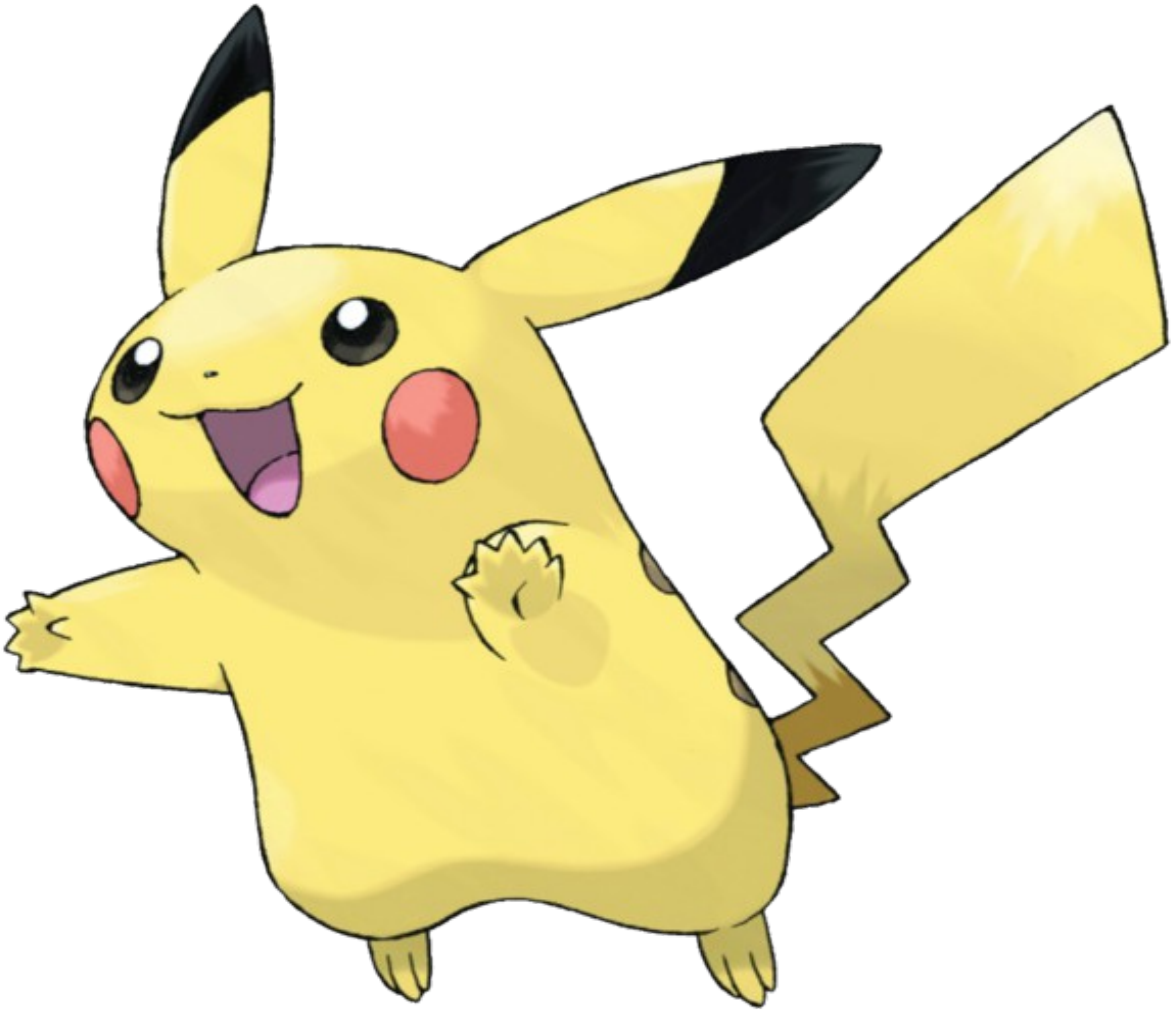
Pokemon Database



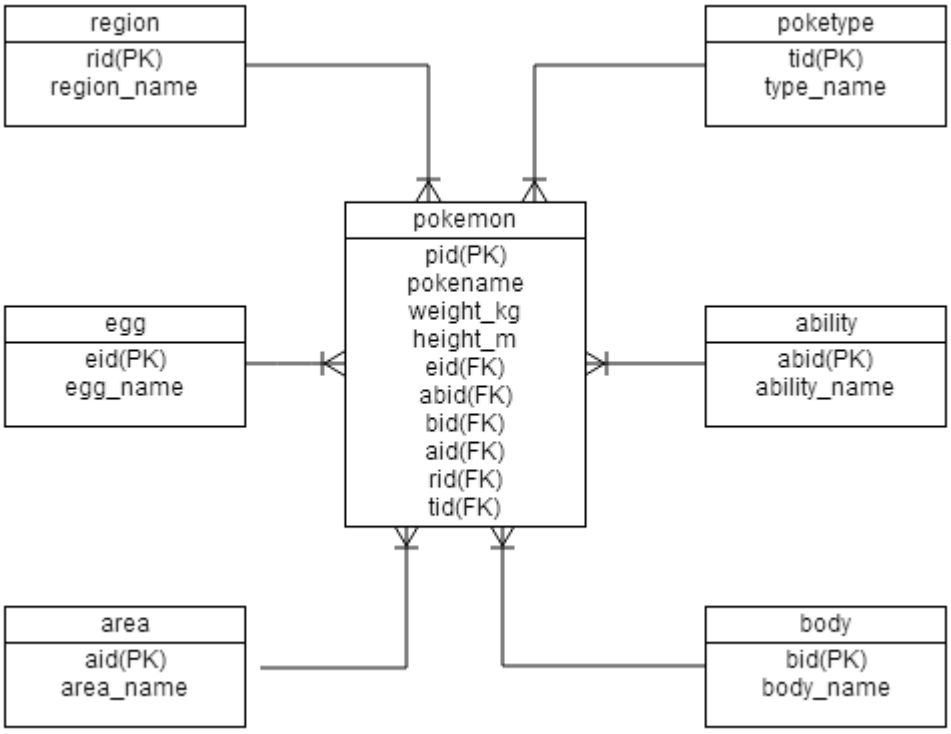
By Miles Welsh
Database Management

Table of Contents

Executive Summary

This document outlines our teams design and implementation of a database for Nintendo, based on their Pokemon series of games. This database will become a central location where all information about current and future Pokemon can be stored. In our ER diagram we will provide our high level solution and describe each one of the entities present. In our entity section we will describe the purpose of each entity and how we plan to meet the requirements given to us by Nintendo. For each entity we will provide create statements and sample data. The intended effect of this sample data is to show Nintendo how there newly designed database works. We also will make this database easier to use by creating procedures that can be used in place of complex SQL commands. We will show how these procedures work and how they can be used. Our final section will detail our possible future plans and some of our design tradeoffs.

Entity Relationship Diagram



Entities

These are the create statements and sample data for the entities seen in the entity relationship diagram

