# SQL Fundamentals Cheat Sheet

## Table of Contents



---

## Basic Query Structure

```sql
SELECT column1, column2
FROM table_name
WHERE condition
GROUP BY column
HAVING condition
ORDER BY column
LIMIT number;
```

**Order of execution:**

1. FROM - Specify table
2. WHERE - Filter rows
3. GROUP BY - Group rows
4. HAVING - Filter groups
5. SELECT - Choose columns

6. ORDER BY - Sort results

7. LIMIT - Limit results

---

# Data Types

## Numeric Types

sql

```sql
INT, INTEGER        -- Whole numbers
DECIMAL(10,2)       -- Fixed-point (10 digits, 2 decimal places)
FLOAT, REAL         -- Floating-point numbers
BOOLEAN             -- True/False values
```

## String Types

sql

```sql
CHAR(10)            -- Fixed-length string (10 characters)
VARCHAR(255)        -- Variable-length string (up to 255 chars)
TEXT                -- Large text data
```

## Date/Time Types

sql

```sql
DATE                -- Date only (YYYY-MM-DD)
TIME                -- Time only (HH:MM:SS)
DATETIME            -- Date and time
TIMESTAMP           -- Date and time with timezone
```

---

# Creating & Modifying Tables

## Create Table

sql

```sql
CREATE TABLE employees (
    id INT PRIMARY KEY AUTO_INCREMENT,
    name VARCHAR(100) NOT NULL,
    email VARCHAR(100) UNIQUE,
    salary DECIMAL(10,2),
    hire_date DATE,
    department_id INT
);
```

### Alter Table

```sql
-- Add column
ALTER TABLE employees ADD phone VARCHAR(20);

-- Drop column
ALTER TABLE employees DROP COLUMN phone;

-- Modify column
ALTER TABLE employees MODIFY COLUMN salary DECIMAL(12,2);

-- Rename column
ALTER TABLE employees RENAME COLUMN name TO full_name;
```

### Drop Table

```sql
DROP TABLE employees;
```

---

## Inserting, Updating & Deleting Data

### Insert Data

```sql
-- Insert single row
INSERT INTO employees (name, email, salary)
VALUES ('John Doe', 'john@email.com', 50000);

-- Insert multiple rows
INSERT INTO employees (name, email, salary)
VALUES
    ('Jane Smith', 'jane@email.com', 60000),
    ('Bob Johnson', 'bob@email.com', 55000);
```

### Update Data

```sql
```

```sql
-- Update specific rows
UPDATE employees
SET salary = 55000
WHERE id = 1;

-- Update multiple columns
UPDATE employees
SET salary = salary * 1.1,
    department_id = 2
WHERE department_id = 1;
```

### Delete Data

```sql
-- Delete specific rows
DELETE FROM employees WHERE id = 1;

-- Delete all rows (but keep table structure)
DELETE FROM employees;
```

---

## Selecting Data

### Basic Select

```sql
-- Select all columns
SELECT * FROM employees;

-- Select specific columns
SELECT name, salary FROM employees;

-- Select with alias
SELECT name AS employee_name, salary AS annual_salary
FROM employees;

-- Select distinct values
SELECT DISTINCT department_id FROM employees;
```

### Calculated Fields

```sql
```

```sql
SELECT
    name,
    salary,
    salary * 12 AS annual_salary,
    CONCAT(name, ' - ', email) AS contact_info
FROM employees;
```

---

# Filtering Data

## WHERE Clause

sql

```sql
-- Basic conditions
SELECT * FROM employees WHERE salary > 50000;
SELECT * FROM employees WHERE department_id = 1;
SELECT * FROM employees WHERE name = 'John Doe';

-- Multiple conditions
SELECT * FROM employees
WHERE salary > 50000 AND department_id = 1;

SELECT * FROM employees
WHERE salary > 60000 OR department_id = 2;

-- NOT condition
SELECT * FROM employees WHERE NOT department_id = 1;
```

## Comparison Operators

sql

```sql
=    -- Equal to
<>   -- Not equal to (also !=)
>    -- Greater than
<    -- Less than
>=   -- Greater than or equal to
<=   -- Less than or equal to
```

## Pattern Matching

sql

```sql
-- LIKE operator
SELECT * FROM employees WHERE name LIKE 'J%';       -- Starts with J
SELECT * FROM employees WHERE name LIKE '%son';     -- Ends with son
SELECT * FROM employees WHERE name LIKE '%oh%';     -- Contains oh

-- Wildcards
%    -- Zero or more characters
_    -- Exactly one character
```

