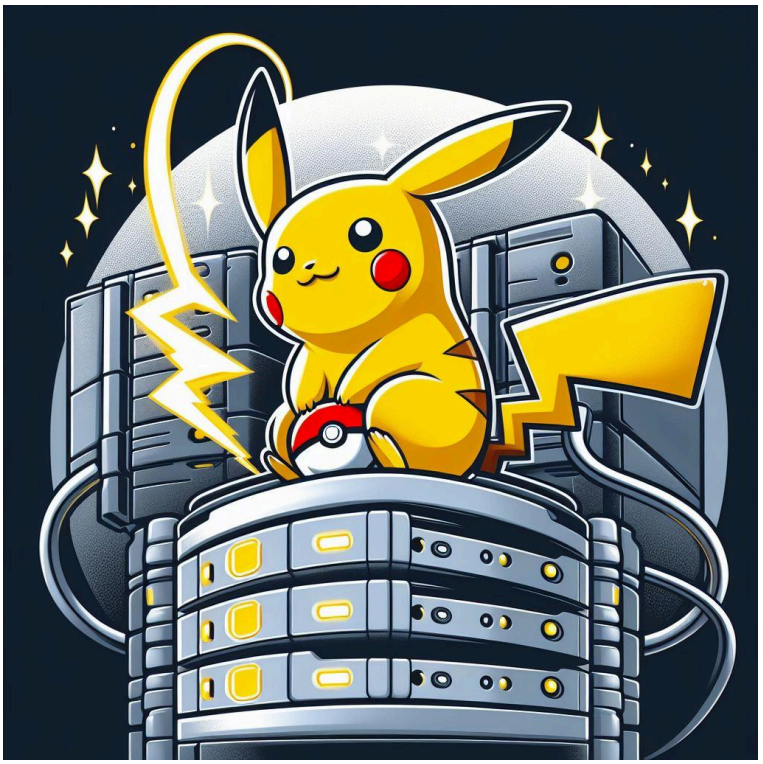


Database Management System

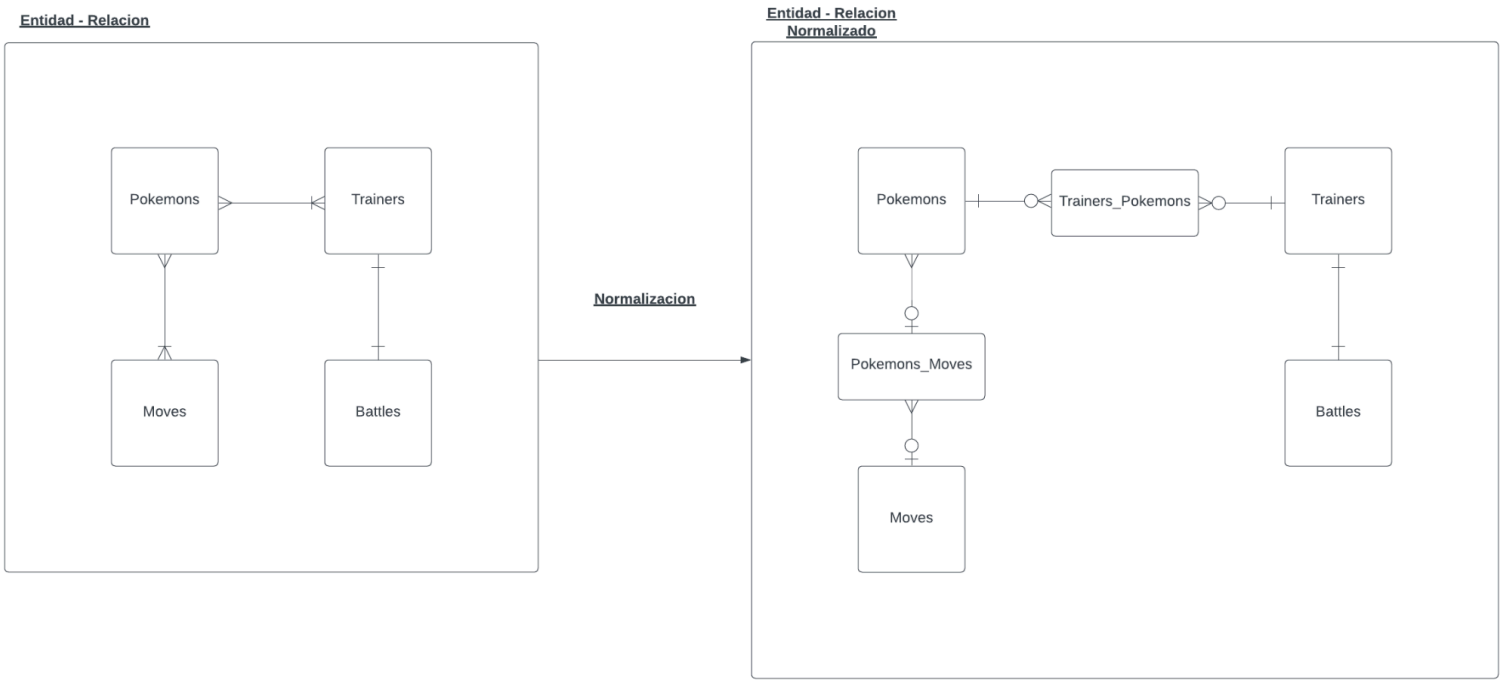
Austin Jenner Beltran
Pedro Capó Lozano
Lluís Muñoz Barceló
1r Daw Presencial

