Cheat Sheet for comprensive SQL

Comprehensive SQL Cheat Sheet

1. Introduction to SQL

- **SQL (Structured Query Language)** is a domain-specific language used for managing and manipulating relational databases.

- **Relational Database**: A collection of tables with rows and columns.

- **Primary Key**: A column or set of columns that uniquely identifies each row in a table.

- **Foreign Key**: A column that refers to the primary key of another table.

---

2. Basic SQL Commands

- **SELECT**: Retrieve data from a database.

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

- **FROM**: Specifies the table from which to retrieve data.

```
SELECT * FROM employees;
```

- **WHERE**: Filters records based on a condition.

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

- **ORDER BY**: Sorts the result set.

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

- **LIMIT**: Limits the number of rows returned.

```
SELECT * FROM employees LIMIT 10;
```

---

3. Data Manipulation Language (DML)

- **INSERT**: Adds new records to a table.

```
INSERT INTO employees (name, department) VALUES ('John Doe', 'HR');
```

- **UPDATE**: Modifies existing records.

```
UPDATE employees SET department = 'Marketing' WHERE name = 'John Doe';
```

- **DELETE**: Removes records from a table.

```
DELETE FROM employees WHERE name = 'John Doe';
```

---

4. Data Definition Language (DDL)

- **CREATE TABLE**: Creates a new table.

```
CREATE TABLE employees (
    id INT PRIMARY KEY,
    name VARCHAR(100),
    department VARCHAR(50)
);
```

- **ALTER TABLE**: Modifies an existing table.

```
ALTER TABLE employees ADD COLUMN email VARCHAR(100);
```

- **DROP TABLE**: Deletes a table.

```
DROP TABLE employees;
```

- **TRUNCATE TABLE**: Removes all records from a table.

```
TRUNCATE TABLE employees;
```

---

5. Data Control Language (DCL)

- **GRANT**: Gives a user specific access rights.

```
GRANT SELECT, INSERT ON employees TO 'user'@'localhost';
```

- **REVOKE**: Takes back access rights from a user.

```
REVOKE INSERT ON employees FROM 'user'@'localhost';
```

---

6. Transaction Control Language (TCL)

- **COMMIT**: Saves the transaction.

```
COMMIT;
```

- **ROLLBACK**: Reverts the transaction.

```
ROLLBACK;
```

- **SAVEPOINT**: Sets a savepoint within a transaction.

```
SAVEPOINT savepoint_name;
```

---

7. Advanced SQL Features

- **GROUP BY**: Groups rows that have the same values.

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

- **HAVING**: Filters groups based on a condition.

```
  SELECT department, AVG(salary) FROM employees GROUP BY department
HAVING AVG(salary) > 50000;
```

- **DISTINCT**: Returns unique values.

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

- **CASE**: Performs conditional logic.

```
  SELECT name, salary,
      CASE
          WHEN salary > 70000 THEN 'High'
          ELSE 'Low'
      END AS salary_level
  FROM employees;
```

---

8. SQL Shortcuts and Tips

- **Aliases**: Use `AS` to create aliases for columns or tables.

```
  SELECT name AS employee_name FROM employees;
```

- **Comments**: Use `--` for single-line comments and `/* */` for multi-line comments.

```
  -- This is a single-line comment
  /*
  This is a
  multi-line comment
  */
```

- **Concatenation**: Use `||` (Oracle) or `CONCAT()` (MySQL).

```
  SELECT CONCAT(first_name, ' ', last_name) AS full_name FROM
employees;
```

---

9. Common SQL Functions

- **Aggregate Functions**:

- `COUNT()`: Counts the number of rows.
- `SUM()`: Adds up the values in a column.
- `AVG()`: Calculates the average value.
- `MIN()`: Finds the minimum value.
- `MAX()`: Finds the maximum value.

- **String Functions**:

- `LENGTH()`: Returns the length of a string.
- `UPPER()`: Converts a string to uppercase.
- `LOWER()`: Converts a string to lowercase.
- `SUBSTRING()`: Extracts a part of a string.

- **Date Functions**:

- `NOW()`: Returns the current date and time.
- `DATE()`: Extracts the date part from a datetime.
- `DATEDIFF()`: Calculates the difference between two dates.

---

10. SQL Joins

- **INNER JOIN**: Returns records that have matching values in both tables.

```
  SELECT * FROM employees INNER JOIN departments ON
employees.department_id = departments.id;
```

- **LEFT JOIN**: Returns all records from the left table and the matched records from the right table.

```
  SELECT * FROM employees LEFT JOIN departments ON
employees.department_id = departments.id;
```

- **RIGHT JOIN**: Returns all records from the right table and the matched records from the left table.

```
  SELECT * FROM employees RIGHT JOIN departments ON
employees.department_id = departments.id;
```

- **FULL OUTER JOIN**: Returns all records when there is a match in either left or right table.

```
  SELECT * FROM employees FULL OUTER JOIN departments ON
employees.department_id = departments.id;
```

---

11. Subqueries and Common Table Expressions (CTEs)

- **Subquery**: A query nested inside another query.

```
  SELECT name FROM employees WHERE department_id IN (SELECT id FROM
departments WHERE name = 'Sales');
```

- **CTE (Common Table Expression)**: A temporary result set.

```
  WITH sales_dept AS (
      SELECT id FROM departments WHERE name = 'Sales'
  )
  SELECT name FROM employees WHERE department_id IN (SELECT id FROM
sales_dept);
```

---

12. Indexes and Constraints

- **INDEX**: Improves the speed of data retrieval operations.

```
  CREATE INDEX idx_name ON employees (name);
```

- **PRIMARY KEY**: Ensures that a column or set of columns is unique.

```
  ALTER TABLE employees ADD PRIMARY KEY (id);
```

- **FOREIGN KEY**: Establishes a link between two tables.

```
  ALTER TABLE employees ADD FOREIGN KEY (department_id) REFERENCES
departments(id);
```

- **UNIQUE**: Ensures that all values in a column are unique.

```
ALTER TABLE employees ADD UNIQUE (email);
```

---

13. Views and Stored Procedures

- **VIEW**: A virtual table based on the result-set of an SQL statement.

```
CREATE VIEW sales_employees AS
SELECT name, salary FROM employees WHERE department = 'Sales';
```

- **STORED PROCEDURE**: A precompiled collection of SQL statements.

```
CREATE PROCEDURE GetEmployeeDetails()
BEGIN
    SELECT * FROM employees;
END;
```

---

14. SQL Best Practices

- **Use Indexes**: For frequently queried columns.
- **Normalize Data**: Reduce redundancy and improve data integrity.
- **Use Transactions**: Ensure data consistency.
- **Avoid SELECT***: Specify columns to retrieve.
- **Use Proper Naming Conventions**: For tables, columns, and variables.
- **Comment Code**: For better readability and maintainability.

This cheat sheet provides a comprehensive overview of SQL, covering essential commands, advanced features, and best practices. Use it as a quick reference guide for your SQL queries and database management tasks.

By Ahmed Baheeg Khorshid

ver 1.0