

---

# **StatAP**

Custom statistics for the NFL

# Contents

<b>1</b>	<b>Introduction</b>	<b>4</b>
1.1	Abstract . . . . .	4
1.2	Problem Definition . . . . .	4
1.3	Application Overview . . . . .	4
<b>2</b>	<b>Application Requirements Specifications</b>	<b>6</b>
2.1	Homepage . . . . .	6
2.2	Players View . . . . .	6
2.3	Teams View . . . . .	6
<b>3</b>	<b>Database Requirements</b>	<b>7</b>
3.1	Entities . . . . .	7
3.1.1	Players . . . . .	7
3.1.2	Teams . . . . .	7
3.1.3	Games . . . . .	7
3.1.4	PlayersStats . . . . .	7
3.1.5	TeamsStats . . . . .	7
3.2	Relations . . . . .	7
3.2.1	PlayersYears . . . . .	7
3.2.2	PlayersTeams . . . . .	7
3.2.3	PlayersGames . . . . .	8
3.2.4	TeamsYears . . . . .	8
<b>4</b>	<b>Integrity Constraints</b>	<b>9</b>
4.0.5	Breakdown . . . . .	9
<b>5</b>	<b>Queries</b>	<b>10</b>
5.1	5 Main Queries . . . . .	10
5.1.1	Other Queries . . . . .	13
<b>6</b>	<b>Entity Relation Model</b>	<b>14</b>
<b>7</b>	<b>Relational Model</b>	<b>16</b>
<b>8</b>	<b>Database Implementation</b>	<b>18</b>
8.1	DBMS . . . . .	18
8.2	Obtaining Data . . . . .	18
8.3	SQL . . . . .	18
<b>9</b>	<b>Relational Database Design</b>	<b>20</b>
9.1	Description . . . . .	20
<b>10</b>	<b>Application Implementation</b>	<b>22</b>
10.1	Overview . . . . .	22
10.2	Installation Manual . . . . .	22

---

<b>11 Appendix</b>	<b>23</b>
11.1 User Manual . . . . .	23
11.2 Programmers Manual . . . . .	30
11.3 Player Statistics . . . . .	31
11.4 Team Statistics . . . . .	32

# 1 Introduction

## 1.1 Abstract

The National Football League (NFL) is one of the most popular and profitable professional sports leagues in the world. The NFL's popularity has led to a surge in the collection of game data and analysis of this data. Football, due to its innate complexity, comprises of hundreds of variables for which data can be collected. These variables include offensive yards, touchdowns scored, passes intercepted, and much more. Furthermore, this data can be associated with individual players, coaches, and the team as whole which creates new ways in which the data can be interpreted. The implications of these statistics have created an avenue for computer applications (largely relying on the internet) where users can play Fantasy Football, predict and bet on games, view games live, and just plainly view the data itself. Furthermore, this data has led to the creation of new statistical measures to characterize players like the Quarterback Rating (QBR) which evaluates the proficiency of a Quarterback, the player that executes the offensive plays.

## 1.2 Problem Definition

The NFL provides access to most of the data it collects online for free. Various 3rd party companies also provide this data, and provide access to special data not easily made available by the NFL. While this data along with statistical measures are made available to the public, there is no major platform available on which users can extract custom statistics. Often on TV, a sports commentator will provide an elaborate statistic (e.g. player x has won the last 5 games against team y when the team of player x has 300 or more offensive yards) which will give more insight into the game, and interests fans in general. Currently, most fans rely on commentators to provide them with such insights as no major platform exists where they can resolve custom statistics. StatAP aims to give the public the ability to analyze NFL data by filling in various parameters themselves, and thereby giving them the ability to perform custom statistical analysis. In this way users can form their own conclusions based upon the information they extract from the StatAP application.

## 1.3 Application Overview

StatAP will consist of a newly created database which will consist of a subset of the data provided by the NFL as it is unrealistic to include everything. The vision is to build a tool where users can generate custom statistics which they may find insightful. The application will center around querying the database to find useful and relevant statistics of which there are too many to list on any single page. The users will have the ability to run thousands of different queries because they will be able to enter a range of parameters to modify the results.

The NFL data used by StatAP was acquired by crawling an online sports reference website<sup>1</sup>. Data crawled and utilized in the application includes player and team statistics for games and whole years. These statistics included things like touchdowns, first downs, passing yards, rushing yards, and other notable football statistics. The user is given access to all the data stored in the application, and has the ability extract data in various ways to gain knowledge and insight

---

<sup>1</sup><http://www.pro-football-reference.com/>

about the player or team at hand.

The application can be expanded in many ways. The most obvious is just to create more and more ways to query the database. Live data extraction can also be incorporated into the application. Furthermore, a model that can potentially predict games can be created and improved using our database by the principles of AI.

---

## 2 Application Requirements Specifications

### 2.1 Homepage

The home page will mainly be used to navigate to either the Players or Teams section of the website which will be the two main sections. The home page may also display some interesting statistics, and explain how these statistics were arrived at to give users an idea on how to use the application. However, the home page itself will not have much other functionality.

### 2.2 Players View

The players page will show a list of every player, sorted by some default setting, such as alphabetical. On the top, there will be a number of ways to filter. There will be a select box and a search box, allowing you to do a text search, which would primarily be used to search by player name. There would be other filters, such as selecting only players on a certain team, or selecting only quarterbacks who have thrown for more than 3000 yards in 2011. The list would show the results in a given amount at a time(defaulting to 30), and then paginate the result

1. Positions(QB, RB, etc...)
2. Search by player name
3. Filter by statistics
4. Filter by team

### 2.3 Teams View

B) The teams page will initially start out as a view of all of the teams sorted alphabetically with a search and filter bar at the top. The search bar will be located in the top right hand side of the page allowing the user to type in a specific team name or city to quickly access team statistics they are looking for. Some examples of the filters that could be entered include: only showing teams with over 60 wins in the last 10 years, teams that have quarterbacks with a rating above 100, etc.

1. Conference, division
2. Filter by statistics
3. Search by team name

### 2.4 Search view

This is the most important view of the application. The page will reflect forms based upon queries defined by the designers. Users will be able to fill in parameters such as names and yardage to extract interesting results for themselves. There will be multiple search pages for different queries. These pages will be oriented towards players and teams.

---

## 3 Database Requirements

### 3.1 Entities

#### 3.1.1 Players

The players table will contain basic information about a player. This will include the id of the player, their name, the position they play and a url to the statistics associated with the player. This table will be used as the entrance point for information about a player.

#### 3.1.2 Teams

The teams entity is used to identify a team by their id or name and retrieve information about that team. The city the team is from, the team's short name, and whether the team is still active will all be contained in this table.

#### 3.1.3 Games

The games entity is for connecting a certain game to the team statistics associated with that game. The games entity will have the id of the home and away teams for reference to the team, as well as the id for the statistics for the home and away teams. It will also contain the date of the game and the results. Since this entity will have an attribute for the home team and away team, how much home-field advantage affects each team can be determined.

#### 3.1.4 PlayersStats

This table contains all statistical data for all players and will be used as the source of data.

#### 3.1.5 TeamsStats

This entity contains all of the statistical data for a team and will be used as the source for team data.

### 3.2 Relations

#### 3.2.1 PlayersYears

This entity will associate a player with a certain year and contain the statistics for that player in this year. This table will be used to retrieve any statistic specific to a player for any given year.

#### 3.2.2 PlayersTeams

The PlayersTeams table will relate a player to the team he played on during a certain time period. This entity will contain an id for the player and the team that can be used to find statistics about a team during a certain player's time with that team.

---

### 3.2.3 **PlayersGames**

The PlayersGames entity connects many other tables. It contains a player id, game id, player statistics id, and team id. This table can be used to find the statistical information of a player or team for a certain game.

