SuperData

28.04.2024

Neal Shiyekar, Saidarsh Tukkadi, Luke Del Rio, Muhammad Ali, Aravindaswamy Anandakumar, Daixing Kuang
CS 4347 - Jalal Omer

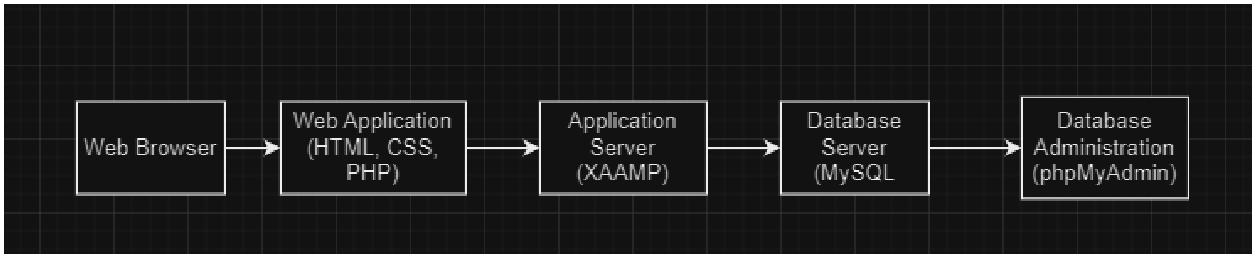
Table of Contents

Introduction

This idea is to track a hypothetical superhero organization based in Richardson to manage the heroes, their missions, villains, agents, etc. Having a database system is vital for the superhero organization because data is constantly changing. Moreover, we want to maintain security and confidentiality, scale horizontally/vertically to accommodate growth of the organization, and allow for concurrent users to access and query the database. Leaders of the organization need direct access to determine the success of their heroes and changes in crime rate. Target User: Agents of the heroes will need access to their hero’s statistics in order to make well-informed decisions for them. The organization's leaders/managers need access to the hero's data and performance, manage salaries of heroes, and hire/fire employees. The SuperAdministrator will maintain the database. Certain heroes based on seniority will have access to info about their team and secret missions.

System Requirements

Diagram:


Web Browser
Web Application (HTML, CSS, PHP)
Application Server (XAAMP)
Database Server (MySQL
Database Administration (phpMyAdmin)

Web Interface:

User-friendly, responsive design for desktop and mobile devices.

Secure forms for CRUD operations on Missions, Heroes, Villains, Agents, and Superpowers.

3. Functional Requirements

CRUD Operations:

Create, read, update, and delete information for Heroes, Villains, Missions, Agents, and Superpowers.

Search and Filtering:

Ability to search database records based on various criteria.

Advanced filtering options for complex queries.

Different access levels depending on user roles (e.g., admin, operator).

4. Non-Functional Requirements

Performance:

Quick response times for all operations.

Efficient handling of concurrent requests.

Reliability:

High availability of the system.

