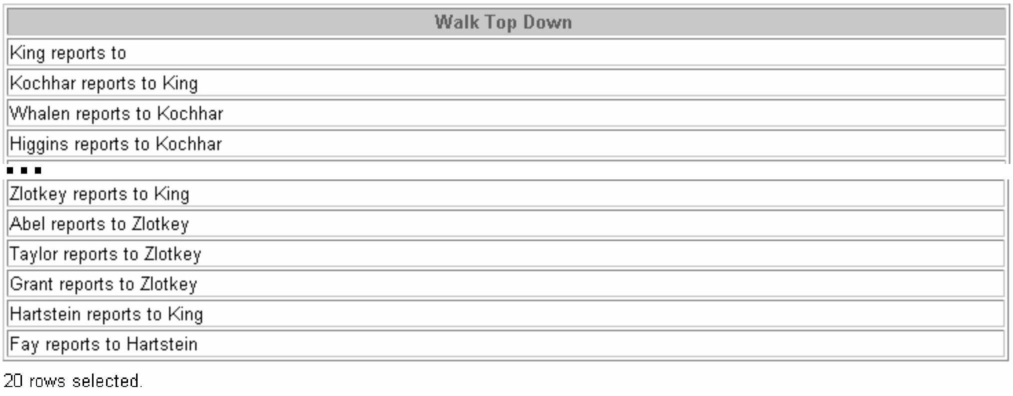
Instructor Note

In the context of the first paragraph, you may wish to include here that the hierarchy will be established to the furthest extremity before the next parent row is evaluated.

In the context of the second paragraph, you may wish to include that additional conditions added to the `CONNECT BY PRIOR` clause potentially eliminated the whole of the branch, hence the `EMPLOYEE_ID AND SALARY` are evaluated for the parent row to determine if it is to be part of the output.

Walking the Tree: From the Top Down

```
SELECT  last_name||' reports to '||
PRIOR   last_name "Walk Top Down"
FROM    employees
START   WITH last_name = 'King'
CONNECT BY PRIOR employee_id = manager_id ;
```

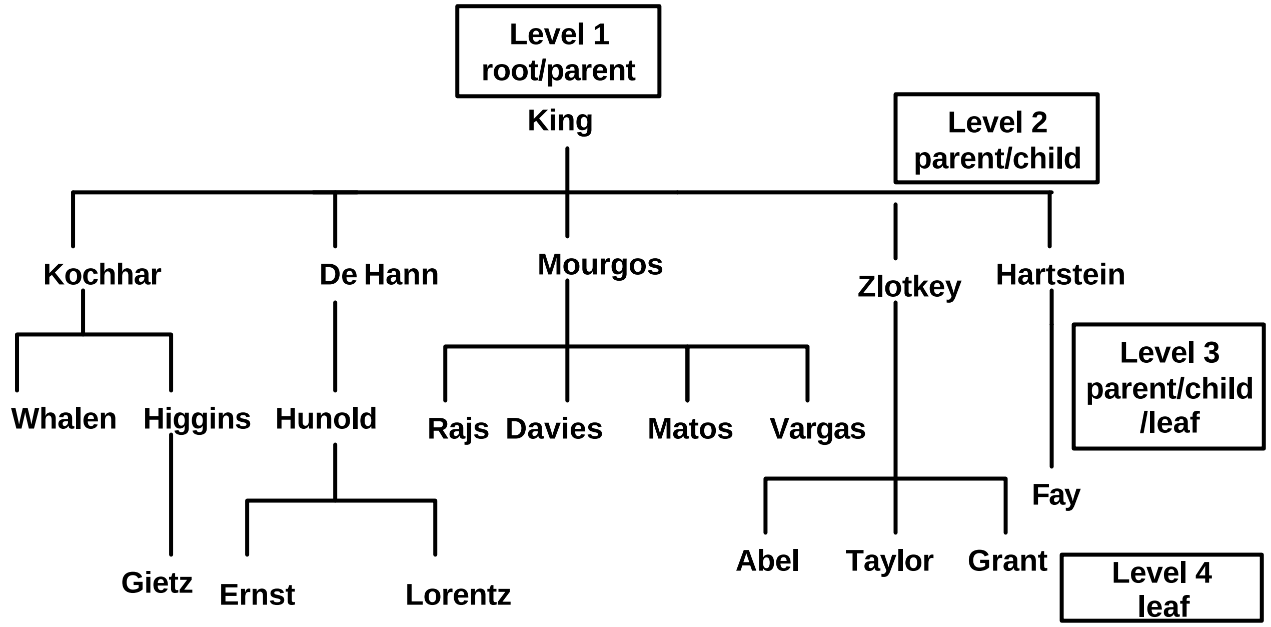


```
Walk Top Down
King reports to
Kochhar reports to King
Whalen reports to Kochhar
Higgins reports to Kochhar
...
Zlotkey reports to King
Abel reports to Zlotkey
Taylor reports to Zlotkey
Grant reports to Zlotkey
Hartstein reports to King
Fay reports to Hartstein
20 rows selected.
```

Walking the Tree: From the Top Down

Walking from the top down, display the names of the employees and their manager. Use employee King as the starting point. Print only one column.

