Trinity College Dublin
Coláiste na Tríonóide, Baile Átha Cliath
The University of Dublin

CSU34041 – Information Management II Project

Stephen Davis

Student Number: #18324401

Table of Contents

Section A: Description of Database Application area and ER Model

1. Application Description

The database I have decided to model is based on the golfing community in the Leinster region. Within the golfing community, I have designed the following entities:

- Player
- Coach
- Club
- Course
- Round
- Shot

Player

A player has a first name, last name, date of birth, handicap and player ID. The player ID is used to identify the specific player. A player's handicap is an indication of how good a player is at golf.

Coach

A coach has a first name, last name, a number of students (s)he teaches, his/her price per lesson and his/her unique coach ID which can be used to identify the specific coach.

Round

For the sake of simplicity, my design assumes a round of golf consists of 18 holes, no more, no less. The attributes of a round are the number of greens in regulation, the gross score, the date played, the number of points scored and the unique round ID, which is used to identify the specific round.

Shot

A shot has the club used (for example a 5-iron), the shot type, the shot number, the hole the shot was taken on, and the round ID the shot corresponds to.

Club

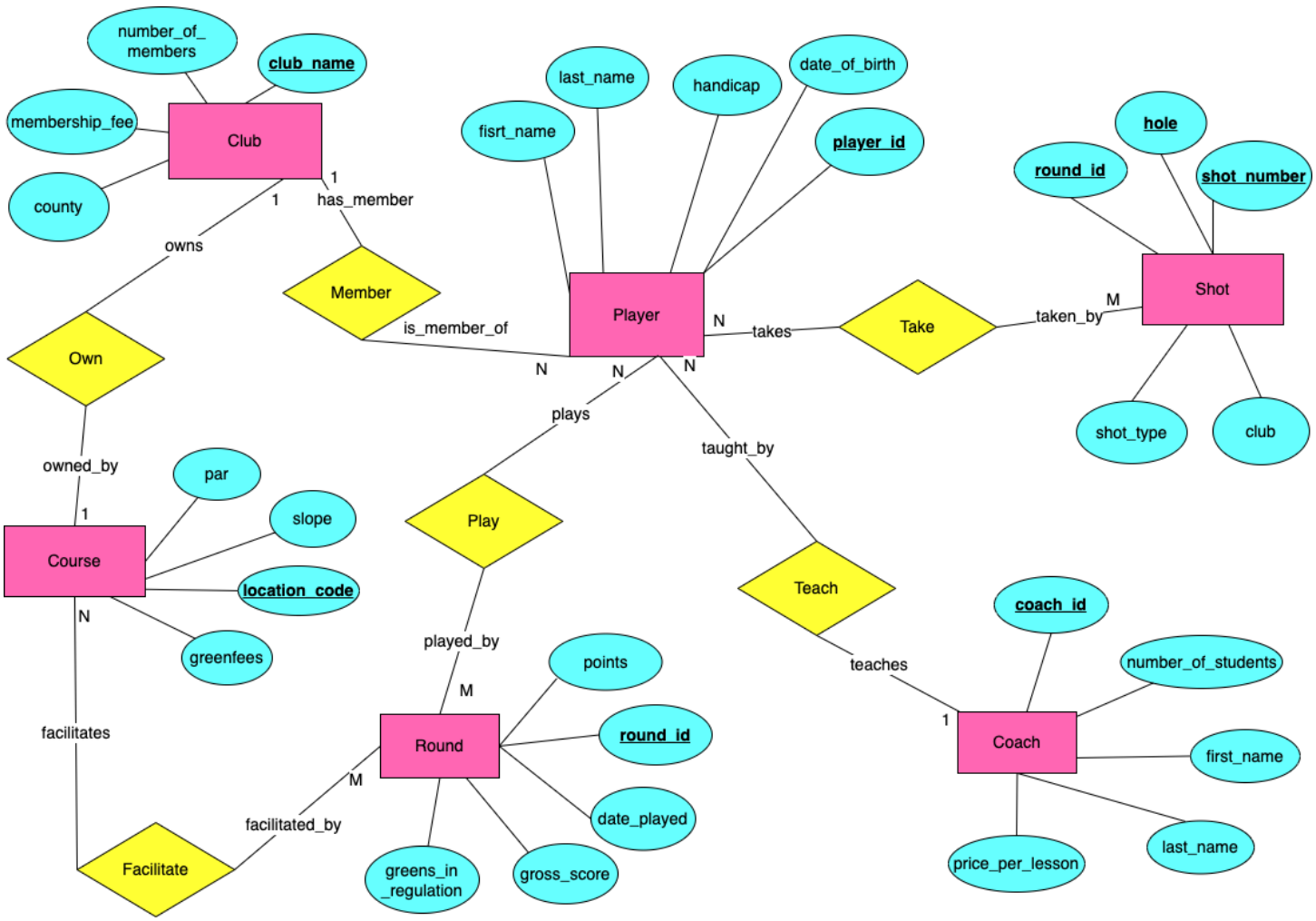
The club entity, which represents the society the player is a part of, is not to be confused with the shot's club attribute, which specifies a specific golf club (for example a 5-iron). The

