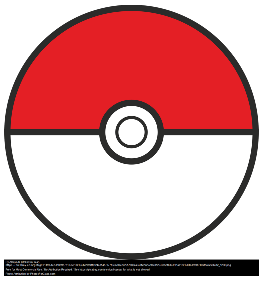
COMP 3380 – Database Project Part 3 - Report

Group Members:

- **Alyssa Gregorash**
- **Henry Wong**
- **Luc Miron**



Due November 27, 2022

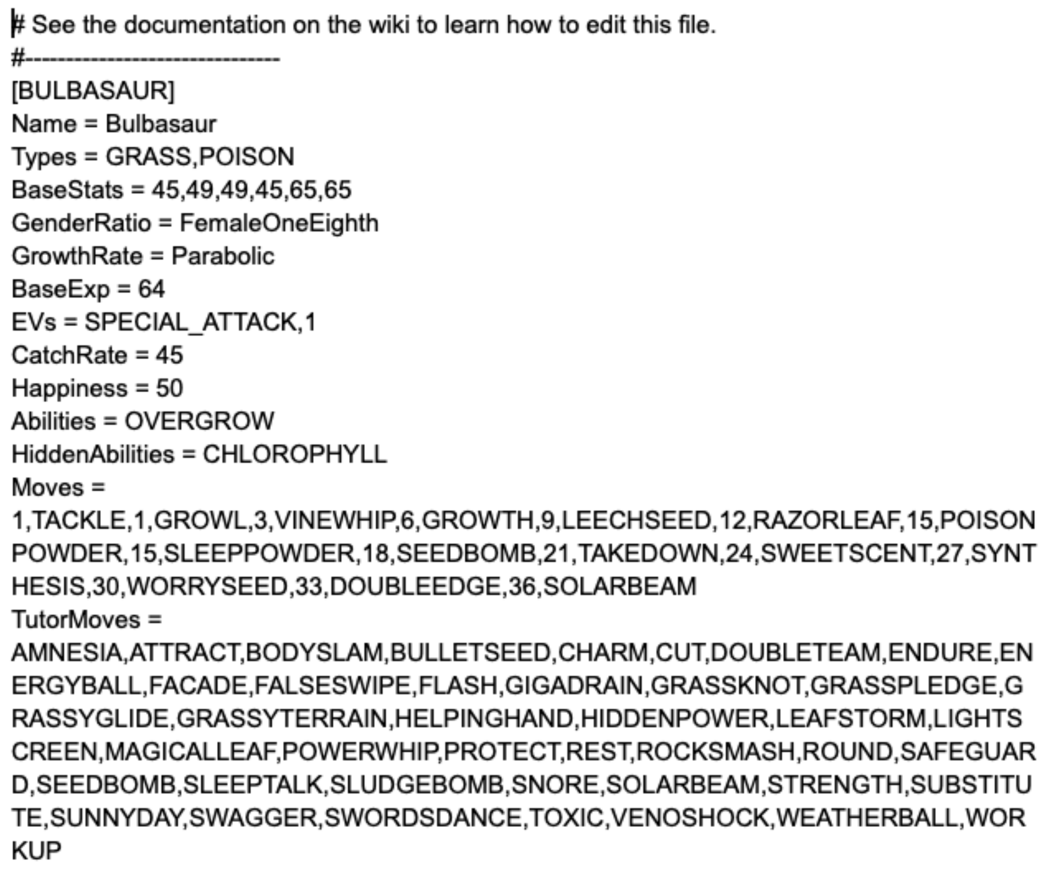
Data Summary

The data contains information about **Pokémon**, their **abilities**, **moves**, **evolution**, **types**, **forms**, and other related elements like the moves themselves and **items**. The data originated from the main series Pokémon games from generations 1-8 inclusive. Side games are not included. Generation 9 wasn't released until the due date for part 2, and therefore was not included. This data was provided by Pokémon Essentials, a fan-made tool used for making Pokémon fan games. This data belongs to Nintendo and the Pokémon Company and they do not endorse its use here. Supplementary data was sourced from Bulbapedia, a Pokémon wiki, and its data is similarly owned by Nintendo and the Pokémon Company.

