Assignment 2
The Pokémon Database

Last updated: **Thursday 28th March 11:31am**
Most recent changes are shown in red ... older changes are shown in brown.
[SQL Schema] [SQL Data]

Introduction

This document contains a description of the Pokémon database. We give both an ER model and an SQL schema.

The database contains a large number of tables and relationships. We summarise what each table represents and the kind of information it holds here.
More information is given in the downloadable schema.

Overview

Pokemon
This table describes general aspects of each Pokémon, including: its unique ID (a combination of a Pokédex number and a variation number), its name, its species, its growth rate, its basic properties (hit points, speed, etc.), and so on.

Games
This table indicates which region a particular game occurs in.
The game ID is also used as part of a Pokédex, and an Encounter.

Types
Each Pokémon has at least one type (e.g. fire, water, ghost, flying).
Some Pokémon may have two types.
All Moves also have a type.

Abilities
Pokémon have a large range of possible abilities (e.g. flame body, gooey, iron fist, neuroforce).
A description of each ability is contained in the `effect` column of this table.
Each Ability a Pokémon knows is given in the `Knowable_Abilities` table.

Moves
Pokémon aso have a large range of possible moves they can make (e.g. blizzard, block, bounce, etc.).
Each move has an associated category, power and accuracy.
Pokémon can potentially learn more move during a game.
Each move a Pokémon can learn given in the `Learnable_Moves` table.

Evolutions
Some Pokémon can change form, and this table describes the starting form and final form.
What conditions are needed before this change can occur, is given in the `Evolution_Requirements` table.

Requirements
Some changes (evolutions, encounters, learnable moves) require certain pre-conditions before they can occur.

The `Requirements` table gives a list of possible pre-conditions which can be applied to the various table associated with "changes".

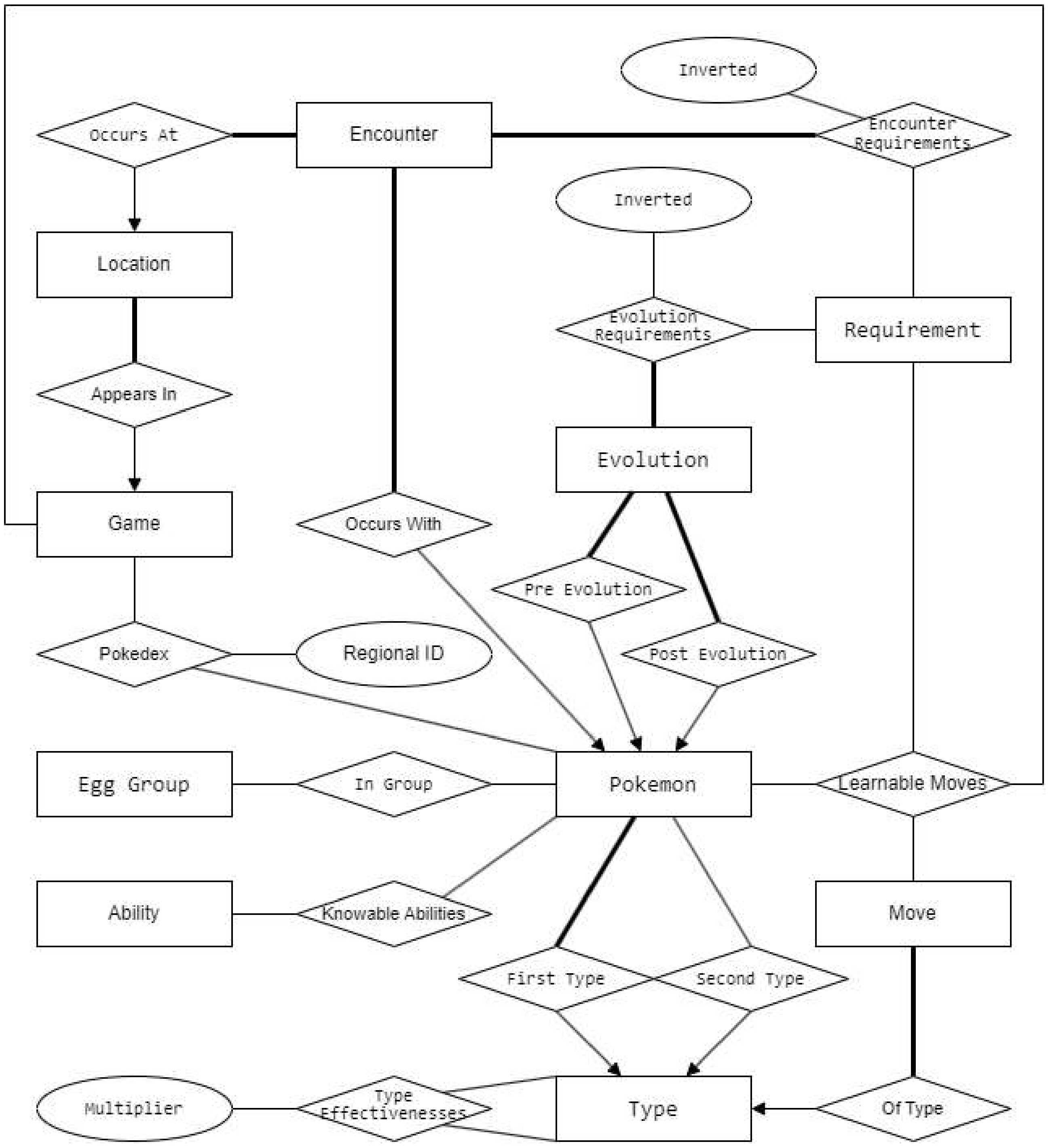
Encounters
Encounters describe where you might find a Pokémon under certain circumstances, how likely is the encounter, and at what level the Pokémon may be.

