DataCamp Intermediate SQL

README

- This is an example of how I work through a DataCamp lesson - no matter what the subject. It's a little more work when there are videos involved. Time to completion about 2 hours.
- I am pretty sure that you won't really understand what's going on without doing it at this level of granularity and with the original data set.
- To make the statements work in SQLite though the original database and sample statements in DataCamp are for PostgreSQL, I used these schema diagrams, which were generated from the SQLite database. I also had to substitute `WHERE` filters for the provided sub-tables.
- This is also a way of keeping the DataCamp lessons interesting because otherwise I'd sometimes get very bored with the pace and/or the subject matter. Gotta keep things buzzing!

We'll take the `CASE`

Selecting from the European Soccer Database

```
SELECT
l.name AS League,
COUNT (m.country_id) as Total_matches
FROM League as l
LEFT OUTER JOIN Match as m
ON l.country_id = m.country_id
GROUP BY l.name;
```

Belgium Jupiler League	1728
England Premier League	3040
France Ligue 1	3040
Germany 1. Bundesliga	2448
Italy Serie A	3017
Netherlands Eredivisie	2448
Poland Ekstraklasa	1920
Portugal Liga ZON Sagres	2052
Scotland Premier League	1824
Spain LIGA BBVA	3040
Switzerland Super League	1422

Feedback: The `JOIN` combines the tables `League` and `Match` using the FK `country_id`, the PK of `Country`, and extracts `League.name` and total number of matches as a `COUNT` of the `League` entries grouped by `name` - this gives you all matches by league.

Compare home, away wins and ties in a season

- Let's say we want to compare the number of home team wins, away team wins, and ties in the 2013/2014 season. The `Match` table has two relevant columns – `home_team_goal`, and `away_team_goal` – for the respective goals of the team that played 'home' vs. the 'away' team that visited them.

```
SELECT date, id, home_team_goal, away_team_goal
FROM match
WHERE season = '2013/2014'
LIMIT 4;
```

```
date                 id    home_team_goal  away_team_goal
-------------------  ----  --------------  --------------
2014-03-29 00:00:00  1237  2               0
2014-03-29 00:00:00  1238  0               1
2014-04-05 00:00:00  1239  1               0
2014-04-05 00:00:00  1240  0               0
```

- We can add filters to the WHERE clause selecting wins, loses, and ties as separate queries, but that's not very efficient if you want to compare these separate outcomes in a single data set.

```
SELECT date, id, home_team_goal, away_team_goal
FROM match
WHERE season = '2013/2014'
AND home_team_goal > away_team_goal  -- filter for home team wins
LIMIT 4;
```

```
date                 id    home_team_goal  away_team_goal
-------------------  ----  --------------  --------------
2014-03-29 00:00:00  1237  2               0
2014-04-05 00:00:00  1239  1               0
2014-04-12 00:00:00  1241  2               1
2014-04-12 00:00:00  1242  2               0
```

- We need a control structure like IF THIS THEN THAT.

  `CASE` syntax

```
CASE
WHEN [cond] THEN [expr]
WHEN [cond] THEN [expr]
...         ELSE [expr] -- default
END AS [new_var] -- new_var is a new table column
```

- In this example, we use a `CASE` statement to create a new variable `outcome` that identifies matches as home team wins, away team wins, or ties. A new column is created with the appropriate text for each match given the outcome.

```
SELECT id, home_team_goal, away_team_goal,
CASE WHEN (home_team_goal > away_team_goal) THEN 'Home Team Win'
WHEN (home_team_goal < away_team_goal) THEN 'Away Team Win'
WHEN (home_team_goal = away_team_goal) THEN 'Tie'
```

```
END AS outcome
FROM Match
WHERE season = '2013/2014'
LIMIT 4;
```

```
id    home_team_goal  away_team_goal  outcome
----  --------------  --------------  -------------
1237  2               0               Home Team Win
1238  0               1               Away Team Win
1239  1               0               Home Team Win
1240  0               0               Tie
```

- Since there are only three outcomes, the last one can be replaced by the default `ELSE`:

```
SELECT id, home_team_goal, away_team_goal,
CASE WHEN (home_team_goal > away_team_goal) THEN 'Home Team Win'
WHEN (home_team_goal < away_team_goal) THEN 'Away Team Win'
ELSE 'Tie'
END AS outcome
FROM Match
WHERE season = '2013/2014'
LIMIT 4;
```

```
id    home_team_goal  away_team_goal  outcome
----  --------------  --------------  -------------
1237  2               0               Home Team Win
1238  0               1               Away Team Win
1239  1               0               Home Team Win
1240  0               0               Tie
```

Exercises

Basic `CASE` statements

- 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!

- `[X]`

  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_long_name,
team_api_id
FROM Team
-- Only include FC Schalke 04 and FC Bayern Munich
WHERE team_long_name IN ('FC Schalke 04', 'FC Bayern Munich');
```

```
team_long_name    team_api_id
----------------  -----------
FC Bayern Munich  9823
FC Schalke 04     10189
```

- Get Germany's `Country.id` from the `Country` table.

```
SELECT id FROM Country WHERE name='Germany';
```

- Explore the `Teams` table

```
.schema Team
```

```
CREATE TABLE IF NOT EXISTS "Team" (
        `id`    INTEGER PRIMARY KEY AUTOINCREMENT,
        `team_api_id`   INTEGER UNIQUE,
        `team_fifa_api_id`      INTEGER,
        `team_long_name`        TEXT,
        `team_short_name`       TEXT
);
```

- `[X]`

  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 home_team_api_id = 10189 THEN 'FC Schalke 04'
WHEN home_team_api_id = 9823  THEN 'FC Bayern Munich'
ELSE 'Other' END AS home_team,
COUNT(id)/2 AS total_matches
FROM Match WHERE country_id = 7809
-- Group by the CASE statement alias
GROUP BY home_team;
```

```
home_team         total_matches
----------------  -------------
FC Bayern Munich  68
FC Schalke 04     68
Other             1088
```

`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.

