**SQL Commands**

Table of Contents

[CREATE TABLE 2](#_Toc177760456)

[INSERT to add values into the table 2](#_Toc177760457)

[SELECT with clauses and functions 4](#_Toc177760458)

[JOINS 5](#_Toc177760459)

[AGGREGATE functions 7](#_Toc177760460)

[WINDOW functions 7](#_Toc177760461)

[Subqueries 7](#_Toc177760462)

[CTE 7](#_Toc177760463)

[Methods to optimize SQL queries 7](#_Toc177760464)

## CREATE TABLE

* **SQL Queries to Create the Tables ‘departments’ and ‘employees’**

**-- departments’**

CREATE TABLE departments (

id INT PRIMARY KEY AUTO\_INCREMENT,

department\_name VARCHAR(50) NOT NULL

);

-- **employees’**

CREATE TABLE employees (

id INT PRIMARY KEY AUTO\_INCREMENT,

first\_name VARCHAR(50) NOT NULL,

last\_name VARCHAR(50) NOT NULL,

age INT,

salary DECIMAL(10, 2),

department\_id INT,

FOREIGN KEY (department\_id) REFERENCES departments(id)

);

**Explanation of the SQL Queries**

* **CREATE TABLE**: This command creates a new table with specified columns and data types. The AUTO\_INCREMENT keyword automatically generates a unique value for the primary key column.
* **FOREIGN KEY**: In the employees table, the department\_id column references the id column in the departments table, creating a relationship between the two tables.

INSERT to add values into the table

* **Sample Data to Insert into ‘department’ Table**

INSERT INTO departments (department\_name) VALUES

('Sales'),

('Marketing'),

('Human Resources'),

('IT'),

('Finance'),

('Customer Service'),

('Research and Development'),

('Logistics'),

('Legal'),

('Public Relations');

* **Sample Data to Insert into Employees Table**

INSERT INTO employees (first\_name, last\_name, age, salary, department\_id) VALUES

('John', 'Doe', 30, 70000.00, 1),

('Jane', 'Smith', 25, 50000.00, 1),

('Emily', 'Davis', 35, 90000.00, 2),

('Michael', 'Brown', 28, 60000.00, 2),

('Chris', 'Johnson', 40, 80000.00, 3),

('Patricia', 'Williams', 32, 65000.00, 3),

('David', 'Jones', 29, 55000.00, 4),

('Linda', 'Garcia', 45, 95000.00, 4),

('Robert', 'Martinez', 38, 72000.00, 5),

('Sarah', 'Hernandez', 27, 48000.00, 5);

**Explanation of the SQL Queries**

* **INSERT INTO**: This command adds new rows to the specified table with the given values.

## SELECT with clauses and functions

* WHERE

Filters the results to include only those employees in the Sales department.

SELECT \* FROM employees WHERE department = 'Sales';

* ORDER BY

Sorts the results by the age column in ascending order. Use DESC for descending order.

SELECT name, age FROM employees ORDER BY age ASC; -- Change to DESC for descending order

* GROUP BY

Counts the number of employees in each department and labels the column as employee\_count

SELECT department, COUNT(\*) AS employee\_count FROM employees GROUP BY department;

* HAVING

Filters the grouped results to include only departments where the average salary exceeds $60,000

SELECT department, AVG(salary) AS avg\_salary FROM employees GROUP BY department HAVING AVG(salary) > 60000;

* LIKE

Filters the results to include only names that start with 'J'. The % wildcard represents any sequence of characters.

SELECT name FROM employees WHERE name LIKE 'J%'; -- Names starting with 'J'

* IN

Filters results to include only those employees who belong to either the Sales or Marketing departments.

SELECT name, department FROM employees WHERE department IN ('Sales', 'Marketing');

* DISTINCT

Removes duplicate values in the results, returning only unique department names.

SELECT DISTINCT department FROM employees;

* CONCAT

Combines the first and last names into a single full\_name column.

SELECT CONCAT(first\_name, ' ', last\_name) AS full\_name FROM employees;

* UPPER/LOWER

Converts the names to uppercase and labels the result as upper\_name.