Egg Groups
Pokémon can breed, but only with other Pokémon in the same Egg Group.

ER Models

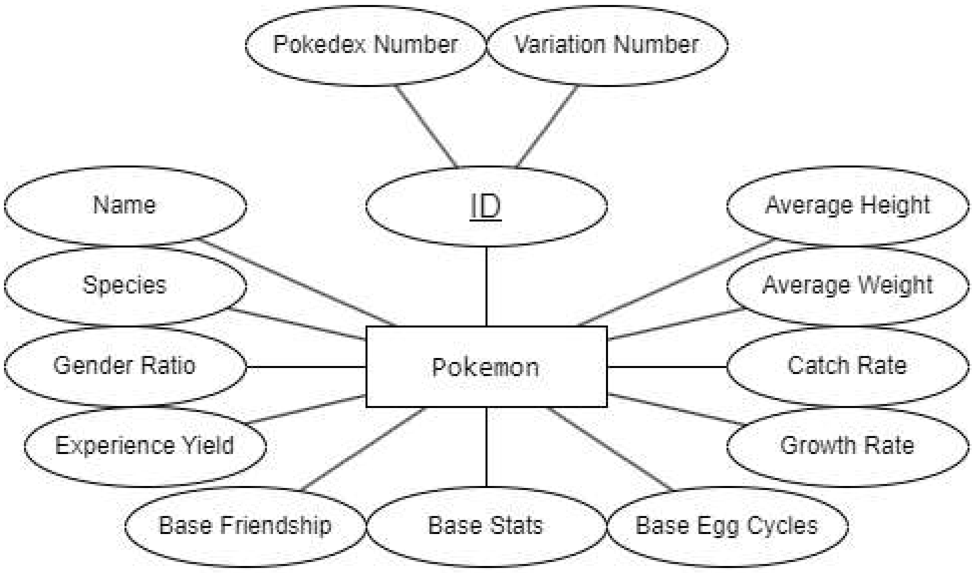
The following ER diagrams show the major components of the Pokémon database:

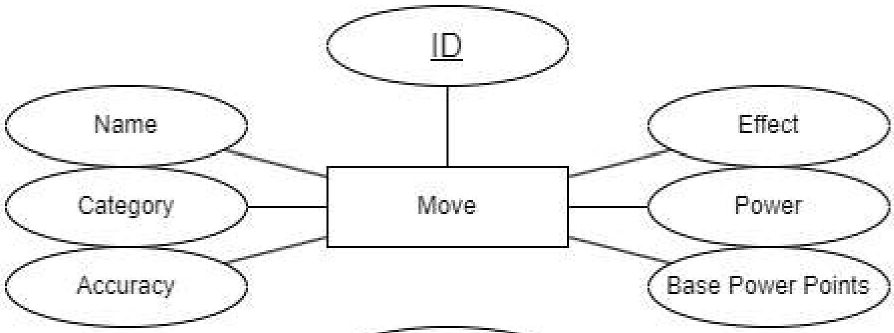
ER diagram of Pokémon relations
Attributes have been removed to make the relations more readable

