Peter Guggisberg 4-19-16

Database Management/Professor Labouseur Design Project

A database of all things

By Peter Guggisberg

Table of Contents

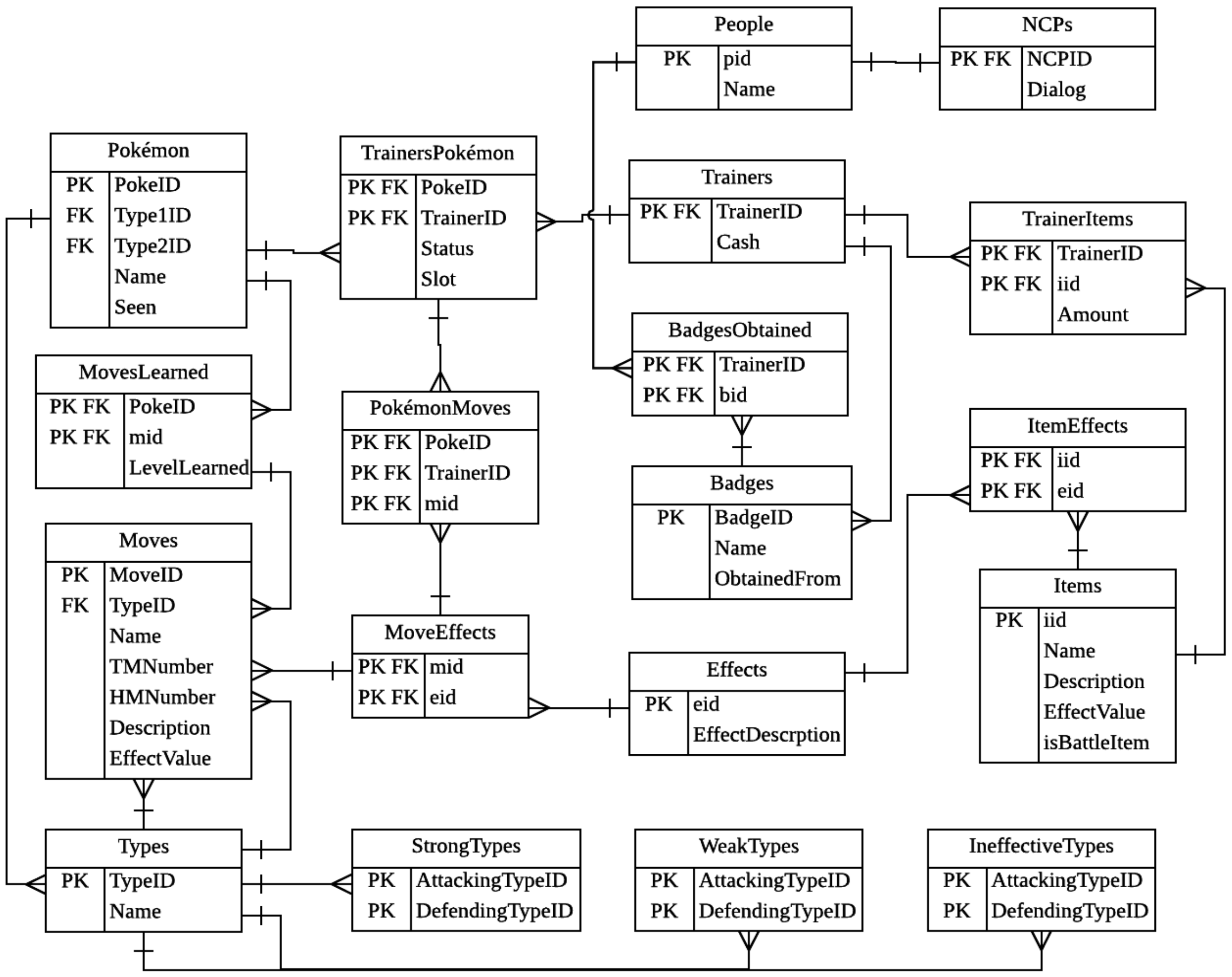
Executive Summary

This document will explain the design and implementation of the database for a Pokémon game. This database will store all of the necessary information for the Pokémon game. The entity relationship diagram below displays how our database will be laid out and we will explain how we will implement it in the sections below. We will even provide example SQL statements to show this implementation. By populating the database with sample data, we can provide examples of how the database will be used in the final product of the game. We will also write sample views for the database for the game and outside users to interact with. We will also show examples of how stored procedures will work with this database. After that, we will explain how this will be used by the game programmers, some known problems with the database, and future enhancements to the database.

