6.12 Final Report

S2G3: Taylor Goldman, Brandon Kinnick, Aidan Matthews

Table of Contents

Executive Summary

This final report document begins with a description of our initial problem statement for the project as well as the specific solution we implemented to solve that problem. It also contains major challenges that we faced while implementing the solution, as well as the overall design for our database. In addition to this, we perform an analysis of the results implemented in our solution which includes both strengths and weaknesses of the process. An appendix containing some of our design components, as well as an index and glossary are included at the end of this document.

Introduction

This document is the final report for our Gym Management database project developed by team S2G3, consisting of Taylor Goldman, Brandon Kinnick, and Aidan Matthews. The purpose of this report is to discuss the effectiveness of the solution to our problem statement that has been outlined in previous documents. We will be revisiting items mentioned in the security analysis and final problem statement and will be providing direct feedback on the strengths and weaknesses of our approach to solving the problem.

Problem Description

The local gym has tons of members, trainers, and programs that have data which need to be kept track of for the gym to continue running smoothly. For example, there are several members paying for memberships, many different trainers working with different members, and many programs for members to be a part of and for trainers to run. Most gyms don't have great systems to keep track of all this data. Our proposed solution is composed of a website that members and employees of the gym can use to access their personal data as well as information about available programs, trainers, and more in hopes to provide a simpler and more centralized platform for people of the local gym to use. This can also make it easier for the gym when they want to create new programs based on data from their members and trainers, and a place for employees of the gym to keep track of members membership billing information which they couldn't do previously.

Expected Functionality

1. Ability to track members of the gym
    a. Personal information
    b. Billing information
    c. Membership information
    d. Trainers and programs involved in
2. Ability to track different trainers

    a. Personal information
    b. Direct deposit information
    c. Programs they teach
    d. Members they train
3. Ability to keep track of different types of programs
    a. Members signed up for the program
    b. Trainers running it
    c. Total open spots left for the program
4. Ability for users to login
5. Ability for the Gym Administrator to create, update, and delete stored data on Trainers, Programs, and Members
6. Limits the access to certain data depending on which user logs in (member vs employee)

Solution Description

The solution decided upon by our team was to develop a web application using HTML, CSS, JavaScript, and Node.js and Express to serve it. We also used a Microsoft SQL Server Management Studio 18 database to store all the information related to the local gym.

Front-End Discussion

The front-end is a simple to use web application and was programmed using HTMl, CSS, and JavaScript. We used Bootstrap for most of our styling choices. To make sure that the site was user friendly for members, trainers, and employees of the local gym we wanted a simple user interface with minimal buttons and drop down menus for users to use to be able to enter and find data. We also made use of tables which display on many of the pages which make it easy for users to be able to view data as well as know which row of data they are editing or deleting. There are separate views on the website depending on if a member, trainer, or administrator is logged in. The member view contains their membership information, billing information, training sessions they are a part of, and programs. The trainer view contains their direct deposit information, training sessions that they teach, as well as programs. The admin view contains tables of data on members, trainers, and programs, which they can perform create, read, update, and delete operations on. The administrator view also allows them to make new memberships, add billing information to members' accounts, and add certain members and trainers to different programs.

Back-End Discussion

Our front-end connects to our Gym Management database on Microsoft SQL Server. We have 10 tables in our database that connect to our front-end for a user friendly display of our data, however the database itself cannot be accessed from the front-end website for security

reasons. These tables include Billing Info, Has Membership, In Program, Member, Membership, Person, Program, Teaches Program, Trainer, and Trains With. The database currently exists on our VM: gymmanagementvm.csse.rose-hulman.edu. Our database design section will have more information on our database.

Key Challenges

- **Challenge:** Connectivity with node.js and express
    - **Solution:** We learned how to host our SQL connection separately from the browser and sent fetch requests to the server to run our stored procedures from the database.
    - **Analysis:** This approach worked pretty well. We all had experience with node js and express before which made the initial learning steps easier. Once we got the hang of making fetch requests and building receiver functions on the server, it was a smooth process.
- **Challenge:** asynchronous callbacks
    - **Solution:** We refreshed our knowledge on asynchronous callbacks and used documentation to help us use them successfully.
    - **Analysis:** In particular, the use of .then() clauses allowed us to chain together procedures to use return values from prior actions. This was not ideal due to concurrency issues. Reading documentation allowed us to research passing a third argument into our sequel calls, a callback function to be executed upon successful executions of stored procedures.
- **Challenge:** Member, Trainer, and Admin Authentication
    - **Solution:** We learned about JWT tokens to be able to maintain a login session for a specific user rather than using global variables for user information.
    - **Analysis:** This approach worked very well. Specifically, the package we used to handle tokens allowed us to encrypt the important tokens that we used to block users from certain pages on the site they would not be allowed access to with their current authentication (i.e. members accessing trainer pages, trainers accessing admin pages, etc.).
- **Challenge:** password hashing
    - **Solution:** We found a new way to hash passwords that was different from the connectivity lab in class since we had a different front-end implementation. We used the bcrypt npm library which made this transition easier.
    - **Analysis:** In particular, the bcrypt npm library allowed easy password hashing and salting through use of nested functions that would use the outputted salted and hashed passwords for use in authentication storage.

