Welcome to Intermediate SQL!

INTERMEDIATE SQL

Mona Khalil
Data Scientist, Greenhouse Software

datacamp

Topics covered:

- CASE statements
- Simple subqueries
- Correlated subqueries
- Window functions

Prerequisites

- Selecting, filtering, and grouping data

```
SELECT user_id, SUM(sales)
FROM sales_data
WHERE user_id BETWEEN 300 AND 400
GROUP BY user_id;
```

- Using joins

```
SELECT c.country, c.team, SUM(m.goals)
FROM countries AS c
LEFT JOIN matches AS m
ON c.team_id = m.home_team_id
WHERE m.year > 1990
GROUP BY c.country, c.team;
```

Selecting from the European Soccer Database

```
SELECT
    l.name AS league,
    COUNT(m.country_id) as matches
FROM league AS l
LEFT JOIN match AS m
ON l.country_id = m.country_id
GROUP BY l.name;
```

```
| league                  | total_matches |
|-------------------------|---------------|
| Belgium Jupiler League  | 732           |
| England Premier League  | 1520          |
| France Ligue 1          | 1520          |
| Germany 1. Bundesliga   | 1224          |
```

Selecting from the European Soccer Database

```
SELECT
    date,
    id,
    home_goal,
    away_goal
FROM match
WHERE season = '2013/2014';
```

```
| date                | id   | home_goal | away_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         |
```

Selecting from the European Soccer Database

```
SELECT
    date,
    id,
    home_goal,
    away_goal
FROM match
WHERE season = '2013/2014'
      AND home_team_goal > away_team_goal;
```

```
| date                | id   | home_goal | away_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         |
```

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           |
```

Let's Practice!

INTERMEDIATE SQL

In CASE things get more complex

INTERMEDIATE SQL

Mona Khalil
Data Scientist, Greenhouse Software

datacamp

Reviewing CASE WHEN

```
SELECT
    date,
    season,
    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;
```

```
| date       | season    | outcome            |
|------------|-----------|--------------------|
| 2011-08-09 | 2011/2012 | Home team win!     |
| 2011-09-01 | 2011/2012 | Away team win!     |
| 2011-09-14 | 2011/2012 | Tie                |
| 2011-10-04 | 2011/2012 | Home team win!     |
```

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! |
```

What ELSE is being excluded?

- What's in your `ELSE` 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;
```

```
| date       | hometeam_id | awayteam_id | outcome        |
|------------|-------------|-------------|----------------|
| 2011-07-29 | 1773        | 8635        | Loss or tie :( |
| 2011-07-30 | 9998        | 9985        | Loss or tie :( |
| 2011-07-30 | 9987        | 9993        | Loss or tie :( |
| 2011-07-30 | 9991        | 9984        | Loss or tie :( |
```

Correctly categorize your data with CASE

```
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! |
```

What's NULL?

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

```
| date       | date_category |
|------------|---------------|
| 2011-11-18 | Older         |
| 2012-02-11 | NULL          |
| 2014-11-07 | NULL          |
| 2015-02-14 | More Recently |
```

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! |
```

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 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;
```

```
| 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! |
```

Let's Practice!

INTERMEDIATE SQL

CASE WHEN with aggregate functions

INTERMEDIATE SQL

Mona Khalil
Data Scientist, Greenhouse Software

datacamp

In CASE you need to aggregate

- `CASE` statements are great for
  - Categorizing data
  - Filtering data
  - Aggregating data

COUNTing CASES

- How many home and away goals did Liverpool score in each season?

season	home_wins	away_wins
2011/2012		
2012/2013		
2013/2014		
2014/2015		

CASE WHEN with COUNT

```
SELECT
    season,
    COUNT(CASE WHEN hometeam_id = 8650
               AND home_goal > away_goal
               THEN id END) AS home_wins
FROM match
GROUP BY season;
```

CASE WHEN with COUNT

```
SELECT
    season,
    COUNT(CASE WHEN hometeam_id = 8650 AND home_goal > away_goal
          THEN id END) AS home_wins,
    COUNT(CASE WHEN awayteam_id = 8650 AND away_goal > home_goal
          THEN id 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         |
```

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         |
```

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 home_goals,
    AVG(CASE WHEN awayteam_id = 8650
          THEN away_goal END) AS away_goals
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 |
```

A ROUNDed AVG

