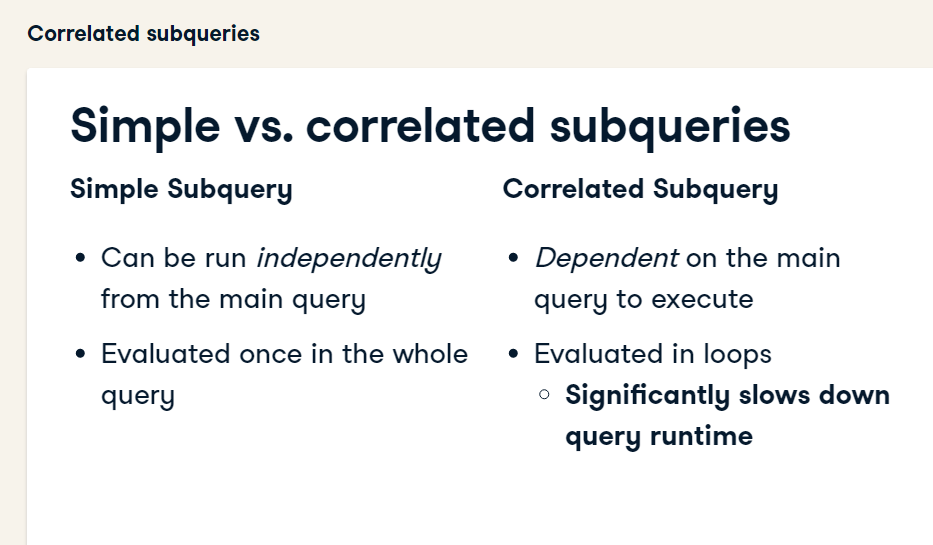
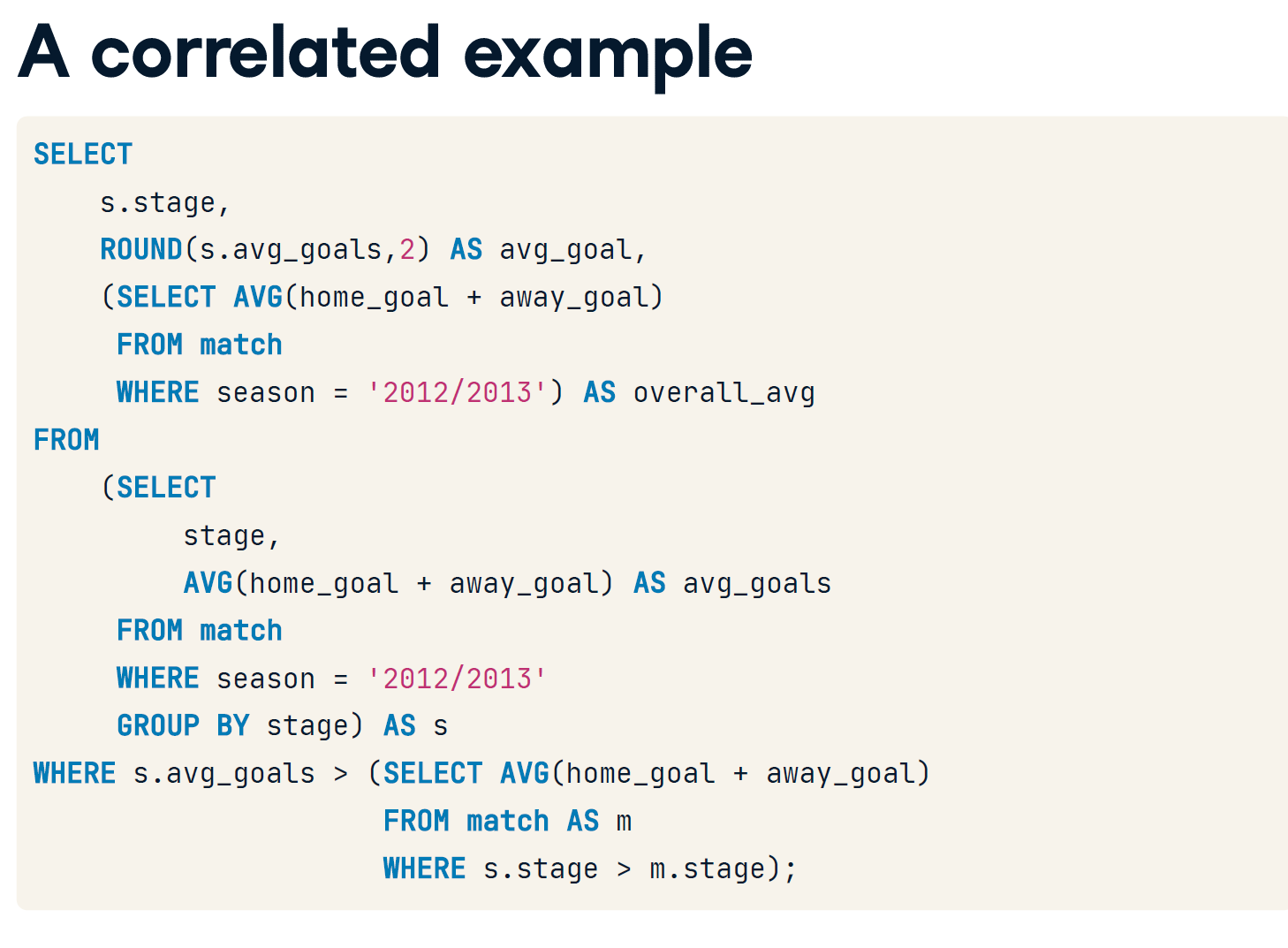
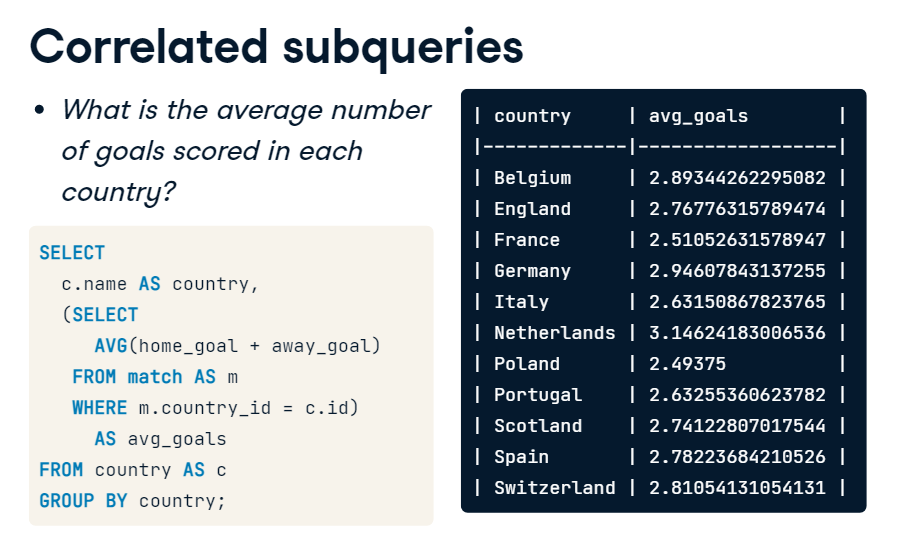
Correlated Subquery







**Basic Correlated Subqueries**

Correlated subqueries are subqueries that reference one or more columns in the main query. Correlated subqueries depend on information in the main query to run, and thus, cannot be executed on their own.

Correlated subqueries are evaluated in SQL once per row of data retrieved -- a process that takes a lot more computing power and time than a simple subquery.

In this exercise, you will practice using correlated subqueries to examine matches with scores that are extreme outliers for each country -- above 3 times the average score!

* Select the country\_id, date, home\_goal, and away\_goal columns in the main query.
* Complete the AVG value in the subquery.
* Complete the subquery column references, so that country\_id is matched in the main and subquery.

SELECT

    -- Select country ID, date, home, and away goals from match

    main.country\_id,

    main.date,

    main.home\_goal,

    main.away\_goal

FROM match AS main

WHERE

    -- Filter the main query by the subquery

    (home\_goal + away\_goal) >

        (SELECT AVG((home\_goal + sub.away\_goal) \* 3)

         FROM match AS sub

         -- Join the main query to the subquery in WHERE

         WHERE main.country\_id = sub.country\_id);

| **country\_id** | **date** | **home\_goal** | **away\_goal** |
| --- | --- | --- | --- |
| 1 | 2011-10-29 | 4 | 5 |
| 1729 | 2011-08-28 | 8 | 2 |
| 1729 | 2012-12-29 | 7 | 3 |
| 1729 | 2013-05-19 | 5 | 5 |
| 1729 | 2013-12-14 | 6 | 3 |

* Select the country\_id, date, home\_goal, and away\_goal columns in the main query.
* Complete the subquery: Select the matches with the highest number of total goals.
* Match the subquery to the main query using country\_id and season.
* Fill in the correct logical operator so that total goals equals the max goals recorded in the subquery.
* SELECT
* -- Select country ID, date, home, and away goals from match
* main.country\_id,
* main.date,
* main.home\_goal,
* main.away\_goal
* FROM match AS main
* WHERE
* -- Filter for matches with the highest number of goals scored
* (home\_goal + away\_goal) =
* (SELECT MAX(home\_goal + sub.away\_goal)
* FROM match AS sub
* WHERE main.country\_id = sub.country\_id
* AND main.season = sub.season);

| **country\_id** | **date** | **home\_goal** | **away\_goal** |
| --- | --- | --- | --- |
| 1 | 2011-10-29 | 4 | 5 |
| 1 | 2012-11-17 | 2 | 6 |
| 1 | 2012-12-09 | 1 | 7 |

<https://github.com/Novartus/Datacamp_SQL/blob/master/Intermediate%20SQL/3.%20Correlated%20Queries%2C%20Nested%20Queries%2C%20and%20Common%20Table%20Expressions.sql>

# 

# 

# 

# Nested simple subqueries

Nested subqueries can be either simple or correlated.

