# ForFit

**Never Quit Again**

Illya Starikov, Jason Young, Claire Trebing

May 2, 2016

# Contents

# Part I.

# The Database Design Manual

# 1. Problem Statement

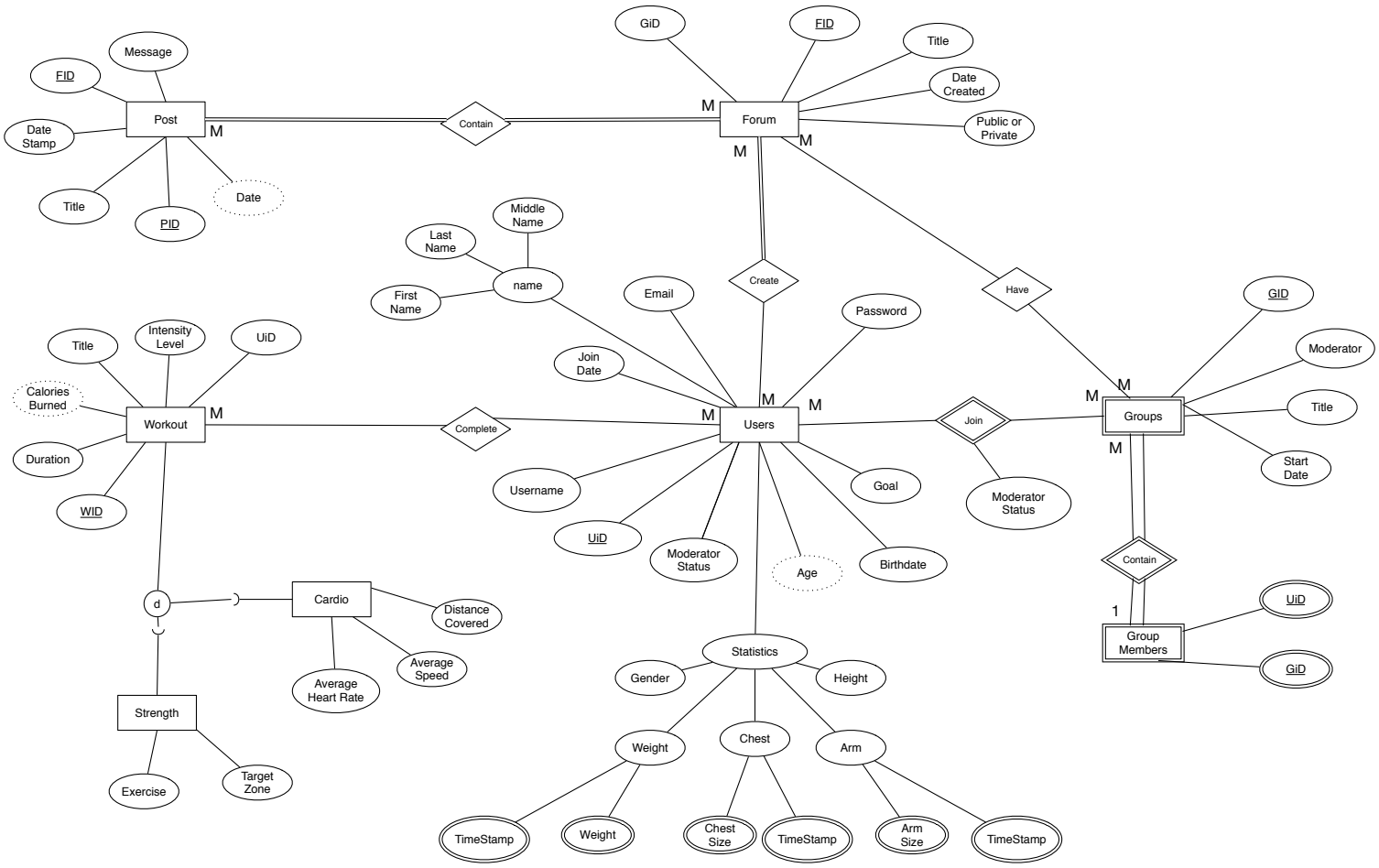
2015 marked a record year for Americans regularly exercising, hitting just over 55%. Keeping that momentum is difficult. As blue-collar jobs continue to decline, it has been more important than ever to keep a weekly regimen of healthy eating and exercising.

