**Cricket**

**🔹 Step-by-step explanation**

**1️⃣ SELECT**

We choose two pieces of data to display in the final result:

P.Player\_Name AS batsman\_Name,

SUM(bs.Runs\_Scored) AS total\_runs

* P.Player\_Name → the name of the batsman (renamed to batsman\_Name for clarity).
* SUM(bs.Runs\_Scored) → adds up all runs scored by that batsman across all balls.

**2️⃣ FROM Ball\_by\_Ball AS bb**

* This table (Ball\_by\_Ball) stores **every ball bowled** in every match — with IDs for match, over, ball, batsman, etc.
* We give it the alias bb so we can refer to it easily later.

**3️⃣ JOIN Batsman\_Scored AS bs**

We connect (JOIN) the Ball\_by\_Ball table with Batsman\_Scored, which contains details about how many runs were scored on each ball.

The ON clause specifies **how** the two tables match up:

ON bb.Match\_Id = bs.Match\_Id

AND bb.Over\_Id = bs.Over\_Id

AND bb.Ball\_Id = bs.Ball\_Id

AND bb.Innings\_No = bs.Innings\_No

These four fields uniquely identify each delivery in the game.  
So this join ensures we match **each ball** with its corresponding **runs record**.

**4️⃣ JOIN Player AS p**

Now we join with the Player table to get the **batsman’s name**.

ON bb.Striker = p.Player\_Id

* bb.Striker → the player ID of the batsman on strike for that ball.
* p.Player\_Id → the player’s unique ID in the Player table.

This join lets us convert an ID number into a human-readable name.

**5️⃣ GROUP BY bb.Striker**

We group all records that belong to the same batsman (Striker).  
That way, the SUM(bs.Runs\_Scored) will add up **all runs for that player**.

Each row in the result will represent one batsman.

**6️⃣ ORDER BY total\_runs DESC**

Finally, we sort the results by total runs in **descending order**,  
so the **highest-scoring batsmen appear first**.

**✅ Result**

| **batsman\_Name** | **total\_runs** |
| --- | --- |
| Virat Kohli | 5400 |
| Rohit Sharma | 4950 |
| Joe Root | 4100 |
| … | … |

**💡 Summary**

| **Clause** | **Purpose** |
| --- | --- |
| Ball\_by\_Ball | Basic info about each delivery |
| Batsman\_Scored | Runs scored on each ball |
| Player | Converts player IDs to names |
| JOIN | Combines all these pieces of info |
| SUM() | Totals runs per player |
| GROUP BY | One row per batsman |
| ORDER BY DESC | Sort from most to least runs |

In plain English:

“For every batsman, sum up all the runs he scored (from all matches and balls), and show the list from the top scorer down.”

## 🔹 Step-by-step explanation

### 1️⃣ WITH player\_match\_runs AS (...)

This is a **Common Table Expression (CTE)** — a temporary result that exists only during the query.  
You can think of it as creating a virtual table called player\_match\_runs.

Inside the parentheses is the query that builds that temporary table.

### 2️⃣ Inside the CTE:

SELECT

p.Player\_Name AS batsman\_name,

bb.Match\_Id,

SUM(bs.Runs\_Scored) AS runs\_in\_match

This part collects, for **each batsman in each match**, the **total runs** they scored.

Let’s break down the joins again:

#### FROM Ball\_by\_Ball AS bb

Contains one row for every delivery in every match.

#### JOIN Batsman\_Scored AS bs

Matches each delivery to the runs scored on it (using Match\_Id, Over\_Id, Ball\_Id, Innings\_No).

#### JOIN Player AS p

Links each striker’s ID to the batsman’s actual name.

### 3️⃣ GROUP BY bb.Striker, bb.Match\_Id

This means:

“For each batsman (Striker) and for each match, sum up all their runs.”

So the **CTE result** player\_match\_runs will look like:

| **batsman\_name** | **Match\_Id** | **runs\_in\_match** |
| --- | --- | --- |
| Virat Kohli | 101 | 78 |
| Virat Kohli | 102 | 45 |
| Rohit Sharma | 101 | 102 |
| Joe Root | 103 | 55 |

### 4️⃣ Main query:

Now we use that CTE to count fifties and hundreds per player.

SELECT

batsman\_name,

SUM(CASE WHEN runs\_in\_match BETWEEN 50 AND 99 THEN 1 ELSE 0 END) AS fifties,

SUM(CASE WHEN runs\_in\_match >= 100 THEN 1 ELSE 0 END) AS hundreds

FROM player\_match\_runs

GROUP BY batsman\_name

ORDER BY hundreds DESC, fifties DESC;

Let’s unpack that.

### 5️⃣ CASE WHEN ... THEN ... ELSE ... END

This is a conditional expression inside SQL.

For each runs\_in\_match:

* If it’s **between 50 and 99**, count 1 (a fifty).
* Else count 0.

Same logic for **hundreds** (runs\_in\_match >= 100).

Then we use SUM() to **add up all the 1s**, effectively counting:

* how many matches each batsman scored a fifty,
* how many matches each batsman scored a hundred.

### 6️⃣ GROUP BY batsman\_name

Groups all matches by batsman — one final row per batsman.