This data was chosen because our group had no previous experience working with a dataset, except for one member who was familiar with this data. It was agreed upon as the best option since the data was large enough and it could easily lead to making interesting queries.

The data we gathered was presented in text files that were constructed to be easily viewed and interpreted by potential readers. This made it easier for the group members with less understanding of the game to navigate. Conversely, it necessitated parsing in order to be used in our database and in some instances, data had to be sourced piece by piece and written into insert statements.

In all, we gathered data from `6` text files which contained raw data and cross references to each other. We split that into our entity and relationship tables. Any "holes" in the data were filled manually by a team member researching the information.

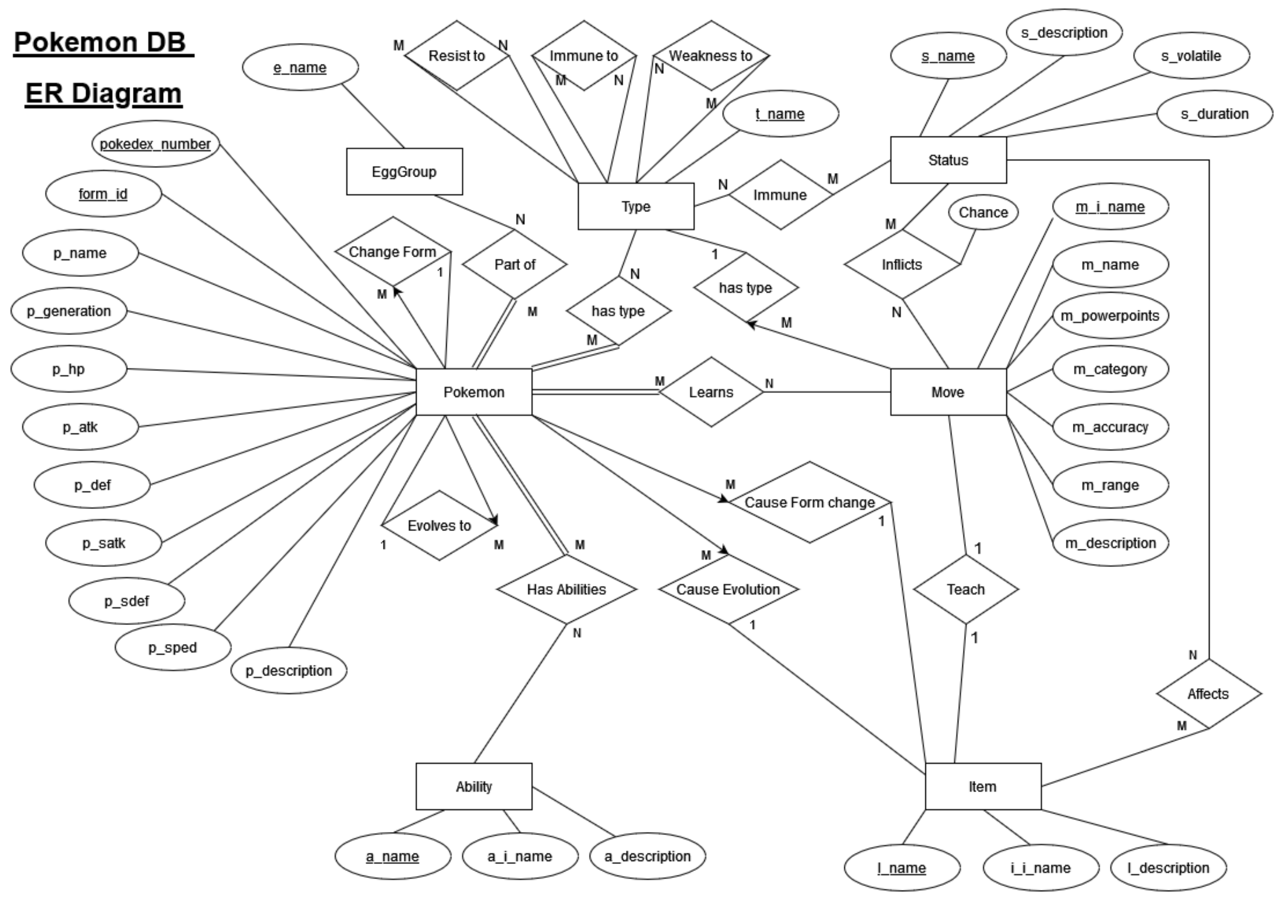


```
# See the documentation on the wiki to learn how to edit this file.
#-------------------------------
[BULBASAUR]
Name = Bulbasaur
Types = GRASS,POISON
BaseStats = 45,49,49,45,65,65
GenderRatio = FemaleOneEighth
GrowthRate = Parabolic
BaseExp = 64
EVs = SPECIAL_ATTACK,1
CatchRate = 45
Happiness = 50
Abilities = OVERGROW
HiddenAbilities = CHLOROPHYLL
Moves =
1,TACKLE,1,GROWL,3,VINEWHIP,6,GROWTH,9,LEECHSEED,12,RAZORLEAF,15,POISON
POWDER,15,SLEEPPOWDER,18,SEEDBOMB,21,TAKEDOWN,24,SWEETSCENT,27,SYNT
HESIS,30,WORRYSEED,33,DOUBLEEDGE,36,SOLARBEAM
TutorMoves =
AMNESIA,ATTRACT,BODYSLAM,BULLETSEED,CHARM,CUT,DOUBLETEAM,ENDURE,EN
ERGYBALL,FACADE,FALSESWIPE,FLASH,GIGADRAIN,GRASSKNOT,GRASSPLEDGE,G
RASSYGLIDE,GRASSYTERRAIN,HELPINGHAND,HIDDENPOWER,LEAFSTORM,LIGHTS
CREEN,MAGICALLEAF,POWERWHIP,PROTECT,REST,ROCKSMASH,ROUND,SAFEGUAR
D,SEEDBOMB,SLEEPTALK,SLUDGEBOMB,SNORE,SOLARBEAM,STRENGTH,SUBSTITU
TE,SUNNYDAY,SWAGGER,SWORDSDANCE,TOXIC,VENOSHOCK,WEATHERBALL,WOR
KUP
```

