

SQL

CHEATSHEET

Standard SQL + PostgreSQL · Select · Join · Aggregate · CTE · Window · DDL · JSON · and more

151 commands · 15 categories

@contentintech

contentintech.com

► SELECT & FILTERING

\$ SELECT col1, col2 FROM tbl
Retrieve specific columns from a table

\$ SELECT * FROM tbl
Retrieve all columns from a table

\$ SELECT DISTINCT col FROM tbl
Return unique values only

\$ WHERE col = 'value'
Filter rows by exact match

\$ WHERE col > 10 AND col < 50
Combine conditions with AND

\$ WHERE col = 1 OR col = 2
Match either condition with OR

\$ WHERE NOT col = 'x'
Negate a condition

\$ WHERE col IN (1, 2, 3)
Match any value in a list

\$ WHERE col BETWEEN 10 AND 20
Match values in a range (inclusive)

\$ WHERE col LIKE 'A%'
Pattern match – % is wildcard

\$ WHERE col IS NULL
Filter rows where column has no value

\$ WHERE col IS NOT NULL
Filter rows where column has a value

\$ LIMIT 10
Return at most 10 rows

\$ LIMIT 10 OFFSET 20
Skip 20 rows then return 10 (paging)

\$ WHERE EXISTS (subquery)
True if subquery returns any rows

\$ WHERE col > ANY (subquery)
True if col > at least one subquery val

\$ WHERE col > ALL (subquery)
True if col > every subquery value

► SORTING & GROUPING

\$ ORDER BY col ASC
Sort ascending (default)

\$ ORDER BY col DESC
Sort descending

\$ ORDER BY col1, col2 DESC
Multi-column sort

\$ ORDER BY col NULLS FIRST
NULLs appear before other values

\$ ORDER BY col NULLS LAST
NULLs appear after other values

\$ GROUP BY col
Aggregate rows sharing the same value

\$ GROUP BY col1, col2
Group by multiple columns

\$ HAVING COUNT(*) > 5
Filter groups after aggregation

\$ DISTINCT ON (col) ...
PostgreSQL: keep first row per group

► JOINS

\$ INNER JOIN t2 ON t1.id = t2.fk
Rows matching in both tables

\$ LEFT JOIN t2 ON t1.id = t2.fk
All t1 rows; NULLs for t2 non-matches

\$ RIGHT JOIN t2 ON t1.id = t2.fk
All t2 rows; NULLs for t1 non-matches

\$ FULL OUTER JOIN t2 ON ...
All rows from both tables

\$ CROSS JOIN t2
Cartesian product of both tables

\$ t1 AS a JOIN t1 AS b ON a.mgr=b.id
Self join using aliases

\$ NATURAL JOIN t2
Join on identically-named columns

\$ JOIN t2 USING (col)
Join on shared column name

\$ JOIN t2 ON t1.id = t2.fk
Explicit join condition with ON

► AGGREGATIONS

\$ COUNT(*)
Count all rows in a group

\$ COUNT(col)
Count non-NULL values in col

\$ COUNT(DISTINCT col)
Count unique non-NULL values

\$ SUM(col)
Sum of all values in col

\$ AVG(col)
Arithmetic mean of col

\$ MIN(col)
Smallest value in col

\$ MAX(col)
Largest value in col

\$ GROUP BY ROLLUP(a, b)
Subtotals + grand total (PostgreSQL)

