

**cSCU9B3 Relational database report**

**Student ID: 2520796**

# **ER Diagram**

A database is a collection of information that is organized so data can be easily stored, managed, updated, and retrieved. However, there are lot of moving information in a database and it can be difficult to grasp how the elements in the database interact with each other. Because of this problem, a visual way to understand how all the separate elements are related to each other and how they are working together is required. So, I created an Entity Relationship Diagram using the diagraming tool Lucidchart. An entity relationship diagram (ERD) is a graphical illustration of entities and their relationships to each other and it is responsible for the organization of data within databases or information systems.

**Components of Entity Relationship Diagram:**

* **Entities:** are objects such as person, thing, or place to be tracked in the database. On this project, our entities are `PLAYERS`, `STATUS`, `PLAYERSKILLS`, `TEAMS`, `TOWNVENUE`, `SCHEDULEDGAMES`, `SKILLS` and `PLAYERSCORES` (see Figure 1).
* **Attributes:** are various properties or traits. On this ER diagram under the `PLAYERS` entity we have a `PlayerID`, a `Surname`, a `Forename`, a `Team`, and a `Statues`. It is important to address that the entities in a database will be the rows and that the attributes will be depicted at the columns.
* **Relationships:** describe how these entities will interact with each other, if at all. And we do that by drawing a line in between them.
* **Cardinality:** defines the relationship in a numerical context, particularly within minimums and maximums (see Figure 2). For example, the relationship between `Players` and `Playerskills` was Many to Many which that leads to an error. I asked my self “Does every player has every skill?” And the answer is no. Therefore, I needed to add another table `Skills` at the end, with a relationship One to Many (`Players` to `Playerskills`), and Many to One (`Playerskills to Skills) to solve this error. Another example, I asked myself again “What is the minimum and maximum number of Players that can score at the same time?”, and” What is the minimum and maximum points can be achieved by a Player?” The answer is One because there is not a possible way for 2 players scoring at the same time, and Zero to Many because a player can achieve 0 or many points, so it is there the optionality (dashed line) that a player may or may not score at all.

