The `USING` clause in SQL is used when performing `JOIN` operations between two tables. It is specifically useful when the two tables have one or more columns with the same name, and you want to match them based on those columns. The `USING` clause simplifies the join condition by allowing you to specify the common columns without needing to fully qualify the column names from each table.

### Syntax of `USING`

```sql

SELECT column\_list

FROM table1

JOIN table2

USING (column\_name);

```

- \*\*`table1` and `table2`\*\*: The two tables that you want to join.

- \*\*`column\_name`\*\*: The column (or columns) common to both tables that will be used to perform the join.

### Example Scenario: Employee and Department Tables

Let’s say we have two tables: `employees` and `departments`. Both tables have a column called `department\_id`, which we can use to join them.

#### Step 1: Create the Tables

\*\*`employees` table:\*\*

```sql

CREATE TABLE employees (

employee\_id SERIAL PRIMARY KEY,

name VARCHAR(100),

department\_id INT

);

```

\*\*`departments` table:\*\*

```sql

CREATE TABLE departments (

department\_id SERIAL PRIMARY KEY,

department\_name VARCHAR(100)

);

```

#### Step 2: Insert Sample Data

\*\*Insert data into `employees`:\*\*

```sql

INSERT INTO employees (name, department\_id)

VALUES

('John Doe', 1),

('Jane Smith', 2),

('Alice Johnson', 1),

('Bob Williams', 3);

```

\*\*Insert data into `departments`:\*\*

```sql

INSERT INTO departments (department\_name)

VALUES

('HR'),

('Engineering'),

('Marketing');

```

#### Step 3: Query with `USING` Clause

We want to join the `employees` table with the `departments` table based on the `department\_id` column, which exists in both tables.

Using the `USING` clause simplifies the `JOIN` condition:

```sql

SELECT name, department\_name

FROM employees

JOIN departments

USING (department\_id);

```

### Explanation:

- The \*\*`USING (department\_id)`\*\* clause tells the database to join the two tables based on the common `department\_id` column.

- In the result, `department\_id` will not appear twice (once from each table); it will only appear once, which simplifies the result set.

### Output:

| name | department\_name |

|----------------|-----------------|

| John Doe | HR |

| Alice Johnson | HR |

| Jane Smith | Engineering |

| Bob Williams | Marketing |

### Step 4: Another Example with Multiple Columns

If two tables share more than one column, you can use the `USING` clause with multiple columns.

Let’s say we have another table `project\_assignments` that contains both `employee\_id` and `department\_id`.

```sql

CREATE TABLE project\_assignments (

employee\_id INT,

department\_id INT,

project\_name VARCHAR(100)

);

```

You can join both `employees` and `departments` to `project\_assignments` on both `employee\_id` and `department\_id` using `USING`:

```sql

SELECT name, project\_name, department\_name

FROM employees

JOIN project\_assignments USING (employee\_id, department\_id)

JOIN departments USING (department\_id);

```

### Key Points:

- \*\*Simplification\*\*: `USING` helps to avoid repeating the column names when the join is based on columns with the same name in both tables.

- \*\*Automatic column removal\*\*: The common column (e.g., `department\_id`) is only displayed once in the result set.

- \*\*Readability\*\*: The `USING` clause makes the query more readable compared to manually specifying join conditions with `ON`.

### Conclusion

The `USING` clause is useful when joining tables that have common column names, as it reduces repetition and improves query readability. It's especially helpful when you are working with natural joins based on columns with the same name in both tables.