attributes of the club entity are the unique club name, the number of members the club has, the membership fee and the county where the club is based. The logic to using the atomic county attribute as oppose to the composite location attribute is to adhere to table normalisation.

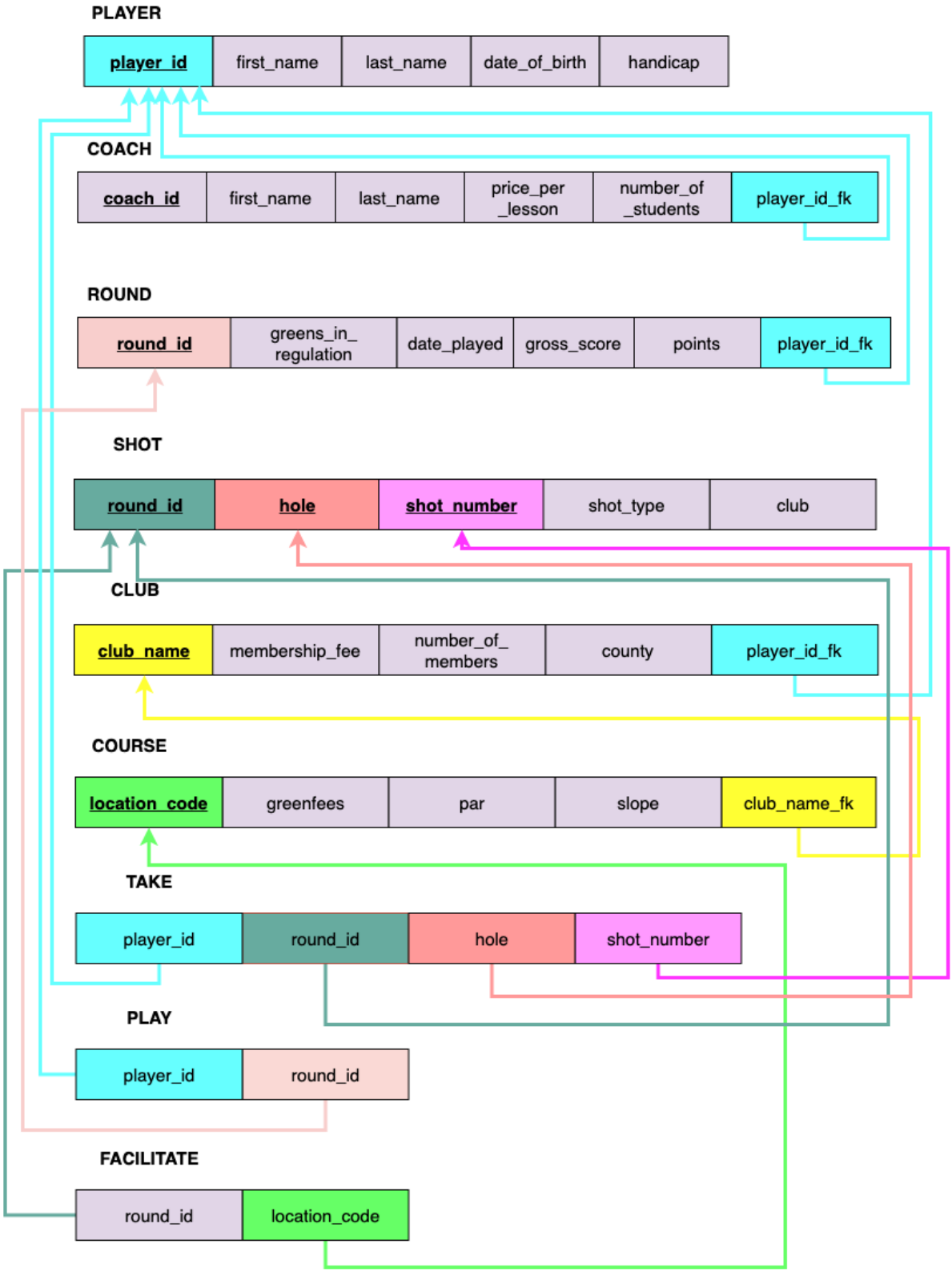
Course

The course entity has a unique location code (an EIRCODE), the greenfees (cost to play a round), the slope (difficulty rating) and the course par (the number of shots it should take you to complete a round). The logic to using the atomic location code attribute as oppose to the composite location attribute is to adhere to table normalisation.