- **Challenge: Complexity of stored procedures**
    - **Solution:** We used online resources and class slideshows to refresh and enhance our understanding of stored procedures. Most importantly, we made effective use of return values as well as a few constraints and rollbacks to ensure that no unexpected changes were made
    - **Analysis:** A more effective approach to this may have been to make more use of triggers. This would have allowed us to write only a few simple triggers on each table to avoid making changes that would adversely affect the database. Because we didn't each stored procedure had numerous checks within it that likely would not have been necessary with smarter use of triggers.

Database Design

The Entity Relationship Diagram and Relational Schema for our database design will be included in the appendix.

Security Measures

Security could possibly be breached if someone without the proper permissions got access to personal data that should be kept private. An example of this is if a member could see everyone's 3 billing information and now has access to people's financial information, which is a huge breach of privacy. Not only this, but trainers could also somehow get access of member's personal information which could be a problem. Therefore, for each specific user ID, you will only be able to view your own private information. Only the administrator of the gym will have access to all the members and trainers but still, only the user themselves will be able to access their personal information and edit it. Another breach of security is if someone got access to edit the database while they were not supposed to be able to. For example, if a member got access to edit someone else's billing information, then the data could be inaccurate and possibly cause referential integrity issues. We have made sure that this won't happen by having different views for members and trainers so that you can only view data that applies to you, rather than being able to view all information, let alone edit other people's information. Not only that, but important personal information will also not be stored in plain text to ensure security. Security could also be breached if someone that isn't a member or employee got into the database and could see data. This could be a problem because if they don't have a specific user ID and aren't marked as an employee or a member, then the database won't know what data they should be able to have access to and what should be private to them. This could cause privacy issues and data integrity problems if they had access to change the data. However, we have also made sure that this won't occur because the first page on the web application requires a user to login or

register a new account. Also, the passwords are hashed and not stored in plain text to make them more secure.

Integrity Constraints

The referential integrity constraints within the database are represented as arrow relationships in our Relational Schema that is in the appendix of this document. On update and delete operations, all operations will cascade. This keeps our database clean without having a bunch of null values for data that references something that has been deleted from the database. It will also be a better option than rejecting certain changes since many of our foreign keys reference ID values from other tables, which won't be edited by the users. Below is the entity integrity analysis for each table in our database.

DOMAIN INTEGRITY CONSTRAINTS:

- For entering billing information for a member, the credit card number, CVV, and expiration date fields cannot be empty.
- When adding a person to the database you must fill out both the first and last name fields.
- For a new program to be successfully created, there must be a description entered, the day of week, the time of the program, and the start date of the program.
- A program's start date must be less than or equal to the value of getdate() and the program's end date.
- New users must have a unique username.

Stored Procedures

STORED PROCEDURE NAME	STORED PROCEDURE PURPOSE
AddBillingInfo	Adds a member's new billing information into the BillingInfo table that can be accessed by that specific member.
AddInstructorToProgram	Adds the specified trainer as the instructor of the program specified in the procedure.
AddMember	Adds a new member to the Member table.
AddMemberToProgram	Adds a member to a program that is specified within the procedure.
AddPerson	This will add a new person to the User table in the database.
AddProgram	Adds a new program to the Program table which can be viewed by the users on the frontend.
AddTrainer	Adds a new trainer to the Trainer table.
AddTrainingSession	Adds a new training session with a member, trainer, and the time they trained for which the user and trainer can view.

