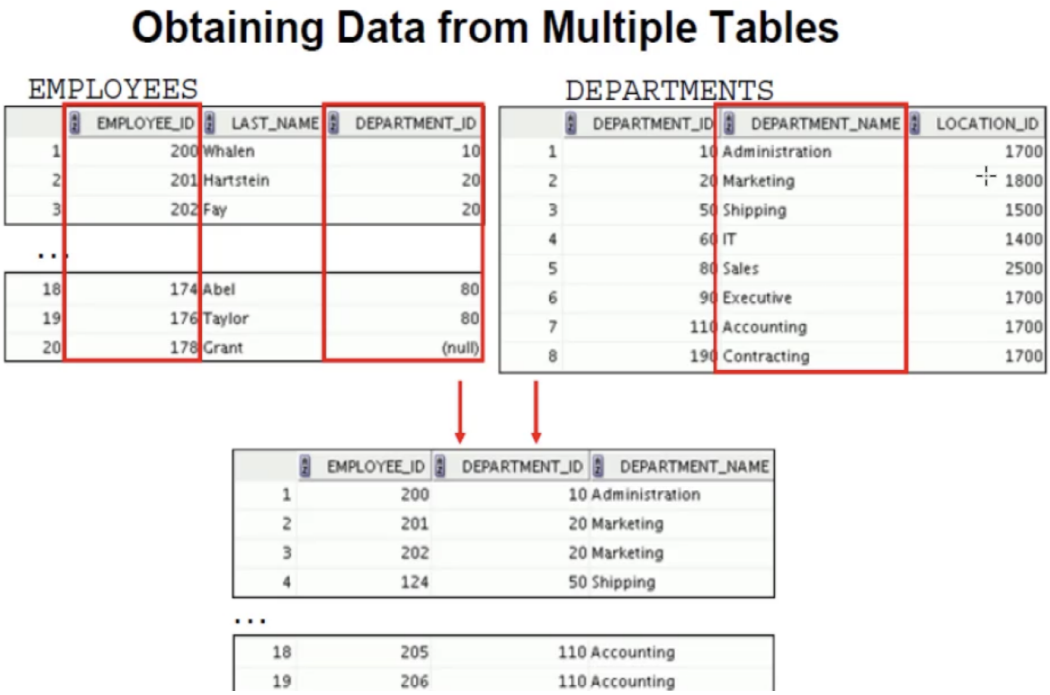
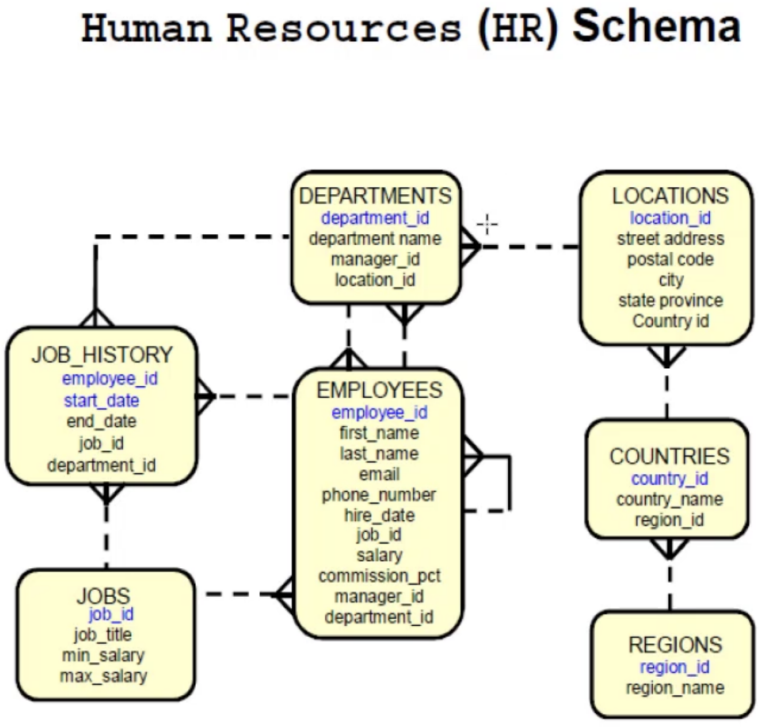
Displaying Data from Multiple Tables Using Joins





What is Cartesian product?

