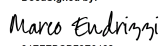


# COM1025 Web and Database Systems

## Coursework Assignment

Marco Endrizzi, 6573978, me00531

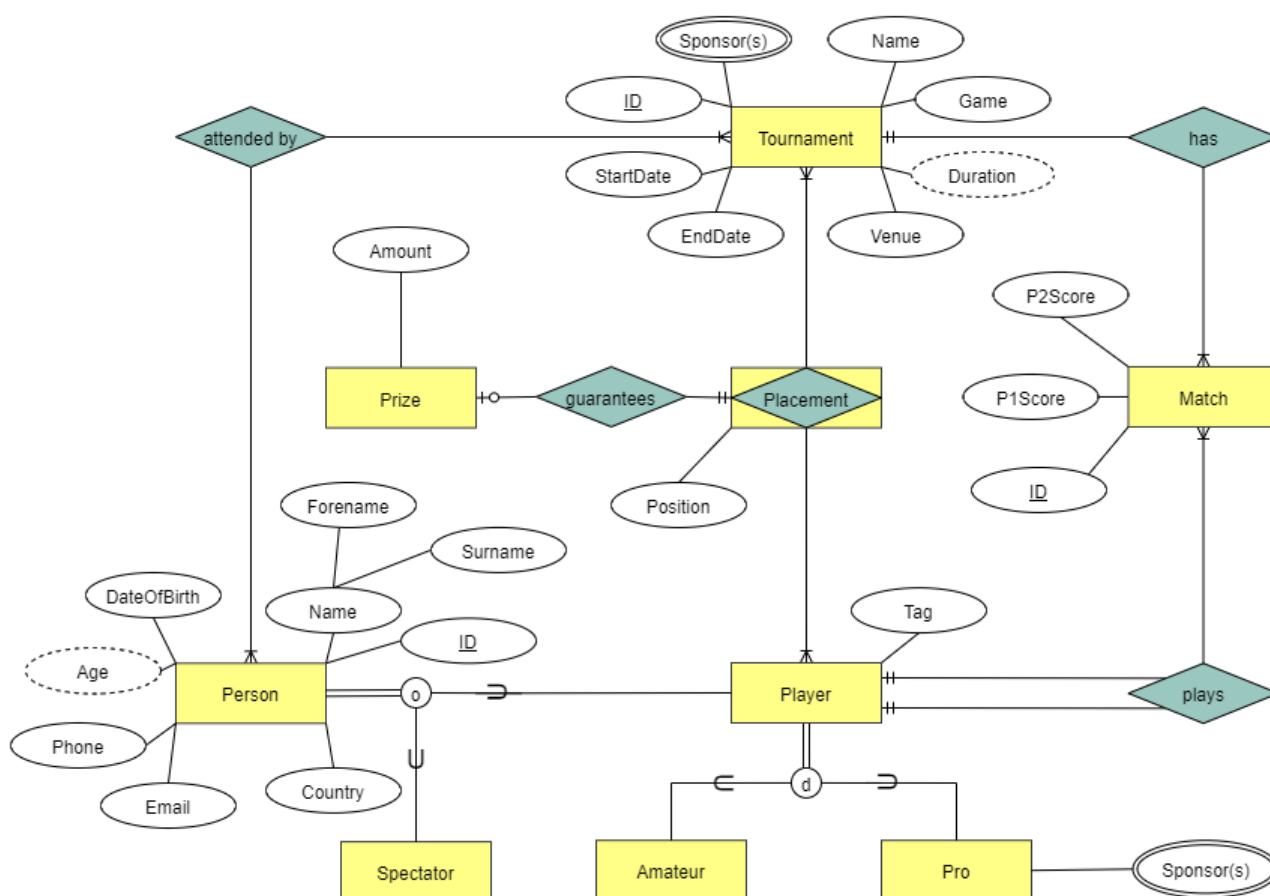
13/01/2020

DocuSigned by:  
  
 94EFEB0D78E0482  
 Marco Endrizzi  
 Mr

## 1 Developing and Testing Environment

- The figure drawing software used to draw the EER diagram and conceptual schema was draw.io.
- XAMPP 7.3.12-0 for Windows and its components were used to create and test the database.
- Google Chrome Beta 80.0.3.3987.16 and Microsoft Edge 44.17763.831.0 were used to test the database
- The operating system used during development and testing processes was Windows 10 Home 1809.