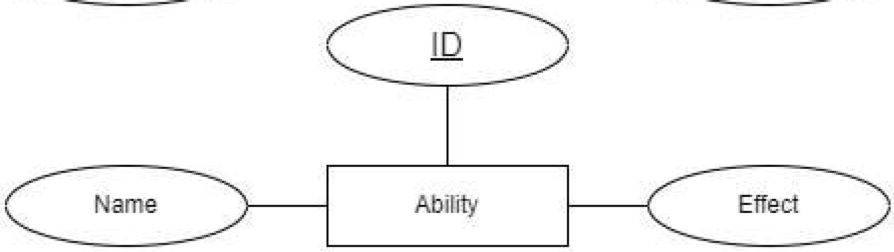


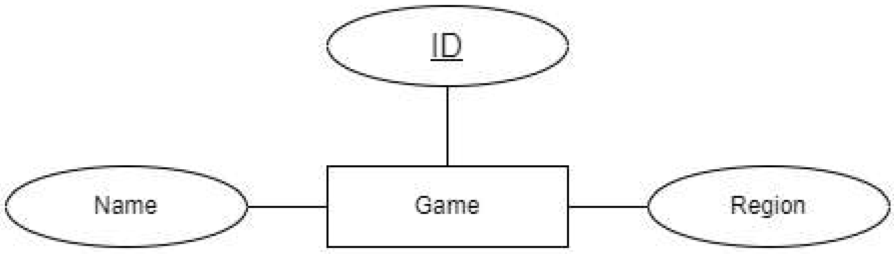
ER diagram of Pokémon attributes

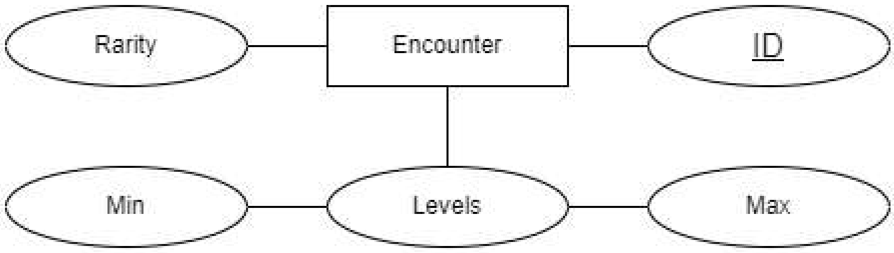
Relations have been removed to make the attributes more readable

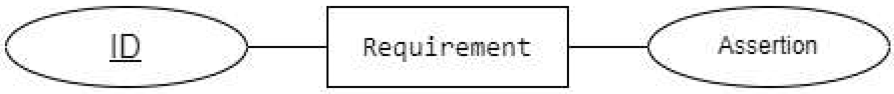

Pokedex Number
Variation Number
ID
Name
Species
Gender Ratio
Experience Yield
Average Height
Average Weight
Catch Rate
Growth Rate
Pokemon
Base Friendship
Base Stats
Base Egg Cycles