Austin Jenner Beltran, Pedro Capó Lozano, Lluís Muñoz Barceló

1.0 Introduction

Pokémon is a popular franchise with a vast array of characters and data associated with them. Managing Pokémon data efficiently is crucial for any database system. This project aims to design a database management system for Pokémon data, focusing on the first generation which would be the Kanto region.

2.0 Entity-Relationship Diagram (ERD)



3.0 Database Schema

The database schema consists of the following tables:

1. **Pokémon:** Contains information about each Pokémon, such as ID, name, type, etc.

```
-- Table: Pokemon
CREATE TABLE Pokemons (
    ID_Pokemon INT AUTO_INCREMENT PRIMARY KEY,
    Name VARCHAR(255),
    Type1 VARCHAR(50),
    Type2 VARCHAR(50),
    Species VARCHAR(255),
    Generation INT,
    Attack INT,
    Defense INT,
    HP INT,
    Speed INT,
    Ability VARCHAR(255),
    Hidden_Ability VARCHAR(255)
);
```

2. **Trainers:** Stores data about Pokémon trainers, including ID, name, and region.

```
-- Table: Trainer
CREATE TABLE Trainers (
    ID_Trainer INT AUTO_INCREMENT PRIMARY KEY,
    Name VARCHAR(255),
    City VARCHAR(255)
);
```

3. **Battles:** Records details of Pokémon battles, such as battle ID, date, winner, etc.

```
-- Table: Battles
CREATE TABLE Battles (
    ID_Battle INT AUTO_INCREMENT PRIMARY KEY,
    ID_Trainer1 INT,
    ID_Trainer2 INT,
    Date DATE,
    Winner INT,
    FOREIGN KEY (ID_Trainer1) REFERENCES trainers(ID_Trainer),
    FOREIGN KEY (ID_Trainer2) REFERENCES trainers(ID_Trainer),
    FOREIGN KEY (Winner) REFERENCES trainers(ID_Trainer)
);
```

4. **Trainers_Pokemon:** A many-to-many relationship table linking trainers to their Pokemon.

```
-- Table: Pokemon_Trainers
CREATE TABLE Trainers_Pokemons (
   ID_Pokemon_Trainers INT AUTO_INCREMENT PRIMARY KEY,
   ID_Trainers INT,
   ID_Pokemon INT,
   FOREIGN KEY (ID_Pokemon) REFERENCES pokemons (ID_Pokemon),
   FOREIGN KEY (ID_Trainers) REFERENCES trainers(ID_Trainer)
);
```

5. **Moves:** Stores information about Pokémon moves, including move ID, name, type, power, etc.

```
-- Table: Moves
CREATE TABLE Moves (
   ID_Move INT AUTO_INCREMENT PRIMARY KEY,
   Name VARCHAR(255),
   Type VARCHAR(50),
   Category VARCHAR(50),
   Power INT,
   Accuracy INT,
   PP INT,
   Description TEXT
);
```

6. **Pokémon_Moves:** A many-to-many relationship table linking Pokémon to their Moves.

```
-- Table: Pokemon_Moves
CREATE TABLE Pokemons_Moves (
   ID_Pokemon_Moves INT AUTO_INCREMENT PRIMARY KEY,
   ID_Pokemon INT,
   ID_Move INT,
   FOREIGN KEY (ID_Pokemon) REFERENCES pokemons (ID_Pokemon),
   FOREIGN KEY (ID_Move) REFERENCES Moves(ID_Move)
);
```

