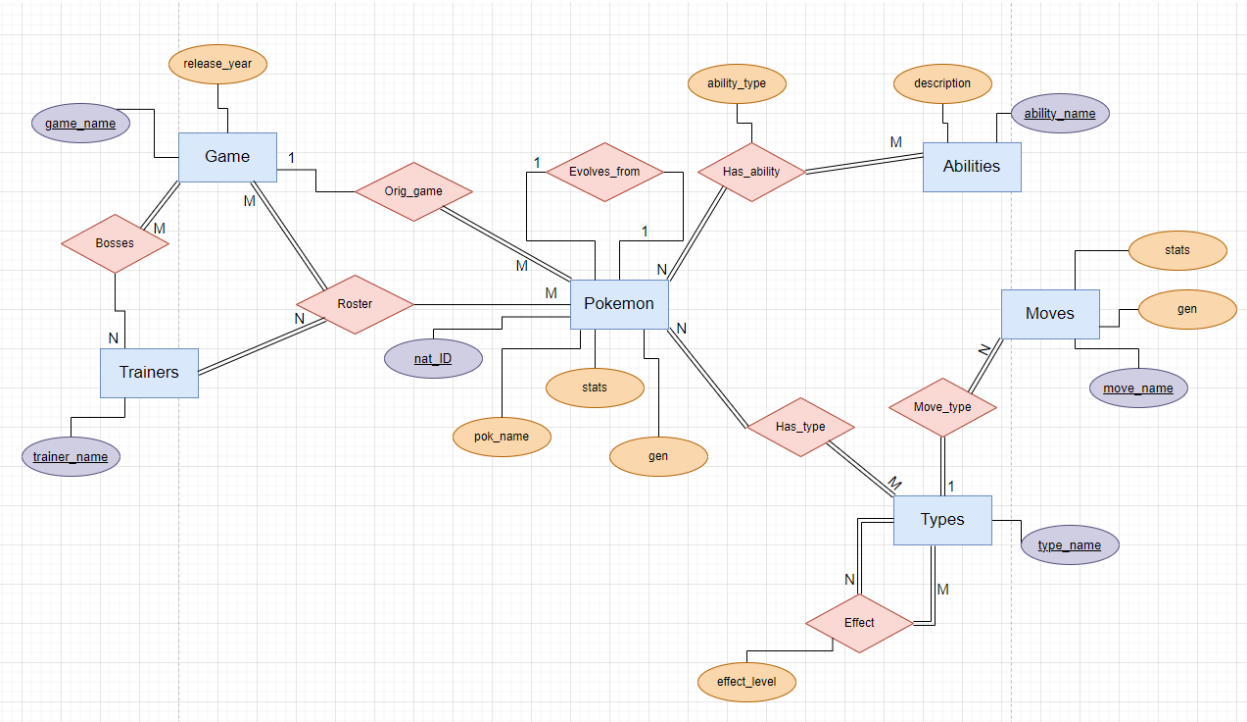
Part 3 Final Report

Members:

- Muhammad Usman Asif (7904857)
- Muhammad Ahmed Asif (7905719)
- Obaid Ali (7884305)

Summary of Data



Pokemon ER Model

Synopsis:

In the Pokemon universe (video games and the animated series), there is a device known as the “Pokedex”, which is a handheld database of all the Pokemon and their pertaining information. However, the Pokedex only includes basic information about each Pokemon, such as their typing, physical attributes, and habits. This project aims to recreate a more extensive, more analytical version of the Pokedex. We try to achieve this by taking the game mechanics into account such as the rosters of all the bosses in the games and the most viable Pokemon that can be used to counter the bosses (by using type-charts) etc.