2. Entity Relationship Diagram

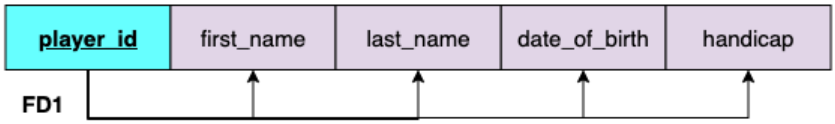


3. Mapping to Relational Schema

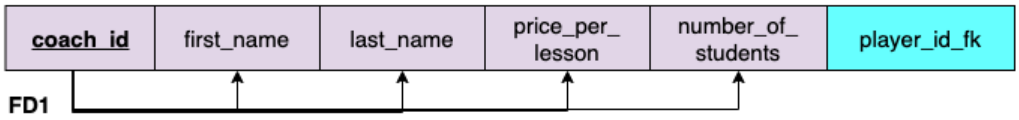


4. Functional Dependency Diagrams

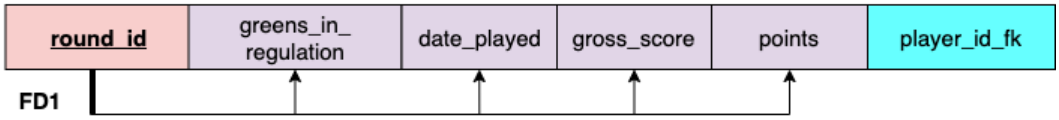
PLAYER



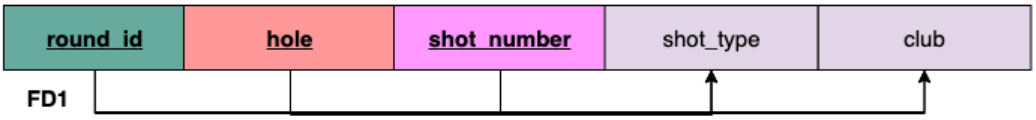
COACH



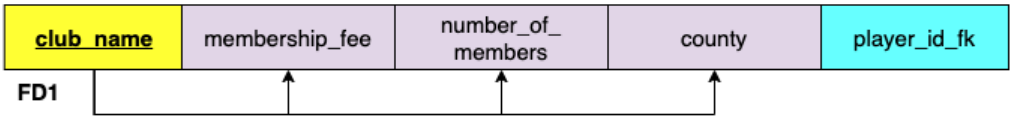
ROUND



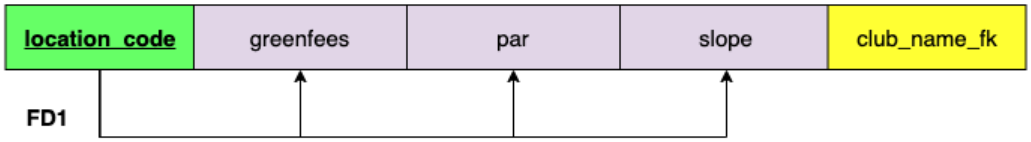
SHOT



CLUB



COURSE



TAKE

PLAY

FACILITATE

Section B Explanation of data and SQL Code:

5. Creation of Database Tables and Constraints

The first thing I do before creating a table, is to ensure the database does not already contain the table. To accomplish this I execute the DROP TABLE IF EXISTS command which will delete the table if it does exist. Then, again, before creating the table, I use the IF NOT EXISTS SQL command as an extra measure of error protection. If the table does not exist, I then create the table using the CREATE TABLE SQL commands. Each attribute is then named, followed by their type and specification CONSTRAINT as to whether the attribute can be NULL or NOT NULL. The CONSTRAINTS I have included in my design are:

1) Entity Constraint (Primary Key)
2) Relation Constraint (Foreign Key)
3) Logical Constraints (Realistic Numerical Ranges and Valid String Values)

Entity Constraint (Primary Key)

When creating a table, I assign one or more values as the primary key of the table using the PRIMARY KEY SQL command. This specifies that no 2 entries in this table may have the same value for this attribute. The PRIMARY KEY for the club relation is the club name, since, logically, no 2 clubs will have the same name.

