Intermediate SQL - 1

Case Statements
CASE WHEN THEN ELSE END
- One Case statement evaluates to one column in the SQL query

CASE statements

- Contains a `WHEN` , `THEN` , and `ELSE` statement, finished with `END`

```
CASE WHEN x = 1 THEN 'a'
     WHEN x = 2 THEN 'b'
     ELSE 'c' END AS new_column
```

CASE WHEN

```
SELECT
  id,
  home_goal,
  away_goal,
  CASE WHEN home_goal > away_goal THEN 'Home Team Win'
       WHEN home_goal < away_goal THEN 'Away Team Win'
       ELSE 'Tie' END AS outcome
FROM match
WHERE season = '2013/2014';
```

```
| id   | home_goal | away_goal | outcome       |
|------|-----------|-----------|---------------|
| 1237 | 2         | 0         | Home Team Win |
| 1238 | 0         | 1         | Away Team Win |
| 1239 | 1         | 0         | Home Team Win |
| 1240 | 0         | 0         | Tie           |
```

-- Identify the home team as Bayern Munich, Schalke 04, or neither
SELECT
    CASE WHEN hometeam_id = 9789 THEN 'FC Schalke 04'

```
      WHEN hometeam_id = 9823 THEN 'FC Bayern Munich'
       ELSE 'Other' END AS home_team,
      COUNT(id) AS total_matches
FROM matches_germany
-- Group by the CASE statement alias
GROUP BY home_team;
```

- In the above case statement we are returning a column called home_team, which contains information regarding which team played the match.

CASE WHEN ... AND then some

- Add multiple logical conditions to your `WHEN` clause!

```
SELECT date, hometeam_id, awayteam_id,
  CASE WHEN hometeam_id = 8455 AND home_goal > away_goal
            THEN 'Chelsea home win!'
       WHEN awayteam_id = 8455 AND home_goal < away_goal
            THEN 'Chelsea away win!'
       ELSE 'Loss or tie :(' END AS outcome
FROM match
WHERE hometeam_id = 8455 OR awayteam_id = 8455;
```

```
| date       | hometeam_id | awayteam_id | outcome           |
|------------|-------------|-------------|-------------------|
| 2011-08-14 | 10194       | 8455        | Loss or tie :(    |
| 2011-08-20 | 8455        | 8659        | Chelsea home win! |
| 2011-08-27 | 8455        | 9850        | Chelsea home win! |
| 2011-09-10 | 8472        | 8455        | Chelsea away win! |
```

We can make use of an and clause inside the when statement to when we want multiple conditions.

What are your NULL values doing?

```
SELECT date, season,
  CASE WHEN hometeam_id = 8455 AND home_goal > away_goal
            THEN 'Chelsea home win!'
       WHEN awayteam_id = 8455 AND home_goal < away_goal
            THEN 'Chelsea away win!'
       END AS outcome
FROM match
WHERE hometeam_id = 8455 OR awayteam_id = 8455;
```

```
| date       | season    | outcome           |
|------------|-----------|-------------------|
| 2011-08-14 | 2011/2012 | NULL              |
| 2011-12-22 | 2011/2012 | NULL              |
| 2012-12-08 | 2012/2013 | Chelsea away win! |
| 2013-03-02 | 2012/2013 | Chelsea home win! |
```

We get NULL values, when Chelsea does not win the match in the above query, to make sure that we do not include NULL value rows, we will add a filter in the WHERE clause.

Where to place your CASE?

