Displaying Data from Multiple Tables Using Joins

Practical 8

Reminder:

Intellectual Property

- Copyright must be seriously protected. The University takes a strong stand against any illegal photocopying and distributing of all materials provided to students. Students are forewarned of the consequences and the penalty that may be meted out if they are “caught in the act”.
- All the materials provided to student **SHOULD NOT** be posted/distributed at any online platform or any other ways possible without the permission.

Lesson Objectives

- Learn how to create SQL queries that join multiple tables.

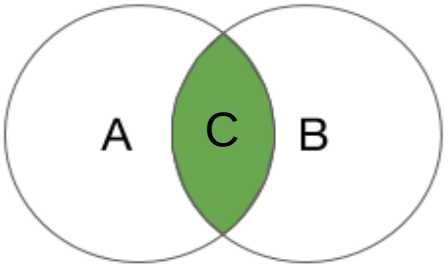
*Run **Northwoods.sql and HR.sql**

Types of Join

- **Cross Join / Cartesian Product**
- **Inner Join / Natural Join / Equijoin**
- Outer Join
  - Left Outer Join
  - Right Outer Join
  - Full Outer Join
- Self Join
- Non-Equijoin

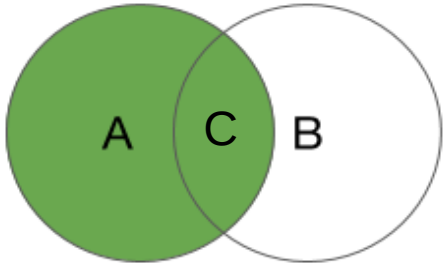
Types of Join

(Select C only)



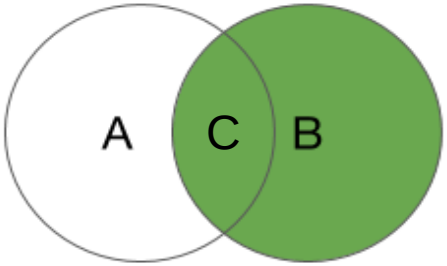
INNER JOIN

(Select A and C only)



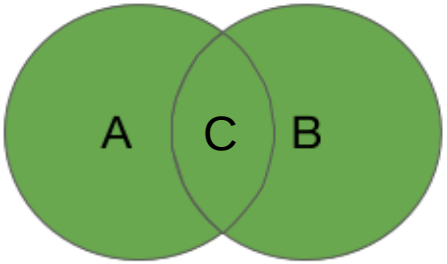
LEFT OUTER JOIN

(Select B and C only)



RIGHT OUTER JOIN

(Select A, B and C)



FULL OUTER JOIN

Cross Join / Cartesian Product

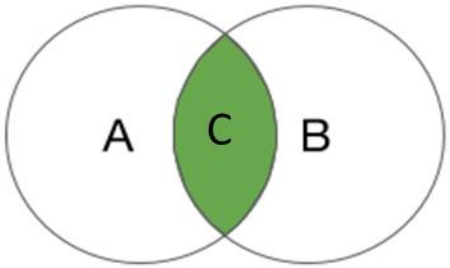
- **Cross Join** or **Cartesian Product** makes every row in one table joined with every row in the other table.

```
SELECT COUNT(*)
FROM employees;

SELECT COUNT(*)
FROM departments;

SELECT last_name, department_name
FROM employees
CROSS JOIN departments;

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

Inner Join

- **Inner Join** is the simplest type of join that occurs when you join two tables based on values in one table being equal to values in another table.
- The NATURAL JOIN clause is based on all the columns in the two tables that have the same name.
- It selects rows from the two tables that have equal values in all matched column.
- If the columns having the same names have different data types, an error is returned.

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



Using Table Alias

- **Table Alias** is an alternate name that you assign to the table in the FROM clause of the query.
- It helps to keep SQL code smaller. Therefore, less memory is used and performance is improved.