Living in the most interconnected generation poses quite a viable idea for social networking: healthy living. We can build a social network that connects users to friends, peers and family to make an online community of healthy living.

Our database will be essential because it will unite something so mundane and uninteresting with familiar faces. This will make exercise more enjoyable and offer a group to hold you accountable. We can shape an entire generation by having motivation a click away.

This database will consist of premade workouts, groups, and reminder emails to help you stay on top of your fitness plan.

# 2. Conceptual Database Design


GiD
FID
Title
Date Created
Public or Private
Message
FID
Post
Date Stamp
Contain
Forum
M
M
M
M
Title
PID
Date
Middle Name
Last Name
name
First Name
Create
Have
Email
Password
Join Date
GID
Moderator
Title
Start Date
Intensity Level
UiD
Title
Calories Burned
Workout
Complete
Users
Join
Groups
Duration
WID
Username
UiD
Moderator Status
Goal
Birthdate
Age
Moderator Status
Contain
1
UiD
Group Members
GiD
d
Cardio
Distance Covered
Average Speed
Average Heart Rate
Statistics
Gender
Height
Weight
Chest
Arm
Strength
Exercise
Target Zone
TimeStamp
Weight
Chest Size
TimeStamp
Arm Size
TimeStamp

# 3. Logical Database Design

## 3.1. Relational Set

Please note primary keys are signified by $^{\mathbf{PK}}$ and foreign keys are signified by $^{FK}$.

**Post**

| PiD $^{\mathbf{PK}}$ | FiD $^{FK}$ | Message | DateStamp | Title | Date |
|---|---|---|---|---|---|

**Forum**

| FiD $^{\mathbf{PK}}$ | Title | DateCreated | PublicOrPrivate | GiD $^{FK}$ |
|---|---|---|---|---|

**Groups**

| GiD $^{\mathbf{PK}}$ | Moderator $^{FK}$ | Title | StartDate |
|---|---|---|---|

**Workouts**

| WiD $^{\mathbf{PK}}$ | Duration | Title | IntensityLevel | CaloriesBurned | UiD $^{FK}$ |
|---|---|---|---|---|---|

**Strength**

| WiD $^{\mathbf{PK}}$ | Duration | Title | IntensityLevel | CaloriesBurned | UiD $^{FK}$ | Target Zone |
|---|---|---|---|---|---|---|

**Cardio**

| WiD $^{\mathbf{PK}}$ | Duration | Title | IntensityLevel | CaloriesBurned | UiD $^{FK}$ | AverageHeartRate |
|---|---|---|---|---|---|---|
| AverageSpeed | DistanceCovered | | | | | |

**Users**

| UiD $^{\mathbf{PK}}$ | Username | Height | Birthdate | Goal | Password | JoinDate | Gender |
|---|---|---|---|---|---|---|---|
| FirstName | MiddleName | LastName | | | | | |

**Weight**

| UiD $^{\mathbf{PK, FK}}$ | TimeStamp $^{\mathbf{PK}}$ | Weight |
|---|---|---|

**Arm Size**

| UiD $^{\mathbf{PK, FK}}$ | TimeStamp $^{\mathbf{PK}}$ | Arm Size |
|---|---|---|

**Chest Size**

| UiD $^{\mathbf{PK, FK}}$ | TimeStamp $^{\mathbf{PK}}$ | ChestSize |
|---|---|---|

