[COP 4710] Project 4: Final Report

Matthew Kramer, Nghiem Ngo, and Sterling Price

Tools

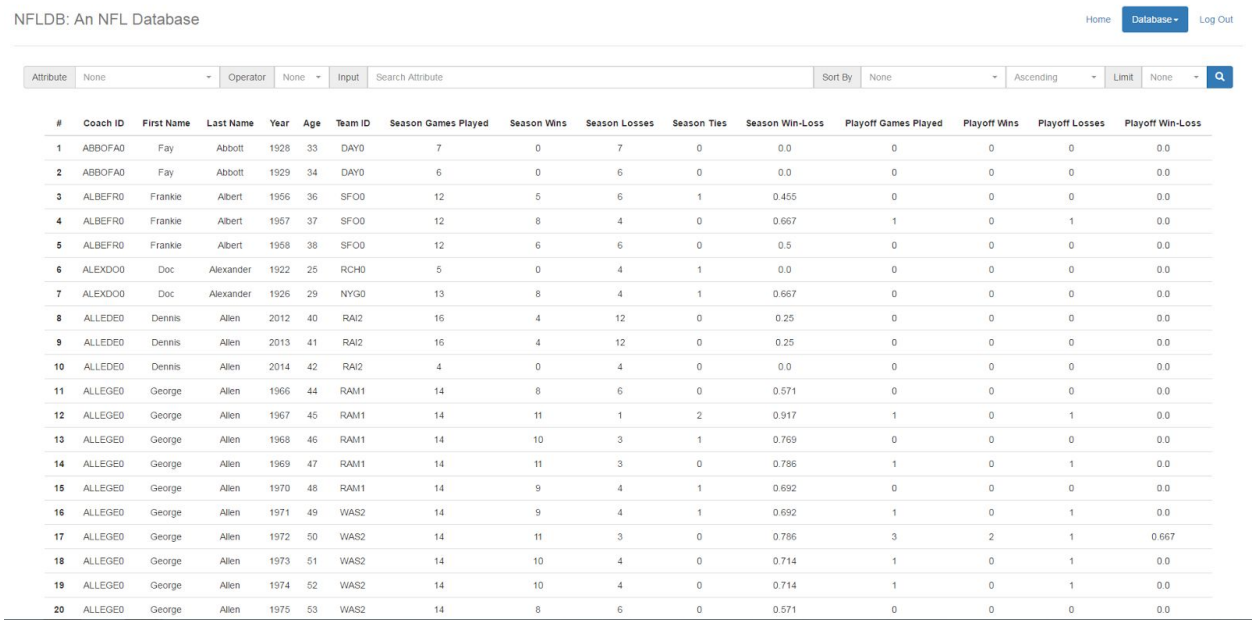
We chose to develop a Python-Flask web application backed by a MySQL database instance hosted through Amazon Web Services (AWS) Relational Database Service (RDS). Our data was aggregated from a number of sources, including pro-football-reference.com and nflsavant.com. In order to interact with the database and its schemas, tables, and stored procedures, we used MySQL Workbench: a user-friendly graphical user interface developed by Oracle for MySQL databases. Lastly, in order to make group collaboration more straightforward, we used a private Github repository in order to maintain version control and keep up to date with the latest changes made by group members. In total, we have used five programming languages to develop this application: MySQL, Python, HTML, CSS, and Javascript.

Interface

The program that we created is hosted on an AWS Elastic Beanstalk Instance (ELI) making it publicly accessible from the internet. Users wishing to access the application need only visit the domain that we have registered: nfldb.me. Once on the home page, users may register for a new account or log in to an existing account. Both of these features operate using queries to a table of usernames and password hashes stored in the database. In order to access the administrator page, you will need to use the admin account (username: admin, password: password).

