

Soccer Practice 1

Intermediate SQL Practice CSC 330 Database Th&Appl Spring 2022

README

- This interactive notebook assumes that you completed and understood the first half of the chapter "We'll take the CASE" of the DataCamp course "Intermediate SQL".
- If you've forgotten stuff (which is normal), check back with the slides and/or the transcript of the [video](#). You can open the transcript below the video and download the slides with a button in the upper right corner.

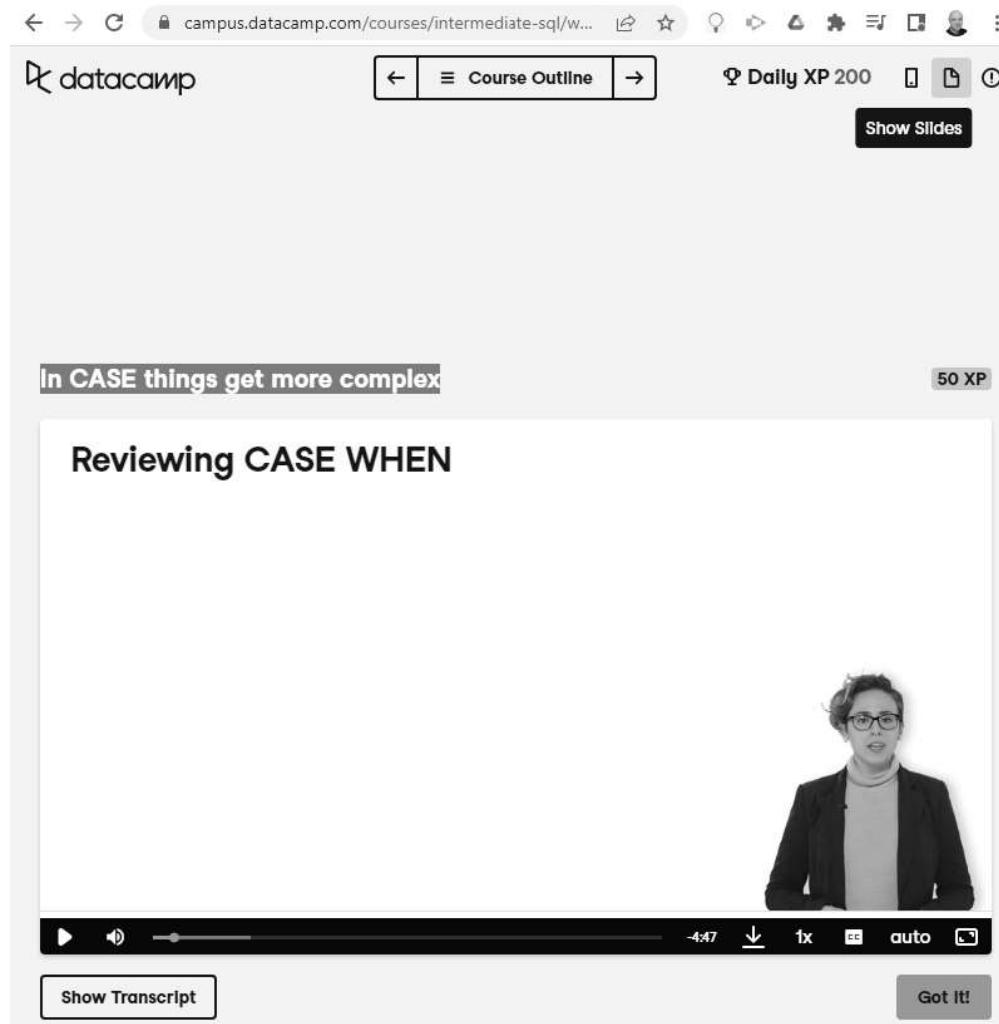


Figure 1: downloading video slides at DataCamp

- It is most convenient to work with the database in an Emacs Org-file. You can simple create a new code block with `< TAB`. To execute it, you must a) load the properties from the first row of this file (enter `C-c C-c` on the line - you should see the message `Local setup has been refreshed` in the minibuffer).
- The **header arguments** for this Org-file includes