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 EMPLOYEE_ID, FIRST_NAME, DEPARTMENT_ID
FROM EMPLOYEES;
```

Number of rows: 107 rows

```
SELECT DEPARTMENT_ID, DEPARTMENT_NAME
FROM DEPARTMENTS;
```

Number of rows: 27 rows

If you try to display data from multiple tables without join, this is called cartesian product

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

Number of rows: 107*27 = 2889 rows

Old joins: Equijoin

Types of Joins

Oracle Proprietary Joins (8i and prior):

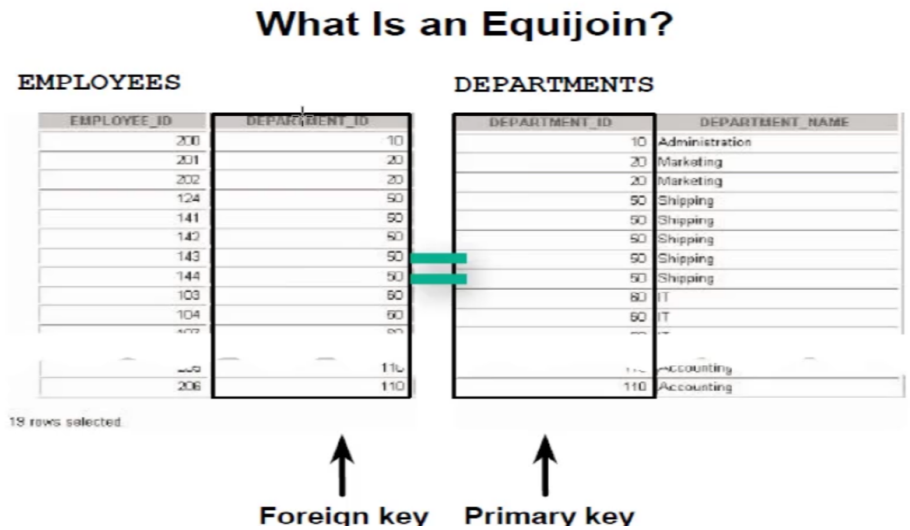
- Equijoin
- Nonequijoin
- Outer join
- Self join

SQL: 1999 Compliant Joins:

- Cross joins
- Natural joins
- Using clause

- Full or two sided outer joins
- Arbitrary join conditions for outer joins

What is an Equijoin?



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 = Simple Joins = Inner Joins

```
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;
```

The following code will give an error

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

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

ORA-00918: column ambiguously defined
00918.00000 - "column ambiguously defined"

Equijoins using additional conditions