Just like an unnested subquery, a nested subquery's components can be executed independently of the outer query, while a correlated subquery requires both the outer and inner subquery to run and produce results.

In this exercise, you will practice creating a nested subquery to examine the highest total number of goals in each season, overall, and during July across all seasons.

* Complete the main query to select the season and the max total goals in a match for each season. Name this max\_goals.
* Complete the first simple subquery to select the max total goals in a match across all seasons. Name this overall\_max\_goals.
* Complete the nested subquery to select the maximum total goals in a match played in July across all seasons.
* Select the maximum total goals in the outer subquery. Name this entire subquery july\_max\_goals.
* SELECT
* -- Select the season and max goals scored in a match
* season,
* MAX(home\_goal + away\_goal) AS max\_goals,
* -- Select the overall max goals scored in a match
* (SELECT MAX(home\_goal+ away\_goal) FROM match) AS overall\_max\_goals,
* -- Select the max number of goals scored in any match in July
* (SELECT MAX(home\_goal + away\_goal)
* FROM match
* WHERE id IN (
* SELECT id FROM match WHERE EXTRACT(MONTH FROM date) = 07)) AS july\_max\_goals
* FROM match
* GROUP BY season;

| **season** | **max\_goals** | **overall\_max\_goals** | **july\_max\_goals** |
| --- | --- | --- | --- |
| 2013/2014 | 10 | 11 | 7 |
| 2012/2013 | 11 | 11 | 7 |
| 2014/2015 | 10 | 11 | 7 |
| 2011/2012 | 10 | 11 | 7 |

# Nest a subquery in FROM

What's the average number of matches per season where a team scored 5 or more goals? How does this differ by country?

Let's use a nested, correlated subquery to perform this operation. In the real world, you will probably find that nesting multiple subqueries is a task you don't have to perform often. In some cases, however, you may find yourself struggling to properly group by the column you want, or to calculate information requiring multiple mathematical transformations (i.e., an AVG of a COUNT).

Nesting subqueries and performing your transformations one step at a time, adding it to a subquery, and then performing the next set of transformations is often the easiest way to yield accurate information about your data. Let's get to it!

* Generate a list of matches where ***at least one team*** scored ***5 or more goals***.
* -- Select matches where a team scored 5+ goals
* SELECT
* country\_id,
* season,
* id
* FROM match
* WHERE home\_goal >=5 OR away\_goal >=5;

| **country\_id** | **season** | **id** |
| --- | --- | --- |
| 1 | 2011/2012 | 764 |
| 1 | 2011/2012 | 766 |
| 1 | 2011/2012 | 781 |
| 1 | 2011/2012 | 791 |

* Turn the query from the previous step into a *subquery* in the FROM statement.
* COUNT the match ids generated in the previous step, and group the query by country\_id and season.
* -- Count match ids
* SELECT
* country\_id,
* season,
* COUNT(id) AS matches
* -- Set up and alias the subquery
* FROM (
* SELECT
* country\_id,
* season,
* id
* FROM match
* WHERE home\_goal >= 5 OR away\_goal >= 5 )AS subquery
* -- Group by country\_id and season
* GROUP BY country\_id, season;

| **country\_id** | **season** | **matches** |
| --- | --- | --- |
| 19694 | 2012/2013 | 5 |
| 21518 | 2012/2013 | 23 |
| 13274 | 2011/2012 | 24 |

* Finally, declare the same query from step 2 as a subquery in FROM with the alias outer\_s.
* Left join it to the country table using the outer query's country\_id column.
* Calculate an AVG of high scoring matches per country in the main query.

SELECT c.name AS country,

    -- Calculate the average matches per season

  AVG(c.id) AS avg\_seasonal\_high\_scores

FROM country AS c

-- Left join outer\_s to country

LEFT JOIN (

  SELECT country\_id, season,

         COUNT(id) AS matches

  FROM (

    SELECT country\_id, season, id

  FROM match

  WHERE home\_goal >= 5 OR away\_goal >= 5) AS inner\_s

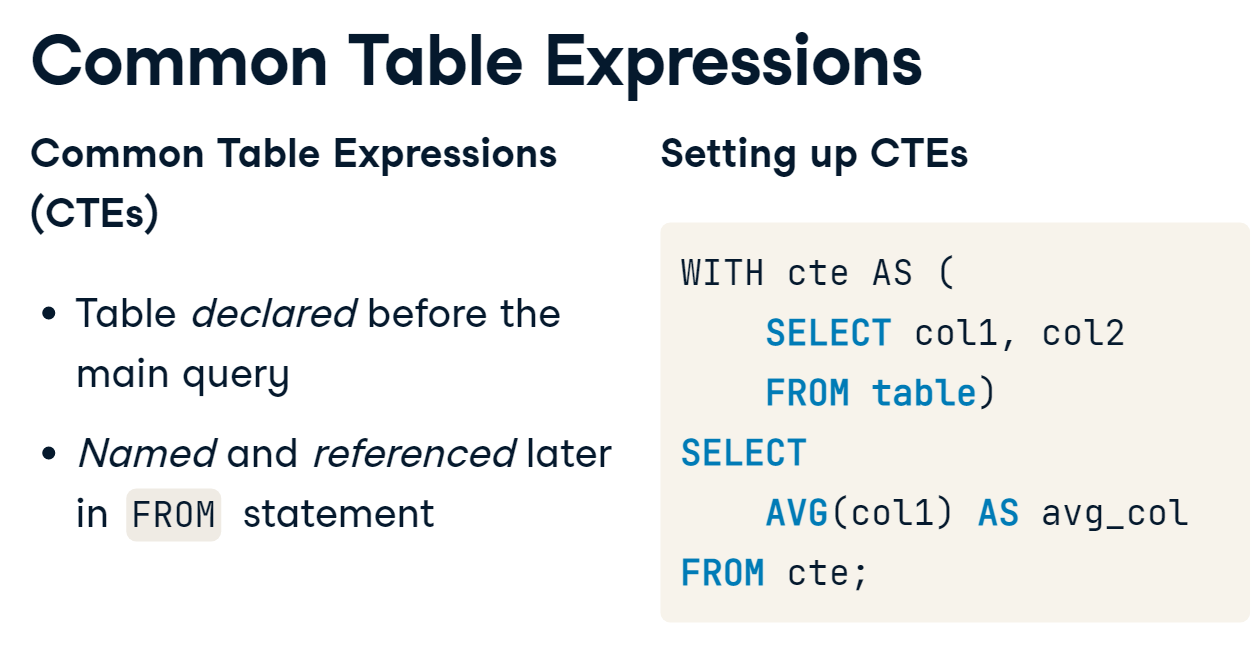
  -- Close parentheses and alias the subquery

  GROUP BY country\_id, season) AS outer\_s

ON c.id = outer\_s.country\_id

GROUP BY country;

| **country** | **avg\_seasonal\_high\_scores** |
| --- | --- |
| Portugal | 17642.000000000000 |
| France | 4769.0000000000000000 |
| Scotland | 19694.000000000000 |





# Clean up with CTEs

In chapter 2, you generated a list of countries and the number of matches in each country with more than 10 total goals. The query in that exercise utilized a subquery in the FROM statement in order to filter the matches before counting them in the main query. Below is the query you created:

SELECT

c.name AS country,

COUNT(sub.id) AS matches

FROM country AS c

INNER JOIN (

SELECT country\_id, id

FROM match

WHERE (home\_goal + away\_goal) >= 10) AS sub

ON c.id = sub.country\_id

GROUP BY country;

You can list one (or more) subqueries as **common table expressions** (CTEs) by declaring them ahead of your main query, which is an excellent tool for organizing information and placing it in a logical order.

In this exercise, let's rewrite a similar query using a CTE.

* Complete the syntax to declare your CTE.
* Select the country\_id and match id from the match table in your CTE.
* Left join the CTE to the league table using country\_id.
* -- Set up your CTE
* WITH match\_list AS (
* SELECT
* country\_id,
* id
* FROM match
* WHERE (home\_goal + away\_goal) >= 10)
* -- Select league and count of matches from the CTE
* SELECT
* l.name AS league,
* COUNT(match\_list.id) AS matches
* FROM league AS l
* -- Join the CTE to the league table
* LEFT JOIN match\_list ON l.id = match\_list.country\_id
* GROUP BY l.name;

| **league** | **matches** |
| --- | --- |
| Switzerland Super League | 0 |
| Poland Ekstraklasa | 0 |
| Netherlands Eredivisie | 1 |

# Organizing with CTEs

Previously, you modified a query based on a statement you completed in chapter 2 using common table expressions.

This time, let's expand on the exercise by looking at details about matches with very high scores using CTEs. Just like a subquery in FROM, you can join tables inside a CTE.

