SQL Cheatsheet

A comprehensive guide to SQL syntax, commands, and best practices for data manipulation and database management.

Basic SQL Commands

SELECT Statement

The `SELECT` statement is used to retrieve data from one or more tables.

Syntax:

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

Example:

```
SELECT customer_id, customer_name
FROM Customers
WHERE city = 'New York';
```

Select all columns using `*`.

```
SELECT *
FROM Employees;
```

INSERT Statement

The `INSERT` statement is used to insert new data into a table.

Syntax:

```
INSERT INTO table_name (column1, column2, ...)
VALUES (value1, value2, ...);
```

Example:

```
INSERT INTO Customers (customer_name, city,
country)
VALUES ('John Doe', 'Los Angeles', 'USA');
```

Insert into all columns (order matters!).

```
INSERT INTO Products
VALUES (1, 'Laptop', 1200.00);
```

UPDATE Statement

The `UPDATE` statement is used to modify existing data in a table.

Syntax:

```
UPDATE table_name
SET column1 = value1, column2 = value2, ...
WHERE condition;
```

Example:

```
UPDATE Employees
SET salary = salary * 1.10
WHERE department = 'Sales';
```

DELETE Statement

The `DELETE` statement is used to delete existing records from a table.

Syntax:

```
DELETE FROM table_name
WHERE condition;
```

Example:

```
DELETE FROM Orders
WHERE order_date < '2023-01-01';
```

Delete all records from a table (use with caution!).

```
DELETE FROM table_name;
```

CREATE TABLE Statement

The `CREATE TABLE` statement is used to create a new table in a database.

Syntax:

```
CREATE TABLE table_name (
    column1 datatype constraints,
    column2 datatype constraints,
    ...
);
```

Example:

```
CREATE TABLE Products (
    product_id INT PRIMARY KEY,
    product_name VARCHAR(255),
    price DECIMAL(10, 2)
);
```

DROP TABLE Statement

The `DROP TABLE` statement is used to delete an existing table in a database.

Syntax:

```
DROP TABLE table_name;
```

Example:

```
DROP TABLE Products;
```

Filtering and Sorting

WHERE Clause

The `WHERE` clause is used to filter records based on specified conditions.

Syntax:

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

Example:

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

Using `AND` and `OR` operators.

```
SELECT *
FROM Products
WHERE category = 'Electronics' AND price <
1000;
```

ORDER BY Clause

The `ORDER BY` clause is used to sort the result-set in ascending or descending order.

Syntax:

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

Example:

```
SELECT *
FROM Customers
ORDER BY customer_name ASC;
```

Sorting by multiple columns.

```
SELECT *
FROM Orders
ORDER BY order_date DESC, customer_id ASC;
```

LIKE Operator

The `LIKE` operator is used in a `WHERE` clause to search for a specified pattern in a column.

Syntax:

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

`%` - Represents zero or more characters
`_` - Represents a single character

Example:

```
SELECT *
FROM Products
WHERE product_name LIKE 'Laptop%'; -- Starts
with 'Laptop'
```

```
SELECT *
FROM Customers
WHERE city LIKE '_ondon'; -- Second character
is 'o' and ends with 'ondon'
```

BETWEEN Operator

The `BETWEEN` operator is used to select values within a given range.

Syntax:

```
SELECT column1, column2, ...
FROM table_name
WHERE columnN BETWEEN value1 AND value2;
```

Example:

```
SELECT *
FROM Orders
WHERE order_date BETWEEN '2023-01-01' AND
'2023-03-31';
```

IN Operator

The `IN` operator is used to specify multiple values in a `WHERE` clause.

Syntax:

```
SELECT column1, column2, ...
FROM table_name
WHERE columnN IN (value1, value2, ...);
```

Example:

```
SELECT *
FROM Customers
WHERE country IN ('USA', 'Canada', 'UK');
```

DISTINCT Keyword

The `DISTINCT` keyword is used to select only distinct (unique) values.

Syntax:

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

Example:

```
SELECT DISTINCT country
FROM Customers;
```

Joins and Unions

INNER JOIN

The `INNER JOIN` keyword selects records that have matching values in both tables.

Syntax:

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

Example:

```
SELECT Orders.order_id,
Customers.customer_name
FROM Orders
INNER JOIN Customers
ON Orders.customer_id = Customers.customer_id;
```

LEFT JOIN

The `LEFT JOIN` keyword returns all records from the left table (table1), and the matched records from the right table (table2). The result is NULL from the right side, if there is no match.