Overview

This database will almost exclusively be used by the game itself. However, we have designed the database so that the client can open up portions of the database to outside users. One example of a use case for this flexibility is creating a Pokémon API. Some websites may want to have some of the data available for players to reference. The list of available Pokémon, move list, and item list could be read by this API, so players can have this data to analyze and change their game play accordingly. Many of the calculations related to the database will be done by the game itself because it needs data that is handled by the game and only the game. With this in mind, we created the database to be very flexible, but also to limit the results in case the game is incorrect. The game and the API also handle all of the interactions with end users, so we do not need to account for as much user error in the database.

E-R Diagram


People
PK pid
Name
NCPs
PK FK NCPID
Dialog
Pokémon
PK PokeID
FK Type1ID
FK Type2ID
Name
Seen
TrainersPokémon
PK FK PokeID
PK FK TrainerID
Status
Slot
Trainers
PK FK TrainerID
Cash
TrainerItems
PK FK TrainerID
PK FK iid
Amount
MovesLearned
PK FK PokeID
PK FK mid
LevelLearned
PokémonMoves
PK FK PokeID
PK FK TrainerID
PK FK mid
BadgesObtained
PK FK TrainerID
PK FK bid
ItemEffects
PK FK iid
PK FK eid
Badges
PK BadgeID
Name
ObtainedFrom
Moves
PK MoveID
FK TypeID
Name
TMNumber
HMNumber
Description
EffectValue
MoveEffects
PK FK mid
PK FK eid
Effects
PK eid
EffectDescrption
Items
PK iid
Name
Description
EffectValue
isBattleItem
Types
PK TypeID
Name
StrongTypes
PK AttackingTypeID
PK DefendingTypeID
WeakTypes
PK AttackingTypeID
PK DefendingTypeID
IneffectiveTypes
PK AttackingTypeID
PK DefendingTypeID

Tables

People Table:

This table will store the ID and name of each character (including the player) in the game. It is the base for NPCs and trainers.

Functional Dependencies: pid → name.

```
CREATE TABLE People (
pid SERIAL PRIMARY KEY,
Name TEXT NOT NULL
                                    );
INSERT INTO People (name)
VALUES
(‘Gary’), (‘Professor Oak’), (‘Nurse Joy’), (‘Brock’), (‘PlayerName’)
```

NPCs Table:

This table will store information pertaining to all of the non-trainer characters you encounter in the game. These people just talk to the player and do not need much data stored for them.

Functional Dependencies: NPCID → dialog.

```
CREATE TABLE NPCs (
NPCID INTEGER PRIMARY KEY REFERENCES People (pid),
Dialog TEXT
                                    );
INSERT INTO NPCs (NPCID, dialog)
VALUES
```

```
(2, ‘Please select a starter Pokémon.’), (3, ‘We hope to see you again.’)
```

Trainers Table:

This table will store any character in the game that handles Pokémon. Cash stores the amount of money the trainer has.

Functional Dependencies: TrainerID → cash.

```
CREATE TABLE Trainers (
TrainerID INTEGER PRIMARY KEY REFERENCES People (pid),
Cash MONEY NOT NULL DEFAULT ‘$0’
                                        );
INSERT INTO Trainers (TrainerID, cash)
VALUES
(1, ‘$0’), (4, ‘$200’), (5, ‘$5000’)
```

Badges Table:

This table stores all the information about each badge including its name and who you obtain it from.

Functional Dependencies: BadgeID → name, ObtainedFrom.

```
CREATE TABLE Badges (
BadgeID SERIAL PRIMARY KEY,
Name TEXT NOT NULL,
ObtainedFrom INTEGER NOT NULL REFERENCES Trainers (TrainerID)
                                        );
INSERT INTO Badges (name, ObtainedFrom)
VALUES
```

(‘BoulderBadge’, 4)

BadgesObtained Table:

This table stores the information about who has earned what badge. Each trainer is matched up with the badges they have earned.

