Jean-Pierre Njock II
Bryan M. Socha
CS 329E//Cohen Fall 2016
GitGudDatabases

NFL PLAYERS DATABASE FINAL REPORT

Introduction

In order to learn more about databases and user interactions with them, we were tasked with finding a dataset we were interested in and designing a schema which would organize the information. After creation of the schema, we then uploaded our dataset to create our database. We then designed interaction with our database. By forming queries from the database, we could retrieve the desired information. Ultimately, this culminated in a real world scenario where we extracted Twitter posts and created questions for those, similar to how business do in the business world. This process exposed us to how databases are formed, how they are organized, and how they are utilized.

Relational Database Design (LAB 1)

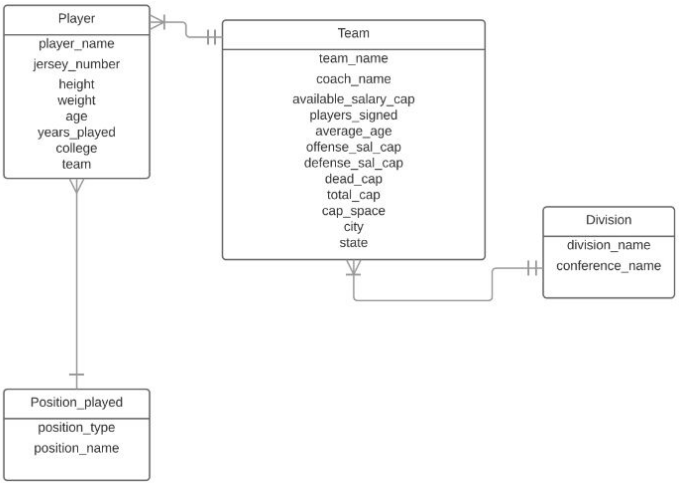
For our initial dataset, we found information for all current National Football League (NFL) players and the teams they are on from each individual team roster. This information would be separated into three main categories: players, teams, divisions. All player information, such as height and weight, would be be placed into a Player table. We then created a table for the positions played by everyone in the NFL, which were divided between offensive and defensive positions. We then found information

regarding each team and placed it into the Team table. Information regarding the team's coach, where they were located, etc. were included in this table. Lastly, we created a table for each division. This table would include the name of each division and how many championships they had won.

Aside from the Player table, each dataset used in the tables were generated ourselves, which had its advantages and disadvantages. By putting the data in manually, we were able to define the information how we saw fit; what information we wanted to include and what we wanted to leave out. This allowed for a more personal dataset, one that we could navigate better. On the other hand, this limited the number of entries overall. For instance, the NFL has 32 teams and 8 divisions. This meant that we would have a limited set of entries to work with later when we developed quieres for either of these tables.

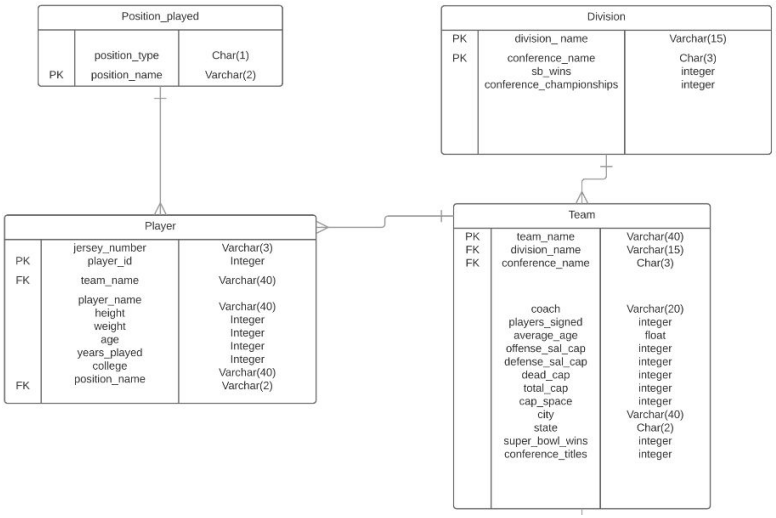
In order to define the relationship between each table, we formed a conceptual entity relationship diagram or ERD. This diagram would be the building blocks of our schema. Shown below is the fixed conceptual ERD for our database following Lab 1 in Figure 1.

