# Database Programming

Self-Joins and Hierarchical Queries

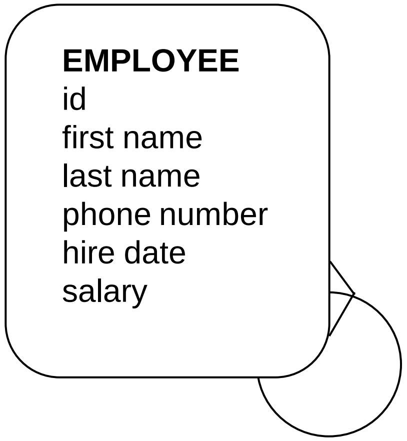
## Objectives

This lesson covers the following objectives:

- Construct and execute a SELECT statement to join a table to itself using a self-join
- Interpret the concept of a hierarchical query
- Create a tree-structured report
- Format hierarchical data
- Exclude branches from the tree structure

## Purpose

In data modeling, it was sometimes necessary to show an entity with a relationship to itself.



```
SELECT worker.last_name || ' works for ' ||
manager.last_name
FROM employees worker JOIN employees manager
ON worker.manager_id = manager.employee_id;
```

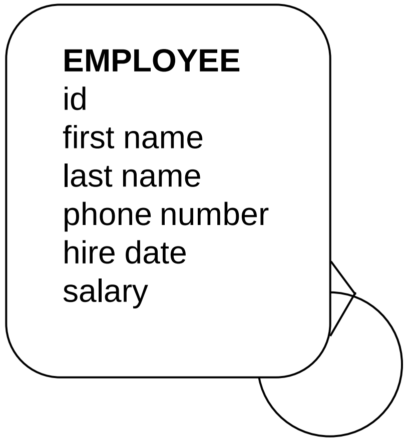
**EMPLOYEES TABLE**

| employee_id | first_name | last_name | email | phone_number | hire_date | job_id | salary | commission_pct | manager_id | department_id |
|---|---|---|---|---|---|---|---|---|---|---|
| 100 | Steven | King | SKING | 515.123.4567 | 17-JUN-1987 | AD_PRES | 24000 | (null) | (null) | 90 |
| 101 | Neena | Kochhar | NKOCHHAR | 515.123.4568 | 21-SEP-1989 | AD_VP | 17000 | (null) | 100 | 90 |

(This EMPLOYEES table output shows columns resulting from a join)

## Purpose (cont.)

For example, an employee can also be a manager. We showed this using the "pig's ear" relationship.



```
SELECT worker.last_name || ' works for ' ||
manager.last_name
FROM employees worker JOIN employees manager
ON worker.manager_id = manager.employee_id;
```

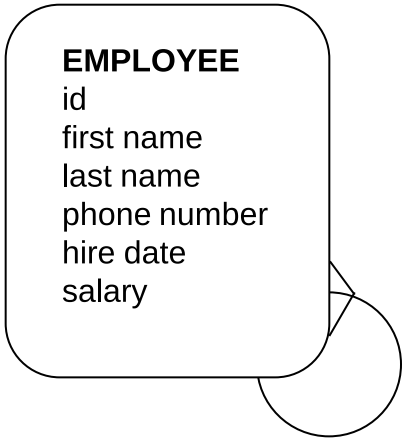
**EMPLOYEES TABLE**

| employee_id | first_name | last_name | email | phone_number | hire_date | job_id | salary | commission_pct | manager_id | department_id |
|---|---|---|---|---|---|---|---|---|---|---|
| 100 | Steven | King | SKING | 515.123.4567 | 17-JUN-1987 | AD_PRES | 24000 | (null) | (null) | 90 |
| 101 | Neena | Kochhar | NKOCHHAR | 515.123.4568 | 21-SEP-1989 | AD_VP | 17000 | (null) | 100 | 90 |

(This EMPLOYEES table output shows columns resulting from a join)

## Purpose (cont.)

Once we have a real employees table, a special kind of join called a self-join is required to access this data.



```
SELECT worker.last_name || ' works for ' ||
manager.last_name
FROM employees worker JOIN employees manager
ON worker.manager_id = manager.employee_id;
```

