SQL Cheat Sheet — From Basics to Advanced

1-SELECT & FILTERS

```
-- Basic selection
SELECT col1, col2
FROM table
WHERE condition
ORDER BY col1 ASC, col2 DESC
OFFSET 0 FETCH FIRST 10 ROWS ONLY;   -- ANSI
-- MySQL: LIMIT 10 OFFSET 0
-- SQL Server: OFFSET 0 ROWS FETCH NEXT 10 ROWS ONLY
```

```
-- Common filters
WHERE col IS NULL
WHERE col IN ('A','B','C')
WHERE col BETWEEN 10 AND 20
WHERE col LIKE 'ABC%'       -- starts with ABC
WHERE col ILIKE '%xyz%'     -- PostgreSQL: case-insensitive
WHERE (colA, colB) IN ((1,2),(3,4))  -- tuple filtering (PostgreSQL)
```

4-CONDITIONAL EXPRESSIONS

```
SELECT
  CASE
    WHEN score >= 90 THEN 'A'
    WHEN score >= 80 THEN 'B'
    WHEN score IS NULL THEN 'No score'
    ELSE 'C'
  END AS grade
FROM exams;
```

2-JOINS (Essential types)

```
-- INNER JOIN
SELECT t1.*, t2.col
FROM t1
JOIN t2 ON t2.id = t1.t2_id;

-- LEFT JOIN (keep all from left)
SELECT t1.*, t2.col
FROM t1
LEFT JOIN t2 ON t2.id = t1.t2_id;

-- RIGHT / FULL JOIN
SELECT *
FROM t1
FULL OUTER JOIN t2 ON t1.id = t2.id;  -- PostgreSQL / SQL Server only

-- SEMI JOIN (existence)
SELECT t1.*
FROM t1
WHERE EXISTS (SELECT 1 FROM t2 WHERE t2.id = t1.t2_id);

-- ANTI JOIN (non-existence)
SELECT t1.*
FROM t1
WHERE NOT EXISTS (SELECT 1 FROM t2 WHERE t2.id = t1.t2_id);
```

3-AGGREGATIONS & HAVING

```
SELECT dept,
       COUNT(*)      AS n_employees,
       SUM(salary)   AS total,
       AVG(salary)   AS avg_salary,
       MIN(salary)   AS min_salary,
       MAX(salary)   AS max_salary
FROM employees
GROUP BY dept
HAVING AVG(salary) > 2000
ORDER BY total DESC;
```

5-WINDOW FUNCTIONS

```
-- Ranking, totals, moving averages
SELECT
  emp_id,
  dept,
  salary,
  RANK()      OVER (PARTITION BY dept ORDER BY salary DESC) AS rk,
  ROW_NUMBER() OVER (PARTITION BY dept ORDER BY salary DESC) AS rn,
  DENSE_RANK() OVER (PARTITION BY dept ORDER BY salary DESC) AS drk,
  SUM(salary)  OVER (PARTITION BY dept)                AS dept_total,
  AVG(salary)  OVER (ORDER BY hire_date
                ROWS BETWEEN 2 PRECEDING AND CURRENT ROW) AS
moving_avg_3
FROM employees;
```

```
-- Previous / next row difference
SELECT
  emp_id,
  sale_date,
  value,
  value - LAG(value) OVER (PARTITION BY emp_id ORDER BY sale_date) AS
diff_prev,
  LEAD(value) OVER (PARTITION BY emp_id ORDER BY sale_date)       AS
next_value
FROM sales;
```

SQL Cheat Sheet — From Basics to Advanced

6-CTEs & SUBQUERIES

```
-- Common Table Expressions (CTEs)
WITH monthly_sales AS (
  SELECT client_id, DATE_TRUNC('month', date) AS month, SUM(value) AS
total_month
  FROM sales
  GROUP BY 1,2
),
top_clients AS (
  SELECT month, client_id, total_month,
      ROW_NUMBER() OVER (PARTITION BY month ORDER BY total_month
DESC) AS rn
  FROM monthly_sales
)
SELECT *
FROM top_clients
WHERE rn <= 10
ORDER BY month, total_month DESC;
```

```
-- Correlated subquery (latest purchase per client)
SELECT c.client_id,
     (SELECT MAX(s.date) FROM sales s WHERE s.client_id =
c.client_id) AS last_purchase
FROM clients c;
```

9-NULL HANDLING & NUMBERS

```
SELECT
  COALESCE(value, 0) AS safe_value,
  NULLIF(field, '')  AS null_if_empty,
  ROUND(price, 2)    AS price_2dec,
  CAST(qty AS NUMERIC(18,2)) AS qty_num,
  CASE WHEN denom = 0 OR denom IS NULL THEN NULL ELSE
num*1.0/denom END AS ratio
FROM t;
```

