📜 SQL Window Functions Cheat Sheet – The Complete Quick Reference

This **SQL window functions cheat sheet** is the single page you can keep open while you write analytic SQL. Every function, every frame clause, every common pattern — with copy-ready syntax and a link to a practice problem you can solve right now.

Whether you're prepping for a **SQL interview**, debugging a dashboard query, or just trying to remember the difference between `RANK()` and `DENSE_RANK()` at 11pm, this guide has you covered.

🚀 Why a Window Functions Cheat Sheet

"Xorthax's quartermaster doesn't memorize every starship manual — they keep a reference card on the bridge. You should too."

Window functions are powerful but unforgiving. One missing `ORDER BY`, one wrong frame clause, and your "running total" returns the same number on every row. This cheat sheet gives you the exact patterns that work, organized so you can find them fast.

- **Scan it** when you remember the pattern but forgot the syntax.
- **Print it** for interview prep — every function comes with a "when to use it" cue.
- **Click through** to the matching practice problem to lock in the muscle memory.

🧬 Window Function Anatomy

Every window function follows the same shape. Memorize this skeleton and you can always fill in the blanks:

The universal window function syntax — every function plugs into this pattern.

```
FUNCTION(expr) OVER (
  [PARTITION BY partition_col, ...]   -- group rows into windows
  [ORDER BY sort_col [ASC|DESC]]      -- order within each window
  [ROWS|RANGE|GROUPS BETWEEN          -- which rows count for each calc
       frame_start AND frame_end]
)
```

Clause	What it does	Required?
`OVER ( ... )`	Marks the function as a window function.	✅ Always
`PARTITION BY`	Splits rows into independent groups.	Optional — defaults to one window over all rows
`ORDER BY`	Defines row order inside each partition.	Required for ranking, value, & distribution functions
`ROWS / RANGE / GROUPS`	Selects which rows feed each calculation.	Optional — has a default that often surprises you

👉 Want the deep dive on each clause? See How Window Functions Work.

🧮 The Four Function Categories

Every window function belongs to one of four families. Knowing the category tells you what defaults to expect.

Category	Functions	Needs ORDER BY?	Accepts frame?
Aggregate	`SUM`, `AVG`, `COUNT`, `MIN`, `MAX`	Optional	✅ Yes
Ranking	`ROW_NUMBER`, `RANK`, `DENSE_RANK`, `NTILE`	✅ Required	❌ No
Value / Navigation	`LAG`, `LEAD`, `FIRST_VALUE`, `LAST_VALUE`, `NTH_VALUE`	✅ Required	`FIRST/LAST/NTH` only
Distribution	`PERCENT_RANK`, `CUME_DIST`	✅ Required	❌ No

➕ Aggregate Window Functions

The same aggregates you already know (`SUM`, `AVG`, etc.) — but applied per-row, without collapsing the result set.

Function	Use it when…
`SUM(expr) OVER (...)`	Running totals, cumulative revenue, partition totals on every row.
`AVG(expr) OVER (...)`	Moving averages, rolling means, partition-level averages.
`COUNT(*) OVER (...)`	Running counts, partition sizes, "how many in my group so far".
`MIN(expr) OVER (...)`	Lowest seen so far, partition minimum repeated on every row.
`MAX(expr) OVER (...)`	Highest seen so far, partition max repeated on every row.

Running total and 7-day moving average — the two patterns you'll write a hundred times.

```
SELECT
  order_date,
  amount,
  SUM(amount) OVER (
    ORDER BY order_date
    ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW
  ) AS running_total,
  AVG(amount) OVER (
    ORDER BY order_date
    ROWS BETWEEN 6 PRECEDING AND CURRENT ROW
  ) AS moving_avg_7d
FROM Orders;
```

👉 Full guide: Aggregate Window Functions.

🥇 Ranking Functions

Assign a position to each row inside its partition. The differences only matter when there are ties.

Function	How it handles ties	Output for values 700, 700, 600
`ROW_NUMBER()`	Always unique — ties broken arbitrarily	1, 2, 3
`RANK()`	Same rank for ties, then skip	1, 1, 3
`DENSE_RANK()`	Same rank for ties, no skip	1, 1, 2
`NTILE(n)`	Splits rows into *n* roughly equal buckets	e.g. `NTILE(4)` → 1–4 quartile bucket

All four ranking functions side by side, partitioned by Region.

```
SELECT
  CustomerID,
  Region,
  OrderAmount,
  ROW_NUMBER() OVER (PARTITION BY Region ORDER BY OrderAmount DESC) AS rn,
  RANK()       OVER (PARTITION BY Region ORDER BY OrderAmount DESC) AS rnk,
  DENSE_RANK() OVER (PARTITION BY Region ORDER BY OrderAmount DESC) AS drnk,
  NTILE(4)     OVER (PARTITION BY Region ORDER BY OrderAmount DESC) AS quartile
FROM Orders;
```

👉 Full guide: Ranking Functions.

