Comprehensive SQL Cheat Sheet

Generated on May 8, 2025

Contents

1 Basic Queries

```sql
-- Select specific columns
SELECT col1, col2, col3
FROM table1;

-- Filter rows
SELECT *
FROM table1
WHERE col4 = 1 AND col5 = 2;

-- Aggregate data
SELECT colA, COUNT(*)
FROM table1
GROUP BY colA
HAVING COUNT(*) > 1;

-- Order results
SELECT col1, col2
FROM table1
ORDER BY col2;

-- Distinct values
SELECT DISTINCT col
FROM table1;

-- Between
SELECT *
FROM table1
WHERE value BETWEEN 10 AND 20;

-- Like
SELECT *
FROM table1
WHERE name LIKE 'A%';

-- In-list
SELECT *
```

```
FROM table1
WHERE category IN ('a','b','c');
```

2 Data Modification

```
-- Update specific rows
UPDATE table1
SET col1 = 'new'
WHERE col2 = 2;

-- Insert values manually
INSERT INTO table1 (ID, FIRST_NAME, LAST_NAME)
VALUES (1, 'Rebel','Labs');

-- Insert from query
INSERT INTO table2
SELECT id, last_name, first_name
FROM table1;
```

3 Views

A **VIEW** is a virtual table based on a query.

```
CREATE VIEW view1 AS
SELECT col1, col2
FROM table1
WHERE ...;
```

4 The Joy of JOINS

4.1 Inner Join

```
SELECT *
FROM A
INNER JOIN B ON A.id = B.a_id;
```

4.2 Left Outer Join

```
SELECT *
FROM A
LEFT JOIN B ON A.id = B.a_id;
-- all rows from A
```

4.3 Right Outer Join

```
SELECT *
FROM A
RIGHT JOIN B ON A.id = B.a_id;
-- all rows from B
```

4.4 Full Outer Join

```
SELECT *
FROM A
FULL JOIN B ON A.id = B.a_id;
-- all rows from A and B
```

4.5 Cross Join

```
SELECT *
FROM A
CROSS JOIN B;
```

4.6 Self Join

```
SELECT e1.name AS Emp, e2.name AS Mgr
FROM Employees e1
JOIN Employees e2 ON e1.manager_id = e2.id;
```

5 Updates on JOINed Queries

```
UPDATE t1
SET col = 1
FROM table1 t1
JOIN table2 t2 ON t1.id = t2.id
WHERE t2.col2 IS NULL;
```

6 Semi-JOINs

Use subqueries instead of JOIN:

```
SELECT col1, col2
FROM table1
WHERE id IN (
  SELECT ref_id
  FROM table2
  WHERE date > CURRENT_TIMESTAMP
);
```

7 Indexes

If you query by a column, index it:

```
CREATE INDEX idx1 ON table1(col1);
```

Recommendations:

- Avoid overlapping indexes
- Avoid indexing too many columns
- Indexes speed DELETE and UPDATE

8 Useful Utility Functions

```
-- Convert strings to dates
TO_DATE('2020-01-01','YYYY-MM-DD');
STR_TO_DATE('01-01-2020','%m-%d-%Y');

-- First non-null
COALESCE(col1, col2, 'default');

-- Current time
CURRENT_TIMESTAMP;
```

9 Reporting

Use aggregation:

```
COUNT(*), SUM(col), AVG(col), MIN(col), MAX(col);
```

10 Selection Queries (LearnSQL)

Fetch all columns:

```
SELECT *
FROM country;
```

Fetch specific columns:

```
SELECT id, name
FROM city;
```

Sort results:

```
SELECT name
FROM city
ORDER BY rating DESC;
```

Aliases:

```
SELECT name AS city_name
FROM city;
```

11 Conditional Queries

```
-- CASE
SELECT
  CASE
    WHEN price < 50 THEN 'Budget'
    WHEN price BETWEEN 50 AND 100 THEN 'Mid'
    ELSE 'Premium'
  END AS category
FROM products;

-- COALESCE
SELECT COALESCE(desc, 'No desc')
FROM products;

-- CAST
SELECT CAST(order_date AS DATE)
FROM orders;
```

12 Window Functions (PostgreSQL)

```
-- Partition
SELECT
  officeCode,
  SUM(length) OVER (PARTITION BY officeCode) AS sum_len
FROM employees;

-- Running total
SELECT
  id,
  SUM(amount) OVER (ORDER BY date) AS running_total
FROM sales;
```

13 Ranking Functions

```
-- DENSE_RANK
SELECT
  product,
  DENSE_RANK() OVER (ORDER BY price DESC) AS rank
FROM products;

-- RANK
SELECT
  employee,
  RANK() OVER (PARTITION BY office ORDER BY length DESC) AS rnk
FROM employees;

-- ROW_NUMBER
SELECT
```

```
  orderNumber,
  ROW_NUMBER() OVER (ORDER BY orderDate) AS rn
FROM orders;
```

14 Subqueries

14.1 SELECT

```
SELECT
  name,
  (SELECT AVG(price) FROM products) AS avg_price
FROM products;
```

14.2 FROM

```
SELECT x.name, x.avg_price
FROM (
  SELECT productLine, AVG(price) AS avg_price
  FROM products
  GROUP BY productLine
) AS x
WHERE x.avg_price > 100;
```

14.3 WHERE

```
SELECT name
FROM products p
WHERE price > (
  SELECT AVG(price)
  FROM products
  WHERE p.productLine = productLine
);
```

14.4 IN / EXISTS

```
SELECT name
FROM products
WHERE code IN (
  SELECT code FROM orderdetails WHERE orderNumber = 10100
);

SELECT name
FROM customers c
WHERE EXISTS (
  SELECT 1 FROM orders o
  WHERE o.customerNumber = c.customerNumber
);
```

