University of British Columbia, Vancouver
Department of Computer Science

CPSC 304 Project Cover Page

Milestone #: 2

Date: Mar 1, 2024

Group Number: 73

Name	Student Number	CS Alias (Userid)	Preferred E-mail Address
Frank Yu	70917505	q2l0q	frankkaiwen.yu@gmail.com
Philip Macau	25060179	d3o3o	macau.philip@gmail.com
Ethan Kenny	58256322	y7b5p	ekenny456@gmail.com

University of British Columbia, Vancouver
Department of Computer Science

Summary:

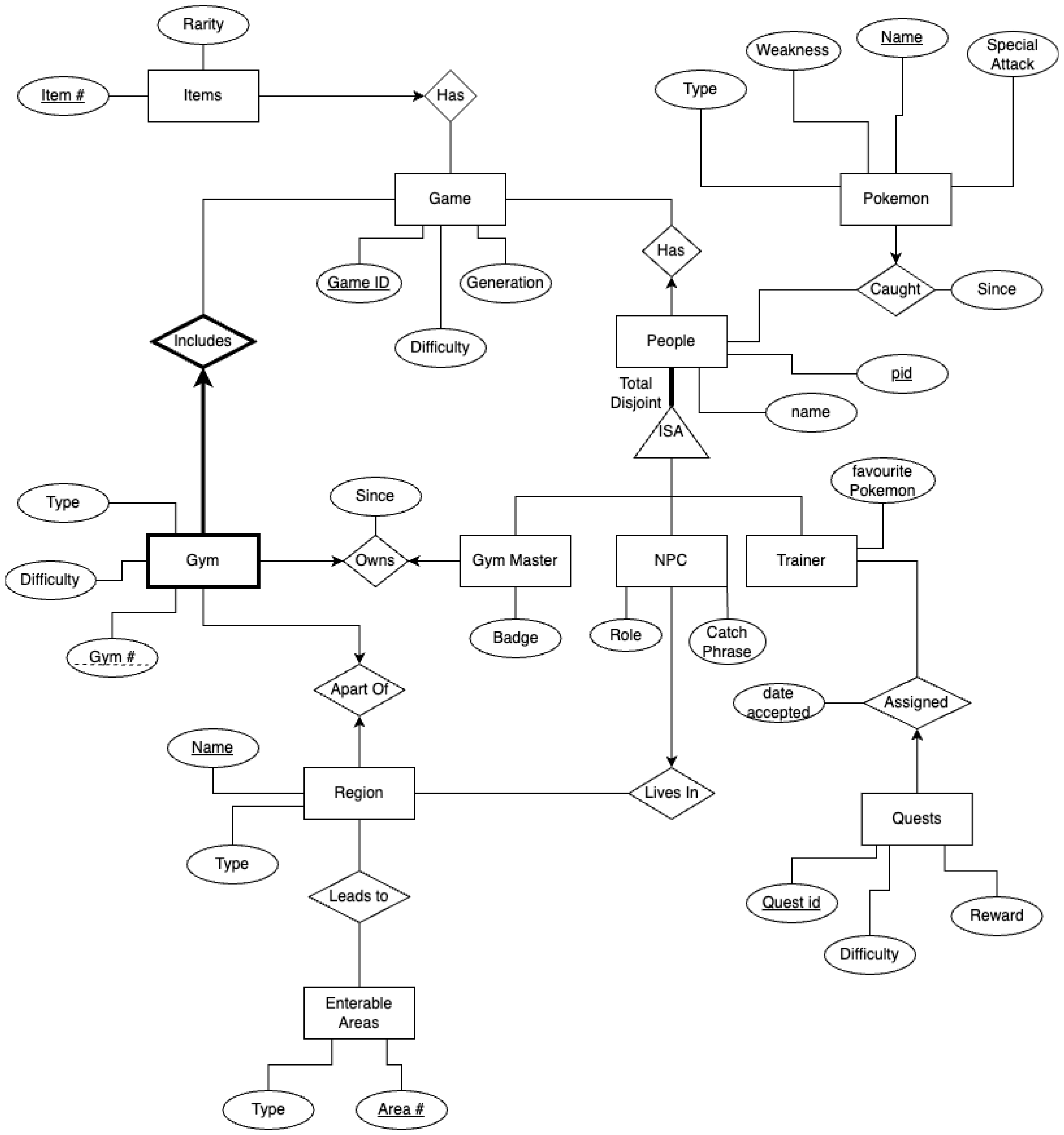
The domain of our application is Pokemon gaming and the tracking of game statistics. With the ability to customize their own Pokemon games, users will be able to tailor their game experiences according to their preferences. Furthermore, in regards to the tracking of game statistics domain, users are able to track individual player progression data, including Pokemons they've captured, quests they completed, etc.

