

# Displaying Data from Multiple Tables Using Joins

# Creating Natural Joins

- The `NATURAL JOIN` clause is based on all the columns that have the same name in two tables.
- 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, an error is returned.

```
SELECT * FROM table1 NATURAL JOIN table2;
```



# Retrieving Records with Natural Joins

```
SELECT employee_id, first_name, job_id, job_title  
from employees NATURAL JOIN jobs;
```



EMPLOYEE_ID	FIRST_NAME	JOB_ID	JOB_TITLE
1	206 William	AC_ACCOUNT	Public Accountant
2	205 Shelley	AC_MGR	Accounting Manager
3	200 Jennifer	AD_ASST	Administration Assistant
4	100 Steven	AD PRES	President
5	102 Lex	AD_VP	Administration Vice President
6	101 Neena	AD_VP	Administration Vice President
7	103 Alexander	IT_PROG	Programmer
8	104 Bruce	IT_PROG	Programmer
9	107 Diana	IT_PROG	Programmer
10	201 Michael	MK_MAN	Marketing Manager
11	202 Pat	MK_REP	Marketing Representative
12	149 Ellen	SA_MAN	Sales Manager
13	174 Ellen	SA_REP	Sales Representative
14	178 Kimberly	SA_REP	Sales Representative
15	176 Jonathon	SA_REP	Sales Representative
16	143 Randall	ST_CLERK	Stock Clerk
17	142 Curtis	ST_CLERK	Stock Clerk
18	141 Tenna	ST_CLERK	Stock Clerk
19	144 Peter	ST_CLERK	Stock Clerk
20	124 Kevin	ST_MAN	Stock Manager

# Creating Joins with the USING Clause

---

- When should you use the `USING` clause?
- If several columns have the same names but the data types do not match, use the `USING` clause to specify the columns for the equijoin.
- Use the `USING` clause to match only one column when more than one column matches.

# Joining Column Names

EMPLOYEES

	EMPLOYEE_ID	DEPARTMENT_ID
1	200	10
2	201	20
3	202	20
4	205	110
5	206	110
6	100	90
7	101	90
8	102	90
9	103	60
10	104	60

...

Foreign key

DEPARTMENTS

	DEPARTMENT_ID	DEPARTMENT_NAME
1	10	Administration
2	20	Marketing
3	50	Shipping
4	60	IT
5	80	Sales
6	90	Executive
7	110	Accounting
8	190	Contracting

Primary key

# Retrieving Records with the USING Clause

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



	EMPLOYEE_ID	LAST_NAME	LOCATION_ID	DEPARTMENT_ID
1	200	Whalen	1700	10
2	201	Hartstein	1800	20
3	202	Fay	1800	20
4	144	Vargas	1500	50
5	143	Matos	1500	50
6	142	Davies	1500	50
7	141	Rajs	1500	50
8	124	Mourgos	1500	50

...

18	206	Gietz	1700	110
19	205	Higgins	1700	110



# Qualifying Ambiguous Column Names

---

- Use table prefixes to:
  - Qualify column names that are in multiple tables
  - Increase the speed of parsing of a statement
- Instead of full table name prefixes, use table aliases.
- Table alias gives a table a shorter name:
  - Keeps SQL code smaller, uses less memory
- Use column aliases to distinguish columns that have identical names, but reside in different tables.



# Using Table Aliases with the USING Clause in Oracle

- Do not qualify a column that is used in the NATURAL join or a join with a USING clause.
- If the same column is used elsewhere in the SQL statement, do not alias it.

```
SELECT l.city, d.department_name  
FROM   locations l JOIN departments d  
USING (location_id)  
WHERE d.location_id = 1400;
```



ORA-25154: column part of USING clause cannot have qualifier  
25154. 00000 - "column part of USING clause cannot have qualifier"  
\*Cause: Columns that are used for a named-join (either a NATURAL join  
or a join with a USING clause) cannot have an explicit qualifier.  
\*Action: Remove the qualifier.  
Error at Line: 4 Column: 6



# Creating Joins with the ON Clause

---

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

# Retrieving Records with the ON Clause

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



	EMPLOYEE_ID	LAST_NAME	DEPARTMENT_ID	DEPARTMENT_ID_1	LOCATION_ID
1	200	Whalen	10	10	1700
2	201	Hartstein	20	20	1800
3	202	Fay	20	20	1800
4	124	Mourgos	50	50	1500
5	144	Vargas	50	50	1500
6	143	Matos	50	50	1500
7	142	Davies	50	50	1500
8	141	Rajs	50	50	1500
9	107	Lorentz	60	60	1400
10	104	Ernst	60	60	1400
11	103	Hunold	60	60	1400

...

