**XJTLU Entrepreneur College (Taicang)**

***School of* *AI and Advanced Computing***

**Lab Manual (Lab 8)**

***CPT103TC:***

***Introduction to Database***

Shaheen Khatoon

Second Part of Semester 1

**Academic Year 2021-2022**

**Preface**

This laboratory manual serves as supplemental material for the laboratory class. Majority of its contents were taken from the materials of Oracle’s DB Design and Programming with SQL training.

The table below shows the delivery plan with reference material.

|  |  |  |
| --- | --- | --- |
| **Labs** | **Topics** | **Reference in DB Programming with SQL** |
| **1** | Introduction to Oracle Application Express  SQL Workshop for uploading and running scripts | Lab set-up  Oracle Application Development Foundation (Self-study)  SQL Scripts |
| **2** | Demonstrate and end to end application building process | Oracle Application Development Foundation (Self-study)  Project OracleFlix-demo |
| **3** | Data modeling using SQL Developer and ER Assistant | SQL Developer and ER Assistant Tutorial |
| **4** | SQL DDL command to create database objects and constraints | Demo CompanyScript.SQl; |
| **5** | Managing constraints and SQL DML | Lab handouts |
| **6** | Basic SQL | Lab handouts |
| 7 | SQL Group functions, subqueries, and set operations | Lab handouts |
| **8** | **SQL Joins** | **Lab handouts** |
| 9 | SQL Single Row functions | Lab handouts |
| 10 | Application development | Lab handouts |

Table of Contents

[**Introduction to Joins** 4](#_Toc95516665)

[SELECT Statement with NATURAL JOIN 5](#_Toc95516666)

[SELECT Statement with CROSS JOIN 6](#_Toc95516667)

[SELECT Statement with JOIN and USING Clause 6](#_Toc95516668)

[SELECT Statement with JOIN and ON Clause 7](#_Toc95516669)

[SELECT Statement to Join 3 Tables 8](#_Toc95516670)

[SELECT Statement with LEFT and RIGHT OUTER JOIN 9](#_Toc95516671)

[SELECT Statement with FULL OUTER JOIN 10](#_Toc95516672)

[SELECT Statement with SELF JOIN 10](#_Toc95516673)

[Oracle Equijoin and Cartesian Product 12](#_Toc95516674)

[SELECT Statement with EQUIJOIN 12](#_Toc95516675)

[SELECT Statement with Cartesian Product Join 13](#_Toc95516676)

[SELECT Statement with Non-equijoin 14](#_Toc95516677)

# **Introduction to Joins**

Up to now, your experience using SQL has been limited to querying and returning information from one database table at a time

This would not be a problem if all data in the database were stored in only one table

But you know from data modeling that separating data into individual tables and being able to associate the tables with one another is the heart of relational database design

Fortunately, SQL provides join conditions that enable information to be queried from separate tables and combined in one report

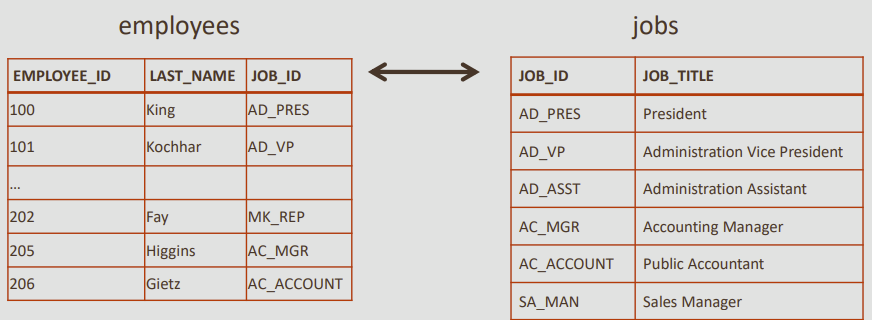
A SQL join clause combines fields from 2 (or more) tables in a relational database as shown in the below figure. The top table is combining departments and employee DEPT\_ID as common attribute in both table.

