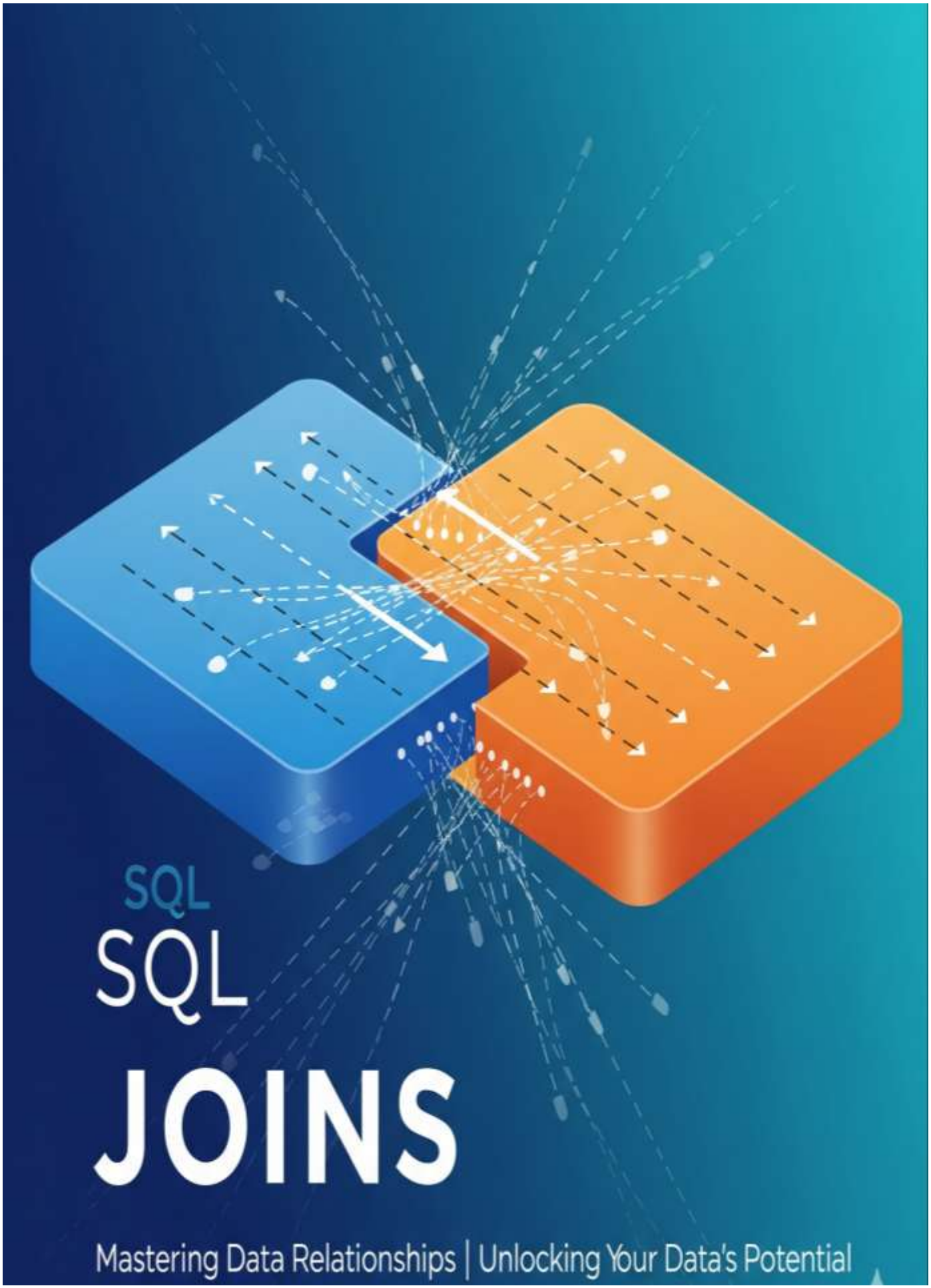
SQL JOINS


SQL
SQL
JOINS
Mastering Data Relationships | Unlocking Your Data's Potential

SQL Joins

SQL joins serve as the weaver's tool, allowing you to seamlessly merge data from multiple tables based on common threads. So explore this section to learn how to use JOIN command.

- JOIN
- Outer Join
- Left Join
- Right Join
- Full Join
- Cross Join
- Self-Join
- UPDATE with JOIN
- DELETE JOIN
- Recursive Join

SQL Joins (Inner, Left, Right and Full Join)

SQL joins are fundamental tools for combining data from multiple tables in relational databases.

- For example, consider two tables where one table (say Student) has student information with id as a key and other table (say Marks) has information about marks of every student id. Now to display the marks of every student with name, we need to join the two tables.
- Please remember, we store data into multiple tables as part of database normalization to avoid anomalies and redundancies.

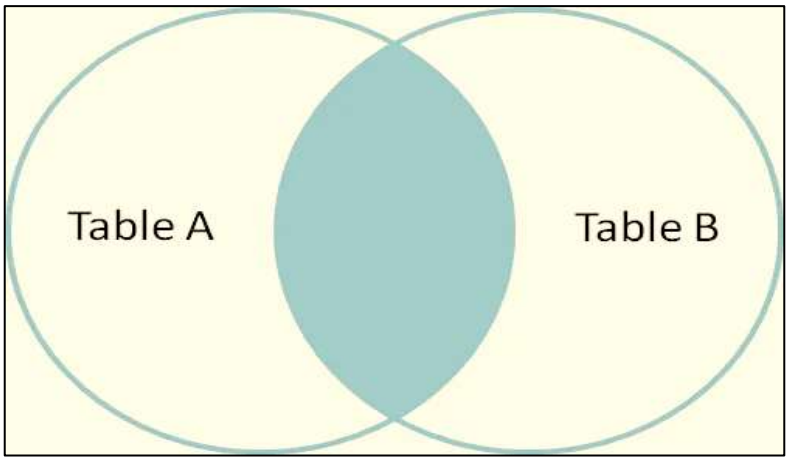
Types of SQL Joins

1. SQL INNER JOIN

The INNER JOIN keyword sele**cts** all rows from both the tables as long as the condition is satisfied. This keyword will create the result set by combining all rows from both the tables where the condition satisfies i.e value of the common field will be the same.