Relation Constraint (Foreign Key)

I place the foreign key constraint on attribute(s) when creating my tables to represent that the 2 (or more) participating entities have a relationship. The foreign key must be the other participating entity's primary key in my design, since this can identify the unique entry.

Logical Constraints

Realistic Numerical Ranges

In order to align with logic, I placed a numerical constraint on the membership fee of a club to be between 100 and 2,000. Anything outside of this range is unrealistic and defies logic, hence the constraint. Similarly, a club can only exist if it has more than 1 member. Only one member would indicate an individual, not a club.

Valid String Values

Since my database is representing the golfing community of Leinster, there are only a finite number of acceptable values for the county attribute of the club. These are, of course, the counties in Leinster, "Louth", "Meath", "Dublin", "Wicklow", "Wexford", "Kilkenny", "Carlow", "Offaly", "Kildare", "Laois", "Westmeath", "Longford".

```sql
CREATE TABLE IF NOT EXISTS Club (
    club_name varchar(255) NOT NULL,
    membership_fee double NOT NULL,
    number_of_members int NULL,
    county varchar(255) NOT NULL,
    player_id_fk int NOT NULL,
    PRIMARY KEY (club_name),
    FOREIGN KEY (player_id_fk) REFERENCES Player(player_id),
    CONSTRAINT check_membership_fee CHECK (membership_fee BETWEEN 100 AND 2000),
    CONSTRAINT check_number_of_members CHECK (number_of_members > 1),
    CONSTRAINT check_county CHECK (county
                IN ('Louth', 'Meath', 'Dublin', 'Wicklow', 'Wexford', 'Kilkenny', 'Carlow', 'Offaly',
                'Kildare', 'Laois', 'Westmeath', 'Longford'))
);
```

Figure 1: Screenshot of creating Club table, highlighting the CONSTRAINTS used in creation.

Every table in my design has constraints; I have not included snippets of all of them for the sake of keeping this report at an appropriate length.

```sql
PRIMARY KEY (round_id, hole, shot_number),
CONSTRAINT check_hole CHECK (hole BETWEEN 1 AND 18),
CONSTRAINT check_shot_number CHECK (shot_number BETWEEN 1 AND 10),
CONSTRAINT check_shot_type CHECK (shot_type IN ('tee-shot', 'approach', 'pitch', 'chip', 'putt')),
CONSTRAINT check_club CHECK (club IN ('driver', 'wood', 'iron', 'wedge', 'putter'))
```

Figure 2: A snippet illustrating the CONSTRAINTS placed on the Shot table. There are only 18 holes on a course, a player will realistically only take between 1 and 10 shots on a hole (1 for a par 3 hole in 1, 10 for a difficult par 5), there are only a finite number of valid shot-types, and similarly, there are only a handful of valid clubs a player can use for a shot.

6. Altering tables

Due to new GDPR regulations in place in the Leinster golfing community, it is now an invasion of privacy to keep a record of a player's date of birth. This decision was the result of a voting system put in place for player's to share their comfort or lack of comfort with sharing their date of birth. Ultimately, a player's date of birth must be removed from the database's records.

```sql
ALTER TABLE Player
DROP COLUMN date_of_birth;
```

Figure 3: SQL Command to remove player's confidential date of birth information.

The tables in my database could have been altered by adding foreign keys, but I made a design decision to include these from the start.

7. Trigger Operations

I have one trigger implemented in my database. After entering a round of golf, the value of the points attribute of the round will affect the player's handicap. Any score higher than 36 is an indication that a player needs to have their handicap reduced and vice versa.

```
DROP TRIGGER IF EXISTS `enter_score`;
DELIMITER $$
CREATE TRIGGER `enter_score`
AFTER INSERT ON Round
FOR EACH ROW
BEGIN
    DECLARE handicap_ int;
    SELECT handicap FROM Player WHERE player_id = NEW.player_id_fk INTO handicap_;
    IF NEW.points > 36
        THEN UPDATE Player SET handicap = (handicap_ - 1) WHERE (player_id = NEW.player_id_fk);
    ELSE
        UPDATE Player SET handicap = (handicap_ + 1) WHERE (player_id = NEW.player_id_fk);
    END IF;
END; $$
```

Figure 4: A trigger which updates a player's handicap after inserting a round of golf. The update is determined by the number of points achieved in the round of golf.