Data Collection:

The data was gathered using both Kaggle and Pokemondb.net and consists of information regarding Pokemon, their Moves, Types, Games and the Trainers that exist in said Games. The reason we were able to combine all these sources with relative ease was that each Pokemon (in the Pokemon universe) has a unique national ID

number. There were a total of 7 tables used, with a total of roughly 5300 records. The tables used were:

1. *Pokemon Dataset.csv* : This extensive dataset, retrieved from Kaggle, relays a plethora of information about all the Pokemon that currently exist, and acted as our “master table” of sorts. We extracted the Pokemon’s name, national ID, game of origin, previous evolution, and some basic statistics (such as attack) from it. The reason we used this particular master table is because it accounts for the evolutionary line of every Pokemon by referencing a table-local ID, which made it easy to identify exactly which Pokemon evolves from another Pokemon. Furthermore, we were able to use this very table to extract information about a Pokemon’s Types and Abilities. The Abilities and Types were separated into different columns as opposed to delimiter-separated single columns, which made the data extraction a lot easier. Moreover, we filtered to only include Pokemon from the mainline games, which means that Pokemon that originated from one of the side games (such as Pokemon GO) and special forms of Pokemon (such as the “mega evolutions” that only existed for Pokemon X and Y) were excluded from our dataset. This was done to achieve consistency in our domain. Other datasets that we found struggled from similar special-case issues but the usefulness of the attributes of our chosen dataset made it comparatively much more viable.
2. *Moves.csv*: In order for us to obtain all the Moves that were introduced in all the Games, we had to manually scrape data from Pokemondb.net for each generation of Games. Once we had all the moves from each Generation, we were able to combine all the tables and add a “generation” column to it for identification purposes. The reason we accounted for all the Moves is because Moves are a cornerstone of the Games’ mechanics and are therefore included in our domain. In our Moves table, we chose to display the statistics of each Move so that an analyst might be able to figure out some of the strongest moves of a certain Type in a certain generation of Games. For example, if an analyst was to teach their Pokemon the strongest “Fire” type move in generation 1, they could do so using our database.
3. *Abilities.csv*: Each Pokemon has at least one and up to three abilities. Primary abilities are owned by every single Pokemon, whereas Secondary and Hidden are only owned by a select few. These abilities give the Pokemon an implicit

advantage in battle and therefore, all the abilities with their respective descriptions were included in our model. As mentioned before, we extracted which Pokemon has which ability in *Pokemon Database.csv* and therefore this table with all the abilities was a necessary component of our relational model and acted as a support table. Compared to other sources (such as pre-existing datasets of Abilities), the description of the Abilities suffered from String encoding issues. Therefore, we decided to scrape data from the website instead.

4. *AllGames.csv*: This simple table was handwritten using information from Pokemondb.net. We only accounted for the mainline Games in our model; excluding any remakes and special games (Pokemon GO). It was easier to make this file by hand than via programmatic extraction and therefore, we chose to take the easier route. This also acted as a support table, storing the "release years" of each game. In the earlier years of the game, the Games were released a year earlier in Japan than in North America and sometimes, they were released at different times in different regions of Japan too. Therefore, for consistency and relevance, we decided to use the North American release years of the Games.

5. *PokemonGymLeaders.csv*: In every Pokemon game, there are several Trainers that one can battle. Each Trainer has a Roster of Pokemon (up to 6) and has a unique name. For the purposes of this project, the only Trainers we accounted for are the main Bosses of the Game. These Bosses are called "Gym Leaders", "The Elite Four", and "Champion", whereby a player must defeat all of the aforementioned (in that particular order) to beat the Game. This is yet another master table we used to extract information about Trainers, Bosses and Rosters. The row entries in this particular csv file corresponded almost exactly to the row entries of the Roster table. Both the Bosses and Trainers (due to the constraints of our domain) can easily be extracted from the same information. We did not include non-Boss Trainers in this instance of the dataset but it is fully viable to do so, given a suitable dataset. However, finding an appropriate table for non-Boss Trainer in all the Games turned out to be too vast. Perhaps a future expansion of this dataset could include that information.

6. *AllTypes.csv*: There are a total of 18 Types in the Pokemon games (fire, water, electric, etcetera.). Each Pokemon must have at least one and up to two Types. This table was also hand-written due to the same reasons as mentioned earlier.

The types were keyed in by visually confirming it using Pokemondb.net. The reason we did not give each Pokemon two different Type attributes is to make our project more futureproof. In that, Pokemon could only have one type in the first generation of games but was later expanded to two. Therefore, in order to account for the possibility of a third Type being introduced in the future games, we decided to keep the Types as a separate entity in our model. Furthermore, having a Type entity separated means that we can add more Types as they are introduced. Historically, new Pokemon types have been introduced frequently.

