**Basic CASE statements**

What is your favorite team?

The *European Soccer Database* contains data about 12,800 matches from 11 countries played between 2011-2015! Throughout this course, you will be shown filtered versions of the tables in this database in order to better explore their contents.

In this exercise, you will identify matches played between *FC Schalke 04* and *FC Bayern Munich*. There are 2 teams identified in each match in the hometeam\_id and awayteam\_id columns, available to you in the filtered matches\_germany table. ID can join to the team\_api\_id column in the teams\_germany table, but you cannot perform a join on both at the same time.

However, you can perform this operation using a CASE statement once you've identified the team\_api\_id associated with each team!

* Select the team's long name and API id from the teams\_germany table.
* Filter the query for *FC Schalke 04* and *FC Bayern Munich* using IN, giving you the team\_api\_IDs needed for the next step.
* SELECT
* -- Select the team long name and team API id
* team\_api\_id,
* team\_long\_name
* FROM teams\_germany
* -- Only include FC Schalke 04 and FC Bayern Munich
* WHERE team\_long\_name IN  ('FC Schalke 04','FC Bayern Munich');

| **team\_api\_id** | **team\_long\_name** |
| --- | --- |
| 9823 | FC Bayern Munich |
| 10189 | FC Schalke 04 |

* Create a CASE statement that identifies whether a match in Germany included FC Bayern Munich, FC Schalke 04, or neither as the home team.
* Group the query by the CASE statement alias, home\_team.
* -- Identify the home team as Bayern Munich, Schalke 04, or neither
* SELECT
* CASE WHEN hometeam\_id = 10189 THEN 'FC Schalke 04'
* WHEN hometeam\_id = 9823 THEN 'FC Bayern Munich'
* ELSE 'Other' END AS home\_team,
* COUNT(id) AS total\_matches
* FROM matches\_germany
* -- Group by the CASE statement alias
* GROUP BY home\_team;

| **home\_team** | **total\_matches** |
| --- | --- |
| FC Bayern Munich | 68 |
| Other | 1088 |
| FC Schalke 04 | 68 |
|  |  |

# CASE statements comparing column values

Barcelona is considered one of the strongest teams in Spain's soccer league.

In this exercise, you will be creating a list of matches in the 2011/2012 season where Barcelona was the home team. You will do this using a CASE statement that compares the values of two columns to create a new group -- wins, losses, and ties.

In 3 steps, you will build a query that identifies a match's winner, identifies the identity of the opponent, and finally filters for Barcelona as the home team. Completing a query in this order will allow you to watch your results take shape with each new piece of information.

The matches\_spain table currently contains Barcelona's matches from the 2011/2012 season, and has two key columns, hometeam\_id and awayteam\_id, that can be joined with the teams\_spain table. However, you can only join teams\_spain to one column at a time.

* Select the date of the match and create a CASE statement to identify matches as home wins, home losses, or ties.
* SELECT
* -- Select the date of the match
* date,
* -- Identify home wins, losses, or ties
* CASE WHEN home\_goal > away\_goal THEN 'Home win!'
* WHEN home\_goal < away\_goal THEN 'Home loss :('
* ELSE 'Tie' END as outcome
* FROM matches\_spain;
* Left join the teams\_spain table team\_api\_id column to the matches\_spain table awayteam\_id. This allows us to retrieve the *away* team's identity.
* Select team\_long\_name from teams\_spain as opponent and complete the CASE statement from Step 1.
* SELECT
* m.date,
* --Select the team long name column and call it 'opponent'
* t.team\_long\_name AS opponent,
* -- Complete the CASE statement with an alias
* CASE WHEN m.home\_goal > away\_goal THEN  'Home win!'
* WHEN m.home\_goal < away\_goal THEN 'Home loss :('
* ELSE 'Tie' END as  outcome
* FROM matches\_spain AS m
* -- Left join teams\_spain onto matches\_spain
* LEFT JOIN teams\_spain AS t
* ON m.awayteam\_id = t.team\_api\_id;

| **date** | **opponent** | **outcome** |
| --- | --- | --- |
| 2011-10-15 | Valencia CF | Tie |
| 2011-09-24 | Valencia CF | Home win! |
| 2011-09-17 | Valencia CF | Home loss :( |
| 2012-05-12 | Valencia CF | Home win! |

* Complete the same CASE statement as the previous steps.
* Filter for matches where the home team is *FC Barcelona* (id = 8634).
* SELECT
* m.date,
* t.team\_long\_name AS opponent,
* -- Complete the CASE statement with an alias
* CASE WHEN m.home\_goal > away\_goal THEN 'Barcelona win!'
* WHEN m.home\_goal < away\_goal THEN 'Barcelona loss :('
* ELSE 'Tie' END as outcome
* FROM matches\_spain AS m
* LEFT JOIN teams\_spain AS t
* ON m.awayteam\_id = t.team\_api\_id
* -- Filter for Barcelona as the home team
* WHERE m.hometeam\_id = 8634 ;

| **ate** | **opponent** | **outcome** |
| --- | --- | --- |
| 2011-10-29 | RCD Mallorca | Barcelona win! |
| 2011-11-19 | Real Zaragoza | Barcelona win! |
| 2011-12-03 | Levante UD | Barcelona win! |
| 2011-11-29 | Rayo Vallecano | Barcelona win! |

# CASE statements comparing two column values part 2

Similar to the previous exercise, you will construct a query to determine the outcome of Barcelona's matches where they played as the ***away team***. You will learn how to combine these two queries in chapters 2 and 3.

