

Hierarchical Retrieval

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

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
105	Austin	IT_PROG	103
106	Pataballa	IT_PROG	103
107	Lorentz	IT_PROG	103
108	Greenberg	FI_MGR	101

...

EMPLOYEE_ID	LAST_NAME	JOB_ID	MANAGER_ID
196	Walsh	SH_CLERK	124
197	Feeney	SH_CLERK	124
198	OConnell	SH_CLERK	124
199	Grant	SH_CLERK	124
200	Whalen	AD_ASST	101
201	Hartstein	MK_MAN	100
202	Fay	MK_REP	201
203	Mavris	HR_REP	101
204	Baer	PR_REP	101
205	Higgins	AC_MGR	101
206	Gietz	AC_ACCOUNT	205

107 rows selected.

Natural Tree Structure

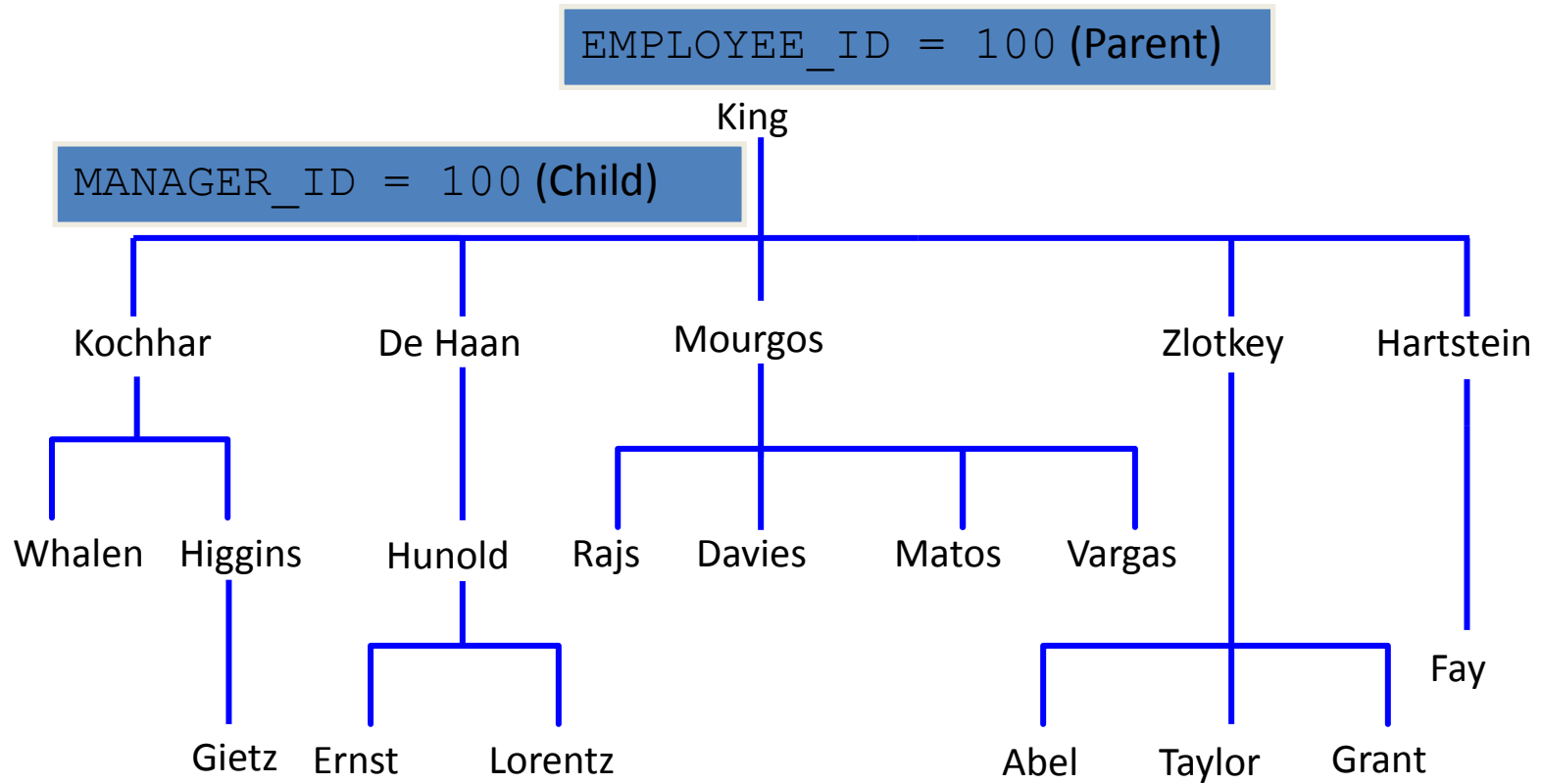
Using **hierarchical queries**, you can retrieve data based on a **natural hierarchical relationship** between rows in a table.

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 possible when a relationship exists between rows in a table

Natural Tree Structure



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.

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

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.

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

CONNECT BY Specifies the columns in which the relationship between parent and child rows exist.

