Module Code: CS2DI17

Assignment Title: Database_GroupCoursework

Student Numbers:

- Nikolaos Dimitriou: 27001654
- Stefanos Stefanou: 270020363
- Loukia Dimosthenous: 27010011
- Petros Kavazis: 27019919
- Kyle Blue: 27004421

Date: 26/11/2019

Actual hrs spent for the assignment: 27

Assignment evaluation: Nice assignment, that test all the aspects of the database module, but it shouldn't be that time consuming while worthing only 25% of the actual course.

Link for the code in gitlab: here

Introduction

This report aims to create a database system to manage the information of the teams, players, fixtures and results of a 6-a-side football tournament organised by the University of Reading throughout the autumn term.

Initially, for the first task we had to create a Chen E-R model of the data, using as reference the lecture notes provided. We created the model using LucidChart *pro* as it's a very useful and intuitive tool for modelling. We also listed the entities, the attributes and the relationships between the entities.

Secondly, for the second task we had to create a Relational model using the E-R model from the first task.

For the third task we had to convert the relational model from the second task into a normalised model in the 3rd Normal form. In order to our analysis to be clear though , we took a step back and we fully analysed the problem ,from an unnormalized structure to 3rd normal form, providing sufficient arguments in each step of normalisation. Our final 3NF schema is consistent as a continuation of our logical and conceptual models, the additional analysis though, gives us a solid argument base to ensure that our schema is the optimal one.Having at least a 3NF schema was necessary in order to remove data anomalies, unnecessary and uncontrolled data redundancy and to ensure data integrity.

For the fourth task we had to implement our aforementioned schema using the PostgreSQL RDBMS. We started off by creating the tables we would need, using the required constraints, and populating them with the data provided in the assignment brief.

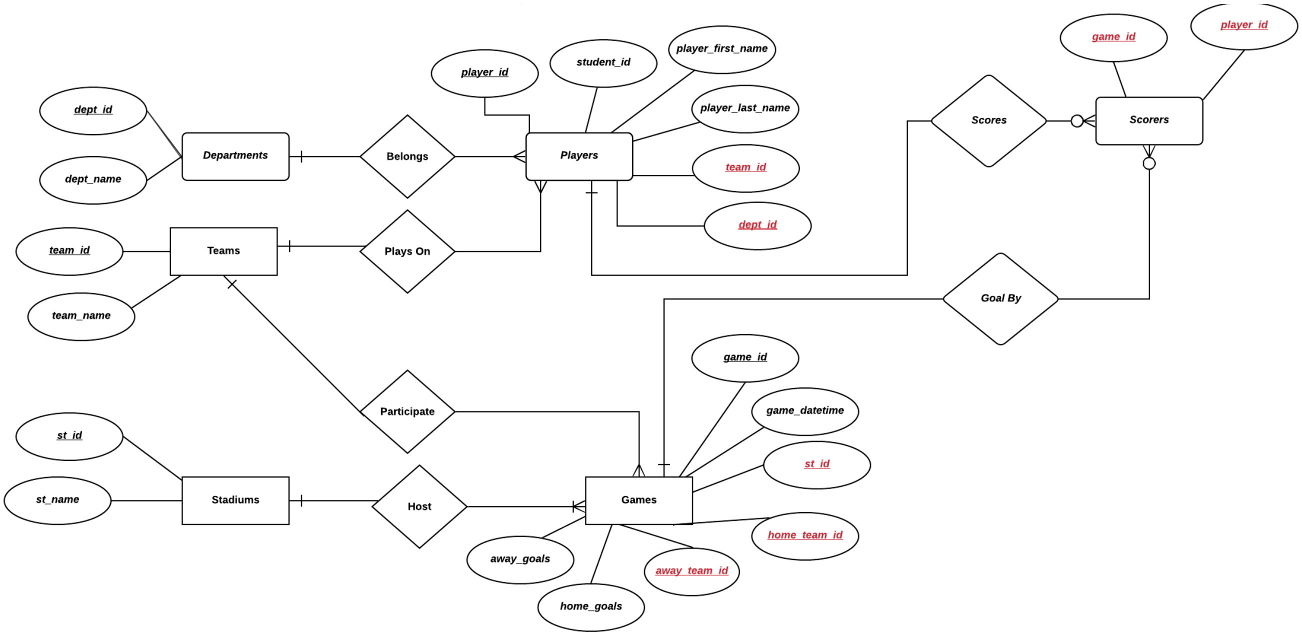
For the fifth task we tested our database implementation using test cases and queries.

Solution Design

Task 1:

In the figure below you can view the Chen E-R diagram which is based on the raw data given in the scenario. It displays the entities,attributes and relationships between entities which will later on be used to create the individual tables. Since there are relationships each entity has the appropriate foreign keys. What can be observed from the diagram is that the Scorers table is made up using foreign keys only. The diagram will be later on be used to create the non-normalised version of the Relational Diagram.

The primary key attributes are marked using the bold and underline formatting method. The formatting of the foreign keys are displayed with the red text colour. The table below shows a list of the key elements of this task.



Entities	**Attributes**	**Domains**	**Relationships**	**Assumptions**	**Constraints**
Departments	**dept_id** dept_name	BIGINT VARCHAR(50)	1:N with Players	-	dept_name NOT NULL UNIQUE
Teams	**team_id** team_name	BIGINT VARCHAR(50)	1:N with Players 1:N with Games	every team plays with each other only once(match (A,B)!=(B,A))	team_name NOT NULL UNIQUE
Stadiums	**st_id** st_name	BIGINT VARCHAR(50)	1:N with Games	-	st_name NOT NULL UNIQUE
Games	**game_id** game_datetime **st_id** **home_team_id** **away_team_id** home_goals away_goals	BIGINT TIMESTAMP INT BIGINT BIGINT	N:1 with Teams N:1 with Games 1:0..N with Scorers	-	x_goals >=0 game_datetime>(1,1,1970) (due to timestamp)

Players	**player_id** student_id player_first_name player_last_name **team_id** **dept_id**	BIGINT INT VARCHAR(50) VARCHAR(50) BIGINT BIGINT	N:1 with Departments N:1 with Teams 1:0..N with Scorers	1)Every player belongs to one Department(Not joint-degrees) 2)Every player belongs to one team	1)student_id NOT NULL UNIQUE 2)player_x_name on Unicode 3)team_id NOT NULL 4)dept_id NOT NULL
Scorers	**game_id** **player_id**	BIGINT BIGINT	0..N:1 with Games 0..N:1 with Players	-	1)player_id NOT NULL

Task 2: The Relational Model was created with reference to the Chen E-R diagram using the Date-Codd form. At first all of the domains to be used in the database are listed along with their respective data types. Then, the relations of the database are listed along with their attributes and domains. The primary key of each relation is also listed and it is assumed that they're not null. The foreign keys are also listed and state on which relation they're referenced from.

Model: University Football Tournament

Domains:

dept_id: BIGINT

dept_name: VARCHAR(50)

team_id: BIGINT

team_name: VARCHAR(50)

st_id: BIGINT

st_name: VARCHAR(50)

player_id:BIGINT

student_id:INT

player_first_name:VARCHAR(50)

player_last_name:VARCHAR(50)

game_id:BIGINT

game_datetime:TIMESTAMP

home_team_id:BIGINT

away_team_id:BIGINT

home_goals:INT

away_goals:INT

relation:**Departments**

relation: **Teams**

dept_id: BIGINT

dept_name: VARCHAR(50)

primary key dept_id

Constraint: dept_name NOT NULL

relation:**Stadiums**

st_id: BIGINT

st_name: VARCHAR(50)

primary key st_id

Constraint: st_name NOT NULL

relation: **Scorers**

game_id: BIGINT

player_id:BIGINT

foreign key game_id references Games not null

foreign key player_id references Players not null

Constraint: player_id NOT NULL

relation: **Games**

game_id:BIGINT

game_datetime: TIMESTAMP

st_id:INT

home_team_id:BIGINT

away_team_id:BIGINT

team_id:BIGINT

team_name:VARCHAR(50)

primary key team_id

Constraint: team_name NOT NULL

Assumption: Every team place with each other only once (A,B)!=(B,A)

relation: **Players**

player_id: BIGINT

student_id:INT

player_first_name:VARCHAR(50)

player_last_name:VARCHAR(50)

dept_id:BIGINT

team_id:BIGINT

primary key: player_id

foreign key dept_id references Departments not null

foreign key team_id references Teams not null

Constraints: student_id NOT NULL, player_x_name on Unicode , team_id NOT NULL, dept_id NOT NULL

Assumptions: Every player belongs to one Department (not joint degrees), Every player belongs to one team home_goals:INT

away_goals:INT

primary key game_id

foreign key st_id references Stadiums not null

foreign key home_team_id references Teams not null

foreign key away_team_id references Teams not null

Constraints: x_goals >=0, game_datetime>(1,1,1970) (due to timestamp)

Task 3: Let's start our analysis , with an unnormalized structure which contains all the necessary data to support our application

Relation : Game

game_id	game_datetime	home_team_id	home_team_name	away_team_id
away_team_name	home_team_goals	away_team_goals	home_player_id	home_player_goals
away_player_id	away_player_goals	home_dept_id	home_dept_name	away_dept_id
away_dept_name	stadium_id	stadium_name		

Relation : Scorers

Game_id	scorer_player_id

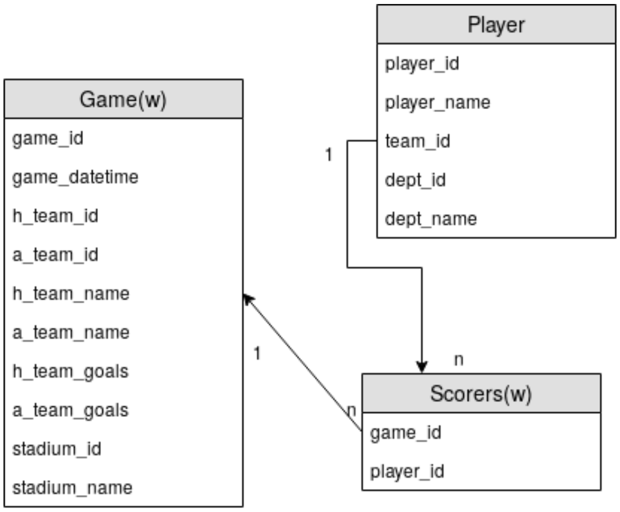
Converting our structure to 1rst Normal Form

The requirements we need to obey to be in 1NF are…

1. No repeating groups
2. No multivalued attributes
3. all non primary-key attributes must be at least partially functionally dependent to primary key

After our initial analysis , we can easily eliminate the repeating groups , by creating a new relation , based on the following functional dependencies

x_player_id -> x_team_id x_player_id -> dept_id



As a result of this , we can now create a new relation called 'Players' , this relation has 1-N connection with the Game Relation , as the following Diagram illustrates.

