Normalized Football Database

IS426

By Jackson Nahom

Introduction:

For this project, I took it upon myself to make a normalized database of historical soccer data that can have its data analyzed. To do this I had to find a data source to get the historical data. This source came to be realized at football-Data.co.uk. This website is mostly used for analyzing soccer for the purpose of betting, it also has the historical betting odds. However, the betting odds are not what I am looking to analyze. On Football Data there are CSV s for leagues in 27 countries.

Building The Database:

To start off I looked at the CSVs in England link on football-Data.co.uk, and realized it would be simplest to web scrape all the CSVs automatically. To do this I had the unique sequence on the lines with the CSV links. For England, this turned out to be: <IMG SRC=”Excel. From there I I isolated the CSV link using the split function. From there I can use a base URL and the link I split out to use request.get to take the data in the CSV. From there I can write it to a file that I name in this format YEAR_CD.csv, where C is country and D is division. Now that I could download one country’s CSV files, the next step was to go through all the countries. This was a similar problem as the last step. I had to parse through and look for the country URL and add it to a base URL to use requests.get to the parse through that webpage using the code I tested on the England page. I hit a roadblock here because Football Data split the leagues into two formats, main leagues, and extra leagues. The way the CSV files are stored in the extra leagues than in main leagues. To get around this I found a sequence that was unique to the lines with CSV files in extra leagues that did not appear in the source page for the

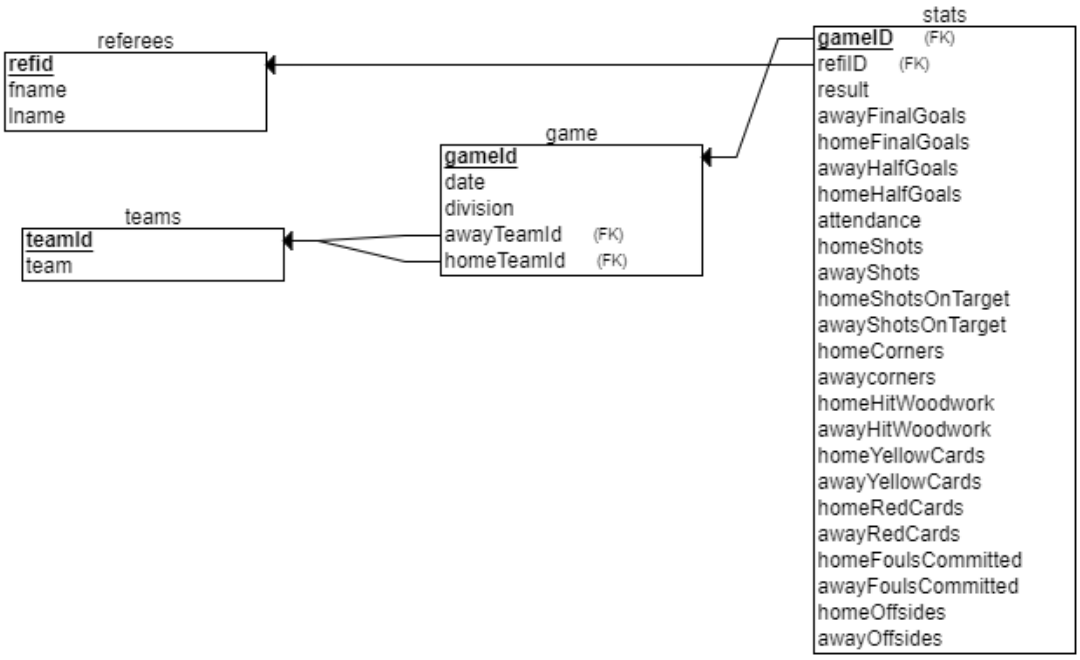
main leagues. This turned out to be: .csv”><IMG. One other thing I did to prevent errors was to replace / with _ when saving files because you can’t name files with slashes. When this script was finished all in all it downloaded 551 CSV files from 27 countries. For the code for this section see downloadCSV.py in Appendix.

