SQL Functions Learnt

SELECT, use to select attributes you want from a df

SELECT x AS y, use to rename fields in query result

FROM, use to select the dataframe you want to select attributes from

SELECT DISTINCT x, use to select only unique values from selection

COUNT(), use to count number of rows of selected (value)

WHERE, use to specify conditions on a selection

AND, use to add conditions

OR, use to add an alternate condition

x BETWEEN, numerical selection conditon based on two values in attribute

WHERE IN, use to streamline selections to anything in an array

IS NULL, to query for missing data

NOT, to query the opposite of a statement

LIKE 'X%', used in a WHERE clause to search for a pattern

'X%' % wildcard will match zero, one, or many characters in text

'X\_' \_ wildcard will match a single character

e.g. LIKE '\_r%' will give rows where the second letter ='r'

'%r\_' will give records where the second to last letter = r

Variable Types, (can be viewed in database 'schemas'):

VARCHAR - variable character (common for strings)

INT - intergers

NUMERIC, DATE etc...

CREATE VIEW x AS, assigns subsequent query to 'x' so that the

selection can be made in shorthand in the future

Aggregate Functions

AVG(), SUM(), MIN(), MAX(), COUNT()

ROUND(x, decimalplaces) , round to specified decimal (leave blank for no dec)

GROUP BY

ORDER BY ... ASC/DESC

HAVING, use to filter grouped records

Order of execution:

FROM, WHERE, GROUP BY, HAVING, SELECT, ORDER BY, LIMIT

INNER JOIN

ON xxx.yyy = xxx.yyy, or USING(yyy), (for when key field is named the same)

e.g.

-- Select fields with aliases

SELECT c.code AS country\_code, name, year, inflation\_rate

FROM countries AS c

-- Join to economies (alias e)

INNER JOIN economies AS e

-- Match on code field using table aliases

ON c.code = e.code

LEFT JOIN x, joins table x as the right to FROM y

RIGHT

FULL JOIN

CROSS JOIN (do not need ON or USING), use to create output of every possible combination of values

<>, operator means 'does not equal'

self joins, used to compare parts of the same table

eg.

FROM x AS x1

INNER JOIN x AS x2

ON x1.key = x2.key

AND x1.key <> x2.key; <-- This removes matching rows with themselves

-- Set Theory --

UNION, set operator that merges two tables excluding duplicates

UNION ALL, merges two tables including duplicates

e.g.

SELECT \*

FROM left\_table

UNION

SELECT \*

FROM right\_table

eg2.

SELECT p.country\_code, p.year

FROM populations AS p

UNION

SELECT e.code, e.year

FROM economies AS e

ORDER BY country\_code , year;

INTERSECT, set operator that only returns values present in both tables

eg.1

-- Return all cities with the same name as a country

SELECT name

FROM cities

INTERSECT

SELECT name

FROM countries;

EXCEPT, set operator that returns unique values from the left table

e.g.

-- Return all cities that do not have the same name as a country

SELECT name

FROM cities

EXCEPT

SELECT name

FROM countries

ORDER BY name;

SUBQUERIES: can occur in WHERE, SELECT and FROM clauses

e.g.

-- Select relevant fields from cities table

SELECT name, country\_code, urbanarea\_pop

FROM cities

-- Filter using a subquery on the countries table

WHERE name IN

(SELECT capital

FROM countries)

ORDER BY urbanarea\_pop DESC;

LEFT JOIN & SUBQUERY FOR SAME TASK:

-- Find top nine countries with the most cities

SELECT countries.name AS country, COUNT(\*) AS cities\_num

FROM countries

LEFT JOIN cities

ON countries.code = cities.country\_code

GROUP BY country

-- Order by count of cities as cities\_num

ORDER BY cities\_num DESC, country

LIMIT 9;

SELECT countries.name AS country,

-- Subquery that provides the count of cities

(SELECT COUNT(country\_code)

FROM cities

WHERE country\_code=code) AS cities\_num

FROM countries

ORDER BY cities\_num DESC, country

LIMIT 9;

