CREATE TABLE departments (

department\_id NUMBER(2) CONSTRAINT departments\_pk PRIMARY KEY,

department\_name VARCHAR2(14),

location VARCHAR2(13)

);

INSERT INTO departments VALUES (10,'ACCOUNTING','NEW YORK');

INSERT INTO departments VALUES (20,'RESEARCH','DALLAS');

INSERT INTO departments VALUES (30,'SALES','CHICAGO');

INSERT INTO departments VALUES (40,'OPERATIONS','BOSTON');

COMMIT;

CREATE TABLE employees (

employee\_id NUMBER(4) CONSTRAINT employees\_pk PRIMARY KEY,

employee\_name VARCHAR2(10),

job VARCHAR2(9),

manager\_id NUMBER(4),

hiredate DATE,

salary NUMBER(7,2),

commission NUMBER(7,2),

department\_id NUMBER(2) CONSTRAINT emp\_department\_id\_fk REFERENCES departments(department\_id)

);

INSERT INTO employees VALUES (7369,'SMITH','CLERK',7902,to\_date('17-12-1980','dd-mm-yyyy'),800,NULL,20);

INSERT INTO employees VALUES (7499,'ALLEN','SALESMAN',7698,to\_date('20-2-1981','dd-mm-yyyy'),1600,300,30);

INSERT INTO employees VALUES (7521,'WARD','SALESMAN',7698,to\_date('22-2-1981','dd-mm-yyyy'),1250,500,30);

INSERT INTO employees VALUES (7566,'JONES','MANAGER',7839,to\_date('2-4-1981','dd-mm-yyyy'),2975,NULL,20);

INSERT INTO employees VALUES (7654,'MARTIN','SALESMAN',7698,to\_date('28-9-1981','dd-mm-yyyy'),1250,1400,30);

INSERT INTO employees VALUES (7698,'BLAKE','MANAGER',7839,to\_date('1-5-1981','dd-mm-yyyy'),2850,NULL,30);

INSERT INTO employees VALUES (7782,'CLARK','MANAGER',7839,to\_date('9-6-1981','dd-mm-yyyy'),2450,NULL,10);

INSERT INTO employees VALUES (7788,'SCOTT','ANALYST',7566,to\_date('13-JUL-87','dd-mm-rr')-85,3000,NULL,20);

INSERT INTO employees VALUES (7839,'KING','PRESIDENT',NULL,to\_date('17-11-1981','dd-mm-yyyy'),5000,NULL,10);

INSERT INTO employees VALUES (7844,'TURNER','SALESMAN',7698,to\_date('8-9-1981','dd-mm-yyyy'),1500,0,30);

INSERT INTO employees VALUES (7876,'ADAMS','CLERK',7788,to\_date('13-JUL-87', 'dd-mm-rr')-51,1100,NULL,20);

INSERT INTO employees VALUES (7900,'JAMES','CLERK',7698,to\_date('3-12-1981','dd-mm-yyyy'),950,NULL,30);

INSERT INTO employees VALUES (7902,'FORD','ANALYST',7566,to\_date('3-12-1981','dd-mm-yyyy'),3000,NULL,20);

INSERT INTO employees VALUES (7934,'MILLER','CLERK',7782,to\_date('23-1-1982','dd-mm-yyyy'),1300,NULL,10);

COMMIT;

An INNER JOIN combines data from two tables where there is a match on the joining column(s) in both tables.



Remember, the INNER keyword is optional. In the examples below, we are returning the DEPARTMENT\_NAME and the EMPLOYEE\_NAME for each employee. The OPERATIONS department has a DEPARTMENT\_ID of 40, so it is not removed by the filter condition, but there are no employees in this department, so there is no match and it is not returned in the result set.

