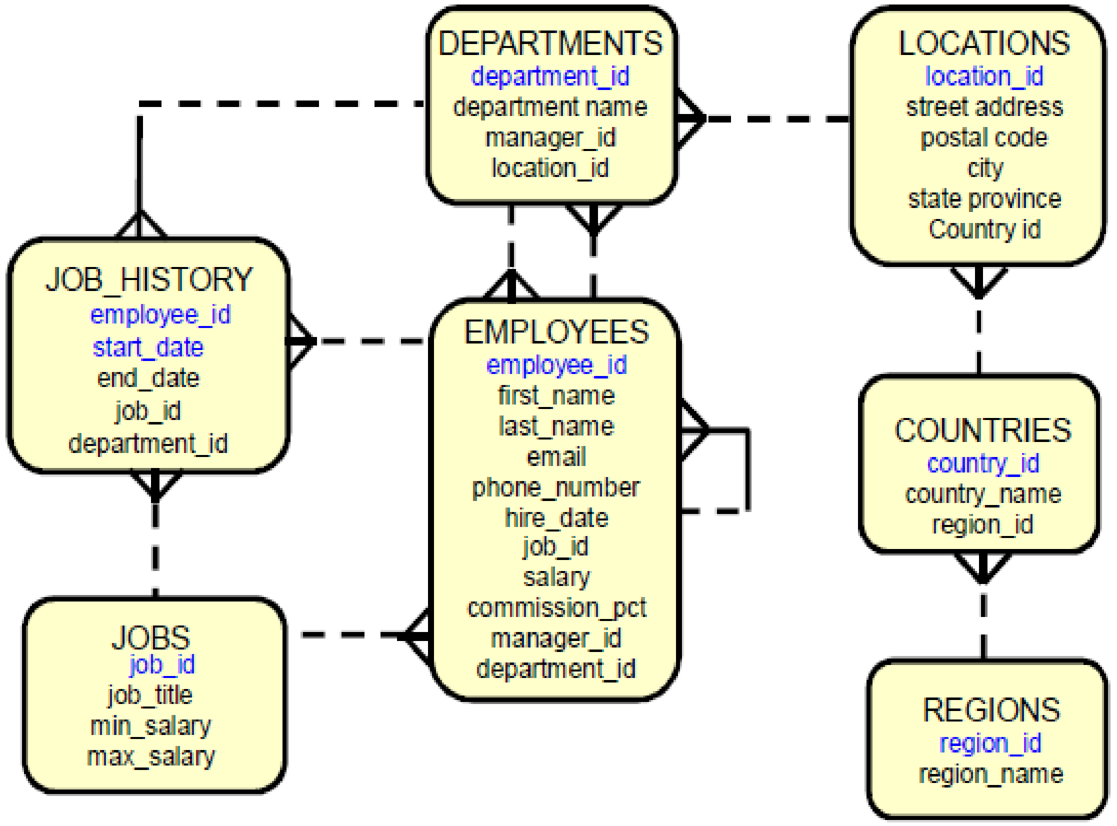
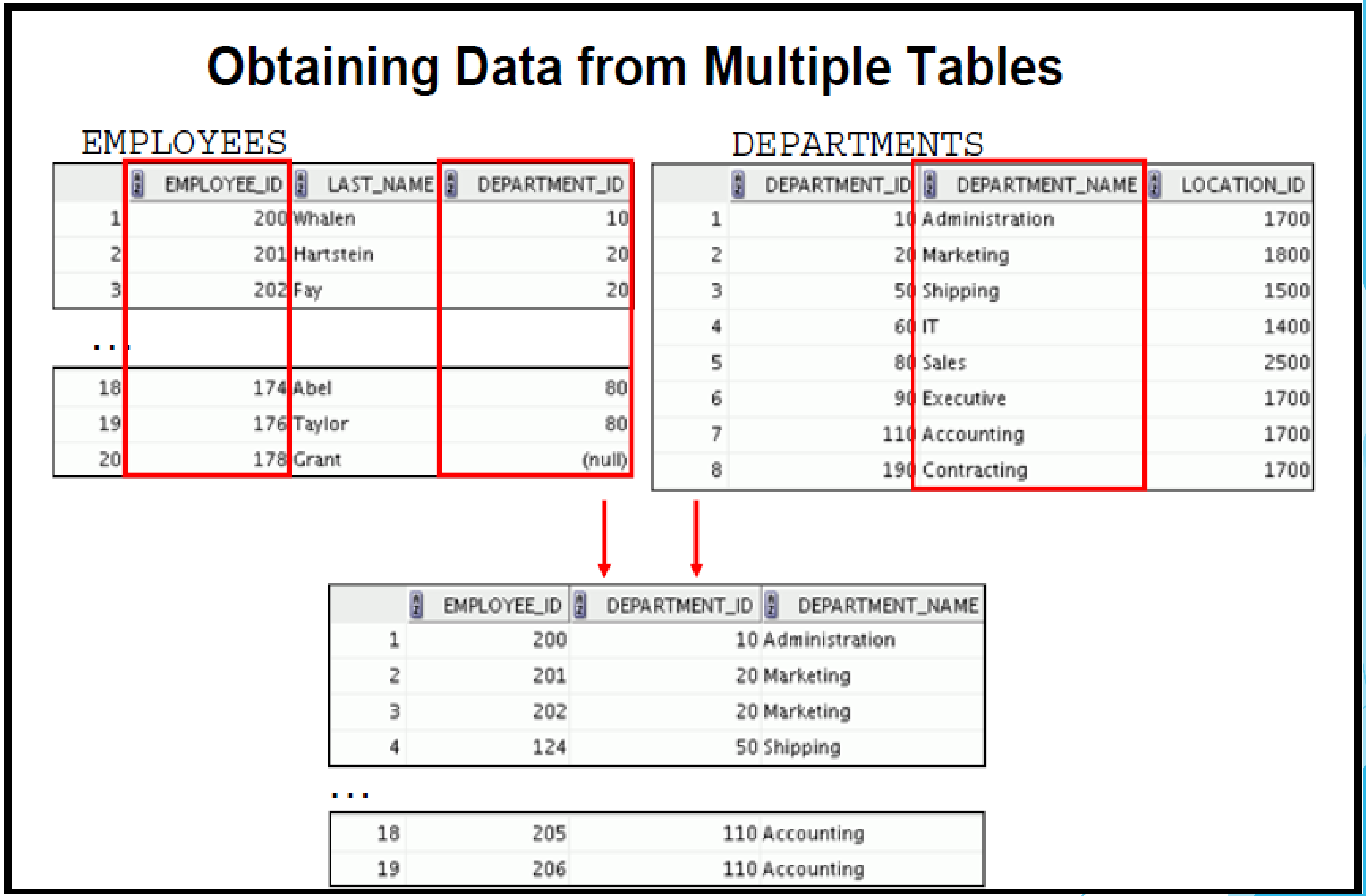
# Displaying Data from multiple tables using joins

## Human Resources (HR) Schema





## Obtaining Data from Multiple Tables

EMPLOYEES

| | EMPLOYEE_ID | LAST_NAME | DEPARTMENT_ID |
|---|---|---|---|
| 1 | 200 | Whalen | 10 |
| 2 | 201 | Hartstein | 20 |
| 3 | 202 | Fay | 20 |

. . .

| | | | |
|---|---|---|---|
| 18 | 174 | Abel | 80 |
| 19 | 176 | Taylor | 80 |
| 20 | 178 | Grant | (null) |

