##### National University of Computer & Emerging Sciences, Karachi



##### Computer Science Department

**Spring 2023, Lab Manual – 05**

|  |  |
| --- | --- |
| **Course Code: CL-2005** | **Course: Database Systems Lab** |
| **Instructor(s):** | **Shakir Hussain** |

# Contents:

1. Introduction to Joins
2. Types of Joins
3. Introduction to Outer Joins and Its Types
4. Introduction to Set Operator
5. Types of Set Operator
6. Exercise

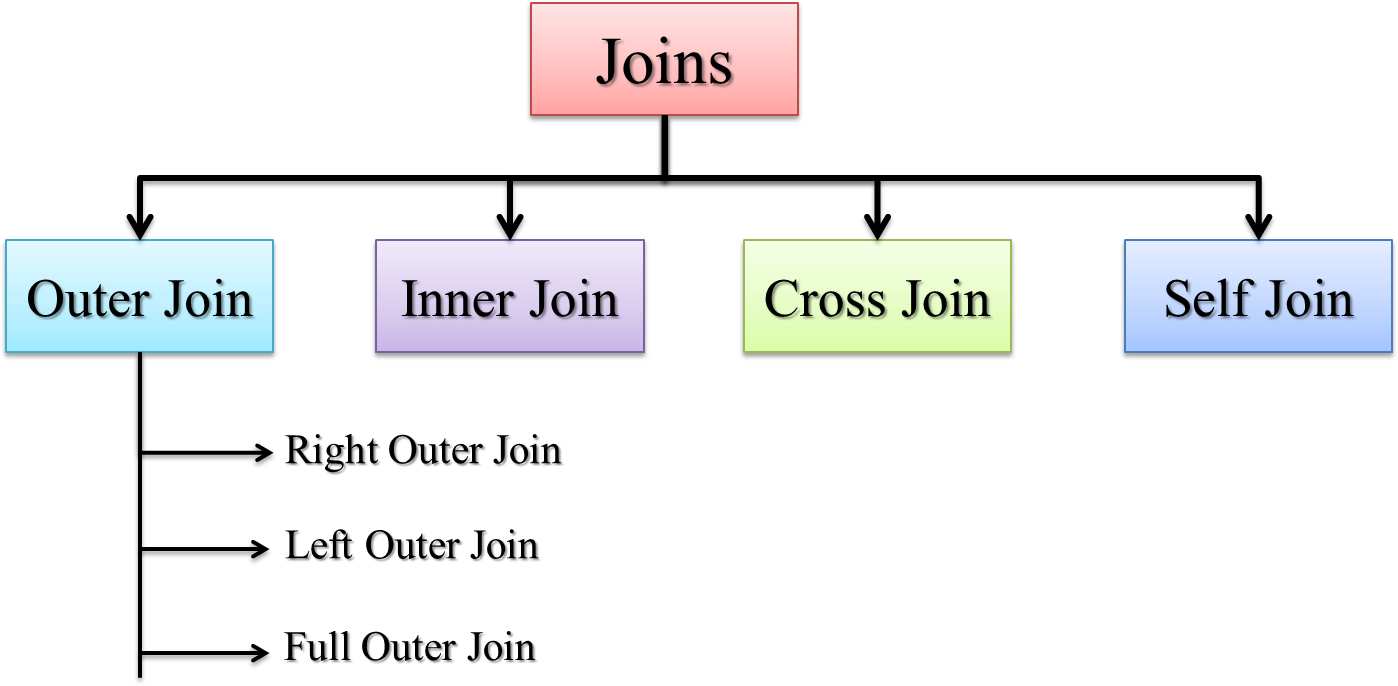
# INTRODUCTION TO JOIN

The JOIN keyword is used in an SQL statement to query data from two or more tables based on a relationship between certain columns in these tables.

# TYPES OF JOINS:

Following are the types of joins. They are:

* Cross Join / Cartesian Join
* Inner Join / Equity Join
* Outer Join
  + Left Outer
  + Right Outer
  + Full Outer
* Self-Join



## Cross Join / Cartesian Join:

In a Cartesian join, also called a Cartesian product or cross join, each record in the first table is matched with each record in the second table.