area

The area entity stores the data about where Pokemon can be found in the wild. For example, Pikachu prefers the grassland area. This entity makes it easy for Nintendo to add new areas and also easy to add Pokemon to new areas.

```
--area--
DROP TABLE IF EXISTS area;
CREATE TABLE area (
  aid         char(4) NOT NULL,
  area_name   TEXT,
PRIMARY KEY(aid)
);
```

	aid	area_name
1	0001	grassland
2	0002	forest
3	0003	near water
4	0004	in water
5	0005	cave
6	0006	mountain
7	0007	rough
8	0008	urban
9	0009	anywhere

Functional Dependencies

area: aid → area_name

egg

The egg entity stores data about what egg group the Pokemon is in. For example, Pikachu is in the field egg group. This entity allow Nintendo to create new egg groups and also add new Pokemon to an existing egg group

```
--egg--
DROP TABLE IF EXISTS egg;
CREATE TABLE egg (
  eid         char(4) NOT NULL,
  egg_name    TEXT,
PRIMARY KEY(eid)
);
```

	eid	egg_name
1	0001	monster
2	0002	amphibious
3	0003	bug
4	0004	flying
5	0005	field
6	0006	fairy
7	0007	grass
8	0008	humanlike
9	0009	aquatic_invertebrates
10	0010	mineral
11	0011	amorphous
12	0012	piscine
13	0013	dragon