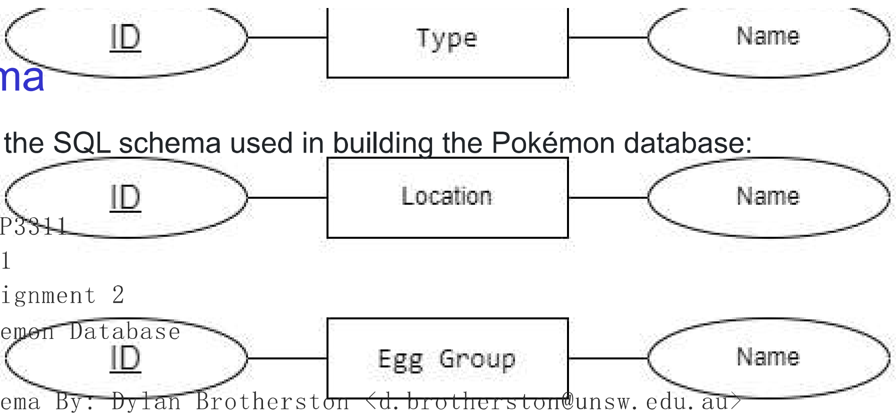

ID
Name
Category
Accuracy
Move
Effect
Power
Base Power Points


ID
Name
Ability
Effect


ID
Name
Game
Region


Rarity
Encounter
ID
Min
Levels
Max


ID
Requirement
Assertion



SQL Schema

The following is the SQL schema used in building the Pokémon database:

```sql
-- COMP3311
-- 24T1
-- Assignment 2
-- Pokemon Database

-- Schema By: Dylan Brotherston <d.brotherston@unsw.edu.au>
-- Version 1.0
-- 2024-03-25

-- This schema is designed to be mostly but not completely accurate to real Pokemo
-- real Pokemon data has an endless amount of edge cases and exceptions
-- so this schema simplifies when necessary

--
-- DOMAINS
--

-- A 1-byte unsigned integer
-- The smallest numeric type postgresql has for us is a Smallint (A 2-byte integer
-- So we need to create a domain to further constrain the value

CREATE DOMAIN Byte AS
    SMALLINT
    CHECK (
        VALUE >= 0
        AND
        VALUE <= 255
    )
;

-- In Pokemon statistic are a value ranging from 1 to 255 inclusive
-- Just a slightly more constrained version of the Byte domain

CREATE DOMAIN Statistic AS
    Byte
    CHECK (
        VALUE >= 1
    )
;

-- A percentage represented as a whole number
-- Mainly used for multipliers
-- so 50 would be a 1/2x multiplier
-- and 200 would be a 2x multiplier
-- etc.
-- As this is mainly used for multipliers, the default value is 100 (1x multiplier

CREATE DOMAIN Percentage AS
    INTEGER
    DEFAULT 100
    CHECK (
        VALUE >= 0
    )
;
```

```

-- A Ratio is a percentage restricted to a maximum of value 100
-- As the name suggests, this is mainly used for ratios
-- eg 47% of something is X (with an implied 53% of the reset being Y)
-- As this is mainly used for ratios, the default value is 50 (50/50)

CREATE DOMAIN Ratio AS
    Percentage
    DEFAULT 50
    CHECK (
        VALUE <= 100
    )
;

-- A Probability is a percentage restricted to a maximum of value 100
-- As the name suggests, this is mainly used for probabilities
-- eg a 76% chance something will happen

CREATE DOMAIN Probability AS
    Percentage
    DEFAULT 0
    CHECK (
        VALUE <= 100
    )
;

-- A distance in meters

CREATE DOMAIN Meters AS
    REAL
    CHECK (
        VALUE >= 0
    )
;

-- A weight in kilograms

CREATE DOMAIN Kilograms AS
    REAL
    CHECK (
        VALUE >= 0
    )
;

--
-- TYPES
--

-- The Primary Key for a Pokemon
-- As some Pokemon have multiple "forms" or "variations" we need two IDs
-- One for the Pokemon itself
-- One for the "forms" or "variations"
--
-- most Pokemon don't have any alternate forms so this is commonly 0
-- If a Pokemon as a "base form" an one (or more) alternative form(s)
-- The base form will have ID 0, and alternative forms will have ID >0

CREATE TYPE _Pokemon_ID AS (
    Pokedex_Number   INTEGER, -- Primary ID:   to differentiate between Pokemon
    Variation_Number INTEGER  -- Secondary ID: to differentiate between forms of
```

```sql
);

-- CREATE TYPE doesn't allow us to add constraints
-- So create a dummy domain (with a leading underscore first)
-- Then create a domain with the same name as the type

CREATE DOMAIN Pokemon_ID AS
    _Pokemon_ID
    CHECK (
        (VALUE).Pokedex_Number   IS NOT NULL
        AND
        (VALUE).Variation_Number IS NOT NULL
    )
;

-- Pokemon (after Generation 1) have 6 different statistic
-- (in Generation 1 Special_Attack and Special_Defense were combined)
-- All Pokemon must have a value for each of these statistics

CREATE TYPE _Stats AS (
    Hit_Points      Statistic,
    Attack          Statistic,
    Defense         Statistic,
    Special_Attack  Statistic,
    Special_Defense Statistic,
    Speed           Statistic
);

-- CREATE TYPE doesn't allow us to add constraints
-- So create a dummy domain (with a leading underscore first)
-- Then create a domain with the same name as the type

CREATE DOMAIN Stats AS
    _Stats
    CHECK (
        (VALUE).Hit_Points      IS NOT NULL
        AND
        (VALUE).Attack          IS NOT NULL