- Get Spain's `Country.id` from the `Country` table.

```
SELECT id FROM Country WHERE name='Spain';
```

- [X]

  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_team_goal > away_team_goal THEN 'Home win!'
WHEN home_team_goal < away_team_goal THEN 'Home loss :('
ELSE 'Tie' END AS outcome
FROM Match
WHERE country_id=21518
LIMIT 5;
```

```
date                 outcome
-------------------  ---------
2008-08-30 00:00:00  Home win!
2008-08-31 00:00:00  Tie
2008-08-31 00:00:00  Home win!
2008-08-31 00:00:00  Home win!
2008-08-31 00:00:00  Tie
```

- [X]

  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_team_goal > m.away_team_goal THEN 'Home win!'
WHEN m.home_team_goal < m.away_team_goal THEN 'Home loss :('
ELSE 'Tie' END AS outcome
FROM Match as m
-- Left join teams_spain onto matches_spain
LEFT JOIN Team AS t
ON m.away_team_api_id = t.team_api_id
WHERE country_id=21518
LIMIT 5;
```

```
date                 opponent        outcome
-------------------  --------------  ---------
2008-08-30 00:00:00  RCD Mallorca    Home win!
2008-08-31 00:00:00  Villarreal CF   Tie
2008-08-31 00:00:00  Real Madrid CF  Home win!
2008-08-31 00:00:00  FC Barcelona    Home win!
2008-08-31 00:00:00  Sevilla FC      Tie
```

- [X]

  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_team_goal > m.away_team_goal THEN 'Barcelona win!'
WHEN m.home_team_goal <m.away_team_goal THEN 'Barcelona loss :('
ELSE 'Tie' END AS outcome
FROM Match AS m
LEFT JOIN Team AS t
ON m.away_team_api_id = t.team_api_id
-- Filter for Barcelona as the home team
WHERE country_id = 21518 AND m.home_team_api_id = 8634
LIMIT 5;
```

```
date                 opponent         outcome
-------------------  ---------------  --------------
2008-11-08 00:00:00  Real Valladolid  Barcelona win!
2008-11-23 00:00:00  Getafe CF        Tie
2008-12-06 00:00:00  Valencia CF      Barcelona win!
2008-12-13 00:00:00  Real Madrid CF   Barcelona win!
2009-01-03 00:00:00  RCD Mallorca     Barcelona win!
```

- 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
m.date, t.team_long_name AS opponent,
CASE WHEN m.home_team_goal > m.away_team_goal THEN 'Barcelona win!'
WHEN m.home_team_goal <m.away_team_goal THEN 'Barcelona loss :('
ELSE 'Tie' END AS outcome
FROM Match AS m
LEFT JOIN Team AS t
ON m.home_team_api_id = t.team_api_id
-- Filter for Barcelona as the AWAY team
WHERE country_id = 21518
AND m.away_team_api_id = 8634
LIMIT 5;
```

```
date                 opponent       outcome
-------------------  -------------  -----------------
2008-08-31 00:00:00  CD Numancia    Barcelona win!
2008-11-16 00:00:00  RC Recreativo  Barcelona loss :(
2008-11-29 00:00:00  Sevilla FC     Barcelona loss :(
2008-12-21 00:00:00  Villarreal CF  Barcelona loss :(
2009-01-11 00:00:00  CA Osasuna     Barcelona loss :(
```

In CASE things get more complex

- [X]

If you want to test multiple logical conditions in a CASE statement, you can use AND inside your WHEN clause. For example, let's see if each match was played, and won, by the team Chelsea.

```
SELECT date, home_team_api_id, away_team_api_id,
CASE
WHEN home_team_api_id = 8455 AND home_team_goal > away_team_goal
THEN 'Chelsea home win!'
WHEN away_team_api_id = 8455 AND home_team_goal < away_team_goal
THEN 'Chelsea away win!'
ELSE 'Loss or tie :-(' END AS outcome
FROM Match
WHERE home_team_api_id = 8455 OR away_team_api_id = 8455
LIMIT 10;
```

```
date                 home_team_api_id  away_team_api_id  outcome
-------------------  ----------------  ----------------  -----------------
2008-08-17 00:00:00  8455              8462              Chelsea home win!
2008-10-29 00:00:00  8667              8455              Chelsea away win!
2008-11-01 00:00:00  8455              8472              Chelsea home win!
2008-11-09 00:00:00  8655              8455              Chelsea away win!
2008-11-15 00:00:00  8659              8455              Chelsea away win!
2008-11-22 00:00:00  8455              10261             Loss or tie :-(
2008-11-30 00:00:00  8455              9825              Loss or tie :-(
2008-12-06 00:00:00  8559              8455              Chelsea away win!
2008-12-14 00:00:00  8455              8654              Loss or tie :-(
2008-12-22 00:00:00  8668              8455              Loss or tie :-(
```

- [X]

  Removing the `WHERE` filter means that any game that does not meet the logical conditions (Chelsea plays) as `'Loss or tie'`. Only with the filter will Chelsea have been part of the game, and its ID will appear in the output table.

```
SELECT date, home_team_api_id, away_team_api_id,
CASE
WHEN home_team_api_id = 8455 AND home_team_goal > away_team_goal
THEN 'Chelsea home win!'
WHEN away_team_api_id = 8455 AND home_team_goal < away_team_goal
THEN 'Chelsea away win!'
ELSE 'Loss or tie :-(' END AS outcome
FROM Match
LIMIT 10;
```

```
date                 home_team_api_id  away_team_api_id  outcome
-------------------  ----------------  ----------------  ---------------
2008-08-17 00:00:00  9987              9993              Loss or tie :-(
2008-08-16 00:00:00  10000             9994              Loss or tie :-(
2008-08-16 00:00:00  9984              8635              Loss or tie :-(
2008-08-17 00:00:00  9991              9998              Loss or tie :-(
2008-08-16 00:00:00  7947              9985              Loss or tie :-(
2008-09-24 00:00:00  8203              8342              Loss or tie :-(
2008-08-16 00:00:00  9999              8571              Loss or tie :-(
2008-08-16 00:00:00  4049              9996              Loss or tie :-(
2008-08-16 00:00:00  10001             9986              Loss or tie :-(
2008-11-01 00:00:00  8342              8571              Loss or tie :-(
```

- [X]

