

# **Database Systems: 02170**

## **Group Project - 10 April 2014**

### **Pokemon Battle Analysis and Counter-Strategy Database**

<b>Thibault Chevalerias</b>	<b>s136103</b>
<b>Kwok Ho Wong</b>	<b>s136055</b>
<b>Ying Tai Lau</b>	<b>s136194</b>
<b>Jacob Lambert</b>	<b>s136215</b>

## **Description**

### **Introduction**

The basis of our database project is the handheld console game Pokémon. Pokémon is a role-playing game, where the player navigates through a virtual world as a Pokémon trainer. The goal of the game is to collect Pokémon, which are creatures of varying size and attributes which are found in the wild but easily domesticated, and to battle other trainers and enemies using these Pokémon.

Our database focuses on the attributes and abilities of individual Pokémon related to performance in battle. Although we could consider additional elements, such as regions of the virtual world or specific trainers and their Pokémon, our database is sufficiently complex without these elements. However, our database could easily be expanded to contain this non-battle related information.

The goal of our database is to allow the user to effectively analyze strategies for winning battles against other trainers. By storing a database of information about individual Pokémon, the user easily identify weaknesses in his opponents Pokémon while capitalizing on his own Pokémon's strengths.

### **Description of data set**

We obtain our dataset from an encyclopedia about pokemon online. Here are some examples:

#	Pokémon	HP	Attack	Defense	Speed	Special	Total	Average
001	 Bulbasaur	45	49	49	45	65	253	50.6
002	 Ivysaur	60	62	63	60	80	325	65
003	 Venusaur	80	82	83	80	100	425	85
004	 Charmander	39	52	43	65	50	249	49.8
005	 Charmeleon	58	64	58	80	65	325	65
006	 Charizard	78	84	78	100	85	425	85
007	 Squirtle	44	48	65	43	50	250	50

Figure1. Properties of different pokemon (source: <http://bulbapedia.bulbagarden.net/>)













Bulbasaur family							
	Bulbasaur	Level 16 →		Ivysaur	Level 32 →		Venusaur
Charmeleon family							
	Charmander	Level 16 →		Charmeleon	Level 36 →		Charizard
Squirtle family							
	Squirtle	Level 16 →		Wartortle	Level 36 →		Blastoise
Caterpie family							
	Caterpie	Level 7 →		Metapod	Level 10 →		Butterfree

Figure2. List of evolution family (source: <http://bulbapedia.bulbagarden.net/>)

We have generated large amount of data with a C++ program by scraping the raw data from this website.

## Entities

We consider two important entities, individual Pokemon and individual moves. Every Pokemon has several specific traits, such as level, attack strength, defense, and most importantly a type, such as fire, grass, or water. Also, some Pokemon may display a combination of two types, such as both grass and water. Finally, some Pokemon have the ability to instantaneously evolve into a stronger Pokemon once they reach a certain level or are exposed to specific conditions.

Each Pokemon can learn up to four moves at a time. These moves also have several specific traits, including power, physical damage, and again a type, such as fire, grass, or water. Not surprisingly, there is a one to one correspondence between the types of Pokemon and the types of moves. A Pokemon's ability to learn a move is based on the type of the pokemon, the type of the move, as well as other factors. For example, it is likely that a Pokemon of type fire will be able to learn most moves of type fire.

One of the most important aspects of a Pokemon battle between trainers is the combination of move types and Pokemon types each trainer possesses. These combinations are important because certain types of moves are more effective against certain types of Pokemon, and vice versa. For example, if a type water move is used against a type fire Pokemon, the move is considered “super-effective”, dealing twice the normal damage. Conversely, if a type grass move is used against a fire type Pokemon, the move is considered “not very effective”, dealing half the normal amount of damage. These combinations become increasingly interesting and complex when Pokemon with two types are used, and our database will be an effective tool for analyzing moves in these situations.

## Relationships

There are many relationships between the entities:

- a pokemon can have one or two types
- a move has one type
- a move type can be effective or non-effective against a Pokemon type (damage x2 or /2)
- if a Pokémon has two types, the effectiveness of a move against each type multiplies
- a move can be learned by many Pokémon
- a Pokémon can learn many moves under some conditions
- a move can either be special or physical
- the power of a move used by a pokemon depends on its attack (resp. special) if it's a physical move (resp. special).
- a Pokémon can have one or many evolutions (transformations into other pokémon), under specific conditions.