Functional Dependencies

egg: eid → egg_name

region

The region entity stores data about the region in which a Pokemon was originally found in. For example, Pikachu originated in the Kanto region. In this example, I have only used the first 3 regions within the game but, this entity makes it easy for Nintendo to add the other regions and add the future regions that will be present in upcoming games.

```
--region--
DROP TABLE IF EXISTS region;
CREATE TABLE region (
  rid         char(4) NOT NULL,
  region_name   TEXT,
PRIMARY KEY(rid)
);
```

	rid	region_name
1	0001	kanto
2	0002	johto
3	0003	hoenn

Functional Dependencies

region: rid → region_name

poketype

The poketype entity stores the data about what type a Pokemon is. For example Pikachu is an electric type and can use electric moves. Please note, that I have not included the fairy type because I am only focusing on the first three generations. Using this entity Nintendo will be able to easily insert new types and put new Pokemon into existing ones.

```
--poketype--
DROP TABLE IF EXISTS poketype;
CREATE TABLE poketype (
  tid         char(4) NOT NULL,
  type_name   TEXT,
PRIMARY KEY(tid)
);
```

	tid	type_name
1	0001	normal
2	0002	fighting
3	0003	flying
4	0004	poison
5	0005	ground
6	0006	rock
7	0007	bug
8	0008	ghost
9	0009	steel
10	0010	fire
11	0011	water
12	0012	grass
13	0013	electric
14	0014	physic
15	0015	ice
16	0016	dragon
17	0017	dark

Functional Dependencies

poketype: pid → type_name

ability

The ability table stores the data about a Pokemons ability. For example Pikachu has the static ability. Please note, that since this is just the first three generations I have not accounted for hidden abilities, but these abilities can be added easily by just putting in new abilities. This enetity will allow Nintendo to insert new abilities, like hidden abilities and pair new Pokemon with existing abilities.

```
--ability--
DROP TABLE IF EXISTS ability;
CREATE TABLE ability (
  abid         char(4) NOT NULL,
  ability_name  TEXT,
PRIMARY KEY(abid)
);
```

	abid	ability_name
1	0001	blaze
2	0002	overgrow
3	0003	torrent
4	0004	static
5	0005	keen eye
6	0006	insomnia
7	0007	guts

Functional Dependencies

ability: abid → ability_name

body

The body entity stores the data about what type of body the Pokemon has. For example, Pikachu has a quadruped body. These entity contains all of the body types as no new body types were added after generation three. This entity will allow Nintendo to add new body types and put new Pokemon into existing ones.

```
--body--
DROP TABLE IF EXISTS body;
CREATE TABLE body (
  bid         char(4) NOT NULL,
  body_name   TEXT,
PRIMARY KEY(bid)
);
```

	bid	body_name
1	0001	head
2	0002	serpentine
3	0003	fins
4	0004	head and arms
5	0005	head and base
6	0006	bipedal and tailed
7	0007	head and legs
8	0008	quadruped
9	0009	single pair wings
10	0010	multiple hands
11	0011	multiple bodies
12	0012	bipedal
13	0013	double pair wings
14	0014	insectoid

Functional Dependencies

body: bid → body_name

pokemon

This entity stores the data about the individual Pokemon and takes all of the other data from the other tables to create a unique Pokemon. In my sample data I have only added Pokemon from the first three generations. Nintendo will be able to use this to add new Pokemon, and its enter height and weight. Nintendo can then use the options in the other tables to give that Pokemon a new type, egg group, etc.

```
--pokemon--
DROP TABLE IF EXISTS pokemon;
CREATE TABLE pokemon (
  pid         char(4) NOT NULL,
  pokename    TEXT,
  weight_kg   INT,
  height_m    INT,
  eid   char(4) NOT NULL REFERENCES egg(eid),
  abid char(4) NOT NULL REFERENCES ability(abid),
  bid char(4) NOT NULL REFERENCES body(bid),
  aid char(4) NOT NULL REFERENCES area(aid),
  rid char(4) NOT NULL REFERENCES region(rid),
  tid char(4) NOT NULL REFERENCES poketype(tid),
PRIMARY KEY(pid)
);
```

