CREATE TABLE attackingpower (

category VARCHAR(20),

Fire FLOAT,

water FLOAT,

Grass FLOAT,

lightning FLOAT,

psychic FLOAT,

metal FLOAT

);

INSERT INTO attackingpower (category, Fire, water, Grass, lightning, psychic, metal)

VALUES

('Fire', 1, 0.5, 2, 0.5, 1, 2),

('Grass', 0.5, 2, 0.5, 1, 1, 0.5),

('Lightning', 1, 2, 0.5, 0.5, 1, 1),

('Metal', 0.5, 0.5, 1, 0.5, 1, 0.5),

('Psychic', 1, 1, 1, 1, 0.5, 0.5),

('Water', 2, 0.5, 0.5, 1, 1, 2);

CREATE TABLE users (

userId INT NOT NULL AUTO\_INCREMENT PRIMARY KEY,

username VARCHAR(20) NOT NULL UNIQUE,

Avatar LONGBLOB,

no\_of\_cards INT DEFAULT 4,

COINS INT DEFAULT 0,

points INT DEFAULT 0,

Played\_matches INT,

won\_matches INT

);

CREATE TABLE pokemon (

PokeId INT NOT NULL AUTO\_INCREMENT PRIMARY KEY,

name VARCHAR(50) NOT NULL UNIQUE,

Health\_Points FLOAT,

Attack\_Points FLOAT,

Category VARCHAR(50) NOT NULL,

poke\_image LONGBLOB NOT NULL

);

CREATE TABLE battle (

matchId INT NOT NULL AUTO\_INCREMENT PRIMARY KEY,

user1\_points INT DEFAULT 0,

user2\_points INT DEFAULT 0,

user1\_id INT NOT NULL UNIQUE,

user2\_id INT NOT NULL UNIQUE,

winner VARCHAR(20) NOT NULL

);

CREATE TABLE owns (

userId INT NOT NULL,

pokeId INT NOT NULL PRIMARY KEY,

matches\_left INT DEFAULT 0,

FOREIGN KEY (userId) REFERENCES users(userId),

FOREIGN KEY (pokeId) REFERENCES pokemon(PokeId)

);

1. ***To fetch the cards***

DELIMITER //

CREATE PROCEDURE GetUserCards(IN user\_id INT)

BEGIN

-- Declare variables

DECLARE done INT DEFAULT FALSE;

DECLARE user\_card\_id INT;

DECLARE user\_card\_name VARCHAR(50);

-- Cursor for fetching user's cards

DECLARE cur CURSOR FOR

SELECT pokemon.PokeId, pokemon.name

FROM owns

Where owns.userId=user\_id;

-- Declare continue handler for cursor

DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;

-- Create temporary table to store user's cards

CREATE TEMPORARY TABLE temp\_user\_cards (

PokeId INT,

name VARCHAR(50)

);

-- Open the cursor

OPEN cur;

-- Fetch cards and insert into temporary table

read\_loop: LOOP

FETCH cur INTO user\_card\_id, user\_card\_name;

IF done THEN

LEAVE read\_loop;

END IF;

INSERT INTO temp\_user\_cards (PokeId, name) VALUES (user\_card\_id, user\_card\_name);

END LOOP;

-- Close the cursor

CLOSE cur;

-- Select the user's cards from the temporary table

SELECT \* FROM temp\_user\_cards;

-- Drop the temporary table

DROP TEMPORARY TABLE IF EXISTS temp\_user\_cards;

END //

DELIMITER ;

***2)fetch 4 cards***

DELIMITER //

CREATE PROCEDURE GetUserSelectedCards(IN user\_id INT, IN card1\_id INT, IN card2\_id INT, IN card3\_id INT, IN card4\_id INT)

BEGIN

-- Declare variables for selected cards

DECLARE card1\_name VARCHAR(50);

DECLARE card2\_name VARCHAR(50);

DECLARE card3\_name VARCHAR(50);

DECLARE card4\_name VARCHAR(50);

-- Get the names of selected cards

SELECT name INTO card1\_name FROM pokemon WHERE PokeId = card1\_id;

SELECT name INTO card2\_name FROM pokemon WHERE PokeId = card2\_id;

SELECT name INTO card3\_name FROM pokemon WHERE PokeId = card3\_id;

SELECT name INTO card4\_name FROM pokemon WHERE PokeId = card4\_id;

-- Select the selected cards

SELECT card1\_name, card2\_name, card3\_name, card4\_name;

END //

DELIMITER ;

***3)create account***

DELIMITER //

CREATE FUNCTION CreateAccount(

p\_username VARCHAR(50)

)

RETURNS INT

DETERMINISTIC

BEGIN

DECLARE userId INT;

DECLARE pokemonCount INT DEFAULT 0;

-- Check if the username already exists

DECLARE username\_count INT;

SELECT COUNT(\*) INTO username\_count FROM users WHERE username = p\_username;

IF username\_count > 0 THEN

SIGNAL SQLSTATE '45000' SET MESSAGE\_TEXT = 'Username already exists';

ELSE

-- Insert new user into the users table

INSERT INTO users (username)

VALUES (p\_username);

-- Get the newly created user's ID

SELECT LAST\_INSERT\_ID() INTO userId;