7. *AllEffects.csv*: Each Type has an effect on another type, when attacking. For example, Water has an effect level of 2 against Fire (it is super-effective) but Fire has a 0.5 effect level against Water (it is not very effective). This table was also handwritten, due to the same reasons mentioned earlier. Although the type-charts of Pokemon are widely available, scraping the table and translating it into csv files programmatically was far more time consuming. Therefore, the type-chart available at (https://pokemondb.net/type) was used as a reference. As mentioned earlier, the Pokedex in the Pokemon games is limited and this is where our version of the Pokedex is an improvement on the original. We take the effects into account, alongside the Bosses mentioned earlier, to give the user information about countering the Bosses' Rosters. Understanding the effect on/of different Types is a major component of beating the Games and therefore, this crucial information is incorporated into our model.

Discussion

Table Breakdown Rationale:

- Pokemon(nat_id, pok_name, orig_game, evolves_from, hp, attack, defense)
  - This table is an entity in our domain so we the need for his table is a priori.
- Game(game_name, release_year)
  - This is also an important entity in our domain. This was kept into a separate entity table so that the database is open to extension where more complex attributes, and hence relations, can be later added to the model. Also this entity is used in several different relations (as can be seen from the ER diagram)

which makes this table useful. This entity participates is several many-to-many relations.

- Abilities(ability_name, description)
  - This is also an entity table. Since it participates in a many-to-many relation, this entity table is needed. Furthermore, it makes the database more extendable. It is also often useful to read the descriptions of an Ability to understand its effects which is impractical to store in several places. Therefore, this table is needed by our model and domain.
- Trainers(trainer_name)
  - This is an entity table that is referenced by multiple many-to-many relations, making it necessary. Although we have only accounted for Boss Trainers in our domain (due to lack of data), modeling this entity separately makes it extendable. As it is, our schema allows for non-Boss Trainers to be added. This is especially useful because Trainers have unique names in Games, making the model even more extendable.
- Types(type_name)
  - This is also an entity table. It is needed because it participates in several many-to-many relations. As before, having this table as a separate entity lends our model to extension. New types can be easily added to the model without many changes which reflects the flexibility requirements of the domain.
- Moves(move_name, move_type, move_power, move_acc, move_pp, orig_gen)
  - This yet another entity table. It is needed not only because it is a major component of our domain but also because it participates in a many-to-one relation where it is on the many side. This is why this table has an attribute move_type which refers to a type_name in the Types table. As before, we model this entity separately not only because of the cardinality of the relations but also because it allows our model to be extended easily. More information about Moves can be added to the table.
- Has_ability(nat_id, ability_name, ability_type)
  - This is a relational table. It is needed because it is a many-to-many relation. A Pokemon can have multiple Abilities and an Ability can be used by many Pokemon. As explained earlier, a Pokemon has 3 types of abilities. It was

debated whether to use 3 separate relations to capture this instead of one. However, since there are a lot of Abilities (at least one per Pokemon), we decided to put an attribute telling us which of the 3 Abilities it is.

- Has_Type(nat_id, type_name)
  - This, as before, is a relational table. It is needed because it is a many-to-many relationship. A Pokemon can have several Types and several Pokemon can be of the same Type. As mentioned before, instead of giving each Pokemon 2 Type attributes we are modeling this relationship in the given manner. Not only does this account for new Types being introduced, but also is extendable in case Pokemon can have more than 2 Types in the future.
- Effect(affecting_type, affected_type, effect_level)
  - This is a self referring many-to-many relationship. Each Type effects all other Types with a certain effect_level. Even if Type X has no effect on Type Y, it is recorded as effect_level = 0. This relation is needed because of its cardinality (it is a many-to-many). Half the strategy in Pokemon games is understanding the effects of different Types on each other, making this table pertinent.
- Bosses(boss_name, boss_game)
  - This is a relational table. It is needed because it is a many-to-many relationship. A Trainer can be in many Games and a Game has many Trainers. This relationship is called Bosses because we could only find data on Boss trainers and not others. This table is important because it records which Trainer was a Boss in which Game. This is crucial because looking up Bosses and their Pokemon would be useful in analysis.
- Roster(roster_entry_id, trainer_name, nat_id, game_name)
  - This is a relational table as well. It is necessary because it is a many-to-many trinary relationship. This table records the different Pokemon that Trainers have in different Games. This, in tandem with Boss and Trainer, can be very helpful to a player of the Game. Understanding the Roster of a Trainer, in combination with Typing data, can be used to perform analysis on Trainers. It connects the Trainer entity to Pokemon and Games, making it very versatile in our domain.