🔁 Value & Navigation Functions

Reach into other rows of the partition without writing a self-join.

Function	What it returns	Common use
`LAG(expr, offset, default)`	Value from *offset* rows before the current row	Month-over-month change, time between events
`LEAD(expr, offset, default)`	Value from *offset* rows after the current row	"Days until next order", forecasting checks
`FIRST_VALUE(expr)`	First row in the window frame	Onboarding price, customer's first order date
`LAST_VALUE(expr)`	Last row in the window frame ⚠️	Latest snapshot — *requires* an explicit unbounded frame
`NTH_VALUE(expr, n)`	The *n*-th row in the window frame	"Second-highest", "third event", etc.

Month-over-month revenue change with LAG().

```
SELECT
  product_id,
  month,
  revenue,
  LAG(revenue) OVER (PARTITION BY product_id ORDER BY month) AS prev_month,
  revenue - LAG(revenue) OVER (PARTITION BY product_id ORDER BY month) AS mom_change
FROM monthly_revenue;
```

⚠️ **LAST_VALUE gotcha:** with the default frame (`RANGE BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW`) it returns the current row, not the partition's last row. Always specify `ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING`.

👉 Full guides: LEAD / LAG · FIRST / LAST / NTH.

📊 Distribution Functions

Express where each row sits relative to its peers, on a 0–1 scale.

Function	Formula	Use it when…
`PERCENT_RANK()`	`(rank - 1) / (rows - 1)`	You want a percentile rank — first row is 0.0, last is 1.0.
`CUME_DIST()`	`rows ≤ current / total rows`	You want cumulative distribution — "what fraction of rows ≤ this value".

Top 10% of orders per region using PERCENT_RANK and a CTE.

```
WITH Ranked AS (
  SELECT *,
         PERCENT_RANK() OVER (PARTITION BY Region ORDER BY Amount DESC) AS pr
  FROM Orders
)
SELECT * FROM Ranked WHERE pr <= 0.10;
```

👉 Full guide: Percentile & Distribution.

📐 Frame Clauses: ROWS vs RANGE vs GROUPS

The frame clause decides which rows are visible to your function for each row's calculation. Most bugs in window queries trace back to this.

Frame	What "current row" means	Use when...
`ROWS`	A specific number of rows before/after the current row	You want exact row counts (e.g. last 7 rows for a 7-row moving avg)
`RANGE`	All rows whose `ORDER BY` value matches the bound	You want value-based bounds (e.g. all rows on the same date)
`GROUPS`	A specific number of *peer groups* (rows with equal sort keys)	Postgres 11+ — when you want N distinct sort values, ties counted together

Frame Boundaries Quick Reference

Boundary	Meaning
`UNBOUNDED PRECEDING`	From the first row of the partition
`n PRECEDING`	*n* rows before the current row
`CURRENT ROW`	The current row itself
`n FOLLOWING`	*n* rows after the current row
`UNBOUNDED FOLLOWING`	Through the last row of the partition

The Defaults That Bite

Situation	Default frame	What it produces
`SUM() OVER (ORDER BY ...)`	`RANGE BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW`	A running total ✅
`SUM() OVER (PARTITION BY ...)` (no ORDER BY)	`UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING`	Partition total on every row ✅
`LAST_VALUE() OVER (ORDER BY ...)`	`RANGE BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW`	Returns the *current* row, not the last ❌

🧰 Common Patterns (One-Liners You'll Reuse Forever)

The handful of patterns that cover ~80% of real analytics SQL.

Pattern	SQL	Practice
Running total	`SUM(amount) OVER (ORDER BY order_date)`	#2
Partition total on every row	`SUM(amount) OVER (PARTITION BY customer_id)`	#1
7-day moving average	`AVG(amount) OVER (ORDER BY date ROWS BETWEEN 6 PRECEDING AND CURRENT ROW)`	#5
Top-N per group	`ROW_NUMBER() OVER (PARTITION BY group ORDER BY metric DESC) AS rn` *(filter `rn <= N`)*	#3
Deduplication (keep latest)	`ROW_NUMBER() OVER (PARTITION BY key ORDER BY updated_at DESC)` *(filter `= 1`)*	#22
Month-over-month change	`amount - LAG(amount) OVER (PARTITION BY id ORDER BY month)`	#6
Days since last event	`order_date - LAG(order_date) OVER (PARTITION BY customer_id ORDER BY order_date)`	#4

Pattern	SQL	Practice
Percent of total	`amount / SUM(amount) OVER () * 100`	#7
Percent of group	`amount / SUM(amount) OVER (PARTITION BY group) * 100`	#7
First / last value per partition	`FIRST_VALUE(price) OVER (PARTITION BY product ORDER BY date)`	#21
Quartile bucketing	`NTILE(4) OVER (ORDER BY revenue DESC)`	#33
Gap between events	`date - LAG(date) OVER (PARTITION BY id ORDER BY date)`	Gap & Island

🧭 Quick Decision Tree: Which Function Do I Need?