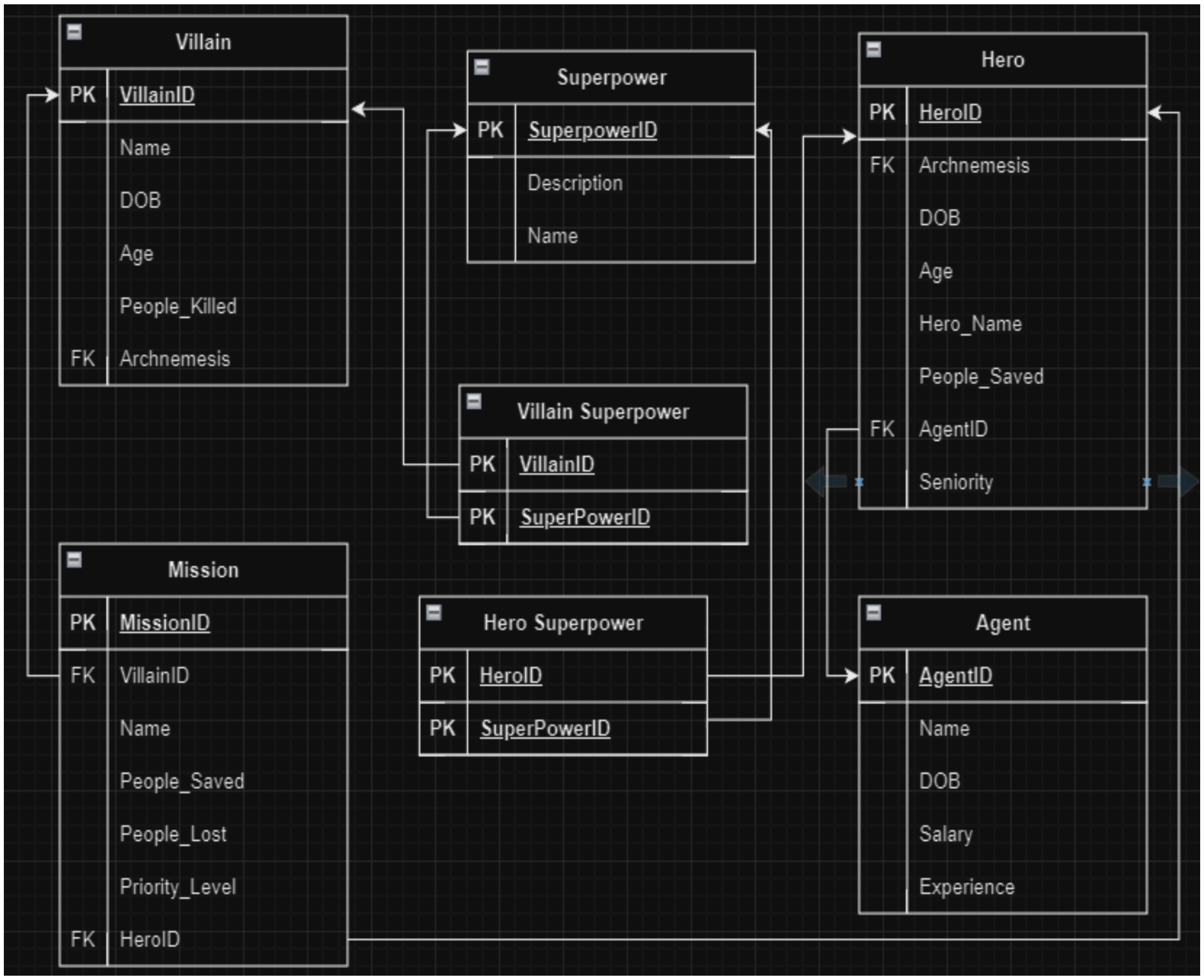
Robust error handling and data recovery mechanisms.

Security:

Data encryption both in transit and at rest.

Regular security audits and compliance with data protection regulations.

Conceptual Design



Database Schema

create.sql

```
CREATE SCHEMA IF NOT EXISTS 4347_ProjectDB_Final;
USE 4347_ProjectDB_Final;
CREATE TABLE IF NOT EXISTS Villain (
VillainID INT AUTO_INCREMENT PRIMARY KEY,
Name varchar(255),
Age INT,
DOB DATE,
People_Killed INT,
Archnemesis INT,
UNIQUE (VillainID, Archnemesis)
);

-- Alter Villain table to add foreign key constraint
ALTER TABLE Villain
ADD CONSTRAINT fk_villain_archnemesis
FOREIGN KEY (Archnemesis) REFERENCES Hero(HeroID)
ON DELETE SET NULL
ON UPDATE CASCADE;
CREATE TABLE IF NOT EXISTS Hero (
HeroID INT AUTO_INCREMENT PRIMARY KEY,
Name varchar(255),
Age INT,
DOB DATE,People_Saved INT,
Archnemesis INT,
AgentID INT default 0,
Seniority ENUM('Trainee', 'Junior Hero', 'Senior Hero', 'Lead Hero', 'Special Hero'),
CONSTRAINT fk_hero_villain FOREIGN KEY (Archnemesis) REFERENCES Villain(VillainID) ON
DELETE SET NULL ON UPDATE CASCADE,
CONSTRAINT fk_hero_agent FOREIGN KEY (AgentID) REFERENCES Agent(AgentID) ON DELETE
SET NULL ON UPDATE CASCADE,
UNIQUE (HeroID, Archnemesis)
);

CREATE TABLE IF NOT EXISTS Agent (
AgentID INT AUTO_INCREMENT PRIMARY KEY,
Age INT,
Salary DECIMAL(10, 2),
Experience INT,
Name varchar(255)
);

CREATE TABLE IF NOT EXISTS Leader (
LeaderID INT AUTO_INCREMENT PRIMARY KEY,
Name varchar(255),
DOB DATE,
Age INT,
Salary DECIMAL(10, 2),Experience INT,
Role varchar(255)
);

CREATE TABLE IF NOT EXISTS Mission (
MissionID INT AUTO_INCREMENT PRIMARY KEY,
Name VARCHAR(255),
HeroID INT,
People_Saved INT,
People_Lost INT,
VillainID INT,
Priority_Level ENUM('1', '2', '3', '4', '5'),
CONSTRAINT fk_mission_hero FOREIGN KEY (HeroID) REFERENCES Hero(HeroID) ON DELETE
CASCADE ON UPDATE CASCADE,
CONSTRAINT fk_mission_villain FOREIGN KEY (VillainID) REFERENCES Villain(VillainID) ON
DELETE CASCADE ON UPDATE CASCADE
);

CREATE TABLE IF NOT EXISTS Superpower (
SuperpowerID INT AUTO_INCREMENT PRIMARY KEY,
Description varchar(255),
Name VARCHAR(255)
);

CREATE TABLE IF NOT EXISTS Hero_Superpowers (
HeroID INT,
SuperpowerID INT,PRIMARY KEY (HeroID, SuperpowerID),
FOREIGN KEY (HeroID) REFERENCES Hero(HeroID)
ON DELETE CASCADE
ON UPDATE CASCADE,
FOREIGN KEY (SuperpowerID) REFERENCES Superpower(SuperpowerID)
ON DELETE CASCADE
ON UPDATE CASCADE
);

CREATE TABLE IF NOT EXISTS Villain_Superpowers (
VillainID INT,
SuperpowerID INT,
PRIMARY KEY (VillainID, SuperpowerID),
FOREIGN KEY (VillainID) REFERENCES Villain(VillainID)
ON DELETE CASCADE
ON UPDATE CASCADE,
FOREIGN KEY (SuperpowerID) REFERENCES Superpower(SuperpowerID)
ON DELETE CASCADE
ON UPDATE CASCADE
```

