UNIVERSIDAD SERGIO ARBOLEDA

Self-Joins and Hierarchical Queries

JOINs

Objectives

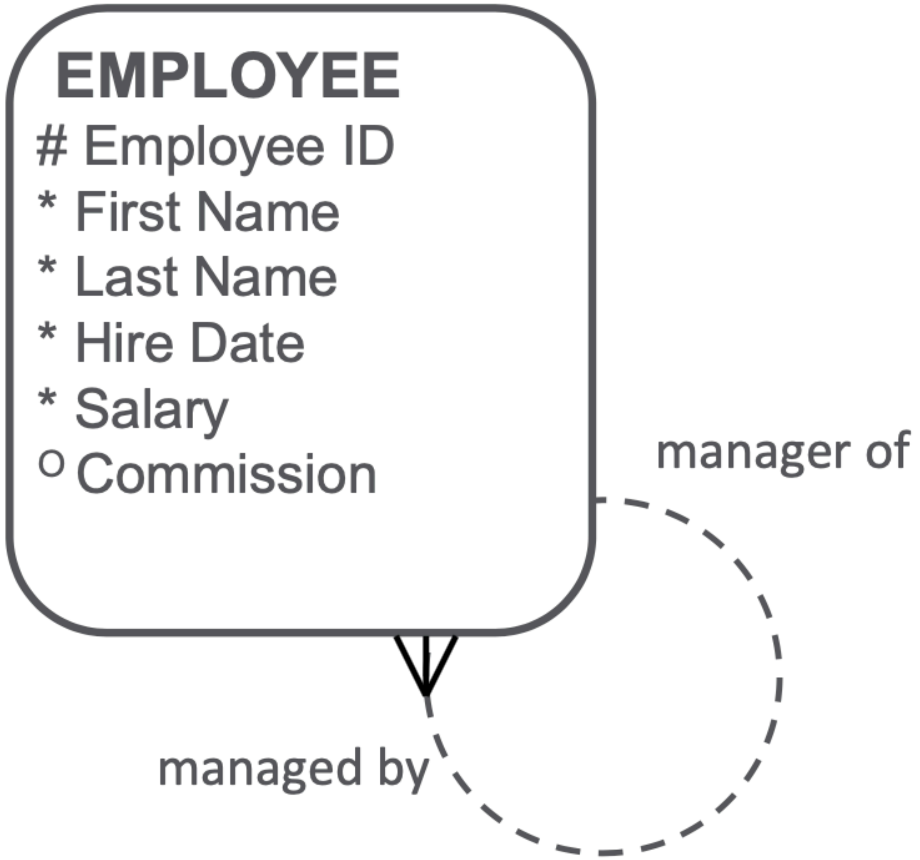
- Construct and execute a SELECT statement to join a table to itself using a self-join
- Interpret the concept of a hierarchical query
- Create a tree-structured report
- Format hierarchical data
- Exclude branches from the tree structure

Purpose

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

For example, an employee can also be a manager.

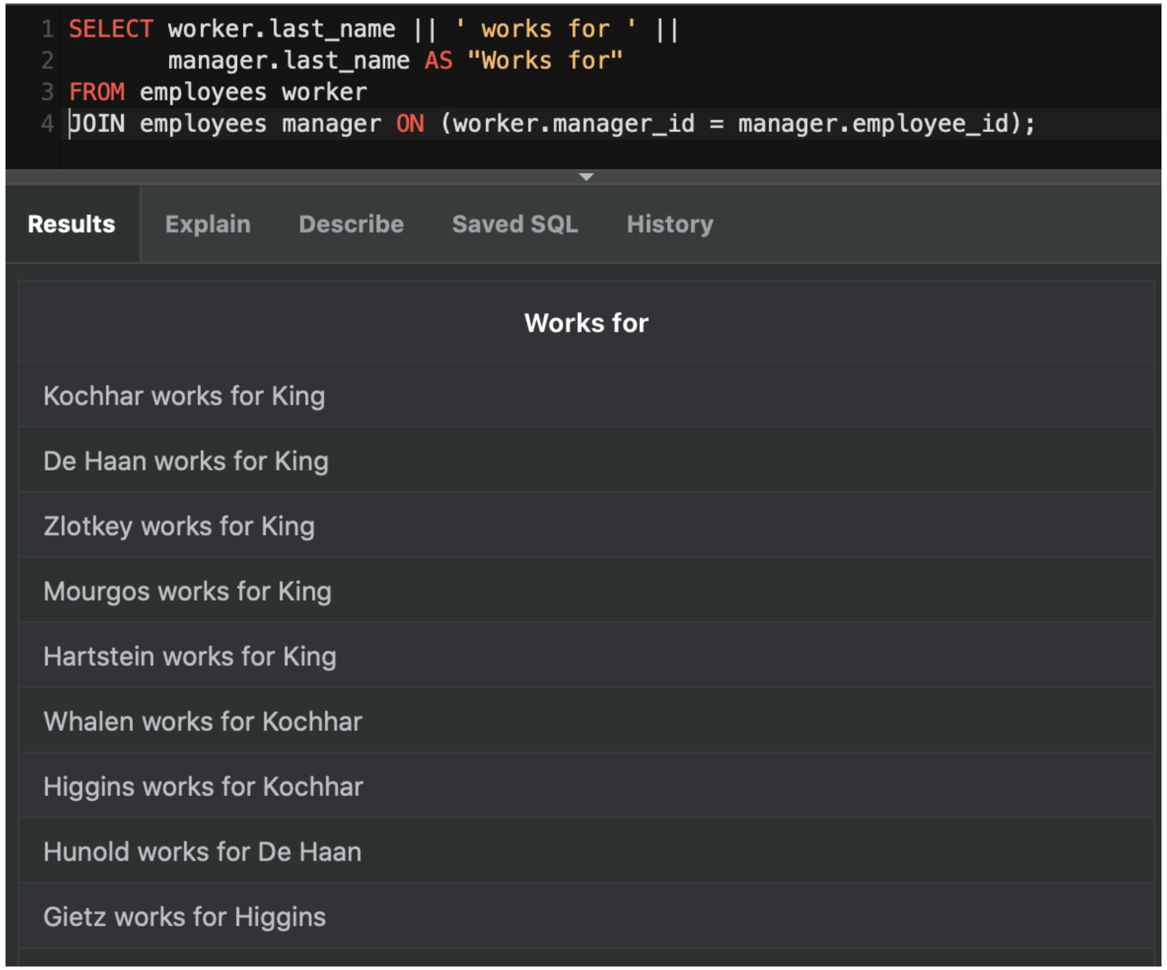
We showed this using the recursive or "pig's ear" relationship.