ER Diagram:

Got rid of the arrows connecting attributes to entities, added ISA hierarchy constraints, and added dotted underline to the primary key of the weak entity. Also added some more attributes to help with making the entities more descriptive and help create better FDs.

- "Since" attribute added for "Owns" relation and "Caught" relation
- "Badge" attribute added for "Gym Master" entity
- "Role" and "Catch Phrase" attribute added for "NPC" entity
- "Favourite Pokemon" attribute added for "Trainer" entity
- "Date accepted" attribute added for "Assigned" relation

⟶ some of the naming of the entity and relations are just slightly different from the names of our tables and relational models below (we removed the spaces, or just added an underscore for the space), however they should all be quite similar to the ER diagram.
⟶ when combining the names of the entity and relations, we used an underscore
⟶ in our tables and relational models, we also added more detail for the naming of the attributes, for example the Region and Pokemon both have the name attribute, we just changed it to regionName and pokemonName, just so that we know which is which when working with the FDs
⟶ Naming remains consistent for all tables/relations following q4.
⟶ ER diagram on the next page


Rarity
Item #
Items
Has
Weakness
Name
Special Attack
Type
Game
Pokemon
Has
Game ID
Generation
Caught
Since
Difficulty
Includes
People
pid
Total Disjoint
name
ISA
favourite Pokemon
Type
Since
Gym
Owns
Gym Master
NPC
Trainer
Difficulty
Badge
Role
Catch Phrase
Gym #
Apart Of
date accepted
Assigned
Name
Region
Lives In
Quests
Type
Leads to
Quest id
Reward
Difficulty
Enterable Areas
Type
Area #

4.
The schema derived from your ER diagram (above). For the translation of the ER diagram to the relational model, follow the same instructions as in your lectures. The process should be reasonably straightforward. For each table:

a. *List the table definition (e.g., Table1(attr1: domain1, attr2: domain2, ...)). Make sure to include the domains for each attribute.*
b. *Specify the primary key (PK), candidate key, (CK) foreign keys (FK), and other constraints (e.g., not null, unique, etc.) that the table must maintain.*

Items_Has (*item#*: integer, *rarity*: string, gameID: integer)
PK = *item#*
FK = *gameID*

Game (*gameID*: integer, game_*difficulty*: string, *generation*: string)
PK = *gameID*

Gym_Includes (*gym#*: integer, *gameID*: integer, *type*: string, gym_*difficulty*: string)
PK = *gym#, gameID*
FK = *gameID (NOT NULL)*

Region_ApartOf (*regionName*: string, *type*: string, *gym#:* integer, *gameID*: integer)
PK = *regionName*
FK = *gym#, gameID (UNIQUE)*

LeadsTo (*area#*: integer, *regionName*: string)
FK = *area#, regionName*

EnterableAreas (*area#*: integer, *type*: string)
PK = *area#*

Pokemon_Caught (*pokemonName*: string, *type*: string, *weakness*: string, *specialAttack*: string, *caught_since*: date, *pid*: integer)
PK = *pokemonName*
FK = *pid*

People_Has (*pid*: integer, *name*: string, *gameID*: integer)
PK = *pid*
FK = *gameID*

GymMaster_Owns (*pid*: integer, *name*: string, *badge*: string, *owns_since*: date, *gym#*: integer, *gameID*: integer)
PK = *pid*
FK = *gym#, gameID (UNIQUE)*

NPC_LivesIn (*pid*: integer, *name*: string, *role*: string, *catch_phrase*: string, *gameID*: integer, *regionName*: string)
PK = *pid*
FK = *regionName, gameID*

Trainer (*pid*: integer, *name*: string, *fav_pokemon*: string, *gameID*: integer)
PK = *pid*
FK = *gameID*

Quests_Assigned (*questID*: integer, *difficulty*: string, *reward*: string, *pid*: integer, *date_accepted*: date)
PK = *questID*
FK = *pid*

5.
Functional Dependencies (FDs)

a. Identify the functional dependencies in your relations, including the ones involving all candidate keys (including the primary key).