Model Design Decision-Making:

- In the earlier stages of the project, we operated under the assumption that each Pokemon would have a Primary Type and Secondary Type attribute. However, as mentioned earlier, the secondary Type was introduced in the second generation of Pokemon games. Furthermore, in Generation 6, a new Type (Fairy) was introduced and the whole game went through major changes in their mechanics. Every time a new Type is introduced, it has it's own unique effect levels against other types, which changes information about ALL other Types. Giving just the attribute of Primary and Secondary Type to a Pokemon would have made our model more limited, and hence we decided to make an entity out of Types.
- Most of the Pokemon can evolve into a different Pokemon, depending on certain requirements that a player must fulfill. Although giving each Pokemon an attribute of "nextEvolution" would have worked, we instead decided to make a circular relationship of "Evolves_From" between Pokemon and itself. The reason we decided to retrieve information about what Pokemon a given Pokemon evolves *from* and not evolves *into* is so that we could work backwards from the final evolution when retrieving the evolutionary line of a Pokemon. If we had used the latter, there would be some problems. For example, the Pokemon Eevee can evolve into several different Pokemon. However, one Pokemon can only ever evolve from exactly one Pokemon which constrains our model in a realistic way.
- Every Game, as mentioned earlier, has Trainers that have their own Rosters. The Trainers can only be identified uniquely by their names. However, a Trainer can reappear in a different Game with a different (or same) Roster. This is an inherently complex relationship to capture, and therefore we made a trinary relationship between Roster, Game and Pokemon. Both the cardinality ratios and the participation constraints were carefully chosen, given the mechanics of the games. Furthermore, a Trainer can have several of the same Pokemon in their Roster in one or more Games. Hence, for each entry in the Roster, we generate an auto-incrementing key. This way, there are no duplicates and the table can be utilized with some clever joins.

Model-Database Integration:

- Although most of the data was fit with relative ease into our relational database, there were some exceptions.
    - As per our current design, each Pokemon has an “evolves_from” attribute, which references an ID of the current Pokemon’s previous evolution. Pokemon, in general, have their IDs in a sequential order. For example, Venusaur (national ID = 3) evolves from Ivysaur (national ID = 2) who evolves from Bulbasaur (national_ID = 1). Hence, whenever a new Pokemon was added, their previous evolution was assumed present. Furthermore, usually, newer evolutions of existing Pokemon are introduced in later Games. However, in some rare cases, a Pokemon’s *previous* evolution was introduced *after* the original was released (Baby Evolutions), which did not fit well for our model (since our inserts relied on order). Therefore, although we included all the Baby Evolution Pokemon, we were not able to connect them to the evolutionary lines of their next forms.
    - There was also some data in the *Pokemon Database.csv* that we could not include in our database. When we initially designed our model, we assumed that every Pokemon originated from one of the mainline Games, and therefore gave it full participation. However, there were some edge cases to this that we could not account for. For example, Melmetal (Pokemon) was introduced in Pokemon GO (not one of the mainline Games). Therefore, we had to exclude a few Pokemon and only included the ones that originated from one of the mainline games to better fit our model (and follow its participation constraints).

Model Decision Retrospect:

- There were some decisions that introduced some technical debt to our model.
    - Initially, we had planned to include a table for the relation called “can_learn” which was supposed to reflect all the moves that can be learned by a certain Pokemon. However, not only were we not able to find a reliable dataset which had this information, we were also not able to scrape this data. For most reliable websites (such as Pokemondb.net), each Pokemon’s moves were on a separate webpage, which would have required far too much work and therefore, this had to be omitted from our final release. Although this takes away some

complexity from our model, Moves and Types are still connected. Not analyzing this bottleneck turned out to reduce our model. However, in spite of this, we were able to keep our database connected.