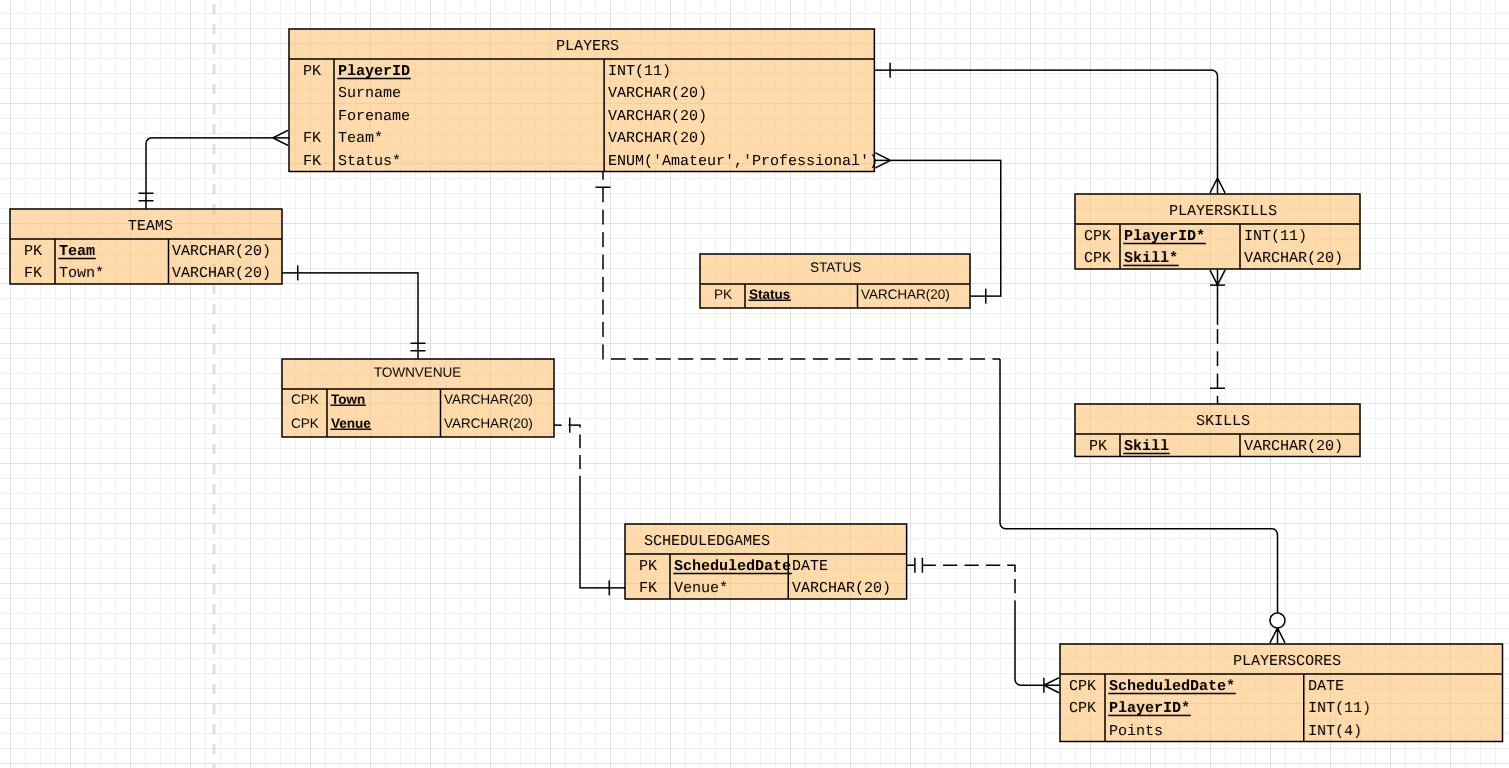


Figure - ER Diagram

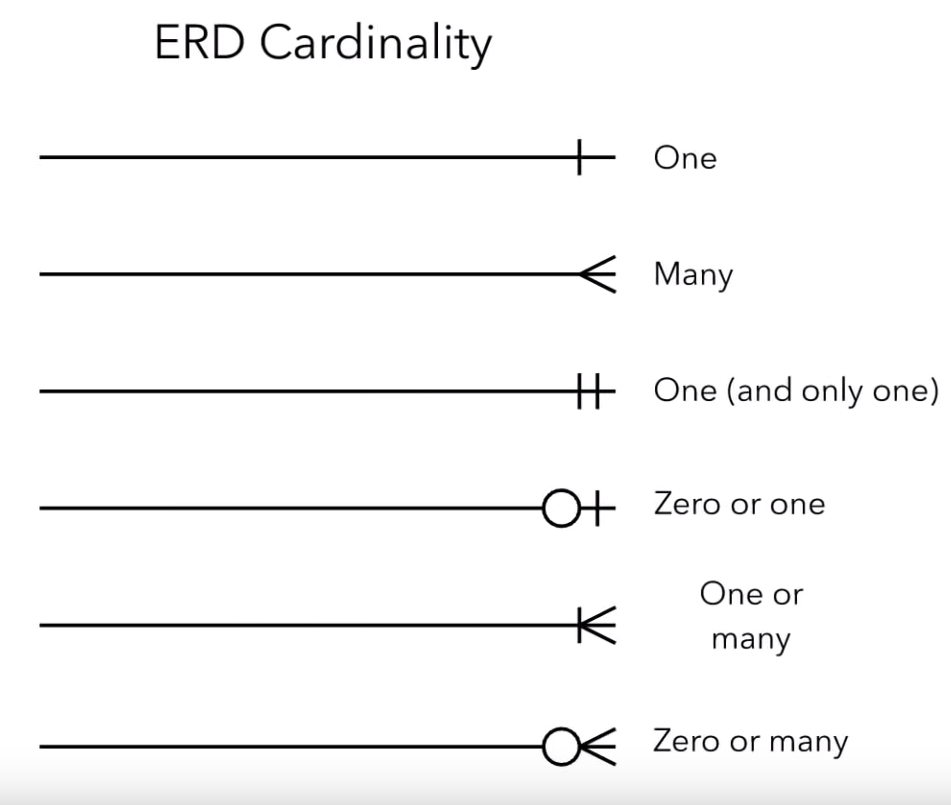


Figure - ERD Cardinality

# **Keys**

Within our entities there are some keywords as “PK”, “FK” and “CPK”, which they are corresponding to Primary Keys, Foreign Keys and Composite Primary Keys.

* **Primary Keys:** If we have a look at our database from our entity `Players` we can notice that there are a lot of players within this table. It would be nice if we could use a single trigger to quickly and accurately distinguish who is who in this massive list. That is where a Primary Key comes in. A Primary Key is an attribute or field that uniquely identifies every record in a certain table. And since a single attribute can accomplish all that, it makes sense that we will need just one Primary Key per entity. Therefore, for our `Players` table, the primary key is going to be a value that makes it, so a player is unlike any other player in our table. For one of our attributes to qualify, there are a few rules:
  + A primary key must be unique, so that it identifies only one record in our table.
  + A primary key must be never-changing. It will be difficult to keep an accurate record of the players if I use an attribute that might not stay consistent.
  + A primary key need to be never-null.

