**CS 4750 Project—Football Database**

Yong Jae Kwon(yk7da), Taehyun Kim(tk9at), Andrew Yang(ahy9ng)

Department of Computer Science, University of Virginia

**Abstract:** This project’s main goal is to create a database and corresponding website that contains the data of all football players in the 6 football teams in Virginia.

**1. Project Information.**

**1. Introduction**

Our project’s goal is to create a database for a college football website. The users of this website will be able to search different football players in the 6 NCAA-1 level football teams in Virginia, and to get the match data of those football teams. Each player information stores the player’s height and weight, main class, address, and team. Each match information stores the date, the participating teams, scores for each team, the passing leader, the rushing leader, and the receiving leader. Each users will have an account that stores information of their favorite players. The administrator will have a separate account that can insert and delete players from the players table, which is linked with other school-specific and position-specific tables.

**2. Requirements**

Users should be able to create, sign into and log out accounts. Accounts will only contain the user’s username, password, and their fav\_player list. The information of individual football players will be stored in the Players table in the football database. The details for each player includes their number, position, class, height, weight, hometown, state, team, and their ID (which was generated respective to their team and number). These details will be stored in the main ‘Players’ table and the tables corresponding to the player’s school and team position. The tables have constraints to detect if a player is truly is in a certain school/position. The information of individual matches, which will be stored in the match\_2016 table, include the details of the two teams, the final score, the date and time, and leading players. Users will be able to search certain players either by their names, or by their team, or by their position. They will also be able to insert certain players in their favorite players list, and to delete items from the favorites list. The user can also export the Players table in JSON form. The admin user will be able to insert/delete players from the Players list, which, using triggers, will also change the tables corresponding to the player.

Security is handled so that it is safe against SQL injection attacks. There is a separate user that will be the only user which will be reading that login table. As mentioned above, there is only one admin that can do the insert/delete operations on the entire database, and the rest of the users can only select some tables or insert/delete the favorites table.

**2. Design Process**

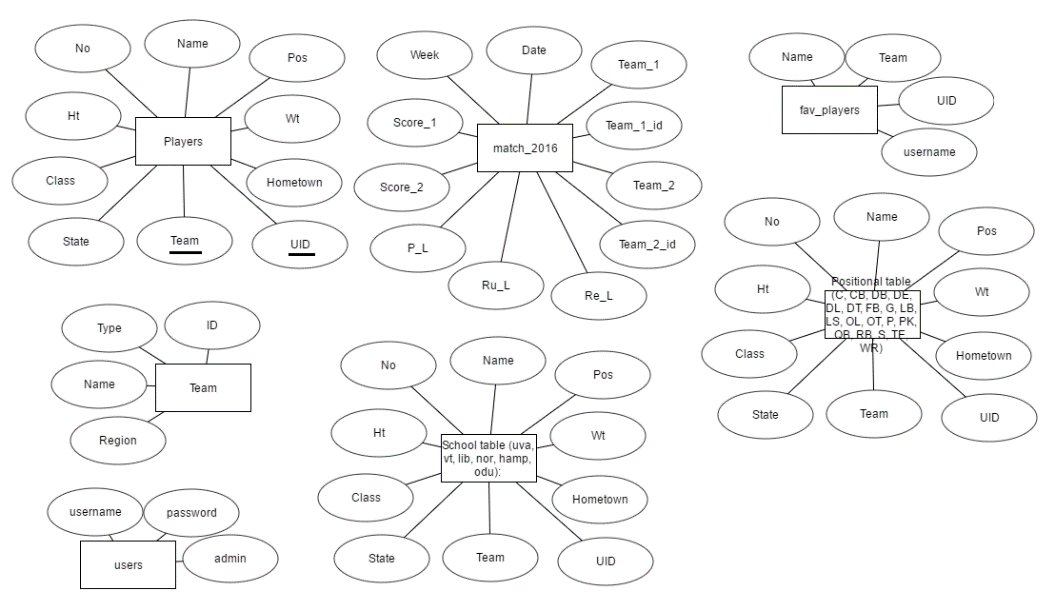
**1.** **Design Decisions**

In order to make things easy for users to access and acquire information, we decided to use website as the interface application. While we didn’t used data encryption, for security we separated the data and code and prepared for possible SQL injection threats. We also granted different types of privileges for different users.