The Data Model

The initial model didn't suffer when moved into a relational model, as it was laid out with the translation in mind.

Changes to the model betweeen parts 1 and 2 were mostly minimal. One of the changes, was to add another attribute for the `Move` , `Ability` , and `Item` entities to include the internal game engine name as there were instances of duplicate in-game display names. By including the internal name, it was easier to construct our queries.



The Data Model Breakdown

Some of the tables were obvious to create. The `Pokémon` entity needed its own table as it was central to the database and has many more attributes than all other entities. **Pokémon**, **Items**, **Abilities**, and **Moves** all have different functions within the game and so it should not be in the same table as those entities or grouped with another as a single entity. For example, it does not make sense for an *item* to have an *'hp'* value.

`Ability` and `Move` entities were each their own table due to different in-game functionality. **Abilities** are considered passive powers while **Moves** are active actions for Pokémon. These two entities deserved their own table as Pokémon can share many combinations of *move* and *ability*.

The `Egg group` entity also had its own table as many Pokémon belong to certain egg groups.

`Type` is an entity that shares a relation to both `Pokémon` and `Move`. Every *move* has a *type* associated with it and every *Pokémon* can have one or more *types*. Due to this, `Type` was given its own table to reduce thousands of possible duplicate entries down to a domain consisting of (currently) only 19 distinct values.

`Status` has relationships with many other entities such as `Type`, `Move`, and `Items`. *Items* can cure many *statuses*, *types* can have immunity to many *statuses*, and *moves* can inflict many *statuses*. As `Status` was referenced multiple times, having its own table was warranted.

Data Modeling Difficulties

Part 1

One of the early hurdles we had to address was how we were going to incorporate and implement **Pokémon forms**. They were very diverse and more common than expected. *Forms* are referenced in the game by sharing a *Pokedex number* with their respective base form. At first, we considered making `Form` a weak entity but quickly realized that the *form* of a *Pokémon* can also change its *stats*, *abilities*, *moves*, etc.. We explored the idea of making the form a subclass of Pokémon. We determined this would make it challenging to model certain relations because any relations needing to reference a specific `Form` would also require a *form ID* value. Since there are many Pokémon that do not have any form changes, we would have been showing null for thousands of attribute values, which was undesirable. Our final solution was to have a Pokémon's base form referred to as form 0 and every other form be given subsequent form numbers.