CREATE TABLE BadgesObtained (

TrainerID INTEGER REFERENCES Trainer (TrainerID),

bid INTEGER REFERENCES Badges (BadgeID), PRIMARY KEY(TrainerID, bid)

);

INSERT INTO BadgesObtained (TrainerID, bid)

VALUES

(1, 1)

Types Table:

This table describes the different types of Pokémon there are. Each type is strong, weak, ineffective, or neutral against every other type. This information is stored in the strong types, weak types, and ineffective types tables.

Functional Dependencies: TypeID → name.

CREATE TABLE Types (

TypeID SERIAL PRIMARY KEY,

Name TEXT NOT NULL UNIQUE

);

INSERT INTO Types (name)

VALUES

(‘Fire’), (‘Electric’), (‘Water’), (‘Ground’), (‘Poison’), (‘Steel’)

StrongTypes Table:

This table stores the information about which attack move types are strong against each Pokémon type.

```
CREATE TABLE StrongTypes (
AttackingTypeID INTEGER NOT NULL REFERENCES Types (TypeID),
DefendingTypeID INTEGER NOT NULL REFERENCES Types (TypeID),
PRIMARY KEY (AttackingTypeID, DefendingTypeID)
                                             );
INSERT INTO StrongTypes (AttackingTypeID, DefendingTypeID)
VALUES
(1, 6), (3, 1), (2, 3), (3, 4), (4, 2), (4, 5), (4, 6)
```

WeakTypes Table:

This table stores the information about which attack move types are weak against each Pokémon type.

```
CREATE TABLE WeakTypes (
AttackingTypeID INTEGER NOT NULL REFERENCES Types (TypeID),
DefendingTypeID INTEGER NOT NULL REFERENCES Types (TypeID),
PRIMARY KEY (AttackingTypeID, DefendingTypeID)
                                           );
INSERT INTO WeakTypes (AttackingTypeID, DefendingTypeID)
VALUES
(1, 3), (5, 4), (6, 2)
```

IneffectiveTypes Table:

This table stores the information about which attack move types are ineffective against each Pokémon type.

```
CREATE TABLE IneffectiveTypes (
AttackingTypeID INTEGER NOT NULL REFERENCES Types (TypeID),
DefendingTypeID INTEGER NOT NULL REFERENCES Types (TypeID),
PRIMARY KEY (AttackingTypeID, DefendingTypeID)
                                        );
INSERT INTO IneffectiveTypes (AttackingTypeID, DefendingTypeID)
VALUES
(2, 4), (5, 6)
```

Pokémon Table:

This table stores all the information describing each Pokémon's characteristics. TypeID1 and TypeID2 reference the types table, but TypeID1 is not nullable because every Pokémon has at least 1 type, but TypeID2 is nullable because not all Pokémon have 2 types.

Functional Dependencies: PokeID → Name, TypeID1, TypeID2, Seen.

```
CREATE TABLE Pokémon (
PokeID SERIAL PRIMARY KEY,
Name TEXT NOT NULL,
TypeID1 INTEGER NOT NULL REFERENCES Types (TypeID),
TypeID2 INTEGER REFERENCES Types (TypeID),
Seen BOOLEAN NOT NULL DEFAULT False
                                );
```

```
 INSERT INTO Pokémon (Name, TypeID1, TypeID2)
VALUES
(‘Magmar’, 1, null), (‘Pikachu’, 2, null), (‘Clawitzer’, 3, null), (‘Steelix’, 4, 6), (‘Arbok’, 5, null)
```

Moves Table:

This table stores the information about every move a Pokémon can use in battle. They have different types and effects. They can have a TMnumber, an HMnumber, or neither, but not both.