Did their performance differ from the matches where they were the home team?

* Complete the CASE statement to identify Barcelona's away team games (id = 8634) as wins, losses, or ties.
* Left join the teams\_spain table team\_api\_id column on the matches\_spain table hometeam\_id column. This retrieves the identity of the *home team* opponent.
* Filter the query to only include matches where Barcelona was the *away* team.

-- Select matches where Barcelona was the away team

SELECT

    m.date,

    t.team\_long\_name AS opponent,

    CASE WHEN m.home\_goal < away\_goal THEN 'Barcelona win!'

        WHEN m.home\_goal > away\_goal THEN 'Barcelona loss :('

        ELSE 'Tie' END as outcome

FROM matches\_spain AS m

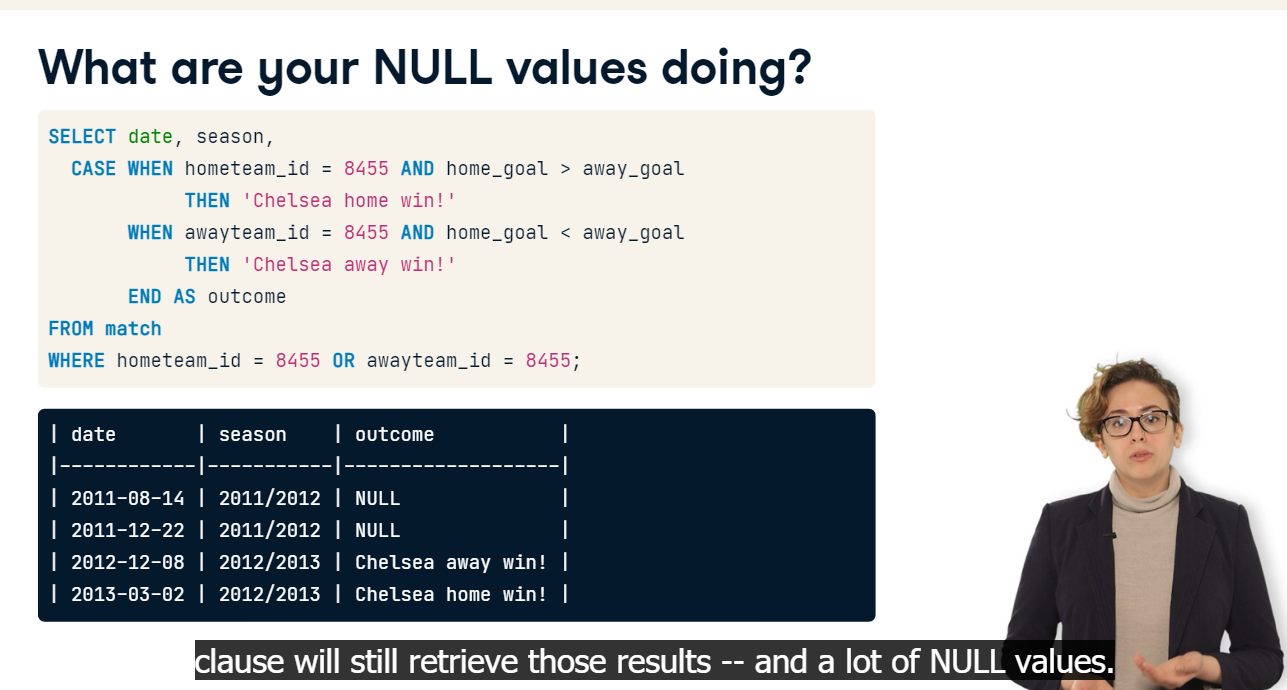
-- Join teams\_spain to matches\_spain

LEFT JOIN teams\_spain AS t

ON m.hometeam\_id = t.team\_api\_id

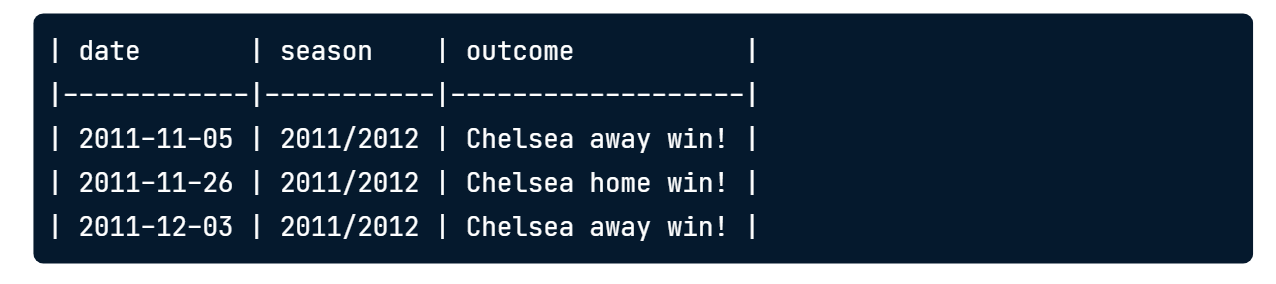
WHERE m.awayteam\_id = 8634;

| **date** | **opponent** | **outcome** |
| --- | --- | --- |
| 2012-01-22 | Málaga CF | Barcelona win! |
| 2011-10-25 | Granada CF | Barcelona win! |
| 2011-11-06 | Athletic Club de Bilbao | Tie |
| 2011-11-26 | Getafe CF | Barcelona loss :( |









# In CASE of rivalry

Barcelona and Real Madrid have been rival teams for more than 80 years. Matches between these two teams are given the name El Clásico (The Classic). In this exercise, you will query a list of matches played between these two rivals.