  What's `NULL`? The following two queries return identical results. When removing the `ELSE` clause,

```
SELECT date,
CASE WHEN date > '2015-01-01' THEN 'More Recently'
WHEN date < '2012-01-01' THEN 'Older'
END AS date_category
FROM Match
LIMIT 4;
```

```
date                 date_category
-------------------  -------------
2008-08-17 00:00:00  Older
2008-08-16 00:00:00  Older
2008-08-16 00:00:00  Older
2008-08-17 00:00:00  Older
```

```
SELECT date,
CASE WHEN date > '2015-01-01' THEN 'More Recently'
WHEN date < '2012-01-01' THEN 'Older'
ELSE NULL END AS date_category
FROM Match
LIMIT 4;
```

```
date                 date_category
-------------------  -------------
2008-08-17 00:00:00  Older
2008-08-16 00:00:00  Older
2008-08-16 00:00:00  Older
2008-08-17 00:00:00  Older
```

- [X]

  Chelsea example without the ELSE clause shows a lot of NULL values. If ELSE is missing, outcome values are replaced by NULL.

```
-- SQLite command to show NULL instead of ""
.nullvalue "[NULL]"

SELECT date, home_team_api_id, away_team_api_id,
CASE WHEN home_team_api_id = 8455 AND home_team_goal > away_team_goal
THEN 'Chelsea home win!'
WHEN away_team_api_id = 8455 AND home_team_goal < away_team_goal
THEN 'Chelsea away win!'
END AS outcome
FROM Match
WHERE home_team_api_id = 8455 OR away_team_api_id = 8455
LIMIT 10;
```

```
date                 home_team_api_id  away_team_api_id  outcome
-------------------  ----------------  ----------------  -----------------
2008-08-17 00:00:00  8455              8462              Chelsea home win!
2008-10-29 00:00:00  8667              8455              Chelsea away win!
2008-11-01 00:00:00  8455              8472              Chelsea home win!
2008-11-09 00:00:00  8655              8455              Chelsea away win!
2008-11-15 00:00:00  8659              8455              Chelsea away win!
2008-11-22 00:00:00  8455              10261             [NULL]
2008-11-30 00:00:00  8455              9825              [NULL]
2008-12-06 00:00:00  8559              8455              Chelsea away win!
2008-12-14 00:00:00  8455              8654              [NULL]
2008-12-22 00:00:00  8668              8455              [NULL]
```

- [X]

  You can filter a query by a CASE statement by placing the entire CASE section after the WHERE, and specifying to exclude rows where the NOT NULL condition is not met.

```
-- SQLite command to show NULL instead of ""
.nullvalue "[NULL]"

SELECT date, home_team_api_id, away_team_api_id,
CASE WHEN home_team_api_id = 8455 AND home_team_goal > away_team_goal
THEN 'Chelsea home win!'
WHEN away_team_api_id = 8455 AND home_team_goal < away_team_goal
THEN 'Chelsea away win!' END AS outcome
FROM Match
WHERE CASE WHEN home_team_api_id = 8455 AND home_team_goal > away_team_goal
THEN 'Chelsea home win!'
WHEN away_team_api_id = 8455 AND home_team_goal < away_team_goal
THEN 'Chelsea away win!' END IS NOT NULL
LIMIT 10;
```

```
date                 home_team_api_id  away_team_api_id  outcome
-------------------  ----------------  ----------------  -----------------
2008-08-17 00:00:00  8455              8462              Chelsea home win!
2008-10-29 00:00:00  8667              8455              Chelsea away win!
2008-11-01 00:00:00  8455              8472              Chelsea home win!
2008-11-09 00:00:00  8655              8455              Chelsea away win!
2008-11-15 00:00:00  8659              8455              Chelsea away win!
2008-12-06 00:00:00  8559              8455              Chelsea away win!
2008-12-26 00:00:00  8455              8659              Chelsea home win!
2008-08-24 00:00:00  8528              8455              Chelsea away win!
2009-01-17 00:00:00  8455              10194             Chelsea home win!
2009-01-28 00:00:00  8455              8549              Chelsea home win!
```

Exercises

In CASE of rivalry

- Query a list of matches played between Barcelona and Real Madrid.
- In this exercise, you will retrieve information about matches played between Barcelona (id = 8634) and Real pMadrid (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 home_team_api_id = 8634 THEN 'FC Barcelona'
ELSE 'Real Madrid CF' END AS home,
-- Identify the away team as Barcelona or Real Madrid
CASE WHEN away_team_api_id = 8634 THEN 'FC Barcelona'
ELSE 'Real Madrid CF' END AS away
FROM Match
WHERE (away_team_api_id = 8634 OR home_team_api_id = 8634)
AND (away_team_api_id = 8633 OR home_team_api_id = 8633)
AND country_id=21518
LIMIT 5;
```

```
date                 home            away
-------------------  --------------  --------------
2008-12-13 00:00:00  FC Barcelona    Real Madrid CF
2009-05-02 00:00:00  Real Madrid CF  FC Barcelona
2009-11-29 00:00:00  FC Barcelona    Real Madrid CF
2010-04-10 00:00:00  Real Madrid CF  FC Barcelona
2010-11-29 00:00:00  FC Barcelona    Real Madrid CF
```

- [ ]

  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.

  Q: what are the combinatorics? Two places, three values, no replacement.

```
SELECT date,
CASE WHEN home_team_api_id = 8634 THEN 'FC Barcelona'
ELSE 'Real Madrid CF' END AS home,
CASE WHEN away_team_api_id = 8634 THEN 'FC Barcelona'
ELSE 'Real Madrid CF' END AS away,
CASE WHEN home_team_goal > away_team_goal AND home_team_api_id = 8634
THEN 'Barcelona win!'
WHEN home_team_goal > away_team_goal AND home_team_api_id = 8633
THEN 'Real Madrid win!'
WHEN home_team_goal < away_team_goal AND away_team_api_id = 8634
THEN 'Barcelona win!'
WHEN home_team_goal < away_team_goal AND away_team_api_id = 8633
THEN 'Real Madrid win!'
ELSE 'Tie!' END AS outcome
FROM Match
WHERE (away_team_api_id = 8634 OR home_team_api_id = 8634)
AND (away_team_api_id = 8633 OR home_team_api_id = 8633)
AND country_id=21518
LIMIT 10;
```

```
date                 home            away            outcome
-------------------  --------------  --------------  ----------------
2008-12-13 00:00:00  FC Barcelona    Real Madrid CF  Barcelona win!
2009-05-02 00:00:00  Real Madrid CF  FC Barcelona    Barcelona win!
2009-11-29 00:00:00  FC Barcelona    Real Madrid CF  Barcelona win!
2010-04-10 00:00:00  Real Madrid CF  FC Barcelona    Barcelona win!
2010-11-29 00:00:00  FC Barcelona    Real Madrid CF  Barcelona win!
2011-04-16 00:00:00  Real Madrid CF  FC Barcelona    Tie!
2011-12-10 00:00:00  Real Madrid CF  FC Barcelona    Barcelona win!
2012-04-21 00:00:00  FC Barcelona    Real Madrid CF  Real Madrid win!
2013-03-02 00:00:00  Real Madrid CF  FC Barcelona    Real Madrid win!
2012-10-07 00:00:00  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.

