

To compare the performance with multiple queries for the below,

List matches where the total goals scored are above the average:

1) Original Subquery-Based Query:

```
SELECT m.*, (s1.noOfGoals + s2.noOfGoals) AS totalGoals
FROM matches m INNER JOIN statistic s1 ON m.homeTeamId = s1.playerId
INNER JOIN statistic s2 ON m.awayTeamId = s2.playerId
WHERE (s1.noOfGoals + s2.noOfGoals) > (
    SELECT AVG(s1.noOfGoals + s2.noOfGoals) AS avgGoals FROM matches m
    INNER JOIN statistic s1 ON m.homeTeamId = s1.playerId
    INNER JOIN statistic s2 ON m.awayTeamId = s2.playerId);
```

Console logs:

```
12:19:32      SELECT m.*, (s1.noOfGoals + s2.noOfGoals) AS totalGoals FROM matches
m INNER JOIN statistic s1 ON m.homeTeamId = s1.playerId INNER JOIN statistic s2 ON
m.awayTeamId = s2.playerId WHERE (s1.noOfGoals + s2.noOfGoals) > ( SELECT
AVG(s1.noOfGoals + s2.noOfGoals) AS avgGoals FROM matches m      INNER JOIN
statistic s1 ON m.homeTeamId = s1.playerId      INNER JOIN statistic s2 ON
m.awayTeamId = s2.playerId) LIMIT 0, 10000      2428 row(s) returned0.141 sec / 0.109
sec
```

The screenshot shows the MySQL Workbench interface. The SQL editor contains the following query:

```
-- List matches where the total goals scored are above the average:
-- Original Subquery-Based Query
SELECT m.*, (s1.noOfGoals + s2.noOfGoals) AS totalGoals
FROM matches m INNER JOIN statistic s1 ON m.homeTeamId = s1.playerId
INNER JOIN statistic s2 ON m.awayTeamId = s2.playerId
WHERE (s1.noOfGoals + s2.noOfGoals) > (
    SELECT AVG(s1.noOfGoals + s2.noOfGoals) AS avgGoals FROM matches m
    INNER JOIN statistic s1 ON m.homeTeamId = s1.playerId
    INNER JOIN statistic s2 ON m.awayTeamId = s2.playerId);
-- Common Table Expression (CTE)
WITH AvgGoals AS (
    SELECT AVG(s1.noOfGoals + s2.noOfGoals) AS avgGoals FROM matches m
    INNER JOIN statistic s1 ON m.homeTeamId = s1.playerId
    INNER JOIN statistic s2 ON m.awayTeamId = s2.playerId);
SELECT m.*, (s1.noOfGoals + s2.noOfGoals) AS totalGoals FROM matches m
INNER JOIN statistic s1 ON m.homeTeamId = s1.playerId
INNER JOIN statistic s2 ON m.awayTeamId = s2.playerId
WHERE (s1.noOfGoals + s2.noOfGoals) > AvgGoals.avgGoals
LIMIT 0, 10000;
```

The result grid shows 24 rows of match data. The console log shows the query execution time as 0.141 sec / 0.109 sec.

Subquery-Based Query:

- The main query selects information about matches (m.*) and calculates the total number of goals for each match by summing the goals scored by the home and away teams (s1.noOfGoals + s2.noOfGoals).
- The WHERE clause filters the results to include only those matches where the total goals are greater than the average.
- The average is calculated using a subquery that joins the matches table with the statistic table twice to get the home and away team statistics, and then calculates the average of the total goals.
- This query achieves the desired result, its performance may be influenced by factors such as subquery overhead, repetitive joins, and indexing.

2) Common Table Expression (CTE):

```
WITH AvgGoals AS (
  SELECT AVG(s1.noOfGoals + s2.noOfGoals) AS avgGoals
  FROM matches m
  INNER JOIN statistic s1 ON m.homeTeamId = s1.playerId
  INNER JOIN statistic s2 ON m.awayTeamId = s2.playerId
)
SELECT m.*, (s1.noOfGoals + s2.noOfGoals) AS totalGoals
FROM matches m
INNER JOIN statistic s1 ON m.homeTeamId = s1.playerId
INNER JOIN statistic s2 ON m.awayTeamId = s2.playerId
JOIN AvgGoals
WHERE (s1.noOfGoals + s2.noOfGoals) > avgGoals;
```