This eliminates the repeating groups , and ensures that every non-primary-key attribute is partially functionally dependent on their respective primary keys

Transforming our structure to 2nd Normal Form

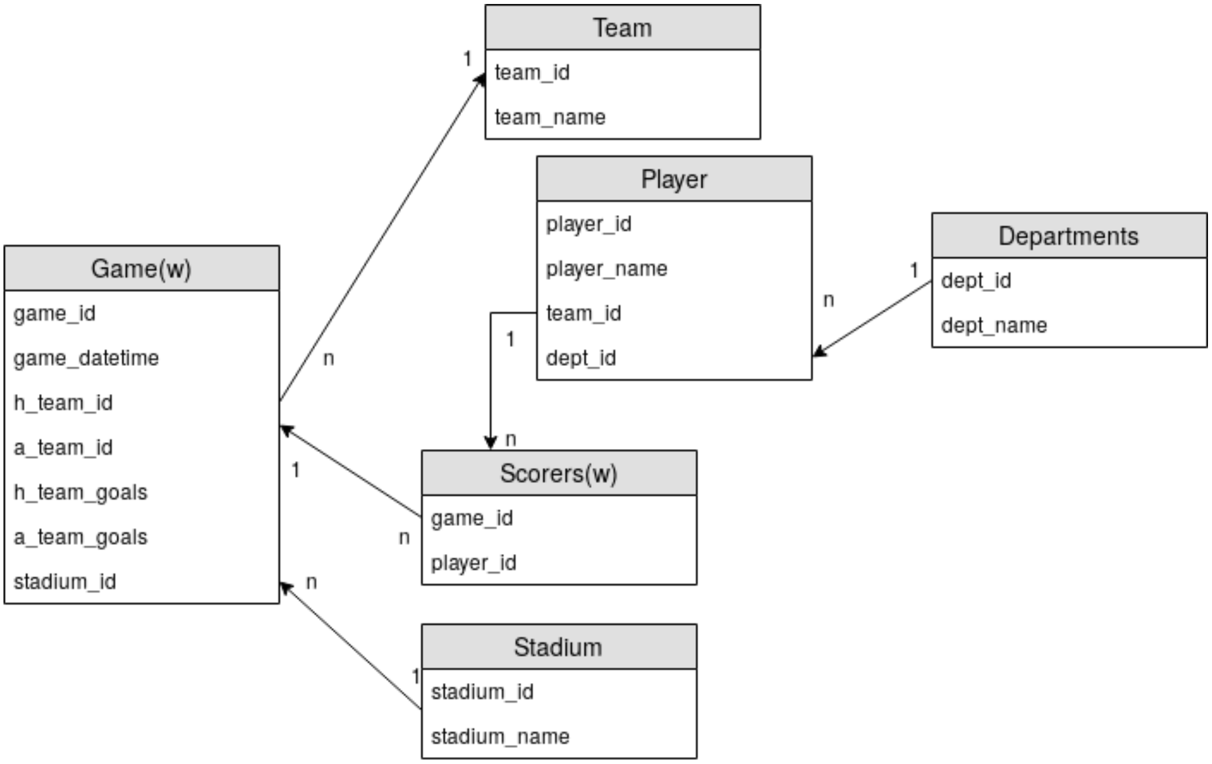
The requirements of the 2NF are ..

1. To be in 1NF
2. All non-primary-key attributes must be *fully* functional dependent to their respective primary keys

After our analysis , we can distinguish the following functional dependencies , on a *non-primary-key* attributes.

team_name -> team_id
dept_name->dept_id
stadium_name -> stadium_id
player_id -> department_id

As a result , we introduce 3 new relations , ‘Department’ , ‘Stadium’ , ‘Teams’ .



After the introduction of those relations ,the respective non-primary-attributes are only described on their primary keys(fully functional dependency).this is the requirement needed to be in 2NF

Transforming our structure to 2nd Normal Form

The requirements for 3NF are..

1. be in 2NF
2. Elimination of all transitive functional dependencies

After our analysis , we conclude that there aren't any transitive functional dependencies in our current schema , that means that our aforementioned schema is already in 3NF , and no further transformations are needed

Solution Implementation

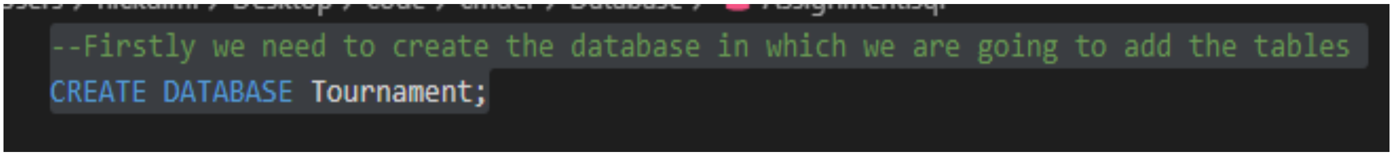
Task 4:

In this task, we will implement a Postgres schema. Some things need to be discussed first:

- Every column specified as a primary key, has a automatic constraint of not being able to have null values
- We are using bigserial for all of our primary keys, that means that all keys are of the type bigint and have a sequence that auto-increments the number (starting from one)

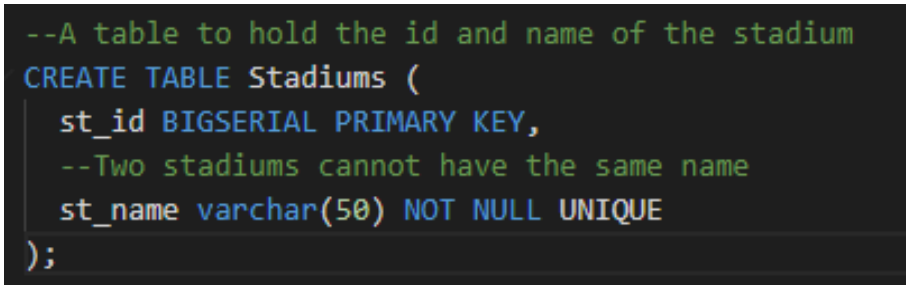
- The table were created with this order, because when you refernce another table it must be already created
- We are going to demonstrating our work by first showing the create table statement and then populating that table,when we created the database though we first created all the tables and then we populated them

Creating the database:



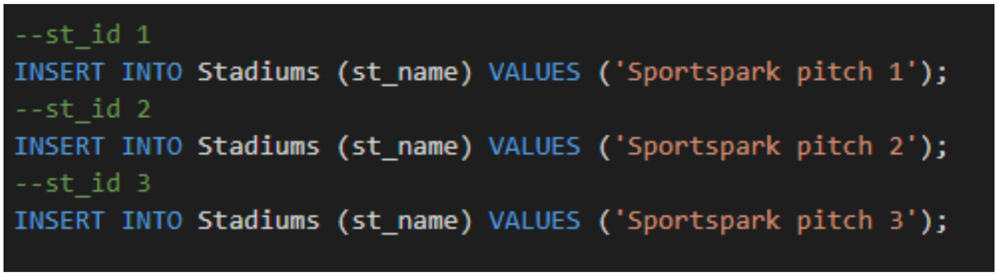
```
--Firstly we need to create the database in which we are going to add the tables
CREATE DATABASE Tournament;
```

Creating the table **Stadiums:**



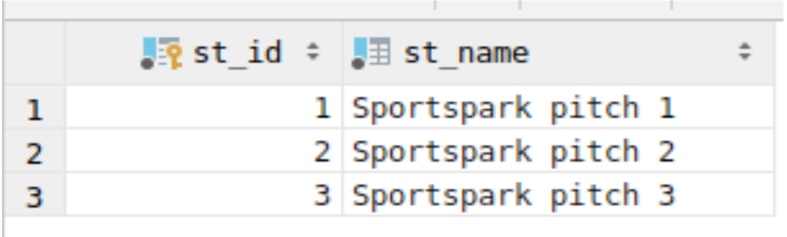
```
--A table to hold the id and name of the stadium
CREATE TABLE Stadiums (
  st_id BIGSERIAL PRIMARY KEY,
  --Two stadiums cannot have the same name
  st_name varchar(50) NOT NULL UNIQUE
);
```

Populating the table **Stadiums:**



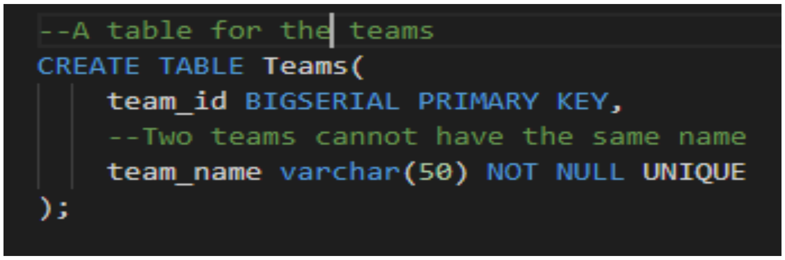
```
--st_id 1
INSERT INTO Stadiums (st_name) VALUES ('Sportspark pitch 1');
--st_id 2
INSERT INTO Stadiums (st_name) VALUES ('Sportspark pitch 2');
--st_id 3
INSERT INTO Stadiums (st_name) VALUES ('Sportspark pitch 3');
```

List of all the data in **Stadiums:**



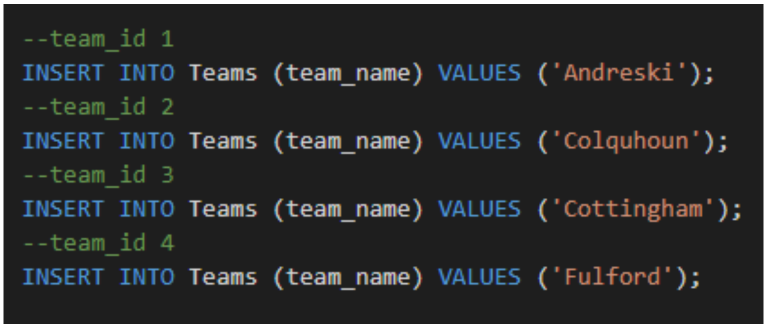
	st_id	st_name
1	1	Sportspark pitch 1
2	2	Sportspark pitch 2
3	3	Sportspark pitch 3

Creating table **Teams:**



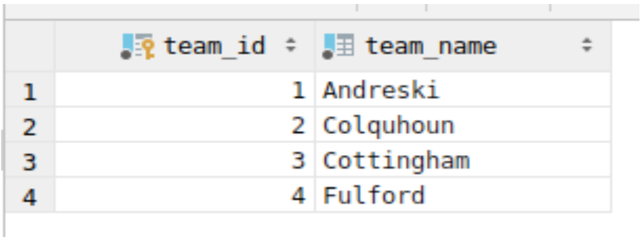
```
--A table for the teams
CREATE TABLE Teams(
    team_id BIGSERIAL PRIMARY KEY,
    --Two teams cannot have the same name
    team_name varchar(50) NOT NULL UNIQUE
);
```