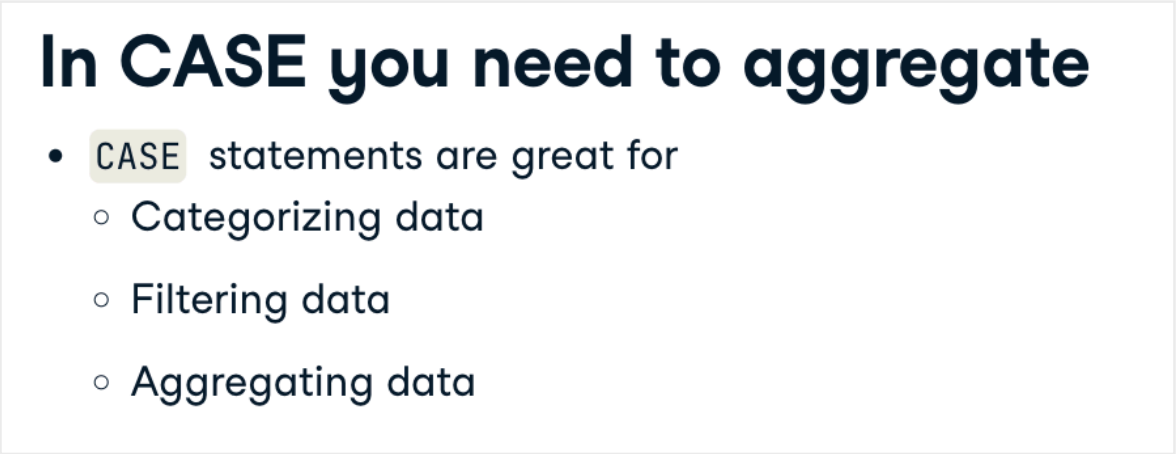
```
SELECT date, season,
  CASE WHEN hometeam_id = 8455 AND home_goal > away_goal
            THEN 'Chelsea home win!'
       WHEN awayteam_id = 8455 AND home_goal < away_goal
            THEN 'Chelsea away win!' END AS outcome
FROM match
WHERE CASE WHEN hometeam_id = 8455 AND home_goal > away_goal
               THEN 'Chelsea home win!'
          WHEN awayteam_id = 8455 AND home_goal < away_goal
               THEN 'Chelsea away win!' END IS NOT NULL;
```

Here, we have included the entire case statement inside the where statement and after the case end keyword we will specify the is not null condition.
The above modification should allow us to not consider any rows that have NULL

values in them.

```
| date       | season    | outcome           |
|------------|-----------|-------------------|
| 2011-11-05 | 2011/2012 | Chelsea away win! |
| 2011-11-26 | 2011/2012 | Chelsea home win! |
| 2011-12-03 | 2011/2012 | Chelsea away win! |
```

CASE WHEN with aggregate functions



Here, id corresponds to the unique match id, therefore, we will be counting the number of matches played by the team in each season, each season because of the group by clause.

CASE WHEN with COUNT

```
SELECT
    season,
    COUNT(CASE WHEN hometeam_id = 8650 AND home_goal > away_goal
          THEN 54321 END) AS home_wins,
    COUNT(CASE WHEN awayteam_id = 8650 AND away_goal > home_goal
          THEN 'Some random text' END) AS away_wins
FROM match
GROUP BY season;
```

```
| season    | home_wins | away_wins |
|-----------|-----------|-----------|
| 2011/2012 | 6         | 8         |
| 2012/2013 | 9         | 7         |
| 2013/2014 | 16        | 10        |
| 2014/2015 | 10        | 8         |
```

count function counts the number of non null rows, so it does not matter whether we return a string or integer or anything else as long as its not a null.

CASE WHEN with SUM

```
SELECT
    season,
    SUM(CASE WHEN hometeam_id = 8650
          THEN home_goal END) AS home_goals,
    SUM(CASE WHEN awayteam_id = 8650
          THEN away_goal END) AS away_goals
FROM match
GROUP BY season;
```

```
| season    | home_goals | away_goals |
|-----------|------------|------------|
| 2011/2012 | 24         | 23         |
| 2012/2013 | 33         | 38         |
| 2013/2014 | 53         | 48         |
| 2014/2015 | 30         | 22         |
```

The CASE is fairly AVG...

```
SELECT
    season,
    AVG(CASE WHEN hometeam_id = 8650
          THEN home_goal END) AS avg_homegoals,
    AVG(CASE WHEN awayteam_id = 8650
          THEN away_goal END) AS avg_awaygoals
FROM match
GROUP BY season;
```

```
| season    | avg_homegoals     | avg_awaygoals     |
|-----------|-------------------|-------------------|
| 2011/2012 | 1.26315789473684  | 1.21052631578947  |
| 2012/2013 | 1.73684210526316  | 2                 |
| 2013/2014 | 2.78947368421053  | 2.52631578947368  |
| 2014/2015 | 1.57894736842105  | 1.15789473684211  |
```

The CASE is fairly AVG...

```
SELECT
    season,
    AVG(CASE WHEN hometeam_id = 8650
          THEN home_goal END) AS avg_homegoals,
    AVG(CASE WHEN awayteam_id = 8650
          THEN away_goal END) AS avg_awaygoals
FROM match
GROUP BY season;
```

```
| season    | avg_homegoals    | avg_awaygoals    |
|-----------|------------------|------------------|
| 2011/2012 | 1.26315789473684 | 1.21052631578947 |
| 2012/2013 | 1.73684210526316 | 2                |
| 2013/2014 | 2.78947368421053 | 2.52631578947368 |
| 2014/2015 | 1.57894736842105 | 1.15789473684211 |
```