- read from the database (which must be in the same directory as the file)
- set .header ON, .mode column
- tangle SQLite code to a file called soccer.sqlite (c-c c-v t tangles all blocks, c-u c-c c-v t tangles the block under the cursor only)
- The European Soccer database 2008-2016 (aka 'soccer db', 300 MB) is available as a SQLite database file on [GDrive](#).

Entity relationship diagrams

- Figure 2 shows the entity relationship diagram for the database that you can find on the net.

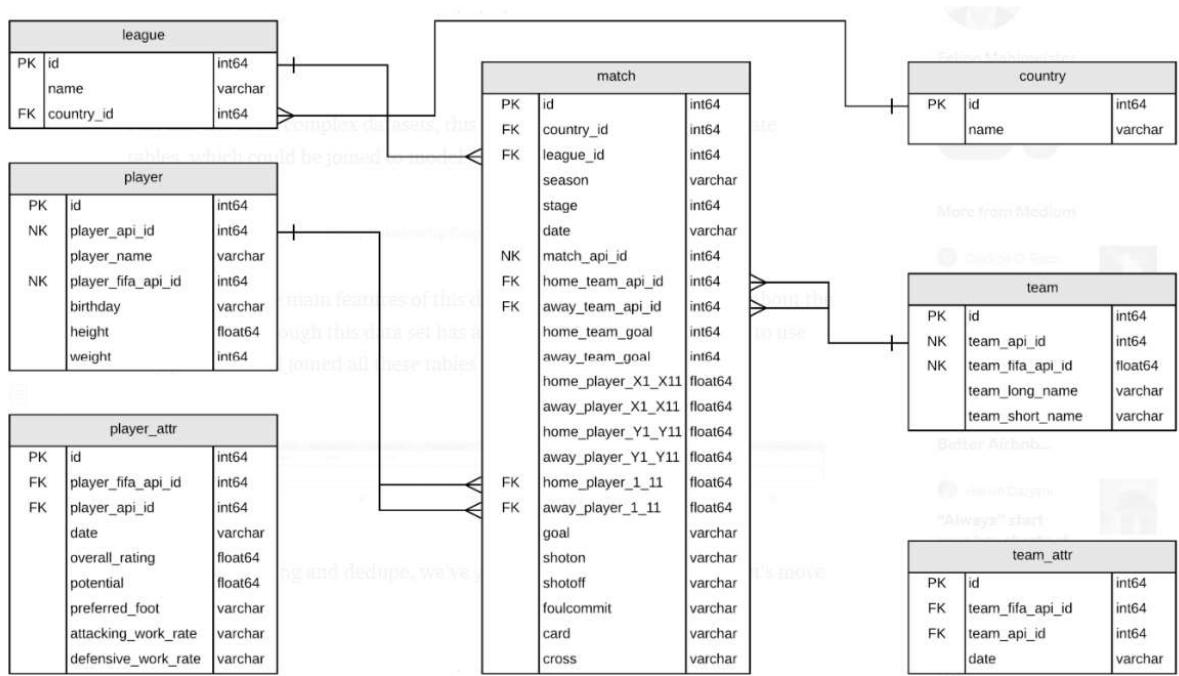


Figure 2: Soccer db ER diagram (Source: Mahlmeister 2019)

- A more accurate [ERD](#) for the soccer db can be autogenerated on the command line using the SchemaCrawler tool (2022). This is a command line (shell) tool, which in turn is based on [Graphviz](#), open source graph visualization software¹.