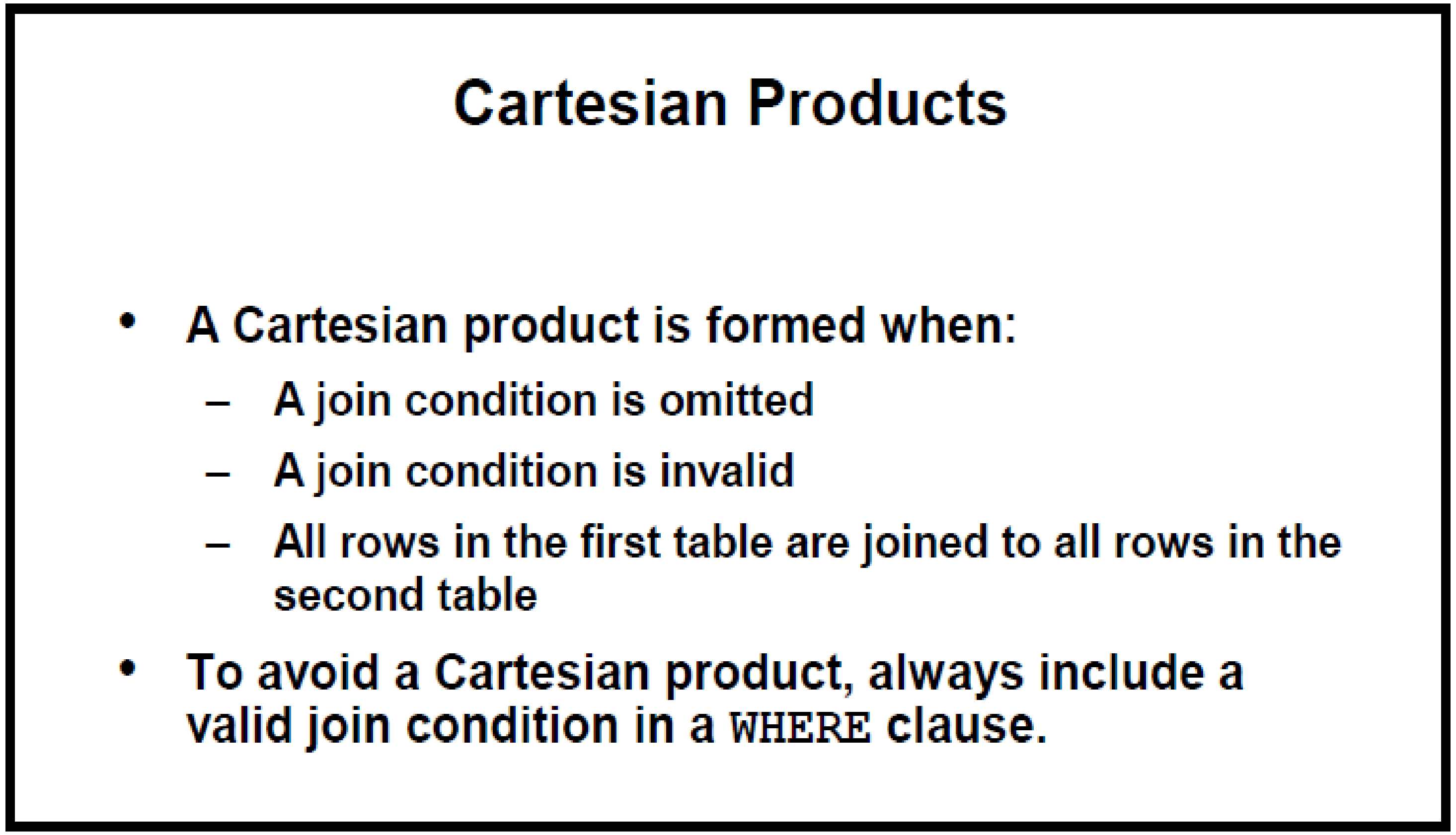
DEPARTMENTS

| | DEPARTMENT_ID | DEPARTMENT_NAME | LOCATION_ID |
|---|---|---|---|
| 1 | 10 | Administration | 1700 |
| 2 | 20 | Marketing | 1800 |
| 3 | 50 | Shipping | 1500 |
| 4 | 60 | IT | 1400 |
| 5 | 80 | Sales | 2500 |
| 6 | 90 | Executive | 1700 |
| 7 | 110 | Accounting | 1700 |
| 8 | 190 | Contracting | 1700 |

| | EMPLOYEE_ID | DEPARTMENT_ID | DEPARTMENT_NAME |
|---|---|---|---|
| 1 | 200 | 10 | Administration |
| 2 | 201 | 20 | Marketing |
| 3 | 202 | 20 | Marketing |
| 4 | 124 | 50 | Shipping |

. . .

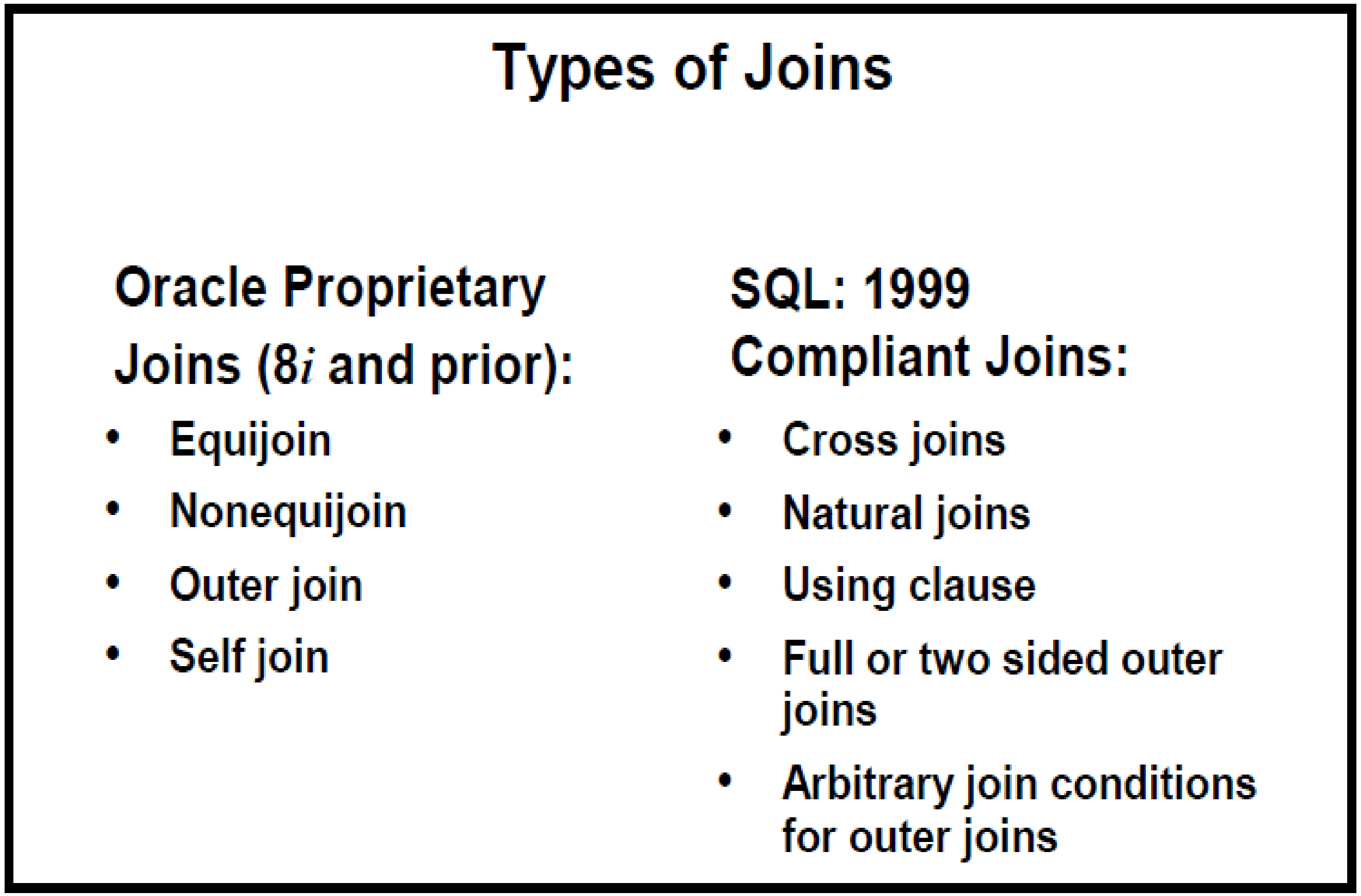
| | | | |
|---|---|---|---|
| 18 | 205 | 110 | Accounting |
| 19 | 206 | 110 | Accounting |