	pid	pokename	weight_kg	height_m	eid	abid	bid	aid	rid	tid
1	0001	charizard	91	2	0001	0001	0006	0006	0001	0010
2	0002	venasaur	100	2	0001	0002	0008	0001	0001	0012
3	0003	blastoise	86	2	0001	0003	0006	0003	0001	0011
4	0004	pikachu	6	0	0005	0004	0008	0002	0001	0013
5	0005	pidgeot	40	2	0004	0005	0009	0002	0001	0003
6	0006	typhlosion	80	2	0005	0001	0008	0001	0002	0010
7	0007	meganium	101	2	0001	0002	0008	0001	0002	0012
8	0008	feraligator	89	2	0001	0003	0006	0003	0002	0011
9	0009	ampharos	62	1	0001	0004	0006	0001	0002	0013
10	0010	noctowl	41	2	0004	0006	0009	0002	0002	0003
11	0011	blaziken	52	2	0005	0001	0006	0001	0003	0010
12	0012	sceptile	52	2	0001	0002	0006	0002	0003	0012
13	0013	swampert	82	2	0001	0003	0006	0003	0003	0011
14	0014	manectric	40	2	0005	0004	0008	0001	0003	0013
15	0015	swellow	20	1	0004	0007	0009	0001	0003	0003

Functional Dependencies

pokemon: aid, eid,r id, pid, abid, bid, pid → pokename, weight_kg, height_m

Views

These views will create windows to the data that contain useful information.

all_view

This view is by far the most convenient because it takes all of the number ids that are in the original Pokemon table and replaces them with their actually names.

```
--all_view
CREATE VIEW all_view
AS
SELECT pokemon.pokename as "name",
egg.egg_name AS"group",
ability.ability_name AS "ability",
body.body_name AS "body type",
area.area_name AS "area",
region.region_name AS "region",
poketype.type_name AS "type"
FROM pokemon
INNER JOIN ability ON pokemon.abid = ability.abid
INNER JOIN egg ON pokemon.eid = egg.eid
INNER JOIN body ON pokemon.bid = body.bid
INNER JOIN area ON pokemon.aid = area.aid
INNER JOIN region ON pokemon.rid = region.rid
INNER JOIN poketype ON pokemon.tid = poketype.tid

SELECT * FROM all_view
```

	name	group	ability	body type	area	region	type
1	charizard	monster	blaze	bipedal and tailed	mountain	kanto	fire
2	venasaur	monster	overgrow	quadruped	grassland	kanto	grass
3	blastoise	monster	torrent	bipedal and tailed	near water	kanto	water
4	pikachu	field	static	quadruped	forest	kanto	electric
5	pidgeot	flying	keen eye	single pair wings	forest	kanto	flying
6	typhlosion	field	blaze	quadruped	grassland	johto	fire
7	meganium	monster	overgrow	quadruped	grassland	johto	grass
8	feraligator	monster	torrent	bipedal and tailed	near water	johto	water
9	ampharos	monster	static	bipedal and tailed	grassland	johto	electric
10	noctowl	flying	insomnia	single pair wings	forest	johto	flying
11	blaziken	field	blaze	bipedal and tailed	grassland	hoenn	fire
12	sceptile	monster	overgrow	bipedal and tailed	forest	hoenn	grass
13	swampert	monster	torrent	bipedal and tailed	near water	hoenn	water
14	manectric	field	static	quadruped	grassland	hoenn	electric
15	swellow	flying	guts	single pair wings	grassland	hoenn	flying

ability_view

This view will allow users to look at the Pokemon in the database and what abilities coincide with those Pokemon

```
--ability_view

CREATE VIEW ability_view
AS
SELECT pokemon.pokename AS "name", ability.ability_name AS "ability"
FROM pokemon
INNER JOIN ability
ON pokemon.abid = ability.abid

SELECT * FROM ability_view
```