### 

### SELECT Statement with NATURAL JOIN

A natural join is based on all columns in two tables that have the same name and selects rows from the two tables that have equal values in all matched columns.

In the tables below the employees table has a job\_id column. This is a reference to the column of the same name in the jobs table.



As shown in the sample code below, when using a natural join, it is possible to join the tables without having to explicitly specify the columns in the corresponding table. However, the names and data types of both columns must be the same

| **Example** | **Explanation** |
| --- | --- |
| SELECT first\_name, last\_name, job\_id, job\_title  FROM employees NATURAL JOIN jobs  WHERE department\_id > 80; | This join will return columns from the employees table and their related job\_title from the jobs table based on the common column job\_id.  The WHERE clause was added to apply an additional restriction to one of the tables, to limit the rows of output. The output is shown below |
|  | |
| Here is another example:  SELECT department\_name, city  FROM departments NATURAL JOIN locations; | The departments and locations table both have a column, location\_id, which is used to join the two tables. Notice that the natural join column does not have to appear in the SELECT clause. |

### SELECT Statement with CROSS JOIN

CROSS JOIN joins each row in one table to every row in the other table. The result set represents all possible row combinations from the two tables. If you CROSS JOIN a table with 20 rows with a table with 100 rows, the query will return 2000 rows.

The result set of a CROSS JOIN brings together data from two tables that are not logically related to each other. CROSS JOINS are useful during testing to create large datasets to measure database performance.

| **Example** | **Explanation** |
| --- | --- |
| SELECT last\_name, department\_name  FROM employees CROSS JOIN departments; | The employees table contains 20 rows and the departments table has 8 rows.  Performing a CROSS JOIN will return 160 rows. |

### SELECT Statement with JOIN and USING Clause

In a natural join, if the tables have columns with the same names but different data types, the join causes an error.

To avoid this situation, the join clause can be modified with a USING clause. The USING clause specifies the columns that should be used for the join.

A USING clause is often preferred to a natural join even when the columns have the same data type as well as the same name, because it clearly states exactly which join column is being used.

| **Example** | **Explanation** |
| --- | --- |
| SELECT first\_name, last\_name, department\_id, department\_name FROM employees JOIN departments USING (department\_id); | The columns referenced in the USING clause should not have a qualifier (table name or alias) anywhere in the SQL statement.  If the column in the USING clause has a qualifier, the following error is returned ***ORA-25154: column part of USING clause cannot have qualifier***. |
| SELECT first\_name, last\_name, department\_id, department\_name  FROM employees JOIN departments USING (department\_id)  WHERE last\_name = 'Higgins'; | The USING clause allows us to use WHERE to restrict rows from one or both tables. |

## SELECT Statement with JOIN and ON Clause

What if the columns to be joined have different names, or if the join uses non-equality comparison operators such as <, >, or BETWEEN ?

We can't use USING, so instead we use an ON clause. This allows a greater variety of join conditions to be specified. The ON clause also allows us to use WHERE to restrict rows from one or both tables.

| **Example** | **Explanation** |
| --- | --- |
| SELECT last\_name, job\_title  FROM employees e JOIN jobs j  ON (e.job\_id = j.job\_id);  The example above uses table aliases as a qualifier e.job\_id = j.job\_id, but could also have been written using the table names (employees.job\_id = jobs.job\_id  When using an ON clause on columns with the same name in both tables, you need to add a qualifier (either the table name or alias) otherwise an error will be returned | In this example, the ON clause is used to join the employees table with the jobs table.  A join ON clause is required when the common columns have different names in the two tables |
| SELECT last\_name, job\_title  FROM employees e JOIN jobs j  ON (e.job\_id = j.job\_id)  WHERE last\_name LIKE 'H%'; | The same query with a WHERE clause to restrict the rows selected.  Note: As the job\_id column in the employees and jobs tables have the same name and data type, this query could also be written with a NATURAL JOIN, or a JOIN USING |
| Sometimes you may need to retrieve data from a table that has no corresponding column in another table  Suppose we want to know the grade\_level for each employees salary  The job\_grades table (shown below) does not have a common column with the employees table . Using an ON clause allows us to join the two tables | |
| SELECT last\_name, salary, grade\_level, lowest\_sal, highest\_sal  FROM employees JOIN job\_grades  ON(salary BETWEEN lowest\_sal AND highest\_sal); | This is to know the grade\_level for each employees salary. Even if the job\_grades table does not have a common column with the employees table. |

### SELECT Statement to Join 3 Tables

Suppose we need a report of our employees, their department, and the city where the department is located?