```
ROUND(3.141592653589,2)
```

```
3.14
```

A ROUNDed AVG

```
SELECT
  season,
  ROUND(AVG(CASE WHEN hometeam_id = 8650
        THEN home_goal END),2) AS home_goals,
  ROUND(AVG(CASE WHEN awayteam_id = 8650
        THEN away_goal END),2) AS away_goals
FROM match
GROUP BY season;
```

```
| season    | avg_homegoals    | avg_awaygoals    |
|-----------|------------------|------------------|
| 2011/2012 | 1.26             | 1.21             |
| 2012/2013 | 1.73             | 2                |
| 2013/2014 | 2.78             | 2.52             |
| 2014/2015 | 1.57             | 1.15             |
```

Percentages with CASE and AVG

```
SELECT
  season,
  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) AS pct_homewins,
  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) AS pct_awaywins
FROM match
GROUP BY season;
```

```
| season    | pct_homewins     | pct_awaywins     |
|-----------|------------------|------------------|
| 2011/2012 | 0.75             | 0.5              |
| 2012/2013 | 0.85714285714286 | 0.66666666666667 |
| 2013/2014 | 0.9375           | 0.66666666666667 |
| 2014/2015 | 1                | 0.78571428571429 |
```

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             |
```

Let's Practice!

INTERMEDIATE SQL

WHERE are the subqueries?

INTERMEDIATE SQL

Mona Khalil
Data Scientist, Greenhouse Software

datacamp

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
```

Simple Subqueries

- Is only processed once in the entire statement

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

Subqueries in the WHERE clause

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

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

```
1.56091291478423
```

```
SELECT date, hometeam_id, awayteam_id, home_goal, away_goal
FROM match
WHERE season = '2012/2013'
      AND home_goal > 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         |
```

Subquery filtering list with IN

- Which teams are part of Poland's league?

```
SELECT
  team_long_name,
  team_short_name AS abbr
FROM team
WHERE
  team_api_id IN
  (SELECT hometeam_id
   FROM match
   WHERE country_id = 15722);
```

team_long_name	abbr
Ruch Chorzów	CHO
Jagiellonia	BIA
Lech Pozna?	POZ
P. Warszawa	PWA
Cracovia	CKR
Górnik ??czna	LEC
Polonia Bytom	GOR
Zag??bie Lubin	ZAG
Pogo? Szczecin	POG
Widzew ?ód?	WID
?l?sk Wroc?aw	SLA

Practice Time!

INTERMEDIATE SQL

Subqueries in the FROM statement

INTERMEDIATE SQL

Mona Khalil
Data Scientist, Greenhouse Software

datacamp

Subqueries in FROM

- Restructure and transform your data
  - Transforming data from long to wide before selecting
  - Prefiltering data
- Calculating *aggregates of aggregates*
  - *Which 3 teams has the highest average of home goals scored?*
    1. Calculate the `AVG` for each team
    2. Get the 3 highest of the `AVG` values

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;
```

```
| team                 | home_avg         |
|----------------------|------------------|
| 1. FC Köln           | 1.13725490196078 |
| 1. FC Nürnberg       | 1.27058823529412 |
| 1. FSV Mainz 05      | 1.43697478991597 |
| AC Ajaccio           | 1.12280701754386 |
```

...to main queries!

```
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)
```

...to main queries!

```
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
```

...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
```

...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
ORDER BY home_avg DESC
LIMIT 3;
```

```
| team           | home_avg |
|----------------|----------|
| FC Barcelona   | 3.8421   |
| Real Madrid CF | 3.6842   |
| PSV            | 3.3529   |
```

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!

Let's Practice!

INTERMEDIATE SQL

Subqueries in SELECT

INTERMEDIATE SQL

Mona Khalil
Data Scientist, Greenhouse Software

datacamp

SELECTing what?