Purpose

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

A self-join is use to join a table to itself as if it was two tables.



```
SELECT worker.last_name || ' works for ' ||
       manager.last_name AS "Works for"
FROM employees worker
JOIN employees manager ON (worker.manager_id = manager.employee_id);
```

```
Works for
Kochhar works for King
De Haan works for King
Zlotkey works for King
Mourgos works for King
Hartstein works for King
Whalen works for Kochhar
Higgins works for Kochhar
Hunold works for De Haan
Gietz works for Higgins
```

SELF-JOIN

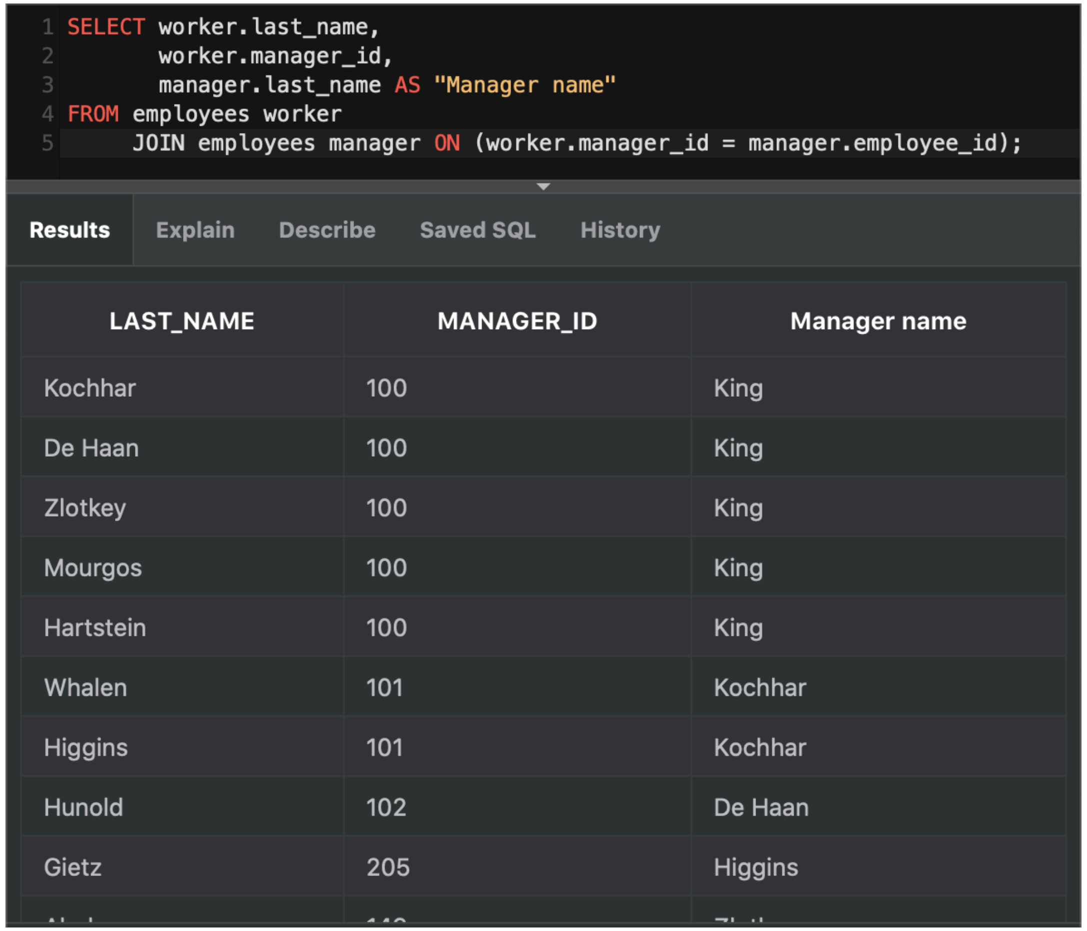
- To join a table to itself,
- the table is given two names or aliases.
- **This will make the database "think" that there are two tables.**