```
SELECT
EMPLOYEES.EMPLOYEE_ID,
EMPLOYEES.FIRST_NAME,
EMPLOYEES.DEPARTMENT_ID,
DEPARTMENTS.DEPARTMENT_NAME
FROM EMPLOYEES,
DEPARTMENTS
WHERE EMPLOYEES.DEPARTMENT_ID = DEPARTMENTS.DEPARTMENT_ID
AND EMPLOYEES.DEPARTMENT_ID > 40
ORDER BY EMPLOYEE_ID;
```

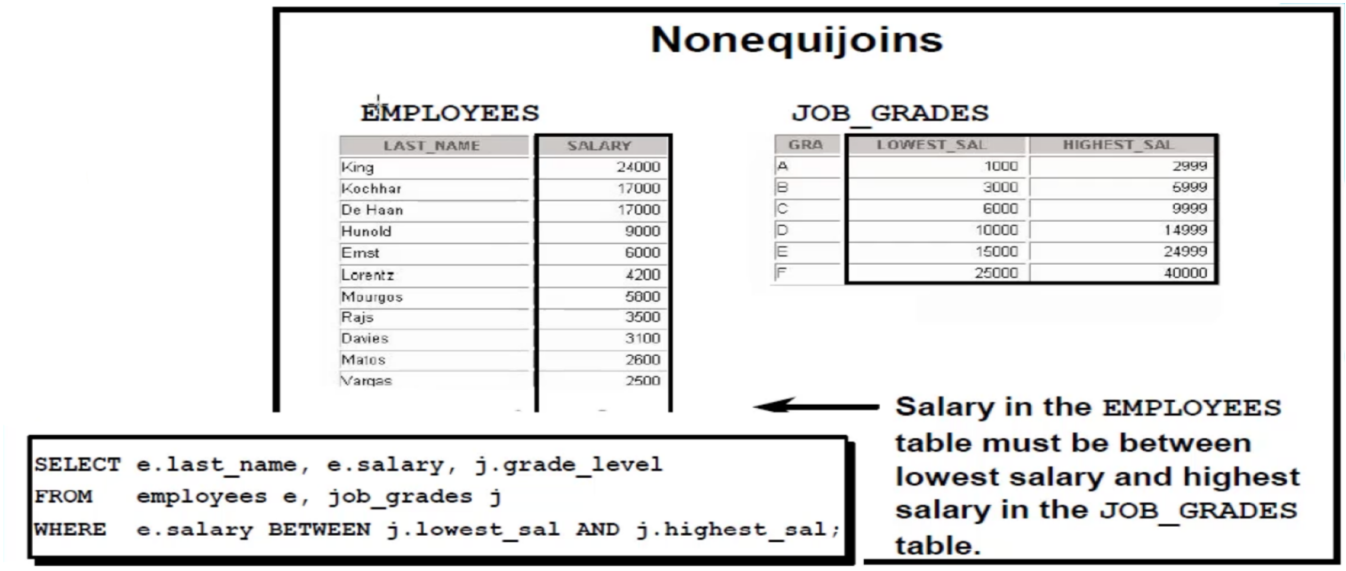
Equijoins using table alias

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

Join more than two tables

```
SELECT
EMP.EMPLOYEE_ID,
EMP.FIRST_NAME,
EMP.DEPARTMENT_ID,
DEPT.DEPARTMENT_NAME,
DEPT.LOCATION_ID,
LOC.CITY
FROM EMPLOYEES EMP,
DEPARTMENTS DEPT,
LOCATIONS LOC
WHERE EMP.DEPARTMENT_ID = DEPT.DEPARTMENT_ID
AND DEPT.LOCATION_ID = LOC.LOCATION_ID
ORDER BY EMPLOYEE_ID;
```

Old Joins: 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 tables 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 equal (=).

```
CREATE TABLE JOB_GRADES
(
GRADE_LEVEL VARCHAR2(3),
LOWEST_SAL NUMBER,
HIGHEST_SAL NUMBER
);
```

Here we insert the records in the table

```
INSERT INTO JOB_GRADES (GRADE_LEVEL, LOWEST_SAL, HIGHEST_SAL)
VALUES ('A', 1000, 2999);
INSERT INTO JOB_GRADES (GRADE_LEVEL, LOWEST_SAL, HIGHEST_SAL)
VALUES ('B', 3000, 5999);
INSERT INTO JOB_GRADES (GRADE_LEVEL, LOWEST_SAL, HIGHEST_SAL)
VALUES ('C', 6000, 9999);
INSERT INTO JOB_GRADES (GRADE_LEVEL, LOWEST_SAL, HIGHEST_SAL)
VALUES ('D', 10000, 14999);
INSERT INTO JOB_GRADES (GRADE_LEVEL, LOWEST_SAL, HIGHEST_SAL)
VALUES ('E', 15000, 24999);
INSERT INTO JOB_GRADES (GRADE_LEVEL, LOWEST_SAL, HIGHEST_SAL)
VALUES ('F', 25000, 40000);
COMMIT;
```

```
SELECT * FROM JOB_GRADES;
```

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