## 2 EER Data Model and Diagram



The database was designed to store information about fighting games tournaments, although it can be used for any 1v1 esports tournament format. It is designed to prevent redundant data storage, which is why a person needs only to be stored once to attend any tournament as spectator or player.

#### **Brief overview of the diagram:**

A tournament is attended by people, which can be spectators or players. Players can themselves be amateurs or pros depending on whether they have a sponsor or not. Every player must play tournament matches and are given a placement based on their performance. The top placements are usually tied with money prizes although this is not always the case.

#### **Tournament:**

- ID: Identifier for the tournament, name can't be used as a tournament can host different games
- Name: Name of the tournament
- Game: Game being played at the tournament
- Venue: Address of location where the tournament is
- StartDate: Date the tournament is set to begin
- EndDate: Date the tournament is set to end
- Duration: Duration in days for the tournament, usually fluctuates between 1 to 3 days depending on participants
- Sponsors: Sponsors for the tournament

#### **Person:**

- ID: Identifier for a person, automatically generated by the database
- Forename: Person's forename
- Lastname: Person's surname
- DateOfBirth: Person's date of birth
- Email: Person's email
- Phone: Person's phone
- Country: Person's country of origin

Person is a supertype entity with 2 subtypes: Spectator and Player. The constraint is overlap total participation as every person in the database can be both spectator and player, depending on the tournament.

#### **Player:**

- Tag: Alias used in game by player

Player is a supertype entity with 2 subtypes: Amateur and Pro. The constraint is disjoint total participation as every player in the database is either sponsored (pro) or not (amateur).

**Match:**

- ID: Identifier for matches, this is needed as multiple matches can be had between 2 players in a double elimination bracket
- P1Score: Games taken by Player 1 in the set (either Best Of 3 or Best of 5)
- P2Score: Games taken by Player 2 in the set (either Best Of 3 or Best of 5)

**Prize:**

- Amount: Money won in dollars

**Placement:**

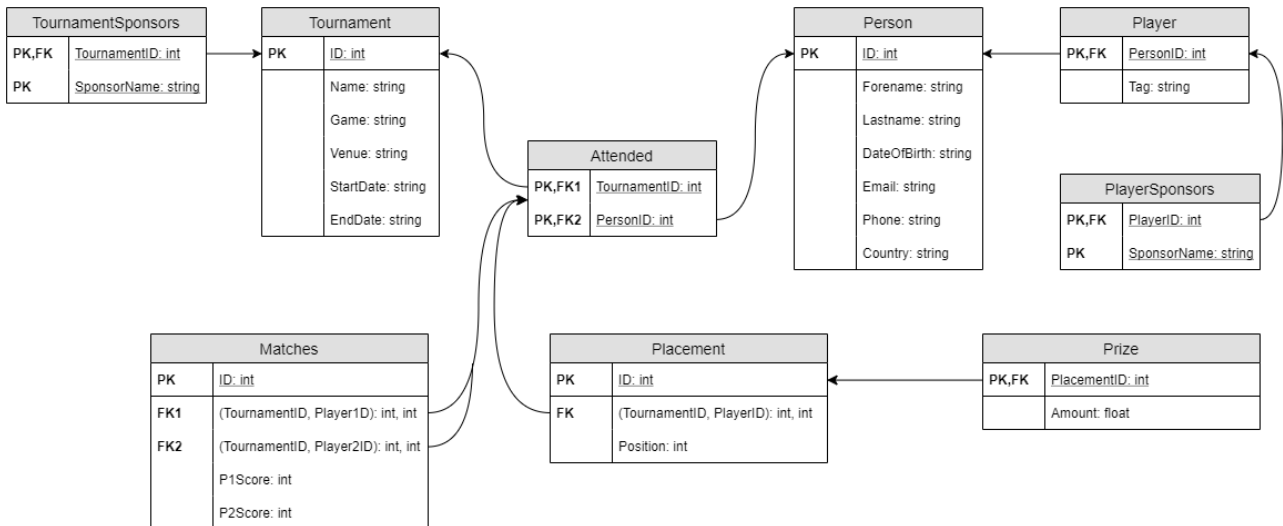
- Position: Final placement in the tournament. For a double elimination bracket, some placements will be repeated and some missing e.g. 2 players are going place tied 5<sup>th</sup> leaving out 4<sup>th</sup> place.