**Group Members**

| UiD **PK** | GiD *FK* | ModeratorStatus |
|---|---|---|

## 3.2. Summary Table

| Attribute | Data Type | Constraints | Meaning |
|---|---|---|---|
| **Message** | varchar | 500 characters | Contents of a post. |
| **FID** | int | unique | Forum ID. |
| **DateStamp** | timestamp | timestamp of creation | Timestamp of post was creation. |
| **Title** | char | 35 characters | Title of a post. |
| **PID** | int | unique | Post ID. |
| **Date** | timestamp | date of creation | Timestamp of post creation. |
| **Title** | char | 35 characters | Title of a forum. |
| **DateCreated** | timestamp | timestamp of creation | Timestamp of forum creation. |
| **PublicOrPrivate** | boolean | - | Is forum public. |
| **GID** | int | unique | Group ID. |
| **Moderator** | char | must match a username | Group owner/moderator. |
| **Title** | char | 35 characters | Title of the group. |
| **StartDate** | timestamp | timestamp of creation | Timestamp of group creation. |
| **IntensityLevel** | int | - | Enumerated value, code for different options. |
| **Title** | char | 35 characters | Title of the workout. |
| **CaloriesBurned** | int | num $> 0$ | How many calories burned during exercise. |
| **Duration** | timestamp | num $> 0$ | Length of exercise. |
| **WID** | int | unique | Workout ID. |
| **Exercise** | char | 35 characters | Exercise name. |
| **TargetZone** | char | 35 characters | Area exercise is intended to workout. |
| **AverageHeartRate** | int | num $> 0$ | Average heart rate recorded during exercise. |
| **AverageSpeed** | int | num $> 0$ | Average speed of cardio exercise. |
| **DistanceCovered** | int | num $> 0$ | Distance covered during cardio exercise. |
| **Password** | char | - | Password of user. |
| **JoinDate** | timestamp | timestamp of creation | Timestamp of users sign up. |
| **Username** | char | unique | Users username, used for sign in. |
| **ModeratorStatus** | boolean | - | Is a moderator. |
| **Birthdate** | date | - | Date of users birth. |
| **Goal** | varchar | 500 characters | Users intended workout goals. |
| **Gender** | char | 1 character, M or F | Users gender, male(M) or female(F). |
| **Weight** | int | num $> 0$ | Users weight at a certain date. |
| **ChestSize** | int | num $> 0$ | Users chest size at a certain date. |
| **ArmSize** | int | num $> 0$ | Users arm size at a certain date. |
| **Height** | int | num $> 0$ | Users height at a certain date. |
| **UID** | int | unique | Users ID. |
| **TimeStamp** | date | unique | Date of users weight mesurement. |
| **TimeStamp** | date | unique | Date of users chest measurement. |
| **TimeStamp** | date | unique | Date of users arm measurement |

# 4. Application Program Design

## 4.1. Create a New User

This function creates a new user, accessing only the `user` table.

| | |
|---|---|
| Input | `Username, Height, Birthdate, Goal, Password, Gender`. |
| Steps | 1. Check to see if the `username` is available. If available, proceed. If not, display appropriate message to notify the user.<br>2. Insert appropriate information to the User table. (Username to `username`, height to `height`)<br>3. Generate a user ID, assign it to the `UiD` attribute.<br>4. Take a time stamp, assign it to the `JoinDate` attribute.<br>5. Calculate the age based on the `BirthDate` attribute. |
| Output | A new `User` entity will be inserted into the table, with appropriate data into proper columns (along with computed properties and derived properties). |
| Assumptions | `Username, Height, Birthdate, Goal, Password, Gender` are all correct (this will be validated in the sign up form). |

## 4.2. Delete A Group

This function deletes a group by updating information in `Forum`, and removing the `Group` and `Group Members` tables.

| | |
|---|---|
| Input | The Group ID (`GiD`) that is to be deleted. |
| Steps | 1. Check to see if the request is made by the moderator via the `Moderator` column in the `Groups` table. If true, approve the request. If not, cancel the request and notify the user.<br>2. Remove the row that has a matching `GiD` that was provided for deletion in the `Groups` table.<br>3. Query the `Group` Members table, removing any row that match the GiD provided for deletion.<br>4. Query the `Forum` table for any matching Group Ids (`GiD`), setting the `GiD` to `null` if matching. |
| Output | The groups are deleted, the group members within that group are deleted, and any reference to the group is deallocated. |
| Assumptions | None. |