> For example, relations like `Galarian Meowth` (pokedex #52, form #2) evolving into `Perrserker` (pokedex #863, form #0) was easy since the numbers could be accessed without the complexity of weak entities or subclass specifics.

In the end, we decided that every entity for `Pokémon` would have two primary keys and every relation that can be formed with a specific *form of Pokémon* would require these two values passed every time.

With `Form`, we had also initially considered giving it a `duration` attribute, but struggled to find an appropriate entity or relation to which it could be tied. `Duration` was thought to be important at the time, but the attribute value for almost every form would be either *indefinite* or *until changed back* which would make the attribute uninteresting. Ultimately, we chose to drop it from the model.

We also had to consider if a `Pokémon` should include its *form name*. We had to ask ourselves, “should Alolan Vulpix be stored as `Alolan Vulpix` or broken up like name=`Vulpix` and formName=`Alolan`?”. Since the name was not used as any key, we merged the form name with the species name to make it more apparent for queries.

Another attribute we decided not to add in our final ER model was the Pokémon’s

`Gender Ratio`. We decided it was not very interesting and it did not depend on form ID's.

Part 2

The original data files we used from our source to build our database were not in the traditional `comma separated value` format.

While writing the parser was not difficult, we quickly found out that it was not as straightforward as simply "**counting lines**". There were many instances in our data where extra attributes (lines) would show up and disrupt the parser. In some cases, this forced us to go into the data file itself and make changes in order for our parses to function properly.

Another setback, was that some data was not present in the data files and had to be manually sourced from Bulbapedia. One such example is, moves that inflict status effects. In these cases, one member opened a wiki and manually wrote the required `insert` statements.

We had initially planned to use the in-game display name as the primary key for many entities but later found edge cases where the in-game display name was duplicated or had different descriptions and functions.

> For example, there were two *abilities* called `As One` that do different things. Using the internal names `ASONEGRIMNEIGH` and `ASONECHILLINGNEIGH` was the best way to distinguish them.

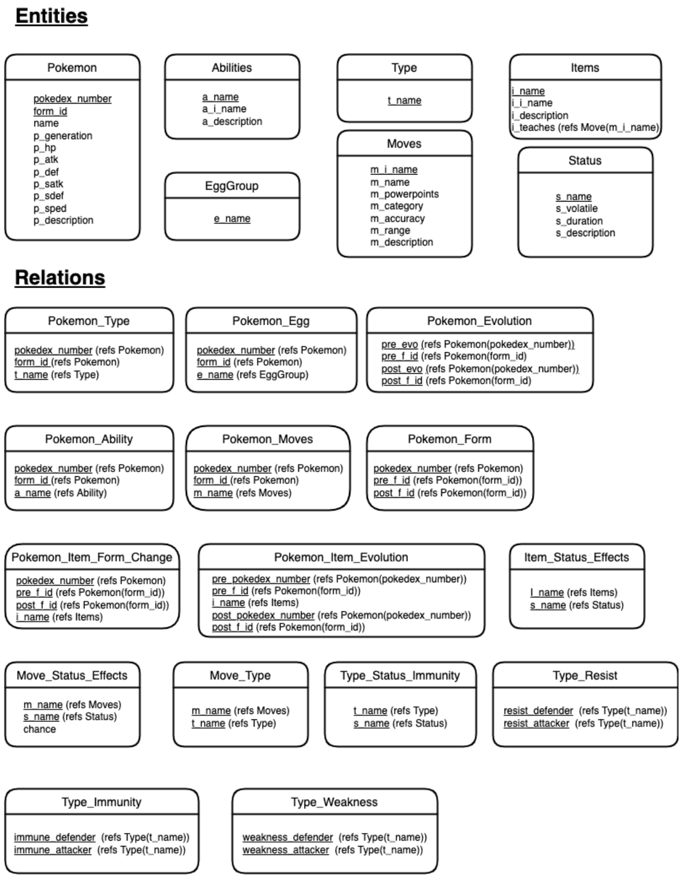
In our ER diagram, we had a relationship where an item can teach a move. We wanted to create a table for this relationship but since it would have been a **1-to-1** relationship between `Item` and `Move`. We ultimately decided to have it attached to the `Item` entity as an attribute and have the items that do not teach any moves have a null value.