Placement is an associative entity as it is a relationship-like entity type with attributes linking entities Tournament and Player together.

**Relationships:**

- Attended by: Many to many total participation relationship. A tournament must be attended by many people and a person must attend at least one tournament.
- Has: One to Many total participation relationship. A tournament must have multiple matches and every match must be linked to single tournament maximum.
- Plays: One to many total participation relationship. Players must play at least a match in a tournament and a match has to be played by 2 different players.
- Guarantees: One to One partial participation relationship. A placement might be tied to a prize and a prize must be tied to a single placement.
- Placement: Many to many total participation relationship: A player must have at least a tournament placing and a tournament must have a placing for each participating player.

### 3 Conceptual/Logical Relational Database Schema



As Spectator and Amateur possess no attributes, it is not necessary to translate them to the schema, since all the data can be derived from Person and Player.

An Attended table is used to link the many to many relationship between Tournament and Person.

I have preferred not to add an ID to the Attended table for two reasons.

Firstly, the data in it is necessary for tables Matches and Placement, which get frequently requested. By not using an ID for Attended, we already have the Player and Tournament IDs in the Matches and Placement tables, leading to one less JOIN statement whenever the tables are fetched, improving execution time.

Secondly, the Attended table acts merely as a referential constraint for other tables, preventing someone not attending a specific tournament to participate in its matches, have a placement or receive a prize.

This is achieved by having tables point at the tuples PersonID and TournamentID in Attended instead of pointing at the singular IDs in the Person and Tournament tables respectively.

Note however that this does not prevent a spectator attending a tournament from placing in it (which is reserved to players). This is because, as spectators and players have both the same PersonID, the only mean to distinguish them is whether they have played any matches in the tournament.

A server side trigger is therefore used to check for this case, so that only players can be added to Placement. Matches and Prize however do not require the same trigger, as every spectator that plays matches automatically becomes a player, and Prize is tied to Placement, which is already checked.

## 4 MySQL Code

### File structure and data sources:

The .sql file is divided in 4 parts. The beginning is dedicated to dropping existing tables and triggers. The order is such that every table can be dropped without causing constraints problems. Next the tables are created with respect to the conceptual schema.

Triggers are created to throw errors server side before incorrect values can be inserted in specific tables. At the end of the file, values are inserted into the newly created tables.

Data for Players, PlayerSponsors, Tournament, Placement and Matches has been taken from [this Tekken tournament](#) and [this BBTtag tournament](#). Dummy data was created for the remaining tables.

Note that even though the two tournaments are under the same name, they are considered different, as a different game is played by different people.

### MySQL code that needs explanation:

#### Triggers:

```
-- Prevents the same player being inserted as player 1 and 2 at the same time for a match
CREATE TRIGGER four_hands BEFORE INSERT ON Matches
....FOR EACH ROW
....BEGIN
....  DECLARE msg varchar(128);
....  IF NEW.Player1ID = NEW.Player2ID THEN
....    SET msg = concat("MyTriggerError: Player ", CAST(NEW.Player1ID AS CHAR), " cant play against himself!");
....    SIGNAL SQLSTATE "45000" SET message_text = msg;
....  END IF;
....END;

(2, 15, 19, 3, 0),
(2, 19, 16, 1, 3),
-- (1, 15, 16, 1, 3),
-- (2, 15, 1, 1, 3),
(2, 15, 15, 3, 0),
(2, 15, 16, 3, 0)
```

**Messaggio di MySQL:** ⓘ

#1644 - MyTriggerError: Player 15 cant play against himself!