SUBQUERY INSIDE FROM

SELECT DISTINCT monarchs.continent, sub.most\_recent

FROM monarchs,

(SELECT

continent,

MAX(indep\_year) AS most\_recent

FROM states

GROUP BY continent) AS sub

WHERE monarchs.continent = sub.continent

ORDER BY continent;

(Count of languages relating to country (key field = code)):

-- Select code, and language count as lang\_num

SELECT code, (SELECT COUNT(\*) FROM languages AS l WHERE l.code = c.code) AS lang\_num

FROM countries AS c

MULTIPLE TABLE IN FROM CLAUSE

e.g. or to prevent table dups:

SELECT left\_table.id, left\_val SELECT DISTINCT left\_table.id, left\_val

FROM left\_table, right\_table FROM left\_table, right\_table

WHERE left\_table.id = right\_table.id WHERE left\_table.id = right\_table.id

INCLUDE SUBQUERY AS A TEMPOARY TABLE IN FROM CLAUSE AND SELECT FROM IT:

eg2. -- Query to return continents with monarchs and the year the most recent country

gained independence:

SELECT DISTINCT monarchs.continent, sub.most\_recent

FROM monarchs,

(SELECT

continent,

MAX(indep\_year) AS most\_recent

FROM states

GROUP BY continent) AS sub

WHERE monarchs.continent = sub.continent

ORDER BY continent;

SUBQUERY INSIDE FROM EXERCISE:

1.

-- Select code, and language count as lang\_num

SELECT code, COUNT(code) AS lang\_num

FROM languages

GROUP BY code; --< Need the GROUP BY clause here to prevent an error

2. Put above query as a subquery and fill in blanks

-- Select local\_name and lang\_num from appropriate tables

SELECT countries.local\_name, sub.lang\_num

FROM countries,

(SELECT code, COUNT(\*) AS lang\_num

FROM languages

GROUP BY code) AS sub

WHERE countries.code = sub.code

ORDER BY lang\_num DESC;

SUBQUERY EXERCISE

1. using where subquery to exclude monarchy or republican countries

SELECT code, inflation\_rate, unemployment\_rate

FROM economies

WHERE year = 2015

AND code NOT IN

-- Subquery returning country codes filtered on gov\_form

(SELECT code

FROM countries

WHERE gov\_form = 'Republic' OR gov\_form = 'Monarchy')

ORDER BY inflation\_rate;

can also use 'LIKE':

(SELECT code

FROM countries

WHERE gov\_form LIKE '%Republic%' OR gov\_form LIKE '%Monarchy%')

Final Exercise for subquery module:

Determine the top 10 capital cities in europe and the Americas by city\_perc,

a metric you'll calculate. city\_perc is a percentage that calculates the "proper" population

in a city as a percentage of the total population in the wider metro area, as follows:

city\_proper\_pop / metroarea\_pop \* 100

-- Select fields from cities

SELECT name, country\_code, city\_proper\_pop,

metroarea\_pop,

city\_proper\_pop / metroarea\_pop \* 100 AS city\_perc

FROM cities

-- Use subquery to filter city name

WHERE country\_code in

(SELECT code

FROM countries

WHERE capital = cities.name ---- (This line was actually not needed)

AND continent LIKE '%America'

OR continent LIKE 'Europe')

-- Add filter condition such that metroarea\_pop does not have null values

AND metroarea\_pop IS NOT NULL

-- Sort and limit the result

ORDER BY city\_perc DESC

LIMIT 10;

In this module, I learned of many types of joins:

INNER JOIN (just JOIN)

Outer joins: LEFT, RIGHT and FULL

CROSS JOIN

Semi joins / anti join

Self join

I learned about set operations:

UNION / UNION ALL

INTERSECT

EXCEPT

I also learnt basic subqueries:

Inside, SELECT, FROM and WHERE clauses

CASE statements, contain WHEN, THEN, ELSE and END

(used as SQLs 'if' statements)

e.g

CASE WHEN x=1 THEN 'a'

WHEN x=2 THEN 'b'

ELSE 'c' END AS new\_column

e.g.2

-- Identify the home team as Bayern Munich, Schalke 04, or neither

SELECT

CASE WHEN hometeam\_id = 10189 THEN 'FC Schalke 04' <-IMPORTANT CASE is a statement

WHEN hometeam\_id = 9823 THEN 'FC Bayern Munich' within the SELECT clause

ELSE 'Other' END AS home\_team,

COUNT(id) AS total\_matches

FROM matches\_germany

-- Group by the CASE statement alias

GROUP BY home\_team;

eg3. Gives the date, opponent (away team) and outcome of every match

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 > m.away\_goal THEN 'Home win!'

