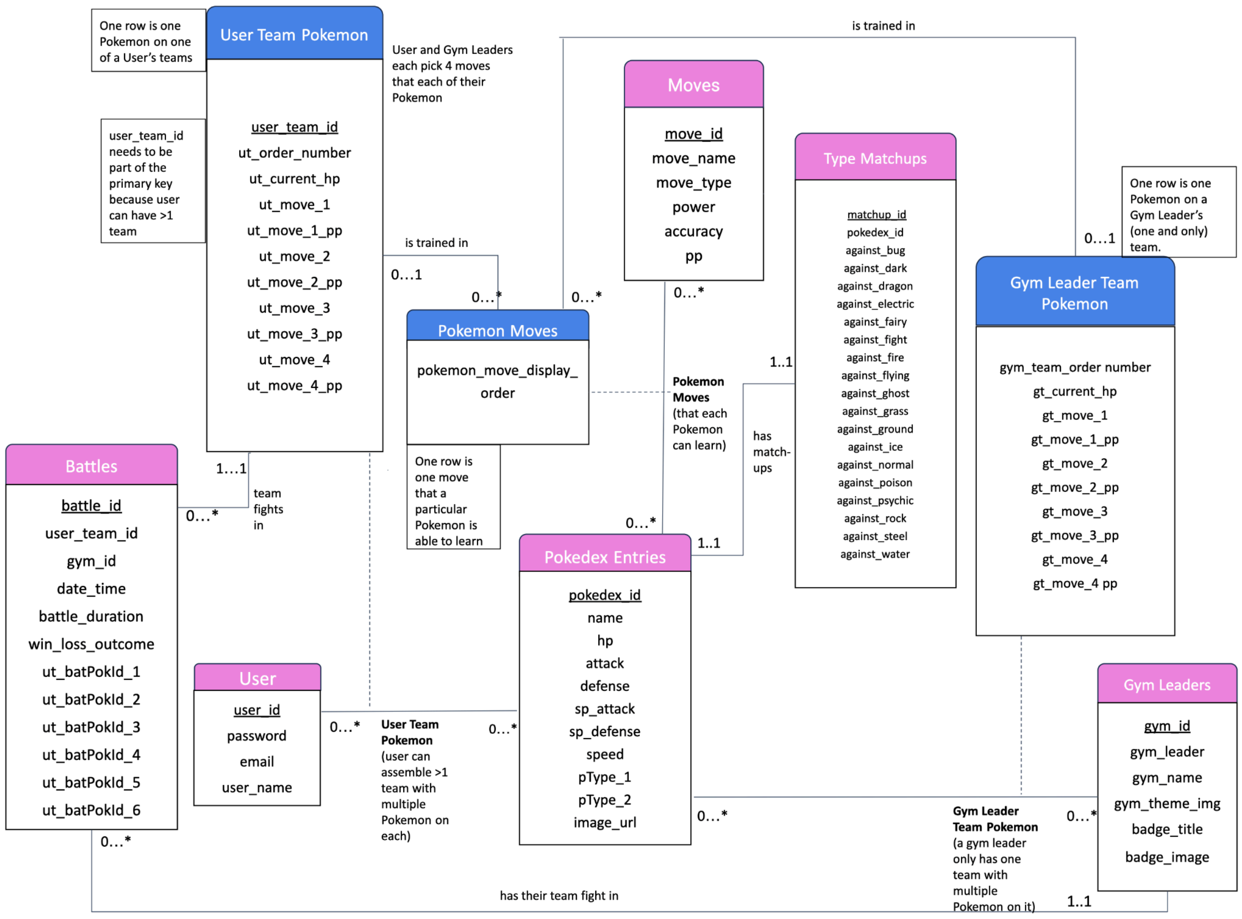
Stage 2: Database Design

I. ER/UML Diagram



II. Assumptions for Entities and Entity Relationships

Entities:

User:
Represents the player's user account in the game. We've modeled this as its own entity, rather than an attribute of another entity because we wanted to separate sensitive information used for authentication of the user and their corresponding game information from the rest of the game.

User Team Pokémon:
Represents the specific Pokémon that a user can have on one of their teams. This entity is a relationship entity representing the many-to-many relationship between User and Pokédex Entries. Because a user can have more than one team, the primary key of this table has three components: user_id from User, Pokédex_id from Pokédex Entries, and user_team_id, which is necessary to uniquely identify an individual Pokémon on a user team. Each row on this table is a single Pokémon that a user has chosen to be on one of their teams. A player can have up to six specific Pokémon on their team. Each Pokémon has its own maximum health value, assigned attack moves, and a predetermined limit on the number of times the Pokémon can use each attack move. We've modeled this as its own entity, rather than an attribute of another entity because we wanted to create something that would represent each row as one Pokémon that is on a user's team. This way, we would know what the corresponding information for that Pokémon was, like what order in the battle it will be played out, its current hp, and their moves.