Although UNSIGNED is used to prevent negative values from being inserted in certain tables, more advanced error handling is needed to check values connected to players.

As players are a subtype of person and spectators have no added attributes, it is difficult to determine whether a person has attended a tournament as a spectator or player. To check for this, a trigger assures whether the person being added has played any matches in the tournament (and is therefore a player).

This trigger prevents a spectator from being able to be added to the Placement table, which is reserved to players. A second trigger is also used to make sure a player cannot play against himself in Matches. All the triggers use BEFORE INSERT ON to stop the insertion of incorrect data before it is added to the respective table. SIGNAL is used to throw an error if the condition from the IF is met. SQLSTATE "45000" simply means unhandled user-defined exception.

## Players vs Spectators

```
$res = $conn->query(
    "SELECT DISTINCT Forename, Lastname, Tag FROM Matches m
    INNER JOIN Player pl ON m.Player1ID=pl.PersonID OR m.Player2ID=pl.PersonID
    INNER JOIN Person p ON p.ID=pl.PersonID
    WHERE m.TournamentID=2"
);
if (!$res) {
    echo "Query failed: (" . $conn->errno . ") " . $conn->error;
} ?>
```

This query returns all the players' names, surnames and tags from the BBTag tournament (ID=2).

A player is defined as a person attending and playing matches.

As playing matches implies attendance (foreign key to Attended in matches ensures this), we can distinguish the players by only looking at the people playing matches.

To check for this, firstly Player (needed for Tag) and Person (needed for Forename and Lastname) are joined to Matches. WHERE is then used to return only the rows with people playing matches from the BBTag tournament. DISTINCT is used to filter out multiple rows with the same values.

```
<?php
$res = $conn->query(
    "SELECT Forename, Lastname FROM attended a
    INNER JOIN person p ON a.PersonID=p.ID
    WHERE a.TournamentID=2 AND NOT EXISTS
    (
        SELECT Player1ID, Player2ID FROM Matches m
        WHERE m.TournamentID=2 AND
        (a.PersonID=m.Player1ID OR a.PersonID=m.Player2ID)
    )"
);
if (!$res) {
    echo "Query failed: (" . $conn->errno . ") " . $conn->error;
} ?>
```

This query returns all the spectators' names and surname from the BBTag tournament (ID=2).

A spectator is defined as a person attending a tournament without playing any matches.

The outer query selects the people attending the BBTag tournament, while the subquery selects all the people playing in matches for the BBTag tournament. By using AND NOT EXISTS we are assuring that the resulting people are attending the tournament (1<sup>st</sup> query) AND NOT playing any matches (2<sup>nd</sup> query), which are the conditions for a spectator.

## Pro players vs Amateurs

```
$res = $conn->query(
    "SELECT DISTINCT Forename, Lastname, Tag, SponsorName FROM Matches m
    INNER JOIN Player pl ON m.Player1ID=pl.PersonID OR m.Player2ID=pl.PersonID
    INNER JOIN Person p ON p.ID=pl.PersonID
    INNER JOIN PlayerSponsors ps ON ps.PlayerID=pl.PersonID
    WHERE m.TournamentID=1"
);
if (!$res) {
    echo "Query failed: (" . $conn->errno . ") " . $conn->error;
} ?>
```

This query returns all the pro players' names, surnames, tags and sponsors from the Tekken tournament (ID=1).

A pro player is defined as a player having a sponsor.