* Declare your CTE, where you create a list of all matches with the league name.
* Select the league, date, home, and away goals from the CTE.
* Filter the main query for matches with 10 or more goals.
* -- Set up your CTE
* WITH match\_list AS (
* -- Select the league, date, home, and away goals
* SELECT
* l.name AS league,
* m.date,
* m.home\_goal,
* m.away\_goal,
* (m.home\_goal + m.away\_goal) AS total\_goals
* FROM match AS m
* LEFT JOIN league as l ON m.country\_id = l.id)
* -- Select the league, date, home, and away goals from the CTE
* SELECT league, date, home\_goal, away\_goal
* FROM match\_list
* -- Filter by total goals
* WHERE total\_goals>=10;

| **league** | **date** | **home\_goal** | **away\_goal** |
| --- | --- | --- | --- |
| England Premier League | 2011-08-28 | 8 | 2 |
| England Premier League | 2012-12-29 | 7 | 3 |
| England Premier League | 2013-05-19 | 5 | 5 |
| Germany 1. Bundesliga | 2013-03-30 | 9 | 2 |

# CTEs with nested subqueries

If you find yourself listing multiple subqueries in the FROM clause with nested statement, your query will likely become long, complex, and difficult to read.

Since many queries are written with the intention of being saved and re-run in the future, proper organization is key to a seamless workflow. Arranging subqueries as CTEs will save you time, space, and confusion in the long run!

* Declare a CTE that calculates the total goals from matches in August of the 2013/2014 season.
* Left join the CTE onto the league table using country\_id from the match\_list CTE.
* Filter the list on the inner subquery to only select matches in August of the 2013/2014 season.

-- Set up your CTE

WITH match\_list AS (

    SELECT

      country\_id,

       (home\_goal + away\_goal) AS goals

    FROM match

    -- Create a list of match IDs to filter data in the CTE

    WHERE id IN (

       SELECT id

       FROM match

       WHERE season = '2013/2014'  AND EXTRACT(MONTH FROM date) = 8))

-- Select the league name and average of goals in the CTE

SELECT

  l.name,

    AVG(match\_list.goals)

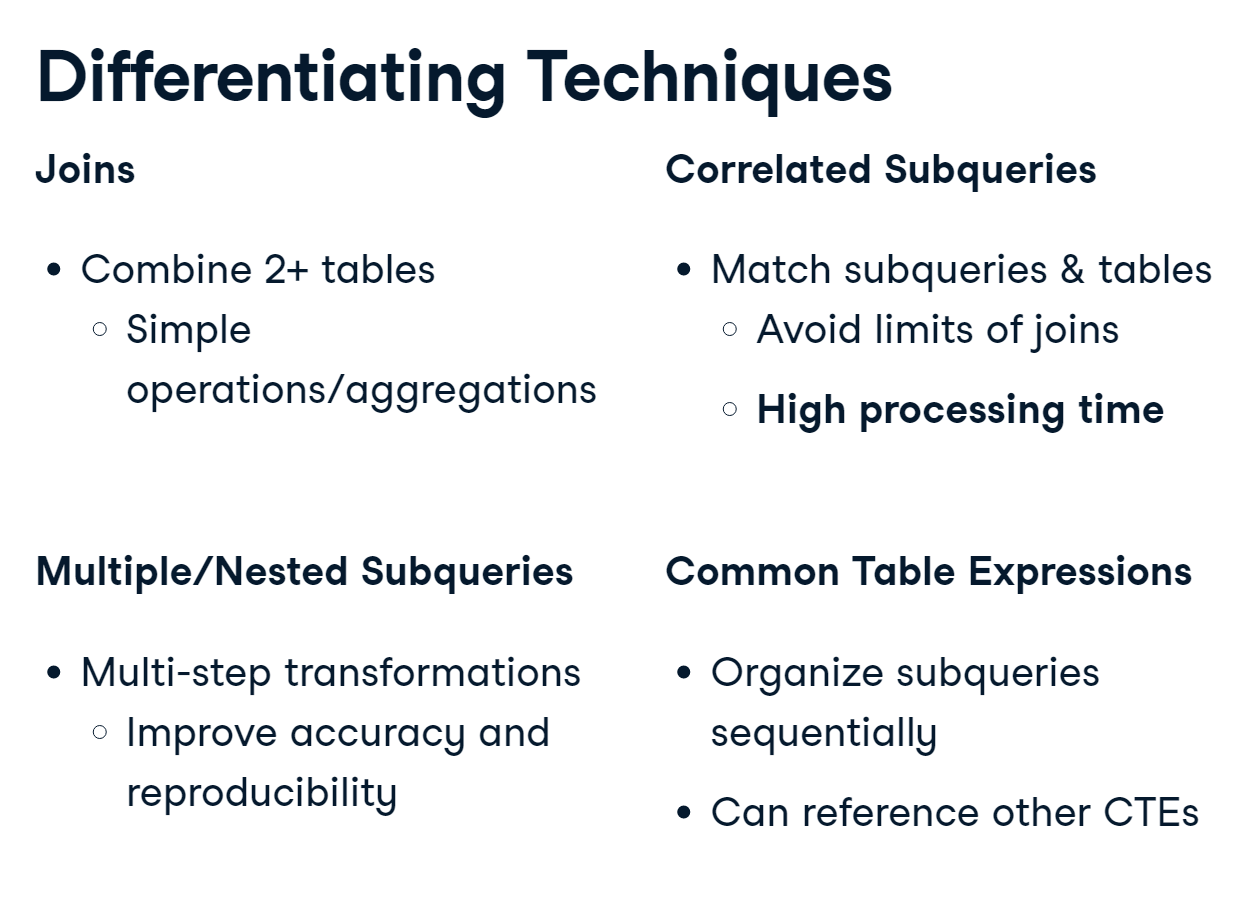
FROM league AS l

-- Join the CTE onto the league table

LEFT JOIN match\_list ON l.id = match\_list.country\_id

GROUP BY l.name;

| **name** | **avg** |
| --- | --- |
| Switzerland Super League | 1.9375000000000000 |
| Poland Ekstraklasa | 2.3103448275862069 |
| Netherlands Eredivisie | 3.4146341463414634 |
| Scotland Premier League | 2.1379310344827586 |
|  |  |



# Get team names with a subquery

Let's solve a problem we've encountered a few times in this course so far -- How do you get both the home and away team names into one final query result?

Out of the 4 techniques we just discussed, this can be performed using subqueries, correlated subqueries, and CTEs. Let's practice creating similar result sets using each of these 3 methods over the next 3 exercises, starting with subqueries in FROM

* Create a query that left joins team to match in order to get the identity of the home team. This becomes the subquery in the next step.

SELECT

    m.id,

    t.team\_long\_name AS hometeam

-- Left join team to match

FROM match AS m

LEFT JOIN team as t

ON m.hometeam\_id = team\_api\_id;

| **date** | **hometeam** | **awayteam** | **home\_goal** | **away\_goal** |
| --- | --- | --- | --- | --- |
| 2011-07-29 | Oud-Heverlee Leuven | RSC Anderlecht | 2 | 1 |
| 2011-07-30 | RAEC Mons | Standard de Liège | 1 | 1 |
| 2011-07-30 | KRC Genk | Beerschot AC | 3 | 1 |

* Add a second subquery to the FROM statement to get the away team name, changing only the hometeam\_id. Left join both subqueries to the match table on the id column.
* SELECT
* m.date,
* -- Get the home and away team names
* hometeam,
* awayteam,
* m.home\_goal,
* m.away\_goal
* FROM match AS m
* -- Join the home subquery to the match table
* LEFT JOIN(
* SELECT match.id, team.team\_long\_name AS hometeam
* FROM match
* LEFT JOIN team
* ON match.hometeam\_id = team.team\_api\_id) AS home
* ON home.id = m.id
* -- Join the away subquery to the match table
* LEFT JOIN (
* SELECT match.id, team.team\_long\_name AS awayteam
* FROM match
* LEFT JOIN team
* -- Get the away team ID in the subquery
* ON match.awayteam\_id  = team.team\_api\_id) AS away
* ON away.id= m.id;

| **date** | **hometeam** | **awayteam** | **home\_goal** | **away\_goal** |
| --- | --- | --- | --- | --- |
| 2011-07-29 | Oud-Heverlee Leuven | RSC Anderlecht | 2 | 1 |
| 2011-07-30 | RAEC Mons | Standard de Liège | 1 | 1 |
| 2011-07-30 | KRC Genk | Beerschot AC | 3 | 1 |
| 2011-07-30 | KAA Gent | KSV Cercle Brugge | 0 | 1 |

# Get team names with correlated subqueries

Let's solve the same problem using correlated subqueries -- How do you get both the home and away team names into one final query result?

This can easily be performed using correlated subqueries. But how might that impact the performance of your query? Complete the following steps and let's find out!

