50+ SQL Practice Problems with Answers

Table of Contents

1. Basic Queries <a name="basic-queries"></a>

For the following problems, assume we have a table called `employees` with columns: `id`, `first_name`, `last_name`, `email`, `department`, `salary`.

1. Retrieve all columns for all employees.

   ```
   SELECT * FROM employees;
   ```

2. Retrieve only the first name and last name of all employees.

   ```
   SELECT first_name, last_name FROM employees;
   ```

3. Get the unique departments from the employees table.

   ```
   SELECT DISTINCT department FROM employees;
   ```

4. Show the first 5 employees in the table.

   ```
   SELECT * FROM employees LIMIT 5;
   ```

5. Get the employee with ID 1000.

   ```
   SELECT * FROM employees WHERE id = 1000;
   ```

2. Filtering and Sorting <a name="filtering-and-sorting"></a>

1. Retrieve all employees in the 'Sales' department.

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

2. Get all employees with a salary greater than 50000.

   ```
   SELECT * FROM employees WHERE salary > 50000;
   ```

3. Find employees whose last name starts with 'S'.

   ```
   SELECT * FROM employees WHERE last_name LIKE 'S%';
   ```

4. Retrieve employees in either the 'Marketing' or 'HR' department.

   ```
   SELECT * FROM employees WHERE department IN ('Marketing', 'HR');
   ```

5. Get employees with salaries between 40000 and 60000.

   ```
   SELECT * FROM employees WHERE salary BETWEEN 40000 AND 60000;
   ```

6. Find all employees not in the 'IT' department.

   ```
   SELECT * FROM employees WHERE department != 'IT';
   ```

7. Retrieve employees ordered by their salary in descending order.

   ```
   SELECT * FROM employees ORDER BY salary DESC;
   ```

8. Get the top 5 highest-paid employees.

   ```
   SELECT * FROM employees ORDER BY salary DESC LIMIT 5;
   ```

3. Joins <a name="joins"></a>

Assume we have another table called `departments` with columns: `id`, `name`, `location`.

1. Join employees with their department information.

   ```
   SELECT e.*, d.location
   FROM employees e
   JOIN departments d ON e.department = d.name;
   ```

2. Find all employees in departments located in 'New York'.

   ```
   SELECT e.*
   FROM employees e
   JOIN departments d ON e.department = d.name
   WHERE d.location = 'New York';
   ```

3. List all departments and employees, including departments with no employees.

   ```
   SELECT d.name, e.first_name, e.last_name
   FROM departments d
   LEFT JOIN employees e ON d.name = e.department;
   ```

4. Find employees who don't have a corresponding department.

   ```
   SELECT e.*
   FROM employees e
   LEFT JOIN departments d ON e.department = d.name
   WHERE d.name IS NULL;
   ```

4. Aggregation and Grouping <a name="aggregation-and-grouping"></a>

1. Count the number of employees in each department.

   ```
   SELECT department, COUNT(*) as employee_count
   FROM employees
   GROUP BY department;
   ```

2. Calculate the average salary for each department.

   ```
   SELECT department, AVG(salary) as avg_salary
   FROM employees
   GROUP BY department;
   ```

3. Find the department with the highest total salary.

   ```
   SELECT department, SUM(salary) as total_salary
   FROM employees
   GROUP BY department
   ORDER BY total_salary DESC
   LIMIT 1;
   ```

4. Count employees with salaries over 50000 in each department.

   ```
   SELECT department, COUNT(*) as high_earners
   FROM employees
   WHERE salary > 50000
   GROUP BY department;
   ```

5. Find departments with more than 10 employees.

   ```
   SELECT department, COUNT(*) as employee_count
   FROM employees
   GROUP BY department
   HAVING COUNT(*) > 10;
   ```

5. Subqueries <a name="subqueries"></a>

1. Find employees who earn more than the average salary.

```
SELECT *
FROM employees
WHERE salary > (SELECT AVG(salary) FROM employees);
```

2. Retrieve employees in the department with the highest average salary.

```
SELECT *
FROM employees
WHERE department = (
    SELECT department
    FROM employees
    GROUP BY department
    ORDER BY AVG(salary) DESC
    LIMIT 1
);
```

3. Find the second highest salary.

```
SELECT MAX(salary)
FROM employees
WHERE salary < (SELECT MAX(salary) FROM employees);
```

4. Retrieve employees who have a higher salary than their department's average.

```
SELECT e.*
FROM employees e
WHERE salary > (
    SELECT AVG(salary)
    FROM employees
    WHERE department = e.department
);
```

6. Data Modification <a name="data-modification"></a>

1. Insert a new employee into the employees table.

```
INSERT INTO employees (first_name, last_name, email, department, salary)
VALUES ('John', 'Doe', 'john.doe@example.com', 'IT', 60000);
```

2. Update the salary of all employees in the 'Marketing' department by 10%.

```
UPDATE employees
SET salary = salary * 1.1
WHERE department = 'Marketing';
```

3. Delete all employees with a salary less than 30000.

```
DELETE FROM employees
WHERE salary < 30000;
```

4. Insert multiple employees at once.

```
INSERT INTO employees (first_name, last_name, email, department, salary)
VALUES
('Alice', 'Smith', 'alice.smith@example.com', 'Sales', 55000),
('Bob', 'Johnson', 'bob.johnson@example.com', 'HR', 50000);
```

7. Table Operations <a name="table-operations"></a>

1. Create a new table called `projects` with columns: id, name, start_date, end_date.

```
CREATE TABLE projects (
    id INT PRIMARY KEY AUTO_INCREMENT,
    name VARCHAR(100) NOT NULL,
    start_date DATE,
    end_date DATE
);
```