You will notice in Step 2 that when you have multiple logical conditions in a CASE statement, you may quickly end up with a large number of WHEN clauses to logically test every outcome you are interested in. It's important to make sure you don't accidentally exclude key information in your ELSE clause.

In this exercise, you will retrieve information about matches played between **Barcelona** (id = 8634) and **Real Madrid** (id = 8633). Note that the query you are provided with already identifies the Clásico matches using a filter in the WHERE clause.

* Complete the first CASE statement, identifying Barcelona or Real Madrid as the home team using the hometeam\_id column.
* Complete the second CASE statement in the same way, using awayteam\_id.

SELECT

        date,

        -- Identify the home team as Barcelona or Real Madrid

        CASE WHEN hometeam\_id = 8634 THEN 'FC Barcelona'

        ELSE 'Real Madrid CF' END AS home,

    -- Identify the away team as Barcelona or Real Madrid

        CASE WHEN awayteam\_id = 8634 THEN 'FC Barcelona'

        ELSE 'Real Madrid CF' END AS away

FROM matches\_spain

WHERE (awayteam\_id = 8634 OR hometeam\_id = 8634)

      AND (awayteam\_id = 8633 OR hometeam\_id = 8633);

| **date** | **home** | **away** |
| --- | --- | --- |
| 2011-12-10 | Real Madrid CF | FC Barcelona |
| 2012-04-21 | FC Barcelona | Real Madrid CF |
| 2013-03-02 | Real Madrid CF | FC Barcelona |

* Construct the final CASE statement identifying who won each match. Note there are 3 possible outcomes, but 5 conditions that you need to identify.
* Fill in the logical operators to identify Barcelona or Real Madrid as the winner.
* SELECT
* date,
* CASE WHEN hometeam\_id = 8634 THEN 'FC Barcelona'
* ELSE 'Real Madrid CF' END as home,
* CASE WHEN awayteam\_id = 8634 THEN 'FC Barcelona'
* ELSE 'Real Madrid CF' END as away,
* -- Identify all possible match outcomes
* CASE WHEN home\_goal > away\_goal AND hometeam\_id = 8634 THEN 'Barcelona win!'
* WHEN home\_goal > away\_goal AND hometeam\_id = 8633 THEN 'Real Madrid win!'
* WHEN home\_goal < away\_goal AND awayteam\_id = 8634 THEN  'Barcelona win!'
* WHEN home\_goal < away\_goal AND awayteam\_id = 8633 THEN  'Real Madrid win!'
* ELSE 'Tie!' END AS outcome
* FROM matches\_spain
* WHERE (awayteam\_id = 8634 OR hometeam\_id = 8634)
* AND (awayteam\_id = 8633 OR hometeam\_id = 8633);

| **date** | **home** | **away** | **outcome** |
| --- | --- | --- | --- |
| 2011-12-10 | Real Madrid CF | FC Barcelona | Barcelona win! |
| 2012-04-21 | FC Barcelona | Real Madrid CF | Real Madrid win! |
| 2013-03-02 | Real Madrid CF | FC Barcelona | Real Madrid win! |
| 2012-10-07 | FC Barcelona | Real Madrid CF | Tie! |

# Filtering your CASE statement

Let's generate a list of matches won by Italy's Bologna team! There are quite a few additional teams in the two tables, so a key part of generating a usable query will be using your CASE statement as a filter in the WHERE clause.

CASE statements allow you to categorize data that you're interested in -- and exclude data you're not interested in. In order to do this, you can use a CASE statement as a filter in the WHERE statement to remove output you don't want to see.

Here is how you might set that up:

SELECT \*

FROM table

WHERE

CASE WHEN a > 5 THEN 'Keep'

WHEN a <= 5 THEN 'Exclude' END = 'Keep';

In essence, you can use the CASE statement as a filtering column like any other column in your database. The only difference is that you don't alias the statement in WHERE.

* Identify Bologna's team ID listed in the teams\_italy table by selecting the team\_long\_name and team\_api\_id.
* -- Select team\_long\_name and team\_api\_id from team
* SELECT
* team\_long\_name  ,
* team\_api\_id
* FROM teams\_italy
* -- Filter for team long name
* WHERE team\_long\_name     = 'Bologna';

| **eam\_long\_name** | **team\_api\_id** |
| --- | --- |
| Bologna | 9857 |

* Select the season and date that a match was played.
* Complete the CASE statement so that only Bologna's home and away wins are identified.

-- Select the season and date columns

SELECT

    season,

    date,

    -- Identify when Bologna won a match

    CASE WHEN hometeam\_id = 9857 AND home\_goal > away\_goal THEN 'Bologna Win'

         WHEN awayteam\_id = 9857 AND away\_goal > home\_goal THEN 'Bologna Win'

         END AS outcome

FROM matches\_italy;

| **eason** | **date** | **outcome** |
| --- | --- | --- |
| 2011/2012 | 2011-12-21 | null |
| 2011/2012 | 2011-12-21 | null |
| 2011/2012 | 2011-12-20 | null |

* Select the home\_goal and away\_goal for each match.
* Use the CASE statement in the WHERE clause to filter all NULL values generated by the statement in the previous step.