4.0 Indexes

Indexes are created to improve the speed of data retrieval operations. In our Pokémon database, we can create indexes on frequently searched columns such as Pokémon name, trainer name, and move name to enhance query performance.

```
-- Index for the Pokémon name column in the Pokemons table
CREATE INDEX idx_pokemon_name ON Pokemons (Name);

-- Index for the Trainer name column in the Trainers table
CREATE INDEX idx_trainer_name ON Trainers (Name);

-- Index for the Move name column in the Moves table
CREATE INDEX idx_move_name ON Moves (Name);

-- Index for the Date column in the Battles table
CREATE INDEX idx_battle_date ON Battles (Date);
```

5.0 Transactions

Transactions ensure data integrity and consistency. In Pokémon battles, it's essential to ensure that changes to the database (such as updating Pokémon stats or recording battle results) are atomic and either fully completed or fully rolled back in case of failure.

```
-- Start a new transaction
START TRANSACTION;

-- Example operations within the transaction
INSERT INTO Trainers (Name, City) VALUES ('Brock', 'Pewter City');

-- Check for errors
IF ROW_COUNT() = 0 THEN
   SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'An error occurred. Rolling
back transaction.';
   ROLLBACK;
ELSE
   COMMIT;
END IF;
```

```
-- Verify changes
SELECT * FROM Trainers WHERE Name = 'Brock';
```

6.0 Users and User Permissions

User management and permissions control access to the database. Different users may have different privileges, such as read-only access or full CRUD (Create, Read, Update, Delete) permissions.

We are going to create 3 users and grant them access to “Pokemon” table.

```
-- Create the users
CREATE USER 'Lluis'@localhost IDENTIFIED BY 'password1';
GRANT SELECT ON Pokemons TO 'Lluis'@'localhost';

CREATE USER 'Austin'@localhost IDENTIFIED BY 'password2';
GRANT SELECT ON Pokemons TO 'Austin'@'localhost';

CREATE USER 'Pedro'@localhost IDENTIFIED BY 'password3';
GRANT SELECT ON Pokemons TO 'Pedro'@'localhost';

-- verify if the users has been made
SELECT USER FROM mysql.user;
```

	User
1	aj
2	root
3	root
4	root
5	root
6	Austin
7	Lluis
8	Pedro
9	mariadb.sys
10	root

7.0 Conclusion

This project demonstrates the importance of indexes, transactions, users, and user permissions in managing Pokémon data effectively. By implementing these database management techniques, we ensure data consistency, integrity, and security, providing a robust foundation for Pokémon data management.

8.0 EXTRAS

We wanted to implement some extras as procedures and triggers to automate our database

PROCEDURES

Table Pokemon

- Add_Pokemon (Name, Type1, Type2, Species, Generation, Attack, Defense, HP, Speed, Ability, Hidden_Abbilityl):

  Procedure to add a new Pokémon to the database.

```
-- Procedure for adding a Pokemon
DELIMITER //
CREATE PROCEDURE AddPokemon(
   IN p_name VARCHAR(255),
   IN p_type1 VARCHAR(50),
   IN p_type2 VARCHAR(50),
   IN p_species VARCHAR(255),
   IN p_generation INT,
   IN p_attack INT,
   IN p_defense INT,
   IN p_hp INT,
   IN p_speed INT,
   IN p_ability VARCHAR(255),
   IN p_hidden_ability VARCHAR(255)
)
BEGIN
   INSERT INTO Pokemons (Name, Type1, Type2, Species, Generation, Attack,
Defense, HP, Speed, Ability, Hidden_Ability)
   VALUES (p_name, p_type1, p_type2, p_species, p_generation, p_attack,
p_defense, p_hp, p_speed, p_ability, p_hidden_ability);
END //
DELIMITER ;
```

- Modify_Pokemon(ID_Pokemon, Name, Type1, Type2, Species, Generation, Attack, Defense, HP, Speed, Ability, Hidden_Abbility):