### 3.2.4 **TeamsYears**

This entity will be used to get the statistics for a team or player in a given year. It contains the id of the team and the id to the statistics of the team and player.



## 4 Integrity Constraints

### 4.0.5 Breakdown

The application does not allow the user to modify the database itself. The main purpose of the application is for the user to extract meaningful subset of the data that is available. The user will be able to run complex queries to generate customized statistics which will not result in any change in the database. The user is restricted from entering data that conforms to what the application expects by using validations. For example for a statistics filter, the user will be limited from entering 1 to 1000 only, due to the fact that this encompasses all possible values for a stat and prevents the user from using malicious input. Another example is that the position of the player is simply selected from a dropdown and cannot be typed by the user.

Since all of the data is crawled off the web, the application designers made sure to collect data in a consistent, accurate, and organized manner. This allows the application to minimally enforce integrity constraints since the application is solely focused on extract data. Time and energy was put in to giving the user meaningful

This is why integrity constraints will not be a huge factor. The integrity constraints that may come into play are listed below:

1. The data utilized by StatAP is crawled from the web which means almost all attributes have NOT NULL restraints.
  2. `player_id`, `team_id`, `game_id`, `players_year_id`, `players_team_id`, `players_game_id`, `team_year_id`, `player_stats_id`, and `team_stats_id` are the primary keys of our relations therefore they are unique.
  3. `game_results` can only be 0, 1, or 2. 0 being the home team, 1 being the away team, and 2 indicating a tie between the teams.
-

## 5 Queries

### 5.1 5 Main Queries

#### English

Find the average number of yards a player has while playing in a game for a team between the years 2008 and 2010.

#### SQL

```
SELECT AVG(yardage) FROM Games G, Players P, Teams T, PlayerYear
py, PlayerGame pg, PlayerStats ps WHERE P.name = "ChrisJohnson
" AND P.pid = py.pid AND py.year >= 2008 AND py.year <= 2010
AND T.team_name = "Titans" AND p.pid = pg.pid AND pg.
player_stats_id = ps.player_stats_id
yardage can equal rushing_yards, passing_yards, receiving
SELECT t
.name, t.id FROM Players p, Team t, PlayerYear py, PlayerGame
pg, PlayerStats ps WHERE p.pid = py.pid AND p.position = "
input_position" AND py.year = "input_year" AND p.pid = pg.pid
AND pg.player_stat_id = ps.player_stat_id AND ps.yards >=
numyards AND pg.tid = t.tid yards, punt_return_yards, or
kick_return_yards
```

#### RA

$Players\_Game\_y \leftarrow (\sigma_{year=2008 \vee year=2009 \vee year=2010}((\sigma_{team\_name=input\_team}((\sigma_{player\_name=input\_player}(Game \bowtie PlayersGame \bowtie Player))) \bowtie Team))))$   
 $Players\_Game\_y_{average}(\sigma(yardage))$   
 Yardage can equal rushing\_yards, passing\_yards, receiving\_yards, punt\_return\_yards, or kick\_return\_yards.

#### TRC

$\{Average(t^{(1)} | (\exists g)(Games(g) \wedge (\exists p)(Player(p) \wedge (\exists T)(Team(T) \wedge (\exists py)(PlayerYear(py) \wedge (\exists pg)(PlayerGame(pg) \wedge (\exists ps)(PlayerStats(ps) \wedge [1] = ps.rushing\_yards \wedge p.name = "input\_player" \wedge p.pid = py.pid \wedge (py.year = "2008" \vee py.year = "2009" \vee py.year = "2010")) \wedge p.pid = pg.pid \wedge pg.tid = T.tid \wedge T.team\_name = "input\_team" \wedge pg.player\_stats\_id = ps.player\_stats\_id))))))\}$

#### English

Find the teams in any specific year that had a player get greater than or equal to a certain number of yards.

#### SQL

```
SELECT t.name, t.id FROM Players p, Team t, PlayerYear py,
       PlayerGame pg, PlayerStats ps WHERE p.pid = py.pid AND p.
       position = "input_position" AND py.year = "input_year" AND p.
       pid = pg.pid AND pg.player_stat_id = ps.player_stat_id AND ps.
       yards >= numyards AND pg.tid = t.tid
```

**RA**

$PositionYear \leftarrow \sigma_{year=input\_year}((\sigma_{position=input\_position} Player) \bowtie PlayersYear)$   
 $Yards \leftarrow \sigma_{passing\_yards \geq numyards}((PositionYear \bowtie PlayersGame) \bowtie PlayerStats)$   
 $\Pi_{team\_name}(Yards \bowtie Team)$

**TRC**

$\{t^{(2)} | (\exists p)(Player(p) \wedge (\exists T)(Team(T) \wedge (\exists py)(PlayerYear(py) \wedge (\exists pg)(PlayerGame(pg) \wedge$   
 $(\exists ps)(PlayerStats(ps) \wedge t[1] = T.name \wedge t[2] = T.tid \wedge$   
 $p.pid = py.pid \wedge p.position = "input\_position" \wedge py.year = "input\_year" \wedge p.pid = pg.pid \wedge$   
 $pg.player\_stat\_id = ps.player\_stat\_id \wedge ps.yards \geq numyards \wedge pg.tid = T.tid))))))\}$

**English**

Find the city that has the most number of away team wins in a given year.

**SQL**

```
SELECT t.city FROM Teams t, Games g WHERE MAX((SELECT COUNT(g.
       result) FROM Teams t, Games g WHERE t.tid = g.away_team_id AND
       g.year = "input_year" AND g.result = 1))
```

**RA**

$AwayGames \leftarrow \sigma_{year=input\_year}((Team \bowtie_{team.id=home\_team.id} Game) \bowtie TeamYear)$   
 $\Pi_{city}(AwayGames \bowtie TeamStats)_{max(\sigma(losses))}$

**TRC**

$\{Max(t^{(1)}) | (\exists T)(Team(T) \wedge (\exists g)(Game(g) \wedge (\exists ty)(TeamYear(ty) \wedge (\exists ts)(TeamStats(ts) \wedge$   
 $t[1] = ts.losses \wedge$   
 $T.tid = G.home\_team\_id \wedge T.tid = ty.tid \wedge ty.year = "input\_year" \wedge ty.team\_stats\_id =$   
 $ts.team\_stats\_id))))\}$   
 $\{Max(u^{(1)}) | (\exists T)(Team(T) \wedge t[1] = T.city \wedge (\exists g)(Game(g) \wedge (\exists ty)(TeamYear(ty) \wedge (\exists ts)(TeamStats(ts) \wedge$   
 $T.tid = G.home\_team\_id \wedge T.tid = ty.tid \wedge ty.year = "input\_year" \wedge ty.team\_stats\_id =$   
 $ts.team\_stats\_id \wedge ts.losses = t[1]))))\}$

**English**

Find the team with the most offensive yards in a given year.

**SQL**

```
SELECT t.id, t.name MAX(ts.offensive_total_yards) FROM Teams t,
       team_stats ts, TeamYear ty WHERE t.tid = ty.tid AND ty.
       team_stats_id = ts.id
```

**RA**

$$\begin{aligned}
YearTeam &\leftarrow (\sigma_{year}(Team \bowtie TeamYear)) \\
TotalYards &\leftarrow (YearTeam \bowtie TeamStats)_{sum(offensive_yards)} \\
\Pi_{team\_name}(TotalYards_{max(\sigma(offensive_yards))})
\end{aligned}$$
**TRC**

$$\begin{aligned}
&\{Max(t^{(1)}) | (\exists ts)(TeamStats(ts) \wedge t[1] = ts.offensive_yards \wedge (\exists T)(Team(T) \wedge (\exists ty)(TeamYear(ty) \wedge \\
&T.tid = ty.tid \wedge ty.team\_stats\_id = ts.team\_stats\_id)))\} \\
&\{u^{(2)} | (\exists T)(Team(T) \wedge u[1] = T.name \wedge u[2] = T.tid \wedge (\exists ty)(TeamYear(ty) \wedge (\exists ts)(TeamStats(ts) \wedge \\
&T.tid = ty.tid \wedge ty.team\_stats\_id = ts.team\_stats\_id \wedge ts.offensive_yards = t[1])))\}
\end{aligned}$$
**English**

Find a teams average number of a specific type of yards for a given position in a given year.