7-STRING FUNCTIONS

```
-- Concatenation
SELECT col1 || ' - ' || col2 AS text FROM t;        -- PostgreSQL
-- MySQL: CONCAT(col1, ' - ', col2)
-- SQL Server: col1 + ' - ' + col2

-- Substrings
SELECT SUBSTRING(name FROM 1 FOR 5);    -- PostgreSQL
-- MySQL / SQL Server: SUBSTRING(name,1,5)

-- Length & case
SELECT LENGTH(name), UPPER(name), LOWER(name),
TRIM(name);

-- Replace & Regex
SELECT REPLACE(name,'-',' ') AS clean_name;
SELECT REGEXP_REPLACE(name, '\s+', ' ', 'g') AS normalized;  --
PostgreSQL

-- Split
SELECT SPLIT_PART(email,'@',1) AS username FROM users;
```

10-SET OPERATIONS

```
SELECT col FROM t1
UNION
SELECT col FROM t2;       -- removes duplicates

SELECT col FROM t1
UNION ALL
SELECT col FROM t2;       -- keeps duplicates

SELECT col FROM t1
INTERSECT
SELECT col FROM t2;

SELECT col FROM t1
EXCEPT
SELECT col FROM t2;       -- SQL Server / PostgreSQL
```

8-DATE & TIME

```
-- Current date/time
SELECT CURRENT_DATE, CURRENT_TIMESTAMP;

-- Truncate to month
SELECT DATE_TRUNC('month', date) AS month;  -- PostgreSQL
-- SQL Server: DATEADD(month, DATEDIFF(month, 0, date), 0)
-- MySQL: DATE_FORMAT(date, '%Y-%m-01')

-- Differences
SELECT AGE(CURRENT_DATE, birth_date);   -- PostgreSQL
SELECT DATEDIFF(day, start_date, end_date);  -- SQL Server
SELECT DATEDIFF(end_date, start_date);      -- MySQL

-- Add intervals
SELECT date + INTERVAL '7 days';      -- PostgreSQL
SELECT DATEADD(day,7,date);         -- SQL Server
SELECT DATE_ADD(date, INTERVAL 7 DAY); -- MySQL
```

11-PIVOT / UNPIVOT

```
-- Manual pivot (portable)
SELECT
  id,
  SUM(CASE WHEN category = 'A' THEN value ELSE 0 END) AS cat_A,
  SUM(CASE WHEN category = 'B' THEN value ELSE 0 END) AS cat_B
FROM facts
GROUP BY id;
```

```
-- Native pivot (SQL Server)
SELECT *
FROM (
  SELECT id, category, value FROM facts
) src
PIVOT (
  SUM(value) FOR category IN ([A],[B],[C])
) p;
```

SQL Cheat Sheet — From Basics to Advanced

12-UPSERT / MERGE

```
-- SQL Server / Oracle / PostgreSQL 15+
MERGE INTO target t
USING (SELECT id, col FROM source) s
ON (t.id = s.id)
WHEN MATCHED THEN UPDATE SET col = s.col
WHEN NOT MATCHED THEN INSERT (id, col) VALUES (s.id, s.col);
```

```
-- PostgreSQL classic
INSERT INTO target (id, col)
VALUES (:id, :col)
ON CONFLICT (id) DO UPDATE
SET col = EXCLUDED.col;
```

```
-- MySQL
INSERT INTO target (id, col)
VALUES (?, ?)
ON DUPLICATE KEY UPDATE col = VALUES(col);
```

15-LATEST ROW PER GROUP

```
WITH ranked AS (
  SELECT *,
       ROW_NUMBER() OVER (PARTITION BY client_id ORDER BY date
DESC) AS rn
  FROM sales
)
SELECT *
FROM ranked
WHERE rn = 1;
```

13-JSON HANDLING

```
-- PostgreSQL
SELECT
  json_col -> 'a'      AS obj_a,
  json_col ->> 'a'     AS obj_a_text,
  (json_col -> 'arr') -> 0 AS first_elem,
  jsonb_pretty(json_col)
FROM t;

SELECT * FROM t
WHERE json_col ->> 'status' = 'active';
```

```
-- MySQL
SELECT JSON_EXTRACT(json_col, '$.a') AS a FROM t;

-- SQL Server
SELECT JSON_VALUE(json_col,'$.a') AS a FROM t;
```

16-DEDUPLICATION

```
WITH ranked AS (
  SELECT *,
       ROW_NUMBER() OVER (PARTITION BY key ORDER BY
updated_at DESC, id DESC) AS rn
  FROM t
)
DELETE FROM t
WHERE id IN (SELECT id FROM ranked WHERE rn > 1);
```