Console logs:

```
12:19:32      WITH AvgGoals AS (  SELECT AVG(s1.noOfGoals + s2.noOfGoals) AS
avgGoals  FROM matches m  INNER JOIN statistic s1 ON m.homeTeamId = s1.playerId
INNER JOIN statistic s2 ON m.awayTeamId = s2.playerId ) SELECT m.*, (s1.noOfGoals +
s2.noOfGoals) AS totalGoals FROM matches m INNER JOIN statistic s1 ON
m.homeTeamId = s1.playerId INNER JOIN statistic s2 ON m.awayTeamId = s2.playerId
JOIN AvgGoals WHERE (s1.noOfGoals + s2.noOfGoals) > avgGoals      2428 row(s)
returned      0.078 sec / 0.031 sec
```

The screenshot shows the MySQL Workbench interface. The SQL editor contains the following query:

```

-- Common Table Expression (CTE)
11
12 WITH AvgGoals AS (
13     SELECT AVG(s1.noOfGoals + s2.noOfGoals) AS avgGoals
14     FROM matches m
15     INNER JOIN statistic s1 ON m.homeTeamId = s1.playerId
16     INNER JOIN statistic s2 ON m.awayTeamId = s2.playerId
17 )
18 SELECT m.*, (s1.noOfGoals + s2.noOfGoals) AS totalGoals
19 FROM matches m
20 INNER JOIN statistic s1 ON m.homeTeamId = s1.playerId
21 INNER JOIN statistic s2 ON m.awayTeamId = s2.playerId
22 WHERE (totalGoals > avgGoals)
23
24

```

The Results Grid shows the following data:

matchId	stadiumId	refereeId	playerOfMatchId	matchName	matchStartDate	matchEndDate	matchStartTime	matchEndTime	homeTeamId	awayTeamId	totalGoals
3	2	632	3632	repelat	2008-10-12	2022-01-12	06:56:50	10:26:13	32	33	61
5	5	515	515	omnis	1972-09-19	2019-09-28	05:51:00	11:16:09	115	116	79
6	7	927	2927	pariatud	1999-12-16	1974-12-28	05:19:54	23:32:39	127	128	54
7	10	1000	1000	commodi	1991-06-25	2023-03-24	07:22:28	18:45:33	200	201	92
8	6	46	4046	officia	1995-05-30	2007-08-30	17:42:28	16:57:27	46	47	49
9	8	228	4228	repudlandae	1988-04-13	1990-07-01	23:05:45	20:45:01	228	229	62
12	10	610	1610	quis	2017-12-12	1993-09-02	21:21:19	16:45:06	10	11	96
13	7	687	1687	possimus	2020-01-15	1970-01-25	14:08:42	08:33:10	87	88	85
21	6	656	4656	nisi	2018-07-09	2019-02-25	13:52:32	17:36:38	256	257	71
22	3	743	4743	repelat	2023-05-25	2012-08-11	04:43:11	03:42:06	343	344	44
23	3	253	253	possimus	1989-05-15	1986-08-30	18:06:24	14:22:06	253	254	80
24	3	753	753	iusto	2015-09-13	1984-05-28	00:02:43	01:42:21	353	354	37

The console logs show the execution of the query with the following messages:

```

30 12:18:27 WITH AvgGoals AS ( SELECT AVG(s1.noOfGoals + s2.noOfGoals) AS avgGoals FROM matches m 2428 row(s) returned 0.156 sec / 0.125 sec
31 12:19:32 SELECT m.*, (s1.noOfGoals + s2.noOfGoals) AS totalGoals FROM matches m INNER JOIN statistic s1 ON m.homeTeamId = s1.playerId 2428 row(s) returned 0.141 sec / 0.109 sec
32 12:19:32 WITH AvgGoals AS ( SELECT AVG(s1.noOfGoals + s2.noOfGoals) AS avgGoals FROM matches m 2428 row(s) returned 0.078 sec / 0.031 sec

```

Common Table Expression (CTE):

A temporary table (AvgGoals) is created to calculate the average number of goals scored in all matches. This CTE is computed once.

Main Query:

- The main query selects information about matches (m.*) and calculates the total number of goals for each match by summing the goals scored by the home and away teams (s1.noOfGoals + s2.noOfGoals).
- The main query then joins with the AvgGoals CTE.
- The WHERE clause filters the results to include only those matches where the total goals are greater than the average calculated in the CTE.
- This query is designed to be efficient by calculating the average only once and then using that result in the main query.