Now That I had collected all the CSVs I had to upload it to a source database. At first, I had errors trying to upload CSV files one at a time. This was because the columns did not match every CSV. So I wrote a script to go through all the CSV files that I wanted to upload to go through getting all the headers and get rid of duplicates and make the source table with nothing in it. I used the glob dictionary to call glob.glob to open up all the CSV files. I needed to populate the table with a different script. To see the create table script see createSourceTable.py in Appendix.

Now that I had created the table with all the headers needed in CSVs that were to be uploaded. For the analysis’s sake of the project, I only uploaded the data from the first division of England. I wrote a function called upload that goes through a csv and puts the data in the source file. This function is very similar to the one we did in class with a few changes. I used the glob.glob function again to collect all the CSV files. The data from England's first division went back to 1993 and resulted in 9804 entries to the source table. The two scripts necessary to achieve this is Upload Function and uploadSource.py in the Appendix. A couple problems that I ran into while going through this process were that a couple CSV files returning blank headers or more rows than in the CSV. I got around these problems by checking if there is anything in the header. I also only uploaded a row if the date existed because there was always a date. After

solving that I had to fix the date format so that it was in the SQL format of YYYY-MM-DD. The way that Football Data had date was DD/MM/YYYY. To fix this I wrote a function to convert the date. This is located in the Appendix as formatDate Function.

Once I have the source table I can Normalize the data. The first step of that is creating a normalized relational schema. This led me to make this relationship schema:



Now that I have my schema I created these table. For the referee's table, none of the values can be NULL. The same goes for the teams, and game tables. However, for the stats table the only values that will never be NULL are gameID, result, awayFinalgGoals, and homeFinalGoals. The rest of the values in stats can be NULL if necessary. To populate the referees, and teams table I had to put all the names for referees and teams into a list, then I could go through and get rid of duplicates. After the

getting rid of the duplicates, I could use and an INSERT to populate the tables with referees and teams. Here is an example of the code I used to get rid of duplicates:

```
#gets rid of duplicates
for input in refList:
    if input not in refListU:
        t2 += 1
        refListU.append(input)
        print input[1] + " not duplicate"
```

The game table was fairly simple I just had retrieved the teamID for the away and home teams. The stats table was even simpler because I generated the gameID the same in both the game and stats table. This made the stats table almost a one by one upload from the source table. See script 4 in the Appendix for the how the tables were populated.

Analysis:

Now that the data has been normalized in a way that it could be used in an application I did an analysis of the data. The first SQL query I wrote was to find out how many games each team had played in England's first division. The query I used to find this was:

```
SELECT t.team, Count(result.team) as Games FROM (
    SELECT awayteamID as team, TgameID
    FROM `game`
    UNION
    SELECT hometeamID as team, TgameID
    From `game`
    GROUP BY TgameID

    ) result
JOIN `teams` t ON t.TteamID = result.team
group BY t.team
ORDER BY `Games`  ASC
```

Top 10 results

Bottom 10 results

team	Games
Chelsea	972
Everton	972
Man United	972
Arsenal	972
Liverpool	972
Tottenham	972
Newcastle	896
Aston Villa	882
West Ham	858
Man City	782

team	Games
Middlesboro	38
Blackpool	38
Barnsley	38
Swindon	42
Oldham	42
Huddersfield	52
Brighton	52
Cardiff	52
Bradford	76
Sheffield United	80

This shows that since 1993 six teams have played in 972 games, while on the other side 5 teams have only played in one season.

Since not every team has played the same amount of games, it does not make sense to look at total goals and shots instead my query looks at the average of goals and shots to compare them. To do this I had to write two queries:

```
SELECT t.team, AVG(result.goals) FROM (
    SELECT awayteamID as team, s.awayFinalGoals as goals, TgameID
    FROM `game`
    JOIN `stats` s ON s.gameID = TgameID
    UNION
    SELECT hometeamID as team, s.homeFinalGoals as goals, TgameID
    FROM `game`
    JOIN `stats` s ON s.gameID = TgameID
    GROUP BY TgameID

    ) result
JOIN `teams` t ON t.TteamID = result.team
group BY t.team
ORDER BY AVG(result.goals)  DESC
```

```
SELECT t.team, AVG(result.shots) FROM (
    SELECT awayteamID as team, s.awayShots as shots, TgameID
    FROM `game`
    JOIN `stats` s ON s.gameID = TgameID
    UNION
    SELECT hometeamID as team, s.homeShots as shots, TgameID
    FROM `game`
    JOIN `stats` s ON s.gameID = TgameID
    GROUP BY TgameID

    ) result
JOIN `teams` t ON t.TteamID = result.team
group BY t.team
ORDER BY AVG(result.shots)  DESC
```