Populating the table **Teams:**



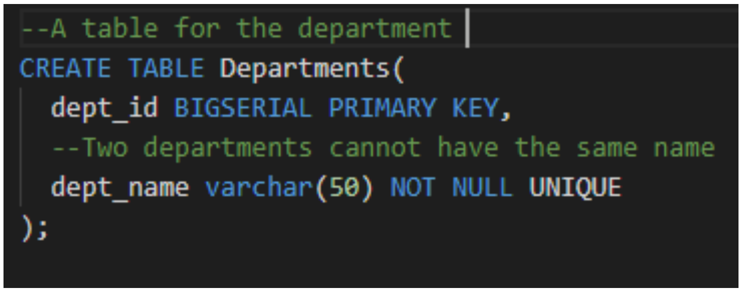
```
--team_id 1
INSERT INTO Teams (team_name) VALUES ('Andreski');
--team_id 2
INSERT INTO Teams (team_name) VALUES ('Colquhoun');
--team_id 3
INSERT INTO Teams (team_name) VALUES ('Cottingham');
--team_id 4
INSERT INTO Teams (team_name) VALUES ('Fulford');
```

List of all the data in **Teams:**



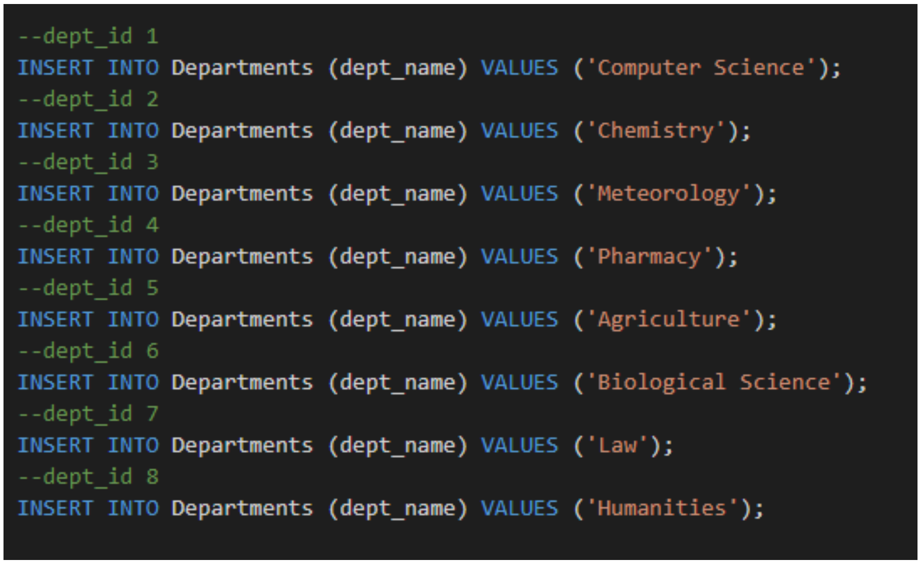
	team_id	team_name
1	1	Andreski
2	2	Colquhoun
3	3	Cottingham
4	4	Fulford

Creating table **Departments:**



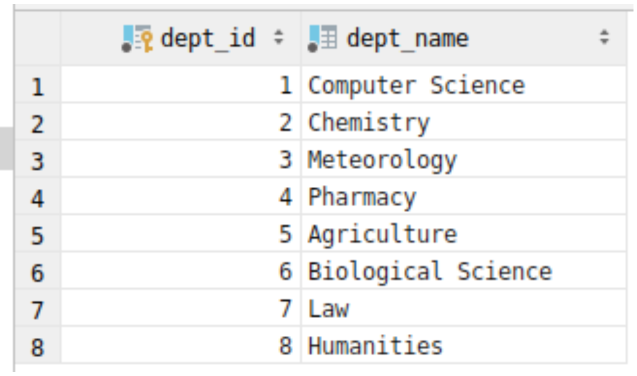
```
--A table for the department
CREATE TABLE Departments(
  dept_id BIGSERIAL PRIMARY KEY,
  --Two departments cannot have the same name
  dept_name varchar(50) NOT NULL UNIQUE
);
```

Populating the table **Departments:**



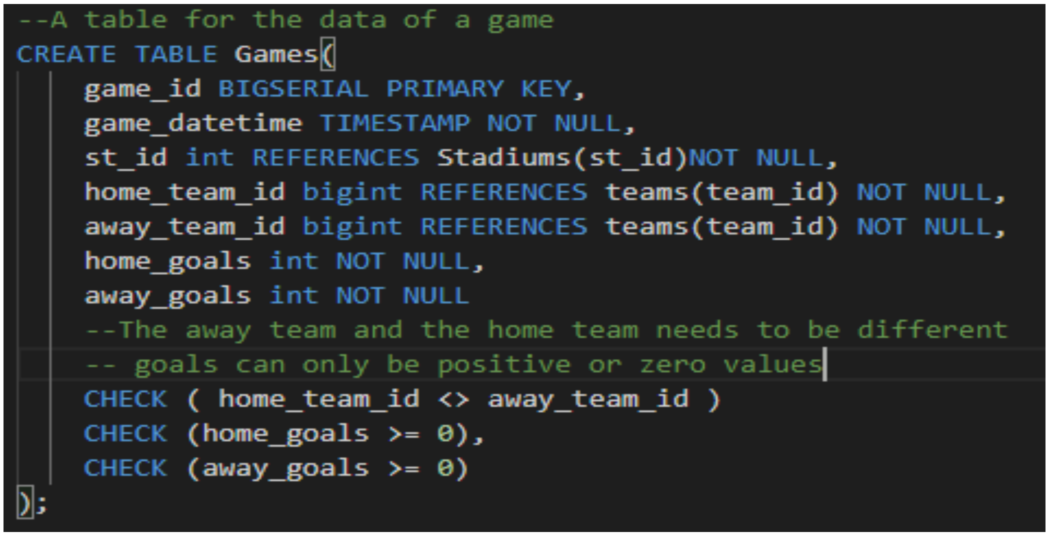
```
--dept_id 1
INSERT INTO Departments (dept_name) VALUES ('Computer Science');
--dept_id 2
INSERT INTO Departments (dept_name) VALUES ('Chemistry');
--dept_id 3
INSERT INTO Departments (dept_name) VALUES ('Meteorology');
--dept_id 4
INSERT INTO Departments (dept_name) VALUES ('Pharmacy');
--dept_id 5
INSERT INTO Departments (dept_name) VALUES ('Agriculture');
--dept_id 6
INSERT INTO Departments (dept_name) VALUES ('Biological Science');
--dept_id 7
INSERT INTO Departments (dept_name) VALUES ('Law');
--dept_id 8
INSERT INTO Departments (dept_name) VALUES ('Humanities');
```

List of all the data in **Departments:**



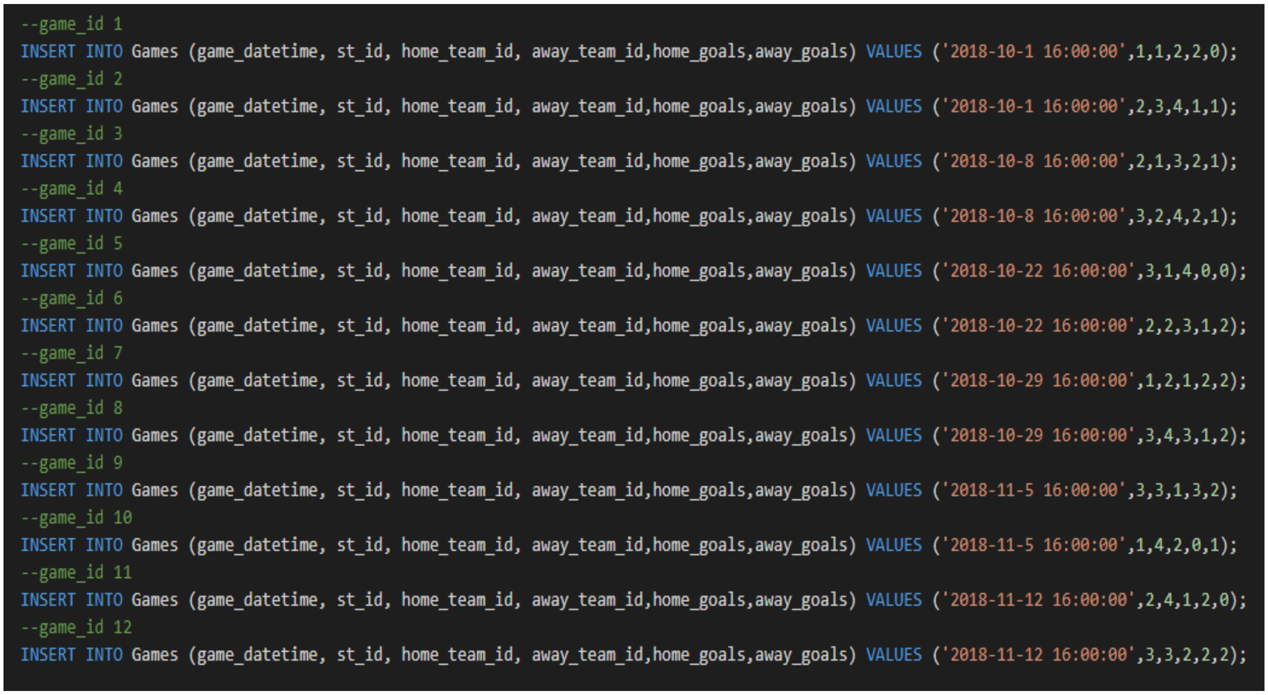
	dept_id	dept_name
1	1	Computer Science
2	2	Chemistry
3	3	Meteorology
4	4	Pharmacy
5	5	Agriculture
6	6	Biological Science
7	7	Law
8	8	Humanities

Creating table **Games:**



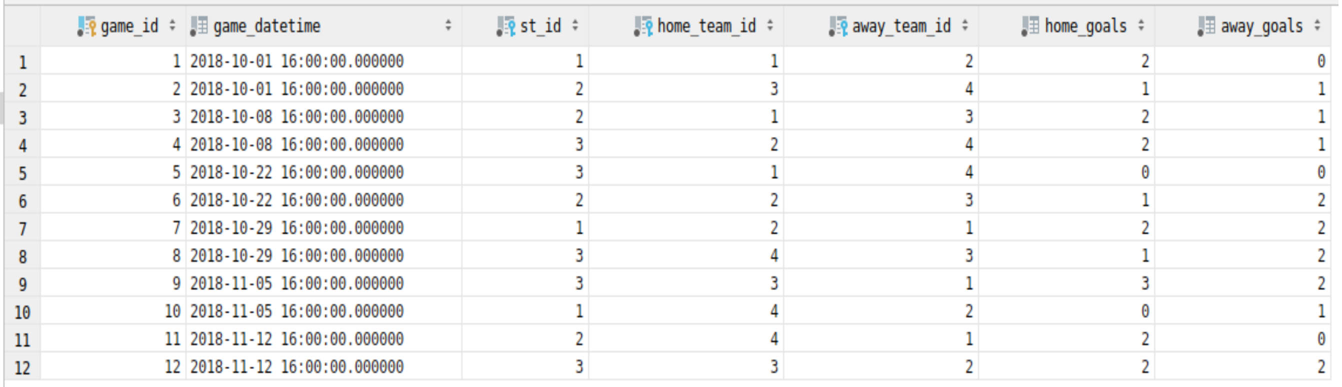
```sql
--A table for the data of a game
CREATE TABLE Games(
    game_id BIGSERIAL PRIMARY KEY,
    game_datetime TIMESTAMP NOT NULL,
    st_id int REFERENCES Stadiums(st_id)NOT NULL,
    home_team_id bigint REFERENCES teams(team_id) NOT NULL,
    away_team_id bigint REFERENCES teams(team_id) NOT NULL,
    home_goals int NOT NULL,
    away_goals int NOT NULL
    --The away team and the home team needs to be different
    -- goals can only be positive or zero values
    CHECK ( home_team_id <> away_team_id )
    CHECK (home_goals >= 0),
    CHECK (away_goals >= 0)
);
```