- The Trainer, Bosses and Roster tables may have redundant data but because of the way our model was set up, making these tables was inevitable. However, perhaps if this part of the model was designed differently, we would be able to make it more expansive and less redundant than it is right now.
- Since the datasets had a lot of limitations, perhaps we could have found cleaner datasets that did not require manual entries or had many special cases.

Potential Modelling Alternatives

- As it has come up before, the main problem of this model lies in Trainer, Bosses and Roster. These could be modelled in a different way. We could replace with these tables:
  - Trainers(trainer_id, trainer_name)
  - Bosses(trainer_id, game_name, position) where position represents if they are a gym leader, elite 4 or a champion.
  - Roster(roster_id, game_name, pok_1, pok_2, pok_3, pok_4, pok_5, pok_6) where pok_i is a national ID. As of right now, a trainer can only have 6 Pokemon.
  - Has_roster(trainer_id, roster_id)
- However, given the work completed and the fact that our current model is more flexible than the above model, we would not choose this model. Although the above model mimics the domain more closely, if a Trainer could have more than 6 Pokemon, it would require changes to our schema. This makes the above model rigid and inflexible. Any changes to the domain would require a lot of refactoring.

Interesting Queries

All Tables

One of the things we provided to the user is access to all individual tables in our relational model.

New Pokemon

We also provided the user with the ability to check how many new Pokemon per Type were introduced in a selected Game. For example, if a user selects the game 'red', they will be given a table which lists all Types and the number of new Pokemon of that Type introduced in that Game. Each Type is in included in the search and it is ordered by Type.type_name. This query is interesting because it allows analysis of aggregate Game and Type data. An analyst may want to observe such overall trends in order to strategize if they are a player of the Game. Here is the SQL query:

```
SELECT Has_Type.type_name, count(*) as numIntroduced
FROM Has_Type
LEFT JOIN Pokemon
    ON Pokemon.nat_id = Has_Type.nat_id
WHERE Pokemon.orig_game = <chosen game_name here>
GROUP BY Has_Type.type_name
ORDER BY numIntroduced DESC;
```

Pokemon Evolutions

Another query we provided was the table all the evolutionary lines of all the Pokemon were given to a user. Since a Pokemon can only evolve 0, 1 or 2 times, we provided a table with 3 columns, listing the names of Pokemon in each column. The first column is the base Pokemon, the second column is its next evolution and the third column is the next evolution of the second evolution. This is done for each Pokemon. This is an extremely useful table since evolutionary lines are a staple of the domain. Knowing which Pokemon evolves into what is crucial for analysis and for players of the Games. Furthermore, as it is, our interface allows for search by name of Pokemon in the first column. This table would be very tedious for a person to decipher from the Pokemon table and its evolves_from attribute. Here is the SQL:

```
SELECT p1.pok_name, p2.pok_name, p3.pok_name

    FROM Pokemon p1

    LEFT JOIN Pokemon p2

    ON p2.evolves_from = p1.nat_id

    LEFT JOIN Pokemon p3

    ON p3.evolves_from = p2.nat_id

    ORDER BY p1.pok_name
```