WHEN m.home\_goal < m.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;

ext. Filter for just barcelona

SELECT

m.date,

t.team\_long\_name AS opponent,

-- Complete the CASE statement with an alias

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

WHEN m.home\_goal < m.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;

\*\*\*

AND statements can also be used in CASE statements

Removing the ELSE statement will work, however all other conditions will be returned as NULL

\*\*\*

By repeating your CASE statement in a WHERE clause will filter your results, removing NULL values

eg.

Graphical user interface

Description automatically generated with medium confidence

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);

^^^ Shows code example using CASE to label team names and all combination of match outcomes, filtering for just those two teams.

-- 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; Graphical user interface, application

Description automatically generated

^^ Labels wins but doesn’t remove nulls

-- 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;

Graphical user interface, application

Description automatically generated

^ Filters results to show only wins of Bologna

It has been shown how CASE statements can be used to categorise/select data and now filter data.

CASE statements can also be used in Aggregating data:

Graphical user interface, text

Description automatically generated

^ CASE statement within Aggregate COUNT function, also works with SUM & AVG

Creating an average goal count\/

Text

Description automatically generated

Creating a percentage win rate:

Text

Description automatically generated

-- \/\/\/\/ Creating

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 country;

A screenshot of a computer

Description automatically generated with medium confidence

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

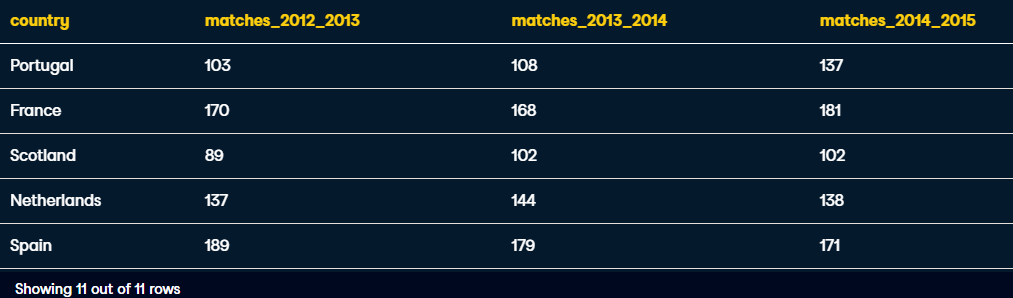
FROM country AS c

LEFT JOIN match AS m

ON c.id = m.country\_id

-- Group by country name alias

GROUP BY country;



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;

Graphical user interface, application

Description automatically generated

Now this query will be used as a subquery within a FROM clause to find all stages where their average goals scored were above the total average goals:

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');

Graphical user interface

Description automatically generated with medium confidence

**Building in another subquery in the SELECT clause:**

subquery in FROM that calculates the average goals scored in each stage during the 2012/2013 season

subquery in SELECT that yields the average goals scored in 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');

Graphical user interface

Description automatically generated

**Correlated subqueries:**

Graphical user interface, text

Description automatically generated

Graphical user interface, text, application

Description automatically generated

**Simple:**

Graphical user interface, application

Description automatically generated

**Correlated:**

Graphical user interface, application

Description automatically generated

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((sub.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);

A screenshot of a computer

Description automatically generated with medium confidence

add an additional column for matching to answer the question -- what was the highest scoring match for each country, in each season:

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

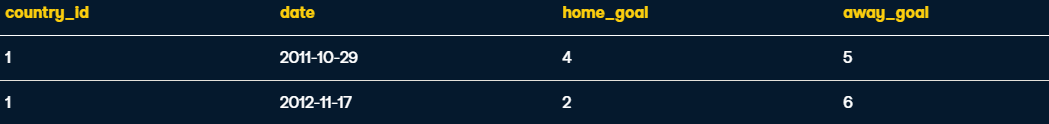
    (home\_goal + away\_goal) =

        (SELECT MAX(sub.home\_goal + sub.away\_goal)