-- Select the season, date, home\_goal, and away\_goal columns

SELECT

    season,

    date,

    home\_goal,

    away\_goal

FROM matches\_italy

WHERE

-- Exclude games not won by Bologna

    CASE WHEN hometeam\_id = 9857 AND home\_goal > away\_goal THEN 'Bologna Win'

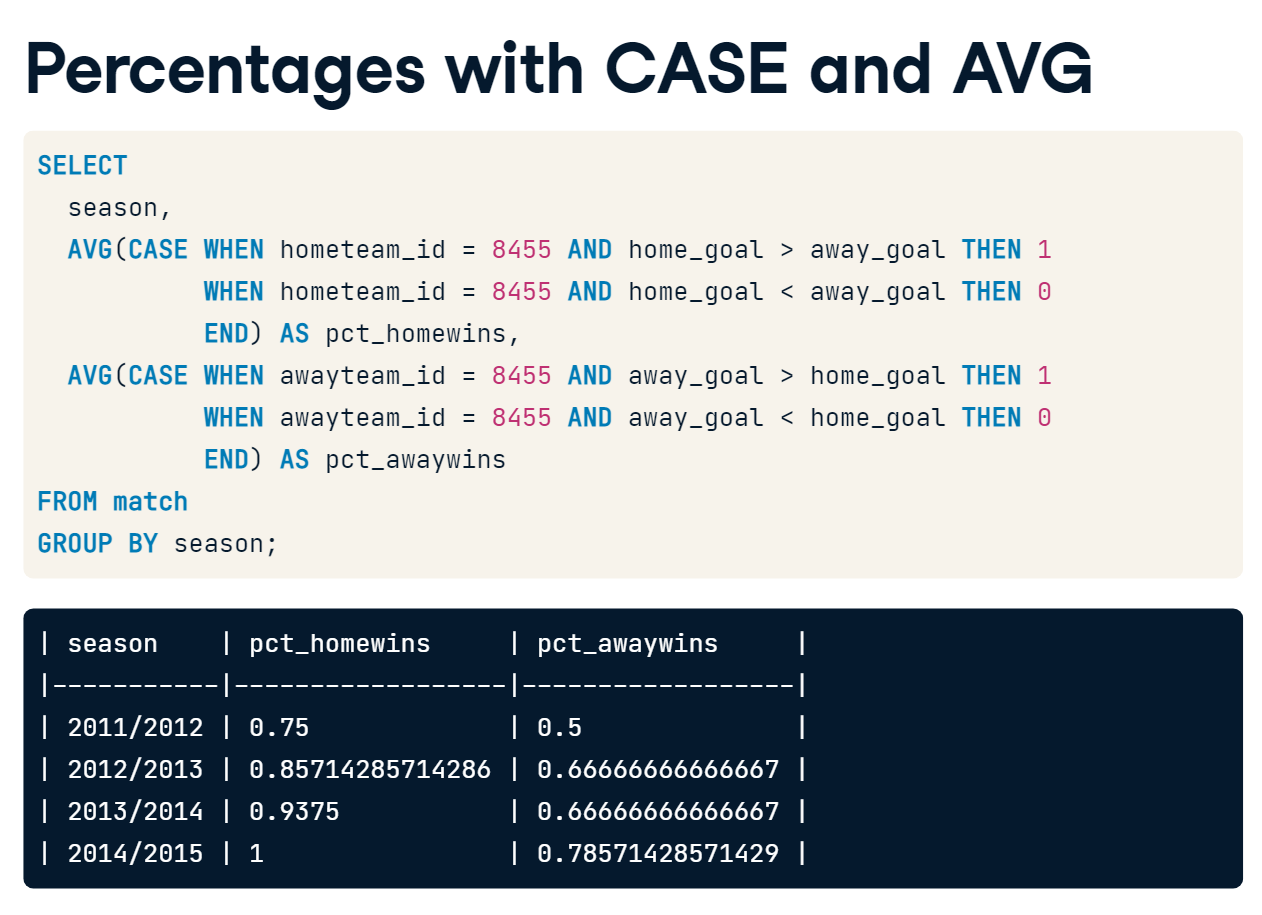
        WHEN awayteam\_id = 9857 AND away\_goal > home\_goal THEN 'Bologna Win'

        END IS NOT NULL;

| **eason** | **date** | **home\_goal** | **away\_goal** |
| --- | --- | --- | --- |
| 2011/2012 | 2011-10-30 | 3 | 1 |
| 2011/2012 | 2011-12-04 | 1 | 0 |
|  |  |  |  |

* Select the home\_goal and away\_goal for each match.
* Use the CASE statement in the WHERE clause to filter all NULL values generated by the statement in the previous step.
* -- Select the season, date, home\_goal, and away\_goal columns
* SELECT
* season,
* date,
* home\_goal,
* away\_goal
* FROM matches\_italy
* WHERE
* -- Exclude games not won by Bologna
* CASE WHEN hometeam\_id = 9857 AND home\_goal > away\_goal THEN 'Bologna Win'
* WHEN awayteam\_id = 9857 AND away\_goal > home\_goal THEN 'Bologna Win'
* END IS NOT NULL ;

| **season** | **date** | **home\_goal** | **away\_goal** |
| --- | --- | --- | --- |
| 2011/2012 | 2011-10-30 | 3 | 1 |
| 2011/2012 | 2011-12-04 | 1 | 0 |
| 2011/2012 | 2012-01-08 | 2 | 0 |
|  |  |  |  |
|  |  |  |  |



# COUNT using CASE WHEN

Do the number of soccer matches played in a given European country differ across seasons? We will use the European Soccer Database to answer this question.