SELECT d.department\_name,

e.employee\_name

FROM **departments d**

**JOIN employees e ON d.department\_id = e.department\_id**

WHERE d.department\_id >= 30

ORDER BY d.department\_name;

DEPARTMENT\_NAM EMPLOYEE\_N

-------------- ----------

SALES ALLEN

SALES BLAKE

SALES JAMES

SALES MARTIN

SALES TURNER

SALES WARD

SELECT d.department\_name,

e.employee\_name

FROM **departments d, employees e**

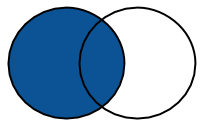
WHERE **d.department\_id = e.department\_id**

AND d.department\_id >= 30

ORDER BY d.department\_name;

LEFT [OUTER] JOIN

A LEFT [OUTER] JOIN returns all valid rows from the table on the left side of the JOIN keyword, along with the values from the table on the right side, or NULLs if a matching row doesn't exist.



Using the previous example, but switching to a LEFT OUTER JOIN means we will see the OPERATIONS department, even though it has no employees.

SELECT d.department\_name,

e.employee\_name

FROM **departments d**

**LEFT OUTER JOIN employees e ON d.department\_id = e.department\_id**

WHERE d.department\_id >= 30

ORDER BY d.department\_name, e.employee\_name;

Here is the non-ANSI equivalent of the previous statement. Notice the "(+)" is used to indicate the side of the join condition that may be missing. For a multi-column join condition, each column must have the "(+)" present. Unlike the ANSI join syntax, the non-ANSI join syntax is not affected by the order of the tables.

SELECT d.department\_name,

e.employee\_name

FROM **departments d, employees e**

WHERE **d.department\_id = e.department\_id (+)**

AND d.department\_id >= 30

ORDER BY d.department\_name, e.employee\_name;

Adding filters to columns returned from an outer joined table is a common cause for confusion. If you test for a specific value, for example "salary >= 2000", but the value for the SALARY column is NULL because the row is missing, a regular condition in the WHERE clause will throw the row away, therefore defeating the object of doing an outer join. Both the ANSI and non-ANSI methods have a way of dealing with this.

Using the ANSI join syntax, filters on columns from the outer joined table are included in the join itself, rather than being placed in the WHERE clause.

SELECT d.department\_name,

e.employee\_name

FROM departments d

LEFT OUTER JOIN employees e ON d.department\_id = e.department\_id **AND e.salary >= 2000**

WHERE d.department\_id >= 30

ORDER BY d.department\_name, e.employee\_name;

Using the non-ANSI join syntax, the "(+)" is used to indicate a column may have a NULL value as a result of an outer join.

SELECT d.department\_name,

e.employee\_name

FROM departments d, employees e

WHERE d.department\_id = e.department\_id (+)

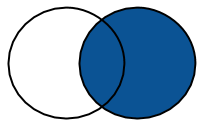
AND **e.salary (+) >= 2000**

AND d.department\_id >= 30

ORDER BY d.department\_name, e.employee\_name;

## RIGHT [OUTER] JOIN

The RIGHT [OUTER] JOIN is the opposite of the LEFT [OUTER] JOIN. It returns all valid rows from the table on the right side of the JOIN keyword, along with the values from the table on the left side, or NULLs if a matching row doesn't exist. All points raised in the previous section apply here also.



SELECT d.department\_name,

e.employee\_name

FROM **employees e**

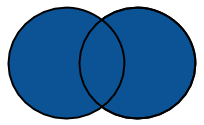
**RIGHT OUTER JOIN departments d ON e.department\_id = d.department\_id**

WHERE d.department\_id >= 30

ORDER BY d.department\_name, e.employee\_name;

## FULL [OUTER] JOIN

A FULL [OUTER] JOIN combines all the rows from the tables on the left and right sides of the join. If there is a conventional match it is made. If either side has missing data, it is replaced by NULLs, rather than throwing the row away.



INSERT INTO employees VALUES (8888,'JONES','DBA',null,to\_date('02-1-1982','dd-mm-yyyy'),1300,NULL,NULL);

COMMIT;

SELECT d.department\_name,

e.employee\_name

FROM **employees e**

**FULL OUTER JOIN departments d ON e.department\_id = d.department\_id**

ORDER BY d.department\_name, e.employee\_name;

SELECT d.department\_name,

e.employee\_name

FROM employees e, departments d

WHERE e.department\_id = d.department\_id (+)

UNION ALL

SELECT d.department\_name,

e.employee\_name

FROM departments d, employees e

WHERE d.department\_id = e.department\_id (+)

AND e.employee\_name IS NULL

ORDER BY 1, 2;