         FROM match AS sub

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

               AND main.season = sub.season);



**Nested Subqueries**

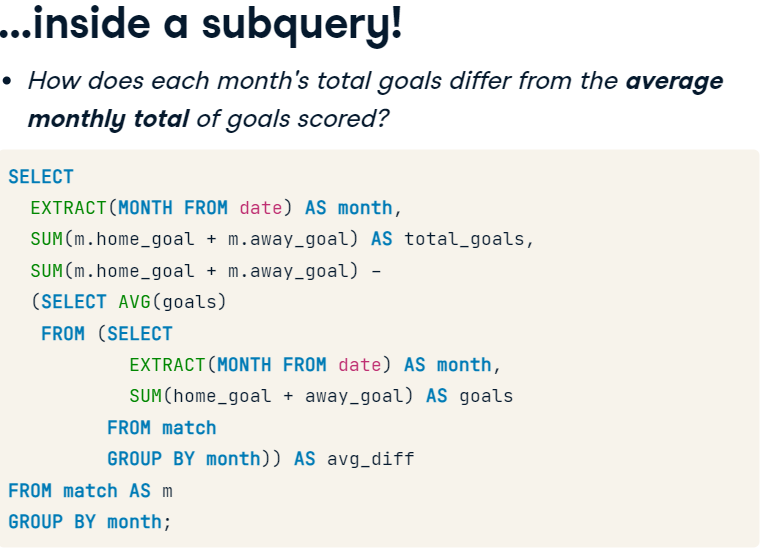
**Subqueries inside another subquery – For performing layers of transformation**

**e.g.**

Graphical user interface, text, application

Description automatically generated A screenshot of a computer

Description automatically generated with medium confidence



2. Gets average of inner query

1. Select the sum of goals from each month

Shape, rectangle

Description automatically generatedEXTRACT, - takes pre-formatted data out of a field eg. Month out of a date

Graphical user interface, text

Description automatically generated

**Nested Subquery where outer query is correlated :**

Graphical user interface, text, application

Description automatically generatedA picture containing text

Description automatically generated

examine the highest total number of goals in each season, overall, and during July across all seasons:

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;

Graphical user interface

Description automatically generated

Average number of matches per season where a team scored >=5 goals:

1. Subquery for FROM:

-- Select matches where a team scored 5+ goals

SELECT

    country\_id,

    season,

    id

FROM match

WHERE home\_goal >= 5 OR away\_goal >= 5;

Graphical user interface

Description automatically generated

2. Putting Subquery into FROM:

-- 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;

Graphical user interface, application

Description automatically generated

3. Nest that query as a query within FROM

SELECT

  c.name AS country,

    -- Calculate the average matches per season

  AVG(outer\_s.matches) 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;

Graphical user interface, application

Description automatically generated

**Common Table Expressions**

A Table declared before the main query for improving readability

Graphical user interface, text, application

Description automatically generated

e.g. Rewriting a subquery in FROM

Results of countries with a count of very few high scoring matches

Graphical user interface, text, application

Description automatically generatedTimeline

Description automatically generated with medium confidence

This subquery can be created and treated like an existing table by taking it out of the main query and referencing it like a table:

Graphical user interface, text, application

Description automatically generatedTimeline

Description automatically generated with medium confidence

We use these instead of subqueries due to:

Performance, as it’s only run once

Organisation

Each CTE can reference the CTEs that came before it

They can also SELF JOIN (recursive CTEs)

-- 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;

Background pattern

Description automatically generated

**Join inside a CTE**

-- 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;

Graphical user interface, application

Description automatically generated

**Nested Subquery within a CTE**

-- 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) = 08))

-- 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;Graphical user interface, application

Description automatically generated

**Differentiating Techniques**

Graphical user interface, text, application

Description automatically generated

# Text, table Description automatically generated with medium confidence

# 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.

**2 subqueries:**

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

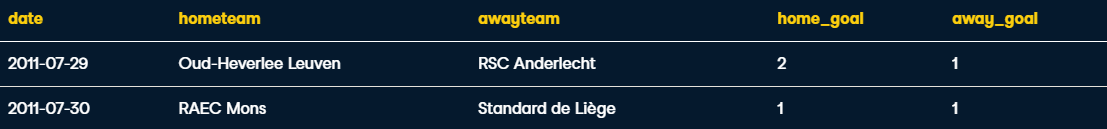
  FROM match

  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;