Syntax:

SELECT table1.column1,table1.column2,table2.column1,.... FROM table1 INNER JOIN table2 ON table1.matching_column = table2.matching_column;



Example of INNER JOIN

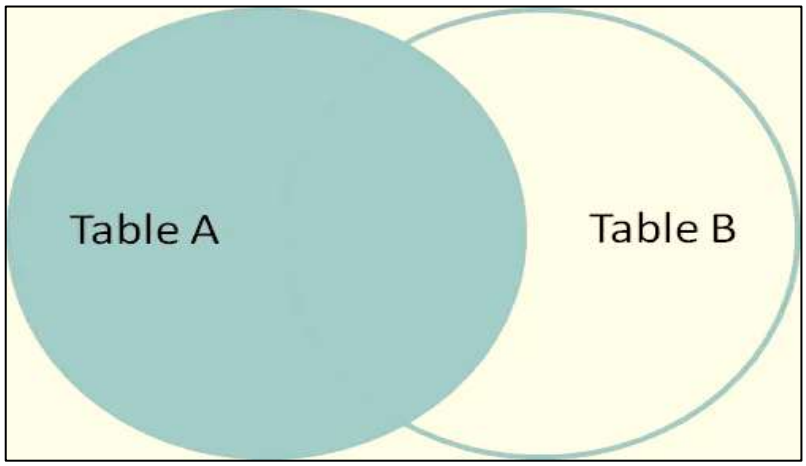
COURSE_ID	NAME	Age
1	HARSH	18
2	PRATIK	19
2	RIYANKA	20
3	DEEP	18
1	SAPTARHI	19

2. SQL LEFT JOIN

A LEFT JOIN returns all rows from the left table, along with matching rows from the right table. If there is no match, NULL values are returned for columns from the right table. LEFT JOIN is also known as LEFT OUTER JOIN.

Syntax

SELECT table1.column1,table1.column2,table2.column1,....
FROM table1
LEFT JOIN table2
ON table1.matching_column = table2.matching_column;



LEFT JOIN Example

In this example, the LEFT JOIN retrieves all rows from the Student table and the matching rows from the StudentCourse table based on the **`ROLL_NO`** column.

Query:

SELECT Student.NAME,StudentCourse.COURSE_ID
FROM Student
LEFT JOIN StudentCourse
ON StudentCourse.ROLL_NO = Student.ROLL_NO;

Output

NAME	COURSE_ID
HARSH	1
PRATIK	2
RIYANKA	2
DEEP	3
SAPTARHI	1
DHANRAJ	*NULL*
ROHIT	*NULL*
NIRAJ	*NULL*

3. SQL RIGHT JOIN

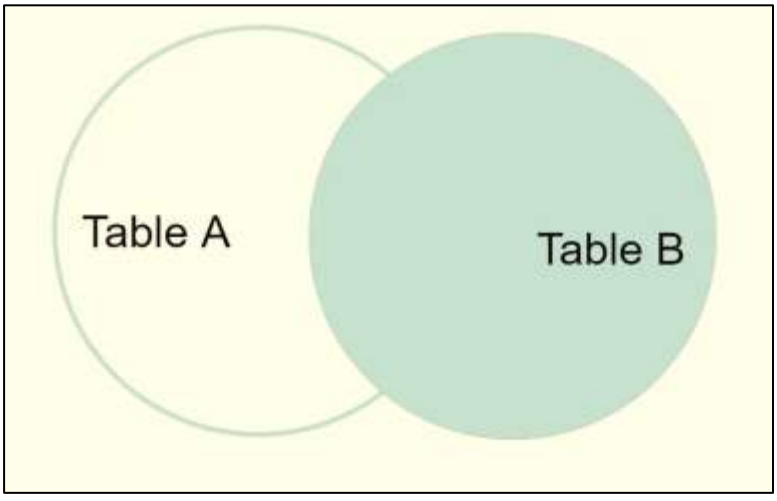
RIGHT JOIN returns all the rows of the table on the right side of the join and matching rows for the table on the left side of the join. It is very similar to LEFT JOIN for the rows for which there is no matching row on the left side, the result-set will contain null. RIGHT JOIN is also known as RIGHT OUTER JOIN.

Syntax

```
SELECT table1.column1,table1.column2,table2.column1,....
FROM table1
RIGHT JOIN table2
ON table1.matching_column = table2.matching_column;
```

Key Terms

- **table1**: First table.
- **table2**: Second table
- **matching_column**: Column common to both the tables.



RIGHT JOIN Example

In this example, the RIGHT JOIN retrieves all rows from the StudentCourse table and the matching rows from the Student table based on the `ROLL_NO` column.

Query:

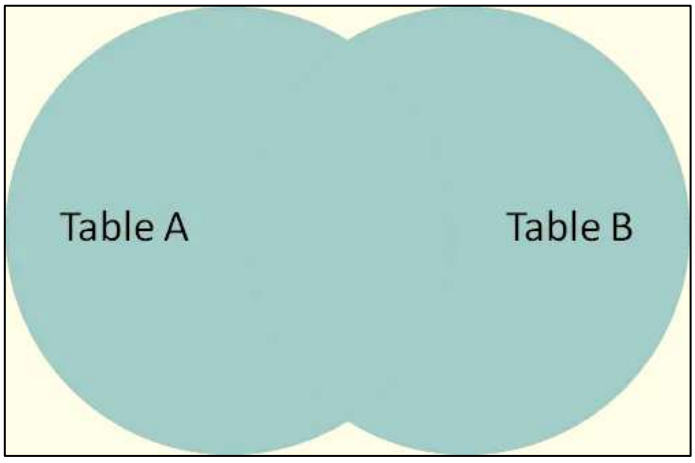
```
SELECT Student.NAME,StudentCourse.COURSE_ID
FROM Student
RIGHT JOIN StudentCourse
ON StudentCourse.ROLL_NO = Student.ROLL_NO;
```

Output

NAME	COURSE_ID
HARSH	1
PRATIK	2
RIYANKA	2
DEEP	3
SAPTARHI	1
NULL	4
NULL	5
NULL	4

4. SQL FULL JOIN

FULL JOIN creates the result-set by combining results of both LEFT JOIN and RIGHT JOIN. The result-set will contain all the rows from both tables. For the rows for which there is no matching, the result-set will contain NULL values.