**SQL**

```

SELECT AVG(yards) FROM Players p, PlayerStats ps, PlayersTeams
pt, Teams t, PlayersYears py WHERE t.name = "input_team" AND
t.id = pt.id AND pt.player_id = p.id AND py.player_id = p.id
AND ps.id = py.player_stats_id AND py.year = "input_year"
yards can equal rushing_yards, passing_yards, receiving_yards,
punt_return_yards, or kick_return_yards

```

**RA**

$$\begin{aligned}
PlayerPosition &\leftarrow (\sigma_{year=input\_year}(\sigma_{position=input\_position}(Player)) \bowtie PlayerYear) \\
PositionStats &\leftarrow (PlayerPosition \bowtie PlayersGame \bowtie PlayerStats) \\
\Pi_{yardage}((PositionStats \bowtie Team)_{average(\sigma(yardage))}) \\
Yardage &\text{ can equal } rushing\_yards, passing\_yards, receiving\_yards, punt\_return\_yards, \text{ or } \\
&kick\_return\_yards.
\end{aligned}$$
**TRC**

$$\begin{aligned}
&\{Average(t^{(1)}) | (\exists ps)(PlayerStats(ps) \wedge t[1] = ps.yardage \wedge (\exists p)(Player(p) \wedge (\exists py)(PlayerYear(py) \wedge \\
&p.position = "input\_position" \wedge p.pid = py.pid \wedge py.year = "input\_year" \wedge (\exists T)(Team(T) \wedge \\
&(\exists pg)(PlayerGame(pg) \wedge \\
&p.pid = pg.pid \wedge pg.tid = T.tid \wedge T.name = "input\_name" \wedge pg.player\_stats\_id = ps.player\_stats\_id))))))\}
\end{aligned}$$
**English**

Find all players who played in all games in a given year for a given team.

**SQL**

```

SELECT p.name p.id FROM Players p, PlayersTeams pt, Teams t WHERE
p.pid = pt.pid AND pt.start >= input_year AND pt.end <=
input_year AND pt.tid = t.tid AND t.name = input_name AND NOT
EXISTS (SELECT * FROM Games g WHERE (g.atid = t.tid OR g.htid
= t.tid) AND g.year = input_year AND NOT EXISTS (SELECT *
FROM PlayersGames pg WHERE pg.gid = g.gid AND g.tid = t.tid
AND p.pid = pg.pid))

```

**RA**

$$\begin{aligned}
Games_y &\leftarrow \Pi_{Game\_id}(\sigma_{name="input\_name"} Team \bowtie_{team\_id} \sigma_{year="input\_year"} Games) \\
Team\_Players_y &\leftarrow \Pi_{player\_id}(\sigma_{start \leq "input\_year" \wedge end \geq "input\_year"} PlayerTeam \bowtie_{team\_id} \sigma_{name="input\_name"} Team) \\
& (Team\_Players_y \bowtie_{player\_id} PlayerGame) / Games_y
\end{aligned}$$
**TRC**

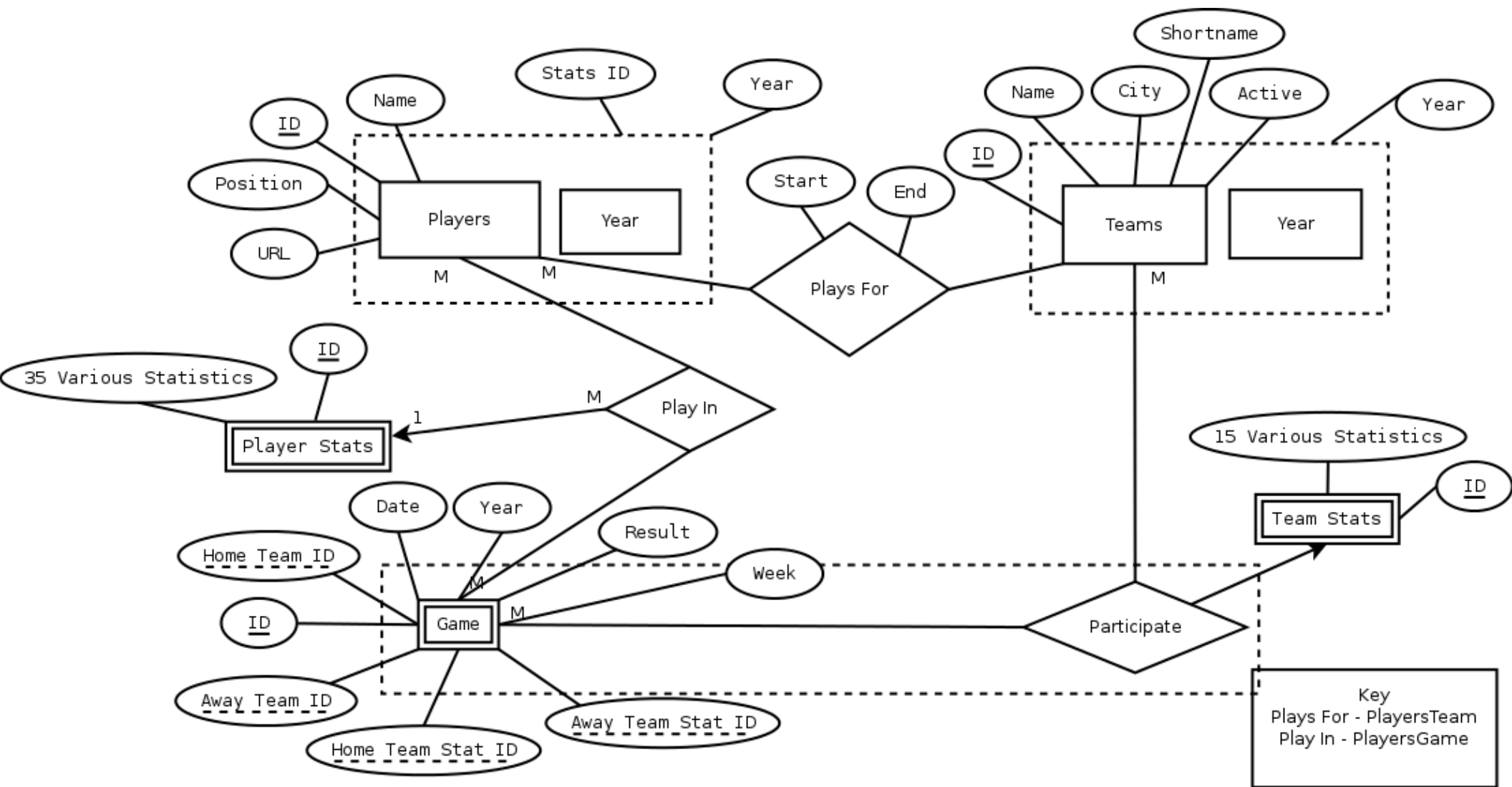
$$\begin{aligned}
&\{t^{(2)} | (\exists p)(Player(p) \wedge (t[1] = p.pid \wedge t[2] = p.name) \wedge (\exists T)(Team(T) \wedge T.name = "input\_name" \wedge \\
&(\exists pt)(PlayerTeam(pt) \wedge pt.pid = p.pid \wedge pt.tid = T.tid \wedge pt.startdate \leq input\_year \wedge \\
&pt.enddate \geq input\_year \wedge \\
&(\forall g)(Game(g) \wedge g.year = input\_year \wedge (g.home\_team\_id = t.tid \vee g.away\_team\_id = t.tid) \rightarrow \\
&(\exists pg)(PlayerGame(pg) \wedge \\
&pg.pid = p.pid \wedge pg.gid = g.gid \wedge pg.tid = t.tid))))))\}
\end{aligned}$$
**5.1.1 Other Queries**