## Hand written ER diagram

By analyzing the entities and relationships between them we have generated a handwritten diagram.

There are 3 main entities, 'pokemon', 'type' and 'moves' in this diagram. For each pokemon, there are 7 attributes: ID, names and their characteristics including HP (Health Points), attack, defense, special, speed. Each pokemon belongs to one or two types (grass, fire, water, etc.) and one type can be shared by more than one pokemon. Therefore, we use many-to-many relationships to represent these connections. (i.e. M:N). Similarly, each pokemon owns more than one move and a move can be learned by more than one pokemon. Therefore, we use many-to-many relationships here as well.

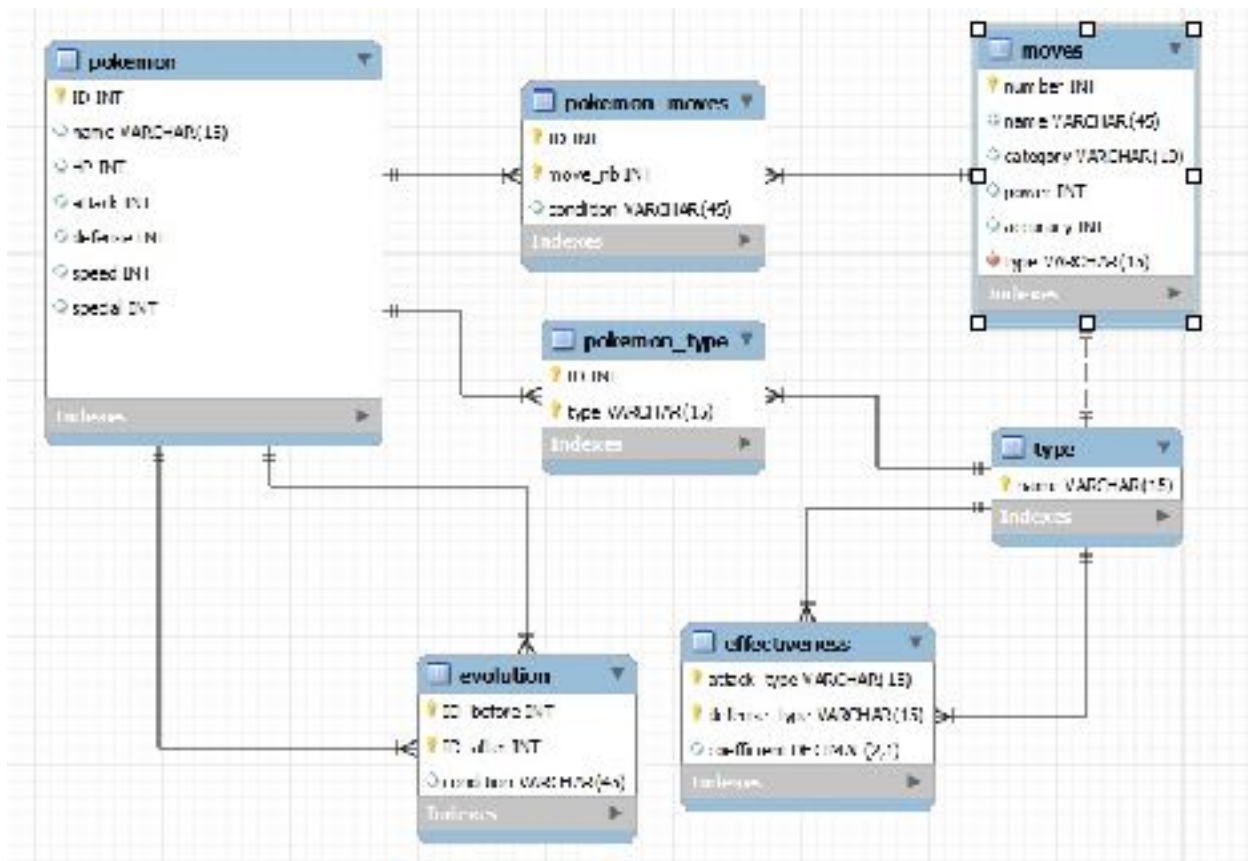
There is also a relation between pokemon themselves, modelling evolutions: it is a many-to-many relationship because one evolution is not unique (there are some rare examples, but we have to consider them), for example Eevee can evolve into 3 different pokemon (and more if you consider recent pokemon), under specific conditions.