Application home page

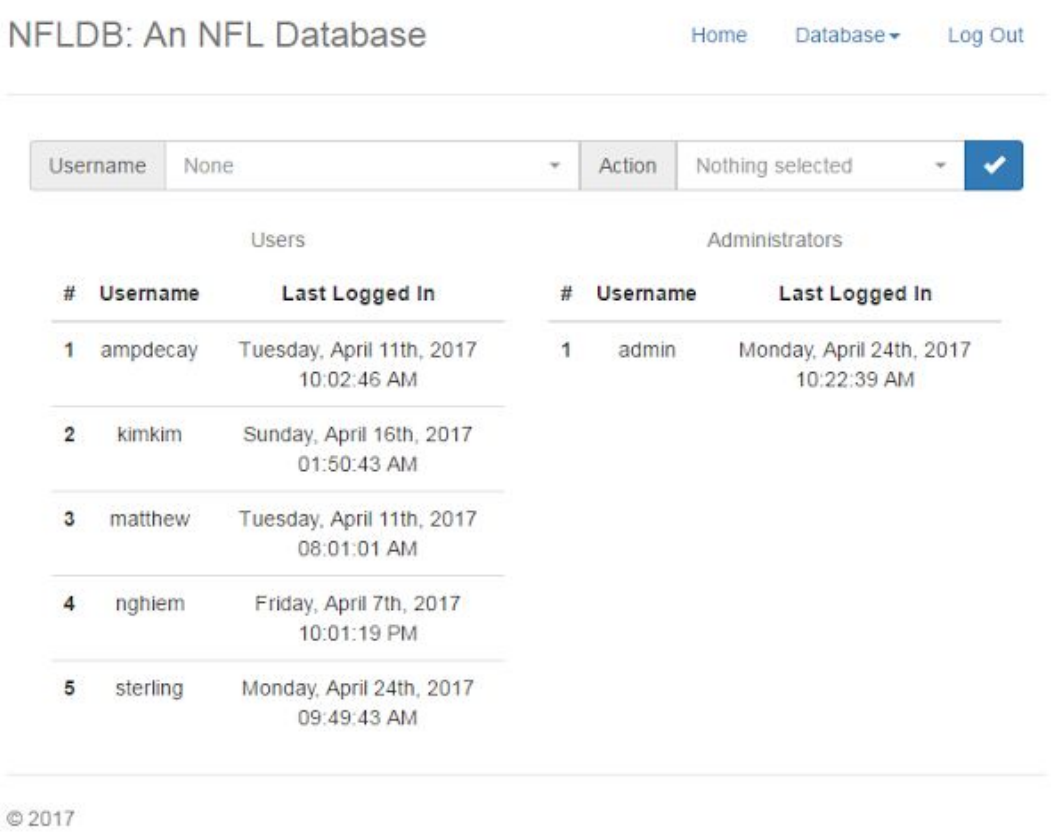
In total, regular users can access six tables: coaches, players, games, super bowls, franchises, and teams. From the page for each table, a user can construct a query using any attribute of that table, an operator with which to compare a given input, the input that the user provides, and then from that query sort and limit the results. For example, a query on the page for the **`coaches`** table could look like: **`Season Win > '10', Order By First Name Desc Limit 10`**. It should be noted that "None" is an acceptable choice for all fields except the input and sort by fields.