        AND
        (VALUE).Defense         IS NOT NULL
        AND
        (VALUE).Special_Attack  IS NOT NULL
        AND
        (VALUE).Special_Defense IS NOT NULL
        AND
        (VALUE).Speed           IS NOT NULL
    )
;

-- A minimum and maximum value

CREATE TYPE _Range AS (
    MIN INTEGER,
    MAX INTEGER
);

-- CREATE TYPE doesn't allow us to add constraints
-- So create a dummy domain (with a leading underscore first)
-- Then create a domain with the same name as the type

-- An open range has either a minimum or a maximum value
-- The other value can be NULL
-- eg. 5 or more

CREATE DOMAIN Open_Range AS
    _Range
    CHECK (
        (VALUE).Min <= (VALUE).Max
        AND
        (
            (VALUE).Min IS NOT NULL
            OR
            (VALUE).Max IS NOT NULL
        )
    )
;

-- A closed range has both a minimum and a maximum value
-- eg. between 5 and 10

CREATE DOMAIN Closed_Range AS
    Open_Range
    CHECK (
        (VALUE).Min IS NOT NULL
        AND
        (VALUE).Max IS NOT NULL
    )
;

-- How quickly does a Pokemon gain experience
-- In a Pokemon game, this value would determine which mathematical formula
-- is used to calculate the pokemon's current level from their current experience

CREATE TYPE Growth_Rates AS ENUM (
    'Erratic',
    'Fast',
    'Medium Fast',
    'Medium Slow',
    'Slow',
    'Fluctuating'
);

-- Each move can have a Category that determines the type of damage it deals
-- Some moves don't have a category
-- moves without a category deal no damage

CREATE TYPE Move_Categories AS ENUM (
    'Physical', -- Deals Physical damage and uses the Attack and Defense statistic
    'Special',  -- Deals Special damage and uses the Special_Attack and Special_De
    'Status'    -- Deals no damage
);

-- Each game takes place in a region (country)
-- (This is a simplification: some games have multiple regions)
-- (This database will only include the "primary" region of each game)

CREATE TYPE Regions AS ENUM (
    'Kanto',
    'Johto',
    'Hoenn',
    'Sinnoh',
    'Unova',
    'Kalos',
    'Alola',
    'Galar',
    'Hisui',
    'Paldea'
);

--
-- Tables
--

-- A Type is an elemental (or otherwise) category that a Pokemon or a move can hav
-- Eg. Fire type, Water type, Ghost type, Flying type, etc.
-- All Pokemon and moves have a type, Some Pokemon have two types

CREATE TABLE Types (
    -- Primary Key field
    -- SERIAL is a PostgreSQL specific type that auto-increments
    -- PostgreSQL will automatically create a sequence for this table
    ID   SERIAL          PRIMARY KEY,

    -- The only information a type has is its name
    Name Text   NOT NULL UNIQUE
);

-- A Type Effectiveness is how two types interact with each other
-- Specifically how much damage a move of one type does to a Pokemon of another ty
-- A Fire type move will have a 2x multiplier to a Grass type Pokemon
-- A Water type move will have a 0.5x multiplier to a Grass type Pokemon
-- If two types are neutral to each other (1x multiplier) there will be no entry f

CREATE TABLE Type_Effectiveness (
    -- Standard N:M relationship
    -- Where the primary key is a composite of the two foreign keys
    Attacking  INTEGER             REFERENCES Types (ID),
    Defending  INTEGER             REFERENCES Types (ID),

    -- Multiplier for the damage a move of the Attacking_Type does to a Pokemon of
    -- If two types are neutral to each other (1x multiplier) there will be no ent
    Multiplier Percentage NOT NULL,

    -- Each combination of types can only have one effectiveness
    -- So the foreign keys together are the primary key
    PRIMARY KEY (Attacking, Defending)
);

-- Requirements are the conditions that must be met for:
-- a Pokemon to evolve
-- a Move to be learned
-- an Encounter to occur

CREATE TABLE Requirements (
    -- Primary Key field
    -- SERIAL is a PostgreSQL specific type that auto-increments
    -- PostgreSQL will automatically create a sequence for this table