The entity type is crucial because the relationships between these types affect the effectiveness of moves used in battles. Specifically, these types are used to determine the most effective attack and defense types for a specific battle situation. Because these different types interact in many ways, the relationships between types is also a many-to-many relationship.

Lastly, several different moves can have the same type, but each move is allowed only one type. For example, there are several different moves of type fire, but no move can be both grass and fire. Therefore, the relationship of type and moves is one to many.



# MySQL EER diagram



## References:

- pokemon (ID) to evolution (ID\_before)
- pokemon (ID) to evolution (ID\_after)
- pokemon (ID) to pokemon\_moves (ID)
- pokemon (ID) to pokemon\_types (ID)
- pokemon\_moves(move\_nb) to moves(number)
- moves(type) to type(name)
- pokemon\_type(type) to type(name)
- effectiveness(attack\_type) to type(name)
- effectiveness(defense\_type) to type(name)

## Justifications for design choices in the EER diagram

As the ID of any pokemon is unique, we use it as the primary key. We use INT for the data, which are numbers (HP, attack, etc. for pokemon, and power, accuracy, etc. for moves), except effectiveness(coefficient) because the data for it may be  $\frac{1}{2}$ , where we use DECIMAL instead. Each move also has a unique ID, so we use this ID as the primary key. Finally, for types, their name is enough because each type is different.

# Database instance

Example data is given through insert, delete and update statements in the next part. Also, we have added some SQL files to our hand-in, so the data is not put here.

The table headers are:

- \* pokemon: ID, name, HP, attack, defense, speed, special
- \* evolution: ID\_before, ID\_after, condition
- \* moves: number, name, category, power, accuracy, type
- \* pokemon\_moves: ID, move\_nb, condition
- \* type: name
- \* pokemon\_type: ID, type
- \* effectiveness: attack\_type, defense\_type, coefficient

## Normalization

We have designed our database with atomic elements from the beginning. For instance, the attributes HP, attack, defense, etc. of pokémon are separated. Also, we have decided to consider a many-to-many relation between pokémon and types, even if a pokémon has only one type. We could have created two attributes “type1” and “type2”, with the second one being null for pokémon having only one type, but our database is more structured by considering the relationship as many-to-many.

Moreover, the attributes have been type-casted such that they correspond perfectly to the data type we have chosen for them: varchar, int, decimal, etc. Therefore there is no problem with the domains where the attributes are taken from. Also, from the discussions above, it is obvious that each primary key is unique and well-defined.

Thus, our database design corresponds at least to the first normal form.

Our database also follows the second normal form restrictions because the only tables having primary keys with two attributes in it are those resulting from many-to-many relationships. The non-primary key attributes which could lead to a problem are the two “condition” attributes, and the “coefficient”. But they depend on both parts of the primary key if we consider the explanations in the introduction, so our database is also in the second normal form.

Similarly, We can use the same arguments for the third normal form. Since the two “condition” attributes depend on each pokemon, or combination of them, there cannot be transitive attributes. Same argument follows for “effectiveness” which depends on each combination of types.

Finally, we have proved that our database is in third normal form. While extracting data for pokemon, moves, etc., there was very little redundancy in data, which indicates that we have appropriately normalized our database.