**(# rows in Table 1) \* (# rows in Table 2)**

****

**Syntax for Cross Join/Cartesian Join:**

SELECT \* FROM TABLE1, TABLE2;

##### ISO Standard:

SELECT \* FROM TABLE1 CROSS JOIN TABLE2;

## Inner Join / Equality Joins:

If the join contains an equality condition, it is also called Equi Join, Natural Join, Inner Join.

### Syntax For Inner Join:

SELECT TABLE1\_COLUMN, TABLE2\_COLUMN FROM TABLE1 T1, TABLE2 T2 WHERE T1. TABLE1\_COLUMN = T2. TABLE2\_COLUMN;

##### Example

* + **To retrieve the employee name, their job and department name, we need to extract data from two tables, EMP and DEPT:**
    - SELECT E.ENAME, E.JOB, D.DNAME FROM EMP E, DEPT D WHERE E.DEPTNO = D.DEPTNO;

##### The SQL-1999 standard:

* + - SELECT ENAME, JOB, DNAME FROM EMP NATURAL JOIN DEPT;

## Using Clause:

* No matter how many common columns are available in the tables, NATURAL JOIN will join with all the common columns.
* Use USING clause to join with specified columns.

### Syntax for Using Clause:

SELECT TABLE1\_COLUMN, TABLE2\_COLUMN FROM TABLE1 JOIN TABLE2 USING (TABLE2\_COLUMN, TABLE1\_COLUMN)

##### Example

SELECT EMPNO, ENAME, MGR, DNAME FROM EMP JOIN DEPT USING (DEPTNO, MGR);

## Self-Join:

When a table is joined to itself then it is called as Self join or in less words we

Can just say “joining a table to itself is called self-join”.

### Syntax for Self-join:

SELECT T1.TABLE1\_COLUMN,T2. TABLE2\_COLUMN FROM TABLE T1,TABLE T2 WHERE T1.COLUMN = T2.COLUMN;

##### Example

SELECT WORKER.ENAME, MANAGER.ENAME FROM EMP WORKER, EMP MANAGER WHERE WORKER.MGR = MANAGER.EMPNO;

# INTRODUCTION TO OUTER JOIN & ITS TYPES

* Use Outer join to return records which don’t have direct match.
* In outer join operation, all records from the source table included in the result even though they don't satisfy the join condition.

### Syntax for Outer Join:

SELECT column names from both tables FROM table name 1 LEFT|RIGHT|FULL OUTER JOIN table name 2 on condition;

Types of Outer Joins:

Outer joins are classified into three types:

* + Left Outer Join
  + Right Outer Join
  + Full Outer Join

## Left Outer Join:

The left outer join produces a table that contains the matched data from the two tables, as well as the remaining rows of the left table and null from the columns of the right table.

### Syntax for Left Outer Join:

SELECT T1. TABLE1\_COLUMN, T2. TABLE2\_COLUMN FROM TABLE1 T1, TABLE2 T2

WHERE T1.TABLE1\_COLUMN = T2. TABLE2\_COLUMN(+);

##### Example

SELECT E.ENAME, D.DEPTNO, D.DNAME FROM EMP E, DEPT D WHERE E.DEPTNO

= D.DEPTNO (+);

**NOTE**: The outer join operator appears on only that side that has information missing.

##### The SQL-1999 standard:

SELECT T1. TABLE1\_COLUMN, T2. TABLE2\_COLUMN FROM TABLE1 T1 LEFT OUTER JOIN TABLE2 T2 ON T1.TABLE1\_COLUMN = T2. TABLE2\_COLUMN;

##### Example

SELECT E.ENAME, D.DEPTNO, D.DNAME FROM EMP E LEFT OUTER JOIN DEPT D ON (E.DEPTNO = D.DEPTNO);

## Right Outer Join:

The right outer join returns a table with the matched data from the two tables being joined, then the remaining rows of the right table and null for the remaining left table's columns.

