Final Report: PokeDB

Sofia Fong, Sophia Pietsch, Kai Sun, Ethan Zhang

Table of Contents

Description of the Application

The users of the application would be Pokemon fans who are looking for information about a specific Pokemon or team suggestions. They can use our application to keep track of the Pokemon they own in some Pokemon game. Using the pokemon that a user owns, the application will utilize past battle statistics to create potential teams for the user. The administrators of the database systems would be the members of this team project.

System support

Our application uses a PostgreSQL database which is set up using an SQL script. We use Python, in particular the pandas library, to import data into the database. Our application uses a Python Flask backend, which uses psycopg2 to communicate with the database. Our frontend uses Vue.js and communicates with the backend using HTTP requests. The front-end consists of several pages the user can navigate and uses Vuex to persistently store information about the user's session.

Database Design

Assumptions about Data

Users:

- Usernames are unique.
- A user can own any number of pokemon.

Pokemon:

- The pokemonID is unique. The name of a pokemon is also unique.
- All base stats (HP, Spd, Atk, Def, SpAtk, SpDef) are greater than or equal to zero.
- A pokemon species has one or two possible abilities and one or two types. If a pokemon species evolves from another species, then it evolves from exactly one other species.

Owned Pokemon:

- The ownedID of owned pokemon is unique.
- All stats (HP, Spd, Atk, Def, SpAtk, SpDef) and the base versions of them are greater than or equal to zero.
- An owned pokemon belongs to exactly one user.
- An owned pokemon is of exactly one species and has exactly one ability. It can have between one and four moves that can be learned by the species. These moves should be unique. The ability must also be one of the abilities that the species can have.
- An owned pokemon's level is between 1 and 100.
- The gender of an owned pokemon is either male, female or unknown.

Moves:

- The moveName of moves is unique.
- The effectiveness of a move against a specific type is one of 0, 0.5, 1 or 2. The effectiveness of a move against a specific type combination (A, B) is obtained by multiplying its effectiveness against A and its effectiveness against B, and is one of 0, 0.25, 0.5, 1, 2, or 4
- The power of a move is between 0 and 250 and a multiple of 5.
- The accuracy of a move is between 0 and 100 and a multiple of 5.

- A move has exactly one type.
- The damage type of a move is either physical or special.

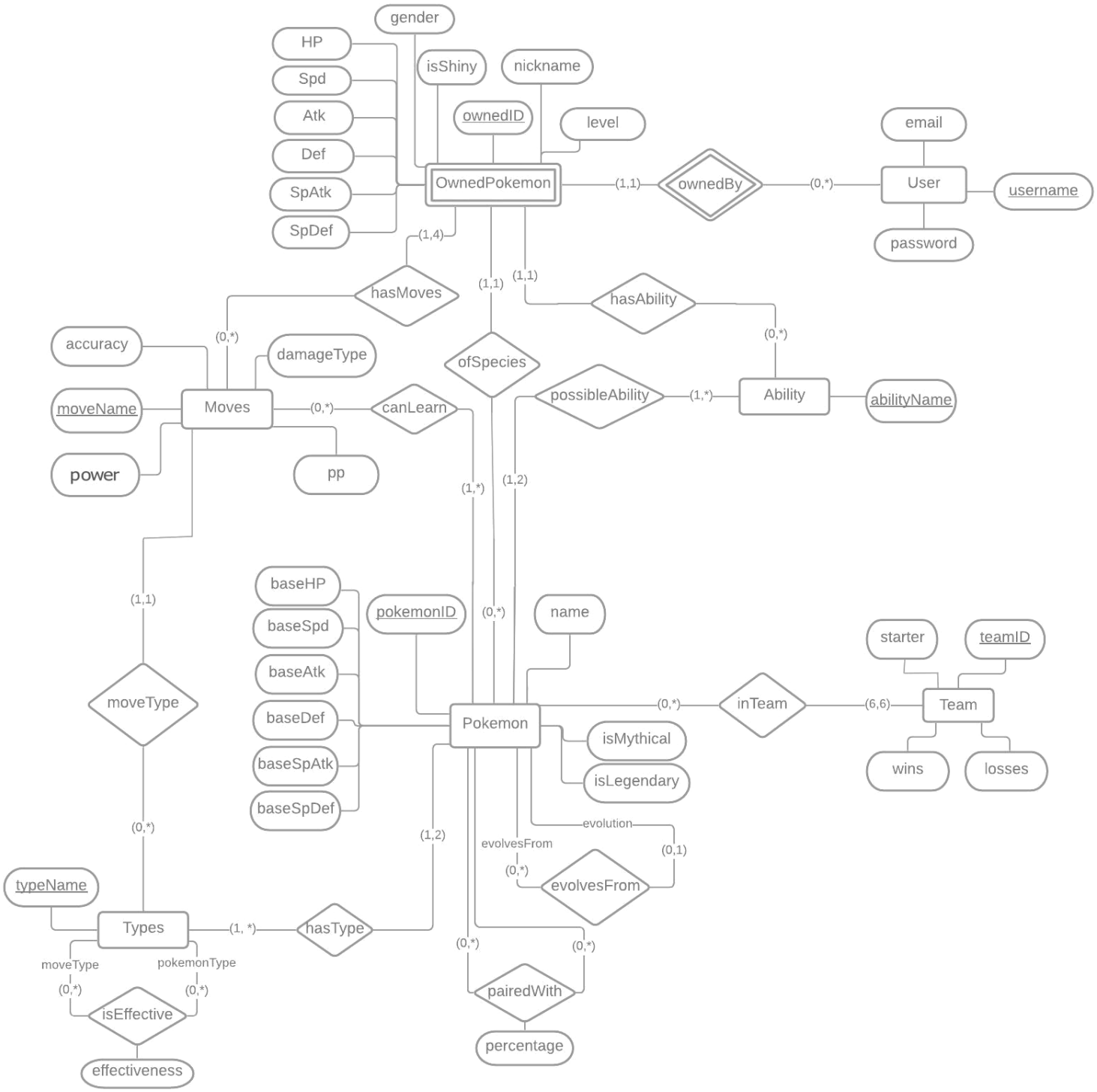
Other:

- The abilityName of abilities is unique. The typeName of types is unique.
- Teams have 6 unique pokemon in them. Teams are considered different if they have a different starter. The teamID is unique.

We choose not to include the following characteristics of pokemon-related data to limit the complexity of our application:

- We do not have maximum values for the stats of user pokemons, since determining the maximum values depends on information we do not keep track of in our application. This limits the amount of information users have to enter when creating an owned pokemon.
- We leave out items to reduce the complexity of the application.
- In pokemon, teams could have multiple pokemon of the same species. We choose to only consider unique pokemon since such teams are more prevalent and easier to model.
- We leave out Hidden Abilities, as they are a relatively new addition to Pokemon and only appear on Pokemon obtained in certain ways outside of normal gameplay.
- We do not include moves with a status type.
- We do not consider moves with multiple possible types.

E/R Diagram


gender
HP
isShiny
nickname
Spd
ownedID
level
Atk
Def
SpAtk
SpDef
OwnedPokemon
(1,1)
ownedBy
(0,*)
email
User
username
password
(1,4)
hasMoves
(1,1)
(1,1)
hasAbility
(0,*)
accuracy
(0,*)
damageType
ofSpecies
moveName
Moves
(0,*)
canLearn
possibleAbility
(1,*)
Ability
abilityName
power
pp
(1,*)
(1,2)
(1,1)
baseHP
pokemonID
(0,*)
name
starter
teamID
moveType
baseSpd
baseAtk
baseDef
Pokemon
(0,*)
inTeam
(6,6)
Team
isMythical
baseSpAtk
isLegendary
wins
losses
baseSpDef
(0,*)
(1,2)
evolution
evolvesFrom
(0,1)
(0,*)
evolvesFrom
typeName
Types
(1, *)
hasType
moveType
pokemonType
(0,*)
(0,*)
(0,*)
(0,*)
isEffective
pairedWith
effectiveness
percentage

The following constraints are not represented in the E/R diagram:

- We do not represent the allowable values for specific attributes.
- The moves a pokemon can have must be learnable by its species. We choose not to represent this constraint since this would require a very complicated relationship set.
- Similarly, we do not represent the constraint that the ability of an owned pokemon must be one of the possible abilities of the corresponding species.

Note that we chose to make OwnedPokemon a weak entity set, since an owned pokemon cannot exist without belonging to some user. However, we decided to make the discriminator unique among all owned pokemon to simplify lookup from the OwnedPokemon table.

Database Tables

Pokemon(id int, name string, baseHp int, baseSpd int, baseAtk int, baseDef int, baseSpAtk int, baseSpDef int, type1 string, type2 string, ability1 string, ability2 string, evolvesFromId int, isLegendary bool, isMythical bool)

- type1, type2 foreign keys to Type
- evolvesFromId is a foreign key to Pokemon

Type(typeName string)

Effectiveness(moveType string, pokemonType string, effectiveness float)

- moveType, pokemonType foreign keys to Type

PokemonPairings(pid1 int, pid2 int, percentage float)

- pid1, pid2 foreign keys to Pokemon with pid1 < pid2
- percentage is the percent Pokemon pid1 is paired with Pokemon pid2 with respect to games involving Pokemon pid1

Team(teamID int, pid1 int, pid2 int, pid3 int, pid4 int, pid5 int, pid6 int, starter int, wins int, losses int)

- pid1 to pid2 foreign keys to Pokemon with pid1 < pid2 < … < pid6

User(username string, email string, password string)

Move(moveName string, moveType string, pp int, power int, damageType string, accuracy int)

- moveType foreign key to Type

CanLearnMove(pid int, moveName string)

- pid foreign key to Pokemon, moveName foreign key to OwnedPokemon

OwnedPokemon(ownedId int, species int, owner string, nickname string, level int, gender string, isShiny bool, hp int, atk int, def int, spAtk int, spDef int, spd int, ability string, move1 string, move2 string, move3 string, move4 string)

- owner foreign key to User, species foreign key to Pokemon, ability foreign key to Ability
- move1, ... , move4 foreign keys to move, move1 NOT NULL

Notes on our design choices:

- For the moves of an owned pokemon, we have four columns in the OwnedPokemon table instead of creating a separate table to keep track of learned moves. This is possible since we know a pokemon can learn at most four moves and simplifies the schema by reducing the number of tables required. We use a similar approach for the type of pokemon, the abilities of pokemon and the IDs of pokemon in each team.
- We chose not to have a separate ability table since each pokemon has at most 2 possible abilities. Hence, we can include all information in the Pokemon table, which we have full control over. We only ever need to look up abilities for one specific species.
- Furthermore, we reduce the complexity of the schema by including an evolvesFromId in the Pokemon table instead of introducing a new table to keep track of evolutions.

Data for the Database

Datasets and Data Generation

For the Type table, we will manually insert all the data, since there are only 18 types of Pokemon. Similarly, for the Effectiveness table we will specify manually which combinations do not have an effectiveness of 1, since this is a small subset of the type combinations. Then, we will write a script to generate all other type combinations and set their effectiveness to 1. The script then inserts all combinations into the Effectiveness table.

For the general information about Pokemon, we use the following dataset:
https://www.kaggle.com/mrdew25/pokemon-database
The following columns of the dataset are used:

- Pokedex Number: This corresponds to the “id” in the Pokemon table
- Pokemon Name: Note that we only look at columns where “Alternate Form Name” is NULL, since we do not support variations of Pokemon
- Legendary Type: From this column, we can extract information for the “isMythical” and “isLegendary” columns of the Pokemon table
- Primary and Secondary Type: These correspond to “type1” and “type2” in our table

- Primary and Secondary Ability: These correspond to “ability1” and “ability2”
- The stats of the pokemon
- Pre-Evolution Pokemon Id: This corresponds to “evolvesFromId”

This gives us most of the information for the Pokemon table. We plan to use a python script, which uses numpy for the data import from the .csv file before extracting the relevant rows and columns and inserting them into the database.

In addition, we would like to show an image for each Pokemon in our table. We scraped these images from Serebii, a wiki for the Pokemon games, as it has images for both the regular and shiny version of each Pokemon. We decided to store these images in an assets folder rather than in the database, using the Pokemon id as the file name. Doing so made the images more accessible, and scalability was not a concern as we do not expect to add any new Pokemon beyond those already in our production dataset.

For the Move and CanLearnMove relations, we use the following dataset:
https://www.kaggle.com/n2cholas/competitive-pokemon-dataset
There are two csv files in this dataset. The “move-data.csv” file contains all information we need for the Move table:

- Name corresponds to “moveName” in the Move table
- Type corresponds to “type”
- Category corresponds to “damageType”
- PP corresponds to “pp”
- Power corresponds to “power”
- Accuracy corresponds to “accuracy”

As above, we will write a python script that extracts these columns from the csv file and inserts all rows into the Move table.
The second “pokemon-data.csv” file can be indexed by Pokemon name and contains a list of possible moves for each Pokemon under the column “Moves”. We will write a python script that for each Pokemon in our database retrieves the list of moves and adds each to the CanLearnMove table.

We will populate the PokemonPairings table based on the most recent usage stats from the pokemon showdown game. Currently, the most recent information can be found here:
https://www.smogon.com/stats/2021-05/moveset/

In particular, we will look at the “ou” tier of Pokemon, which includes almost all Pokemon and has the most data. For each Pokemon, the files specify the number of games in which the pokemon was used as well as the percent of times that Pokemon was paired with other

pokemon. We will extract this information from the text files using a python script and then add the entries to the table.

To obtain information on team usage, we scrape data from https://replay.pokemonshowdown.com/ using the Beautiful Soup Python library. The “recent replays” section shows replays from the last 5 minutes, and a log of each replay is available at https://replay.pokemonshowdown.com/<battle type>-<id>.log. By checking for new replays every couple of minutes, we can obtain a large amount of data on popular teams in all game modes. Each replay shows the Pokemon in each team, the starter Pokemon as well as who won the battle. For PokeDB, we only consider single battles, where each player sends out one Pokemon at a time. We ignore double or triple battles where multiple Pokemon are sent out at once, as adding support for them would increase the complexity of our application logic and database schema. We also ignore any battles with random or custom pokemon, as these do not provide any insight into good team compositions. After scraping this data, we insert it into the Team table and initialize wins and losses. If we already have an identical team in the table, we instead update the wins and losses of that entry.

We will generate the OwnedPokemon table randomly, using the information from the tables we filled above. We create a script that randomly picks a pokemon from the Pokemon table and randomly picks one of its abilities. Then, the stats of the pokemon and the number of learned moves are randomized. A single random move that the species can learn is chosen from the CanLearnMove table. We use this procedure to generate 50 owned pokemon for user1 and 10 owned pokemon for user2.

For the user relation, we generate our own data. Since there are no interactions between users, we can simply use a small set of users we create manually.