    ID        SERIAL          PRIMARY KEY,

    -- An assertion is a condition that must be met
```

```sql
    -- eg. "Level: 52", "Time of Day: Night", etc.
    Assertion Text   NOT NULL UNIQUE
);

-- A Pokemon
-- The meat and potatoes of the database
-- Each row in this table represents a Pokemon (or variation of a Pokemon)
-- There are a LOT of N:M relationships to this table,
-- so a lot of related information is stored in other tables

CREATE TABLE Pokemon (
    -- Primary Key field
    -- This is a composite type
    -- consisting of the Pokemon's ID and the variation's ID
    ID                Pokemon_ID           PRIMARY KEY,

    -- Basic information
    Name              Text         NOT NULL UNIQUE,                -- Name of the
    Species           Text         NOT NULL,                      -- Species of t
    First_Type        INTEGER      NOT NULL REFERENCES Types (ID), -- Primary t
    Second_Type       INTEGER               REFERENCES Types (ID), -- Secondary
    -- All Pokemon have a primary type, but not all have a secondary type

    -- Characteristics - These are always the same for all pokemon of the same ID
    Average_Height    Meters       NOT NULL, -- Average height of the Pokemon spec
    Average_Weight    Kilograms    NOT NULL, -- Average weight of the Pokemon spec
    Catch_Rate        Statistic    NOT NULL, -- Base catch rate of the Pokemon spe
    Growth_Rate       Growth_Rates NOT NULL, -- Experience curve type of the Pokem
    Experience_Yield  INTEGER      NOT NULL, -- Base experience yield from defeat
    Gender_Ratio      Ratio,                 -- Population gender ratio of the Poke
                                             -- stored as the percentage of the pop
                                             -- a NULL value represents an un-gende

    -- Base stats - These are the starting values for all Pokemon of the same ID
    --            - But each "instance" of a Pokemon can have different stats
    Base_Stats        Stats        NOT NULL, -- Base stats of the Pokemon
    Base_Friendship   Byte         NOT NULL, -- Base friendship of the Pokemon
    Base_Egg_Cycles   INTEGER      NOT NULL  -- Base number of cycles to hatch an
);

-- Egg Groups are a way to categorize what Pokemon can breed with each other
-- Any pokemon with a common egg group can breed with each other
-- With the exception of the Ditto group (just Ditto), which can breed with any ot
-- And the Undiscovered group, which can't breed with any other group (including t

CREATE TABLE Egg_Groups (
    -- Primary Key field
    -- SERIAL is a PostgreSQL specific type that auto-increments
    -- PostgreSQL will automatically create a sequence for this table
    ID   SERIAL          PRIMARY KEY,

    -- Name of the egg group
    -- eg. "Monster", "Human-Like", "Amorphous", etc.
    Name Text   NOT NULL UNIQUE
);

-- What pokemon are in what egg groups

CREATE TABLE In_Group (
    -- Standard N:M relationship
    -- Where the primary key is a composite of the two foreign keys
    Pokemon   Pokemon_ID REFERENCES Pokemon (ID),
    Egg_Group INTEGER    REFERENCES Egg_Groups (ID),

    PRIMARY KEY (Pokemon, Egg_Group)
);

-- An Evolution is a way for a Pokemon to change into another Pokemon
-- This is almost an N:M relationship from Pokemon to Pokemon
-- Except:
-- There are additional requirements for the evolution to occur (`Evolution_Requir
-- Unlike many N:M relationships, this table doesn't use its foreign keys as a com
-- This is because there can be multiple ways for a Pokemon A to evolve into Pokem
-- That is Pokemon A can evolve into Pokemon B via method A,B,C or by X,Y,Z
-- This would be represented by 2 rows in the table, one for each method
-- For an evolution to occur, the Pokemon must meet certain requirements
-- These requirements are stored in the Evolution_Requirements N:M relationship ta

CREATE TABLE Evolutions (
    -- Primary Key field
    -- SERIAL is a PostgreSQL specific type that auto-increments