### Syntax for Right Outer Join:

SELECT T1. TABLE1\_COLUMN, T2. TABLE2\_COLUMN FROM TABLE1 T1, TABLE2 T2

WHERE T1.TABLE1\_COLUMN (+)= T2. TABLE2\_COLUMN;

### Example:

SELECT E.ENAME, D.DEPTNO, D.DNAME FROM EMP E, DEPT D WHERE E.DEPTNO(+) = D.DEPTNO;

##### SQL-1999 standard:

SELECT T1. TABLE1\_COLUMN, T2. TABLE2\_COLUMN FROM TABLE1 T1 RIGHT OUTER JOIN TABLE2 T2 ON T1.TABLE1\_COLUMN = T2. TABLE2\_COLUMN;

##### Example:

SELECT E.ENAME, D.DEPTNO, D.DNAME FROM EMP E RIGHT OUTER JOIN DEPT D ON (E.DEPTNO = D.DEPTNO);

## Full Outer Join:

The full outer join returns a table with the matched data of two table then remaining rows of both left table and then the right table.

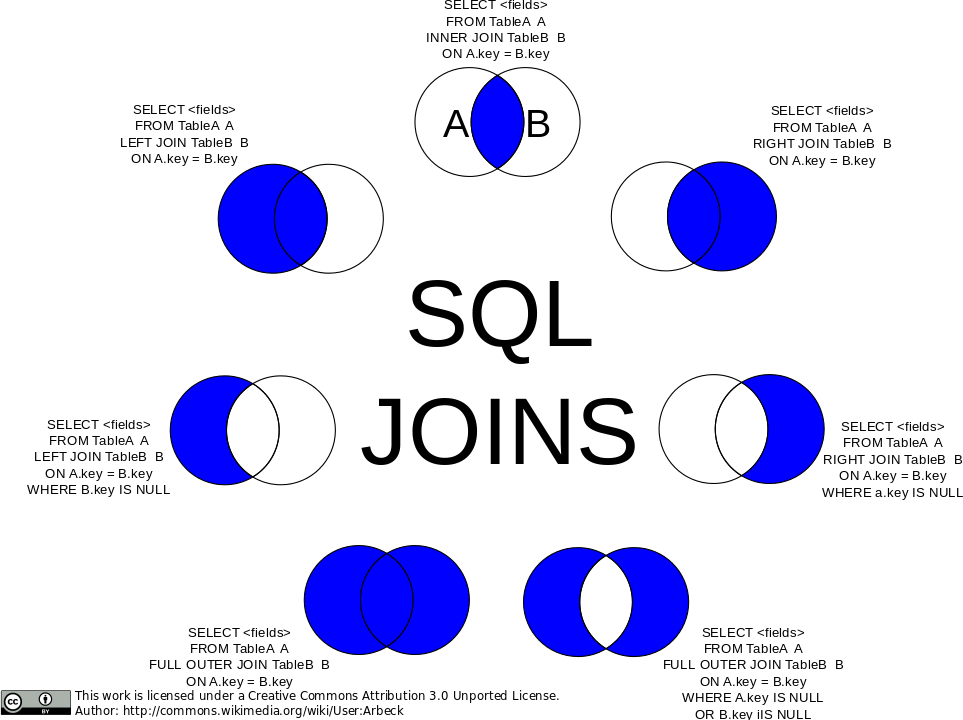
### Syntax for Full Outer Join:

SELECT T1. TABLE1\_COLUMN, T2. TABLE2\_COLUMN FROM TABLE1 T1 FULL OUTER JOIN TABLE2 T2 ON T1.TABLE1\_COLUMN = T2. TABLE2\_COLUMN;

##### Example

SELECT E.ENAME, D.DEPTNO, D.DNAME FROM EMP E FULL OUTER JOIN DEPT D ON (E.DEPTNO = D.DEPTNO);

Conclusion for Joins:



# INTRODUCTION TO SET OPERATOR

Set operators are used to join the results of two (or more) SELECT statements. The SET operators available in Oracle 11g are UNION, UNION ALL, INTERSECT and MINUS.

All of the SET operators have the same order of precedence. Instead, Oracle evaluates queries from left to right or top to bottom during execution. If parentheses are used explicitly, the order may change because parentheses take precedence over dangling operators.

# TYPES OF SET OPERATOR

Following are the types of operators that are used for set in oracle. They are:

* + Union
  + Union all
  + Intersect
  + Minus

#### Union Operator:

The SQL Union function joins the results of two or more SQL SELECT queries. The number of datatypes and columns in both tables on which the UNION operation is performed must be the same in order to perform the union operation. The duplicate rows are removed from the result of the union operation.