## Cartesian Products

- A Cartesian product is formed when:
  - A join condition is omitted
  - A join condition is invalid
  - All rows in the first table are joined to all rows in the second table
- To avoid a Cartesian product, always include a valid join condition in a `WHERE` clause.

```
SELECT EMPLOYEES.EMPLOYEE_ID ,
EMPLOYEES.FIRST_NAME,
EMPLOYEES.DEPARTMENT_ID,
DEPARTMENTS.DEPARTMENT_NAME
FROM EMPLOYEES,
DEPARTMENTS
ORDER BY EMPLOYEE_ID;
```



# Types of Joins

**Oracle Proprietary Joins (8*i* and prior):**

- Equijoin
- Nonequijoin
- Outer join
- Self join

**SQL: 1999 Compliant Joins:**

- Cross joins
- Natural joins
- Using clause
- Full or two sided outer joins
- Arbitrary join conditions for outer joins

## Joining Tables Using Oracle Syntax

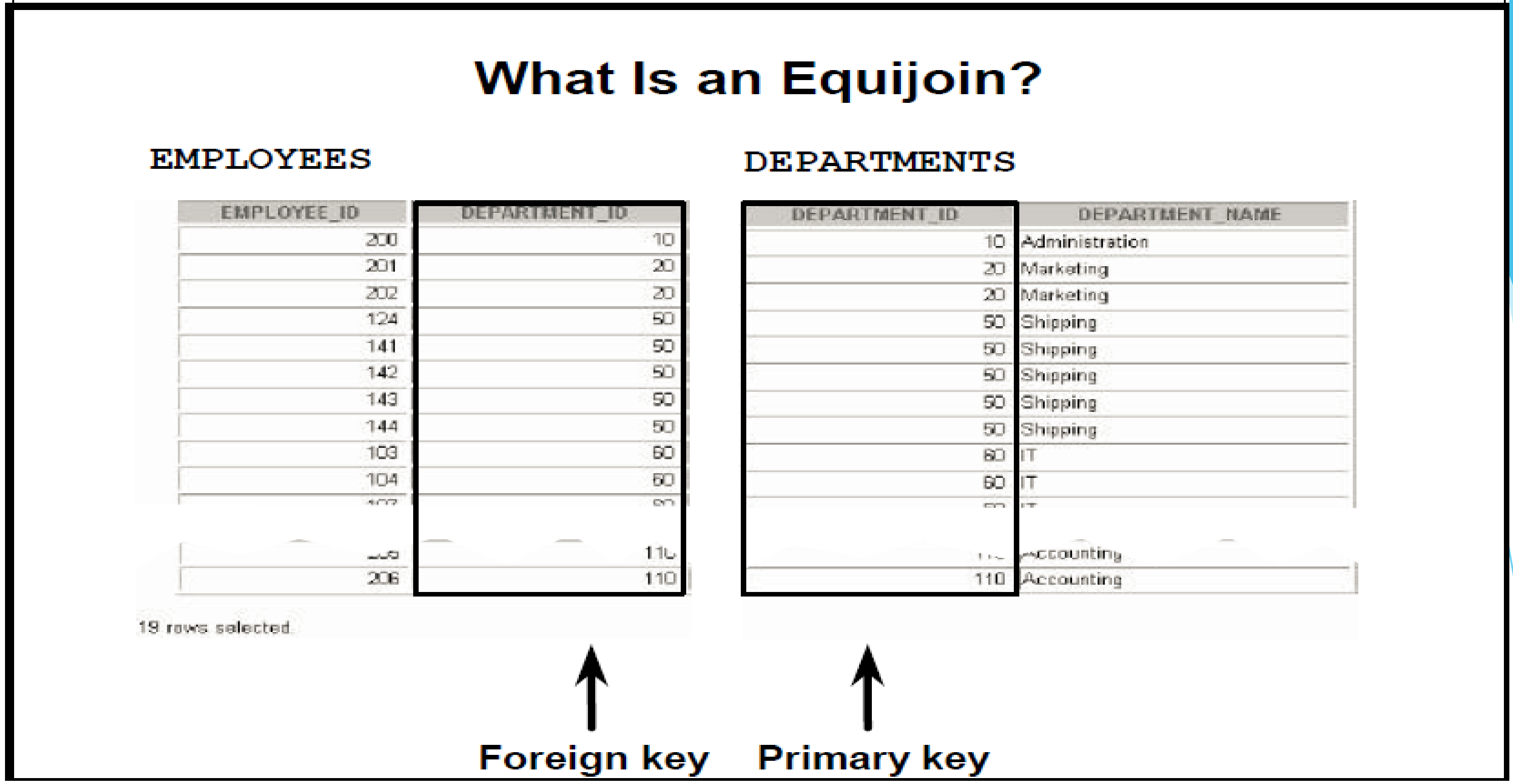
Use a join to query data from more than one table.

```
SELECT    table1.column, table2.column
FROM      table1, table2
WHERE     table1.column1 = table2.column2;
```

- Write the join condition in the `WHERE` clause.
- Prefix the column name with the table name when the same column name appears in more than one table.

**Guidelines**

- When writing a `SELECT` statement that joins tables, precede the column name with the table name for clarity and to enhance database access.
- If the same column name appears in more than one table, the column name must be prefixed with the table name.
- To join *n* tables together, you need a minimum of `n-1` join conditions. For example, to join four tables, a minimum of three joins is required. This rule may not apply if your table has a concatenated primary key, in which case more than one column is required to uniquely identify each row.



## What Is an Equijoin?

**EMPLOYEES**

| EMPLOYEE_ID | DEPARTMENT_ID |
|---|---|
| 200 | 10 |
| 201 | 20 |
| 202 | 20 |
| 124 | 50 |
| 141 | 50 |
| 142 | 50 |
| 143 | 50 |
| 144 | 50 |
| 103 | 60 |
| 104 | 60 |
| 107 | 60 |
| 205 | 110 |
| 206 | 110 |

19 rows selected.