# Creating Three-Way Joins

```
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;
```



	EMPLOYEE_ID	CITY	DEPARTMENT_NAME
1	100	Seattle	Executive
2	101	Seattle	Executive
3	102	Seattle	Executive
4	103	Southlake	IT
5	104	Southlake	IT
6	107	Southlake	IT
7	124	South San Francisco	Shipping
8	141	South San Francisco	Shipping
9	142	South San Francisco	Shipping

# Applying Additional Conditions to a Join

Use the **AND** clause or the **WHERE** clause to apply additional conditions:

```
SELECT e.employee_id, e.last_name, e.department_id,  
       d.department_id, d.location_id  
FROM   employees e JOIN departments d  
ON     (e.department_id = d.department_id)  
AND    e.manager_id = 149 ;
```

OR

```
SELECT e.employee_id, e.last_name, e.department_id,  
       d.department_id, d.location_id  
FROM   employees e JOIN departments d  
ON     (e.department_id = d.department_id)  
WHERE  e.manager_id = 149 ;
```

# 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.

# Self-Joins Using the ON Clause

---

```
SELECT worker.last_name emp, manager.last_name mgr
FROM   employees worker JOIN employees manager
ON     (worker.manager_id = manager.employee_id);
```



	EMP	MGR
1	Hunold	De Haan
2	Fay	Hartstein
3	Gietz	Higgins
4	Lorentz	Hunold
5	Ernst	Hunold
6	Zlotkey	King
7	Mourgos	King
8	Kochhar	King

...



# Returning Records with No Direct Match Using OUTER Joins

DEPARTMENTS

	DEPARTMENT_NAME	DEPARTMENT_ID
1	Administration	10
2	Marketing	20
3	Shipping	50
4	IT	60
5	Sales	80
6	Executive	90
7	Accounting	110
8	Contracting	190

There are no employees in department 190.

Employee "Grant" has not been assigned a department ID.

Therefore, the above two records do not appear in the equijoin result.

Equijoin with EMPLOYEES

	DEPARTMENT_ID	LAST_NAME
1	10	Whalen
2	20	Hartstein
3	20	Fay
4	110	Higgins
5	110	Gietz
6	90	King
7	90	Kochhar
8	90	De Haan
9	60	Hunold
10	60	Ernst

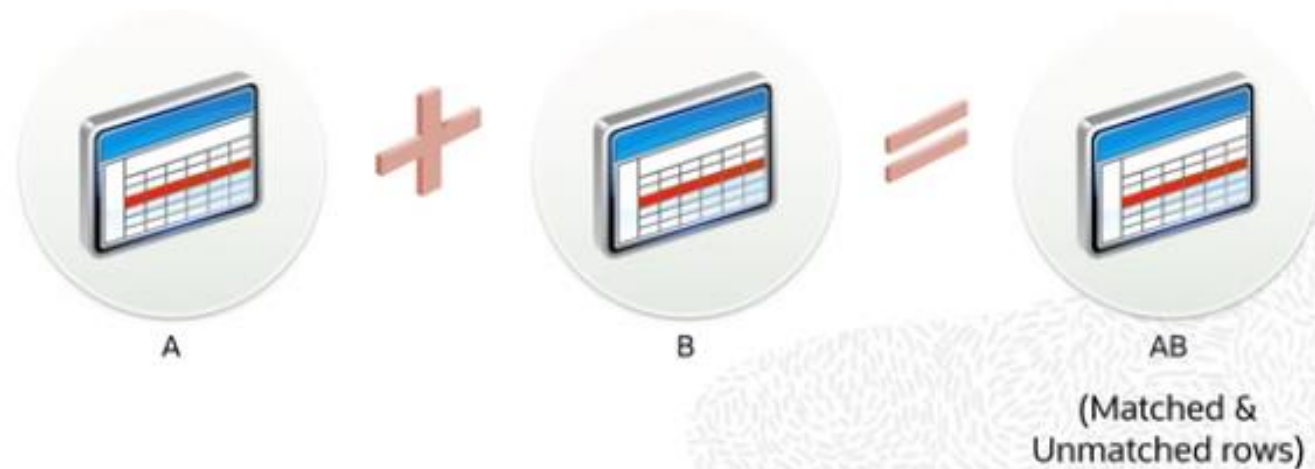
...