Figure 1: Conceptual ERD



After we had defined the relationships between each table, we then created a logical ERD to define the minute relationships. Shown below is the logical ERD for Lab 1 in Figure 2.

Figure 2: Logical ERD



The logical ERD shows the primary key, foreign key relationships between each table and what variable type each attribute possesses. To summarize, the Division table holds primary keys of “division_name” and “conference_Name”. The Team table takes the “division_Name” and “conference_Name” as foreign keys from the Division table.

“Team_name” is the primary key for this table. The Position_Played table has “position_name” as the primary key, and both this and “team_name” from Team are foreign keys on the Player table. The Player table has an auto-incremented “player_id” as its primary key.

After creation of the schema for our project, we had to create the tables within MySQL. For each table, a create table script was run. The script for one of the tables is shown below in Figure 3. We did not encounter any problems with running the script.

Figure 3: MySQL table creation script

```
create table Division(
conference_name Char(3),
division_name varchar(15),
sb_wins integer,
conference_championships integer,
unique(division_name,conference_name),
primary key (division_name,conference_name)
);
```

Unfortunately, we had an issue with our original dataset that had to be remedied before execution of Lab 2. Within the relationship between the Team, Player and Division tables, we had incorrectly placed foreign key constraints. Within the Player table, “team_name” initially had a primary key restraint. This meant uniqueness for that attribute. Because many players could be on the same team, “team_name” could not be unique. Additionally, Division had “team_name” as a foreign key constraint, but because it was the parent table to Team, it should not have it as an attribute at all. The Team table needed the “division_name” and “conference_name” as foreign key restraints to

correctly implement the table. These changes were made and we continued on to Lab 2.

At the end of our project, we edited both of our conceptual and logical ERDs to include the Twitter API. Both updated ERDs, in Figures 4 and 5, are on the following page. The Tweet table was added with attributes such as “Tweet_id” and, most importantly, “tweet_doc”. This would be used later in the Twitter quieres. The table has a primary key of “Tweet_id” and a foreign key constraint of “team_name” from the Team table.

Figure 4: Conceptual ERD updated with Twitter API

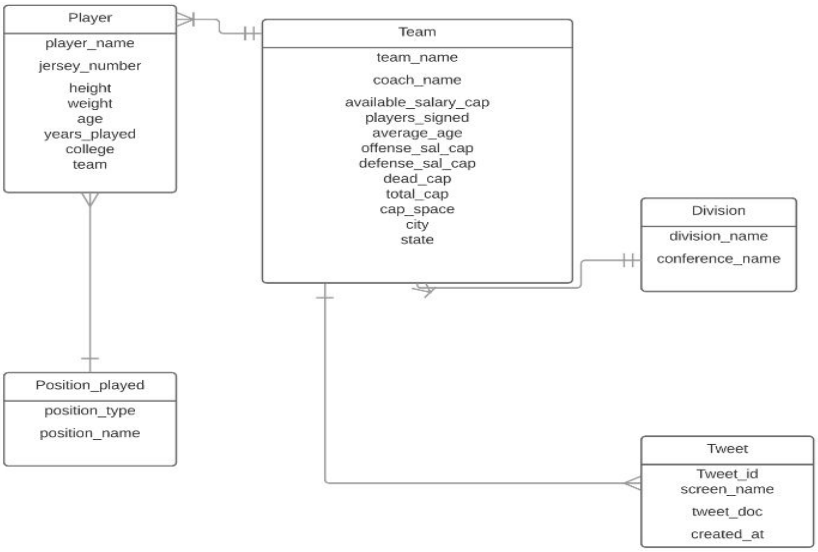
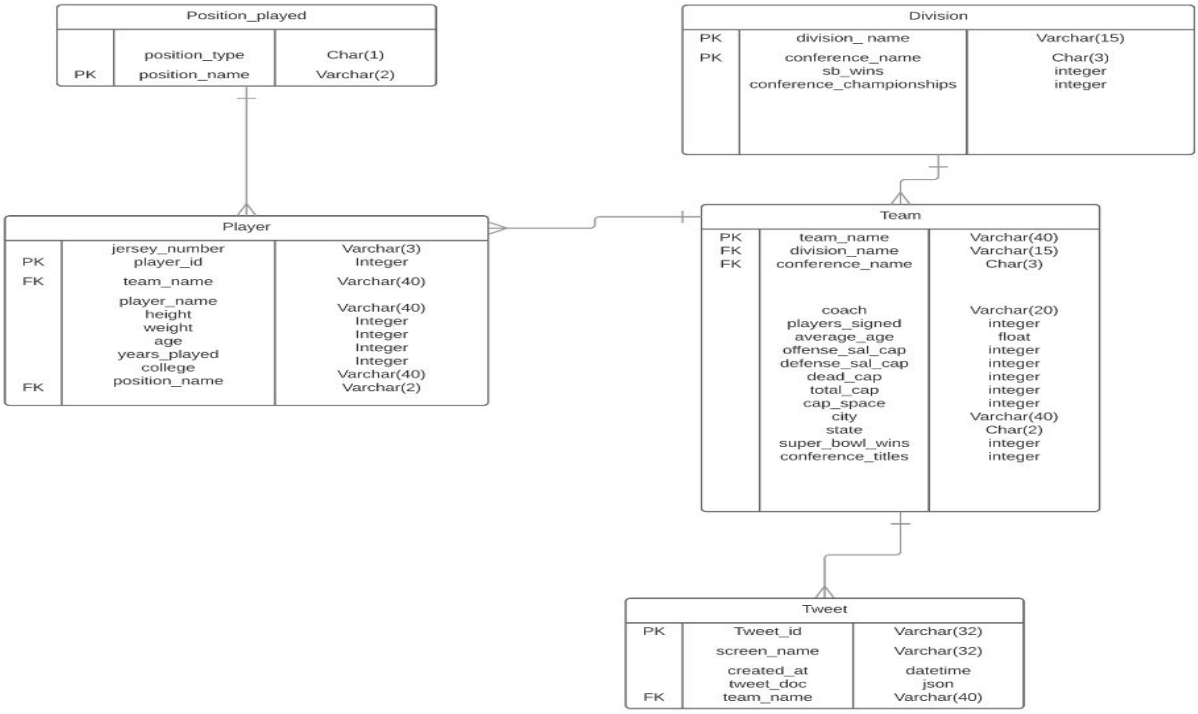