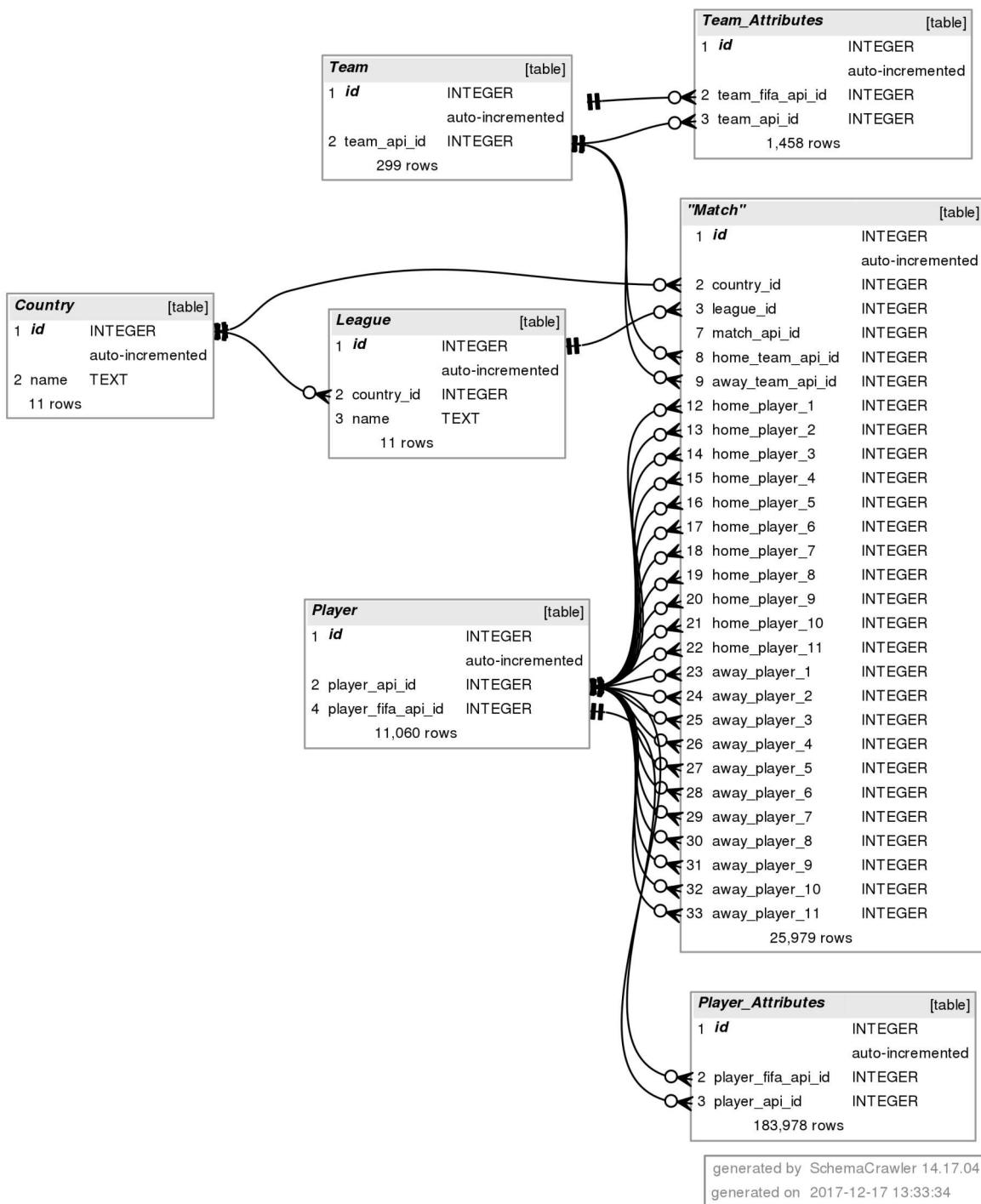
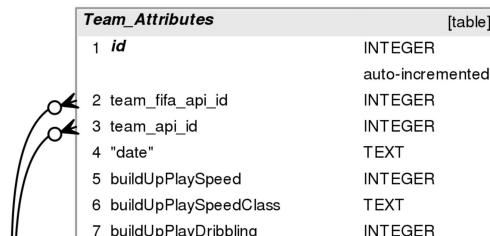