8. Creation of Views

I have implemented 3 views in my project:

1) best_scores
2) coaches
3) competitors

best_scores

The logic behind this view is when a player wants to see what the best (lowest) scores have in common. By viewing this view, a player can then determine that increasing the number of greens in regulation you have during a round is vital to get a good score in a round of golf.

points	greens_in_regulation
37	10
24	2
42	18
37	11
25	1
26	0
33	4

Figure 5: best_scores view. This highlights the relationship between greens in regulation and number of points scored in a round.

coaches

The logic to this view is for when a player wants to find coaches who (s)he could possibly go to for lessons. In order to protect the coaches' privacy, their personal information such as the number of students they have and their personal ID are not visible to the player searching for the coach. All a player needs to know when deciding on which coach to go to for lessons is their name and how much they charge for a lesson to see if the player can afford the coach; players can assume that the higher the cost per lesson of the coach, the better quality/standard the coach is. One design decision I made here is that a coach's name is not unique. Additionally, I have decided to exclude a phone number for a coach, again for privacy issues. My logic is here that you can only get lessons from a coach if you have been referred from a current student, who would provide the coach's contact details.

first_name	last_name	price_per_lesson
Tom	Jones	50
Arnold	Palmer	40
Robert	Hammer	35
Ethan	Fitzpatrick	44.99
Mark	Tumble	34.99
Tom	Jones	59.99

Figure 6: coaches view. This shows public information about a coach (excluding private attribute 'number of students').

competitors

When playing a round of golf, players often want to know how good their competitors are. Additionally, players often want to find other players of a similar standard, or to see how good their friends are. This is why I have designed the competitors view. The only information a player needs on their competitor is their name and their handicap, since, by definition, a handicap is used to determine the quality or standard of a player. Giving players access to other players' date of births is unnecessary and also invades the privacy of the players whose data is being shared.

first_name	last_name	handicap
John	Lennon	18
James	Rodrigo	6
Lee	Jones	23
Albert	Smith	26
Shane	Eager	1

Figure 7: competitors view. This shows public information of other players such as their handicap.

9. One of Your Commands to Populate Tables

To populate my tables I make use of the INSERT SQL command. Only values which adhere to the table's CONSTRAINTS in place will be accepted.

```
INSERT INTO Club VALUES ('Tulfarris', 330, 400, 'Wicklow', 1);
INSERT INTO Club VALUES ('Royal Newlands', 600, 700, 'Dublin', 3);
INSERT INTO Club VALUES ('Faithlegg', 330, 450, 'Kildare', 4);
INSERT INTO Club VALUES ('Rosslare Strand', 650, 800, 'Wexford', 2);
INSERT INTO Club VALUES ('The K Club', 1200, 1000, 'Carlow', 5);
```

Figure 8: The attributes passed in to the Club table are: club name, membership fee, number of members, county and the player id (foreign key).

10. Retrieving Information from the Database

To retrieve all entries in a table the SELECT * FROM 'table-name' command can be used:

SELECT * FROM Round retrieves all the Rounds

Additionally, the previously mentioned views that have been created can retrieve information:

SELECT * FROM best_scores -> see all best scores (which contains points vs greens in regulation)

SELECT points, greens_in_regulation FROM Round achieves the same as above command.

As well as wanting to see the number of greens in regulation and points a player got in a given round, a player would also want to view a more detailed breakdown of his or her performance. This can be achieved by joining the Round and Shot tables. By analysing the result, a player can see where there is room for them to improve and they can identify what area to focus on, or what club they need to practice using.

```
SELECT Shot.round_id, hole, shot_number, shot_type FROM Shot
INNER JOIN
Round ON
Round.round_id = Shot.round_id;
```

Figure 9: The built-in INNER JOIN SQL command in action with the Shot and Round tables.

round_id	hole	shot_number	shot_type
1	1	1	tee-shot
1	1	2	approach
1	1	3	approach
1	1	4	pitch
1	1	5	putt
1	2	1	tee-shot
1	2	2	chip
1	2	3	putt
2	2	3	putt
3	15	1	tee-shot

Figure 10: The result of using the above built-in INNER JOIN SQL command.