Top 10 Average goals

team	AVG(result.goals)
Man United	1.9331
Arsenal	1.8148
Chelsea	1.7346
Liverpool	1.6975
Man City	1.6189
Tottenham	1.4866
Blackpool	1.4474
Leeds	1.3709
Newcastle	1.3594
Blackburn	1.3135

Top 10 Average shots

team	AVG(result.shots)
Liverpool	14.9914
Chelsea	14.8954
Man United	14.5688
Arsenal	14.2206
Man City	13.8470
Tottenham	13.6734
QPR	13.0877
Southampton	12.2847
Bournemouth	12.0547
Everton	11.8481

Interestingly the top 6 results have the same teams in the but the next 4 results in the top 10 do not contain any of the same team. A couple things to note are the Liverpool averages the most shots per game, however Liverpool average only the fourth most goals per game. The other thing of note is that only two teams match their position of both queries: Man City and Tottenham. While looking at my results I noticed something interesting while looking at the lowest averages for goals:

team	AVG(result.goals)
Huddersfield	0.7115
Cardiff	0.8654
Bradford	0.8947
Sheffield United	0.9250
Burnley	0.9518
Hull	0.9526
Brighton	0.9615
Barnsley	0.9737
Oldham	1.0000
Watford	1.0049

Oddly in the bottom ten only two of the five teams that only played one season appeared, Barnsley, and Oldham. I would of expected all 5 teams to be there.

The Next thing I wanted to look at was how dirty some teams have been. Once again because not every team has played the same amount of games so totals do not give a very good view. The first query I looked at was on average how many yellow cards did a team receive:

```
SELECT t.team, AVG(result.YellowCards) FROM (
    SELECT awayteamID as team, s.awayYellowCards as
YellowCards, TgameID
    FROM `game`
    JOIN `stats` s ON s.gameID = TgameID
    UNION
    SELECT hometeamID as team, s.homeYellowCards as
YellowCards, TgameID
    FROM `game`
    JOIN `stats` s ON s.gameID = TgameID
    GROUP BY TgameID

    ) result
JOIN `teams` t ON t.TteamID = result.team
group BY t.team
ORDER BY AVG(result.YellowCards)  DESC
```

Top 3 results

team	AVG(result.YellowCards)
Coventry	2.0000
Leeds	1.8816
Derby	1.8509

Looking at this result Coventry is by far the dirtiest playing team averaging 2 yellow cards a game. I gets even more compelling when looking at the average red cards per game:

team	AVG(result.YellowCards)	AVG(result.RedCards)
Middlesboro	1.3158	0.1316
QPR	1.6754	0.1316
Hull	1.7368	0.1263
Blackburn	1.8014	0.1196
Leicester	1.5500	0.1107
Coventry	2.0000	0.1053

This query is ordered by the most red cards on average. On average about every \10 games Coventry receives a red card. Looking at the other top results Blackburn is the only other team that could give Coventry a run for its money. However, based on this data Coventry appears to be on average the most cautioned team in the first division.

Now soccer is a game so there is a winner and loser, sometimes there is a tie. So I looked at who has had the most wins, loses, and ties since 1993. This took three queries because of problems with UNION.

Wins:

```
SELECT t.team, COUNT(result.wins) FROM (
    SELECT awayteamID as team, TgameID, s.result as wins
    FROM `game`
    JOIN `stats` s ON s.gameID = TgameID
    WHERE s.result = 'A'
    UNION
    SELECT hometeamID as team, TgameID, s.result as wins
    FROM `game`
    JOIN `stats` s ON s.gameID = TgameID
    WHERE s.result = 'H'
    GROUP BY TgameID
    ) result
JOIN `teams` t ON t.TteamID = result.team
group BY t.team
ORDER BY COUNT(result.wins)  DESC
```

team	COUNT(result.wins)
Man United	611
Arsenal	538
Chelsea	532
Liverpool	494

For wins I included the top four to show how far ahead Man United is from the rest of the field.