18	80	Abel
19	80	Taylor



# INNER Versus OUTER Joins

- The join of two tables returning only matched rows is called an `INNER` join.
- A join between two tables that returns the results of the `INNER` join as well as the unmatched rows from the left (or right) table is called a `LEFT` (or `RIGHT`) `OUTER` join.
- In Oracle, a join between two tables that returns the results of an `INNER` join as well as the results of a left and right join is a `FULL OUTER JOIN`.



# LEFT OUTER JOIN

```
SELECT e.last_name, e.department_id, d.department_name
FROM   employees e LEFT OUTER JOIN departments d
ON     (e.department_id = d.department_id) ;
```



	LAST_NAME	DEPARTMENT_ID	DEPARTMENT_NAME
1	Whalen	10	Administration
2	Fay	20	Marketing
3	Hartstein	20	Marketing
4	Vargas	50	Shipping
5	Matos	50	Shipping

...

16	Kochhar	90	Executive
17	King	90	Executive
18	Gietz	110	Accounting
19	Higgins	110	Accounting
20	Grant	(null)	(null)

# RIGHT OUTER JOIN

```
SELECT e.last_name, d.department_id, d.department_name
FROM   employees e RIGHT OUTER JOIN departments d
ON     (e.department_id = d.department_id) ;
```



	1 LAST_NAME	2 DEPARTMENT_ID	3 DEPARTMENT_NAME
1	Whalen	10	Administration
2	Hartstein	20	Marketing
3	Fay	20	Marketing
4	Davies	50	Shipping
5	Vargas	50	Shipping
6	Rajs	50	Shipping
7	Mourgos	50	Shipping
8	Matos	50	Shipping

...

18	Higgins	110	Accounting
19	Gietz	110	Accounting
20	(null)	190	Contracting

# FULL OUTER JOIN in Oracle

```
SELECT e.last_name, d.department_id, d.department_name
FROM   employees e FULL OUTER JOIN departments d
ON     (e.department_id = d.department_id) ;
```



	LAST_NAME	DEPARTMENT_ID	DEPARTMENT_NAME
1	King	90	Executive
2	Kochhar	90	Executive
3	De Haan	90	Executive
4	Hunold	60	IT

...

15	Grant	(null)	(null)
16	Whalen	10	Administration
17	Hartstein	20	Marketing
18	Fay	20	Marketing
19	Higgins	110	Accounting
20	Gietz	110	Accounting
21	(null)	190	Contracting

# Cartesian Products

A Cartesian product:

- Is a join of every row of one table to every row of another table
- Generates a large number of rows and the result is rarely useful



# Generating a Cartesian Product

EMPLOYEES (20 rows)

EMPLOYEE_ID	LAST_NAME	DEPARTMENT_ID
1	200 Whalen	10
2	201 Hartstein	20
3	202 Fay	20
4	205 Higgins	110
...		
19	176 Taylor	80
20	178 Grant	(null)

DEPARTMENTS (8 rows)

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

Cartesian product:  
20 x 8 = 160 rows

EMPLOYEE_ID	DEPARTMENT_ID	LOCATION_ID
1	200	10
2	201	20
...		
21	200	10
22	201	20
...		
159	176	80
160	178	(null)



# Creating Cross Joins

- A `CROSS JOIN` is a `JOIN` operation that produces a Cartesian product of two tables.
- To create a Cartesian product, specify `CROSS JOIN` in your `SELECT` statement.