#	Coach ID	First Name	Last Name	Year	Age	Team ID	Season Games Played	Season Wins	Season Losses	Season Ties	Season Win-Loss	Playoff Games Played	Playoff Wins	Playoff Losses	Playoff Win-Loss
1	ABBOFA0	Fay	Abbott	1928	33	DAY0	7	0	7	0	0.0	0	0	0	0.0
2	ABBOFA0	Fay	Abbott	1929	34	DAY0	6	0	6	0	0.0	0	0	0	0.0
3	ALBEFR0	Frankie	Albert	1956	36	SFO0	12	5	6	1	0.455	0	0	0	0.0
4	ALBEFR0	Frankie	Albert	1957	37	SFO0	12	8	4	0	0.667	1	0	1	0.0
5	ALBEFR0	Frankie	Albert	1958	38	SFO0	12	6	6	0	0.5	0	0	0	0.0
6	ALEXDO0	Doc	Alexander	1922	25	RCH0	5	0	4	1	0.0	0	0	0	0.0
7	ALEXDO0	Doc	Alexander	1926	29	NYG0	13	8	4	1	0.667	0	0	0	0.0
8	ALLEDE0	Dennis	Allen	2012	40	RAI2	16	4	12	0	0.25	0	0	0	0.0
9	ALLEDE0	Dennis	Allen	2013	41	RAI2	16	4	12	0	0.25	0	0	0	0.0
10	ALLEDE0	Dennis	Allen	2014	42	RAI2	4	0	4	0	0.0	0	0	0	0.0
11	ALLEGE0	George	Allen	1966	44	RAM1	14	8	6	0	0.571	0	0	0	0.0
12	ALLEGE0	George	Allen	1967	45	RAM1	14	11	1	2	0.917	1	0	1	0.0
13	ALLEGE0	George	Allen	1968	46	RAM1	14	10	3	1	0.769	0	0	0	0.0
14	ALLEGE0	George	Allen	1969	47	RAM1	14	11	3	0	0.786	1	0	1	0.0
15	ALLEGE0	George	Allen	1970	48	RAM1	14	9	4	1	0.692	0	0	0	0.0
16	ALLEGE0	George	Allen	1971	49	WAS2	14	9	4	1	0.692	1	0	1	0.0
17	ALLEGE0	George	Allen	1972	50	WAS2	14	11	3	0	0.786	3	2	1	0.667
18	ALLEGE0	George	Allen	1973	51	WAS2	14	10	4	0	0.714	1	0	1	0.0
19	ALLEGE0	George	Allen	1974	52	WAS2	14	10	4	0	0.714	1	0	1	0.0
20	ALLEGE0	George	Allen	1975	53	WAS2	14	8	6	0	0.571	0	0	0	0.0

Coaches table page, showing data preview and query options

In addition to normal users, our web application also features administrative users who have an enhanced amount of control over registered users. Administrators are indicated in the users table by a **1** in the **`admin`** attribute. These administrators have access to a hidden configuration page allowing them to promote a user to administrator, demote an administrator to a user, and delete users by removing them from the **`users`** table.



Administrator configuration page

Application Logic

The application logic of our program is largely handled by a Flask application written in Python. This program defines functions based on the page that is being navigated to and the HTTP methods that are being used (**GET** and **POST**). Typically, these methods fall into one of two categories: rendering HTML pages when a user requests them or performing back-end application logic for communicating with the database, performing user authentication, manipulating session cookies, etc.

Database

As stated before, the majority of the data in our application was aggregated from different websites. Some of the data was already in CSV format, making it easily accessible. However, other data had to be scraped from pages before being converted to a usable form. To assemble our data prior to migrating to the database instance, we used Google Sheets. This allowed us to store and manipulate the data before downloading it as CSV files. Once downloaded, we could then import these CSV files into the MySQL database. Our application interacts directly with the database stored on AWS, not with any local files or SQL servers.

Tables

Coaches

(*Coach ID*: VARCHAR(7), *First Name*: TEXT, *Last Name*: TEXT, *Year*: YEAR, *Age*: INT, *Team ID*: TEXT, *Season Games Played*: INT, *Season Wins*: INT, *Season Losses*: INT, *Season Ties*: INT, *Season Win-Loss*: DOUBLE, *Playoff Games Played*: BIGINT(20), *Playoff Wins*: BIGINT(20), *Playoff Losses*: BIGINT(20), *Playoff Win-Loss*: DOUBLE)

Contains information about coaches in the NFL; each entry represents a single coach during a particular season of the NFL.