Syntax

SELECT table1.column1,table1.column2,table2.column1,....
FROM table1
FULL JOIN table2
ON table1.matching_column = table2.matching_column;

Key Terms

- **table1**: First table.
- **table2**: Second table
- **matching_column**: Column common to both the tables.

FULL JOIN Example

This example demonstrates the use of a FULL JOIN, which combines the results of both LEFT JOIN and RIGHT JOIN. The query retrieves all rows from the Student and StudentCourse tables. If a record in one table does not have a matching record in the other table, the result set will include that record with `NULL` values for the missing fields

Query:

SELECT Student.NAME,StudentCourse.COURSE_ID
FROM Student
FULL JOIN StudentCourse
ON StudentCourse.ROLL_NO = Student.ROLL_NO;

Output

NAME	COURSE_ID
HARSH	1
PRATIK	2
RIYANKA	2
DEEP	3
SAPTARHI	1
DHANRAJ	NULL
ROHIT	NULL
NIRAJ	NULL

NAME	COURSE_ID
NULL	4
NULL	5
NULL	4

5. SQL Natural Join

A Natural Join is a type of INNER JOIN that automatically joins two tables based on columns with the same name and data type. It returns only the rows where the values in the common columns match.

- It returns rows where the values in these common columns are the same in both tables.
- Common columns appear only once in the result, even if they exist in both tables.
- Unlike a CROSS JOIN, which creates all possible combinations of rows, a Natural Join only includes rows with matching values

Example:

Look at the two tables below: Employee and Department

Employee		
Emp_id	Emp_name	Dept_id
1	Ram	10
2	Jon	30
3	Bob	50

Department	
Dept_id	**Dept_name**
10	IT
30	HR
40	TIS

Find all Employees and their respective departments.
(Employee) ? (Department)

Output:

Emp_id	Emp_name	Dept_id	Dept_id	Dept_name
1	Ram	10	10	IT
2	Jon	30	30	HR

SQL Outer Join

Outer Joins allow retrieval of rows from two or more tables based on a related column. Unlike Inner Joins, Outer Joins also include rows that do not have a corresponding match in one or both of the tables. This makes them especially useful when dealing with incomplete data, ensuring that no records are missed during reporting or analysis.

Types of Outer Joins
There are three main types of Outer Joins in SQL:

- LEFT OUTER JOIN (or LEFT JOIN)
- RIGHT OUTER JOIN (or RIGHT JOIN)
- FULL OUTER JOIN

Each of these join types handles unmatched rows differently and understanding how they work will help you use them effectively in your SQL queries.
Let's Consider the two tables, Employees and Departments for understanding all types of Outer Joins with examples.

1. **Employees Table**

```
CREATE TABLE Employees (
    EmployeeID INT PRIMARY KEY,
    Name VARCHAR(50),
    DepartmentID INT );

INSERT INTO Employees (EmployeeID, Name, DepartmentID) VALUES
(1, 'John', 101),
(2, 'Sarah', 102),
(3, 'Michael', NULL),
(4, 'Emma', 103);
```

EmployeeID	Name	DepartmentID
1	John	101

EmployeeID	Name	DepartmentID
2	Sarah	102
3	Michael	NULL
4	Emma	103

2. Departments Table

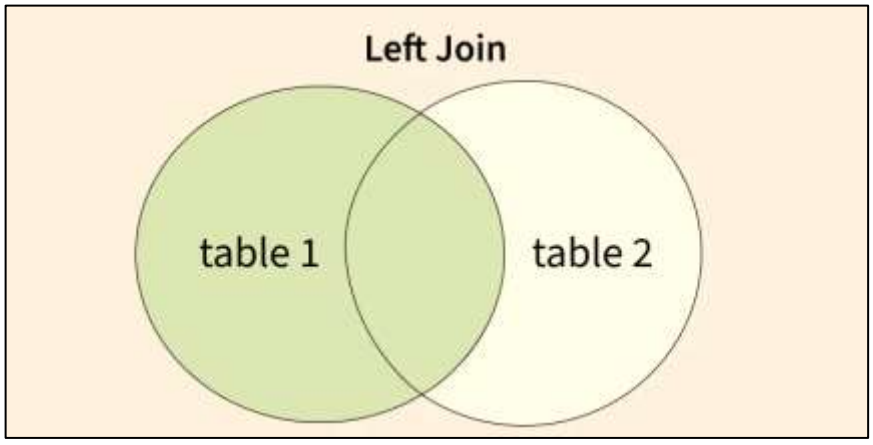
```
CREATE TABLE Departments (
    DepartmentID INT PRIMARY KEY,
    DepartmentName VARCHAR(50));

INSERT INTO Departments (DepartmentID, DepartmentName) VALUES
(101, 'HR'),
(102, 'IT'),
(103, 'Marketing')
(104, 'Finance');
```

DepartmentID	DepartmentName
101	HR
102	IT
103	Marketing
104	Finance

1. LEFT OUTER JOIN (or LEFT JOIN)

LEFT OUTER JOIN (referred to as LEFT JOIN) returns all rows from the left table and the matching rows from the right table. If there is no match, the result will include NULL values for columns from the right table.