15 Combine Data: Set Operations

```
-- UNION / UNION ALL
SELECT name FROM cycling WHERE country='DE'
UNION
SELECT name FROM skating WHERE country='DE';

-- INTERSECT
SELECT name FROM cycling WHERE country='DE'
INTERSECT
SELECT name FROM skating WHERE country='DE';

-- EXCEPT / MINUS
SELECT name FROM cycling WHERE country='DE'
EXCEPT
SELECT name FROM skating WHERE country='DE';
```

16 Database-Specific Commands

16.1 SQLite Commands

```
.tables             -- list tables
.open filename      -- open DB file
.save filename      -- save DB file
.schema table       -- show table schema
.mode column        -- column mode output
.headers on         -- show headers
.quit               -- exit
```

16.2 PostgreSQL Commands

```
\l                  -- list databases
\c dbname           -- connect to database
\dt                 -- list tables
\d table            -- describe table
\du                 -- list roles
\timing             -- toggle timing
\i filename         -- run file
\q                  -- quit
```

17 GeeksforGeeks Extended SQL Commands

17.1 Create a Database in SQL

```
CREATE DATABASE company;
USE company;
ALTER DATABASE database_name;
DROP DATABASE company;
```

Source: GeeksforGeeks :contentReference[oaicite:0]index=0

17.2 Creating Data in SQL

```
CREATE TABLE employees (
  employee_id INT PRIMARY KEY,
  first_name VARCHAR(50),
  last_name VARCHAR(50),
  department VARCHAR(50),
  salary DECIMAL(10,2)
);
INSERT INTO employees (employee_id, first_name, last_name, department, salary)
VALUES
  (1, 'John', 'Doe', 'HR', 50000.00),
  (2, 'Jane', 'Smith', 'IT', 60000.00);
ALTER TABLE employees ADD COLUMN new_column INT;
DROP TABLE employees;
```

Source: GeeksforGeeks :contentReference[oaicite:1]index=1

17.3 Reading / Querying Data in SQL

```
SELECT * FROM employees;
SELECT DISTINCT department FROM employees;
SELECT * FROM employees WHERE salary > 55000.00;
SELECT * FROM employees LIMIT 3;
SELECT * FROM employees LIMIT 10000 OFFSET 2;
SELECT * FROM employees FETCH FIRST 3 ROWS ONLY;
SELECT first_name, last_name,
  CASE
    WHEN salary > 55000 THEN 'High'
    WHEN salary > 50000 THEN 'Medium'
    ELSE 'Low'
  END AS salary_category
FROM employees;
```

Source: GeeksforGeeks :contentReference[oaicite:2]index=2

17.4 Updating / Deleting Data

```
UPDATE employees SET salary = 55000.00 WHERE employee_id = 1;
DELETE FROM employees WHERE employee_id = 5;
```

Source: GeeksforGeeks :contentReference[oaicite:3]index=3

17.5 Filtering Data

```
SELECT * FROM employees WHERE department = 'IT';
SELECT * FROM employees WHERE first_name LIKE 'J%';
SELECT * FROM employees WHERE department IN ('HR','Finance');
SELECT * FROM employees WHERE salary BETWEEN 50000 AND 60000;
SELECT * FROM employees WHERE department IS NULL;
SELECT * FROM employees ORDER BY salary DESC;
```

Source: GeeksforGeeks :contentReference[oaicite:4]index=4

17.6 SQL Operators

```
SELECT * FROM employees WHERE department = 'IT' AND salary > 60000;
SELECT * FROM employees WHERE department = 'HR' OR department = 'Finance';
SELECT * FROM employees WHERE NOT department = 'IT';
```

Source: GeeksforGeeks :contentReference[oaicite:5]index=5

17.7 Utility Functions

String functions:

```
SELECT SUBSTR(first_name,1,3) AS short_name FROM employees;
SELECT INSERT(CONCAT(first_name,' ',last_name),6,0,'Amazing ') AS modified_name
FROM employees;
```

Source: GeeksforGeeks :contentReference[oaicite:6]index=6

Date/Time and Math functions:

```
SELECT CURRENT_DATE AS current_date;
SELECT SQRT(25) AS square_root;
```

Source: GeeksforGeeks :contentReference[oaicite:7]index=7

17.8 Subqueries

```
SELECT first_name, last_name
FROM employees
WHERE salary = (SELECT MAX(salary) FROM employees);

SELECT first_name, last_name
FROM employees
WHERE department_id IN (
  SELECT department_id FROM departments WHERE department_name = 'IT'
);
```

Source: GeeksforGeeks :contentReference[oaicite:8]index=8

17.9 Views, Indexes, and Transactions

```
CREATE VIEW high_paid_employees AS
  SELECT * FROM employees WHERE salary > 60000;
DROP VIEW IF EXISTS high_paid_employees;

CREATE INDEX idx_department ON employees(department);
DROP INDEX IF EXISTS idx_department;

BEGIN TRANSACTION;
COMMIT;
ROLLBACK;
```

Source: GeeksforGeeks :contentReference[oaicite:9]index=9

17.10 Advanced Commands

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

CREATE TRIGGER before_employee_insert
BEFORE INSERT ON employees
FOR EACH ROW
BEGIN
  SET NEW.creation_date = NOW();
END;

CREATE FUNCTION calculate_bonus(salary DECIMAL) RETURNS DECIMAL
BEGIN
  RETURN salary * 0.1;
END;

WITH high_paid_employees AS (
  SELECT * FROM employees WHERE salary > 60000
)
SELECT * FROM high_paid_employees;
```

Source: GeeksforGeeks :contentReference[oaicite:10]index=10