- 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**.

```
SELECT *
FROM table
WHERE
CASE WHEN a > 5 THEN 'Keep'
     WHEN a <= 5 THEN 'Exclude' END = 'Keep';
```

- `[ ]`

  Identify Bologna's team ID listed in the teams_italy table by selecting the team_long_name and team_api_id.

Get the equivalent attributes from the schema for `Team`.

```
.schema Team
```

```
CREATE TABLE IF NOT EXISTS "Team" (
        `id`    INTEGER PRIMARY KEY AUTOINCREMENT,
        `team_api_id`   INTEGER UNIQUE,
        `team_fifa_api_id`      INTEGER,
        `team_long_name`        TEXT,
        `team_short_name`       TEXT
);
```

Solution:

```
SELECT
t.team_long_name,
t.team_api_id
FROM Team AS t
WHERE t.team_long_name = 'Bologna';
```

```
team_long_name  team_api_id
--------------  -----------
Bologna         9857
```

- [ ]

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

Get the equivalent attributes from the schema for `Match`.

```
.schema Match
```

```
CREATE TABLE `Match` (
        `id`    INTEGER PRIMARY KEY AUTOINCREMENT,
        `country_id`    INTEGER,
        `league_id`     INTEGER,
        `season`        TEXT,
        `stage` INTEGER,
        `date`  TEXT,
        `match_api_id`  INTEGER UNIQUE,
        `home_team_api_id`      INTEGER,
        `away_team_api_id`      INTEGER,
        `home_team_goal`        INTEGER,
        `away_team_goal`        INTEGER,
        `home_player_X1`        INTEGER,
        `home_player_X2`        INTEGER,
        `home_player_X3`        INTEGER,
        `home_player_X4`        INTEGER,
        `home_player_X5`        INTEGER,
        `home_player_X6`        INTEGER,
        `home_player_X7`        INTEGER,
        `home_player_X8`        INTEGER,
        `home_player_X9`        INTEGER,
```

```
        `home_player_X10`       INTEGER,
        `home_player_X11`       INTEGER,
        `away_player_X1`        INTEGER,
        `away_player_X2`        INTEGER,
        `away_player_X3`        INTEGER,
        `away_player_X4`        INTEGER,
        `away_player_X5`        INTEGER,
        `away_player_X6`        INTEGER,
        `away_player_X7`        INTEGER,
        `away_player_X8`        INTEGER,
        `away_player_X9`        INTEGER,
        `away_player_X10`       INTEGER,
        `away_player_X11`       INTEGER,
        `home_player_Y1`        INTEGER,
        `home_player_Y2`        INTEGER,
        `home_player_Y3`        INTEGER,
        `home_player_Y4`        INTEGER,
        `home_player_Y5`        INTEGER,
        `home_player_Y6`        INTEGER,
        `home_player_Y7`        INTEGER,
        `home_player_Y8`        INTEGER,
        `home_player_Y9`        INTEGER,
        `home_player_Y10`       INTEGER,
        `home_player_Y11`       INTEGER,
        `away_player_Y1`        INTEGER,
        `away_player_Y2`        INTEGER,
        `away_player_Y3`        INTEGER,
        `away_player_Y4`        INTEGER,
        `away_player_Y5`        INTEGER,
        `away_player_Y6`        INTEGER,
        `away_player_Y7`        INTEGER,
        `away_player_Y8`        INTEGER,
        `away_player_Y9`        INTEGER,
        `away_player_Y10`       INTEGER,
        `away_player_Y11`       INTEGER,
        `home_player_1` INTEGER,
        `home_player_2` INTEGER,
        `home_player_3` INTEGER,
        `home_player_4` INTEGER,
        `home_player_5` INTEGER,
        `home_player_6` INTEGER,
        `home_player_7` INTEGER,
        `home_player_8` INTEGER,
        `home_player_9` INTEGER,
        `home_player_10`        INTEGER,
        `home_player_11`        INTEGER,
        `away_player_1` INTEGER,
        `away_player_2` INTEGER,
        `away_player_3` INTEGER,
        `away_player_4` INTEGER,
        `away_player_5` INTEGER,
        `away_player_6` INTEGER,
        `away_player_7` INTEGER,
        `away_player_8` INTEGER,
        `away_player_9` INTEGER,
        `away_player_10`        INTEGER,
        `away_player_11`        INTEGER,
        `goal`  TEXT,
        `shoton`        TEXT,
        `shotoff`       TEXT,
        `foulcommit`    TEXT,
        `card`  TEXT,
        `cross` TEXT,
        `corner`        TEXT,
        `possession`    TEXT,
```

```
        `B365H` NUMERIC,
        `B365D` NUMERIC,
        `B365A` NUMERIC,
        `BWH`   NUMERIC,
        `BWD`   NUMERIC,
        `BWA`   NUMERIC,
        `IWH`   NUMERIC,
        `IWD`   NUMERIC,
        `IWA`   NUMERIC,
        `LBH`   NUMERIC,
        `LBD`   NUMERIC,
        `LBA`   NUMERIC,
        `PSH`   NUMERIC,
        `PSD`   NUMERIC,
        `PSA`   NUMERIC,
        `WHH`   NUMERIC,
        `WHD`   NUMERIC,
        `WHA`   NUMERIC,
        `SJH`   NUMERIC,
        `SJD`   NUMERIC,
        `SJA`   NUMERIC,
        `VCH`   NUMERIC,
        `VCD`   NUMERIC,
        `VCA`   NUMERIC,
        `GBH`   NUMERIC,
        `GBD`   NUMERIC,
        `GBA`   NUMERIC,
        `BSH`   NUMERIC,
        `BSD`   NUMERIC,
        `BSA`   NUMERIC,
        FOREIGN KEY(`country_id`) REFERENCES `country`(`id`),
        FOREIGN KEY(`league_id`) REFERENCES `League`(`id`),
        FOREIGN KEY(`home_team_api_id`) REFERENCES `Team`(`team_api_id`),
        FOREIGN KEY(`away_team_api_id`) REFERENCES `Team`(`team_api_id`),
        FOREIGN KEY(`home_player_1`) REFERENCES `Player`(`player_api_id`),
        FOREIGN KEY(`home_player_2`) REFERENCES `Player`(`player_api_id`),
        FOREIGN KEY(`home_player_3`) REFERENCES `Player`(`player_api_id`),
        FOREIGN KEY(`home_player_4`) REFERENCES `Player`(`player_api_id`),
        FOREIGN KEY(`home_player_5`) REFERENCES `Player`(`player_api_id`),
        FOREIGN KEY(`home_player_6`) REFERENCES `Player`(`player_api_id`),
        FOREIGN KEY(`home_player_7`) REFERENCES `Player`(`player_api_id`),
        FOREIGN KEY(`home_player_8`) REFERENCES `Player`(`player_api_id`),
        FOREIGN KEY(`home_player_9`) REFERENCES `Player`(`player_api_id`),
        FOREIGN KEY(`home_player_10`) REFERENCES `Player`(`player_api_id`),
        FOREIGN KEY(`home_player_11`) REFERENCES `Player`(`player_api_id`),
        FOREIGN KEY(`away_player_1`) REFERENCES `Player`(`player_api_id`),
        FOREIGN KEY(`away_player_2`) REFERENCES `Player`(`player_api_id`),
        FOREIGN KEY(`away_player_3`) REFERENCES `Player`(`player_api_id`),
        FOREIGN KEY(`away_player_4`) REFERENCES `Player`(`player_api_id`),
        FOREIGN KEY(`away_player_5`) REFERENCES `Player`(`player_api_id`),
        FOREIGN KEY(`away_player_6`) REFERENCES `Player`(`player_api_id`),
        FOREIGN KEY(`away_player_7`) REFERENCES `Player`(`player_api_id`),
        FOREIGN KEY(`away_player_8`) REFERENCES `Player`(`player_api_id`),
        FOREIGN KEY(`away_player_9`) REFERENCES `Player`(`player_api_id`),
        FOREIGN KEY(`away_player_10`) REFERENCES `Player`(`player_api_id`),
        FOREIGN KEY(`away_player_11`) REFERENCES `Player`(`player_api_id`)
);
```

```
.nullvalue "[NULL]"
SELECT
m.season, m.date,
CASE WHEN home_team_api_id = 9857 AND home_team_goal > away_team_goal
THEN 'Bologna Win'
```

```
WHEN away_team_api_id = 9857 AND home_team_goal < away_team_goal
THEN 'Bologna Win'
END AS outcome
FROM Match as m
WHERE country_id = 10257
LIMIT 8;
```

```
season     date                 outcome
---------  -------------------  -----------
2008/2009  2008-08-31 00:00:00  [NULL]
2008/2009  2008-08-31 00:00:00  [NULL]
2008/2009  2008-08-31 00:00:00  [NULL]
2008/2009  2008-08-31 00:00:00  [NULL]
2008/2009  2008-08-31 00:00:00  [NULL]
2008/2009  2008-08-31 00:00:00  Bologna Win
2008/2009  2008-08-31 00:00:00  [NULL]
2008/2009  2008-08-30 00:00:00  [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.