**Union and Union all**

The difference between **Union and Union all** is that **Union all** will not eliminate duplicate rows, instead it just pulls **all** rows from **all** tables fitting your query specifics and combines them into a table. A **UNION** statement effectively does a SELECT DISTINCT on the results set

**SELECT to\_date(sysdate, 'yyyy-mm-dd') FROM dual**

**UNION**

**SELECT to\_date(sysdate, 'yyyy-mm-dd') FROM dual;**

**SELECT to\_date(sysdate, 'yyyy-mm-dd') FROM dual**

**UNION ALL**

**SELECT to\_date(sysdate, 'yyyy-mm-dd') FROM dual;**

Interestingly, when you run an ANSI FULL OUTER JOIN, the Oracle optimizer rewrites it to a non-ANSI join equivalent, so there is no performance improvement associated with it. It's just easier on the eye.

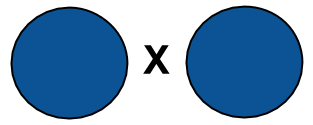
Let's remove that extra employee so it doesn't affect any other examples.

DELETE FROM employees WHERE employee\_id = 8888;

COMMIT;

## CROSS JOIN

A CROSS JOIN is the deliberate creation of a Cartesian product. There are no join columns specified, so every possible combination of rows between the two tables is produced.



SELECT e.employee\_name,

d.department\_name

FROM **employees e**

**CROSS JOIN departments d**

ORDER BY e.employee\_name, d.department\_name;

Here is the non-ANSI equivalent of the previous statement. Notice, there are no join conditions in the WHERE clause.

SELECT e.employee\_name,

d.department\_name

FROM **employees e, departments d**

ORDER BY e.employee\_name, d.department\_name;

## NATURAL JOIN

A NATURAL JOIN is a variant on an INNER JOIN. The join columns are determined implicitly, based on the column names. Any columns that share the same name between the two tables are assumed to be join columns. Here is an example using the ANSI join syntax.

SELECT e.employee\_name,

d.department\_name

FROM **employees e**

**NATURAL JOIN departments d**

ORDER BY e.employee\_name, d.department\_name;

There is no non-ANSI equivalent of this, as all join conditions must be specified.

Using a NATURAL JOIN is a bad idea. If someone adds a new column to one of the tables that happens to have the same name as a column in the other table, they may break any existing natural joins. It is effectively a bug waiting to happen.

You can't apply any aliased filters to columns used in natural joins, as shown in the following example.

SELECT e.employee\_name,

d.department\_name

FROM employees e

NATURAL JOIN departments d

WHERE **d.department\_id = 20**

ORDER BY e.employee\_name;

**WHERE d.department\_id = 20**

**\***

**ERROR at line 5:**

**ORA-25155: column used in NATURAL join cannot have qualifier**

Instead you must remove the alias, which in other circumstances would result in an ambiguous reference error.

SELECT e.employee\_name,

d.department\_name

FROM employees e

NATURAL JOIN departments d

WHERE **department\_id = 20**

ORDER BY e.employee\_name;

## [INNER] JOIN ... USING

The INNER JOIN ... USING is almost a half-way house between a conventional INNER JOIN and a NATURAL JOIN. The join is made using columns with matching names in each table, but you have to specify the columns to be used, not the whole condition. This allows you to join on a subset of the columns common to both tables.

SELECT e.employee\_name,

d.department\_name

FROM **employees e**

**JOIN departments d USING (department\_id)**

ORDER BY e.employee\_name;

This is a safe join syntax as it can't be affected by addition of columns to either table. Similar to the NATURAL JOIN, you can't apply any aliased filters to columns used in the join, but if you remove the alias it works.

SELECT e.employee\_name,

d.department\_name

FROM employees e

JOIN departments d USING (department\_id)

WHERE **d.department\_id = 20**

ORDER BY e.employee\_name;

**WHERE d.department\_id = 20**

**\***

**ERROR at line 5:**

**ORA-25154: column part of USING clause cannot have qualifier**

SELECT e.employee\_name,

d.department\_name

FROM employees e

JOIN departments d USING (department\_id)

WHERE **department\_id = 20**

ORDER BY e.employee\_name;