Populating table **Games:**



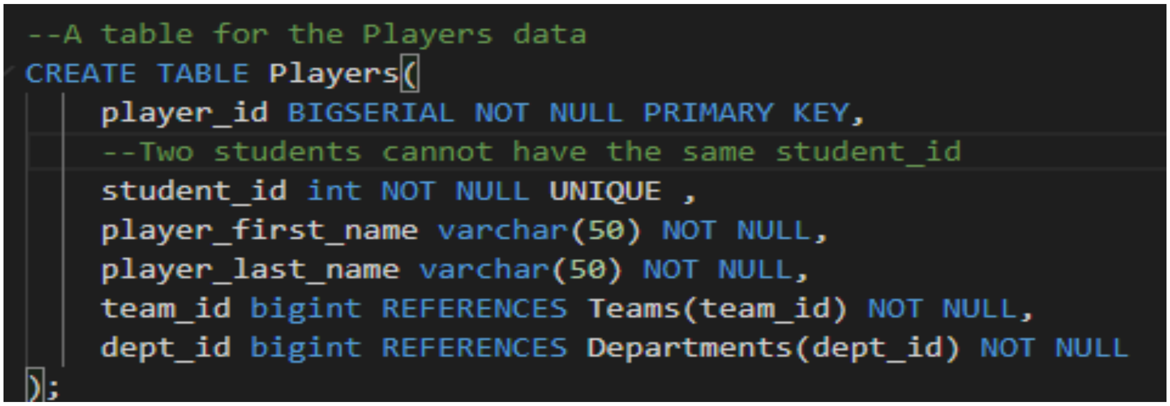
```sql
--game_id 1
INSERT INTO Games (game_datetime, st_id, home_team_id, away_team_id,home_goals,away_goals) VALUES ('2018-10-1 16:00:00',1,1,2,2,0);
--game_id 2
INSERT INTO Games (game_datetime, st_id, home_team_id, away_team_id,home_goals,away_goals) VALUES ('2018-10-1 16:00:00',2,3,4,1,1);
--game_id 3
INSERT INTO Games (game_datetime, st_id, home_team_id, away_team_id,home_goals,away_goals) VALUES ('2018-10-8 16:00:00',2,1,3,2,1);
--game_id 4
INSERT INTO Games (game_datetime, st_id, home_team_id, away_team_id,home_goals,away_goals) VALUES ('2018-10-8 16:00:00',3,2,4,2,1);
--game_id 5
INSERT INTO Games (game_datetime, st_id, home_team_id, away_team_id,home_goals,away_goals) VALUES ('2018-10-22 16:00:00',3,1,4,0,0);
--game_id 6
INSERT INTO Games (game_datetime, st_id, home_team_id, away_team_id,home_goals,away_goals) VALUES ('2018-10-22 16:00:00',2,2,3,1,2);
--game_id 7
INSERT INTO Games (game_datetime, st_id, home_team_id, away_team_id,home_goals,away_goals) VALUES ('2018-10-29 16:00:00',1,2,1,2,2);
--game_id 8
INSERT INTO Games (game_datetime, st_id, home_team_id, away_team_id,home_goals,away_goals) VALUES ('2018-10-29 16:00:00',3,4,3,1,2);
--game_id 9
INSERT INTO Games (game_datetime, st_id, home_team_id, away_team_id,home_goals,away_goals) VALUES ('2018-11-5 16:00:00',3,3,1,3,2);
--game_id 10
INSERT INTO Games (game_datetime, st_id, home_team_id, away_team_id,home_goals,away_goals) VALUES ('2018-11-5 16:00:00',1,4,2,0,1);
--game_id 11
INSERT INTO Games (game_datetime, st_id, home_team_id, away_team_id,home_goals,away_goals) VALUES ('2018-11-12 16:00:00',2,4,1,2,0);
--game_id 12
INSERT INTO Games (game_datetime, st_id, home_team_id, away_team_id,home_goals,away_goals) VALUES ('2018-11-12 16:00:00',3,3,2,2,2);
```

List of all the data in **Games:**



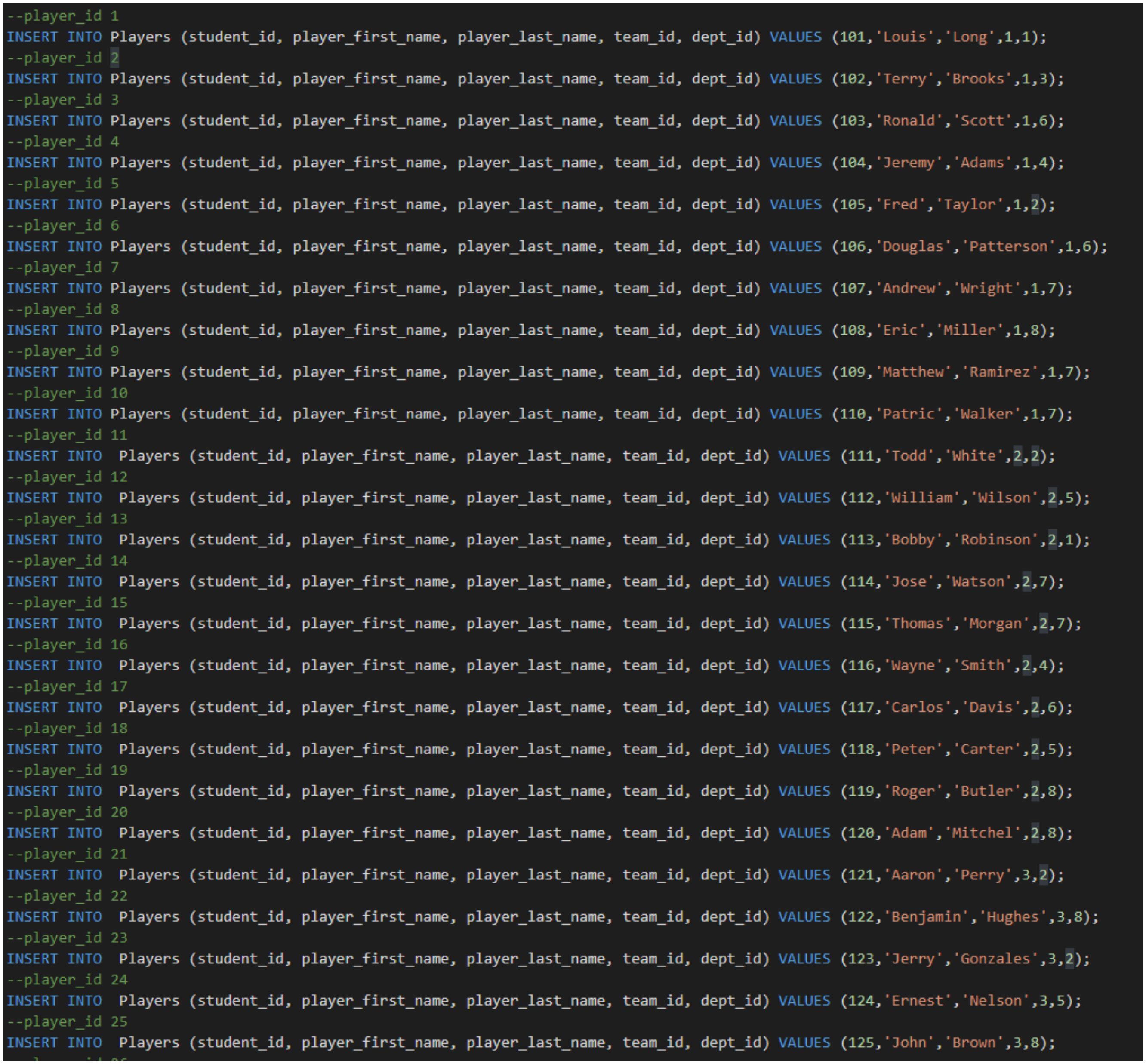
	game_id	game_datetime	st_id	home_team_id	away_team_id	home_goals	away_goals
1	1	2018-10-01 16:00:00.000000	1	1	2	2	0
2	2	2018-10-01 16:00:00.000000	2	3	4	1	1
3	3	2018-10-08 16:00:00.000000	2	1	3	2	1
4	4	2018-10-08 16:00:00.000000	3	2	4	2	1
5	5	2018-10-22 16:00:00.000000	3	1	4	0	0
6	6	2018-10-22 16:00:00.000000	2	2	3	1	2
7	7	2018-10-29 16:00:00.000000	1	2	1	2	2
8	8	2018-10-29 16:00:00.000000	3	4	3	1	2
9	9	2018-11-05 16:00:00.000000	3	3	1	3	2
10	10	2018-11-05 16:00:00.000000	1	4	2	0	1
11	11	2018-11-12 16:00:00.000000	2	4	1	2	0
12	12	2018-11-12 16:00:00.000000	3	3	2	2	2