Franchises

(*Franchise ID*: VARCHAR(3), *Name*: TEXT, *Year (Start)*: YEAR, *Year (End)*: YEAR, *Active?*: VARCHAR(3), *Wins*: INT, *Losses*: INT, *Ties*: INT, *Win-Loss*: DOUBLE, *Top Passer*: TEXT, *Top Rusher*: TEXT, *Top Receiver*: TEXT, *Top Coach*: TEXT, *Years in Playoffs*: INT *Championship Wins*: INT, *Super Bowl Wins*: INT, *Conference Wins*: INT, *Division Wins*: INT)

Contains information about NFL franchises. Franchises are closely related to teams, except teams are a specification of a franchise. Franchises may have any number of teams over the course of their existence in the NFL. While team names and locations may vary, the corresponding franchise always stays the same.

Games

(*Game ID*: VARCHAR(12), *Date*: TEXT, *Home Team ID*: TEXT, *Home Score*: INT, *Away Team ID*: TEXT, *Away Score*: INT, *Temperature (F)*: INT, *Humidity*: DOUBLE, *Wind Speed (MPH)*: INT)

Contains information about every NFL game played, including the home and away teams, final score, and weather.

Players

(*First Name*: TEXT, *Last Name*: TEXT, *Player ID*: VARCHAR(8), *Birth City*: TEXT, *Birth State*: TEXT, *Birth Country*: TEXT, *Birth Date*: TEXT, *College*: TEXT, *Position*: TEXT, *Height (FT)*: INT, *Height (IN)*: INT, *Weight (LBS)*: INT)

Contains personal information about NFL players. While we would have liked to have entries for each player during each season (like the coaches table), the data was simply not available.

Super Bowls

(*Date*: DATE, *Roman Numeral*: TEXT, *Number*: INT, *Winning Team ID*: TEXT, *Winning Score*: INT, *Losing Team ID*: TEXT, *Losing Score*: INT, *MVP*: TEXT, *MVP Player ID*: TEXT , *Stadium*: TEXT, *City*: TEXT, *State*: TEXT)

Contains information about every Super Bowl played from 1967 to 2017.

Teams

(*Team ID*: VARCHAR(4), *Team Name*: TEXT, *Franchise ID*: TEXT, *League*: TEXT, *Year (Start)*: YEAR, *Year (End)*: YEAR, *Active?*: VARCHAR(3), *Wins*: INT, *Losses:* INT, *Ties*: INT, *Win-Loss*: DOUBLE, *Years in Playoffs*: INT *Championship Wins*: INT, *Super Bowl Wins*: INT, *Conference Wins*: INT, *Division Wins*: INT)

Contains information about every NFL Team from 1920 to 2017. A team belongs to a franchise.

Users

(*username*: TEXT, *password*: LONGBLOB, *last_login*: TIMESTAMP, *admin*: TINYINT(1))

Contains the username and password of users registered on the web application as well as whether or not they are an administrator.

Stored Procedures

check_user()

*Performs a check of the **users** table to see if a given user exists. Used during authentication.*

```
PROCEDURE `check_user`(IN username VARCHAR(12))
BEGIN
	IF(SELECT EXISTS(SELECT * FROM `users` WHERE `users`.`username` =
		username)) THEN
		SELECT 'TRUE';
	ELSE
		SELECT 'FALSE';
	END IF;
END
```

---

create_user()
*Creates a new entry in the **users** table given a username and password hash. This procedure checks if the user exists already before insertion.*

```
PROCEDURE `create_user`(IN username VARCHAR(12), IN password LONGBLOB)
BEGIN
	IF(SELECT EXISTS(SELECT * FROM `users` WHERE `users`.`username` =
		username))
	THEN
		SELECT 'FALSE';
	ELSE
		INSERT INTO `users` VALUES(username, password, now(), 0);
		SELECT 'TRUE';
	END IF;
END
```

---

delete_user()
*Removes a given user from the **users** table. Used on the admin configuration page.*

```
PROCEDURE `delete_user`(IN username VARCHAR(12))
BEGIN
	DELETE FROM `users` WHERE `users`.`username` = username;
END
```