Pokemon Abilities

Similar to the previous query, this one is also an aggregate table. This query provides the primary, secondary and hidden Abilities of each Pokemon. It provides 3 columns; one for Pokemon name, one for its Primary Ability, one for its Secondary Ability and one for its Hidden Ability. This is useful because, as before, these abilities are an important part of the domain. Therefore, having all the abilities of each Pokemon compiled into a single table is useful for analysis and for players looking to find Ability information on certain Pokemon. As before, the name of the Pokemon (first column) is searchable in our given interface. Moreover, extracting information about each Pokemon's all 3 abilities from the Has_ability table would be tedious, making this table very convenient. Here is the SQL:

```
WITH primaries AS (
  SELECT Pokemon.nat_id, Has_ability.ability_name
  FROM Pokemon
  JOIN Has_ability
    ON Pokemon.nat_id = Has_ability.nat_id
  WHERE Has_ability.ability_type = 'primary'
),
secondaries AS (
  SELECT Pokemon.nat_id, Has_ability.ability_name
  FROM Pokemon
  JOIN Has_ability
    ON Pokemon.nat_id = Has_ability.nat_id
  WHERE Has_ability.ability_type = 'secondary'
),
hiddens AS (
  SELECT Pokemon.nat_id, Has_ability.ability_name
  FROM Pokemon
```

```
  JOIN Has_ability
    ON Pokemon.nat_id = Has_ability.nat_id
  WHERE Has_ability.ability_type = 'hidden'
)
SELECT Pokemon.nat_id, Pokemon.pok_name,
primaries.ability_name, secondaries.ability_name,
hiddens.ability_name
FROM Pokemon
LEFT JOIN primaries
  ON Pokemon.nat_id = primaries.nat_id
LEFT JOIN secondaries
  ON Pokemon.nat_id = secondaries.nat_id
LEFT JOIN hiddens
  ON Pokemon.nat_id = hiddens.nat_id
ORDER BY Pokemon.nat_id;
```

Type Search

We also allow users to search the Pokemon database based on a given Game. The user can specify the Game they want to search, two Types to filter on and the operation (and/or) between the Types. For example, if users choose Game as 'red', primary Type as 'fire', the operation as 'AND' and the secondary Type as 'flying', this will provide a list of Pokemon that are from the Game 'red', are 'fire' Type and 'flying' Type. The user can elect to ignore a secondary type, in which case Pokemon with that Type will be shown. If the user selects Game as 'red', primary Type as 'fire', operation as 'OR' and secondary Type as 'water', all Pokemon from 'red' with 'fire' or 'water' type will return. This tool is useful because searching for Pokemon with particular Types in a Game is often something players of the Games value. As mentioned earlier, Types are a major mechanism of the Games. Therefore, searching based on Types like this is crucial for analysis. Here is the SQL:

```
- SELECT Pokemon.pok_name, Pokemon.hp, Pokemon.attack, Pokemon.defense
FROM Pokemon
JOIN Has_Type
  ON Pokemon.nat_id = Has_Type.nat_id
WHERE Has_Type.type_name = <chosen primary_type here>

    AND Pokemon.orig_game = <chosen game_name here>
    INTERSECT --or union (based on operation selected)
    SELECT Pokemon.pok_name, Pokemon.hp, Pokemon.attack, Pokemon.defense
    FROM Pokemon
    JOIN Has_Type
      ON Pokemon.nat_id = Has_Type.nat_id
    WHERE Has_Type.type_name = <chosen secondary_type here>
```

```
    AND Pokemon.orig_game = <chosen game_name here>
```

Boss Favorites

We also provided aggregate data for Bosses in Games. In this query, the user enters a Game and we provide a table with all the Boss names in that Game, their most favored Type in that Game and the number of Pokemon they have of that Type in that Game. Most favored Type is the Type which is most commonly present in the Roster of that Boss in that Game. This data is extremely useful to players since, as mentioned earlier, Type information is a major part of strategizing in the Games. Also, to beat the Game, players need to beat all Bosses in that Game. Therefore, having this table is very pertinent and pragmatic for players. If a player knows the favoured Type of a Boss, they can beat them much more easily. Furthermore, this query aggregates data in a way that would be extremely hard for a person to do from the given tables, making it very useful for analysis. Here is the SQL:

```
SELECT b.boss_name, ht.type_name as favType, count(*) as numPokemon
FROM Bosses b
JOIN Roster r
    ON b.boss_name = r.trainer_name
    AND b.boss_game = r.game_name
JOIN Has_Type ht ON r.nat_id = ht.nat_id
WHERE b.boss_game = <chosen game_name here>
AND ht.type_name IN (
    SELECT TOP(1) Has_Type.type_name
    FROM Roster
    JOIN Has_Type
      ON Roster.nat_id = Has_Type.nat_id
    WHERE Roster.trainer_name = r.trainer_name
    GROUP BY type_name
    ORDER BY COUNT(*) DESC
)
GROUP BY b.boss_name, ht.type_name;
```

Relational Database Required?