	name	ability
1	charizard	blaze
2	venasaur	overgrow
3	blastoise	torrent
4	pikachu	static
5	pidgeot	keen eye
6	typhlosion	blaze
7	meganium	overgrow
8	feraligator	torrent
9	ampharos	static
10	noctowl	insomnia
11	blaziken	blaze
12	sceptile	overgrow
13	swampert	torrent
14	manectric	static
15	swellow	guts

egg_view

This view will allow you see all of the egg groups associated with the different Pokemon.

```
--egg_view
CREATE VIEW egg_view
AS
SELECT pokemon.pokename AS "name", egg.egg_name AS "group"
FROM pokemon
INNER JOIN egg
ON pokemon.eid = egg.eid

SELECT * FROM egg_view
```

	name	group
1	charizard	monster
2	venasaur	monster
3	blastoise	monster
4	pikachu	field
5	pidgeot	flying
6	typhlosion	field
7	meganium	monster
8	feraligator	monster
9	ampharos	monster
10	noctowl	flying
11	blaziken	field
12	sceptile	monster
13	swampert	monster
14	manectric	field
15	swellow	flying

area_view

This view will allow users to see each area a Pokemon lives in.

```
--area_view

CREATE VIEW area_view
AS
SELECT pokemon.pokename AS "name", area.area_name AS"area"
FROM pokemon
INNER JOIN area
ON pokemon.aid = area.aid

SELECT * FROM area_view
```

	name	area
1	charizard	mountain
2	venasaur	grassland
3	blastoise	near water
4	pikachu	forest
5	pidgeot	forest
6	typhlosion	grassland
7	meganium	grassland
8	feraligator	near water
9	ampharos	grassland
10	noctowl	forest
11	blaziken	grassland
12	sceptile	forest
13	swampert	near water
14	manectric	grassland
15	swellow	grassland

region_view

This view will show were all the sample Pokemon and what region of the world they were originally found in.

```
--region_view

CREATE VIEW region_view
AS
SELECT pokemon.pokename AS "name", region.region_name AS "region"
FROM pokemon
INNER JOIN region
ON pokemon.rid = region.rid

SELECT * FROM region_view
```

	name	region
1	charizard	kanto
2	venasaur	kanto
3	blastoise	kanto
4	pikachu	kanto
5	pidgeot	kanto
6	typhlosion	johto
7	meganium	johto
8	feraligator	johto
9	ampharos	johto
10	noctowl	johto
11	blaziken	hoenn
12	sceptile	hoenn
13	swampert	hoenn
14	manectric	hoenn
15	swellow	hoenn

poketype_view

This view will show each type associated with the different pokemon.

```
--poketype_view

CREATE VIEW poketype_view
AS
SELECT pokemon.pokename AS "name", poketype.type_name AS "type"
FROM pokemon
INNER JOIN poketype
ON pokemon.tid = poketype.tid

SELECT * FROM poketype_view
```

	name	type
1	charizard	fire
2	venasaur	grass
3	blastoise	water
4	pikachu	electric
5	pidgeot	flying
6	typhlosion	fire
7	meganium	grass
8	feraligator	water
9	ampharos	electric
10	noctowl	flying
11	blaziken	fire
12	sceptile	grass
13	swampert	water
14	manectric	electric
15	swellow	flying

body_view

This view will show users the different body types associated with different Pokemon

```
--body_view

CREATE VIEW body_view
AS
SELECT pokemon.pokename AS "name", body.body_name AS "body type"
FROM pokemon
INNER JOIN body
ON pokemon.bid = body.bid

SELECT * FROM body_view
```

	name	body type
1	charizard	bipedal and tailed
2	venasaur	quadruped
3	blastoise	bipedal and tailed
4	pikachu	quadruped
5	pidgeot	single pair wings
6	typhlosion	quadruped
7	meganium	quadruped
8	feraligator	bipedal and tailed
9	ampharos	bipedal and tailed
10	noctowl	single pair wings
11	blaziken	bipedal and tailed
12	sceptile	bipedal and tailed
13	swampert	bipedal and tailed
14	manectric	quadruped
15	swellow	single pair wings