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

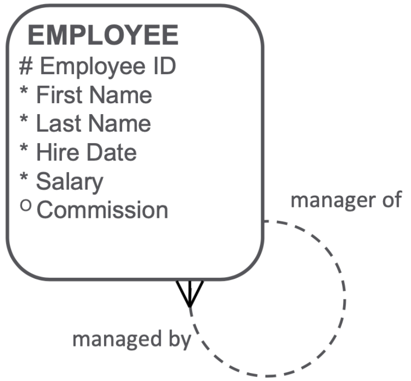
EMPLOYEES (manager)

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



```
SELECT worker.last_name,
       worker.manager_id,
       manager.last_name AS "Manager name"
FROM employees worker
     JOIN employees manager ON (worker.manager_id = manager.employee_id);
```

LAST_NAME	MANAGER_ID	Manager name
Kochhar	100	King
De Haan	100	King
Zlotkey	100	King
Mourgos	100	King
Hartstein	100	King
Whalen	101	Kochhar
Higgins	101	Kochhar
Hunold	102	De Haan
Gietz	205	Higgins



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 type of 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, you can retrieve data based on a natural hierarchical relationship between rows in a table.

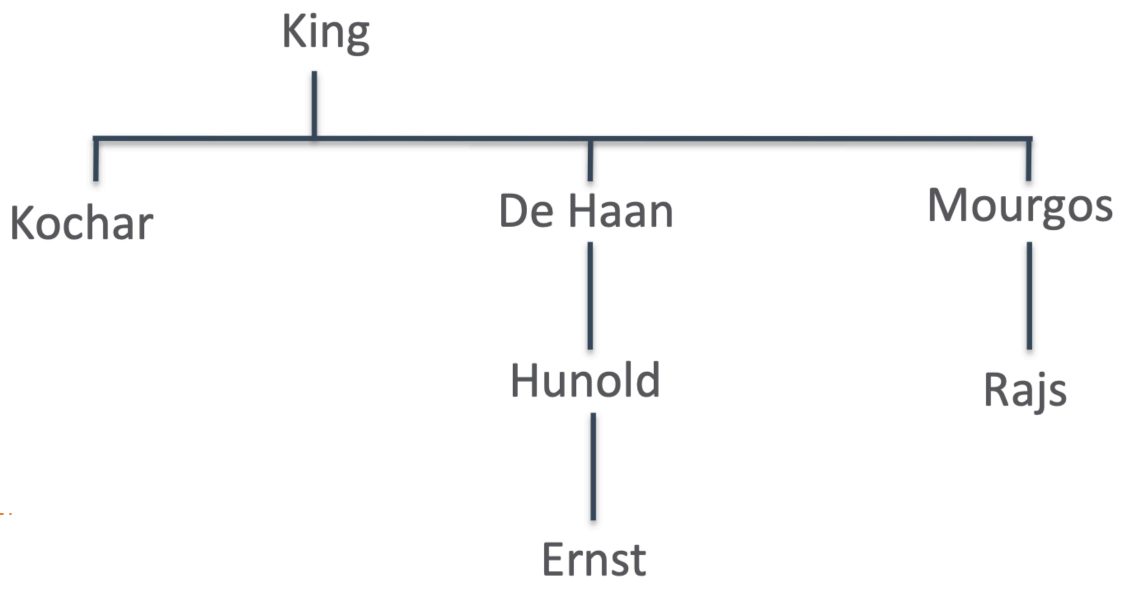
A relational database does not store records in a hierarchical way.