```
.nullvalue "[NULL]"
SELECT
m.season, m.date, home_team_goal, away_team_goal
FROM Match as m
WHERE country_id = 10257 AND CASE
      WHEN home_team_api_id = 9857 AND home_team_goal > away_team_goal
      THEN 'Bologna Win'
      WHEN away_team_api_id = 9857 AND home_team_goal < away_team_goal
      THEN 'Bologna Win'
      END IS NOT NULL
LIMIT 10;
```

```
season     date                 home_team_goal  away_team_goal
---------  -------------------  --------------  --------------
2008/2009  2008-08-31 00:00:00  1               2
2008/2009  2008-12-13 00:00:00  5               2
2008/2009  2009-01-18 00:00:00  1               2
2008/2009  2009-01-28 00:00:00  0               1
2008/2009  2009-03-08 00:00:00  3               0
2008/2009  2009-04-26 00:00:00  2               0
2008/2009  2009-05-17 00:00:00  2               1
2008/2009  2009-05-31 00:00:00  3               1
2008/2009  2008-10-19 00:00:00  3               1
2009/2010  2009-10-28 00:00:00  2               1
```

`CASE WHEN` with aggregate functions

Schema check

- `[X]`

  Check the `Match` and `Country` schemas for alignment between column names (PostgreSQL db different from SQLite db).

```
.schema Match
```

```
CREATE TABLE `Match` (
        `id`    INTEGER PRIMARY KEY AUTOINCREMENT,
        `country_id`    INTEGER,
        `league_id`     INTEGER,
        `season`        TEXT,
        `stage` INTEGER,
        `date`  TEXT,
        `match_api_id`  INTEGER UNIQUE,
        `home_team_api_id`      INTEGER,
        `away_team_api_id`      INTEGER,
        `home_team_goal`        INTEGER,
        `away_team_goal`        INTEGER,
        `home_player_X1`        INTEGER,
        `home_player_X2`        INTEGER,
        `home_player_X3`        INTEGER,
        `home_player_X4`        INTEGER,
        `home_player_X5`        INTEGER,
        `home_player_X6`        INTEGER,
        `home_player_X7`        INTEGER,
        `home_player_X8`        INTEGER,
        `home_player_X9`        INTEGER,
        `home_player_X10`       INTEGER,
        `home_player_X11`       INTEGER,
        `away_player_X1`        INTEGER,
        `away_player_X2`        INTEGER,
        `away_player_X3`        INTEGER,
        `away_player_X4`        INTEGER,
        `away_player_X5`        INTEGER,
        `away_player_X6`        INTEGER,
        `away_player_X7`        INTEGER,
        `away_player_X8`        INTEGER,
        `away_player_X9`        INTEGER,
        `away_player_X10`       INTEGER,
        `away_player_X11`       INTEGER,
        `home_player_Y1`        INTEGER,
        `home_player_Y2`        INTEGER,
        `home_player_Y3`        INTEGER,
        `home_player_Y4`        INTEGER,
        `home_player_Y5`        INTEGER,
        `home_player_Y6`        INTEGER,
        `home_player_Y7`        INTEGER,
        `home_player_Y8`        INTEGER,
        `home_player_Y9`        INTEGER,
        `home_player_Y10`       INTEGER,
        `home_player_Y11`       INTEGER,
        `away_player_Y1`        INTEGER,
        `away_player_Y2`        INTEGER,
        `away_player_Y3`        INTEGER,
        `away_player_Y4`        INTEGER,
        `away_player_Y5`        INTEGER,
        `away_player_Y6`        INTEGER,
        `away_player_Y7`        INTEGER,
        `away_player_Y8`        INTEGER,
        `away_player_Y9`        INTEGER,
        `away_player_Y10`       INTEGER,
        `away_player_Y11`       INTEGER,
        `home_player_1` INTEGER,
        `home_player_2` INTEGER,
        `home_player_3` INTEGER,
        `home_player_4` INTEGER,
```