## SQL updates

### Database insert, delete, update statements

#### Examples of some insert statements:

- insert into pokemon values ('Bulbasaur', 1, 45, 49, 49, 45, 65)
- insert into pokemon values ('Ivysaur', 2, 60, 62, 63, 60, 80)
- insert into pokemon values ('Venusaur', 3, 80, 82, 83, 80, 100)
  
- insert into type values ('Grass')
- insert into type values ('Poison')
  
- insert into pokemon\_type values ('Bulbasaur', 'Grass')
- insert into pokemon\_type values ('Bulbasaur', 'Poison')
  
- insert into effectiveness values ('Fire', 'Grass', 2)
- insert into effectiveness values ('Water', 'Grass', 0.5)
- insert into effectiveness values ('Fighting', 'Grass', 1)
  
- insert into evolution values (1, 2, 'level 16')
- insert into evolution values (2, 3, 'level 32')
  
- insert into moves (33, 'Tackle', 'physical', 35, 95, 'Normal')
- insert into moves (75, 'Razor Leaf', 'physical', 55, 95, 'Grass')
- insert into moves (76, 'Solar Beam', 'special', 120, 100, 'Grass')
  
- insert into pokemon\_moves (1, 33, 'basic attack')
- insert into pokemon\_moves (1, 75, 'level 19')
- insert into pokemon\_moves (1, 76, 'TM 22')

In the new pokémon games, the damage of some moves have been modified, so we could do this kind of updates. Also, add new pokémon would be simple.

#### Examples of some delete statements:

- delete from pokemon;
- delete from pokemon where name = 'Bulbasaur';
  
- delete from type where name = 'Normal';
- delete from type where name = 'Fighting';
  
- delete from effectiveness where attack\_type = 'Fire';
- delete from effectiveness where defense\_type = 'Water';
  
- delete from evolution;
  
- delete from moves where number = 1;
- delete from moves where name = 'Double Slap';

#### Examples of some update statements:

- update pokemon set HP = 100 where name = 'Ivysaur';



- update pokemon  
set HP = case  
    when HP <= 50 then HP\*1.05  
    else HP\*1.03  
end;
- update pokemon\_type set type = 'Grass' where type = 'Poison';

# SQL queries

## Database queries

We can, for example, want the list of all pokémon:

```
select * from pokemon;
```

We may want to have the list of water-type pokémon:

```
select * from pokemon
      join pokemon_type.ID = pokemon_type.ID
where pokemon_type.type = 'water';
```

List of pokémon with more than 100 in attack and more than 80 in speed:

```
select * from pokemon
where attack > 100 and speed > 80;
```

List of physical moves:

```
select * from moves
where category = 'physical';
```

List of grass pokémon, ordered by their special:

```
select * from pokemon
      join pokemon_type.ID = pokemon_type.ID
where pokemon_type.type = 'grass'
order by special;
```

List of electric and special moves, ordered by power\*accuracy:

```
select number, name, type, category, power*accuracy
from moves
where type = 'electric' and category = 'special'
order by power*accuracy;
```

Calculate the average characteristics for each type of pokemon:

```
select pokemon_type.type, avg(HP), avg(attack), avg(defense), avg(speed), avg(special)
from pokemon
      join pokemon_type on pokemon.ID = pokemon_type.ID
group by pokemon_type.type;
```

A very useful query could be the list of types effective against fire for example:

```
select name from type
      join effectiveness on type.name = effectiveness.attack_type
where defense_type = 'fire' and coefficient = 2;
```

## Example of triggers

Imagine we want to add new moves. It is possible that a user could attempt to enter the accuracy of the move as a probability ratio instead of a percentage. If such an entry is made, we

could then activate a trigger to rectify the mistake.

```
delimiter //
create trigger accuracy_check
before insert on moves for each row
begin
if (new.accuracy < 30 or new.accuracy > 100) then set new.accuracy = null;
end if;
end; //
delimiter ;
```

Thanks to this, we will know that there is a problem with this move (remark : accuracy cannot be under 30).

Also, a move is either 'physical' or 'special', so we can create a trigger for that:

```
delimiter //
create trigger category_check
before insert on moves for each row
begin
if new.category not in ('physical', 'special') then set new.category = null;
end if;
end; //
delimiter ;
```

Another useful trigger would check if the power of an move is positive (it can be 0 because some moves have special effects instead of inflicting damage):

```
delimiter //
create trigger power_check
before insert on moves for each row
begin
if new.power < 0 then set new.power = null;
end if;
end; //
delimiter ;
```

