**Problem Statement**

Many people who are app users are also gamers. The may play casually, or they may play in a more focused setting. One thing that is common though, is that they use their internet connected devices to check information about the game they’re playing; how to find secret paths and tips and tricks. One game that is common amongst gamers is the popular Pokémon franchise. In this franchise, one can collect and battle with creatures called Pokémon. The user can build teams of Pokémon, to battle other Pokémon. These creatures have different stats or values which determine how strong they are against other creatures in the game. As players of this series, we can say that we have definitely kept our computers by our side while we play, checking these stats as we battle. The goal of this assignment is to showcase a database that reflects a real-world problem. The issue of continuously searching for the information about the Pokémon is one that affects us, and one that we think can be resolved with such a project. To this end, we are creating a web-application which will allow users to curate multiple Pokémon. They will be able to see their stats, their weaknesses, their strengths, their locations, and will even be able to build teams out of the Pokémon they have encountered.

The database will consist of 6 entities, 2 of which are weak entities. The entities follow as: Pokémon, Region, Type, Trainer, Form, and Stats. The Pokémon entity consists of 3 primary attributes: an ID number, a name, and a climate. The ID number uniquely identifies each creature. Though the name is unique as well, it provided unnecessary additional information in identifying a creature, so it does not need to be part of the entity’s primary key. The climate identifies what type of climate within a region (an in-game designation for different worlds, indicating what game the creature can be found in) the Pokémon can be caught. The Pokémon entity has two weak entities connected to it: Form and Stats. Both of these are identified using the primary key of the Pokémon they descend from (i.e. their ID number). The stats entity consists of 5 6 attributes: HP, Attack, Defense, Special Attack, Special Defense, and Speed. These statistics determine how the Pokémon fares in battle against another Pokémon. The stat spread does not need to be unique to a single Pokémon, but a Pokémon has a unique stat spread. The only instance in which the creatures has a different stat spread, is if it changes forms. The is marked by a relationship between stats and form which updates the creature’s stats. The form change is simply an alternate version of the same creature. The Pokémon entity is connected to a Type entity as well. The type entity is used to determine the strengths and weaknesses between Pokémon in battle. The weaknesses and strengths are defined in the type entity itself instead of the Pokémon. This minimizes the amount of data to store in the database.

The other primary entity is the Trainer entity. The trainer entity is the one that the user takes on, and contains the users name and their unique trainer ID. The trainer ID/username is used as this entity’s primary key. In addition to the user ID, the trainer entity contains the name of the trainer. The trainer is allowed to register multiple Pokémon, thus building up their Pokédex. The user can also select an in-game region that they are from. This will allow for further searching by allowing the user to narrow down Pokémon close to them. The username is inherited from a user class, which also contains the password credentials to log in to the database. The region is another entity, whose use has been described through both the Trainer and the Pokémon entities. It is used to help narrow down Pokémon to trainer’s regions and help find unique Pokémon.

Finally, a tertiary relationship exists between the Pokémon, Region, and Trainer entities. This allows the trainer to have teams in multiple regions (i.e. multiple games) consisting of many different Pokémon. This is a key feature of the games, and something that is a quality of life enhancement for players who own and play multiple versions of the game. This relationship can also be represented as a multi-valued dependency where the User entity can have multiple trainer entities, each which are from a different region, with different collections.

Due to the implementation of this project as a webapp, it allows for maximum user penetration. Any gamer of the series will have access to the utility. Additionally, most people who would use this tool, are already querying multiple sites at a time. With this tool, they can conveniently access all the information they need in one tool, on any deice available to them.

**Documentation**

The webapp was built on the Django platform. Django is a Python module that allows for powerful user customization and efficient website construction. The database construction in Django is different from standard tools. It allows the user to construct their tables/entities as “Data Object Models”. It implicitly creates the table from the user constraints, but does allow for users to construct the databases using standard SQL commands as well. The convenience of this tool, is that it simplifies the process of splitting up multivalued dependencies into separate tables, by splitting them up in the background. After this, the user can write decision logic in python/Django syntax. The program requires certain dependencies to be installed before running: Python 3.6, the following pip packages (pillow, django, django-multiselectfields). The virtual environment for this program is stored in the source folder. The server must be started up through a terminal or command line window with the following command from within the source folder: python manage.py runserver. In the console, we have set it to output all the SQL commands being run to load our elements. The site can be loaded by navigating to a Chrome or Firefox browser and navigating to 127.0.0.1:8000. User profiles can be added at the admin panel: 127.0.0.1:8000/admin. The log in for this site is: tangjl, pwd: Welcome11. From this panel, the user must create first a new user, and then a new trainer. After doing so, the user can return to the main site, and log out of the admin account, and into their account. Using the checkboxes, the user can add and remove creatures from their personal collection (which can be accessed from the collection button located under the username). The search functionality allows the user to search for Pokémon using different search parameter. If nothing is selected, and the user presses search, all Pokémon are displayed in numerical order. The user can select a combination of Region, Type, and Climate to search by. The user can also specify what order they want the information sorted in. Additionally, the user can search part of a Pokémon name, and see all Pokémon that contain that segment.

