

SQL through cricket: a free SQL cheat sheet with IPL-style examples

Learn SQL the way you follow a cricket match. Each idea below comes with a cricket analogy, a short explanation and a query you can run on a fictional IPL dataset of teams, players, matches, batting_stats and bowling_stats. No sign-up needed.

Online version and the free SQL Super Over game: skillancy.in/resources/sql-through-cricket

The dataset

teams: team_id, team_name, city, home_ground, titles_won

players: player_id, player_name, team_id, role, nationality, age

matches: match_id, season, match_date, team1_id, team2_id, venue, winner_id, win_margin, win_type, player_of_match_id

batting_stats: player_id, season, innings, runs, balls_faced, fours, sixes, not_outs

bowling_stats: player_id, season, innings, overs, runs_conceded, wickets

SELECT and WHERE

Cricket analogy: Picking your playing XI: from the full squad, keep only the players who fit today's plan.

SELECT chooses the columns you want to see. WHERE keeps only the rows that match a condition. Combine conditions with AND and OR.

```
SELECT player_name, role, age
FROM players
WHERE team_id = 'CSK' AND role = 'Bowler';
```

Tip: Text values go in single quotes. Use IS NULL, not = NULL, to find missing values.

ORDER BY and LIMIT

Cricket analogy: The Orange Cap table: sort batters by runs and show the top few.

ORDER BY sorts rows (ASC is the default, DESC for highest first). LIMIT keeps the first n rows. Add a second sort column to break ties.

```
SELECT player_id, runs
FROM batting_stats
WHERE season = 2024
ORDER BY runs DESC, player_id
LIMIT 5;
```

Tip: Without a tie-breaker, two players on the same runs can swap places between runs of the query.

GROUP BY and aggregates

Cricket analogy: The season summary: collapse every match into one line per team.

GROUP BY puts rows into groups. COUNT, SUM, AVG, MIN and MAX then give one value per group. Every selected column must be grouped or aggregated.

```
SELECT winner_id, COUNT(*) AS wins
FROM matches
WHERE winner_id IS NOT NULL
GROUP BY winner_id
ORDER BY wins DESC;
```

Tip: COUNT(*) counts rows; COUNT(col) skips NULLs in that column.

JOINS

Cricket analogy: Matching the scorecard to the squad list: the scorecard has player IDs, the squad list has names.

INNER JOIN keeps rows that match in both tables. LEFT JOIN keeps every row from the left table and fills gaps with NULL.

```
SELECT p.player_name, b.runs
FROM batting_stats b
JOIN players p ON p.player_id = b.player_id
WHERE b.season = 2023
ORDER BY b.runs DESC
LIMIT 5;
```

Tip: Give tables short aliases (p, b) and always say which table a column comes from.

HAVING

Cricket analogy: Qualifying for the playoffs: first count each team's points, then keep teams above the cut-off.

WHERE filters rows before grouping. HAVING filters groups after they are summarised, so it can use COUNT, SUM and other aggregates.

```
SELECT player_id, SUM(wickets) AS wickets
FROM bowling_stats
GROUP BY player_id
HAVING SUM(wickets) >= 30
ORDER BY wickets DESC;
```

Tip: If the condition uses an aggregate, it belongs in HAVING.

Subqueries

Cricket analogy: Comparing every batter with the team average: first work out the average, then compare.

A subquery is a query inside another query. Use it in WHERE to compare with a single value, or with IN to compare with a list.

```
SELECT player_id, runs
FROM batting_stats
WHERE season = 2024
      AND runs > (SELECT AVG(runs) FROM batting_stats WHERE season = 2024)
ORDER BY runs DESC;
```

Tip: A subquery after = or > must return exactly one value.

CASE WHEN

Cricket analogy: The umpire's signal: four, six or dot ball, every delivery gets a label.

CASE WHEN adds a label based on conditions. Inside SUM or COUNT it counts only the rows you care about.

```
SELECT player_name,
      CASE WHEN age < 25 THEN 'Young gun'
            WHEN age < 32 THEN 'Prime'
            ELSE 'Veteran' END AS bracket
FROM players
WHERE team_id = 'MI';
```

Tip: Always add ELSE, otherwise unmatched rows get NULL.

Window functions

Cricket analogy: The leaderboard inside each team: rank batters within their own franchise without losing any rows.

Functions like RANK, DENSE_RANK, ROW_NUMBER and SUM OVER work across related rows but keep every row. PARTITION BY restarts the count per group.

```
SELECT p.team_id, p.player_name, b.runs,
       RANK() OVER (PARTITION BY p.team_id ORDER BY b.runs DESC) AS team_rank
FROM batting_stats b
JOIN players p ON p.player_id = b.player_id
WHERE b.season = 2022;
```

Tip: RANK leaves gaps after ties (1, 1, 3); DENSE_RANK doesn't (1, 1, 2).

CTEs (WITH)

Cricket analogy: Team strategy meeting: plan the powerplay, the middle overs and the death overs as separate steps.

WITH name AS (...) creates a named step you can use like a table. Chain several steps to keep a hard query readable.

```
WITH season_runs AS (
  SELECT player_id, SUM(runs) AS total
  FROM batting_stats
  GROUP BY player_id
)
SELECT p.player_name, s.total
FROM season_runs s
JOIN players p ON p.player_id = s.player_id
ORDER BY s.total DESC
LIMIT 5;
```

Tip: Name each CTE after what it contains, e.g. season_runs, top_bowlers.

Want structured, live SQL training? Book a free demo at skillancy.in/contact-us.