```
SELECT l.city, d.department_name
FROM locations l NATURAL JOIN departments d;
```

Creating Joins with the USING Clause

- If several columns have the same names but the data type do not match, use the USING clause to specify the column for the equijoin.
- Use the USING clause to match only one column when more than one column matches.
- The NATURAL JOIN and USING clauses are mutually exclusive.

Creating Joins with the USING Clause

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

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

ERROR at line 4:

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

Creating Joins with the USING Clause

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

CITY	DEPARTMENT_NAME
Southlake	IT

Creating Joins with the USING Clause

- Only simple column name is allowed to be used in the USING clause.

```
SELECT first_name, department_name, d.manager_id
FROM employees e JOIN departments d
USING (d.department_id)
WHERE department_id = 50;
```

```
USING (d.department_id)
       *
ERROR at line 3:
ORA-01748: only simple column
names allowed here
```

```
SELECT first_name, department_name, d.manager_id
FROM employees e JOIN departments d
USING (department_id)
WHERE department_id = 50;
```

Creating Joins with the USING Clause

- If the same column is used elsewhere in the SQL statement, do not alias it.

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

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

Creating Joins with the USING Clause

- The column that is common in both tables, but not used in the USING clause, must be prefix with a table alias.

```
SELECT first_name, department_name, manager_id
FROM employees e JOIN departments d
USING (department_id)
WHERE department_id = 50;
```

```
SELECT first_name,
department_name, manager_id

*
ERROR at line 1:
ORA-00918: column ambiguously
defined
```

```
SELECT first_name, department_name, d.manager_id
FROM employees e JOIN departments d
USING (department_id)
WHERE department_id = 50;
```

Creating Joins with the ON Clause

- Use the ON clause to specify arbitrary conditions or specific column to join.
- The join condition is separated from other search conditions.
- The ON clause makes code easy to understand.

SQL 1999 vs. Oracle

SQL 1999:

```
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;
```

Oracle:

```
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;
```

Practice 8.1

Write the following questions using Natural join, Using clause, On clause and Oracle:

1. Identify the city of every departments.
2. List all employee last name with their respective job title.
3. List all countries name with their respective regions name.

Joining More Than 2 Tables

SQL 1999:

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

Oracle:

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

Applying Additional Conditions to Join

SQL 1999:

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

Oracle:

```
SELECT e.employee_id, e.last_name, e.department_id, d.location_id
FROM employees e, departments d
WHERE e.department_id = d.department_id
AND e.manager_id = 149;
```

Practice 8.2

Write the following questions using Using clause, On clause and Oracle:

1. List the location_id, city and departments name that have location_id 1400.
2. List the address for all department include department name, postal code, city, and country name.
3. List all employees first name and department name from department_id 50 as well as their respective manager_id.

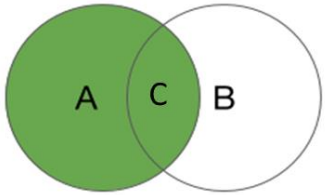
Practice 8.3

Why the following statements yield different results?

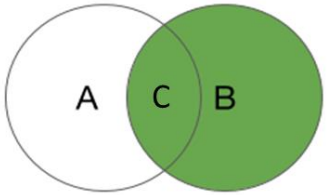
```
SELECT e.employee_id, e.last_name, e.department_id
FROM employees e JOIN departments d
ON e.department_id = d.department_id;

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

SELECT employee_id, department_name
FROM employees e
NATURAL JOIN departments d;
```

INNER Join vs Outer Join

- Inner Join returns rows only if values exist in all tables that are joined. If no values exist for a row in one of the joined tables, the inner join does not retrieve the row.
- **Outer Join** returns all rows from one table, and also retrieves matching rows from a second table.
- The outer join operator (+) inserts a NULL value for the columns that do not have matching rows. (Oracle syntax)



LEFT OUTER JOIN



RIGHT OUTER JOIN

Employee(s) who doesn’t belong to any department