- Returns a **single value**
  - Include aggregate values to compare to individual values
- Used in mathematical calculations
  - Deviation from the average

Subqueries in SELECT

- Calculate the total matches across all seasons

```
SELECT COUNT(id) FROM match;
```

```
12837
```

Subqueries in SELECT

```
SELECT
  season,
  COUNT(id) AS matches,
  12837 as total_matches
FROM match
GROUP BY season;
```

```
| season    | matches | total_matches |
|-----------|---------|---------------|
| 2011/2012 | 3220    | 12837         |
| 2012/2013 | 3260    | 12837         |
| 2013/2014 | 3032    | 12837         |
| 2014/2015 | 3325    | 12837         |
```

Subqueries in SELECT

```
SELECT
  season,
  COUNT(id) AS matches,
  (SELECT COUNT(id) FROM match) as total_matches
FROM match
GROUP BY season;
```

```
| season    | matches | total_matches |
|-----------|---------|---------------|
| 2011/2012 | 3220    | 12837         |
| 2012/2013 | 3260    | 12837         |
| 2013/2014 | 3032    | 12837         |
| 2014/2015 | 3325    | 12837         |
```

SELECT subqueries for mathematical calculations

```
SELECT AVG(home_goal + away_goal)
FROM match
WHERE season = '2011/2012';
```

```
2.72
```

```
SELECT
  date,
  (home_goal + away_goal) AS goals,
  (home_goal + away_goal) - 2.72 AS diff
FROM match
WHERE season = '2011/2012';
```

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 mind

- 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';
```

Let's Practice!

INTERMEDIATE SQL

Subqueries everywhere! And best practices!

INTERMEDIATE SQL

Mona Khalil
Data Scientist, Greenhouse Software

datacamp

As many subqueries as you want...

- Can include multiple subqueries in `SELECT`, `FROM`, `WHERE`

```
SELECT
    country_id,
    ROUND(AVG(matches.home_goal + matches.away_goal),2) AS avg_goals,
    (SELECT ROUND(AVG(home_goal + away_goal),2)
     FROM match WHERE season = '2013/2014') AS overall_avg
FROM (SELECT
        id,
        home_goal,
        away_goal,
        season
      FROM match
      WHERE home_goal > 5) AS matches
WHERE matches.season = '2013/2014'
     AND (AVG(matches.home_goal + matches.away_goal) >
         (SELECT AVG(home_goal + away_goal)
          FROM match WHERE season = '2013/2014')
GROUP BY country_id;
```

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
```

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;
```

Holywell's SQL Style Guide

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?

Properly filter each subquery!

- Watch your filters!

```
SELECT
    country_id,
    ROUND(AVG(m.home_goal + m.away_goal),2) AS avg_goals,
    (SELECT ROUND(AVG(home_goal + away_goal),2)
     FROM match WHERE season = '2013/2014') AS overall_avg
FROM match AS m
WHERE
    m.season = '2013/2014'
    AND (AVG(m.home_goal + m.away_goal) >
        (SELECT AVG(home_goal + away_goal)
         FROM match WHERE season = '2013/2014')
GROUP BY country_id;
```

Let's Practice!

INTERMEDIATE SQL

Correlated Subqueries

INTERMEDIATE SQL

Mona Khalil
Data Scientist, Greenhouse Software

datacamp

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 simple example

- *Which match stages tend to have a higher than average number of goals scored?*

```
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
                     WHERE season = '2012/2013');
```

A simple example

- *Which match stages tend to have a higher than average number of goals scored?*

```
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 -- Subquery in FROM
WHERE s.avg_goals > (SELECT AVG(home_goal + away_goal)
                     FROM match
                     WHERE season = '2012/2013'); -- Subquery in WHERE
```

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);
```

A correlated example

stage	avg_goals
3	2.83
4	2.8
6	2.78
8	3.09
10	2.96

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?*

```
SELECT
  c.name AS country,
  AVG(m.home_goal + m.away_goal)
     AS avg_goals
FROM country AS c
LEFT JOIN match AS m
ON c.id = m.country_id
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  |
```

Correlated subqueries

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

```
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  |
```

Let's practice!

INTERMEDIATE SQL

Nested Subqueries

INTERMEDIATE SQL

Mona Khalil
Data Scientist, Greenhouse Software

datacamp

Nested subqueries??

- Subquery inside another subquery
- Perform multiple layers of transformation

A subquery...