At a logical level, *egg groups* do not need a *form id* since all *forms* of a particular **Pokémon** will belong to the same **egg group**. When writing our insert statements, we quickly found that we had to include both primary keys from the `Pokémon` entity in order for it to work smoothly.

While brainstorming ideas for queries, we thought about making a query that would display the `evolution line` of a **Pokémon**. The concept was to show all pre and post evolutions of a **Pokémon** as a type of lineage. We determined this type of query was not possible without multiple recursive calls or repeating calls inside a while loop. This would be further complicated with branching evolutions and forms that might result in an evolution **tree** or worse, a **cycle**! This idea was ultimately dropped in favor of a simplified version that returns all **Pokémon forms** that can evolve and what **Pokémon** they evolve into.

Database Summary

The data is more than `4,200 KB` with `72,624` combined entries into `22` tables formed from `7` entities and `15` relations.



There are no disconnected portions in our database. It is entirely connected through key dependencies. The longest seperation is between `Pokémon` and `Status` which are connected through `4` tables.

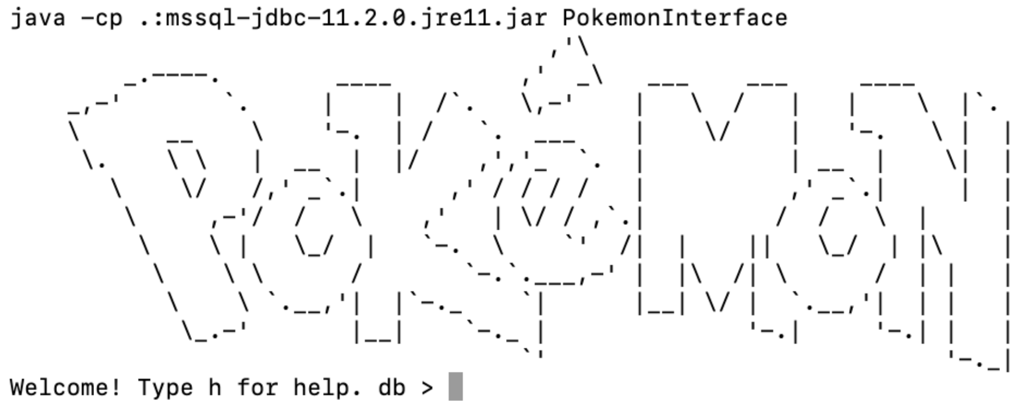
The `Pokémon` table is central to the database. It is constructed of `1,182` entries for each individual form of Pokémon and is tied to `8` relationship tables.

`Pokémon_Move` is the largest table by far, having `56,973` entries. This averages over `48` moves per **Pokémon form**. The `sql` file for inserting all the entries into this table was split into six parts to improve performance during insertion.

> **NOTE**: There is a real Pokémon called `Type:Null`. This is not an error.