Finally, since "This version of MySQL doesn't yet support multiple triggers with the same action time and event for one table", as kindly said by MySQL debugger, we can create a big trigger which does all these tasks:

```
delimiter //
create trigger moves_check
before insert on moves for each row
begin
    if new.power < 0 then set new.power = null;
end if;
```

```

        if new.category not in ('physical', 'special') then set new.category = null;
        end if;
        if (new.accuracy < 30 or new.accuracy > 100) then set new.accuracy = null;
        end if;
    end; //
delimiter ;

```

## **Ideas of functions, procedures, views around a common goal: to win a battle**

We can create an useful view:

```

create view extended_effectiveness as
select A.attack_type as attack, A.defense_type as type1, if(A.defense_type <> B.defense_type,
B.defense_type, 'none') as type2, if(A.defense_type <> B.defense_type,
A.coefficient*B.coefficient, A.coefficient) as coefficient
    from effectiveness as A
    join effectiveness as B on A.attack_type = B.attack_type;

```

Indeed, many pokémon have two types, so it is very useful to know the effect of opponent's attacks on the combination of those types!

For example, if we take 'Bulbasaur' (ID = 1) as our opponent, we have the types 'Grass' and 'Poison', so we can do this simple query:

```

select attack, coefficient from extended_effectiveness where type1 = 'Grass' and type2 =
'Poison' and coefficient =
(select max(coefficient) from extended_effectiveness
where type1 = 'Grass' and type2 = 'Poison');

```

Thus we get the list of the most effective kind of attacks on this particular combination of types: namely 'Fire', 'Ice', 'Psy', 'Flying'.

Now, since pokémon tend to learn mostly attacks of their types, we can be interested in finding a pokémon which has one of the most effective types against a given opponent, and also one type against which the attacks of the opponent will be inefficient:

```

delimiter //
create procedure best_types
(in test_type varchar(15), in attack_type_in varchar(15), out result varchar(5))
begin
if (test_type in
(select defense_type from effectiveness where attack_type = attack_type_in and
coefficient = (select min(coefficient) from effectiveness where attack_type = attack_type_in)))
    then set result = 'yes';
else set result = 'no';
end if;

```

```
end; //
delimiter ;
```

This procedure takes one type that we want to test and the type of the attacks: 'Grass' as input (we assume that Bulbasaur has mostly 'Grass'-type attacks), and give the result: 'yes' if the test-type is among the most resisting types to 'Grass' attacks, 'no' if not.

We have for example the results:

```
call best_types('Ice','Grass', @result);
select @result;
```

gives 'no'

```
call best_types('Fire','Grass', @result);
select @result;
```

gives 'yes'

So, we will know that it may be better to choose a Fire pokémon instead of an Ice pokémon: even if both types have effective attacks against Bulbasaur, Fire will resist better than Ice against Bulbasaur's attacks.

So, if we want our pokémon to have a fire type, we may want to know the best fire moves:

```
delimiter //
create procedure best_moves
(in type_in varchar(15))
begin
select name, category, power, accuracy from moves where type = type_in order by
power*accuracy limit 5;
end; //
delimiter ;
```

This procedure gives us the five 'best' moves of a given type, ranked by power\*accuracy. It is better to show five results instead of just one, because we can choose between those five if we want to maximize damage or accuracy or supplementary effects...

For example, call best\_moves('fire'); gives us 'fireblast' in first, this is the one we will choose.

Then, the only thing left to do is to choose the pokémon. So, we need the list of pokémon that can learn the move we have chosen.

```
delimiter //
create procedure best_pokemon
(in move_name_in varchar(45))
```

```
begin
select pokemon.name, HP, attack, defense, speed, special, pokemon_moves.condition
from pokemon
      join pokemon_moves on pokemon.ID = pokemon_moves.ID
      join moves on moves.number = pokemon_moves.move_nb
where moves.name = move_name_in
order by if(moves.category = 'physical', attack, special) desc limit 5;
end; //
delimiter ;
```

We obtain a small list like in the example before. If we want to make precise choices. The pokemon that can learn the move are ranked by their attack if it is a physical move (because this is the relevant characteristic) and by their special if it is a special move. Thus we will be able to inflict maximum damage.

Remark : the names of the moves are unique (otherwise, the procedure would not perform optimally).

So, in conclusion, we see that the best choice for this battle is Moltres, but Arcanine and Charizard are also very good options.