```
SELECT EMP.EMPLOYEE_ID, EMPL.FIRST_NAME, EMP.SALARY, GRADES.GRADE_LEVEL
FROM
EMPLOYEES EMP,
JOB_GRADES GRADES
WHERE EMP.SALARY BEWTEEN GRADES.LOWEST_SAL AND GRADES.HIGHEST_SAL
```

EMPLOYEE_ID	FIRST_NAME	SALARY	GRADE_LEVEL
203	Susan	6500	C
165	David	6800	C
113	Luis	6900	C

The following code will give the same result

```
SELECT EMP.EMPLOYEE_ID, EMPL.FIRST_NAME, EMP.SALARY, GRADES.GRADE_LEVEL
FROM
EMPLOYEES EMP,
JOB_GRADES GRADES
WHERE EMP.SALARY >= GRADES.LOWEST_SAL
AND EMP.SALARY <= GRADES.HIGHEST_SAL
```

Old Joins: Outer Join

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 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 table 1 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

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

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

Outer Join - Case 1

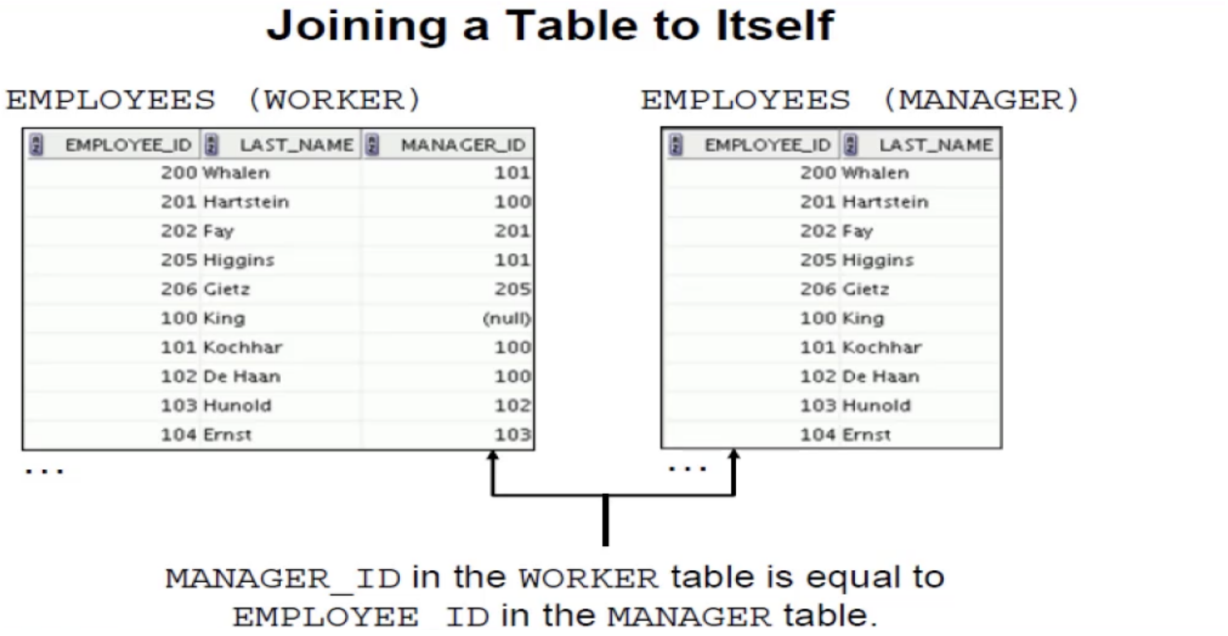
```
SELECT
EMPLOYEES.EMPLOYEE_ID,
EMPLOYEES.FIRST_NAME,
```

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

Outer Join - Case 2

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

Outer Joins: Self Join




```
SELECT EMPLOYEE_ID, FIRST_NAME, MANAGER_ID
FROM EMPLOYEES;
```

EMPLOYEE_ID	FIRST_NAME	MANAGER_ID
100	Steven	(null)

101	Neena	100
102	Lex	100
103	Alexander	102

Number of rows: 107 rows

