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**](https://csgitlab.reading.ac.uk/xc001654/Database)

**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.

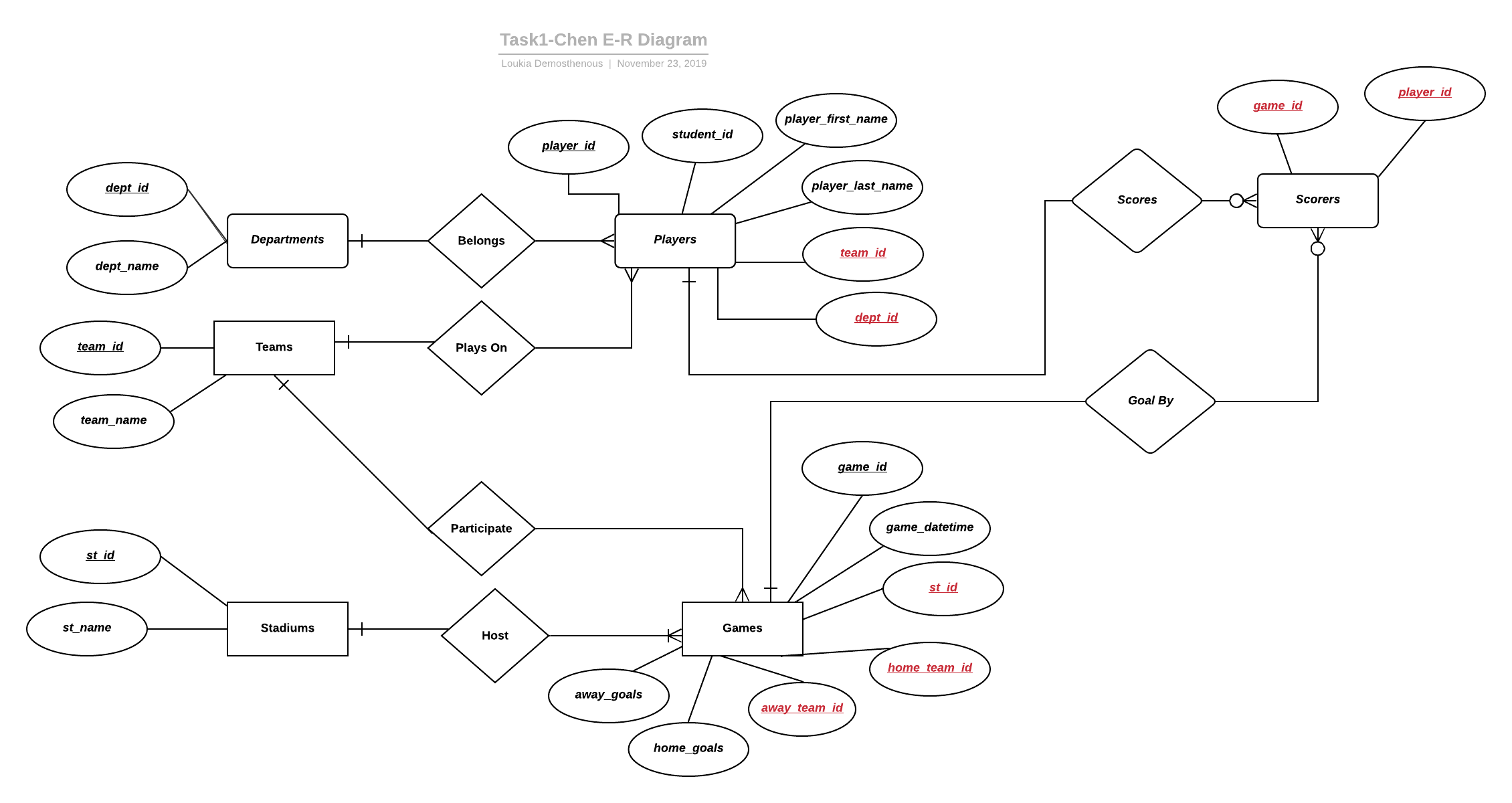
For the fifth task we tested our database implementation using test cases and queries.