Sample Data

User:

	username [PK] character varying(30)	email character varying(50)	password character varying(20)
1	user1	user1@gmail.com	pass1
2	user2	user1@gmail.com	pass2
3	user3	user1@gmail.com	pass3
*			

Type:

	typename [PK] character varying(16)
1	Bug
2	Dark
3	Dragon
4	Electric
5	Fairy
6	Fighting
7	Fire
8	Flying
9	Ghost
10	Grass
11	Ground
12	Ice
13	Normal
14	Poison
15	Psychic
16	Rock
17	Steel
18	Water
*	

Pokemon:

	id [PK]	name character vary	basehp integer	basespd integer	baseatk integer	basedef integer	basespatk integer	basespdef integer	type1 character va	type2 character v	ability1 character varyir	ability2 characte	evolvesfromid integer	islegendary boolean	ismythical boolean
1	1	Bulbasaur	45	45	49	49	65	65	Grass	Poison	Overgrow			FALSE	FALSE
2	2	Ivysaur	60	60	62	63	80	80	Grass	Poison	Overgrow		1	FALSE	FALSE
3	3	Venusaur	80	80	100	123	122	120	Grass	Poison	Overgrow		2	FALSE	FALSE
4	4	Charmander	39	65	52	43	60	50	Fire		Blaze			FALSE	FALSE
5	5	Charmeleon	58	80	64	58	80	65	Fire		Blaze		4	FALSE	FALSE
6	6	Charizard	78	100	104	78	159	115	Fire	Flying	Blaze		5	FALSE	FALSE
7	7	Squirtle	44	43	48	65	50	64	Water		Torrent			FALSE	FALSE
8	8	Wartortle	59	58	63	80	65	80	Water		Torrent		7	FALSE	FALSE
9	9	Blastoise	79	78	103	120	135	115	Water		Torrent		8	FALSE	FALSE

Move:

	movename [PK] character varying(30)	movetype character varying(16)	pp integer	power integer	damagetype character varying(10)	accuracy integer
1	Bubble	Water	30	40	special	100
2	Ember	Fire	25	40	special	100
3	Fire Blast	Fire	5	110	special	85
4	Hydro Pump	Water	5	110	special	80
5	Pound	Normal	35	40	physical	100
6	Razor Leaf	Grass	25	55	physical	95
7	Solar Beam	Grass	10	120	special	100
*						

CanLearnMove:

	pid [PK] integer	movename [PK] character varying(30)
1	1	Razor Leaf
2	2	Razor Leaf
3	2	Solar Beam
4	3	Razor Leaf
5	3	Solar Beam
6	4	Ember
7	6	Ember
8	6	Fire Blast
9	6	Solar Beam
10	7	Bubble
11	8	Bubble
12	9	Bubble
13	9	Hydro Pump
*		

Effectiveness:

	movetype [PK] character varying(16)	pokemontype [PK] character varying(16)	effectiveness numeric(2,1)
1	Fighting	Normal	2.0
2	Normal	Ghost	0.0
3	Normal	Normal	1.0
4	Rock	Fighting	0.5
*			

PokemonPairings:

	pid1 [PK] integer	pid2 [PK] integer	percentage real
1	3	6	0.1
2	3	9	0.2
3	6	9	0.05
*			

Team:

	teamid [PK] serial	pid1 integer	pid2 integer	pid3 integer	pid4 integer	pid5 integer	pid6 integer	starter integer	wins integer	losses integer
1	1	1	2	3	4	5	6	1	1	1
2	2	2	3	4	5	6	9	2	6	2
3	3	3	5	6	7	8	9	3	10	5
*										

OwnedPokemon:

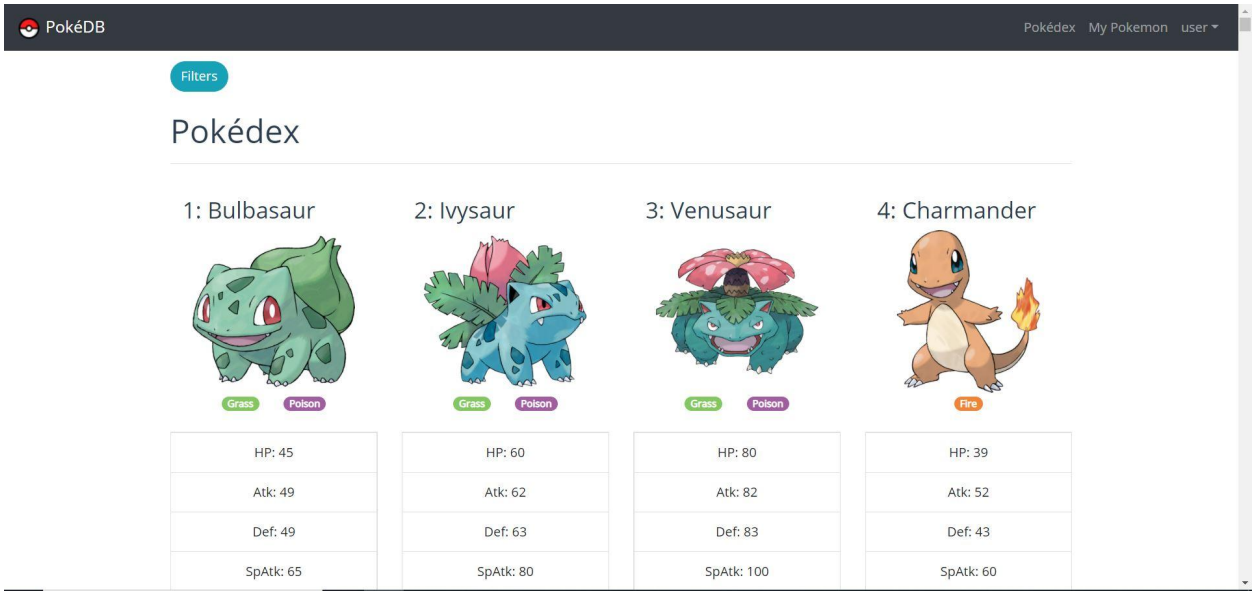
	ownedid [PK] seri	species integer	owner characte	nickname character	level integer	gender character	isshiny boolean	hp integer	atk integer	def integer	spatk integer	spdef integer	spd integer	ability character va	move1 character vary	move2 characte	move3 characte	move4 characte
1	1	1	user1	Bulby	2	male	FALSE	10	10	10	10	10	10	Overgrow	Razor Leaf			
2	2	4	user2	Char	2	female	FALSE	11	11	11	11	11	11	Blaze	Ember			
3	3	7	user3	Turtle	2	male	FALSE	12	12	12	12	12	12	Torrent	Bubble			
*																		

Key Feature Descriptions

Feature 1: Pokedex

This feature would allow users to search for Pokemon in our database. They would be able to search for Pokemon with a given id, with a certain string in their name, with stats in a given range, or with a specific ability. They can also add a filter to only show Pokemon with specific types, Legendary Pokemon, and Mythical Pokemon. The user can also query for Pokemon that know moves with a certain name, type, PP, power, damage type, or accuracy. Resulting Pokemon are returned in order of Pokemon id.

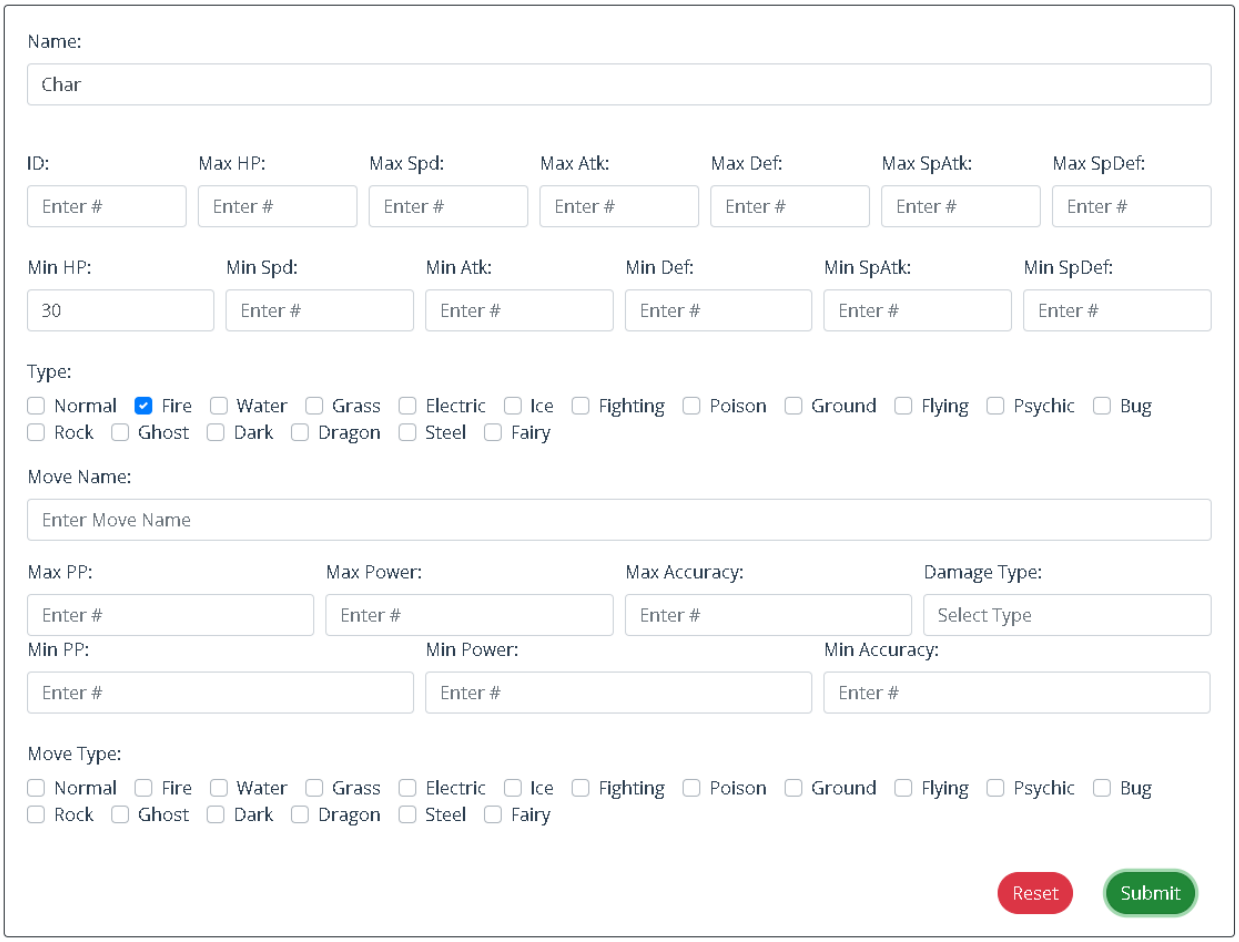
The search result would contain some basic details about each Pokemon selected, as well as a link that leads to a more detailed page about the Pokemon. If the user queried by move, the move(s) known by the Pokemon that match the query will also be shown.



Here is the query structure if the user does not query by moves.

SELECT id, name, baseHp, baseSpd, baseAtk, baseDef, baseSpAtk, baseSpDef, type1, type2
FROM Pokemon
WHERE <specifications>
ORDER BY id;

Here, specifications are the constraints on the columns that the user specified. Here is an example query. Note that when the user supplies strings, we will look for all strings containing the user-entered string, ignoring case.




```
SELECT id, name, baseHp, baseSpd, baseAtk, baseDef, baseSpAtk, baseSpDef, type1, type2
    FROM Pokemon
    WHERE name ILIKE '%char%' AND baseHp >= 30
    AND (type1 = 'Fire' OR type2 = 'Fire')
    ORDER BY id;
```

Out of the Pokemon in the production data set, we expect this to output all the information about Charmander, Charmeleon, Charizard, and Chimchar, since these are the only Fire-type Pokemon with “char” in their name and baseHp >= 30.

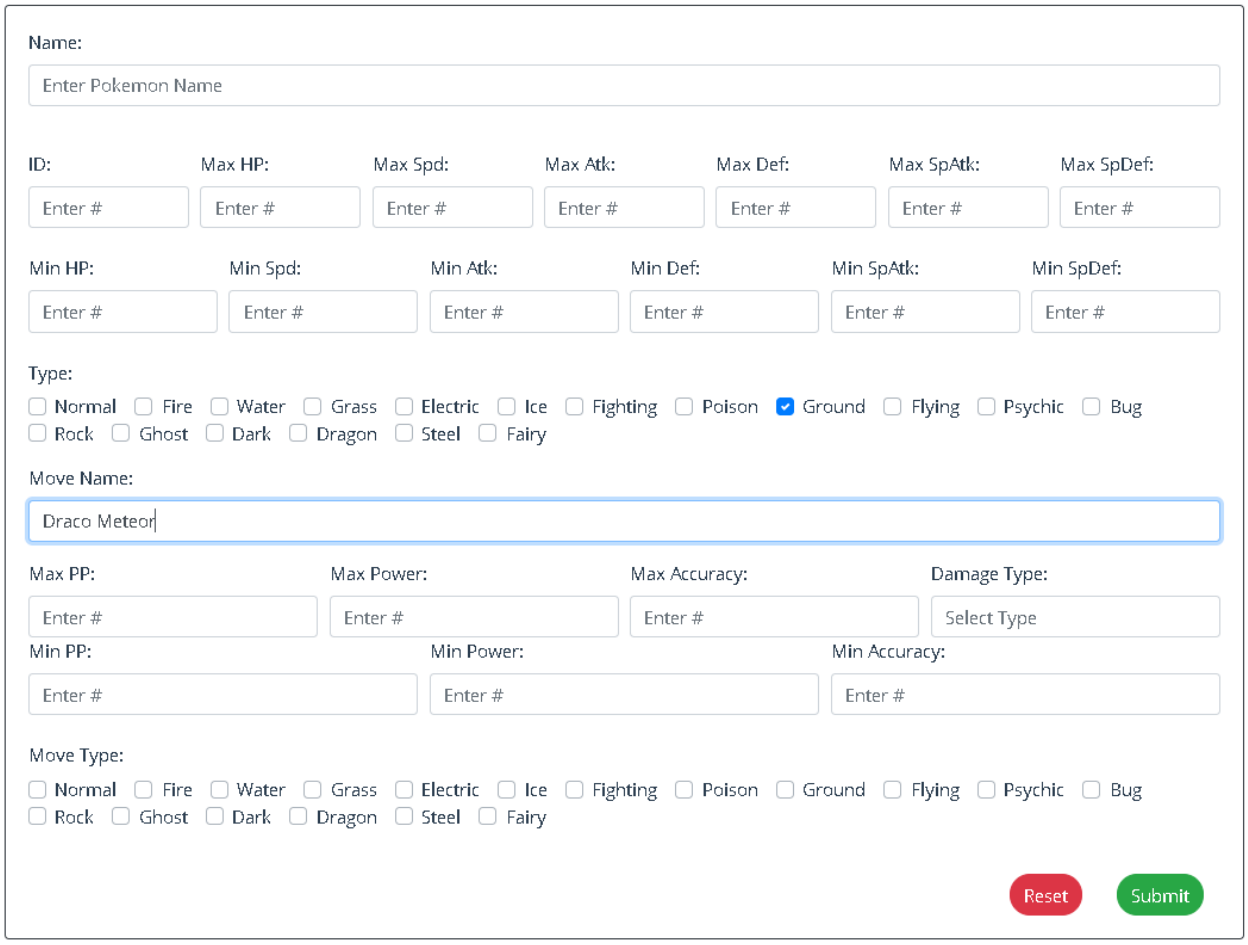
Pokédex

4: Charmander	5: Charmeleon	6: Charizard	390: Chimchar
			
Fire	Fire	Fire Flying	Fire
HP: 39	HP: 58	HP: 78	HP: 44
Atk: 52	Atk: 64	Atk: 84	Atk: 58
Def: 43	Def: 58	Def: 78	Def: 44
SpAtk: 60	SpAtk: 80	SpAtk: 109	SpAtk: 58
SpDef: 50	SpDef: 65	SpDef: 85	SpDef: 44
Spd: 65	Spd: 80	Spd: 100	Spd: 61

Here is the query for Pokemon that have specific features AND know specific moves. Notice that we group the entries so the query outputs one tuple for each Pokemon. The matching move names are aggregated into a list.

```
SELECT id, name, baseHp, baseSpd, baseAtk, baseDef, baseSpAtk, baseSpDef, type1, type2,
STRING_AGG(Move.moveName, ', ' ORDER BY Move.moveName) FROM (
        (
                Pokemon
                JOIN
                CanLearnMove
                ON id=pid
        )
        JOIN
        Move
        ON Move.moveName = CanLearnMove.moveName
)
WHERE
<pokemon features> AND <move features>
GROUP BY id, name, baseHp, baseSpd, baseAtk, baseDef, baseSpAtk, baseSpDef, type1, type2
ORDER BY id;
```

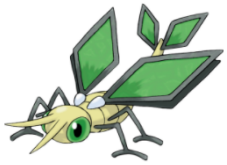
Here is an example query using this structure. The user is interested in Pokemon with the Ground type that can learn the move Draco Meteor.



