Pokemon Information Database

By: Top CS Dawgs
(Cesar H., Star W., Vitaliy K.)

Last Updated: 12/11/2023

Table of Contents

Database Proposal and Description

Our database will be about Pokemon, specifically their evolutions and background information. This database will cater to both new and old Pokemon fans as it will hold Pokemon from different generations. Pokemon fans will be able to learn more about any Pokemon they want. Only information from either the Pokemon Company or Nintendo will be used to design our database around. Fan-made pokemon or unofficial information regarding a Pokemon will not be considered accurate and therefore not be included in the database.The following data our database will hold are numerous.

First we have the Pokemon themselves which will contain information about their name, legendary or mythical status, gender ratio, and what evolution stage they are in. Strongly tied into the Pokemon is the typing system. A Pokemon can have 1 or 2 types associated with it. Each type has other types that they are weak, strong, or ineffective against.

A looser connection to Pokemon is provided in the form of game mechanics. These vary wildly from Pokemon to Pokemon so they include quite the differing amount of data. It includes information on evolution items needed, level it evolves at, time of day it evolves at, and/or gimmick that causes it to evolve.

A Game entity is provided to detail the environment that a Pokemon was introduced in, including attributes such as the region it debuted in and from which game. Lastly, we have the Pokedex which catalogs all the official known Pokemon in the Pokemon Universe. We will not go too much into detail such as include informational passages about the Pokemon but we will include its national dex number and the regional dex number from its first appearance in the Pokemon series.

Learned moves are also another entity that is stored in our database. THese moves themselves have various attributes such as the move's name, damage type, PP usage, and power. Each move will also have a Type associated with it.

The central and most important piece of data in our database will be the Pokemon. The game itself will be considered its own separate entity as will the Pokedex and Pokemon typings. Most of the data types present in our database will take the form of ints and strings.

The limitations of this database is that since each Pokemon game has different logic and rules regarding ***how*** pokemon are evolved (such as different levels, time of day, item needed). This database will attempt to provide as much information into these evolution methods but its main purpose will be to serve as a general knowledge base for looking up Pokemon evolutions and their information.

Another limitation is the sheer volume of Pokemon that exist. We are unsure of how many Pokemon we will be able to insert into the database that would be feasible enough for this class. Yet another limitation that we imposed on the database is that certain pokemon information is excluded. For example, pokemon stats and special abilities aren't listed here so queries such as "Filter by 'Fire' type pokemon with the ability 'sturdy' " would not be able to provide a valid result.

Queries Examples

Queries will provide a convenient method for users to learn more about a certain pokemon. Common queries that our database will be able to answer include:

- Filter by “Legendary” Pokemon in “Hoenn” region and “Johto” region.
- Filter by “Fire” and “Fighting” type Pokemon that have a “stage 3” evolution.
- Filter by all non-legendary pokemon that cannot evolve.
- Filter by pokemon that can be only evolved using an item.
- Filter by pokemon that have multiple evolution forms.
- Filter by “Generation 3” Pokemon.

Project Timeline

In order to ensure that group efforts are being made to complete tasks in a timely manner, Table 1 illustrated below outlines a work schedule that breaks down due dates in a more manageable format.

Official Canvas Due Dates are included and colored in red to give the group an indication of when the smaller deliverables are due overall for the complete iteration

Table 1: Pokemon Information Database Work Schedule

Deliverable	Description	Date
Team ByLaws	Bylaws doc completed and filled Signed and reviewed by all members	10/13/2023
Project Proposal	Describe the Database Project Provide examples and usage of the database Outline the timeline of Project deliverables	10/14/2023
Project Iteration 0 Due		10/15/2023
Updated Proposal	Incorporate feedback into Proposal V2	10/17/2023
ER Diagram	Create ER Diagram based on Database Proposal and Description	10/19/2023

Final Project: Pokemon Informational Database

Relational Data Model	Create RM based on ER Diagram	10/21/2023
Project Iteration 1 Due		10/22/2023
Create Progress Presentation	Create intro slide Describe project, goals, and purpose State tools used for Project *Total 3 min timed presentation*	10/30/2023
Tooling and Big Picture Due		11/6/2023
Updated Proposal	Incorporate feedback into Proposal V3	11/8/2023
Add Tooling Assessment	Include tools used for Database (based on Presentation): ● Database software ● UI ● Database hosting ● Other tools	11/10/2023
Project Iteration 2 Due		11/11/2023
Updated Iteration	Incorporate feedback into Iteration V4	11/14/2023
Finish Database	Create database based on previous iterations Include SQL statements using standard naming	11/18/2023
Populate Database	Create file to regenerate and populate DB on use	11/20/2023
Create report	List name of tables used in Report Explain methodology used to create data in report	11/24/2023
Project Iteration 3 Due		11/26/2023

Update Database and Files as Needed	Bundle all SQL code in sql/text files and UI code	11/28/2023
Present Short Demo	Presentation and Demo of Database ● 5 min intro ● Scenarios and Reasoning ● Show and Tell ● Challenges	TBD (Final Exam Date)
Update Iteration	Incorporate feedback into Iteration V5	12/04/2023
Submit Final Report (Final Iteration)	Discuss: ● Design and Results ● Normalization ● SQL Statements ● Testing	12/10/2023
Project Demo Due		12/11/2023
Project Iteration 4 Due		12/13/2023

Roles and Responsibilities

As of the writing of this document, all members will equally split the work for each ‘sprint’ amongst one another. For example, the time period between 10/15 and 10/22 is considered a sprint. Each member of the team will assign themselves to complete certain tasks detailed in the sprint. In this example, Cesar could be in charge of updating the project proposal, Vitaliy can work on completing the ER model, and Star would complete the Relational Data Model. If a member completes their section with time to spare and it is deemed to be of good quality, another member may request them to help out on their section. In the case of major setbacks, such as a member becoming sick or unable to complete their assigned work, the remaining members will divide the 3rd member's work amongst themselves.

Diagram Info and Assumptions

For those who are not knowledgeable in Pokemon, there are some game mechanics and terms that have to be explained in order to understand why the diagram is the way it is. To start off, there are many different pokemon that exist. Here, we only concern ourselves with each Pokemon species as a whole, not individual Pokemon. Pokemon are always introduced in a new game which is tied to a certain region and generation of the Pokemon series. In each region, they have a regional dex number assigned to them and a permanent National Pokedex number given. These Pokemon are able to evolve into other Pokemon using a game mechanic. This mechanic can be a combination of reaching a certain level, using an item, leveling up at a certain time, or completing a specific task. Pokemon are able to have either one or two types assigned to them. They also learn a variety of moves as they level up, with each move having its own typing and effectiveness.

Here's also some terminology to understand our diagram better:

Pokemon:

- Pname: A pokemon's official name
- Evo_Stage: The evolution stage of the current pokemon
- Rarity: The type of pokemon it is classified as (Legendary, Mythical, or Basic)
- Gender_Ratio: The probability of a Pokemon being male. The remaining percentage indicates how many are female
- Previous_Evo: The previous evolution of the current Pokemon

Pokedex:

- Nation_Num: ID of a pokemon based on the national pokedex
- Region_Num: ID of a pokemon based on what was given to them in their first regional dex

Game:

- Gname: Name of a Pokemon game
- Region: Location the game takes place
- Generation: Generation the game was in
- Release_Date: When the game was released

Moves:

- Move_Name: Name of the Pokemon move
- Damage_Type: What type of move it is (Special, Physical, or Status)
- PP: Power points of a move (How many times it can be used in battle
- Power: Base damage caused by the move

Type:

- ❖ Type 1 “Elements” that a pokemon is classified to be (Can be single type or dual type which is why it is many to many)
- ❖ Effective: All the “Elements” that a type does 2x damage to
- ❖ Weak: All the “Elements” that a type does 0.5x damage to
- ❖ Not_Effective: All the “Elements” that a type does no damage to

Mechanic:

- ❖ Next_Evo: The Evolution that results from evolving that is related to this specific evolution method.
- ❖ Level: The level at which the current pokemon evolves into their next stage
- ❖ Item: An item that allows a pokemon to evolve into another stage
- ❖ Time: Time of day which evolves a pokemon
- ❖ Gimmick: Any other evolution method that causes a pokemon to evolve (ie walking 200 steps, doing 5 spins, etc.)

That said, here are some assumptions we have made:

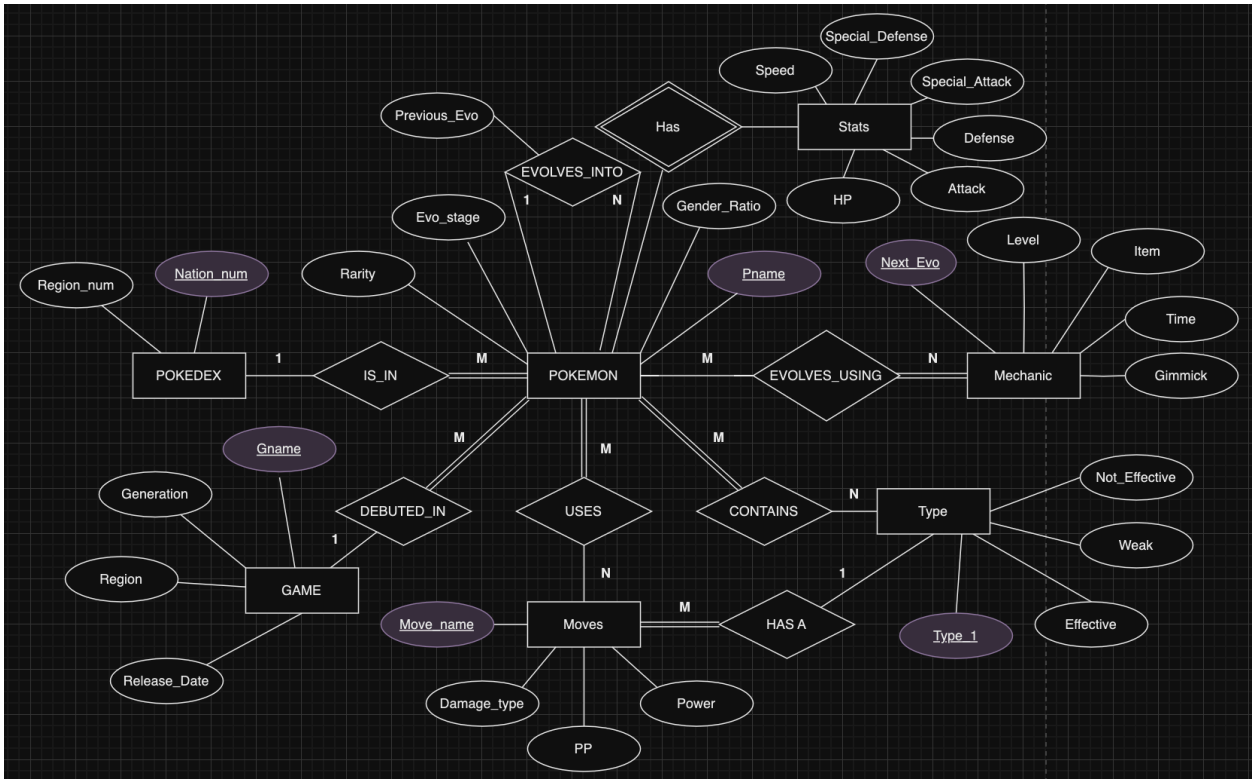
- Pokemon may or may not have an evolution (They might only have 1 stage in their evolution or be the final stage in their evolution line)
- A Pokemon must be introduced in a main game and their regional number corresponds to where they debuted
    - As such, we are only considering the National Dex as the only main pokedex, the regional number is included for extra information
- Other forms of Pokemon, such as the Pikachu’s with different hats or regional forms, are not considered as separate Pokemon (In order for a Pokemon to be considered distinct, they need to have their own National Dex number)
- Mega Evolutions, Gigantamax forms, and other special evolutions or form changes do not count as a separate pokemon as they do not have their own National Dex Number and in most cases are only temporary during a battle
    - They also do not count as the next evolution of a pokemon as they do not have their own National dex Number. They would not be able to properly fit into the “Evolves Into” recursive relationship
- There will never be any pokemon with the same name yet different dex numbers
- Terastallizing to change type does not count towards what a pokemon’s original typing is
    - Nor do type changes such as certain items (like the Arceus plates)
- Type 2 can be null as some pokemon are single typing
    - In addition, there can’t be a duplicate type in both type 1 and 2 (ie not steel-steel type pokemon, in that case it would just be steel)
- Only official pokemon up to Pokemon Scarlet and Violet’s Teal Mask DLC are considered for this database (as of this writing, Teal Mask DLC brings the highest National Dex Number to 1018)
- The shortest possible name for a pokemon game name is Red (ie Pokemon Red)

- Shortest possible region name is 5 chars (ie Johto) and longest is 6 (ie Sinnoh)
- Only Pokemon up to the latest generation (generation 9) are considered for this database

- Exclusive versions of the mainline games have minimal differences between the two
- It is possible for Pokemon to not have any type advantages or disadvantages.
- Status moves that do no damage (such as growth) have a Power of 0
- Other regions that take place in the same location (ie Sinnoh being the present-day version of Hisui) are considered to be one region
    - Hisui would be grouped under Sinnoh
- DLC regions that take place outside of a game's main region (ie Kitakami in Paldea) are also grouped under the same main region
    - DLCs are not considered to be their own separate games

Entity-Relationship (ER) Diagram

Figure 1: Pokemon Database Entity Relationship Diagram



Relational Model (RM) Diagram

Figure 2: Pokemon Database Relational Data Model
Key for RM Diagram:
Italics = Foreign key
Underline = Primary Key

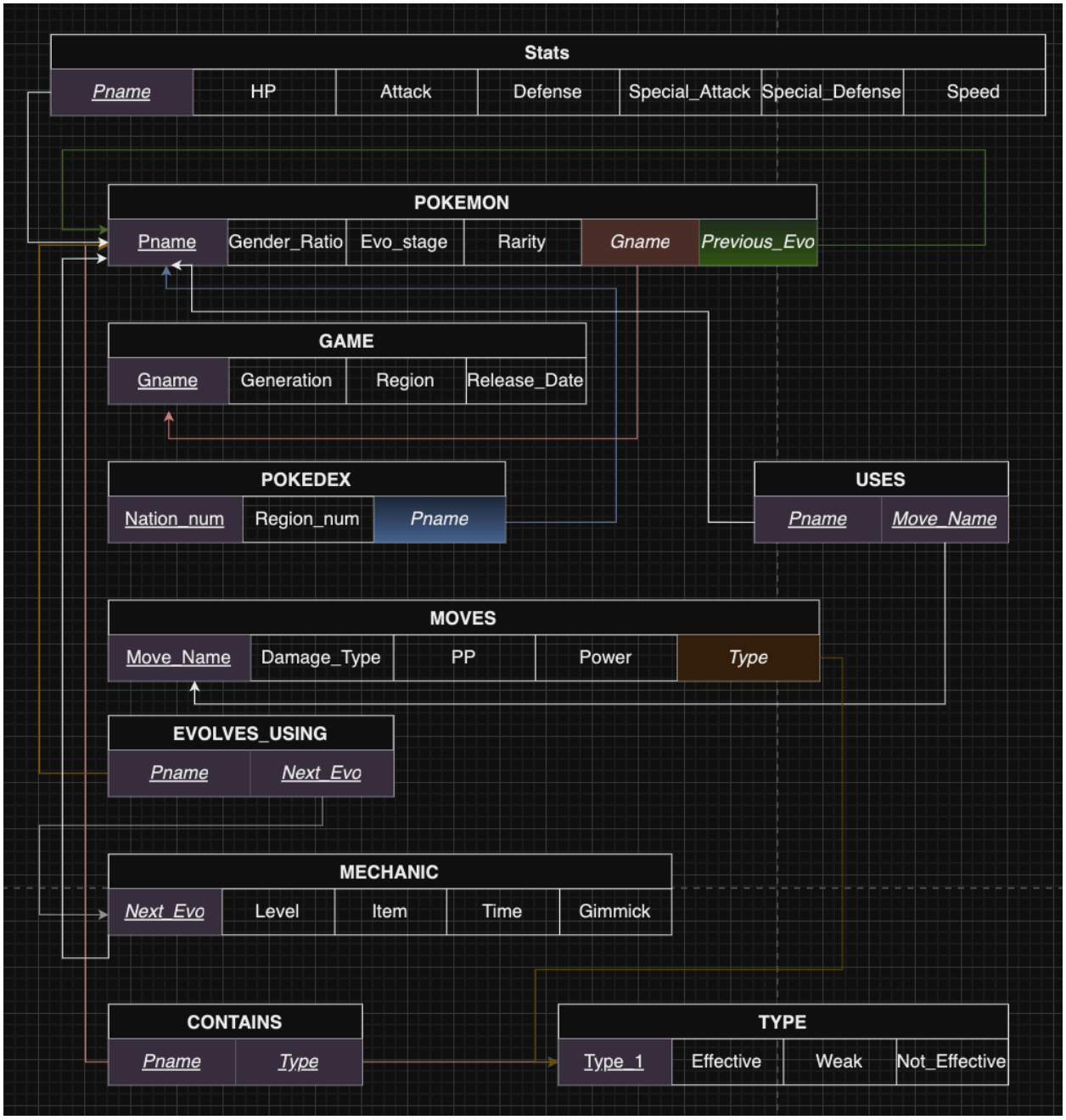


Diagram Constraints

Key Constraints

- National Number must be a unique number
- Gname must be a unique string
- Pname must be a unique string
- Type 1 must be a unique string
- Next_Evo must be a unique string
- Move_Name must be a unique string
- Type 1 and Pname along with another different Type1 and same Pname must be a unique combination not repeating each other (ie NOT “Ralts: Psychic” and another “Ralts: Psychic”)
- The same Pname can only appear twice in the CONTAINS table
- Pname and Move_Name must be a unique combination
- Pname and Next_Evo must be a unique combination

