The Completely Original Game Created for the GameGuy Color

Database by Zack Meath

Table of Contents

Executive Summary:

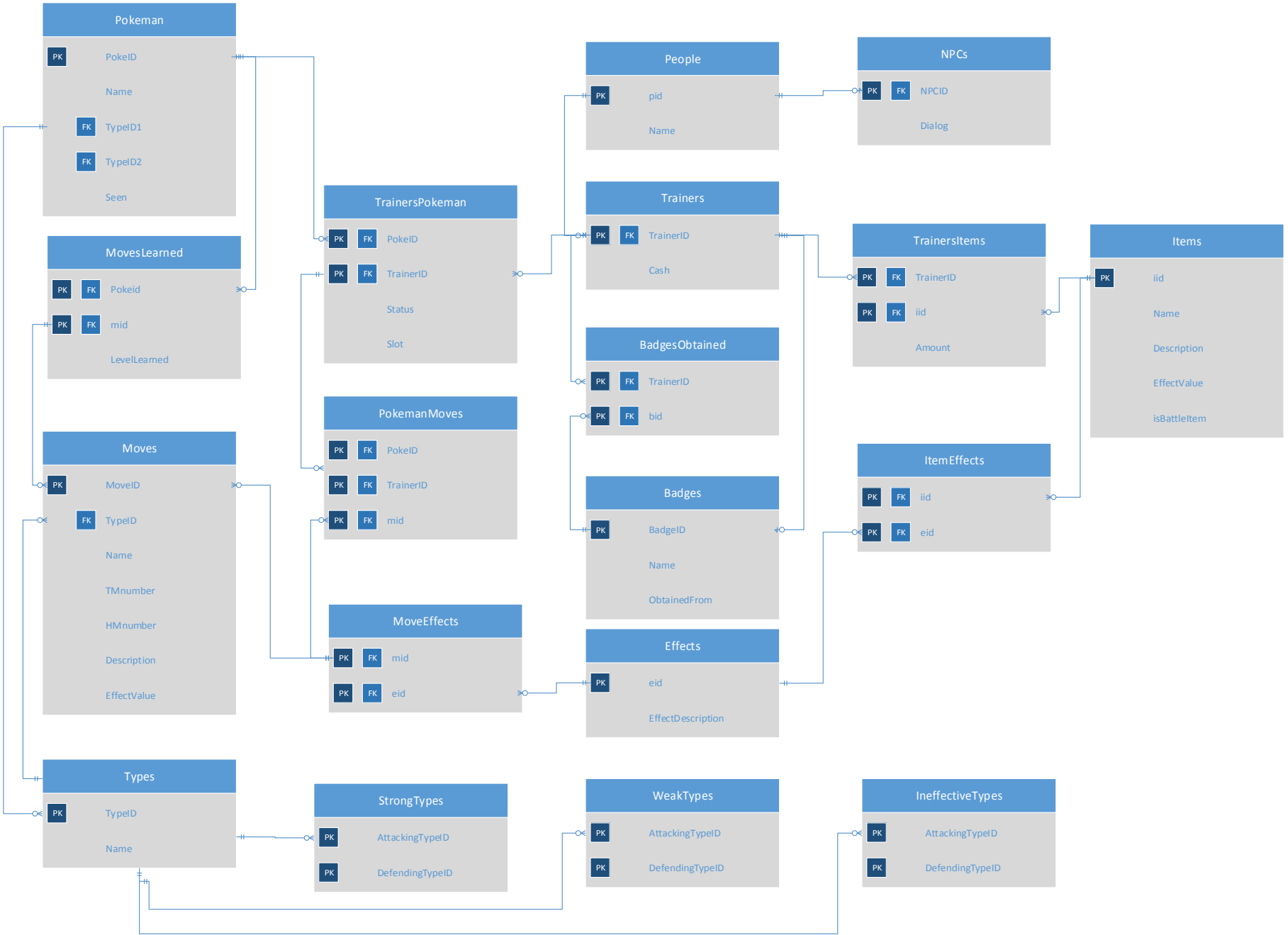
This document will explain the design and implementation of the database for the Pokeman GameGuy game. This database will store all of the necessary information for the Pokeman game. The entity relationship diagram below displays how our database will be laid out and we will explain how we will implement 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 and some known problems with 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 Pokeman API. Some websites may want to have some of the data available for players to reference. The list of available pokeman, movelist, 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 the game only. 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 interaction with end users so we do not need to account for as much user error in the database.