PKs and CKs are considered functional dependencies and should be included in the list of FDs. You do not need to include trivial FDs such as A ⟶ A.

***Note:** In your list of FDs, there must be some kind of valid FD other those identified by a PK or CK. If you observe that no relations have FDs other than the PK and CK(s), then you will have to intentionally add some (meaningful) attributes to show valid FDs. We want you to get a good normalization exercise. Your design must go through a normalization process. You do not need to have a non-PK/CK FD for each relation but be reasonable. If your TA feels that some non-PK/CK FDs have been omitted, your grade will be adjusted accordingly*

Items_Has:
- item# ⟶ rarity, gameID
  CK = {item#}
  PK = {item#}

Game:
- gameID ⟶ game_difficulty, generation
  CK = {gameID}
  Pk = {gameID}

Game_Includes:
- gym#, gameID ⟶ type, gym_difficulty
  CK = {gym#, gameID}

PK = {gym#, gameID}

Region_ApartOf:
- regionName ⟶ type, gym#, gameID
  CK = {regionName}
  PK = {regionName}

LeadsTo:
- regionName, area# ⟶ none
  CK = {regionName, area#}
  PK = {regionName, area#}

EnterableAreas:
- area# ⟶ type
  CK = {area#}
  PK = {area#}

Pokemon_Caught:
- pokemonName ⟶ Type, Weakness,Special_Attack, since
- type ⟶ Weakness
  CK = {pokemonName, pid}
  PK = {pokemonName}

People_has:
- pid ⟶ name, gameID
  CK = {pid}
  PK = {pid}

GymMaster_Owns:
- pid ⟶ name, owns_since
- badge ⟶ gym#, gameID
  CK = {pid, badge}
  PK = {pid}

NPC_LivesIn:
- pid ⟶ name, regionName, gameID
- role ⟶ catchPhrase
  CK = {pid, role}
  PK = {pid}

Trainer:
- pid ⟶ name, favouritePokemon, gameID
  CK = {pid}
  PK = {pid}

Quests_Assigned:
- quest_Id ⟶ difficulty, dateAccepted, pid
- difficulty ⟶ reward

CK = {quest_Id}
PK = {quest_Id}

6.
Normalization

a. Normalize each of your tables to be in 3NF or BCNF. Give the list of tables, their primary keys, their candidate keys, and their foreign keys after normalization.

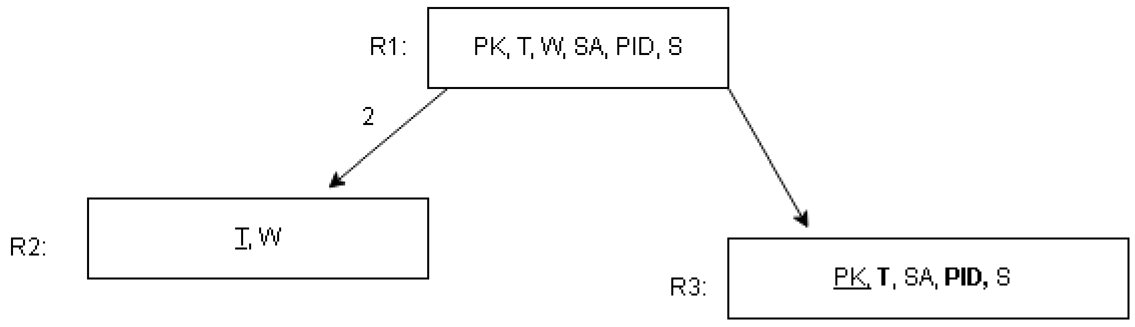
You should show the steps taken for the decomposition. Should there be errors, and no work is shown, no partial credit can be awarded without steps shown. The format should be the same as Step 3, with tables listed similar to Table1(attr1:domain1, attr2:domain2, ...). ALL Tables must be listed, not only the ones post normalization.