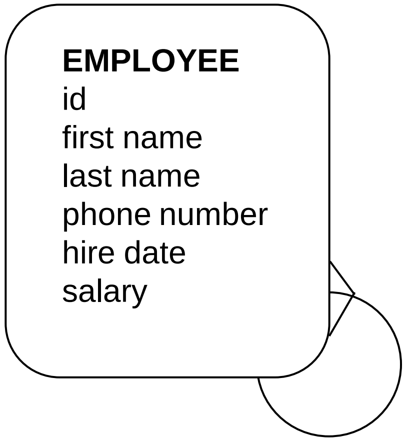
**EMPLOYEES TABLE**

| employee_id | first_name | last_name | email | phone_number | hire_date | job_id | salary | commission_pct | manager_id | department_id |
|---|---|---|---|---|---|---|---|---|---|---|
| 100 | Steven | King | SKING | 515.123.4567 | 17-JUN-1987 | AD_PRES | 24000 | (null) | (null) | 90 |
| 101 | Neena | Kochhar | NKOCHHAR | 515.123.4568 | 21-SEP-1989 | AD_VP | 17000 | (null) | 100 | 90 |

(This EMPLOYEES table output shows columns resulting from a join)

## Purpose (cont.)

You probably realize by now the importance of a data model once it becomes a database.



```
SELECT worker.last_name || ' works for ' ||
manager.last_name
FROM employees worker JOIN employees manager
ON worker.manager_id = manager.employee_id;
```

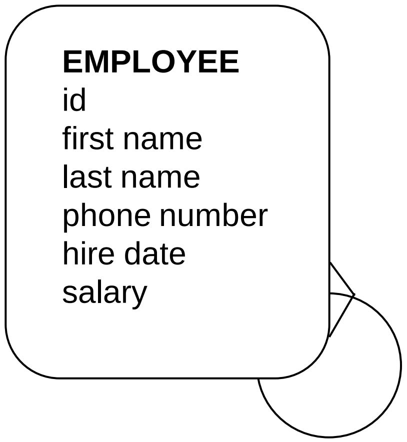
**EMPLOYEES TABLE**

| employee_id | first_name | last_name | email | phone_number | hire_date | job_id | salary | commission_pct | manager_id | department_id |
|---|---|---|---|---|---|---|---|---|---|---|
| 100 | Steven | King | SKING | 515.123.4567 | 17-JUN-1987 | AD_PRES | 24000 | (null) | (null) | 90 |
| 101 | Neena | Kochhar | NKOCHHAR | 515.123.4568 | 21-SEP-1989 | AD_VP | 17000 | (null) | 100 | 90 |

(This EMPLOYEES table output shows columns resulting from a join)

## Purpose (cont.)

It’s no coincidence that the data model looks similar to the tables we now have in the database.



```
SELECT worker.last_name || ' works for ' ||
manager.last_name
FROM employees worker JOIN employees manager
ON worker.manager_id = manager.employee_id;
```

**EMPLOYEES TABLE**

| employee_id | first_name | last_name | email | phone_number | hire_date | job_id | salary | commission_pct | manager_id | department_id |
|---|---|---|---|---|---|---|---|---|---|---|
| 100 | Steven | King | SKING | 515.123.4567 | 17-JUN-1987 | AD_PRES | 24000 | (null) | (null) | 90 |
| 101 | Neena | Kochhar | NKOCHHAR | 515.123.4568 | 21-SEP-1989 | AD_VP | 17000 | (null) | 100 | 90 |

(This EMPLOYEES table output shows columns resulting from a join)

## SELF-JOIN

To join a table to itself, the table is given two names or aliases. This will make the database “think” that there are two tables.

EMPLOYEES (worker)

| employee_id | last_name | manager_id |
|---|---|---|
| 100 | King | |
| 101 | Kochar | 100 |
| 102 | De Haan | 100 |
| 103 | Hunold | 102 |
| 104 | Ernst | 103 |
| 107 | Lorentz | 103 |
| 124 | Mourgos | 100 |

EMPLOYEES (manager)

| employee_id | last_name |
|---|---|
| 100 | King |
| 101 | Kochar |
| 102 | De Haan |
| 103 | Hunold |
| 104 | Ernst |
| 107 | Lorentz |
| 124 | Mourgos |

Manager_id in the worker table is equal to employee_id in the manager table.

## SELF-JOIN (cont.)

Choose alias names that relate to the data's association with that table.