ER DIAGRAM


Pokeman
PK PokeID
Name
FK TypeID1
FK TypeID2
Seen
MovesLearned
PK FK Pokeid
PK FK mid
LevelLearned
Moves
PK MoveID
FK TypeID
Name
TMnumber
HMnumber
Description
EffectValue
Types
PK TypeID
Name
TrainersPokeman
PK FK PokeID
PK FK TrainerID
Status
Slot
PokemanMoves
PK FK PokeID
PK FK TrainerID
PK FK mid
MoveEffects
PK FK mid
PK FK eid
StrongTypes
PK AttackingTypeID
PK DefendingTypeID
People
PK pid
Name
Trainers
PK FK TrainerID
Cash
BadgesObtained
PK FK TrainerID
PK FK bid
Badges
PK BadgeID
Name
ObtainedFrom
Effects
PK eid
EffectDescription
WeakTypes
PK AttackingTypeID
PK DefendingTypeID
NPCs
PK FK NPCID
Dialog
TrainersItems
PK FK TrainerID
PK FK iid
Amount
ItemEffects
PK FK iid
PK FK eid
Items
PK iid
Name
Description
EffectValue
isBattleItem
IneffectiveTypes
PK AttackingTypeID
PK DefendingTypeID

Tables

People table:
The people 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'),
('Guy Number 7'),
('Nurse Joy'),
('Brock'),
('PlayerName')
```

NPCs table:
The NPC (Non-Playable Character) table will store information pertaining to all of the non-trainer characters you encounter in the game. These people can only really talk and do not need much data stored for them. If dialog is null, then they do not speak.

Functional Dependencies:
NPCID → dialog

```
CREATE TABLE NPCs (
      NPCID INTEGER PRIMARY KEY REFERENCES People(pid),
      dialog TEXT
      );
INSERT INTO NPCs (NPCID, dialog)
VALUES  (2, 'I will let you have a secret Pokeman – a MAGIKARP – for just $500!'),
      (3, 'We hope to see you again.')
```

Trainers table:
The Trainers table will store any character in the game that handles pokeman. 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
(5,'$3000'),
(1,'$0'),
(4,'$500')
```

Badges table:
The badges table stores all the information about each badge including it 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:
The BadgesObtained 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:
The Types table describes the different types of pokeman there are. Each type is strong, weak, ineffective, or neutral against every other type. This information is stored in the StrongTypes, WeakTypes, and InneffectiveTypes tables.

Functional Dependencies:
TypeID → name

```
CREATE TABLE Types (
      TypeID SERIAL PRIMARY KEY,
      name TEXT NOT NULL UNIQUE
      );

INSERT INTO Types(name)
VALUES
('Ghost'),
('Normal'),
('Water'),
('Fire'),
('Grass'),
('Poison')
```

StrongTypes table:
This table stores the information about which attack move type is strong against each pokeman 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
(5,3),
(4,5),
(6,5)
```

WeakTypes table:
This table stores the information about which attack move type is weak against each pokeman 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
(4,3),
(5,6),
(3,5)
```

IneffectiveTypes table:
This table stores the information about which attack move type is ineffective against each pokeman 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
(2,1)
```

Pokeman table:
The Pokeman table stores all the information describing each pokeman's characteristics. TypeID1 and TypeID2 reference the Types table but TypeID1 is not nullable because every Pokeman has at least 1 type but TypeID2 is nullable because not all pokeman have 2 types.

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

```
CREATE TABLE Pokeman (
      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 Pokeman (Name, TypeID1, TypeID2)
VALUES
('Gengar', 1, 6),
('Snorlax', 2, null),
('Blastoise', 3, null),
('Victreebell',5,6),
('Magmar', 4, null)
```

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

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

```
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,
      pp INTEGER NOT NULL,
      accuracy INTEGER NOT NULL
      );