\$ GROUPING SETS ((a),(b),())
Multiple grouping combos in one query

\$ GROUP BY CUBE(a, b)
All possible grouping combinations

► SUBQUERIES

\$ SELECT (SELECT MAX(x) FROM t2)
Scalar subquery – returns single value

\$ WHERE col = (SELECT MAX(x) FROM t2)
Filter using scalar subquery result

\$ WHERE col IN (SELECT id FROM t2)
Filter using a list from subquery

\$ WHERE EXISTS (SELECT 1 FROM t2 ...)
True when subquery returns rows

\$ FROM (SELECT ... FROM t2) AS sub
Derived table / inline view

\$ WHERE t1.col = (SELECT ... WHERE t2.col = t1.col)
Correlated subquery

\$ LATERAL (SELECT ... LIMIT 3) AS s
Lateral join – ref outer cols inside

► CTEs & WINDOW FUNCTIONS

\$ WITH cte AS (SELECT ...) SELECT ...
Named temporary result set

\$ WITH RECURSIVE cte AS (...)
Recursive CTE for hierarchical data

\$ ROW_NUMBER() OVER (...)
Sequential row number in partition

\$ RANK() OVER (...)
Rank with gaps on ties

\$ DENSE_RANK() OVER (...)
Rank without gaps on ties

\$ LEAD(col, 1) OVER (...)
Value of col in the next row

\$ LAG(col, 1) OVER (...)
Value of col in the previous row

\$ FIRST_VALUE(col) OVER (...)
First value in the window frame

\$ LAST_VALUE(col) OVER (...)
Last value in the window frame

\$ PARTITION BY col
Divide rows into partitions

\$ OVER (PARTITION BY a ORDER BY b)
Full window spec — partition + order

► INSERT UPDATE DELETE

\$ INSERT INTO t (c1,c2) VALUES (v1,v2)
Insert a single row

\$ INSERT INTO t SELECT ... FROM src
Insert rows from a query result

\$ UPDATE t SET col = val WHERE ...
Update rows matching condition

\$ UPDATE t SET col = s.val FROM src s WHERE t.id = s.id
Update from another table

\$ DELETE FROM t WHERE ...
Delete rows matching condition

\$ DELETE FROM t USING t2 WHERE ...
Delete using join (PostgreSQL)

\$ INSERT ... ON CONFLICT DO NOTHING
Upsert — ignore duplicate key

\$ INSERT ... ON CONFLICT (col) DO UPDATE SET col = EXCLU...
Upsert — update on conflict

\$ RETURNING id, col
Return values from modified rows

\$ TRUNCATE TABLE t
Delete all rows fast (no rollback scan)