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



	LAST_NAME	DEPARTMENT_NAME
1	Abel	Administration
2	Davies	Administration
3	De Haan	Administration
4	Ernst	Administration
5	Fay	Administration
...		
158	Vargas	Contracting
159	Whalen	Contracting
160	Zlotkey	Contracting





<https://www.techonthenet.com/oracle/joins.php>

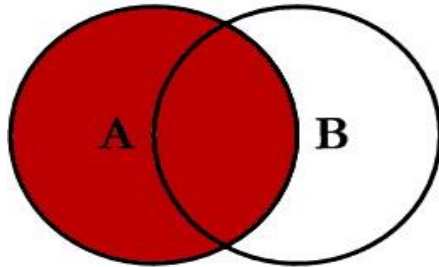
# Outer Join and ANSI equivalents

- The table below shows ANSI/ISO SQL: 99 joins and their equivalent Oracle outer joins

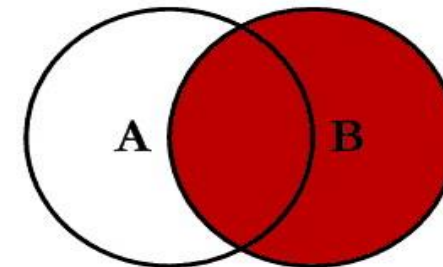
ANSI/ISO SQL	Oracle Syntax
LEFT OUTER JOIN departments d ON (e.department_id = d.department_id);	WHERE e.department_id = d.department_id(+);
RIGHT OUTER JOIN departments d ON (e.department_id = d.department_id);	WHERE e.department_id(+) = d.department_id;
FULL OUTER JOIN departments d ON (e.department_id = d.department_id);	No direct equivalent.

# Tipuri de join-uri

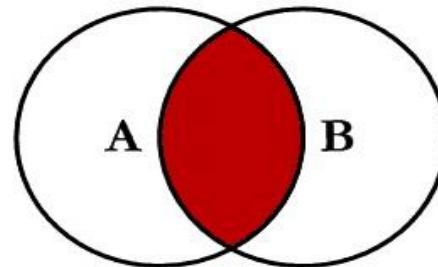
## SQL JOINS



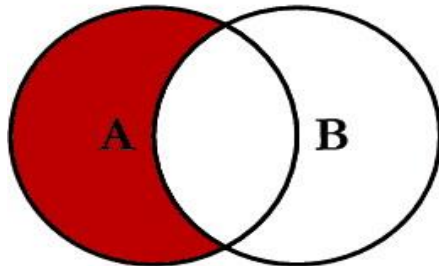
```
SELECT <select_list>  
FROM TableA A  
LEFT JOIN TableB B  
ON A.Key = B.Key
```



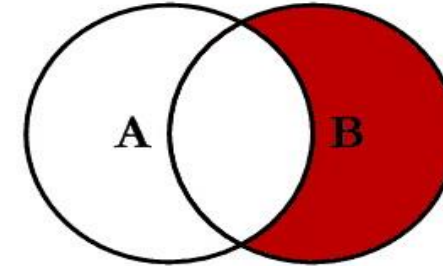
```
SELECT <select_list>  
FROM TableA A  
RIGHT JOIN TableB B  
ON A.Key = B.Key
```



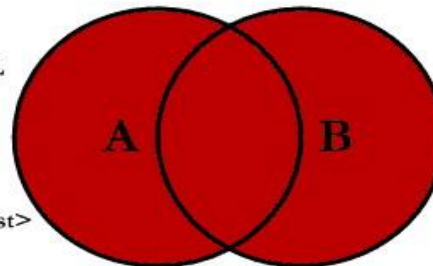
```
SELECT <select_list>  
FROM TableA A  
INNER JOIN TableB B  
ON A.Key = B.Key
```



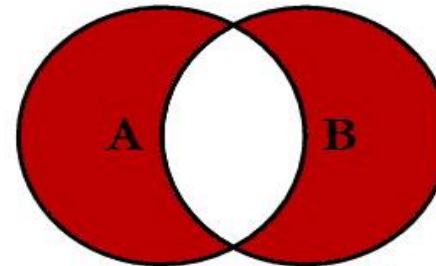
```
SELECT <select_list>  
FROM TableA A  
LEFT JOIN TableB B  
ON A.Key = B.Key  
WHERE B.Key IS NULL
```



```
SELECT <select_list>  
FROM TableA A  
RIGHT JOIN TableB B  
ON A.Key = B.Key  
WHERE A.Key IS NULL
```



```
SELECT <select_list>  
FROM TableA A  
FULL OUTER JOIN TableB B  
ON A.Key = B.Key
```



```
SELECT <select_list>  
FROM TableA A  
FULL OUTER JOIN TableB B  
ON A.Key = B.Key  
WHERE A.Key IS NULL  
OR B.Key IS NULL
```