Ranking Rows with the LEVEL Pseudocolumn



Ranking Rows with the LEVEL Pseudocolumn

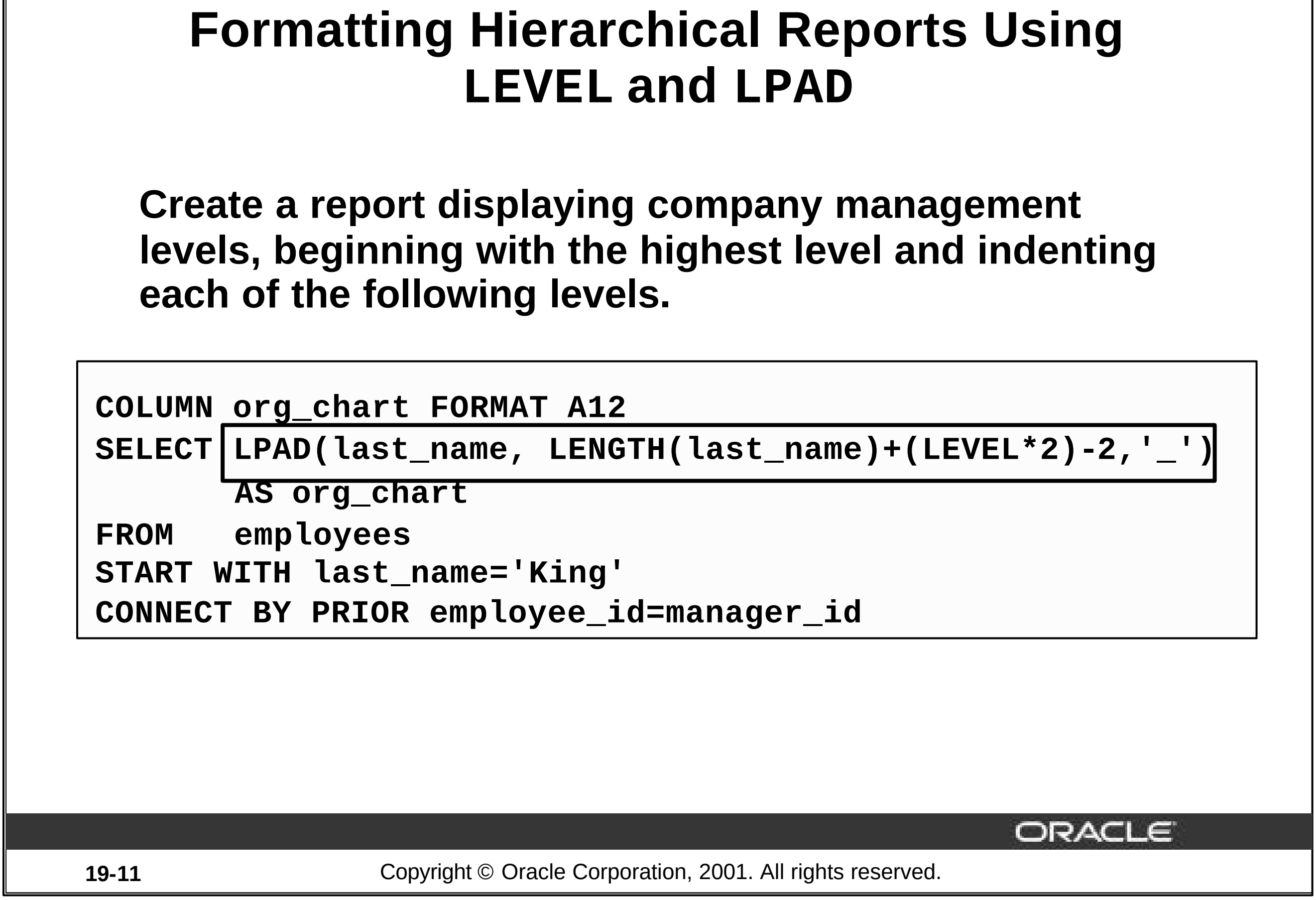
You can explicitly show the rank or level of a row in the hierarchy by using the LEVEL pseudocolumn. This will make your report more readable. The forks where one or more branches split away from a larger branch are called nodes, and the very end of a branch is called a leaf, or leaf node. The diagram in the slide shows the nodes of the inverted tree with their LEVEL values. For example, employee Higgens is a parent and a child, while employee Davies is a child and a leaf.