Moves:
Represents a static record of all the possible moves of all different Pokémon, in the Pokédex (see Pokédex Entries description below). Not every Pokémon can learn every move, and more than one Pokémon are able to learn the same move. This entity will be used to give the user more information about the moves described in the Pokémon Moves relation (a relationship entity that shows the specific moves that each Pokémon in the Pokédex is able to learn). We've modeled this as its own entity, rather than an attribute of another entity because we wanted a static record that all users could look into. In this entity, we have a large static record of all the possible moves that a Pokémon could have. These possible attack moves are available to all Pokémon and all users who add Pokémon to their teams.

Pokémon Moves:
Represents the Pokémon and one of its attack moves. This entity is a relationship entity representing the many-to-many relationship between Moves and Pokédex Entries. We've modeled this as its own entity, rather than an attribute of another entity because we wanted to create

something that would represent each row as a move that, for each Pokémon, that particular Pokémon is able to learn. Similar to the reason for creating the User Team Pokémon relation, we wanted to easily access corresponding information, for each Pokémon's move.

Type Matchups:
Represents a Pokémon's affinity against different Pokémon opponents. Each Pokémon has its own affinity against different elemental Pokémons. Depending on a Pokémon's affinity to an opponent Pokémon's elemental type, they may have a stronger advantage or disadvantage against that opponent Pokémon. This will affect the amount of damage, or how much the Pokémon's hp lowers, taken by the Pokémon inflicted by the opponent. We've modeled this as its own entity, rather than an attribute of another entity because, like the moves entity, we wanted a static record that would determine the elemental affinity of all the Pokémons in the Pokédex. Due to the large amount of possible elemental affinities, we decided to separate the type matchups attributes from Pokédex entries.

Gym Leader Team Pokémon:
Represents the specific Pokémon that a specific gym leader can have on their team. This entity is a relationship entity representing the many-to-many relationship between Gym Leaders and Pokédex Entries. A gym leader only has one team and can have up to six specific Pokémon on their team. Each Pokémon has its own maximum health value, assigned attack moves, and a predetermined limit on the number of times each Pokémon can use each attack move. The users will use their Pokémon teams to fight against the team of each Gym Leader. We've modeled this as its own entity, rather than an attribute of another entity because we wanted to create something that would represent each row as one Pokémon that is on a Gym Leader's team. This way, we would know what the corresponding information for that Pokémon was, like what order in the battle it will be played out, its current hp, and their moves.

Gym Leaders:
Represents unique gyms, where each gym has its own gym leader and gym team of Pokémon. Along with this, when a gym is defeated, users are able to obtain the badge associated with that gym, as proof of their

valiant fight. We’ve modeled this as its own entity, rather than an attribute of another entity because each gym will have its own unique gym leader, name, theme, image, and badge. We wanted a record in the gym leaders' relation to represent one gym leader person.

Pokédex Entries:
Represents a static record of all the possible Pokémon a user can choose from to create a team. This also represents a static record from which the gym leader’s team members will come from. Each Pokémon has its own descriptive attributes: name, maximum health (hp), attack power (attack), defense power (defense), special attack power (sp_attack), special defense power (sp_defense), attack speed (speed), Pokémon type 1 (pType_1), Pokémon type 2 (pType_2), and image of themselves (image_url). We’ve modeled this as its own entity, rather than an attribute of another entity because we wanted to have a large static record of all the possible Pokémon that a user could choose from, to add to their team. Alongside this, the gym leaders’ teams will only be made up of Pokémon that are in the same Pokédex.

Battles:
Represents a battle that the user initiated against a gym. This will tell us the results of that battle and the team the user used to fight in the gym. We’ve modeled this as its own entity, rather than an attribute of another entity because we wanted each record to represent a battle that a user has fought. Each record would store corresponding statistical information like the battle duration and battle outcome.

<u>Relationships</u> (hyphens indicate relationships)
User - Pokédex Entries: Many-to-many relationship. This is represented by the relationship entity User Team Pokémon. A user can choose from many different Pokémon in the Pokédex to assemble their team. Users can form more than one team, and have up to six Pokémon from the Pokédex Entries to make up each team.
Pokédex Entries - User: Many-to-many relationship. A Pokémon can be chosen from many different types of users, when the users are assembling their Pokémon team. This allows all users of our game to choose from the same pool of Pokémon.

User Team Pokémon - Pokémon Moves: At-most-one to many relationship. A Pokémon can be trained in many different attack moves, via choosing from a large pool of different moves.
Pokémon Moves - User Team Pokémon: Many to at-most-one relationship. A Pokémon move will be assigned to a user's team Pokémon, if that move is chosen for that Pokémon. Not all the Pokémon moves in the pool of Pokémon moves will be chosen for the Pokémon nor will they all be available during battle for the Pokémon.

Pokémon Moves - Gym Leader Team Pokémon: Many to at-most-one relationship. A Pokémon move will be assigned to a gym leader's team Pokémon, if that move is chosen for that Pokémon. Not all the Pokémon moves in the pool of Pokémon moves will be chosen for the Pokémon nor will they all be available during battle for the Pokémon.
Gym Leader Team Pokémon - Pokémon Moves: At-most-one to many relationship. A gym leader team Pokémon can be trained in many attack moves, choosing from a large pool of different moves.