**Please note that your query will run more slowly than the previous exercise!**

* Using a correlated subquery in the SELECT statement, match the team\_api\_id column from team to the hometeam\_id from match.
* SELECT
* m.date,
* (SELECT team\_long\_name
* FROM team AS t
* -- Connect the team to the match table
* WHERE t.team\_api\_id = m.hometeam\_id) AS hometeam
* FROM match AS m;

| **date** | **hometeam** |
| --- | --- |
| 2011-07-29 | Oud-Heverlee Leuven |
| 2011-07-30 | RAEC Mons |
| 2011-07-30 | KRC Genk |

* Create a second correlated subquery in SELECT, yielding the away team's name.
* Select the home and away goal columns from match in the main query.
* SELECT
* m.date,
* (SELECT team\_long\_name
* FROM team AS t
* WHERE t.team\_api\_id = m.hometeam\_id) AS hometeam,
* -- Connect the team to the match table
* (SELECT team\_long\_name
* FROM team AS t
* WHERE t.team\_api\_id = m.awayteam\_id) AS awayteam,
* -- Select home and away goals
* home\_goal,
* away\_goal
* FROM match AS m;

| **date** | **hometeam** | **awayteam** | **home\_goal** | **away\_goal** |
| --- | --- | --- | --- | --- |
| 2011-07-29 | Oud-Heverlee Leuven | RSC Anderlecht | 2 | 1 |
| 2011-07-30 | RAEC Mons | Standard de Liège | 1 | 1 |
| 2011-07-30 | KRC Genk | Beerschot AC | 3 | 1 |
| 2011-07-30 | KAA Gent | KSV Cercle Brugge | 0 | 1 |

# Get team names with CTEs

You've now explored two methods for answering the question, How do you get both the home and away team names into one final query result?

Let's explore the final method - common table expressions. Common table expressions are similar to the subquery method for generating results, mainly differing in syntax and the order in which information is processed.

* Select id from match and team\_long\_name from team. Join these two tables together on hometeam\_id in match and team\_api\_id in team.
* SELECT
* -- Select match id and team long name
* m.id,
* t.team\_long\_name AS hometeam
* FROM match AS m
* -- Join team to match using team\_api\_id and hometeam\_id
* LEFT JOIN team AS t
* ON m.hometeam\_id = t.team\_api\_id ;

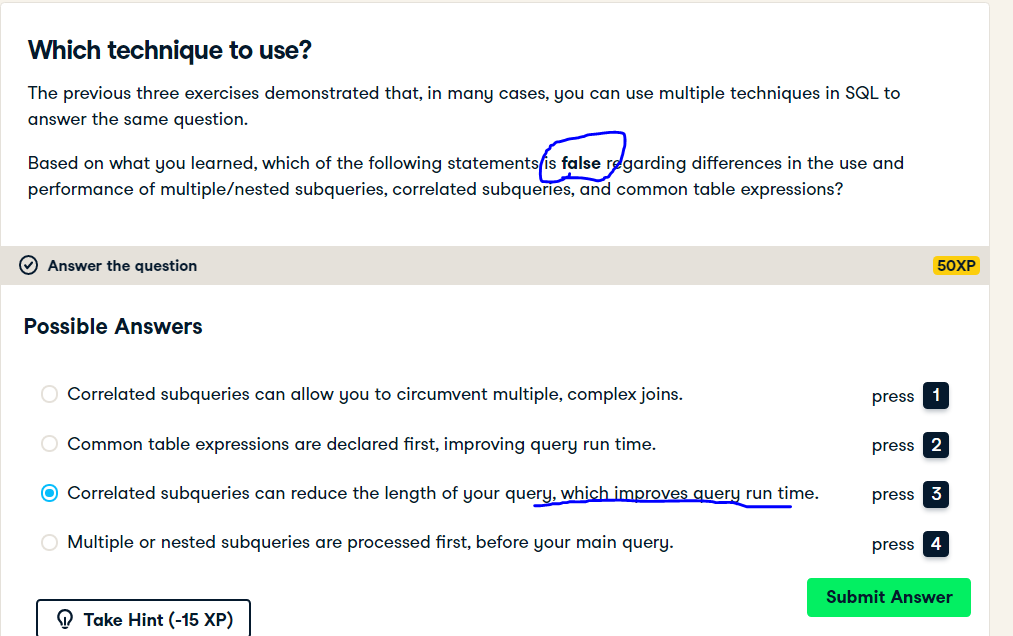
| **id** | **hometeam** |
| --- | --- |
| 757 | Oud-Heverlee Leuven |
| 758 | RAEC Mons |
| 759 | KRC Genk |

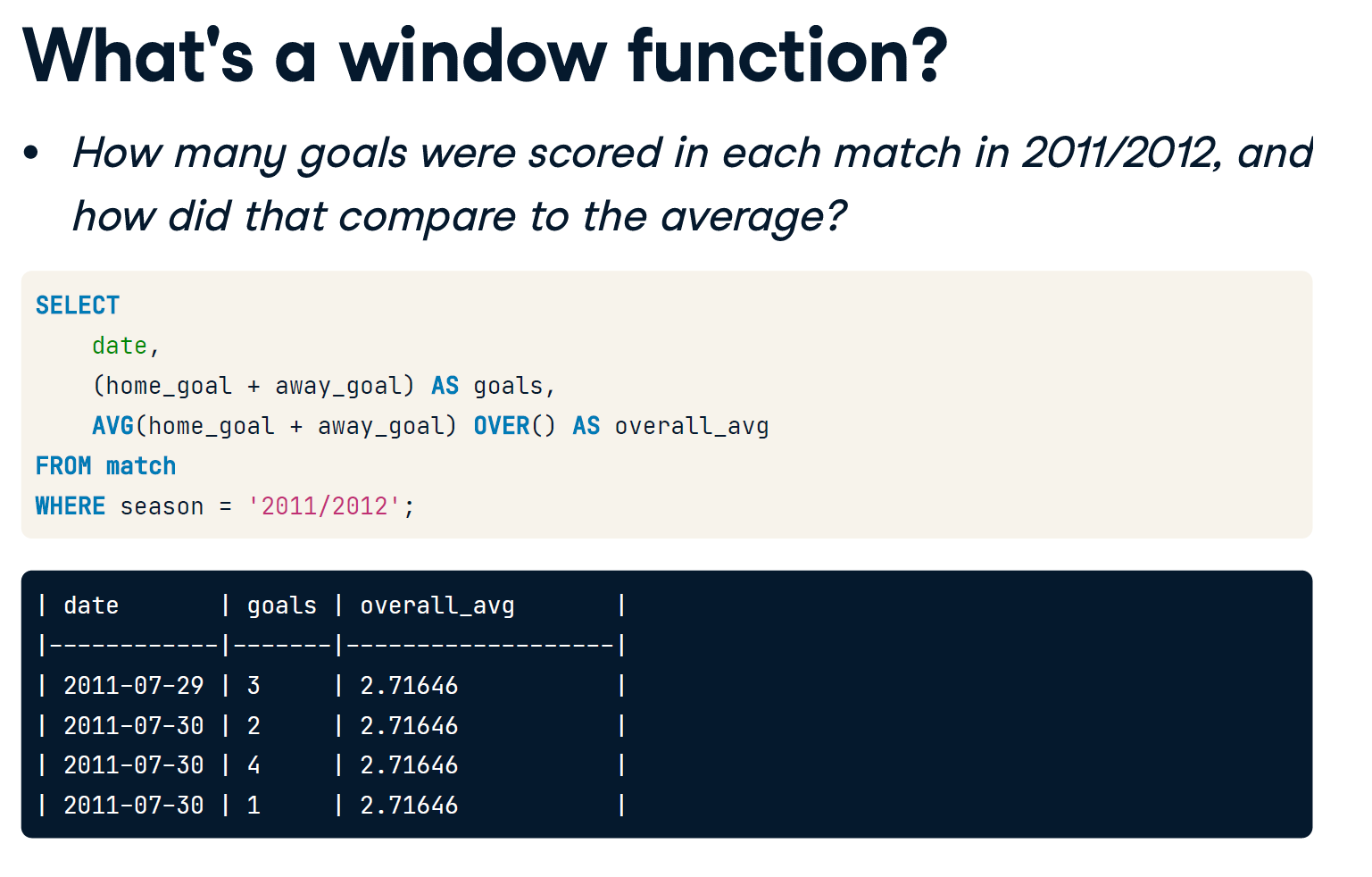
* Declare the query from the previous step as a common table expression. SELECT everything from the CTE into the main query. **Your results will not change at this step!**
* -- Declare the home CTE
* WITH home AS (
* SELECT m.id, t.team\_long\_name AS hometeam
* FROM match AS m
* LEFT JOIN team AS t
* ON m.hometeam\_id = t.team\_api\_id)
* -- Select everything from home
* SELECT \*
* FROM home;

| **id** | **hometeam** |
| --- | --- |
| 757 | Oud-Heverlee Leuven |
| 758 | RAEC Mons |
| 759 | KRC Genk |
| 760 | KAA Gent |

* Let's declare the second CTE, away. Join it to the first CTE on the id column.
* The date, home\_goal, and away\_goal columns have been added to the CTEs. SELECT them into the main query.
* WITH home AS (
* SELECT m.id, m.date,
* t.team\_long\_name AS hometeam, m.home\_goal
* FROM match AS m
* LEFT JOIN team AS t
* ON m.hometeam\_id = t.team\_api\_id),
* -- Declare and set up the away CTE
* away AS (
* SELECT m.id, m.date,
* t.team\_long\_name AS awayteam, m.away\_goal
* FROM match AS m
* LEFT JOIN team AS t
* ON m.awayteam\_id = t.team\_api\_id)
* -- Select date, home\_goal, and away\_goal
* SELECT
* home.date,
* home.hometeam,
* away.awayteam,
* home.home\_goal,
* away.away\_goal
* -- Join away and home on the id column
* FROM home
* INNER JOIN away
* ON home.id = away.id;

| **date** | **hometeam** | **awayteam** | **home\_goal** | **away\_goal** |
| --- | --- | --- | --- | --- |
| 2011-07-29 | Oud-Heverlee Leuven | RSC Anderlecht | 2 | 1 |
| 2011-07-30 | RAEC Mons | Standard de Liège | 1 | 1 |
| 2011-07-30 | KRC Genk | Beerschot AC | 3 | 1 |
| 2011-07-30 | KAA Gent | KSV Cercle Brugge | 0 | 1 |
| 2011-07-30 | Sporting Lokeren | SV Zulte-Waregem | 0 | 0 |







# The match is OVER

The OVER() clause allows you to pass an aggregate function down a data set, similar to subqueries in SELECT. The OVER() clause offers significant benefits over subqueries in select -- namely, your queries will run faster, and the OVER() clause has a wide range of additional functions and clauses you can include with it that we will cover later on in this chapter.