The LEVEL Pseudocolumn

Value	Level
1	A root node
2	A child of a root node
3	A child of a child, and so on

Note: A *root node* is the highest node within an inverted tree. A *child node* is any nonroot node. A parent node is any node that has children. A leaf node is any node without children. The number of levels returned by a hierarchical query may be limited by available user memory.

In the slide, King is the root or parent (LEVEL = 1). Kochhar, De Hann, Mourgos, Zlotkey, Hartstein, Higgens, and Hunold are children and also parents (LEVEL = 2). Whalen, Rajs, Davies, Matos, Vargas, Gietz, Ernst, Lorentz, Abel, Taylor, Grant, and Fay are children and leaves.
(LEVEL = 3 and LEVEL = 4)



Formatting Hierarchical Reports Using LEVEL and LPAD

Create a report displaying company management levels, beginning with the highest level and indenting each of the following levels.

```
COLUMN org_chart FORMAT A12
SELECT LPAD(last_name, LENGTH(last_name)+(LEVEL*2)-2,'_')
       AS org_chart
FROM   employees
START WITH last_name='King'
CONNECT BY PRIOR employee_id=manager_id
```

Formatting Hierarchical Reports Using LEVEL

The nodes in a tree are assigned level numbers from the root. Use the LPAD function in conjunction with the pseudocolumn LEVEL to display a hierarchical report as an indented tree.

In the example on the slide:

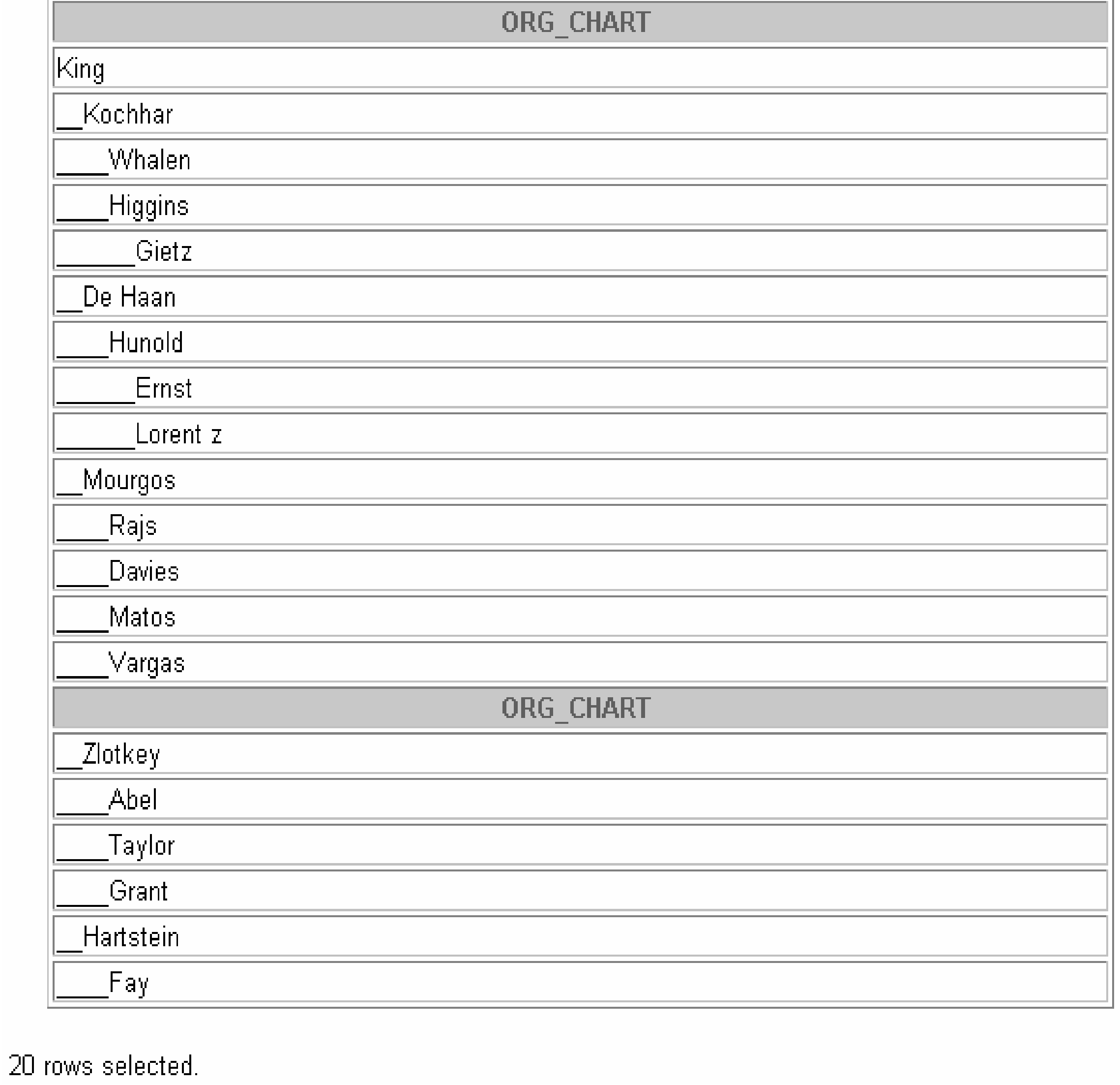
- LPAD(*char1*,*n* [,*char2*]) returns *char1*, left-padded to length *n* with the sequence of characters in *char2*. The argument *n* is the total length of the return value as it is displayed on your terminal screen.
- LPAD(last_name, LENGTH(last_name)+(LEVEL*2)-2,'_')defines the display format.
- *char1* is the LAST_NAME , *n* the total length of the return value, is length of the LAST_NAME +(LEVEL*2)-2 ,and *char2* is '_'.