Pokédex Entries - Moves: Many-to-many relationship. This is represented by the relationship entity Pokémon Moves. A Pokémon in the Pokédex can learn many different moves, choosing from a large pool of moves.
Moves - Pokédex Entries: Many-to-many relationship. A move can be learned by a large number of different Pokémon in the Pokédex.

Pokédex Entries - Type Matchups: One-to-one relationship. A Pokémon in the Pokédex will have one type matchup, which will determine their affinity to different elements. This will come into play when this Pokémon battles against other Pokémons with similar or contrasting elemental affinities.
Type Matchups - Pokédex Entries: One-to-one relationship. A type matchup will be assigned to one Pokémon in the Pokédex. This will come into play when this Pokémon battles against other Pokémons with similar or contrasting elemental affinities.

Pokédex Entries - Gym Leaders: Many-to-many relationship. This is represented by the relationship entity Gym Leader Team Pokémon. Many

Pokémon from the Pokédex can be chosen, or assembled into a team, by many different gym leaders.
Gym Leaders - Pokédex Entries: Many-to-many relationship. Many different gym leaders can assemble a gym team by choosing from many different Pokémon in the Pokédex.

Gym Leaders - Battles: One-and-only-one to many. Many different gym leaders can have their Pokémon team fight in many different battles. This allows users to replay a certain battle with the gym leader, especially if that battle resulted in a loss.
Battles - Gym Leaders: Many to one-and-only-one relationship. A battle will be associated with one gym leader. Each win with each gym leader will result in a unique badge being earned by the user.

Battles - User Team Pokémon: Many to one-and-only-one relationship. A battle is fought by one team of Pokémon, created by the user.
User Team Pokémon – Battles: One-and-only-one to many relationship. A user's team of Pokémon can fight in multiple battles. For example, the user can use the same team to fight subsequent battles with gym leaders.

III. Normalize Database

We will go through each entity of our database, list its functional dependencies, and decompose it into 3NF.

Entity: User

FDs:
user_id -> password, email, username

Candidate Keys:

left	middle	right	none
user_id		password, email, username	

What appears only on left-hand side and on neither side: user_id
From FDs,user_id gives you password, email, username.
Therefore, user_id is the only candidate key, which makes sense because we have chosen it as our primary key.

To normalize this schema to 3NF:

1. Conversion to minimal basis:
   a. Only singletons on the right hand side.

user_id -> password
user_id -> email
user_id -> username

   b. Remove unnecessary attributes from left hand side

There are no FDs with more than one attribute on the left hand side, so this step is already done.

   c. Remove functional dependencies that can be inferred from the rest.

None of these FDs can be inferred from any of the others, so there are no FDs to remove.

Combining FDs with the same left-hand side, together, this gives us the minimal basis:

user_id -> password, email, username

2. For each functional dependency A -> B in the minimal basis, use AB as the schema of a new relation.

This gives back the User entity that we had originally.

User(user_id, password, email, username)

3. If none of the schemas from Step 2 is a superkey, add another relation whose schema is a key for the original relation.

There is no need to add any other relation, because User contains the attribute user_id, which is a key for the entire relation.

<u>Entity: User Team Pokémon</u>

(This is a relationship entity for the many-to-many relationship between User and Pokédex Entries, so the table for it will include user_id and Pokédex_id as part of its primary key. Also, since users are able to have more than one team, user_team_id from this entity will also be needed as a third component of the primary key to uniquely identify an individual Pokémon on one of a user's teams. One row on this table does not represent an entire team but instead an individual Pokémon on one of a user's teams.)

FDs:
user_id, Pokédex_id, user_team_id -> ut_order_number, ut_current_hp, ut_move_1, ut_move_1_pp, ut_move_2, ut_move_2_pp, ut_move_3, ut_move_3_pp, ut_move_4, ut_move_4_pp

(Note that ut_move_1_pp is not functionally dependent on ut_move_1, etc., since move pp is depleted as moves are used in battle and this attribute is a current battle status.)

Candidate Keys:

left	middle	right	none
user_id, Pokédex_id, user_team_id		ut_order_number, ut_current_hp, ut_move_1, ut_move_1_pp, ut_move_2, ut_move_2_pp, ut_move_3, ut_move_3_pp, ut_move_4,	

		ut_move_4_pp	

What appears only on left-hand side and on neither side: user_id, Pokédex_id, user_team_id

From FDs,user_id, Pokédex_id, user_team_id gives you ut_order_number, ut_current_hp, ut_move_1, ut_move_1_pp, ut_move_2, ut_move_2_pp, ut_move_3, ut_move_3_pp, ut_move_4, ut_move_4_pp

Therefore, user_id, Pokédex_id, user_team_id is the only candidate key, which makes sense because we have chosen it as our primary key.