In this exercise, you will revise some queries from previous chapters using the OVER() clause.

* Select the match ID, country name, season, home, and away goals from the match and country tables.
* Complete the query that calculates the average number of goals scored overall and then includes the aggregate value in each row using a window function.
* SELECT
* -- Select the id, country name, season, home, and away goals
* m.id,
* c.name AS country,
* m.season,
* m.home\_goal,
* m.away\_goal,
* -- Use a window to include the aggregate average in each row
* AVG(m.home\_goal + m.away\_goal) OVER() AS overall\_avg
* FROM match AS m
* LEFT JOIN country AS c ON m.country\_id = c.id;

| **id** | **country** | **season** | **home\_goal** | **away\_goal** | **overall\_avg** |
| --- | --- | --- | --- | --- | --- |
| 1 | Belgium | 2011/2012 | 2 | 1 | 2.7321025161642128 |
| 1 | Belgium | 2011/2012 | 1 | 1 | 2.7321025161642128 |
| 1 | Belgium | 2011/2012 | 3 | 1 | 2.7321025161642128 |
| 1 | Belgium | 2011/2012 | 0 | 1 | 2.7321025161642128 |

# What's OVER here?

Window functions allow you to create a RANK of information according to any variable you want to use to sort your data. When setting this up, you will need to specify what column/calculation you want to use to calculate your rank. This is done by including an ORDER BY clause inside the OVER() clause. Below is an example:

SELECT

id,

RANK() OVER(ORDER BY home\_goal) AS rank

FROM match;

In this exercise, you will create a data set of ranked matches according to which leagues, on average, score the most goals in a match.

* Select the league name and average total goals scored from league and match.
* Complete the window function so it calculates the rank of average goals scored across all leagues in the database.
* Order the rank by the average total of home and away goals scored.
* SELECT
* -- Select the league name and average goals scored
* l.name AS league,
* AVG(m.home\_goal + m.away\_goal) AS avg\_goals,
* -- Rank each league according to the average goals
* Rank() OVER(ORDER BY AVG(m.home\_goal + m.away\_goal)) AS league\_rank
* FROM league AS l
* LEFT JOIN match AS m
* ON l.id = m.country\_id
* WHERE m.season = '2011/2012'
* GROUP BY l.name
* -- Order the query by the rank you created
* ORDER BY league\_rank;

| **league** | **avg\_goals** | **league\_rank** |
| --- | --- | --- |
| Poland Ekstraklasa | 2.1958333333333333 | 1 |
| France Ligue 1 | 2.5157894736842105 | 2 |
| Italy Serie A | 2.5837988826815642 | 3 |
| Switzerland Super League | 2.6234567901234568 | 4 |

# Flip OVER your results

In the last exercise, the rank generated in your query was organized from smallest to largest. By adding DESC to your window function, you can create a rank sorted from largest to smallest.

SELECT

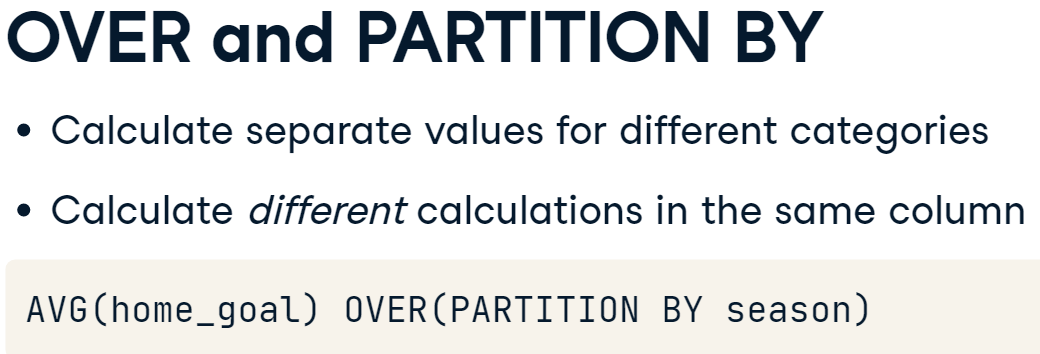
id,

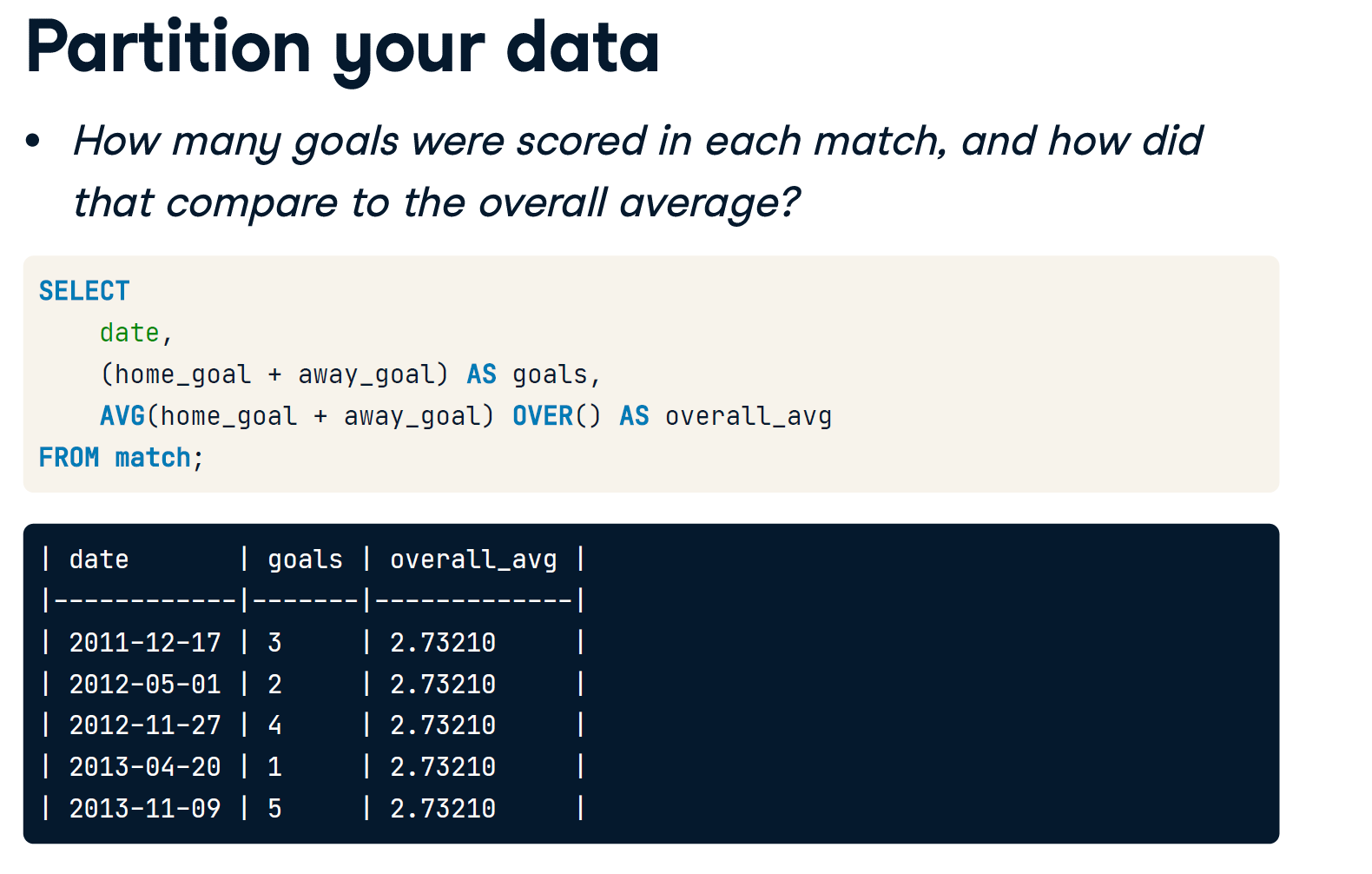
RANK() OVER(ORDER BY home\_goal DESC) AS rank