INSERT INTO Moves(TypeID, name, TMnumber, HMnumber, description, EffectValue, pp,accuracy)
VALUES
(2,'Cut', null, 1, 'Cut inflicts damage and has no secondary effect.', 50, 30, 95 ),
(4, 'Fire Blast', 38, null, 'Fire Blast inflicts damage and has a 30% chance of burning the
      target.',120, 5, 85),
(5, 'Vine Whip', null, null, 'Vine Whip inflicts damage and has no secondary effect', 45,25,100),
(6, 'Smog', null,null,'Smog inflicts damage and has a 40% chance of poisoning the target.',30,20,70)
```

MovesLearned table:
This table stores the information about what move each pokeman learns at which level.

Functional Dependencies:
PokeID, mid → LevelLearned

```
CREATE TABLE MovesLearned(
      PokeID INTEGER NOT NULL REFERENCES Pokeman(PokeID),
      mid INTEGER NOT NULL REFERENCES Moves(mid),
      LevelLearned INTEGER NOT NULL,
      PRIMARY KEY(PokeID,mid)
      );

INSERT INTO MovesLearned(PokeID, mid, LevelLearned)
VALUES
(5,2,56),
(1,4,31)
```

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 (example, this move does 30 damage) but the value of the effect is not always the same, so the value is stored with the move or item but what the value means is 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 pokeman with poison type X'),
('Damage opposing pokeman with X hitpoints'),
('Heal selected pokeman by X hitpoints'),
('Decrease speed of opposing pokeman 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 TrainerItems table because it depends on what 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 pokeman with a 100% success rate', null, True),
('Hyper Potion','Restores health to your pokeman',100, True),
('Bicycle','Doubles travel speed',null,False)
```

ItemEffects table:
This table links the Items table to the Effects table. It also stores the amount of each item that the trainer posesses 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)
```

TrainerPokeman table:
This table stores data about which pokeman are in each trainer's party. This also stores information about that specific pokeman because it is dependent on this information.

Functional dependencies:
PokeID, TrainerID → status, slot

```
CREATE TABLE TrainerPokeman(
      PokeID INTEGER NOT NULL REFERENCES Pokeman(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 TrainerPokeman(PokeID, TrainerID, status, slot)
VALUES
(1,1,'normal',1),  (3,1,'normal',2),  (2,5,'poisoned',1),  (4,5,'normal',2),  (5,5,'normal',3)
```

PokemanMoves table:
This table stores the information about which pokeman knows which move, which depends on the trainer so it needs to be combined with trainers as well.

```
CREATE TABLE PokemanMoves(
      PokeID INTEGER NOT NULL REFERENCES TrainerPokeman(PokeID),
      TrainerID INTEGER NOT NULL REFERENCES TrainerPokeman(TrainerID),
      mid INTEGER NOT NULL REFERENCES Moves(mid),
      PRIMARY KEY (PokeID,TrainerID,mid)
      );

INSERT INTO MoveEffects(PokeID,TrainerID,mid)
VALUES
(1,1,4),
(2,2,1),
(4,2,3),
(5,2,2)
```

TrainerItems table:
This table stores the information about each trainer's items ans 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

Pokedex:
This view shows all of the data for pokeman.

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

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

```
SELECT PokeID, p.name, t1.name, t2.name, seen
FROM Pokeman 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 pokeman and their data (in order) in the player's party.

```
SELECT tp.PokeID, p.name, t1.name, t2.name, tp.status, tp.slot
FROM TrainerPokeman tp INNER JOIN Pokeman 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 get 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 Pokeman a new move, it checks to see if the pokeman already knows the move.

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

TRIGGERS

```
CREATE TRIGGER CheckAlreadyKnown BEFORE INSERT ON PokemanMoves
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 pokeman, 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 Pokeman 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, TrainersPokeman, BadgesObtained, TrainersItems
TO game

REVOKE UPDATE, INSERT
ON StrongTypes, WeakTypes, IneffectoveTypes, Types, Pokeman, 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, Pokeman, 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 serve data for the 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 can not 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. Pokeman, move, and items are only added in each new release of the game, so when the team decides to add a new Pokeman, more, or item, the database administrator is responsible for adding the pokeman, move, or item data to the database.

KNOWN PROBLEMS

- Each pokeman 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 location in the database if had 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.