To normalize this schema to 3NF:

1. *Conversion to minimal basis:*
    a. *Only singletons on the right hand side.*

user_id, Pokédex_id, user_team_id -> ut_order_number
user_id, Pokédex_id, user_team_id -> ut_current_hp,
user_id, Pokédex_id, user_team_id -> ut_move_1
user_id, Pokédex_id, user_team_id -> ut_move_1_pp
user_id, Pokédex_id, user_team_id -> ut_move_2
user_id, Pokédex_id, user_team_id -> ut_move_2_pp
user_id, Pokédex_id, user_team_id -> ut_move_3
user_id, Pokédex_id, user_team_id -> ut_move_3_pp
user_id, Pokédex_id, user_team_id -> ut_move_4
user_id, Pokédex_id, user_team_id -> ut_move_4_pp

    b. *Remove unnecessary attributes from left hand side*

All FDs have the same three attributes on the left hand side: user_id, Pokédex_id, user_team_id. *Since all three of these are needed to uniquely give you each of the other attributes in this relation, none of them can be removed from any of the FDs.*

    c. *Remove functional dependencies that can be inferred from the rest.*

None of these FDs can be inferred from any of the others, so there are no FDs to remove.

Combining FDs with the same left-hand side, together, this gives us the minimal basis:

user_id, Pokédex_id, user_team_id -> ut_order_number, ut_current_hp, ut_move_1, ut_move_1_pp, ut_move_2, ut_move_2_pp, ut_move_3, ut_move_3_pp, ut_move_4, ut_move_4_pp

2. For each functional dependency A -> B in the minimal basis, use AB as the schema of a new relation.

This gives us the original User Team Pokémon entity:
User Team Pokémon(user_id, Pokédex_id, user_team_id, ut_order_number, ut_current_hp, ut_move_1, ut_move_1_pp, ut_move_2, ut_move_2_pp, ut_move_3, ut_move_3_pp, ut_move_4, ut_move_4_pp)

3. If none of the schemas from Step 2 is a superkey, add another relation whose schema is a key for the original relation.

There is no need to add any other relation, because User Team Pokémon contains the attributes user_id, user_id, and Pokédex_id, user_team_id which together are a key for the entire relation.

<u>Entity: Pokédex Entries</u>

FDs:
Pokédex_id -> name, hp, attack, defense, sp_attack, sp_defense, speed, pType_1, pType_2, image_url
name -> Pokédex_id, hp, attack, defense, sp_attack, sp_defense, speed, pType_1, pType_2, image_url
image_url -> Pokédex_id, name, hp, attack, defense, sp_attack, sp_defense, speed, pType_1, pType_2

Candidate Keys:

left	middle	right	none
	Pokédex_id, name, image_url	hp, attack, defense, sp_attack, sp_defense, speed, pType_1, pType_2	

What appears only on left-hand side and on neither side: no attributes
What appears in the middle: Pokédex_id, name, image_url

From FDs, Pokédex_id, name, image_url each give you all of the other attributes in the relation, so they are all candidate keys, although we have chosen Pokédex_id to be our primary key.

To normalize this schema to 3NF:

1. Conversion to minimal basis:
   d. Only singletons on the right hand side.

Pokédex_id -> name
Pokédex_id -> hp
Pokédex_id -> attack
Pokédex_id -> defense
Pokédex_id -> sp_attack
Pokédex_id -> sp_defense
Pokédex_id -> speed
Pokédex_id -> pType_1
Pokédex_id -> pType_2
Pokédex_id -> image_url
name -> Pokédex_id
name -> hp
name -> attack
name -> defense
name -> sp_attack
name -> sp_defense

```
name -> speed
name -> pType_1
name -> pType_2
name -> image_url
image_url -> Pokédex_id
image_url -> name
image_url -> hp
image_url -> attack
image_url -> defense
image_url -> sp_attack
image_url -> sp_defense
image_url -> speed
image_url -> pType_1
image_url -> pType_2
```

e. Remove unnecessary attributes from left hand side

There are no FDs with more than one attribute on the left hand side, so this step is already done.

f. Remove functional dependencies that can be inferred from the rest.

There is a circular set of FDs with Pokédex_id -> name -> image_url -> Pokédex_id -> image_url -> name -> Pokédex_id, *etc. All three of these attributes are candidate keys for the entire relation, as shown. This circularity can be removed by simply retaining the FDs with the primary key we have chosen, Pokédex_id, on the left hand side.*

```
Pokédex_id -> name
Pokédex_id -> hp
Pokédex_id -> attack
Pokédex_id -> defense
Pokédex_id -> sp_attack
Pokédex_id -> sp_defense
Pokédex_id -> speed
Pokédex_id -> pType_1
Pokédex_id -> pType_2
```

Pokédex_id -> image_url

Combining FDs with the same left-hand side, together, this gives us the minimal basis:

Pokédex_id -> name, hp, attack, defense, sp_attack, sp_defense, speed, pType_1, pType_2, image_url

2. For each functional dependency A -> B in the minimal basis, use AB as the schema of a new relation.

This gives back the Pokédex Entries entity that we had originally.

Pokédex Entries(Pokédex_id, name, hp, attack, defense, sp_attack, sp_defense, speed, pType_1, pType_2, image_url)

3. If none of the schemas from Step 2 is a superkey, add another relation whose schema is a key for the original relation.

There is no need to add any other relation, because Pokédex_id contains the attributes Pokédex_id, name, and image_url, each of which is a candidate key for the entire relation.