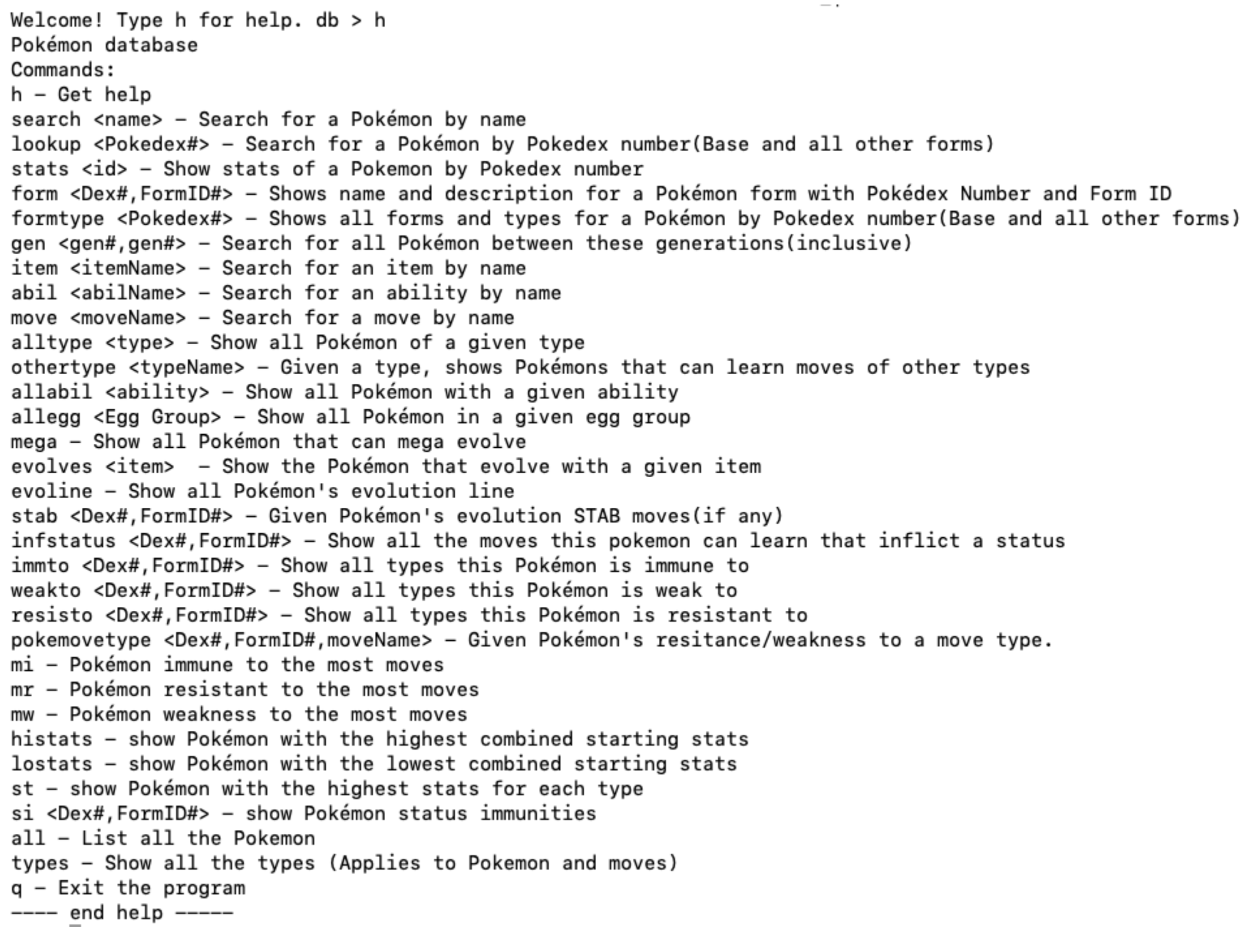
Interface Summary

We chose to use a **command line** interface for its simplicity, its security and because our group did not have much experience implementing different types of interfaces. We modeled it closely after the interface in assignment 2 Part 5 and chose to use **Java** since we had examples of functioning command line interfaces in **Java** from assignments.



```
java -cp .:mssql-jdbc-11.2.0.jre11.jar PokemonInterface
                                  ,'\
    _.----.        ____         ,'  _\   ___    ___     ____
_,-'       `.     |    |  /`.   \,-'    |   \  /   |   |    \  |`.
\      __    \    '-.  | /   `.  ___    |    \/    |   '-.   \ |  |
 \.    \ \   |  __  |  |/    ,','_  `.  |          | __  |    \|  |
   \    \/   /,' _`.|      ,' / / / /   |          ,' _`.|     |  |
    \     ,-'/  /   \    ,'   | \/ / ,`.|         /  /   \  |     |
     \    \ |   \_/  |   `-.  \    `'  /|  |    ||   \_/  | |\    |
      \    \ \      /       `-.`.___,-' |  |\  /| \      /  | |   |
       \    \ `.__,'|  |`-._    `|      |__| \/ |  `.__,'|  | |   |
        \_.-'       |__|    `-._ |              '-.|     '-.| |   |
                                `'                            '-._|
Welcome! Type h for help. db > 
```