**Constraints & Domain**

* User
  + Username (VARCHAR 25) – D: {null, rohatgia, tangjl, etc.}, unique, null not allowed, Primary Key
  + Password (VARCHAR 25) – D: {null, Abcd123!, …, etc.} not unique, null not allowed
* Trainer
  + Username (VARCHAR 25) – D: {null, rohatgia, tangjl, etc.}, unique, null not allowed, Foreign Key, Primary Key
  + Name (VARCHAR 25) – D: {null, Brendan, Gen, …, etc.}, not unique, null not allowed
  + Region (VARCHAR 10) – D: {null, Kanto, Johto, Hoenn, Sinnoh, Unova, Kalos}, not unique, null not allowed, Foreign Key
  + NumRegions (INT 1) – D: { null, 0-6}, not unique, null allowed, Sequence Value
  + Pokemon (INT 3) – D: {null, 001, 002, …, 802}, not unique, null allowed, Multi Valued Dependency, Foreign Key, Multi-Valued Dependency (802 Max)
* Pokemon
  + Number (INT 3) - D: {null, 000, …, 802},unique, null not allowed, Primary Key
  + Name (VARCHAR 25) – D: {null, Bulbasaur, Ivysaur, …, Marshadow}, unique, null not allowed
  + Climate (VARCHAR 10)– D: {null, Ocean, Grass, Mountain, Tropical, Snow}, not unique, null not allowed
  + Region (VARCHAR 10) – D: {null, Kanto, Johto, Hoenn, Sinnoh, Unova, Kalos}, not unique, null not allowed, Foreign Key
  + Type (VARCHAR 10) – D: {null, Normal, Fighting, Grass, Water, Fire, Ground, Rock, Bug, Psychic, Dragon, Poison, Dark, Normal/Fighting, etc.}, not unique, null not allowed, Foreign Key, Multi-Valued Dependency (2 Max)
  + Evolves Into (INT 3) – D: {null, 002, 003, etc.}, unique, null allowed, Foreign Key, Unary
  + Evolves From (INT 3) – D: {null, 001, 002, etc.}, unique, null allowed, Foreign Key, Unary
* Stats
  + Number (INT 3) – D: {null, 000, …, 802}, unique, null not allowed, Primary Key, Foreign Key
  + HP (INT 3) – D: {null, 001, …, 255}, not unique, null not allowed
  + Attack (INT 3) – D: {null, 001, …, 255}, not unique, null not allowed
  + Defense (INT 3) – D: {null, 001, …, 255}, not unique, null not allowed
  + Special Attack (INT 3) – D: {null, 001, …, 255}, not unique, null not allowed
  + Special Defense (INT 3) – D: {null, 001, …, 255}, not unique, null not allowed
  + Speed (INT 3) – D: {null, 001, …, 255}, not unique, null not allowed
* Region
  + Name (VARCHAR 10) – D: {null, Kanto, Johto, Hoenn, Sinnoh, Unova, Kalos}, unique, null not allowed, Primary Key
* Type
  + Type (VARCHAR 20) – D: {null, Normal, Fighting, Grass, Water, Fire, Ground, Rock, Bug, Psychic, Dragon, Poison, Dark, Normal/Fighting, etc.}, unique, null not allowed, Primary Key
  + Weakness (VARCHAR 10) – D: {null, Normal, Fighting, Grass, Water, Fire, Ground, Rock, Bug, Psychic, Dragon, Poison, Dark}, not unique, null allowed, Multi-Valued Dependency (18 Max), Foreign Key, Unary
  + Weakness (VARCHAR 10)– D: {null, Normal, Fighting, Grass, Water, Fire, Ground, Rock, Bug, Psychic, Dragon, Poison, Dark}, not unique, null allowed, Multi-Valued Dependency (18 Max), Foreign Key, Unary

