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;