1. Determining a teams win/loss record over a given period of time.
2. Determining a teams record against an opposing team over a given period of time.
3. Determining a teams record without a given player over a given period of time.
4. Calculating a teams offensive/defensive yardage with or without a given player.
5. Determining a teams performance based upon the weather conditions.
6. Determining a teams performance at a given location/stadium.
7. Finding out a teams best performance offensively or defensively in a given period of time.
8. Selecting all players who match a given statistics filter (e.g select a player with x or more yards of offense).
9. Find all players who have had x or more wins against team A.
10. Rank all players based upon a given stat (e.g rushing yardage).
11. Find two players that have combined for the most offensive yards in a given period of time.
12. Determine a players career record against a given team.
13. Determine a players career stats against a given team.
14. Determine a players win/loss ratio if he generates x or more yards against a given team.

## 6 Entity Relation Model

The ER model of this application is displayed on the next page. The model is formatted using standard practices to indicate primary keys, foreign keys, weak entities, participation constraints etc. A thorough overview of the database model is provided in the next section. Please note that the **Players Team** relation and **PlayersGame** relation were renamed to **Plays for** and **Plays in** in the ER diagram. This was done for the purposes of semantics so that the ER diagram read better.

---



## 7 Relational Model

The ER model above was translated into a relational model that accurately describes our actual database implementation. Please note that all of the data used to fill our database was acquired via web crawling and no user input is currently accepted in the application. Furthermore, none of the attributes in any relation are unfilled in any case. Only primary keys of relations are unique, and these keys are depicted as underlined. Foreign keys are indicated in red.

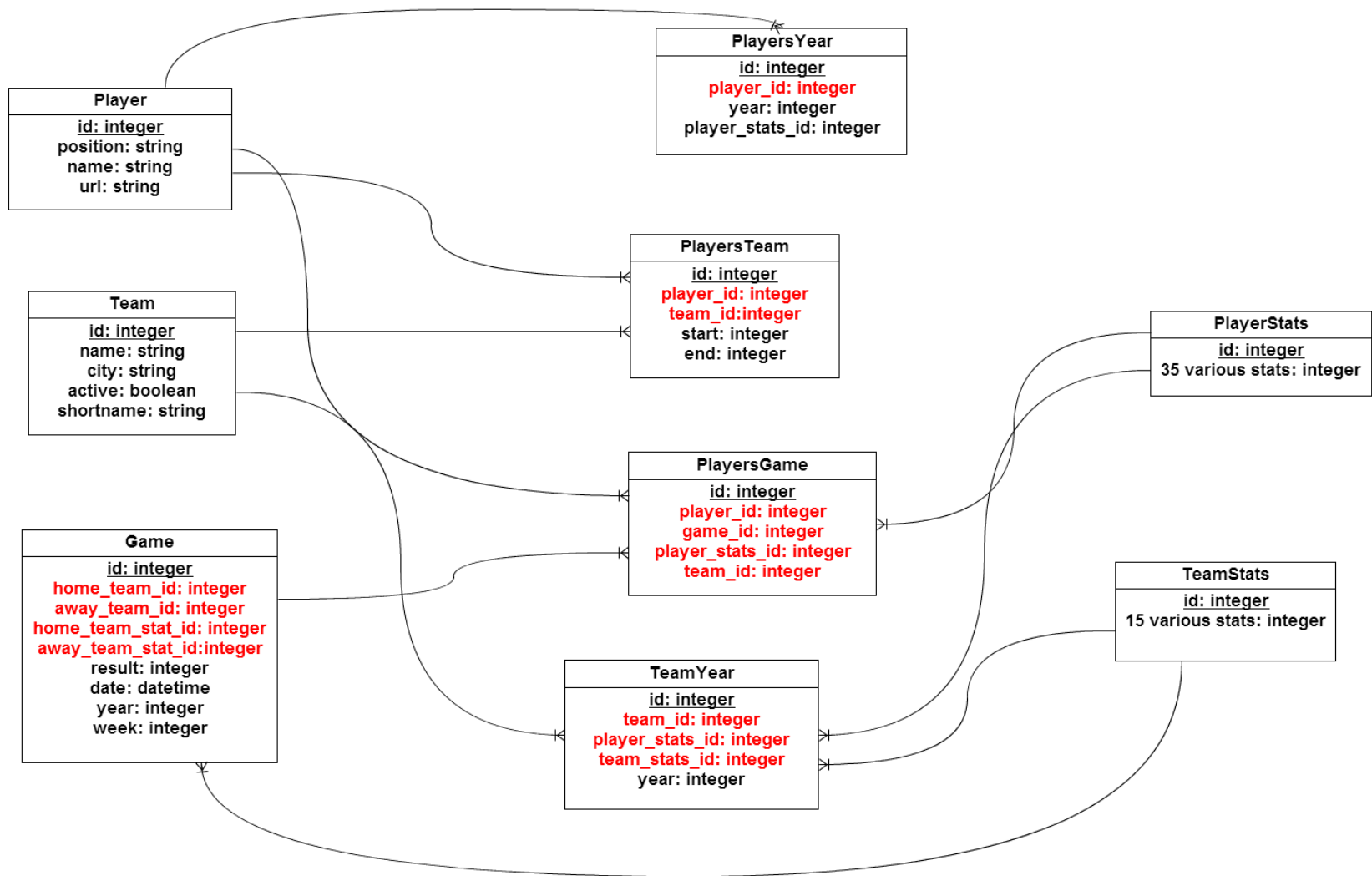
Note that the Game relation not only stores information about the game itself but foreign key references to the stats of the teams for that game and the teams themselves. Furthermore, the year field is stored as an attribute instead of its own entity. This is due to the fact that no attributes can be stored in a year entity however the year itself is important in determining yearly information in our database.

The player team relation has a start and end date because a player can not only play for multiple teams, but for the same team in one or more periods.

The player statistics stored are more detailed than the team statistics scored because the statistics that the author expects the users to find interesting reflect this. Also the data acquired was the most easily crawlable so that querying would be prioritized.

Lastly, the statistics attributes are not displayed in either the ER or relational model. This is simply due to the fact that there are too many to list and listing all of them does not reveal any insight about the database model. The player statistics and team statistics that are stored in our database can be viewed in section 11.3 of the appendix.





## 8 Database Implementation

### 8.1 DBMS

The DBMS used for StatAP was Sqlite3. The reasoning behind using Sqlite3 was the lack of need for features offered by more heavy weight DBMS such as MySQL and PostgreSQL, as well as the convenience offered in transfer of data. The easy transfer of data is important for development, as to properly develop, you need the database to be populated, and populating the database takes a fair amount of time.

Due to the nature of the application (extensive query), the primary keys of all tables were indexed using a built in Sqlite3 feature. Indexing greatly reduces the amount of time spent executing the querying and greatly improved the applications usability.

### 8.2 Obtaining Data

One of the large parts of the project was retrieving the data to populate the database. To do this, a web spider was built that scraped many pages of a popular sports reference website known as Pro Football Reference, and retrieved various information about players, teams, and games for professional football, dating as far back as 1920.

The spider was one of the more challenging parts of the project, as it required extensive studying of the data on the website being scraped, to properly account for all possible cases on the various different pages. As data was extracted various changes were made to the spider and bugs had to be solved. While Pro Football Reference displays data in a very standardized manner there were some issues. For example offensive stats for players were stored in a table that was called "Rushing and Receiving", but also "Receiving and Rushing".