**DEPARTMENTS**

| DEPARTMENT_ID | DEPARTMENT_NAME |
|---|---|
| 10 | Administration |
| 20 | Marketing |
| 20 | Marketing |
| 50 | Shipping |
| 50 | Shipping |
| 50 | Shipping |
| 50 | Shipping |
| 50 | Shipping |
| 60 | IT |
| 60 | IT |
| 60 | IT |
| 110 | Accounting |
| 110 | Accounting |

**Equijoins**

To determine an employee's department name, you compare the value in the DEPARTMENT_ID column in the EMPLOYEES table with the DEPARTMENT_ID values in the DEPARTMENTS table. The relationship between the EMPLOYEES and DEPARTMENTS tables is an *equijoin*, that is, values in the DEPARTMENT_ID column on both tables must be equal. Frequently, this type of join involves primary and foreign key complements.

**Note:** Equijoins are also called *simple joins* or *inner joins*.

## Qualifying Ambiguous Column Names

- Use table prefixes to qualify column names that are in multiple tables.
- Improve performance by using table prefixes.
- Distinguish columns that have identical names but reside in different tables by using column aliases.

```
SELECT
EMPLOYEES.EMPLOYEE_ID ,
EMPLOYEES.FIRST_NAME,
EMPLOYEES.DEPARTMENT_ID,
DEPARTMENTS.DEPARTMENT_NAME
FROM

EMPLOYEES,
DEPARTMENTS
WHERE EMPLOYEES.DEPARTMENT_ID=DEPARTMENTS.DEPARTMENT_ID
ORDER BY EMPLOYEE_ID;
```

## Using Table Aliases

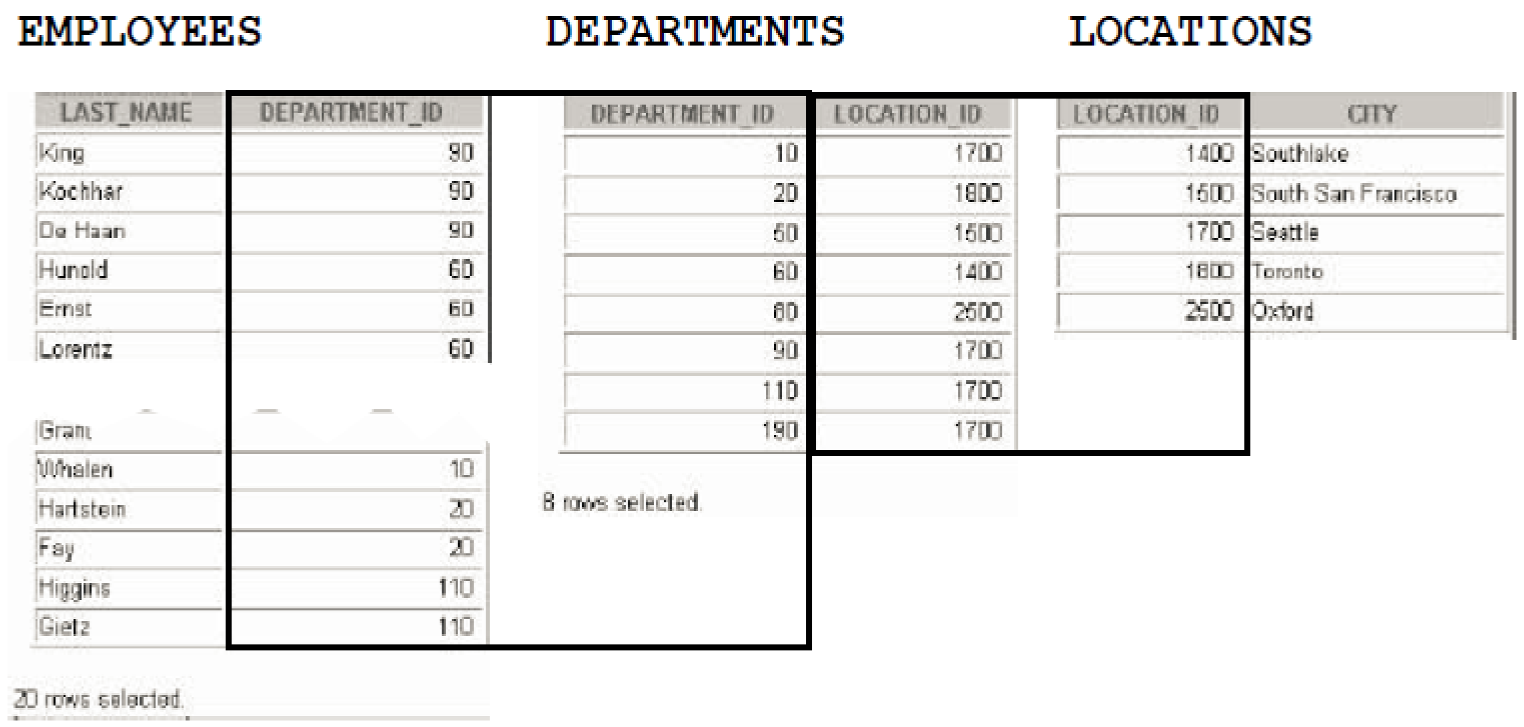
- Simplify queries by using table aliases
- Improve performance by using table prefixes

```
SELECT e.employee_id, e.last_name, e.department_id,
       d.department_id, d.location_id
FROM   employees e, departments d
WHERE  e.department_id = d.department_id;
```

**Guidelines**

- Table aliases can be up to 30 characters in length, but the shorter they are the better.
- If a table alias is used for a particular table name in the `FROM` clause, then that table alias must be substituted for the table name throughout the `SELECT` statement.
- Table aliases should be meaningful.
- The table alias is valid only for the current `SELECT` statement.

## Joining More than Two Tables



EMPLOYEES

| LAST_NAME | DEPARTMENT_ID |
|---|---|
| King | 90 |
| Kochhar | 90 |
| De Haan | 90 |
| Hunold | 60 |
| Ernst | 60 |
| Lorentz | 60 |
| Grant | |
| Whalen | 10 |
| Hartstein | 20 |
| Fay | 20 |
| Higgins | 110 |
| Gietz | 110 |

20 rows selected.

DEPARTMENTS

| DEPARTMENT_ID | LOCATION_ID |
|---|---|
| 10 | 1700 |
| 20 | 1800 |
| 50 | 1500 |
| 60 | 1400 |
| 80 | 2500 |
| 90 | 1700 |
| 110 | 1700 |
| 190 | 1700 |

8 rows selected.