Most of the database is consisted of player data. The ‘Players’ table is consisted of the information of football players, including their current team and main position. There exists other tables that includes information of football players in a certain school or in a certain position (Example: table ‘uva’ contains all players in Virginia Cavaliers, table ‘QB’ contains all quarterbacks).

**2. E-R Diagram.**

Because there are a lot of tables and attributes for each table, the tables with same structures (i.e. school table and positional table) was grouped up. The E-R Diagram is shown below.



**3. Database Schema**

Players:

`No` varchar (2) CHARACTER SET utf8 DEFAULT NULL,

`Name` varchar (23) CHARACTER SET utf8 DEFAULT NULL,

`Pos` varchar (2) CHARACTER SET utf8 DEFAULT NULL,

`Ht` varchar (6) CHARACTER SET utf8 DEFAULT NULL,

`Wt` int (11) DEFAULT NULL,

`Class` varchar (2) CHARACTER SET utf8 DEFAULT NULL,

`Hometown` varchar (21) CHARACTER SET utf8 DEFAULT NULL,

`State` varchar (2) CHARACTER SET utf8 DEFAULT NULL,

`Team` varchar (3) CHARACTER SET utf8 NOT NULL DEFAULT ,

`UID` varchar (5) NOT NULL DEFAULT

PRIMARY KEY (`Team`,`UID`)

CREATE TRIGGER `Players\_trigger` AFTER INSERT ON `Players`

FOR EACH ROW BEGIN

IF (new.Team = ‘player\_team)

THEN

INSERT INTO ` player\_team ` (`No`, `Name`, `Pos`, `Ht`, `Wt`, `Class`, `Hometown`, `State`, `Team`, `UID`) VALUES ( new.No, new.Name, new.Pos, new.Ht, new.Wt, new.Class, new.Hometown , new.State , new.Team, new.UID );

END IF;

IF (new.Pos = ‘player\_pos’)

THEN

INSERT INTO `player\_pos` (`No`, `Name`, `Pos`, `Ht`, `Wt`, `Class`, `Hometown`, `State`, `Team`, `UID`) VALUES ( new.No, new.Name, new.Pos, new.Ht, new.Wt, new.Class, new.Hometown , new.State , new.Team, new.UID );

END IF;

END

School table (uva, vt, lib, nor, hamp, odu):

`No` varchar (2) CHARACTER SET utf8 DEFAULT NULL,

`Name` varchar (23) CHARACTER SET utf8 DEFAULT NULL,

`Pos` varchar (2) CHARACTER SET utf8 DEFAULT NULL,

`Ht` varchar (6) CHARACTER SET utf8 DEFAULT NULL,

`Wt` int (11) DEFAULT NULL,

`Class` varchar (2) CHARACTER SET utf8 DEFAULT NULL,

`Hometown` varchar (21) CHARACTER SET utf8 DEFAULT NULL,

`State` varchar (2) CHARACTER SET utf8 DEFAULT NULL,

`Team` varchar (3) CHARACTER SET utf8 NOT NULL DEFAULT ,

`UID` varchar (5) NOT NULL DEFAULT

PRIMARY KEY (`Team`,`UID`)

ADD CONSTRAINT `FK\_PLAYER\_(school\_name)` FOREIGN KEY (`Team`, `UID`) REFERENCES `Players` (`Team`, `UID`) ON DELETE CASCADE ON UPDATE CASCADE;

Positional table (C, CB, DB, DE, DL, DT, FB, G, LB, LS, OL, OT, P, PK, QB, RB, S, TE, WR):

`No` varchar (2) CHARACTER SET utf8 DEFAULT NULL,

`Name` varchar (23) CHARACTER SET utf8 DEFAULT NULL,

`Pos` varchar (2) CHARACTER SET utf8 DEFAULT NULL,

`Ht` varchar (6) CHARACTER SET utf8 DEFAULT NULL,

`Wt` int (11) DEFAULT NULL,

`Class` varchar (2) CHARACTER SET utf8 DEFAULT NULL,

`Hometown` varchar (21) CHARACTER SET utf8 DEFAULT NULL,

`State` varchar (2) CHARACTER SET utf8 DEFAULT NULL,

`Team` varchar (3) CHARACTER SET utf8 NOT NULL DEFAULT ,

`UID` varchar (5) NOT NULL DEFAULT

PRIMARY KEY (`Team`,`UID`)