## 4.3. Modifying User Statistics

Our user statistics have the ability to fluctuate. We would like to accommodate for this fluctuation by allowing users to update their respective statics; specifically, we would like to let users update `Username, Height, Goal, Password, Gender` in the `Users` table.

| | |
|---|---|
| Input | The specific attribute(s) of the set `Username, Height, Goal, Password, Gender` that would like to be updated with the new value. |
| Steps | 1. Ensure the data is valid (e.g. is not `null` when applicable, in the proper domain). If it is valid, continue. If not, prompt the user with an error message and try again.<br>2. Modify the attribute to reflect the new value.<br>3. Repeat for any additional attributes provided. |
| Output | The attribute(s) should now reflect the new value provided. |
| Assumptions | The data is within a proper range (will not overflow). |

## 4.4. Query Other Users

This function allows for users to query other users; this can be done via `Username` or `FirstName, MiddleName`, and `LastName` from the `Users` table.

| | |
|---|---|
| Input | Either a `Username` xor any subset of `FirstName, MiddleName`, or `LastName`. |
| Steps | 1. Check to see if input is valid. If is, proceed. If not, display error message to the user.<br>2. Query the `Users` table to see if the user exists. If the user exists, proceed. If not, display appropriate message to the user.<br>3. Project the profile. |
| Output | Either the search user will be projected or an error message is the user does not exist. |
| Assumptions | The first, middle and last name are all provided. The names are unique (solely for the testing purposes). |

## 4.5. User Leaderboards

Generate the leaderboard based on the workouts accomplished; specifically aggregating data from the `Strength` and `Cardio` table. *Note this is the function that requires multiple tables.*

| | |
|---|---|
| Input | None. |
| Steps | 1. Merge the `User` and `Workouts` table, call the new table `Merged`.<br>2. Add up the total duration (call the new property `TotalDuration`) in the `Merged` table based on the `UiD` attribute, making a new table named `Sums`.<br>3. Sort the `Sums` by the `TotalDuration` attribute.<br>4. Display the top 10 on the sorted `Sums` table to the user.<br>5. Display the user their current rank. |
| Output | An eleven-row table displaying the top 10 leaderboards and the users current rank. |
| Assumptions | There is a bare minimum of eleven users. |

# Part II.

# User Manual

# 5. A Brief Introduction To ForFit

ForFit is a new way to workout. Over the past year, just about 55% of Americans exercised regularly. ForFit is a way to connect you to your friend, family, and other exercise enthusiasts in your area. These exercises can range from traditional bike riding and swimming to newer forms of exercise such as parkour and yoga. ForFit encourages group activity and connection so that workouts are never dull and so that users will be continually motivated and pushed toward their workout goals.

# 6. How It Works

ForFit has two main components. The first part is the users themselves. Users sign up for the service and then begin recording their workouts. Workouts can be logged as many times as liked, from once a month to several times a day. The more workouts logged, the better data and the more progress toward your exercise goals.

Users can view other users and the workouts they've done. This leads into the second component of ForFit, which is the encouragement of community. Users can create groups and join existing groups. These groups can help users focus on specific areas. For example, a user wanting to run a marathon might join a group of other runners training for a marathon. This way the user can keep in touch and stay motivated through the group and also have a resource to ask questions or get other workouts.

Groups allow users to make forums, for in-depth discussion, and posts, for quick updates. These groups are the main area of community creation that ForFit focuses on in the design. Users can find other users and join groups with users they have been exercising with before. This creates a network of community interaction.