You will examine the number of matches played in 3 seasons within each country listed in the database. This is much easier to explore with each season's matches in separate columns. Using the country and unfiltered match table, you will count the number of matches played in each country during the 2012/2013, 2013/2014, and 2014/2015 match seasons.

##### Instructions 1/2

**50 XP**

* [1](javascript:void(0))
* [2](javascript:void(0))
* Create a CASE statement that identifies the id of matches played in the 2012/2013 season. Specify that you want ELSE values to be NULL.
* Wrap the CASE state
* SELECT
* c.name AS country,
* -- Count games from the 2012/2013 season
* COUNT(CASE WHEN m.season = '2012/2013'
* THEN m.id ELSE NULL END) AS matches\_2012\_2013
* FROM country AS c
* LEFT JOIN match AS m
* ON c.id = m.country\_id
* -- Group by country name alias
* GROUP BY c.name;

| **country** | **matches\_2012\_2013** |
| --- | --- |
| Portugal | 240 |
| France | 380 |
| Scotland | 228 |

* Create 3 CASE WHEN statements counting the matches played in each country across the 3 seasons.
* END your CASE statement without an ELSE clause.

SELECT

    c.name AS country,

    -- Count matches in each of the 3 seasons

    COUNT(CASE WHEN m.season = '2012/2013' THEN m.id END) AS matches\_2012\_2013,

    COUNT(CASE WHEN m.season = '2013/2014' THEN m.id END) AS matches\_2013\_2014,

    COUNT(CASE WHEN m.season = '2014/2015' THEN m.id END) AS matches\_2014\_2015

FROM country AS c

LEFT JOIN match AS m

ON c.id = m.country\_id

-- Group by country name alias

GROUP BY c.name;

| **country** | **matches\_2012\_2013** | **matches\_2013\_2014** | **matches\_2014\_2015** |
| --- | --- | --- | --- |
| Portugal | 240 | 240 | 306 |
| France | 380 | 380 | 380 |
| Scotland | 228 | 228 | 228 |

* Create 3 CASE statements to "count" matches in the '2012/2013', '2013/2014', and '2014/2015' seasons, respectively.
* Have each CASE statement return a 1 for every match you want to include, and a 0 for every match to exclude.
* Wrap the CASE statement in a SUM to return the total matches played in each season.
* Group the query by the country name alias.
* SELECT
* c.name AS country,
* -- Sum the total records in each season where the home team won
* SUM(CASE WHEN m.season = '2012/2013' AND m.home\_goal > m.away\_goal THEN 1
* ELSE 0 END ) AS matches\_2012\_2013,
* SUM(CASE WHEN m.season = '2013/2014' AND m.home\_goal >m.away\_goal THEN 1
* ELSE 0 END) AS matches\_2013\_2014,
* SUM(CASE WHEN m.season = '2014/2015'  AND m.home\_goal >m.away\_goal THEN 1
* ELSE 0 END) AS matches\_2014\_2015
* FROM country AS c
* LEFT JOIN match AS m
* ON c.id = m.country\_id
* -- Group by country name alias
* GROUP BY c.name;

| **country** | **matches\_2012\_2013** | **matches\_2013\_2014** | **matches\_2014\_2015** |
| --- | --- | --- | --- |
| Portugal | 103 | 108 | 137 |
| France | 170 | 168 | 181 |
| Scotland | 89 | 102 | 102 |

# Calculating percent with CASE and AVG

CASE statements will return any value you specify in your THEN clause. This is an incredibly powerful tool for robust calculations and data manipulation when used in conjunction with an aggregate statement. One key task you can perform is using CASE inside an AVG function to calculate a percentage of information in your database.

Here's an example of how you set that up:

AVG(CASE WHEN condition\_is\_met THEN 1

WHEN condition\_is\_not\_met THEN 0 END)

With this approach, it's important to accurately specify which records count as 0, otherwise your calculations may not be correct!

Your task is to examine the number of wins, losses, and ties in each country. The matches table is filtered to include all matches from the 2013/2014 and 2014/2015 seasons.

* Create 3 CASE statements to COUNT the total number of home team wins, away team wins, and ties, which will allow you to examine the total number of records.
* SELECT
* c.name AS country,
* -- Count the home wins, away wins, and ties in each country
* COUNT(CASE WHEN m.home\_goal > m.away\_goal THEN m.id
* END) AS home\_wins,
* COUNT(CASE WHEN m.home\_goal < m.away\_goal THEN m.id
* END) AS away\_wins,
* COUNT(CASE WHEN m.home\_goal = m.away\_goal THEN m.id
* END) AS ties
* FROM country AS c
* LEFT JOIN matches AS m
* ON c.id = m.country\_id
* GROUP BY country;

| **country** | **home\_wins** | **away\_wins** | **ties** |
| --- | --- | --- | --- |
| Portugal | 245 | 156 | 145 |
| France | 349 | 215 | 196 |
| Scotland | 204 | 158 | 94 |
|  |  |  |  |

* Calculate the percentage of matches tied using a CASE statement inside AVG.
* Fill in the logical operators for each statement. Alias your columns as ties\_2013\_2014 and ties\_2014\_2015, respectively.

