SQL- Queries

Comprehensive Lesson on SQL Queries (Beginner to Advanced)

This lesson provides an in-depth guide to SQL queries, starting with the basics and advancing to more complex concepts like subqueries, `DISTINCT`, `LIMIT`, and advanced filtering techniques.

---

1. Basic SQL Queries

1.1. Introduction to SQL

- SQL (Structured Query Language) is the standard language used to interact with relational databases.
- You use SQL to retrieve, manipulate, and manage data within databases like MySQL, PostgreSQL, Oracle, SQL Server, etc.

1.2. Basic `SELECT` Queries

The `SELECT` statement is used to retrieve data from a database. You can specify the columns you want to fetch or use `*` to retrieve all columns.

Syntax:

```sql
SELECT column1, column2, ...
FROM table_name;
```

Example:

```sql
SELECT name, department, salary
FROM employees;
```

---

2. Filtering Data with `WHERE` Clause

The `WHERE` clause filters records based on specified conditions. You can use operators like `=`, `>`, `<`, `>=`, `<=`, `<>` (not equal).

Syntax:

```sql
SELECT column1, column2, ...
FROM table_name
WHERE condition;
```

Example:

```sql
SELECT name, salary
FROM employees
WHERE salary > 50000;
```

Common Operators:

- `=`: Equal to

- `<>` or `!=`: Not equal to
- `>` / `<`: Greater than / Less than
- `>=` / `<=`: Greater than or equal to / Less than or equal to
- `BETWEEN`: Filters within a range.
- `IN`: Matches any value in a list.
- `LIKE`: Performs pattern matching (used with wildcards `%` and `_`).

Example with `IN` and `BETWEEN`:

```sql
SELECT name, department
FROM employees
WHERE department IN ('HR', 'Finance')
AND salary BETWEEN 50000 AND 80000;
```

---

3. Sorting Results with `ORDER BY`

`ORDER BY` allows you to sort the result set by one or more columns. You can sort in ascending (ASC) or descending (DESC) order.

Syntax:

```sql
SELECT column1, column2, ...
FROM table_name
ORDER BY column1 ASC|DESC;
```

Example:

```sql
SELECT name, salary
FROM employees
ORDER BY salary DESC;
```

---

4. Eliminating Duplicates with `DISTINCT`

The `DISTINCT` keyword removes duplicate records from the result set. This is useful when you need to see only unique values.

Syntax:

```sql
SELECT DISTINCT column1, column2, ...
FROM table_name;
```

Example:

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

---

5. Limiting Results with `LIMIT`

The `LIMIT` clause allows you to restrict the number of rows returned by the query, which is especially useful for large datasets.

Syntax (MySQL, PostgreSQL):

```sql
SELECT column1, column2, ...
```

```sql
FROM table_name
LIMIT number_of_rows;
```

Example:

```sql
SELECT name, salary
FROM employees
ORDER BY salary DESC
LIMIT 5;
```

---

6. Aggregating Data with Functions

SQL provides built-in functions to perform calculations on a set of records.

6.1. Aggregate Functions:

- `COUNT()`: Counts the number of rows.
- `SUM()`: Sums the values.
- `AVG()`: Calculates the average value.
- `MIN()` / `MAX()`: Finds the minimum or maximum value.

Example:

```sql
SELECT department, COUNT(*) AS num_employees, AVG(salary) AS avg_salary
FROM employees
GROUP BY department;
```

- **GROUP BY** groups records by a specified column, allowing you to perform aggregate functions like `COUNT`, `AVG`, etc.

6.2. Filtering Groups with `HAVING`

`HAVING` is used to filter groups based on aggregate values. It is similar to `WHERE` but applies after aggregation.

Example:

```sql
SELECT department, COUNT(*), AVG(salary)
FROM employees
GROUP BY department
HAVING COUNT(*) > 10;
```

---

7. Joining Tables

7.1. Inner Join

An `INNER JOIN` returns records that have matching values in both tables.