Figure 5: Logical ERD updated with Twitter API



Data Loader (LAB 2)

With the updated ERDs and tables generated in MySQL, we began work on populating our database with our datasets. In order to do this, we had to create Python 2 import functions. Initially, these scripts would connect to the MySQL server that held our information. If any problems occurred, an error would be returned. Then we would choose an insert statement that would be transmitted to the MySQL database and place information into the table. Figure 6 shows a the insert statement for entering data into the Division table.

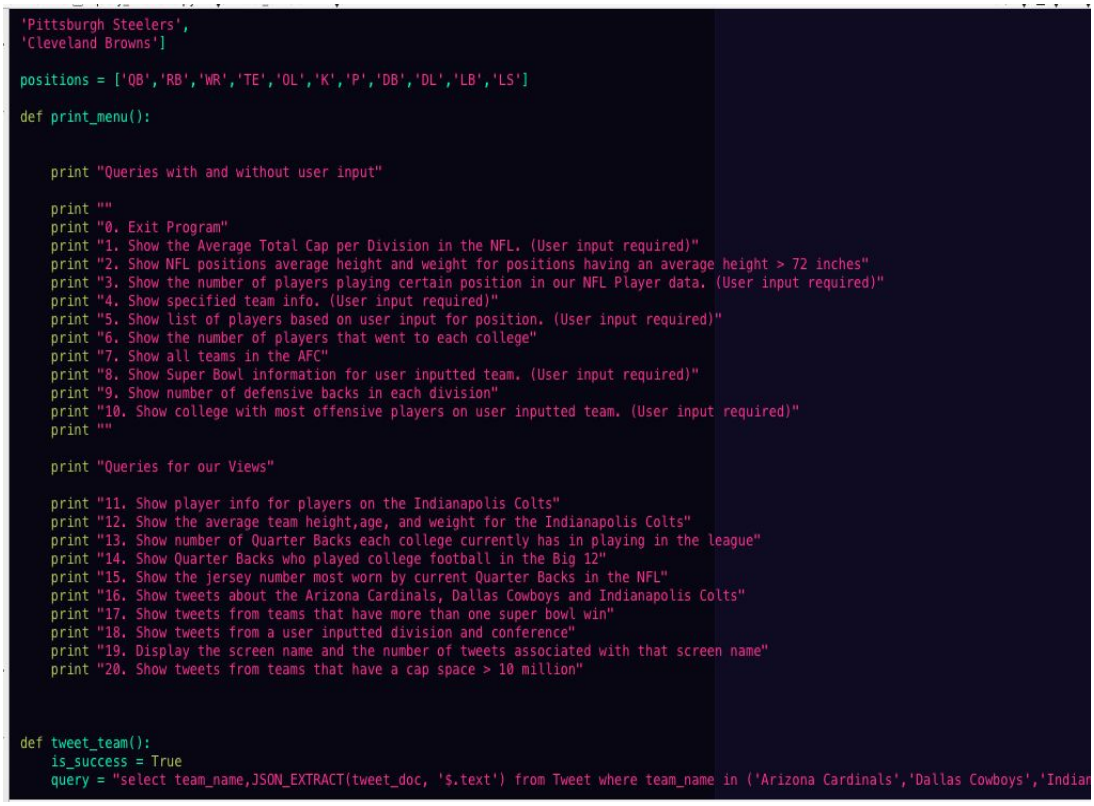
Query Interface (LAB 3)