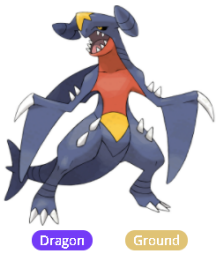

```
SELECT id, name, baseHp, baseSpd, baseAtk, baseDef, baseSpAtk, baseSpDef, type1, type2,
STRING_AGG(Move.moveName, ', ' ORDER BY Move.moveName) FROM (
        (
                Pokemon
                JOIN
                CanLearnMove
                ON id=pid
        )
        JOIN
        Move
        ON Move.moveName = CanLearnMove.moveName
)
```

WHERE
(type1 = 'Ground' OR type2 = 'Ground') AND Move.moveName ILIKE '%Draco Meteor%'
GROUP BY id, name, baseHp, baseSpd, baseAtk, baseDef, baseSpAtk, baseSpDef, type1, type2
ORDER BY id;

There are six matching Pokemon in the production dataset: Vibrava, Flygon, Gible, Gabite, Garchomp, and Zygarde all have the Ground type and can learn Draco Meteor. For all Pokemon in the result set, the query should return basic information, as well as a comma-separated list of moves that match the given criteria. In this case, all Pokemon in the result set will only be listed alongside the move Draco Meteor.

329: Vibrava	330: Flygon	443: Gible	444: Gabite
			
Ground Dragon	Ground Dragon	Dragon Ground	Dragon Ground
HP: 50	HP: 80	HP: 58	HP: 68
Atk: 70	Atk: 100	Atk: 70	Atk: 90
Def: 50	Def: 80	Def: 45	Def: 65
SpAtk: 50	SpAtk: 80	SpAtk: 40	SpAtk: 50
SpDef: 50	SpDef: 80	SpDef: 45	SpDef: 55
Spd: 70	Spd: 100	Spd: 42	Spd: 82
Selected Moves Known: Draco Meteor	Selected Moves Known: Draco Meteor	Selected Moves Known: Draco Meteor	Selected Moves Known: Draco Meteor

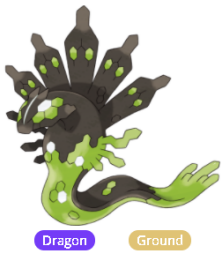
445: Garchomp



Dragon Ground

HP: 108
Atk: 130
Def: 95
SpAtk: 80
SpDef: 85
Spd: 102
Selected Moves Known: Draco Meteor

718: Zygarde



Dragon Ground

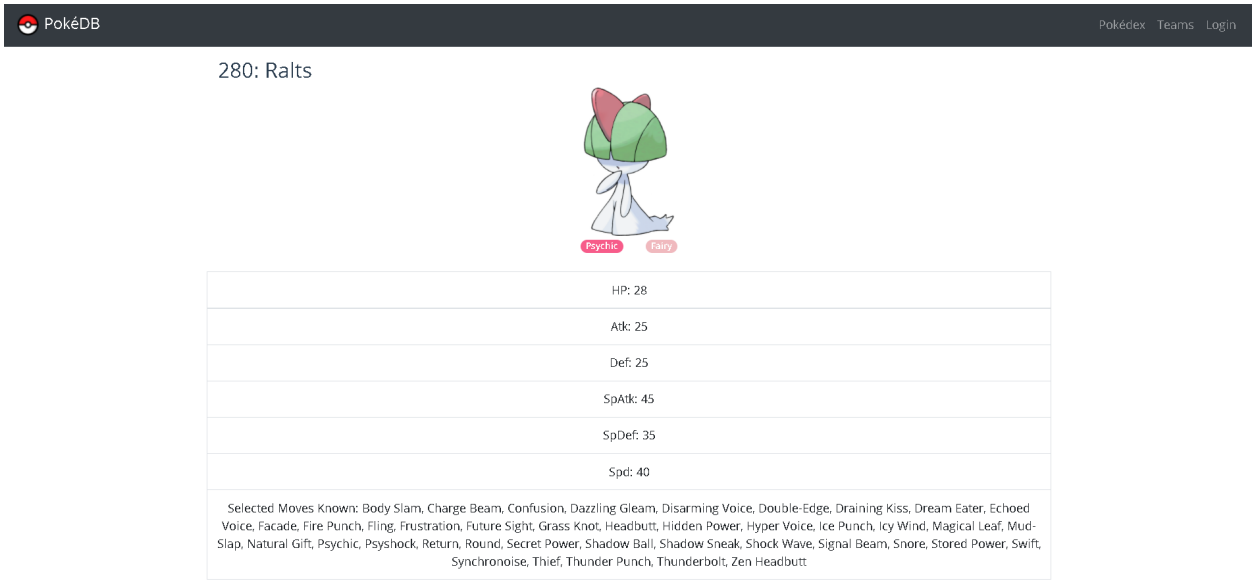
HP: 108
Atk: 100
Def: 121
SpAtk: 81
SpDef: 95
Spd: 95
Selected Moves Known: Draco Meteor

Feature 2: Pokemon Evolution

When the user clicks on a link in a search result, they should be shown a page with more details about the Pokemon. One of these details should be what Pokemen the current Pokemon can evolve into, presented as a list. Since each Pokemon only stores what Pokemon they evolve from, this information can be retrieved using a recursive query.

```
WITH RECURSIVE
Evolution(evolvesFrom, evolvesInto) AS (
        (
                SELECT evolvesFromId, id FROM Pokemon
        )
        UNION
        (
                SELECT e1.evolvesFrom, id
                FROM Evolution e1, Pokemon p
                WHERE e1.evolvesInto = p.evolvesFromId
        )
)
SELECT * FROM Pokemon
WHERE id IN (
        SELECT evolvesInto
        FROM Evolution
        WHERE evolvesFrom = <pokemon id>
);
```

Here is an example query. The user is interested in all Pokemon that can evolve from the Pokemon Ralts, which has id 280.




```
WITH RECURSIVE
Evolution(evolvesFrom, evolvesInto) AS (
        (
                SELECT evolvesFromId, id FROM Pokemon
        )
        UNION
        (
                SELECT e1.evolvesFrom, id
                FROM Evolution e1, Pokemon p
                WHERE e1.evolvesInto = p.evolvesFromId
        )
)
SELECT * FROM Pokemon
WHERE id IN (
        SELECT evolvesInto
        FROM Evolution
        WHERE evolvesFrom = 280
);
```

Ralts evolves into Kirlia, which can then evolve into Gardevoir or Gallade. Thus, this query should return information about Kirlia, Gardevoir, and Gallade.

Evolutions

Click on a Pokémon's image to learn more.

281: Kirlia

Psychic Fairy

HP: 38
Atk: 35
Def: 35
SpAtk: 65
SpDef: 55
Spd: 50
Evolves from: Ralts

282: Gardevoir

Psychic Fairy

HP: 68
Atk: 65
Def: 65
SpAtk: 125
SpDef: 115
Spd: 80
Evolves from: Kirlia

475: Gallade

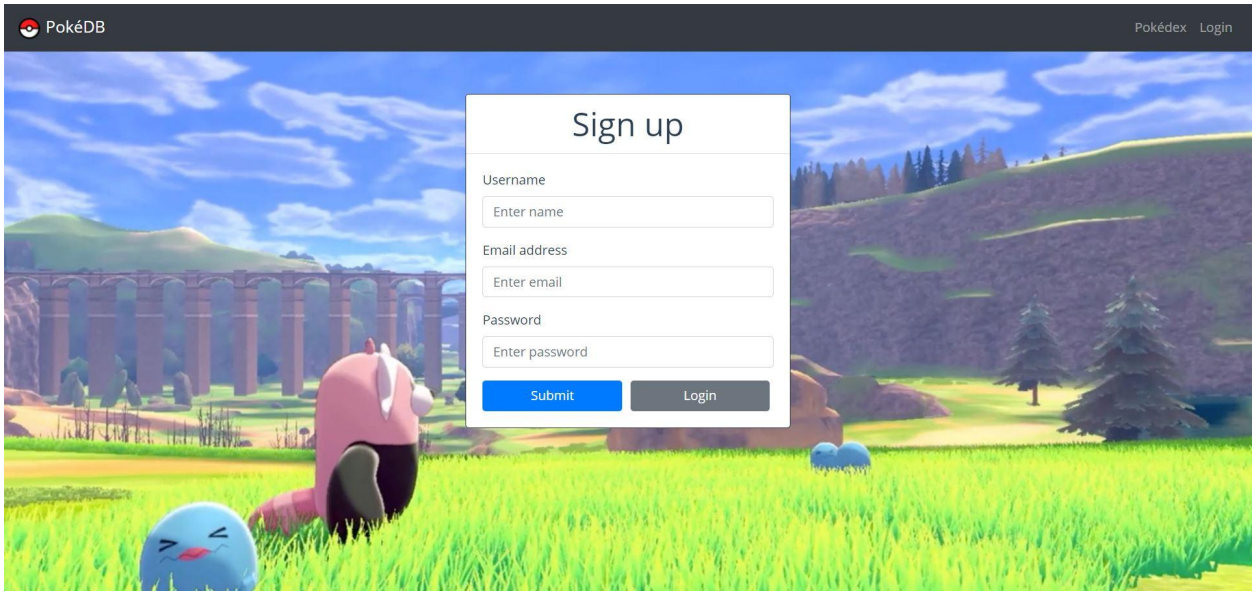
Psychic Fighting

HP: 68
Atk: 125
Def: 65
SpAtk: 65
SpDef: 115
Spd: 80
Evolves from: Kirlia

Feature 3: User signup / login with authentication and encrypted passwords

Signup / adding new users

New users can click a “Signup” button at the top of the website. They will be taken to a form page where there are fields of a new username, their email address, and password for them to fill in. Upon submission, their information will be stored onto the User database. Thus, this user’s account has been added and they will be logged in**.** If the username already exists, an error message will pop-up and the user will need to choose a different username.

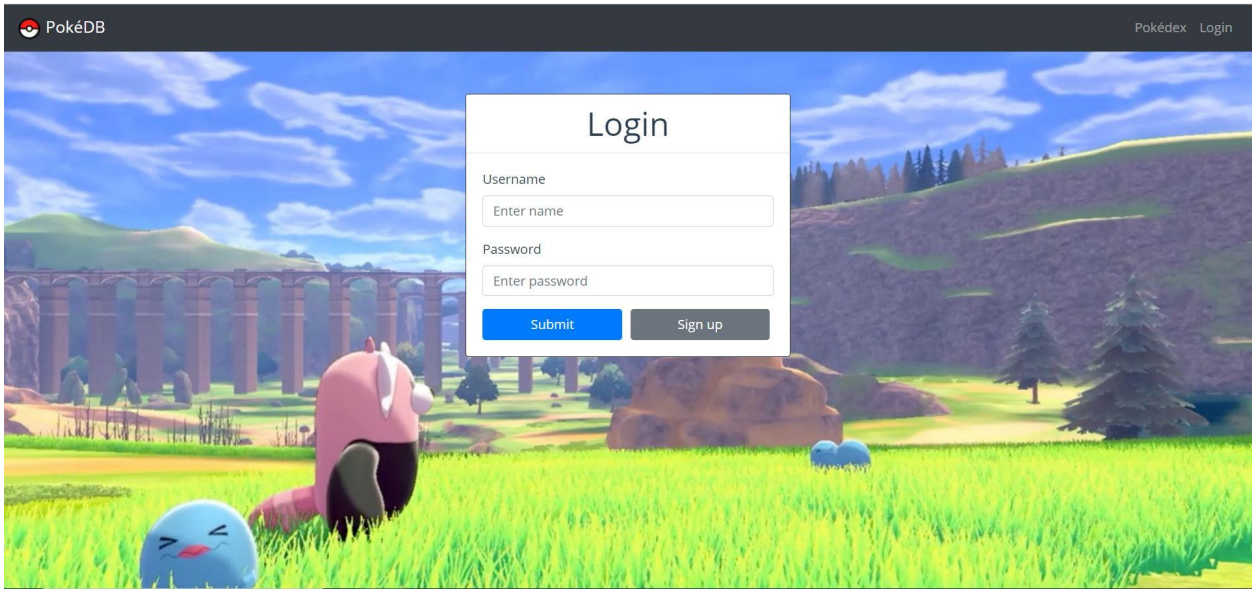


To insert the user’s information:
INSERT INTO User(username, email, password)
VALUES(‘username’, ‘email@address’, ‘password’);

We expect this query to create a new tuple in the User table with values (‘username’, ‘email@address’, ‘password’). Upon successful signup, the user is redirected to the Pokedex page.

Login with authentication / verifying old password

Existing users can click a “Login” button at the top of the website. They will be taken to a form page where there are fields of their username and password for them to fill in.
Upon submission, the password associated with their username will be retrieved from the database and the password they entered will be matched with the retrieved password. If the passwords match, then the user is logged in.