Loses:

```
SELECT t.team, COUNT(result.loses) FROM (
    SELECT awayteamID as team, TgameID, s.result as
loses
    FROM `game`
    JOIN `stats` s ON s.gameID = TgameID
    WHERE s.result = 'H'
    UNION
    SELECT hometeamID as team, TgameID, s.result as
loses
    FROM `game`
    JOIN `stats` s ON s.gameID = TgameID
    WHERE s.result = 'A'
    GROUP BY TgameID
    ) result
JOIN `teams` t ON t.TteamID = result.team
group BY t.team
ORDER BY COUNT(result.loses)  DESC
```

team	COUNT(result.loses)
West Ham	355

Ties:

```
SELECT t.team, COUNT(result.ties) FROM (
    SELECT awayteamID as team, TgameID, s.result as ties
    FROM `game`
    JOIN `stats` s ON s.gameID = TgameID
    WHERE s.result = 'D'
    UNION
    SELECT hometeamID as team, TgameID, s.result as ties
    FROM `game`
    JOIN `stats` s ON s.gameID = TgameID
    WHERE s.result = 'D'
    GROUP BY TgameID
    ) result
JOIN `teams` t ON t.TteamID = result.team
group BY t.team
ORDER BY COUNT(result.ties)  DESC
```

team	COUNT(result.ties)
Everton	283

This is just to the top team in each category. All three teams in each category have been around the English first division a long time with West Ham the only team of the three missing in any of the seasons since 1993.

Finally, the last analysis that I did was checking what referee officiated the most home wins. Though, the referee data is far from complete with multiple seasons having the value for referee NULL. The query I used for this analysis was:

```
SELECT r.Original, COUNT(*)
FROM `game`
JOIN `stats` s ON s.gameID = TgameID
Left JOIN `referee` r ON s.refID = r.TrefID
WHERE s.result = 'H' AND s.refID IS NOT NULL
GROUP BY r.TrefID
ORDER BY COUNT(*)  DESC
```

Referee with most Home wins

Original	COUNT(*)
M Dean	189

After looking at this result I realized that there is not enough context to see if there is any significance to the result. To try and get more context I ran the query for ties and losses as well.

Referee with most ties

Original	COUNT(*)
M Dean	126

Referee with most Away wins

Original	COUNT(*)
M Dean	118

After running the three queries and seeing all three have the same referee I can only concluded that M Dean has just officiated more games than everyone else.

Conclusion:

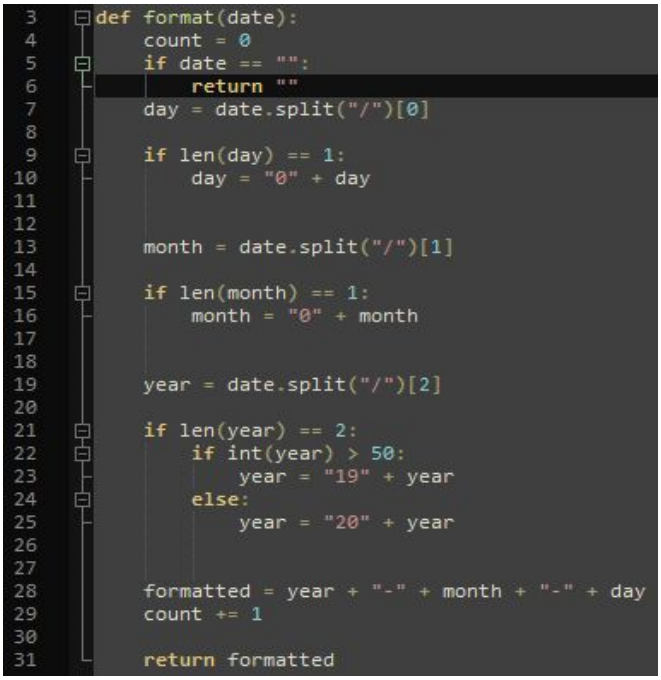
This project to a significant amount of work for me to complete. Though I believe I did the most I could given the time I was able to commit to it. It also taught me some very beneficial skills from building a database from scratch to being able web scrape hundreds of CSVs automatically from a website. Finally because of the way I normalize my database I had to write some fairly complicated SQL queries to complete my analysis. All in all I enjoyed doing this project because this is similar to work I would like to do after graduating.