The domain of the Pokemon universe consists of a lot of relations. Pokemon evolve to and from each other, can have several Types, have several abilities and can learn many moves. Furthermore, Moves have Types (maybe there will be moves with several Types in the future) and Trainers can have several Pokemon. Due to the highly relational nature of the domain, a graph dataset would be a good fit. In a graph database, relationship would not only be simplified but also would be more extendible which is a common requirement of the domain. For example, adding new Types, Pokemon having 3 Types and Pokemon having more than 2 evolutions could be easy to implement. Not only that but a graph database would allow for instances of Pokemon instead of one entry per unique Pokemon. This way, we can model the domain more closely. For example, we could have a Pikachu (Pokemon) named ‘bob’ belong to Trainer ‘Ash’, and a Pikachu ‘rob’ could belong Trainer ‘Misty’. As for the queries, they may be a bit more complex in a graph database but would be able to capture much more information. The types of queries possible would be more if the domain was modelled in a graph database context. However, for the scope of this project and the data available, a relational model is perhaps more constrained and suitable.

Database as Teaching Tool:

This could be a good teaching tool for this course because the domain knowledge is very common. However, having some domain knowledge is necessary for understanding the model which may be an issue. The flexibility and extendibility of this database may prove useful as well. Moreover, since national ID’s for Pokemon are standardized and other entities have similar unique identifiers (usually names), it would be very easy to mold the model to student needs. The biggest problem to solve that comes to mind is accounting for special cases of Pokemon and accounting for wild and trained Pokemon.

Member Contributions

- Muhammad Ahmed Asif (7905719):
    - Part 1: Found the gym leaders dataset. Took meeting notes and went to instructor for feedback on progress. Designed initial ER diagram and worked

with teammates to normalize our model. Also wrote the report of the more complex relations and their respective cardinality and participation requirements.

- Part 2: I created the ‘allGames’ table by hand. I also implemented the string validator for preventing SQL injection. Furthermore, I also wrote the script that was responsible for extracting information from CSV files and writing them into an SQL file which populates our database. However, it is worth noting that I had help from my team catching errors and special cases. I also helped in the decision making for the front-end of the project. In that, I helped decide which queries to choose and how to display them. I designed some of the pages from All Tables and some from our other interesting queries. I also personally came up with 2 interesting queries that we implemented in our project. Moreover, I helped with version control issues and branching strategies that we used to work on the project together.
- Part 3: Wrote the part about interesting queries. Also answered the questions about using this project as a teaching source and whether this project requires a relational database. Lastly, I helped the team write the rest of the report.

- Obaid Ali (7884305):
  - Part 1: Found nine datasets of moves for each generation. Also constructed the ER/EER diagram with its entities and relationships. I also worked with my teammates to normalize the database. I also wrote two entities and three relationships alongside their participation constraints and cardinality.
  - Part 2: I created the 'allTypes' database by hand, combining data from nine moves datasets into a unified 'moves’ table with a generation column. I also oversaw the development of the project's backend and front end, providing logic for seamless database interaction and table presentation. I took the initiative to train peers on Python, HTML, and crucial tools like as Flask and Bootstrap, therefore improving our collaborative efforts and increasing overall project productivity.
  - Part 3: Helped take notes about the dataset as we worked on the project. Contributed to answering a few questions
- Muhammad Usman Asif (7904857):

- Part 1: Found the *Pokemon Database.csv* database and wrote the brief summary section of the proposal. Also wrote two of the Entities and three of the Relationships (with all their participation constraints and cardinality). Last but not least, collaborated with my teammates to come up with the ER diagram, normalizing our dataset, and writing the final relational model.
- Part 2: Designed the webpages that view all our tables in their complete form (including the logic code in Python). I also wrote 2 of the 6 unique queries used in this project, alongside the pages that display the result of the queries. Hand-wrote the allEffects table by referencing from Pokemondb.net. Also did the logic behind displaying the two aforementioned queries. Lastly, I also helped my groupmates debug errors pertaining to implementing the interface.
- Part 3: Did the summary component of the dataset, answered a few of the discussion questions, and did the references.