Stored Procedures

These stored procedures are functions that will make finding data within the database easier and more convenient

ability_procedure

This procedure will allow users to find out which Pokemon have a certain ability.

```
--ability_procedure

CREATE OR REPLACE FUNCTION ability_procedure(abilityName text)
RETURNS TABLE("Name" text, "Ability Name" text) as $$
BEGIN
RETURN QUERY SELECT pokemon.pokename AS "Name",
ability.ability_name AS "Ability Name"
FROM pokemon,ability
WHERE ability.ability_name = abilityName AND pokemon.abid = ability.abid;
END;
$$ LANGUAGE plpgsql;

SELECT ability_procedure('blaze');
```

	ability_procedure
1	(charizard,blaze)
2	(typhlosion,blaze)
3	(blaziken,blaze)

egg_procedure

This procedure will allow users to find out what egg group a Pokemon belongs to.

```
--egg_procedure

CREATE OR REPLACE FUNCTION egg_procedure(eggName text)
RETURNS TABLE("Name" text, "Group Name" text) as $$
BEGIN
RETURN QUERY SELECT pokemon.pokename AS "Name",
egg.egg_name AS "Group Name"
FROM pokemon,egg
WHERE egg.egg_name = eggName AND pokemon.eid = egg.eid;
END;
$$ LANGUAGE plpgsql;

SELECT  egg_procedure('monster)
```

	egg_procedure
1	(charizard,monster)
2	(venasaur,monster)
3	(blastoise,monster)
4	(meganium,monster)
5	(feraligator,monster)
6	(ampharos,monster)
7	(sceptile,monster)
8	(swampert,monster)

area_procedure

This procedure will find all Pokemon associated with a given area.

```
--area_procedure

CREATE OR REPLACE FUNCTION area_procedure(areaName text)
RETURNS TABLE("Name" text, "Area Name" text) as $$
BEGIN
RETURN QUERY SELECT pokemon.pokename AS "Name",
area.area_name AS "Area Name"
FROM pokemon,area
WHERE area.area_name = areaName AND pokemon.aid = area.aid;
END;
$$ LANGUAGE plpgsql;

SELECT area_procedure('grassland')
```

	area_procedure
1	(venasaur,grassland)
2	(typhlosion,grassland)
3	(meganium,grassland)
4	(ampharos,grassland)
5	(blaziken,grassland)
6	(manectric,grassland)
7	(swellow,grassland)

region_procedure

This procedure will allow the user to see all the Pokemon that were originally found in this region.

```
--region_procedure

CREATE OR REPLACE FUNCTION region_procedure(regionName text)
RETURNS TABLE("Name" text, "Region Name" text) as $$
BEGIN
RETURN QUERY SELECT pokemon.pokename AS "Name",
region.region_name AS "Region Name"
FROM pokemon,region
WHERE region.region_name = regionName AND pokemon.rid = region.rid;
END;
$$ LANGUAGE plpgsql;

SELECT region_procedure('kanto')
```

	region_procedure
1	(charizard,kanto)
2	(venasaur,kanto)
3	(blastoise,kanto)
4	(pikachu,kanto)
5	(pidgeot,kanto)

poketype_procedure

This procedure will allow users to find all Pokemon that belong to a certain type.

```
--poketype_procedure

CREATE OR REPLACE FUNCTION poketype_procedure(typeName text)
RETURNS TABLE("Name" text, "Type Name" text) as $$
BEGIN
RETURN QUERY SELECT pokemon.pokename AS "Name",
poketype.type_name AS "Type Name"
FROM pokemon,poketype
WHERE poketype.type_name = typeName AND pokemon.tid = poketype.tid;
END;
$$ LANGUAGE plpgsql;

SELECT poketype_procedure('electric')
```

	poketype_procedure
1	(pikachu,electric)
2	(ampharos,electric)
3	(manectric,electric)

body_procedure

This procedure will find all Pokemon that are associated to a given body type.

```
--body_procedure

CREATE OR REPLACE FUNCTION body_procedure(bodyName text)
RETURNS TABLE("Name" text, "Body Name" text) as $$
BEGIN
RETURN QUERY SELECT pokemon.pokename AS "Name",
body.body_name AS "Body Name"
FROM pokemon,body
WHERE body.body_name = bodyName AND pokemon.bid = body.bid;
END;
$$ LANGUAGE plpgsql;

SELECT body_procedure('quadruped')
```

	body_procedure
1	(venasaur,quadruped)
2	(pikachu,quadruped)
3	(typhlosion,quadruped)
4	(meganium,quadruped)
5	(manectric,quadruped)

Reports and Queries

These queries show how the database can be used to organize and output useful information.

order by weight

This query will order the Pokemon for heaviest to lightest and then output the data.