**Normalization**

* **Relations**
  + Pokémon(number, name, FK region, climate, FK type, FK Evolution, FK Previous Form, {Type})
  + User(Username, Password)
  + Trainer(User.Username, name, numRegions, FK Region, {Pokémon})
  + Region(Name)
  + Type(Name, FK {Weakness}, FK {Strength})
  + Stats(Pokémon.Number, HP, Attack, Defense, Special Attack, Special Defense, Speed)

1. **User(Username, Password, Name, Trainer.Region, numRegions, {Pokémon})**
   1. ***1NF: Is every element atomic?***  
      The Pokémon attribute is a Multi-Valued Dependency, so a new relation must be defined PokemonCollection(User.Username, Pokemon.Number).
   2. ***2NF: Do any non-key attributes depend on only parts of the Primary Key?***  
      There is a single Functional Dependency (Username -> Password, Name, Trainer.Region, numRegions). All attributes depend on it.  
      There is a single Multi-Valued Dependency, whose attributes are the primary key.
   3. ***3NF: Are there any attributes which are dependent on a non-key attribute?***  
      The region name is derived from the region, as it is a foreign key. This ensures that it is not dependent on only parts of the primary key. Username acts as a superkey. This is in 3NF
   4. ***BCNF: Is there any redundancy in functional dependencies?***  
      Username can used to be identify all Pokemon.Number values in the Multi-Valued Dependencies. The Pokemon.Number values are all foreign keys. It is a superkey for both the Functional Dependency and the Multi-Valued Dependency. This is in BCNF
2. **Region(Name)**
   1. ***1NF: Is every element atomic?***  
      Name is the attribute and the primary key. This is in 1NF
   2. ***2NF: Do any non-key attributes depend on only parts of the Primary Key?***

The primary key is the only attribute. This is in 2NF

* 1. ***3NF: Are there any attributes which are dependent on a non-key attribute?***

The primary key is the only attribute. This is in 3NF

* 1. ***BCNF: Is there any redundancy in functional dependencies?***

The primary key is the only attribute, there is only one functional dependency. This is in BCNF

1. **Type(Name, {Weakness}, {Strength})**
   1. ***1NF: Is every element atomic?***  
      The name attribute can be used to identify the weaknesses and strengths for that specific elements. Weakness and Strength can have multiple values, so they are represented as Multi-Valued Dependencies. To be indivisible, they must be represented as their own tables. The relations follows as Weakness(Type1.Name, Type2.Name), Strength(Type1.Name, Type2.Name).
   2. ***2NF:* *Do any non-key attributes depend on only parts of the Primary Key?***In the functional dependency and multi-valued dependencies, the only dependencies are on the type name, which is a unary relationship. This is in 2NF
   3. ***3NF:* *Are there any attributes which are dependent on a non-key attribute?***The name of the base type can be used to identify the type itself, as well as all the elements it is weak against and strong against. This is in 3NF
   4. ***BCNF:* *Is there any redundancy in functional dependencies?***In all 3 dependencies, the name of the base type can be used to identify unique elements. The Multi-Value Dependencies are identified with two type name, so they represent different information. This is in BCNF
2. **Pokémon(Number, Name, Evolution, Previous Evolution, Pokemon.Region, HP, Attack, Defense, Special Attack, Special Defense, Speed, {Type})**
   1. ***1NF: Is every element atomic?***  
      The Number attribute can be used to identify every other attribute. Some attributes are foreign keys, ensuring that none of their sub terms would make Pokémon non-atomic. There is a Multi-Valued Dependency in Type, which can be split into the relation PokemonType(Pokemon.Number, Type.Name)
   2. ***2NF:* *Do any non-key attributes depend on only parts of the Primary Key?***  
      In both the functional dependency and the multi-valued dependencies, the attributes can be determined by the Number attribute. This is in 2NF.
   3. ***3NF:* *Are there are any attributes which are dependent on a non-key attribute?***  
      The Multi-Value Dependency is dependent on another attribute to uniquely identify it, but this value is a foreign key. Thus, this is not directly an attribute of Pokémon. This is in 3NF
   4. ***BCNF:* *Is there any redundancy in functional dependencies?***  
      There is one functional dependency and one multi-valued dependency. Both these can be identified with the Number attribute. There is no redundancy. This is in BCNF.