Functional Dependencies: mid → TypeID, name, TMnumber, HMnumber, description, EffectValue, accuracy, PP.

```
CREATE TABLE Moves (
mid SERIAL PRIMARY KEY,
 TypeID INTEGER NOT NULL REFERENCES Types (TypeID),
Name TEXT NOT NULL UNIQUE,
TMnumber INTEGER,
HMnumber INTEGER,
Description TEXT,
EffectValue DOUBLE PRECISION NOT NULL,
Accuracy INTEGER NOT NULL
PP INTEGER NOT NULL,
                                        );
INSERT INTO Moves (TypeID, name, TMnumber, HMnumber, description, EffectValue,
accuracy, PP)
VALUES
```

(2, ‘Thunderbolt’, 24, null, ‘May cause paralysis to the target.’, 90, 100, 15), (3, ‘Surf’, null, 3, null, 90, 100, 15), (5, ‘Poison Sting’, null, null, ‘May cause poison to the target.’, 15, 100, 35), (6, ‘Metal Sound’, null, null, ‘Lowers target’s Sp. Def stat by 2 stages.’, null, 85, 40)

MovesLearned Table:

This table stores the information about what moves each Pokémon learns and at which level they are learned.

Functional Dependencies: PokeID, mid → LevelLearned.

```
CREATE TABLE MovesLearned (
PokeID INTEGER NOT NULL REFERENCES Pokémon (PokeID),
mid INTEGER NOT NULL REFERENCES Moves (mid),
LevelLearned INTEGER NOT NULL,
PRIMARY KEY (PokeID, mid)
                                        );
INSERT INTO MovesLearned (PokeID, mid, LevelLearned)
VALUES
(2, 1, 29), (5, 3, 4)
```

Effects Table:

This table stores data about each effect that is present in the game. Most items and all moves have an effect associated with them, but the value of the effect is not always the same, so the value is stored with the move or item, however the meaning of the values are in the effects table.

Functional Dependencies: eid → description.

```
CREATE TABLE Effects (
eid SERIAL PRIMARY KEY,
Description TEXT NOT NULL
                                        );
INSERT INTO Effects (description)
VALUES
(‘Poison opposing Pokémon with poison type X’), (‘Damage opposing Pokémon with X HP’),
(‘Heal selected Pokémon by X HP’), (‘Decrease speed of opposing Pokémon by X stages’)
```

Items Table:

This table stores data about all items in the game. All of the items have effects, which are stored in a similar fashion to moves. The number of items each trainer has is stored in the trainer items table because it depends on which trainer it is and what item it is.

Functional Dependencies: iid → description, name, EffectValue, isBattleItem.

```
CREATE TABLE Items (
iid SERIAL PRIMARY KEY,
Name TEXT NOT NULL UNIQUE,
Description TEXT NOT NULL,
EffectValue INTEGER,
isBattleItem BOOLEAN NOT NULL
                                        );
INSERT INTO Items (name, description, EffectValue, isBattleItem)
VALUES
(‘Master Ball’, ‘Catch any Pokémon with a 100% success rate’, null, True), (‘Hyper Potion’,
‘Restores health to your Pokémon by 200 HP’, 100, True), (‘Bicycle’, ‘Doubles travel speed’,
null, False)
```

ItemEffects Table:

This table links the items table to the effects table. It also stores the amounts of each item that the trainer possesses because it is dependent on these 2 factors.

```
CREATE TABLE ItemEffects (
eid INTEGER NOT NULL REFERENCES Effects (eid),
iid INTEGER NOT NULL REFERENCES Items (iid),
PRIMARY KEY (eid, iid)
                                        );
INSERT INTO ItemEffects (eid, iid, amount)
VALUES
(3, 2, 3)
```

MoveEffects Table:

This table links the moves table to the effects table. All moves have an effect to go with their effect value.

```
CREATE TABLE MoveEffects (
eid INTEGER NOT NULL REFERENCES Effects (eid),
mid INTEGER NOT NULL REFERENCES Moves (mid),
PRIMARY KEY (eid, mid)
                                          );
INSERT INTO MoveEffects (eid, mid)
VALUES
(1, 2), (2, 2), (3, 2), (4, 2)
```

TrainerPokémon Table:

This table stores data about which Pokémon are in each trainer’s party. This also stores information about that specific Pokémon because it is dependent on this information.