Additionally, I have added in my own custom function which allows the net score to be calculated after a Round has been entered.

```
DELIMITER $$
CREATE FUNCTION Calculate_net_score(gross_score int, rounds_player_id int)
RETURNS INT DETERMINISTIC
BEGIN
    DECLARE handicap_ int;
    SELECT handicap FROM Player WHERE player_id = rounds_player_id INTO handicap_;
    RETURN gross_score - handicap_;
END
$$
```

Figure 11: Custom function calculating the net score based on gross score and player id for a round.

This function can then be called as seen in the below snippet, to view Rounds with a net score of <75.

```
SELECT * FROM Round WHERE Calculate_net_score(Round.gross_score, Round.player_id_fk) < 75;
```

Figure 12: SQL Command calling custom function to calculate net score of round of golf for a player.

11. Security Commands (roles & permissions)

In modelling my Leinster golf database, I have created various roles with corresponding permissions. The roles I created are:

1) chairperson
2) scorekeeper
3) scorekeeper_assistant
4) club_secretary
5) player

chairperson

The chairperson has permission to do everything in this system. The chairperson is at the top of the societal structure and hence does not need permission to do anything.

```
CREATE ROLE IF NOT EXISTS chairperson;
GRANT ALL ON golfer_database.* TO chairperson;
CREATE USER IF NOT EXISTS stephen IDENTIFIED BY 'stephen1';
GRANT chairperson TO stephen;
```

scorekeeper

The scorekeeper's duty is to enter and maintain a record of completed rounds in golf. This ensures players cannot cheat because the scorekeeper ultimately determines whether a round is inserted or not.

```
CREATE ROLE IF NOT EXISTS scorekeeper;
GRANT INSERT ON Round to scorekeeper;
CREATE USER IF NOT EXISTS ian IDENTIFIED BY 'ian1';
GRANT scorekeeper TO ian;
```

scorekeeper_assistant

While the scorekeeper is busy inserting the rounds into the database, (s)he will have an assistant who can order rounds from best to worst in order to possibly determine a winner.

```
CREATE ROLE IF NOT EXISTS scorekeeper_assistant;
GRANT SELECT ON Round TO scorekeeper_assistant;
CREATE USER IF NOT EXISTS shane IDENTIFIED BY 'shane1';
GRANT scorekeeper_assistant TO shane;
```

club_secretary

The club secretary is responsible for inputting players, new and old, into the system. The club secretary has access to a player's personal information such as their date of birth because this will allow the secretary to determine whether a player can enter a certain competition which may be for a certain age cohort.

```
CREATE ROLE IF NOT EXISTS club_secretary;
GRANT INSERT ON Player TO club_secretary;
CREATE USER IF NOT EXISTS imelda IDENTIFIED BY 'imelda1';
GRANT club_secretary TO imelda;
```

player

Finally there is the player role. A user with the role of a player can view the public details of other players, coaches and view what the best scores all have in common. Unlike the chairperson, scorekeeper, scorekeeper's assistant and the club secretary, a player has the GRANT OPTION. The logic here is that since the information a player can view is public, and does not breach any privacy concerns, there is no danger of harmful propagation forming. This is an important security feature I have incorporated into my system.

```
CREATE ROLE IF NOT EXISTS player;
GRANT SELECT ON Competitors TO player WITH GRANT OPTION;
GRANT SELECT ON Coaches TO player WITH GRANT OPTION;
GRANT SELECT ON Best_Scores TO player WITH GRANT OPTION;
CREATE USER IF NOT EXISTS mark IDENTIFIED BY 'mark1';
CREATE USER IF NOT EXISTS john IDENTIFIED BY 'john1';
CREATE USER IF NOT EXISTS liam IDENTIFIED BY 'liam1';
GRANT player TO mark;
GRANT player TO john;
GRANT player TO liam;
```

12. Additional SQL Features

Functions

As seen in the "retrieving information" section, an additional feature which I have implemented is a custom function called Calculate_net_score which takes in the gross score and player id (which is used to find the player's handicap), to calculate the net score of the round of golf achieved by the player.

```
DELIMITER $$
CREATE FUNCTION Calculate_net_score(gross_score int, rounds_player_id int)
RETURNS INT DETERMINISTIC
BEGIN
    DECLARE handicap_ int;
    SELECT handicap FROM Player WHERE player_id = rounds_player_id INTO handicap_;
    RETURN gross_score - handicap_;
END
$$
```

Figure 13: Calculate_net_score function which takes in gross score and player id to calculate net score of a player's round of golf.

Section C Listing of SQL Code for the Database

Please see attached SQL code on Blackboard.