We need to join three tables: employees, departments and locations

Both USING and ON can be used to join three or more tables

| **Example** | **Explanation** |
| --- | --- |
| SELECT last\_name, department\_name AS "Department", city  FROM employees JOIN departments USING (department\_id)  JOIN locations USING (location\_id); | To produce a report of employees, their department, and the city where the department is located, we need to join three tables: employees, departments and locations. |

### SELECT Statement with LEFT and RIGHT OUTER JOIN

A join of two or more tables that returns only the matched rows is called an inner join. All the join discussed are called inner join. When a join returns the unmatched rows as well as the matched rows, it is called an outer join.

| **Example** | **Explanation** |
| --- | --- |
| SELECT e.last\_name, d.department\_id, d.department\_name  FROM employees e  LEFT OUTER JOIN departments d ON (e.department\_id = d.department\_id); | note that the table name listed to the left of the words "left outer join" is referred to as the "left table."  This query will return all employee last names, both those that are assigned to a department and those that are not |
| 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); | This right outer join would return all department IDs and department names, both those that have employees assigned to them and those that do not. |

### SELECT Statement with FULL OUTER JOIN

It is possible to create a join condition to retrieve all matching rows and all unmatched rows from both tables. Using a full outer join solves this problem. The result set of a full outer join includes all rows from a left outer join and all rows from a right outer join combined together without duplication.

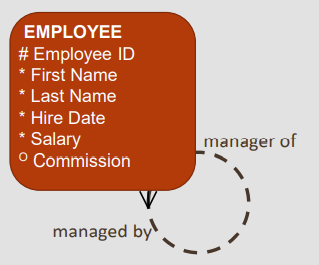
| **Example** | **Explanation** |
| --- | --- |
| SELECT e.last\_name, d.department\_id, d.department\_name  FROM employees e  FULL OUTER JOIN departments d ON (e.department\_id = d.department\_id); | Employees and departments tables are fully joined on their department\_id using FULL OUTER JOIN. |

### SELECT Statement with SELF JOIN

In data modeling, it was sometimes necessary to show an entity with a relationship to itself

For example, an employee can also be a manager.

We showed this using the recursive or "pig's ear" relationship



A self-join is use to join a table to itself as if it was two tables

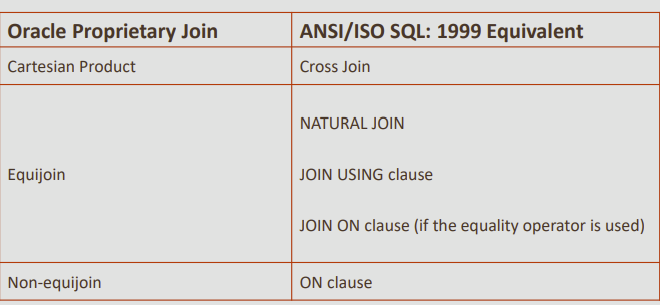
| **Example** | **Explanation** |
| --- | --- |
| SELECT worker.last\_name, worker.manager\_id, manager.last\_name  AS "Manager name"  FROM employees worker JOIN employees manager  ON (worker.manager\_id = manager.employee\_id);  To join a table to itself, the table is given two names or aliases. This will make the database "think" that there are two tables shown below | “Manager name” is alias names that relate to the data's association with that table. Manager\_id in the worker table is equal to employee\_id in the manager table. |

## Oracle Equijoin and Cartesian Product

The previous section looked at querying and returning data from more than one table in a relational database using ANSI/ISO SQL: 99 syntax