    -- PostgreSQL will automatically create a sequence for this table
    ID             SERIAL              PRIMARY KEY,

    -- Pre_Evolution is the Pokemon that starts the evolution
    Pre_Evolution  Pokemon_ID NOT NULL REFERENCES Pokemon (ID),
    -- Post_Evolution is the Pokemon that is the result of the evolution
    Post_Evolution Pokemon_ID NOT NULL REFERENCES Pokemon (ID)
    -- The combination of Pre_Evolution and Post_Evolution is *not* unique
    -- because of references from the Evolution_Requirements table
);

-- What conditions must be met for an evolution to occur
-- There may be multiple requirements for an evolution to occur
-- If there are multiple requirements, all of them must be met

CREATE TABLE Evolution_Requirements (
    -- Standard N:M relationship
    -- Where the primary key is a composite of the two foreign keys
    Evolution   INTEGER          REFERENCES Evolutions (ID),
    Requirement INTEGER          REFERENCES Requirements (ID),

    -- Whether the requirement is inverted
    -- ie. the evolution can occur if the requirement is *not* met
    Inverted    BOOLEAN NOT NULL DEFAULT FALSE,

    PRIMARY KEY (Evolution, Requirement)
);

-- A game in the Pokemon series
-- eg. Pokemon Red, Pokemon Gold, Pokemon Black, etc.
-- the common word "Pokemon" is omitted from the name

CREATE TABLE Games (
    -- Primary Key field
    -- SERIAL is a PostgreSQL specific type that auto-increments
    -- PostgreSQL will automatically create a sequence for this table
    ID     SERIAL           PRIMARY KEY,

    -- Name of the game
    Name   Text    NOT NULL UNIQUE,
    -- The region the game is set in
    Region Regions NOT NULL
);

-- A location in a game
-- eg. Route 1, Route 2, Azalea Town, Saffron City, etc.

CREATE TABLE Locations (
    -- Primary Key field
    -- SERIAL is a PostgreSQL specific type that auto-increments
    -- PostgreSQL will automatically create a sequence for this table
    ID         SERIAL          PRIMARY KEY,

    -- Name of the location
    Name       Text    NOT NULL,
    -- The game the location is in
    Appears_In INTEGER NOT NULL REFERENCES Games (ID),

    -- Each named location can only appear in a game once
    UNIQUE (Name, Appears_In)
);

-- A "Pokedex entry" is a description of a Pokemon with a specific game
-- With a specific game, pokemon can have different pokedex numbers
-- The pokedex number that is stored in the Pokemon table is called the "National
-- The game specific pokedex number that is stored in the Pokedex table is called

CREATE TABLE Pokedex (
    -- standard N:M relationship
    -- Where the primary key is a composite of the two foreign keys
    National_ID Pokemon_ID          REFERENCES Pokemon (ID),
    Game        INTEGER             REFERENCES Games (ID),

    -- The Variation_Number is still the same as in the National_ID
    -- ie if Ash-Greninja as National_ID 658 and a Variation_Number of 1
    -- and a Regional_ID of 9 in X they still have the same Variation_Number of 1
    Regional_ID INTEGER    NOT NULL,

    PRIMARY KEY (National_ID, Game)
);

-- An encounter is a way for a pokemon to found within a game
-- (only wild pokemon are recorded in this database)
-- (not gift pokemon, interactions with the environment, or trainer battles)

CREATE TABLE Encounters (
    -- Primary Key field
    -- SERIAL is a PostgreSQL specific type that auto-increments
    -- PostgreSQL will automatically create a sequence for this table
    ID          SERIAL                PRIMARY KEY,

    -- An encounter occurs with a specific pokemon
    Occurs_With Pokemon_ID   NOT NULL REFERENCES Pokemon (ID),
    -- An encounter occurs in a specific location
    Occurs_At   INTEGER      NOT NULL REFERENCES Locations (ID),

    -- Encounters have a rarity
    -- the rarity how likely it is to encounter a specific pokemon out of all the
    -- (rarities (especially for early games) are estimates)
    -- (the sum of all the rarities in a location *should* be 100, but will most l
    Rarity      Probability  NOT NULL,