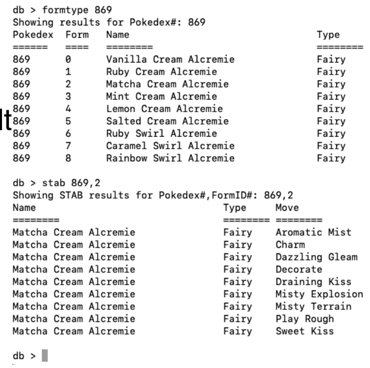
We expanded the `help menu` and added multiple queries to the interface. The menu currently has `31` different queries a user can run. These range in complexity from a simple `search` query that searches for a Pokémon by name, to complex queries that include a logical layer and return an outcome like `pokemovetype`. Due to it's relative simplicity, it would be easy to expand the interface and add more queries in the future.



```
Welcome! Type h for help. db > h
Pokémon database
Commands:
h - Get help
search <name> - Search for a Pokémon by name
lookup <Pokedex#> - Search for a Pokémon by Pokedex number(Base and all other forms)
stats <id> - Show stats of a Pokemon by Pokedex number
form <Dex#,FormID#> - Shows name and description for a Pokémon form with Pokédex Number and Form ID
formtype <Pokedex#> - Shows all forms and types for a Pokémon by Pokedex number(Base and all other forms)
gen <gen#,gen#> - Search for all Pokémon between these generations(inclusive)
item <itemName> - Search for an item by name
abil <abilName> - Search for an ability by name
move <moveName> - Search for a move by name
alltype <type> - Show all Pokémon of a given type
othertype <typeName> - Given a type, shows Pokémons that can learn moves of other types
allabil <ability> - Show all Pokémon with a given ability
allegg <Egg Group> - Show all Pokémon in a given egg group
mega - Show all Pokémon that can mega evolve
evolves <item>  - Show the Pokémon that evolve with a given item
evoline - Show all Pokémon's evolution line
stab <Dex#,FormID#> - Given Pokémon's evolution STAB moves(if any)
infstatus <Dex#,FormID#> - Show all the moves this pokemon can learn that inflict a status
immto <Dex#,FormID#> - Show all types this Pokémon is immune to
weakto <Dex#,FormID#> - Show all types this Pokémon is weak to
resisto <Dex#,FormID#> - Show all types this Pokémon is resistant to
pokemovetype <Dex#,FormID#,moveName> - Given Pokémon's resitance/weakness to a move type.
mi - Pokémon immune to the most moves
mr - Pokémon resistant to the most moves
mw - Pokémon weakness to the most moves
histats - show Pokémon with the highest combined starting stats
lostats - show Pokémon with the lowest combined starting stats
st - show Pokémon with the highest stats for each type
si <Dex#,FormID#> - show Pokémon status immunities
all - List all the Pokemon
types - Show all the types (Applies to Pokemon and moves)
q - Exit the program
---- end help -----
```

Error checking is done by verifying the validity of *user inputs* for both the *command* and any *arguments*, and by using *prepared statements* in the methods for each `query`. This makes it difficult for someone to gain unintended access to the database using *sql injection*.

Results are output in formatted `strings` and printed directly to the terminal in a tabular layout.



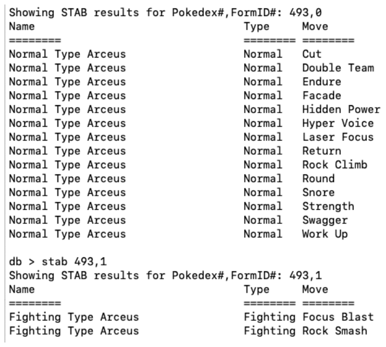
```
db > formtype 869
Showing results for Pokedex#: 869
Pokedex  Form   Name                                Type
======   ====   ========                            ========
869      0      Vanilla Cream Alcremie              Fairy
869      1      Ruby Cream Alcremie                 Fairy
869      2      Matcha Cream Alcremie               Fairy
869      3      Mint Cream Alcremie                 Fairy
869      4      Lemon Cream Alcremie                Fairy
869      5      Salted Cream Alcremie               Fairy
869      6      Ruby Swirl Alcremie                 Fairy
869      7      Caramel Swirl Alcremie              Fairy
869      8      Rainbow Swirl Alcremie              Fairy

db > stab 869,2
Showing STAB results for Pokedex#,FormID#: 869,2
Name                           Type     Move
========                       ======== ========
Matcha Cream Alcremie          Fairy    Aromatic Mist
Matcha Cream Alcremie          Fairy    Charm
Matcha Cream Alcremie          Fairy    Dazzling Gleam
Matcha Cream Alcremie          Fairy    Decorate
Matcha Cream Alcremie          Fairy    Draining Kiss
Matcha Cream Alcremie          Fairy    Misty Explosion
Matcha Cream Alcremie          Fairy    Misty Terrain
Matcha Cream Alcremie          Fairy    Play Rough
Matcha Cream Alcremie          Fairy    Sweet Kiss

db > 
```