Domain Constraints

Subset Domains

Pokemon:

- Pname: {D2 between length 2 and 10} (ie: Porygon2)
- Evo_Stage: {D1 ranging from 1 to 3}
- Rarity: {“Legendary”, “Mythical”, “Basic”}
- Gender-Ratio: {Decimal in the format of XXX.XX where it is no greater than 100 and no less than 0 unless it is genderless in which cause it will be categorized as -1}
- Previous_Evo: {A currently existing Pokemon in the database (using Pname)}

Stats

- HP: {D1 ranging from 1 to 255}
- Attack: {D1 ranging from 1 to 255}
- Defense: {D1 ranging from 1 to 255}
- Special_Attack: {D1 ranging from 1 to 255}
- Special_Defense: {D1 ranging from 1 to 255}
- Speed: {D1 ranging from 1 to 255}

Pokedex:

- National_num: {D1 below 1018}
- Regional_num: {D1 below 400}

Game:
- Gname: {D2 from range 11 - 50}
- Generation: {D1 only consisting of numbers 1-9}
- Region: {‘Kanto’, ’Johto’, ‘Hoenn’, ‘Sinnoh’, ‘Unova’, ‘Kalos’, ‘Alola’, ‘Galar’, ‘Paldea’}
- Release_Date: {Date in the format of “YYYY-MM-DD”}

Moves:
- Move_Name: {D2 of length 3 to 18}
- Damage_Type: {“Status”, “Physical”, or “Special”}
- PP: {D1 from 1 to 30}
- Power: {D1 from 1 to 250}

Type:
- Type1: {D3 between length 3 and length 8} (shortest is “Ice”, longest is “Fighting)
- Effective: {D3 between length 3 and 50 consisting of Types in Type1}
- Weak: {D3 between length 3 and 50 consisting of Types in Type1}
- Not_Effective: {D3 between length 3 and 50 consisting of Types in Type1}

Mechanic:
- Next_Evo {A currently existing Pokemon in the database (using Pname)}
- Level: {D1 up to 100}
- Item: {D3 between length 3 and 10}
- Time: {“Midday”, “Night”, “Dusk”, “Dawn”, “Midnight”}
- Gimmick: {D2 between length 7 and 25}

Common Domains

D1: {Non-zero positive integer}
D2: {Alphanumeric string}
D3: {Alphabetical string}

Integrity Constraints

Pokemon:
- Pname can’t be null
- Evo_Stage can’t be null
- Gname can’t be null

Stats:
- Pname can’t be null
- HP can’t be null
- Attack can’t be null

- Defense can't be null
- Special_Attack can't be null
- Special_Defense can't be null
- Speed can't be null

Pokedex:
- National number can't be null
- Pname can't be null

Game:
- Gname can't be null
- Generation can't be null
- Region can't be null

Moves:
- Move_Name can't be null
- Damage_Type can't be null
- PP can't be null
- Power can't be null
- Type can't be null

Types:
- Type 1 can't be null

Mechanics:
- Next_Evo can't be null
- Level, Item, Time, and Gimmick can't all be null (At least one must have a value)

CONTAIN, USES, and EVOLVES_USING:
- No attributes can be null

Diagram Discussion

The central point of our database is of course the pokemon themselves. As there are multiple pokemon in existence, most relationships from the pokemon's view are many to one. There is only 1 National Pokedex and 1 of each game. A pokemon has to be part of the National Dex in order to be considered an official pokemon. In order to be in the National Dex, a pokemon must debut in a game. Many pokemon are able to debut in one game. It is also not possible for a pokemon to have 0 types which is why it must have at least one type assigned to

it. The pokedex, games, and pokemon types are all strong entities as they can exist separately from an individual pokemon and be distinctly recognizable on their own. Mechanics are a bit tricky but we decided to keep it as a strong entity as the same Pokemon can evolve using different mechanics.

Most of our difficulties with the diagrams revolved around the entity “Mechanics”. We did consider it to be a weak entity as there was no one single key that could choose to uniquely identify different data. Multiple pokemon evolve at the same level but with different items, time of day, or some other gimmick so we couldn’t use level as a unique identifier. We also couldn’t use items or time of day as those areas could potentially be null for certain pokemon. We brainstormed possible solutions on how to give “Mechanics” a primary key(s) and we ultimately decided that we would give it a Mechanic_ID attribute to have it stand by itself. Since “Mechanics” and “Pokemon” have a many to many relationship, we made another table called “Evolves Using” to help make the relation between the two. Same goes with the tables “Uses” and “Contains”.

One minor concern is how much data the “Pokemon” table will hold. As it has a wide variety of attributes, both main and foreign, we worry that it may be too complex. The only issue with reversing these foreign keys (for example, making type have Pname) is that it wouldn’t make logical sense to do so mainly because of the many to one relationship “Pokemon” has with other entities. The only tables that do have Pname as a foreign key are the many-to-many tables and the Pokedex table. The Pokedex table is able to have a Pname foreign because of our assumption that no 2 Pokemon would share the same National_Num.

There are still some concerns left over about the “Mechanics” entity tables and its relations but hopefully we’re able to iron out any issues before completing our database.

Normalization of Tables:

Now that our ER diagram, relational model and overall database design is complete, it’s time to evaluate our tables to see which normal form our tables are in. In order for a table to be in 1NF it must have no multivalued attributes. For it to be in 2NF every key must be fully functional dependent on all the primary keys. To reach 3NF there must be no transient dependencies in the table.

Moves:

The Moves table is currently in 3NF as there are no multivalued attributes, every key is fully dependent on Move_Name, and there exists no transitive dependencies. Type is a foreign key yet it is still functionally dependent to Move_Name. That is not to say that the Type table is functionally dependent on Moves, just this one attribute.

The dependencies of Moves goes as follows:

- Move_Name → Damage_Type, PP, Power, Type

Figure 3 further down below includes the functional dependencies of Moves in visual form for a clearer understanding

Pokemon:

The Pokemon table is currently in 3NF as there are no multivalued attributes, every key is fully dependent on Pname, and there exists no transitive dependencies. Gname and Previous_Evo are foreign keys yet they are still functionally dependent on Pname. In this case, Pname is the pre-evolution and Gname is referring to the game this Pokemon debuted in. This property is what makes Gname in Pokemon functionally dependent on Pname.

The dependencies of Pokemon goes as follows:

- Pokemon → Gender_Ratio, Evo_Stage, Rarity, Gname, Previous_Evo

Figure 3 further down below includes the functional dependencies of Pokemon in visual form for a clearer understanding

Pokedex:

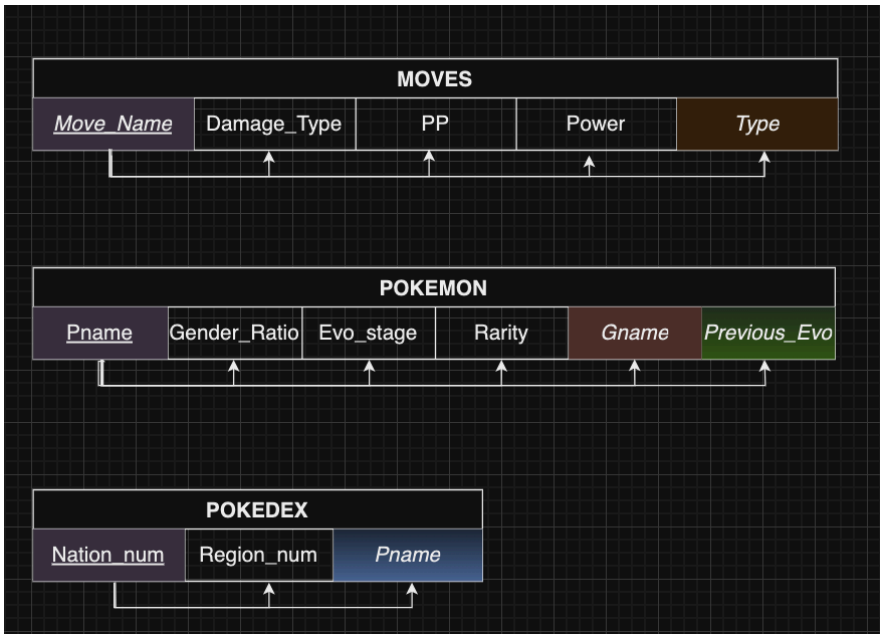
The Pokedex table is currently in 3NF as there are no multivalued attributes, every key is fully dependent on National_num, and there exists no transitive dependencies. Pname is a foreign key yet it is still functionally dependent to Nation_num. Every Pokemon is assigned to only one National Dex Number so that is what explicitly makes it fully dependent on Nation_num.

The dependencies of Pokedex goes as follows:

- Nation_num → Region_Num, Pname

Figure 3 further down below includes the functional dependencies of Pokedex in visual form for a clearer understanding

Figure 3: Functional Dependencies of Moves, Pokemon, and Pokedex



Game:

The Game table is currently in 3NF as there are no multivalued attributes, every key is fully dependent on Gname, and there exists no transitive dependencies. Many may think that Region is dependent on Generation or vice-versa but that is not the case here. Since we are talking about games, the region may have originally been from Generation 4 but the current game may be a remake taking place in a later Generation. It is a common misconception that one attribute determines the other.

The dependencies of Game goes as follows:

- Gname → Generation, Region, Release_Date

Figure 6 further down below includes the functional dependencies of Game in visual form for a clearer understanding

Mechanic:

The Mechanic table is currently in no normal form as there exists multivalued attributes in the form of Gimmick. Gimmick may be confusing for some but it is meant to be a catch-all for any evolution methods not covered by the other attributes. This means that it could be a string like “Turn upside down, run 100 steps” which indicates that there can be multiple values in this

attribute. To improve Mechanic into 1NF we'd have to break Gimmick apart and turn it into multiple individual attributes. For example, we could have a bool called Turn_Upside_Down and set it to either yes or no. "Run 100 steps" could be an int attribute called Steps. The only issue with splitting up Gimmick in this method is that we'd end up with multiple nullable attributes.

Once we reach 1NF, we essentially should be in 3NF. As far as our knowledge goes, none of the split attributes from Gimmick should be functionally dependent on anything else but the Next_Evo.

The original dependencies of Mechanic goes as follows:

- Next_Evo → Level, Item, Time, Gimmick

Figure 4: 3NF form of Mechanic

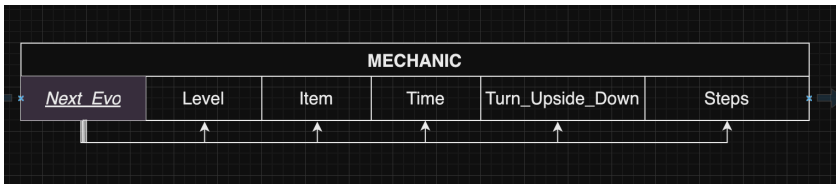


Figure 6 further down below includes the original functional dependencies of Mechanic in graphic form for a clearer understanding

Type:

The Type table is also currently not in any normal form as there are multiple multivalued attributes which are Effective, Weak, and Not_Effective. The reason being that each of these attributes contain many Types listed in a string. For example, Effective could be "Rock, Fairy, Poison" and the same goes for the other 2 attributes I've listed. One remedy to get our table to 1NF would be to split these attributes into smaller categories such as Weak → Weak1, Weak2, Weak 3, etc..

Another solution would be to make these Effectiveness attributes into another relationship table. This could be a table called Weakness that has Type as a Primary Key along with Weak_To as another Primary Key (ie "Rock" then "Water"). We'd still keep the original Type table only that it would have just one attribute. The reason why is that Moves still needs to have a Type associated with it along with Pokemon. It would also make it easier and more convenient to add weaknesses or new Types into the database.

The original dependencies of Type goes as follows:

- Type_1 → Effective, Weak, Not_Effective

Figure 5: 3NF form of Type

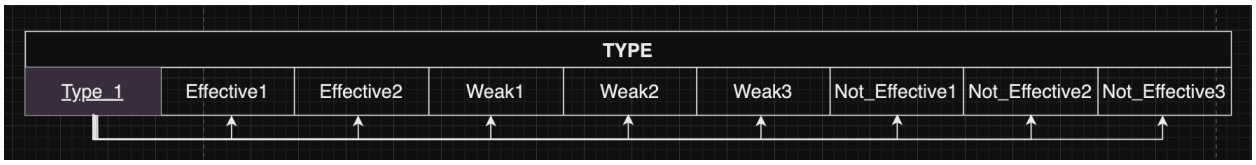
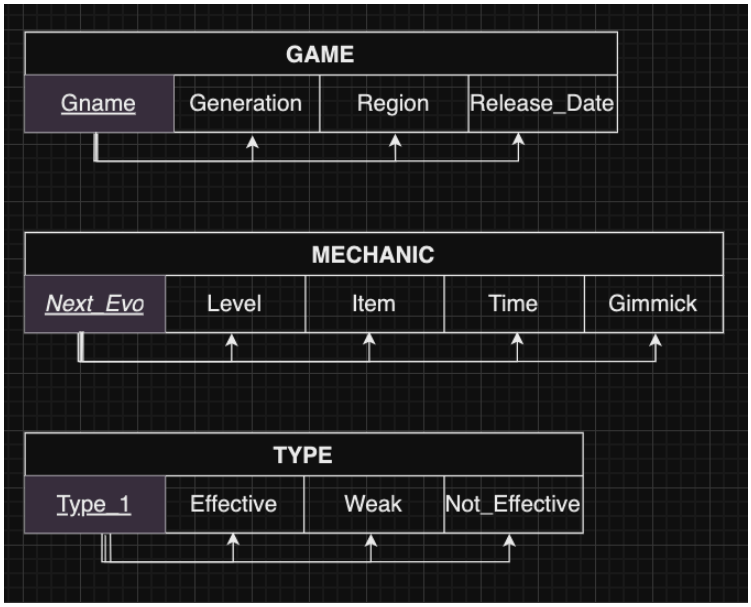


Figure 6 down below includes the functional dependencies of Type in visual form for a clearer understanding

Figure 6: Original Functional Dependencies of Game, Mechanic, and Type



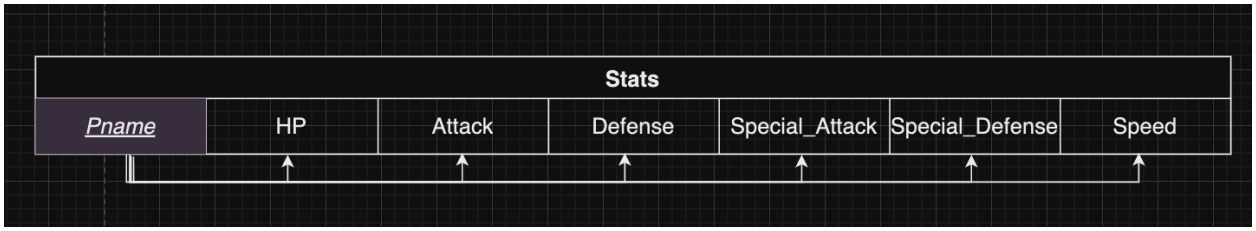
Stats:

The Stats table is in 3NF as there are no multivalued attributes, every key is dependent on Pname, and no transient dependencies present in the table. Without a Pokemon to assign stats to, each stat would be meaningless meaning that they are heavily dependent on the Pokemon.

The original dependencies of Type goes as follows:

- Pname → HP, Attack, Defense, Special_Attack, Special_Defense, Speed

Figure 7: Functional Dependency of Stats



Tooling

Database Management Software:

MySQL

MySQL is open source, and beginner-friendly among our team members. It is easy to learn and shouldn't have as steep of a learning curve as other DBMS.

Hosting and Web Dev:

Microsoft Azure

The team has plenty of reasons to use Microsoft Azure for database hosting. It's easy to integrate with VSC as there are plenty of extensions that make it easy to manage Azure resources such as storage and cloud management. Not to mention Azure is free to use and accessible on any device.