**Diagrammatic view of Union operator**

**Syntax for Union Operator**

SELECT TABLE1\_COLUMN, TABLE2\_COLUMN FROM TABLE1 **UNION** SELECT TABLE1\_COLUMN\_ID, TABLE2\_COLUMN\_ID FROM TABLE2

##### Example

SELECT employee\_id, job\_id FROM employees **UNION** SELECT employee\_id, job\_id FROM job\_history;

Union All Operator:

With one exception, UNION and UNION ALL operate in a similar manner. UNION ALL, on the other hand, returns the result set without eliminating duplication or sorting the data.

**Diagrammatic view of Union all operator**

### Syntax for Union All Operator

SELECT TABLE1\_COLUMN, TABLE2\_COLUMN FROM TABLE1 UNION ALL SELECT TABLE1\_COLUMN\_ID, TABLE2\_COLUMN\_ID FROM TABLE2

##### Example

SELECT employee\_id, job\_id, department\_id FROM employees **UNION ALL**

SELECT employee\_id, job\_id, department\_id FROM job\_history

Intersect Operator

It's used to join two SELECT statements together. The common rows from both SELECT statements are returned by the Intersect procedure. The number of datatypes and columns in the Intersect operation must be the same. There are no duplicates, and the data is arranged in ascending order by default.

**Diagrammatic view of Intersect operator**

### Syntax for Intersect Operator

SELECT TABLE1\_COLUMN, TABLE2\_COLUMN FROM TABLE1 **INTERSECT** SELECT TABLE1\_COLUMN\_ID, TABLE2\_COLUMN\_ID FROM TABLE2

##### Example

SELECT employee\_id, job\_id FROM employees **INTERSECT** SELECT employee\_id, job\_id FROM job\_history;

Minus Operator

It combines the results of two SELECT statements into a single statement. The minus operator is used to show rows that are present in the first query but not in the second. There are no duplicates, and the data is sorted ascending by default.

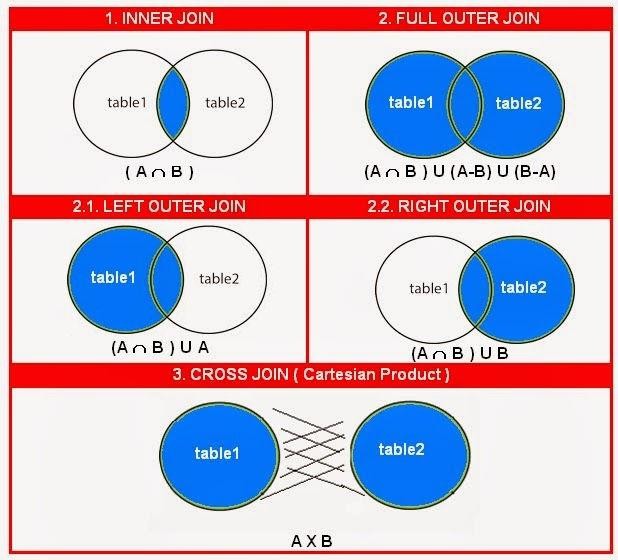
**Diagrammatic view of Minus operator**

### Syntax for Minus Operator

SELECT TABLE1\_COLUMN, TABLE2\_COLUMN FROM TABLE1 **MINUS** SELECT TABLE1\_COLUMN\_ID, TABLE2\_COLUMN\_ID FROM TABLE2

##### Example

SELECT employee\_id, job\_id FROM employees **MINUS** SELECT employee\_id, job\_id FROM job\_history

How to implement joins as set operator?

**Additional Information**

**Maintaining referential integrity:**

In the context of relational databases, the **"ON DELETE"** clause is used in conjunction with foreign keys to define what action should be taken when a record in the referenced (parent) table is deleted. The **ON DELETE** clause helps maintain referential integrity, which ensures that relationships between tables remain consistent and valid. There are several options for the **ON DELETE** clause:

1. **CASCADE:** When you specify "**ON DELETE CASCADE**" it means that when a record in the parent table is deleted, all related records in the child table(s) will also be automatically deleted. This cascading effect continues through all levels of child tables that have foreign keys referencing the parent table. This option is useful when you want to ensure that no orphaned records are left behind.