References:

Dew Gaming, 2022, “Pokemon Database”, *Kaggle*, viewed 4 October 2023, available at https://www.kaggle.com/datasets/mrdew25/pokemon-database

Montano, H. M. L., 2023, “Pokemon Gen 1-9 Gym Leaders + Elite Four”, *Kaggle*, viewed 4 October 2023, available at https://www.kaggle.com/datasets/maxiboo/pokemon-gen-1-9-gym-leaders-elite-four

Pokemon Database, 2023, “Pokemon Black & White”, *pokemondb.net*, viewed 4 October 2023, available at https://pokemondb.net/black-white

Pokemon Database, 2023, “Pokemon Crystal”, *pokemondb.net*, viewed 4 October 2023, available at https://pokemondb.net/crystal

Pokemon Database, 2023, “Pokemon Diamond & Pearl”, *pokemondb.net*, viewed 4 October 2023, available at https://pokemondb.net/diamond-pearl

Pokemon Database, 2023, “Pokemon Emerald”, *pokemondb.net*, viewed 4 October 2023, available at https://pokemondb.net/emerald

Pokemon Database, 2023, “Pokemon Gold & Silver”, *pokemondb.net*, viewed 4 October 2023, available at https://pokemondb.net/gold-silver

Pokemon Database, 2023, “Pokemon moves from Generation 1”, *pokemondb.net*, viewed 4 October 2023, available at https://pokemondb.net/move/generation/1

Pokemon Database, 2023, “Pokemon moves from Generation 2”, *pokemondb.net*, viewed 4 October 2023, available at https://pokemondb.net/move/generation/2

Pokemon Database, 2023, “Pokemon moves from Generation 3”, *pokemondb.net*, viewed 4 October 2023, available at https://pokemondb.net/move/generation/3

Pokemon Database, 2023, “Pokemon moves from Generation 4”, *pokemondb.net*, viewed 4 October 2023, available at https://pokemondb.net/move/generation/4

Pokemon Database, 2023, “Pokemon moves from Generation 5”, *pokemondb.net*, viewed 4 October 2023, available at https://pokemondb.net/move/generation/5

Pokemon Database, 2023, “Pokemon moves from Generation 6”, *pokemondb.net*, viewed 4 October 2023, available at https://pokemondb.net/move/generation/6

Pokemon Database, 2023, “Pokemon moves from Generation 7”, *pokemondb.net*, viewed 4 October 2023, available at https://pokemondb.net/move/generation/7

Pokemon Database, 2023, “Pokemon moves from Generation 8”, *pokemondb.net*, viewed 4 October 2023, available at https://pokemondb.net/move/generation/8

Pokemon Database, 2023, “Pokemon moves from Generation 9”, *pokemondb.net*, viewed 4 October 2023, available at https://pokemondb.net/move/generation/9

Pokemon Database, 2023, “Pokemon Platinum”, *pokemondb.net*, viewed 4 October 2023, available at https://pokemondb.net/platinum

Pokemon Database, 2023, “Pokemon Red & Blue”, *pokemondb.net*, viewed 4 October 2023, available at https://pokemondb.net/red-blue

Pokemon Database, 2023, “Pokemon Ruby & Sapphire”, *pokemondb.net*, viewed 4 October 2023, available at https://pokemondb.net/ruby-sapphire

Pokemon Database, 2023, “Pokemon Scarlet & Violet”, *pokemondb.net*, viewed 4 October 2023, available at https://pokemondb.net/scarlet-violet

Pokemon Database, 2023, “Pokemon Sun & Moon”, *pokemondb.net*, viewed 4 October 2023, available at https://pokemondb.net/sun-moon

Pokemon Database, 2023, “Pokemon Sword & Shield”, *pokemondb.net*, viewed 4 October 2023, available at https://pokemondb.net/sword-shield

Pokemon Database, 2023, “Pokemon types & type chart”, *pokemondb.net*, viewed 4 October 2023, available at https://pokemondb.net/type

Pokemon Database, 2023, “Pokemon X & Y”, *pokemondb.net*, viewed 4 October 2023, available at https://pokemondb.net/x-y

Pokemon Database, 2023, “Pokemon Yellow”, *pokemondb.net*, viewed 4 October 2023, available at https://pokemondb.net/yellow