Updated Tables:

(to be decomposed)
Pokemon_Caught (*pokemonName*: string, *type*: string, *weakness*: string, *specialAttack*: string, *pid*: integer, *caught_since*: date)
PK = *pokemonName*
FK = *pid*

FDs:
1. pokemonName ⟶ Type, Weakness, Special_Attack, pid, since
2. Type ⟶ Weakness VIOLATES BCNF

Candidate key: {pokemonName}
{pokemonName, type, weakness, specialAttack, pid, since} ~ {PK, T, W, SA, PID, S}

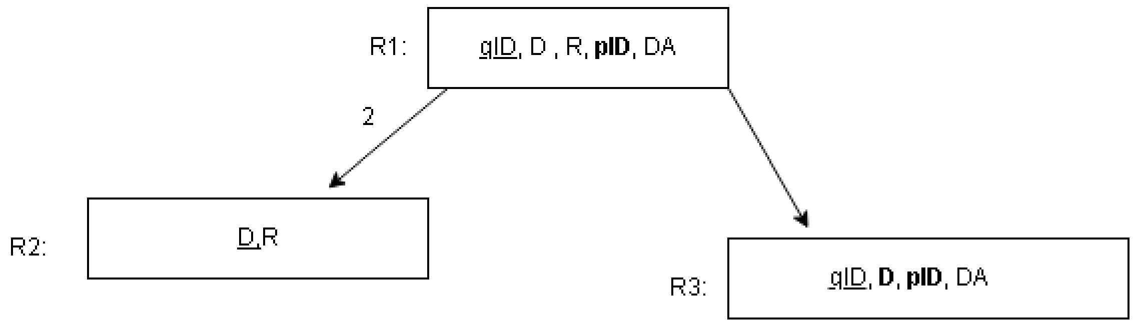


R2: Type_weakness(*type*: string, *weakness*: string)
PK = *type*

R3: Pokemon_Caught (*pokemonName*: string, *type*: string, *specialAttack*: string, *pid*: integer, caught_since: date)
PK = pokemonName

FK = type, pid

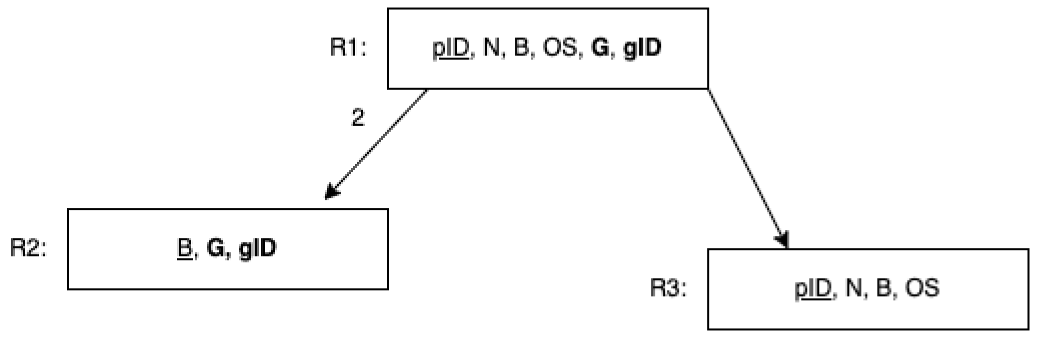
(to be decomposed)
Quests_Assigned (*questID*: integer, *difficulty*: string, *reward*: string, *pid*: integer, *date_accepted*: string)
PK = *questID*
FK = *pid*
FDs:
1. Quest_Id ⟶ Difficulty, dateAccepted, pid
2. Difficulty ⟶ Reward VIOLATES BCNF
Candidate key: {Quest_Id}
{questID, difficulty, reward, pid, date_accepted} ~ {qID, D, R, pID, DA}



R2: Quests_Assigned (*questID*: integer, *difficulty*: string, *pid*: integer, *date_accepted*: string)
PK: questID
FK: difficulty, pid

R3: Difficulty_reward (*difficulty*: string, *reward*: string)
PK: difficulty

(to be decomposed)
GymMaster_Owns (*pid*: integer, *name*: string, *badge*: string, *owns_since*: date, *gym#*: integer, *gameID*: integer)
PK = *pid*
FK = *gym#, gameID (UNIQUE)*
FDs:
1. pid ⟶ name, owns_since VIOLATES BCNF
2. badge ⟶ gym#, gameID VIOLATES BCNF
Candidate key: {pid, badge}
{pid, name, badge, owns_since, gym#, gameID} ~ {pID, N, B, OS, G, gID}



R2: Badge_Gym (*badge*: string, *gym#*: integer, *gameID*: integer)