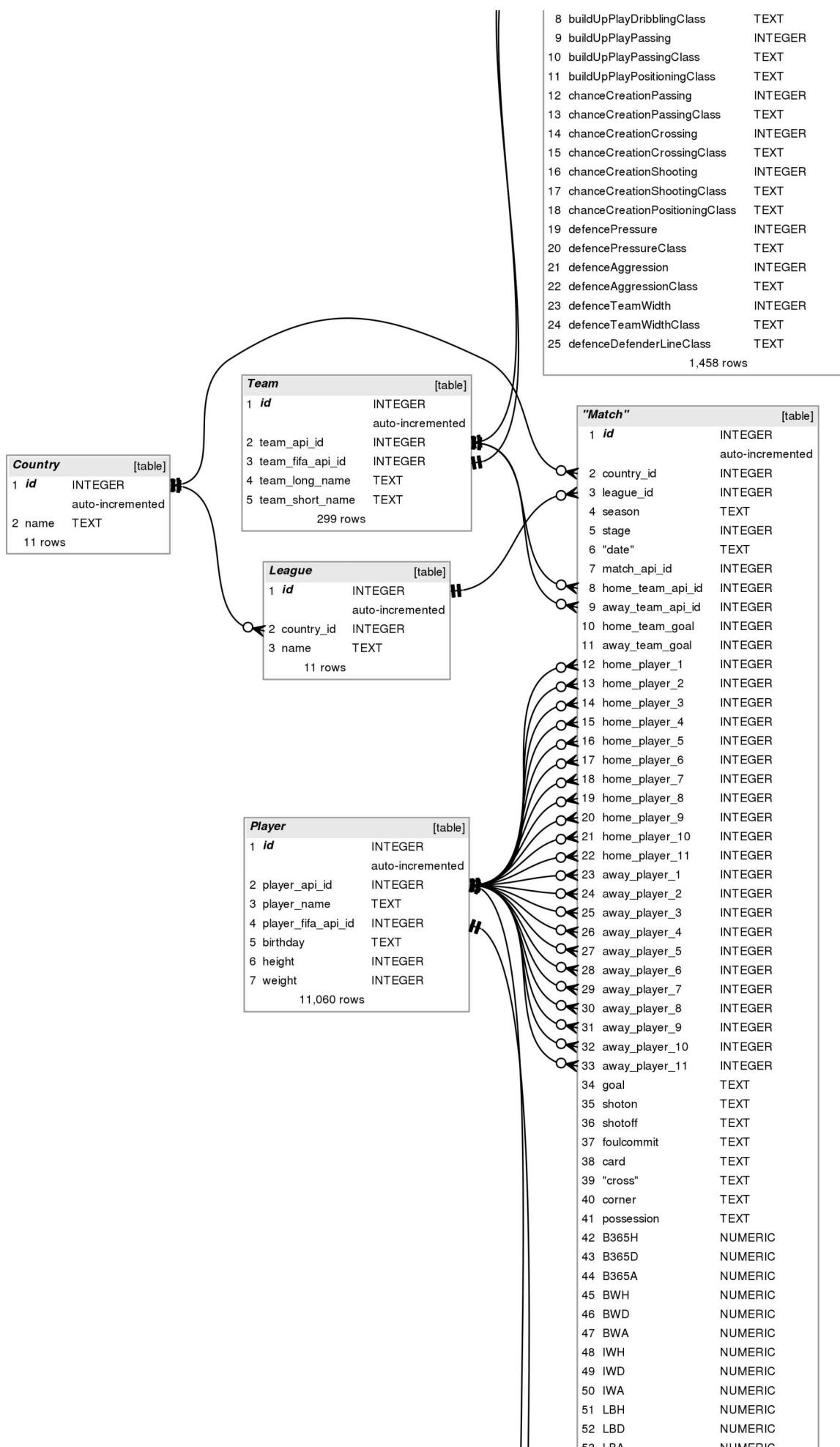