### 8.3 SQL

The SQL creation statements used to create our database are listed below.

```
CREATE TABLE games (id INTEGER PRIMARY KEY AUTOINCREMENT NOT NULL,  
    date date, year integer, week integer, home_team_id integer,  
    away_team_id integer, home_team_stats_id integer,  
    away_team_stats_id integer, result integer, created_at datetime,  
    updated_at datetime);
```

```
CREATE TABLE player_stats (id INTEGER PRIMARY KEY AUTOINCREMENT NOT  
    NULL, rushing_attempts integer, rushing_yards integer,  
    rushing_touchdowns integer, fumbles integer, receptions integer,  
    receiving_yards integer, receiving_touchdowns integer, completions  
    integer, attempts integer, passing_yards integer,  
    passing_touchdowns integer, interceptions_thrown integer,  
    times_sacked integer, qb_rating float, solo_tackles integer,  
    assist_tackles integer, sacks float, passes_defended integer,  
    interceptions integer, interception_touchdowns integer,  
    fumbles_forced integer, fumbles_recovered integer,  
    fumbles_touchdowns integer, kick_return_attempts integer,  
    kick_return_yards integer, kick_return_touchdowns integer,
```

---

```
punt_return_attempts integer, punt_return_yards integer,
punt_return_touchdowns integer, field_goals_made integer,
field_goals_attempted integer, extra_points_made integer,
extra_points_attempted integer, punts integer, punt_yards integer,
created_at datetime, updated_at datetime);
CREATE TABLE players (id INTEGER PRIMARY KEY AUTOINCREMENT NOT NULL,
position varchar(255), name varchar(255), url varchar(255),
created_at datetime, updated_at datetime);
CREATE TABLE players_games (id INTEGER PRIMARY KEY AUTOINCREMENT NOT
NULL, player_id integer, game_id integer, team_id integer,
player_stats_id integer, created_at datetime, updated_at datetime)
;
CREATE TABLE players_teams (id INTEGER PRIMARY KEY AUTOINCREMENT NOT
NULL, player_id integer, team_id integer, start integer, end
integer, created_at datetime, updated_at datetime);
CREATE TABLE players_years (id INTEGER PRIMARY KEY AUTOINCREMENT NOT
NULL, player_id integer, year integer, player_stats_id integer,
created_at datetime, updated_at datetime);
CREATE TABLE schema_migrations (version varchar(255) NOT NULL);
CREATE TABLE team_stats (id INTEGER PRIMARY KEY AUTOINCREMENT NOT
NULL, wins integer, losses integer, ties integer, points_scored
integer, points_allowed integer, first_downs_made integer,
offensive_total_yards integer, offensive_passing_yards integer,
offensive_rushing_yards integer, turnovers_lost integer,
first_downs_allowed integer, total_yards_allowed integer,
passing_yards_allowed integer, rushing_yards_allowed integer,
turnovers_gained integer, created_at datetime, updated_at datetime
);
CREATE TABLE team_years (id INTEGER PRIMARY KEY AUTOINCREMENT NOT
NULL, team_id integer, team_stats_id integer, year integer,
created_at datetime, updated_at datetime);
CREATE TABLE teams (id INTEGER PRIMARY KEY AUTOINCREMENT NOT NULL,
name varchar(255), city varchar(255), active boolean, shortname
varchar(255), created_at datetime, updated_at datetime);
```

---

## 9 Relational Database Design

### 9.1 Description

The database schema was analyzed, and the authors found it to be compliant with BCNF and therefore 3NF as well. The data model established is simple yet elegant. The vast majority of queries that we tested responded fairly quickly.

There is some redundancy in our database in that we are storing yearly data as well as data on a game by game basis. This was done on purpose because the user is expected to very frequently query for yearly information which would have to be computed very redundantly. Furthermore, some of the queries the authors wanted to support would run more efficiently and strain our application much less with yearly data.

#### 1) Game

Keys:

$gid \rightarrow \{date, year, week, home\_team\_id, away\_team\_id, home\_team\_stat\_id, away\_team\_stat\_id, result\}$

$home\_team\_stat\_id \rightarrow \{determineseverything\}$

$away\_team\_stat\_id \rightarrow \{determineseverything\}^2$

#### 2) Player

Keys:

$pid \rightarrow \{position, name, url\}$

$url \rightarrow \{determineseverything\}^3$

#### 3) Team

Keys:

$tid \rightarrow \{name, city, active\}$

$shortname \rightarrow \{determineseverything\}$

$name \rightarrow \{determineseverything\}^4$

#### 4) PlayerTeam

Keys:

$\{\}$ emptyset

#### 5) PlayerGame

Keys:

$\{\}$ emptyset

#### 6) PlayerStats

Keys:

$player\_stats\_id \rightarrow \{determineseverything\}^5$

---

<sup>2</sup>8 total attributes

<sup>3</sup>3 total attributes

<sup>4</sup>4 total attributes

<sup>5</sup>35 total attributes

### 7) TeamStats

Keys:

$team\_stats\_id \rightarrow \{determineseverything\}^6$

### 8) PlayersYear

Keys:

$player\_year\_id \rightarrow \{player\_id, year, player\_stats\_id\}$

### 9) TeamYear

Keys:

$team\_year\_id \rightarrow \{team\_id, player\_stats\_id, team\_stats\_id, year\}$

---

<sup>6</sup>15 total attributes

---

## 10 Application Implementation

### 10.1 Overview

This application using the Ruby on Rails framework which integrates several components together for quick development and deployment of a scalable web application. As indicated before, Sqlite3 was used to implement the database, and Rails makes it easy to setup and interact with the database directly from the application level code. Rails provides several gems which are essentially software packages that can be easily integrated into the application to add functionality. These gems allow the application to include powerful open source features including interfacing with the database, crawling web data, parsing HTML, and other important tasks.

Very little work was put into the actual layout and appearance of the application. Twitter Bootstrap<sup>7</sup> was used to provide a fluid and simple yet elegant design. Bootstrap is simply a CSS and Javascript collection that can easily be integrated into any web application providing almost every front-end functionality necessary in an minimalist but organized style.

A significant portion of the application implementation work was done the database end for crawling the data. An extensive crawler was built using Nokogiri<sup>8</sup> which is useful in parsing HTML, XML, and other popular languages. Essentially the crawler visited web pages in a sequential manner specified by the authors. For each age using table headers, data was obtained and directly inserted into the database as Nokogiri is based off of the Rails framework.

### 10.2 Installation Manual

In order to install and run our application you must install the ruby and rails development environment on your machine. This environment is open source and compatible with Windows/-Mac/Linux. We recommend using the Ruby Version Control Manager for installation.<sup>9</sup> Once you have installed the Ruby on Rails environment on your machine, please follow the following steps to run our application.

1. Visit our Github repository, and download the source code.<sup>10</sup>
2. Open up a terminal and run **bundle install** after going inside the eecs-341 directory (note that rails will automatically detect that this is a rails application).
3. Now run **rake db:create** followed by **rake db:migrate**.
4. Start the server by running **rails s**.
5. Finally open a web browser and visit **localhost:3000**.<sup>11</sup>

---

<sup>7</sup><http://twitter.github.io/bootstrap/>

<sup>8</sup><http://nokogiri.org/>

<sup>9</sup>You can get rails at: [http://guides.rubyonrails.org/getting\\_started.html](http://guides.rubyonrails.org/getting_started.html)

<sup>10</sup>Github url: <https://github.com/Aaronneyer/eecs-341>

<sup>11</sup>Please note rails uses port 3000 by default, you can specify the port with -p.

---

## 11 Appendix

### 11.1 User Manual

In this section, the main functionality of the application is explained for a naive user. In the next few pages, each of the main views is examined and the important features are illustrated. Not all views are shown, including redundant search views that where other various queries can be executed. However, these pages have the same user interaction as the main query page displayed. The views are taken from running the application on a local machine.