PK: badge

FK: game#, gameID

R3: GymMaster_Owns (*pid*: integer, *name*: string, *badge*: string, *owns_since*: date)

PK: pid

FK: badge

(to be decomposed)

NPC_LivesIn (*pid*: integer, *name*: string, *role*: string, *catch_phrase*: string, *gameID*: integer, *regionName*: string)

PK = *pid*

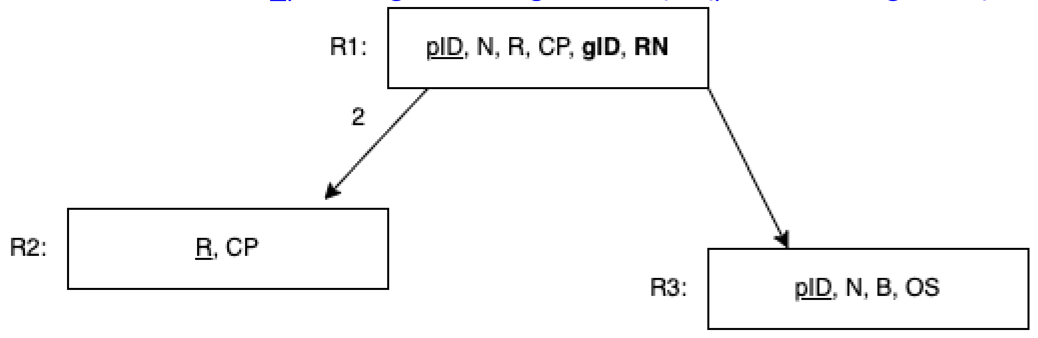
FK = *regionName, gameID*

FDs:

1. pid ⟶ name, regionName, gameID VIOLATES BCNF
2. role ⟶ catchPhrase VIOLATES BCNF

Candidate key: {pid, role}

{pid, name, role, catch_phrase, gameID, regionName} ~ {pID, N, R, CP, gID, RN}



R2: Role_catchPhrase (*role*: string, *catch_phrase*: string)

PK: role

R3: NPC_LivesIn (*pid*: integer, *name*: string, *role*: string, *gameID*: integer, *regionName*: string)

PK: pid

FK: role, gameID, regionName

Non-updated Tables:

Items_Has (*item#*: integer, *rarity*: string, gameID: integer)
PK = *item#*
FK = *gameID*

Game (*gameID*: integer, game_*difficulty*: string, *generation*: string)
PK = *gameID*

Gym_Includes (*gym#*: integer, *gameID*: integer, *type*: string, gym_*difficulty*: string)
PK = *gym#, gameID*
FK = *gameID (NOT NULL)*

Region_ApartOf (*regionName*: string, *type*: string, *gym#:* integer, *gameID*: integer)
PK = *regionName*
FK = *gym#, gameID (UNIQUE)*

LeadsTo (*area#*: integer, *regionName*: string)
FK = *area#, regionName*

EnterableAreas (*area#*: integer, *type*: string)
PK = *area#*

People_Has (*pid*: integer, *name*: string, *gameID*: integer)
PK = *pid*
FK = *gameID*

Trainer (*pid*: integer, *name*: string, *fav_pokemon*: string, *gameID*: integer)
PK = *pid*
FK = *gameID*