- *How much did each country's average differ from the overall average?*

```
SELECT
    c.name AS country,
    AVG(m.home_goal + m.away_goal) AS avg_goals,
    AVG(m.home_goal + m.away_goal) -
        (SELECT AVG(home_goal + away_goal)
         FROM match) AS avg_diff
FROM country AS c
LEFT JOIN match AS m
ON c.id = m.country_id
GROUP BY country;
```

A subquery...

country	avg_goals	avg_diff
Belgium	2.8015	0.096
England	2.7105	0.005
France	2.4431	-0.2624
Germany	2.9016	0.196
Italy	2.6168	-0.0887
Netherlands	3.0809	0.3754
Poland	2.425	-0.2805
Portugal	2.5346	-0.1709
Scotland	2.6338	-0.0718
Spain	2.7671	0.0616
Switzerland	2.9297	0.2241

...inside a subquery!

- *How does each month's total goals differ from the **average monthly total** of goals scored?*

```
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 avg_diff
FROM match AS m
GROUP BY month;
```

Inner subquery

```
SELECT
  EXTRACT(MONTH from date) AS month,
  SUM(home_goal + away_goal) AS goals
FROM match
GROUP BY month;
```

```
| month | goals |
|-------|-------|
| 01    | 2988  |
| 02    | 3768  |
| 03    | 3936  |
| 04    | 4055  |
| 05    | 2719  |
| 06    | 84    |
| 07    | 366   |
```

Outer subquery

```
SELECT AVG(goals)
FROM (SELECT
        EXTRACT(MONTH from date) AS month,
        AVG(home_goal + away_goal) AS goals
FROM match
GROUP BY month) AS s;
```

```
2944.75
```

Final query

```
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  |
```

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*

Correlated nested subqueries

- *What is the each country's average goals scored in the 2011/2012 season?*

```
SELECT
  c.name AS country,
  (SELECT AVG(home_goal + away_goal)
   FROM match AS m
   WHERE m.country_id = c.id
         AND id IN (
             SELECT id
             FROM match
             WHERE season = '2011/2012')) AS avg_goals
FROM country AS c
GROUP BY country;
```

Correlated nested subqueries

- *What is the each country's average goals scored in the 2011/2012 season?*

```
SELECT
  c.name AS country,
  (SELECT AVG(home_goal + away_goal)
   FROM match AS m
   WHERE m.country_id = c.id
         AND id IN (
             SELECT id -- Begin inner subquery
             FROM match
             WHERE season = '2011/2012')) AS avg_goals
FROM country AS c
GROUP BY country;
```

Correlated nested subquery

- *What is the each country's average goals scored in the 2011/2012 season?*

```
SELECT
  c.name AS country,
  (SELECT AVG(home_goal + away_goal)
   FROM match AS m
   WHERE m.country_id = c.id -- Correlates with main query
         AND id IN (
             SELECT id -- Begin inner subquery
             FROM match
             WHERE season = '2011/2012')) AS avg_goals
FROM country AS c
GROUP BY country;
```

Correlated nested subqueries

country	avg_goals
Belgium	2.87916666666667
England	2.80526315789474
France	2.51578947368421
Germany	2.85947712418301
Italy	2.58379888268156
Netherlands	3.25816993464052
Poland	2.19583333333333
Portugal	2.64166666666667
Scotland	2.6359649122807
Spain	2.76315789473684
Switzerland	2.62345679012346

Let's practice!

INTERMEDIATE SQL

Common Table Expressions

INTERMEDIATE SQL

Mona Khalil
Data Scientist, Greenhouse Software

datacamp

When adding subqueries...

- Query complexity increases quickly!
  - Information can be difficult to keep track of

Solution: **Common Table Expressions!**

Common Table Expressions

Common Table Expressions (CTEs)

- Table *declared* before the main query
- *Named* and *referenced* later in `FROM` statement

Setting up CTEs

```
WITH cte AS (
    SELECT col1, col2
    FROM table)
SELECT
    AVG(col1) AS avg_col
FROM cte;
```

Take a subquery in FROM

```
SELECT
  c.name AS country,
  COUNT(s.id) AS matches
FROM country AS c
INNER JOIN (
  SELECT country_id, id
  FROM match
  WHERE (home_goal + away_goal) >= 10) AS s
ON c.id = s.country_id
GROUP BY country;
```

```
| country     | matches |
|-------------|---------|
| England     | 3       |
| Germany     | 1       |
| Netherlands | 1       |
| Spain       | 4       |
```

Place it at the beginning

```
(
  SELECT country_id, id
  FROM match
  WHERE (home_goal + away_goal) >= 10
)
```