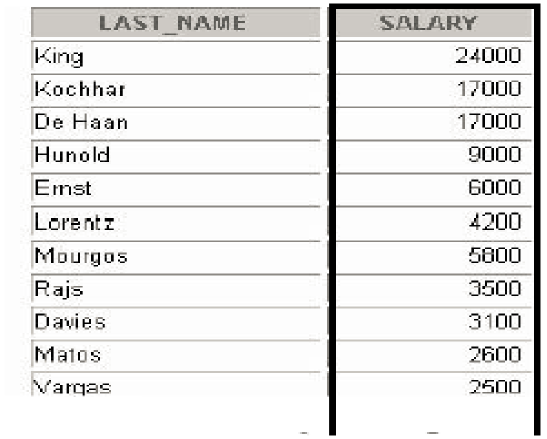
LOCATIONS

| LOCATION_ID | CITY |
|---|---|
| 1400 | Southlake |
| 1500 | South San Francisco |
| 1700 | Seattle |
| 1800 | Toronto |
| 2500 | Oxford |

**To join *n* tables together, you need a minimum of *n*-1 join conditions. For example, to join three tables, a minimum of two joins is required.**

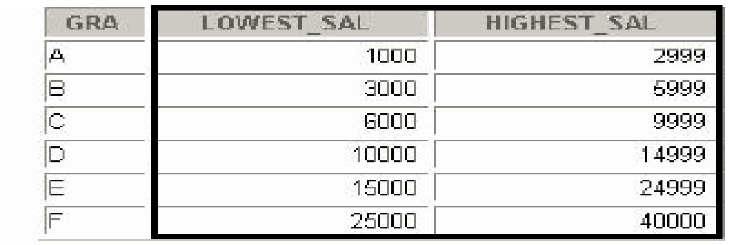
## Nonequijoins

EMPLOYEES



| LAST_NAME | SALARY |
|---|---|
| King | 24000 |
| Kochhar | 17000 |
| De Haan | 17000 |
| Hunold | 9000 |
| Ernst | 6000 |
| Lorentz | 4200 |
| Mourgos | 5800 |
| Rajs | 3500 |
| Davies | 3100 |
| Matos | 2600 |
| Vargas | 2500 |

JOB_GRADES



| GRA | LOWEST_SAL | HIGHEST_SAL |
|---|---|---|
| A | 1000 | 2999 |
| B | 3000 | 5999 |
| C | 6000 | 9999 |
| D | 10000 | 14999 |
| E | 15000 | 24999 |
| F | 25000 | 40000 |

← **Salary in the EMPLOYEES table must be between lowest salary and highest salary in the JOB_GRADES table.**

```
SELECT e.last_name, e.salary, j.grade_level
FROM   employees e, job_grades j
WHERE  e.salary BETWEEN j.lowest_sal AND j.highest_sal;
```

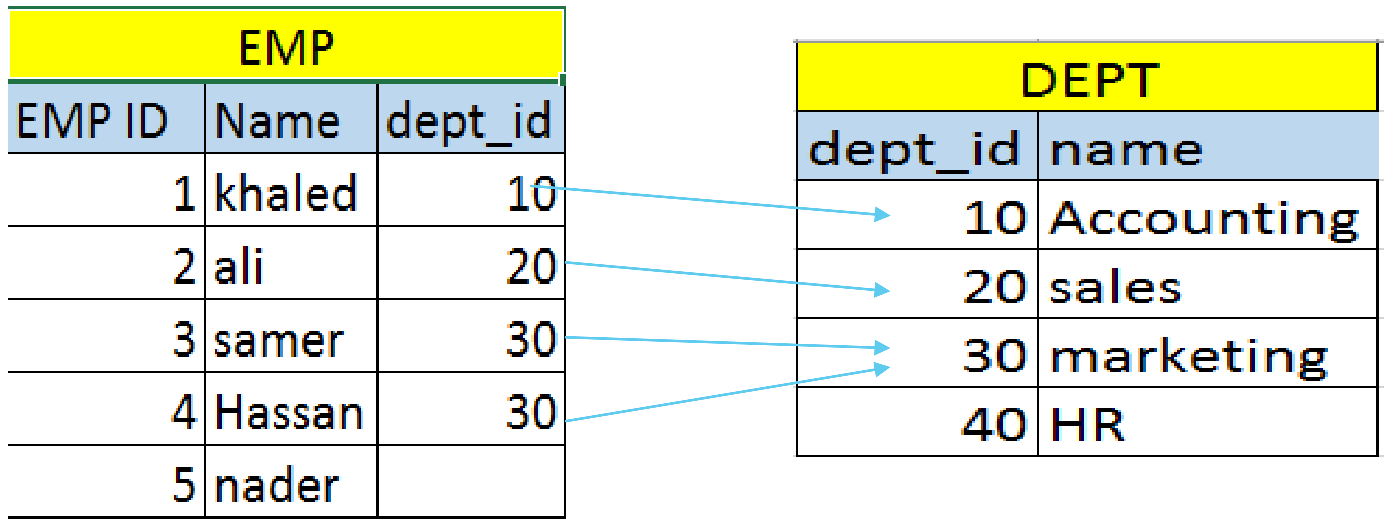
### Nonequijoins

A nonequijoin is a join condition containing something other than an equality operator.

The relationship between the EMPLOYEES table and the JOB_GRADES table has an example of a nonequijoin. A relationship between the two tables is that the SALARY column in the EMPLOYEES table must be between the values in the LOWEST_SALARY and HIGHEST_SALARY columns of the JOB_GRADES table. The relationship is obtained using an operator other than equals (=).

# Outer Join

Main table



| EMP | | |
|---|---|---|
| EMP ID | Name | dept_id |
| 1 | khaled | 10 |
| 2 | ali | 20 |
| 3 | samer | 30 |
| 4 | Hassan | 30 |
| 5 | nader | |

| DEPT | |
|---|---|
| dept_id | name |
| 10 | Accounting |
| 20 | sales |
| 30 | marketing |
| 40 | HR |

- Emp ID 5 has no department

So if you used natural join emp.dept_id=dept.dept_id, then he will not appear

- So we have to use outer join (+), always place it on the side that have the missing data

  where emp.dept_id=dept.dept_id(+)

- If a FK in table1 has null values and it is the main table that you want to display the data from, then the outer join should be table2 side which has the reference from table1

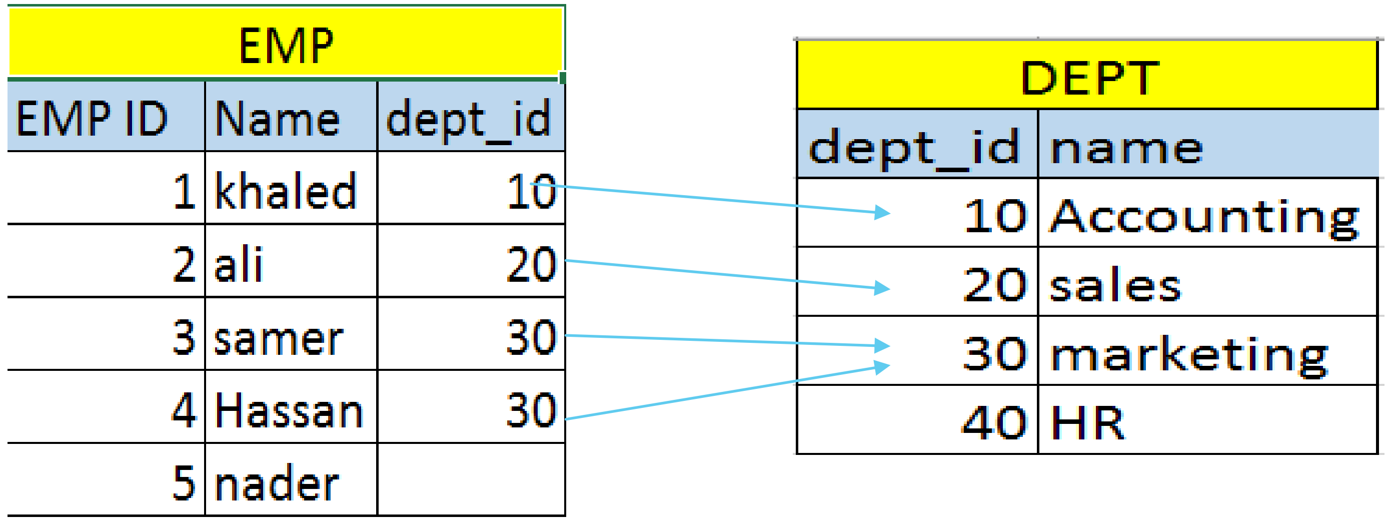
## Outer Joins Syntax

- You use an outer join to also see rows that do not meet the join condition.
- The outer join operator is the plus sign (+).

```
SELECT  table1.column, table2.column
FROM    table1, table2
WHERE   table1.column(+) = table2.column;
```

```
SELECT  table1.column, table2.column
FROM    table1, table2
WHERE   table1.column = table2.column(+);
```