Creating table **Players:**

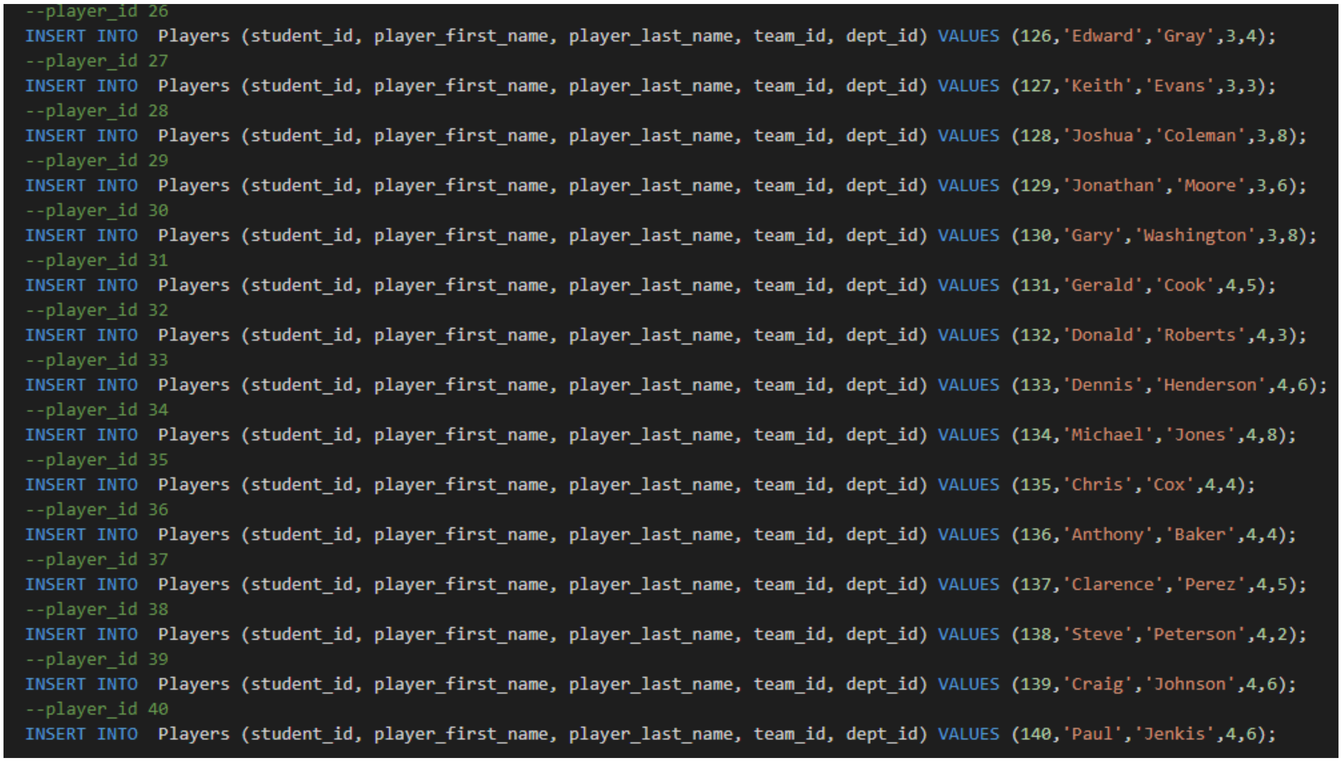


```
--A table for the Players data
CREATE TABLE Players(
    player_id BIGSERIAL NOT NULL PRIMARY KEY,
    --Two students cannot have the same student_id
    student_id int NOT NULL UNIQUE ,
    player_first_name varchar(50) NOT NULL,
    player_last_name varchar(50) NOT NULL,
    team_id bigint REFERENCES Teams(team_id) NOT NULL,
    dept_id bigint REFERENCES Departments(dept_id) NOT NULL
);
```

Populating table **Players:**

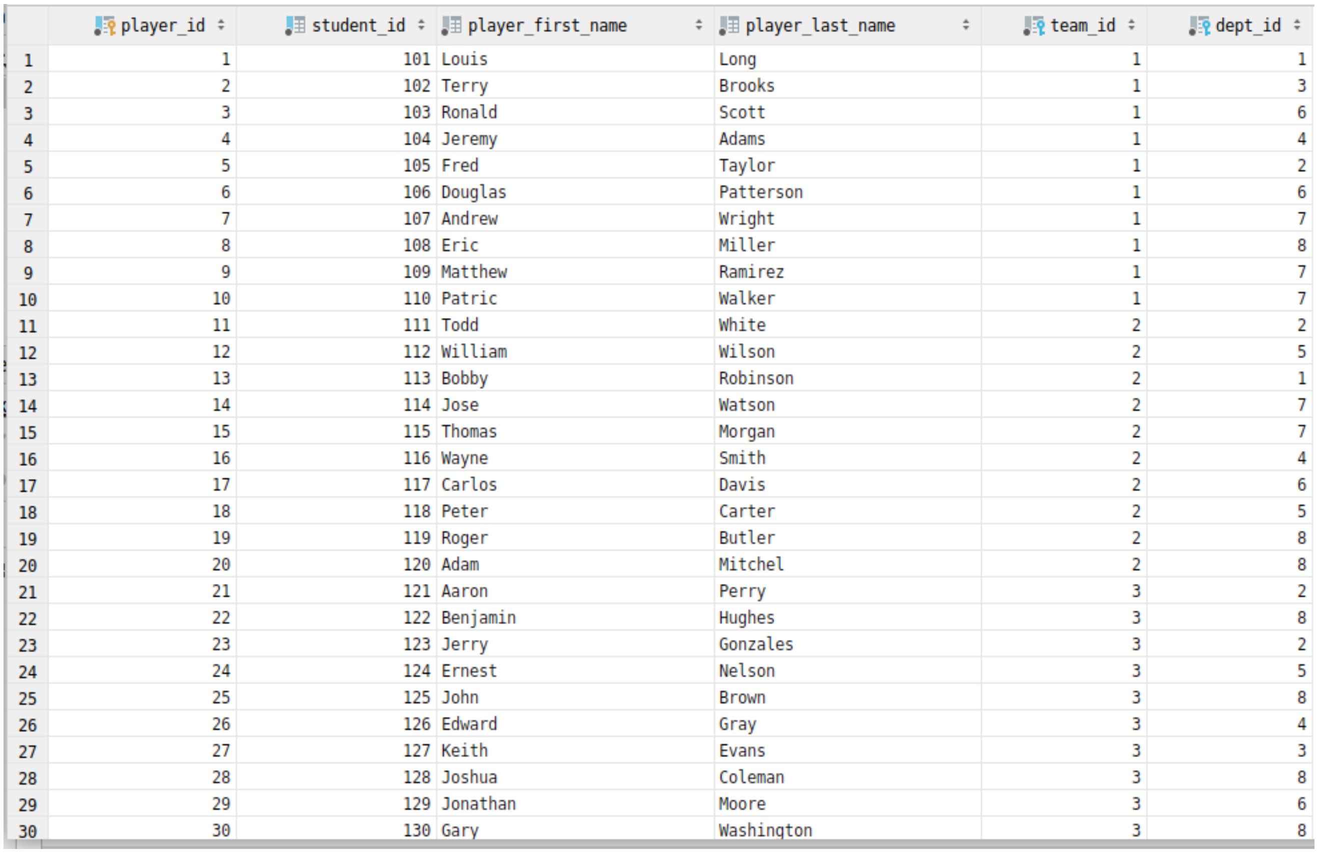


```
--player_id 1
INSERT INTO Players (student_id, player_first_name, player_last_name, team_id, dept_id) VALUES (101,'Louis','Long',1,1);
--player_id 2
INSERT INTO Players (student_id, player_first_name, player_last_name, team_id, dept_id) VALUES (102,'Terry','Brooks',1,3);
--player_id 3
INSERT INTO Players (student_id, player_first_name, player_last_name, team_id, dept_id) VALUES (103,'Ronald','Scott',1,6);
--player_id 4
INSERT INTO Players (student_id, player_first_name, player_last_name, team_id, dept_id) VALUES (104,'Jeremy','Adams',1,4);
--player_id 5
INSERT INTO Players (student_id, player_first_name, player_last_name, team_id, dept_id) VALUES (105,'Fred','Taylor',1,2);
--player_id 6
INSERT INTO Players (student_id, player_first_name, player_last_name, team_id, dept_id) VALUES (106,'Douglas','Patterson',1,6);
--player_id 7
INSERT INTO Players (student_id, player_first_name, player_last_name, team_id, dept_id) VALUES (107,'Andrew','Wright',1,7);
--player_id 8
INSERT INTO Players (student_id, player_first_name, player_last_name, team_id, dept_id) VALUES (108,'Eric','Miller',1,8);
--player_id 9
INSERT INTO Players (student_id, player_first_name, player_last_name, team_id, dept_id) VALUES (109,'Matthew','Ramirez',1,7);
--player_id 10
INSERT INTO Players (student_id, player_first_name, player_last_name, team_id, dept_id) VALUES (110,'Patric','Walker',1,7);
--player_id 11
INSERT INTO  Players (student_id, player_first_name, player_last_name, team_id, dept_id) VALUES (111,'Todd','White',2,2);
--player_id 12
INSERT INTO  Players (student_id, player_first_name, player_last_name, team_id, dept_id) VALUES (112,'William','Wilson',2,5);
--player_id 13
INSERT INTO  Players (student_id, player_first_name, player_last_name, team_id, dept_id) VALUES (113,'Bobby','Robinson',2,1);
--player_id 14
INSERT INTO  Players (student_id, player_first_name, player_last_name, team_id, dept_id) VALUES (114,'Jose','Watson',2,7);
--player_id 15
INSERT INTO  Players (student_id, player_first_name, player_last_name, team_id, dept_id) VALUES (115,'Thomas','Morgan',2,7);
--player_id 16
INSERT INTO  Players (student_id, player_first_name, player_last_name, team_id, dept_id) VALUES (116,'Wayne','Smith',2,4);
--player_id 17
INSERT INTO  Players (student_id, player_first_name, player_last_name, team_id, dept_id) VALUES (117,'Carlos','Davis',2,6);
--player_id 18
INSERT INTO  Players (student_id, player_first_name, player_last_name, team_id, dept_id) VALUES (118,'Peter','Carter',2,5);
--player_id 19
INSERT INTO  Players (student_id, player_first_name, player_last_name, team_id, dept_id) VALUES (119,'Roger','Butler',2,8);
--player_id 20
INSERT INTO  Players (student_id, player_first_name, player_last_name, team_id, dept_id) VALUES (120,'Adam','Mitchel',2,8);
--player_id 21
INSERT INTO  Players (student_id, player_first_name, player_last_name, team_id, dept_id) VALUES (121,'Aaron','Perry',3,2);
--player_id 22
INSERT INTO  Players (student_id, player_first_name, player_last_name, team_id, dept_id) VALUES (122,'Benjamin','Hughes',3,8);
--player_id 23
INSERT INTO  Players (student_id, player_first_name, player_last_name, team_id, dept_id) VALUES (123,'Jerry','Gonzales',3,2);
--player_id 24
INSERT INTO  Players (student_id, player_first_name, player_last_name, team_id, dept_id) VALUES (124,'Ernest','Nelson',3,5);
--player_id 25
INSERT INTO  Players (student_id, player_first_name, player_last_name, team_id, dept_id) VALUES (125,'John','Brown',3,8);
```