Place it at the beginning

```
WITH s AS (
  SELECT country_id, id
  FROM match
  WHERE (home_goal + away_goal) >= 10
)
```

Show me the CTE

```
WITH s AS (
  SELECT country_id, id
  FROM match
  WHERE (home_goal + away_goal) >= 10
)
SELECT
  c.name AS country,
  COUNT(s.id) AS matches
FROM country AS c
INNER JOIN s
ON c.id = s.country_id
GROUP BY country;
```

```
| country     | matches |
|-------------|---------|
| England     | 3       |
| Germany     | 1       |
| Netherlands | 1       |
| Spain       | 4       |
```

Show me all the CTEs

```
WITH s1 AS (
  SELECT country_id, id
  FROM match
  WHERE (home_goal + away_goal) >= 10),
s2 AS (                              -- New subquery
  SELECT country_id, id
  FROM match
  WHERE (home_goal + away_goal) <= 1
)
SELECT
  c.name AS country,
  COUNT(s1.id) AS high_scores,
  COUNT(s2.id) AS low_scores         -- New column
FROM country AS c
INNER JOIN s1
ON c.id = s1.country_id
INNER JOIN s2                        -- New join
ON c.id = s2.country_id
GROUP BY country;
```

Why use CTEs?

- Executed once
  - CTE is then stored in memory
  - Improves query performance
- Improving organization of queries
- Referencing other CTEs
- Referencing itself (`SELF JOIN`)

Let's Practice!

INTERMEDIATE SQL

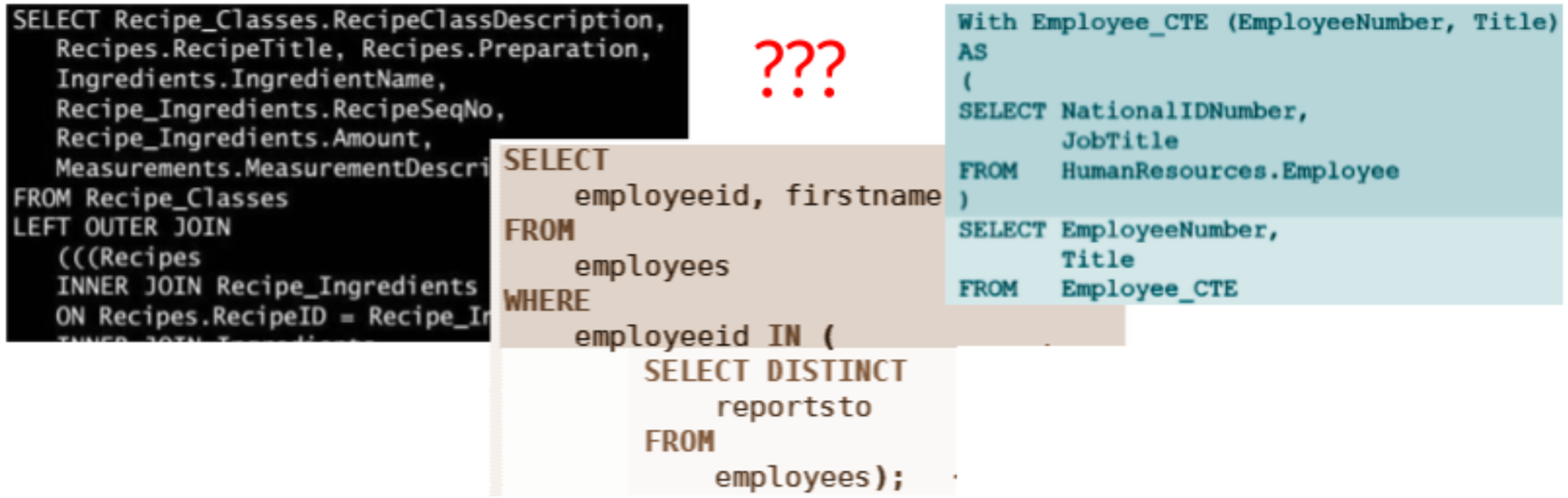
Deciding on techniques to use

INTERMEDIATE SQL

Mona Khalil
Data Scientist, Greenhouse Software

Different names for the same thing?

- Considerable overlap...



- ...but **not** identical!