Syntax:

```
SELECT table1.column1, table1.column2, table2.column1, ...
FROM table1
LEFT JOIN table2
ON table1.matching_column = table2.matching_column;
```

Example: To retrieve all employees along with their respective departments, even if they don't belong to any department (i.e., the department is NULL), we can use the LEFT OUTER JOIN

Query:

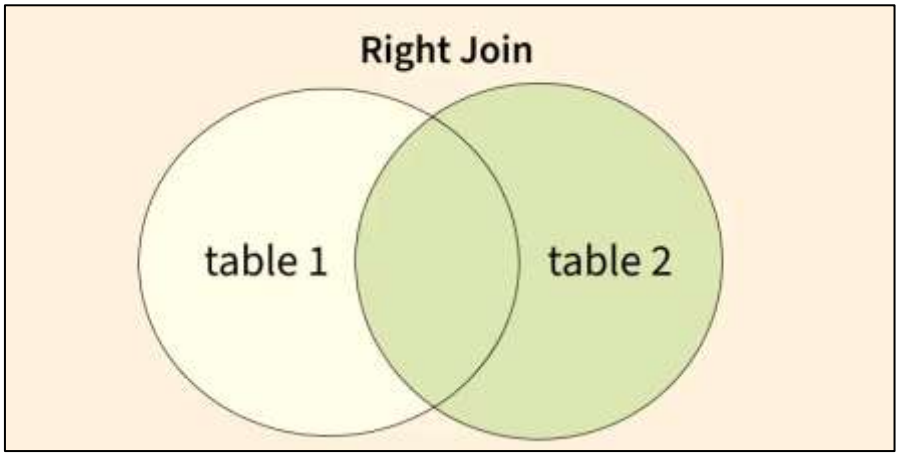
```
SELECT Employees.Name, Employees.DepartmentID, Departments.DepartmentName
FROM Employees
LEFT JOIN Departments
ON Employees.DepartmentID = Departments.DepartmentID;
```

Output

Name	DepartmentID	DepartmentName
John	101	HR
Sarah	102	IT
Michael	NULL	NULL
Emma	103	Marketing

2. RIGHT OUTER JOIN (RIGHT JOIN)

RIGHT OUTER JOIN (often called RIGHT JOIN) returns all rows from the right table and the matching rows from the left table. If there is no match, the result will include NULL values for columns from the left table.



Syntax:

```
SELECT table1.column1, table1.column2, table2.column1, ...
FROM table1
```

```
RIGHT JOIN table2
ON table1.matching_column = table2.matching_column;
```

Example: Let's now look at a RIGHT OUTER JOIN on the Employees and Departments tables. Suppose we want to retrieve all

Query:

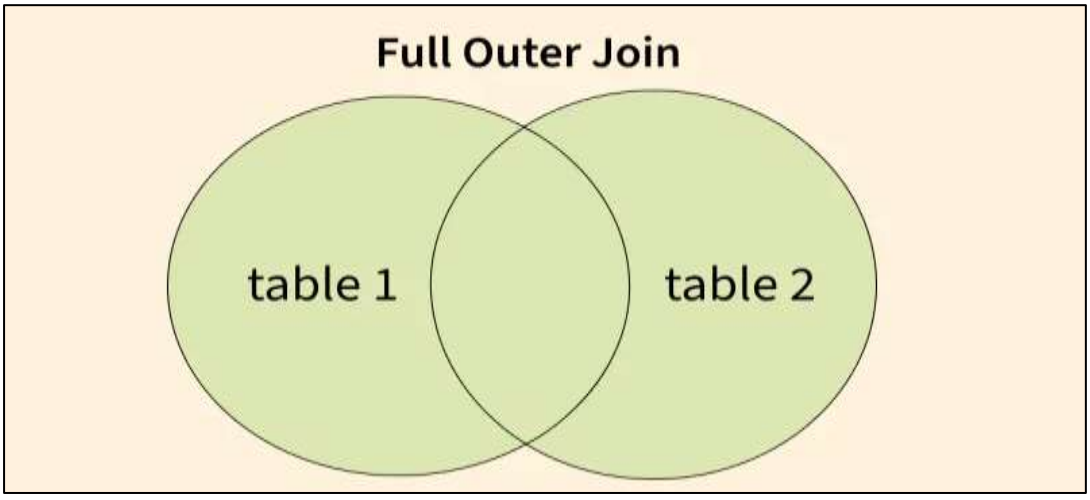
```
SELECT Employees.Name, Employees.DepartmentID, Departments.DepartmentName
FROM Employees
RIGHT JOIN Departments
ON Employees.DepartmentID = Departments.DepartmentID;
```

Output

Name	DepartmentID	DepartmentName
John	101	HR
Sarah	102	IT
Emma	103	Marketing
NULL	NULL	Finance

3. FULL OUTER JOIN

FULL OUTER JOIN returns all rows when there is a match in either the left or right table. If there is no match, the result will include NULL for the missing side of the table. Essentially, it combines the results of both LEFT JOIN and RIGHT JOIN.



Syntax:

```
SELECT table1.column1, table1.column2, table2.column1, ...
FROM table1
FULL JOIN table2
ON table1.matching_column = table2.matching_column;
```

Example: Let's now use a FULL OUTER JOIN to get all employees and all departments, regardless of whether an employee belongs to a department or a department has employees.

Query:

```
SELECT Employees.Name, Employees.DepartmentID, Departments.DepartmentName
FROM Employees
FULL JOIN Departments
ON Employees.DepartmentID = Departments.DepartmentID;
```

Output

Name	DepartmentID	DepartmentName
John	101	HR
Sarah	102	IT
Michael	NULL	NULL
Emma	103	Marketing
NULL	NULL	Finance

Difference Between INNER JOIN and OUTER JOIN

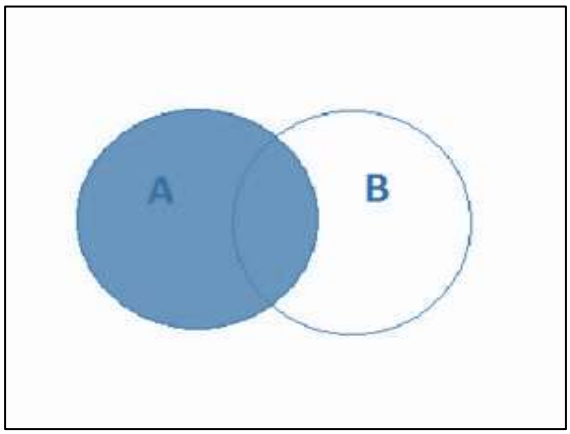
INNER JOIN	OUTER JOIN
Returns only records with matching values in both tables.	Returns records even if there is no match in one or both tables.
Excludes non-matching rows.	Includes non-matching rows (NULLs fill missing values).
Produces a smaller, filtered result set.	Produces a larger, more comprehensive result set.
Only one type: INNER JOIN.	Three types: LEFT, RIGHT, FULL OUTER JOIN.
Used when focusing strictly on relationships between tables.	Used when dealing with incomplete data or ensuring no records are lost.
Common in transactional queries.	Common in reporting, analytics and data integration tasks.

