

SQL Cheat Sheet for Analysts

Everyday queries, plus the credit-risk patterns used in loan reporting

Adekunle Okunade
Data Analyst
adekunleokunade.com

1. The shape of a query

You write it in this order, but the database reads it in a different order.

```
SELECT  columns, aggregates
FROM    table
JOIN    other_table ON ...
WHERE   row filter           -- before grouping
GROUP BY columns
HAVING  group filter        -- after grouping
ORDER BY column DESC
LIMIT 10;                  -- TOP 10 in SQL Server
```

Read order: FROM, JOIN, WHERE, GROUP BY, HAVING, SELECT, ORDER BY, LIMIT.

2. Filtering

```
WHERE amount > 50000
  AND product IN ('BNPL', 'Overdraft')
  AND start_date BETWEEN '2026-01-01' AND '2026-03-31'
  AND name LIKE 'A%'           -- starts with A
  AND closed_date IS NULL;     -- never "= NULL"
```

Mix AND with OR? Use brackets: (a OR b) AND c.

3. Aggregating

```
SELECT product,
  COUNT(*)           AS loans,
  COUNT(DISTINCT customer_id) AS customers,
  SUM(balance)       AS total_balance,
  AVG(balance)       AS avg_balance,
  MIN(start_date), MAX(start_date)
FROM loans
GROUP BY product
HAVING SUM(balance) > 1000000;
```

COUNT(col) skips NULLs. COUNT(*) counts rows.

4. Joins

Join	Keeps
INNER JOIN	Only rows that match in both tables
LEFT JOIN	All left rows, matches from right (else NULL)
RIGHT JOIN	All right rows, matches from left
FULL JOIN	All rows from both

```
SELECT l.loan_id, c.segment
FROM loans l
LEFT JOIN customers c ON c.customer_id = l.customer_id;
```

Find rows with no match: LEFT JOIN, then WHERE c.customer_id IS NULL.

Watch out: joining on a non-unique key multiplies rows and inflates SUMs. Check counts before and after.

5. CASE WHEN: buckets and flags

```
CASE
  WHEN days_past_due = 0 THEN '0 Current'
  WHEN days_past_due <= 30 THEN '1-30'
  WHEN days_past_due <= 60 THEN '31-60'
  WHEN days_past_due <= 90 THEN '61-90'
  ELSE '90+'
END AS bucket
```

Conditional total: SUM(CASE WHEN dpd > 30 THEN balance ELSE 0 END)

6. NULLs and cleaning

```
COALESCE(phone, 'unknown') -- first non-NULL
NULLIF(a, 0)               -- NULL if a = 0 (avoids
divide by zero)
CAST(amount AS DECIMAL(12,2))
TRIM(name), UPPER(name), LOWER(name)
SUBSTRING(code, 1, 3), LENGTH(code)
a || b                     -- join text (CONCAT in MySQL, SQL
Server)
```

7. Dates

Date functions differ most between databases. Pick yours:

	PostgreSQL	SQL Server	MySQL
Today	CURRENT_DATE	CAST(GETDATE() () AS date)	CURDATE()
Month start	DATE_TRUNC('month', d)	DATEFROMPART S(YEAR(d), MONTH(d), 1)	DATE_FORMAT(d, '%Y-%m- 01')
Difference	d2 - d1 (days)	DATEDIFF(day, d1, d2)	DATEDIFF(d2, d1)
Add days	d + 30	DATEADD(day, 30, d)	DATE_ADD(d, INTERVAL 30 DAY)

8. Subqueries and CTEs

A CTE gives a step a name, so a long query reads top to bottom.

```
WITH monthly AS (
  SELECT product, month_end, SUM(balance) AS bal
  FROM snapshots
  GROUP BY product, month_end
)
SELECT * FROM monthly
WHERE bal > 1000000;
```

9. Window functions

They add a calculation to each row without collapsing rows.
Pattern: `fn() OVER (PARTITION BY ... ORDER BY ...)`

```
ROW_NUMBER() OVER (PARTITION BY customer_id
                    ORDER BY start_date DESC) AS rn
RANK()           OVER (ORDER BY balance DESC)
LAG(balance)     OVER (PARTITION BY loan_id ORDER BY
month_end)
LEAD(bucket)     OVER (PARTITION BY loan_id ORDER BY
month_end)
SUM(amount)      OVER (ORDER BY pay_date)          -- running
total
AVG(balance)     OVER (PARTITION BY product)       -- group
average on every row
```

Keep the latest row per customer: compute `ROW_NUMBER()` in a CTE, then `WHERE rn = 1`.

10. Remove duplicates

```
WITH ranked AS (
  SELECT *, ROW_NUMBER() OVER (
    PARTITION BY loan_id ORDER BY updated_at DESC)
  AS rn
  FROM loans_raw
)
SELECT * FROM ranked WHERE rn = 1;
```

Find duplicates first: `GROUP BY loan_id HAVING COUNT(*) > 1`

11. Credit risk: portfolio at risk (PAR)

PAR30 = balance of loans more than 30 days past due, as a share of total balance.

```
SELECT
  SUM(CASE WHEN days_past_due > 30 THEN balance END)
  * 1.0 / SUM(balance) AS par_30,
  SUM(CASE WHEN days_past_due > 90 THEN balance END)
  * 1.0 / SUM(balance) AS par_90
FROM loans;
```

* `1.0` stops integer division from returning 0.

12. Credit risk: roll rates

Where were last month's loans this month? One row per loan per month-end.

```
WITH b AS (
  SELECT loan_id, month_end, balance,
    CASE WHEN dpd = 0 THEN '0 Current'
         WHEN dpd <= 30 THEN '1-30'
         WHEN dpd <= 60 THEN '31-60'
         ELSE '61+' END AS bucket
  FROM loan_snapshots
), p AS (
  SELECT *, LEAD(bucket) OVER (
    PARTITION BY loan_id ORDER BY month_end) AS
  next_bucket
  FROM b
)
SELECT bucket, COALESCE(next_bucket, 'Left the book') AS
next_bucket,
  COUNT(*) AS loans,
  ROUND(100.0 * COUNT(*) / SUM(COUNT(*))
        OVER (PARTITION BY bucket), 1) AS pct
FROM p
WHERE month_end < (SELECT MAX(month_end) FROM
loan_snapshots)
GROUP BY bucket, COALESCE(next_bucket, 'Left the book');
```

Full walk-through with a worked example:
adekunleokunade.com/insights/roll-rates-in-sql

13. Habits that save you

- Count rows before and after every join.
- Test a query on a small slice (`LIMIT 100`) first.
- Reconcile totals against a report you already trust.
- Name your columns (`AS`) and format your SQL so a colleague can read it.
- Never use real customer data in examples you share.