ASP.NET and C#

Considering that Microsoft Azure has great compatibility with ASP.NET, we've decided to use it as our UI elements to program user interfaces and make it easy to access the database. ASP.NET supports MySQL and can also be used in VSC with some added extensions.

Editor:

Visual Studio Code (VSC) + Visual Studio

The team is familiar with using VSC. All of us have been continuously using it throughout our classes and have deep-rooted experience using it. The various extensions provided by VSC also work well with our other tooling options selected.

Git

In order for the team to collaborate easier, we'll be using Github integration and Live Share extension (for pair programming). Not only will it allow us better cooperation and teamwork, but we'll be able keep track of any changes we make to our code.

Additional Tools:
Pokemon API (https://pokeapi.co/docs/v2)

Since generating data about Pokemon can be large and tedious, we'll make use of a Pokemon API data retrieval. This is a read-only API that contains up-to-date information about the Pokemon database. We'll still need to make our own database according to our design but this is a right step forward for generating data quickly and efficiently.

Python

In order to retrieve information from this API, we will be using Python. One of our team members has experience with using Python so we're confident that we'll be able to make the most of it.

Methodology for Data Creation, Population, and Testing

Most of our data was collected using a Pokemon API(https://pokeapi.co/docs/v2) that was publicly available online. Using this API, we were able to extract data that would be useful for insertions formatted into our database. Different methods were used such as Python coding and Postman. We'd use these languages to create text files for insertion into different tables. If any issues arose in the creation of these text files, we'd go back and update our Python code or manually do some quality checks. Before we fully committed to using this data, we used a variety of different resources to cross reference our data. Websites such as Bulbapedia proved to be the most useful especially when finding Pokemon and Pokedex data.

When testing the database, we manually created our own queries to confirm if the database was acting accordingly. Occasionally we did also use some 3rd party services to create mock sample data. This was mostly to test violation constraints. Essentially, we'd ask an AI service (such as ChatGPT or Bing AI) to create a list of insertions to insert into a MySQL database. From there, we'd pick out the queries that made the most sense and provided the most use to testing our database. We'd move these testing queries into a text file, testing specific tables and situations with each use of the AI.

Code for our Python scripts are also included along with the PokemonDB SQL files.

Pokemon Database Table List

Overall, there are a total of 10 tables in our database. 3 of these tables are for M-N relationships and as such are denoted by using all caps in their Table Name. The remaining 7 tables are the main entities consisting of their own data and attributes.

Table 2: List of all Tables in Pokemon DB

Game
Pokemon
Stats
Pokedex
Type
Moves
Mechanic
CONTAIN
USES
EVOLVES_USING

Pokemon Database Queries

Data Creation:

Creating the Game Entity

```
CREATE table Game
(Gname VARCHAR(50) NOT NULL, Generation TINYINT NOT NULL, Region VARCHAR(6) NOT NULL,
Release_Date DATE,
PRIMARY KEY (Gname),
CONSTRAINT Chk_Generation CHECK (Generation >= 1 and Generation <= 9),
CONSTRAINT Chk_Region CHECK (Region = 'Kanto' or Region = 'Johto' or Region = 'Hoenn' or
Region = 'Sinnoh' or Region = 'Unova' or Region = 'Kalos' or Region = 'Alola' or Region
= 'Galar' or Region = 'Paldea')
);
```

Creating Pokemon Entity

```
CREATE table Pokemon
(Pname VARCHAR(15) NOT NULL, Gender_Ratio DECIMAL(5,2), Evo_Stage TINYINT NOT NULL,
Rarity VARCHAR(9), Gname VARCHAR(50) NOT NULL, Previous_Evo VARCHAR(15),
PRIMARY KEY (Pname),
FOREIGN KEY (Gname) REFERENCES Game(Gname) ON UPDATE cascade ON DELETE cascade,
FOREIGN KEY (Previous_Evo) REFERENCES Pokemon(Pname),
CONSTRAINT Chk_Ratio CHECK ((Gender_Ratio >= 0 and Gender_Ratio <= 100) or Gender_Ratio = -1),
CONSTRAINT Chk_Stage CHECK (Evo_Stage = 1 or Evo_Stage = 2 or Evo_Stage = 3),
```

```
CONSTRAINT Chk_Rarity CHECK (Rarity = 'Basic' or Rarity = 'Mythical' or Rarity = 'Legendary')
);
```

Creating the Pokedex Entity

```
CREATE table Pokedex
(Nation_Num SMALLINT NOT NULL, Region_Num SMALLINT, Pname VARCHAR(15) NOT NULL,
PRIMARY KEY (Nation_Num),
UNIQUE (Pname),
FOREIGN KEY (Pname) REFERENCES Pokemon (Pname) ON DELETE cascade ON UPDATE cascade,
CONSTRAINT Chk_Nation CHECK (Nation_Num > 0 and Nation_Num < 1018),
CONSTRAINT Chk_Region_Num CHECK (Region_Num > 0 and Region_Num < 400)
);
```

Creating the Type Entity

```
CREATE table Type
(Type_1 VARCHAR(8) NOT NULL, Effective VARCHAR(50), Weak VARCHAR(50), Not_Effective VARCHAR(50),
PRIMARY KEY (Type_1),
CONSTRAINT Chk_Type CHECK (Type_1 = 'Bug' or Type_1 = 'Dark' or Type_1 = 'Dragon' or Type_1 = 'Electric' or
Type_1 = 'Fairy' or Type_1 = 'Fighting' or Type_1 = 'Fire' or Type_1 = 'Flying' or Type_1 = 'Ghost' or Type_1 = 'Grass'
or Type_1 = 'Ground' or Type_1 = 'Ice' or Type_1 = 'Normal' or Type_1 = 'Poison' or Type_1 = 'Psychic' or Type_1 =
'Rock' or Type_1 = 'Steel' or Type_1 = 'Water')
);
```

Creating the Moves Entity

```
CREATE table Moves
(Move_Name VARCHAR(18) NOT NULL, Damage_Type VARCHAR(8) NOT NULL, PP TINYINT NOT NULL, Power
TINYINT UNSIGNED NOT NULL, Type VARCHAR(8) NOT NULL,
PRIMARY KEY (Move_Name),
FOREIGN KEY (Type) REFERENCES TYPE(Type_1) ON DELETE cascade ON UPDATE cascade,
CONSTRAINT Chk_Damage CHECK (Damage_Type = 'Physical' or Damage_Type = 'Special' or Damage_Type =
'Status'),
CONSTRAINT Chk_PP CHECK (PP >= 1 and PP <= 45),
CONSTRAINT Chk_Power CHECK (Power >= 0 and Power <= 250)
);
```

Creating the Mechanic Entity

```
CREATE table Mechanic
(Next_Evo VARCHAR(15) NOT NULL, Level TINYINT, Item VARCHAR(10), Time VARCHAR(10), Gimmick
VARCHAR(25),
PRIMARY KEY (Next_Evo),
FOREIGN KEY (Next_Evo) REFERENCES Pokemon(Pname) ON UPDATE cascade ON DELETE cascade,
CONSTRAINT Chk_Level CHECK (Level >= 1 and Level <= 100),
CONSTRAINT Chk_Time CHECK (Time = 'Night' or Time = 'Midday' or Time = 'Dusk' or Time = 'Dawn' or Time =
'Midnight')
);
```

Creating the CONTAIN Entity *(All caps to signify it is a M to N table)*

```
CREATE table CONTAIN
(Pname VARCHAR(15) NOT NULL, Type VARCHAR(8) NOT NULL,
PRIMARY KEY (Pname, Type),
FOREIGN KEY (Pname) REFERENCES Pokemon(Pname) ON UPDATE cascade ON DELETE cascade,
FOREIGN KEY (Type) REFERENCES Type(Type_1) ON UPDATE cascade ON DEIETE cascade
);
```

Creating the USES Entity *(All caps to signify it is a M to N table)*

```
CREATE table USES
(Pname VARCHAR(15) NOT NULL, Move_Name VARCHAR(18) NOT NULL,
PRIMARY KEY (Pname, Move_Name),
FOREIGN KEY (Pname) REFERENCES Pokemon(Pname) ON UPDATE cascade ON DELETE cascade,
FOREIGN KEY (Move_Name) REFERENCES Moves(Move_Name) ON UPDATE cascade ON DELETE cascade
);
```

Creating the EVOLVES_USING Entity *(All caps to signify it is a M to N table)*

```
CREATE table EVOLVES_USING
(Pname VARCHAR(15) NOT NULL, Next_Evo VARCHAR(15),
PRIMARY KEY (Pname, Next_Evo),
FOREIGN KEY (Pname) REFERENCES Pokemon(Pname) ON UPDATE cascade ON DELETE cascade,
FOREIGN KEY (Next_Evo) REFERENCES Mechanic(Next_Evo) ON UPDATE cascade ON DEIETE cascade
);
```

Data Insertions

For populating the database we have included the following SQL insertions. In some cases, these are not the entirety of our insertions used. We have included ****Notes**** at the end of a Table section to indicate if certain insertions were omitted for the document. This omitted information is still present in the SQL files so there is no need to worry there.

Table Pokemon:

```
/*Generation 1*/

INSERT INTO Pokemon VALUES ('Bulbasaur', 87.50, 1, 'Basic', 'Pokemon Red and Blue', NULL);
INSERT INTO Pokemon VALUES ('Ivysaur', 87.50, 2, 'Basic', 'Pokemon Red and Blue', 'Bulbasaur');
INSERT into Pokemon VALUES ('Venusaur', 87.5, 3, 'Basic', 'Pokemon Red and Blue', 'Ivysaur');

INSERT into Pokemon VALUES ('Charmander', 87.5, 1, 'Basic', 'Pokemon Red and Blue', NULL);
INSERT INTO POKEMON VALUES ('Charmeleon', 87.5, 2, 'Basic', 'Pokemon Red and Blue', 'Charmander');
INSERT INTO POKEMON VALUES ('Charizard', 87.5, 3, 'Basic', 'Pokemon Red and Blue', 'Charmeleon');

INSERT into Pokemon VALUES ('Squirtle', 87.5, 1, 'Basic', 'Pokemon Red and Blue', NULL);
INSERT into Pokemon VALUES ('Wartortle', 87.5, 2, 'Basic', 'Pokemon Red and Blue', 'Squirtle');
INSERT into Pokemon VALUES ('Blastoise', 87.5, 3, 'Basic', 'Pokemon Red and Blue', 'Wartortle');
```

```
INSERT into Pokemon VALUES ('Caterpie', 50, 1, 'Basic', 'Pokemon Red and Blue', NULL);
INSERT into Pokemon VALUES ('Metapod', 50, 2, 'Basic', 'Pokemon Red and Blue', 'Caterpie');
INSERT into Pokemon VALUES ('Butterfree', 50, 3, 'Basic', 'Pokemon Red and Blue', 'Metapod');

INSERT into Pokemon VALUES ('Weedle', 50, 1, 'Basic', 'Pokemon Red and Blue', NULL);
INSERT into Pokemon VALUES ('Kakuna', 50, 2, 'Basic', 'Pokemon Red and Blue', 'Weedle');
INSERT into Pokemon VALUES ('Beedrill', 50, 3, 'Basic', 'Pokemon Red and Blue', 'Kakuna');

INSERT into Pokemon VALUES ('Pidgey', 50, 1, 'Basic', 'Pokemon Red and Blue', NULL);
INSERT into Pokemon VALUES ('Pidgeotto', 50, 2, 'Basic', 'Pokemon Red and Blue', 'Pidgey');
INSERT into Pokemon VALUES ('Pidgeot', 50, 3, 'Basic', 'Pokemon Red and Blue', 'Pidgeotto');

INSERT into Pokemon VALUES ('Rattata', 50, 1, 'Basic', 'Pokemon Red and Blue', NULL);
INSERT into Pokemon VALUES ('Raticate', 50, 2, 'Basic', 'Pokemon Red and Blue', 'Rattata');

INSERT into Pokemon VALUES ('Spearow', 50, 1, 'Basic', 'Pokemon Red and Blue', NULL);
INSERT into Pokemon VALUES ('Fearow', 50, 2, 'Basic', 'Pokemon Red and Blue', NULL);

INSERT into Pokemon VALUES ('Ekans', 50, 1, 'Basic', 'Pokemon Red and Blue', NULL);
INSERT into Pokemon VALUES ('Arbok', 50, 2, 'Basic', 'Pokemon Red and Blue', 'Ekans');

INSERT into Pokemon VALUES ('Sandshrew', 50, 1, 'Basic', 'Pokemon Red and Blue', NULL);
INSERT into Pokemon VALUES ('Sandslash', 50, 2, 'Basic', 'Pokemon Red and Blue', 'Sandshrew');

INSERT into Pokemon VALUES ('Nidoran♀', 0, 1, 'Basic', 'Pokemon Red and Blue', NULL);
INSERT into Pokemon VALUES ('Nidorina', 0, 2, 'Basic', 'Pokemon Red and Blue', 'Nidoran♀');
INSERT into Pokemon VALUES ('Nidoqueen', 0, 3, 'Basic', 'Pokemon Red and Blue', 'Nidorina');

INSERT into Pokemon VALUES ('Nidoran♂', 100, 1, 'Basic', 'Pokemon Red and Blue', NULL);
INSERT into Pokemon VALUES ('Nidorino', 100, 2, 'Basic', 'Pokemon Red and Blue', 'Nidoran♂');
INSERT into Pokemon VALUES ('Nidoking', 100, 3, 'Basic', 'Pokemon Red and Blue', 'Nidorino');

INSERT into Pokemon VALUES ('Cleffa', 25, 1, 'Basic', 'Pokemon Gold and Silver', NULL);
INSERT into Pokemon VALUES ('Clefairy', 25, 2, 'Basic', 'Pokemon Red and Blue', NULL);
INSERT into Pokemon VALUES ('Clefable', 25, 3, 'Basic', 'Pokemon Red and Blue', NULL);

INSERT into Pokemon VALUES ('Igglybuff', 25, 1, 'Basic', 'Pokemon Gold and Silver', NULL);
INSERT into Pokemon VALUES ('Jigglypuff', 25, 2, 'Basic', 'Pokemon Red and Blue', 'Igglybuff');
INSERT into Pokemon VALUES ('Wigglytuff', 25, 3, 'Basic', 'Pokemon Red and Blue', 'Jigglypuff');

INSERT into Pokemon VALUES ('Zubat', 50, 1, 'Basic', 'Pokemon Red and Blue', NULL);
INSERT into Pokemon VALUES ('Golbat', 50, 2, 'Basic', 'Pokemon Red and Blue', 'Zubat');

INSERT into Pokemon VALUES ('Oddish', 50, 1, 'Basic', 'Pokemon Red and Blue', NULL);
INSERT into Pokemon VALUES ('Gloom', 50, 2, 'Basic', 'Pokemon Red and Blue', 'Oddish');
INSERT into Pokemon VALUES ('Vileplume', 50, 3, 'Basic', 'Pokemon Red and Blue', 'Gloom');

INSERT into Pokemon VALUES ('Paras', 50, 1, 'Basic', 'Pokemon Red and Blue', NULL);
INSERT into Pokemon VALUES ('Parasect', 50, 2, 'Basic', 'Pokemon Red and Blue', 'Paras');

INSERT into Pokemon Values ('Venonat', 50, 1, 'Basic', 'Pokemon Red and Blue', NULL);
INSERT into Pokemon VALUES ('Venomoth', 50, 2, 'Basic', 'Pokemon Red and Blue', 'Venonat');
```

```
INSERT into Pokemon VALUES ('Mew', -1, 1, 'Mythical', 'Pokemon Red and Blue', NULL);
INSERT into Pokemon VALUES ('Mewtwo', -1, 1, 'Legendary', 'Pokemon Red and Blue', NULL);

INSERT into Pokemon VALUES ('Vulpix', 25.0, 1, 'Basic', 'Pokemon Red and Blue', NULL);
INSERT into Pokemon VALUES ('Ninetales', 25.0, 2, 'Basic', 'Pokemon Red and Blue', 'Vulpix');

/*Was added in later generation*/
INSERT into Pokemon Values ('Pichu', 50.0, 1, 'Basic', 'Pokemon Gold and Silver', NULL);
INSERT into Pokemon VALUES ('Pikachu', 50.0, 2, 'Basic', 'Pokemon Red and Blue', 'Pichu');
INSERT into Pokemon VALUES ('Raichu', 50.0, 3, 'Basic', 'Pokemon Red and Blue', 'Pikachu');

INSERT into Pokemon VALUES ('Cubone', 50.0, 1, 'Basic', 'Pokemon Red and Blue', NULL);
INSERT into Pokemon VALUES ('Marowak', 50.0, 2, 'Basic', 'Pokemon Red and Blue', 'Cubone');

INSERT into Pokemon VALUES ('Eevee', 87.5, 1, 'Basic', 'Pokemon Red and Blue', NULL);
INSERT into Pokemon VALUES ('Vaporeon', 87.5, 2, 'Basic', 'Pokemon Red and Blue', 'Eevee');
INSERT into Pokemon VALUES ('Jolteon', 87.5, 2, 'Basic', 'Pokemon Red and Blue', 'Eevee');
INSERT into Pokemon VALUES ('Flareon', 87.5, 2, 'Basic', 'Pokemon Red and Blue', 'Eevee');

INSERT into Pokemon VALUES ('Porygon', -1, 1, 'Basic', 'Pokemon Red and Blue', NULL);


/*Generation 2*/

INSERT into Pokemon VALUES ('Chikorita', 87.5, 1, 'Basic', 'Pokemon Gold and Silver', NULL);
INSERT into Pokemon VALUES ('Bayleef', 87.5, 2, 'Basic', 'Pokemon Gold and Silver', 'Chikorita');
INSERT into Pokemon VALUES ('Meganium', 87.5, 3, 'Basic', 'Pokemon Gold and Silver', 'Chikorita');

INSERT into Pokemon VALUES ('Cyndaquil', 87.5, 1, 'Basic', 'Pokemon Gold and Silver', NULL);
INSERT into Pokemon VALUES ('Quilava', 87.5, 2, 'Basic', 'Pokemon Gold and Silver', 'Cyndaquil');
INSERT into Pokemon VALUES ('Typhlosion', 87.5, 3, 'Basic', 'Pokemon Gold and Silver', 'Quilava');

INSERT into Pokemon VALUES ('Totodile', 87.5, 1, 'Basic', 'Pokemon Gold and Silver', NULL);
INSERT into Pokemon VALUES ('Croconaw', 87.5, 2, 'Basic', 'Pokemon Gold and Silver', 'Totodile');
INSERT into Pokemon VALUES ('Feraligatr', 87.5, 3, 'Basic', 'Pokemon Gold and Silver', 'Croconaw');

INSERT into Pokemon VALUES ('Togepi', 87.5, 1, 'Basic', 'Pokemon Gold and Silver', NULL);
INSERT into Pokemon VALUES ('Togetic', 87.5, 2, 'Basic', 'Pokemon Gold and Silver', 'Togepi');

INSERT into Pokemon VALUES ('Wooper', 50, 1, 'Basic', 'Pokemon Gold and Silver', NULL);
INSERT into Pokemon VALUES ('Quagsire', 50, 2, 'Basic', 'Pokemon Gold and Silver', 'Wooper');

INSERT into Pokemon VALUES ('Celebi', -1, 1, 'Mythical', 'Pokemon Gold and Silver', NULL);

INSERT into Pokemon VALUES ('Ho-Oh', -1, 1, 'Legendary', 'Pokemon Gold and Silver', NULL);

INSERT into Pokemon VALUES ('Lugia', -1, 1, 'Legendary', 'Pokemon Gold and Silver', NULL);

INSERT into Pokemon VALUES ('Raikou', -1, 1, 'Legendary', 'Pokemon Gold and Silver', NULL);
INSERT into Pokemon VALUES ('Enter', -1, 1, 'Legendary', 'Pokemon Gold and Silver', NULL);
INSERT into Pokemon VALUES ('Suicune', -1, 1, 'Legendary', 'Pokemon Gold and Silver', NULL);
```