Legacy versions of Oracle databases required joins to use Oracle Proprietary join syntax, and many of these older databases are still in use

This lesson introduces Oracle Proprietary join syntax for Equijoins and Cartesian Product, and their ANSI/ISO SQL: 99 counterparts

Join Comparison 

To query data from more than one table using the Oracle proprietary syntax, use a join condition in the WHERE clause

*SELECT table1.column, table2.column*

*FROM table1, table2*

*WHERE table1.column1 = table2.column2;*

### SELECT Statement with EQUIJOIN

| **Example** | **Explanation** |
| --- | --- |
| SELECT employees.last\_name, employees.job\_id, jobs.job\_title  FROM employees, jobs  WHERE employees.job\_id = jobs.job\_id; | Joins 2 tables using their common column job\_id in WHERE Clause |
| SELECT employees.last\_name, departments.department\_name  FROM employees, departments  WHERE employees.department\_id = departments.department\_id; | Joins 2 tables using their common column department\_id |
| **Restricting Rows In a Join** | |
| SELECT employees.last\_name, employees.job\_id, jobs.job\_title FROM employees, jobs  WHERE employees.job\_id = jobs.job\_id  AND employees.department\_id = 80; | As with single-table queries, the WHERE clause can be used to restrict the rows considered in one or more tables of the join.  The query shown uses the AND operator to restrict the rows returned. |
| **Aliases** | |
| SELECT last\_name, e.job\_id, job\_title  FROM employees e, jobs j  WHERE e.job\_id = j.job\_id AND department\_id = 80; | Table aliases are used in the query. But when column names are not duplicated between two tables, you do not need to add the table name or alias to the column name. |
| **Join 3 Tables** | |
| SELECT last\_name, city  FROM employees e, departments d, locations l  WHERE e.department\_id = d.department\_id  AND d.location\_id = l.location\_id | To produce a report of employees, their department, and the city where the department is located, we need to join three tables: employees, departments and locations. |

### SELECT Statement with Cartesian Product Join

* If two tables in a join query have no join condition specified in the WHERE clause or the join condition is invalid, the Oracle Server returns the Cartesian product of the two tables.
* This is a combination of each row of one table with each row of the other.
* A Cartesian product is equivalent to an ANSI CROSS JOIN.
* To avoid a Cartesian product, always include a valid join condition in a WHERE clause.

| **Example** | **Explanation** |
| --- | --- |
| SELECT employees.last\_name, departments.department\_name  FROM employees, departments; | In this query, the join condition has been omitted |

### SELECT Statement with Non-equijoin

| **Example** | **Explanation** |
| --- | --- |
| SELECT last\_name, salary, grade\_level, lowest\_sal, highest\_sal  FROM employees, job\_grades  WHERE (salary BETWEEN lowest\_sal AND highest\_sal); | To know the grade\_level for each employee's salary, while the job\_grades table does not have a common column with the employees table, we can use a nonequijoin to join the two tables. |

# Exercises

Apply the concepts learn in this lab on company database for retrieving following information.

1. Retrieve the name and address of all employees who work for the ‘Research’ department without using join
2. Modify query 1 and observe results by using join
3. Modify query 1 and use subquery instead of join or equijoin
4. Retrieve the last name of employees and their supervisors (natural join)
5. Modify query 4 to display all employees with supervisor and also that employee where a supervisor is not assigned (left outer join)
6. Modify query 4 to display all the supervisors with and without employees assigned to them (Right outer join)
7. Modify query 4 to display all employees with and without supervisors and all supervisors with and without employees (Full outer join)
8. For every project located in ‘Stafford’, list the project number, the controlling department number, and the department manager’s last name, address, and birth date.

select e.employee\_id, d.department\_name

from employees e, departments d

where e.department\_id = d.department\_id

and d.department\_name = 'IT';

select employee\_id, department\_name

from employees join departments

using (department\_id)

where department\_name = 'IT';

select e.employee\_id, d.department\_name, d.department\_id

from employees e, departments d

where e.department\_id = d.department\_id

and e.department\_id =

(select department\_id from departments where department\_name = 'IT' );