```
        `home_player_5` INTEGER,
        `home_player_6` INTEGER,
        `home_player_7` INTEGER,
        `home_player_8` INTEGER,
        `home_player_9` INTEGER,
        `home_player_10`        INTEGER,
        `home_player_11`        INTEGER,
        `away_player_1` INTEGER,
        `away_player_2` INTEGER,
        `away_player_3` INTEGER,
        `away_player_4` INTEGER,
        `away_player_5` INTEGER,
        `away_player_6` INTEGER,
        `away_player_7` INTEGER,
        `away_player_8` INTEGER,
        `away_player_9` INTEGER,
        `away_player_10`        INTEGER,
        `away_player_11`        INTEGER,
        `goal`  TEXT,
        `shoton`        TEXT,
        `shotoff`       TEXT,
        `foulcommit`    TEXT,
        `card`  TEXT,
        `cross` TEXT,
        `corner`        TEXT,
        `possession`    TEXT,
        `B365H` NUMERIC,
        `B365D` NUMERIC,
        `B365A` NUMERIC,
        `BWH`   NUMERIC,
        `BWD`   NUMERIC,
        `BWA`   NUMERIC,
        `IWH`   NUMERIC,
        `IWD`   NUMERIC,
        `IWA`   NUMERIC,
        `LBH`   NUMERIC,
        `LBD`   NUMERIC,
        `LBA`   NUMERIC,
        `PSH`   NUMERIC,
        `PSD`   NUMERIC,
        `PSA`   NUMERIC,
        `WHH`   NUMERIC,
        `WHD`   NUMERIC,
        `WHA`   NUMERIC,
        `SJH`   NUMERIC,
        `SJD`   NUMERIC,
        `SJA`   NUMERIC,
        `VCH`   NUMERIC,
        `VCD`   NUMERIC,
        `VCA`   NUMERIC,
        `GBH`   NUMERIC,
        `GBD`   NUMERIC,
        `GBA`   NUMERIC,
        `BSH`   NUMERIC,
        `BSD`   NUMERIC,
        `BSA`   NUMERIC,
        FOREIGN KEY(`country_id`) REFERENCES `country`(`id`),
        FOREIGN KEY(`league_id`) REFERENCES `League`(`id`),
        FOREIGN KEY(`home_team_api_id`) REFERENCES `Team`(`team_api_id`),
        FOREIGN KEY(`away_team_api_id`) REFERENCES `Team`(`team_api_id`),
        FOREIGN KEY(`home_player_1`) REFERENCES `Player`(`player_api_id`),
        FOREIGN KEY(`home_player_2`) REFERENCES `Player`(`player_api_id`),
        FOREIGN KEY(`home_player_3`) REFERENCES `Player`(`player_api_id`),
        FOREIGN KEY(`home_player_4`) REFERENCES `Player`(`player_api_id`),
        FOREIGN KEY(`home_player_5`) REFERENCES `Player`(`player_api_id`),
```

```
        FOREIGN KEY(`home_player_6`) REFERENCES `Player`(`player_api_id`),
        FOREIGN KEY(`home_player_7`) REFERENCES `Player`(`player_api_id`),
        FOREIGN KEY(`home_player_8`) REFERENCES `Player`(`player_api_id`),
        FOREIGN KEY(`home_player_9`) REFERENCES `Player`(`player_api_id`),
        FOREIGN KEY(`home_player_10`) REFERENCES `Player`(`player_api_id`),
        FOREIGN KEY(`home_player_11`) REFERENCES `Player`(`player_api_id`),
        FOREIGN KEY(`away_player_1`) REFERENCES `Player`(`player_api_id`),
        FOREIGN KEY(`away_player_2`) REFERENCES `Player`(`player_api_id`),
        FOREIGN KEY(`away_player_3`) REFERENCES `Player`(`player_api_id`),
        FOREIGN KEY(`away_player_4`) REFERENCES `Player`(`player_api_id`),
        FOREIGN KEY(`away_player_5`) REFERENCES `Player`(`player_api_id`),
        FOREIGN KEY(`away_player_6`) REFERENCES `Player`(`player_api_id`),
        FOREIGN KEY(`away_player_7`) REFERENCES `Player`(`player_api_id`),
        FOREIGN KEY(`away_player_8`) REFERENCES `Player`(`player_api_id`),
        FOREIGN KEY(`away_player_9`) REFERENCES `Player`(`player_api_id`),
        FOREIGN KEY(`away_player_10`) REFERENCES `Player`(`player_api_id`),
        FOREIGN KEY(`away_player_11`) REFERENCES `Player`(`player_api_id`)
);
```

```
.schema Country
```

```
CREATE TABLE `Country` (
        `id`    INTEGER PRIMARY KEY AUTOINCREMENT,
        `name`  TEXT UNIQUE
);
```

COUNT

- [X]

  Question: "How many home and away **wins** did Liverpool score in each season?"

  To answer this, you can use a `CASE WHEN` condition as an argument of `COUNT`.

```
SELECT
season,
COUNT(CASE WHEN home_team_api_id = 8650 -- Liverpool
AND home_team_goal > away_team_goal
THEN id END) AS home_team_wins,
COUNT(CASE WHEN home_team_api_id = 8650
AND away_team_goal > home_team_goal
THEN id END) AS away_team_wins
FROM Match
GROUP BY season;
```

```
season     home_team_wins  away_team_wins
---------  --------------  --------------
2008/2009  12              0
2009/2010  13              3
2010/2011  12              3
2011/2012  6               4
2012/2013  9               4
2013/2014  16              2
2014/2015  10              4
2015/2016  8               3
```

- [X]

  Trying the same thing with WHERE. This will work for one or the other condition, but not for both of them in the same statement.