7.
The SQL DDL statements required to create all the tables from item #6. The statements should use the appropriate foreign keys, primary keys, UNIQUE constraints, etc. Unless you know that you will always have exactly x characters for a given character, it is better to use the VARCHAR data type as opposed to a CHAR(Y). For example, UBC courses always use four characters to represent which department offers a course. In that case, you will want to use CHAR(4) for the department attribute in your SQL DDL statement. If you are trying to represent the name of a UBC course, you will want to use VARCHAR as the number of characters in a course name can vary greatly.

```
CREATE TABLE Items_Has (
        item#           INT,
        rarity          VARCHAR,
        gameID          INT,
        PRIMARY KEY (item#),
        FOREIGN KEY (gameID) REFERENCES (Game)
                ON DELETE CASCADE
                ON UPDATE CASCADE
);
CREATE TABLE Game (
        gameID                  INT,
        game_difficulty         VARCHAR,
        generation              VARCHAR,
        PRIMARY KEY (gameID)
);
CREATE TABLE Gym_Includes (
        gym#            INT,
        difficulty      VARCHAR,
        type            VARCHAR,
        gameID          INT             NOT NULL,
        PRIMARY KEY (gameID, gym#),
        FOREIGN KEY (gameID) REFERENCES (Game)
                ON DELETE CASCADE
);
CREATE TABLE Region_ApartOf (
        regionName              VARCHAR,
        type                    VARCHAR,
        gym#                    INT     UNIQUE,
        gameID                  INT     UNIQUE,
        PRIMARY KEY (regionName),
        FOREIGN KEY (gym#, gameID) REFERENCES (Gym_Includes)
);
CREATE TABLE LeadsTo (
        regionName      VARCHAR,
        area#           INT,
        FOREIGN KEY (regionName) REFERENCES (Region_ApartOf),
        FOREIGN KEY (area#) REFERENCES (EnterableAreas)
);
CREATE TABLE EnterableAreas (
        area#           INT,
        type            VARCHAR,
        PRIMARY KEY (Area_#)
```

```
);
CREATE TABLE Pokemon_Caught (
        name                VARCHAR,
        type                VARCHAR,
        specialAttack       VARCHAR,
        caught_since        VARCHAR,
        pid                 INT,
        PRIMARY KEY (name),
        FOREIGN KEY (pid) REFERENCES (People)
                ON DELETE CASCADE
                ON UPDATE CASCADE,
        FOREIGN KEY (type) REFERENCES Type_weakness
                ON DELETE CASCADE
                ON UPDATE CASCADE
);
CREATE TABLE Type_weakness (
        type                VARCHAR,
        weakness            VARCHAR,
        PRIMARY KEY (type)
);
CREATE TABLE People_Has (
        pid         INT,
        name        VARCHAR,
        gameID      VARCHAR,
        PRIMARY KEY (pid),
        FOREIGN KEY (gameID) REFERENCES (Game)
                ON DELETE CASCADE
                ON UPDATE CASCADE
);
CREATE TABLE GymMaster_Owns (
        pid                 INT,
        name                VARCHAR,
        badge               VARCHAR,
        owns_since          VARCHAR,
        PRIMARY KEY (pid),
        FOREIGN KEY (badge) REFERENCES Badge_Gym
                ON DELETE CASCADE
                ON UPDATE CASCADE
);
CREATE TABLE Badge_Gym (
        badge               VARCHAR,
        gym#                INT             UNIQUE,
        gameID              INT             UNIQUE,
        PRIMARY KEY (badge),
        FOREIGN KEY (gym#, gameID) REFERENCES Gym_Includes
                ON DELETE CASCADE
                ON UPDATE CASCADE
);
CREATE TABLE NPC_LivesIn (
        pid                     INT,
        name                    VARCHAR,
        role                    VARCHAR,
        gameID                  INT,
        regionName              VARCHAR,
        PRIMARY KEY (pid),
        FOREIGN KEY (gameID) REFERENCES Game
                ON DELETE CASCADE
                ON UPDATE CASCADE,
        FOREIGN KEY (regionName) REFERENCES Region
                ON DELETE CASCADE
                ON UPDATE CASCADE,
        FOREIGN KEY (role) REFERENCES Role_catchPhrase
                ON DELETE CASCADE
                ON UPDATE CASCADE
);
CREATE TABLE Role_catchPhrase (
        role                    VARCHAR,
        catch_phrase            VARCHAR,
        PRIMARY KEY (role)
);
CREATE TABLE Trainer (
        pid                     INT,
        name                    VARCHAR,
        Fav_pokemon             VARCHAR,
        gameID                  INT,
        PRIMARY KEY (pid),
        FOREIGN KEY (gameID) REFERENCES Game
                ON DELETE CASCADE
                ON UPDATE CASCADE
);
CREATE TABLE Quest_Assigned (
        questID                 INT,
        difficulty              VARCHAR,
        pid                     INT,
        date_accepted           VARCHAR,
        PRIMARY KEY (questID),
		FOREIGN KEY (pid) REFERENCES Trainer
			ON DELETE CASCADE
			ON UPDATE CASCADE,
		FOREIGN KEY (difficulty) REFERENCES Difficulty_reward
			ON DELETE CASCADE
			ON UPDATE CASCADE
	);
	CREATE TABLE Difficulty_reward (
		difficulty		VARCHAR,
		reward			VARCHAR,
		PRIMARY KEY (difficulty)
	);
```

8.
INSERT statements to populate each table with at least 5 tuples. You will likely want to have more than 5 tuples so that you can have meaningful queries later.

