Reading: Examples for Multiple Tables

Estimated Time: 15 minutes

Objectives

At the end of this reading, you will be able to:

- Explain the concept of working with multiple tables in SQL queries
- Identify scenarios where joining multiple tables is necessary for data retrieval and manipulation
- Determine how to use different types of joins to combine data from multiple tables effectively

Sample data

The database is designed to store and manage employee information within an organization. It includes tables for employee details, department information, and salary records.

Employees

This table contains information about employees. The `emp_id` column represents the unique identifier for each employee, `emp_name` stores the employee's name, and `dept_id` indicates the department to which the employee belongs.

emp_id	emp_name	dept_id
1	Alice	101
2	Bob	102
3	Charlie	101
4	David	103

Departments

This table contains information about departments. The `dept_id` column serves as the unique identifier for each department, while `dept_name` holds the name of the department.

dept_id	dept_name
101	HR
102	Finance
103	IT

Salaries

This table contains information about employees' salaries. The `emp_id` column links each salary entry to the respective employee, while the `salary` column denotes the salary amount for each employee.

emp_id	salary
1	50000
2	60000
3	55000
4	65000

Working with multiple tables

In database management, working with multiple tables is essential for storing and organizing complex data sets. It involves retrieving, manipulating, and analyzing data from two or more tables simultaneously to derive meaningful insights. Combining related information from different sources allows for more comprehensive analysis and reporting.

Implicit join

An implicit join refers to the joining of tables using a condition in the WHERE clause without explicitly using the JOIN keyword.

Syntax statement for implicit join

The general syntax for implicit join:

```
SELECT columns
FROM table1, table2
WHERE table1.column_name = table2.column_name;
```

Explanation:

- In an implicit join, tables are listed in the FROM clause and separated by commas.
- The WHERE clause specifies the join condition, connecting columns from the joined tables.

Example

Consider the following example where the objective is to retrieve the employee names and their corresponding salaries.

```
SELECT employees.emp_name, salaries.salary
FROM employees, salaries
WHERE employees.emp_id = salaries.emp_id;
```

Explanation:

- This query retrieves employee names and their corresponding department names by implicitly joining the employees and department tables based on matching employee IDs `(emp_id)`.
- The condition `employees.emp_id = departments.emp_id` in the `WHERE` clause ensures that only rows with matching employee IDs from both tables are included in the result set.

Response:
The output represents the result set obtained from executing the SQL query. It displays employees names `(emp_name)` alongside their corresponding salaries `(salary)`. Each row in the result set represents an employee and their respective salary.

emp_name	salary
Alice	50000
Bob	60000
Charlie	55000
David	65000

Example

Consider the following example, in which the objective is to retrieve the employees names along with their corresponding department names and salaries.

```
SELECT e.emp_name, d.dept_name, s.salary
FROM Employees e, Departments d, Salaries s
WHERE e.dept_id = d.dept_id
AND e.emp_id = s.emp_id;
```

Explanation:

- List the employees, departments, and salaries tables in the FROM clause without specifying a join condition.
- Join conditions are specified in the `WHERE` clause:

1. `e.dept_id = d.dept_id`: Matches employees with their respective departments based on the equality of department IDs.
2. `emp_id = s.emp_id`: Matches employees with their respective salaries based on the equality of employee IDs.

Response:
The output represents comprehensive employee information, including department affiliation and salary details.

emp_name	dept_name	salary
Alice	HR	50000
Bob	Finance	60000
Charlie	HR	55000
David	IT	65000

Example

Consider the following example, in which the objective is to retrieve the employees names along with their corresponding department names and the average salary for each department.

```
SELECT e.emp_name, d.dept_name, AVG(s.salary) AS avg_salary
FROM Employees e, Departments d, Salaries s
WHERE e.dept_id = d.dept_id
AND e.emp_id = s.emp_id
GROUP BY d.dept_name;
```

Explanation:

- List the employees, departments, and salaries tables in the FROM clause without specifying a join condition.
- Join conditions are specified in the `WHERE` clause:

1. `e.dept_id = d.dept_id`: Matches employees with their respective departments based on the equality of department IDs.
2. `e.emp_id = s.emp_id`: Matches employees with their respective salaries based on the equality of employee IDs.

- The `GROUP BY` clause groups the result by department name (dept_name).
- The `AVG()` function calculates the average salary (salary) for each department.

Response:
The output represents the names of employees `(emp_name)`, their corresponding department names `(dept_name)`, and the average salary `(avg_salary)` for each department. Each row in the result set represents a department along with the average salary for that department.

emp_name	dept_name	avg_salary
Alice	HR	52500
Bob	Finance	60000
Charlie	HR	52500
David	IT	65000

Inner join

An inner join returns rows from both tables with matching values in the specified column.

Syntax statement for inner join

The general syntax for inner join:

```
SELECT columns
FROM table1
INNER JOIN table2 ON table1.column_name = table2.column_name;
```

Explanation:

- The `INNER JOIN` keyword combines rows from both table 1 and table 2 based on the specified join condition.
- The condition `table1.column_name = table2.column_name` ensures that only rows with matching values in the specified column are included in the result set.
- Columns selected in the SELECT statement are retrieved from table 1 and table 2, providing a comprehensive view of the combined data.

Example

Consider the following example where the objective is to retrieve the employees names and their corresponding department names.

```
SELECT emp_name, dept_name
FROM employees
INNER JOIN departments ON employees.dept_id = departments.dept_id;
```

Explanation:

- The `INNER JOIN` operation ensures that the result set includes only rows with matching department IDs `(dept_id)` from both the employees and departments tables.
- The condition `employees.dept_id = departments.dept_id` specifies the join criterion, linking rows from the employees table to corresponding rows in the departments table based on their department IDs.
- The selected columns, `emp_name` from the employees table and `dept_name` from the departments table, provide a comprehensive view of employee names and their corresponding department names.

Response:
The output represents the result set obtained from executing the SQL query. It displays employee names `(emp_name)` alongside their corresponding department names `(dept_name)`.

emp_name	dept_name
Alice	HR
Bob	Finance
Charlie	HR
David	IT

Each row in the output represents an employee and their respective department. For example:

- Alice works in the HR department.
- Bob works in the Finance department.
- Charlie also works in the HR department.
- David works in the IT department.

These examples demonstrate the flexibility and power of using joins to combine data from multiple tables in SQL, enabling comprehensive analysis and reporting across related data sets. In the lab that follows later in the module, you will apply these concepts and practice accessing multiple tables with subqueries and implicit join hands-on.

Good luck!

Author(s)

Pratiksha Verma

Other Contributors

Malika Singla, Lakshmi Holla


Skills Network