Let’s see at a specific player like Atkins Paul and determine what data might uniquely identify him. We cannot rely on Forename and Surname because 2 different players could very easily share the same first name or last name (eg. Atkins Paul, Atkins Surinder and Agg Paul). So, it is not a unique record in this table. We cannot rely on Team either because 2 separate players can be in the same team. Or a player can change team, so Team is not unique and breaks the rule of never changing. Same thing for the Status. This leaves us with PlayerID. By default, any sort of ID is typically programmed to increment for each addition to the table. We can see how PlayerID passes all the rules Paul Atkins’s ID will completely identify him as a particular instance in our database, and that value will not ever be repeated in this table. So, we are going to make that our primary key.

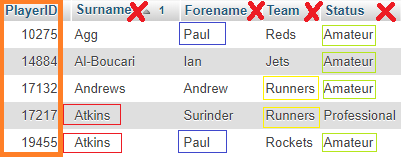


Figure - Primary Key

* **Foreign Keys:** A foreign key is the same as a primary key, but just located in a foreign place. Let’s say we have a primary key in one entity, but it would be really helpful to pull these data into another entity. That is where we get a foreign key. I already established `Team` as the Primary Key for the `Teams` entity. But that same attribute is also in the `Players` entity and the reason is because I want to know exactly in which team is each player. If we look at a specific team, it has a player of ‘Woolhouse’. This is a foreign key because it references the primary key in the `Teams` table. Now we know exactly which team we are dealing with when we look at this specific player. But the same Team is repeater right below because Sharks has another player in that team.



Figure - Foreign Key

Unlike a primary key, foreign keys do not have to be unique, they can be repeated in a table and are allowed be multiple in one entity.

* **Composite Primary Keys:** Composite primary keys can be used when two or more attributes are necessary to uniquely identify every record in a table. Let’s say ‘Gillian’ possess a skill of ‘catching’ and a skill of ‘Jumping’. When we look at these two rows and each of these attributes, we would not be able to rely on any single one to give us a unique record. The `PlayerID` is duplicated when someone has 1 or more same skills, so it is not a unique record in this table. A skill could be parsed into a couple different players so `Skills` is also not unique. In order to create a unique primary key, we need to combine these 2 attributes and create a new unique value that will not be repeated.

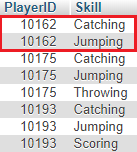


Figure - Composite Primary Key

When creating composite primary keys, we should use the fewest number of attributes possible and we should not use attributes that are apt to change because it can make things messy.

# **Table Creation**

CREATE TABLE `STATUS` (

`Status` VARCHAR(20),

PRIMARY KEY (`Status`)

);

CREATE TABLE `PLAYERSCORES` (

`ScheduledDate` DATE,

`PlayerID` INT(11),

`Points` INT(4),

KEY `CPK` (`ScheduledDate`, `PlayerID`)

);

CREATE TABLE `SCHEDULEDGAMES` (

`ScheduledDate` DATE,

`Venue` VARCHAR(20),

PRIMARY KEY (`ScheduledDate`),

KEY `FK` (`Venue`)

);

CREATE TABLE `SKILLS` (

`Skill` VARCHAR(20),

PRIMARY KEY (`Skill`)

);

CREATE TABLE `TEAMS` (

`Team` VARCHAR(20),

`Town` VARCHAR(20),

PRIMARY KEY (`Team`),

KEY `FK` (`Town`)

);

CREATE TABLE `PLAYERS` (

`PlayerID` INT(11),

`Surname` VARCHAR(20),

`Forename` VARCHAR(20),

`Team` VARCHAR(20),

`Status` ENUM('Amateur','Professional'),

PRIMARY KEY (`PlayerID`),

KEY `FK` (`Team`, `Status`)

);

CREATE TABLE `TOWNVENUE` (

`Town` VARCHAR(20),

`Venue` VARCHAR(20),

KEY `CPK` (`Town`, `Venue`)

);