### Range and List Conditions

```sql
-- BETWEEN
SELECT * FROM employees WHERE salary BETWEEN 45000 AND 65000;

-- IN
SELECT * FROM employees WHERE department_id IN (1, 2, 3);

-- NOT IN
SELECT * FROM employees WHERE department_id NOT IN (1, 2);
```

### NULL Values

```sql
-- Check for NULL
SELECT * FROM employees WHERE phone IS NULL;

-- Check for NOT NULL
SELECT * FROM employees WHERE phone IS NOT NULL;
```

---

## Sorting & Limiting

### ORDER BY

```sql
```

```sql
-- Sort ascending (default)
SELECT * FROM employees ORDER BY salary;

-- Sort descending
SELECT * FROM employees ORDER BY salary DESC;

-- Multiple columns
SELECT * FROM employees ORDER BY department_id, salary DESC;
```

### LIMIT

sql

```sql
-- Limit results
SELECT * FROM employees ORDER BY salary DESC LIMIT 10;

-- Offset and limit (pagination)
SELECT * FROM employees ORDER BY id LIMIT 10 OFFSET 20;
```

---

## Grouping & Aggregation

### Aggregate Functions

sql

```sql
COUNT(*)          -- Count all rows
COUNT(column)     -- Count non-null values
SUM(column)       -- Sum of values
AVG(column)       -- Average of values
MIN(column)       -- Minimum value
MAX(column)       -- Maximum value
```

### GROUP BY

sql

```sql
-- Basic grouping
SELECT department_id, COUNT(*) as employee_count
FROM employees
GROUP BY department_id;

-- Multiple aggregations
SELECT
    department_id,
    COUNT(*) as employee_count,
    AVG(salary) as avg_salary,
    MAX(salary) as max_salary
FROM employees
GROUP BY department_id;
```

### HAVING

```sql
-- Filter groups (use HAVING instead of WHERE for aggregates)
SELECT department_id, COUNT(*) as employee_count
FROM employees
GROUP BY department_id
HAVING COUNT(*) > 5;

SELECT department_id, AVG(salary) as avg_salary
FROM employees
GROUP BY department_id
HAVING AVG(salary) > 55000;
```

---

## Joins

### Inner Join

```sql
-- Returns only matching rows from both tables
SELECT e.name, d.department_name
FROM employees e
INNER JOIN departments d ON e.department_id = d.id;
```

### Left Join

```sql
```

```sql
-- Returns all rows from left table, matching rows from right
SELECT e.name, d.department_name
FROM employees e
LEFT JOIN departments d ON e.department_id = d.id;
```

### Right Join

```sql
-- Returns all rows from right table, matching rows from left
SELECT e.name, d.department_name
FROM employees e
RIGHT JOIN departments d ON e.department_id = d.id;
```

### Full Outer Join

```sql
-- Returns all rows from both tables
SELECT e.name, d.department_name
FROM employees e
FULL OUTER JOIN departments d ON e.department_id = d.id;
```

### Self Join

```sql
-- Join table with itself
SELECT e1.name as employee, e2.name as manager
FROM employees e1
JOIN employees e2 ON e1.manager_id = e2.id;
```

---

## Subqueries

### Subquery in WHERE

```sql
```

```sql
-- Find employees with above-average salary
SELECT name, salary
FROM employees
WHERE salary > (SELECT AVG(salary) FROM employees);

-- Find employees in specific departments
SELECT name
FROM employees
WHERE department_id IN (
    SELECT id FROM departments WHERE location = 'New York'
);
```

### Subquery in FROM

```sql
-- Use subquery as a table
SELECT dept_stats.department_id, dept_stats.avg_salary
FROM (
    SELECT department_id, AVG(salary) as avg_salary
    FROM employees
    GROUP BY department_id
) as dept_stats
WHERE dept_stats.avg_salary > 55000;
```

### Correlated Subquery

```sql
-- Subquery references outer query
SELECT name, salary
FROM employees e1
WHERE salary > (
    SELECT AVG(salary)
    FROM employees e2
    WHERE e2.department_id = e1.department_id
);
```

---

## Common Functions

### String Functions

```sql
```

```
CONCAT(str1, str2)          -- Concatenate strings
UPPER(string)               -- Convert to uppercase
LOWER(string)               -- Convert to lowercase
LENGTH(string)              -- String length
SUBSTRING(string, start, length)  -- Extract substring
TRIM(string)                -- Remove leading/trailing spaces
REPLACE(string, old, new)   -- Replace text
```

## Numeric Functions

sql

```
ROUND(number, decimals)     -- Round number
CEILING(number)             -- Round up
FLOOR(number)               -- Round down
ABS(number)                 -- Absolute value
MOD(number, divisor)        -- Modulo operation
```