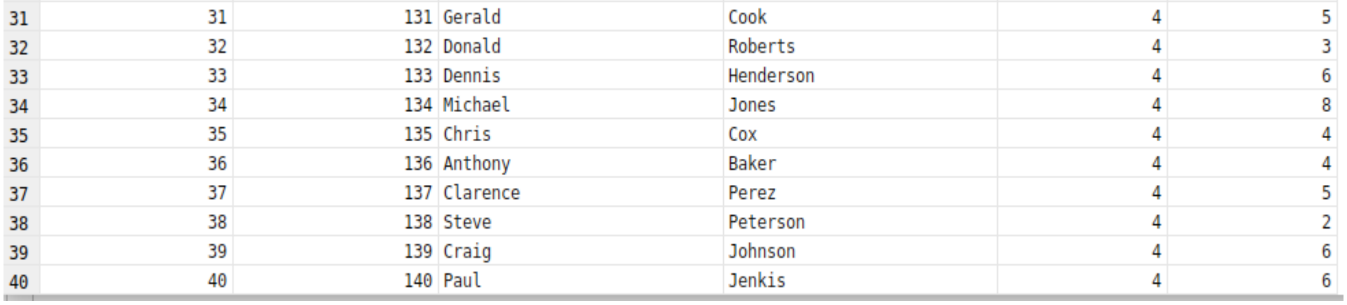


```
--player_id 26
INSERT INTO  Players (student_id, player_first_name, player_last_name, team_id, dept_id) VALUES (126,'Edward','Gray',3,4);
--player_id 27
INSERT INTO  Players (student_id, player_first_name, player_last_name, team_id, dept_id) VALUES (127,'Keith','Evans',3,3);
--player_id 28
INSERT INTO  Players (student_id, player_first_name, player_last_name, team_id, dept_id) VALUES (128,'Joshua','Coleman',3,8);
--player_id 29
INSERT INTO  Players (student_id, player_first_name, player_last_name, team_id, dept_id) VALUES (129,'Jonathan','Moore',3,6);
--player_id 30
INSERT INTO  Players (student_id, player_first_name, player_last_name, team_id, dept_id) VALUES (130,'Gary','Washington',3,8);
--player_id 31
INSERT INTO  Players (student_id, player_first_name, player_last_name, team_id, dept_id) VALUES (131,'Gerald','Cook',4,5);
--player_id 32
INSERT INTO  Players (student_id, player_first_name, player_last_name, team_id, dept_id) VALUES (132,'Donald','Roberts',4,3);
--player_id 33
INSERT INTO  Players (student_id, player_first_name, player_last_name, team_id, dept_id) VALUES (133,'Dennis','Henderson',4,6);
--player_id 34
INSERT INTO  Players (student_id, player_first_name, player_last_name, team_id, dept_id) VALUES (134,'Michael','Jones',4,8);
--player_id 35
INSERT INTO  Players (student_id, player_first_name, player_last_name, team_id, dept_id) VALUES (135,'Chris','Cox',4,4);
--player_id 36
INSERT INTO  Players (student_id, player_first_name, player_last_name, team_id, dept_id) VALUES (136,'Anthony','Baker',4,4);
--player_id 37
INSERT INTO  Players (student_id, player_first_name, player_last_name, team_id, dept_id) VALUES (137,'Clarence','Perez',4,5);
--player_id 38
INSERT INTO  Players (student_id, player_first_name, player_last_name, team_id, dept_id) VALUES (138,'Steve','Peterson',4,2);
--player_id 39
INSERT INTO  Players (student_id, player_first_name, player_last_name, team_id, dept_id) VALUES (139,'Craig','Johnson',4,6);
--player_id 40
INSERT INTO  Players (student_id, player_first_name, player_last_name, team_id, dept_id) VALUES (140,'Paul','Jenkis',4,6);
```

List of all the data in **Players:**

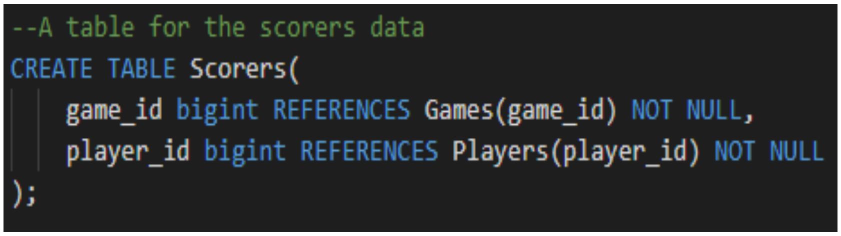


	player_id	student_id	player_first_name	player_last_name	team_id	dept_id
1	1	101	Louis	Long	1	1
2	2	102	Terry	Brooks	1	3
3	3	103	Ronald	Scott	1	6
4	4	104	Jeremy	Adams	1	4
5	5	105	Fred	Taylor	1	2
6	6	106	Douglas	Patterson	1	6
7	7	107	Andrew	Wright	1	7
8	8	108	Eric	Miller	1	8
9	9	109	Matthew	Ramirez	1	7
10	10	110	Patric	Walker	1	7
11	11	111	Todd	White	2	2
12	12	112	William	Wilson	2	5
13	13	113	Bobby	Robinson	2	1
14	14	114	Jose	Watson	2	7
15	15	115	Thomas	Morgan	2	7
16	16	116	Wayne	Smith	2	4
17	17	117	Carlos	Davis	2	6
18	18	118	Peter	Carter	2	5
19	19	119	Roger	Butler	2	8
20	20	120	Adam	Mitchel	2	8
21	21	121	Aaron	Perry	3	2
22	22	122	Benjamin	Hughes	3	8
23	23	123	Jerry	Gonzales	3	2
24	24	124	Ernest	Nelson	3	5
25	25	125	John	Brown	3	8
26	26	126	Edward	Gray	3	4
27	27	127	Keith	Evans	3	3
28	28	128	Joshua	Coleman	3	8
29	29	129	Jonathan	Moore	3	6
30	30	130	Gary	Washington	3	8



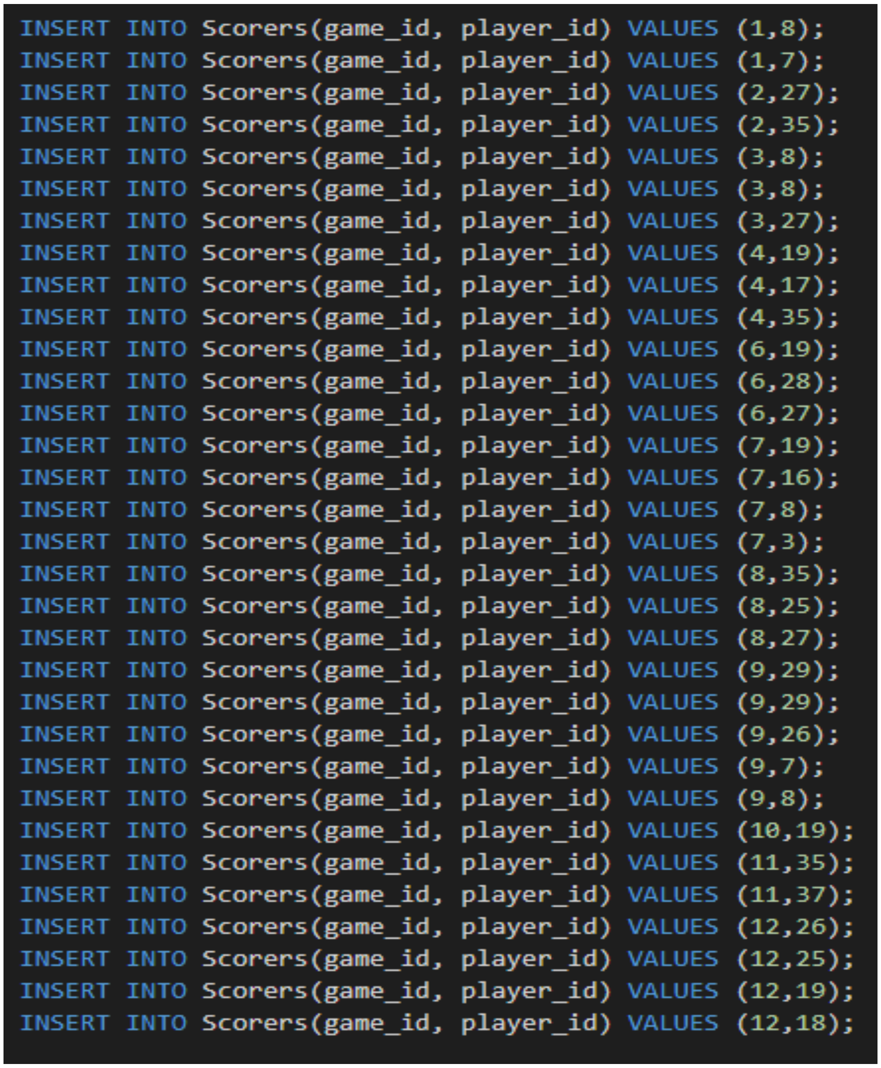
31	31	131	Gerald	Cook	4	5
32	32	132	Donald	Roberts	4	3
33	33	133	Dennis	Henderson	4	6
34	34	134	Michael	Jones	4	8
35	35	135	Chris	Cox	4	4
36	36	136	Anthony	Baker	4	4
37	37	137	Clarence	Perez	4	5
38	38	138	Steve	Peterson	4	2
39	39	139	Craig	Johnson	4	6
40	40	140	Paul	Jenkis	4	6

Creating table **Scorers:**



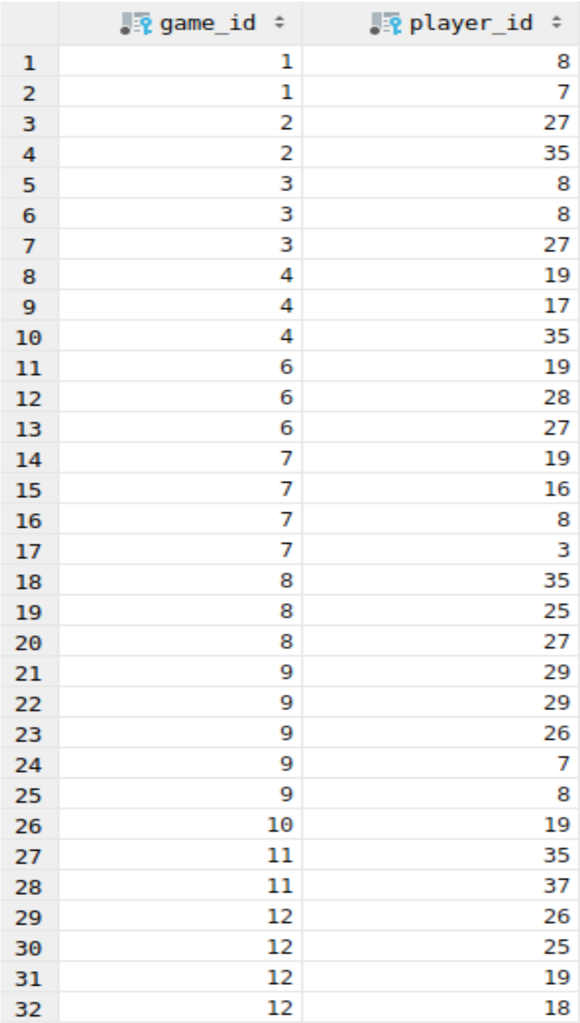
```
--A table for the scorers data
CREATE TABLE Scorers(
    game_id bigint REFERENCES Games(game_id) NOT NULL,
    player_id bigint REFERENCES Players(player_id) NOT NULL
);
```