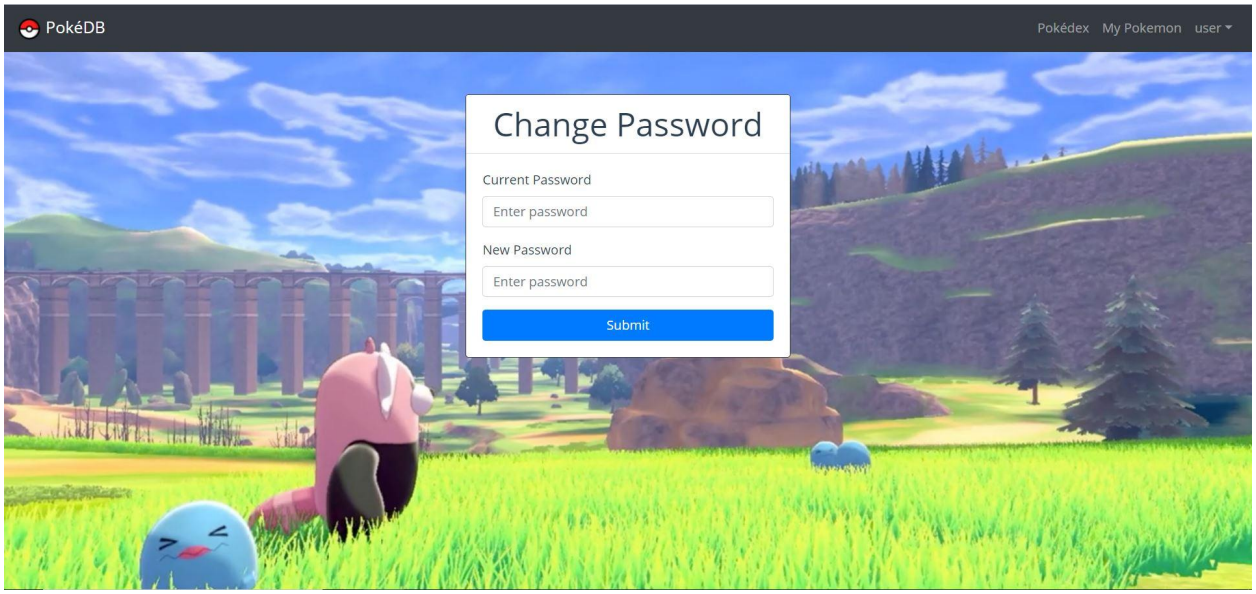


Moreover, existing users can change their password by clicking on an option in a dropdown menu. They will be taken to a form page where there are fields for the old password and the new password. Upon submission, the password associated with their username will be retrieved from the database and the old password they entered will be matched with the retrieved password.

Retrieving the password associated to the entered username:
SELECT password
FROM User
WHERE username = 'username';

With the above INSERT, the output should be a row containing the password 'password'. Upon successful login, the user is redirected to the Pokedex page.

Changing password
As mentioned above, existing users can change their password. After verifying the old password entered matches, the password stored on the database will be updated to the new password.



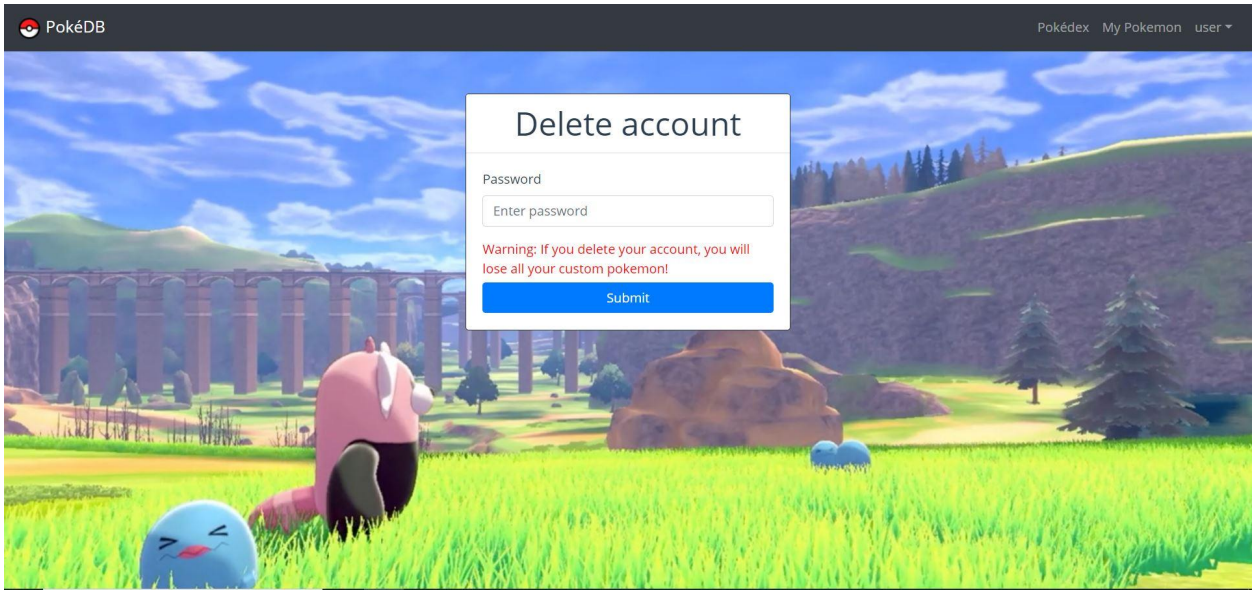
Update password associated to a username:

UPDATE User
SET password = ‘new_password’
WHERE username = ‘username’;

If we ran the above SELECT on the database now, the output should be a row containing the password ‘new_password’.

Deleting user

Users can delete their account by clicking on an option on a dropdown menu. Upon clicking this option, they can enter their password to delete their account from the database.



Delete user account associated to a username:

DELETE FROM User

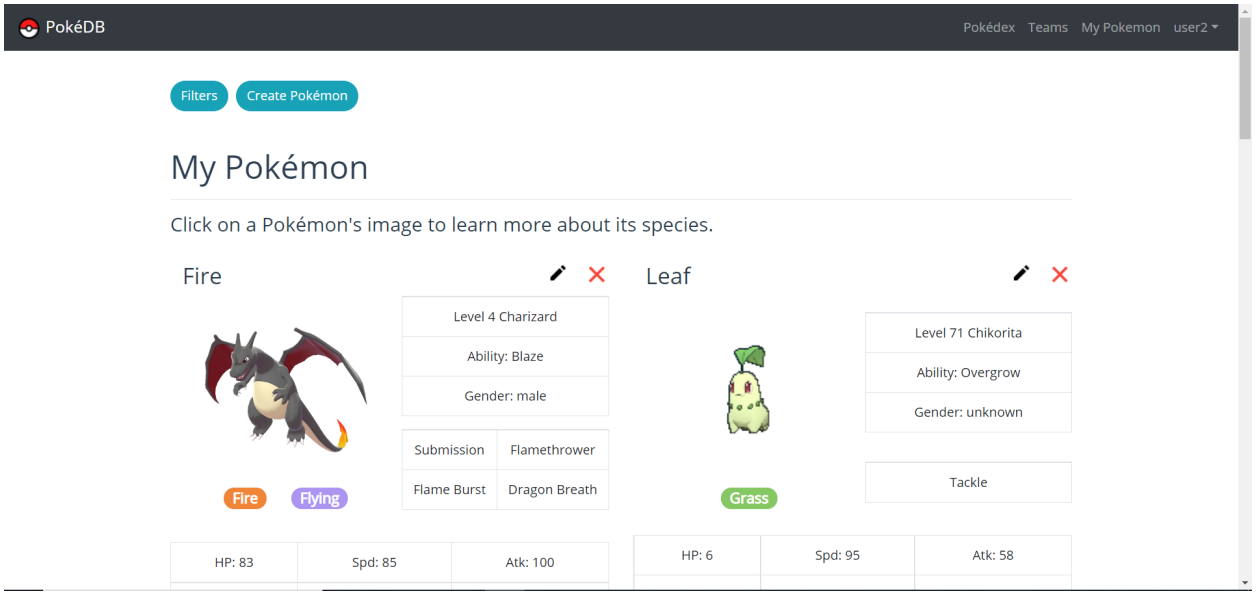
WHERE username = ‘username’;

If we ran the above SELECT on the database now, the output should be 0 rows.

Feature 4: User-owned Pokemon

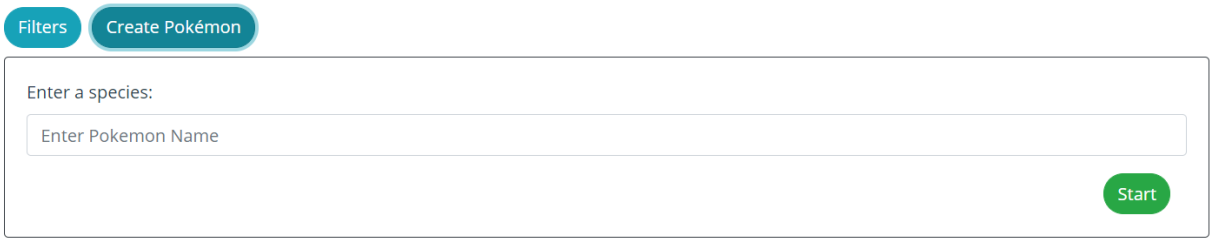
This feature allows users to create custom user owned-pokemon, lets users view and modify their owned pokemon and enables deletion of pokemon of choice.

There is one page of the application from which the users can complete these operations. Creation of pokemon occurs through a button at the top of the page. Underneath, the page displays the pokemon already created by the user along with their information, retrieved using a simple select statement with the current username specified. Each owned pokemon has edit and delete icons which let the user access the other operations of this feature.

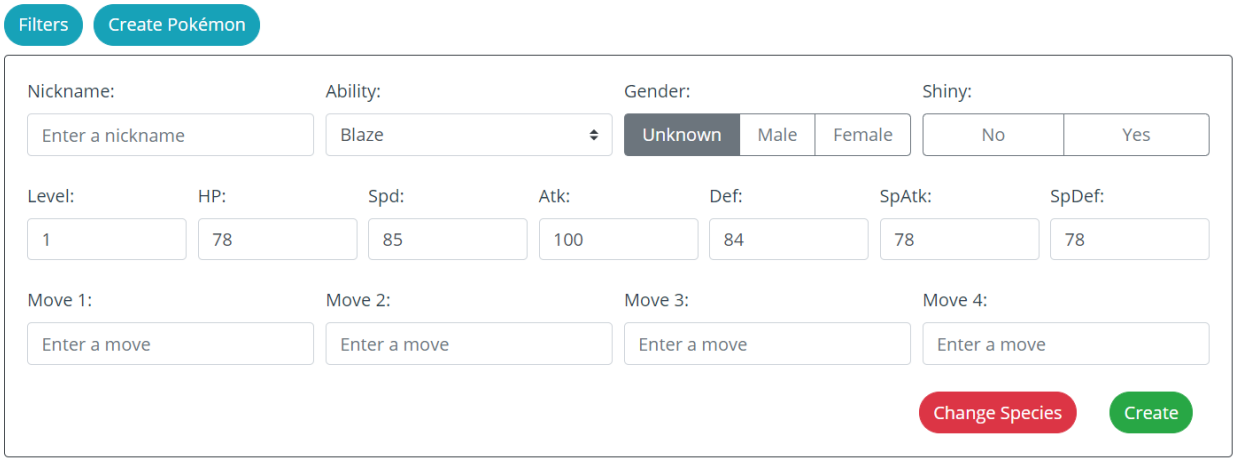


Creation of owned pokemon:

The user clicks on the “create a new owned pokemon” button. The application then asks the user to choose a species of Pokemon.



The application uses this to load some information about that species and displays a form which lets the user input the following information, followed by clicking a “Create” button: name of the ability (required), up to four move names (first move is required), nickname, level, gender, isShiny and pokemon stats.



After the user submits the information, the application attempts to use the insertion query below to create the owned pokemon. The schema uses foreign keys to ensure that the specified moves and species are valid. Furthermore, CHECKs ensure that the user specifies sensible values for the gender and level, and that the moves are unique. One trigger ensures that the ability is consistent with the Pokemon table and one trigger inserts base values for the pokemon stats if the user left these empty.

If the query succeeds, then the page reloads. Otherwise, the front-end displays the error and the user is able to edit and resubmit the form or cancel the operation.

Here is the trigger that supplies default stats when they are not specified by the user:

```
CREATE OR REPLACE FUNCTION fill_stats() RETURNS TRIGGER AS $$
DECLARE
        bHp INTEGER;
        bAtk INTEGER;
        bDef INTEGER;
        bSpAtk INTEGER;
        bSpDef INTEGER;
        bSpd INTEGER;
BEGIN
        SELECT baseHp, baseAtk, baseDef, baseSpAtk, baseSpDef, baseSpd
        INTO bHp, bAtk, bDef, bSpAtk, bSpDef, bSpd
        FROM Pokemon
        WHERE id = NEW.species;

        IF NEW.hp IS NULL THEN
```

```
            NEW.hp = bHp;
        END IF;
        IF NEW.atk IS NULL THEN
            NEW.atk = bAtk;
        END IF;
        IF NEW.def IS NULL THEN
            NEW.def = bDef;
        END IF;
        IF NEW.spAtk IS NULL THEN
            NEW.spAtk = bSpAtk;
        END IF;
        IF NEW.spDef IS NULL THEN
            NEW.spDef = bSpDef;
        END IF;
        IF NEW.spd IS NULL THEN
            NEW.spd = bSpd;
        END IF;

        RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER DefaultStats
BEFORE INSERT ON OwnedPokemon
FOR EACH ROW
EXECUTE PROCEDURE fill_stats();
```

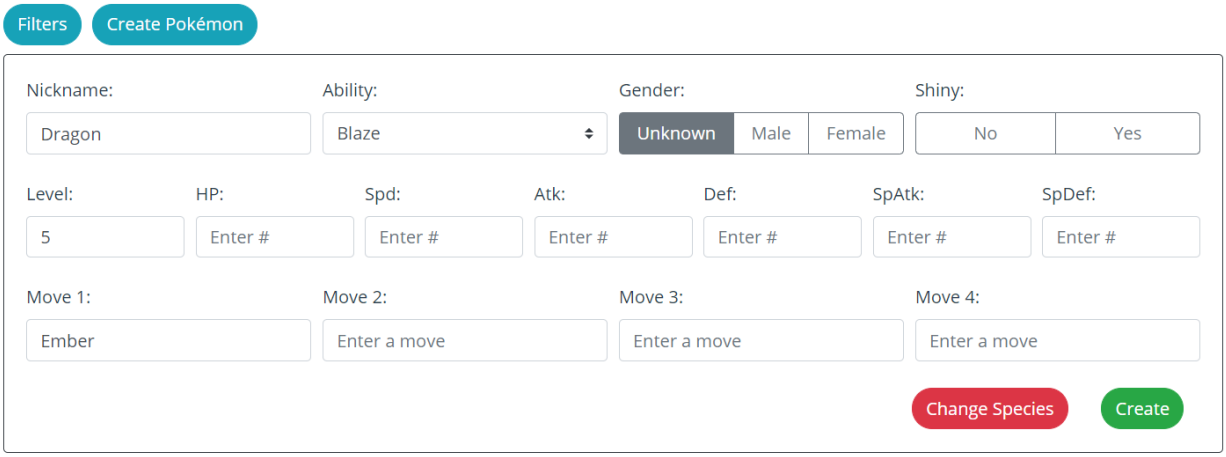
The query for creating owned pokemon is flexible and depends on the columns that were specified by the user:

```
INSERT INTO OwnedPokemon (species, owner, other specified columns)
        SELECT id, <username>, other specified values FROM Pokemon
        WHERE name = <specified species name>;
```