**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 player\_first\_name:VARCHAR(50)

dept\_name: VARCHAR(50) player\_last\_name:VARCHAR(50)

team\_id: BIGINT game\_id:BIGINT

team\_name: VARCHAR(50) game\_datetime:TIMESTAMP

st\_id: BIGINT home\_team\_id:BIGINT

st\_name: VARCHAR(50) away\_team\_id:BIGINT

player\_id:BIGINT home\_goals:INT

student\_id:INT away\_goals:INT

relation:**Departments** relation: **Teams**

dept\_id: BIGINT team\_id:BIGINT

dept\_name: VARCHAR(50) team\_name:VARCHAR(50)

primary key dept\_id primary key team\_id

Constraint: dept\_name NOT NULL Constraint: team\_name NOT NULL

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

relation:**Stadiums** relation: **Players**

st\_id: BIGINT player\_id: BIGINT

st\_name: VARCHAR(50) student\_id:INT

primary key st\_id player\_first\_name:VARCHAR(50)

Constraint: st\_name NOT NULL 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

relation: **Scorers** Constraints: student\_id NOT NULL, player\_x\_name on

game\_id: BIGINT Unicode , team\_id NOT NULL, dept\_id NOT NULL

player\_id:BIGINT Assumptions: Every player belongs to one Department

foreign key game\_id references Games not null (not joint degrees), Every player belongs to one team

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

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 |

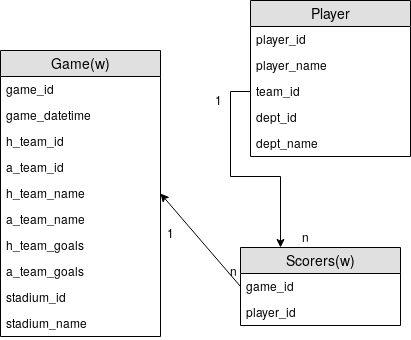
Converting our structure to 1rst Normal Form

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

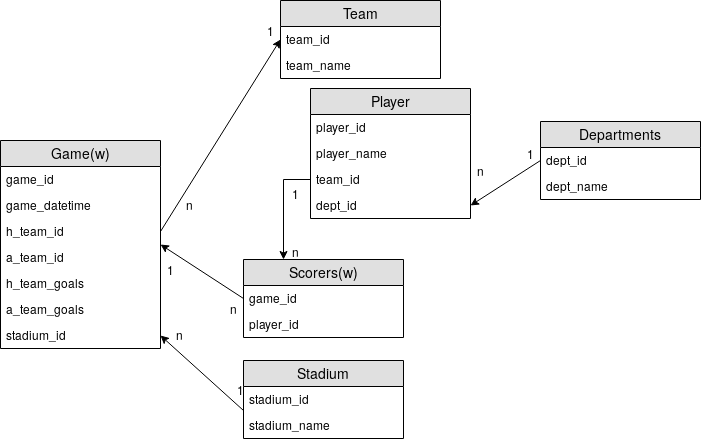
The requirements of the 2NF are ..

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:**

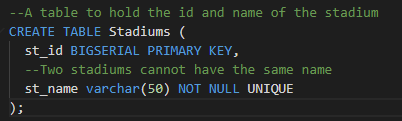
In this task, we will implement a Postgres schema. Some things need to be discussed first:

* 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

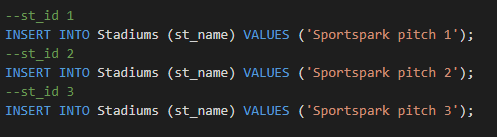
Creating the database:



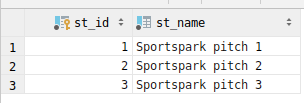
Creating the table **Stadiums:**

****

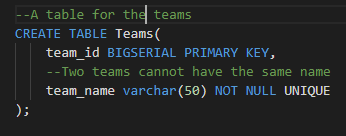
Populating the table **Stadiums:**

****

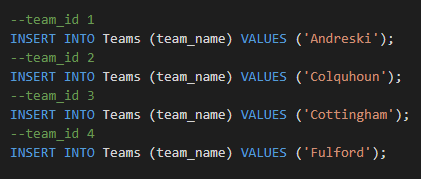
List of all the data in **Stadiums:**

****

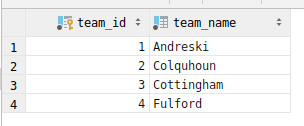
Creating table **Teams:**

****

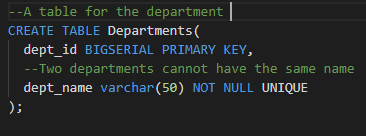
Populating the table **Teams:**

****

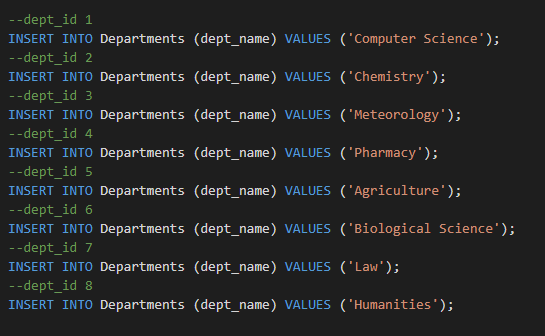
List of all the data in **Teams:**

****

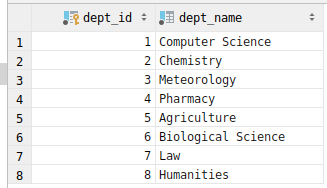
Creating table **Departments:**

****

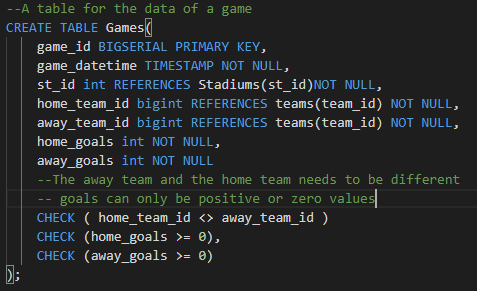
Populating the table **Departments:**

****

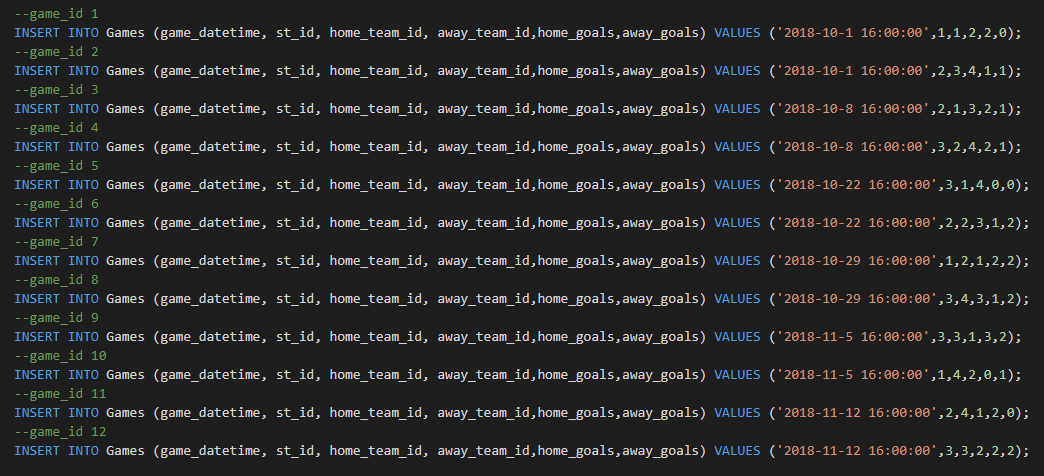
List of all the data in **Departments:**