SELECT

    c.name AS country,

    -- Round the percentage of tied games to 2 decimal points

    \_\_\_(\_\_\_(CASE WHEN m.season='2013/2014' AND m.home\_goal = m.away\_goal THEN 1

             WHEN m.season='2013/2014' AND m.home\_goal != m.away\_goal THEN 0

             END),\_\_\_) AS pct\_ties\_2013\_2014,

    \_\_\_(\_\_\_(CASE WHEN m.season='2014/2015' AND m.home\_goal = m.away\_goal THEN 1

             WHEN m.season='2014/2015' AND m.home\_goal != m.away\_goal THEN 0

             END),\_\_\_) AS pct\_ties\_2014\_2015

FROM country AS c

LEFT JOIN matches AS m

ON c.id = m.country\_id

GROUP BY country;

| **country** | **ties\_2013\_2014** | **ties\_2014\_2015** |
| --- | --- | --- |
| Portugal | 0.25000000000000000000 | 0.27777777777777777778 |
| France | 0.28421052631578947368 | 0.23157894736842105263 |
|  |  |  |

* The previous "ties" columns returned values with 14 decimal points, which is not easy to interpret. Use the ROUND function to round to 2 decimal points.
* SELECT
* c.name AS country,
* -- Round the percentage of tied games to 2 decimal points
* ROUND(AVG(CASE WHEN m.season='2013/2014' AND m.home\_goal = m.away\_goal THEN 1
* WHEN m.season='2013/2014' AND m.home\_goal != m.away\_goal THEN 0
* END),2) AS pct\_ties\_2013\_2014,
* ROUND(AVG(CASE WHEN m.season='2014/2015' AND m.home\_goal = m.away\_goal THEN 1
* WHEN m.season='2014/2015' AND m.home\_goal != m.away\_goal THEN 0
* END),2) AS pct\_ties\_2014\_2015
* FROM country AS c
* LEFT JOIN matches AS m
* ON c.id = m.country\_id
* GROUP BY country;

| **country** | **pct\_ties\_2013\_2014** | **pct\_ties\_2014\_2015** |
| --- | --- | --- |
| Portugal | 0.25 | 0.28 |
| France | 0.28 | 0.23 |

# Filtering using scalar subqueries

Subqueries are incredibly powerful for performing complex filters and transformations. You can filter data based on single, scalar values using a subquery in ways you cannot by using WHERE statements or joins. Subqueries can also be used for more advanced manipulation of your data set. You will likely encounter subqueries in any real-world setting that uses relational databases.

In this exercise, you will generate a list of matches where the total goals scored (for both teams in total) is more than 3 times the average for games in the matches\_2013\_2014 table, which includes all games played in the 2013/2014 season.

* Calculate *triple* the average home + away goals scored across all matches. This will become your subquery in the next step. Note that this column does not have an alias, so it will be called ?column? in your results.
* -- Select the average of home + away goals, multiplied by 3
* SELECT
* 3 \* AVG(home\_goal + away\_goal)
* FROM matches\_2013\_2014;

| **?column?** |
| --- |
| 8.3004617414248020 |

* Select the date, home goals, and away goals in the main query.
* Filter the main query for matches where the total goals scored exceed the value in the subquery.

SELECT

    -- Select the date, home goals, and away goals scored

    date,

    home\_goal,

    away\_goal

FROM  matches\_2013\_2014

-- Filter for matches where total goals exceeds 3x the average

WHERE (home\_goal + away\_goal) >

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

        FROM matches\_2013\_2014);

| **date** | **home\_goal** | **away\_goal** |
| --- | --- | --- |
| 2013-12-14 | 6 | 3 |
| 2014-03-22 | 3 | 6 |
| 2013-10-30 | 7 | 3 |

# Filtering using a subquery with a list

Your goal in this exercise is to generate a list of teams that never played a game in their home city. Using a subquery, you will generate a list of unique hometeam\_ID values from the unfiltered match table to exclude in the team table's team\_api\_ID column.

In addition to filtering using a single-value (scalar) subquery, you can create a list of values in a subquery to filter data based on a complex set of conditions. This type of subquery generates a one column reference list for the main query. As long as the values in your list match a column in your main query's table, you don't need to use a join -- even if the list is from a separate table.

* Create a subquery in the WHERE clause that retrieves all unique hometeam\_ID values from the match table.
* Select the team\_long\_name and team\_short\_name from the team table. Exclude all values from the subquery in the main query.
* SELECT
* -- Select the team long and short names
* team\_long\_name,
* team\_short\_name
* FROM team
* -- Exclude all values from the subquery
* WHERE team\_api\_id NOT IN
* (SELECT DISTINCT hometeam\_id  FROM match);

| **team\_long\_name** | **team\_short\_name** |
| --- | --- |
| FCV Dender EH | DEN |
| KSV Roeselare | ROS |
| Tubize | TUB |
| Royal Excel Mouscron | MOU |

# Filtering with more complex subquery conditions

In the previous exercise, you generated a list of teams that have no home matches listed in the soccer database using a subquery in WHERE. Let's do some further exploration in this database by creating a list of teams that scored 8 or more goals in a home match.

In order to do this, you will construct a subquery in the WHERE statement with its own filtering condition.

* Create a subquery in WHERE clause that retrieves all hometeam\_ID values from match with a home\_goal score greater than or equal to *8*.
* Select the team\_long\_name and team\_short\_name from the team table. *Include* all values from the subquery in the main query.
* SELECT
* -- Select the team long and short names
* team\_long\_name,
* team\_short\_name
* FROM team
* -- Filter for teams with 8 or more home goals
* WHERE team\_api\_id  IN
* (SELECT hometeam\_id
* FROM match
* WHERE home\_goal >= 8);

| **team\_long\_name** | **team\_short\_name** |
| --- | --- |
| Manchester United | MUN |
| Chelsea | CHE |
| Southampton | SOU |

# Joining Subqueries in FROM

The match table in the European Soccer Database does not contain country or team names. You can get this information by joining it to the country table, and use this to aggregate information, such as the number of matches played in each country.