### 7️⃣ ORDER BY hundreds DESC, fifties DESC

Sorts the final result:

1. Most hundreds first,
2. If tied, by most fifties.

## ✅ Final Output Example

| **batsman\_name** | **fifties** | **hundreds** |
| --- | --- | --- |
| Rohit Sharma | 25 | 10 |
| Virat Kohli | 20 | 8 |
| Joe Root | 18 | 6 |

## 🧠 Summary

| **Step** | **Purpose** |
| --- | --- |
| WITH player\_match\_runs AS (...) | Build a temporary table with total runs per batsman per match |
| SUM(bs.Runs\_Scored) | Calculate total runs in that match |
| CASE WHEN runs\_in\_match ... | Check if those runs are a 50 or a 100 |
| SUM() | Count how many 50s or 100s each batsman has |
| GROUP BY batsman\_name | One row per player |
| ORDER BY | Sorts best performers first |

In plain English:

“First, calculate how many runs each batsman scored in each match.  
Then, for each batsman, count how many times they made a fifty (50–99) or a hundred (100+), and list them in order of top performers.”

**🔹 Step 1: WITH bowling\_stats AS (...)**

This part defines a **Common Table Expression (CTE)** — a temporary result set named bowling\_stats.  
It calculates **how many runs and wickets each bowler had in each match**.

**🔹 Step 2: Inside the CTE**

**SELECT**

p.Player\_Name AS bowler\_name,

bb.Match\_Id,

SUM(bs.Runs\_Scored) AS runs\_conceded,

COUNT(wt.Player\_Out) AS wickets

For each bowler and match:

* p.Player\_Name → name of the bowler.
* bb.Match\_Id → which match it was.
* SUM(bs.Runs\_Scored) → total runs given away by that bowler in that match.
* COUNT(wt.Player\_Out) → total wickets taken by that bowler in that match.

**FROM Ball\_by\_Ball AS bb**

Each row represents **one ball** bowled in a match (contains bowler, striker, over, ball number, etc.).

**JOIN Player p ON bb.Bowler = p.Player\_Id**

Connects each bowler’s ID to their name from the Player table.  
This gives human-readable bowler names.

**LEFT JOIN Batsman\_Scored AS bs**

Connects each ball to the **runs scored** on that ball.

The join uses multiple fields to ensure the correct match and delivery:

bb.Match\_Id = bs.Match\_Id

AND bb.Over\_Id = bs.Over\_Id

AND bb.Ball\_Id = bs.Ball\_Id

AND bb.Innings\_No = bs.Innings\_No

✅ LEFT JOIN ensures we still keep the ball record even if **no runs were scored** (e.g. a wicket or dot ball).

**LEFT JOIN Wicket\_Taken AS wt**

Adds wicket information, also matched by ball identifiers.

✅ Another LEFT JOIN — so if a ball didn’t take a wicket, it still appears in the results.

**GROUP BY bb.Bowler, bb.Match\_Id**

Groups all deliveries bowled by the same bowler in the same match.

After grouping:

* SUM(bs.Runs\_Scored) → total runs conceded that match,
* COUNT(wt.Player\_Out) → total wickets in that match.

**🧩 The CTE result (bowling\_stats) looks like:**

| **bowler\_name** | **Match\_Id** | **runs\_conceded** | **wickets** |
| --- | --- | --- | --- |
| Bumrah | 101 | 32 | 3 |
| Bumrah | 102 | 28 | 2 |
| Starc | 101 | 40 | 4 |
| Starc | 102 | 25 | 1 |

**🔹 Step 3: Main Query**

Now we use that CTE to find **each bowler’s best match figures**.

SELECT

bowler\_name,

MAX(wickets) AS best\_wickets,

MIN(runs\_conceded) AS least\_runs\_in\_best

FROM bowling\_stats

GROUP BY bowler\_name

ORDER BY best\_wickets DESC;

Let’s unpack it:

**MAX(wickets) AS best\_wickets**

For each bowler, find the **highest number of wickets** they’ve ever taken in a single match.

**MIN(runs\_conceded) AS least\_runs\_in\_best**

Finds the **lowest number of runs conceded** — the fewest runs the bowler gave away in their best matches.

*(Note: this part doesn’t strictly ensure “least runs when taking best wickets” — it just finds the overall minimum runs in any match. But it’s often used as a simple “best economy” indicator.)*

**GROUP BY bowler\_name**

Groups results by bowler, so we get one row per player.

**ORDER BY best\_wickets DESC**

Sorts the final table so that bowlers with the most wickets in a single match come first.

**✅ Example Output**

| **bowler\_name** | **best\_wickets** | **least\_runs\_in\_best** |
| --- | --- | --- |
| Mitchell Starc | 6 | 22 |
| Jasprit Bumrah | 5 | 18 |
| Pat Cummins | 4 | 25 |

**🧠 Summary**

| **Step** | **Purpose** |
| --- | --- |
| Ball\_by\_Ball | Every delivery (who bowled, over, ball, etc.) |
| Batsman\_Scored | Runs scored on each ball |
| Wicket\_Taken | Wickets on each ball |
| Player | Bowler names |
| CTE bowling\_stats | Calculates runs conceded & wickets per bowler per match |
| MAX(wickets) | Finds bowler’s best wicket performance |
| MIN(runs\_conceded) | Finds lowest runs conceded |
| ORDER BY | Shows best bowlers first |

**🗣️ In plain English:**

“For each bowler, calculate how many runs they gave and wickets they took in each match.  
Then show, for each bowler, the match where they took the most wickets, and how few runs they’ve ever conceded.”