```
SELECT
season,
COUNT(*) as home_team_wins
FROM Match
WHERE home_team_api_id = 8650
AND home_team_goal > away_team_goal
GROUP BY season;
```

```
season     home_team_wins
---------  --------------
2008/2009  12
2009/2010  13
2010/2011  12
2011/2012  6
2012/2013  9
2013/2014  16
2014/2015  10
2015/2016  8
```

- [X]

  Not equivalent:

```
SELECT
season,
COUNT(*) as "home_team_wins or away_team_wins"
FROM Match
WHERE
(home_team_api_id = 8650
AND home_team_goal > away_team_goal) OR
(away_team_api_id = 8650
AND home_team_goal < away_team_goal)
GROUP BY season;
```

```
season     home_team_wins or away_team_wins
---------  --------------------------------
2008/2009  25
2009/2010  18
2010/2011  17
2011/2012  14
2012/2013  16
2013/2014  26
2014/2015  18
2015/2016  16
```

SUM

- [X]

  Question: "Number of home and away **goals** that Liverpool scored in each season?"

```
SELECT
season,
SUM(CASE WHEN home_team_api_id = 8650 -- Liverpool
THEN home_team_goal END) AS home_goals,
SUM(CASE WHEN away_team_api_id = 8650
THEN away_team_goal END) AS away_goals
FROM Match
GROUP BY season;
```

```
season     home_goals  away_goals
---------  ----------  ----------
2008/2009  41          36
2009/2010  43          18
2010/2011  37          22
2011/2012  24          23
2012/2013  33          38
2013/2014  53          48
2014/2015  30          22
2015/2016  33          30
```

AVG and percentages

- [ ]

  "How many goals did Liverpool score on average per season?"

```
SELECT
season,
AVG(CASE WHEN home_team_api_id = 8650 -- Liverpool
THEN home_team_goal END) AS avg_home_goals,
AVG(CASE WHEN away_team_api_id = 8650
THEN away_team_goal END) AS avg_away_goals
FROM Match
GROUP BY season;
```

```
season     avg_home_goals    avg_away_goals
---------  ----------------  ----------------
2008/2009  2.15789473684211  1.89473684210526
2009/2010  2.26315789473684  0.947368421052632
2010/2011  1.94736842105263  1.15789473684211
2011/2012  1.26315789473684  1.21052631578947
2012/2013  1.73684210526316  2.0
2013/2014  2.78947368421053  2.52631578947368
2014/2015  1.57894736842105  1.15789473684211
2015/2016  1.73684210526316  1.57894736842105
```

- [ ]

  Same question but rounded output (to 2 decimals) with ROUND.

```
SELECT
season,
ROUND(AVG(CASE WHEN home_team_api_id = 8650 -- Liverpool
THEN home_team_goal END), 2) AS avg_home_goals,
ROUND(AVG(CASE WHEN away_team_api_id = 8650
```

```
THEN away_team_goal END), 2) AS avg_away_goals
FROM Match
GROUP BY season;
```

```
season     avg_home_goals  avg_away_goals
---------  --------------  --------------
2008/2009  2.16            1.89
2009/2010  2.26            0.95
2010/2011  1.95            1.16
2011/2012  1.26            1.21
2012/2013  1.74            2.0
2013/2014  2.79            2.53
2014/2015  1.58            1.16
2015/2016  1.74            1.58
```

- [ ]

  "Which percentage of home/away games did Liverpool win?"

```
SELECT
season,
ROUND(AVG(CASE WHEN home_team_api_id = 8650 AND home_team_goal > away_team_goal THEN 1
WHEN home_team_api_id = 8650 AND home_team_goal < away_team_goal THEN 0
END), 2) * 100 AS pct_home_team_wins,
ROUND(AVG(CASE WHEN away_team_api_id = 8650 AND home_team_goal < away_team_goal THEN 1
WHEN away_team_api_id = 8650 AND home_team_goal > away_team_goal THEN 0
END), 2) * 100 AS pct_away_team_wins
FROM Match
GROUP BY season;
```

```
season     pct_home_team_wins  pct_away_team_wins
---------  ------------------  ------------------
2008/2009  100.0               87.0
2009/2010  81.0                38.0
2010/2011  80.0                31.0
2011/2012  60.0                44.0
2012/2013  69.0                58.0
2013/2014  89.0                71.0
2014/2015  71.0                50.0
2015/2016  73.0                53.0
```

Exercises

`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.

- [X]

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 statement in a COUNT function and group the query by the country alias.

```
SELECT
c.name AS country,
COUNT -- count games from 2012/13 season
(CASE WHEN m.season = '2012/2013' THEN m.id ELSE NULL END) AS matches_12_13
FROM Country as c LEFT JOIN Match as m ON c.id = m.country_id
-- group by country
GROUP BY country;
```

```
country      matches_12_13
-----------  -------------
Belgium      240
England      380
France       380
Germany      306
Italy        380
Netherlands  306
Poland       240
Portugal     240
Scotland     228
Spain        380
Switzerland  180
```

- [ ]

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
(CASE WHEN m.season = '2012/2013' THEN m.id END) AS matches_12_13,
COUNT
(CASE WHEN m.season = '2013/2014' THEN m.id END) AS matches_13_14,
COUNT
(CASE WHEN m.season = '2014/2015' THEN m.id END) AS matches_14_15
FROM Country as c LEFT JOIN Match as m ON c.id = m.country_id
-- group by country
GROUP BY country;
```

```
country      matches_12_13  matches_13_14  matches_14_15
-----------  -------------  -------------  -------------
Belgium      240            12             240
England      380            380            380
France       380            380            380
Germany      306            306            306
Italy        380            380            379
Netherlands  306            306            306
Poland       240            240            240
Portugal     240            240            306
Scotland     228            228            228
Spain        380            380            380
Switzerland  180            180            180
```

`COUNT` using `CASE WHEN` with multiple conditions

- In R or Python, you have the ability to calculate a SUM of logical values (i.e., TRUE/FALSE) directly. In SQL, you have to convert these values into 1 and 0 before calculating a sum. This can be done using a CASE statement.
- Your goal here is to use the country and match table to determine the total number of matches won by the home team in each country during the 2012/2013, 2013/2014, and 2014/2015 seasons.
- `[X]`

  Create 3 CASE statements to "count" matches in the '2012/2013', '2013/2014', and '2014/2015' seasons, resp.