If you're interested in filtering data from one of these tables, you can also create a subquery from one of the tables, and then join it to an existing table in the database. A subquery in FROM is an effective way of answering detailed questions that requires filtering or transforming data before including it in your final results.

Your goal in this exercise is to generate a subquery using the match table, and then join that subquery to the country table to calculate information about matches with 10 or more goals in total!

* Create the subquery to be used in the next step, which selects the country ID and match ID (id) from the match table.
* Filter the query for matches with greater than or equal to 10 goals.
* SELECT
* -- Select the country ID and match ID
* country\_id,
* id
* FROM match
* -- Filter for matches with 10 or more goals in total
* WHERE (home\_goal + away\_goal) >= 10;

| **country\_id** | **id** |
| --- | --- |
| 1729 | 3093 |
| 1729 | 3369 |
| 1729 | 3566 |

* Construct a subquery that selects only matches with 10 or more total goals.
* Inner join the subquery onto country in the main query.
* Select name from country and count the id column from match.
* SELECT
* -- Select country name and the count match IDs
* c.name AS country\_name,
* COUNT(sub.id) AS matches
* FROM country AS c
* -- Inner join the subquery onto country
* -- Select the country id and match id columns
* INNER JOIN (SELECT country\_id, id
* FROM match
* -- Filter the subquery by matches with 10+ goals
* WHERE (home\_goal+away\_goal)>=10) AS sub
* ON c.id = sub.country\_id
* GROUP BY country\_name;

| **country\_name** | **matches** |
| --- | --- |
| Netherlands | 1 |
| Spain | 4 |
| Germany | 1 |
| England | 3 |

# Building on Subqueries in FROM

In the previous exercise, you found that England, Netherlands, Germany and Spain were the only countries that had matches in the database where 10 or more goals were scored overall. Let's find out some more details about those matches -- when they were played, during which seasons, and how many of the goals were home vs. away goals.

You'll notice that in this exercise, the table alias is excluded for every column selected in the main query. This is because the main query is extracting data from the subquery, which is treated as a single table.

* Complete the subquery inside the FROM clause. Select the country name from the country table, along with the date, the home goal, the away goal, and the total goals columns from the match table.
* Create a column in the subquery that adds home and away goals, called total\_goals. This will be used to filter the main query.
* Select the country, date, home goals, and away goals in the main query.
* Filter the main query for games with 10 or more total goals.
* SELECT
* -- Select country, date, home, and away goals from the subquery
* country,
* date,
* home\_goal,
* away\_goal
* FROM
* -- Select country name, date, home\_goal, away\_goal, and total goals in the subquery
* (SELECT c.name AS country,
* m.date,
* m.home\_goal,
* m.away\_goal,
* (m.home\_goal + m.away\_goal) AS total\_goals
* FROM match AS m
* LEFT JOIN country AS c
* ON m.country\_id = c.id) AS subq
* -- Filter by total goals scored in the main query
* WHERE total\_goals  >= 10;

| **country** | **date** | **home\_goal** | **away\_goal** |
| --- | --- | --- | --- |
| England | 2011-08-28 | 8 | 2 |
| England | 2012-12-29 | 7 | 3 |
| England | 2013-05-19 | 5 | 5 |

# Add a subquery to the SELECT clause

Subqueries in SELECT statements generate a single value that allow you to pass an aggregate value down a data frame. This is useful for performing calculations on data within your database.

In the following exercise, you will construct a query that calculates the average number of goals per match in each country's league

* In the subquery, select the average total goals by adding home\_goal and away\_goal.
* Filter the results so that only the average of goals in the 2013/2014 season is calculated.
* In the main query, select the average total goals by adding home\_goal and away\_goal. This calculates the average goals for each league.
* Filter the results in the main query the same way you filtered the subquery. Group the query by the league name.
* SELECT
* l.name AS league,
* -- Select and round the league's total goals
* ROUND(AVG(m.home\_goal + m.away\_goal), 2) AS avg\_goals,
* -- Select & round the average total goals for the season
* (SELECT ROUND(AVG(home\_goal + away\_goal), 2)
* FROM match
* WHERE season = '2013/2014') AS overall\_avg
* FROM league AS l
* LEFT JOIN match AS m
* ON l.country\_id = m.country\_id
* -- Filter for the 2013/2014 season
* WHERE m.season = '2013/2014'
* GROUP BY l.name;

| **league** | **avg\_goals** | **overall\_avg** |
| --- | --- | --- |
| Switzerland Super League | 2.89 | 2.77 |
| Poland Ekstraklasa | 2.64 | 2.77 |
| Netherlands Eredivisie | 3.20 | 2.77 |
| Scotland Premier League | 2.75 | 2.77 |

# Subqueries in Select for Calculations

Subqueries in SELECT are a useful way to create calculated columns in a query. A subquery in SELECT can be treated as a single numeric value to use in your calculations. When writing queries in SELECT, it's important to remember that filtering the main query does not filter the subquery -- and vice versa.

In the previous exercise, you created a column to compare each league's average total goals to the overall average goals in the 2013/2014 season. In this exercise, you will add a column that directly compares these values by subtracting the overall average from the subquery.