SQL LEFT JOIN

In SQL, the LEFT JOIN (also called LEFT OUTER JOIN) retrieves all records from the left table and only the matching records from the right table. If no match is found in the right table, the query will return `NULL` values for its columns.

- Returns all rows from the left table.
- Includes only matching rows from the right table.
- Non-matching rows in the right table are represented as `NULL`

SQL LEFT JOIN Venn Diagram



Syntax:

```
SELECT column_name(s)
FROM tableA
LEFT JOIN tableB ON tableA.column_name = tableB.column_name;
```

Examples of SQL LEFT JOIN

Let's look at an example of **LEFT JOIN in SQL** to understand it better. Consider two tables: `Emp` (employees) and `department` (departments). The `Emp` table contains employee details, while the `department` table holds department details.

Employee Table

```
CREATE TABLE Emp (
    EmpID INT PRIMARY KEY,
    Name VARCHAR(50),
    Country VARCHAR(50),
    Age INT,
    Salary INT,
    department_id INT
);

INSERT INTO Emp (EmpID, Name, Country, Age, Salary, department_id)
VALUES (1, 'Shubham', 'India', 23, 30000, 101),
       (2, 'Aman', 'Australia', 21, 45000, 102),
       (3, 'Naveen', 'Sri Lanka', 24, 40000, 103),
       (4, 'Aditya', 'Austria', 21, 35000, 104),
       (5, 'Nishant', 'Spain', 22, 25000, 101);
```

Output:

EmpID	Name	Country	Age	Salary	department_id
1	Shubham	India	23	30000	101
2	Aman	Australia	21	45000	102
3	Naveen	Sri Lanka	24	40000	103
4	Aditya	Austria	21	35000	104
5	Nishant	Spain	22	25000	101

Department Table

```
 CREATE TABLE department (
    department_id INT PRIMARY KEY,
    department_name VARCHAR(50),
    department_head VARCHAR(50),
    location VARCHAR(50)
);

INSERT INTO department (department_id, department_name, department_head,
location)
VALUES (101, 'Sales', 'Sarah', 'New York'),
       (102, 'Marketing', 'Jay', 'London'),
       (103, 'Finance', 'Lavish', 'San Francisco'),
       (104, 'Engineering', 'Kabir', 'Bangalore');
SELECT * FROM department;
```

Output:

department_id	department_name	department_head	location
101	Sales	Sarah	New York
102	Marketing	Jay	London
103	Finance	Lavish	San Francisco
104	Engineering	Kabir	Bangalore

Example : Performing a LEFT JOIN

To perform left-join on Employee and Department Tables we will use the following SQL query:

Query:

```
SELECT Emp.EmpID, Emp.Name, department.
department_name, department.department_head,
department.location
FROM Emp
LEFT JOIN department ON Emp.department_id = department.department_id;
```

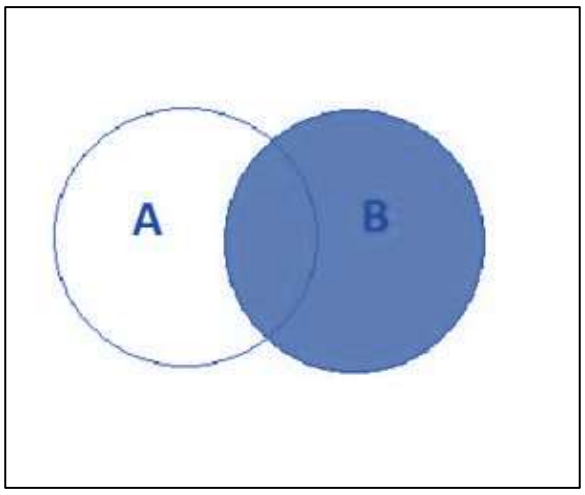
Output:

EmpID	Name	department_name	department_head	location
1	Shubham	Sales	Sarah	New York
2	Aman	Marketing	Jay	London
3	Naveen	Finance	Lavish	San Francisco
4	Aditya	Engineering	Kabir	Bangalore
5	Nishant	Sales	Sarah	New York

SQL RIGHT JOIN

In SQL, the RIGHT JOIN (also called RIGHT OUTER JOIN) is used to combine rows from two tables based on a related column. It returns all records from the right table and only the matching records from the left table. If there is no match in the left table, the result will show `NULL` values for the left table's columns.

- Returns all rows from the right table.
- Includes only matching rows from the left table.
- Non-matching rows from the left table appear as `NULL`



Syntax:

```
SELECT column_name(s)
FROM tableA
RIGHT JOIN tableB
ON tableA.column_name = tableB.column_name;
```

Examples of SQL RIGHT JOIN

In this example, we will consider two tables employee table containing details of the employees working in the particular department the and department table containing the details of the department

Employee Table:

emp_no	emp_name	dept_no
E1	Varun Singhal	D1
E2	Amrita Aggarwal	D2
E3	Ravi Anand	D3

Department Table:

dept_no	d_name	location
D1	IT	Delhi
D2	HR	Hyderabad
D3	Finance	Pune
D4	Testing	Noida
D5	Marketing	Mathura

Query:

```
SELECT emp_no , emp_name ,d_name, location
FROM employee
RIGHT  JOIN dept
ON employee.dept_no = department.dept_no;
```

Output:

emp_no	emp_name	d_name	location
E1	Varun Singhal	IT	Delhi
E2	Amrita Aggarwal	HR	Hyderabad

emp_no	emp_name	d_name	location
E3	Ravi Anand	Finance	Pune
[NULL]	[NULL]	Testing	Noida
[NULL]	[NULL]	Marketing	Mathura

Applications of SQL RIGHT JOIN

- **Merging Data:** Combines related data from different tables.
- **Ensuring Completeness:** Guarantees all records from the right table are included.
- **Handling Missing Values:** Identifies records without matches in the left table.
- **Analyzing Relationships:** Helps detect data gaps and dependencies across tables.