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

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



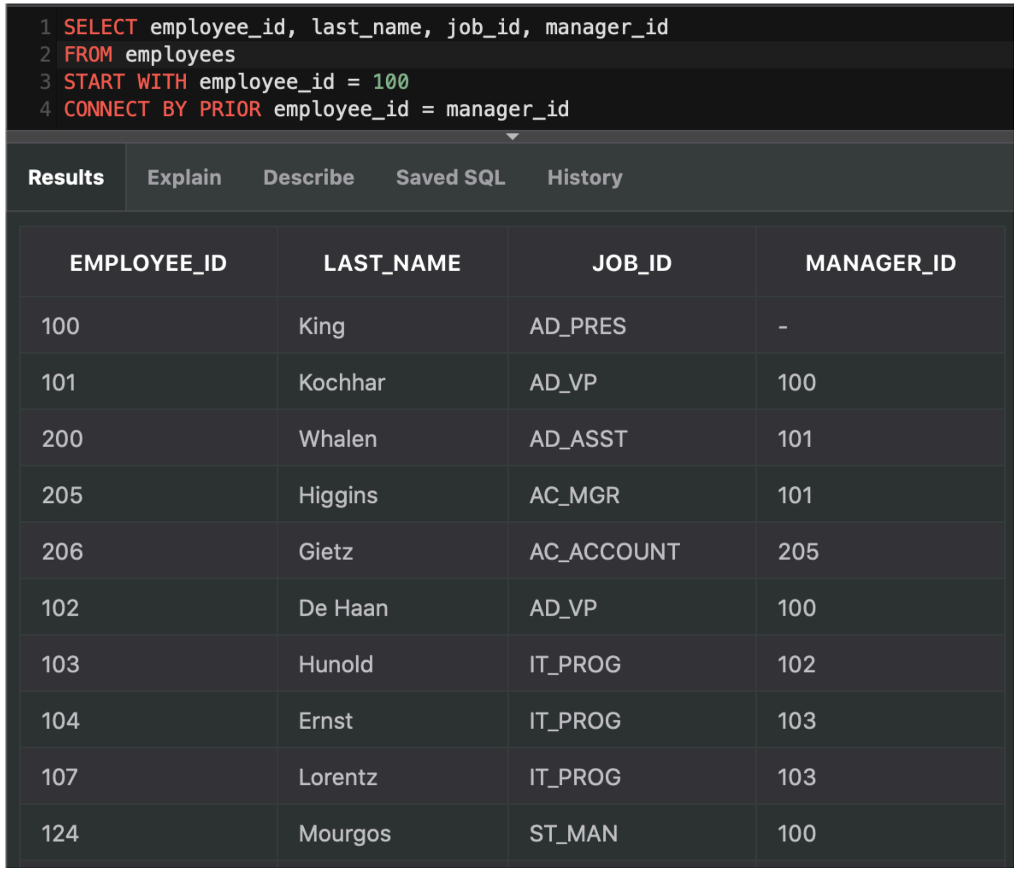
Hierarchical Queries Keywords

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

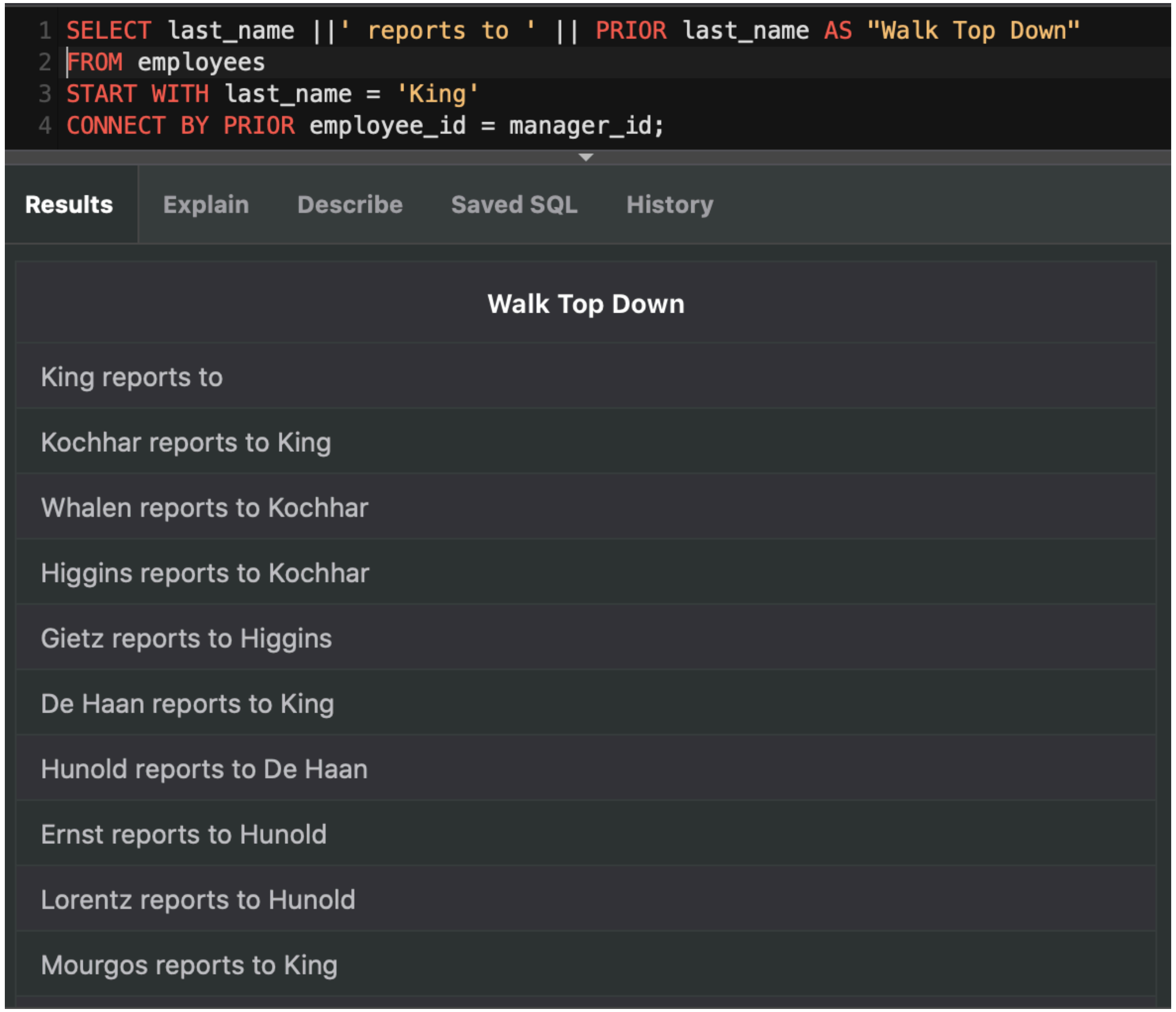
LEVEL
- specifies how many branches deep the tree will traverse.



```
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	-