INSERT into Pokemon VALUES ('Espeon', 87.5, 2, 'Basic', 'Pokemon Gold and Silver', 'Eevee');
INSERT into Pokemon VALUES ('Umbreon', 87.5, 2, 'Basic', 'Pokemon Gold and Silver', 'Eevee');

INSERT into Pokemon VALUES ('Porygon2', -1, 2, 'Basic', 'Pokemon Gold and Silver', 'Porygon');

INSERT into Pokemon VALUES ('Swinub', 87.5, 1, 'Basic', 'Pokemon Gold and Silver', NULL);
INSERT into Pokemon VALUES ('Piloswine', 87.5, 2, 'Basic', 'Pokemon Gold and Silver', 'Swinub');

*****Note: Pokemon from Generation 3 and 4 are present in the database. For the sake of efficiency, we have not included these inserts here but they are in the "Pokemon.sql" file*****

Table Game:

INSERT into Game VALUES ('Pokemon Red and Blue', 1, 'Kanto', '1998-09-28');
INSERT into Game VALUES ('Pokemon Yellow', 1, 'Kanto', '1999-10-18');
INSERT into Game VALUES ('Pokemon Gold and Silver', 2, 'Johto', '2000-10-15');
INSERT into Game VALUES ('Pokemon Crystal', 2, 'Johto', '2001-06-29');
INSERT into Game VALUES ('Pokemon Ruby and Sapphire', 3, 'Hoenn', '2003-03-19');
INSERT into Game VALUES ('Pokemon FireRed and LeafGreen', 3, 'Kanto', '2004-09-09');
INSERT into Game VALUES ('Pokemon Emerald', 3, 'Hoenn', '2005-05-01');
INSERT into Game VALUES ('Pokemon Diamond and Pearl', 4, 'Sinnoh', '2007-04-22');

Table Pokedex:

INSERT INTO POKEDEX VALUES (1, NULL, 'Bulbasaur');
INSERT INTO POKEDEX VALUES (2, NULL, 'Ivysaur');
INSERT INTO POKEDEX VALUES (3, NULL, 'Venusaur');
INSERT INTO POKEDEX VALUES (4, NULL, 'Charmander');
INSERT INTO POKEDEX VALUES (5, NULL, 'Charmeleon');
INSERT INTO POKEDEX VALUES (6, NULL, 'Charizard');
INSERT INTO POKEDEX VALUES (7, NULL, 'Squirtle');
INSERT INTO POKEDEX VALUES (8, NULL, 'Wartortle');
INSERT INTO POKEDEX VALUES (9, NULL, 'Blastoise');
INSERT INTO POKEDEX VALUES (10, NULL, 'Caterpie');
INSERT INTO POKEDEX VALUES (11, NULL, 'Metapod');
INSERT INTO POKEDEX VALUES (12, NULL, 'Butterfree');
INSERT INTO POKEDEX VALUES (13, NULL, 'Weedle');
INSERT INTO POKEDEX VALUES (14, NULL, 'Kakuna');
INSERT INTO POKEDEX VALUES (15, NULL, 'Beedrill');
INSERT INTO POKEDEX VALUES (16, NULL, 'Pidgey');
INSERT INTO POKEDEX VALUES (17, NULL, 'Pidgeotto');
INSERT INTO POKEDEX VALUES (18, NULL, 'Pidgeot');
INSERT INTO POKEDEX VALUES (19, NULL, 'Rattata');
INSERT INTO POKEDEX VALUES (20, NULL, 'Raticate');

Table Type:

INSERT INTO TYPE VALUES ('Normal', Null, 'Fighting', 'Rock');
INSERT INTO TYPE VALUES ('Fighting', 'Normal', 'Flying', 'Flying');
INSERT INTO TYPE VALUES ('Flying', 'Fighting', 'Rock', 'Rock');
INSERT INTO TYPE VALUES ('Poison', 'Grass', 'Ground', 'Poison');
INSERT INTO TYPE VALUES ('Ground', 'Poison', 'Water', 'Bug');

INSERT INTO TYPE VALUES ('Rock', 'Flying', 'Fighting', 'Fighting');
INSERT INTO TYPE VALUES ('Bug', 'Grass', 'Flying', 'Fighting');
INSERT INTO TYPE VALUES ('Ghost', 'Ghost', 'Ghost', 'Dark');
INSERT INTO TYPE VALUES ('Steel', 'Rock', 'Fighting', 'Steel');
INSERT INTO TYPE VALUES ('Fire', 'Bug', 'Ground', 'Rock');
INSERT INTO TYPE VALUES ('Water', 'Ground', 'Grass', 'Water');
INSERT INTO TYPE VALUES ('Grass', 'Ground', 'Flying', 'Flying');
INSERT INTO TYPE VALUES ('Electric', 'Flying', 'Ground', 'Grass');
INSERT INTO TYPE VALUES ('Psychic', 'Fighting', 'Bug', 'Steel');
INSERT INTO TYPE VALUES ('Ice', 'Flying', 'Fighting', 'Steel');
INSERT INTO TYPE VALUES ('Dragon', 'Dragon', 'Ice', 'Steel');
INSERT INTO TYPE VALUES ('Dark', 'Ghost', 'Fighting', 'Fighting');
INSERT INTO TYPE VALUES ('Fairy', 'Fighting', 'Poison', 'Poison');

Table Moves:

INSERT INTO MOVES VALUES ('Pound', 'Physical', 35, 40, 'Normal');
INSERT INTO MOVES VALUES ('Karate-chop', 'Physical', 25, 50, 'Fighting');
INSERT INTO MOVES VALUES ('Double-slap', 'Physical', 10, 15, 'Normal');
INSERT INTO MOVES VALUES ('Comet-punch', 'Physical', 15, 18, 'Normal');
INSERT INTO MOVES VALUES ('Mega-punch', 'Physical', 20, 80, 'Normal');
INSERT INTO MOVES VALUES ('Pay-day', 'Physical', 20, 40, 'Normal');
INSERT INTO MOVES VALUES ('Fire-punch', 'Physical', 15, 75, 'Fire');
INSERT INTO MOVES VALUES ('Ice-punch', 'Physical', 15, 75, 'Ice');
INSERT INTO MOVES VALUES ('Thunder-punch', 'Physical', 15, 75, 'Electric');
INSERT INTO MOVES VALUES ('Scratch', 'Physical', 35, 40, 'Normal');
INSERT INTO MOVES VALUES ('Vice-grip', 'Physical', 30, 55, 'Normal');
INSERT INTO MOVES VALUES ('Guillotine', 'Physical', 5, 0, 'Normal');
INSERT INTO MOVES VALUES ('Razor-wind', 'Special', 10, 80, 'Normal');
INSERT INTO MOVES VALUES ('Swords-dance', 'Status', 20, 0, 'Normal');
INSERT INTO MOVES VALUES ('Cut', 'Physical', 30, 50, 'Normal');
INSERT INTO MOVES VALUES ('Gust', 'Special', 35, 40, 'Flying');
INSERT INTO MOVES VALUES ('Wing-attack', 'Physical', 35, 60, 'Flying');
INSERT INTO MOVES VALUES ('Whirlwind', 'Status', 20, 0, 'Normal');
INSERT INTO MOVES VALUES ('Fly', 'Physical', 15, 90, 'Flying');
INSERT INTO MOVES VALUES ('Bind', 'Physical', 20, 15, 'Normal');
INSERT INTO MOVES VALUES ('Slam', 'Physical', 20, 80, 'Normal');
INSERT INTO MOVES VALUES ('Vine-whip', 'Physical', 25, 45, 'Grass');
INSERT INTO MOVES VALUES ('Stomp', 'Physical', 20, 65, 'Normal');
INSERT INTO MOVES VALUES ('Double-kick', 'Physical', 30, 30, 'Fighting');
INSERT INTO MOVES VALUES ('Mega-kick', 'Physical', 5, 120, 'Normal');
INSERT INTO MOVES VALUES ('Jump-kick', 'Physical', 10, 100, 'Fighting');
INSERT INTO MOVES VALUES ('Rolling-kick', 'Physical', 15, 60, 'Fighting');
INSERT INTO MOVES VALUES ('Sand-attack', 'Status', 15, 0, 'Ground');
INSERT INTO MOVES VALUES ('Headbutt', 'Physical', 15, 70, 'Normal');
INSERT INTO MOVES VALUES ('Horn-attack', 'Physical', 25, 65, 'Normal');
INSERT INTO MOVES VALUES ('Fury-attack', 'Physical', 20, 15, 'Normal');
INSERT INTO MOVES VALUES ('Horn-drill', 'Physical', 5, 0, 'Normal');
INSERT INTO MOVES VALUES ('Tackle', 'Physical', 35, 40, 'Normal');
INSERT INTO MOVES VALUES ('Body-slam', 'Physical', 15, 85, 'Normal');
INSERT INTO MOVES VALUES ('Wrap', 'Physical', 20, 15, 'Normal');
INSERT INTO MOVES VALUES ('Take-down', 'Physical', 20, 90, 'Normal');

```
INSERT INTO MOVES VALUES ('Thrash', 'Physical', 10, 120, 'Normal');
INSERT INTO MOVES VALUES ('Double-edge', 'Physical', 15, 120, 'Normal');
INSERT INTO MOVES VALUES ('Tail-whip', 'Status', 30, 0, 'Normal');
INSERT INTO MOVES VALUES ('Poison-sting', 'Physical', 35, 15, 'Poison');
INSERT INTO MOVES VALUES ('Twineedle', 'Physical', 20, 25, 'Bug');
```

*****Note: Not all insertions detailed here in report. "Moves.sql" file includes all insertions used*****

Table Mechanic:

```
INSERT INTO MECHANIC VALUES ('Ivysaur', 16, NULL, NULL, NULL);
INSERT INTO MECHANIC VALUES ('Charmeleon', 16, NULL, NULL, NULL);
INSERT INTO MECHANIC VALUES ('Wartortle', 16, NULL, NULL, NULL);
INSERT INTO MECHANIC VALUES ('Metapod', 7, NULL, NULL, NULL);
INSERT INTO MECHANIC VALUES ('Kakuna', 7, NULL, NULL, NULL);
INSERT INTO MECHANIC VALUES ('Pidgeotto', 18, NULL, NULL, NULL);
INSERT INTO MECHANIC VALUES ('Raticate', 20, NULL, NULL, NULL);
INSERT INTO MECHANIC VALUES ('Fearow', 20, NULL, NULL, NULL);
INSERT INTO MECHANIC VALUES ('Arbok', 22, NULL, NULL, NULL);
INSERT INTO MECHANIC VALUES ('Pikachu', NULL, NULL, NULL, NULL);
INSERT INTO MECHANIC VALUES ('Sandslash', 22, NULL, NULL, NULL);
INSERT INTO MECHANIC VALUES ('Nidorina', 16, NULL, NULL, NULL);
INSERT INTO MECHANIC VALUES ('Nidorino', 16, NULL, NULL, NULL);
INSERT INTO MECHANIC VALUES ('Clefairy', NULL, NULL, NULL, NULL);
INSERT INTO MECHANIC VALUES ('Ninetales', NULL, 'fire-stone', NULL, NULL);
INSERT INTO MECHANIC VALUES ('Jigglypuff', NULL, NULL, NULL, NULL);
INSERT INTO MECHANIC VALUES ('Golbat', 22, NULL, NULL, NULL);
INSERT INTO MECHANIC VALUES ('Gloom', 21, NULL, NULL, NULL);
INSERT INTO MECHANIC VALUES ('Parasect', 24, NULL, NULL, NULL);
INSERT INTO MECHANIC VALUES ('Venomoth', 31, NULL, NULL, NULL);
```