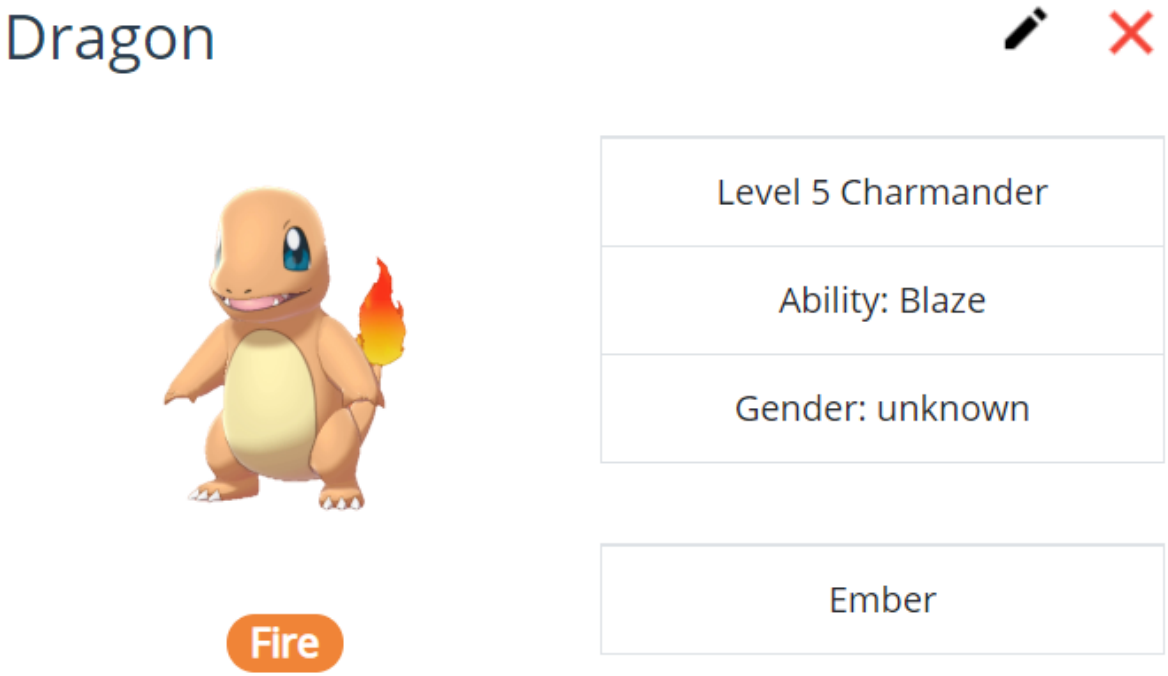
A concrete example is the following query:

```
INSERT INTO OwnedPokemon (species, owner, nickname, level, ability, move1)
        SELECT id, 'user1', 'Dragon', 5, 'Blaze', 'Ember' FROM Pokemon
```

WHERE name = 'Charmander';

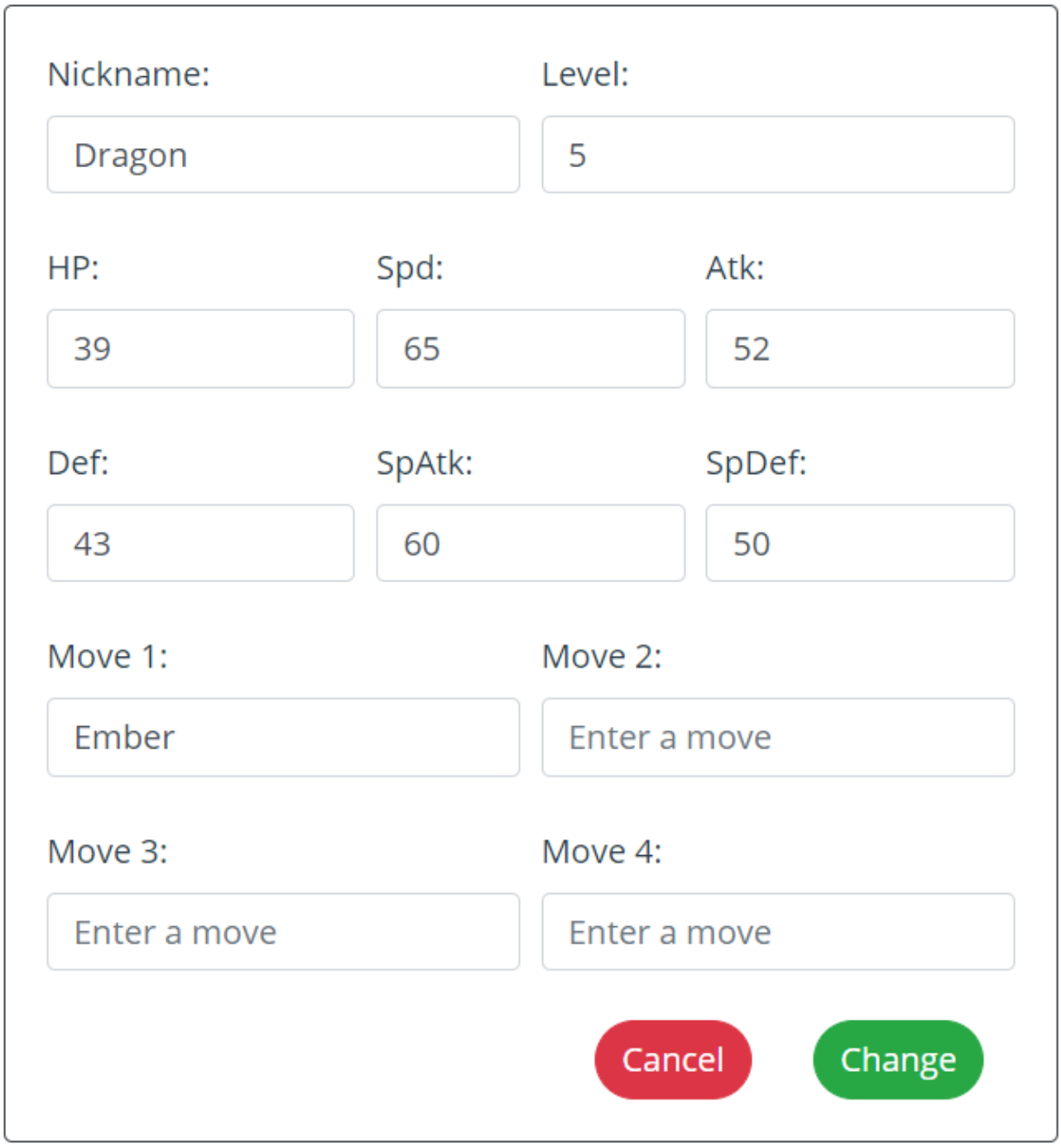


Note that the user did not specify any of the stats, so the trigger will retrieve these from the Pokemon table. We expect the following tuple to be inserted into the OwnedPokemon table: (56, 4, user1, Dragon, 5, unknown, f, 39, 52, 43, 60, 50, 65, Blaze, Ember, NULL, NULL, NULL) Here, the first 56 is the unique ID that the database generates internally. The gender and isShiny flags are set to their default values, and moves 2 to 4 are NULL.



Modification of owned pokemon:

Once the user clicks the edit icon on an owned pokemon, a new form appears, pre-populated with the current values. The user can edit the moves, the nickname and the stats and then submit. As before, if all constraints are satisfied then the page reloads. Otherwise, the user can try again or cancel the operation.

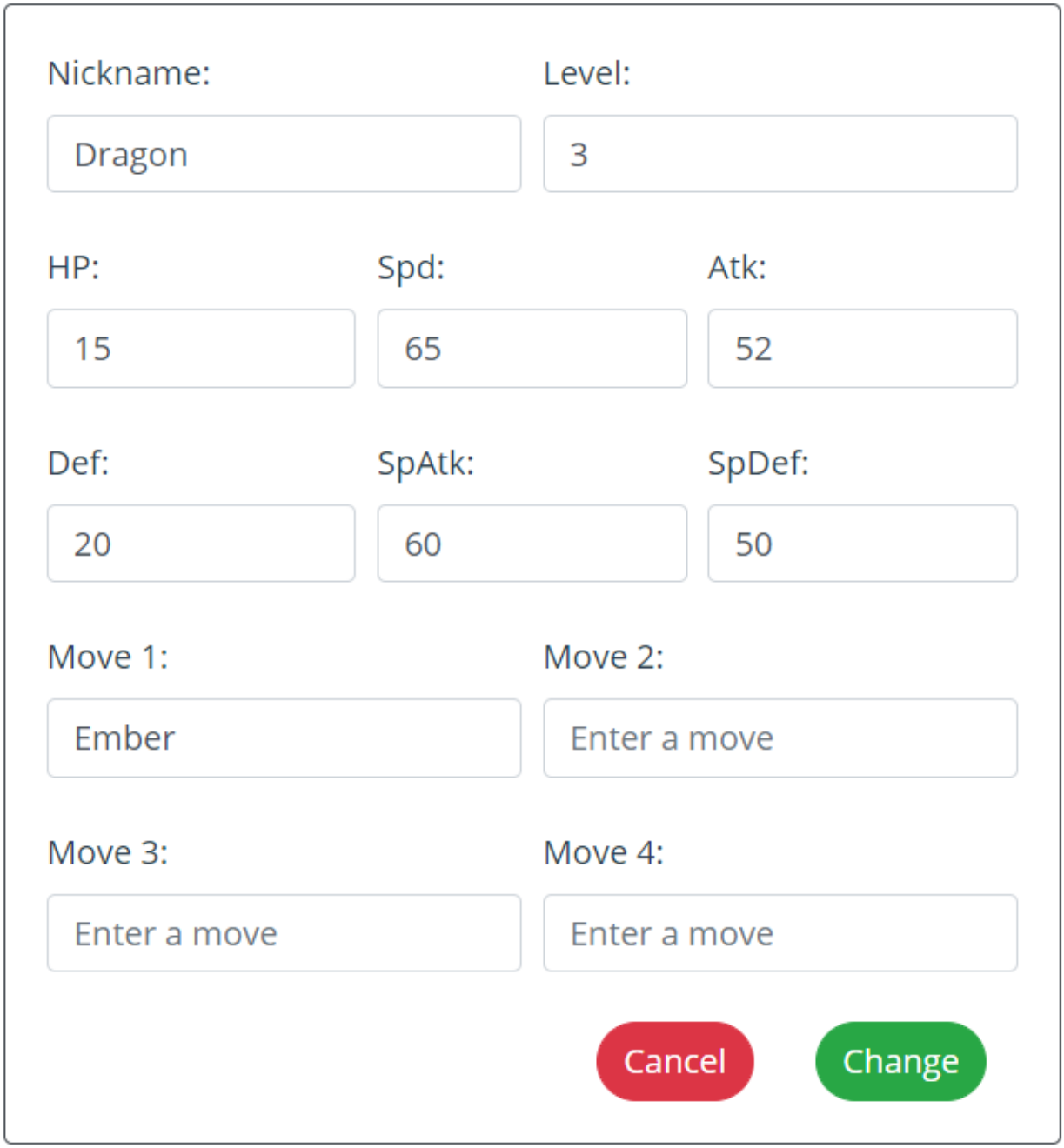


Here is the query for modifying owned pokemon. The query uses the ownedID of the pokemon, which the front-end keeps track of in a hidden element.

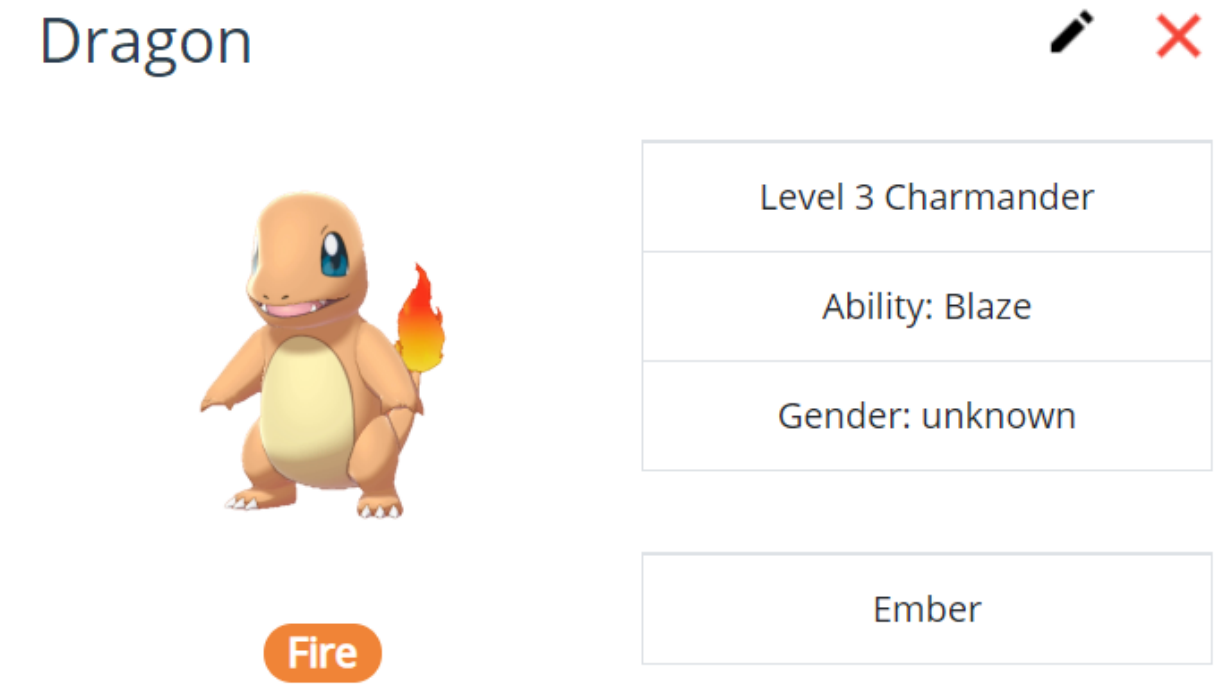
UPDATE OwnedPokemon SET
species = (SELECT id FROM Pokemon WHERE name=specifiedName),
specifiedColumn1 = specifiedValue1, specifiedColumn2 = specifiedValue2, ...
WHERE ownedID = <given ID>;

A concrete example is the following query. The user changes the level, hp and def of the pokemon with ownedID 56. The species ID is looked up in the subquery.

```
UPDATE OwnedPokemon SET species = (SELECT id FROM Pokemon WHERE
name='Charmander'),
        level = 3, hp = 15, def = 20
WHERE ownedID = 56;
```



Before execution, the sample dataset includes following tuple in the OwnedPokemon table:
(56, 4, user1, Dragon, 5, unknown, f, 39, 52, 43, 60, 50, 65, Blaze, Ember, NULL, NULL, NULL)
After execution, we expect this tuple to look like the following:
(56, 4, user1, Dragon, 3, unknown, f, 15, 52, 20, 60, 50, 65, Blaze, Ember, NULL, NULL, NULL)



Deletion of owned pokemon:
Once the user clicks the delete icon on a pokemon, the backend of the application will send the deletion query to the database. After deletion, the owned pokemon page reloads without the deleted pokemon.