To check for this the query is very similar to the Players' one, except an additional JOIN is used with PlayerSponsors to filter out all the players without sponsors.

```
<?php
$res = $conn->query(
    "SELECT DISTINCT Forename, Lastname, Tag FROM Matches m
    INNER JOIN Player pl ON m.Player1ID=pl.PersonID OR m.Player2ID=pl.PersonID
    INNER JOIN Person p ON p.ID=pl.PersonID
    LEFT JOIN PlayerSponsors ps ON ps.PlayerID=pl.PersonID
    WHERE m.TournamentID=1 AND ISNULL(ps.PlayerID)"
);
if (!$res) {
    echo "Query failed: (" . $conn->errno . ") " . $conn->error;
} ?>
```

This query returns all the amateurs players' names, surnames and tags from the Tekken tournament (ID=1).

An amateur player is defined as having no sponsor.

The query is very similar to the Pro Players' one, except this time a left join is used for PlayerSponsors. This will result in the rows with an unsponsored player having null value in the SponsorName column, which we can then filter using ISNULL() to find the unsponsored players.

## 5 Website Working with MySQL Database

### **main.php**

This file manages the connection to the mysql database, setting up and releasing the connection at the end of the script execution and displaying a message in case of an error.

To make the script portable, the variables used to connect to the database are configurable at the beginning of the script.

The HTML skeleton is also contained within this file.

To improve readability, functionality has been split in 4 different files, which are included where necessary.

### **checkbox.php.**

This file processes the input of the checkboxes in the page. It outputs a table based on the received input.

### **select.php**

This file processes the input from the dropdown selector in the page. It outputs a table based on the received input.

### **update.php**

This file is included before any select statement, to ensure that any changes to the database can be seen by the other queries, preventing data staleness.

The possible operations on the table PlayerSponsors table are UPDATE, INSERT and DELETE.

The user can interact with the table and modify data by using a simple and intuitive HTML interface.

Note that even though the database table requires the players' IDs, the user only interacts with the players' names. This is because a dropdown select mapping the player names to each respective player ID is used, so that the user only interacts with names while IDs are passed to the server.

It is therefore not necessary to perform input validation on the players' IDs, as it is impossible to input wrong or non-present values into the database.

SQL Prepared statements are used for faster execution and to protect from SQL injection attacks, as the sponsor's name is a text field. This also means that the sponsor's name is susceptible to XSS, which is prevented by sanitizing the input with htmlspecialchars.



Website

☐ Person Table

☐ Player Table

☐ Tournaments Table

☐ Invia

Tournaments table (No Venue column):

ID	Name	Game	StartDate	EndDate	Duration (days)
1	FGC AL #40	Tekken 7	2019-12-26	2019-12-26	1
2	FGC AL #40	BBTag	2019-12-26	2019-12-26	1

Tag	Sponsor
Meeks	DTS
iVesperX	FS
BigDame21	WD
Lex Steel	WISH
AxEL	DTS
RyenV	DTS
boom	DTS
boom	PRO
Meeks	

BBTag Standings

Invia

The website is separated vertically in 3 parts: on the left a checkbox can be used to print unmodified database tables, in the middle a table can be used to directly affect database values, and on the right a dropdown select can be used to print tables based on more complicated select statements.

The middle table can be used to change the values of the current row by selecting a different tag, sponsor or both, and then pressing the button. This will cause the selected row to be updated in the database.

The delete button can be pressed to delete any selected row from the database.

Values can also be inserted in the database table by choosing a player Tag, adding a Sponsor and pressing the add button.

We can check the buttons are really affecting database entries (and not just the table) by looking at the below table, which fetches its values directly from the database.

6 Advanced Tasks

All advanced tasks proposed have been attempted.

The database is normalised to BCNF as:

- There are no repeating groups (multi valued attributes TournamentSponsors and PlayerSponsors have their own tables)
- All non-key attributes are fully functional dependent on the primary key
- There is no transitive functional dependency
- For every functional dependency X->Y, X is the super key of the table.

7 References (Optional)

<https://smash.gg/tournament/fgc-al-40/events/tekken-7/overview>  
<https://smash.gg/tournament/fgc-al-40/events/bbtag/overview>