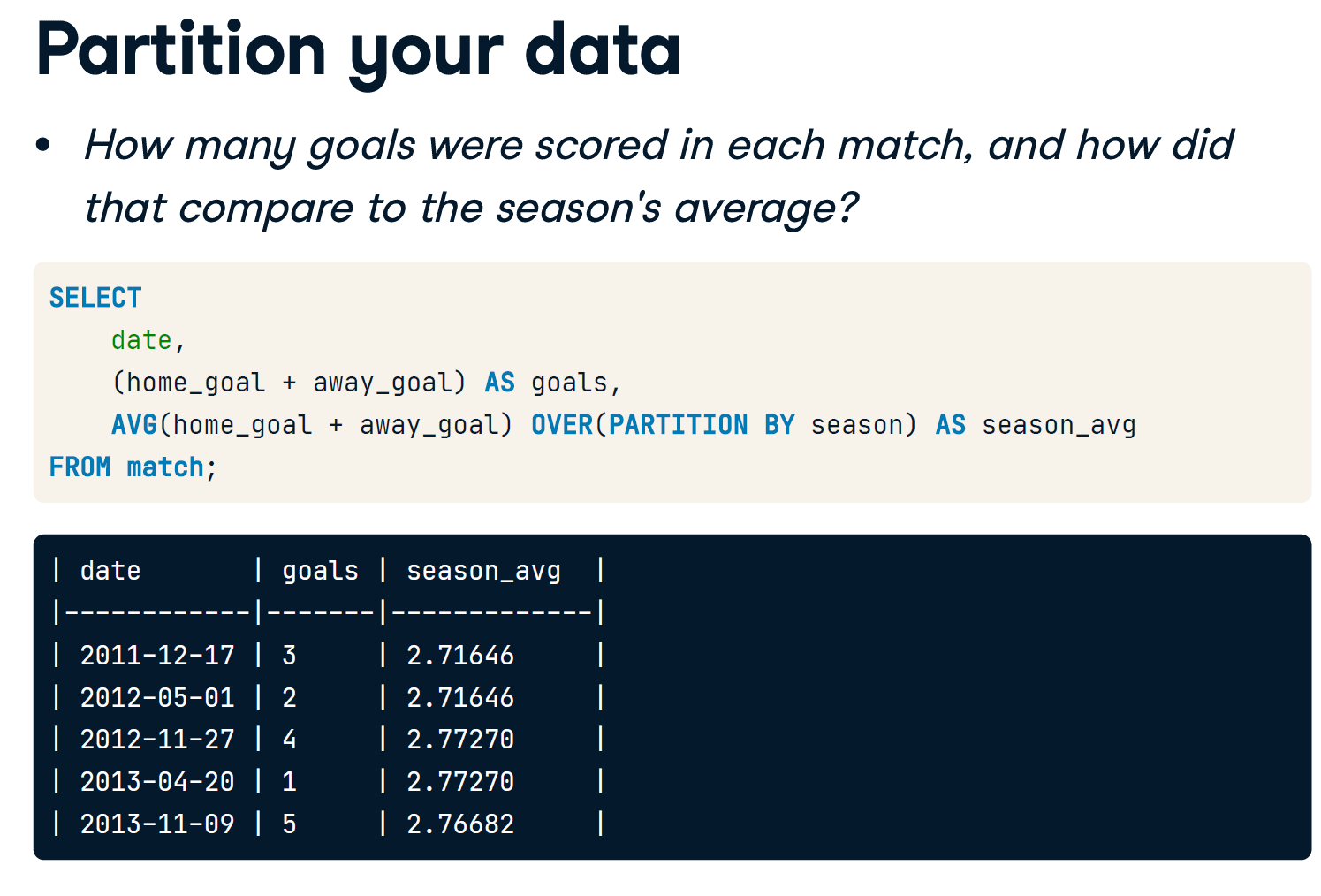
FROM match;

* Complete the same parts of the query as the previous exercise.
* Complete the window function to rank each league from highest to lowest average goals scored.
* Order the main query by the rank you just created.
* SELECT
* -- Select the league name and average goals scored
* l.name AS league,
* AVG(m.home\_goal + m.away\_goal) AS avg\_goals,
* -- Rank leagues in descending order by average goals
* RANK() OVER(ORDER BY AVG(m.home\_goal + m.away\_goal) DESC) AS league\_rank
* FROM league AS l
* LEFT JOIN match AS m
* ON l.id = m.country\_id
* WHERE m.season = '2011/2012'
* GROUP BY l.name
* -- Order the query by the rank you created
* ORDER BY league\_rank;

| **league** | **avg\_goals** | **league\_rank** |
| --- | --- | --- |
| Netherlands Eredivisie | 3.2581699346405229 | 1 |
| Belgium Jupiler League | 2.8791666666666667 | 2 |
| Germany 1. Bundesliga | 2.8594771241830065 | 3 |







# PARTITION BY a column

The PARTITION BY clause allows you to calculate separate "windows" based on columns you want to divide your results. For example, you can create a single column that calculates an overall average of goals scored for each season.

In this exercise, you will be creating a data set of games played by Legia Warszawa (Warsaw League), the top ranked team in Poland, and comparing their individual game performance to the overall average for that season.

Where do you see more outliers? Are they Legia Warszawa's home or away games?

* Complete the two window functions that calculate the home and away goal averages. Partition the window functions by season to calculate separate averages for each season.
* Filter the query to only include matches played by *Legia Warszawa*, id = 8673.

SELECT

    date,

    season,

    home\_goal,

    away\_goal,

    CASE WHEN hometeam\_id = 8673 THEN 'home'

         ELSE 'away' END AS warsaw\_location,

    -- Calculate the average goals scored partitioned by season

    AVG(home\_goal) OVER(PARTITION BY season) AS season\_homeavg,

    AVG(away\_goal) OVER(PARTITION BY season) AS season\_awayavg

FROM match

-- Filter the data set for Legia Warszawa matches only

WHERE

    hometeam\_id =   8673

    OR awayteam\_id = 8673

ORDER BY (home\_goal + away\_goal) DESC;

SELECT

    date,

    season,

    home\_goal,

    away\_goal,

    CASE WHEN hometeam\_id = 8673 THEN 'home'

         ELSE 'away' END AS warsaw\_location,

    -- Calculate the average goals scored partitioned by season

    ROUND (AVG(home\_goal) OVER(PARTITION BY season),3) AS season\_homeavg,

    ROUND( AVG(away\_goal) OVER(PARTITION BY season),3) AS season\_awayavg

FROM match

-- Filter the data set for Legia Warszawa matches only

WHERE

    hometeam\_id =   8673

    OR awayteam\_id = 8673

ORDER BY (home\_goal + away\_goal) DESC;

| **date** | **season** | **home\_goal** | **away\_goal** | **warsaw\_location** | **season\_homeavg** | **season\_awayavg** |
| --- | --- | --- | --- | --- | --- | --- |
| 2013-09-14 | 2013/2014 | 3 | 5 | away | 1.767 | 1.233 |
| 2014-09-13 | 2014/2015 | 4 | 3 | home | 1.567 | 1.333 |
| 2013-07-20 | 2013/2014 | 5 | 1 | home | 1.767 | 1.233 |
| 2013-10-20 | 2013/2014 | 4 | 1 | home | 1.767 | 1.233 |

# PARTITION BY multiple columns

The PARTITION BY clause can be used to break out window averages by multiple data points (columns). You can even calculate the information you want to use to partition your data! For example, you can calculate average goals scored by season and by country, or by the calendar year (taken from the date column).

In this exercise, you will calculate the average number home and away goals scored Legia Warszawa, and their opponents, partitioned by the month in each season.

* Construct two window functions partitioning the average of home and away goals by season and month.
* Filter the data set by *Legia Warszawa*'s team ID (8673) so that the window calculation excludes all teams who did not play against them.

SELECT

    date,

    season,

    home\_goal,

    away\_goal,

    CASE WHEN hometeam\_id = 8673 THEN 'home'