Differentiating Techniques

Joins

- Combine 2+ tables
  - Simple operations/aggregations

Correlated Subqueries

- Match subqueries & tables
  - Avoid limits of joins
  - **High processing time**

Multiple/Nested Subqueries

- Multi-step transformations
  - Improve accuracy and reproducibility

Common Table Expressions

- Organize subqueries sequentially
- Can reference other CTEs

So which do I use?

- Depends on your database/question
- The technique that best allows you to:
  - Use and reuse your queries
  - Generate clear and accurate results

Different use cases

Joins

- 2+ tables (*What is the total sales per employee?*)

Correlated Subqueries

- *Who does each employee report to in a company?*

Multiple/Nested Subqueries

- *What is the average deal size closed by each sales representative in the quarter?*

Common Table Expressions

- *How did the marketing, sales, growth, & engineering teams perform on key metrics?*

Let's Practice!

INTERMEDIATE SQL

Window Functions

INTERMEDIATE SQL

Mona Khalil
Data Scientist, Greenhouse Software

datacamp

Working with aggregate values

- Requires you to use `GROUP BY` with **all** non-aggregate columns

```
SELECT
  country_id,
  season,
  date,
  AVG(home_goal) AS avg_home
FROM match
GROUP BY country_id;
```

```
ERROR: column "match.season" must appear in the GROUP BY
clause or be used in an aggregate function
```

Introducing window functions!

- Perform calculations on an already generated result set (a *window*)
- Aggregate calculations
  - Similar to subqueries in `SELECT`
  - Running totals, rankings, moving averages

What's a window function?

- *How many goals were scored in each match in 2011/2012, and how did that compare to the average?*

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

```
| date       | goals | overall_avg       |
|------------|-------|-------------------|
| 2011-07-29 | 3     | 2.71646           |
| 2011-07-30 | 2     | 2.71646           |
| 2011-07-30 | 4     | 2.71646           |
| 2011-07-30 | 1     | 2.71646           |
```

What's a window function?

- *How many goals were scored in each match in 2011/2012, and how did that compare to the average?*

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

```
| date       | goals | overall_avg       |
|------------|-------|-------------------|
| 2011-07-29 | 3     | 2.71646           |
| 2011-07-30 | 2     | 2.71646           |
| 2011-07-30 | 4     | 2.71646           |
| 2011-07-30 | 1     | 2.71646           |
```

Generate a RANK

- *What is the rank of matches based on number of goals scored?*

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

```
| date       | goals |
|------------|-------|
| 2011-07-29 | 3     |
| 2011-07-30 | 2     |
| 2011-07-30 | 4     |
| 2011-07-30 | 1     |
```

Generate a RANK

- *What is the rank of matches based on number of goals scored?*

```
SELECT
    date,
    (home_goal + away_goal) AS goals,
    RANK() OVER(ORDER BY home_goal + away_goal) AS goals_rank
FROM match
WHERE season = '2011/2012';
```

```
| date       | goals | goals_rank |
|------------|-------|------------|
| 2012-04-28 | 0     | 1          |
| 2011-12-26 | 0     | 1          |
| 2011-09-10 | 0     | 1          |
| 2011-08-27 | 0     | 1          |
```

Generate a RANK

- *What is the rank of matches based on number of goals scored?*

```
SELECT
    date,
    (home_goal + away_goal) AS goals,
    RANK() OVER(ORDER BY home_goal + away_goal DESC) AS goals_rank
FROM match
WHERE season = '2011/2012';
```

```
| date       | goals | goals_rank |
|------------|-------|------------|
| 2011-11-06 | 10    | 1          |
| 2011-08-28 | 10    | 1          |
| 2012-05-12 | 9     | 3          |
| 2012-02-12 | 9     | 3          |
```

Key Differences

- Processed *after* every part of query except `ORDER BY`
  - Uses information in result set rather than database
- Available in PostgreSQL, Oracle, MySQL, SQL Server...
  - ...but NOT SQLite

Let's Practice!