Populating table **Scorers:**



```
INSERT INTO Scorers(game_id, player_id) VALUES (1,8);
INSERT INTO Scorers(game_id, player_id) VALUES (1,7);
INSERT INTO Scorers(game_id, player_id) VALUES (2,27);
INSERT INTO Scorers(game_id, player_id) VALUES (2,35);
INSERT INTO Scorers(game_id, player_id) VALUES (3,8);
INSERT INTO Scorers(game_id, player_id) VALUES (3,8);
INSERT INTO Scorers(game_id, player_id) VALUES (3,27);
INSERT INTO Scorers(game_id, player_id) VALUES (4,19);
INSERT INTO Scorers(game_id, player_id) VALUES (4,17);
INSERT INTO Scorers(game_id, player_id) VALUES (4,35);
INSERT INTO Scorers(game_id, player_id) VALUES (6,19);
INSERT INTO Scorers(game_id, player_id) VALUES (6,28);
INSERT INTO Scorers(game_id, player_id) VALUES (6,27);
INSERT INTO Scorers(game_id, player_id) VALUES (7,19);
INSERT INTO Scorers(game_id, player_id) VALUES (7,16);
INSERT INTO Scorers(game_id, player_id) VALUES (7,8);
INSERT INTO Scorers(game_id, player_id) VALUES (7,3);
INSERT INTO Scorers(game_id, player_id) VALUES (8,35);
INSERT INTO Scorers(game_id, player_id) VALUES (8,25);
INSERT INTO Scorers(game_id, player_id) VALUES (8,27);
INSERT INTO Scorers(game_id, player_id) VALUES (9,29);
INSERT INTO Scorers(game_id, player_id) VALUES (9,29);
INSERT INTO Scorers(game_id, player_id) VALUES (9,26);
INSERT INTO Scorers(game_id, player_id) VALUES (9,7);
INSERT INTO Scorers(game_id, player_id) VALUES (9,8);
INSERT INTO Scorers(game_id, player_id) VALUES (10,19);
INSERT INTO Scorers(game_id, player_id) VALUES (11,35);
INSERT INTO Scorers(game_id, player_id) VALUES (11,37);
INSERT INTO Scorers(game_id, player_id) VALUES (12,26);
INSERT INTO Scorers(game_id, player_id) VALUES (12,25);
INSERT INTO Scorers(game_id, player_id) VALUES (12,19);
INSERT INTO Scorers(game_id, player_id) VALUES (12,18);
```

List of all tha data in **Scorers:**



	game_id	player_id
1	1	8
2	1	7
3	2	27
4	2	35
5	3	8
6	3	8
7	3	27
8	4	19
9	4	17
10	4	35
11	6	19
12	6	28
13	6	27
14	7	19
15	7	16
16	7	8
17	7	3
18	8	35
19	8	25
20	8	27
21	9	29
22	9	29
23	9	26
24	9	7
25	9	8
26	10	19
27	11	35
28	11	37
29	12	26
30	12	25
31	12	19
32	12	18

Task 5:

Test 1 - Listing all students who play for a particular department

1. This test will validate that we are able to list all students who play in a particular department (e.g. Computer Science or Biological Science) through the use of join between two related tables (departments and players). We will test for the Biological Science department.

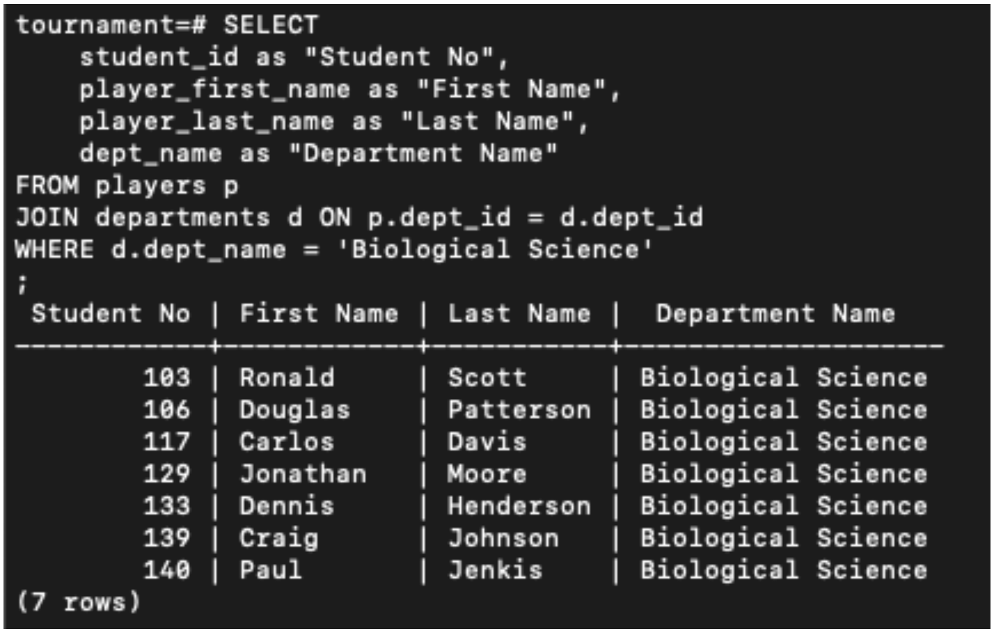
2. The expected outcome of the test is:

Student No	First Name	Last Name	Department Name

103	Ronald	Scott	Biological Sciences
106	Douglas	Patterson	Biological Sciences
117	Carlos	Davis	Biological Sciences
129	Jonathan	Moore	Biological Sciences
133	Dennis	Henderson	Biological Sciences
139	Craig	Johnson	Biological Sciences
140	Paul	Jenkins	Biological Sciences

This was extracted and compiled from the original set of data provided.

3. Both the query used in the test and the resultant table are provided in the screenshot below:



```
tournament=# SELECT
    student_id as "Student No",
    player_first_name as "First Name",
    player_last_name as "Last Name",
    dept_name as "Department Name"
FROM players p
JOIN departments d ON p.dept_id = d.dept_id
WHERE d.dept_name = 'Biological Science'
;
 Student No | First Name | Last Name |  Department Name
------------+------------+-----------+--------------------
        103 | Ronald     | Scott     | Biological Science
        106 | Douglas    | Patterson | Biological Science
        117 | Carlos     | Davis     | Biological Science
        129 | Jonathan   | Moore     | Biological Science
        133 | Dennis     | Henderson | Biological Science
        139 | Craig      | Johnson   | Biological Science
        140 | Paul       | Jenkis    | Biological Science
(7 rows)
```

4. As you can see, the resulting table directly matches and mirrors the expected result, proving the test successful.

Test 2 - Listing all fixtures for a specific date (i.e. 29th of October 2018)

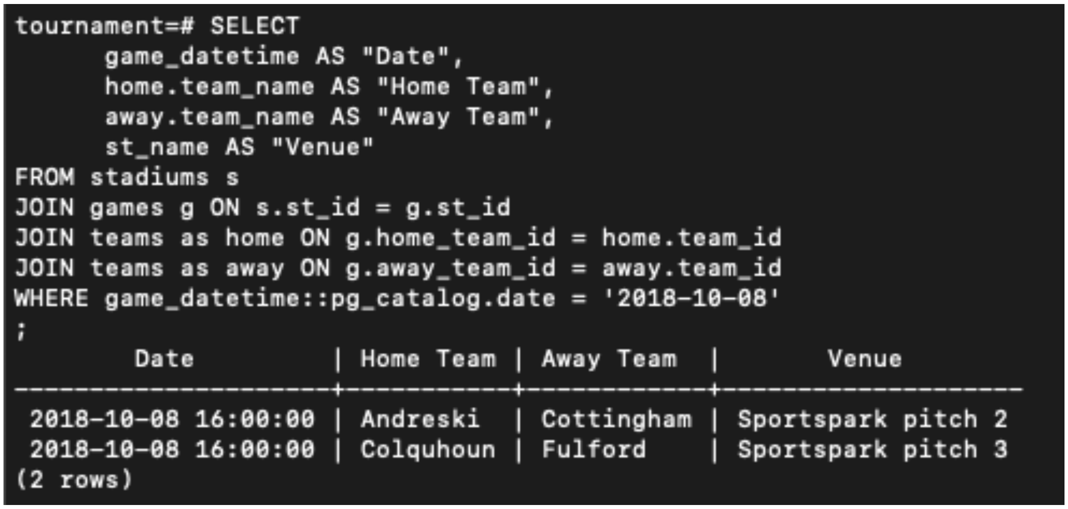
1. This test aims to ensure that we are able to list all fixtures (games / events) that are associated with a specific date. The date we decided to test through our query was the 8th October 2018 (08/10/2018).

2. The expected outcome for this test in particular is:

Date - Time	Home Team	Away Team	Venue
08/10/2018 16:00	Andreski	Cottingham	Pitch 2
08/10/2018 16:00	Colquhoun	Fulford	Pitch 3

Again, the expected result was extracted from the data provided.

3. Both the query used in the test and the resultant table are provided in the screenshot below:



```
tournament=# SELECT
      game_datetime AS "Date",
      home.team_name AS "Home Team",
      away.team_name AS "Away Team",
      st_name AS "Venue"
FROM stadiums s
JOIN games g ON s.st_id = g.st_id
JOIN teams as home ON g.home_team_id = home.team_id
JOIN teams as away ON g.away_team_id = away.team_id
WHERE game_datetime::pg_catalog.date = '2018-10-08'
;
        Date         | Home Team | Away Team  |       Venue
---------------------+-----------+------------+---------------------
 2018-10-08 16:00:00 | Andreski  | Cottingham | Sportspark pitch 2
 2018-10-08 16:00:00 | Colquhoun | Fulford    | Sportspark pitch 3
(2 rows)
```

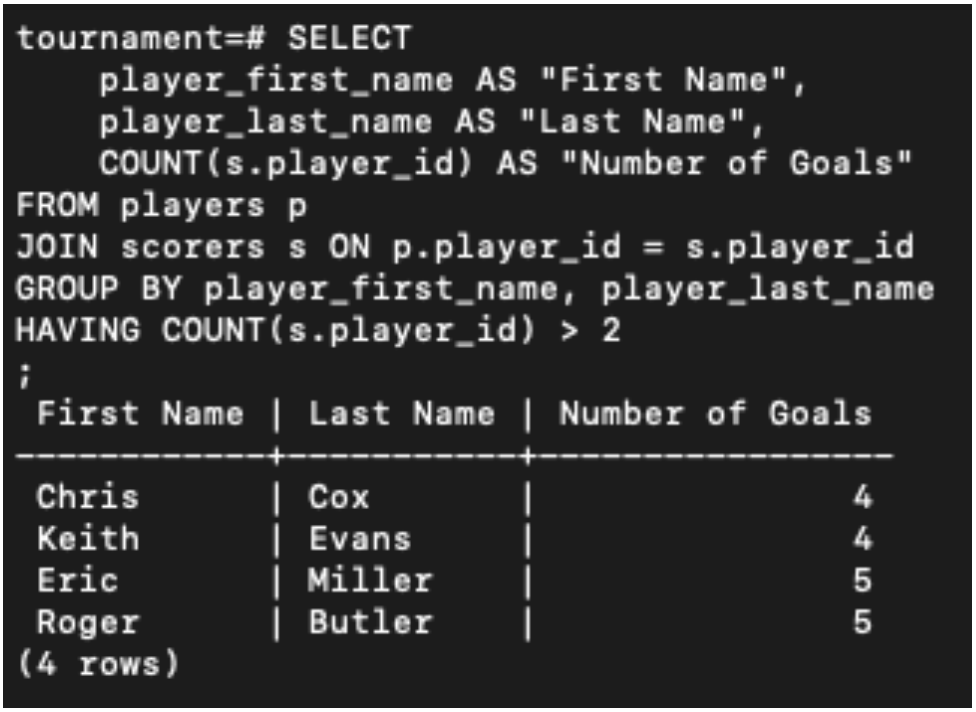
4. In this instance, the result of the query executed exactly matches the foreseen / expected outcome (should we account for the fact that Sportpark pitch 2 maps to Pitch 2 in the provided data, and a slight date_time representation difference). This implies a pass on this test.