Percentages with CASE and AVG

```
SELECT
  season,
  ROUND(AVG(CASE WHEN hometeam_id = 8455 AND home_goal > away_goal THEN 1
            WHEN hometeam_id = 8455 AND home_goal < away_goal THEN 0
            END),2) AS pct_homewins,
  ROUND(AVG(CASE WHEN awayteam_id = 8455 AND away_goal > home_goal THEN 1
            WHEN awayteam_id = 8455 AND away_goal < home_goal THEN 0
            END),2) AS pct_awaywins
FROM match
GROUP BY season;
```

season	pct_homewins	pct_awaywins
2011/2012	0.75	0.5
2012/2013	0.86	0.67
2013/2014	0.94	0.67
2014/2015	1	0.79

SUBQUERY/ NESTED QUERY

What is a subquery?

- A query *nested* inside another query

```
SELECT column
FROM (SELECT column
      FROM table) AS subquery;
```

- Useful for intermediary transformations

What do you do with subqueries?

- Can be in *any* part of a query
  - `SELECT`, `FROM`, `WHERE`, `GROUP BY`
- Can return a variety of information
  - Scalar quantities (`3.14159`, `-2`, `0.001`)
  - A list (`id = (12, 25, 392, 401, 939)`)
  - A table

Why subqueries?

- Comparing groups to summarized values
  - *How did Liverpool compare to the English Premier League's average performance for that year?*
- Reshaping data
  - *What is the highest monthly average of goals scored in the Bundesliga?*
- Combining data that cannot be joined
  - *How do you get both the home and away team names into a table of match results?*

Simple subqueries

- Can be evaluated independently from the outer query

```
SELECT home_goal
FROM match
WHERE home_goal > (
    SELECT AVG(home_goal)
    FROM match);
SELECT AVG(home_goal) FROM match;
```

```
1.56091291478423
```

Subqueries in the WHERE clause

- Which matches in the 2012/2013 season scored home goals *higher than overall average*?

```
SELECT date, hometeam_id, awayteam_id, home_goal, away_goal
FROM match
WHERE season = '2012/2013'
      AND home_goal > (SELECT AVG(home_goal)
                       FROM match);
```

```
| date       | hometeam_id | awayteam_id | home_goal | away_goal |
|------------|-------------|-------------|-----------|-----------|
| 2012-07-28 | 9998        | 1773        | 5         | 2         |
| 2012-07-29 | 9987        | 9984        | 3         | 3         |
| 2012-10-05 | 9993        | 9991        | 2         | 2         |
```

FROM subqueries...

```
SELECT
  t.team_long_name AS team,
  AVG(m.home_goal) AS home_avg
FROM match AS m
LEFT JOIN team AS t
ON m.hometeam_id = t.team_api_id
WHERE season = '2011/2012'
GROUP BY team;
```

...to main queries!

```
SELECT team, home_avg
FROM (SELECT
          t.team_long_name AS team,
          AVG(m.home_goal) AS home_avg
      FROM match AS m
      LEFT JOIN team AS t
      ON m.hometeam_id = t.team_api_id
      WHERE season = '2011/2012'
      GROUP BY team) AS subquery
```

Things to remember

- You can create multiple subqueries in one `FROM` statement
  - Alias them!
  - Join them!
- You can join a subquery to a table in `FROM`
  - Include a joining columns in both tables!

Subqueries in SELECT

Subqueries in SELECT

```
SELECT
  date,
  (home_goal + away_goal) AS goals,
  (home_goal + away_goal) -
     (SELECT AVG(home_goal + away_goal)
      FROM match
      WHERE season = '2011/2012') AS diff
FROM match
WHERE season = '2011/2012';
```

```
| date       | goals | diff              |
|------------|-------|-------------------|
| 2011-07-29 | 3     | 0.28354037267081  |
| 2011-07-30 | 2     | -0.71645962732919 |
| 2011-07-30 | 4     | 1.28354037267081  |
| 2011-07-30 | 1     | -1.71645962732919 |
```

SELECT subqueries -- things to keep in m

- Need to return a **SINGLE** value
  - Will generate an error otherwise
- Make sure you have all filters in the right places
  - Properly filter **both** the main and the subquery!

```
SELECT
    date,
    (home_goal + away_goal) AS goals,
    (home_goal + away_goal) -
       (SELECT AVG(home_goal + away_goal)
        FROM match
        WHERE season = '2011/2012') AS diff
FROM match
WHERE season = '2011/2012';
```

Format your queries

- Line up `SELECT`, `FROM`, `WHERE`, and `GROUP BY`

```
SELECT
    col1,
    col2,
    col3
FROM table1
WHERE col1 = 2;
```