  Procedure to modify the data of an existing Pokémon.

```
-- Procedure for updating a Pokemon
DELIMITER //
CREATE PROCEDURE UpdatePokemon(
   IN p_id INT,
   IN p_name VARCHAR(255),
   IN p_type1 VARCHAR(50),
   IN p_type2 VARCHAR(50),
   IN p_species VARCHAR(255),
   IN p_generation INT,
   IN p_attack INT,
   IN p_defense INT,
   IN p_hp INT,
   IN p_speed INT,
   IN p_ability VARCHAR(255),
   IN p_hidden_ability VARCHAR(255)
)
BEGIN
   UPDATE Pokemons
   SET Name = p_name,
       Type1 = p_type1,
       Type2 = p_type2,
       Species = p_species,
       Generation = p_generation,
       Attack = p_attack,
       Defense = p_defense,
       HP = p_hp,
       Speed = p_speed,
       Ability = p_ability,
       Hidden_Ability = p_hidden_ability
   WHERE ID_Pokemon = p_id;
END //
DELIMITER ;
```

- Delete_Pokemon(ID_Pokemon):

  Procedure to delete a Pokémon from the database.

```
-- Procedure for deleting a Pokemon
DELIMITER //
CREATE PROCEDURE DeletePokemon(
   IN p_id INT
)
BEGIN
   DELETE FROM Pokemons WHERE ID_Pokemon = p_id;
END //
DELIMITER ;
```

Table Trainers

- Add_Trainer(Name, City):

  Procedure to add a new coach to the database.

```
-- Procedure for adding a Trainer
DELIMITER //
CREATE PROCEDURE AddTrainer(
   IN p_name VARCHAR(255),
   IN p_city VARCHAR(255)
)
BEGIN
   INSERT INTO Trainers (Name, City) VALUES (p_name, p_city);
END //
DELIMITER ;
```

- Modify_Trainer(ID_Trainer, Name, City):

  Procedure to modify the data of an existing trainer.

```
-- Procedure for updating a Trainer
DELIMITER //
CREATE PROCEDURE UpdateTrainer(
   IN p_id INT,
   IN p_name VARCHAR(255),
   IN p_city VARCHAR(255)
)
BEGIN
   UPDATE Trainers SET Name = p_name, City = p_city WHERE ID_Trainer =
p_id;
END //
DELIMITER ;
```

- Delete_Trainer(ID_Trainer):

  Procedure to delete a trainer from the database.

```
-- Procedure for deleting a Trainer
DELIMITER //
CREATE PROCEDURE DeleteTrainer(
   IN p_id INT
)
BEGIN
   DELETE FROM Trainers WHERE ID_Trainer = p_id;
END //
DELIMITER ;
```

Table Moves

- Add_Movement (Name, Type, Category, Power, Accuracy, PP, Description):

  Procedure to add a new movement to the database.

```
-- Procedure for adding a Move
DELIMITER //
CREATE PROCEDURE AddMove(
    IN p_name VARCHAR(255),
    IN p_type VARCHAR(50),
    IN p_category VARCHAR(50),
    IN p_power INT,
    IN p_accuracy INT,
    IN p_pp INT,
    IN p_description TEXT
)
BEGIN
    INSERT INTO Moves (Name, Type, Category, Power, Accuracy, PP,
Description)
    VALUES (p_name, p_type, p_category, p_power, p_accuracy, p_pp,
p_description);
END //
DELIMITER ;
```

- Modify_Movement (ID_Moves, Name, Type, Category, Power, Accuracy, PP, Description):

  Procedure to modify the data of an existing movement.

```
DELIMITER //
CREATE PROCEDURE UpdateMove(
    IN p_id INT,
    IN p_name VARCHAR(255),
    IN p_type VARCHAR(50),
    IN p_category VARCHAR(50),
    IN p_power INT,
    IN p_accuracy INT,
    IN p_pp INT,
    IN p_description TEXT
)
BEGIN
    UPDATE Moves
    SET Name = p_name, Type = p_type, Category = p_category, Power =
p_power,
        Accuracy = p_accuracy, PP = p_pp, Description = p_description
    WHERE ID_Move = p_id;
END //
DELIMITER ;
```

- Delete_Movement(ID_Move):

  Procedure to delete a movement from the database.

```
-- Procedure for deleting a Move
DELIMITER //
CREATE PROCEDURE DeleteMove(
    IN p_id INT
)
BEGIN
    DELETE FROM Moves WHERE ID_Move = p_id;
END //
```

```
DELIMITER ;
```