# Outer Join



| EMP | | |
|---|---|---|
| EMP ID | Name | dept_id |
| 1 | khaled | 10 |
| 2 | ali | 20 |
| 3 | samer | 30 |
| 4 | Hassan | 30 |
| 5 | nader | |

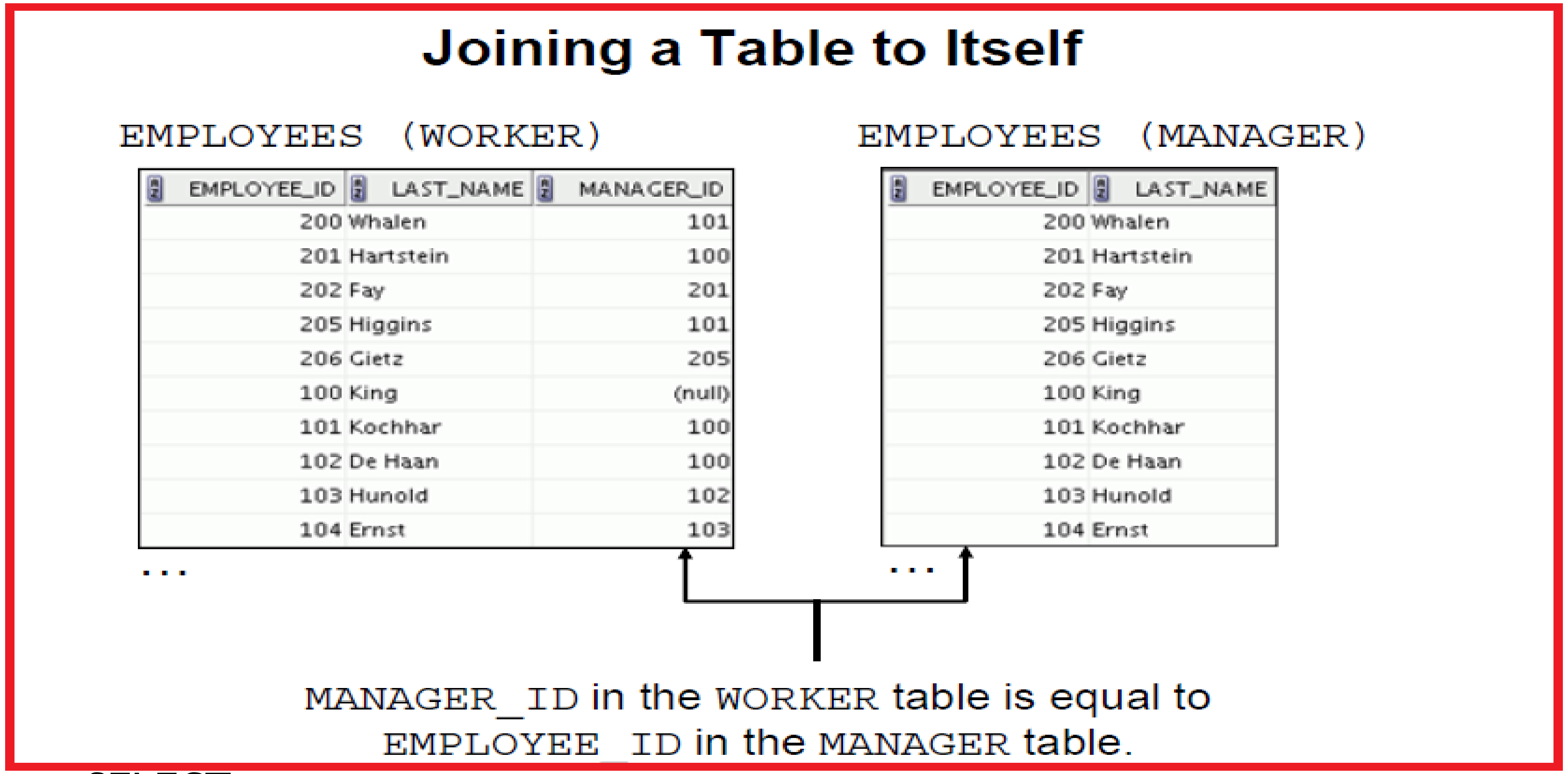
| DEPT | |
|---|---|
| dept_id | name |
| 10 | Accounting |
| 20 | sales |
| 30 | marketing |
| 40 | HR |

Where emp.dept_id=dept.dept_id(+)

| emp.EMP ID | emp.Name | emp.dept_id | dept.name |
|---|---|---|---|
| 1 | khaled | 10 | Accounting |
| 2 | ali | 20 | sales |
| 3 | samer | 30 | marketing |
| 4 | Hassan | 30 | marketing |
| 5 | nader | | |

Where emp.dept_id(+)=dept.dept_id

| emp.EMP ID | emp.Name | dept.dept_id | dept.name |
|---|---|---|---|
| 1 | khaled | 10 | Accounting |
| 2 | ali | 20 | sales |
| 3 | samer | 30 | marketing |
| 4 | Hassan | 30 | marketing |
| | | 40 | HR |



## Joining a Table to Itself

EMPLOYEES (WORKER)

| EMPLOYEE_ID | LAST_NAME | MANAGER_ID |
|---|---|---|
| 200 | Whalen | 101 |
| 201 | Hartstein | 100 |
| 202 | Fay | 201 |
| 205 | Higgins | 101 |
| 206 | Gietz | 205 |
| 100 | King | (null) |
| 101 | Kochhar | 100 |
| 102 | De Haan | 100 |
| 103 | Hunold | 102 |
| 104 | Ernst | 103 |

...

EMPLOYEES (MANAGER)

| EMPLOYEE_ID | LAST_NAME |
|---|---|
| 200 | Whalen |
| 201 | Hartstein |
| 202 | Fay |
| 205 | Higgins |
| 206 | Gietz |
| 100 | King |
| 101 | Kochhar |
| 102 | De Haan |
| 103 | Hunold |
| 104 | Ernst |

...

MANAGER_ID in the WORKER table is equal to EMPLOYEE_ID in the MANAGER table.

```
SELECT
worker.EMPLOYEE_ID,
WORKER.FIRST_NAME,
WORKER.MANAGER_ID,
manager.first_name
FROM
EMPLOYEES WORKER,
EMPLOYEES MANAGER
WHERE WORKER.MANAGER_ID=MANAGER.EMPLOYEE_ID;
```

maxvlearn

## Joining Tables Using SQL: 1999 Syntax

**Use a join to query data from more than one table.**

```
SELECT    table1.column, table2.column
FROM      table1
[CROSS JOIN table2] |
[NATURAL JOIN table2] |
[JOIN table2 USING (column_name)] |
[JOIN table2
  ON(table1.column_name = table2.column_name)] |
[LEFT|RIGHT|FULL OUTER JOIN table2
  ON (table1.column_name = table2.column_name)];
```

### Defining Joins

Using the SQL: 1999 syntax, you can obtain the same results as what was shown in the prior pages.