In other words, this tells SQL to take the LAST_NAME and left-pad it with the '_' character till the length of the resultant string is equal to the value determined by LENGTH(last_name)+(LEVEL*2)-2.

For King, LEVEL = 1. Hence, (2 * 1) - 2 = 2 - 2 = 0. So King does not get padded with any '_' character and is displayed in column 1.

For Kochhar, LEVEL = 2. Hence, (2 * 2) - 2 = 4 - 2 = 2 . So Kochhar gets padded with 2 '_' characters and is displayed indented.

The rest of the records in the EMPLOYEES table are displayed similarly.



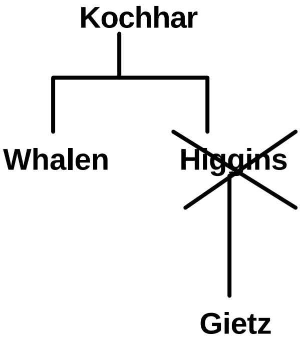
```
ORG_CHART
King
__Kochhar
____Whalen
____Higgins
______Gietz
__De Haan
____Hunold
______Ernst
______Lorent z
__Mourgos
____Rajs
____Davies
____Matos
____Vargas
ORG_CHART
__Zlotkey
____Abel
____Taylor
____Grant
__Hartstein
____Fay

20 rows selected.
```

Pruning Branches

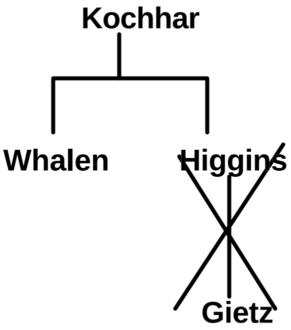
Use the `WHERE` clause to eliminate a node.

```
WHERE last_name != 'Higgins'
```



Use the `CONNECT BY` clause to eliminate a branch.

```
CONNECT BY PRIOR
employee_id = manager_id
AND last_name != 'Higgins'
```



Pruning Branches

You can use the `WHERE` and `CONNECT BY` clauses to prune the tree; that is, to control which nodes or rows are displayed. The predicate you use acts as a Boolean condition.

Examples

Starting at the root, walk from the top down, and eliminate employee Higgins in the result, but process the child rows.

```
SELECT  department_id, employee_id,last_name, job_id, salary
FROM    employees
WHERE   last_name  != 'Higgins'
START   WITH manager_id IS NULL
CONNECT BY PRIOR employee_id = manager_id;
```

Starting at the root, walk from the top down, and eliminate employee Higgins and all child rows.

```
SELECT  department_id, employee_id,last_name, job_id, salary
FROM    employees
START   WITH manager_id IS NULL
CONNECT BY PRIOR employee_id = manager_id
AND     last_name != 'Higgins';
```

Instructor Note

You may wish to add here that using a `WHERE` clause to restrict a node could result in the hierarchy not being reflected truly by the output.