****

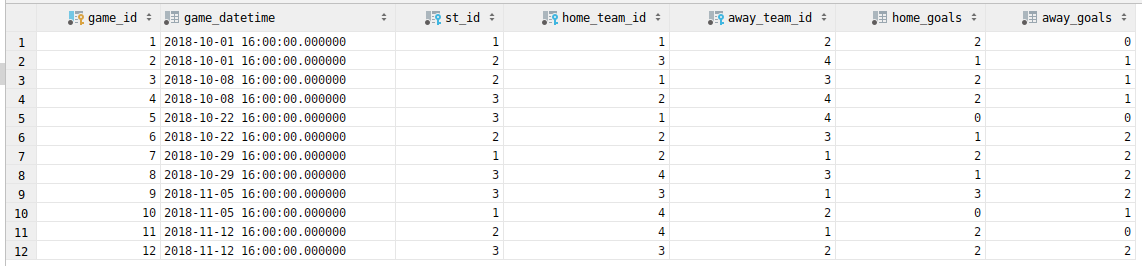
Creating table **Games:**

****

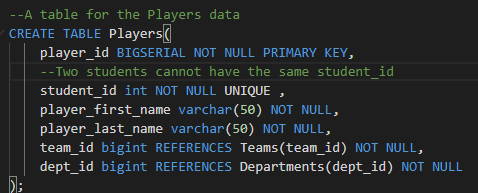
Populating table **Games:**

****

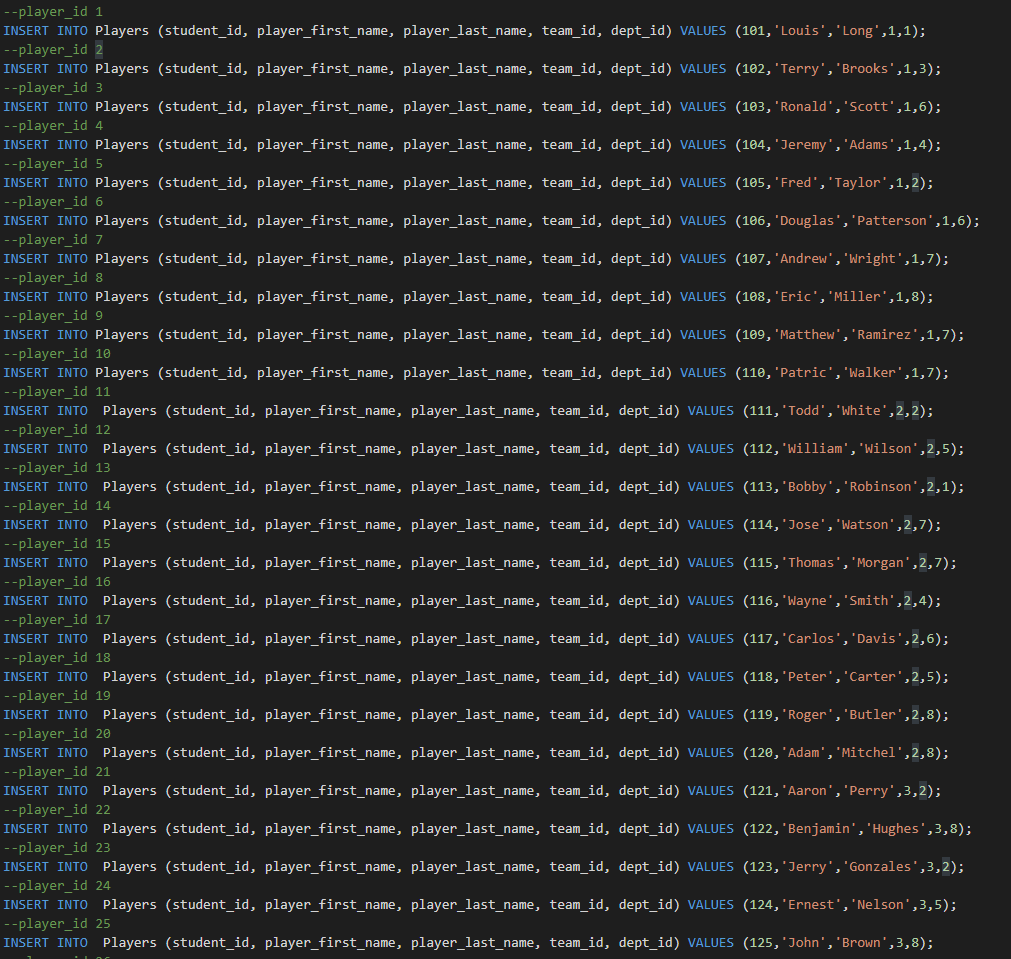
List of all the data in **Games:**

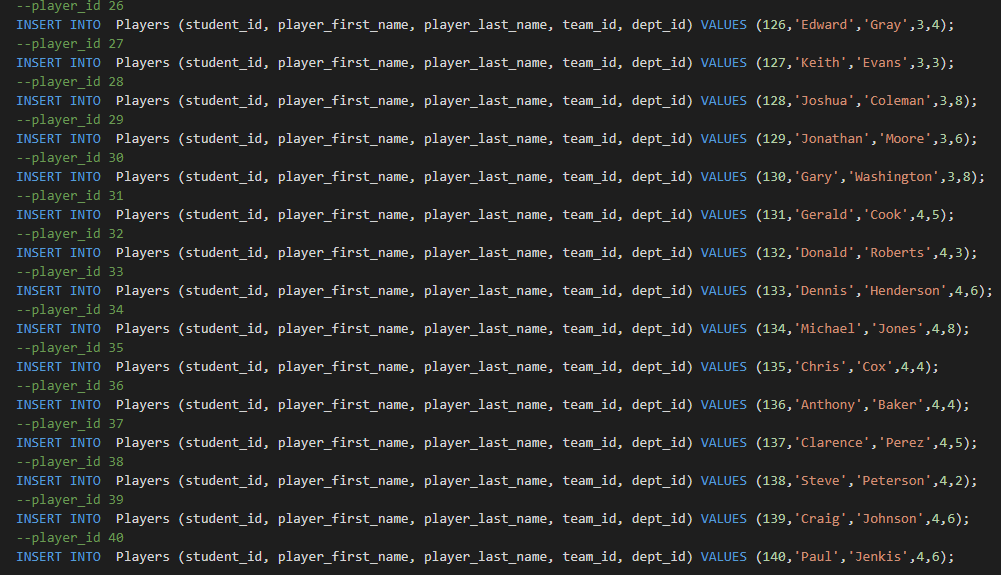
****

Creating table **Players:**

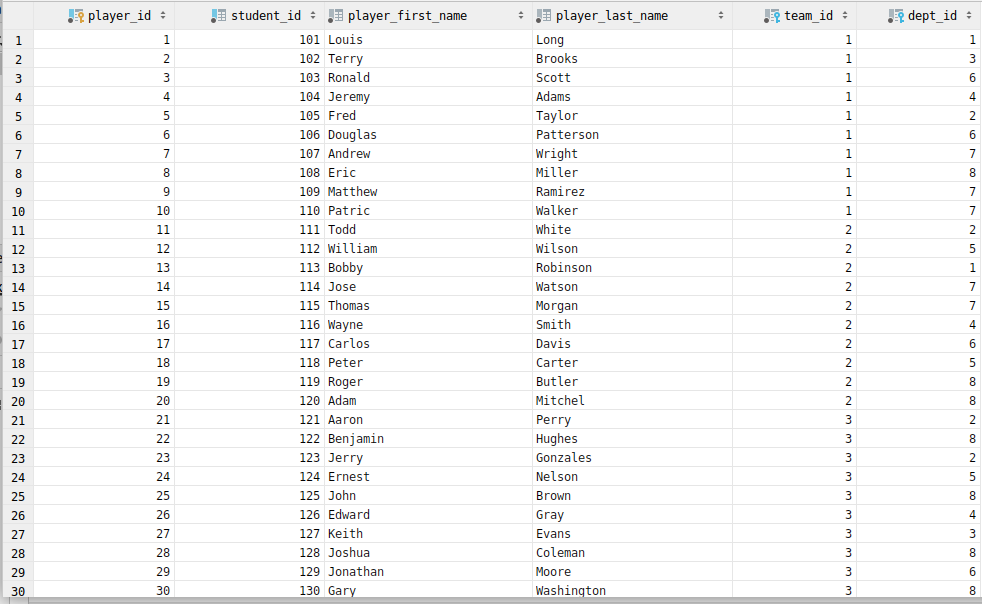
****

Populating table **Players:**

****

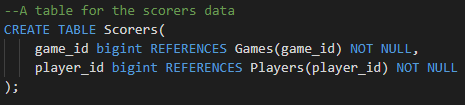
****

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

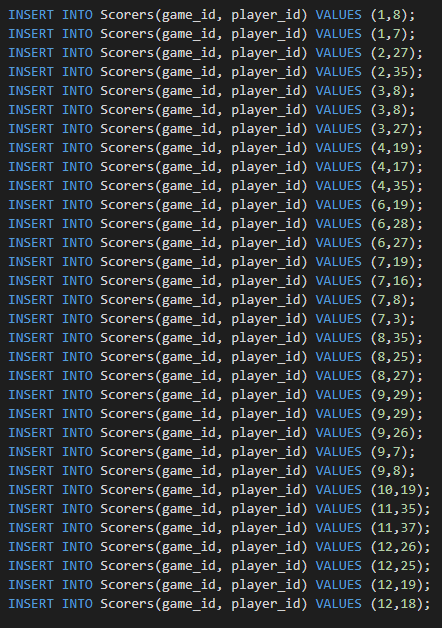
****

****

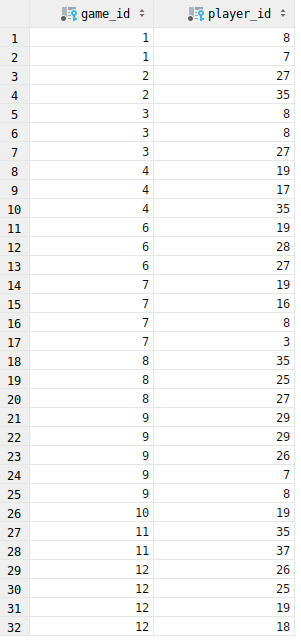
Creating table **Scorers:**

****

Populating table **Scorers:**

****

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

****

**Task 5:**

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

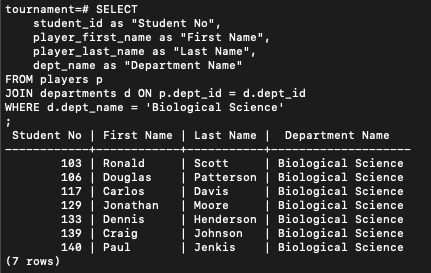