Interesting Queries

- `stab` is one of our most interesting queries. **Stab moves** are learned by and share a *type* with the *Pokémon* learning them. There is an additional layer of complexity as **Pokémon** can have different forms and can change their `Type`.



```
Showing STAB results for Pokedex#,FormID#: 493,0
Name                                  Type     Move
========                              ======== ========
Normal Type Arceus                    Normal   Cut
Normal Type Arceus                    Normal   Double Team
Normal Type Arceus                    Normal   Endure
Normal Type Arceus                    Normal   Facade
Normal Type Arceus                    Normal   Hidden Power
Normal Type Arceus                    Normal   Hyper Voice
Normal Type Arceus                    Normal   Laser Focus
Normal Type Arceus                    Normal   Return
Normal Type Arceus                    Normal   Rock Climb
Normal Type Arceus                    Normal   Round
Normal Type Arceus                    Normal   Snore
Normal Type Arceus                    Normal   Strength
Normal Type Arceus                    Normal   Swagger
Normal Type Arceus                    Normal   Work Up

db > stab 493,1
Showing STAB results for Pokedex#,FormID#: 493,1
Name                                  Type     Move
========                              ======== ========
Fighting Type Arceus                  Fighting Focus Blast
Fighting Type Arceus                  Fighting Rock Smash
```

> This becomes clear when we compare the *stab moves* between different forms of *Pokémon #493* for instance. The list of moves that a Pokémon can learn changes depending on their form number. Counting the number of *stab moves* is similarly interesting.

- `pokeMoveType` is set up in a way to account for both of a Pokémon's types when deciding how effective a given move is. It will print whether the Pokémon is immune to, weak to, resistant to, or affected normally by a given move.

> Again, looking at Pokémon #489 again we query: `pokemovetype 493,0,fighting` and get the result: `Normal Type Arceus is weak to Fighting.`

- `othertype` is a query that will list **Pokémon** that can learn moves from a given *type* to which they do not belong.

- `st` shows the highest stats for **Pokémon** of each type by taking the sum of `hp`, `atk`, `def`, `satk`, `sdef`, and `sped` as **total base stats**. Do any **Pokémon** top both of their types? Try it for yourself. Do you notice any patterns?

The Data Model in Another Form

This data set does not explicitly require the use of a relational database but it was the most appropriate due the amount of referencing between entities. Our database is static since we will not be frequently updating values. Therefore, it would naturally be more structured. We are not likely to be deleting entries, so with a relational model, we can ensure greater data integrity compared to **Graph** or **NoSQL** databases.

As mentioned earlier, we felt the Pokémon `evolution line` query would have clearly worked better within a *graph database*. The nature of *graphs* and their ability to represent *trees*, *cycles* and *paths* would be better suited to this type of query. Displaying the `evolution line` is possible in a *relational database* but we were not satisfied with how it would be represented. Further, the complexity of this query in a *relational database* combined with the error checking required, makes it a daunting task, whereas, within a *graph database*, the query can be written in a single line.

Would this be a good tool for others?

This **Pokémon** database may work well for individuals that are familiar with Pokémon. It may not cover topics such as region but it has enough depth to be interesting. Many examples we saw in class work with having only one primary key referenced for a relation between two entities. This database can be used as an example of how to make relations using more than one primary key.

The main drawback is that it requires a bit of knowledge of the **Pokémon** universe. Certain entities can be confusing to the uninitiated, such as the difference between an ability and a move. Second, although the database size is modest by database standards, it may be too large for in-class exercises.

```
db > q
Exiting...
```