► TABLE DDL

\$ CREATE TABLE t (id SERIAL PRIMARY KEY, name TEXT NOT NULL)
Create table

\$ ALTER TABLE t ADD COLUMN col TYPE
Add a new column

\$ ALTER TABLE t DROP COLUMN col
Remove a column

\$ ALTER TABLE t ALTER COLUMN col TYPE new_type
Change column data type

\$ ALTER TABLE t ADD CONSTRAINT ...
Add a named constraint

\$ DROP TABLE t
Permanently delete a table

\$ DROP TABLE IF EXISTS t
Drop table only if it exists

\$ ALTER TABLE t RENAME TO new_name
Rename a table

\$ CREATE TEMP TABLE t AS SELECT ...
Temp table from query result

► INDEXES

\$ CREATE INDEX idx ON t (col)
Create a B-tree index

\$ CREATE UNIQUE INDEX idx ON t (col)
Enforce uniqueness via index

\$ CREATE INDEX idx ON t (lower(col))
Expression / functional index

\$ CREATE INDEX idx ON t (a, b)
Composite multi-column index

\$ CREATE INDEX idx ON t (col) WHERE active
Partial index — subset of rows

\$ DROP INDEX idx
Remove an index

\$ REINDEX TABLE t
Rebuild all indexes on a table

\$ EXPLAIN SELECT ...
Show query execution plan

\$ EXPLAIN ANALYZE SELECT ...
Run query and show actual timings

► TRANSACTIONS

\$ BEGIN
Start a transaction block

\$ COMMIT
Persist all changes in the transaction

\$ ROLLBACK
Undo all changes in the transaction

\$ SAVEPOINT sp1
Create a named rollback point

\$ RELEASE SAVEPOINT sp1
Remove a savepoint

\$ ROLLBACK TO SAVEPOINT sp1
Roll back to a savepoint only

\$ SET TRANSACTION ISOLATION LEVEL SERIALIZABLE
Set isolation level

\$ SELECT ... FOR UPDATE
Lock selected rows for update

\$ SELECT ... FOR SHARE
Lock rows allowing concurrent reads

► CONSTRAINTS

\$ col TYPE PRIMARY KEY
Unique not-null row identifier

\$ FOREIGN KEY (col) REFERENCES t2(id)
Enforce referential integrity

\$ col TYPE UNIQUE
All values in col must be unique

\$ col TYPE NOT NULL
Column must have a value

\$ CHECK (col > 0)
Row-level boolean constraint

\$ col TYPE DEFAULT 'value'
Default value if none supplied

\$ CONSTRAINT c DEFERRABLE INITIALLY DEFERRED
Defer check to commit time

\$ ON DELETE CASCADE
Delete child rows when parent deleted

\$ ON DELETE SET NULL
Set FK to NULL when parent deleted

► VIEWS & MATERIALIZED

\$ CREATE VIEW v AS SELECT ...
Create a virtual table from a query

\$ CREATE OR REPLACE VIEW v AS ...
Update view definition in place

\$ DROP VIEW v
Remove a view

```
$ DROP VIEW IF EXISTS v
Remove view only if it exists

$ CREATE MATERIALIZED VIEW mv AS ...
Cache query result on disk

$ REFRESH MATERIALIZED VIEW mv
Update cached data from source query

$ REFRESH MATERIALIZED VIEW CONCURRENTLY mv
Refresh without locking reads
```

► STRING FUNCTIONS

```
$ CONCAT(a, ' ', b)
Join strings together

$ LENGTH(col)
Number of characters in string

$ UPPER(col) / LOWER(col)
Convert case

$ TRIM(col)
Remove leading and trailing spaces

$ LTRIM(col) / RTRIM(col)
Remove spaces from one side only

$ SUBSTRING(col FROM 1 FOR 5)
Extract part of a string

$ REPLACE(col, 'old', 'new')
Replace substring occurrences

$ REGEXP_REPLACE(col, 'pat', 'new')
Replace using a regex pattern

$ SPLIT_PART(col, ',', 2)
Split string and return Nth part

$ POSITION('x' IN col)
Find character position (1-based)

$ COALESCE(col, 'default')
Return first non-NULL argument

$ NULLIF(col, 0)
Return NULL if col equals value

$ CAST(col AS INTEGER)
Convert value to another type
```

► DATE FUNCTIONS

```
$ NOW()
Current date and time with timezone

$ CURRENT_DATE
Today's date (no time)

$ CURRENT_TIMESTAMP
Current timestamp with timezone

$ DATE_TRUNC('month', ts)
Truncate timestamp to precision

$ DATE_PART('year', ts)
Extract numeric part from timestamp

$ EXTRACT(EPOCH FROM ts)
Seconds since Unix epoch

$ AGE(ts1, ts2)
Difference between two timestamps

$ ts + INTERVAL '7 days'
Add an interval to a timestamp

$ TO_DATE('2024-01-01', 'YYYY-MM-DD')
Parse string to date

$ TO_TIMESTAMP('...', 'YYYY-...')
Parse string to timestamp

$ ts AT TIME ZONE 'UTC'
Convert timestamp to timezone
```

► JSON (PostgreSQL)

```
$ JSON_BUILD_OBJECT('k', val)
Build a JSON object from key-value

$ JSON_AGG(row)
Aggregate rows into a JSON array

$ col -> 'key'
Get JSON field as JSON

$ col ->> 'key'
Get JSON field as text

$ col #> '{a,b}'
Get nested JSON by path as JSON

$ jsonb_set(col, '{key}', '"v"')
Update a key in a JSONB column

$ jsonb_array_elements(col)
Expand JSONB array to rows

$ col @> '{"k":"v"}'::jsonb
JSONB contains value check

$ col <@ '{"k":"v"}'::jsonb
JSONB is contained by check

$ col ? 'key'
JSONB has key check

$ jsonb_each(col)
Expand JSONB to (key, value) rows
```

STAY CONNECTED

Follow us for daily tech content, tutorials, and resources

INSTAGRAM

@contentintech

[instagram.com/contentintech](https://www.instagram.com/contentintech)

YOUTUBE

@contenintech

[youtube.com/@contenintech](https://www.youtube.com/@contenintech)

WHATSAPP

Contentintech Channel

[whatsapp.com/channel/0029VbCZJQU05MUijd2ofT1D](https://www.whatsapp.com/channel/0029VbCZJQU05MUijd2ofT1D)

Thanks for learning with us – share this cheatsheet with your team!

contentintech.com