1. This test will validate that we are able to list all students who play in a particular department (e.g. Computer Science or Biological Science) through the use of join between two related tables (departments and players). We will test for the Biological Science department.

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:



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)

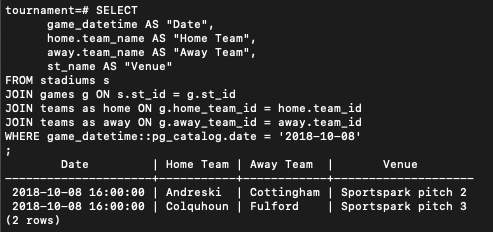
1. This test aims to ensure that we are able to list all fixtures (games / events) that are associated with a specific date. The date we decided to test through our query was the 8th October 2018 (08/10/2018).

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:



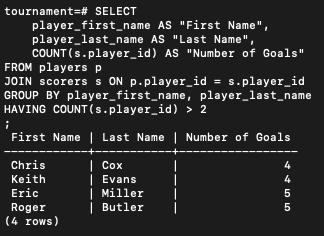
4. In this instance, the result of the query executed exactly matches the foreseen / expected outcome (should we account for the fact that Sportpark pitch 2 maps to Pitch 2 in the provided data, and a slight date\_time representation difference). This implies a pass on this test.

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 |

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



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:

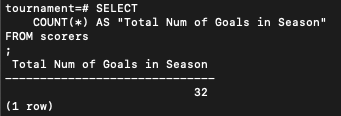
    2+2+3+3+0+3+4+3+5+1+2+4

    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



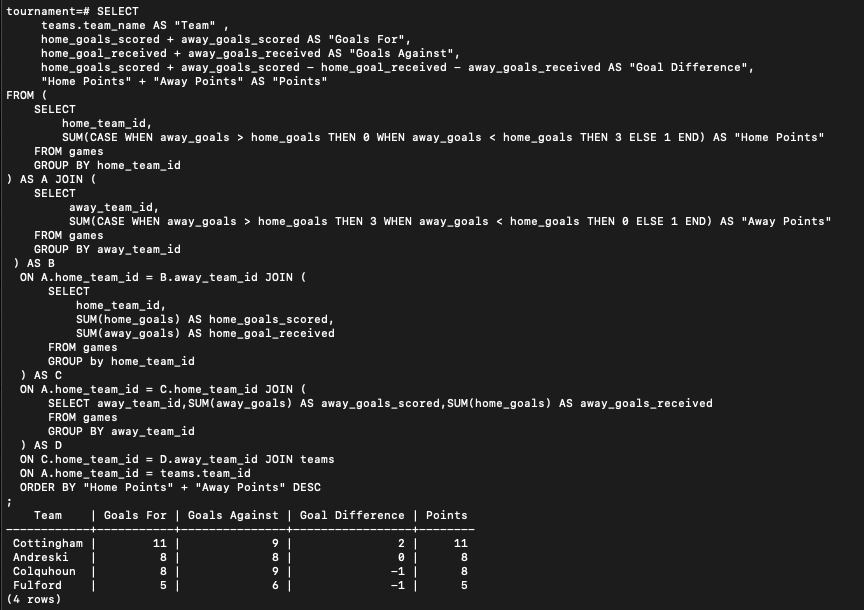
1. As we can see, the result of the query and expected output match, yet again implying test success. The result was simply extracted through counting all records in the scorers table.

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



1. 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.