SQL FULL JOIN

The FULL JOIN (or FULL OUTER JOIN) in SQL returns all rows from both tables, combining matched rows and filling unmatched rows with NULL values. It is basically the combination of LEFT JOIN and RIGHT JOIN

- Retrieves all rows from both tables.
- Matches rows where conditions meet.
- Fills NULLs where no match exists.
- Combines results of LEFT JOIN + RIGHT JOIN.
- Can be used sequentially for multiple tables.

Syntax:

```
SELECT columns
FROM table1
FULL JOIN table2
ON table1.column = table2.column;
```

Parameters:

- **`SELECT columns`**: Specifies the columns to retrieve.
- **`FROM table1:`** The first table to be joined.
- **`FULL JOIN table2`**: Specifies the second table to join with the first table using a FULL JOIN.
- **`ON table1.column = table2.column`**: Defines the condition to match rows between the two tables.

Examples of SQL FULL JOIN

Let's look at some examples of the FULL JOIN in SQL and understand it's working. First, let's create a demo database and two tables on which we will perform the JOIN.

1. Books Table
This table Represents the list of books in the system

BOOK_ID	BOOK_NAME	ISSUED_ON	DUE_DATE
1	RD SHARMA	2023-01-01	2023-01-08
2	GATE CRACKER	2023-02-02	2023-02-09
3	MORRIS MANO	2023-03-03	2023-03-10
4	NK PUBLICATIONS	2023-04-04	2023-04-11
5	BIG BANG THEORY	2023-05-05	2023-05-12

2. Authors Table
This table represents the authors who have written the books.

AUTHOR_ID	AUTHOR_NAME
1	Ram Kumar
2	Shyam Sunder
3	Sita Singh
4	Mohan Gupta
5	Raj Kapoor

3. `Publishers` Table
This table represents the authors who have published the books.

PUBLISHER_ID	PUBLISHER_NAME
1	Pearson

PUBLISHER_ID	PUBLISHER_NAME
2	Wiley
3	McGraw-Hill
4	Springer
5	Elsevier

Example : FULL JOIN on Multiple Tables
In this example, we perform a FULL JOIN across the **Books**, **Authors**, and **Publishers** tables to combine all related records into a single result set.

Query:

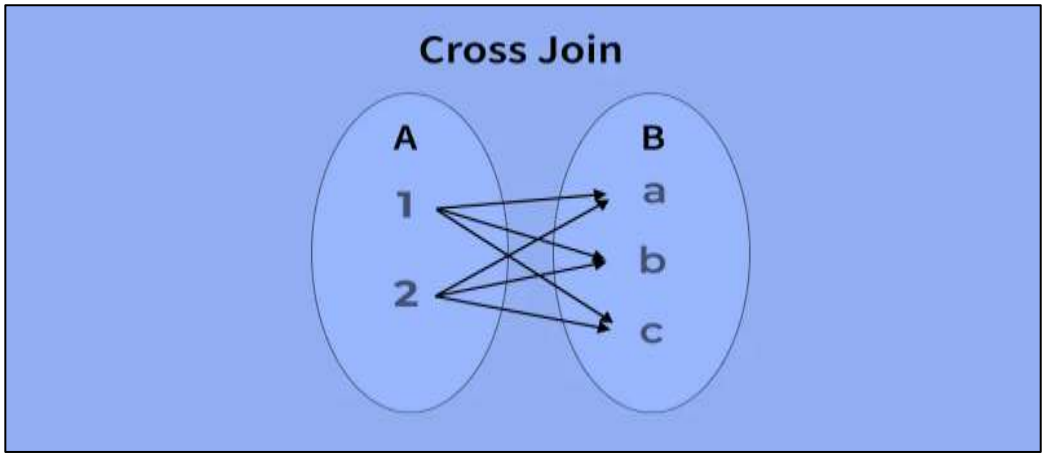
```
SELECT
    b.BOOK_ID,
    b.BOOK_NAME,
    a.AUTHOR_NAME,
    p.PUBLISHER_NAME
FROM Books b
FULL JOIN Authors a ON b.BOOK_ID = a.AUTHOR_ID
FULL JOIN Publishers p ON b.BOOK_ID = p.PUBLISHER_ID;
```

Output:

BOOK_ID	BOOK_NAME	AUTHOR_NAME	PUBLISHER_NAME
1	RD SHARMA	Ram Kumar	Pearson
2	GATE CRACKER	Shyam Sunder	Wiley
3	MORRIS MANO	Sita Singh	McGraw-Hill
4	NK PUBLICATIONS	Mohan Gupta	Springer
5	BIG BANG THEORY	Raj Kapoor	Elsevier

SQL CROSS JOIN

CROSS JOIN in SQL generates the Cartesian product of two tables, meaning each row from the first table is paired with every row from the second. This is useful when you want all possible combinations of records. Since result grows as rows_in_table1 × rows_in_table2, it can get very large, so it's best used with smaller tables or alongside a WHERE clause to filter results into meaningful pairs.



Syntax:

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

Examples of SQL CROSS JOIN

Before diving into queries, let's create two sample tables: Customer and Orders. These tables will help us understand how CROSS JOIN combines data into multiple combinations.

Table 1- Customer

```
CREATE TABLE Customer (
   ID INT,
   NAME VARCHAR(50),
   AGE INT,
   PHONE VARCHAR(10) );

INSERT INTO Customer (ID, NAME, AGE, PHONE) VALUES
(1, 'AMIT JAIN', 21, 98474),
(2, 'JATIN VERMA', 47, 63996);
```

ID	NAME	AGE	PHONE
1	AMIT JAIN	21	98474
2	JATIN VERMA	47	63996

Table 2- Orders

```
CREATE TABLE Orders (
   ORDER_ID INT,
   AMOUNT INT,
   PLACED_ON DATE );

INSERT INTO Orders (ORDER_ID, AMOUNT, PLACED_ON) VALUES
(101, 999, '2023-04-19'),
(102, 4999, '2023-04-20');
```

ORDER_ID	AMOUNT	PLACED_ON
101	999	2023-04-19
102	4999	2023-04-20

Query:

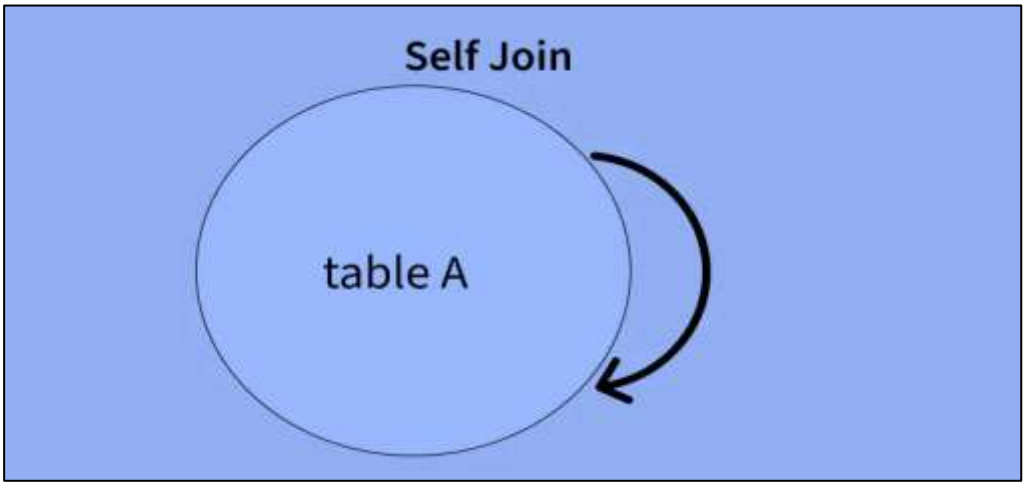
```
SELECT *
FROM Customer
CROSS JOIN Orders;
```

Output:

ID	NAME	AGE	PHONE	ORDER_ID	AMOUNT	PLACED_ON
1	AMIT JAIN	21	98474	101	999	2023-04-19
1	AMIT JAIN	21	98474	102	4999	2023-04-20
2	JATIN VERMA	47	63996	101	999	2023-04-19
2	JATIN VERMA	47	63996	102	4999	2023-04-20

SQL Self Join

A Self Join is a regular join where a table is joined with itself. It is particularly useful when comparing rows within the same table, such as retrieving employee-manager relationships from an employee table.



Syntax:

```
SELECT columns
FROM table AS alias1
JOIN table AS alias2
ON alias1.column = alias2.related_column;
```

- **columns:** Columns to retrieve in the result.
- **alias1:** First reference (alias) of the table.
- **alias2:** Second reference (alias) of the same table.
- **related_column:** condition that links rows within same table (e.g., Employee.ManagerID = Manager.EmployeeID).

Applications of SQL Self Join

SQL Self Join is widely used in different scenarios, such as:

- **Hierarchical Data:** Representing organizational structures like employee–manager, category–subcategory or parent–child relationships.
- **Finding Relationships:** Identifying relationships within the same table, for example linking friends in a social network or mapping task dependencies.
- **Data Comparison:** Comparing rows within same table, such as salaries of employees in same department.
- **Detecting Duplicates:** Finding duplicate records in a table by joining it with itself on duplicate criteria.
- **Sequential Data Analysis:** Comparing current rows with previous or next rows, useful for analyzing trends in sales or time-series data.

SQL | UPDATE with JOIN

What is SQL UPDATE with JOIN?

The **UPDATE with JOIN** statement enables us to modify records in a table using a **JOIN operation**. It combines data from **two** or **more tables** based on a **matching condition** (e.g., INNER JOIN, LEFT JOIN) and updates the specified columns in one table. SQL UPDATE JOIN could be used to update one table using another table and **join condition**.

Syntax

```
UPDATE target_table
SET target_table.column_name = source_table.column_name,
    target_table.column_name2 = source_table.column_name2
FROM target_table
INNER JOIN source_table
ON target_table.column_name = source_table.column_name
WHERE condition;
```

Key Terms

- **`target_table`:** The table whose records you want to update.
- **`source_table`:** The table containing the data you want to use for the update.
- **`SET`:** Specifies the columns in the target table that will be updated.
- **`INNER JOIN`:** Ensures only matching rows from both tables are considered.
- **`ON`:** The condition that specifies how the tables are related.
- **`WHERE`:** An optional clause to filter which rows to update.

Example: SQL UPDATE with JOIN Using Two Tables

col1	col2	col3
1	11	FIRST
11	12	SECOND
21	13	THIRD
31	14	FOURTH

col1	col2	col3
1	21	TWO-ONE
11	22	TWO-TWO
21	23	TWO-THREE
31	24	TWO-FOUR

Query:

```
UPDATE tech1
SET col2 = tech2.col2,
col3 = Tech2.col3
FROM tech1
INNER JOIN tech2 ON tech1.col1 = tech2.col1
WHERE Tech1.col1 IN (21, 31);
```

Output

col1	col2	col3
1	11	FIRST
11	12	SECOND
21	23	TWO-THREE
31	24	TWO-FOUR

Explanation:

- We are updating **col2** and **col3** in table `Tech1`.
- We use an **INNER JOIN** to match rows from `Tech1` and `Tech2` where `col1` is the same.
- The **WHERE clause** filters the rows for `col1 = 21` and `col1 = 31`.

SQL UPDATE with JOIN Using LEFT JOIN

Sometimes you may need to update records in the target table even if there is no match in the source table. In such cases, we can use **LEFT JOIN**.

Syntax

UPDATE target_table
SET target_table.column_name = source_table.column_name
FROM target_table
LEFT JOIN source_table
ON target_table.column_name = source_table.column_name;

Example

```
UPDATE tech1
SET col2 = ISNULL(tech2.col2, 0)
FROM tech1
LEFT JOIN tech2
ON tech1.col1 = tech2.col1;
```

Output

col1	col2	col3
1	0	FIRST
11	22	SECOND
21	23	TWO-THREE
31	24	TWO-FOUR

SQL DELETE JOIN

The **DELETE JOIN** operation in SQL delete rows from a table while considering conditions that involve another table. By combining DELETE with **JOIN**, we can efficiently **manage** related data across tables. This method allows us to **join two** or **more** tables and delete rows from the **primary table** that meet specific criteria, ensuring that the **integrity** of related data is maintained while **removing unnecessary** or **outdated information**.