         ELSE 'away' END AS warsaw\_location,

    -- Calculate average goals partitioned by season and month

    AVG(home\_goal) OVER(PARTITION BY season,

            EXTRACT(month FROM date)) AS season\_mo\_home,

    AVG(away\_goal) OVER(PARTITION BY season,

            EXTRACT(month FROM date)) AS season\_mo\_away

FROM match

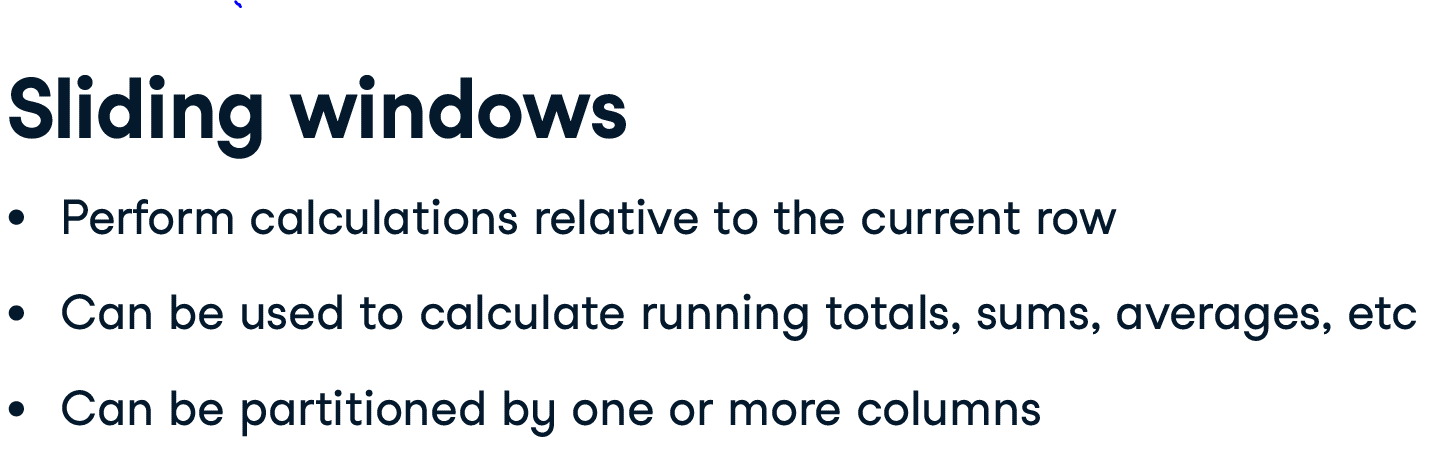
WHERE

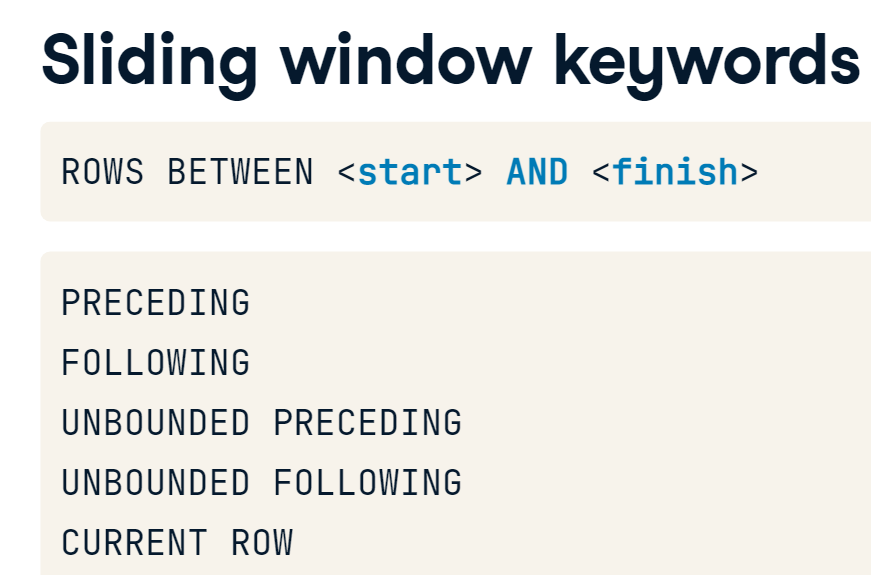
    hometeam\_id = 8673

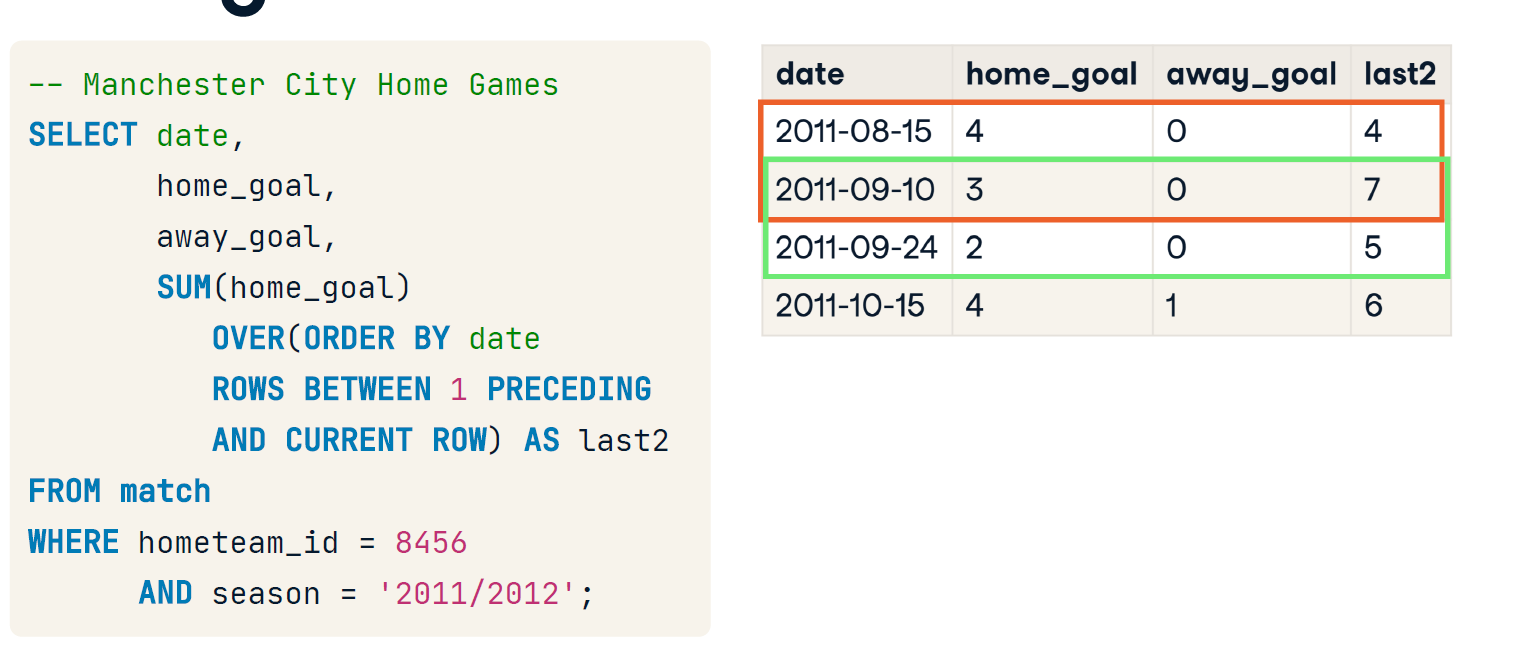
    OR awayteam\_id = 8673

ORDER BY (home\_goal + away\_goal) DESC;

| **date** | **season** | **home\_goal** | **away\_goal** | **warsaw\_location** | **season\_mo\_home** | **season\_mo\_away** |
| --- | --- | --- | --- | --- | --- | --- |
| 2013-09-14 | 2013/2014 | 3 | 5 | away | 2.250 | 2.50 |
| 2014-09-13 | 2014/2015 | 4 | 3 | home | 2.000 | 2.66 |
| 2013-07-20 | 2013/2014 | 5 | 1 | home | 2.500 | 2.00 |









# Slide to the left

Sliding windows allow you to create running calculations between any two points in a window using functions such as PRECEDING, FOLLOWING, and CURRENT ROW. You can calculate running counts, sums, averages, and other aggregate functions between any two points you specify in the data set.

In this exercise, you will expand on the examples discussed in the video, calculating the running total of goals scored by the FC Utrecht when they were the home team during the 2011/2012 season. Do they score more goals at the end of the season as the home or away team?

* Complete the window function by:
  + Assessing the running *total* of home goals scored by *FC Utrecht*.
  + Assessing the running *average* of home goals scored.
  + Ordering both the running average and running total by date.

SELECT

     date,

     home\_goal,

     away\_goal,

    -- Create a running total and running average of home goals

    SUM(home\_goal) OVER(ORDER BY date

         ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW) AS running\_total,

    AVG(home\_goal) OVER(ORDER BY date

         ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW) AS running\_avg

FROM match

WHERE

     hometeam\_id = 9908

     AND season = '2011/2012';

| **date** | **home\_goal** | **away\_goal** | **running\_total** | **running\_avg** |
| --- | --- | --- | --- | --- |
| 2011-08-14 | 2 | 2 | 2 | 2.0000000000000000 |
| 2011-08-27 | 3 | 1 | 5 | 2.5000000000000000 |
| 2011-09-18 | 2 | 2 | 7 | 2.3333333333333333 |

# Slide to the right

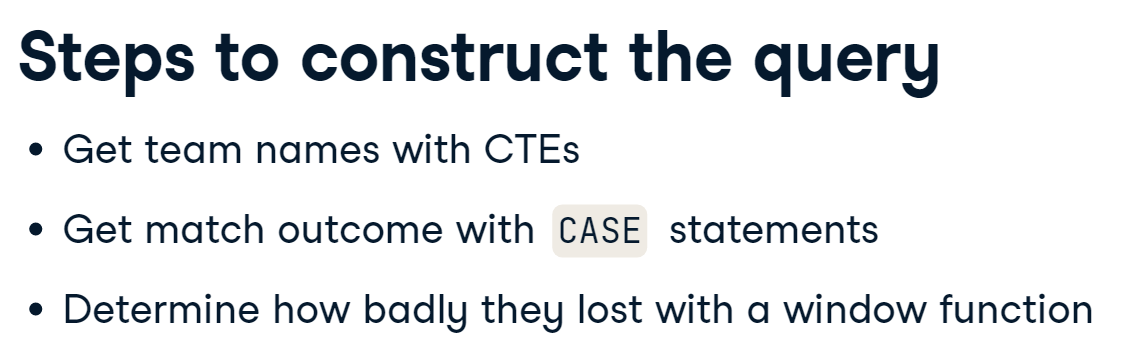
Now let's see how FC Utrecht performs when they're the away team. You'll notice that the total for the season is at the bottom of the data set you queried. Depending on your results, this could be pretty long, and scrolling down is not very helpful.

In this exercise, you will slightly modify the query from the previous exercise by sorting the data set in reverse order and calculating a backward running total **from the** CURRENT ROW **to the end of the data set** (earliest record).