Items_Has:

```
INSERT INTO Items_Has VALUES(10, “Common”, 0);
INSERT INTO Items_Has VALUES(6, “Rare”, 1);
INSERT INTO Items_Has VALUES(4, “Rare Holo”, 1);
INSERT INTO Items_Has VALUES(3, “Secret Rare”, 3);
INSERT INTO Items_Has VALUES(15, “Common”, 0);
```

Game:

```
INSERT INTO Game VALUES(0, ‘Hard’, ‘Johto’);
INSERT INTO Game VALUES(1, ‘Easy’, ‘Hoenn’);
INSERT INTO Game VALUES(2, ‘Beginner’, ‘Time and Space’);
INSERT INTO Game VALUES(3, ‘Hardcore’ , ‘Truth and Ideals’);
INSERT INTO Game VALUES(4, ‘Medium’, ‘Day and Night’);
```

Gym_Includes:

```
INSERT INTO Gym_Includes VALUES(0, Easy, ‘Water’, 1);
INSERT INTO Gym_Includes VALUES(1, Easy, ‘Ground’, 1);
INSERT INTO Gym_Includes VALUES(2, Medium, ‘Normal’, 2);
INSERT INTO Gym_Includes VALUES(3, Medium, ‘Fire’, 2);
INSERT INTO Gym_Includes VALUES(4, Hard, ‘Electric’, 2);
```

Region_ApartOf:

```
INSERT INTO Region_ApartOf VALUES(‘Celestic City’, ‘City’, 1, 1);
INSERT INTO Region_ApartOf VALUES(‘Jubilife City’, ‘City’, 2, 1);
INSERT INTO Region_ApartOf VALUES(‘Full Moon Island’, ‘Island’, 3, 1);
INSERT INTO Region_ApartOf VALUES(‘Twin Leaf Town’, ‘town’, 1, 2);
INSERT INTO Region_ApartOf VALUES(‘Hearthrome City’, ‘town’, 2, 2);
```

LeadsTo:

```
INSERT INTO LeadsTo VALUES(‘Celestic City’, 1);
INSERT INTO LeadsTo VALUES(‘Celestic City’, 2);
INSERT INTO LeadsTo VALUES(‘Hearthrome City’, 3);
INSERT INTO LeadsTo VALUES(‘Hearthrome City’, 4);
INSERT INTO LeadsTo VALUES(‘Floroma Town’, 5);
```

EnterableAreas:

```
INSERT INTO EnterableAreas VALUES(0, ‘house’);
INSERT INTO EnterableAreas VALUES(1, ‘forest’);
INSERT INTO EnterableAreas VALUES(2, ‘cave’);
INSERT INTO EnterableAreas VALUES(3, ‘store’);
INSERT INTO EnterableAreas VALUES(4, ‘store’);
```

Pokemon_Caught:

```
INSERT INTO Pokemon_Caught VALUES(‘bulbasaur’, ‘grass’, 65, 2020-03-21, 0);
INSERT INTO Pokemon_Caught VALUES(‘ivysaur’, ‘grass’, 70, 2020-04-08, 0);
INSERT INTO Pokemon_Caught VALUES(‘venusaur’, ‘grass’, 75, 2020-05-17, 0);
INSERT INTO Pokemon_Caught VALUES(‘charmander’, ‘fire’, 65, 2020-03-18, 1);
INSERT INTO Pokemon_Caught VALUES(‘charmeleon’, ‘fire’, 70, 2020-04-05, 1);
```

Type_Weakness:

INSERT INTO Type_Weakness VALUES(‘fire’, ‘water’);
INSERT INTO Type_Weakness VALUES(‘water’, ‘grass’);
INSERT INTO Type_Weakness VALUES(‘grass’, ‘fire’);
INSERT INTO Type_Weakness VALUES(‘rock’, ‘fighting’);
INSERT INTO Type_Weakness VALUES(‘bug’, ‘fighting’);

People_Has:

INSERT INTO People_Has VALUES(0, ‘Trainer Jimmy’, 0);
INSERT INTO People_Has VALUES(1, ‘Trainer John’, 0);
INSERT INTO People_Has VALUES(2, ‘Store Owner Bill’, 0);
INSERT INTO People_Has VALUES(3, ‘Nurse Jaimie’, 1);
INSERT INTO People_Has VALUES(4, ‘Gym Master Brock’, 1);