3) Window Function Query:

WITH MatchGoals AS (

SELECT m.*, (s1.noOfGoals + s2.noOfGoals) AS totalGoals, **AVG(s1.noOfGoals + s2.noOfGoals) OVER () AS avgGoals**

FROM matches m INNER JOIN statistic s1 ON m.homeTeamId = s1.playerId

INNER JOIN statistic s2 ON m.awayTeamId = s2.playerId

)

SELECT mg.* FROM MatchGoals mg WHERE (totalGoals > avgGoals);

Console logs:

```

12:19:32 WITH MatchGoals AS ( SELECT m.*, (s1.noOfGoals + s2.noOfGoals) AS totalGoals, AVG(s1.noOfGoals + s2.noOfGoals) OVER () AS avgGoals FROM matches m INNER JOIN statistic s1 ON m.homeTeamId = s1.playerId INNER JOIN statistic s2 ON

```

m.awayTeamId = s2.playerId) SELECT mg.* FROM MatchGoals mg WHERE (totalGoals > avgGoals) **2428 row(s) returned 0.078 sec / 0.015 sec**

The screenshot shows the MySQL Workbench interface. The SQL editor contains a query that uses a Common Table Expression (CTE) named 'MatchGoals' and a window function 'AVG(...) OVER ()' to calculate the average of total goals for all matches. The query is as follows:

```

-- Window Function Query
30 WITH MatchGoals AS (
31     SELECT m.*, (s1.noOfGoals + s2.noOfGoals) AS totalGoals, AVG(s1.noOfGoals + s2.noOfGoals) OVER () AS avgGoals
32     FROM matches m INNER JOIN statistic s1 ON m.homeTeamId = s1.playerId
33     INNER JOIN statistic s2 ON m.awayTeamId = s2.playerId
34 )
35 SELECT mg.* FROM MatchGoals mg WHERE (totalGoals > avgGoals);

```

The results are displayed in a table with columns: matchId, stadiumId, refereeId, playerOfMatchId, matchName, matchStartDate, matchEndDate, matchStartTime, matchEndTime, homeTeamId, awayTeamId, and totalGoals. The table shows 24 rows of data.

The bottom of the screenshot shows the 'Output' tab with the execution details for the query:

#	Time	Action	Message	Duration / Fetch
31	12:19:32	SELECT m.*, (s1.noOfGoals + s2.noOfGoals) AS totalGoals FROM matches m INNER JOIN statistic ...	2428 row(s) returned	0.141 sec / 0.109 sec
32	12:19:32	WITH AvgGoals AS (SELECT AVG(s1.noOfGoals + s2.noOfGoals) AS avgGoals FROM match...	2428 row(s) returned	0.078 sec / 0.031 sec
33	12:19:32	WITH MatchGoals AS (SELECT m.*, (s1.noOfGoals + s2.noOfGoals) AS totalGoals, AVG(s1.noOfGoals + s2.noOfGoals) AS avgGoals FROM matches m INNER JOIN statistic s1 ON m.homeTeamId = s1.playerId INNER JOIN statistic s2 ON m.awayTeamId = s2.playerId) SELECT mg.* FROM MatchGoals mg WHERE (totalGoals > avgGoals);	2428 row(s) returned	0.078 sec / 0.015 sec

Window Function Query:

- A Common Table Expression (CTE) named MatchGoals is created.
- Within the CTE, the query selects information about matches (m.*) and calculates the total number of goals for each match by summing the goals scored by the home and away teams (s1.noOfGoals + s2.noOfGoals).
- A window function (AVG(...) OVER ()) is applied to calculate the average of total goals for all matches.

Main Query:

- The main query selects all columns from the MatchGoals CTE.
- The WHERE clause filters the results to include only those matches where the total goals (totalGoals) are greater than the calculated average (avgGoals).
- This query efficiently calculates the average total goals using a window function and then filters matches based on this average.

Summary:

All three queries appear to achieve the same result—selecting matches where the total number of goals is greater than the average number of goals.

The differences lie in the syntax and structure of the queries, with the second and third queries utilizing CTEs for better readability and potentially improved performance.

The execution times are also provided for each query. The third query, in particular, seems to have the lowest execution time, indicating potential efficiency.