Annotate your queries

```
/* This query filters for col1 = 2
and only selects data from table1 */
SELECT
    col1,
    col2,
    col3
FROM table1
WHERE col1 = 2;
```

Annotate your queries

```
SELECT
    col1,
    col2,
    col3
FROM table1 -- this table has 10,000 rows
WHERE col1 = 2; -- Filter WHERE value 2
```

/* */ - Multiline comment
— Inline comment

Indent your queries

- Indent your subqueries!

```
SELECT
    col1,
    col2,
    col3
FROM table1
WHERE col1 IN
        (SELECT id
         FROM table2
         WHERE year = 1991);
```

Indent your queries

```
SELECT
  date,
  hometeam_id,
  awayteam_id,
  CASE WHEN hometeam_id = 8455 AND home_goal > away_goal
            THEN 'Chelsea home win'
       WHEN awayteam_id = 8455 AND home_goal < away_goal
            THEN 'Chelsea away win'
       WHEN hometeam_id = 8455 AND home_goal < away_goal
            THEN 'Chelsea home loss'
       WHEN awayteam_id = 8455 AND home_goal > away_goal
            THEN 'Chelsea away loss'
       WHEN (hometeam_id = 8455 OR awayteam_id = 8455)
            AND home_goal = away_goal THEN 'Chelsea Tie'
       END AS outcome
FROM match
WHERE hometeam_id = 8455 OR awayteam_id = 8455;
```

Is that subquery necessary?

- Subqueries require computing power
  - How big is your database?
  - How big is the table you're querying from?
- Is the subquery *actually* necessary?

Correlated subqueries

Correlated subqueries are subqueries that reference one or more columns in the main query. Correlated subqueries depend on information in the main query to run, and thus, cannot be executed on their own.
Correlated subqueries are evaluated in SQL once per row of data retrieved -- a process that takes a lot more computing power and time than a simple subquery.

Correlated subquery

- Uses values from the *outer* query to generate a result
- Re-run for every row generated in the final data set
- Used for advanced joining, filtering, and evaluating data

A correlated example

```
SELECT
    s.stage,
    ROUND(s.avg_goals,2) AS avg_goal,
    (SELECT AVG(home_goal + away_goal)
     FROM match
     WHERE season = '2012/2013') AS overall_avg
FROM
    (SELECT
         stage,
         AVG(home_goal + away_goal) AS avg_goals
     FROM match
     WHERE season = '2012/2013'
     GROUP BY stage) AS s
WHERE s.avg_goals > (SELECT AVG(home_goal + away_goal)
                     FROM match AS m
                     WHERE s.stage > m.stage);
```

In the example above we can observe that in the where clause we are using the result of the subquery of the from clause.

Simple vs. correlated subqueries

Simple Subquery

- Can be run *independently* from the main query
- Evaluated once in the whole query

Correlated Subquery

- *Dependent* on the main query to execute
- Evaluated in loops
  - **Significantly slows down query runtime**

Correlated subqueries

- *What is the average number of goals scored in each country?*

```sql
SELECT
  c.name AS country,
  (SELECT
     AVG(home_goal + away_goal)
   FROM match AS m
   WHERE m.country_id = c.id)
     AS avg_goals
FROM country AS c
GROUP BY country;
```

country	avg_goals
Belgium	2.89344262295082
England	2.76776315789474
France	2.51052631578947
Germany	2.94607843137255
Italy	2.63150867823765
Netherlands	3.14624183006536
Poland	2.49375
Portugal	2.63255360623782
Scotland	2.74122807017544
Spain	2.78223684210526
Switzerland	2.81054131054131

Nested subqueries?

- Subquery inside another subquery
- Perform multiple layers of transformation

```
SELECT
  EXTRACT(MONTH FROM date) AS month,
  SUM(m.home_goal + m.away_goal) AS total_goals,
  SUM(m.home_goal + m.away_goal) -
  (SELECT AVG(goals)
   FROM (SELECT
           EXTRACT(MONTH FROM date) AS month,
           SUM(home_goal + away_goal) AS goals
         FROM match
         GROUP BY month) AS s) AS diff
FROM match AS m
GROUP BY month;
```

```
| month | goals | diff     |
|-------|-------|----------|
| 01    | 5821  | -36.25   |
| 02    | 7448  | 1590.75  |
| 03    | 7298  | 1440.75  |
| 04    | 8145  | 2287.75  |
```

Note: The extract statement here is used to get from the date column, here date is in the date time format.

Correlated nested subqueries

- Nested subqueries can be correlated or uncorrelated
  - Or...a combination of the two
  - Can reference information from the *outer subquery* or *main query*