Match the question you're trying to answer to the function family.

If you need to...	Reach for
Add an aggregate alongside detail rows	`SUM/AVG/COUNT OVER (PARTITION BY ...)`
Compute a running total	`SUM OVER (ORDER BY ...)`
Compute a moving / rolling window	`AVG OVER (ORDER BY ... ROWS BETWEEN n PRECEDING AND CURRENT ROW)`
Number every row uniquely	`ROW_NUMBER()`
Rank with ties + gaps (1, 1, 3)	`RANK()`
Rank with ties, no gaps (1, 1, 2)	`DENSE_RANK()`
Bucket into N equal-sized groups	`NTILE(n)`
Compare against the previous row	`LAG()`
Compare against the next row	`LEAD()`
Get the first or last value in a group	`FIRST_VALUE()` / `LAST_VALUE()`
Express a row as a percentile	`PERCENT_RANK()` / `CUME_DIST()`
Find streaks or gaps in dates	`ROW_NUMBER()` + `LAG()` — see Gap & Island

⚠️ Common Pitfalls (Quick Fixes)

- ❌ **Forgetting `ORDER BY` for ranking/value functions** — results are non-deterministic. *Fix:* always include `ORDER BY`, with a tie-breaker if values can repeat.
- ❌ **`LAST_VALUE()` returning the current row** — the default frame stops at `CURRENT ROW`. *Fix:* add `ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING`.
- ❌ **Using a window function in `WHERE`** — window functions run *after* `WHERE`. *Fix:* wrap the query in a CTE or subquery and filter outside.
- ❌ **Confusing `RANK` with `DENSE_RANK`** — `RANK` skips numbers after ties; `DENSE_RANK` doesn't. *Fix:* pick based on whether downstream consumers expect contiguous ranks.
- ❌ **`ROW_NUMBER()` changing across runs** — the engine breaks ties however it wants. *Fix:* add a stable secondary column to `ORDER BY` (e.g. an `id`).
- ❌ **Using `ROWS` when you needed `RANGE` (or vice versa)** — different semantics with duplicate `ORDER BY` values. *Fix:* ROWS = exact row count, RANGE = value-based.
- ❌ **Heavy `PARTITION BY` on huge tables** — sort spills kill performance. *Fix:* filter aggressively first; consider `LATERAL` joins for top-N at scale.

🌐 Database Differences (At a Glance)

Window function support is broadly consistent — but the syntactic sugar differs. The Reference section has dedicated pages for each function in each database.

Feature	PostgreSQL	MySQL 8+	SQL Server	Snowflake / BigQuery
All standard window functions	✅	✅	✅	✅
`QUALIFY` clause (filter window results)	❌	❌	❌	✅ (huge readability win)
`GROUPS` frame	✅ (11+)	❌	❌	Snowflake ✅ / BigQuery ❌
`EXCLUDE` frame option	✅ (11+)	❌	❌	❌
Named windows (`WINDOW w AS (...)`)	✅	✅	❌	✅

👉 For exact syntax in your dialect, browse the Reference section — every function has a dedicated page per database (Postgres, MySQL, SQL Server, MariaDB, SQLite, Snowflake, BigQuery, and more).

🧪 Practice Problems by Concept

Reference is good. Reps are better. Pick a row from the cheat sheet above, then come down here and solve the matching problem.

Aggregate Window Functions

- **1.** Galactic Customer Revenue Breakdown
- **2.** The Interstellar Sales Tracker 9000
- **5.** Customer Spending, Averaged and Analyzed

Ranking Functions

- **3.** Top 5 Spenders in the Galactic Bazaar
- **20.** Species Revenue Rankings
- **28.** Species Revenue Leaderboard

LEAD / LAG

- **4.** Two Orders Before, Two Orders After
- **6.** How Much More Did They Order?

FIRST / LAST / NTH VALUE

- **21.** The Peak Stock Protocol
- **41.** The First Order Ever Placed
- **43.** The Pioneer Product in Every Category

NTILE / Distribution

- **33.** Galactic Spending Tiers
- **35.** Customer Frequency Classes
- **40.** Species Revenue Classifications

🔗 Related Learning Pages

This cheat sheet is a quick reference. The deep dives live here:

- Beginner Series — start here if window functions are brand new.
- How Window Functions Work — the OVER, PARTITION BY, ORDER BY mental model.
- Aggregate Window Functions — running totals, moving averages, percent-of-total.
- Ranking Functions — ROW_NUMBER vs RANK vs DENSE_RANK in depth.
- LEAD / LAG — comparing across rows.
- FIRST / LAST / NTH — boundary values inside a window.
- Percentile & Distribution — PERCENT_RANK, CUME_DIST, NTILE.

- Gap and Island — streaks, gaps, and consecutive-date patterns.