* Select the average goals scored in a match for each league in the main query.
* Select the average goals scored in a match overall for the 2013/2014 season in the subquery.
* Subtract the subquery from the average number of goals calculated for each league.
* Filter the main query so that only games from the 2013/2014 season are included.
* SELECT
* -- Select the league name and average goals scored
* l.name AS league,
* ROUND(AVG(m.home\_goal + m.away\_goal),2) AS avg\_goals,
* -- Subtract the overall average from the league average
* ROUND(AVG(m.home\_goal + m.away\_goal) -
* (SELECT AVG(home\_goal + away\_goal)
* FROM match
* WHERE season= '2013/2014'),2) AS diff
* FROM league AS l
* LEFT JOIN match AS m
* ON l.country\_id = m.country\_id
* -- Only include 2013/2014 results
* WHERE season= '2013/2014'
* GROUP BY l.name;

| **league** | **avg\_goals** | **diff** |
| --- | --- | --- |
| Switzerland Super League | 2.89 | 0.12 |
| Poland Ekstraklasa | 2.64 | -0.13 |
| Netherlands Eredivisie | 3.20 | 0.43 |

# ALL the subqueries EVERYWHERE

In soccer leagues, games are played at different stages. Winning teams progress from one stage to the next, until they reach the final stage. In each stage, the stakes become higher than the previous one. The match table includes data about the different stages that each match took place in.

In this lesson, you will build a final query across 3 exercises that will contain three subqueries -- one in the SELECT clause, one in the FROM clause, and one in the WHERE clause. In the final exercise, your query will extract data examining the average goals scored in each stage of a match. Does the average number of goals scored change as the stakes get higher from one stage to the next?

* Extract the average number of home and away team goals in two SELECT subqueries.
* Calculate the average home and away goals for the specific stage in the main query.
* Filter both subqueries and the main query so that only data from the 2012/2013 season is included.
* Group the query by the m.stage column.
* SELECT
* -- Select the stage and average goals for each stage
* m.stage,
* ROUND(AVG(m.home\_goal + m.away\_goal),2) AS avg\_goals,
* -- Select the average overall goals for the 2012/2013 season
* ROUND((SELECT AVG(home\_goal  + away\_goal)
* FROM match
* WHERE season = '2012/2013'),2) AS overall
* FROM match AS m
* -- Filter for the 2012/2013 season
* WHERE season = '2012/2013'
* -- Group by stage
* GROUP BY m.stage ;

| **stage** | **avg\_goals** | **overall** |
| --- | --- | --- |
| 29 | 2.63 | 2.77 |
| 4 | 2.80 | 2.77 |
| 34 | 2.68 | 2.77 |

# Add a subquery in FROM

In the previous exercise, you created a data set listing the average home and away goals in each match stage of the 2012/2013 match season.

In this next step, you will turn the main query into a subquery to extract a list of stages where the average home goals in a stage is higher than the overall average for home goals in a match.

* Calculate the average home goals and average away goals from the match table for each stage in the FROM clause subquery.
* Add a subquery to the WHERE clause that calculates the overall average home goals.
* Filter the main query for stages where the average home goals is higher than the overall average.
* Select the stage and avg\_goals columns from the s subquery into the main query.

SELECT

    -- Select the stage and average goals from the subquery

    s.stage,

    ROUND(s.avg\_goals,2) AS avg\_goals

FROM

    -- Select the stage and average goals in 2012/2013

    (SELECT

         stage,

         AVG(home\_goal + away\_goal) AS avg\_goals

     FROM match

     WHERE season = '2012/2013'

     GROUP BY stage) AS s

WHERE

    -- Filter the main query using the subquery

    s.avg\_goals > (SELECT AVG(home\_goal + away\_goal)

                    FROM match WHERE season = '2012/2013');

| **stage** | **avg\_goals** |
| --- | --- |
| 4 | 2.80 |
| 10 | 2.96 |
| 38 | 3.17 |
| 6 | 2.78 |

# Add a subquery in SELECT

In the previous exercise, you added a subquery to the FROM statement and selected the stages where the number of average goals in a stage exceeded the overall average number of goals in the 2012/2013 match season. In this final step, you will add a subquery in SELECT to compare the average number of goals scored in each stage to the total.

* Create a subquery in SELECT that yields the average goals scored in the 2012/2013 season. Name the new column overall\_avg.
* Create a subquery in FROM that calculates the average goals scored in each stage during the 2012/2013 season.
* Filter the main query for stages where the average goals exceeds the overall average in 2012/2013.
* SELECT
* -- Select the stage and average goals from s
* s.stage,
* ROUND(s.avg\_goals,2) AS avg\_goal,
* -- Select the overall average for 2012/2013
* (SELECT AVG(home\_goal + away\_goal) FROM match WHERE season = '2012/2013') AS overall\_avg
* FROM
* -- Select the stage and average goals in 2012/2013 from match
* (SELECT
* stage,
* AVG(home\_goal + away\_goal) AS avg\_goals
* FROM match
* WHERE season = '2012/2013'
* GROUP BY stage) AS s
* WHERE
* -- Filter the main query using the subquery
* s.avg\_goals > (SELECT AVG(home\_goal + away\_goal)
* FROM match WHERE season = '2012/2013');

| **stage** | **avg\_goal** | **overall\_avg** |
| --- | --- | --- |
| 4 | 2.80 | 2.7726993865030675 |
| 10 | 2.96 | 2.7726993865030675 |
| 38 | 3.17 | 2.7726993865030675 |