# 7. Test Procedure

This database was well tested before it reached you, the user. This section will explain how the test database was created and what tests the database went through before completion. The mock data was created using C++ code to randomly generate data. Data from online was used to create First Names, Last Names, Usernames, and Group Names as well as to populate the workout subclasses in strength and cardio. Using C++ code to randomly select and combine information, creating unique data. This mock data was then put through a series of tests to make sure the functions would work properly. Each function was tested over 500 times to error or improper analysis of the data. After confirming that these functions were working properly, they were implemented into the database.

# 8. Functions

## 8.1. Delete Group

Delete Group allows you to remove a group from the database. Once a group has been deleted, you can no longer make forums or posts in this group.

### Parameters

**groupID** The `GiD` of the group that is to be deleted.

### Results

The group who's ID matches the given `groupID` will be deleted. If the group does not exist, nothing is displayed.

```
rMBP:~ postgres$ '/Applications/Postgres.app/Contents/Versions/9.5/bin'/psql -p5432
psql (9.5.2)
Type "help" for help.

postgres=# select delete_group(42);
 delete_group
--------------
(0 rows)

postgres=#
```

## 8.2. Modify User

Modify User allows users to change certain values in the user's data by providing the user name. Users can change their username, height, goal, password, and gender.

### Parameters

**userID** The `UiD` of the user to who's statistics that will be modified. If this is not provided, there is no way of changing the user.

**newUsername** The new username property of the user. If this statistic will not change, pass a `'0'` to the function. Otherwise, proceed with changing.

**newHeightFeet** See above.

**newHeightInches** See above.

**newGoal** See above.

**newPassword** See above.

**newGender** See above.

### Results

For all attributes that do not have the value of `'0'`, the value of said attribute will be updated (provided it upholds all constraint).

```
rMBP:~ postgres$ '/Applications/Postgres.app/Contents/Versions/9.5/bin'/psql -p5432
psql (9.5.2)
Type "help" for help.

postgres=# select modify_user(1, 'IllyaStarikov', 0, 0, 42, '0', 'F');
 modify_user
-------------
(0 rows)

postgres=# select * from USERS where UiD = 1;
 uid |   username    |          email           | heightfeet | heightinches | birthdate  | goal |   password    |  joindate  | gender | fname | minit |  lname
-----+---------------+--------------------------+------------+--------------+------------+------+---------------+------------+--------+-------+-------+----------
   1 | IllyaStarikov | CasieScarlett1@gmail.com |          6 |            2 | 1967-03-31 |   42 | EtZEdlnfNiBAI | 1999-12-31 | F      | Casie | M     | Scarlett
(1 row)
```

## 8.3. New User

New User creates new users in the database. The user must input a username, email, height, birthdate, goal, and password. If any of those are not provided, an error message will be generated.

### Parameters

**id, username, email, heightFeet, heightInches, birthdate, goal, password** All the attributes that can be inserted into user table.

**joinDate, gender, firstName, mInitial, lastName** See above.

### Results

Provided all the attributes meet database constraints, a new user will be inserted into the database. If a constraint is not meant, there will be a message outputting the error and the insert will be rejected.

```
postgres=# select new_user(404, 'IllyaStarikov', 'IllyaStarikov@iCloud.com', 5, 10, '07-20-1994', 165, '42', '07-21-1994', 'M', 'Illya', 'A', 'Starikov');
 new_user
----------
(0 rows)

postgres=# select * from USERS where UiD = 404;
 uid |   username    |          email           | heightfeet | heightinches | birthdate  | goal | password |  joindate  | gender | fname | minit |  lname
-----+---------------+--------------------------+------------+--------------+------------+------+----------+------------+--------+-------+-------+----------
 404 | IllyaStarikov | IllyaStarikov@iCloud.com |          5 |           10 | 1994-07-20 |  165 | 42       | 1994-07-21 | M      | Illya | A     | Starikov
(1 row)
```

## 8.4. Query User

Query User allows users to find others by inputting a first and last name. If a user does not exist than the results are empty.