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	Hunold	IT_PROG	102
104	Ernst	IT_PROG	103
107	Lorentz	IT_PROG	103
124	Mourgos	ST_MAN	100

Hierarchical Queries Another Example

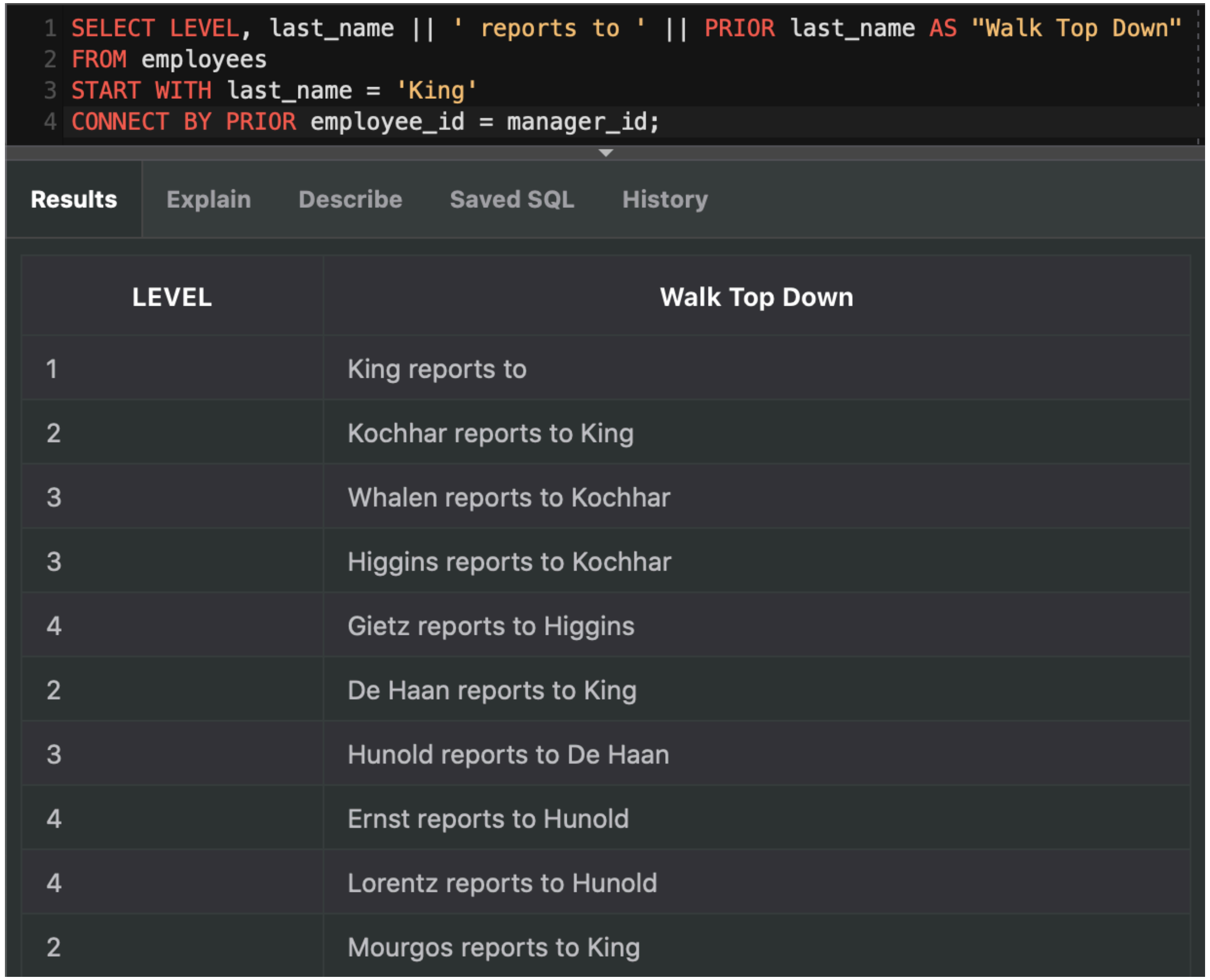


```sql
SELECT last_name ||' reports to ' || PRIOR last_name AS "Walk Top Down"
FROM employees
START WITH last_name = 'King'
CONNECT BY PRIOR employee_id = manager_id;
```