EMPLOYEES (worker)

| employee_id | last_name | manager_id |
|---|---|---|
| 100 | King | |
| 101 | Kochar | 100 |
| 102 | De Haan | 100 |
| 103 | Hunold | 102 |
| 104 | Ernst | 103 |
| 107 | Lorentz | 103 |
| 124 | Mourgos | 100 |

EMPLOYEES (manager)

| employee_id | last_name |
|---|---|
| 100 | King |
| 101 | Kochar |
| 102 | De Haan |
| 103 | Hunold |
| 104 | Ernst |
| 107 | Lorentz |
| 124 | Mourgos |

Manager_id in the worker table is equal to employee_id in the manager table.

## SELF-JOIN Example

Here is another example of a self-join. In this example of a band, we have members of the band who play an instrument, and members of the band who both play an instrument and are their section's lead player or chair. A readable way to show this self-join is:

```
SELECT chair.last_name
               || ' is the section chair of ' ||
               player.last_name
FROM    band_members chair JOIN band_members player
ON         (player.chair_id =  chair.member_id);
```

## Hierarchical Queries

Closely related to self-joins are hierarchical queries. On the previous pages, you saw how you can use self-joins to see each employee’s direct manager. With hierarchical queries, we can also see who that manager works for, and so on.

With this query, we can build an Organization Chart showing the structure of a company or a department. 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.

## Using Hierarchical Queries

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 the branches of a tree in a specific order.

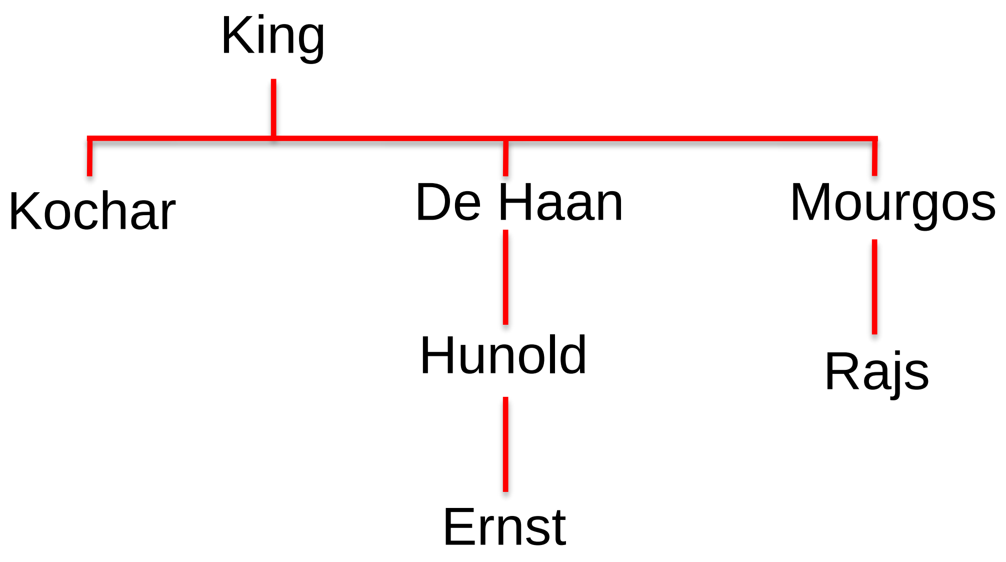
## Hierarchical Queries Data

Examine the sample data from the EMPLOYEES table below, and look at how you can manually make the connections to see who works for whom starting with Steven King and moving through the tree from there.

| EMPLOYEE_ID | FIRST_NAME | LAST_NAME | EMAIL | PHONE_NUMBER | HIRE_DATE | JOB_ID | SALARY | COMM_PCT | MGR_ID | DEPT_ID |
|---|---|---|---|---|---|---|---|---|---|---|
| 100 | Steven | King | SKING | 515.123.4567 | 17-JUN-1987 | AD_PRES | 24000 | (null) | (null) | 90 |
| 101 | Neena | Kochhar | NKOCHHAR | 515.123.4568 | 21-SEP-1989 | AD_VP | 17000 | (null) | 100 | 90 |
| 102 | Lex | De Haan | LDEHAAN | 515.123.4569 | 13-JAN-1993 | AD_VP | 17000 | (null) | 100 | 90 |
| 103 | Alexander | Hunold | AHUNOLD | 590.423.4567 | 03-JAN-1990 | IT_PROG | 9000 | (null) | 102 | 60 |
| 104 | Bruce | Ernst | BERNST | 590.423.4568 | 21-MAY-1991 | IT_PROG | 6000 | (null) | 103 | 60 |
| 124 | Kevin | Mourgos | KMOURGOS | 650.123.5234 | 16-NOV-1999 | ST_MAN | 5800 | (null) | 100 | 50 |
| 141 | Trenna | Rajs | TRAJS | 650.121.8009 | 17-OCT-1995 | ST_CLERK | 3500 | (null) | 124 | 50 |