Figure 3: Soccer db ER diagram - high level (Source: Mahlmeister 2019)



Soccer Practice 1



33	LDR	NUMERIC
54	PSH	NUMERIC
55	PSD	NUMERIC
56	PSA	NUMERIC
57	WHH	NUMERIC
58	WHD	NUMERIC
59	WHA	NUMERIC
60	SJH	NUMERIC
61	SJD	NUMERIC
62	SJA	NUMERIC
63	VCH	NUMERIC
64	VCD	NUMERIC
65	VCA	NUMERIC
66	GBH	NUMERIC
67	GBD	NUMERIC
68	GBA	NUMERIC
69	BSH	NUMERIC
70	BSD	NUMERIC
71	BSA	NUMERIC

25,979 rows

Player_Attributes [table]	
1	<i>id</i> INTEGER auto-incremented
2	player_fifa_api_id INTEGER
3	player_api_id INTEGER
4	"date" TEXT
5	overall_rating INTEGER
6	potential INTEGER
7	preferred_foot TEXT
8	attacking_work_rate TEXT
9	defensive_work_rate TEXT
10	crossing INTEGER
11	finishing INTEGER
12	heading_accuracy INTEGER
13	short_passing INTEGER
14	volleys INTEGER
15	dribbling INTEGER
16	curve INTEGER
17	free_kick_accuracy INTEGER
18	long_passing INTEGER
19	ball_control INTEGER
20	acceleration INTEGER
21	sprint_speed INTEGER
22	agility INTEGER
23	reactions INTEGER
24	balance INTEGER
25	shot_power INTEGER
26	jumping INTEGER
27	stamina INTEGER
28	strength INTEGER
29	long_shots INTEGER
30	aggression INTEGER
31	interceptions INTEGER
32	positioning INTEGER
33	vision INTEGER
34	penalties INTEGER
35	marking INTEGER
36	standing_tackle INTEGER
37	sliding_tackle INTEGER
38	gk_diving INTEGER
39	gk_handling INTEGER
40	gk_kicking INTEGER
41	gk_positioning INTEGER
42	gk_reflexes INTEGER

183,978 rows

generated by SchemaCrawler 14.17.04
generated on 2017-12-17 13:43:22

Figure 4: Soccer db ER diagram - all columns (Source: Mahlmeister 2019)

Open soccer db

- Complete the following exercises as best you can. You find the solutions as a PDF on GitHub.
- [X]

When executed (`C-c C-c` or `M-x org-babel-execute-buffer`), the code block below shows the list of tables in the soccer SQLite database.

```
.tables
```

Country League	Match Player	Player_Attributes	Team_Attributes
		Team	

- []

Which SQLite command show you all the attributes of each table? Print out the attributes for `Country` and `Team`.

```
.schema Country
.schema Team
```

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

DataCamp: We'll take the CASE

- [] In the introductory video, a query is shown. Recreated it from these requirements:
 - Count the number of matches (as `total_matches`) played in each league listed in the `league` table.
 - When you look at the ERD, you see that the `country_id` is a foreign key in the `league` table and in the `match` table. This means that you can use it in a `JOIN` to query both tables.
 - In this query, you don't actually need a `LEFT OUTER JOIN`. Check this by altering the command and running an inner `JOIN` instead.
 - Reminder: the `LEFT [OUTER] JOIN` takes two relations or tables, A and B, and returns the inner join of A and B along with the unmatched rows of A.
 - At the end, the `GROUP BY` command runs the `COUNT` operation on each league.

```
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;
```

league	total_matches
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

- []

Next, compare the number of home team wins, away team wins, and ties in the 2013/2014 season. Unfortunately, some of the attributes have different names in our version of the database.