**Syntax:**

FOREIGN KEY (order\_id) REFERENCES orders(order\_id) ON DELETE CASCADE

1. **SET NULL:** With "**ON DELETE SET NULL"** if a record in the parent table is deleted, the corresponding foreign key values in the child table(s) are set to **NULL**. This is useful when you have nullable foreign key columns in the child table, and you want to indicate that the relationship no longer exists without deleting the child records.

**Syntax:**

FOREIGN KEY (order\_id) REFERENCES orders(order\_id) ON DELETE SET NULL

1. **SET DEFAULT:** Like "**ON DELETE SET NULL**", "**ON DELETE SET DEFAULT"** sets the foreign key values in the child table(s) to their default values when a record in the parent table is deleted. This option is typically used when the foreign key column(s) have default values defined.

**Syntax:**

FOREIGN KEY (order\_id) REFERENCES orders(order\_id) ON DELETE SET DEFAULT (will set value to default value set while creating table)

1. **NO ACTION/RESTRICT**: When you specify "**ON DELETE NO ACTION**" or "**ON DELETE RESTRICT"** it means that the database will prevent the deletion of a parent record if there are related child records that reference it. This option enforces referential integrity by prohibiting actions that would break the foreign key relationship.

**Syntax**:

FOREIGN KEY (order\_id) REFERENCES orders(order\_id) ON DELETE NO ACTION

1. **SET (value):** You can also specify a specific value (other than **NULL** or **DEFAULT**) to be set in the child table(s) when a parent record is deleted. This can be a specific constant value or an expression.

**Syntax:**

FOREIGN KEY (order\_id) REFERENCES orders(order\_id) ON DELETE SET -1

# LAB EXERCISE

1. Write a query to list the name, job name, department name, salary of the employees according to the department in ascending order.
2. Write a query to list the department where at least two employees are working.
3. Fetch all the records where the salary of employee is less than the lowest salary.
4. Write a query to list the name, job name, annual salary, department id, department name and city who earn 60000 in a year or not working as an ANALYST.
5. Write a query to print details of the employees who are also Managers.
6. List department number, Department name for all the departments in which there are no employees in the department.
7. Display employee name, salary, department name where all employees(unique) have matching departments.
8. Display name, job ID, department name, street address and city of the employee in which there is no state province.
9. Write an SQL query to show records from one table that another table does not have.
10. Display all employees who belong to the country “US” but not belongs to state province Washington.

**Home Tasks**

(You can use any join, other than cross **OR** cartesian join)

1. Create a table named **Students** having attributes (Student\_id, FullName, Age, Gender, EnrollmentDate, and CGPA). **Student\_id** being primary key.
2. Create another table named **Courses** having attributes (Course\_id, course\_title, Credits, and OfferedinSemester). **Course\_id** being primary.
3. Make another table **Transcript** having attributes (student\_id, course\_id, GPA and Grade).
4. Grade ranges from ‘A’ to ‘F’ such that if GPA is:
   1. Less than 2 -> grade F.
   2. 2 -> grade E.
   3. 2.5 -> grade D.
   4. 3 -> grade C.
   5. 3.5 -> grade B.
   6. 4 -> grade A.
5. Create a relationship between Courses ->Transcript, Courses being the parent table. Create another relationship between Students->Transcript, Students being the parent table.
6. Filled the students table with 6 rows of data, the first containing your own data, and the remaining 5 containing random data (excluding your classmates).
7. Fill the Courses table with data from your curriculum.
8. Fill in the transcript table in such a way that each student must be enrolled in at least 3 courses.
9. Make sure that the GPA in **Transcript** is never less than 1 or greater than 4. Also, make sure that the Grade lies within the range specified above in **task#4**.
10. The CGPA in the student table should be the average GPA of all courses the student is enrolled in.
11. Display details of each student along with the details of courses they are enrolled in, their GPA, CGPA, GRADE etc. In short, display all details in the 3 tables.
12. Display the number of courses each student is enrolled in, along with their CGPA, Name and Student\_id in ascending order.
13. Delete the details of students with less than 2 CGPA and **either** nullify the values in referencing table **OR** delete the records completely.
14. Display the number of students enrolled in each Course, along with the Course\_title and Course\_id in ascending order.
15. Count the number of students having the same Grade.
16. List courses which don’t have any students enrolled in them. Similarly, list students which aren’t enrolled in any courses.