Here is the query for deleting owned pokemon. The query uses the ownedID of the pokemon, which the front-end keeps track of in a hidden element:

DELETE FROM OwnedPokemon WHERE ownedID = <given ID>;

A concrete example is the following query:

DELETE FROM OwnedPokemon WHERE ownedID = 56;

After execution, the following tuple in the sample data should no longer be present in the OwnedPokemon dataset:
(56, 4, user1, Dragon, 3, unknown, f, 15, 52, 20, 60, 50, 65, Blaze, Ember, NULL, NULL, NULL)

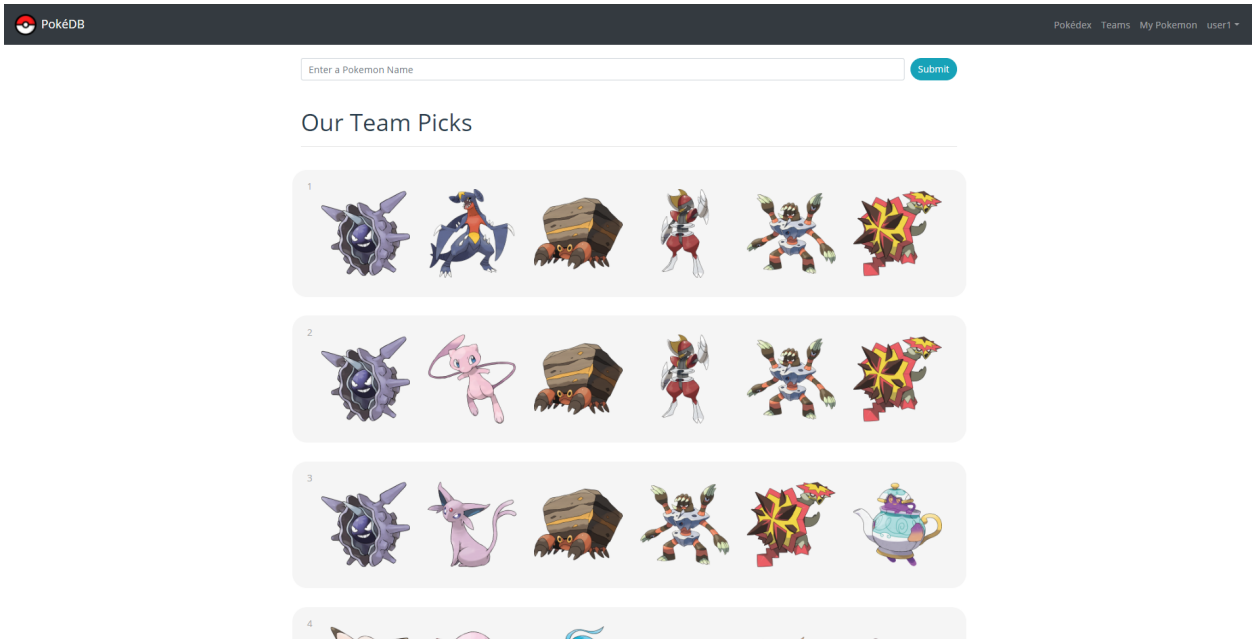
Feature 5: Recommending Program-Generated Pokemon Teams

When users click on the "Teams" button, the application will generate up to 5 program-generated teams of 6 Pokemon by selecting each Pokemon greedily based on highest sum of percentages that each pair of Pokemon are matched up, while restricting that Pokemon must have different types i.e.

- Suppose we have selected a table T of teams with n-1 Pokemon
- To select the nth team, we join T with a Pokemon table to get the nth Pokemon
- We also left-join the PokemonPairings table n-1 times to get pairing synergies between Pokemon (i, n) i.e. the percentage that the ith Pokemon and nth Pokemon are paired together (in the case that the two Pokemon do not have a percentage pairing in the PokemonPairings table, left-join allows the percentage to be NULL which we will treat as 0)
- These Pokemon should have different types for a stronger team
- The score of a team becomes the sum of these pairing percentages
- If n < 6, then we choose the top 1000 teams by score (to prevent memory overflow). If n = 6, then we have complete teams, in which case we choose the top 5 teams by score.

The user can also filter on teams containing a specific Pokemon i.e. if the user wishes to find teams containing Charizard, they can search "Charizard" in the search bar and the program will only generate teams where the first Pokemon is Charizard.

Snapshot of no filter search:



```
SELECT
   p1id,
   p1name,
   p2id,
   p2name,
   p3id,
   p3name,
   p4id,
   p4name,
   p5id,
   p5name,
   Pokemon6.id as p6id,
   Pokemon6.name as p6name
FROM
   (SELECT
      p1id,
      p1name,
      p1type,
      p2id,
      p2name,
      p2type,
      p3id,
      p3name,
      p3type,
      p4id,
      p4name,
      p4type,
      Pokemon5.id as p5id,
      Pokemon5.name as p5name,
      Pokemon5.type1 as p5type
   FROM
      (SELECT
         p1id,
         p1name,
         p1type,
         p2id,
         p2name,
         p2type,
```

```
  p3id,
  p3name,
  p3type,
  Pokemon4.id as p4id,
  Pokemon4.name as p4name,
  Pokemon4.type1 as p4type
FROM
  (SELECT
    p1id,
    p1name,
    p1type,
    p2id,
    p2name,
    p2type,
    Pokemon3.id as p3id,
    Pokemon3.name as p3name,
    Pokemon3.type1 as p3type
  FROM
    (SELECT
      Pokemon1.id as p1id,
      Pokemon1.name as p1name,
      Pokemon1.type1 as p1type,
      Pokemon2.id as p2id,
      Pokemon2.name as p2name,
      Pokemon2.type1 as p2type
    FROM Pokemon as Pokemon1
    INNER JOIN Pokemon as Pokemon2
      ON Pokemon1.id <> Pokemon2.id
      AND Pokemon1.id < Pokemon2.id
      AND Pokemon1.type1 <> Pokemon2.type1
    LEFT JOIN PokemonPairings as Pokemon12
      ON        Pokemon12.pid1 = Pokemon1.id
      AND       Pokemon12.pid2 = Pokemon2.id
    ORDER BY
      COALESCE(Pokemon12.percentage, 0) DESC
    LIMIT 1000) as Pokemon12
  INNER JOIN Pokemon as Pokemon3
    ON  p1id <> Pokemon3.id
```

```
      AND  p2id < Pokemon3.id
      AND p1type <> Pokemon3.type1
      AND p2type <> Pokemon3.type1
    LEFT JOIN PokemonPairings as Pokemon13
      ON    Pokemon13.pid1 = p1id
      AND  Pokemon13.pid2 = Pokemon3.id
    LEFT JOIN PokemonPairings as Pokemon23
      ON    Pokemon23.pid1 = p2id
      AND Pokemon23.pid2 = Pokemon3.id
    ORDER BY
      COALESCE(Pokemon13.percentage, 0) +
      COALESCE(Pokemon23.percentage, 0) DESC
    LIMIT 1000) as Pokemon123
  INNER JOIN Pokemon as Pokemon4
    ON  p1id <> Pokemon4.id
    AND p3id < Pokemon4.id
    AND p1type <> Pokemon4.type1
    AND p2type <> Pokemon4.type1
    AND p3type <> Pokemon4.type1
  LEFT JOIN PokemonPairings as Pokemon14
    ON      Pokemon14.pid1 = p1id
    AND     Pokemon14.pid2 = Pokemon4.id
  LEFT JOIN PokemonPairings as Pokemon24
    ON      Pokemon24.pid1 = p2id
    AND     Pokemon24.pid2 = Pokemon4.id
  LEFT JOIN PokemonPairings as Pokemon34
    ON      Pokemon34.pid1 = p3id
    AND     Pokemon34.pid2 = Pokemon4.id
  ORDER BY
    COALESCE(Pokemon14.percentage, 0) +
    COALESCE(Pokemon24.percentage, 0) +
    COALESCE(Pokemon34.percentage, 0) DESC
  LIMIT 1000) as Pokemon1234
INNER JOIN Pokemon as Pokemon5
  ON  p1id <> Pokemon5.id
  AND p4id < Pokemon5.id
  AND p1type <> Pokemon5.type1
  AND p2type <> Pokemon5.type1
```

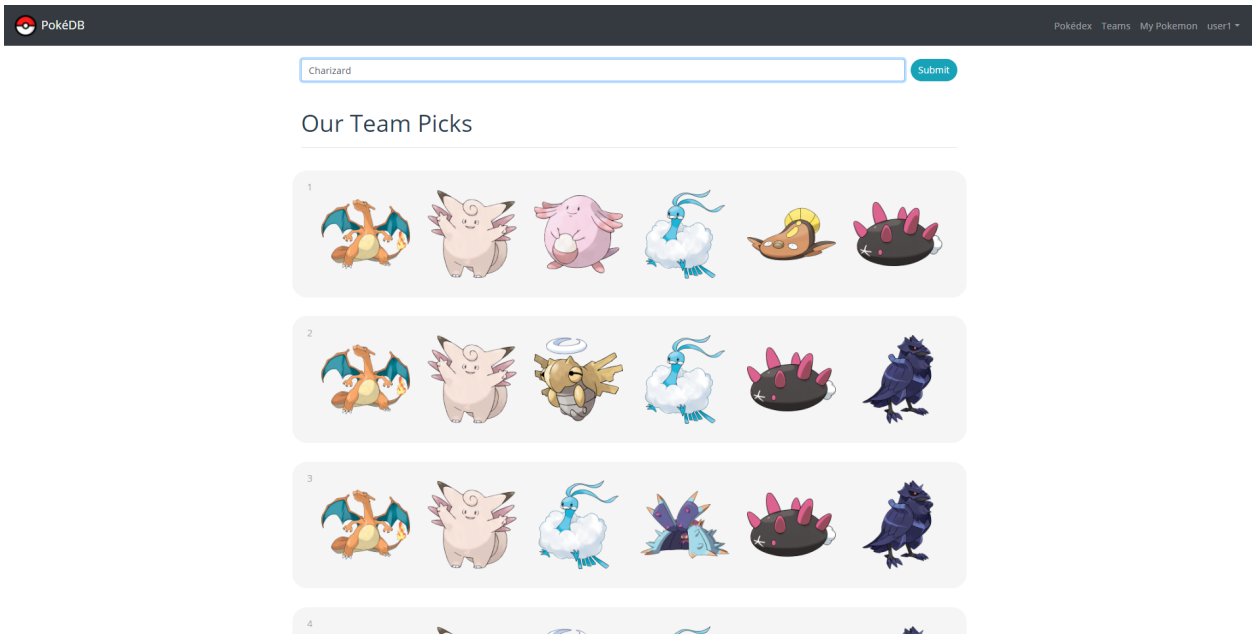
```
      AND p3type <> Pokemon5.type1
      AND p4type <> Pokemon5.type1
   LEFT JOIN PokemonPairings as Pokemon15
      ON       Pokemon15.pid1 = p1id
      AND      Pokemon15.pid2 = Pokemon5.id
   LEFT JOIN PokemonPairings as Pokemon25
      ON       Pokemon25.pid1 = p2id
      AND      Pokemon25.pid2 = Pokemon5.id
   LEFT JOIN PokemonPairings as Pokemon35
      ON       Pokemon35.pid1 = p3id
      AND      Pokemon35.pid2 = Pokemon5.id
   LEFT JOIN PokemonPairings as Pokemon45
      ON       Pokemon45.pid1 = p4id
      AND      Pokemon45.pid2 = Pokemon5.id
   ORDER BY
      COALESCE(Pokemon15.percentage, 0) +
      COALESCE(Pokemon25.percentage, 0) +
      COALESCE(Pokemon35.percentage, 0) +
      COALESCE(Pokemon45.percentage, 0) DESC
   LIMIT 1000) as Pokemon12345
INNER JOIN Pokemon as Pokemon6
   ON  p1id <> Pokemon6.id
   AND p5id < Pokemon6.id
   AND p1type <> Pokemon6.type1
   AND p2type <> Pokemon6.type1
   AND p3type <> Pokemon6.type1
   AND p4type <> Pokemon6.type1
   AND p5type <> Pokemon6.type1
LEFT JOIN PokemonPairings as Pokemon16
   ON   Pokemon16.pid1 = p1id
   AND Pokemon16.pid2 = Pokemon6.id
LEFT JOIN PokemonPairings as Pokemon26
   ON   Pokemon26.pid1 = p2id
   AND Pokemon26.pid2 = Pokemon6.id
LEFT JOIN PokemonPairings as Pokemon36
   ON   Pokemon36.pid1 = p3id
   AND Pokemon36.pid2 = Pokemon6.id
LEFT JOIN PokemonPairings as Pokemon46
```

```
   ON   Pokemon46.pid1 = p4id
   AND Pokemon46.pid2 = Pokemon6.id
LEFT JOIN PokemonPairings as Pokemon56
   ON   Pokemon56.pid1 = p5id
   AND Pokemon56.pid2 = Pokemon6.id
ORDER BY
   COALESCE(Pokemon16.percentage, 0) +
   COALESCE(Pokemon26.percentage, 0) +
   COALESCE(Pokemon36.percentage, 0) +
   COALESCE(Pokemon46.percentage, 0) +
   COALESCE(Pokemon56.percentage, 0) DESC
LIMIT 5;
```

Test Output:

```
 p1id |  p1name  | p2id |  p2name  | p3id | p3name  | p4id |  p4name  | p5id |  p5name  | p6id |  p6name
------+----------+------+----------+------+---------+------+----------+------+----------+------+-------------
  91 | Cloyster |  445 | Garchomp |  558 | Crustle |  625 | Bisharp  |  689 | Barbaracle |  776 | Turtonator
  91 | Cloyster |  151 | Mew      |  558 | Crustle |  625 | Bisharp  |  689 | Barbaracle |  776 | Turtonator
  91 | Cloyster |  196 | Espeon   |  558 | Crustle |  689 | Barbaracle |  776 | Turtonator |  855 | Polteageist
  36 | Clefable |  113 | Chansey  |  334 | Altaria |  618 | Stunfisk |  748 | Toxapex  |  771 | Pyukumuku
  36 | Clefable |  292 | Shedinja |  334 | Altaria |  748 | Toxapex  |  771 | Pyukumuku |  823 | Corviknight
```