This portion of the project had us utilize all the skills in SQL that we learned in class to create an interactive query interface. Below will be screenshots of what are query interface code looked like with brief descriptions of what's going on.

1. ***Code used to produce Query Interface (Menu):***

   a. Below is an image of code that is used to create the menu for our query interface (Figure 6). As you see there are 20 options to choose from within our query interface. Some options require user input while others do not. And if the user wants to quit the interface at anytime, he or she can do so by selecting option 0.

Figure 6: Code used to display menu options



```
'Pittsburgh Steelers',
'Cleveland Browns']

positions = ['QB','RB','WR','TE','OL','K','P','DB','DL','LB','LS']

def print_menu():


    print "Queries with and without user input"

    print ""
    print "0. Exit Program"
    print "1. Show the Average Total Cap per Division in the NFL. (User input required)"
    print "2. Show NFL positions average height and weight for positions having an average height > 72 inches"
    print "3. Show the number of players playing certain position in our NFL Player data. (User input required)"
    print "4. Show specified team info. (User input required)"
    print "5. Show list of players based on user input for position. (User input required)"
    print "6. Show the number of players that went to each college"
    print "7. Show all teams in the AFC"
    print "8. Show Super Bowl information for user inputted team. (User input required)"
    print "9. Show number of defensive backs in each division"
    print "10. Show college with most offensive players on user inputted team. (User input required)"
    print ""

    print "Queries for our Views"

    print "11. Show player info for players on the Indianapolis Colts"
    print "12. Show the average team height,age, and weight for the Indianapolis Colts"
    print "13. Show number of Quarter Backs each college currently has in playing in the league"
    print "14. Show Quarter Backs who played college football in the Big 12"
    print "15. Show the jersey number most worn by current Quarter Backs in the NFL"
    print "16. Show tweets about the Arizona Cardinals, Dallas Cowboys and Indianapolis Colts"
    print "17. Show tweets from teams that have more than one super bowl win"
    print "18. Show tweets from a user inputted division and conference"
    print "19. Display the screen name and the number of tweets associated with that screen name"
    print "20. Show tweets from teams that have a cap space > 10 million"



def tweet_team():
    is_success = True
    query = "select team_name,JSON_EXTRACT(tweet_doc, '$.text') from Tweet where team_name in ('Arizona Cardinals','Dallas Cowboys','Indian
```

2. ***Code used to produce Query functions:***

a. Here is an example piece of code used in our function portion of our query interface. Here, we're focused on building a query that displays players who played for the Indianapolis Colts. To accomplish this we first built our query of interest which can be seen in the image below (Figure 7).

Figure 7: Functional Query entry

```
is_success = True
query = "select player,position_name,age,college from colts_players order by age desc"
```

b. After building the desired query, we implement the functions used in our server script to connect to our mysql server and execute the query (Figure 8). As you see, we include formatting where the code begins with (Print “”). This allowed our output to look clean when the user used our interface. We then iterated through our query results and selected portions that we wanted; in our case we wanted to gather the first four column values for row 1 (Player Name, Position,Age, and University Attended). We also included error checks that notified the user if the query failed and what the error message displayed. If no error occurred, then a successful message would be returned.