I have attached a file zipped with all the code I used to complete this project.

APPENDIX:

```python
import csv
import re
import pymysql
import glob

tablename = 'soccerDataSource'

conn = pymysql.connect(host='mysql.clarksonmsda.org', port=3306, user='nahomjd', passwd='X7phenom5!', db='nahomjd_is426', autocommit=True) #setup our credentials
cur = conn.cursor(pymysql.cursors.DictCursor)

inputs = []  # etc
count = 0

for files in glob.glob('C:/Users/User/Desktop/FinalProject/backupCSVs/*E0.csv'):
    inputs.append(files)
    print files

#determine field names
fieldnames = []
for filename in inputs:
  with open(filename, "r") as f_in:
    reader = csv.reader(f_in)
    headers = next(reader)
    for header in headers:
      if header not in fieldnames:
        if len(header) > 0:
            fieldnames.append(header)
            count += 1
            print header


sql = "CREATE TABLE IF NOT EXISTS `"+tablename+"` ("
i=0
c2m = {}
m2c = {}

for field in fieldnames:
    fn = field.lower().replace(" ","_")
    fn = field.lower().replace("    ","")

    c2m[field] = fn
    m2c[fn] = field
    if i > 0:
        sql+= ",\n"
    sql += "`"+fn+"`" + " VARCHAR(255) NULL"
    i+=1
sql += ") ;"

cur.execute(sql)

conn.close()
```

createSourceTable.py



```
def format(date):
    count = 0
    if date == "":
        return ""
    day = date.split("/")[0]

    if len(day) == 1:
        day = "0" + day


    month = date.split("/")[1]

    if len(month) == 1:
        month = "0" + month


    year = date.split("/")[2]

    if len(year) == 2:
        if int(year) > 50:
            year = "19" + year
        else:
            year = "20" + year


    formatted = year + "-" + month + "-" + day
    count += 1

    return formatted
```

formatDate Function

```
import json
import urllib2
from readCSV import upload

test = 0
database = "soccerDataSource"
#http://www.football-data.co.uk/new/ARG.csv
count = 0
base = requests.get("http://www.football-data.co.uk/data.php")
#glob function
rows = base.text.split("\n")
#http://www.football-data.co.uk/mmz4281/1819/E0.csv
baseURL = 'http://www.football-data.co.uk/'
for row in rows:
    if '<tr><td>' in row:
        country = row.split("HREF=\"")[1].split("\">")[0]
        #print country

        r =requests.get("http://www.football-data.co.uk/" + country)

        lines = r.text.split("\n")
        for line in lines:
            #print line
            if '<IMG SRC="Excel' in line:
                field = line.split("HREF=\"")[1].split("\">")[0]

                link = baseURL + field
                csv_ = field.split("/", 1)[1]
                file = csv_.replace("/", "_")
                print file

                data = requests.get(link)









                #print file
                f = open(file, 'wb')
                f.write(data.text.encode('ascii', 'replace'))
                f.close

                print file + " csv created!"
                count += 1



            if '.csv"><IMG' in line:
                field = line.split("HREF=\"")[1].split("\">")[0]

                link = baseURL + field
                file = field.split("/", 1)[1]

                print file
                data = requests.get(link)

                f = open(file, 'wb')
                f.write(data.text.encode('ascii', 'replace'))
                f.close

                print file + " csv created!"
                count += 1

print count
```