In the syntax:

| | |
|---|---|
| *table1.column* | Denotes the table and column from which data is retrieved |
| CROSS JOIN | Returns a Cartesian product from the two tables |
| NATURAL JOIN | Joins two tables based on the same column name |
| JOIN *table* USING *column_name* | Performs an equijoin based on the column name |
| JOIN *table* ON *table1.column_name* = *table2.column_name* | Performs an equijoin based on the condition in the ON clause |
| *LEFT/RIGHT/FULL OUTER* | |

## Creating Cross Joins

- The `CROSS JOIN` clause produces the cross-product of two tables.
- This is the same as a Cartesian product between the two tables.

```
SELECT last_name, department_name
FROM   employees
CROSS JOIN departments;
```

This is the same but in old join format

```
SELECT last_name, department_name
FROM employees, departments;
```

## Creating Natural Joins

- The `NATURAL JOIN` clause is based on all columns in the two tables that have the same name.
- It selects rows from the two tables that have equal values in all matched columns.
- If the columns having the same names have different data types, then an error is returned.

1999 Format

```
SELECT department_id, department_name,
       location_id, city
FROM   departments
NATURAL JOIN locations;
```

Old Format as Equijoin

The natural join can also be written as an equijoin:

```
SELECT department_id, department_name,
       departments.location_id, city
FROM   departments, locations
WHERE  departments.location_id = locations.location_id;
```

## Creating Joins with the USING Clause

- If several columns have the same names but the data types do not match, the NATURAL JOIN clause can be modified with the USING clause to specify the columns that should be used for an equijoin.
  Note: Use the USING clause to match only one column when more than one column matches.
- Do not use a table name or alias in the referenced columns.
- The NATURAL JOIN and USING clauses are mutually exclusive.

```
SELECT e.employee_id, e.last_name, d.location_id
FROM   employees e JOIN departments d
USING (department_id);
```

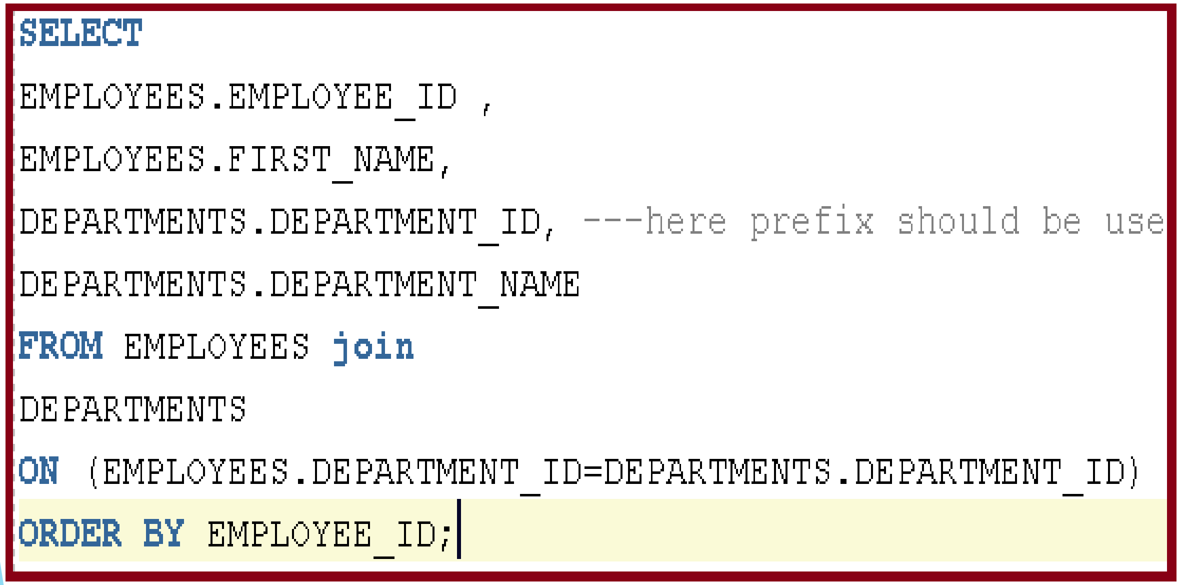
This can also be written as an equijoin:

```
   SELECT employee_id, last_name,
          employees.department_id, location_id
   FROM   employees, departments
   WHERE  employees.department_id = departments.department_id;
```

## Creating Joins with the ON Clause

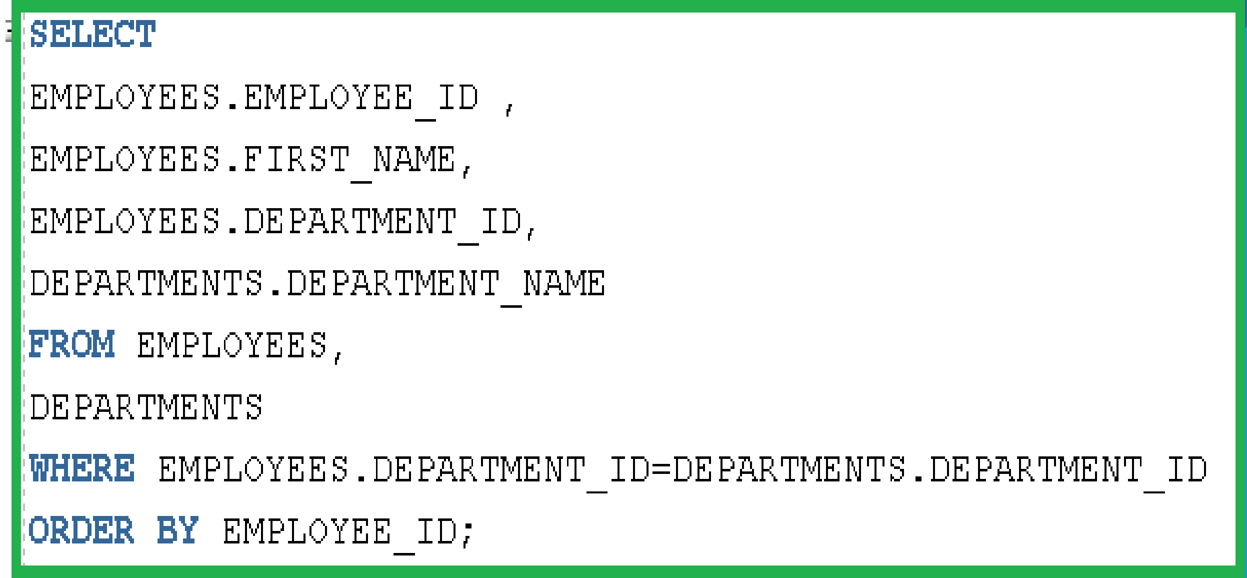
- The join condition for the natural join is basically an equijoin of all columns with the same name.
- To specify arbitrary conditions or specify columns to join, the ON clause is used.
- Separates the join condition from other search conditions.
- The ON clause makes code easy to understand.