We can confirm this particular relation is in 3NF from the definition of 3NF: "A relation R is in 3rd normal form if : Whenever there is a nontrivial dependency A1, A2, ..., An -> B for R, then {A1, A2, ..., An } is a super-key for R, OR B is part of a key."
(from pre-lecture slides for this class)

Looking at the FD that is our minimal basis:

Pokédex_id -> name, hp, attack, defense, sp_attack, sp_defense, speed, pType_1, pType_2, image_url

The left hand side, Pokédex_id, is a candidate key for the relation. This is also true for all of the FDs we looked at earlier where name or image_url were on the left hand side. They are also candidate keys for the relation. Therefore, all FDs for this relation show that the relation is in 3NF.

Entity: Gym Leader Team Pokémon

(This is a relationship entity for the many-to-many relationship between Gym Leader and Pokédex Entries, so the table for it will include gym_id and Pokédex_id as part of its primary key. Also note that each gym leader has only one team.)

FDs:
gym_id, Pokédex_id -> gym_team_order_number, gt_current_hp, gt_move_1, gt_move_1_pp, gt_move_2, gt_move_2_pp, gt_move_3, gt_move_3_pp, gt_move_4, gt_move_4_pp

(Note that gt_move_1_pp is not functionally dependent on gt_move_1, etc., since move pp is depleted as moves are used in battle and this attribute is a current battle status.)

Candidate Keys:

left	middle	right	none
gym_id, Pokédex_id,		gym_team_order_number, gt_current_hp, gt_move_1, gt_move_1_pp, gt_move_2, gt_move_2_pp, gt_move_3, gt_move_3_pp, gt_move_4, gt_move_4_pp ut_move_4_pp	

What appears only on left-hand side and on neither side: gym_id, Pokédex_id

From FDs, gym_id, Pokédex_id gives you gym_team_order_number, gt_current_hp, gt_move_1, gt_move_1_pp, gt_move_2, gt_move_2_pp, gt_move_3, gt_move_3_pp, gt_move_4, gt_move_4_pp (ie, all other attributes)

Therefore, gym_id, Pokédex_id is the only candidate key, which makes sense because it is our primary key for this relation.

To normalize this schema to 3NF:

2. *Conversion to minimal basis:*
    d. *Only singletons on the right hand side.*

gym_id, Pokédex_id -> gt_order_number
gym_id, Pokédex_id -> gt_current_hp,
gym_id, Pokédex_id -> gt_move_1
gym_id, Pokédex_id -> gt_move_1_pp
gym_id, Pokédex_id -> gt_move_2
gym_id, Pokédex_id -> gt_move_2_pp
gym_id, Pokédex_id -> gt_move_3
gym_id, Pokédex_id -> gt_move_3_pp
gym_id, Pokédex_id -> gt_move_4
gym_id, Pokédex_id -> gt_move_4_pp

    e. *Remove unnecessary attributes from left hand side*

All FDs have the same two attributes on the left hand side: gym_id, Pokédex_id. *Since both of these are needed to uniquely give you each of the other attributes in this relation, none of them can be removed from any of the FDs.*

    f. *Remove functional dependencies that can be inferred from the rest.*

None of these FDs can be inferred from any of the others, so there are no FDs to remove.

Combining FDs with the same left-hand side, together, this gives us the minimal basis:

gym_id, Pokédex_id -> gym_team_order_number, gt_current_hp, gt_move_1, gt_move_1_pp, gt_move_2, gt_move_2_pp, gt_move_3, gt_move_3_pp, gt_move_4, gt_move_4_pp

2. For each functional dependency A -> B in the minimal basis, use AB as the schema of a new relation.

This gives us the original Gym Leader Team Pokémon entity:
Gym Leader Team Pokémon(gym_id, Pokédex_id, gym_team_order_number, gt_current_hp, gt_move_1, gt_move_1_pp, gt_move_2, gt_move_2_pp, gt_move_3, gt_move_3_pp, gt_move_4, gt_move_4_pp)

3. If none of the schemas from Step 2 is a superkey, add another relation whose schema is a key for the original relation.

There is no need to add any other relation, because Gym Leader Team Pokémon contains the attributes gym_id and Pokédex_id, which together are a key for the entire relation.

Entity: Gym Leaders

FDs:
gym_id -> gym_leader, gym_name, gym_theme_img, badge_title, badge_image

(Here we have a similar situation that we have with Pokédex Entries: we have multiple candidate keys. In fact, since every attribute for this entity is not null and unique, all attributes for this entity are candidate keys. Rather than list functional dependencies with each attribute on its own on the left hand side and all other attributes on the right hand side, as we did for Pokédex Entries, we will only list this FD with the chosen primary key on the left hand side, since the circular nature every attribute being a candidate key that has an FD that gives you all other attributes will result in us reducing down to this single original FD anyway in the process of normalizing to 3NF.)