* Complete the window function by:
  + Assessing the running *total* of home goals scored by *FC Utrecht*.
  + Assessing the running *average* of home goals scored.
  + Ordering both the running average and running total by date, *descending*.
* SELECT
* -- Select the date, home goal, and away goals
* date,
* home\_goal,
* away\_goal,
* -- Create a running total and running average of home goals
* SUM(home\_goal) OVER(ORDER BY date DESC
* ROWS BETWEEN CURRENT ROW AND  UNBOUNDED FOLLOWING  ) AS running\_total,
* AVG(home\_goal) OVER(ORDER BY date DESC
* ROWS BETWEEN  CURRENT ROW  AND UNBOUNDED FOLLOWING) AS running\_avg
* FROM match
* WHERE
* awayteam\_id = 9908
* AND season = '2011/2012';

| **date** | **home\_goal** | **away\_goal** | **running\_total** | **running\_avg** |
| --- | --- | --- | --- | --- |
| 2012-05-06 | 1 | 3 | 25 | 1.4705882352941176 |
| 2012-04-21 | 0 | 2 | 24 | 1.5000000000000000 |
| 2012-04-12 | 3 | 0 | 24 | 1.6000000000000000 |
| 2012-03-25 | 3 | 1 | 21 | 1.5000000000000000 |





# Setting up the home team CTE

In this course, we've covered ways in which you can use CASE statements, subqueries, common table expressions, and window functions in your queries to structure a data set that best meets your needs. For this exercise, you will be using all of these concepts to generate a list of matches in which Manchester United was defeated during the 2014/2015 English Premier League season.

Your first task is to create the first query that filters for matches where Manchester United played as the home team. This will become a common table expression in a later exercise.

* Create a CASE statement that identifies each match as a win, lose, or tie for Manchester United.
* Fill out the logical operators for each WHEN clause in the CASE statement (equals, greater than, less than).
* Join the tables on home team ID from match, and team\_api\_id from team.
* Filter the query to only include games from the 2014/2015 season where Manchester United was the home team.
* SELECT
* m.id,
* t.team\_long\_name,
* -- Identify matches as home/away wins or ties
* CASE WHEN m.home\_goal > m.away\_goal THEN 'MU Win'
* WHEN m.home\_goal < m.away\_goal THEN 'MU Loss'
* ELSE 'Tie' END
* AS outcome
* FROM match AS m
* -- Left join team on the home team ID and team API id
* LEFT JOIN team AS t
* ON m.hometeam\_id = t.team\_api\_id
* WHERE
* -- Filter for 2014/2015 and Manchester United as the home team
* m.season =  '2014/2015'
* AND t.team\_long\_name = 'Manchester United';

| **id** | **team\_long\_name** | **outcome** |
| --- | --- | --- |
| 4013 | Manchester United | MU Loss |
| 4031 | Manchester United | MU Win |
| 4051 | Manchester United | MU Win |
| 4062 | Manchester United | MU Win |

# Setting up the away team CTE

Great job! Now that you have a query identifying the home team in a match, you will perform a similar set of steps to identify the away team. Just like the previous step, you will join the match and team tables. Each of these two queries will be declared as a Common Table Expression in the following step.

The primary difference in this query is that you will be joining the tables on awayteam\_id, and reversing the match outcomes in the CASE statement.

When altering CASE statement logic in your own work, you can reverse either the logical condition (i.e., home\_goal > away\_goal) or the outcome in THEN -- just make sure you only reverse one of the two!

* Complete the CASE statement syntax.
* Fill out the logical operators identifying each match as a win, loss, or tie for Manchester United.
* Join the table on awayteam\_id, and team\_api\_id.
* SELECT
* m.id,
* t.team\_long\_name,
* -- Identify matches as home/away wins or ties
* CASE WHEN m.home\_goal  > m.away\_goal THEN 'MU Loss'
* WHEN m.home\_goal  < m.away\_goal THEN 'MU Win'
* ELSE 'Tie' END AS outcome
* -- Join team table to the match table
* FROM match AS m
* LEFT JOIN team AS t
* ON awayteam\_id = t.team\_api\_id
* WHERE
* -- Filter for 2014/2015 and Manchester United as the away team
* m.season  = '2014/2015'
* AND t.team\_long\_name = 'Manchester United';

| **id** | **team\_long\_name** | **outcome** |
| --- | --- | --- |
| 4026 | Manchester United | MU Loss |
| 4039 | Manchester United | Tie |
| 4075 | Manchester United | Tie |
| 4089 | Manchester United | Tie |

# Putting the CTEs together

Now that you've created the two subqueries identifying the home and away team opponents, it's time to rearrange your query with the home and away subqueries as Common Table Expressions (CTEs). You'll notice that the main query includes the phrase, SELECT DISTINCT. Without identifying only DISTINCT matches, you will return a duplicate record for each game played.

Continue building the query to extract all matches played by Manchester United in the 2014/2015 season.

* Declare the home and away CTEs before your main query.
* Join your CTEs to the match table using a LEFT JOIN.
* Select the relevant data from the CTEs into the main query.
* Select the date from match, team names from the CTEs, and home/ away goals from match in the main query.
* -- Set up the home team CTE
* WITH home AS  (
* SELECT m.id, t.team\_long\_name,
* CASE WHEN m.home\_goal > m.away\_goal THEN 'MU Win'
* WHEN m.home\_goal < m.away\_goal THEN 'MU Loss'
* ELSE 'Tie' END AS outcome
* FROM match AS m
* LEFT JOIN team AS t ON m.hometeam\_id = t.team\_api\_id),
* -- Set up the away team CTE
* away AS (
* SELECT m.id, t.team\_long\_name,
* CASE WHEN m.home\_goal > m.away\_goal THEN 'MU Win'
* WHEN m.home\_goal < m.away\_goal THEN 'MU Loss'
* ELSE 'Tie' END AS outcome
* FROM match AS m
* LEFT JOIN team AS t ON m.awayteam\_id = t.team\_api\_id)
* -- Select team names, the date and goals
* SELECT DISTINCT
* m.date,
* home.team\_long\_name AS home\_team,
* away.team\_long\_name AS away\_team,
* m.home\_goal,
* m.away\_goal
* -- Join the CTEs onto the match table
* FROM match AS m
* LEFT JOIN home ON m.id = home.id
* LEFT JOIN away ON m.id = away.id
* WHERE m.season = '2014/2015'
* AND (home.team\_long\_name = 'Manchester United'
* OR away.team\_long\_name = 'Manchester United');

| **date** | **home\_team** | **away\_team** | **home\_goal** | **away\_goal** |
| --- | --- | --- | --- | --- |
| 2014-08-16 | Manchester United | Swansea City | 1 | 2 |
| 2014-08-24 | Sunderland | Manchester United | 1 | 1 |
| 2014-08-30 | Burnley | Manchester United | 0 | 0 |
| 2014-09-14 | Manchester United | Queens Park Rangers | 4 | 0 |

# Add a window function

Fantastic! You now have a result set that retrieves the match date, home team, away team, and the goals scored by each team. You have one final component of the question left -- how badly did Manchester United lose in each match?

In order to determine this, let's add a window function to the main query that ranks matches by the absolute value of the difference between home\_goal and away\_goal. This allows us to directly compare the difference in scores without having to consider whether Manchester United played as the home or away team!

The equation is complete for you -- all you need to do is properly complete the window function!

* Set up the CTEs so that the home and away teams each have a name, ID, and score associated with them.
* Select the date, home team name, away team name, home goal, and away goals scored in the main query.
* Rank the matches and order by the difference in scores in *descending* order.

-- Set up the home team CTE

WITH home AS  (

  SELECT m.id, t.team\_long\_name,

    CASE WHEN m.home\_goal > m.away\_goal THEN 'MU Win'

       WHEN m.home\_goal < m.away\_goal THEN 'MU Loss'

         ELSE 'Tie' END AS outcome

  FROM match AS m

  LEFT JOIN team AS t ON m.hometeam\_id = t.team\_api\_id),

-- Set up the away team CTE

away AS (

  SELECT m.id, t.team\_long\_name,

    CASE WHEN m.home\_goal > m.away\_goal THEN 'MU Loss'

       WHEN m.home\_goal < m.away\_goal THEN 'MU Win'

         ELSE 'Tie' END AS outcome

  FROM match AS m

  LEFT JOIN team AS t ON m.awayteam\_id = t.team\_api\_id)

-- Select columns and and rank the matches by date

SELECT DISTINCT

    m.date,

    home.team\_long\_name AS home\_team,

    away.team\_long\_name  AS away\_team,

    m.home\_goal, m.away\_goal,

    RANK() OVER(ORDER BY ABS(home\_goal - away\_goal) DESC) as match\_rank

-- Join the CTEs onto the match table

FROM match AS m

LEFT JOIN home ON m.id = home.id

LEFT JOIN away ON m.id = away.id

WHERE m.season = '2014/2015'

      AND ((home.team\_long\_name = 'Manchester United' AND home.outcome = 'MU Loss')

      OR (away.team\_long\_name = 'Manchester United' AND away.outcome = 'MU Loss'));

| **date** | **home\_team** | **away\_team** | **home\_goal** | **away\_goal** | **match\_rank** |
| --- | --- | --- | --- | --- | --- |
| 2014-08-16 | Manchester United | Swansea City | 1 | 2 | 3 |
| 2014-09-21 | Leicester City | Manchester United | 5 | 3 | 2 |
| 2014-11-02 | Manchester City | Manchester United | 1 | 0 | 3 |
| 2015-01-11 | Manchester United | Southampton | 0 | 1 | 3 |