Table USES:

```
INSERT INTO USES VALUES ('Bulbasaur', 'Swords-dance');
INSERT INTO USES VALUES ('Bulbasaur', 'Cut');
INSERT INTO USES VALUES ('Bulbasaur', 'Bind');
INSERT INTO USES VALUES ('Bulbasaur', 'Vine-whip');
INSERT INTO USES VALUES ('Bulbasaur', 'Headbutt');
INSERT INTO USES VALUES ('Ivysaur', 'Cut');
INSERT INTO USES VALUES ('Ivysaur', 'Bind');
INSERT INTO USES VALUES ('Ivysaur', 'Vine-whip');
INSERT INTO USES VALUES ('Ivysaur', 'Headbutt');
INSERT INTO USES VALUES ('Ivysaur', 'Tackle');
INSERT INTO USES VALUES ('Venusaur', 'Cut');
INSERT INTO USES VALUES ('Venusaur', 'Bind');
INSERT INTO USES VALUES ('Venusaur', 'Vine-whip');
INSERT INTO USES VALUES ('Venusaur', 'Headbutt');
INSERT INTO USES VALUES ('Venusaur', 'Tackle');
INSERT INTO USES VALUES ('Charmander', 'Fire-punch');
INSERT INTO USES VALUES ('Charmander', 'Thunder-punch');
INSERT INTO USES VALUES ('Charmander', 'Scratch');
INSERT INTO USES VALUES ('Charmander', 'Swords-dance');
INSERT INTO USES VALUES ('Charmander', 'Cut');
INSERT INTO USES VALUES ('Charmeleon', 'Fire-punch');
```

```
INSERT INTO USES VALUES ('Charmeleon', 'Thunder-punch');
INSERT INTO USES VALUES ('Charmeleon', 'Scratch');
INSERT INTO USES VALUES ('Charmeleon', 'Swords-dance');
INSERT INTO USES VALUES ('Charmeleon', 'Cut');
INSERT INTO USES VALUES ('Charizard', 'Fire-punch');
INSERT INTO USES VALUES ('Charizard', 'Thunder-punch');
INSERT INTO USES VALUES ('Charizard', 'Scratch');
INSERT INTO USES VALUES ('Charizard', 'Swords-dance');
INSERT INTO USES VALUES ('Charizard', 'Cut');
INSERT INTO USES VALUES ('Squirtle', 'Ice-punch');
INSERT INTO USES VALUES ('Squirtle', 'Mega-kick');
INSERT INTO USES VALUES ('Squirtle', 'Headbutt');
INSERT INTO USES VALUES ('Squirtle', 'Tackle');
INSERT INTO USES VALUES ('Squirtle', 'Body-slam');
INSERT INTO USES VALUES ('Wartortle', 'Ice-punch');
INSERT INTO USES VALUES ('Wartortle', 'Mega-kick');
INSERT INTO USES VALUES ('Wartortle', 'Headbutt');
INSERT INTO USES VALUES ('Wartortle', 'Tackle');
INSERT INTO USES VALUES ('Wartortle', 'Body-slam');
INSERT INTO USES VALUES ('Blastoise', 'Ice-punch');
INSERT INTO USES VALUES ('Blastoise', 'Mega-kick');
INSERT INTO USES VALUES ('Blastoise', 'Headbutt');
INSERT INTO USES VALUES ('Blastoise', 'Tackle');
INSERT INTO USES VALUES ('Blastoise', 'Body-slam');
INSERT INTO USES VALUES ('Caterpie', 'String-shot');
INSERT INTO USES VALUES ('Caterpie', 'Snore');
INSERT INTO USES VALUES ('Caterpie', 'Bug-bite');
INSERT INTO USES VALUES ('Caterpie', 'Electroweb');
INSERT INTO USES VALUES ('Metapod', 'Harden');
INSERT INTO USES VALUES ('Metapod', 'Iron-defense');
INSERT INTO USES VALUES ('Metapod', 'Bug-bite');
INSERT INTO USES VALUES ('Metapod', 'Electroweb');
INSERT INTO USES VALUES ('Butterfree', 'Gust');
INSERT INTO USES VALUES ('Butterfree', 'Whirlwind');
INSERT INTO USES VALUES ('Butterfree', 'Headbutt');
INSERT INTO USES VALUES ('Butterfree', 'Tackle');
INSERT INTO USES VALUES ('Butterfree', 'Take-down');
INSERT INTO USES VALUES ('Weedle', 'String-shot');
INSERT INTO USES VALUES ('Weedle', 'Bug-bite');
INSERT INTO USES VALUES ('Weedle', 'Electroweb');
INSERT INTO USES VALUES ('Kakuna', 'Harden');
INSERT INTO USES VALUES ('Kakuna', 'Iron-defense');
INSERT INTO USES VALUES ('Kakuna', 'Bug-bite');
INSERT INTO USES VALUES ('Kakuna', 'Electroweb');
INSERT INTO USES VALUES ('Beedrill', 'Cut');
INSERT INTO USES VALUES ('Beedrill', 'Headbutt');
INSERT INTO USES VALUES ('Beedrill', 'Fury-attack');
INSERT INTO USES VALUES ('Beedrill', 'Take-down');
INSERT INTO USES VALUES ('Beedrill', 'Double-edge');
INSERT INTO USES VALUES ('Pidgey', 'Gust');
INSERT INTO USES VALUES ('Pidgey', 'Wing-attack');
INSERT INTO USES VALUES ('Pidgey', 'Whirlwind');
INSERT INTO USES VALUES ('Pidgey', 'Fly');
```

INSERT INTO USES VALUES ('Pidgey', 'Sand-attack');
INSERT INTO USES VALUES ('Pidgeotto', 'Gust');
INSERT INTO USES VALUES ('Pidgeotto', 'Wing-attack');
INSERT INTO USES VALUES ('Pidgeotto', 'Whirlwind');
INSERT INTO USES VALUES ('Pidgeotto', 'Fly');
INSERT INTO USES VALUES ('Pidgeotto', 'Sand-attack');
INSERT INTO USES VALUES ('Pidgeot', 'Gust');
INSERT INTO USES VALUES ('Pidgeot', 'Wing-attack');
INSERT INTO USES VALUES ('Pidgeot', 'Whirlwind');
INSERT INTO USES VALUES ('Pidgeot', 'Fly');
INSERT INTO USES VALUES ('Pidgeot', 'Sand-attack');
INSERT INTO USES VALUES ('Rattata', 'Headbutt');
INSERT INTO USES VALUES ('Rattata', 'Tackle');
INSERT INTO USES VALUES ('Rattata', 'Body-slam');
INSERT INTO USES VALUES ('Rattata', 'Take-down');
INSERT INTO USES VALUES ('Rattata', 'Double-edge');
INSERT INTO USES VALUES ('Raticate', 'Cut');
INSERT INTO USES VALUES ('Raticate', 'Headbutt');
INSERT INTO USES VALUES ('Raticate', 'Tackle');
INSERT INTO USES VALUES ('Raticate', 'Body-slam');
INSERT INTO USES VALUES ('Raticate', 'Take-down');

Table CONTAINS:

INSERT INTO CONTAIN VALUES ('Bulbasaur', 'Grass');
INSERT INTO CONTAIN VALUES ('Ivysaur', 'Grass');
INSERT INTO CONTAIN VALUES ('Venusaur', 'Grass');
INSERT INTO CONTAIN VALUES ('Charmander', 'Fire');
INSERT INTO CONTAIN VALUES ('Charmeleon', 'Fire');
INSERT INTO CONTAIN VALUES ('Charizard', 'Fire');
INSERT INTO CONTAIN VALUES ('Squirtle', 'Water');
INSERT INTO CONTAIN VALUES ('Wartortle', 'Water');
INSERT INTO CONTAIN VALUES ('Blastoise', 'Water');
INSERT INTO CONTAIN VALUES ('Caterpie', 'Bug');
INSERT INTO CONTAIN VALUES ('Metapod', 'Bug');
INSERT INTO CONTAIN VALUES ('Butterfree', 'Bug');
INSERT INTO CONTAIN VALUES ('Weedle', 'Bug');
INSERT INTO CONTAIN VALUES ('Kakuna', 'Bug');
INSERT INTO CONTAIN VALUES ('Beedrill', 'Bug');
INSERT INTO CONTAIN VALUES ('Pidgey', 'Normal');
INSERT INTO CONTAIN VALUES ('Pidgeotto', 'Normal');
INSERT INTO CONTAIN VALUES ('Pidgeot', 'Normal');
INSERT INTO CONTAIN VALUES ('Rattata', 'Normal');
INSERT INTO CONTAIN VALUES ('Raticate', 'Normal');

Table EVOLVES_USING:

INSERT INTO EVOLVES_USING VALUES ('Bulbasaur', 'Ivysaur');
INSERT INTO EVOLVES_USING VALUES ('Charmander', 'Charmeleon');
INSERT INTO EVOLVES_USING VALUES ('Squirtle', 'Wartortle');
INSERT INTO EVOLVES_USING VALUES ('Caterpie', 'Metapod');
INSERT INTO EVOLVES_USING VALUES ('Weedle', 'Kakuna');
INSERT INTO EVOLVES_USING VALUES ('Pidgey', 'Pidgeotto');
INSERT INTO EVOLVES_USING VALUES ('Rattata', 'Raticate');
INSERT INTO EVOLVES_USING VALUES ('Spearow', 'Fearow');

INSERT INTO EVOLVES_USING VALUES ('Ekans', 'Arbok');
INSERT INTO EVOLVES_USING VALUES ('Pichu', 'Pikachu');
INSERT INTO EVOLVES_USING VALUES ('Sandshrew', 'Sandslash');
INSERT INTO EVOLVES_USING VALUES ('Nidoran♀', 'Nidorina');
INSERT INTO EVOLVES_USING VALUES ('Nidoran♂', 'Nidorino');
INSERT INTO EVOLVES_USING VALUES ('Cleffa', 'Clefairy');
INSERT INTO EVOLVES_USING VALUES ('Vulpix', 'Ninetales');
INSERT INTO EVOLVES_USING VALUES ('Igglybuff', 'Jigglypuff');
INSERT INTO EVOLVES_USING VALUES ('Zubat', 'Golbat');
INSERT INTO EVOLVES_USING VALUES ('Oddish', 'Gloom');
INSERT INTO EVOLVES_USING VALUES ('Paras', 'Parasect');
INSERT INTO EVOLVES_USING VALUES ('Venonat', 'Venomoth');

Table Contains:
INSERT INTO CONTAIN VALUES ('Bulbasaur', 'Grass');
INSERT INTO CONTAIN VALUES ('Ivysaur', 'Grass');
INSERT INTO CONTAIN VALUES ('Venusaur', 'Grass');
INSERT INTO CONTAIN VALUES ('Charmander', 'Fire');
INSERT INTO CONTAIN VALUES ('Charmeleon', 'Fire');
INSERT INTO CONTAIN VALUES ('Charizard', 'Fire');
INSERT INTO CONTAIN VALUES ('Squirtle', 'Water');
INSERT INTO CONTAIN VALUES ('Wartortle', 'Water');
INSERT INTO CONTAIN VALUES ('Blastoise', 'Water');
INSERT INTO CONTAIN VALUES ('Caterpie', 'Bug');
INSERT INTO CONTAIN VALUES ('Metapod', 'Bug');
INSERT INTO CONTAIN VALUES ('Butterfree', 'Bug');
INSERT INTO CONTAIN VALUES ('Weedle', 'Bug');
INSERT INTO CONTAIN VALUES ('Kakuna', 'Bug');
INSERT INTO CONTAIN VALUES ('Beedrill', 'Bug');
INSERT INTO CONTAIN VALUES ('Pidgey', 'Normal');
INSERT INTO CONTAIN VALUES ('Pidgeotto', 'Normal');
INSERT INTO CONTAIN VALUES ('Pidgeot', 'Normal');
INSERT INTO CONTAIN VALUES ('Rattata', 'Normal');
INSERT INTO CONTAIN VALUES ('Raticate', 'Normal');

SQL Multi-Table User Queries Examples

The queries listed below are not the only queries that a user can run on our database. In our UI, the user may implement any query they write to use our database. The ones listed below are given example queries that a user can run on our database. For example, the first query can be modified by the user to show the total number of each type in Generation 2 instead of 1 or they can even do a different search criteria such as looking through a certain Game Title such as those Pokemon from ‘Pokemon Diamond and Pearl’.

Table 3: Multi-Table Queries

SQL Statement	Purpose
SELECT Contain.Type, Count(Pokemon.Pname) as Total FROM Game, Pokemon, Contain WHERE Pokemon.Gname = Game.Gname AND Pokemon.Pname = Contain.Pname AND Pokemon.Previous_Evo IS NOT NULL AND Game.Generation = 1 GROUP BY Contain.Type;	English: Show the total number of Each Type Pokemon from Generation 1 who have a Previous Evolution Involved Tables: Game, Pokemon, CONTAIN Scenario: For the casual player who loves the 1st Generation of Pokemon. They're looking for a Pokemon with an uncommon type to add to their team.
SELECT Game.Generation, Count(*) as 'Total Pokemon' FROM Game JOIN Pokemon on Game.Gname = Pokemon.Gname JOIN CONTAIN on Pokemon.Pname = Contain.Pname Where CONTAIN.Type = 'Grass' GROUP BY Game.Generation ORDER BY Game.Generation;	English: Show the total number Pokemon in each generation who are Grass types. Involved Tables: Game, Pokemon, CONTAIN, Type Scenario: A casual player who's curious about how common one of the starter types is in each generation
SELECT Pokedex.Nation_Num as 'National Number', Pokemon.Pname as 'Pokemon Name' FROM Game JOIN Pokemon on Game.Gname = Pokemon.Gname JOIN Pokedex on Pokemon.Pname = Pokedex.Pname WHERE Pokemon.Evo_Stage = 3 AND Pokemon.Pname In (SELECT Pokemon.Pname FROM Pokemon JOIN Uses on Pokemon.Pname = Uses.Pname JOIN Moves on Uses.Move_Name = Moves.Move_Name JOIN Type on Moves.Type = Type.Type_1 WHERE Type.Effective like '%Water%') ORDER BY Pokedex.Nation_Num ASC;	English: Show all the Pokemon who are a stage 3 Evolution and can learn a move strong against Water Types Involved Tables: Game, Pokemon, Pokedex, USES, Moves Scenario: For a competitive player who wants to fill a hole in their team that can outmatch any Water type Pokemon

SELECT Pokedex.Nation_Num as 'National Number', Pokemon.Pname as 'Pokemon Name', Pokemon.Gname as 'Game', Game.Generation, Game.Region, Type.Type_1 as Type FROM Game JOIN Pokemon on Game.Gname = Pokemon.Gname JOIN Pokedex on Pokemon.Pname = Pokedex.Pname JOIN Contain on CONTAIN.Pname = Pokemon.Pname JOIN Type on CONTAIN.Type = Type.Type_1 WHERE Pokemon.Pname = 'Ralts';	English: Show all the defining characteristics about Ralts (Type, Game Debut, etc.) Involved Tables: Game, Pokemon, Pokedex, CONTAIN, Type Scenario: For a casual player whose favorite Pokemon is Ralts and they want to learn as much about them as possible.
SELECT Game.Generation, Pokemon.Pname as 'Pokemon Name', Contain.Type FROM Game JOIN Pokemon on Game.Gname = Pokemon.Gname JOIN Contain on Pokemon.Pname = Contain.Pname WHERE Game.Generation < 5 AND Pokemon.Rarity like 'Legendary' AND Pokemon.Pname in (Select Pokemon.Pname FROM Uses JOIN Moves on Uses.Move_Name = Moves.Move_Name WHERE Moves.Power > 10) ORDER BY Game.Generation, Pokemon.Pname ASC;	English: Show all Legendary Pokemon from before Generation 5 who can learn a move with over 10 power Involved Tables: Game, Pokemon, USES, Contains, Moves Scenario: A competitive player who wants to form a Legendaries only team using Pokemon from their childhood.
SELECT DISTINCT Pokedex.Nation_Num as 'National Number', Pokedex.Region_Num as 'Region Number', Pokemon.Gname as 'Game', Pokemon.Pname as 'Pokemon Name', Mechanic.Next_Evo as 'Next Evo' FROM Pokedex JOIN Pokemon on Pokedex.Pname = Pokemon.Pname JOIN Evolves_Using on Pokemon.Pname = EVOLVES_USING.Pname JOIN Mechanic on EVOLVES_USING.Next_Evo = Mechanic.Next_Evo WHERE (Pokemon.Gname = 'Pokemon Diamond and Pearl' OR Pokemon.Gname = 'Pokemon X and Y') and Mechanic.Level < 55 ORDER BY Pokedex.Nation_Num ASC;	English: Show all the Pokemon who debuted in Pokemon Diamond and Pearl or Pokemon X and Y and start evolving before level 55 Involved Tables: Pokedex, Pokemon, EVOLVES_USING, Mechanic Scenario: For a beginner player who has just started playing Diamond and Pearl or Pokemon X and Y and needs to know what Pokemon he can evolve to the next level before finishing the game

SELECT DISTINCT Pokedex.Nation_Num, Game.Gname, Game.Region, Pokemon.Pname FROM Pokedex JOIN Pokemon on Pokedex.Pname = Pokemon.Pname JOIN Game on Game.Gname = Pokemon.Gname JOIN Uses on USES.Pname = Pokemon.Pname JOIN Moves on Moves.Move_Name = Uses.Move_Name WHERE Game.Generation = 6 and Moves.Type = 'Psychic' ORDER BY Pokedex.Nation_Num ASC;	English: Show the National Number, Game name, Region, and Pokemon Name from all the Generation 6 Pokemon who can learn a Psychic type move Involved Tables: Pokedex, Pokemon, Game, Uses, Moves Scenario: For a competitive user playing “Pokemon X or Pokemon Y” who wants to use a new Pokemon from that Generation who can beat Poison Types
SELECT Pokedex.Nation_Num, Pokemon.Pname, Evolves_Using.Next_Evo as 'Next Evo' FROM Pokedex JOIN Pokemon on Pokedex.Pname = Pokemon.Pname JOIN Game on Pokemon.Gname = Game.Gname JOIN Contain on CONTAIN.Pname = Pokemon.Pname JOIN Type on CONTAIN.Type = Type.Type_1 JOIN EVOLVES_USING on EVOLVES_USING.Pname = Pokemon.Pname WHERE Type.Type_1 = 'Steel' or Type.Type_1 = 'Fairy' and Game.Release_Date > '2001-12-31' ORDER BY Pokedex.Nation_Num;	English: Show the name, number, and next evolution of every Steel or Fairy Type Pokemon released after 2001 who can still evolve Involved Tables: Pokedex, Pokemon, Game, CONTAIN, Type, EVOLVES_USING Scenario: For a curious player who wants to use more recent Steel or Fairy type Pokemon for their next team.
SELECT Pokedex.Nation_Num as 'National Number', Pokemon.Pname as 'Pokemon Name', Game.Generation, Contain.Type, Evolves_Using.Next_Evo, Mechanic.Item FROM Pokedex JOIN Pokemon on Pokemon.Pname = Pokedex.Pname JOIN Game on Pokemon.Gname = Game.Gname JOIN EVOLVES_USING on EVOLVES_USING.Pname = Pokemon.Pname JOIN Mechanic on EVOLVES_USING.Next_Evo = Mechanic.Next_Evo JOIN CONTAIN on	English: Show the National Number, Name, Generation, Type, Next Evolution, and Item of every Pokemon who evolves using an item Involved Tables: Pokedex, Pokemon, Game, CONTAIN, Type, EVOLVES_USING, Mechanic Scenario: For a user who is wondering if maybe his Pokemon hasn’t evolved yet because it