## Date Functions

sql

```
NOW()                       -- Current date and time
CURDATE()                   -- Current date
CURTIME()                   -- Current time
DATE_ADD(date, INTERVAL 1 DAY)  -- Add time interval
DATE_SUB(date, INTERVAL 1 MONTH) -- Subtract time interval
DATEDIFF(date1, date2)      -- Difference in days
DATE_FORMAT(date, format)   -- Format date
```

## Conditional Functions

sql

```sql
-- CASE statement
SELECT name,
    CASE
        WHEN salary > 60000 THEN 'High'
        WHEN salary > 40000 THEN 'Medium'
        ELSE 'Low'
    END as salary_category
FROM employees;

-- COALESCE (return first non-null value)
SELECT name, COALESCE(phone, email, 'No contact') as contact
FROM employees;

-- NULLIF (return null if values are equal)
SELECT name, NULLIF(salary, 0) as salary
FROM employees;
```

---

## Constraints

### Primary Key

sql

```sql
-- Single column primary key
CREATE TABLE employees (
    id INT PRIMARY KEY,
    name VARCHAR(100)
);

-- Composite primary key
CREATE TABLE order_items (
    order_id INT,
    product_id INT,
    quantity INT,
    PRIMARY KEY (order_id, product_id)
);
```

### Foreign Key

sql

```sql
CREATE TABLE employees (
    id INT PRIMARY KEY,
    department_id INT,
    FOREIGN KEY (department_id) REFERENCES departments(id)
);

-- With actions
ALTER TABLE employees
ADD CONSTRAINT fk_department
FOREIGN KEY (department_id) REFERENCES departments(id)
ON DELETE CASCADE
ON UPDATE CASCADE;
```

### Other Constraints

```sql
-- NOT NULL
name VARCHAR(100) NOT NULL

-- UNIQUE
email VARCHAR(100) UNIQUE

-- CHECK
salary DECIMAL(10,2) CHECK (salary > 0)

-- DEFAULT
hire_date DATE DEFAULT CURRENT_DATE
```

---

## Indexes

### Create Index

```sql
-- Simple index
CREATE INDEX idx_salary ON employees(salary);

-- Composite index
CREATE INDEX idx_dept_salary ON employees(department_id, salary);

-- Unique index
CREATE UNIQUE INDEX idx_email ON employees(email);
```

### Drop Index

sql

```sql
DROP INDEX idx_salary;
```

---

## Common Patterns

### Pagination

sql

```sql
-- MySQL/PostgreSQL
SELECT * FROM employees ORDER BY id LIMIT 10 OFFSET 20;

-- SQL Server
SELECT * FROM employees ORDER BY id OFFSET 20 ROWS FETCH NEXT 10 ROWS ONLY;
```

### Top N Records

sql

```sql
-- Top 5 highest paid employees
SELECT * FROM employees ORDER BY salary DESC LIMIT 5;

-- Top 3 employees per department
SELECT *
FROM (
    SELECT *,
        ROW_NUMBER() OVER (PARTITION BY department_id ORDER BY salary DESC) as rn
    FROM employees
) ranked
WHERE rn <= 3;
```

### Duplicate Detection

sql

```sql
-- Find duplicates
SELECT email, COUNT(*)
FROM employees
GROUP BY email
HAVING COUNT(*) > 1;

-- Remove duplicates (keep one)
DELETE e1 FROM employees e1
INNER JOIN employees e2
WHERE e1.id > e2.id AND e1.email = e2.email;
```

### Conditional Aggregation

```sql
-- Count employees by salary range
SELECT
    department_id,
    COUNT(*) as total_employees,
    COUNT(CASE WHEN salary > 50000 THEN 1 END) as high_salary_count,
    COUNT(CASE WHEN salary <= 50000 THEN 1 END) as low_salary_count
FROM employees
GROUP BY department_id;
```

### Running Totals

```sql
-- Running sum of salaries
SELECT
    name,
    salary,
    SUM(salary) OVER (ORDER BY id) as running_total
FROM employees;
```

---

## Tips for Strong Fundamentals

1. **Always use proper JOIN syntax** instead of comma-separated tables in FROM clause
2. **Use aliases** to make queries more readable
3. **Be specific with column names** in SELECT statements
4. **Use appropriate data types** and constraints
5. **Index frequently queried columns** for better performance
6. **Use LIMIT** when testing queries on large datasets
7. **Always backup before running DELETE or UPDATE** statements
8. **Use transactions** for multiple related operations
9. **Understand NULL behavior** in comparisons and functions
10. **Practice query optimization** by analyzing execution plans

This cheat sheet covers the essential SQL fundamentals. Practice these concepts regularly to build a strong foundation!