Syntax:

```
SELECT column1, column2, ...
FROM table1
LEFT JOIN table2
ON table1.column_name = table2.column_name;
```

Example:

```
SELECT Customers.customer_name,
Orders.order_id
FROM Customers
LEFT JOIN Orders
ON Customers.customer_id = Orders.customer_id;
```

RIGHT JOIN

The `RIGHT JOIN` keyword returns all records from the right table (table2), and the matched records from the left table (table1). The result is NULL from the left side, when there is no match.

Syntax:

```
SELECT column1, column2, ...
FROM table1
RIGHT JOIN table2
ON table1.column_name = table2.column_name;
```

Example:

```
SELECT Customers.customer_name,
Orders.order_id
FROM Customers
RIGHT JOIN Orders
ON Customers.customer_id = Orders.customer_id;
```

FULL OUTER JOIN

The `FULL OUTER JOIN` keyword returns all records when there is a match in either left (table1) or right (table2) table records.

Syntax:

```
SELECT column1, column2, ...
FROM table1
FULL OUTER JOIN table2
ON table1.column_name = table2.column_name;
```

Example:

```
SELECT Customers.customer_name,
Orders.order_id
FROM Customers
FULL OUTER JOIN Orders
ON Customers.customer_id = Orders.customer_id;
```

UNION Operator

The `UNION` operator is used to combine the result-set of two or more `SELECT` statements. It removes duplicate rows.

Syntax:

```
SELECT column1, column2, ...
FROM table1
WHERE condition
UNION
SELECT column1, column2, ...
FROM table2
WHERE condition;
```

Example:

```
SELECT city FROM Customers
UNION
SELECT city FROM Suppliers
ORDER BY city;
```

UNION ALL Operator

The `UNION ALL` operator is used to combine the result-set of two or more `SELECT` statements. It does not remove duplicate rows.

Syntax:

```
SELECT column1, column2, ...
FROM table1
WHERE condition
UNION ALL
SELECT column1, column2, ...
FROM table2
WHERE condition;
```

Example:

```
SELECT city FROM Customers
UNION ALL
SELECT city FROM Suppliers
ORDER BY city;
```

Aggregate Functions and Grouping

Aggregate Functions

Aggregate functions perform a calculation on a set of values and return a single value.

- `COUNT()` - Returns the number of rows.
- `AVG()` - Returns the average value.
- `SUM()` - Returns the sum of all values.
- `MIN()` - Returns the minimum value.
- `MAX()` - Returns the maximum value.

Example:

```
SELECT COUNT(customer_id) AS TotalCustomers
FROM Customers;
```

```
SELECT AVG(salary) AS AverageSalary
FROM Employees;
```

GROUP BY Clause

The `GROUP BY` clause is used to group rows that have the same values in specified columns into summary rows.

Syntax:

```
SELECT column1, column2, ...
FROM table_name
WHERE condition
GROUP BY column1, column2, ...
ORDER BY column1, column2, ...;
```

Example:

```
SELECT department, AVG(salary) AS
AverageSalary
FROM Employees
GROUP BY department;
```

HAVING Clause

The `HAVING` clause is used to filter the results of a `GROUP BY` query based on a specified condition.

Syntax:

```
SELECT column1, column2, ...
FROM table_name
WHERE condition
GROUP BY column1, column2, ...
HAVING condition
ORDER BY column1, column2, ...;
```

Example:

```
SELECT department, AVG(salary) AS
AverageSalary
FROM Employees
GROUP BY department
HAVING AVG(salary) > 60000;
```

Subqueries

A subquery is a query nested inside another query.

Syntax:

```
SELECT column1, column2, ...
FROM table_name
WHERE columnN IN (SELECT columnN FROM
another_table WHERE condition);
```

Example:

```
SELECT product_name
FROM Products
WHERE price > (SELECT AVG(price) FROM
Products);
```

EXISTS Operator

The `EXISTS` operator is used to test for the existence of any record in a subquery.

Syntax:

```
SELECT column1, column2, ...
FROM table_name
WHERE EXISTS (SELECT columnN FROM
another_table WHERE condition);
```

Example:

```
SELECT customer_name
FROM Customers
WHERE EXISTS (SELECT order_id FROM Orders
WHERE Orders.customer_id =
Customers.customer_id);
```