Snapshot of search with filter for teams with Charizard:



```
SELECT
   p1id,
   p1name,
   p2id,
   p2name,
   p3id,
   p3name,
   p4id,
   p4name,
   p5id,
   p5name,
   Pokemon6.id as p6id,
   Pokemon6.name as p6name
FROM
   (SELECT
      p1id,
      p1name,
      p1type,
      p2id,
      p2name,
      p2type,
      p3id,
      p3name,
      p3type,
      p4id,
      p4name,
      p4type,
      Pokemon5.id as p5id,
      Pokemon5.name as p5name,
      Pokemon5.type1 as p5type
   FROM
      (SELECT
         p1id,
         p1name,
         p1type,
         p2id,
         p2name,
```

```
  p2type,
  p3id,
  p3name,
  p3type,
  Pokemon4.id as p4id,
  Pokemon4.name as p4name,
  Pokemon4.type1 as p4type
FROM
  (SELECT
    p1id,
    p1name,
    p1type,
    p2id,
    p2name,
    p2type,
    Pokemon3.id as p3id,
    Pokemon3.name as p3name,
    Pokemon3.type1 as p3type
  FROM
    (SELECT
      Pokemon1.id as p1id,
      Pokemon1.name as p1name,
      Pokemon1.type1 as p1type,
      Pokemon2.id as p2id,
      Pokemon2.name as p2name,
      Pokemon2.type1 as p2type
    FROM Pokemon as Pokemon1
    INNER JOIN Pokemon as Pokemon2
      ON Pokemon1.id <> Pokemon2.id
      AND Pokemon1.type1 <> Pokemon2.type1
    LEFT JOIN PokemonPairings as Pokemon12
      ON        Pokemon12.pid1 = Pokemon1.id
      AND       Pokemon12.pid2 = Pokemon2.id
    WHERE LOWER(Pokemon1.name) = LOWER('Charizard')
    ORDER BY
      COALESCE(Pokemon12.percentage, 0) DESC
    LIMIT 1000) as Pokemon12
  INNER JOIN Pokemon as Pokemon3
```

```
      ON  p1id <> Pokemon3.id
      AND  p2id < Pokemon3.id
      AND p1type <> Pokemon3.type1
      AND p2type <> Pokemon3.type1
    LEFT JOIN PokemonPairings as Pokemon13
      ON    Pokemon13.pid1 = p1id
      AND  Pokemon13.pid2 = Pokemon3.id
    LEFT JOIN PokemonPairings as Pokemon23
      ON    Pokemon23.pid1 = p2id
      AND Pokemon23.pid2 = Pokemon3.id
    ORDER BY
      COALESCE(Pokemon13.percentage, 0) +
      COALESCE(Pokemon23.percentage, 0) DESC
    LIMIT 1000) as Pokemon123
  INNER JOIN Pokemon as Pokemon4
    ON  p1id <> Pokemon4.id
    AND p3id < Pokemon4.id
    AND p1type <> Pokemon4.type1
    AND p2type <> Pokemon4.type1
    AND p3type <> Pokemon4.type1
  LEFT JOIN PokemonPairings as Pokemon14
    ON      Pokemon14.pid1 = p1id
    AND     Pokemon14.pid2 = Pokemon4.id
  LEFT JOIN PokemonPairings as Pokemon24
    ON      Pokemon24.pid1 = p2id
    AND     Pokemon24.pid2 = Pokemon4.id
  LEFT JOIN PokemonPairings as Pokemon34
    ON      Pokemon34.pid1 = p3id
    AND     Pokemon34.pid2 = Pokemon4.id
  ORDER BY
    COALESCE(Pokemon14.percentage, 0) +
    COALESCE(Pokemon24.percentage, 0) +
    COALESCE(Pokemon34.percentage, 0) DESC
  LIMIT 1000) as Pokemon1234
INNER JOIN Pokemon as Pokemon5
  ON  p1id <> Pokemon5.id
  AND p4id < Pokemon5.id
  AND p1type <> Pokemon5.type1
```

```
    AND p2type <> Pokemon5.type1
    AND p3type <> Pokemon5.type1
    AND p4type <> Pokemon5.type1
  LEFT JOIN PokemonPairings as Pokemon15
    ON      Pokemon15.pid1 = p1id
    AND     Pokemon15.pid2 = Pokemon5.id
  LEFT JOIN PokemonPairings as Pokemon25
    ON      Pokemon25.pid1 = p2id
    AND     Pokemon25.pid2 = Pokemon5.id
  LEFT JOIN PokemonPairings as Pokemon35
    ON      Pokemon35.pid1 = p3id
    AND     Pokemon35.pid2 = Pokemon5.id
  LEFT JOIN PokemonPairings as Pokemon45
    ON      Pokemon45.pid1 = p4id
    AND     Pokemon45.pid2 = Pokemon5.id
  ORDER BY
    COALESCE(Pokemon15.percentage, 0) +
    COALESCE(Pokemon25.percentage, 0) +
    COALESCE(Pokemon35.percentage, 0) +
    COALESCE(Pokemon45.percentage, 0) DESC
  LIMIT 1000) as Pokemon12345
INNER JOIN Pokemon as Pokemon6
  ON  p1id <> Pokemon6.id
  AND p5id < Pokemon6.id
  AND p1type <> Pokemon6.type1
  AND p2type <> Pokemon6.type1
  AND p3type <> Pokemon6.type1
  AND p4type <> Pokemon6.type1
  AND p5type <> Pokemon6.type1
LEFT JOIN PokemonPairings as Pokemon16
  ON   Pokemon16.pid1 = p1id
  AND Pokemon16.pid2 = Pokemon6.id
LEFT JOIN PokemonPairings as Pokemon26
  ON   Pokemon26.pid1 = p2id
  AND Pokemon26.pid2 = Pokemon6.id
LEFT JOIN PokemonPairings as Pokemon36
  ON   Pokemon36.pid1 = p3id
  AND Pokemon36.pid2 = Pokemon6.id
```

```
LEFT JOIN PokemonPairings as Pokemon46
   ON   Pokemon46.pid1 = p4id
   AND Pokemon46.pid2 = Pokemon6.id
LEFT JOIN PokemonPairings as Pokemon56
   ON   Pokemon56.pid1 = p5id
   AND Pokemon56.pid2 = Pokemon6.id
ORDER BY
   COALESCE(Pokemon16.percentage, 0) +
   COALESCE(Pokemon26.percentage, 0) +
   COALESCE(Pokemon36.percentage, 0) +
   COALESCE(Pokemon46.percentage, 0) +
   COALESCE(Pokemon56.percentage, 0) DESC
LIMIT 5;
```

Test Output

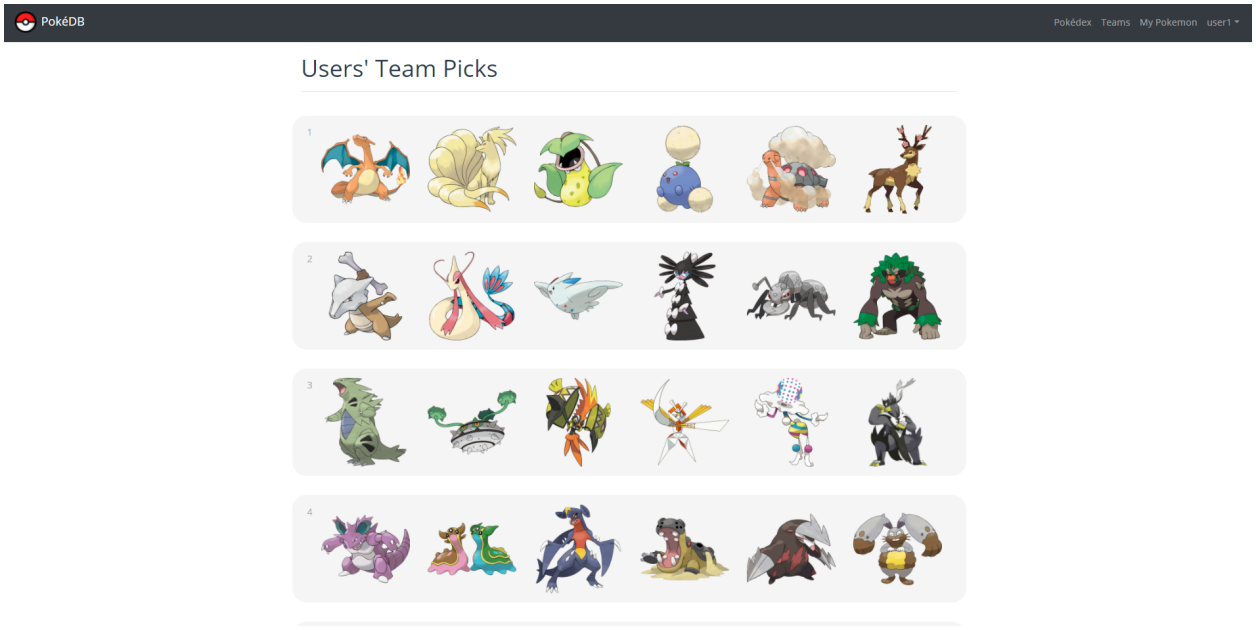
```
 p1id |  p1name   | p2id |  p2name  | p3id |  p3name  | p4id | p4name  | p5id |  p5name   | p6id |   p6name
------+-----------+------+----------+------+----------+------+---------+------+-----------+------+-------------
   6 | Charizard |   36 | Clefable |  113 | Chansey  |  334 | Altaria |  618 | Stunfisk  |  771 | Pyukumuku
   6 | Charizard |   36 | Clefable |  292 | Shedinja |  334 | Altaria |  771 | Pyukumuku |  823 | Corviknight
   6 | Charizard |   36 | Clefable |  334 | Altaria  |  748 | Toxapex |  771 | Pyukumuku |  823 | Corviknight
   6 | Charizard |   36 | Clefable |  292 | Shedinja |  334 | Altaria |  748 | Toxapex   |  823 | Corviknight
   6 | Charizard |   36 | Clefable |  292 | Shedinja |  748 | Toxapex |  771 | Pyukumuku |  823 | Corviknight
```

Feature 6: Recommending User-Generated Pokemon Teams

When users click on the “Teams” button, the application will also recommend a list of user-generated Pokemon teams (retrieved from Pokemon Showdown games) sorted by win rate where the number of games played with this team is at least 2. We inner-join the Teams table with the Pokemon table 6 times to get the names of the Pokemon on the team that will appear when you hover over each Pokemon image. Finally, we show the 5 teams with the highest win rate.

The user can also filter on teams containing a specific Pokemon i.e. if the user wishes to find teams containing Charizard, they can search “Charizard” in the search bar and the program will only generate teams that contain Charizard.

Snapshot of no filter search:



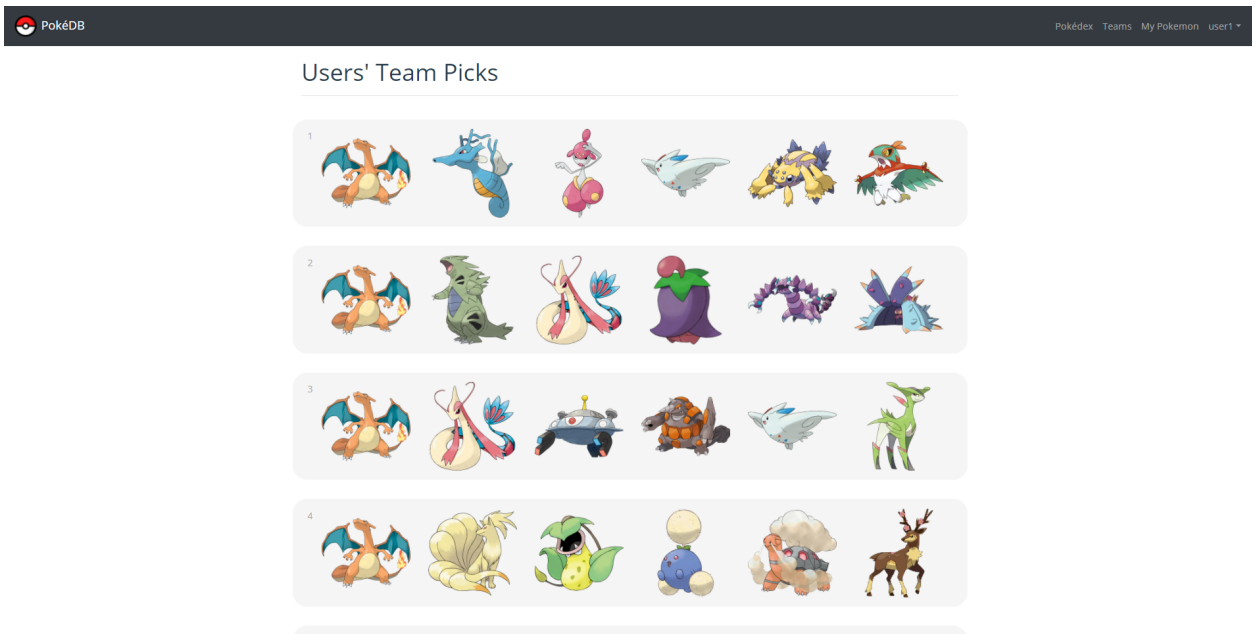

```
SELECT pid1,
    Pokemon1.name,
    pid2,
    Pokemon2.name,
    pid3,
    Pokemon3.name,
    pid4,
    Pokemon4.name,
    pid5,
    Pokemon5.name,
    pid6,
    Pokemon6.name
FROM Team
INNER JOIN Pokemon as Pokemon1
ON pid1 = Pokemon1.id
INNER JOIN Pokemon as Pokemon2
ON pid2 = Pokemon2.id
INNER JOIN Pokemon as Pokemon3
ON pid3 = Pokemon3.id
INNER JOIN Pokemon as Pokemon4
ON pid4 = Pokemon4.id
INNER JOIN Pokemon as Pokemon5
ON pid5 = Pokemon5.id
```

INNER JOIN Pokemon as Pokemon6
ON pid6 = Pokemon6.id
WHERE wins + losses >= 2
ORDER BY CAST(wins AS FLOAT) / CAST(wins + losses AS FLOAT) DESC
LIMIT 5;

Test Output

```
pid1 |   name    | pid2 |    name    | pid3 |    name    | pid4 |    name    | pid5 |    name     | pid6 |   name
------+-----------+------+------------+------+------------+------+------------+------+-------------+------+-----------
 248 | Tyranitar |  598 | Ferrothorn |  785 | Tapu Koko  |  798 | Kartana    |  806 | Blacephalon |  892 | Urshifu
   6 | Charizard |   38 | Ninetales  |   71 | Victreebel |  189 | Jumpluff   |  324 | Torkoal     |  586 | Sawsbuck
  50 | Diglett   |  177 | Natu       |  240 | Magby      |  559 | Scraggy    |  629 | Vullaby     |  696 | Tyrunt
   3 | Venusaur  |   36 | Clefable   |  530 | Excadrill  |  681 | Aegislash  |  784 | Kommo-o     |  892 | Urshifu
 105 | Marowak   |  350 | Milotic    |  468 | Togekiss   |  576 | Gothitelle |  632 | Durant      |  812 | Rillaboom
```