1999 format



```
SELECT
EMPLOYEES.EMPLOYEE_ID ,
EMPLOYEES.FIRST_NAME,
DEPARTMENTS.DEPARTMENT_ID, ---here prefix should be use
DEPARTMENTS.DEPARTMENT_NAME
FROM EMPLOYEES join
DEPARTMENTS
ON (EMPLOYEES.DEPARTMENT_ID=DEPARTMENTS.DEPARTMENT_ID)
ORDER BY EMPLOYEE_ID;
```

Old Format



```
SELECT
EMPLOYEES.EMPLOYEE_ID ,
EMPLOYEES.FIRST_NAME,
EMPLOYEES.DEPARTMENT_ID,
DEPARTMENTS.DEPARTMENT_NAME
FROM EMPLOYEES,
DEPARTMENTS
WHERE EMPLOYEES.DEPARTMENT_ID=DEPARTMENTS.DEPARTMENT_ID
ORDER BY EMPLOYEE_ID;
```

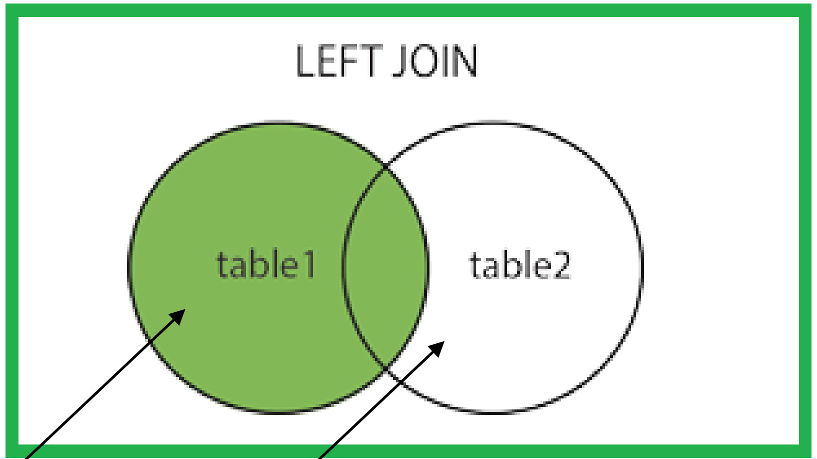
## Creating Three-Way Joins with the ON Clause

```
SELECT employee_id, city, department_name
FROM   employees e
JOIN   departments d
ON     d.department_id = e.department_id
JOIN   locations l
ON     d.location_id = l.location_id;
```

### Three-Way Joins

A three-way join is a join of three tables. In SQL: 1999 compliant syntax, joins are performed from left to right, so the first join to be performed is EMPLOYEES JOIN DEPARTMENTS. The first join condition can reference columns in EMPLOYEES and DEPARTMENTS but cannot reference columns in LOCATIONS. The second join condition can reference columns from all three tables.

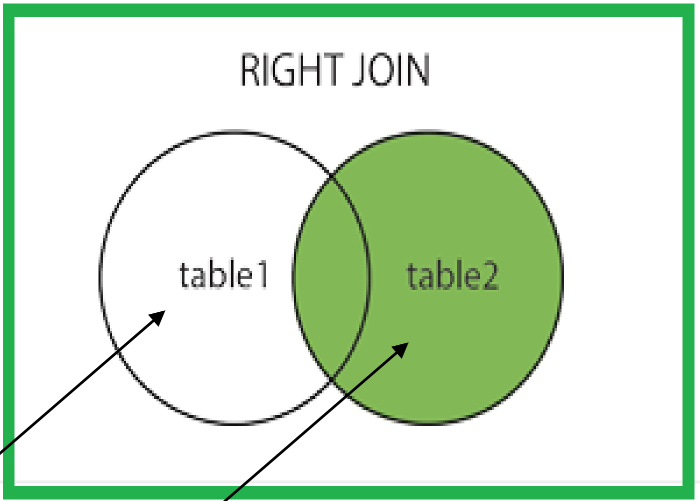
# Left Outer Join



```
SELECT
EMPLOYEES.EMPLOYEE_ID ,
EMPLOYEES.FIRST_NAME,
EMPLOYEES.DEPARTMENT_ID,
DEPARTMENTS.DEPARTMENT_NAME
FROM EMPLOYEES
left OUTER JOIN DEPARTMENTS
on( EMPLOYEES.DEPARTMENT_ID=DEPARTMENTS.DEPARTMENT_ID)
ORDER BY EMPLOYEE_ID;
```

```
SELECT
EMPLOYEES.EMPLOYEE_ID ,
EMPLOYEES.FIRST_NAME,
EMPLOYEES.DEPARTMENT_ID,
DEPARTMENTS.DEPARTMENT_NAME
FROM EMPLOYEES,
DEPARTMENTS
WHERE EMPLOYEES.DEPARTMENT_ID=DEPARTMENTS.DEPARTMENT_ID(+)
ORDER BY EMPLOYEE_ID;
```

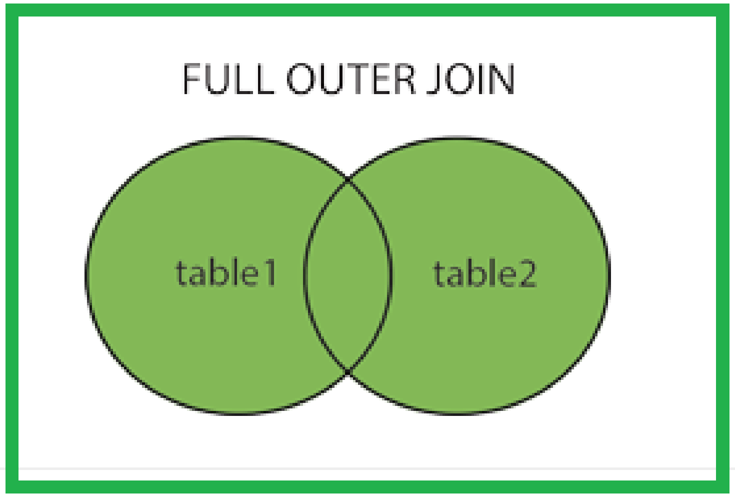
# Right Outer Join



```
SELECT
EMPLOYEES.EMPLOYEE_ID ,
EMPLOYEES.FIRST_NAME,
EMPLOYEES.DEPARTMENT_ID,
DEPARTMENTS.DEPARTMENT_NAME
FROM EMPLOYEES
right OUTER JOIN DEPARTMENTS
ON( EMPLOYEES.DEPARTMENT_ID=DEPARTMENTS.DEPARTMENT_ID)
ORDER BY EMPLOYEE_ID;
```

```
SELECT
EMPLOYEES.EMPLOYEE_ID ,
EMPLOYEES.FIRST_NAME,
EMPLOYEES.DEPARTMENT_ID,
DEPARTMENTS.DEPARTMENT_NAME
FROM EMPLOYEES,
DEPARTMENTS
WHERE EMPLOYEES.DEPARTMENT_ID(+)=DEPARTMENTS.DEPARTMENT_ID
ORDER BY EMPLOYEE_ID;
```

# full Outer Join



```
SELECT
EMPLOYEES.EMPLOYEE_ID ,
EMPLOYEES.FIRST_NAME,
EMPLOYEES.DEPARTMENT_ID,
DEPARTMENTS.DEPARTMENT_NAME
FROM EMPLOYEES
full OUTER JOIN DEPARTMENTS
ON( EMPLOYEES.DEPARTMENT_ID=DEPARTMENTS.DEPARTMENT_ID)
ORDER BY EMPLOYEE_ID;
```

## ▶ Thank You