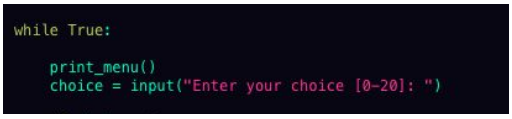
Figure 8: Query output and formatting code

```
try:
    connection = create_connection()
    cursor = connection.cursor()

    query_status = run_stmt(cursor,query)
    if query_status is False:
        is_success = False

    results = cursor.fetchall()

    print ""
    print(format('Player Name','>19s') + format('Position','>15s') + format('AGE','>9s') + format('University Attended','>26s'))
    print ""
    for row in results:
        print(format(str(row[0]),'>19s') + format(str(row[1]),'>12s') + format(str(row[2]),'>12s') + format(str(row[3]),'>26s'))

    print""
except pymysql.Error as e:
    is_success = False
    print "Colts Players failed: " + e.strerror
return is_success
```

3. Code used to create an interactive menu option. User input check included

a. Below is the code used to generate the entire query interface with interactive menu options. With regard to the screenshot below (Figure 9), the code shows how we first ask the user to select a choice. As you may remember, the user must choose from the selected options mentioned before.

Figure 9: Query Interface menu code



b. The following screenshot (Figure 10) displays how we provided output and results based on the user's selected option. In the case for this screenshot, if the user selects option 1 then we execute the following code. In this case, option 1 is one of our queries requiring user input, so we first ask them to select a division from a set of options. We then have them pick a conference. After the user inserts their inputs we error check the values to make sure the inputs are valid, which can be seen in the "while" statements below. If the inputs are not valid we ask the user to re-enter their values.

Figure 10: Query Interface code for choice 1

```
if choice==1:
    division = raw_input("Select the Division from the following list: North, South, East, West: ")
    conference = raw_input("Select conference from the following list: AFC, NFC: ")
    conference = str(conference.lower())
    division = division.lower()
    while(conference != 'afc' and conference != 'nfc'):
        conference = raw_input("Incorrect input. Select conference from the following list: AFC, NFC: ")
        conference = conference.lower()
    while(division != 'north' and division != 'south' and division != 'east' and division != 'west'):
        division = raw_input("Incorrect input. Select the Division from the following list: North, South, East, West: ")
        division = division.lower()
    print "You have chosen to show the Average Total Cap for the " + conference + " " + division + ". Here are the results:"
    avg_total_cap(division, conference)
```

c. Once the inserted values are valid, we print out a display of the results, which can be seen below. Here the user selects to display the average salary cap for the AFC North division in the NFL (Figure 11). Also below Figure 11 is the output to one of our twitter queries, which were added material used in our final project (Figure 12). In this query, we're interested in presenting the number of tweets based on the screen_name used and the results are shown below.

Figure 11: Query menu and results for option 1: Average Total cap for the AFC North

```
Queries with and without user input

0. Exit Program
1. Show the Average Total Cap per Division in the NFL. (User input required)
2. Show NFL positions average height and weight for positions having an average height > 72 inches
3. Show the number of players playing certain position in our NFL Player data. (User input required)
4. Show specified team info. (User input required)
5. Show list of players based on user input for position. (User input required)
6. Show the number of players that went to each college
7. Show all teams in the AFC
8. Show Super Bowl information for user inputted team. (User input required)
9. Show number of defensive backs in each division
10. Show college with most offensive players on user inputted team. (User input required)

Queries for our Views
11. Show player info for players on the Indianapolis Colts
12. Show the average team height,age, and weight for the Indianapolis Colts
13. Show number of Quarter Backs each college currently has in playing in the league
14. Show Quarter Backs who played college football in the Big 12
15. Show the jersey number most worn by current Quarter Backs in the NFL
16. Show tweets about the Arizona Cardinals, Dallas Cowboys and Indianapolis Colts
17. Show tweets from teams that have more than one super bowl win
18. Show tweets from a user inputted division and conference
19. Display the screen name and the number of tweets associated with that screen name
20. Show tweets from teams that have a cap space > 10 million
Enter your choice [0-20]: 1
```