## Hierarchical Queries Illustrated

The organization chart that we can draw from the data in the EMPLOYEES table will look like this:



## Hierarchical Queries Keywords

Hierarchical queries have their own new keywords: START WITH, CONNECT BY PRIOR, and LEVEL.

START WITH identifies which row to use as the Root for the tree it is constructing, CONNECT BY PRIOR explains how to do the inter-row joins, and LEVEL specifies how many branches deep the tree will traverse.

## Hierarchical Queries Keyword Example

```
SELECT employee_id, last_name, job_id, manager_id
FROM   employees
START  WITH  employee_id = 100
CONNECT BY PRIOR employee_id = manager_id
```

| employee_id | last_name | job_id | manager_id |
|---|---|---|---|
| 100 | King | AD_PRES | (null) |
| 101 | Kochhar | AD_VP | 100 |
| 200 | Whalen | AD_ASST | 101 |
| 205 | Higgins | AC_MGR | 101 |
| 206 | Gietz | AC_ACCOUNT | 205 |
| 102 | De Haan | AD_VP | 100 |
| 103 | Hunould | IT_PROG | 102 |
| 141 | Rajs | ST_CLERK | 124 |

## Hierarchical Queries Another Example

```
SELECT last_name || ' reports to ‘ ||  PRIOR last_name as "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 |
| Gietz reports to Higgins |
| De Haan reports to King |
| Hunold reports to De Haan |
| Ernst reports to Hunold |

## Hierarchical Queries Level Example

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

LEVEL is a pseudo-column used with hierarchical queries, and it counts the number of steps it has taken from the root of the tree.

| LEVEL | Walk Top Down |
|---|---|
| 1 | King reports to |
| 2 | Kochhar reports to King |
| 3 | Whalen reports to Kochhar |
| 3 | Higgins reports to Kochhar |
| 4 | Gietz reports to Higgins |
| 2 | De Haan reports to King |
| 3 | Hunold reports to De Haan |
| 4 | Ernst reports to Hunold |

## Hierarchical Query Report

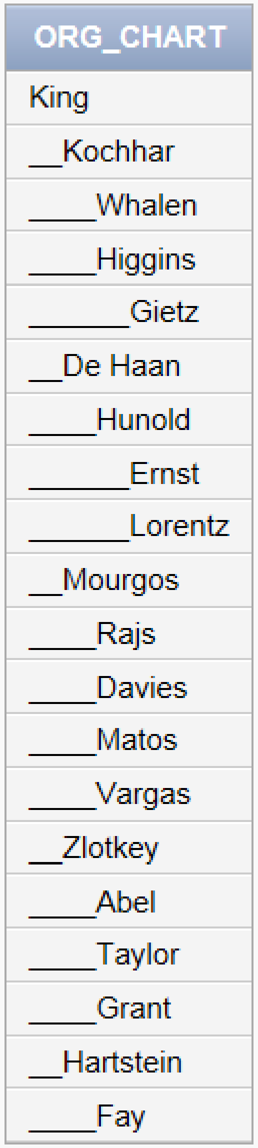
If you wanted to create a report displaying company management levels, beginning with the highest level and indenting each of the following levels, then this would be easy to do using the LEVEL pseudo column and the LPAD function to indent employees based on their level.