Candidate Keys:

left	middle	right	none
gym_id		gym_leader, gym_name, gym_theme_img, badge_title, badge_image	

What appears only on left-hand side and on neither side: gym_id
From FDs, gym_id gives you gym_leader, gym_name, gym_theme_img, badge_title, badge_image.
Therefore, gym_id is a candidate key, which makes sense because we have chosen it as our primary key. In fact, as stated above, for this relation, all attributes are a candidate key.

To normalize this schema to 3NF:

1. Conversion to minimal basis:
   g. Only singletons on the right hand side.

gym_id -> gym_leader
gym_id -> gym_name
gym_id -> gym_theme_img
gym_id -> badge_title
gym_id -> badge_image

   h. Remove unnecessary attributes from left hand side

There are no FDs with more than one attribute on the left hand side, so this step is already done.

   i. Remove functional dependencies that can be inferred from the rest.

The circularity of FDs resulting from every attribute being a candidate key results in us eliminating all FDs that do not have the primary key on the left hand side, which this time,we have already done.

Combining FDs with the same left-hand side, together, this gives us the minimal basis:

gym_id -> gym_leader, gym_name, gym_theme_img, badge_title, badge_image

2. For each functional dependency A -> B in the minimal basis, use AB as the schema of a new relation.

This gives back the Gym Leaders entity that we had originally.

Gym Leaders(gym_id, gym_leader, gym_name, gym_theme_img, badge_title, badge_image)

3. If none of the schemas from Step 2 is a superkey, add another relation whose schema is a key for the original relation.

There is no need to add any other relation, because every attribute in Gym Leaders is a candidate key for the entire relation.

<u>Entity: Battles</u>

FDs:
battle_id -> user_team_id, gym_id, date_time, battle_duration, win_loss_outcome, ut_batPokId_1, ut_batPokId_2, ut_batPokId_3, ut_batPokId_4, ut_batPokId_5, ut_batPokId_6

Candidate Keys:

left	middle	right	none
battle_id		user_team_id, gym_id, date_time, battle_duration, win_loss_outcome, ut_batPokId_1, ut_batPokId_2, ut_batPokId_3,	

		ut_batPokId_4, ut_batPokId_5, ut_batPokId_6	

What appears only on left-hand side and on neither side: battle_id
From FDs, battle_id gives you user_team_id, gym_id, date_time, battle_duration, win_loss_outcome, ut_batPokId_1, ut_batPokId_2, ut_batPokId_3, ut_batPokId_4, ut_batPokId_5, ut_batPokId_6.
Therefore, battle_id is the only candidate key, which makes sense because we have chosen it as our primary key.

To normalize this schema to 3NF:

1. Conversion to minimal basis:
j. Only singletons on the right hand side.

battle_id -> user_team_id
battle_id -> gym_id
battle_id -> date_time
battle_id -> battle_duration
battle_id -> win_loss_outcome
battle_id -> ut_batPokId_1
battle_id -> ut_batPokId_2
battle_id -> ut_batPokId_3
battle_id -> ut_batPokId_4
battle_id -> ut_batPokId_5
battle_id -> ut_batPokId_6

k. Remove unnecessary attributes from left hand side

There are no FDs with more than one attribute on the left hand side, so this step is already done.

l. Remove functional dependencies that can be inferred from the rest.

None of these FDs can be inferred from any of the others, so there are no FDs to remove.

Combining FDs with the same left-hand side, together, this gives us the minimal basis:

battle_id -> user_team_id, gym_id, date_time, battle_duration, win_loss_outcome, ut_batPokId_1, ut_batPokId_2, ut_batPokId_3, ut_batPokId_4, ut_batPokId_5, ut_batPokId_6

2. For each functional dependency A -> B in the minimal basis, use AB as the schema of a new relation.

This gives back the Battles entity that we had originally.

Battles(battle_id, user_team_id, gym_id, date_time, battle_duration, win_loss_outcome, ut_batPokId_1, ut_batPokId_2, ut_batPokId_3, ut_batPokId_4, ut_batPokId_5, ut_batPokId_6)

3. If none of the schemas from Step 2 is a superkey, add another relation whose schema is a key for the original relation.

There is no need to add any other relation, because Battles contains the attribute battle_id, which is a key for the entire relation.

<u>Entity: Type Matchups</u>

FDs:
matchup_id -> Pokédex_id, against_bug, against_dark, against_dragon, against_electric, against_fairy, against_fight, against_fire, against_flying, against_ghost, against_grass, against_ground, against_ice, against_normal, against_poison, against_psychic, against_rock, against_steel, against_water

Candidate Keys:

left	middle	right	none
matchup_id		Pokédex_id, against_bug,	

		against_dark, against_dragon, against_electric, against_fairy, against_fight, against_fire, against_flying, against_ghost, against_grass, against_ground, against_ice, against_normal, against_poison, against_psychic, against_rock, against_steel, against_water	

What appears only on left-hand side and on neither side: matchup_id
From FDs, matchup_id gives you Pokédex_id, against_bug, against_dark, against_dragon, against_electric, against_fairy, against_fight, against_fire, against_flying, against_ghost, against_grass, against_ground, against_ice, against_normal, against_poison, against_psychic, against_rock, against_steel, against_water.