It shouldn't be too hard to find them though - use the detailed ERD or the schema to identify them and alter the code to get the right result:

```
: 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
```

Fix the code:

```
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

- []

Next, filter those events out for whom the home team wins were greater than the away team wins.

Change the code:

```

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

CASE statements

- The CASE syntax is reminiscent of the `base::cbind` and the `dplyr::mutate` commands in R, and of the IF structure in C-type languages.

```

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

```

- []

Using CASE and only the `match` table, create a new variable 'outcome' that identifies matches as home team wins, away team wins, or ties (as the default outcome) in the '2013/2014' season. Print only the first 5 lines.

Here is some [documentation on CASE for SQLite](#).

```

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 5;

```

id	home_team_goal	away_team_goal	outcome
1237	2	0	Home Team Win
1238	0	1	Tie
1239	1	0	Home Team Win
1240	0	0	Tie
1241	2	1	Home Team Win

Practice simple CASE statements

- [] Explore the matches for Germany - this is a table that DataCamp has already prepared for you.
 - write a query to find the country.id for "Germany"
 - write a query to find the total number of matches for Germany
 - write a query to find the matches for Germany
- []

Country ID for Germany: print it as "Germany ID".

```
SELECT id AS "Germany ID"
FROM country AS c
WHERE c.name="Germany";
```

```
Germany ID
-----
7809
```

- []

Total number of matches for Germany: print it as "Matches"

```
SELECT COUNT(*) AS "Matches"
FROM match AS m
WHERE m.country_id = 7809;
```

```
Matches
-----
2448
```

- []

Matches for Germany: print the date, the season, and the home team id.

```
SELECT
date,
season,
home_team_api_id AS home_id
FROM match AS m
WHERE m.country_id = 7809
LIMIT 5;
```

date	season	home_id
2008-08-15 00:00:00	2008/2009	9823
2008-08-16 00:00:00	2008/2009	8178
2008-08-16 00:00:00	2008/2009	10189
2008-08-16 00:00:00	2008/2009	8721
2008-08-17 00:00:00	2008/2009	9810

- []

Use CASE to add a column `Ergebnis` (German for 'outcome') that shows if the game was a win, a loss or a tie ('`Unentschieden`') for Germany. Print only date and `Ergebnis`.

```
SELECT
date,
CASE WHEN home_team_goal > away_team_goal THEN 'Heimatgewinn! :-)'
WHEN home_team_goal < away_team_goal THEN 'Heimatverlust :-('
ELSE 'Unentschieden' END AS Ergebnis
FROM match AS m
WHERE m.country_id = 7809
LIMIT 5;
```

date	Ergebnis
2008-08-15 00:00:00	Unentschieden
2008-08-16 00:00:00	Heimatverlust :-(
2008-08-16 00:00:00	Heimatgewinn! :-)
2008-08-16 00:00:00	Heimatgewinn! :-)
2008-08-17 00:00:00	Heimatverlust :-(

NULL

- []

What was `NULL` again, and why do you get `NULL` in a table with the `CASE` statement?

ANSWER: when you enter `NULL` as the default value after `ELSE`.

Figure 5 has no default clause and rows that do not meet the filter condition have the outcome `NULL` (undefined).

```
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!

Figure 5: NULL and CASE (Source: DataCamp)

- []

What was the strategy to remove rows with NULL from your output?

ANSWER: you treat the *entire* CASE~ statement as an argument to the WHERE filter and end with IS NOT NULL (instead of the new column name).

In figure 6 the entire CASE statement is the filter condition and NULL is explicitly filtered out, just as if the filter looked like this:

```
WHERE attribute IS NOT NULL
```

```
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!