**Correlated Subquery:**

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

     m.home\_goal,

     m.away\_goal

FROM match AS m;



**Common Table Expressions: (*Note that you only need to write* WITH *once)***

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;



**Window Functions:**

Graphical user interface, text, application

Description automatically generated

**OVER(), clause to initiate window function**

**Eg. Subquery in Select (complex version)**

Graphical user interface, text, application, chat or text message

Description automatically generated**\/**

**Window Function (more concise)**

Graphical user interface, text

Description automatically generated Text

Description automatically generated with low confidence

**RANK window function example:** Text

Description automatically generated

Graphical user interface, text

Description automatically generated

SELECT

    -- Select the id, country name, season, home, and away goals

    m.id,

    c.name AS country,

    m.season,

    m.home\_goal,

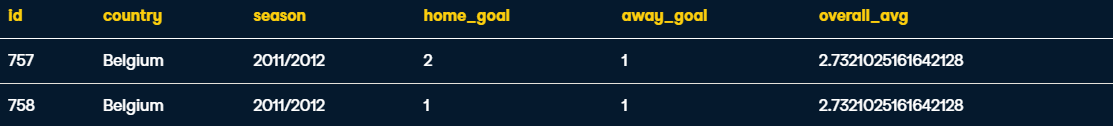
    m.away\_goal,

    -- 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;



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;



**Add DESC to order the RANK**



**Window Partitions**

**OVER** and **PARTITION BY**:

Graphical user interface, text

Description automatically generated with medium confidence

**e.g.**

Graphical user interface, text

Description automatically generated

**e.g.2**

Graphical user interface, text

Description automatically generated

**PARTITION BY** multiple columns:

Graphical user interface, text

Description automatically generated

Text

Description automatically generated

Ex1.

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

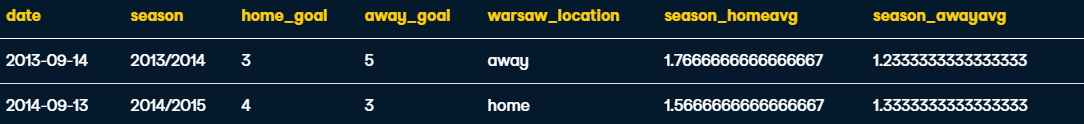
-- Filter the data set for Legia Warszawa matches only

WHERE

    hometeam\_id = 8673

    OR awayteam\_id = 8673

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



Ex2. (Partition by 2 columns\_)

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

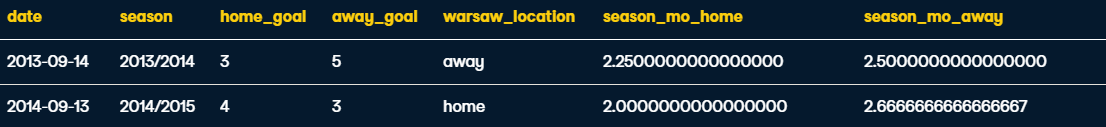
FROM match

WHERE

    hometeam\_id = 8673

    OR awayteam\_id = 8673

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



**Sliding Windows**

Graphical user interface, text, application

Description automatically generated

Graphical user interface, text, application

Description automatically generated

e.g.

Graphical user interface, text, application

Description automatically generated

^ Here the query first calculates the sum of goals when Manchester city played as the home team during the 2011/2012 season. Then states you want the calculation into a running total ordered by date of match, oldest -> most recent and calculated from the beginning of the dataset to the current row.

Table

Description automatically generated

^ Creates a running total of every two

Ex1. UNBOUNDED PRECEDING AND CURRENT ROW

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';

Graphical user interface

Description automatically generated with low confidence

Ex2. Backward running total/avg CURRENT ROW AND UNBOUNDED FOLLOWING

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';

Graphical user interface

Description automatically generated with medium confidence