Table Pokemon_Moves

- Add_Pokemon_Moves(ID_Pokemon, ID_Move):
  Procedure to add a move to a specific Pokémon.

```
DELIMITER //
CREATE PROCEDURE AddPokemonMove(
   IN p_pokemon_id INT,
   IN p_move_id INT
)
BEGIN
   INSERT INTO Pokemons_Moves (ID_Pokemon, ID_Move)
   VALUES (p_pokemon_id, p_move_id);
END //
DELIMITER ;
```

- Delete_Pokemon_Moves(ID_Pokemon, ID_Move):
  Procedure to remove a move from a specific Pokémon.

```
DELIMITER //
CREATE PROCEDURE DeletePokemonMove(
   IN p_pokemon_move_id INT
)
BEGIN
   DELETE FROM Pokemons_Moves WHERE ID_Pokemon_Moves = p_pokemon_move_id;
END //
DELIMITER ;
```

Table Battles

- Register_Battle(ID_Trainer1, ID_Trainer2, Date, Winner):

```
-- Procedure for adding a battle
DELIMITER //
CREATE PROCEDURE AddBattle(
   IN p_trainer1_id INT,
   IN p_trainer2_id INT,
   IN p_date DATE,
   IN p_winner_id INT
)
BEGIN
   INSERT INTO Battles (ID_Trainer1, ID_Trainer2, Date, Winner)
   VALUES (p_trainer1_id, p_trainer2_id, p_date, p_winner_id);
END //
DELIMITER ;
```

TRIGGERS

Table Pokemon:

- **Trigger_Before_Add_Pokemon**:Executed before adding a new Pokémon to the database. This trigger can be used to verify that the data entered is valid, for **example**, that the Pokémon name is not empty or that the attack, defense, HP and speed values are within a valid range.

```
DELIMITER //
CREATE TRIGGER Trigger_Before_Add_Pokemon
BEFORE INSERT ON Pokemons
FOR EACH ROW
```

```
BEGIN
    -- Check if the Pokémon name is not empty
    IF NEW.Name = '' OR NEW.Name IS NULL THEN
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Pokemon name cannot be
empty';
    END IF;

    -- Check if attack, defense, HP, and speed values are within a valid range
    IF NEW.Attack < 0 OR NEW.Attack > 255 THEN
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Invalid Attack value';
    END IF;

    IF NEW.Defense < 0 OR NEW.Defense > 255 THEN
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Invalid Defense value';
    END IF;

    IF NEW.HP < 0 OR NEW.HP > 255 THEN
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Invalid HP value';
    END IF;

    IF NEW.Speed < 0 OR NEW.Speed > 255 THEN
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Invalid Speed value';
    END IF;
END //
DELIMITER ;
```

Table Trainer

- **Trigger_Before_Add_Trainer**: Executed before adding a new trainer to the database. This trigger can be used to verify that the data entered is valid, for example, that the coach name is not empty or that the coach city is a valid city.

```
DELIMITER //
CREATE TRIGGER Trigger_Before_Add_Trainer
BEFORE INSERT ON Trainers
FOR EACH ROW
BEGIN
    -- Check if the trainer name is not empty
    IF NEW.Name = '' OR NEW.Name IS NULL THEN
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Trainer name cannot be
empty';
    END IF;

    -- Check if the city name is not empty
    IF NEW.City = '' OR NEW.City IS NULL THEN
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Trainer name cannot be
empty';
    END IF;

END //
DELIMITER ;
```

Table Moves

- **Trigger_Before_Add_Movement**: Executed before adding a new movement to the database. This trigger can be used to verify that the data entered is valid, for example, that the movement name is not empty or that the movement type is valid.

```
DELIMITER //
CREATE TRIGGER Trigger_Before_Add_Movement
BEFORE INSERT ON Moves
FOR EACH ROW
BEGIN
   -- Check if the movement name is not empty
   IF NEW.Name = '' OR NEW.Name IS NULL THEN
       SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Movement name cannot be
empty';
   END IF;

   -- Check if the movement type is valid
   IF NEW.Type NOT IN ('Normal', 'Fire', 'Water', 'Electric', 'Grass', 'Ice',
'Fighting', 'Poison', 'Ground', 'Flying', 'Psychic', 'Bug', 'Rock', 'Ghost',
'Dragon', 'Dark', 'Steel', 'Fairy') THEN
       SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Invalid movement type';
   END IF;
END //
DELIMITER ;
```