Functional dependencies: PokeID, TrainerID → status, slot.

```
CREATE TABLE TrainerPokémon (
PokeID INTEGER NOT NULL REFERENCES Pokémon (Pokeid),
TrainerID INTEGER NOT NULL REFERENCES Trainers (TrainerID),
Status TEXT NOT NULL DEFAULT ‘normal’ CHECK (status = ‘normal’ OR status = ‘asleep’
OR status = ‘burned’ OR status = ‘paralyzed’ OR status = ‘frozen’ OR status = ‘poisoned’ OR
status = ‘fainted’),
Slot INTEGER NOT NULL DEFAULT 1 CHECK (slot > 0 and slot < 7),
PRIMARY KEY (PokeID, TrainerID)
                                        );
INSERT INTO TrainerPokémon (PokeID, TrainerID, status, slot)
VALUES
(1, 1, ‘normal’, 1), (3, 1, ‘normal’, 2), (2, 5, ‘poisoned’, 1), (4, 5, ‘fainted’, 6), (5, 5, ‘burned’, 3)
```

PokémonMoves Table:

This table stores the information about which Pokémon knows which moves, which depends on the trainer, so it needs to be combined with trainers as well.

```
CREATE TABLE PokémonMoves (
PokeID INTEGER NOT NULL REFERENCES TrainerPokémon (PokeID),
TrainerID INTEGER NOT NULL REFERENCES TrainerPokémon (TrainerID),
mid INTEGER NOT NULL REFERENCES Moves (mid),
PRIMARY KEY (PokeID, TrainerID, mid)
                                        );
INSERT INTO MoveEffects (PokeID, TrainerID, mid)
VALUES
(2, 1, 1), (5, 1, 3), (6, 2, 4)
```

TrainerItems Table:

This table stores the information about each trainers’ items and how many they have.

```
CREATE TABLE TrainerItems (
ItemID INTEGER NOT NULL REFERENCES Items (iid),
TrainerID INTEGER NOT NULL REFERENCES Trainers (TrainerID),
Amount INTEGER NOT NULL DEFULT 0,
PRIMARY KEY (ItemID, TrainerID)
                                 );
INSERT INTO TrainerItems (ItemID, TrainerID, amount)
VALUES
(1, 5, 99)
```

Views and Reports

Pokédex:

This view shows all of the data for Pokémon.

```
SELECT p.PokeID, p.name, t1.name, t2.name
FROM Pokémon INNER JOIN Types t1 on p.TypeID1 = t1.TypeID
LEFT OUTER JOIN Types t2 on p.TypeID2 = t2.TypeID;
```

PlayerPokédex:

This view shows all of the data for any Pokémon that the player has seen or caught only.

```
SELECT PokeID, p.name, t1.name, t2.name, seen
FROM Pokémon p
INNER JOIN Types t1 ON p.TypeID1 = t1.TypeID
LEFT OUTER JOIN Types t2 ON p.TypeID2 = t2.TypeID
WHERE p.seen = True;
```

PlayersParty:

This view shows all of the Pokémon and their data in order in the player’s party.

```
SELECT tp.PokeID, p.name, t1.name, t2.name, tp.status, tp.slot
FROM TrainerPokémon tp
INNER JOIN Pokémon p on (p.PokeID = tp.PokeID)
INNER JOIN Types t1 ON p.TypeID1 = t1.TypeID
LEFT OUTER JOIN Types t2 ON p.TypeID2 = t2.TypeID
WHERE tp.TrainerID = 5 ORDER BY tp.slot;
```

PlayersItems:

This view gets the player’s inventory of items.

```
SELECT i.name, i.description, ti.amount
FROM TrainerItems ti
INNER JOIN Items I ON (i.iid = ti.ItemID)
WHERE ti.TrainerID = 5;
```

Stored Procedures

isStrong, isWeak, isIneffective:

These three stored procedures can be used to see if the attack is super effective, not very effective, or ineffective.

```
CREATE OR REPLACE FUNCTION isStrong (AttackTypeID INTEGER, DefenderTypeID
INTEGER)
RETURNS BOOLEAN AS $$
FOR R IN
SELECT st.AttackingTypeID, st.DefendingTypeID
FROM StrongTypes st
WHERE st.AttackingTypeID = AttackTypeID AND st.DefendingTypeID = DefenderTypeID
LOOP
RETURN True;
END LOOP;
$$ LANGUAGE PLPGSQL;
CREATE OR REPLACE FUNCTION isWeak (AttackTypeID INTEGER, DefenderTypeID
INTEGER)
RETURNS BOOLEAN AS $$
FOR R IN
SELECT wt.AttackingTypeID, wt.DefendingTypeID
FROM WeakTypes wt
WHERE wt.AttackingTypeID = AttackTypeID AND wt.DefendingTypeID = DefenderTypeID
LOOP
```

```
RETURN True;
END LOOP;
$$ LANGUAGE PLPGSQL;
CREATE OR REPLACE FUNCTION isIneffective (AttackTypeID INTEGER, DefenderTypeID
INTEGER)
RETURNS BOOLEAN AS $$
FOR R IN
SELECT it.AttackingTypeID, it.DefendingTypeID
FROM IneffectiveTypes it
WHERE it.AttackingTypeID = AttackTypeID AND it.DefendingTypeID = DefenderTypeID
LOOP
RETURN True;
END LOOP;
$$ LANGUAGE PLPGSQL;
```