CancelMembership	Cancels the membership for the specified member in the procedure.
CheckIfUsernameAvailable	Checks if the username entered to register with is unique to be able to use for a new account.
DeleteBillingInfo	Deletes specified billing information for the specified member from the BillingInfo table
DeleteMember	Deletes the specified member from the Member table.
DeletePerson	Deletes the specified person from the database.
DeleteProgram	Deletes the program from the Program table.
DeleteTrainer	Deletes the trainer from the Trainer table.
EditMember	Edits the specified member's personal information and adds the updated information into the Member table, replacing the old information.
EditPerson	Edits the specified person and puts their new information in the database, replacing the old information.
EditProgram	Edits the specified program and updates the program's information in the Program table.
EditTrainer	Edits the specified trainer's personal information and updates the trainer's information in the Trainer table.
GetActiveMembership	Selects the membership for the specified member where there is no end date, so the membership is active.
GetLoginID	Selects the login ID for a person in the User table.
GetMemberInfo	Selects personal information about the specified member from the Member table joined with the Person table.
GetMembersOfProgram	Selects members of the specified program.
GetProgramInfo	Selects information about the program specified in the procedure.
GetTrainerInfo	Selects personal information about the specified trainer from the Trainer table joined with the Person table.
Register	Inserts a new username and password salt and hash combination with the person's ID into the User table.
RegisterAsMember	Same as register stored procedure but specifically for a member.
RemoveMemberFromProgram	Deletes the specified member from the InProgram table for the specified program.
RemoveTrainerFromProgram	Deletes the specified trainer from the TeachesProgram table for the specified program.
RetrieveLoginInfo	Selects the password salt and hash and the UID from the Person table.
ShowAllMemberInformation	Shows the top 20 member's information in a table and has an option to filter by username.
ShowAllProgramInformation	Shows the top 20 programs in a table and has an option to filter by different things like program ID, description, day of week, total spots left, etc.
ShowAllTrainerInformation	Shows the top 20 trainer's information in a table and has an option to filter by username.

ShowMemberBillingInformation	Selects all information from the BillingInfo table for the specified member.
ShowMembershipsByMember	Selects all memberships from the Membership table for the specified member, in order from least to most recent.
StartMembership	Activates a membership for the specified user to the default tier and current date and inserts into the Membership table.

Design Analysis

Strengths

- The system has a high level of security since there is a different view for members, trainers, and administrators which makes sure users can only access their data based on their type of account.
- Appropriate referential and entity integrity constraints are in place.
- Most stored procedures use primary key values to compare and execute, which eliminates concern with referential integrity issues.

Weaknesses

- We have a lot of stored procedures, and some aren't used often due to cutting down the scope, so they are taking up a lot of file space within the database. We could have combined some of these stored procedures or used other methods such as triggers that could do something like a stored procedure but solve the same problem that multiple stored procedures can take on with one trigger to save space and simplify our solution.
- We could have implemented views to make our tables that are displayed on the front-end easier to create.

Appendix

Relational Schema

Person(UID, Username, PasswordSalt, PasswordHash, FName, LName, DOB)

Member(UID, isVIP)

Membership(MembershipID, MemberID, MembershipTier, MembershipStartDate, MembershipEndDate, ExpectedTotal, PaidTotal)

BillingInfo(MemberID, CreditCardNumber, CVV, ExpirationDate, NameOnCard, ZIPCode)

Trainer(UID, Salar, Specialty, AccountNum, RoutingNum)

TrainsWith(MemberID, TrainerID, TimeTrained)

Program(PID, Description, DayOfWeek, TimeOfMeet, ProgramStartDate, ProgramEndDate, TotalSpots)

InProgram(MemberID, ProgramID)

TeachesProgram(TrainerID, ProgramID)

FOREIGN KEYS

Member(UID) → Person(UID)

Trainer(UID) → Person(UID)

Membership(MemberID) → Member(UID)

BillingInfo(MemberID) → Member(UID)

TrainsWith(MemberID) → Member(UID)

TrainsWith(TrainerID) → Trainer(UID)