Functional Dependencies

The functional dependencies before the normal third form:
- Villain: VillainID -> DOB, People_Killed, Archnemesis, SuperpowerID
- Hero: HeroID -> DOB, People_Saved, Archnemesis, SuperpowerID, AgentID, Seniority
- Agent: AgentID -> DOB, Salary, Experience
- Mission: MissionID -> Name, HeroID, People_Saved, People_Lost, VillainID, Priority_Level - Superpower: SuperpowerID -> Description, Name, VillainID, HeroID
- Organization: OrgName -> LeaderID, HeroID, VillainID

The new Table to resolve Superpower ID problem the new functional dependencies will look like this:

● Villain

- VillainID → DOB, Age, People_Killed, Archnemesis
- Hero
- HeroID → DOB, Age, People_Saved, Archnemesis, AgentID, Seniority
- Agent
- AgentID → DOB, Salary, Experience
- Mission

● MissionID → Name, HeroID, People_Saved, People_Lost, VillainID,

Priority_Level ● Superpower

● SuperpowerID → Description, Name ● Hero_Superpowers

- (HeroID, SuperpowerID) → None
- HeroID → SuperpowerID
- SuperpowerID → HeroID
- Villain_Superpowers
- (VillainID, SuperpowerID) → None
- VillainID → SuperpowerID
- SuperpowerID → VillainID

The new tables will now look like this:

- Villain: VillainID (PK), DOB, Age, People_Killed, Archnemesis (FK, references Hero(HeroID))

- Hero: HeroID (PK), DOB, Age, People_Saved, Archnemesis (FK, references Villain(VillainID)), AgentID (FK, references Agent(AgentID)), Seniority
- Agent: AgentID (PK), DOB, Salary, Experience
- Mission: MissionID (PK), Name, HeroID (FK, references Hero(HeroID)), People_Saved,
People_Lost, VillainID (FK, references Villain(VillainID)), Priority_Level
- Superpower: SuperpowerID (PK), Description, Name
- Hero_Superpowers: HeroID (FK, references Hero(HeroID)), SuperpowerID (FK, references Superpower(SuperpowerID)), Primary Key (HeroID, SuperpowerID)
- Villain_Superpowers: VillainID (FK, references Villain(VillainID)), SuperpowerID (FK, references Superpower(SuperpowerID)), Primary Key (VillainID, SuperpowerID)

User Application Interface

We built the interface using html, css, php, and mysql. The user of the system can perform CRUD operations on the database by inputting information in the text fields. On the navigation bar there are different options to choose from and perform crud operations on. We provide add, update and delete buttons for each option.

Conclusion

Overall, this project was a great step-by-step implementation of a full stack application focusing on performing CRUD operations on a database. It was also fun to create our own fictional database with made-up data. Some future work for this Superhero database is to add more comprehensive crud operations for each option(hero, villain, superpower, etc.). The interface can also be improved to show pictures of heroes, villains, and agents. This database can be scaled vertically to account for more volumes of data as an organization has more employees and goes on more missions.

References

We utilized YouTube tutorials for connecting the front-end and database via PHP. For ER diagrams and schema, we utilized content from the lecture slides.

Appendix

In the zip folder