Syntax:

```sql
SELECT table1.column1, table2.column2, ...
FROM table1
INNER JOIN table2
ON table1.common_column = table2.common_column;
```

Example:

```sql
SELECT employees.name, departments.department_name
FROM employees
INNER JOIN departments
ON employees.department_id = departments.department_id;
```

7.2. Left Join

A `LEFT JOIN` returns all records from the left table and matching records from the right table. If there's no match, NULL is returned.

Example:

```sql
SELECT employees.name, departments.department_name
FROM employees
LEFT JOIN departments
ON employees.department_id = departments.department_id;
```

7.3. Right Join

A `RIGHT JOIN` returns all records from the right table and matching records from the left table.

Example:

```sql
SELECT employees.name, departments.department_name
FROM employees
RIGHT JOIN departments
ON employees.department_id = departments.department_id;
```

7.4. Full Outer Join

A `FULL OUTER JOIN` returns all records when there is a match in either left or right table.

Example:

```sql
SELECT employees.name, departments.department_name
FROM employees
FULL OUTER JOIN departments
ON employees.department_id = departments.department_id;
```

---

8. Subqueries

A **subquery** is a query within another query. Subqueries can be used in various parts of a SQL statement such as `SELECT`, `WHERE`, and `FROM` clauses.

8.1. Subquery in `WHERE` Clause

A subquery in the `WHERE` clause is used to filter the main query based on the results of another query.

Example:

```sql
SELECT name, salary
FROM employees
WHERE salary > (SELECT AVG(salary) FROM employees);
```

8.2. Subquery in `FROM` Clause

A subquery can be treated as a table in the `FROM` clause.

Example:

```sql
SELECT department, avg_salary
FROM (
    SELECT department, AVG(salary) AS avg_salary
    FROM employees
    GROUP BY department
) AS dept_avg
WHERE avg_salary > 60000;
```

8.3. Subquery with `IN`

The `IN` operator is often used with subqueries to filter results based on a list of values.

Example:

```sql
SELECT name
FROM employees
WHERE department_id IN (SELECT department_id FROM departments WHERE location = 'New York');
```

---

9. Advanced SQL: Window Functions

Window functions perform calculations across rows that are related to the current row.

9.1. `ROW_NUMBER()`

`ROW_NUMBER()` assigns a unique number to each row based on the specified ordering.

Example:

```sql
SELECT name, salary, ROW_NUMBER() OVER (ORDER BY salary DESC) AS row_num
FROM employees;
```

9.2. `RANK()`

`RANK()` provides a ranking for rows, but rows with the same values get the same rank, and the next rank is skipped.

Example:

```sql
SELECT name, salary, RANK() OVER (ORDER BY salary DESC) AS rank
FROM employees;
```

9.3. `PARTITION BY`

You can use `PARTITION BY` to group rows by a specific column before applying window functions.

Example:

```sql
SELECT department, name, salary, ROW_NUMBER() OVER (PARTITION BY department ORDER BY salary DESC) AS rank
FROM employees;
```

---

10. Common SQL Use Cases

10.1. Finding the Nth Highest Salary

To find the 4th highest salary using a subquery:

```sql
SELECT DISTINCT salary
FROM employees
ORDER BY salary DESC
LIMIT 1 OFFSET 3;
```

10.2. Deleting Duplicate Rows

To delete duplicate rows while keeping the first occurrence:

```sql
DELETE FROM employees
WHERE employee_id NOT IN (
    SELECT MIN(employee_id)
    FROM employees
    GROUP BY name, department, salary
);
```

Summary

This lesson has covered the full spectrum of SQL queries, from basic `SELECT` statements and filtering data to more advanced concepts like subqueries, window functions, and joins. Here's a quick recap:

- **Basic Queries**: Using `SELECT`, `WHERE`, and `ORDER BY`.
- **Aggregating Data**: Using `COUNT()`, `SUM()`, `AVG()`, and `GROUP BY`.
- **Eliminating Duplicates**: Using `DISTINCT`.
- **Limiting Results**: Using `LIMIT`.
- **Joins**: Performing `INNER`, `LEFT`, `RIGHT`, and `FULL OUTER JOIN`.
- **Subqueries**: Using subqueries for complex filtering and aggregation.
- **Window Functions**: Using `ROW_NUMBER()` and `RANK()` for advanced ranking.