GymMaster_Owns:

INSERT INTO GymMaster_Owns VALUES(0, ‘Gym Master Brock’, ‘Boulder Badge’, 2002-03-21);
INSERT INTO GymMaster_Owns VALUES(1, ‘Gym Master Misty’, ‘Cascade Badge’, 2005-03-10);
INSERT INTO GymMaster_Owns VALUES(2, ‘Gym Master Erika’, ‘Rainbow Badge’, 2001-02-20);
INSERT INTO GymMaster_Owns VALUES(3, ‘Gym Master Sabrina, ‘Marsh Badge’, 2000-01-01);
INSERT INTO GymMaster_Owns VALUES(4, ‘Gym Master Blaine, ‘Volcano Badge’, 2007-01-13);

Badge_Gym:

INSERT INTO Badge_Gym VALUES(‘Boulder Badge’, 0, 0);
INSERT INTO Badge_Gym VALUES(‘Cascade Badge’, 1, 0);

```
INSERT INTO Badge_Gym VALUES(‘Rainbow Badge’, 2, 1);
INSERT INTO Badge_Gym VALUES(‘Marsh Badge’, 3, 2);
INSERT INTO Badge_Gym VALUES(‘Volcano Badge’, 4, 2);
```

NPC_LivesIn:

```
INSERT INTO NPC_LivesIn VALUES(0, ‘Store Owner Jim’, ‘Store Owner’, 0, ‘Celestic City’);
INSERT INTO NPC_LivesIn VALUES(1, ‘Store Owner Bill’, ‘Store Owner’, 0, ‘Celestic City’);
INSERT INTO NPC_LivesIn VALUES(2, ‘Nurse Jenny’, ‘Nurse’, 0, ‘Celestic City’);
INSERT INTO NPC_LivesIn VALUES(3, ‘Nurse Joy’, ‘Nurse’, 1, ‘Floroma Town’);
INSERT INTO NPC_LivesIn VALUES(4, ‘Cyclist Jerry’, ‘Cyclist’, 1, ‘Hearthrome City’);
```

Role_CatchPhrase:

```
INSERT INTO Role_CatchPhrase VALUES(‘Store Owner’, ‘Welcome to the shop!’);
INSERT INTO Role_CatchPhrase VALUES(‘Nurser’, ‘What seems to be the problem?’);
INSERT INTO Role_CatchPhrase VALUES(‘Cyclist’, ‘On your left!’);
INSERT INTO Role_CatchPhrase VALUES(‘Walker’, ‘Hey, slow down!’);
INSERT INTO Role_CatchPhrase VALUES(‘Professor’, ‘Any new information for me today?’);
```

Trainer:

```
INSERT INTO Trainer VALUES(0, ‘Trainer Josh’, ‘Pikachu’, 0);
INSERT INTO Trainer VALUES(1, ‘Trainer Jimmy’, ‘Charizard’, 0);
INSERT INTO Trainer VALUES(2, ‘Trainer Janet’, ‘Psyduck’, 1);
INSERT INTO Trainer VALUES(3, ‘Trainer Mary’, ‘Gible’, 2);
INSERT INTO Trainer VALUES(4, ‘Trainer Will’, ‘Machamp’, 1);
```

Quest_Assigned:

```
INSERT INTO Quest_Assigned VALUES(0, ‘Easy’, 0, 2020-01-13);
INSERT INTO Quest_Assigned VALUES(0, ‘Easy’, 0, 2020-01-13);
INSERT INTO Quest_Assigned VALUES(0, ‘Medium’, 0, 2021-01-12);
INSERT INTO Quest_Assigned VALUES(0, ‘Hard’, 1, 2020-05-18);
INSERT INTO Quest_Assigned VALUES(0, ‘Easy’, 2, 2020-02-10);
```

Difficulty_Reward:

```
INSERT INTO Difficulty_Reward VALUES(‘Easy’, 200);
INSERT INTO Difficulty_Reward VALUES(‘Medium’, 300);
INSERT INTO Difficulty_Reward VALUES(‘Hard’, 400);
INSERT INTO Difficulty_Reward VALUES(‘Super Hard’, 800);
INSERT INTO Difficulty_Reward VALUES(‘Impossible’, 2000);
```