Snapshot of search with filter for teams with Charizard:



SELECT pid1,
Pokemon1.name,
pid2,
Pokemon2.name,
pid3,
Pokemon3.name,
pid4,

```
    Pokemon4.name,
    pid5,
    Pokemon5.name,
    pid6,
    Pokemon6.name
FROM Team
INNER JOIN Pokemon as Pokemon1
ON pid1 = Pokemon1.id
INNER JOIN Pokemon as Pokemon2
ON pid2 = Pokemon2.id
INNER JOIN Pokemon as Pokemon3
ON pid3 = Pokemon3.id
INNER JOIN Pokemon as Pokemon4
ON pid4 = Pokemon4.id
INNER JOIN Pokemon as Pokemon5
ON pid5 = Pokemon5.id
INNER JOIN Pokemon as Pokemon6
ON pid6 = Pokemon6.id
WHERE wins + losses >= 2
AND (LOWER(Pokemon1.name) = LOWER('Charizard')
OR LOWER(Pokemon2.name) = LOWER('Charizard')
OR LOWER(Pokemon3.name) = LOWER('Charizard')
OR LOWER(Pokemon4.name) = LOWER('Charizard')
OR LOWER(Pokemon5.name) = LOWER('Charizard')
OR LOWER(Pokemon6.name) = LOWER('Charizard'))
ORDER BY CAST(wins AS FLOAT) / CAST(wins + losses AS FLOAT) DESC
LIMIT 5;
```

Test Output

```
 pid1 |   name    | pid2 |   name    | pid3 |    name    | pid4 |   name    | pid5 |    name    | pid6 |   name
------+-----------+------+-----------+------+------------+------+-----------+------+------------+------+----------
   6 | Charizard |  230 | Kingdra   |  308 | Medicham   |  468 | Togekiss  |  596 | Galvantula |  701 | Hawlucha
   6 | Charizard |  248 | Tyranitar |  350 | Milotic    |  421 | Cherrim   |  452 | Drapion    |  748 | Toxapex
   6 | Charizard |   38 | Ninetales |   71 | Victreebel |  189 | Jumpluff  |  324 | Torkoal    |  586 | Sawsbuck
   6 | Charizard |  199 | Slowking  |  442 | Spiritomb  |  646 | Kyurem    |  888 | Zacian     |  892 | Urshifu
   6 | Charizard |  350 | Milotic   |  462 | Magnezone  |  464 | Rhyperior |  468 | Togekiss   |  640 | Virizion
```

Additional Feature 1: Better security

All passwords are hashed before getting stored in the tables to ensure that even if the tables get leaked, the password information is still secure. We use the bcrypt library to compute the hashes of the passwords. The login and change password features were modified to compare the hashes instead of comparing the passwords directly.

Here is an example of passwords stored as hashes:

	username [PK] character varying (30)	email character varying (50)	password character varying (80)
1	user1	user1@gmail.com	$2b$12$WQkPRDNuqyg15s74dTPcAehU/ok.x8Zpq3U3jGQ7NIIP3hIFF0ZE2
2	user2	user2@gmail.com	$2b$12$E6iV0NWWBSwr.I36.QsKMuDBG86ornMDYQLnVrAkxuocRuIPfM45W
3	user3	user3@gmail.com	$2b$12$VdChJ/GZ6/XUGy/qfBgc8.92MR0G5gqx5IrddTHmP89w.TKDvkBsm

Additionally, all calls to queries were modified to ensure that SQL injection is not possible. We used the functionalities available in psycopg2 to do so.

Additional Feature 2: Type weaknesses and resistances

Using the data stored in the Effectiveness table, we added functionalities that use a Pokemon's innate weakness or resistance to attacks of a given type.

First, on the Pokemon details page we list how effective a move of each type is against a Pokemon with the given type combination. For example, a Fire and Flying type Pokemon like Charizard is completely immune to Ground type attacks, but incredibly weak to Rock type attacks - they deal 4 times the normal damage.

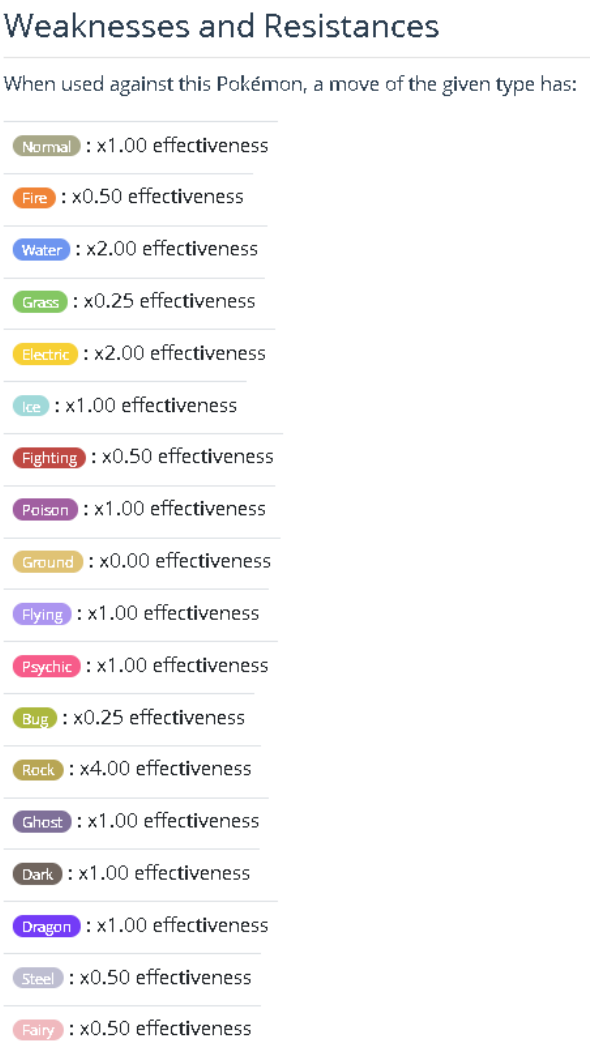
6: Charizard

Fire Flying

Base Stats

HP: 78	Spd: 100
Atk: 84	Def: 78
SpAtk: 109	SpDef: 85

Selected Moves Known: Aerial Ace, Air Cutter, Air Slash, Ancient Power, Beat Up, Bide, Bite, Blast Burn, Body Slam, Brick Break, Brutal Swing, Bulldoze, Counter, Crunch, Cut, Dig, Double-Edge, Dragon Breath, Dragon Claw, Dragon Pulse, Dragon Rage, Dragon Rush, Dragon Tail, Dynamic Punch, Earthquake, Echoed Voice, Ember, Facade, Fire Blast, Fire Fang, Fire Pledge, Fire Punch, Fire Spin, Fissure, Flame Burst, Flame Charge, Flamethrower, Flare Blitz, Fling, Fly, Focus Blast, Focus Punch, Frustration, Fury Cutter, Giga Impact, Headbutt, Heat Wave, Hidden Power, Hyper Beam, Incinerate, Inferno, Iron Tail, Mega Kick, Mega Punch, Metal Claw, Mud-Slap, Natural Gift, Ominous Wind, Outrage, Overheat, Power-Up Punch, Rage, Return, Rock Slide, Rock Smash, Rock Tomb, Round, Scratch, Secret Power, Seismic Toss, Shadow Claw, Skull Bash, Sky Drop, Slash, Snore, Solar Beam, Steel Wing, Strength, Submission, Swift, Take Down, Thunder Punch, Twister, Wing Attack



If the Pokemon has 2 types, the query used to get its weaknesses and resistances is:

SELECT e1.moveType, (e1.effectiveness * e2.effectiveness)
FROM Effectiveness AS e1, Effectiveness AS e2
WHERE e1.pokemonType = <type1> AND e2.pokemonType = <type2> AND e1.moveType = e2.moveType

If the Pokemon has 1 type, the query used instead is:

SELECT moveType, effectiveness
FROM Effectiveness
WHERE pokemonType = <type>

Second, on the Pokedex page we can now query for Pokemon that are weak to or resist attacks of a certain type. For example, suppose our opponent is a Charizard, and we want to find a Pokemon that is resistant to both Fire and Flying type attacks. Any Rock type Pokemon that does not have another type weak to Fire or Flying type attacks should match our query.

Name:

Enter Pokemon Name

ID: Enter #

Max HP: Enter #

Max Spd: Enter #

Max Atk: Enter #

Max Def: Enter #

Max SpAtk: Enter #

Max SpDef: Enter #

Min HP: Enter #

Min Spd: Enter #

Min Atk: Enter #

Min Def: Enter #

Min SpAtk: Enter #

Min SpDef: Enter #

Type:

☐ Normal ☐ Fire ☐ Water ☐ Grass ☐ Electric ☐ Ice ☐ Fighting ☐ Poison ☐ Ground ☐ Flying ☐ Psychic ☐ Bug
☐ Rock ☐ Ghost ☐ Dark ☐ Dragon ☐ Steel ☐ Fairy

Move Name:

Enter Move Name

Max PP: Enter #

Max Power: Enter #

Max Accuracy: Enter #

Damage Type: Select Type

Min PP: Enter #

Min Power: Enter #

Min Accuracy: Enter #

Move Type:

☐ Normal ☐ Fire ☐ Water ☐ Grass ☐ Electric ☐ Ice ☐ Fighting ☐ Poison ☐ Ground ☐ Flying ☐ Psychic ☐ Bug
☐ Rock ☐ Ghost ☐ Dark ☐ Dragon ☐ Steel ☐ Fairy

Weaknesses:

☐ Normal ☐ Fire ☐ Water ☐ Grass ☐ Electric ☐ Ice ☐ Fighting ☐ Poison ☐ Ground ☐ Flying ☐ Psychic ☐ Bug
☐ Rock ☐ Ghost ☐ Dark ☐ Dragon ☐ Steel ☐ Fairy

Resistances:

☐ Normal ☑ Fire ☐ Water ☐ Grass ☐ Electric ☐ Ice ☐ Fighting ☐ Poison ☐ Ground ☑ Flying ☐ Psychic ☐ Bug
☐ Rock ☐ Ghost ☐ Dark ☐ Dragon ☐ Steel ☐ Fairy

Reset Submit

Pokédex

Click on a Pokémon's image to learn more.

74: Geodude

Rock Ground

Base Stats

HP: 40	Spd: 20
Atk: 80	Def: 100
SpAtk: 30	SpDef: 30

75: Graveler

Rock Ground

Base Stats

HP: 55	Spd: 35
Atk: 95	Def: 115
SpAtk: 45	SpDef: 45

76: Golem

Rock Ground

Base Stats

HP: 80	Spd: 45
Atk: 120	Def: 130
SpAtk: 55	SpDef: 65

95: Onix

Rock Ground

Base Stats

HP: 35	Spd: 70
Atk: 45	Def: 160
SpAtk: 30	SpDef: 45

111: Rhyhorn

Ground Rock

112: Rhydon

Ground Rock

138: Omanyte

Rock Water

139: Omastar

Rock Water

This functionality was implemented by first querying for type combinations that match the weakness/resistance criteria, then selecting only Pokemon with those type combinations. The query to get all type combinations (including individual types) that are weak to/resist attacks of a given type is:

(

SELECT e1.pokemonType, e2.pokemonType

FROM Effectiveness AS e1, Effectiveness AS e2

WHERE e1.moveType = <attack type>

AND e2.moveType = <attack type>

AND (e1.effectiveness * e2.effectiveness) <less than for resistance, greater than for weakness> 1

) UNION (

SELECT pokemonType, NULL
FROM Effectiveness
WHERE moveType = <attack type>
AND effectiveness <less than for resistance, greater than for weakness> 1
)

When the user selects multiple resistances through filtering, we return any Pokemon that are resistant to *all* of the selected types. (Similarly for weaknesses) To do so, we execute the above query for each attack type selected, and find the intersection of their outputs as sets in Python. This was done in Python instead of SQL to avoid complicated queries with a variable number of joins, and so that we can stop querying early if we realize that no Pokemon can have the required weaknesses/resistances.

Additional Feature 3: Multi-User Access Control

To ensure multiple users can access PokeDB and see expected and consistent reads / writes, we added a lock that all SQL queries must acquire before and release after. By doing so, this lock protects the shared variables (critical sections) - connection to the database and cursor - thus ensuring that the results retrieved from reads and writes to the database are consistent and as expected even if multiple users access PokeDB at a time, achieving multi-user access control.

Performance

Many of our queries rely on the static data present in the Pokemon table. These can be sped up using indices without having additional update costs, as we do not need to update the table.

For the Pokemon table, an index on the pokemon's name speeds up almost all our queries. This index is created by adding the "UNIQUE" constraint to the "name" attribute. In particular, this addition speeds up the following queries:

- Pokedex querying, as the user can query pokemon by their name
- Insertion and updating of owned pokemon, as the user specifies species by name
- Recommending program-generated pokemon teams, as we filter the results based on pokemon names specified by the user
- Recommending user-generated pokemon teams, as we filter the results based on pokemon names specified by the user

Even with the first index, the recommendation of program-generated pokemon teams requires up to 4 seconds. The query uses multiple joins with a join condition that includes the primary type of the pokemon. We add an index on the "type1" attribute of the Pokemon table to speed this query up. After the optimization, the time required for the query is reduced to around 2 seconds. Notice that this index also helps speed up queries for our Pokedex feature.

With the indices mentioned above, all other queries can be executed in under half a second on the sample dataset. Since this performance is sufficient for our application, we did not add any additional indices.

In addition to these two indices, Feature 5 requires a join with 21 tables (6 Pokemon tables with 15 PokemonPairings tables for each pair of Pokemon). Originally, we used cross-joins between the 6 Pokemon tables, meaning that we would potentially search 800^6 tuples (there are approximately 800 Pokemon), which is too expensive in terms of memory and computation. Thus, as a heuristic optimization, we join the tables one-by-one and greedily select the top 1000 intermediate results with the highest score at each level. By limiting the intermediate results table to 1000, the memory and computation requirements are reduced drastically, allowing us to program-generate Pokemon teams in a reasonable amount of time.

Member Contributions

Sophia Pietsch (stpietsc)

- Implemented Owned Pokemon feature
- Prevented SQL injection
- Cleanup of front end
- Wrote script and recorded video for relevant sections of the presentation
- Edited video presentation for all members

Sofia Fong (s22fong)

- Updated relevant sections of the report
- Updated front-end for teams page
- Added password hashing
- Wrote script and recorded video for relevant sections of presentation

Ethan Zhang (y2788zha)

- Implemented additional feature for type weaknesses/resistances
    - List weaknesses/resistances on Pokemon details page
    - Query for pokemon by weaknesses/resistances
- Added image scraper and front end support for shiny Pokemon
- Improved user interface of features 1 and 2
- Updated relevant sections of report
- Wrote script and recorded video for relevant sections of presentation

Kai Sun (k49sun)

- Added a lock to protect critical sections in the code to allow multi-user access control
- Updated relevant sections of the report
- Wrote a script and recorded video for relevant sections of presentation