Pokemon.Pname = CONTAIN.Pname WHERE Mechanic.Item is NOT NULL ORDER by Pokedex.Nation_Num ASC;	needs an item
SELECT Pokemon.Pname, Game.Region, Contain.Type, Mechanic.Next_Evo, Mechanic.Level, Mechanic.Item, Mechanic.Time, Mechanic.Gimmick FROM Pokedex JOIN Pokemon on Pokemon.Pname = Pokedex.Pname JOIN Game on Pokemon.Gname = Game.Gname JOIN EVOLVES_USING on EVOLVES_USING.Pname = Pokemon.Pname JOIN Mechanic on EVOLVES_USING.Next_Evo = Mechanic.Next_Evo JOIN CONTAIN on Pokemon.Pname = CONTAIN.Pname WHERE Pokedex.Nation_Num = 52 or Pokedex.Nation_Num = 236 or Pokedex.Nation_Num = 743;	English: Show the Pokemon name, Region, Type(s), and Mechanic attributes of Pokemon Number 52, 236, and 743 Involved Tables: Pokedex, Pokemon, Game, EVOLVES_USING, Mechanic, CONTAIN Scenario: For a user who wants to know how to evolve certain Pokemon in their party.
SELECT Pokedex.Nation_Num, Pokemon.Pname, Contain.Type FROM Pokemon JOIN CONTAIN on Pokemon.Pname = CONTAIN.Pname JOIN Game on Pokemon.Gname = Game.Gname JOIN Pokedex on Pokedex.Pname = Pokemon.Pname WHERE Game.Gname = 'Pokemon Scarlet and Violet' and Pokemon.Pname in (Select Contain.Pname From Contain Join Type on Contain.Type = Type.Type_1 Where Type.Weak like '%Fairy%') ORDER BY Pokedex.Nation_Num ASC;	English: Show the National Number, Pokemon Name, and Type(s) of all the Pokemon who debuted in 'Pokemon Scarlet and Violet' who are weak to Fairy Types Involved Tables: Pokedex, Pokemon, Game, CONTAIN, Type Scenario: For a user who isn’t knowledgeable about Pokemon Scarlet and Violet but wants to avoid using Pokemon weak to Fairy types.
SELECT Distinct Pokedex.Region_Num, Pokemon.Pname FROM Pokedex JOIN Pokemon on Pokedex.Pname = Pokemon.Pname JOIN CONTAIN on CONTAIN.Pname = Pokemon.Pname JOIN USES on Pokemon.Pname = USES.Pname JOIN	English: Show the Regional Number and name of all the Normal/Flying and Fire/Flying dual-type Pokemon who can learn a 'Physical' Type move Involved Tables:

Moves on USES.Move_Name = Moves.Move_Name WHERE (CONTAIN.Type = 'Normal' OR CONTAIN.Type = ‘Fire’) and Moves.Damage_Type = 'Physical' and Pokemon.Pname in (SELECT Pokemon.Pname FROM Pokemon JOIN CONTAIN on CONTAIN.Pname = Pokemon.Pname WHERE CONTAIN.Type = ‘Flying');	Pokedex, Pokemon, CONTAIN, USES, Moves Scenario: For a user who wants to figure out what Normal/Flying or Fire/Flying dual-type Pokemon he can use that learn Physical moves.

Data Testing Queries

Some, not all, of our testing queries were done with the help of AI. As such, we’ve included both ChatGPT and BingAI in our References section at the end of our document.

Table Pokedex:

Uniqueness constraint violation because Pokedex entry 1 and 2 are already filled in the database. Another violation is also present in that “Ralts” is assigned two different dex entries.

```
--*Ralts should already be in table so there's also another violation here*
INSERT into Pokedex VALUES (1, NULL, 'Ralts');
INSERT into Pokedex VALUES (2, NULL, 'Ralts');
```

A violation of Domain Constraints. There are wrong values in both queries as the format should be integer, integer (can be null), and string.

```
INSERT into Pokedex VALUES ('a', NULL, 'Ralts');
INSERT into Pokedex VALUES (21, NULL, 10);
```

A violation of Domain Constraints. The National Number can’t be null nor can the referenced pokemon be null. A number must always be associated with a Pokemon already existing in the database.

```
--Testing for NOT NULL violations
```

```
INSERT into Pokedex VALUES (NULL, 10, 'Ralts');
INSERT into Pokedex VALUES (1000, 10, NULL);
INSERT into Pokedex VALUES (NULL, 20, 'Kirlia');
INSERT into Pokedex VALUES (123, 10, NULL);
```

This is a violation of the Referential integrity constraint. “Mario”, “Luigi”, “Peach” and “Bowser” are not anywhere in the Pokemon table. Valid values would be names such as “Lugia” or “Piplup”

```
--Testing for FK Pname not in another table
INSERT into Pokedex VALUES (1011, 12, 'Mario');
INSERT into Pokedex VALUES (1011, 12, 'Luigi');
INSERT into Pokedex VALUES (1011, 12, 'Peach');
INSERT into Pokedex VALUES (1011, 12, 'Bowser');
```

Clear violation of Domain Constraints. It is not possible to have a National Pokedx number less than or equal to 0 or greater than 1018

```
--Test for out of bounds nums
UPDATE Pokedex SET Nation_Num = 5000 where Nation_Num = 20;
UPDATE Pokedex SET Nation_Num = 0 where Nation_Num = 20;
UPDATE Pokedex SET Region_Num = 5000 where Nation_Num = 10;
UPDATE Pokedex SET Region_Num = 0 where Nation_Num = 10;
```

Table Game:

These queries violate the PK constraint of the Game Table. There must always be a value associated with Gname (attribute 1) to create a valid record in the table

```
--Test for INSERTING with missing PK
INSERT into Game VALUES (NULL, 1, 'Kanto', '1998-09-28');
INSERT into Game VALUES (NULL, 2, 'Johto', '2001-12-20');
INSERT into Game VALUES (NULL, 3, 'Kanto', '1998-09-28');
INSERT into Game VALUES (NULL, 4, 'Hoenn', '2001-12-20');
```

A violation of the Domain Constraints. Each query has one of two attributes that are of the wrong data type. It should be string, integer, string, and date.

```
--Test for wrong data types
INSERT into Game VALUES ('Pokemon Red and Blue', 2, 3, '1998-10-14');
INSERT into Game VALUES (21, 'Red', 3, 2002);
INSERT into Game VALUES ('Po', 2, 3, '1998-10-14');
INSERT into Game VALUES (-1, 'Red', 3, 2002);
```

Another violation of Domain Constraints. None of these are one of the nine valid regions (such as ‘Alola’ or ‘Unova’)

```
--Test for not valid region
INSERT into Game VALUES ('Pokemon Red and Blue', 1, 'Cali', '1998-09-28');
INSERT into Game VALUES ('Pokemon Emerald', 3 'Texas', '2005-05-01');
INSERT into Game VALUES ('Pokemon Red and Blue', 1, 'Pizza', '1998-09-28');
INSERT into Game VALUES ('Pokemon Emerald', 3 'Yellow', '2005-05-01');
INSERT into Game VALUES ('Pokemon Red and Blue', 1, 'Soda', '1998-09-28');
INSERT into Game VALUES ('Pokemon Emerald', 3 'Bluy', '2005-05-01');
INSERT into Game VALUES ('Pokemon Red and Blue', 1, 'Poppy', '1998-09-28');
INSERT into Game VALUES ('Pokemon Emerald', 3 'Moons', '2005-05-01');
```

Yet another violation of Domain Constraints. None are a valid integer for a generation. They should be a value ranging from 1-9.

```
--Test for not valid generation number
INSERT into Game VALUES ('Pokemon Sun and Moon', 11, 'Alola', '2016-11-18');
INSERT into Game VALUES ('Pokemon Sword and Shield', 0, 'Galar' '2019-11-15');
INSERT into Game VALUES ('Pokemon Scarlet and Violet', -1, 'Paldea', '2022-11-18');
INSERT into Game VALUES ('Pokemon Sun and Moon', 30, 'Alola', '2016-11-18');
INSERT into Game VALUES ('Pokemon Sword and Shield', 200, 'Galar' '2019-11-15');
INSERT into Game VALUES ('Pokemon Scarlet and Violet', -1, 'Paldea', '2022-11-18');
```

A violation of the Entity Integrity Constraints are shown above. We've already tested for NULL in the PK so these test for NULL in the Generation attribute. By design it is not allowed to be null.

```
--Test for NOT NULL violation
INSERT into Game VALUES ('Pokemon Platinum', NULL, 'Sinnoh', '2009-03-22');
INSERT into Game VALUES ('Pokemon Ultra Sun and Ultra Moon', NULL, 7, '2017-11-17');
```

This one is not a violation constraint but merely a test to see if adding another game would be accepted into the Game Table. These are valid queries that should pass through

```
--Next ones should work (date is optional) *Just for testing this game doesn't actually exist*
SET SQL_SAFE_UPDATES = 0;
INSERT into Game VALUES ('Pokemon Legends: Celebi', 9, 'Sinnoh', NULL);
DELETE from Game where Gname like '%Celebi%';
INSERT into Game VALUES ('Pokemon Legends: Kyurem', 9, 'Unova', NULL);
DELETE from Game where Gname like '%Kyurem%';
SET SQL_SAFE_UPDATES = 1;
```

Each of these queries violate the Uniqueness Constraint. These games have already been previously added into the database so there should be no repeats and these

queries should not pass through. This is even considering if they have different attributes.

```
--Test for not valid uniqueness
INSERT into Game VALUES ('Pokemon X and Y', 6, 'Kalos', NULL);
INSERT into Game VALUES ('Pokemon X and Y', 9, 'Kalos', NULL);
INSERT into Game VALUES ('Pokemon X and Y', 9, 'Unova', NULL);
INSERT into Game VALUES ('Pokemon X and Y', 6, 'Kalos', '1999-10-11');
INSERT into Game VALUES ('Pokemon Omega Ruby and Alpha Sapphire', 6, 'Hoenn', NULL);
```

Table Moves:

Each of these moves are violating the Uniqueness constraint as they are already present in the table.

```
--Testing for Uniqueness Violation
--*Assuming these are already in table*
INSERT into Moves VALUES ('Absorb', 'Special', 25, 50, 'Rock');
INSERT into Moves VALUES ('Flame Wheel', 'Special', 25, 50, 'Rock');
INSERT into Moves VALUES ('Splash', 'Special', 25, 50, 'Rock');
INSERT into Moves VALUES ('Leer', 'Special', 25, 50, 'Rock');
INSERT into Moves VALUES ('Attract', 'Special', 25, 50, 'Rock');
```

The Domain Constraint is being violated here with each query. It is not possible to have a Power less than 0 or greater than 250.

```
--Testing for out of bounds values
UPDATE Moves SET Power = -1 WHERE Move_Name = 'Absorb';
UPDATE Moves SET Power = 1000 WHERE Move_Name = 'Absorb';
UPDATE Moves SET Power = -10 WHERE Move_Name = 'Absorb';
UPDATE Moves SET Power = 3000 WHERE Move_Name = 'Absorb';
UPDATE Moves SET Power = -1.2 WHERE Move_Name = 'Absorb';
UPDATE Moves SET Power = 1000.4 WHERE Move_Name = 'Absorb';
```

Similar to the last violation above (Domain Constraint Violation), it is also not possible to have PP less than 0 or greater than 250.

```
UPDATE Moves Set PP = -3 WHERE Move_Name = 'Absorb';
UPDATE Moves Set PP = 251 WHERE Move_Name = 'Absorb';
UPDATE Moves Set PP = -1 WHERE Move_Name = 'Absorb';
UPDATE Moves Set PP = 300 WHERE Move_Name = 'Absorb';
```

This one is a Referential Integrity Constraint. None of these types are present in the Type Table of our database.

```
/*Testing for Type not present in tables*/
```

```
UPDATE Moves SET Type = 'Soul' WHERE Move_Name = 'Absorb';
UPDATE Moves SET Type = 'Mental' WHERE Move_Name = 'Absorb';
UPDATE Moves SET Type = 'Cosmic' WHERE Move_Name = 'Absorb';
UPDATE Moves SET Type = 'Stellar' WHERE Move_Name = 'Absorb';
```

A Domain Constraint violation as none of these damage types are valid types. Only “Physical”, “Special”, and “Status” are accepted data.

```
/*Testing for invalid Damage_Type*/
UPDATE Moves SET Damage_Type = 'Mental' WHERE Move_Name = 'Absorb';
UPDATE Moves SET Damage_Type = 'Soul' WHERE Move_Name = 'Absorb';
UPDATE Moves SET Damage_Type = 'Cosmic' WHERE Move_Name = 'Absorb';
UPDATE Moves SET Damage_Type = 'Stellar' WHERE Move_Name = 'Absorb';
```

Another Domain Constraint Violation. These violations take the form of being the wrong type of data.

```
/*Testing for wrong data types*/
UPDATE Moves SET Damage_Type = 1 WHERE Move_Name = 'Absorb';
UPDATE Moves SET PP = 'twenty' WHERE Move_Name = 'Absorb';
UPDATE Moves SET Power = 'a lot' WHERE Move_Name = 'Absorb';
UPDATE Moves SET Damage_Type = 67 WHERE Move_Name = 'Absorb';
UPDATE Moves SET PP = 'forty' WHERE Move_Name = 'Absorb';
UPDATE Moves SET Power = 'a little' WHERE Move_Name = 'Absorb';
```

Table Type

Constraints violated in these queries are all because these types already exist in the database.

```
/*Testing for Uniqueness Violation*/
INSERT into Type VALUES ('Rock', NULL, NULL, NULL);
INSERT into Type VALUES ('Fairy', NULL, NULL, NULL);
INSERT into Type VALUES ('Steel', NULL, NULL, NULL);
INSERT into Type VALUES ('Dragon', NULL, NULL, NULL);
INSERT into Type VALUES ('Fighting', NULL, NULL, NULL);
INSERT into Type VALUES ('Water', NULL, NULL, NULL);
INSERT into Type VALUES ('Fire', NULL, NULL, NULL);
INSERT into Type VALUES ('Grass', NULL, NULL, NULL);
```

None of these queries are one of the 18 valid Pokemon typings. Correct examples would be “Rock”, “Ground”, or “Water”

```
/*Testing for invalid type*/
INSERT into Type VALUES ('Sound', NULL, NULL, NULL);
INSERT into Type VALUES ('Quantum', NULL, NULL, NULL);
INSERT into Type VALUES ('Imaginary', NULL, NULL, NULL);
INSERT into Type VALUES ('2019-11-02', NULL, NULL, NULL);
```

A Type requires you to have primary typing. The other three attributes may be null but not the primary typing. You need to have some primary typing to act as a primary key or else it becomes a Key Constraint.

```
/*Testing for no PK*/
INSERT into Type VALUES (NULL, 'Rock', NULL, NULL);
INSERT into Type VALUES (NULL, NULL, 'Fairy', NULL);
INSERT into Type VALUES (NULL, NULL, 'Rock,Fairy', NULL);
INSERT into Type VALUES (NULL, NULL, NULL, 'Ground');
INSERT into Type VALUES (NULL, NULL, 'Water,Fire', 'Grass');
```

Numbers such as 2 or 3 are not valid data types for the attribute “Effective”. Same with Type_1 along with Type_1 not being able to be a non-existent Type.

```
--Testing for wrong data types
UPDATE Type SET Effective = 2 WHERE Type_1 = 'Rock';
UPDATE Type SET Effective = 'Rock' WHERE Type_1 = 'Sound';
UPDATE Type SET Effective = 2 WHERE Type_1 = 'Fairy';
UPDATE Type SET Effective = 'Rock' WHERE Type_1 = 3;
```

Database Access

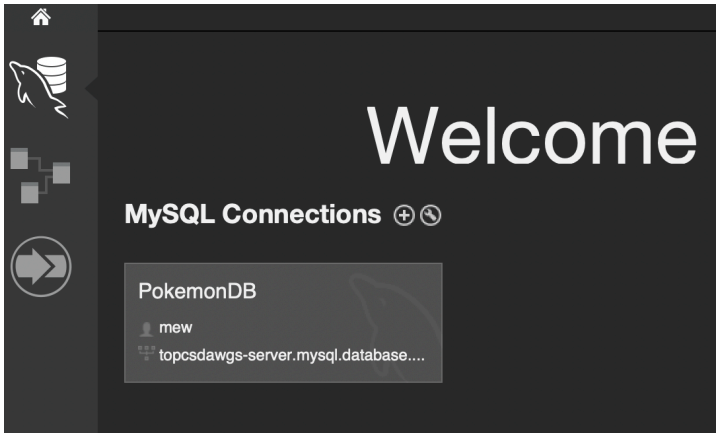
While we do provide all the sql files necessary to create and populate the database, we are using Microsoft Azure to host our database online. We have given access to all graders and Professor Parsons so viewing the database should not be an issue.