```
SELECT
WORKER.EMPLOYEE_ID,
WORKER.FIRST_NAME,
WORKER.MANAGER_ID,
MANAGER.FIRST_NAME -- the manager name (first name) will be from the Manager alias
FROM
EMPLOYEES WORKER,
EMPLOYEES MANAGER
WHERE WORKER.MANAGER_ID = MANAGER.EMPLOYEE_ID;
```

EMPLOYEE_ID	FIRST_NAME	MANAGER_ID	FIRST_NAME_1
168	Lisa	148	Gerald
169	Harrison	148	Gerald
170	Tayler	148	Gerald

Number of rows: 106 rows because there is one employee without manager (i.e. Steven)

```
SELECT
WORKER.EMPLOYEE_ID,
WORKER.FIRST_NAME,
WORKER.MANAGER_ID,
MANAGER.FIRST_NAME -- the manager name (first name) will be from the Manager alias
FROM
EMPLOYEES WORKER,
EMPLOYEES MANAGER
WHERE WORKER.MANAGER_ID = MANAGER.EMPLOYEE_ID(+);
```

Apply the outer join to the MANAGER table because it does not have null values
Number of rows: 107 rows

1999 Syntax: Cross Join (Cartesian Product)

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;
```

1999 Syntax: USING Clause

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 EMPLOYEE E JOIN DEPARTMENTS D
USING (DEPARTMENT_ID);
```

We can get the same result using equijoin

```
SELECT EMPLOYEE_ID, LAST_NAME, EMPLOYEES.DEPARTMENT_ID, LOCATION_ID
FROM EMPLOYEES, DEPARTMENTS
WHERE EMPLOYEES.DEPARTMENT_ID = DEPARTMENTS.DEPARTMENT_ID;
```

Note: you can use the USING clause with multiple columns

```
SELECT *
FROM EMPLOYEES
JOIN JOB_HISTORY
USING (EMPLOYEE_ID, DEPARTMENT_ID);
```

1999 Syntax: ON Clause

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.

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;
```

1999 Format

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

Creating Three-Way Joins with the ON Clause

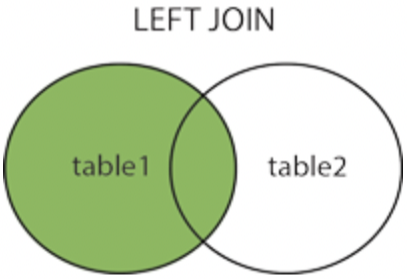
```
SELECT EMPLOYEE_ID, CITY, DEPARTMENT_NAME
FROM EMPLOYEES E
JOIN DEPARTMENTS D
ON D.DEPARTMENT_ID = E.DEPARTMENT_ID
JOIN LOACTION L
ON D.LOCATION_ID = L.LOCATION_ID;
```

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.

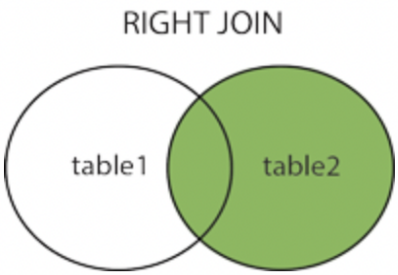
1999 Syntax: Left / Right / Full Outer Join

Left Outer Join



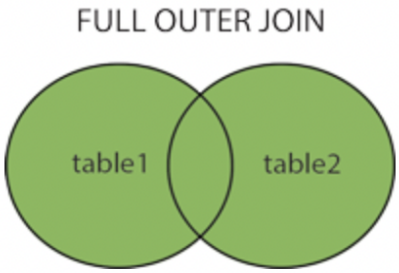

```
SELECT
EMPLOYEES.EMPLOYEE_ID,
EMPLOYEES.FIRST_NAME,
EMPLOYEES.DEPARTMENT_ID,
DEPARTMENTS.DEPARTMENT_NAME
FROM EMPLOYEES
LEFT OUTER JOIN DEPARTMENTS
ON (EMPLOYEES.DEPARTMENT_ID = DEPARTMENTS.DEPARTMENT_ID)
ORDER BY EMPLOYEE_ID;
```

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;
```

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;
```