### Parameters

**firstName, lastName** The first and last name of the user to be queried.

### Results

If the user exists, the column of the user will be returned. If the user does not exist, not such column will be returned.

```
rMBP:~ postgres$ '/Applications/Postgres.app/Contents/Versions/9.5/bin'/psql -p5432
psql (9.5.2)
Type "help" for help.

postgres=# select query_user('Illya', 'Starikov');
                                                                                                  query_user
-------------------------------------------------------------------------------------------------------------------------------------------
 (117,IllyaStarikov,IllyaStarikov@iCloud.com,5,10,1994-07-20,165,42,1994-07-21,M,Illya,A,Starikov)
(1 row)
```

## 8.5. Total Users

Total Users returns how many users are in the whole database. In the future, this function will give a user count for an individual group.

### Parameters

None.

### Results

Returns the total count of users, a maintenance function used to compare to the total count in the database.

```
rMBP:~ postgres$ '/Applications/Postgres.app/Contents/Versions/9.5/bin'/psql -p5432
psql (9.5.2)
Type "help" for help.

postgres=# select total_users();
 total_users
-------------
          52
(1 row)
```

## 8.6. User Leaderboards

User Leaderboards gives you a top 10 user count. This is calculated by a count of workouts done by a user. This count includes all the users in a database and is not specific to a group.

### Parameters

None.

### Results

Returns the top 10 users (calculated via count of workouts done) in the database.

```
rMBP:~ postgres$ '/Applications/Postgres.app/Contents/Versions/9.5/bin'/psql -p5432
psql (9.5.2)
Type "help" for help.

postgres=# select user_leaderboards();
    user_leaderboards
-----------------------
 (Gregory,Patty,6)
 (Agueda,Heather,4)
 (Bertram,Gerardoy,4)
 (Gay,Arthury,4)
 (Garry,Shanony,4)
 (Brad,Lana,4)
 (Francisco,Bruce,4)
 (Edgar,Dennisy,4)
 (Alvina,Carter,4)
 (Darlene,Salvatore,3)
(10 rows)
```

# 9. Shell

We know doing tasks such as creating the database, generating a new set of random inputs, or even running all the function files to makes sure all of the function are created can tedious. Forunatly, we have a solution — that solution is Shell scripting.

We quickly realized that repetitively running the same commands[1] could get tedious, so we have provided multiple shell scripts that automate most of the monotonous tasks.

To run a shell script, first you must give it privileges to be able to run `Bash` commands. This is accomplished by `chmod -x /path/to/file.sh`. This give read/write access to the shell script, which then allows the shell script to be run via `./path/to/file.sh`. This then allows the user to do tasks like setting up the entirety of the database (which spans multiple files!) one file path away.

Below are the shell scripts we provide.

## 9.1. Create Database

A lot of testing requires the use of setting up the database from scratch. Unfortunately, this usually means creating the database scheme, followed by inserting the data, followed by inserting the functions, and so forth. Although an easy fix would be to have all the code in a single file, this would combine concerns. We would rather have a lot of function specific files apposed to one giant file. Our solution: `./create_database.sh`

The create database script works like so:

1. Create the database schema by `create`ing all the tables and `modify`ing all for foreign key constraints.
2. Insert data, which is artifically generated (as will be mentioned below).
3. Go through the *entirety* of the functions directory, and running each function `sql` script to ensure all functions are inserted into the database.

This allows for easily bringing a database back up after it has been reset.

## 9.2. Random Input

Let face it, every social network occasionally needs some help. That is why we have a shell script that executes our C++ codebase to auto-populate the table. The shell script runs like so:

[1] Like running `'/Applications/Postgres.app/Contents/Versions/9.5/bin'/psql -p5432 -f` to create every file!

1. Change to the directory of the C++ codebase.
2. Run the makefile
3. Run the executable, pipe that to an output file.
4. Move the shell script to the directory of all the other `sql` files.

This makes for easily produced, new data.

## 9.3. Drop Database

Drop database simply runs the `sql` script to drop all tables in a cascade fashion. It's a simple script but is much quicker than manually running the command to drop the database.