Here is the link for easy access to the database:
https://portal.azure.com/?quickstart=true#@cloud.washington.edu/resource/subscriptions/d59d7608-2011-4c37-b80c-0f8784ffc78a/resourceGroups/Pokemon/providers/Microsoft.DBforMySQL/flexibleServers/topcsdawgs-server/databases

If you wish to tinker with the database and view the tables yourself, we recommend you use MySQL workbench and connect to the database through that application. You can download it through this link down below:
https://dev.mysql.com/downloads/workbench/

Once you have it set-up on your computer, you can establish a new connection by pressing the + button next to **MySQL Connections**.

Figure 8: MySQL Workbench



It'll prompt for a username and password to access the database with MySQLWorkbench. Input **mew** for user and the password is **Pokemonz123**. Also for hostname input **topcsdawgs-server.mysql.database.azure.com**.

Here is the official MySQL Workbench steps for connecting to the database in case you want to follow these steps instead:

Figure 9: Official MySQL Workbench Steps

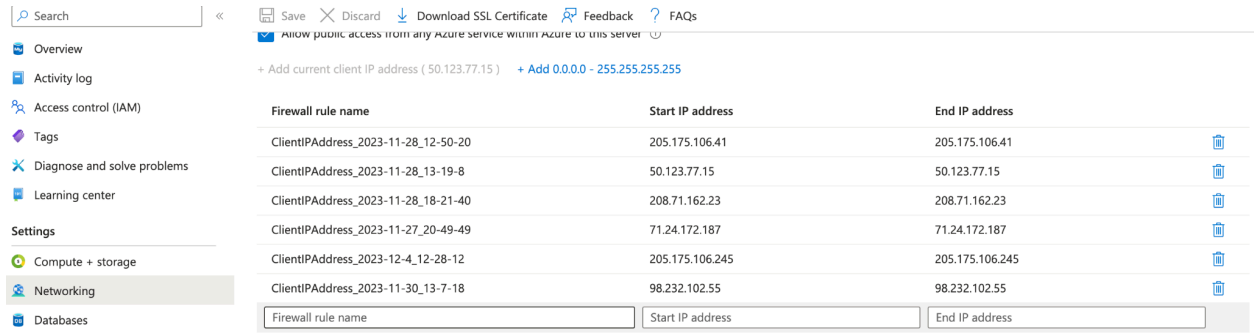
MySQL Workbench

To connect with MySQL workbench client, follow the steps below.

1. Click the + symbol in the **MySQL Connections** tab to add a new connection.
2. Enter a name for the connection in the **Connection name** field.
3. Select **Standard (TCP/IP)** as the Connection Type.
4. Enter **topcsdawgs-server.mysql.database.azure.com** in hostname field.
5. Enter **mew** as username and then enter your **Password**.

If the connection fails, it may be because you need to add your IP address to the **Networking** tab in Azure. Go over to the Azure link we have and navigate to **Network** then just press **+Add current client IP address.**

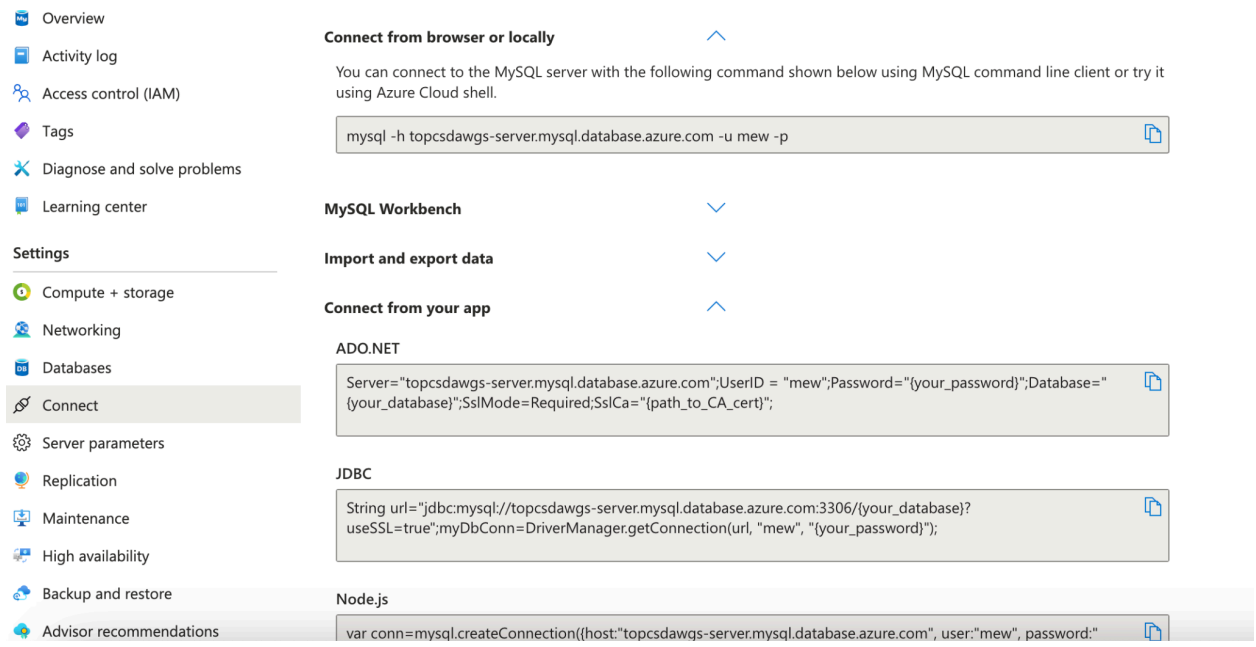
Figure 10: Azure Database Networking Tab.



If you wish to create the database yourself, that is fine too. All the sql files have been included as a zip file alongside this report **(PokemonDBTopCSDawgs.zip).**

In the connections tab, there are also more options recommended by Azure on how to connect to the database in the **Connect** page. You may also attempt any of these methods if you so choose.

Figure 11: Other Connection Options



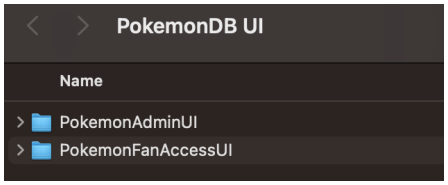
UI Usage:

We did manage to make a UI for our database. There's two different versions. One is called "PokemonFanAccess" which is for regular users of our database. These users have read-only permission so they are only allowed to do SELECT Queries. This user is able to choose out of eight queries. Options 1-6 are premade queries with no modifications allowed. Whereas option 7 allows the user to create their own query which follows MySQL syntax. Lastly, there's option 8 which simply allows the user to exit the program.

The second version is called "PokemonAdminAccess" which was made for people who can directly interact with the database. This user is able to use the select just like the PokemonFanAccess; but with the rest of the functionalities as well. The functionalities include creating, inserting, updating, deleting, and dropping tables.

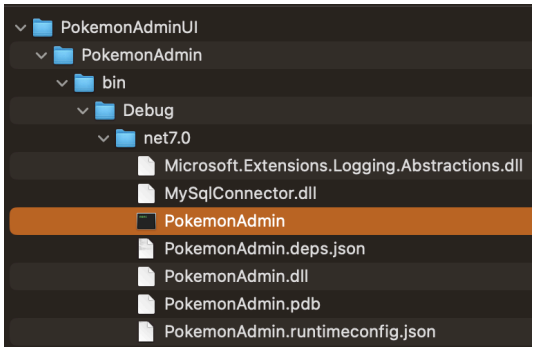
To access either one of these UIs, open up the zip folder labeled "PokemonDB UI"

Figure 11: PokemonDB UI



Then, open the Terminal Application called "Pokemon Admin" or "PokemonFanAccess".

Figure 12: PokemonAdmin / FanAccess UI



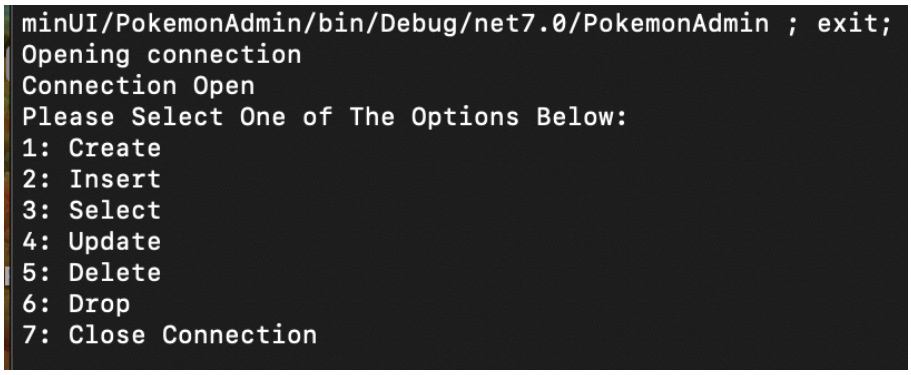
If you run into a .NET Error Message, make sure to follow the download instructions your Terminal provides through the link (Ex: Figure 13). Make sure to download .NET 7 (Any other version may not work).

Figure 13: .NET Error Message

```
To install missing framework, download:
https://aka.ms/dotnet-core-applaunch?framework=Microsoft.NETCore.App&framework_v
ersion=7.0.0&arch=x64&rid=osx-x64&os=osx.13
```

Depending on which User Application you open, there are a variety of options to choose from. Figures 14 and 15 show the main menu for “PokemonAdmin” and “PokemonFanAccess” respectively.

Figure 14: “PokemonAdmin”



Depending on which option you choose in ‘PokemonAdmin” you’ll have to complete the query selected. For example, choosing option 1 (Create) means that you’ll finish the statement with the name of the table and its attributes (much like line 2 in Figure 15). Similar formats are also in the other options (Insert, Select, Update, etc.). For example, Drop also asks that you complete the query for a drop. An example of this is given in line 17 of Figure 15.

Figure 15: “PokemonAdmin” Query Example

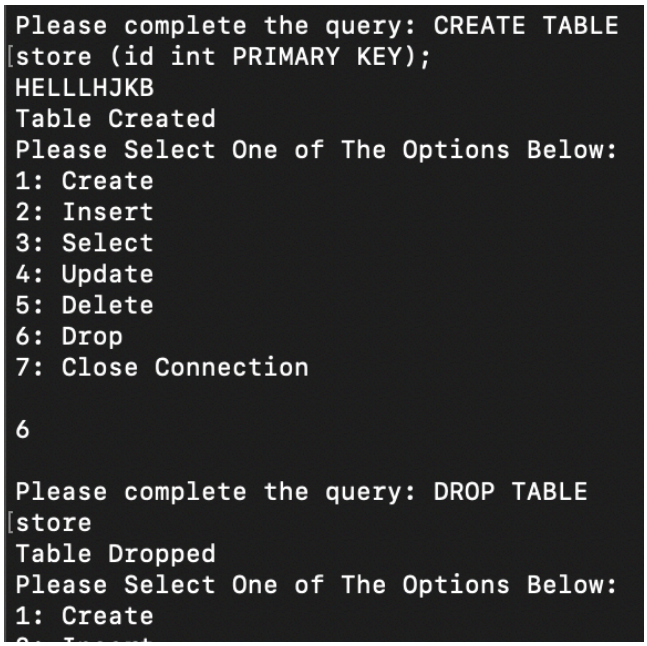
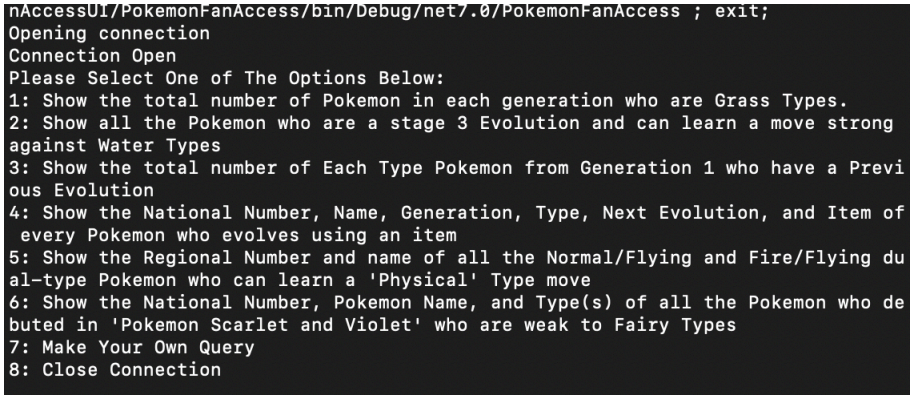


Figure 16: “PokemonFanAccess”



Choosing options 1-6 in “PokemonFanAccess” will run the hard-programmed query corresponding to the prompt while option 8 will close the connection.

Option 7 allows the user to input their own selection prompts to the database. The UI will run any prompt given that it’s a SELECT prompt and properly written (i.e SELECT * from Pokemon —>Correct, Select from Pokemon —-> Incorrect);

From here make your selection and enjoy using the database!

Database Creation and Insertion Order:

Here is the recommended order to use the sql files to ensure that no violations are created:

1) CreateTables.sql
2) Type.sql
3) GameData.sql
4) Pokemon.sql
5) Stats.sql
6) Pokedex.sql
7) Moves.sql
8) Mechanic.sql
9) Uses.sql
10) EvolvesUsing.sql
11) Contains.sql

******If there are any issues in accessing the database, feel free to message any team member through our UW emails, Canvas, or Discord******

Demo Screenshots:

In our demo, we plan to show off our PokemonFanAccess UI. This one has built in queries that'll make it fast for showing off our project. PokemonFanAccess and PokemonAdmin both have different privileges and UIs but our main user will be the PokemonFanAccess so we've included that here. Figure 17 shows the main menu of the PokemonFanAccess UI.

Figure 17: PokemonFanAccess UI Demo

```
nAccessUI/PokemonFanAccess/bin/Debug/net7.0/PokemonFanAccess ; exit;
Opening connection
Connection Open
Please Select One of The Options Below:
1: Show the total number of Pokemon in each generation who are Grass Types.
2: Show all the Pokemon who are a stage 3 Evolution and can learn a move strong
against Water Types
3: Show the total number of Each Type Pokemon from Generation 1 who have a Previ
ous Evolution
4: Show the National Number, Name, Generation, Type, Next Evolution, and Item of
 every Pokemon who evolves using an item
5: Show the Regional Number and name of all the Normal/Flying and Fire/Flying du
al-type Pokemon who can learn a 'Physical' Type move
6: Show the National Number, Pokemon Name, and Type(s) of all the Pokemon who de
buted in 'Pokemon Scarlet and Violet' who are weak to Fairy Types
7: Make Your Own Query
8: Close Connection
```

Selecting option 4 will run the corresponding on our database and display the results on the console as shown in Figure 18:

Figure 18: Option 4 Results

```
National Number, Pokemon Name, Generation, Type, Next Evo, Item
(25, Pikachu, 1, Electric, Raichu, thunder-stone)
(30, Nidorina, 1, Poison, Nidoqueen, moon-stone)
(33, Nidorino, 1, Poison, Nidoking, moon-stone)
(35, Clefairy, 1, Fairy, Clefable, moon-stone)
(37, Vulpix, 1, Fire, Ninetales, fire-stone)
(39, Jigglypuff, 1, Normal, Wigglytuff, moon-stone)
(39, Jigglypuff, 1, Fairy, Wigglytuff, moon-stone)
(44, Gloom, 1, Poison, Vileplume, leaf-stone)
(44, Gloom, 1, Grass, Vileplume, leaf-stone)
(58, Growlithe, 1, Fire, Arcanine, fire-stone)
(61, Poliwhirl, 1, Water, Poliwrath, water-stone)
(70, Weepinbell, 1, Grass, Victreebel, leaf-stone)
(70, Weepinbell, 1, Poison, Victreebel, leaf-stone)
(82, Magneton, 1, Electric, Magnezone, thunder-stone)
(82, Magneton, 1, Steel, Magnezone, thunder-stone)
(90, Shellder, 1, Water, Cloyster, water-stone)
(95, Onix, 1, Rock, Steelix, metal-coat)
(95, Onix, 1, Ground, Steelix, metal-coat)
(102, Exeggcute, 1, Grass, Exeggutor, leaf-stone)
(102, Exeggcute, 1, Psychic, Exeggutor, leaf-stone)
(112, Rhydon, 1, Rock, Rhyperior, Protector)
(112, Rhydon, 1, Ground, Rhyperior, Protector)
(117, Seadra, 1, Water, Kingdra, dragon-scale)
(120, Staryu, 1, Water, Starmie, water-stone)
(123, Scyther, 1, Bug, Scizor, metal-coat)
(123, Scyther, 1, Flying, Scizor, metal-coat)
(125, Electabuzz, 1, Electric, Electivire, electririzer)
(126, Magmar, 1, Fire, Magmortar, magmarizer)
(133, Eevee, 1, Normal, Vaporeon, water-stone)
(137, Porygon, 1, Normal, Porygon2, upgrade)
(176, Togetic, 2, Flying, Togekiss, shiny-stone)
```

This isn't the complete table displayed in the UI but it manages to show off enough to see how the results are formatted. These results are made by connected to our MySQL database which is hosted on Azure and running the query displayed in Figure 19.