ADD CONSTRAINT `FK\_PLAYER\_(pos\_name)` FOREIGN KEY (`Team`, `UID`) REFERENCES `Players` (`Team`, `UID`) ON DELETE CASCADE ON UPDATE CASCADE;

Users:

`username` varchar (16) CHARACTER SET utf8 NOT NULL DEFAULT,

`password` varchar (16) CHARACTER SET utf8 NOT NULL DEFAULT,

`admin` tinyint(1) NOT NULL DEFAULT

Fav\_players:

`Name` varchar (18) CHARACTER SET utf8 NOT NULL DEFAULT,

`Team` varchar (3) CHARACTER SET utf8 NOT NULL DEFAULT ',

`UID` varchar (5) NOT NULL DEFAULT,

`username` varchar (16) CHARACTER SET utf8 NOT NULL DEFAULT

Match\_2016:

`Week` varchar(4) CHARACTER SET utf8 DEFAULT NULL,

`Date` varchar(22) CHARACTER SET utf8 DEFAULT NULL,

`Team\_1` varchar(37) CHARACTER SET utf8 DEFAULT NULL,

`Team\_1\_id` varchar(5) CHARACTER SET utf8 DEFAULT NULL,

`Team\_2` varchar(34) CHARACTER SET utf8 DEFAULT NULL,

`Team\_2\_id` varchar(5) CHARACTER SET utf8 DEFAULT NULL,

`Score\_1` int(11) DEFAULT NULL,

`Score\_2` int(11) DEFAULT NULL,

`P\_L` varchar(16) CHARACTER SET utf8 DEFAULT NULL,

`Ru\_L` varchar(16) CHARACTER SET utf8 DEFAULT NULL,

`Re\_L` varchar(19) CHARACTER SET utf8 DEFAULT NULL

Teams:

`ID` varchar(5) CHARACTER SET utf8 NOT NULL DEFAULT '',

`Type` varchar(11) CHARACTER SET utf8 DEFAULT NULL,

`Name` varchar(44) CHARACTER SET utf8 DEFAULT NULL,

`Region` varchar(21) CHARACTER SET utf8 DEFAULT NULL

**4. 3NF Proof**

1) Players:

Calculating F+:

Functional Dependencies:

No → No

Name → Name

Pos → Pos

Ht → Ht

Wt → Wt

Class → Class

Hometown → Hometown

State → State

Team → Team

UID → UID

All the dependencies are trivial since all attributes are linearly independent of each other. It can be argued that Hometown and State can be considered linearly dependent, but for convenience factor, we considered the two attributes linearly independent.

Calculating Fc:

All dependencies are trivial, so Fc is the empty set.

Deriving 3NF tables from Fc:

Since all dependencies are trivial, the table containing all of the attribute is 3NF-compliant.

2) School table (uva, vt, lib, nor, hamp, odu):

Calculating F+:

Functional Dependencies:

No → No

Name → Name

Pos → Pos

Ht → Ht

Wt → Wt

Class → Class

Hometown → Hometown

State → State

Team → Team

UID → UID

All the dependencies are trivial since all attributes are linearly independent of each other. It can be argued that Hometown and State can be considered linearly dependent, but for convenience factor, we considered the two attributes linearly independent.

Calculating Fc:

All dependencies are trivial, so Fc is the empty set.

Deriving 3NF tables from Fc:

Since all dependencies are trivial, the table containing all of the attribute is 3NF-compliant.

3) Positional table (C, CB, DB, DE, DL, DT, FB, G, LB, LS, OL, OT, P, PK, QB, RB, S, TE, WR):

Calculating F+:

Functional Dependencies:

No → No

Name → Name

Pos → Pos

Ht → Ht

Wt → Wt

Class → Class

Hometown → Hometown

State → State

Team → Team

UID → UID

All the dependencies are trivial since all attributes are linearly independent of each other. It can be argued that Hometown and State can be considered linearly dependent, but for convenience factor, we considered the two attributes linearly independent.

Calculating Fc:

All dependencies are trivial, so Fc is the empty set.

Deriving 3NF tables from Fc:

Since all dependencies are trivial, the table containing all of the attribute is 3NF-compliant.

4) Users:

Calculating F+:

Functional Dependencies:

username → username

password → password

admin → admin

All the dependencies are trivial since all attributes are linearly independent of each other.