    -- The range of levels the pokemon can be encountered at (inclusive on both en
    -- eg (5, 10) means the pokemon can be encountered at levels 5, 6, 7, 8, 9, 10
    -- (7, 7) means the pokemon can only be encountered at level 7
    Levels      Closed_Range NOT NULL
);

-- What conditions must be met for an encounter to occur
-- generally this is the requirements is what method of movement the player is usi
-- eg. walking, surfing, fishing, etc.

CREATE TABLE Encounter_Requirements (
    -- Standard N:M relationship
    -- Where the primary key is a composite of the two foreign keys
    Encounter   INTEGER         REFERENCES Encounters (ID),
    Requirement INTEGER         REFERENCES Requirements (ID),

    -- Whether the requirement is inverted
    -- ie. the encounter can occur if the requirement is *not* met
    Inverted    BOOLEAN NOT NULL DEFAULT FALSE,

    PRIMARY KEY (Encounter, Requirement)
);

-- A move is a way for a pokemon to attack, or otherwise effect another pokemon (e

CREATE TABLE Moves (
    -- Primary Key field
    -- SERIAL is a PostgreSQL specific type that auto-increments
    -- PostgreSQL will automatically create a sequence for this table
    ID                 SERIAL                   PRIMARY KEY,

    -- Name of the move
    Name               Text            NOT NULL UNIQUE,
    -- The effect the move has (if any) (this is just some flavour text)
    Effect             Text,

    -- The type of the move
    Of_Type            INTEGER         NOT NULL REFERENCES Types (ID),

    -- The category of the move (physical, special, or status)
    Category           Move_Categories,
    -- The power of the move (amount of damage it does)
    POWER               Statistic,
    -- The accuracy of the move (how likely it is to hit)
    Accuracy           Probability,
    -- The power points of the move (how many times the move can be used)
    Base_Power_Points INTEGER
);

-- This is one hell of a table
-- it has over 1M rows
-- This table is used to determine what moves a pokemon can learn
-- *But* that set of moves is dependent on the game the pokemon is in
-- So this is a *4* way relationship
-- an N:M:O:P relationship if you will

CREATE TABLE Learnable_Moves (
    -- Definitely *not* a standard N:M relationship
    -- but the same principle applies
    Learnt_By   Pokemon_ID NOT NULL REFERENCES Pokemon (ID),
    Learnt_In   INTEGER    NOT NULL REFERENCES Games (ID),
    Learnt_When INTEGER    NOT NULL REFERENCES Requirements (ID),
    Learns      INTEGER    NOT NULL REFERENCES Moves (ID),

    PRIMARY KEY (Learnt_By, Learnt_In, Learnt_When, Learns)
);

-- An ability is a special effect that a pokemon can have
-- This is different from a move in pokemon cannot learn new abilities
-- they are either *born* with an ability and they will always have it

CREATE TABLE Abilities (
    -- Primary Key field
    -- SERIAL is a PostgreSQL specific type that auto-increments
    -- PostgreSQL will automatically create a sequence for this table
    ID     SERIAL          PRIMARY KEY,

    -- Name of the ability
    Name   Text   NOT NULL UNIQUE,
    -- The effect the ability has (if any) (this is just some flavour text)
    Effect Text   NOT NULL
);

-- A pokemon can only have one ability
-- But they can have one of multiple abilities
-- eg Dreepy can have one of "Clear Body" or "Infiltrator" or "Cursed Body"

CREATE TABLE Knowable_Abilities (
    -- Standard N:M relationship
    -- Where the primary key is a composite of the two foreign keys
    Known_By Pokemon_ID           REFERENCES Pokemon (ID),
    Knows    INTEGER              REFERENCES Abilities (ID),

    -- Some pokemon can hidden abilities
    -- this is an ability that the pokemon itself doesn't have
    -- but it's "children" can have (a recessive gene if you will)
    Hidden   BOOLEAN     NOT NULL,

    -- As a pokemon cannot learn a new ability
    -- there is no foreign key to Requirements table

    PRIMARY KEY (Known_By, Knows)
);

```