FIgure 19: Option 4 Query

```
/*Show the National Number, Name, Generation,
Type, Next Evolution, and Item of every Pokemon who evolves using an item*/

SELECT Pokedex.Nation_Num as 'National Number', Pokemon.Pname as 'Pokemon Name', Game.Generation,
Contain.Type, Evolves_Using.Next_Evo, Mechanic.Item
FROM Pokedex JOIN Pokemon on Pokemon.Pname = Pokedex.Pname JOIN Game on Pokemon.Gname = Game.Gname
JOIN EVOLVES_USING on EVOLVES_USING.Pname = Pokemon.Pname JOIN Mechanic on EVOLVES_USING.Next_Evo = Mechanic.Next_Evo
JOIN CONTAIN on Pokemon.Pname = CONTAIN.Pname
WHERE Mechanic.Item is NOT NULL
ORDER by Pokedex.Nation_Num ASC;
```

Final Database Design Reflection: Challenges and Possible Improvements

Challenges:

The biggest challenge to this database project was definitely the data creation. We did have the use of an API to extract data from but we encountered difficulties in extracting this data in the correct format. Eventually, a different team member was able to extract the API data for various tables using Python to create text files for insertions. Despite making this tremendous leap in progress, we kept having to redo our Python files as we kept realizing some pitfalls in our data collection. For example, we realized that the type effectiveness (weak, strong, no damage) were incomplete in our insertions. Some types only had one value in their Weak attribute when they should've had multiple.

Once we started populating the database more, we kept running into some data type issues. One example is when we underestimate how long the name of a Pokemon game would be. We had a varchar(18) data type for Gname which was too short for our new Game Table requirements. As such we had to alter the table in order to insert our new data.

A similar issue happened with some of our INT data types. We didn't take into account the ranges of our TINYINT Power attribute in Moves. We had it signed so the maximum value it could take was 127 which was insufficient for Moves like Explosion which have a Power of 250.

As we had to keep fixing small details such as these, we had to keep updating our schemes and diagrams to reflect it. Altering these tables was difficult but in some instances we were able to simplify drop and recreate the new table as there were no records inserted yet.

Another challenge we faced was creating the UI for our database. During the duration of this project the team kept switching between which UI we were going to implement. At first we had agreed on using ASP.net but then we began considering using Wordpress since it was built in as part of Azure. Afterwards, we realized that UI was especially challenging for the team. There was a major learning curve in understanding how to make a UI. The UI aspect was a big roadblock to our progress as most of our members weren't experienced with the front end. In the end, we sort of managed to make a usable UI. There were still difficulties in getting others to

access it.There was some difficulty in setting up the right environment but by following the steps we've outlined hopefully there are no problems.

Changes:

If we had another week to work on this project, we'd contribute more time towards polishing our UI and potentially updating more of our database schemas. We'd be able to add in triggers or other quality control checks to ensure that the right string formats were used. Attributes such as Gname are subject to incorrect data in our current interaction of the database. Hypothetically, you could insert 'Waffles' as a Gname and it would work (given that you fill the other attributes correctly). Admittedly, it would prove extremely difficult to ensure it's a valid game name as it is not out of the realm of possibility for there to be 'Pokemon Waffles' and 'Pokemon Pancakes'. At the very least, we could ensure that the Gname starts with 'Pokemon'.

We'd also work on normalizing some of our tables to reach higher levels, with a strong priority on the 'Type' table as its Weak, Effective, and Not_Effective attributes are essentially multivalued because it is a string of multiple types. We're not sure if this would be the most effective method but we could change the 'Type' table to only be a single attribute (Type_1) and create 3 other M to N tables for each effectiveness type (1 for Weak, 1 for Strong, and 1 for Not_Effective). It would be a sort of recursive relationship with Type. Each table would have Type_1 as a foreign key and a new attribute called weak/strong/not_effective depending on which table we're discussing which are also foreign keys. Both these keys would be the primary key so that we could include every effectiveness relationship. For example, the 'Weak' table would have records such as

Additions:

Outside of that one week, there are multiple other changes we could make to further expand on our database. This could be either more design updates and/or even additions to expand on our vision.

For starters, we could include another table related to Pokemon called 'Characteristics' that would include attributes such as weight, height, egg group, and more. We could even transfer some of our attributes from the 'Pokemon' table to this 'Characteristics' table if they make more sense there. This would help incorporate more

Another addition would be expanding on the region_Num attribute in 'Pokedex' to be specific to each regional dex (i.e. there'd be an attribute called Alolan_Num,another called Unova_Num, and so on). We could even make each regional dex their own table (in a 1 to 1 relationship) in order to include the Pokedex dialog entries.

To cater to our more competitive Pokemon players, we can add another table called Trainers which would include characters such as Gym Leaders and Rivals for users to check what Pokemon Teams they have. To make the connection between trainers and Pokemon, we could either have 6 foreign key attributes labeled PokemonX (where X is a number from 1-6) or make another table called Team that has the Trainer Name as a Primary & Foreign Key and the 6 foreign key attributes for each team member. I could continue on with this one addition but it would require a lot of time to make assumptions and explain the logic so I'll leave it as is for the current moment.

Data Population:

We've already discussed how we went about generating data in the section: *Methodology for Data Creation, Population, and Testing.* As such, we won't go over this again but we will reflect more on our methods here.

The Python scripting ended up being a lifesaver. Once we managed to figure out how to extract data from the API for one table it was only a matter of altering our script file to extract data for a different table. There were small difficulties in accessing the data correctly as the API documentation was a little unclear in certain aspects.

The biggest challenges in scripting came from certain tables that had to be logically extracted differently. For example, the API had no information for the original regional number of Pokemon so we had to do several loops through sections of the API to input the correct regional number.

Another difficulty was that certain information was not in the API such as Pokemon rarity. In situations such as these, we had to devise certain strategies to fill this information correctly. Most of these solutions required quite a bit of manual work. For example, when creating inserts for Pokemon, we left the rarity as 'basic' and then went into the files ourselves and changed those that were actually 'legendary' or 'mythical'.

Testing:

When testing, we asked AI such as ChatGPT and BingAI to create inserts or update statements that were not acceptable in our database. We attempted to cover as many tables as possible along with certain situations but we did still hope we could do more testing. Most situations we covered dealt with violating table constraints ranging from referential integrity to domain constraints. A more comprehensive list of how we tested is included in the section: *Methodology for Data Creation, Population, and Testing.*

We did also include testing certain operations such as insertions and updates. Deletions of actual records were successful in a way. That is, we weren't allowed to delete integral parts of our database (such as a Pokemon) without possibly deleting other records in other tables. If we were to turn off MySQL's Safe Updates, we would be able to delete a Pokemon along with related entries in other tables. We didn't include these actual deletions as they would mess up our database if someone were to turn off the system checks (i.e Safe Updates and Foreign Keys).

Overall, there would be times when we would unknowingly encounter a flaw in our design. This led to us having to rethink our data or update some of our tables.

Through these various tests and methods, we were able to ensure that our database acted according to our requirements. We tried to be as thorough as possible in our tests and table constraints. We will admit that there is an oversight in certain situations. If a Pokemon were to get a pre-evolution added later on (such as the case with Cleffa and Pichu) there would be no way to increment the evo_stage for their next evolutions. Triggers would help ensure data integrity yet we did not manage to implement them.

Reflections:

In the beginning for the first iteration, most of our effort was spent on creating the database design and schemas. The Relational Data model was more difficult to implement as we previously had our ‘Mechanics’ table as a weak entity. We spent a lot of time discussing how we were going to handle this relationship and created a couple different diagrams before finally landing on one.

In iteration 2, we spent a moderate amount of effort in choosing our tools for the project. Had we known how complex certain tools were going to be, we would’ve researched more before deciding to stick with them. For example, Postman should’ve been simple to use but we found Python to be an even better alternative. This was especially helpful considering that we were running into difficulties with Postman and JSON.

As part of our feedback from our first presentation, we added a new table called ‘Moves’ to our database. Not too much effort was spent on this as ‘Moves’ was straightforward in its design and data.

What did eat up a lot of our effort and time was trying to create data for our database. A lot of effort was put in from everyone to clean up our database design and get our data in the correct format. Just in case we didn’t manage to finish our Python scripting in time, some members manually created data insertions to populate our database for iteration 3. Thankfully, we managed to get most of our scripts done so we didn’t end up using most of our backup data files.

In retrospect, an avoidable amount of time was spent fine-tuning our project. If we had only spent some more time in the beginning with our database design, we would’ve saved a lot of time and effort. The same goes for iteration 2, if we had chosen our tools better, we wouldn’t have spent so much effort attempting to make certain tools work out.

Version Control (Updates and Changes)

For Iteration 3:

Table Creation:

- Added Table Moves:
  Created attributes: Move_Name, Damage_Type, PP, Power, and FK Type
  - Reason:
    Given feedback during our Tooling Presentation led the team to create another table detailing Pokemon moves.

    Due to the subsequent changes in how the Type table was designed, it now made it easier to use Type_1 as a foreign key.

Table Updates:

- Table Game:
  Update Gname from VARCHAR(18) to VARCHAR(50)

- Reason:
  We decided to include both versions of Pokemon games to act as one title. The reason being that differences between the two are very minimal, with only version exclusives being the biggest difference. Keeping the different versions of the games separate from each other would also complicate the data and cause confusion.

  For Example: Pikachu debuted in both “Pokemon Red” and “Pokemon Blue”. If we want to include the game that a Pokemon debuted from we’d need to add a foreign key that is multivariable. This is slightly unnecessary as both games are exactly the same in the database minus their names. As such, we decided to combine both versions into one title (EX: “Pokemon Red and Blue”) to simplify the data. Now that the titles are longer in length, we had to change the maximum length of a Pokemon game title.

Add constraint checks to Region

- Reason:
  Our main goal in adding these constraint checks was to do a bit of quality control. Since there aren’t that many regions, we decided to check and make sure the region name added was one of the 9 possible mainline Pokemon regions. That way we don’t get any data outside of the proper domain such as “Texas”.

- Table Pokemon:
  Update Gender_Ratio
  - Reason:
    Changed data type to instead be a decimal in the format of XXX.XX to represent the probability of the Pokemon being male. A string acting as a ratio overcomplicated what was needed to represent the gender ratio.
    - Also, with the realization that genderless Pokemon exist, we decided to use ‘-1’ as an indication of a genderless or purposely unknown gender. The string format would’ve been unnecessarily difficult to account for.

  Delete Next_Evo
  - Reason:
    Our initial design overlooked how Pokemon can evolve into MULTIPLE other evolutions. We would’ve needed to make Next_Evo a multivariable foreign key to account for these branched evolution. We opted to instead just stick with Previous_Evo as this was enough information for queries. As long as there is one connection between a Pokemon and its evolution, there will be a way to format queries to follow along an evolution line.

- Table Type:
  Added a constraint to check if Type_1 is a valid Pokemon Type

Reason:
All the possible Pokemon types are already included in the game. In the event that a new type is added in the future, we will simply alter the constraint check to also include this new type

Deleted Type_2
Reason:
Simplification of data. With our updated database schema, it no longer made sense to include dual types together in one table. Not only that, but since we added 'Moves' as a table, it made no sense to have them be coupled together like that. A Pokemon move is only a single type so this further supported this decision.

- Table Mechanics:
  Created Mechanic_ID attribute
  - Reason:
    Mechanics was a weak entity so in order to make it a strong entity, we added a new attribute to act as a primary key. This way we would also be able to keep track of each different evolution method far easier

- Table: Contain
  Updated for the new M to N relationship resulting from splitting up the Type table to refer to types individually
  - Reason:
    Many different Pokemon can make use of many different moves. This necessitated the creation of another table between Pokemon and Moves

Assumption Additions:

- Exclusive versions of the mainline games have minimal differences between the two
- It is possible for Pokemon to not have any type advantages or disadvantages.
- Status moves that do no damage (such as growth) have a Power of 0
- Other regions that take place in the same location (ie Sinnoh being the present-day version of Hisui) are considered to be one region
  - Hisui would be grouped under Sinnoh
- DLC regions that take place outside of a game's main region (ie Kitakami in Paldea) are also grouped under the same main region
  - DLCs are not considered to be their own separate games

Important Reasons:
To avoid data redundancy and simplify our database, sub-regions were grouped together. Pokemon such as Sinistcha were specifically released in DLC regions. This lets us keep our scope relatively tight and not branch out too far. If we were to consider DLC

as their own categories, then we might branch out even further and count spin-off games as part of this database.

Overall Changes:

- Changed data types of small data for storage efficiency (i.e. using SMALLINT and TINYINT instead of INTEGER for single digit data)
- Changed a lot of data types to be NOT NULL as they should always have a value for their attributes (EX: A Pokemon will always be a certain Evolution Stage, regardless of whether or not they have any pre-evolutions or evolutions)

For Iteration 4:

Table Creation:

- Added Stats
  Reason:
  We felt like we didn't quite reach the minimum 7 tables with 3 attributes requirement so we decided to add one last table to meet it.

Document Outline Changes and Additions:

- Version Control and Data Access Sections
  Reason:
  Swapped locations for easier reading and better flow. Version Control section now goes before Data Access

- Final Database Design Reflection: Challenges and Possible Improvements
  Reason:
  Added reflections and further discussion on the process over the quarter of creating this database

Tooling Changes:

- Replaced Postman with Python Scripting to retrieve database data insertions.
  Reason: We were unable to get Postman and JSON scripting to work correctly. Another team member was able to problem-solve a different method to gather data using Python Scripting.

UI Changes:

- Replaced Previous UI Implementation with .NET(C#) Console Application
  Reason:
  We ran into many struggles attempting to make a web-based UI application. None of our team members were really experienced with UI so we were unable to make much progress. Eventually, we decided to go the minimal UI route and

go forward with a console based application. We found a great tutorial and guide to connect our database to our UI and create different user access privileges.

- Added Visual Studio into our Tooling Section
  Reason:
  Since we were now using a different UI method, we needed to use a different IDE to accomplish our Console Application. Unlike VSC, Visual Studio lets us make a project specifically tailored to create a Console Application.

References

Bing. (2023). BingAI (March 14 2023 version) [Large language model]. https://www.bing.com/chat

Cloud Knowledge. “Connect & Query Azure Database for Mysql Flexible Server Using .Net | Use C# to Connect and Query.” *YouTube*, YouTube, 10 Aug. 2023, www.youtube.com/watch?v=d25JQCEKzrQ.

“Documentation.” *PokéAPI*, pokeapi.co/docs/v2. Accessed 11 Dec. 2023.

Elmasri, Ramez, and Sham Navathe. *Fundamentals of Database Systems*. Pearson, 2020.

Nintendo, Contributors to. “List of Pokémon Games.” *Nintendo*, Fandom, Inc., nintendo.fandom.com/wiki/List_of_Pok%C3%A9mon_games. Accessed 11 Dec. 2023.

OpenAI. (2023). ChatGPT (November 21 2023 version) [Large language model]. https://chat.openai.com/chat/

“The Official Pokémon Website.” *Pokemon.Com*, www.pokemon.com/us/pokedex. Accessed 11 Dec. 2023.

“Region.” *Region - Bulbapedia, the Community-Driven Pokémon Encyclopedia*, bulbapedia.bulbagarden.net/wiki/Region. Accessed 11 Dec. 2023.

Shreyaaithal. “Quickstart: Connect Using C# - Azure Database for MySQL - Flexible Server.” *Quickstart: Connect Using C# - Azure Database for MySQL - Flexible Server | Microsoft Learn*, learn.microsoft.com/en-us/azure/mysql/flexible-server/connect-csharp. Accessed 12 Dec. 2023.

"Where Legends Come to Life." *Serebii.Net*, www.serebii.net/index2.shtml. Accessed 11 Dec. 2023.