---

get_admins()
Retrieves the usernames and a formatted last login timestamp for all administrators. Used to populate admin table on configuration page.

```
PROCEDURE `get_admins`()
BEGIN
	SELECT `username`, DATE_FORMAT(`last_login`, '%W, %M %D, %Y %r')
		FROM `users` WHERE `admin` = 1;
END
```

---

get_all_usernames()
Retrieves the usernames of all users.

```
PROCEDURE `get_all_usernames`()
BEGIN
	SELECT `username` FROM `users` ORDER BY `username`;
END
```

---

get_headers()
Returns the column/attribute names for a given table. This is used when rendering the headers for a table on a page.

```
PROCEDURE `get_headers`(IN tablename VARCHAR(45))
BEGIN
	SELECT `COLUMN_NAME` FROM `INFORMATION_SCHEMA`.`COLUMNS` WHERE
		`TABLE_NAME`= tablename;
END
```

---

get_password()
*Retrieves the password hash of a given user in the **users** table. Used to provide authentication.*

```
PROCEDURE `get_password`(IN username VARCHAR(45))
BEGIN
	SELECT `password` FROM `users` WHERE `users`.`username` =
		username LIMIT 1;
END
```

---

get_users()
Fetches the usernames and a formatted last login timestamp for all users.

```
PROCEDURE `get_users`()
BEGIN
	SELECT `username`, DATE_FORMAT(`last_login`, '%W, %M %D, %Y %r')
		FROM `users` WHERE `admin` = 0;
END
```

---

login()
Updates a user's last log-in timestamp and returns whether or not they are an admin.

```
PROCEDURE `login`(IN username VARCHAR(12))
BEGIN
	UPDATE `users` SET `users`.`last_login` = now() WHERE
		`users`.`username` = username;
	SELECT `admin` FROM `users` WHERE `users`.`username` = username;
END
```

---

preview_table()
Returns the first 100 rows of a given table. This procedure is used to display a table when no query has been performed.

```
PROCEDURE `preview_table`(IN tablename VARCHAR(45))
BEGIN
	SET @SQL = CONCAT('SELECT * FROM `', tablename, '` LIMIT 100');
	PREPARE statement FROM @SQL;
	EXECUTE statement;
	DEALLOCATE PREPARE statement;
END
```

---

set_user()
Modifies a user's administrative privileges. Used to promote and demote users.

```
PROCEDURE `set_user`(IN username VARCHAR(12), IN admin TINYINT(1))
BEGIN
	IF(SELECT EXISTS(SELECT * FROM `users` WHERE `users`.`username` =
		username))
	THEN
		UPDATE `users` SET `users`.`admin` = admin WHERE
			`users`.`username` = username;
		SELECT 'TRUE';
	ELSE
		SELECT 'FALSE';
	END IF;
END
```

---

query()