Extended Lesson: `LIKE` with Wildcards and `OFFSET` with `LIMIT` in SQL

In this extension of the lesson, we'll focus on how to use the `LIKE` operator with wildcards for pattern matching and explore how `LIMIT` with `OFFSET` can help paginate or skip rows in query results.

---

1. Using `LIKE` with Wildcards

The `LIKE` operator in SQL is used to search for a specified pattern in a column. It's commonly used with **wildcards** to allow partial matches within text fields.

1.1. Wildcards Used with `LIKE`

- `%` (percent): Matches **zero or more characters**.
- `_` (underscore): Matches **exactly one character**.

1.2. Syntax

```sql
SELECT column1, column2, ...
FROM table_name
WHERE column LIKE pattern;
```

1.3. Examples of `LIKE` with Wildcards

Example 1: Search for Names that Start with 'A'

- Here, the wildcard `%` is used to match any sequence of characters after the letter 'A'. This query finds all names that start with 'A'.

```sql
SELECT name
FROM employees
WHERE name LIKE 'A%';
```

- **Explanation**: The `LIKE 'A%'` means "find all names where the first letter is 'A' and the rest can be anything (including nothing)."

Example 2: Search for Names that End with 'n'

- Use `%` at the beginning to match any sequence of characters leading up to the final letter 'n'.

```sql
SELECT name
FROM employees
WHERE name LIKE '%n';
```

- **Explanation**: The `LIKE '%n'` means "find all names where the last letter is 'n', and there can be any number of characters before it."

Example 3: Search for Names that Contain 'an'

- Here, `%` is used on both sides of the search string to match names containing the substring "an" anywhere in the name.

```sql
SELECT name
FROM employees
WHERE name LIKE '%an%';
```

- **Explanation**: The `LIKE '%an%'` means "find all names that contain 'an' anywhere."

Example 4: Search for Names with Exactly 4 Letters

- Use `_` to represent any single character, repeated four times.

```sql
SELECT name
FROM employees
WHERE name LIKE '____';
```

- **Explanation**: The `LIKE '____'` means "find all names with exactly 4 characters, where each `_` matches any single character."

Example 5: Search for Names that Start with 'A' and Have Any Second Character

- The `_` wildcard is used for a single character match.

```sql
SELECT name
FROM employees
WHERE name LIKE 'A_';
```

- **Explanation**: The `LIKE 'A_'` means "find all names where the first character is 'A' and the second character can be any single character."

1.4. Case Sensitivity of `LIKE`

- In **MySQL**, the `LIKE` operator is **case-insensitive** by default. To make it case-sensitive, you would use the `BINARY` keyword.

Example (Case-Sensitive Search):

```sql
SELECT name
FROM employees
WHERE BINARY name LIKE 'A%';
```

- In **PostgreSQL** and **Oracle**, `LIKE` is **case-sensitive** by default, but `ILIKE` can be used for case-insensitive matches.

Example (PostgreSQL - Case-Insensitive Search):

```sql
SELECT name
FROM employees
WHERE name ILIKE 'A%';
```

---

2. Using `LIMIT` and `OFFSET` for Pagination

The `LIMIT` clause is used to limit the number of rows returned by a query, while `OFFSET` allows you to skip a specific number of rows before starting to return the results. This combination is particularly useful for **pagination**, where you retrieve results in chunks (pages).

2.1. Syntax

```sql
SELECT column1, column2, ...
FROM table_name
LIMIT number_of_rows OFFSET number_of_rows_to_skip;
```

- `LIMIT`: Specifies the maximum number of rows to return.
- `OFFSET`: Specifies the number of rows to skip before returning rows.