To normalize this schema to 3NF:

1. Conversion to minimal basis:
   m. Only singletons on the right hand side.

matchup_id -> Pokédex_id
matchup_id -> against_bug
matchup_id -> against_dark
matchup_id -> against_dragon
matchup_id -> against_electric
matchup_id -> against_fairy
matchup_id -> against_fight
matchup_id -> against_fire

matchup_id -> against_flying
matchup_id -> against_ghost
matchup_id -> against_grass
matchup_id -> against_ground
matchup_id -> against_ice
matchup_id -> against_normal
matchup_id -> against_poison
matchup_id -> against_psychic
matchup_id -> against_rock
matchup_id -> against_steel
matchup_id -> against_water

n. Remove unnecessary attributes from left hand side

There are no FDs with more than one attribute on the left hand side, so this step is already done.

o. Remove functional dependencies that can be inferred from the rest.

None of these FDs can be inferred from any of the others, so there are no FDs to remove.

Combining FDs with the same left-hand side, together, this gives us the minimal basis:

matchup_id -> Pokédex_id, against_bug, against_dark, against_dragon, against_electric, against_fairy, against_fight, against_fire, against_flying, against_ghost, against_grass, against_ground, against_ice, against_normal, against_poison, against_psychic, against_rock, against_steel, against_water

2. For each functional dependency A -> B in the minimal basis, use AB as the schema of a new relation.

This gives back the Type Matchups entity that we had originally.

Type Matchups(matchup_id, Pokédex_id, against_bug, against_dark, against_dragon, against_electric, against_fairy, against_fight, against_fire, against_flying, against_ghost, against_grass, against_ground, against_ice, against_normal, against_poison, against_psychic, against_rock, against_steel, against_water)

3. If none of the schemas from Step 2 is a superkey, add another relation whose schema is a key for the original relation.

There is no need to add any other relation, because Type Matchups contains the attribute matchup_id, which is a key for the entire relation.

<u>Entity: Moves</u>

FDs:
move_id -> move_name, move_type, power, accuracy, pp

Candidate Keys:

left	middle	right	none
move_id		move_name, move_type, power, accuracy, pp	

What appears only on left-hand side and on neither side: move_id
From FDs,move_id gives you move_name, move_type, power, accuracy, pp.
Therefore, move_id is the only candidate key, which makes sense because we have chosen it as our primary key.

To normalize this schema to 3NF:

1. Conversion to minimal basis:
   p. Only singletons on the right hand side.

move_id -> move_name

move_id -> move_type
move_id -> power
move_id -> accuracy
move_id -> pp

q. Remove unnecessary attributes from left hand side

There are no FDs with more than one attribute on the left hand side, so this step is already done.

r. Remove functional dependencies that can be inferred from the rest.

None of these FDs can be inferred from any of the others, so there are no FDs to remove.

Combining FDs with the same left-hand side, together, this gives us the minimal basis:

move_id -> move_name, move_type, power, accuracy, pp

2. For each functional dependency A -> B in the minimal basis, use AB as the schema of a new relation.

This gives back the Moves entity that we had originally.

Moves(move_id, move_name, move_type, power, accuracy, pp)

3. If none of the schemas from Step 2 is a superkey, add another relation whose schema is a key for the original relation.

There is no need to add any other relation, because Moves contains the attribute move_id, which is a key for the entire relation.

<u>Entity: Pokémon moves</u>

(This is a relationship entity for the many-to-many relationship between Moves and Pokédex Entries, so the table for it will include move_id and Pokédex_id as part of its primary key. One row on this

table represents a single move that a Pokémon is able to learn (not every Pokémon can learn every move, and more than one Pokémon are able to learn the same move.)

FDs:
Pokédex_id, move_id -> Pokémon_move_display_order

Candidate Keys:

left	middle	right	none
Pokédex_id, move_id		Pokémon_move_display_order	

What appears only on left-hand side and on neither side: Pokédex_id, move_id

From FDs,Pokédex_id, move_id gives you Pokémon_move_display_order.

Therefore, Pokédex_id, move_id is the only candidate key, which makes sense because it is the primary key for this relation.

To normalize this schema to 3NF:

3. *Conversion to minimal basis:*
    g. *Only singletons on the right hand side.*

Pokédex_id, move_id -> Pokémon_move_display_order

h. *Remove unnecessary attributes from left hand side*

The only FD has two attributes on the left hand side: Pokédex_id, move_id. *Since both of these are needed to uniquely give you the only other attribute in this relation,* Pokémon_move_display_order, *neither of them can be removed from this FD.*

i. *Remove functional dependencies that can be inferred from the rest.*

There is only one FD, so there are no other FDs that can be inferred from it to remove.

Combining FDs with the same left-hand side, together, this gives us the minimal basis:

Pokédex_id, move_id -> Pokémon_move_display_order

2. For each functional dependency A -> B in the minimal basis, use AB as the schema of a new relation.

This gives us the original User Team Pokémon entity:
Pokémon Moves(Pokédex_id, move_id, Pokémon_move_display_order)

3. If none of the schemas from Step 2 is a superkey, add another relation whose schema is a key for the original relation.