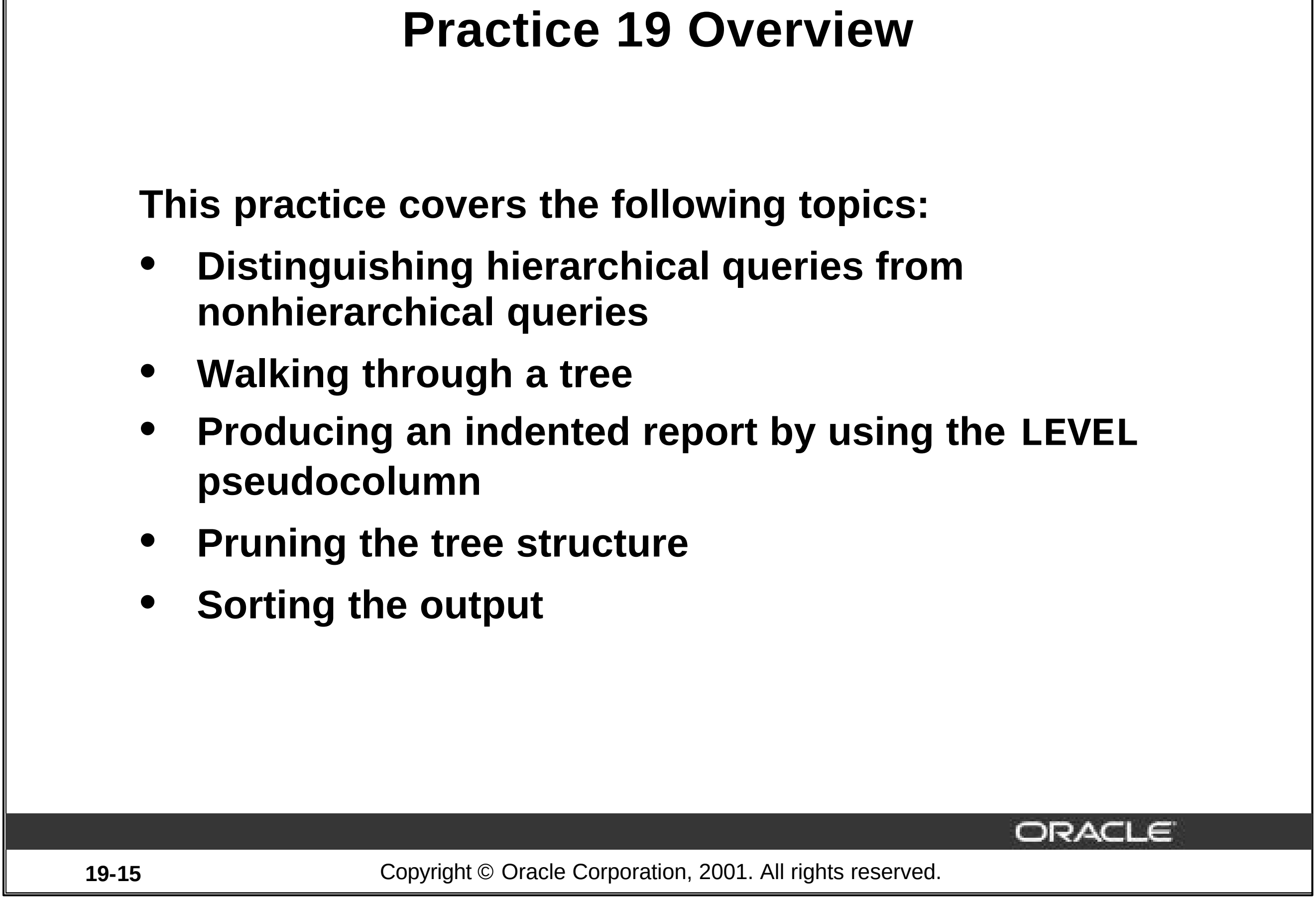
Summary

In this lesson, you should have learned the following:

- **You can use hierarchical queries to view a hierarchical relationship between rows in a table.**
- **You specify the direction and starting point of the query.**
- **You can eliminate nodes or branches by pruning.**

Summary

You can use hierarchical queries to retrieve data based on a natural hierarchical relationship between rows in a table. The `LEVEL` pseudocolumn counts how far down a hierarchical tree you have traveled. You can specify the direction of the query using the `CONNECT BY PRIOR` clause. You can specify the starting point using the `START WITH` clause. You can use the `WHERE` and `CONNECT BY` clauses to prune the tree branches.



Practice 19 Overview

This practice covers the following topics:

- Distinguishing hierarchical queries from nonhierarchical queries
- Walking through a tree
- Producing an indented report by using the LEVEL pseudocolumn
- Pruning the tree structure
- Sorting the output

Practice 19 Overview

In this practice, you gain practical experience in producing hierarchical reports.