```
SELECT first_name, last_name
FROM employees
WHERE department_id IS NULL;
```

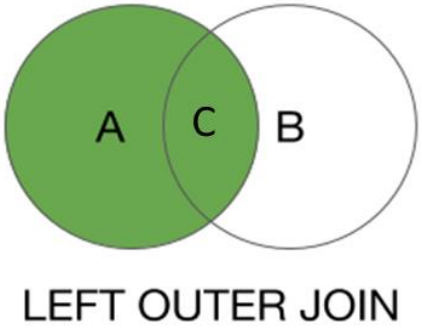
```
FIRST_NAME           LAST_NAME
-------------------  -------------------
Kimberely            Grant
```

Department which does not have employee

```
SELECT department_name, department_id
FROM departments
WHERE NOT EXISTS (SELECT * FROM employees
WHERE departments.department_id = employees.department_id)
```

DEPARTMENT_NAME	DEPARTMENT_ID
Contracting	190
IT Helpdesk	230

Left Outer Join



- **Left Outer Join** shows all the rows from the left table even though there are no matching rows.

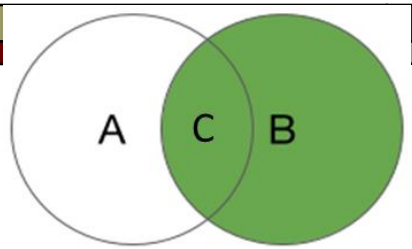
SQL 1999

```
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);
```

Oracle

```
SELECT e.last_name, e.department_id, d.department_name
FROM employees e, departments d
WHERE e.department_id = d.department_id(+);
```

Right Outer Join



RIGHT OUTER JOIN

- **Right Outer Join** shows all the rows from the right table even though there are no matching rows.

SQL 1999

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

Oracle

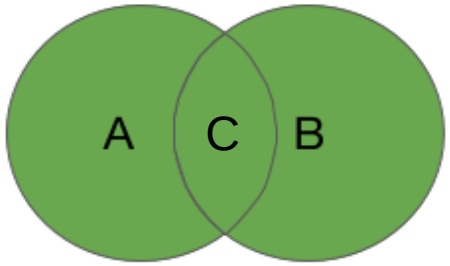
```
SELECT e.last_name, e.department_id, d.department_name
FROM employees e, departments d
WHERE e.department_id(+) = d.department_id;
```

Full Outer Join

- **Full Outer Join** shows all the rows from both tables even though there are no matching rows.

SQL 1999

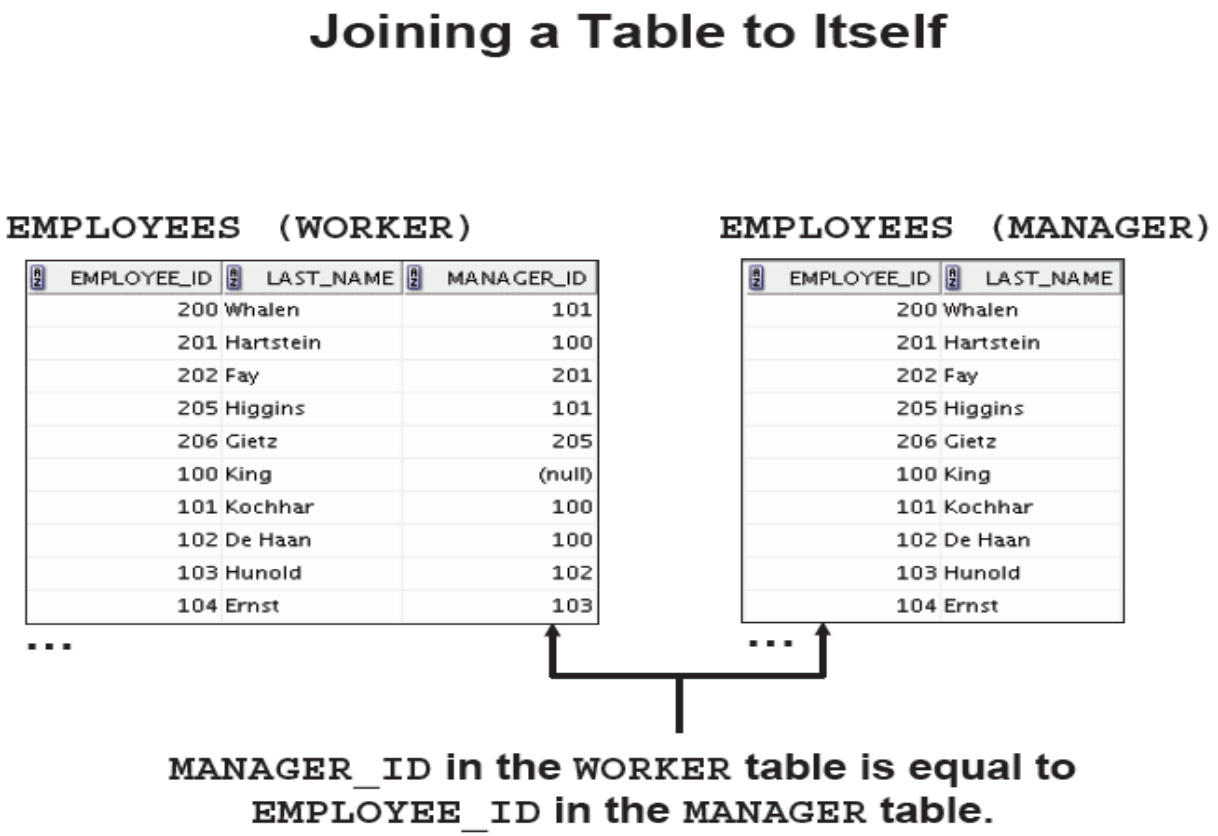
```
SELECT e.last_name, e.department_id, d.department_name
FROM employees e FULL OUTER JOIN departments d
ON (e.department_id = d.department_id);
```



FULL OUTER JOIN

Self Join

- When you create a query that joins a table to itself, you create a **Self Join**.



Self Join

SQL 1999

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

Oracle