While we can **join multiple tables** in the `FROM` clause, we can only specify one table in the **DELETE clause**. This means that although multiple tables can be used to define conditions for deletion, the rows can only be **deleted from a single**, explicitly stated table.

Key Features of DELETE JOIN

- Deletes rows from one primary table based on conditions set by joining additional tables.
- Only one table is listed explicitly in the DELETE clause.
- Ensures efficient data management and integrity across relational tables.
- Uses **INNER JOIN**, LEFT JOIN, or **USING** clauses to match data between tables.
- Can target specific rows using the **WHERE** clause.

Syntax:

DELETE table1
FROM table1 JOIN table2
ON table1.attribute_name = table2.attribute_name
WHERE condition

Key Terms:

- **table1**: The primary table from which rows will be deleted.
- **table2**: The table used for comparison or condition.
- **ON**: Specifies the condition for the JOIN.
- **WHERE**: Optional; filters which rows to delete.

Demo SQL Database

For this **DELETE JOIN** tutorial, we will use two example tables: **students** and **library_books**. These tables contain related data, where each student has an **associated library** book entry. This example demonstrates how to delete records from the **library_books** table based on matching **student IDs** from the **students** table.

1. **Student Table**

student_id	student_name	student_branch
1001	PRADEEP	E.C.E

student_id	student_name	student_branch
1002	KIRAN	E.C.E
1003	PRANAV	E.C.E
2001	PADMA	C.S.E
2002	SRUTHI	C.S.E
2003	HARSITHA	C.S.E
3001	SAI	I.T
3002	HARSH	I.T
3003	HARSHINI	I.T

2. **Library Books Table**

lib_id	book_taken
1001	2
1002	3
1003	4
2001	2
3001	3

Step-by-Step Example: Deleting Rows with DELETE JOIN

This section demonstrates how to use the **DELETE JOINsyntax** to remove rows from **one table** based on **conditions** involving **another table**. This example will walk you through the process of **setting up** the necessary **tables**, **inserting data**, and executing the **DELETE JOIN** query to delete specific rows efficiently.

Step 1: Create Tables

```
CREATE DATABASE TechforTech;
USE TechforTech

CREATE TABLE student (
student_id VARCHAR(8),
student_name VARCHAR(20),
student_branch VARCHAR(20)
)

CREATE TABLE library_books(
lib_id VARCHAR(20),
book_taken INT
)
```

Step 2: Insert Sample Data

```
INSERT INTO students
VALUES( '1001','PRADEEP','E.C.E'),
( '1002','KIRAN','E.C.E'),
( '1003','PRANAV','E.C.E'),
( '2001','PADMA','C.S.E'),
( '2002','SRUTHI','C.S.E'),
( '2003','HARSITHA','C.S.E'),
( '3001','SAI','I.T'),
( '3002','HARSH','I.T'),
( '3003','HARSHINI','I.T')

INSERT INTO library_books
VALUES( '1001',2),
( '1002',3),
( '1003',4),
( '2001',2),
( '3001',3)
```

Step 3: Delete a Row Using DELETE JOIN

Suppose we want to delete a row from the **library_books** table where the **`lib_id`** matches **`student_id`** 1001 in the `students` table for a specific student ID

```
DELETE library_books
FROM  library_books JOIN students ON
students.student_id =library_books.lib_id
WHERE lib_id= 1001
SELECT * FROM library_books
```

Recursive Join in SQL

Recursive joins are implemented using recursive **common table expressions** (CTEs). CTEs are temporary result sets that can be referred to within the execution scope of a **SELECT**, **INSERT**, **UPDATE**, or DELETE statement. In a recursive CTE, a query is repeatedly executed to gather related data, making it possible to handle hierarchical relationships like parent-child data.

Syntax:

```
WITH RECURSIVE cte_name AS (
    -- Anchor Query: Select the root or starting point
    SELECT columns
    FROM table
    WHERE condition

    UNION ALL

    -- Recursive Query: Join the CTE with the table to fetch related data
    SELECT t.columns
    FROM table t
    INNER JOIN cte_name cte ON t.column = cte.column
)
```

Example of Recursive Join in SQL

Let's walk through an example where we create an **employee-manager hierarchy** using a recursive join in SQL. Assume we have a table of employees where each employee has a manager_id pointing to their manager's employee_id. The goal is to retrieve a list of employees along with their managers, all the way up the chain.

employee_id	employee_name	manager_id	age
1	Ankit	NULL	32
2	Ayush	1	31
3	Piyush	1	42
4	Ramesh	2	31
5	Rohan	3	29
6	Harry	3	28
7	Rohit	4	32
8	Gogi	4	32
9	Tapu	5	33
10	Sonu	5	40

Now, we will use a recursive join to get a list of all employees and their managers, starting with Ankit (employee with employee_id = 1).

Query:

```
WITH RECURSIVE employee_hierarchy AS (
    -- Anchor query: Start with Ankit (employee_id = 1)
    SELECT employee_id, employee_name, manager_id, age
    FROM employees
    WHERE employee_id = 1

    UNION ALL

    -- Recursive query: Join the employees table with itself to get the
employees reporting to each manager
    SELECT e.employee_id, e.employee_name, e.manager_id, e.age
    FROM employees e
    INNER JOIN employee_hierarchy eh ON e.manager_id = eh.employee_id
)
SELECT * FROM employee_hierarchy;
```

Output:

id	name	age
1	Ankit	32
2	Ayush	31
3	Piyush	42
4	Ramesh	31
5	Rohan	29
6	Harry	28
7	Rohit	32
8	Gogi	32
9	Tapu	33
10	Sonu	40

Applications of Recursive Joins

- **Hierarchical Data Representation:** Recursive joins are commonly used to represent and query hierarchical structures, such as employee-manager relationships, organizational charts, and bill of materials.
- **Parent-Child Relationships:** Recursive queries help retrieve data that represents parent-child relationships, such as categories and subcategories in a product catalog.
- **Graph Traversal:** Recursive joins are also used for traversing graphs or networks, such as social networks or transportation networks