CREATE TABLE `PLAYERSKILLS` (

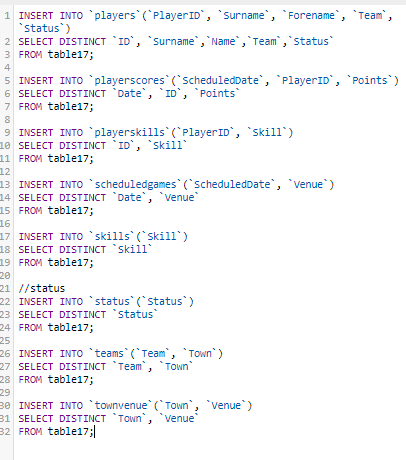
`PlayerID` INT(11),

`Skill` VARCHAR(20),

KEY `CPK` (`PlayerID`, `Skill`)

);

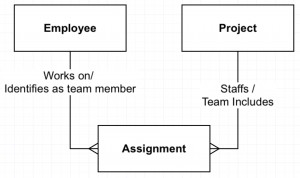
This is the SQL statement I used to create the tables in mySQL. As you can see, the table commands with the entity it is referencing are highlighted with red colour, whereas, the attributes for this entity are highlighted with green. Similarly, the data parameters that we set are highlighted with blue and the primary and foreign keys with grey colours. For `Town`, `Skill, `Forename` etc I use ***VARCHAR*** which can hold letters and numbers, whilst for `PlayerID` and `Points` I use ***INT***. However, for `Status`, due to the limited set of possible values, I decided to use ***ENUM***, and for `ScheduledDate` I use ***DATE***. Below, I used the SQL Statement to import the data from `table17` into the correct tables:



SQL Statement – Insertion

## **Resolving Many-to-Many**

Relational database management systems cannot house M:M relationships, therefore before changing a data model to a database in SQL Server the modeler should convert the M:M into something else. As shown below, the M:M between project and employee is understandably corresponding to two One-to-Many relationships among employee, project and a new entity that it is called Assignment.

[](http://robertlambert.net/wp-content/uploads/2014/10/Resolved-M-M.jpg)

The new entity that resolves a Many-to-Many is called an “association” or “intersection” entity, suggesting that it is just an index linking the other two entities. If this remains unresolved, it may reveal unexpected complexity of great interest to the business. It would have hidden important elements like `start date`, `end date`, `expected hours`, `billing rate` etc.

# **Normalization**

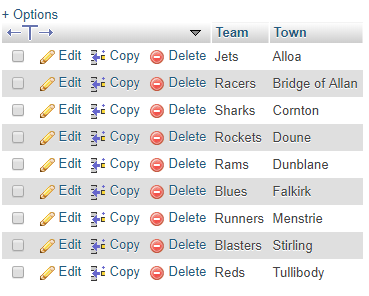
After transferring the data into the tables I will make sure that the diagram obeys the rules of 3rd Normal Form. A 3rd normal form means that all fields(columns) can be determined **only** by the Key in the table and no other column. For example, before editing my diagram, the players were so many and If I wanted to add more players to it the Skills are going to repeat again and again every single time. It would be even worse if I wanted to change one skill. I would need to go through the massive table and change the specific skill name from everywhere. Thus, they need to be separated so both of the tables are normalized properly. Obviously, the `Skill` is now Primary Key on its own table and foreign key on `Playerskills`.

# **Searches**

## **Search A**

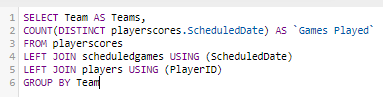


SQL Statement - Search A

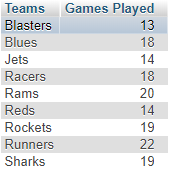


Results - Search A

## **Search B**



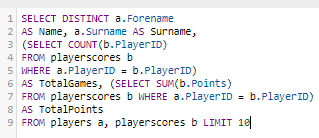
SQL Statement - Search B





Results - Search B

## **Search C**



SQL Statement - Search C

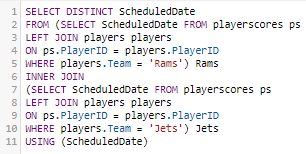




Results - Search C



## **Search D**

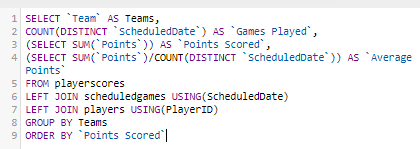


SQL Statement - Search D

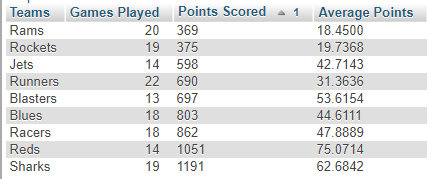


Results – Search D

## **Search E**



SQL Code - Search E



Results - Search E

# **PHP**

In the following figure you can find the PHP source code that I used:

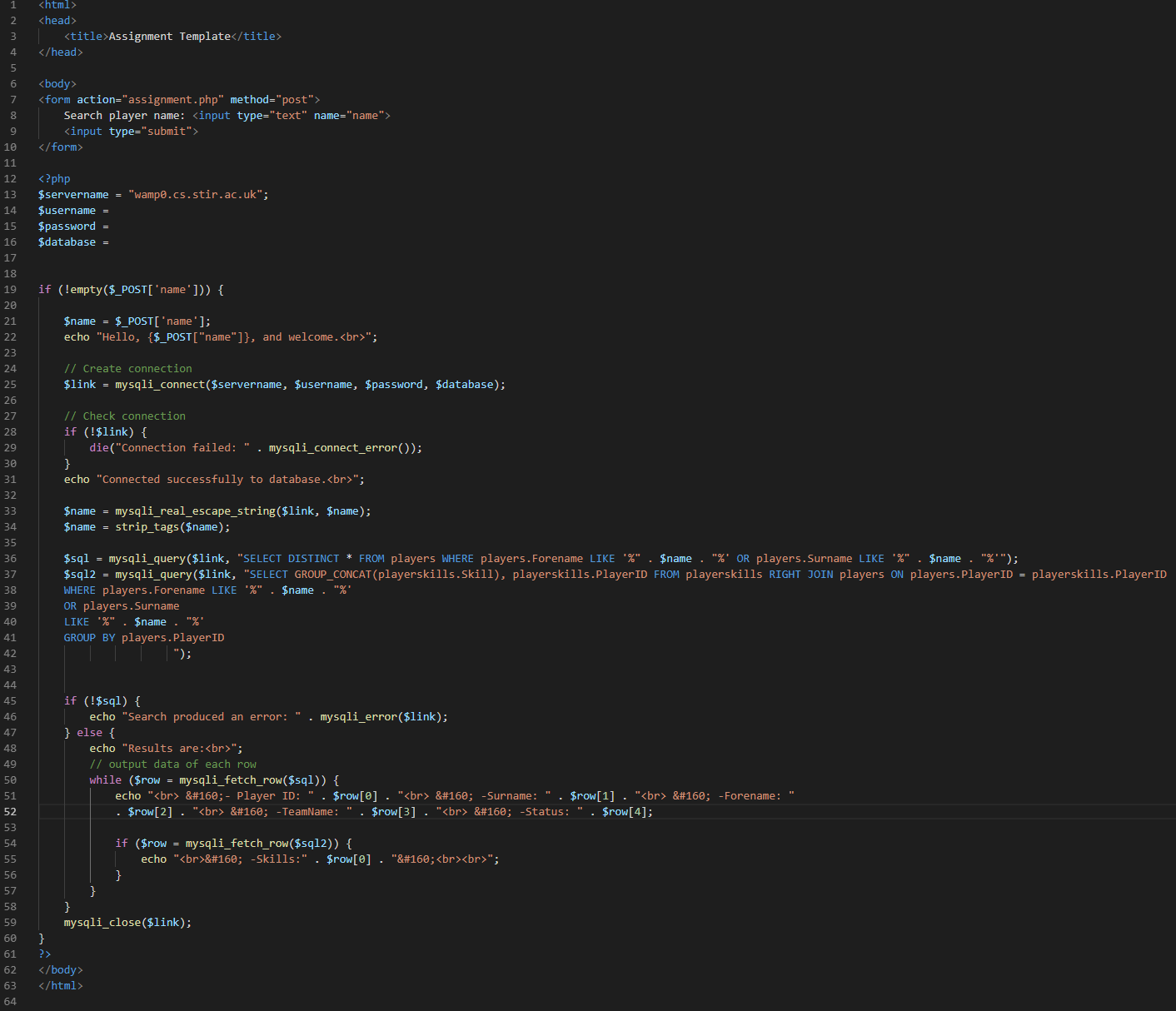


Figure - PHP Source Code

I created 2 queries ($sql and $sql2) to search for any players names (forename or surname) and display in the web page the following characteristics of all matching players found: `PlayerID`, `Forename`, `Surname`, `Team`, `Status`, `Skills`. The first one is searching from one table and retrieves all the data while the second query searches another table and concatenates all the skills that a player may have (one or more) into a single row and display it near each player correspondingly.