```
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
Lorentz reports to Hunold
Mourgos reports to King
```

Hierarchical Queries Level Example

- 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.

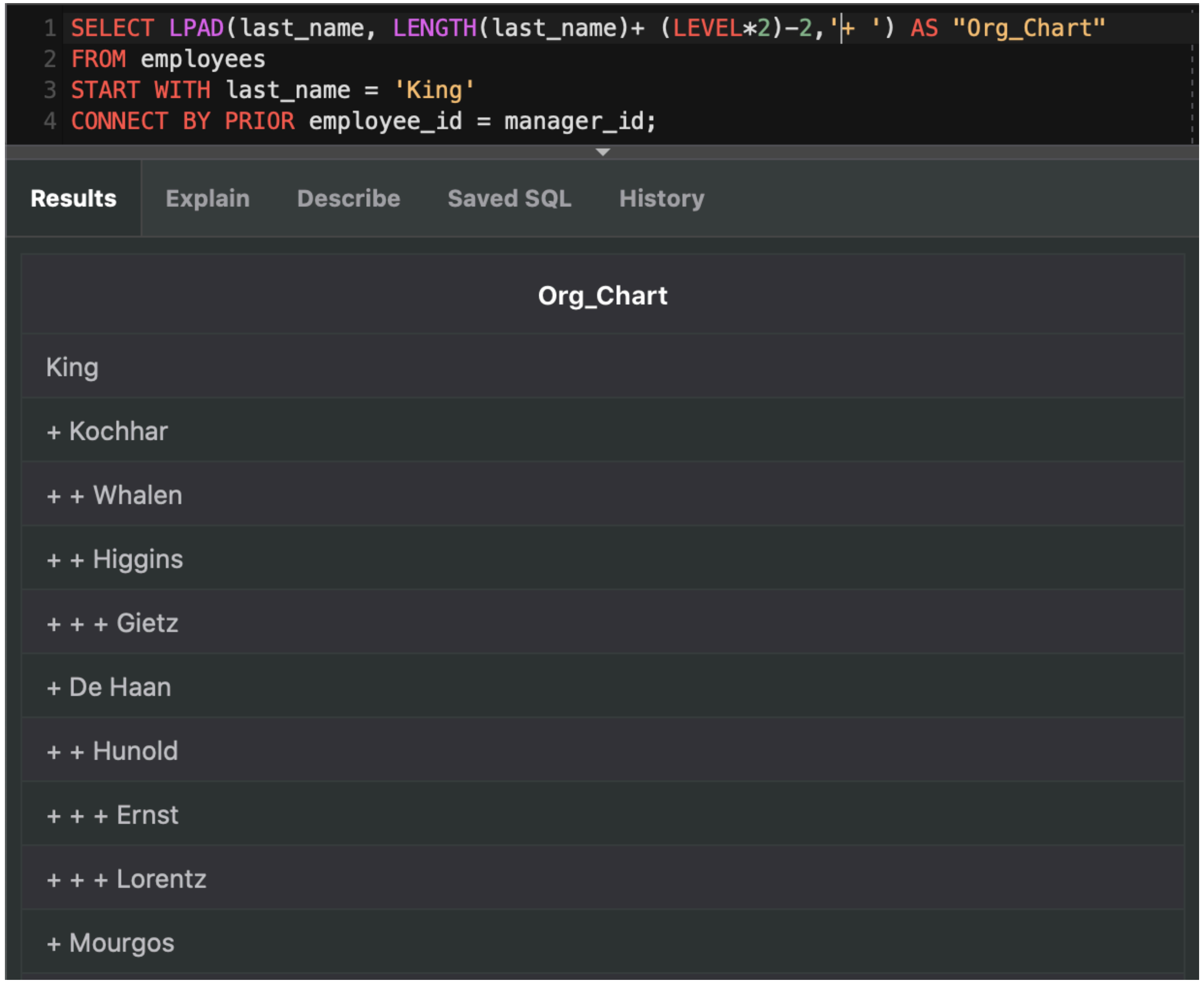


```
SELECT LEVEL, last_name || ' reports to ' || PRIOR last_name AS "Walk Top Down"
FROM employees
START WITH last_name = 'King'
CONNECT BY PRIOR employee_id = manager_id;
```

LEVEL	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
4	Lorentz reports to Hunold
2	Mourgos reports to King

Hierarchical Query Report

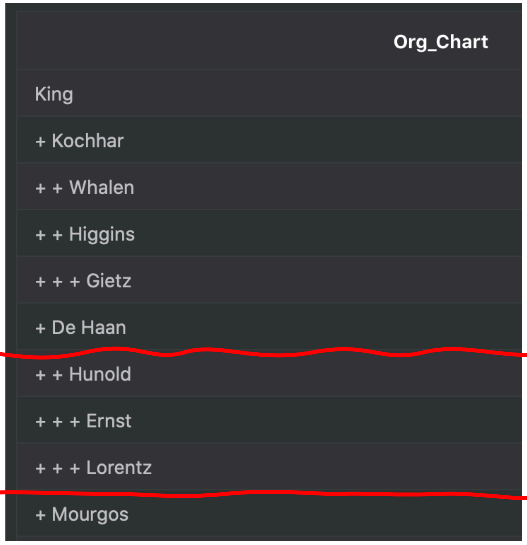
- If you wanted to create a report displaying
  - company management levels,
  - **beginning with the highest level and**
  - indenting each of the following levels,
  - then this would be easy to do using the LEVEL pseudo column and the LPAD function to indent employees based on their level.



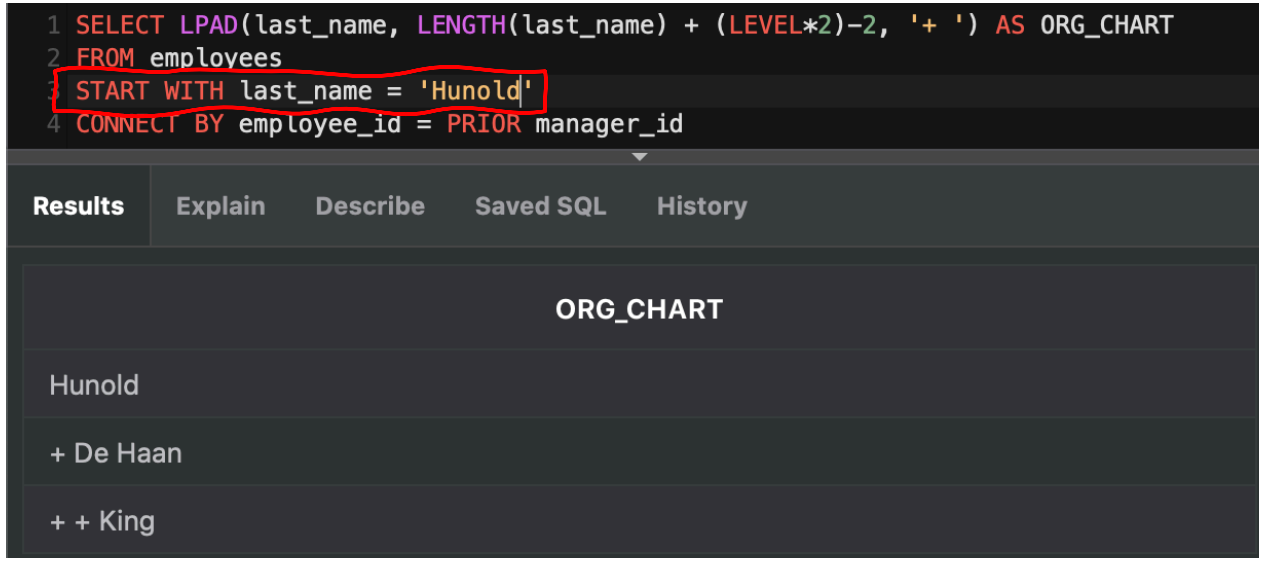
```
SELECT LPAD(last_name, LENGTH(last_name)+ (LEVEL*2)-2,'+ ') AS "Org_Chart"
FROM employees
START WITH last_name = 'King'
CONNECT BY PRIOR employee_id = manager_id;
```