```
SELECT worker.last_name emp, manager.last_name mgr
FROM employees worker, employees manager
WHERE worker.manager_id = manager.employee_id;
```

Non-Equi Join

- **Non-Equi Join** is a join condition that contains something other than an equality operator.

EMPLOYEES

	LAST_NAME	SALARY
1	Whalen	4400
2	Hartstein	13000
3	Fay	6000
4	Higgins	12000
5	Gietz	8300
6	King	24000
7	Kochhar	17000
8	De Haan	17000
9	Hunold	9000
10	Ernst	6000
...		
19	Taylor	8600
20	Grant	7000

JOB_GRADES

	GRADE_LEVEL	LOWEST_SAL	HIGHEST_SAL
1	A	1000	2999
2	B	3000	5999
3	C	6000	9999
4	D	10000	14999
5	E	15000	24999
6	F	25000	40000

The `JOB_GRADES` table defines the `LOWEST_SAL` and `HIGHEST_SAL` range of values for each `GRADE_LEVEL`. Therefore, the `GRADE_LEVEL` column can be used to assign grades to each employee.

Non-Equi Join

SQL 1999

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

Oracle

```
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;
```

Practice 8.4

- List the number of course section offered in each term

Term	Section Offered
Fall 2006	4
Spring 2007	6
Summer 2007	3

Practice 8.5

Write a query to list all the subjects and the number of students who had taken the course.

Subject	#
Database Management	4
Intro. to Info. Systems	6
Web-Based Systems	3
Systems Analysis	6

Practice 8.6

Modify practice 8.5, include only those subjects which is taken by more than 4 students.

Subject	#
Database Management	4
Intro. to Info. Systems	6
Systems Analysis	6

Do it yourself

- List all employee id with the respective department names and city.
- List the information of students; include the student last name and their enrollment subject.
- List all the call id and course name which are offered in Summer 2007, together with the faculty member name who taught the courses.

- Try the exercise given.