ASSIGNMENT 4

1. Difference Between Weak and Strong Entity Sets

An entity set is a collection of similar entities that share the same attributes. Entity sets can be classified as either strong or weak, depending on whether they have enough attributes to uniquely identify their entities.

Strong Entity Set

A strong entity set is an entity set that has a primary key that uniquely identifies each entity in the set. It does not depend on any other entity set for its identification. Strong entities are independent and can exist on their own.

Example: An Employee entity set with attributes `Employee_ID`, `Name`, and `Department`. Each employee can be uniquely identified by `Employee_ID`, making this a strong entity set.

Weak Entity Set

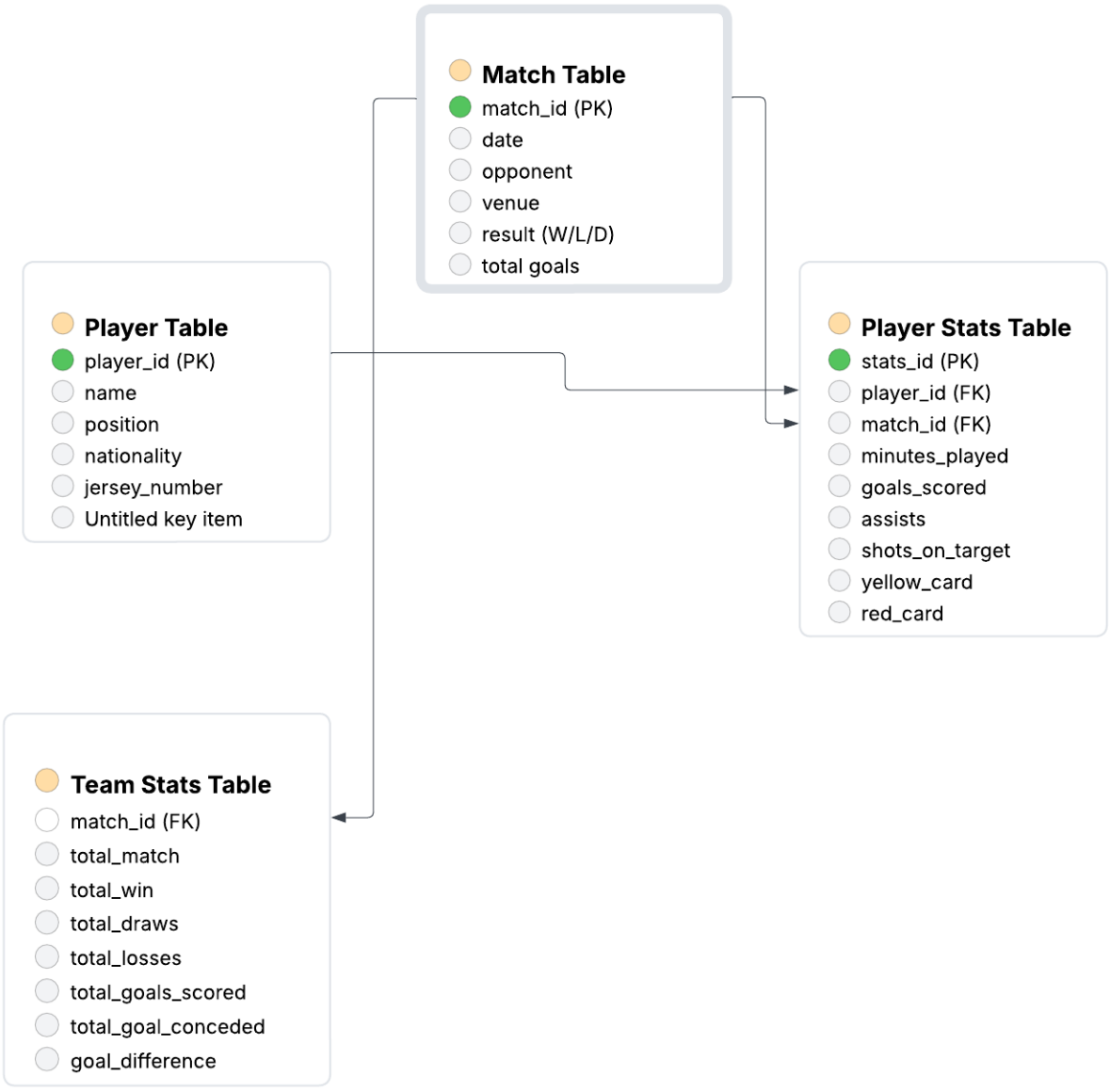
A weak entity set does not have sufficient attributes to form a primary key on its own. Instead, it depends on a strong entity set for its identification. A weak entity set is always associated with a strong entity set through an identifying relationship. The primary key of a weak entity set is formed by combining:

1. The primary key of the strong entity set (to which it is related).
2. A discriminator attribute (also called a partial key) that uniquely identifies the weak entities within the strong entity set.

Example: An entity set Dependent that stores information about the dependents of employees. A Dependent entity might have attributes `Dependent_Name` and `Relationship` but lacks a unique identifier. Instead, it is identified by the Employee_ID (from the Employee entity) and `Dependent_Name` as a discriminator.

- **Strong entity:** `Employee(Employee_ID, Name, Department)`
- **Weak entity:** `Dependent(Employee_ID, Dependent_Name, Relationship)`

QUESTION 2: MANCHESTER UNITED



QUESTION 3:

Appending `NATURAL JOIN section` would not change the result because the `section` table only contains extra details about courses, like classroom and time slot. The query is already grouping by `course_id, semester, year, sec_id`, which uniquely defines each section.

Since the query is calculating the average total credits of students (`avg(tot_cred)`) and counting the number of students in each section (`having count(ID) >= 2`), all the needed data comes from `takes` and `student`. The `section` table does not add any new student-related information, so joining it does not affect the final result.

B.

Enter SQL commands here

```
1 -- enter yoselect course_id, semester, year, sec_id, avg (tot_cred)
2 select course_id, semester, year, sec_id, avg (tot_cred)
3 from takes natural join student
4 where year = 2017
5 group by course_id, semester, year, sec_id
6 having count (ID) >= 2;
```

Execute | Save the db | Load an SQLite database file: Choose File No file chosen

course_id	semester	year	sec_id	avg (tot_cred)
CS-101	Fall	2017	1	65
CS-190	Spring	2017	2	43
CS-347	Fall	2017	1	67