INTERMEDIATE SQL

Window Partitions

INTERMEDIATE SQL

Mona Khalil
Data Scientist, Greenhouse Software

datacamp

OVER and PARTITION BY

- Calculate separate values for different categories
- Calculate *different* calculations in the same column

```
AVG(home_goal) OVER(PARTITION BY season)
```

Partition your data

- *How many goals were scored in each match, and how did that compare to the overall average?*

```
SELECT
    date,
    (home_goal + away_goal) AS goals,
    AVG(home_goal + away_goal) OVER() AS overall_avg
FROM match;
```

```
| date       | goals | overall_avg |
|------------|-------|-------------|
| 2011-12-17 | 3     | 2.73210     |
| 2012-05-01 | 2     | 2.73210     |
| 2012-11-27 | 4     | 2.73210     |
| 2013-04-20 | 1     | 2.73210     |
| 2013-11-09 | 5     | 2.73210     |
```

Partition your data

- *How many goals were scored in each match, and how did that compare to the season's average?*

```
SELECT
    date,
    (home_goal + away_goal) AS goals,
    AVG(home_goal + away_goal) OVER(PARTITION BY season) AS season_avg
FROM match;
```

```
| date       | goals | season_avg  |
|------------|-------|-------------|
| 2011-12-17 | 3     | 2.71646     |
| 2012-05-01 | 2     | 2.71646     |
| 2012-11-27 | 4     | 2.77270     |
| 2013-04-20 | 1     | 2.77270     |
| 2013-11-09 | 5     | 2.76682     |
```

PARTITION by Multiple Columns

```
SELECT
  c.name,
  m.season,
  (home_goal + away_goal) AS goals,
  AVG(home_goal + away_goal)
      OVER(PARTITION BY m.season, c.name) AS season_ctry_avg
FROM country AS c
LEFT JOIN match AS m
ON c.id = m.country_id
```

```
| name        | season    | goals     | season_ctry_avg |
|-------------|-----------|-----------|-----------------|
| Belgium     | 2011/2012 | 1         | 2.88            |
| Netherlands | 2014/2015 | 1         | 3.08            |
| Belgium     | 2011/2012 | 1         | 2.88            |
| Spain       | 2014/2015 | 2         | 2.66            |
```

PARTITION BY considerations

- Can partition data by 1 or more columns
- Can partition aggregate calculations, ranks, etc

Let's Practice!

INTERMEDIATE SQL

Sliding Windows

INTERMEDIATE SQL

Mona Khalil
Data Scientist, Greenhouse Software

datacamp

Sliding Windows

- Perform calculations relative to the current row
- Can be used to calculate running totals, sums, averages, etc
- Can be partitioned by one or more columns

Sliding Window Keywords

```
ROWS BETWEEN <start> AND <finish>
```

```
PRECEDING
FOLLOWING
UNBOUNDED PRECEDING
UNBOUNDED FOLLOWING
CURRENT ROW
```

Sliding Window Example

```
-- Manchester City Home Games
SELECT
  date,
  home_goal,
  away_goal,
  SUM(home_goal)
     OVER(ORDER BY date ROWS BETWEEN
          UNBOUNDED PRECEDING AND CURRENT ROW) AS running_total
FROM match
WHERE hometeam_id = 8456 AND season = '2011/2012';
```

```
| date       | home_goal | away_goal | running_total |
|------------|-----------|-----------|---------------|
| 2011-08-15 | 4         | 0         | 4             |
| 2011-09-10 | 3         | 0         | 7             |
| 2011-09-24 | 2         | 0         | 9             |
| 2011-10-15 | 4         | 1         | 13            |
```

Sliding Window Frame

```
-- Manchester City Home Games
SELECT date,
       home_goal,
       away_goal,
       SUM(home_goal)
          OVER(ORDER BY date
          ROWS BETWEEN 1 PRECEDING
          AND CURRENT ROW) AS last2
FROM match
WHERE hometeam_id = 8456
      AND season = '2011/2012';
```

date	home_goal	away_goal	last2
2011-08-15	4	0	4
2011-09-10	3	0	7
2011-09-24	2	0	5
2011-10-15	4	1	6

Let's Practice!

INTERMEDIATE SQL

Bringing it all Together

INTERMEDIATE SQL

Mona Khalil
Data Scientist, Greenhouse Software

datacamp

What you've learned so far

- `CASE` statements
- Simple subqueries
- Nested and correlated subqueries
- Common table expressions
- Window functions

Let's do a case study!

Who defeated Manchester United in the 2013/2014 season?

Steps to construct the query

- Get team names with CTEs
- Get match outcome with `CASE` statements
- Determine how badly they lost with a window function

Getting the database for yourself

Full European Soccer Database

Let's Practice!

INTERMEDIATE SQL