downloadCSV.py

```
if game:
    hTeamID = 0
    aTeamID = 0

    for row in curall:
        sql = 'SELECT * FROM `teams` WHERE team = %s'
        #getting hometeam ID
        curteam = conn.cursor(pymysql.cursors.DictCursor)
        curteam.execute(sql,(row['hometeam']))
        for team in curteam:
            htid = team['teamID']

        sql = 'SELECT * FROM `teams` WHERE team = %s'
        #getting awayteam ID
        curteam = conn.cursor(pymysql.cursors.DictCursor)
        curteam.execute(sql,(row['awayteam']))
        for team in curteam:
            atid = team['teamID']
        curwareInsert = conn.cursor(pymysql.cursors.DictCursor)

        sql = 'INSERT INTO game (`date`,`division`,`homeTeamID`,`awayTeamID`) VALUES \
        (%s,%s,%s,%s);'
        curwareInsert.execute(sql,(row['date'],row['div'],htid, atid))
        print "intserted"
```

gameUpload

```
if ref:
    for row in curall:
        if row['referee']:
            count = row['referee'].count(' ')
            #dealing with last name first
            if ',' in row['referee']:

                refList.append([row['referee'].split(" ")[1], row['referee'].split(",")[0], row['referee']])

            else:
                #if len(row['referee'].split(" ")[0]) > 1:
                #print row['referee'].split()
                #dealing with middle intial
                if len(row['referee'].split()) == 3:
                    refList.append([row['referee'].split(" ")[0], row['referee'].split(" ")[2],row['referee']])
                elif len(row['referee'].split()) == 2:
                    refList.append([row['referee'].split(" ")[0], row['referee'].split(" ")[1],row['referee']])
                else:
                    print "error"
            t += 1

        #gets rid of duplicates
        for input in refList:
            if input not in refListU:
                t2 += 1
                refListU.append(input)
                print input[1] + " not duplicate"

    print t, t2

    #print refListU
    #insert list
    for name in refListU:
        if len(name) != 0:
            #print name
            print name[0] +" "+ name[1] + " " + name[2]
            curwareInsert = conn.cursor(pymysql.cursors.DictCursor)

            sql = 'INSERT INTO referee (`fName`,`lName`, `Original`) VALUES \
            (%s,%s,%s);'
            curwareInsert.execute(sql,(name[0],name[1],name[2]))
            print "intserted"
```

refereesUpload

```
if stats:

    for row in curall:
        rid = 0

        if row['referee'] is not None:
            sql = 'SELECT * FROM `referee` WHERE `Original` = %s'
            #print row['referee']
            curgame = conn.cursor(pymysql.cursors.DictCursor)
            curgame.execute(sql,(row['referee']))

            for ref in curgame:
                rid = ref['refID']
            curwareInsert = conn.cursor(pymysql.cursors.DictCursor)
            sql = 'INSERT INTO `stats` (`refID`,`result`,`homeFinalGoals`,`awayFinalGoals`,`homeHalfGoals`,\
            `awayHalfGoals`,`halfTimeResult`,`attendance`,`homeShots`,`awayShots`,`homeShotsOnTarget`,`awayShotsOnTarget`,\
            `homeHitWoodwork`,`awayHitWoodwork`,`homeYellowCards`,`awayYellowCards`,`homeRedCards`,`awayRedCards`,\
            `homeCorners`,`awayCorners`,`homeFoulsCommitted`,`awayFoulsCommitted`,`homeOffsides`,`awayOffsides`)\
            VALUES (%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s);'
            print rid
            curwareInsert.execute(sql,(rid,row['ftr'],row['fthg'],row['ftag'],row['hthg'],row['htag'],row['htr']\
            ,row['attendance'],row['hs'],row['as'],row['hst'],row['ast'],row['hhw'],row['ahw'],row['hy'],row['ay']\
            ,row['hr'],row['ar'],row['hc'],row['ac'],row['hf'],row['af'],row['ho'],row['ao']))
            print "intserted"
        else:
            curwareInsert = conn.cursor(pymysql.cursors.DictCursor)
            sql = 'INSERT INTO `stats` (`result`,`homeFinalGoals`,`awayFinalGoals`,`homeHalfGoals`,\
            `awayHalfGoals`,`halfTimeResult`,`attendance`,`homeShots`,`awayShots`,`homeShotsOnTarget`,`awayShotsOnTarget`,\
            `homeHitWoodwork`,`awayHitWoodwork`,`homeYellowCards`,`awayYellowCards`,`homeRedCards`,`awayRedCards`,\
            `homeCorners`,`awayCorners`,`homeFoulsCommitted`,`awayFoulsCommitted`,`homeOffsides`,`awayOffsides`)\
            VALUES (%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s);'

            curwareInsert.execute(sql,(row['ftr'],row['fthg'],row['ftag'],row['hthg'],row['htag'],row['htr'],\
            row['attendance'],row['hs'],row['as'],row['hst'],row['ast'],row['hhw'],row['ahw'],row['hy'],row['ay']\
            ,row['hr'],row['ar'],row['hc'],row['ac'],row['hf'],row['af'],row['ho'],row['ao']))
            print "intserted"