```
Org_Chart
King
+ Kochhar
+ + Whalen
+ + Higgins
+ + + Gietz
+ De Haan
+ + Hunold
+ + + Ernst
+ + + Lorentz
+ Mourgos
```

- This example shows how to create a **Bottom Up Hierarchical Query** by **<u>moving the keyword PRIOR</u>** to after the equals sign, and using 'Hunold' in the START WITH clause.



Org_Chart
King
+ Kochhar
+ + Whalen
+ + Higgins
+ + + Gietz
+ De Haan
+ + Hunold
+ + + Ernst
+ + + Lorentz
+ Mourgos



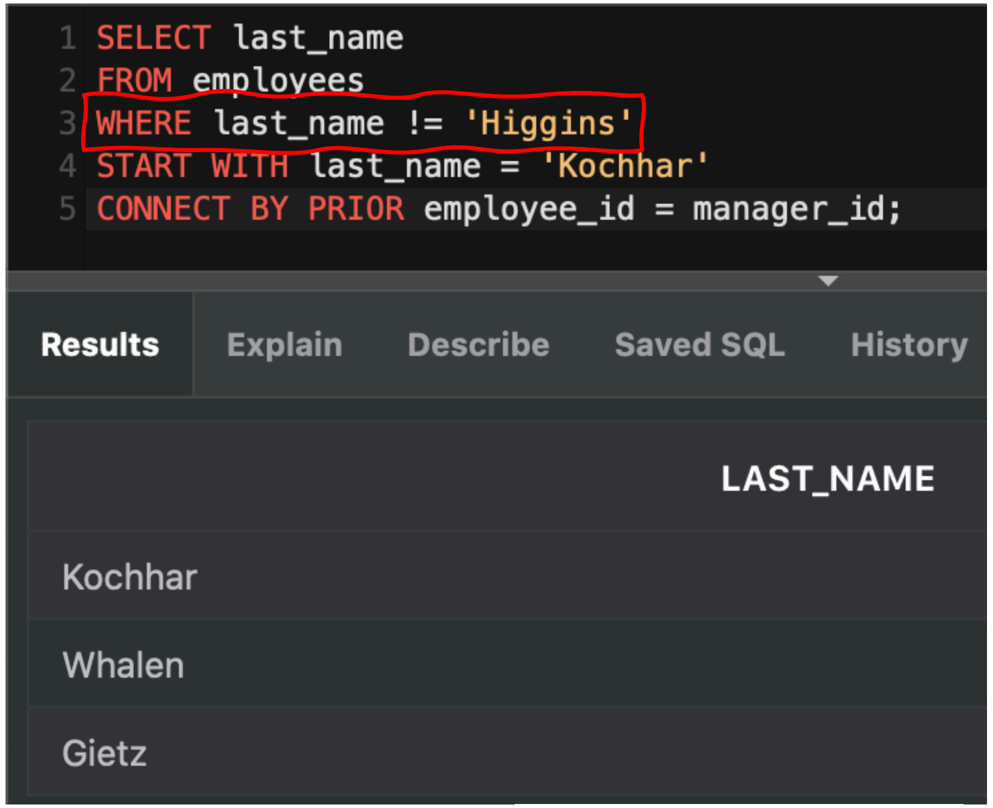
```
SELECT LPAD(last_name, LENGTH(last_name) + (LEVEL*2)-2, '+ ') AS ORG_CHART
FROM employees
START WITH last_name = 'Hunold'
CONNECT BY employee_id = PRIOR manager_id
```

ORG_CHART
Hunold
+ De Haan
+ + King

Hierarchical Queries Pruning

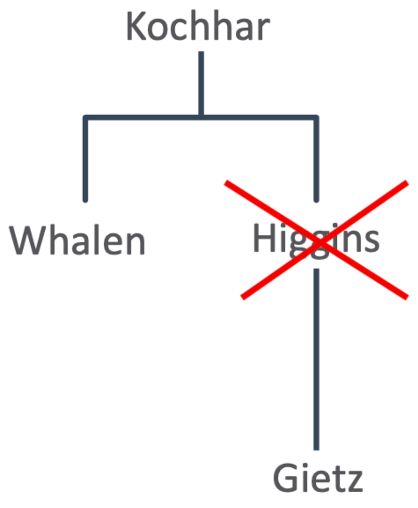
- Pruning branches from the tree can be done using either
  - the WHERE clause or
  - the CONNECT BY PRIOR clause.
- If the WHERE clause is used,
  - only the row named in the statement is excluded;
- if the CONNECT BY PRIOR clause is used,
  - the entire branch is excluded.

- 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.



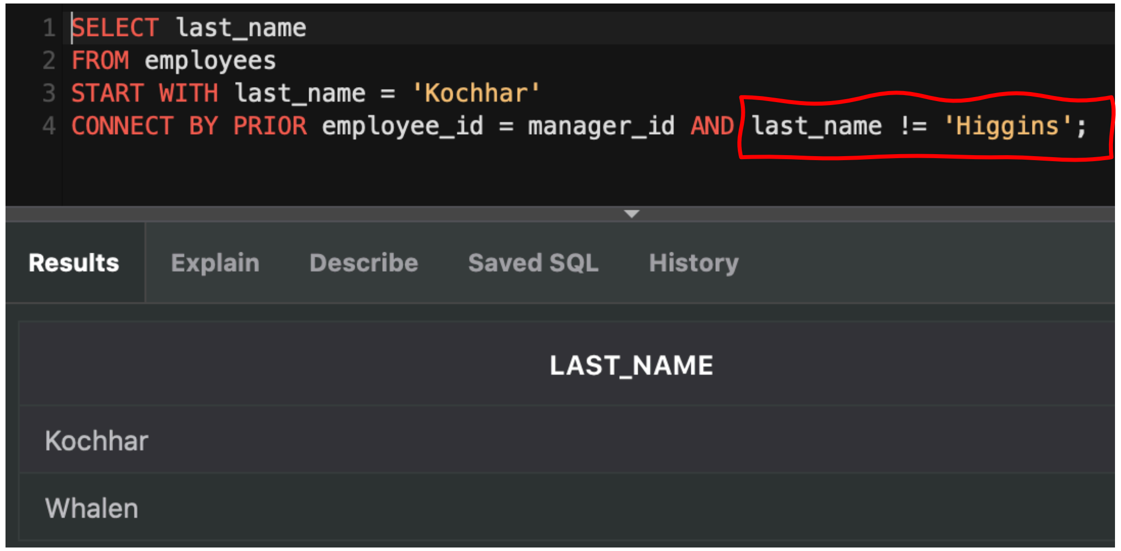
```
SELECT last_name
FROM employees
WHERE last_name != 'Higgins'
START WITH last_name = 'Kochhar'
CONNECT BY PRIOR employee_id = manager_id;
```