Figure 6: NULL and CASE (Source: DataCamp)

Create a database with NULL values and query it with CASE

- To make this more interesting and revisit possibly long-forgotten skills, let's create a database from scratch, generate some NULL values and query them using our new CASE skills.
- [] Create a database null.db using the code block below (you don't have to do anything - running the code block will create an empty database with that name)
 - Create a table tnull with two attributes:
 - id as an integer primary key column
 - name as a text column
 - check that the table is there

```
CREATE TABLE tnull (id INT PRIMARY KEY, name TEXT);
.tables
```

```
tnull
```

- [] Now `INSERT` a row of `VALUES` into the table `tnull`: insert 1 and "Jim Jones" (or any other name you like).
 - In the same block, query the table.

```
INSERT INTO tnull VALUES (1, "Jim Jones");
SELECT * FROM tnull;
```

```
id  name
--  -----
1   Jim Jones
```

- []

Once you've done this, try to execute it again. You should see this error message:

```
Error: near line 1: UNIQUE constraint failed: tnull.id
```

Why do you get this error message? Write the answer into the block below:

ANSWER: the `PRIMARY KEY tnull.id` is a unique constraint - there can be only one `id` with the value 1.

- []

Insert another three rows into `tnull`.

```
INSERT INTO tnull VALUES (2, "Jane Doe");
INSERT INTO tnull VALUES (3, "John Smith");
INSERT INTO tnull VALUES (4, "Paul Potts");
SELECT * FROM tnull;
```

```
id  name
--  -----
1   Jim Jones
2   Jane Doe
3   John Smith
4   Paul Potts
```

You should see something like this (with your choice of `name` values):

```
: id  name
: --  -----
: 1   Jim Jones
```

```
: 2  Jane Doe
: 3  John Smith
: 4  Paul Potts
```

- []

Insert NULL values by adding another column using the `ALTER TABLE [table] ADD COLUMN [column definition]` command.

- add another `TEXT` column, call it `city`.
- insert another row of values
- then change the `NULL` display value with `.nullvalue` to "[NULL]"
- display your table

```
ALTER TABLE tnull ADD COLUMN city TEXT;
INSERT INTO tnull VALUES (5, "Harry Houdini", "Batesville");
.nullvalue "[NULL]"
SELECT * FROM tnull;
```

id	name	city
1	Jim Jones	[NULL]
2	Jane Doe	[NULL]
3	John Smith	[NULL]
4	Paul Potts	[NULL]
5	Harry Houdini	Batesville

You should now see something like this:

```
: id  name          city
: --  -----
: 1   Jim Jones    [NULL]
: 2   Jane Doe     [NULL]
: 3   John Smith   [NULL]
: 4   Paul Potts   [NULL]
: 5   Harry Houdini Batesville
```

- []

Now write a `CASE` command that shows a new column 'No NULL' whose values are `0` if the corresponding `city` value is `[NULL]` and `1` otherwise.

```
: id  name          No NULL
: --  -----
: 1   Jim Jones    0
: 2   Jane Doe     0
: 3   John Smith   0
: 4   Paul Potts   0
: 5   Harry Houdini 1
```

Code:

```
SELECT name,  
CASE WHEN city IS NOT NULL THEN 1  
END IS NOT NULL AS 'No NULL'  
FROM tnull;
```

name	No NULL
Jim Jones	0
Jane Doe	0
John Smith	0
Paul Potts	0
Harry Houdini	1

References

- Mathien H (2017). European Soccer Database [dataset]. [URL: kaggle.com](#).
- Mahlmeister F (Dec 4, 2019). European Soccer Data Analysis [blog]. [URL: medium.com](#)
- SchemaCrawler (2022). Free database schema discovery and comprehension tool [website]. [URL: schemacrawler.com](#)

Footnotes:

¹ This is an example of software to look at in the Data Visualization course, because visualization of data structures stands in the foreground.

Author: Soccer Practice

Created: 2022-04-18 Mon 12:36