```

statsUpload

```
if team:
    for row in curall:
        if row['hometeam']:
            #print row['hometeam']
            teamList.append(row['hometeam'])
        if row['hometeam']:
            #print row['hometeam']
            teamList.append(row['awayteam'])
        #gets rid of duplicates
        for input in teamList:
            if input not in teamListU:
                t2 += 1
                teamListU.append(input)
                print input + " not duplicate"
    #upload teams
    for name in teamListU:
        if len(name) != 0:
            #print name
            print name
            curwareInsert = conn.cursor(pymysql.cursors.DictCursor)

            sql = 'INSERT INTO teams (`team`) VALUES \
            (%s);'
            curwareInsert.execute(sql,(name))
            print "intserted"

```

teamsUpload

```python
def upload(filename, tablename):
    count = 0
    conn = pymysql.connect(host='mysql.clarksonmsda.org', port=3306, user='nahomjd', passwd='X7phenom5!', db='nahomjd_is426', autocommit=True) #setup our credentials
    cur = conn.cursor(pymysql.cursors.DictCursor)
    with open(filename) as f:
        data = [{k: str(v) for k, v in row.items()}
            for row in csv.DictReader(f, skipinitialspace=True)]

    fields = data[0].keys()
    date = ""
    sql = "CREATE TABLE IF NOT EXISTS `soccerDataSource` ("

    failure = 0
    i=0
    c2m = {}
    m2c = {}
    for field in fields:
        if field == "":
            failure += 1
        else:
            fn = field.lower().replace(" ","_")
            fn = field.lower().replace("    ","")

            c2m[field] = fn
            m2c[fn] = field
            if i > 0:
                sql+= ",\n"
            sql += "`"+fn+"`" + " VARCHAR(255) NULL"
            i+=1
    sql += ") ;"
    cur.execute(sql)

    f = open("mytable.sql",'w')
    f.write(sql)
    f.close()

    fl = '`,`'.join(m2c.keys())
    fl = '`' + fl + '`'
    tl = ''
    tl = str('%s,' * len(m2c.keys()))[0:-1]
    sql = 'INSERT INTO `'+tablename+'` ('+fl+') \
    VALUES ('+tl+');'

    for row in data:
        data_ok = False
        fdl = []
        for fn in m2c.keys():
            if fn == 'date':
                date = format(row[m2c[fn]])
                if len(date) > 0:
                    data_ok = True
                    fdl.append(date)
                    count += 1
            else:
                fdl.append(row[m2c[fn]])
        #print sql
        #print fdl
        if data_ok:
            #print sql
            cur.execute(sql,fdl)
    print count
    cur.close()
    conn.close()
    #print "failed columns" + str(failure)
```

Upload Function

```python
import requests
import csv
import json
import urllib2
import glob
from readCSV import upload
import pymysql

conn = pymysql.connect(host='mysql.clarksonmsda.org', port=3306, user='nahomjd', passwd='X7phenom5!', db='nahomjd_is426', autocommit=True) #setup our credentials
curall = conn.cursor(pymysql.cursors.DictCursor)

database = "soccerDataSource"
trun = conn.cursor(pymysql.cursors.DictCursor)
sql = 'TRUNCATE TABLE `soccerDataSource`;'
trun.execute(sql)

for files in glob.glob('C:/Users/User/Desktop/FinalProject/backupCSVs/*_E0.csv'):

    field = files.split('\\')[1]
    print field
    upload(files, database)
    print "uploaded"
```

uploadSource.py