The application is not in production, thus if a user must download and install it locally from Github. Please note that the data we have crawled is available on Github as well.

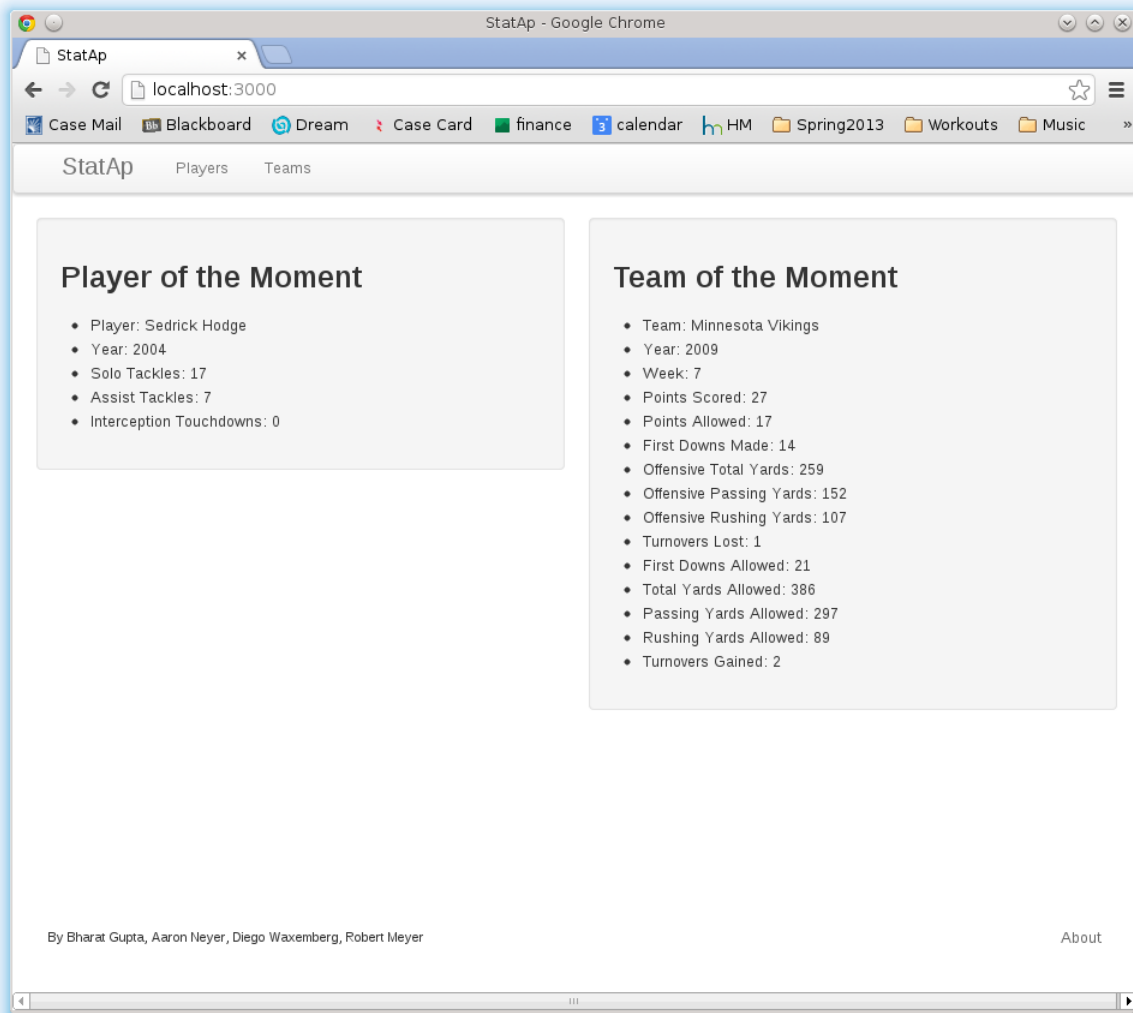


Figure 1: This is the home page view where you are shown a player and team "of the moment" (a randomly selected entity). From here you can navigate to the two main pages of interest, the Player page and the Team page.



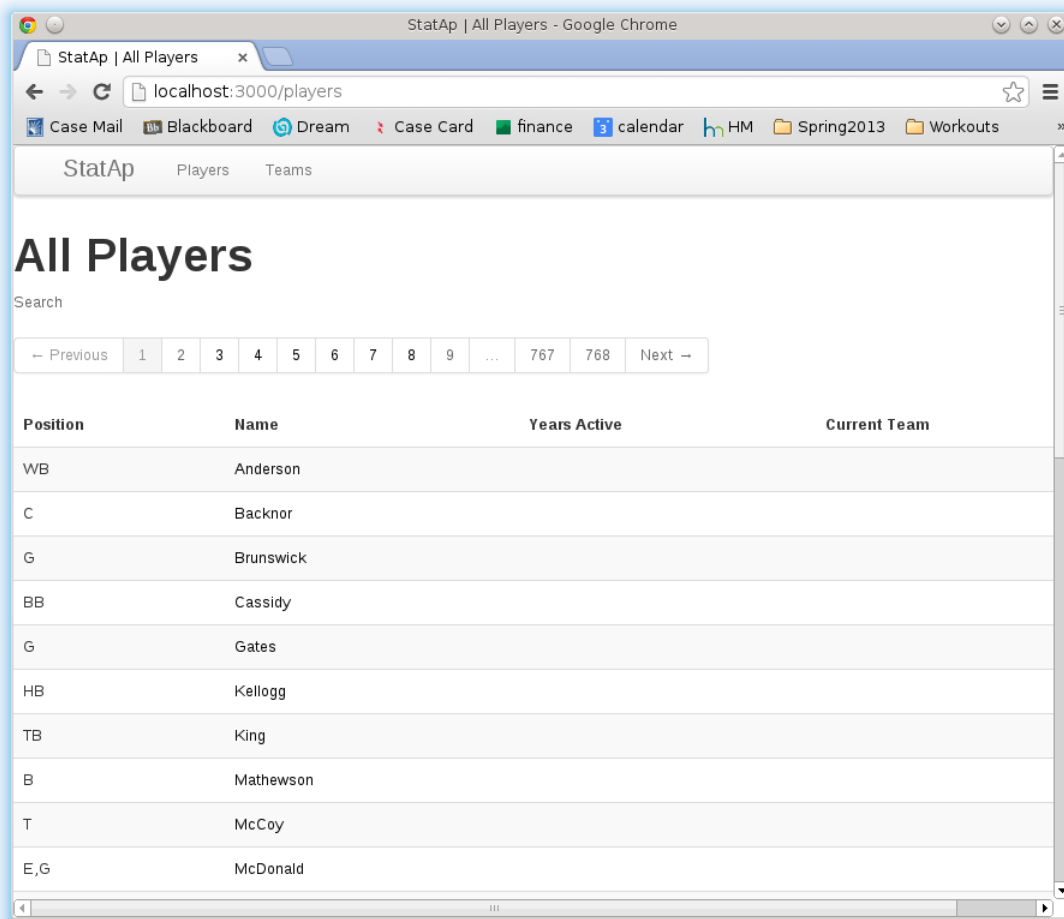
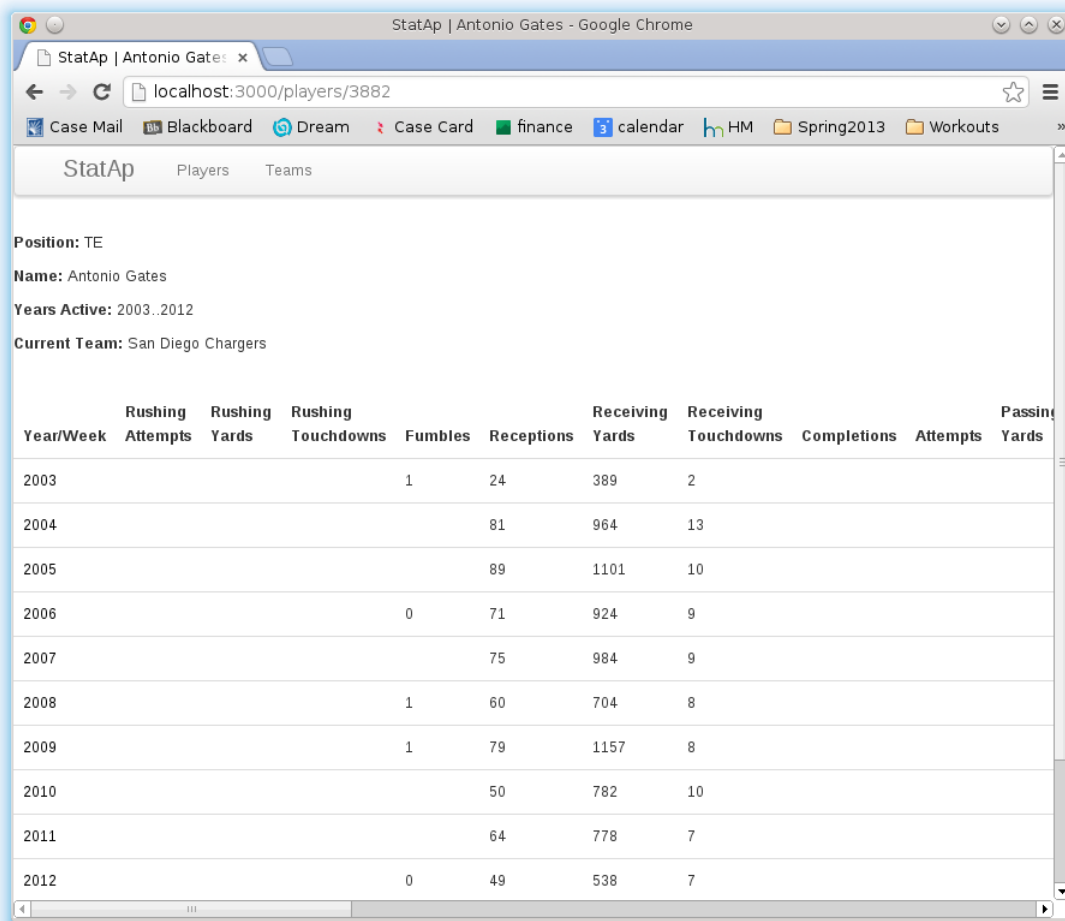