**2.2. Examples of `LIMIT` with `OFFSET`

Example 1: Fetch the First 5 Rows

```sql
SELECT name, salary
FROM employees
ORDER BY salary DESC
```

```sql
LIMIT 5;
```

- **Explanation**: This query returns the top 5 employees with the highest salaries by ordering the result set by salary in descending order and limiting the result to 5 rows.

Example 2: Skip the First 10 Rows and Fetch the Next 5 Rows

```sql
SELECT name, salary
FROM employees
ORDER BY salary DESC
LIMIT 5 OFFSET 10;
```

- **Explanation**: This query skips the first 10 rows and then returns the next 5 rows (i.e., rows 11 through 15 after ordering the employees by salary).

Example 3: Paginating Results

To paginate results, you can combine `LIMIT` and `OFFSET`. For example, if you want to display **5 rows per page**:

- **Page 1**: Fetch the first 5 rows

  ```sql
  SELECT name, salary
  FROM employees
  ORDER BY salary DESC
  LIMIT 5 OFFSET 0;   -- Skip 0 rows
  ```

- **Page 2**: Fetch the next 5 rows

  ```sql
  SELECT name, salary
  FROM employees
  ORDER BY salary DESC
  LIMIT 5 OFFSET 5;   -- Skip 5 rows
  ```

- **Page 3**: Fetch the next 5 rows

  ```sql
  SELECT name, salary
  FROM employees
  ORDER BY salary DESC
  LIMIT 5 OFFSET 10;   -- Skip 10 rows
  ```

2.3. Combining `DISTINCT`, `LIMIT`, and `OFFSET`

If you want to limit distinct rows, you can combine the `DISTINCT`, `LIMIT`, and `OFFSET` clauses.

Example: Fetch 3 Distinct Salaries, Skipping the Top 2

```sql
SELECT DISTINCT salary
FROM employees
ORDER BY salary DESC
LIMIT 3 OFFSET 2;
```

- **Explanation**: This query first retrieves distinct salaries, orders them by salary in descending order, skips the top 2 salaries, and returns the next 3 distinct salaries.

---

3. Use Case: Finding the 4th Highest Salary Using `LIMIT` and `OFFSET`

Now that you understand both `LIKE` and `LIMIT/OFFSET`, let's revisit the query to find the 4th highest salary, using `LIMIT` and `OFFSET`.

Example:

```sql
SELECT DISTINCT salary
FROM employees
ORDER BY salary DESC
LIMIT 1 OFFSET 3;
```

- **Explanation**: This query orders the distinct salaries in descending order, skips the top 3 salaries, and retrieves the next salary (which will be the 4th highest).

---

4. Summary

- `LIKE` is an important SQL operator for pattern matching, especially when you want to find records that partially match a string.
  - `%` matches any sequence of characters (including none).
  - `_` matches exactly one character.
- `LIMIT` and `OFFSET` are used to restrict the number of rows returned and to skip rows for pagination or other purposes.
  - `LIMIT` specifies how many rows to return.
  - `OFFSET` specifies how many rows to skip before starting to return rows.

By using `LIKE`, `LIMIT`, and `OFFSET`, you can perform sophisticated data filtering, partial matches, and controlled result retrieval.

```sql
select e.first_name, e.last_name, a.department_name from
(SELECT department_id, max(salary) as maxsal,
  d.name as department_name
FROM employees   AS e
JOIN departments AS d ON e.department_id = d.id
group by department_id) AS a
join employees AS e on e.department_id = a.department_id and e.salary = a.maxsal
```

```sql
WITH DepartmentMaxSalaries AS (
    SELECT
        department_id,
        MAX(salary) AS max_salary,
        d.name AS department_name
    FROM
        employees AS e
    JOIN
        departments AS d ON e.department_id = d.id
    GROUP BY
        department_id, d.name -- Group by department name to prevent ambiguity if two departments have the same name.
)
SELECT
```

```sql
    e.id,
    e.first_name,
    e.last_name,
    e.salary,
    dms.department_name
FROM
    employees AS e
JOIN
    DepartmentMaxSalaries AS dms ON e.department_id = dms.department_id AND e.salary = dms.max_salary;
```