PRIOR 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'
```

Starting Point(s)

The row or rows to be used as the root of the tree are determined by the `START WITH` clause.

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.

Parent-Child relationship

The **PRIOR** operator refers to the **parent row**. To find the child rows 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 child rows of the parent.

The Oracle server always selects child rows by evaluating the `CONNECT BY` condition with respect to a current parent row.

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 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

King reports to

Kochhar reports to King

Greenberg reports to Kochhar

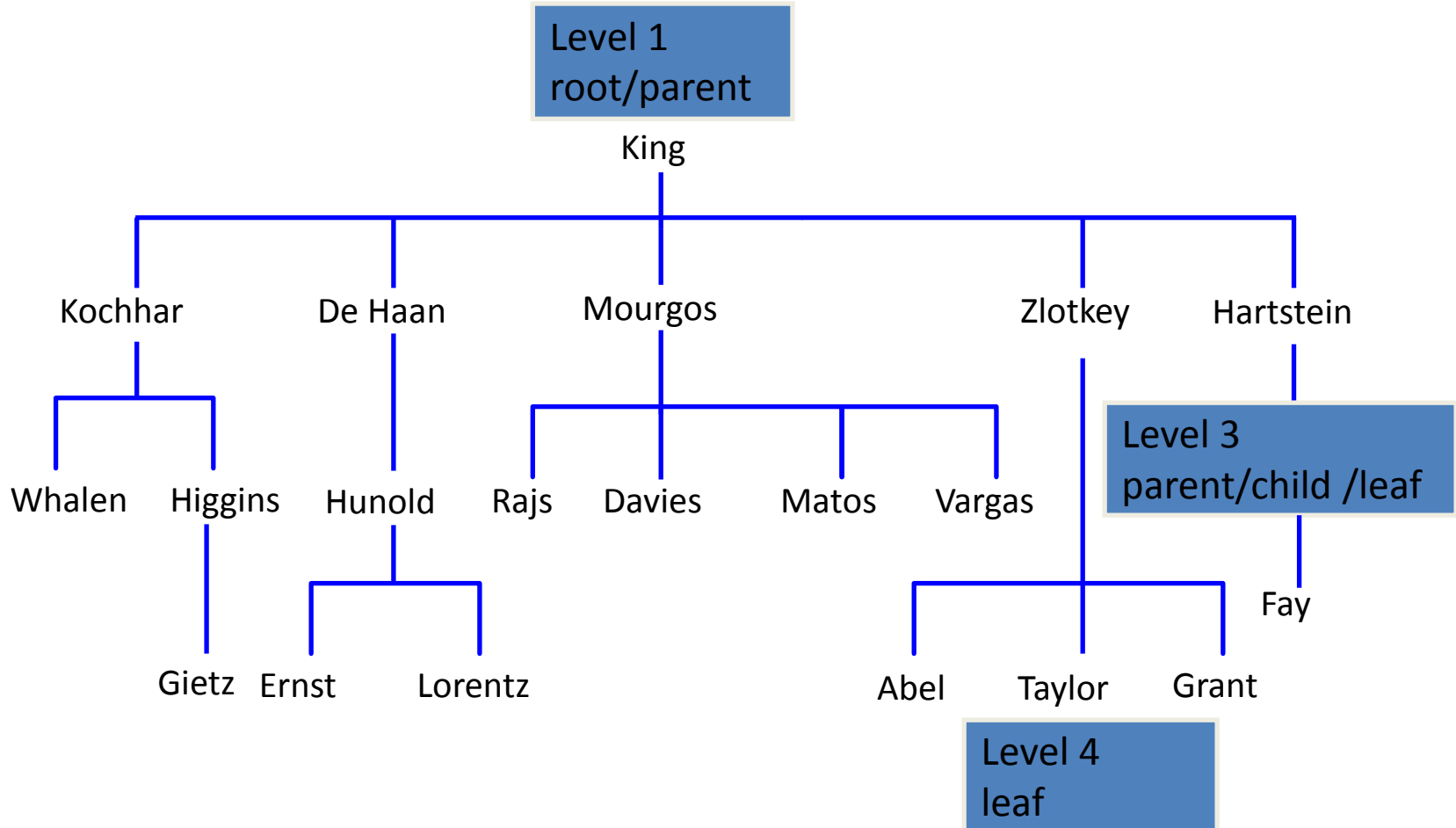
Faviet reports to Greenberg

Chen reports to Greenberg

...

108 rows selected.

Ranking Rows with the LEVEL Pseudocolumn



Formatting Hierarchical Reports Using LEVEL and LPAD

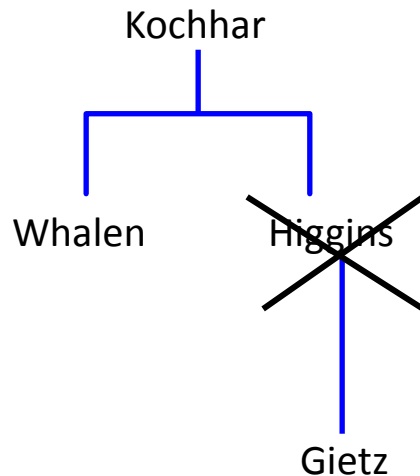
- Create a report displaying company management levels, beginning with the highest level and indenting each of the following levels.

```
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
```


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'
```