Figure 12: Twitter query results for screen_name and number of tweets

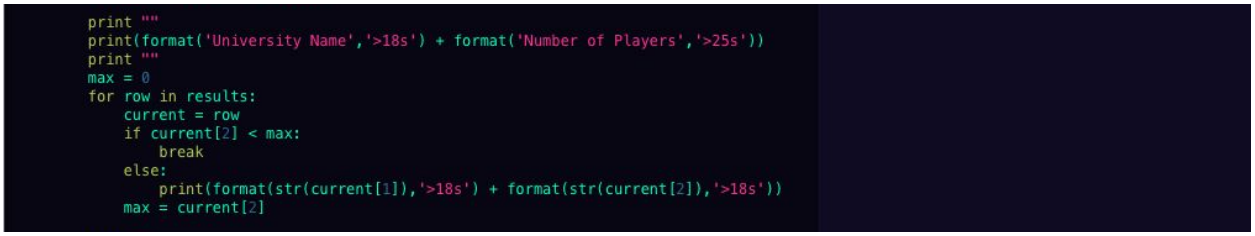
```
Enter your choice [0-20]: 19
You have chosen to display the number of tweets posted per screen name. Here are the results
        frugalmaniac    6
             USN1337    5
       PatriotsViews    4
         RavensViews    4
       BuffaloBills1v   4
       panthers_fanly   3
       RedskinsFanMVP   3
        ChargersViews   3
        TheNFLHotline   3
     SteelersNewsNow    3
        falcons_fanly   3
     PhilEaglesViews    3
         CardsFanMVP    3
           franktfox    2
       DaBearsOnFire    2
     ChopShopTheatre    2
          BrownsBots    2
     TexansSupporter    2
        VikingsViews    2
        JaguarsViews    2
       TexansFanMVP1    2
      CardinalsViews    2
     ColtsSupporters    2
          hippykid88    2
        BengalsViews    2
```

<u>Technical Challenges:</u>

Luckily throughout the duration of the lab we never encountered any serious, earth shattering technical challenges. However that's not say that we didn't encounter any challenges. One example of our technical challenges occurred in Lab 3. In order to provide a clean presentation of our data once it was queried required us to format the output accordingly. My partner and I didn't know enough to format strings accordingly, so a little use of google helped us solve this miniscule challenge. Like mentioned before, the technical challenge wasn't serious but it required a lot of patience and testing to make sure output was formatted correctly. Another technical challenged that

we faced was also associated with Lab 3. In this challenge, we wanted to display the University that had the most offensive players for a given NFL team. For example, The University of Tennessee possessed 4 offensive players for the Miami Dolphins which was the greatest amount recorded out of the list of colleges associated with that NFL team. Thankfully the solution to this challenge was to implement a couple “if” statements while we selected data from the query. Below is a code snippet of how we created our solution (Figure 13).

Figure 13: Query solution for University with max offensive players



```
print ""
print(format('University Name','>18s') + format('Number of Players','>25s'))
print ""
max = 0
for row in results:
    current = row
    if current[2] < max:
        break
    else:
        print(format(str(current[1]),'>18s') + format(str(current[2]),'>18s'))
    max = current[2]
```

Conclusion:

1. ***Lessons learned***

   a. Throughout the process of creating and implementing our database, we’ve learned a few lessons. First, good database design begins with understanding what you want to model in your database. In order to figure that out, it’s necessary to brainstorm ideas and put them to use in conceptual and logical models as we did in lab1. Your conceptual and logical models will change, so don’t expect to be perfect on the first go, it’s very uncommon. Once the conceptual and logical models are created and agreed upon, the whole process of implementing the database and

querying results become a lot easier. Second, it's important to be interested in the data you plan on using for your database. Of course in the real-world you don't have the luxury of choosing your data, but in the classroom setting for this semester it was crucial that you had interest in your data. NFL players and statistics were data that my partner and I were both interested in, so the entire process of creating the database was rather enjoyable. And third, don't be afraid of getting data that is messy or data that requires reformatting. Luckily our data was fairly clean to begin with, but in reality most data you get won't be as friendly as ours was. And most of the time your data won't be in the easily formatted CSV, but maybe JSON or XML. Thankfully we learned throughout the course how to deal with data that is formatted differently, so data gathering shouldn't be an issue in future designs and practice.