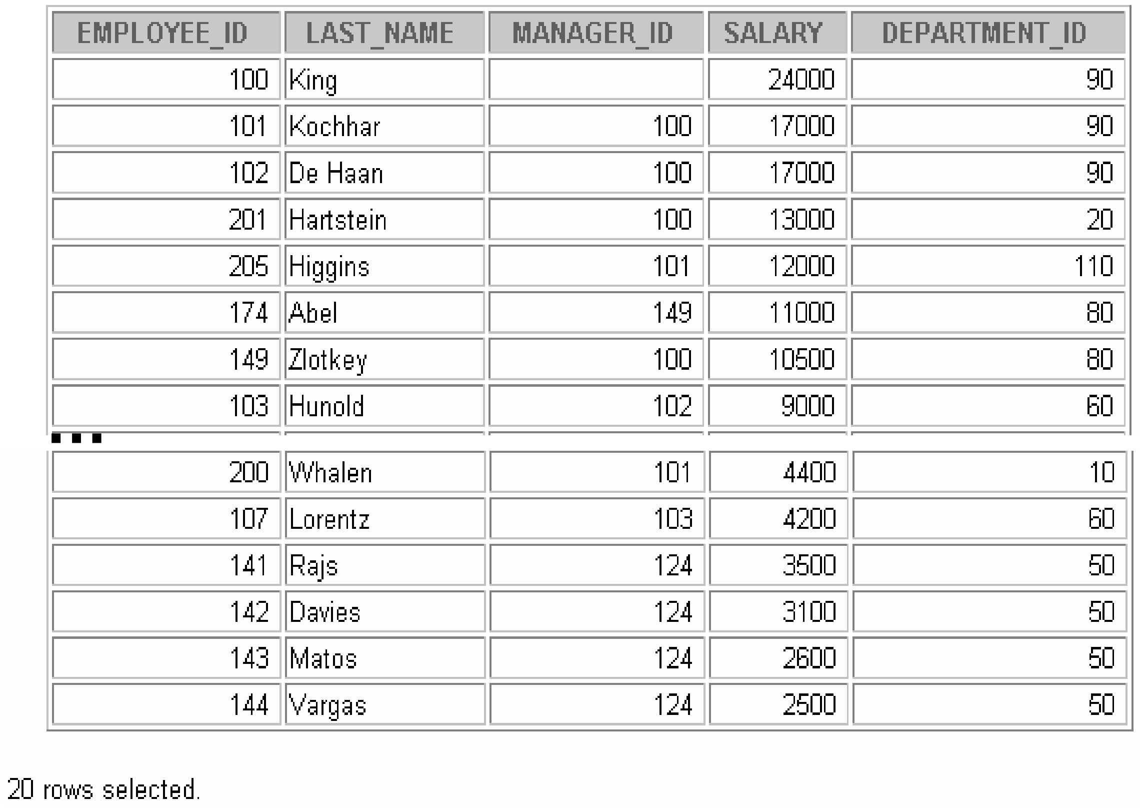
Paper-Based Questions

Question 1 is a paper-based question.

Practice 19

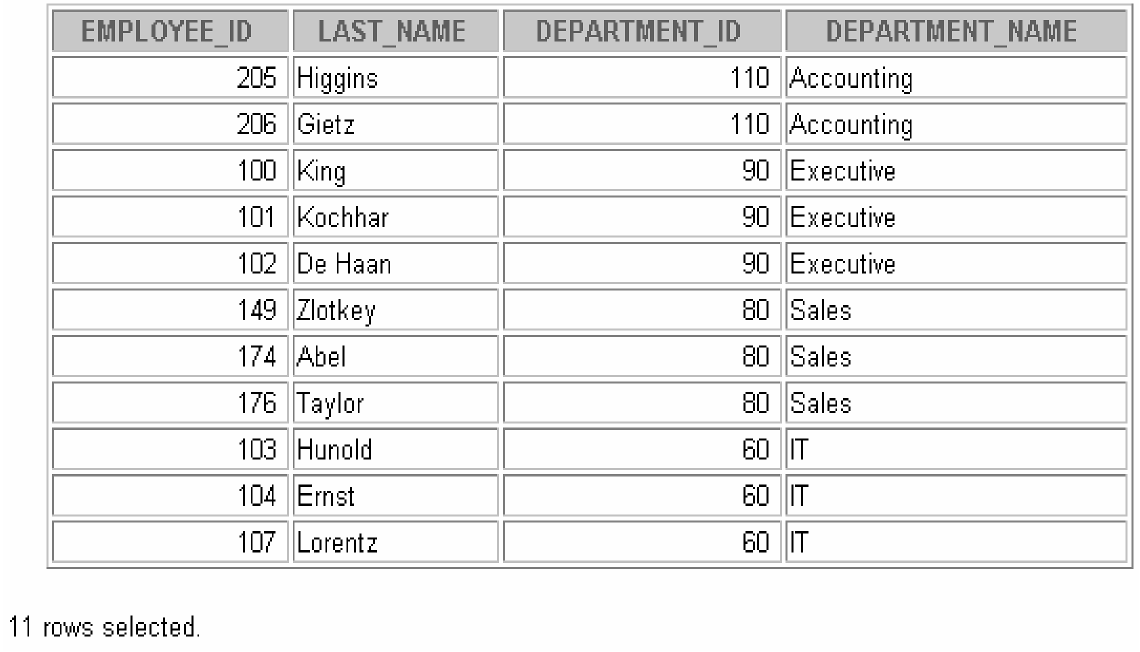
1. Look at the following outputs. Are these outputs the result of a hierarchical query? Explain why or why not.

Exhibit 1:



EMPLOYEE_ID	LAST_NAME	MANAGER_ID	SALARY	DEPARTMENT_ID
100	King		24000	90
101	Kochhar	100	17000	90
102	De Haan	100	17000	90
201	Hartstein	100	13000	20
205	Higgins	101	12000	110
174	Abel	149	11000	80
149	Zlotkey	100	10500	80
103	Hunold	102	9000	60
■■■				
200	Whalen	101	4400	10
107	Lorentz	103	4200	60
141	Rajs	124	3500	50
142	Davies	124	3100	50
143	Matos	124	2600	50
144	Vargas	124	2500	50

20 rows selected.