2. Add a new column 'manager_id' to the employees table.

```
ALTER TABLE employees
ADD COLUMN manager_id INT;
```

3. Create an index on the 'email' column of the employees table.

```
CREATE INDEX idx_email ON employees(email);
```

4. Remove the 'end_date' column from the projects table.

```
ALTER TABLE projects
DROP COLUMN end_date;
```

8. Advanced Queries <a name="advanced-queries"></a>

1. Rank employees by salary within each department.

```
SELECT
    first_name,
    last_name,
    department,
    salary,
    RANK() OVER (PARTITION BY department ORDER BY salary DESC) as salary_rank
FROM employees;
```

2. Calculate a running total of salary by department.

```
SELECT
    first_name,
    last_name,
    department,
    salary,
    SUM(salary) OVER (PARTITION BY department ORDER BY id) as running_total
FROM employees;
```

3. Find employees with salaries higher than their department's average.

```
WITH dept_avg AS (
    SELECT department, AVG(salary) as avg_salary
    FROM employees
    GROUP BY department
)
SELECT e.*
FROM employees e
JOIN dept_avg da ON e.department = da.department
WHERE e.salary > da.avg_salary;
```

4. Get the nth highest salary.

```
-- Replace 'n' with the desired number
SELECT DISTINCT salary
FROM (
    SELECT salary,
           DENSE_RANK() OVER (ORDER BY salary DESC) as salary_rank
    FROM employees
) ranked_salaries
WHERE salary_rank = n;
```

5. Find employees who have the same name.

```
SELECT first_name, last_name, COUNT(*) as name_count
FROM employees
```

```
GROUP BY first_name, last_name
HAVING COUNT(*) > 1;
```

6. List all employees and their direct reports.

```
SELECT
    m.first_name as manager_first_name,
    m.last_name as manager_last_name,
    e.first_name as employee_first_name,
    e.last_name as employee_last_name
FROM employees e
LEFT JOIN employees m ON e.manager_id = m.id;
```

7. Calculate the difference between each employee's salary and their department's average salary.

```
SELECT
    e.first_name,
    e.last_name,
    e.department,
    e.salary,
    e.salary - avg_salary as salary_difference
FROM employees e
JOIN (
    SELECT department, AVG(salary) as avg_salary
    FROM employees
    GROUP BY department
) dept_avg ON e.department = dept_avg.department;
```

8. Find employees who have worked on all projects.

```
-- Assuming a 'employee_projects' table linking employee
s and projects
SELECT e.first_name, e.last_name
FROM employees e
WHERE NOT EXISTS (
    SELECT p.id
    FROM projects p
    WHERE NOT EXISTS (
        SELECT 1
        FROM employee_projects ep
        WHERE ep.employee_id = e.id AND ep.project_id = p.id
    )
);
```

9. Get the top 3 earners in each department.

```
WITH ranked_employees AS (
    SELECT
        *,
        RANK() OVER (PARTITION BY department ORDER BY salary DESC) as salary_rank
    FROM employees
)
SELECT *
FROM ranked_employees
WHERE salary_rank <= 3;
```

10. Find departments where the total salary is higher than the company's average department total salary.

```
WITH dept_totals AS (
    SELECT department, SUM(salary) as total_salary
    FROM employees
    GROUP BY department
)
SELECT *
FROM dept_totals
WHERE total_salary > (
    SELECT AVG(total_salary)
    FROM dept_totals
);
```

11. List employees who have a salary higher than their manager.

```
SELECT e.first_name, e.last_name, e.salary as employee_salary, m.salary as manager_salary
FROM employees e
JOIN employees m ON e.manager_id = m.id
WHERE e.salary > m.salary;
```

12. Find the employee(s) with the longest tenure in each department.

```
-- Assuming there's a 'hire_date' column
WITH ranked_employees AS (
    SELECT
        *,
        RANK() OVER (PARTITION BY department ORDER BY hire_date ASC) as tenure_rank
    FROM employees
)
SELECT *
FROM ranked_employees
WHERE tenure_rank = 1;
```

13. Calculate the percentage of total salary each employee represents in their department.

```
WITH dept_totals AS (
    SELECT department, SUM(salary) as total_salary
    FROM employees
    GROUP BY department
)
SELECT
    e.first_name,
    e.last_name,
    e.department,
    e.salary,
    (e.salary / dt.total_salary * 100) as salary_percentage
FROM employees e
JOIN dept_totals dt ON e.department = dt.department;
```

14. Find employees who have changed departments.

```
-- Assuming there's a 'employee_history' table with past
department info
SELECT DISTINCT e.first_name, e.last_name
FROM employees e
JOIN employee_history eh ON e.id = eh.employee_id
WHERE e.department != eh.department;
```

15. List projects along with the number of employees working on each.

```
-- Assuming an 'employee_projects' table
SELECT
    p.name as project_name,
    COUNT(DISTINCT ep.employee_id) as employee_count
FROM projects p
LEFT JOIN employee_projects ep ON p.id = ep.project_id
GROUP BY p.id, p.name;
```

16. Find employees who have worked in all departments.

```
-- Assuming an 'employee_history' table with past depart
ment info
SELECT e.first_name, e.last_name
FROM employees e
WHERE (
    SELECT COUNT(DISTINCT department)
    FROM (
        SELECT department FROM employees
        UNION
        SELECT department FROM employee_history
    ) all_departments
) = (
    SELECT COUNT(DISTINCT department)
    FROM (
        SELECT department FROM employees WHERE id = e.id
        UNION
        SELECT department FROM employee_history WHERE em
ployee_id = e.id
    ) employee_departments
);
```

These 50+ SQL practice problems cover a wide range of SQL concepts and should provide you with a good set of exercises to enhance your SQL skills. Remember to adapt the table and column names to match your specific database schema when practicing these queries.