```
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
```

## Hierarchical Query Output Levels

```
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
```

As you can see in the result on the right, each row is indented by two underscores per level.

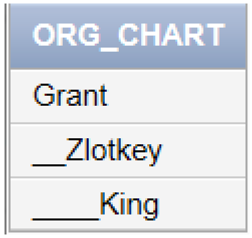


```
ORG_CHART
King
__Kochhar
____Whalen
____Higgins
______Gietz
__De Haan
____Hunold
______Ernst
______Lorentz
__Mourgos
____Rajs
____Davies
____Matos
____Vargas
__Zlotkey
____Abel
____Taylor
____Grant
__Hartstein
____Fay
```

## Bottom Up Hierarchical Query

```
SELECT LPAD(last_name, LENGTH(last_name) +
(LEVEL*2) -2, '_') AS ORG_CHART
FROM employees
START WITH last_name = 'Grant'
CONNECT BY employee_id = PRIOR manager_id
```

As you can see in the result on the right, this example shows how to create a Bottom Up Hierarchical Query by moving the keyword PRIOR to after the equals sign, and using 'Grant' in the START WITH clause.



| ORG_CHART |
|---|
| Grant |
| __Zlotkey |
| ____King |

## Hierarchical Queries Pruning

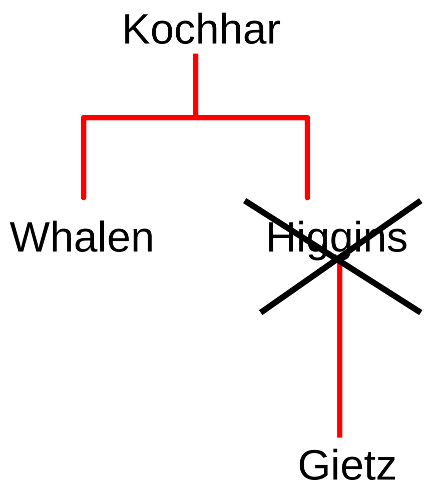
Pruning branches from the tree can be done using either the WHERE clause or the CONNECT BY PRIOR clause.

If the WHERE clause is used, only the row named in the statement is excluded; if the CONNECT BY PRIOR clause is used, the entire branch is excluded.

## Hierarchical Queries Pruning (cont.)

For example, if you want to exclude a single row from your result, you would use the WHERE clause to exclude that row; however, in the result, it would then look like Gietz worked directly for Kochhar, which he does not.

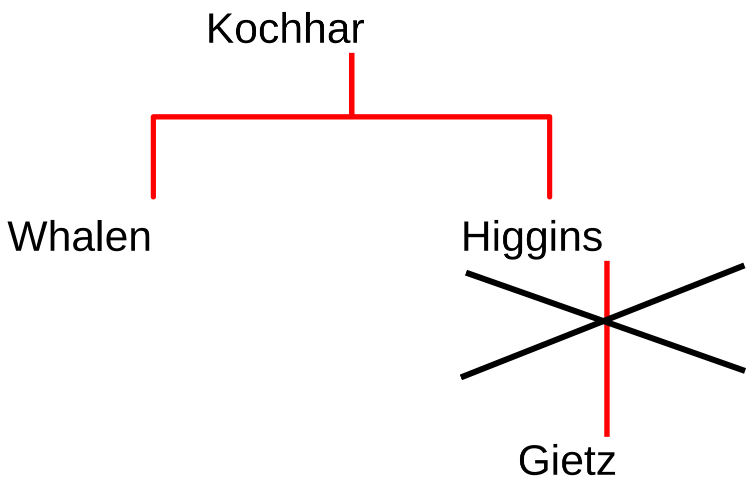
```
SELECT last_name
FROM employees
WHERE last_name != 'Higgins‘
START WITH last_name = ‘Kochhar’
CONNECT BY PRIOR  employee_id = manager_id
```



## Hierarchical Queries Pruning (cont.)

If, however, you wanted to exclude one row and all the rows below that one, you should make the exclusion part of the CONNECT BY statement.  In this example that excludes Higgins, we are also excluding Gietz in the result.

```
SELECT last_name
FROM employees
START  WITH last_name =
‘Kochhar’
CONNECT BY PRIOR  employee_id =
manager_id
AND last_name != 'Higgins
```



## Terminology

Key terms used in this lesson included:

- Connect By prior
- Hierarchical queries
- Level
- Self join
- Start with

## Summary

In this lesson, you should have learned how to:

- Construct and execute a SELECT statement to join a table to itself using a self-join
- Interpret the concept of a hierarchical query
- Create a tree-structured report
- Format hierarchical data
- Exclude branches from the tree structure