CheckAlreadyKnown:

When you are teaching a Pokémon a new move, it checks to see if the Pokémon already knows that move.

```
CREATE FUNCTION CheckAlreadyKnown ()
RETURNS TRIGGER AS $CheckAlreadyKnown$
BEGIN
IF EXIST (
SELECT mid
FROM PokémonMoves
WHERE PokeID = NEW.PokeID AND TrainerID = NEW.TrainerID AND mid= NEW.mid
            )
THEN
RAISE EXCEPTION “This Pokémon already knows this move”';
END IF;
RETURN NEW;
END;
$CheckAlreadyKnown$ LANGUAGE PLPGSQL;
```

Triggers

```
CREATE TRIGGER CheckAlreadyKnown BEFORE INSERT ON PokémonMoves
FOR EACH ROW EXECUTE PROCEDURE CheckAlreadyKnown ();
```

Security

Admin:

The administrator of the database needs to have privileges to change anything in the database. Any new moves, new Pokémon, or new items will be added or updated by the admin.

```
CREATE ROLE Admin
GRANT SELECT, INSERT, UPDATE ON ALL TABLES IN SCHEMA public
TO Admin;
```

Game:

The game itself must have read access to everything to be able to get the data required to run. The game also has insertion rights for some things because the game will change the data around a lot. However, the game cannot change many things, such as what Pokémon there are, their types, their moves, what kind of items there are, etc., so those privileges are revoked.

CREATE ROLE Game

GRANT SELECT ON ALL TABLES IN SCHEMA public

TO Game;

GRANT UPDATE, INSERT

ON People, Trainers, TrainersPokémon, BadgesObtained, TrainersItems

TO Game;

REVOKE UPDATE, INSERT ON StrongTypes, WeakTypes, IneffectiveTypes, Types, Pokémon, MovesLearned, Moves, MoveEffects, Effects, ItemEffects, Items, NPCs, Badges

TO Game;

API:

The API is an optional addition to the database. It does not allow the public to change the information in the database, but it does allow them to read from specific tables that could aid in their playing of the game.

CREATE ROLE API

GRANT SELECT ON Types, StrongTypes, WeakTypes, IneffectiveTypes, Pokémon, MovesLearned, Moves, MoveEffects, Effects, ItemEffects, Items, Badges

TO API;

REVOKE UPDATE, INSERT

ON ALL TABLES IN SCHEMA public

TO API;

Implementation Notes:

This database's main purpose is to provide data for this game to manipulate, so many calculations will happen in the game itself. This database can be hosted over the web, so it can be accessed from anywhere, but the player's device must always be connected to the Internet. Another possible option is to host it locally on the game device. The obvious drawbacks are that you will no longer have an API and once the database is on the game, you cannot add to any administrator-only tables. The game itself is fully responsible for user input and handling events while it utilizes the database for information. Pokémon, moves, and items are only added in each new release of the game, so when the team decides to add a new Pokémon, move, or item, the database administrator is responsible for adding the Pokémon, move, or item data to the database.

Known Problems:

Each Pokémon has 2 fields for types. This could be separated into a separate table to avoid nulls. Because of the game mechanics, locks are not needed in this database, but they could be added for additional security. We could separate players from trainers, but it is unnecessary because they have the same characteristics.

Future Enhancements:

We could add the game maps to the database. We could store NPC and character locations in the database if they have the maps. The game currently handles all battle information, but we could bring that info to the database if we wanted to store that info or record battle history.