Calculating Fc:

All dependencies are trivial, so Fc is the empty set.

Deriving 3NF tables from Fc:

Since all dependencies are trivial, the table containing all of the attribute is 3NF-compliant.

5) fav\_players

Calculating F+:

Functional Dependencies:

Name → Name

Team → Team

UID → UID

username → username

All the dependencies are trivial since all attributes are linearly independent of each other.

Calculating Fc:

All dependencies are trivial, so Fc is the empty set.

Deriving 3NF tables from Fc:

Since all dependencies are trivial, the table containing all of the attribute is 3NF-compliant.

6) match\_2016

Calculating F+:

Functional Dependencies:

Week → Week

Date → Date

Team\_1 → Team\_1

Team\_1\_id → Team\_1\_id

Team\_2 → Team\_2

Team\_2\_id → Team\_2\_id

Score\_1 → Score\_1

Score\_2 → Score\_2

P\_L → P\_L

Ru\_L → Ru\_L

Re\_L → Re\_L

All the dependencies are trivial since all attributes are linearly independent of each other. It can be argued that Team\_1 and Team\_1\_id along with Team\_2 and Team\_2\_id can be considered linearly dependent, but for convenience factor, we considered these attributes linearly independent.

Calculating Fc:

All dependencies are trivial, so Fc is the empty set.

Deriving 3NF tables from Fc:

Since all dependencies are trivial, the table containing all of the attribute is 3NF-compliant.

7) teams

Calculating F+:

Functional Dependencies:

ID → ID

Type → Type

Name → Name

Region → Region

All the dependencies are trivial since all attributes are linearly independent of each other.

Calculating Fc:

All dependencies are trivial, so Fc is the empty set.

Deriving 3NF tables from Fc:

Since all dependencies are trivial, the table containing all of the attribute is 3NF-compliant.

**3. Evaluation of Product**

**1. Testing Procedure.**

Sample queries listed in the section below were tested through the front end application and the database in phpMyAdmin. Checks and Triggers were also tested through inserting rows and testing the behavior of the checks and triggers. The tests done are listed as the sample data and queries below.

**2. Sample data and Sample Queries.**

A. Logging in with the provided username.

COUNT(SELECT \* FROM users WHERE username = $myusername AND password = $mypassword) → 1 for any successful login and 0 for unsuccessful login.

B. Insert a player into database

INSERT INTO `Players` (`No`, `Name`, `Pos`, `Ht`, `Wt`, `Class`, `Hometown`, `State`, `Team`, `UID`) VALUES

('1', 'DeCarlo Hamiltona', 'DT', ' 06-03', 335, 'FR', 'Plantation', 'FL', 'Lib', '3001'); →

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | DeCarlo Hamiltona | DT | 06-03 | 335 | FR | Plantation | FL | Lib | 3001 |

Inserted into table Players, DT, and lib.

C. Deleting a player from database.

DELETE FROM `Players` WHERE Team = Lib AND UID = 3001→

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | DeCarlo Hamiltona | DT | 06-03 | 335 | FR | Plantation | FL | Lib | 3001 |

Deleted from table Players, DT, and lib.

D. Searching a player

SELECT \* FROM Players WHERE Name LIKE Hamiltona →

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | DeCarlo Hamiltona | DT | 06-03 | 335 | FR | Plantation | FL | Lib | 3001 |

Shown as the result of the query.

E. Searching a position

SELECT \* from C →

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **No** | **Name** | **Ht** | **Wt** | **Class** | **Hometown** | **Pos** | **State** | **Team** | **UID** | **Check** |
| 50 | Jackson Matteoa | 06-05 | 290 | SR | Ashburn | C | VA | UVA | 1071 |  |
| 78 | R.J. Proctora | 06-05 | 305 | FR | Bealeton | C | VA | UVA | 1089 |  |

Above table shown as result of query.

F. Searching by school

SELECT \* FROM UVA →

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **No** | **Name** | **Ht** | **Wt** | **Class** | **Hometown** | **Pos** | **State** | **Team** | **UID** | **Check** |
| -- | Naji Abdullaha | 6-5 | 235 | FR | Jacksonville | DE | FL | UVA | 1001 |  |
| -- | Sonny Abramsona | 6-2 | 210 | FR | Chester | QB | NJ | UVA | 1002 |  |

Above table shown as result of query.