There is no need to add any other relation, because Pokémon Moves contains the attributes Pokédex_id and move_id which together are a key for the entire relation.

IV. Logical Design

- User (user_id:VARCHAR(30)[PK], password:VARCHAR(255), email:VARCHAR(100), user_name:VARCHAR(30))
- User Team Pokémon (user_id:VARCHAR(30)[PK],ut_Pokédex_id:INT [PK], user_team_id:INT[PK], ut_team_member_id:INT[PK], ut_current_hp:REAL, ut_move_1:VARCHAR(100), ut_move_1_pp:INT,ut_move_2:VARCHAR(100), ut_move_2_pp:INT,ut_move_3:VARCHAR(100), ut_move_3_pp:INT,ut_move_4:VARCHAR(100), ut_move_4_pp:INT)
- Pokémon Moves (Pokédex_id:INT[PK], move_id:INT [PK], Pokémon_move_id:INT [PK])

- Moves (move_id:INT[PK], move_name:VARCHAR(30), move_type:VARCHAR(30), power:INT, accuracy:FLOAT, pp:INT)
- Type Matchups (matchup_id:INT[PK], Pokédex_id:INT, against_bug:REAL, against_dark:REAL, against_dragon:REAL, against_electric:REAL, against_fairy:REAL, against_fight:REAL, against_fire:REAL, against_flying:REAL, against_ghost:REAL, against_grass:REAL, against_ground:REAL, against_ice:REAL, against_normal:REAL, against_poison:REAL, against_psychic:REAL, against_rockV, against_steel:REAL, against_water:REAL)
- Pokédex Entries (Pokédex_id:INT[PK], name:VARCHAR(30), hp:INT, attack:INT, defense: INT, sp_attack:INT, sp_defense:INT, speed:INT,pType_1:VARCHAR(30), pType_2:VARCHAR(30), image_url:VARCHAR(255))
- Gym Leader Team Pokémon (gym_id:INT[PK], gt_Pokédex_id:INT [PK], gym_team_member_id:INT[PK], gt_current_hp:INT,gt_move_1:VARCHAR(100), gt_move_1_pp:INT,gt_move_2:VARCHAR(100), gt_move_2_pp:INT,gt_move_3:VARCHAR(100), gt_move_3_pp:INT,gt_move_4:VARCHAR(100), gt_move_4_pp:INT)
- Gym Leaders (gym_id:INT[PK], gym_leader:VARCHAR(30), gym_name:VARCHAR(30), gym_theme_img:VARCHAR(255), badge_title:VARCHAR(30), badge_image:VARCHAR(255))
- Battles (battle_id:INT[PK], user_team_id:INT, gym_id:INT, date_time:DATE_TIME, battle_duration:REAL, win_loss_outcome:VARCHAR(30), ut_batPokId_1:INT, ut_batPokId_2:INT, ut_batPokId_3:INT, ut_batPokId_4:INT, ut_batPokId_5:INT, ut_batPokId_6:INT)

V. Fixed suggestions from Stage 1

This is the feedback received from Stage 1:

1. The image dataset wouldn't perhaps allow you much of database operations. Can you see if you can find another real dataset? Maybe that of GYM leaders?

2. You are encouraged to balance out tasks that involve DB operations a bit more evenly.

<u>Here is how we addressed the first feedback statement:</u>

In response to the feedback on our proposal, we have decided to include two additional datasets (in addition to the two we included from our proposal) from Kaggle:
https://www.kaggle.com/datasets/lunamcbride24/Pokémon-type-matchup-data
https://www.kaggle.com/datasets/n2cholas/competitive-Pokémon-dataset

The first, which includes Pokémon Type Matchup Data, will be used to create a richer and more accurate battle simulation by including elemental type multipliers for each Pokémon. The second, the Competitive Pokémon Dataset, includes key battle attributes like pp (power point move counters) which are not present in our original datasets. We have made additional changes to our UML diagram and entity relationships to reflect the addition of this data.

Here is how we addressed the second feedback statement:
We have decided to split the work streams for this project up by feature instead of specific technical role. Each of our team members will complete the end-to-end code (including database work, backend, API, and frontend) for each of the important main features for our game. Below is the revised project work distribution overview:

Project Work Distribution Overview

- Design of database, QA Testing
  - *Person(s): Everyone*
  - Create UML, write SQL queries, indexing, testing
- Data Acquisition
  - *Person(s): Everyone*
  - Retrieve data from Kaggle Pokémon datasets and input into project databases
- Full Stack Engineer
  - *Person: Mengmeng Fang*
  - Pokédex, Pokémon Team Selection
- Full Stack Engineer
  - *Person: Pierre Font*
  - Battling, Battle Mechanics

- Full Stack Engineer & UI/UX Designer
    - *Person: Hannah Kim*
    - User Profile, User Battle Summary
- Team Captain & Project Manager
    - *Person: Hannah Kim*
    - Manages division of tasks, keeps track of milestones, works to maintain team cohesion
- Tech Lead & Full Stack Engineer
    - *Person: Tanjie McMeans*
    - Technical mentorship, lead code reviews, resolving technical blockers, Gym Leaders, Battle AI