Figure 2: On the players page you can view every single player in the database. You can view more players by going to the next page. You can also go to the search or query page from this page.



StatAp | Antonio Gates - Google Chrome

localhost:3000/players/3882

Case Mail Blackboard Dream Case Card finance calendar HM Spring2013 Workouts

StatAp Players Teams

**Position:** TE  
**Name:** Antonio Gates  
**Years Active:** 2003..2012  
**Current Team:** San Diego Chargers

Year/Week	Rushing Attempts	Rushing Yards	Rushing Touchdowns	Fumbles	Receptions	Receiving Yards	Receiving Touchdowns	Completions	Attempts	Passing Yards
2003				1	24	389	2			
2004					81	964	13			
2005					89	1101	10			
2006				0	71	924	9			
2007					75	984	9			
2008				1	60	704	8			
2009				1	79	1157	8			
2010					50	782	10			
2011					64	778	7			
2012				0	49	538	7			

Figure 3: This page shows you an individual players statistics organized by year. By clicking on each year you can view the players per game stats.

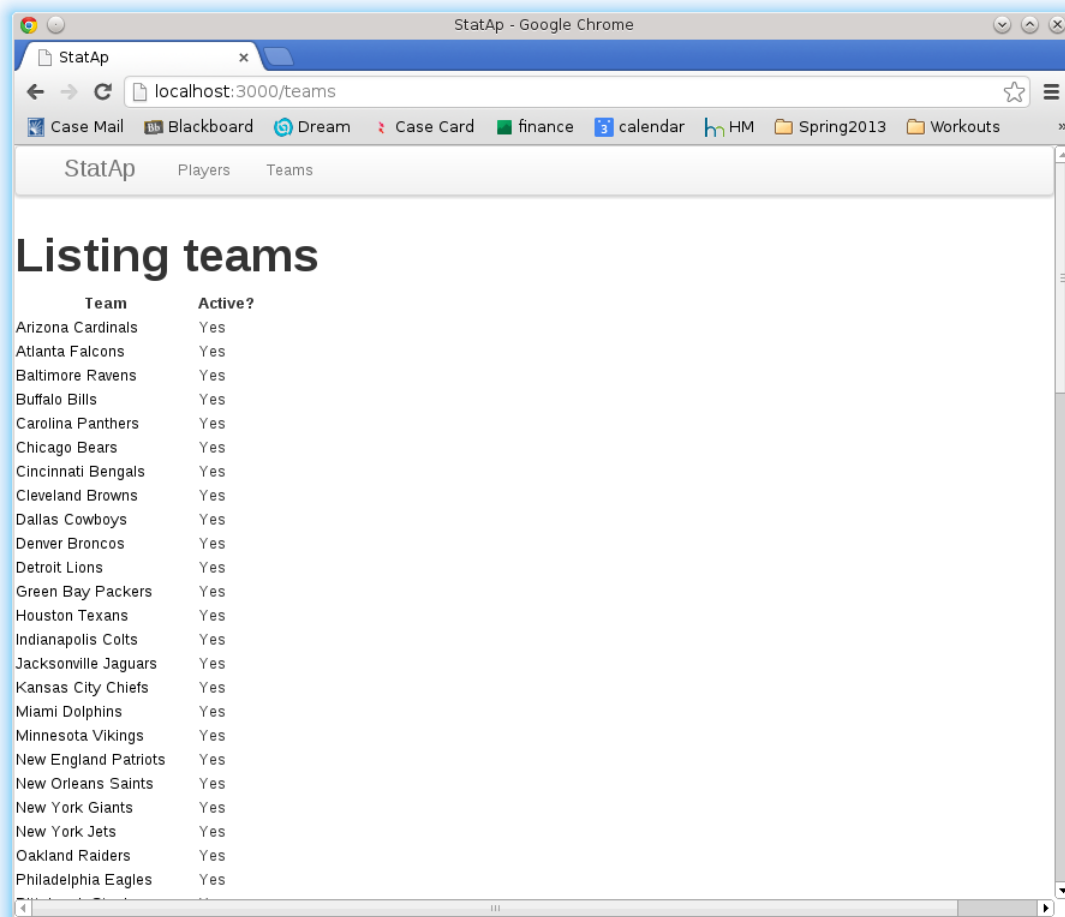


Figure 4: The teams page displays all of the teams similarly to the players page. It is not as functional as the players page, however, you can view all teams and get more detailed information.

StatAp - Google Chrome

localhost:3000/teams/12

Case Mail Blackboard Dream Case Card finance calendar HM Spring2013 Workouts

StatAp Players Teams

City: Green Bay  
Name: Packers

Year/Week	Wins	Losses	Ties	Points Scored	Points Allowed	First Downs Made	Offensive Total Yards	Offensive Passing Yards	Offensive Rushing Yards	Turnovers Lost	First Downs Allowed	Total Yards Allowed
2012	11	5	0	433	336	341	5751	4049	1702	16	308	5388
2011	15	1	0	560	359	353	6482	4924	1558	14	358	6585
2010	10	6	0	388	240	312	5730	4124	1606	22	270	4945
2009	11	5	0	461	297	335	6065	4180	1885	16	272	4551
2008	6	10	0	419	380	299	5618	3813	1805	21	295	5349
2007	13	3	0	435	291	307	5931	4334	1597	24	297	5013
2006	8	8	0	301	366	301	5458	3795	1663	33	291	5134
2005	4	12	0	298	344	318	5118	3766	1352	45	280	4690
2004	10	6	0	424	380	354	6357	4449	1908	29	307	5541
2003	10	6	0	442	307	315	5798	3240	2558	32	288	5101
2002	12	4	0	398	328	318	5560	3627	1933	28	294	4985
2001	12	4	0	388	366	303	5482	3370	1602	27	278	4897

Figure 5: Similar to the player view the view displays a teams statistical information organized by year. By clicking on each view you can view the statistical information on a per game basis.