SELECT UPPER(name) AS upper\_name FROM employees;

SELECT LOWER(name) AS upper\_name FROM employees;

* CASE

Labels the resulting column as salary\_category, categorizing salaries into 'Low', 'Medium', or 'High

SELECT name,

CASE

WHEN salary < 50000 THEN 'Low'

WHEN salary BETWEEN 50000 AND 100000 THEN 'Medium'

ELSE 'High'

END AS salary\_category

FROM employees;

## JOINS

* INNER JOIN: Matches rows in both tables.

SELECT e.first\_name, e.last\_name, d.department\_name

FROM employees e

INNER JOIN departments d ON e.department\_id = d.id;

This query retrieves the first and last names of employees along with their department names, but only for employees who are assigned to a department.

* LEFT JOIN: All rows from the left table and matched rows from the right.

SELECT e.first\_name, e.last\_name, d.department\_name

FROM employees e

LEFT JOIN departments d ON e.department\_id = d.id;

This query retrieves all employees, including those who may not belong to any department. If an employee does not belong to a department, the department\_name will be NULL.

* RIGHT JOIN: All rows from the right table and matched rows from the left.

SELECT e.first\_name, e.last\_name, d.department\_name

FROM employees e

RIGHT JOIN departments d ON e.department\_id = d.id;

This query retrieves all departments, including those that do not have any employees. If a department has no employees, the first\_name and last\_name will be NULL.

* FULL JOIN: All rows from both tables, with NULLs for unmatched rows.

SELECT e.first\_name, e.last\_name, d.department\_name

FROM employees e

FULL OUTER JOIN departments d ON e.department\_id = d.id;

This query retrieves all employees and all departments. If an employee does not belong to a department or a department has no employees, the corresponding columns will be NULL.

* CROSS JOIN: All combinations of rows from both tables.

SELECT e.first\_name, d.department\_name

FROM employees e

CROSS JOIN departments d;

This query returns a combination of each employee with every department. If there are 10 employees and 10 departments, the result will contain 100 rows.

* SELF JOIN: Joins a table with itself to compare rows.

SELECT a.first\_name AS Employee1, b.first\_name AS Employee2

FROM employees a

INNER JOIN employees b ON a.department\_id = b.department\_id AND a.id <> b.id;

This query retrieves pairs of employees who work in the same department. It uses an alias (a and b) to differentiate between the two instances of the employees table

## AGGREGATE functions

* COUNT: Counts rows or non-null values.

SELECT COUNT(\*) AS total\_employees FROM employees;

* SUM: Sums numeric values.

SELECT SUM(salary) AS total\_salaries FROM employees;

* AVG: Calculates the average of numeric values.

SELECT AVG(salary) AS average\_salary FROM employees;

* MIN: Finds the minimum value.

SELECT MIN(salary) AS lowest\_salary FROM employees;

* MAX: Finds the maximum value.

SELECT MAX(salary) AS highest\_salary FROM employees;

## WINDOW functions

Window functions perform calculations across a set of table rows that are somehow related to the current row. Unlike regular aggregate functions, they do not group rows into a single output but instead provide a value for each row.

* SELECT employee\_id, department\_id, salary,

AVG(salary) OVER (PARTITION BY department\_id) AS avg\_department\_salary

FROM employees;

Returns average salary in each department

* SELECT employee\_id, name, department\_id, salary,

RANK() OVER (PARTITION BY department\_id ORDER BY salary DESC) AS salary\_rank

FROM employees;

RANK(): Assigns a rank to each employee based on their salary within each department (partitioned by department\_id). Higher salaries get lower rank numbers.

## Subqueries

* Return only those employees whose salary is above the average.

SELECT name FROM employees WHERE salary > (SELECT AVG(salary) FROM employees);

Explanation:

* Subquery (SELECT AVG(salary) FROM employees): Calculates the average salary of all employees.
* WHERE salary > (subquery): Filters the main query to return only those employees whose salary is above the average.

## CTE

A CTE is a temporary result set that you can reference within a SELECT, INSERT, UPDATE, or DELETE statement. It’s defined using the WITH clause and can help break complex queries into simpler parts, improving readability and maintainability.

WITH DepartmentCount AS (

SELECT department\_id, COUNT(\*) AS employee\_count

FROM employees

GROUP BY department\_id

)

SELECT d.department\_name, dc.employee\_count

FROM departments d

JOIN DepartmentCount dc ON d.department\_id = dc.department\_id;