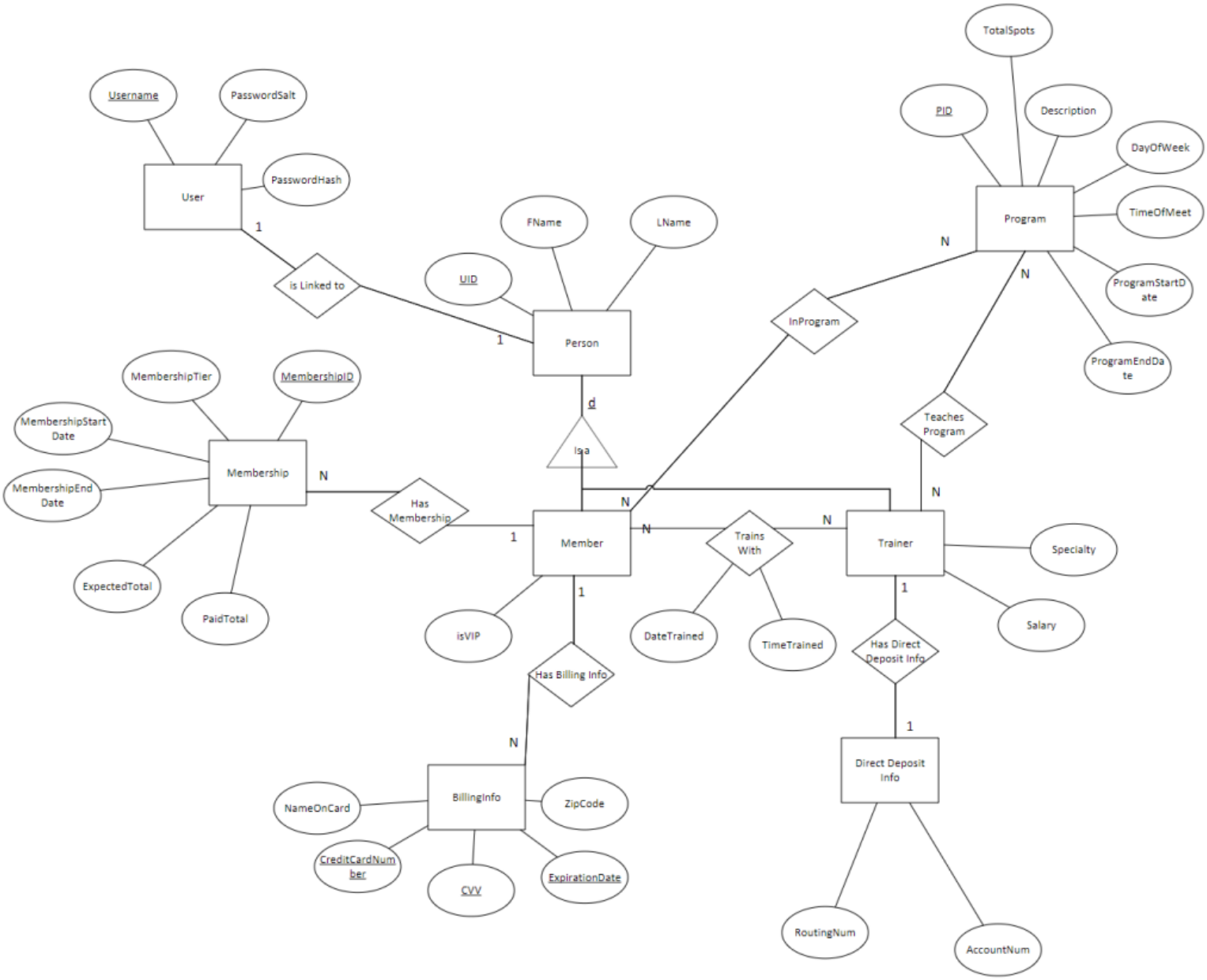
InProgram(MemberID) → Member(UID)

InProgram(ProgramID) → Program(PID)

TeachesProgram(TrainerID) → Trainer(UID)

TeachesProgram(ProgramID) → Program(PID)

Entity Relationship Diagram



Entity Relationship Diagram Explanation

The database is built from the existence of users when they register for an account, which is linked to a person so that they can store their personal information. Then, depending on if they are creating a trainer or member account, they will reference one of those objects and inherit those attributes for

either a trainer or member. The rest of the database revolves around the members and trainers within the database. Payment information is stored for both members and trainers with relationships to the Billing Info and Direct Deposit Info tables. Members also have memberships and can have multiple memberships. Programs are the other main table in the database and trainers and members have relationships with this table based on if they are in the program or if they teach the program.

Index

Glossary

SQL: a structured query language used for standard database manipulation

Stored Procedure: a set of SQL statements with an assigned name that are stored in our database

Entity Relationship Diagram: a way of illustrating the various relations and objects within a database in an abstract, easy to understand view

Relational Schema: the data that describes the structure of a database within a certain domain

Database: a structured set of data held in a computer, especially one that is accessible in various ways

Front end: the part of an application with which the user interacts directly

References

1. Team S2G3 Security and Data Integrity Analysis document, delivered on: 04/23/2022
2. Team S2G3 Final Problem Statement document, delivered on: 05/14/2022
3. Bootstrap 5.0 Documentation Introduction · Bootstrap v5.0 (getbootstrap.com)