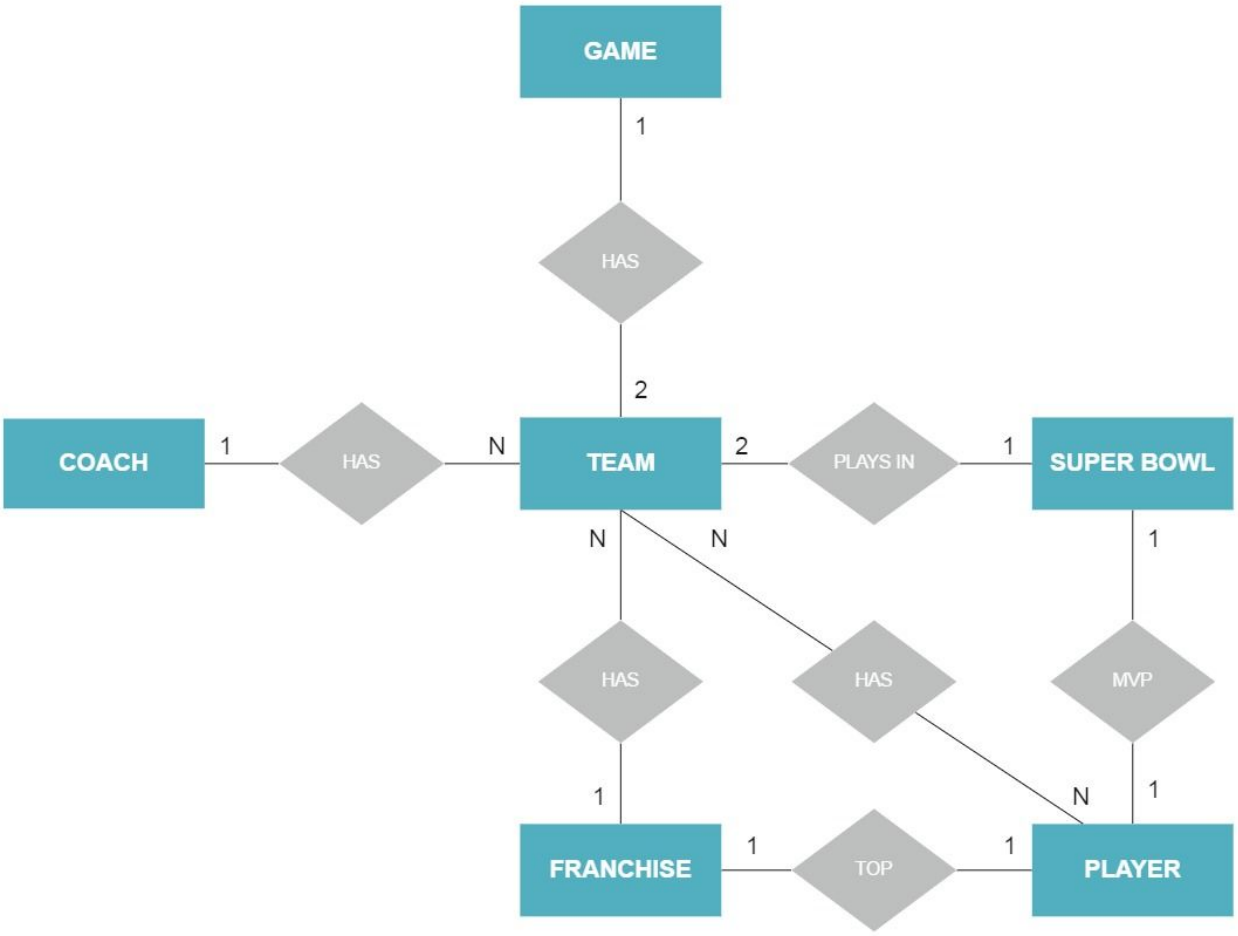
Takes the parameters given from the web application, constructs an SQL query, and then executes it - returning the results. This procedure is used for every table query.

```
PROCEDURE `query`(IN tablename VARCHAR(45), IN attribute VARCHAR(45),
IN operator VARCHAR(2), IN input VARCHAR(45), IN sortby VARCHAR(45),
IN sorttype VARCHAR(45), IN entries INT)
BEGIN
	SET @SQL = CONCAT('SELECT * FROM `', tablename, '`');
	IF attribute != '' AND operator != '' AND input != ''
	THEN
		SET @SQL = CONCAT(@SQL, ' WHERE `', attribute, '` ',
		operator, ' \'', input, '\'');
	END IF;

	IF sortby != ''
	THEN
		SET @SQL = CONCAT(@SQL, ' ORDER BY `', sortby, '` ',
		sorttype);
	END IF;

	IF entries != 0
	THEN
		SET @SQL = CONCAT(@SQL, ' LIMIT ', entries);
	END IF;

	PREPARE statement FROM @SQL;
	EXECUTE statement;
	DEALLOCATE PREPARE statement;
END
```



Entity relationship diagram; attributes not shown.