Test 3 - Listing all the players who have scored more than 2 goals

1. This particular test aims to affirm that we can list all players who have scored more than two goals in our database.This test would require a join between two tables (scorers and players) and calculation using the COUNT sql aggregated function.
2. The expected result, which has been put together through the extraction of the provided data, is displayed below:

First Name	Last Name	Number of Goals
Chris	Cox	4
Keith	Evans	4
Eric	Miller	5
Roger	Butler	5

3. The screenshot below of terminal output displays both the query used for this test, and the returned result.



```
tournament=# SELECT
    player_first_name AS "First Name",
    player_last_name AS "Last Name",
    COUNT(s.player_id) AS "Number of Goals"
FROM players p
JOIN scorers s ON p.player_id = s.player_id
GROUP BY player_first_name, player_last_name
HAVING COUNT(s.player_id) > 2
;
 First Name | Last Name | Number of Goals
------------+-----------+-----------------
 Chris      | Cox       |               4
 Keith      | Evans     |               4
 Eric       | Miller    |               5
 Roger      | Butler    |               5
(4 rows)
```

1. The result of the query turned out to completely remember the expected result, allowing us to conclude test 4 as a success. No unnecessary data, such as names of players without more than 2 goals, was displayed.

Test 4 - Return the total number of goals scored in the season.

1. This test will validate whether or not our database enables us to extract the number of goals scored throughout the entire season, by any team. Our schema allowed us to execute this query with relative ease, omitting the need for any joins.
2. Adding the total goals scored from each game consecutively (using data provided) like so:

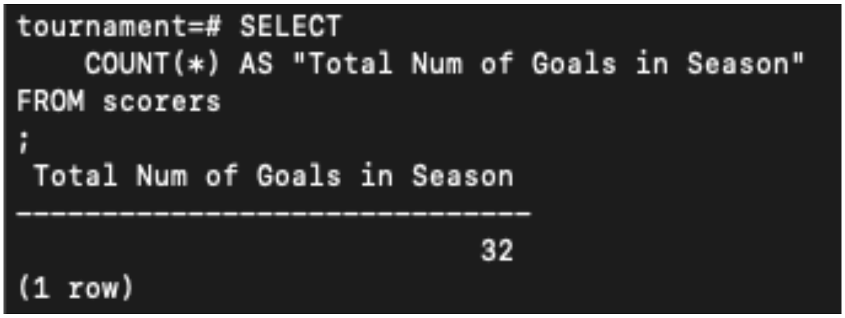
2+2+3+3+0+3+4+3+5+1+2+4

Results in a total of 32 goals throughout the entire season.

This means the resultant table should look like so:

Total Num of Goals in Season
32

1. Below displays the query used for this test and its corresponding result



```
tournament=# SELECT
    COUNT(*) AS "Total Num of Goals in Season"
FROM scorers
;
 Total Num of Goals in Season
------------------------------
                           32
(1 row)
```

1. As we can see, the result of the query and expected output match, yet again implying test success. The result was simply extracted through counting all records in the scorers table.

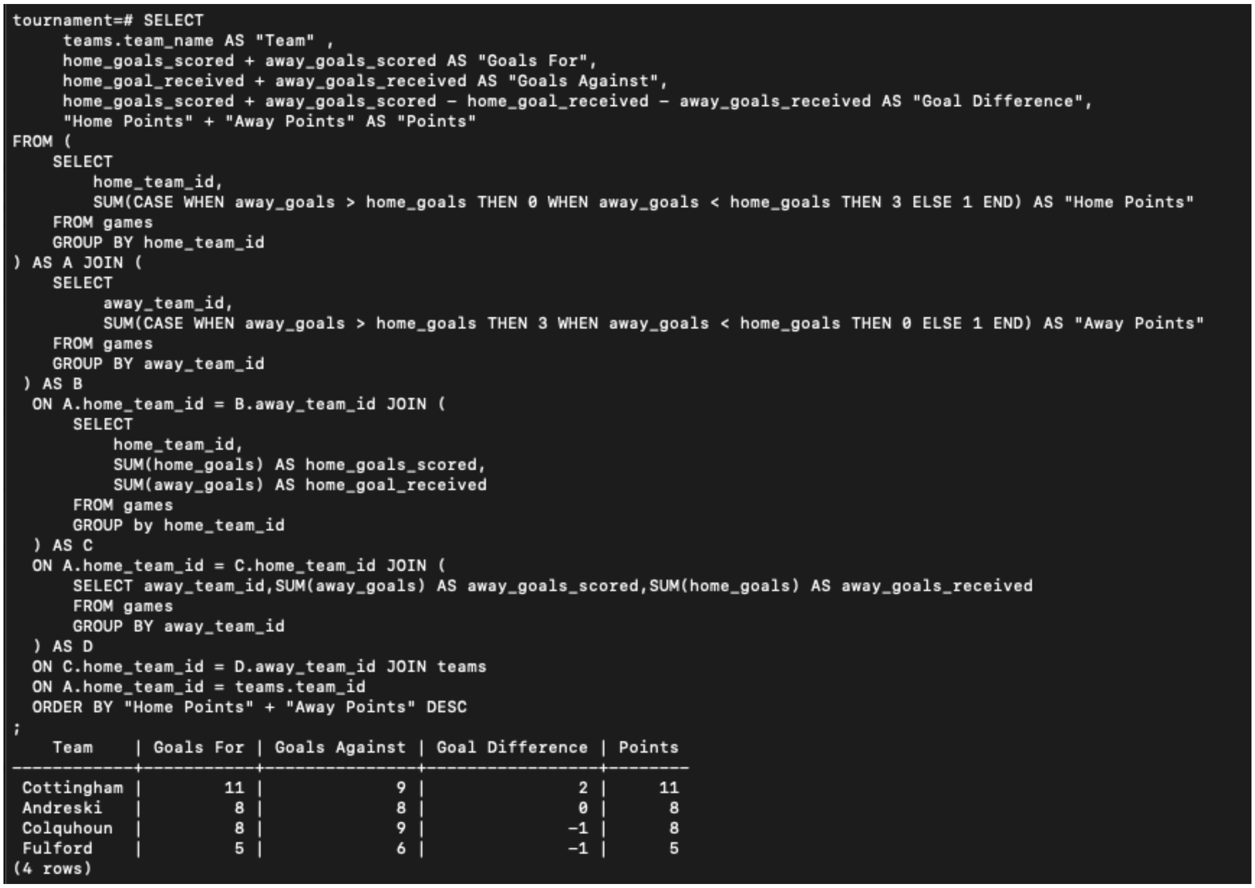
Test 5 - Return the number of goals in favour, goals against, goals difference and points by team.

1. The final test consists of obtaining the number of goals in favour, number of goals against, the goal difference, and points by team (resembling a league table). To obtain such data in one query required multiple subqueries, at least to far as we could see, and using our schema.
2. The expected outcome of our test, can yet again be extracted through the data provided, though not so simply. Calculations must be made to infer expected values. After all calculations, the table should look like (Sort by points):

Team	Goals For	Goals Against	Goal Difference	Points
Cottingham	11	9	2	11
Andreski	8	8	0	8
Colquhoun	8	9	-1	8

Fulford	5	6	-1	5

1. Below is both the query used to validate this test, and the resultant output



```
tournament=# SELECT
     teams.team_name AS "Team" ,
     home_goals_scored + away_goals_scored AS "Goals For",
     home_goal_received + away_goals_received AS "Goals Against",
     home_goals_scored + away_goals_scored - home_goal_received - away_goals_received AS "Goal Difference",
     "Home Points" + "Away Points" AS "Points"
FROM (
    SELECT
        home_team_id,
        SUM(CASE WHEN away_goals > home_goals THEN 0 WHEN away_goals < home_goals THEN 3 ELSE 1 END) AS "Home Points"
    FROM games
    GROUP BY home_team_id
) AS A JOIN (
    SELECT
         away_team_id,
         SUM(CASE WHEN away_goals > home_goals THEN 3 WHEN away_goals < home_goals THEN 0 ELSE 1 END) AS "Away Points"
    FROM games
    GROUP BY away_team_id
 ) AS B
  ON A.home_team_id = B.away_team_id JOIN (
      SELECT
          home_team_id,
          SUM(home_goals) AS home_goals_scored,
          SUM(away_goals) AS home_goal_received
      FROM games
      GROUP by home_team_id
  ) AS C
  ON A.home_team_id = C.home_team_id JOIN (
      SELECT away_team_id,SUM(away_goals) AS away_goals_scored,SUM(home_goals) AS away_goals_received
      FROM games
      GROUP BY away_team_id
  ) AS D
  ON C.home_team_id = D.away_team_id JOIN teams
  ON A.home_team_id = teams.team_id
  ORDER BY "Home Points" + "Away Points" DESC
;
    Team    | Goals For | Goals Against | Goal Difference | Points
------------+-----------+---------------+-----------------+--------
 Cottingham |        11 |             9 |               2 |     11
 Andreski   |         8 |             8 |               0 |      8
 Colquhoun  |         8 |             9 |              -1 |      8
 Fulford    |         5 |             6 |              -1 |      5
(4 rows)
```

2. Expected outcome and actual output match on every field of every record, implying success. Our query was able to extract the necessary data, without delivering any redundant data. Most fields displayed are calculated / derived data, as opposed to the primary records from the fields themselves. Data is sorted by points, much like the arrangement of a real-life league table.

Conclusion

The first task helped us because it was a way for us to visualise what we would need to do for the later tasks. It essentially enabled us to do the other tasks with ease and it gave us clarity on our project. We also had to keep in mind all the criteria needed to create an optimal data model.

The second task made us view the data logically in the form of structured textual form using the Codd – Date definition. We mapped the entities, attributes and the relationships between them. Using controlled redundancy, we didn't include redundant information such as duplication of data.

The third task allowed us to strip down each relation to its essential attributes using normalisation. Firstly, we converted our table into the first normal form, the second normal form and eventually the third normal form. We use the third normal form in order for us to do non loss decomposition meaning we will not delete any attributes and we will move them into new relations instead.

The fourth task enables us to define and manipulate existing database tables and constraints using PostgreSQL. We essentially implemented the relational model from the second task. We used various variables such as BIGINT and TIMESTAMP to better accommodate for specific data types. We maintained database integrity throughout the database. We added constraints to the primary and foreign keys as well as some other important constraints.

The fifth task supported all the previous tasks by making us test the implementation of the database schema using queries. This is useful practise as it taught us to test our implementation before making sure that it works properly. It also helped us identify bugs earlier and easier in the code. We also saved a lot of time on tracing bugs on specific parts of the code.

Overall, this assignment helped us to understand what it takes to build a database management system from scratch using various techniques for data modelling, normalising, implementing and testing the SQL code needed. It taught us some techniques used in the industry today and how to use them for a more effective and efficient implementation.