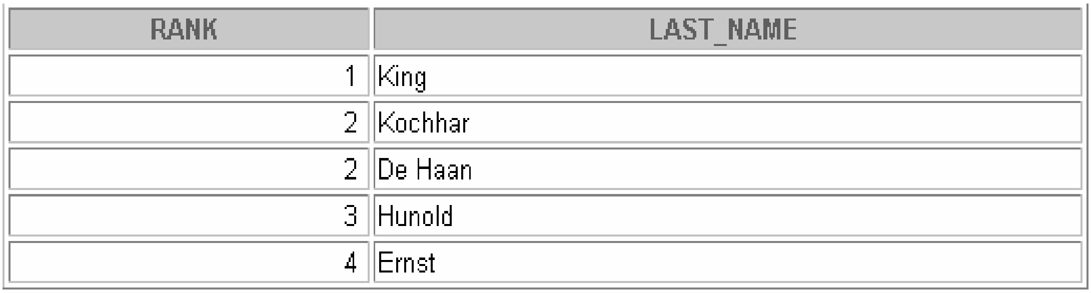


EMPLOYEE_ID	LAST_NAME	DEPARTMENT_ID	DEPARTMENT_NAME
205	Higgins	110	Accounting
206	Gietz	110	Accounting
100	King	90	Executive
101	Kochhar	90	Executive
102	De Haan	90	Executive
149	Zlotkey	80	Sales
174	Abel	80	Sales
176	Taylor	80	Sales
103	Hunold	60	IT
104	Ernst	60	IT
107	Lorentz	60	IT

11 rows selected.

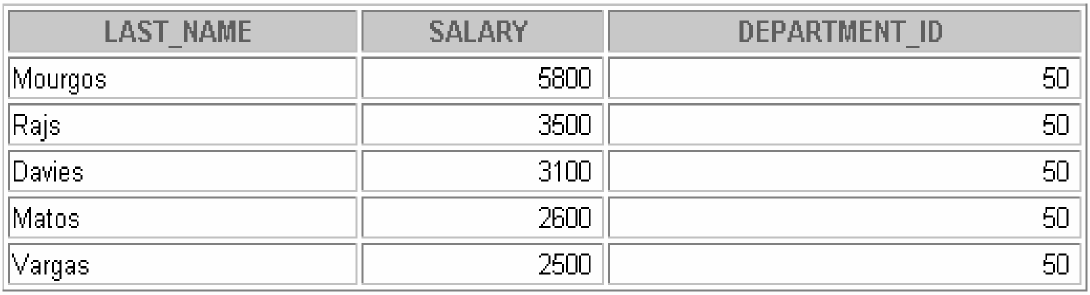
Practice 19 (continued)

Exhibit 3:



RANK	LAST_NAME
1	King
2	Kochhar
2	De Haan
3	Hunold
4	Ernst

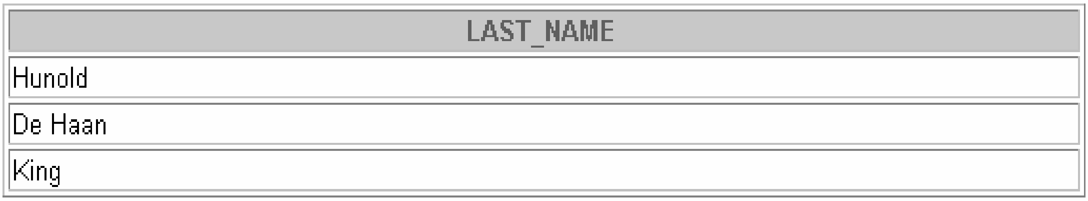
2. Produce a report showing an organization chart for Mourgos’s department. Print last names, salaries, and department IDs.



LAST_NAME	SALARY	DEPARTMENT_ID
Mourgos	5800	50
Rajs	3500	50
Davies	3100	50
Matos	2600	50
Vargas	2500	50

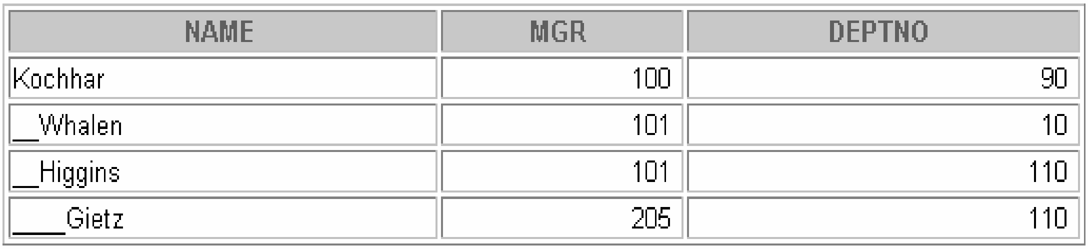
ıy

his immediate manager first.