14-ARRAYS & REGEX (PostgreSQL)

```
SELECT unnest(tags) AS tag FROM articles;
SELECT * FROM articles WHERE 'sql' = ANY(tags);

-- Regex filter
SELECT * FROM t WHERE col ~ '^\d{4}-\d{2}-\d{2}$';
```

17-PERFORMANCE TIPS

- ✅ Filter early (reduce data before joins)
- ✅ Index JOIN / WHERE / ORDER BY columns
- ✅ Avoid functions on indexed columns
- ✅ Use EXPLAIN to inspect query plans
- ✅ Batch inserts instead of row-by-row
- ✅ Partition large tables (e.g., by date)
- ✅ Avoid SELECT * in production;

18-TRANSACTIONS

```
BEGIN;

UPDATE accounts SET balance = balance - 100 WHERE id = 1;
UPDATE accounts SET balance = balance + 100 WHERE id = 2;

COMMIT;  -- or ROLLBACK;
```

19-SECURITY & BEST PRACTICES

- 🔹 Use **parameterized queries** (prevent SQL injection)
- 🔹 Grant **least privilege** possible
- 🔹 Expose data through **views/materialized views**
- 🔹 Log or capture changes via **CDC / triggers**

SQL Cheat Sheet — From Basics to Advanced

20-DDL ESSENTIALS

```
CREATE TABLE sales (
  id         BIGSERIAL PRIMARY KEY,
  client_id    BIGINT NOT NULL,
  date       TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,
  value       NUMERIC(18,2) NOT NULL,
  status      VARCHAR(20),
  UNIQUE (client_id, date)
);

CREATE INDEX ix_sales_client_date ON sales (client_id, date);

ALTER TABLE sales ADD COLUMN channel VARCHAR(20);
ALTER TABLE sales ALTER COLUMN value TYPE NUMERIC(19,4);
```

23-OPTIMIZING JOINS

```
WITH filtered_t2 AS (
  SELECT id, colX FROM t2 WHERE status = 'active'
)
SELECT t1.*, filtered_t2.colX
FROM t1
JOIN filtered_t2 ON filtered_t2.id = t1.t2_id;
```

24-PARAMETERIZED FILTERS (ETL/BI)

```
-- Date range
SELECT *
FROM facts
WHERE date >= :start_date
  AND date < :end_date;

-- Rolling 12-month window
SELECT DATE_TRUNC('month', date) AS month,
      SUM(value) AS total_month
FROM facts
WHERE date >= DATE_TRUNC('month', CURRENT_DATE) - INTERVAL '11
months'
GROUP BY 1
ORDER BY 1;
```

21-COMMON PATTERNS

```
-- Top N per group
WITH ranked AS (
  SELECT *,
       ROW_NUMBER() OVER (PARTITION BY group_id ORDER BY metric DESC)
AS rn
  FROM t
)
SELECT * FROM ranked WHERE rn <= 3;
```

```
-- Gaps & Islands
WITH ordered AS (
  SELECT *,
       ROW_NUMBER() OVER (PARTITION BY key ORDER BY date) AS rn
  FROM t
),
grp AS (
  SELECT *,
       (EXTRACT(EPOCH FROM date)::BIGINT - rn) AS group_key
  FROM ordered
)
SELECT key, group_key, MIN(date) AS start_date, MAX(date) AS end_date
FROM grp
GROUP BY key, group_key;
```

```
-- Slowly Changing Dimension (Type 2)
SELECT *
FROM dim_customer
WHERE natural_id = :id
  AND :ref_date >= valid_from
  AND (valid_to IS NULL OR :ref_date < valid_to);
```

22-DATA QUALITY CHECKS

```
-- Duplicates
SELECT key, COUNT(*) c
FROM t
GROUP BY key
HAVING COUNT(*) > 1;

-- Missing required fields
SELECT *
FROM t
WHERE col IS NULL OR TRIM(col) = '';

-- Outliers (PostgreSQL)
SELECT *
FROM t
WHERE value NOT BETWEEN
  PERCENTILE_CONT(0.01) WITHIN GROUP (ORDER BY value)
  AND
  PERCENTILE_CONT(0.99) WITHIN GROUP (ORDER BY value);
```

💡 Dialect Notes

Feature	PostgreSQL	SQL Server	MySQL 8+
ILIKE	✅	❌	❌
DATE_TRUNC	✅	⚙️	⚙️
JSON	✅ (jsonb)	✅	✅
FULL JOIN	✅	✅	❌
CTE / WINDOW	✅	✅	✅
FILTER in aggregates	✅	❌	❌