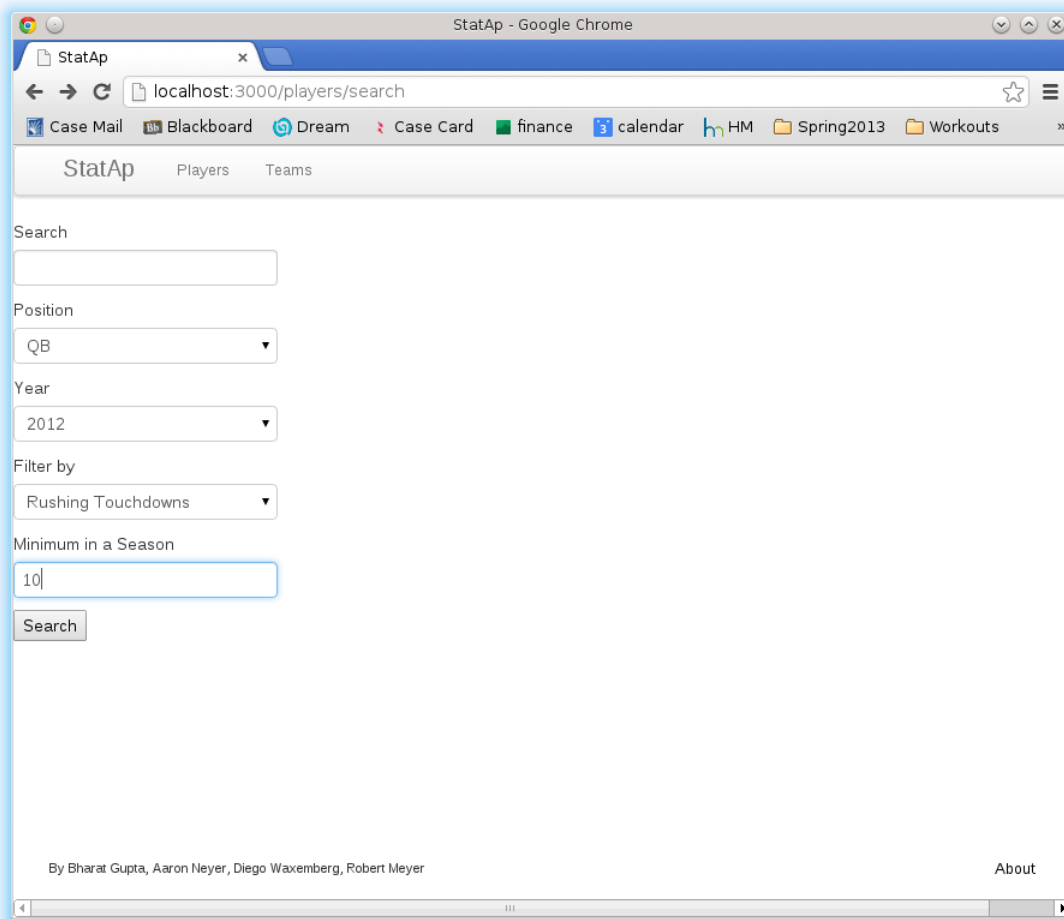


Figure 6: The query or search page can be used to extract data based upon parameters. As shown once you enter the parameters simply click search to view the results. Other query pages each have their own view with a similar style.

## 11.2 Programmers Manual

StatAP is implemented using Ruby on Rails. Rails uses an MVC framework, which has Models, Views, and Controllers. The models are the classes, which represent our tables in the database. For example, we have a Player class in our app/models directory, which maps to the players table in the database.

Each of the models has code in it, identifying it's relationships with other tables, to allow for relational queries to be made. There can also be scopes defined, allowing for shortcuts for commonly used partial queries. For example, player.rb has the following scope:

```
scope :years_stats, -> {joins(:players_years => :players_stats)}
```

The controllers handle rendering views with proper information. The controller containing the most logic is the players\_controller. This handles rendering everything within players, which is the index page, which shows all players, search page, which renders a search form to perform queries over the players, results, which handles the results of the search, and then show, which handles showing information about individual players.

The player results controller takes information passed in from the user, and depending on what items are passed, it filters down the @players variable. For example, if there is a year or a statistic passed, it will take a natural join on players\_years and players\_stats using the years\_stats scope previously defined, and will filter @players where it matches the user input. It will then paginate the results, by limiting them to 30, and offsetting them dependent on the page number passed in the parameters.

The views handle showing the actual information to the user. Again, the primary views are with players, although the team views are also heavily used. The views primarily correlate to the controller actions, although some are reused. For example, results renders the index view, because the method of showing information is equivalent for both, it just shows different information, dependent on the @players variable. The players show view will retrieve all stats for a given player by year and game and render them, with the games being initially hidden, and then shown when the user clicks on the year. The teams show view acts in a similar manner.

To populate the database, a spider is used. This scrapes information from [www.pro-football-reference.com](http://www.pro-football-reference.com) using the 'open-uri' ruby gem, and then parses it with Nokogiri, which generates an HTML parse tree and allows for css selection. Using this, information is gathered on every player, team, and game. With the information gathered, active\_record statements are called to insert rows into the database for these various entities and relationships. This is a lengthy process and can take a few hours, as there are over 20,000 players, as well as many different games and teams.

## 11.3 Player Statistics

t.integer	"rushing_attempts"
t.integer	"rushing_yards"
t.integer	"rushing_touchdowns"
t.integer	"fumbles"
t.integer	"receptions"
t.integer	"receiving_yards"
t.integer	"receiving_touchdowns"
t.integer	"completions"
t.integer	"attempts"
t.integer	"passing_yards"
t.integer	"passing_touchdowns"
t.integer	"interceptions_thrown"
t.integer	"times_sacked"
t.float	"qb_rating"
t.integer	"solo_tackles"
t.integer	"assist_tackles"
t.float	"sacks"
t.integer	"passes_defended"
t.integer	"interceptions"
t.integer	"interception_touchdowns"
t.integer	"fumbles_forced"
t.integer	"fumbles_recovered"
t.integer	"fumbles_touchdowns"
t.integer	"kick_return_attempts"
t.integer	"kick_return_yards"
t.integer	"kick_return_touchdowns"
t.integer	"punt_return_attempts"
t.integer	"punt_return_yards"
t.integer	"punt_return_touchdowns"
t.integer	"field_goals_made"
t.integer	"field_goals_attempted"
t.integer	"extra_points_made"
t.integer	"extra_points_attempted"
t.integer	"punts"
t.integer	"punt_yards"

## 11.4 Team Statistics

```
t.integer  "wins"
t.integer  "losses"
t.integer  "ties"
t.integer  "points_scored"
t.integer  "points_allowed"
t.integer  "first_downs_made"
t.integer  "offensive_total_yards"
t.integer  "offensive_passing_yards"
t.integer  "offensive_rushing_yards"
t.integer  "turnovers_lost"
t.integer  "first_downs_allowed"
t.integer  "total_yards_allowed"
t.integer  "passing_yards_allowed"
t.integer  "rushing_yards_allowed"
t.integer  "turnovers_gained"
```

---