LAST_NAME
Hunold
De Haan
King

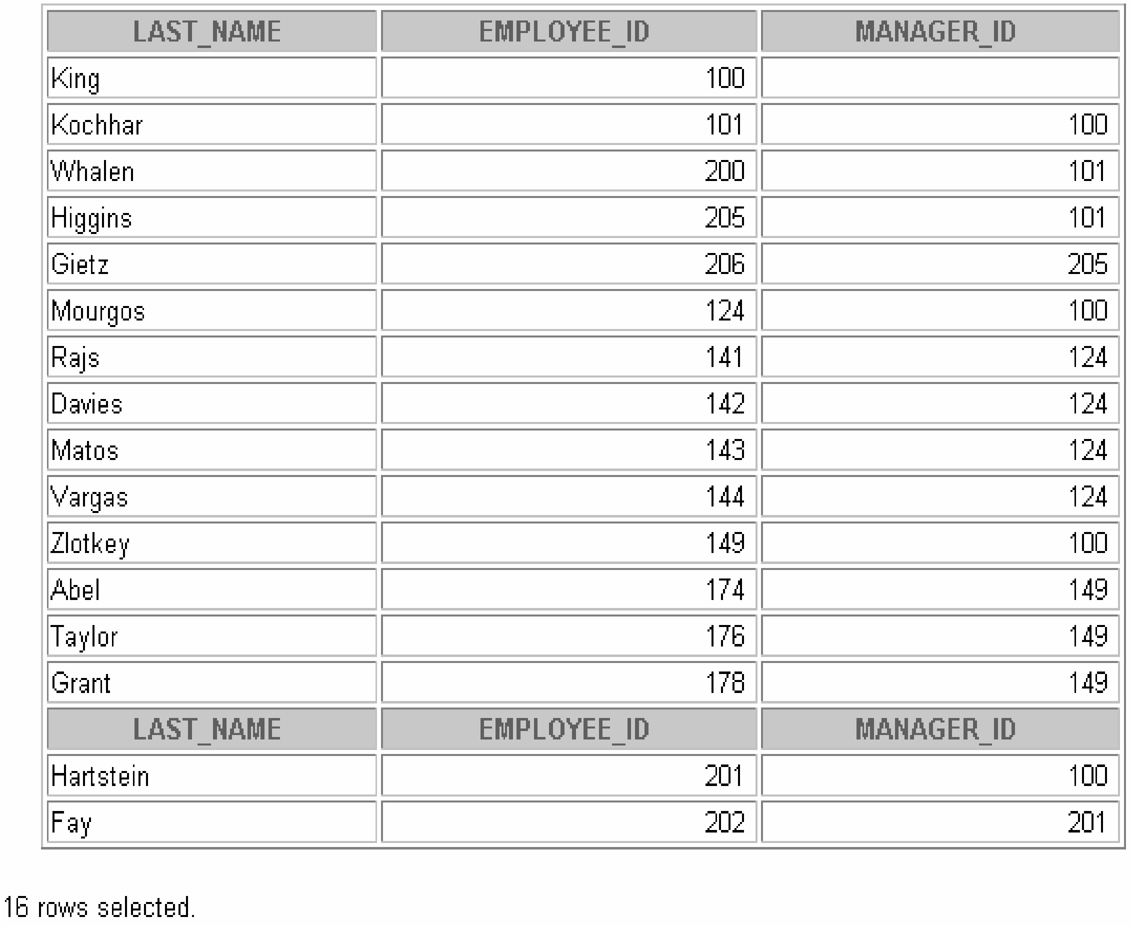
4. Create an indented report showing the management hierarchy starting from the employee whose `LAST_NAME` is Kochhar. Print the employee’s last name, manager ID, and department ID. Give alias names to the columns as shown in the sample output.



NAME	MGR	DEPTNO
Kochhar	100	90
__Whalen	101	10
__Higgins	101	110
____Gietz	205	110

If you have time, complete the following exercise:

5. Produce a company organization chart that shows the management hierarchy. Start with the person at the top level, exclude all people with a job ID of `IT_PROG`, and exclude De Haan and those employees who report to De Haan.



LAST_NAME	EMPLOYEE_ID	MANAGER_ID
King	100	
Kochhar	101	100
Whalen	200	101
Higgins	205	101
Gietz	206	205
Mourgos	124	100
Rajs	141	124
Davies	142	124
Matos	143	124
Vargas	144	124
Zlotkey	149	100
Abel	174	149
Taylor	176	149
Grant	178	149
LAST_NAME	EMPLOYEE_ID	MANAGER_ID
Hartstein	201	100
Fay	202	201

16 rows selected.