```
LAST_NAME
Kochhar
Whalen
Gietz
```



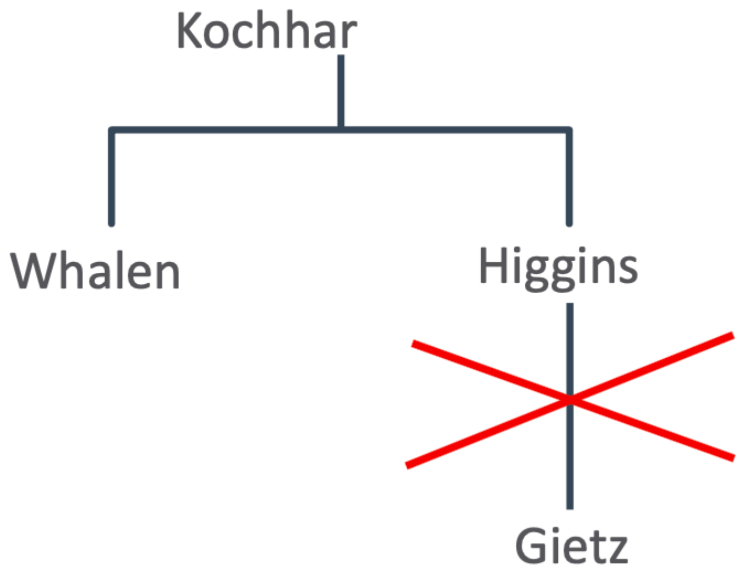
Hierarchical Queries Pruning

- 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';
```

LAST_NAME
Kochhar
Whalen


Kochhar
Whalen
Higgins
Gietz