--order by weight

```
SELECT pokename,
weight_kg FROM pokemon
ORDER BY weight_kg DESC
```

	pokename	weight_kg
1	meganium	100.5
2	venasaur	100
3	charizard	90.5
4	feraligator	88.8
5	blastoise	85.5
6	swampert	81.9
7	typhlosion	79.5
8	ampharos	61.5
9	sceptile	52.2
10	blaziken	52
11	noctowl	40.8
12	manectric	40.2
13	pidgeot	39.5
14	swellow	19.8
15	pikachu	6

order by height

This query will return the Pokemon in order of tallest to shortest.

```
--order by height

SELECT pokename,
height_m FROM pokemon
ORDER BY height_m DESC
```

	pokename	height_m
1	feraligator	2.3
2	venasaur	2
3	blaziken	1.9
4	meganium	1.8
5	charizard	1.7
6	typhlosion	1.7
7	sceptile	1.7
8	noctowl	1.6
9	blastoise	1.6
10	manectric	1.5
11	pidgeot	1.5
12	swampert	1.5
13	ampharos	1.4
14	swellow	0.7
15	pikachu	0.4

order by name

This query will sort the Pokemon's names alphabetically.

```
--order by name

SELECT pokename,
height_m,weight_kg FROM pokemon
ORDER BY pokename ASC
```

	pokename	height_m	weight_kg
1	ampharos	1.4	61.5
2	blastoise	1.6	85.5
3	blaziken	1.9	52
4	charizard	1.7	90.5
5	feraligator	2.3	88.8
6	manectric	1.5	40.2
7	meganium	1.8	100.5
8	noctowl	1.6	40.8
9	pidgeot	1.5	39.5
10	pikachu	0.4	6
11	sceptile	1.7	52.2
12	swampert	1.5	81.9
13	swellow	0.7	19.8
14	typhlosion	1.7	79.5
15	venasaur	2	100

Security

There will be types of users in this database

1. The admin who can change, update, and maintain the database.

```
CREATE ROLE admin_account
GRANT SELECT, INSERT, UPDATE, ALTER
ON ALL TABLES IN SCHEMA POKEMON
TO admin_account
```

2. The public user who can see the database and perform queries on it.

```
CREATE ROLE public_account
GRANT SELECT
ON ALL TABLES IN SCHEMA POKEMON
TO public_account
```

End Notes

Implementation/Known Problems/Future Plans

The implementation of the database went fairly smoothly. I wanted to add much more to the database but with over 700 Pokemon, that would have been impossible with the time I was giving. The hardest part of the implementation was making sure that I was complex enough without going to in depth. I wanted to create a database that accurately showcased the Pokemon universe but, didn't want to take it to far.

One of the main problems of this database is the exclusion of dual types. I mentioned before that I did not go past Generation three and this is for good reason. If I had decided to go past Generation three I would have to deal with over 400 new Pokemon, four new regions and a plethora of new dual type Pokemon. My database currently only accounts for single type Pokemon but, I do have a plan to fix this in the future.

In the future I want to be able to add all of the existing Pokemon and regions but, most importantly add dual type Pokemon. My database currently cannot take dual type Pokemon but I can solve this by considering each dual type to just be another type. For example, if I have a Pokemon that is fire type and flying type, instead of creating a second type column I can just call fire/flying its own type. This actually works out well because not all types can mix together, for example fire and water can't mix. This would also fix the problem of where a dual type Pokemon would fall. For example, if I had a fire/flying Pokemon what type would it be apart of? Fire or flying or both? The best answer appears to be both, but this doesn't work. If I put a fire/flying Pokemon in the fire group then that would imply that it can only learn fire type moves and vice versa with flying. This is incorrect. My dual type as a single type solution fixes these problems