  - 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
(CASE WHEN m.season = '2012/2013' AND m.home_team_goal > m.away_team_goal
THEN 1 ELSE 0 END) AS matches_12_13,
SUM
(CASE WHEN m.season = '2013/2014' THEN 1 ELSE 0 END) AS matches_13_14,
SUM
(CASE WHEN m.season = '2014/2015' THEN 1 ELSE 0 END) AS matches_14_15
FROM Country as c LEFT JOIN Match as m ON c.id = m.country_id
-- group by country
GROUP BY country;
```

```
country      matches_12_13  matches_13_14  matches_14_15
-----------  -------------  -------------  -------------
Belgium      102            12             240
England      166            380            380
France       170            380            380
Germany      130            306            306
Italy        177            380            379
Netherlands  137            306            306
Poland       97             240            240
Portugal     103            240            306
Scotland     89             228            228
Spain        189            380            380
Switzerland  84             180            180
```

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.

- In the code chunk below, the average is computed over any target column.

```
AVG(CASE WHEN condition_is_met THEN 1
         WHEN condition_is_not_met THEN 0 END)
```

- Your task is to examine the number of wins, losses, and ties in each country.
- [ ]

  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. The `Match` table is filtered to include all matches from the 2013/2014 and 2014/2015 seasons.

```
SELECT
c.name AS country,
COUNT
(CASE WHEN (m.season = '2013/2014' OR m.season = '2014/2015') AND
m.home_team_goal > m.away_team_goal THEN m.id END) AS home_wins,
COUNT
(CASE WHEN (m.season = '2013/2014' OR m.season = '2014/2015') AND
m.home_team_goal < m.away_team_goal THEN m.id END) AS away_wins,
COUNT
(CASE WHEN (m.season = '2013/2014' OR m.season = '2014/2015') AND
m.home_team_goal = m.away_team_goal THEN m.id END) AS ties
FROM Country as c LEFT JOIN Match as m ON c.id = m.country_id
GROUP BY country;
```

```
country      home_wins  away_wins  ties
-----------  ---------  ---------  ----
Belgium      112        78         62
England      351        238        171
France       349        215        196
Germany      290        176        146
Italy        333        216        210
Netherlands  282        173        157
Poland       224        117        139
Portugal     245        156        145
Scotland     204        158        94
Spain        350        233        177
Switzerland  158        113        89
```

- [X]

  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.
  - Use the ROUND function to round to 2 decimal points.

```
SELECT
c.name AS country,
ROUND (
AVG (CASE WHEN m.season = '2013/2014' AND m.home_team_goal  = m.away_team_goal THEN 1
WHEN m.season = '2013/2014' AND m.home_team_goal != m.away_team_goal THEN 0
END),2) * 100 AS ties_13_14,
ROUND (
AVG (CASE WHEN m.season = '2014/2015' AND m.home_team_goal  = m.away_team_goal THEN 1
WHEN m.season = '2014/2015' AND m.home_team_goal != m.away_team_goal THEN 0
END),2) * 100 AS ties_14_15
FROM Country as c LEFT JOIN Match as m ON c.id = m.country_id
GROUP BY country;
```

```
country      ties_13_14  ties_14_15
-----------  ----------  ----------
Belgium      17.0        25.0
England      21.0        24.0
France       28.0        23.0
Germany      21.0        27.0
Italy        24.0        32.0
Netherlands  27.0        24.0
Poland       30.0        28.0
Portugal     25.0        28.0
Scotland     22.0        19.0
Spain        23.0        24.0
Switzerland  23.0        27.0
```

TODO Short and Simple Subqueries

Practice

These are examples of DataCamp practice exercises reworked for SQLite. You can find these and many others in the Practice menu on the DataCamp dashboard. Link to Intermediate SQL practice.

Subquery joins

The match column contains 22 columns with the ID of each home and away team player. How might you set up a query to start identifying the first two home team players?

```
SELECT
p1.date,
p1.player_name AS home_pl1,
p2.player_name AS home_pl2
FROM (
SELECT m.id, p.player_name, m.date
FROM match AS m
INNER JOIN player as p
ON m.home_player_1 = p.player_api_id) as p1
INNER JOIN (
SELECT m.id, p.player_name
FROM match AS m
INNER JOIN player as p
ON m.home_player_2 = p.player_api_id) as p2
ON p1.id = p2.id
LIMIT 5;
```

```
date                 home_pl1             home_pl2
-------------------  -------------------  ------------------
2009-02-27 00:00:00  Wouter Biebauw       Kenny van Hoevelen
2009-03-01 00:00:00  Boubacar Barry Copa  Olivier Doll
2009-02-28 00:00:00  Stijn Stijnen        Koen Daerden
2009-02-28 00:00:00  Silvio Proto         Pieterjan Monteyne
2009-02-28 00:00:00  Cedric Berthelin     Eric Deflandre
```

Feedback: the player_name is stored in the table player, the corresponding player number is stored in match. The subqueries extract two sub-tables and link them as an INNER JOIN.

Subquery filter

- What is the correct way to use a subquery to filter the query for `player_id` taller than 175cm? Here's a preview of the players table:

```
player_name             id    height  penalties
----------------------  ----  ------  ---------
Khadare Guirassy Abdou  5812  175.26  64
Lukas Zelenka           6406  175.26  72
```

```
SELECT DISTINCT p.player_name, p.id, p.height, pa.penalties
FROM Player AS p INNER JOIN Player_Attributes AS pa
ON p.player_api_id = pa.player_api_id
WHERE player_name IN ("Khadare Guirassy Abdou", "Lukas Zelenka")
GROUP BY p.player_name;
```

```
player_name             id    height  penalties
----------------------  ----  ------  ---------
Khadare Guirassy Abdou  5812  175.26  64
Lukas Zelenka           6406  175.26  72
```

Correlated subqueries

- In a table of soccer matches, which matches have a total number of goals scored more than 3 times the average ? Can you determine this using a correlated subquery? Below is a preview of the match table.

```
country_id        date   home_goal  away_goal
        10     11/1/08           4          1
        20     11/8/08           0          0
```

Author: DataCamp (PostgreSQL version) / M Birkenkrahe (SQLite adaptation)
Created: 2022-04-26 Tue 22:53