-- Attempt to assign four random unassigned Pokémon to the new user

WHILE pokemonCount < 4 DO

-- Select a random unassigned Pokémon

SET @rand\_poke := (SELECT pokeId FROM pokemon WHERE pokeId NOT IN (SELECT pokeId FROM owns) ORDER BY RAND() LIMIT 1);

-- Assign the selected Pokémon to the new user

UPDATE owns

SET userId = userId

WHERE pokeId = @rand\_poke AND userId IS NULL;

-- Check if the assignment was successful

SET pokemonCount = (SELECT COUNT(\*) FROM owns WHERE userId = userId);

END WHILE;

SELECT userId; -- Return the newly created user's ID

END IF;

RETURN userId;

END //

DELIMITER ;

***4)RANK***

DELIMITER //

CREATE FUNCTION `RANK`(user\_Id INT)

RETURNS INT

DETERMINISTIC

BEGIN

DECLARE user\_rank INT;

-- Create a temporary table to store rankings

CREATE TEMPORARY TABLE temp\_ranking (

userId INT,

points INT,

rank INT AUTO\_INCREMENT PRIMARY KEY

);

-- Populate temporary table with user rankings based on points

INSERT INTO temp\_ranking (userId, points)

SELECT userId, points

FROM users

ORDER BY points DESC;

-- Retrieve the rank of the specified user from the temporary table

SELECT rank INTO user\_rank

FROM temp\_ranking

WHERE userId = user\_Id;

-- Drop the temporary table

DROP TEMPORARY TABLE IF EXISTS temp\_ranking;

RETURN user\_rank;

END //

DELIMITER ;

**5) update owner table after match**

DELIMITER //

CREATE PROCEDURE UpdateOwnershipAfterMatch(

IN p\_userId INT,

IN p\_pokeId INT

)

BEGIN

DECLARE remaining\_matches INT;

-- Get the remaining matches of the Pokémon for the user

SELECT remaining\_matches\_of\_pokemon INTO remaining\_matches

FROM owns

WHERE userId = p\_userId AND pokeId = p\_pokeId;

-- Check if the Pokémon is owned by the user and has remaining matches

IF remaining\_matches IS NOT NULL AND remaining\_matches > 0 THEN

-- Decrement the remaining matches of the Pokémon

UPDATE owns

SET remaining\_matches\_of\_pokemon = remaining\_matches - 1

WHERE userId = p\_userId AND pokeId = p\_pokeId;

-- Check if the remaining matches become zero and delete the row

IF remaining\_matches - 1 = 0 THEN

DELETE FROM owns

WHERE userId = p\_userId AND pokeId = p\_pokeId;

END IF;

SELECT 'Ownership updated successfully.' AS message;

ELSE

SELECT 'Invalid user ID or Pokémon ID, or Pokémon has no remaining matches.' AS message;

END IF;

END //

DELIMITER ;

**6) check pokemon expiry**

DELIMITER //

CREATE FUNCTION CheckPokemonExpiry(

p\_userId INT,

p\_pokeId INT

)

RETURNS VARCHAR(100)

BEGIN

DECLARE matches\_left INT;

-- Get the remaining matches of the Pokémon for the user

SELECT remaining\_matches\_of\_pokemon INTO matches\_left

FROM owns

WHERE userId = p\_userId AND pokeId = p\_pokeId;

IF matches\_left IS NOT NULL THEN

RETURN CONCAT('Number of matches left: ', matches\_left);

ELSE

RETURN 'Invalid user ID or Pokémon ID.';

END IF;

END //

DELIMITER ;

**7)purchase pokemon**

DELIMITER //

CREATE PROCEDURE PurchaseCard(

IN p\_userId INT,

IN p\_pokeName VARCHAR(50);

)

BEGIN

DECLARE card\_cost INT;

DECLARE user\_coins INT;

DECLARE p\_pokeId INT;

-- Get the cost of the card (assuming each Pokémon costs 100 coins)

SET card\_cost = 100;

-- Get the user's current coins

SELECT COINS INTO user\_coins FROM users WHERE userId = p\_userId;

SELECT pokeId INTO p\_pokeId FROM pokemon WHERE name=p\_pokeName;

-- Check if the user has enough coins to purchase the card

IF user\_coins >= card\_cost THEN

-- Deduct the card cost from the user's coins

UPDATE users SET COINS = user\_coins - card\_cost WHERE userId = p\_userId;

INSERT INTO owns (userId, pokeId, remaining\_matches\_of\_pokemon)

VALUES (p\_userId, p\_pokeId, 3);

SELECT 'Card purchased successfully.' AS message;

ELSE

SELECT 'Insufficient coins to purchase the card.' AS message;

END IF;

END //

DELIMITER ;

**8)battle**

DELIMITER //

CREATE PROCEDURE StartBattleWithCategory(

IN p\_userId1 INT,

IN p\_userId2 INT

)

BEGIN

-- Temporary tables to store selected Pokémon for each user, including category

CREATE TEMPORARY TABLE temp\_user1\_cards (

PokeId INT,

name VARCHAR(50),

Attack\_Points FLOAT,

Health\_Points FLOAT,

Category VARCHAR(50)

);

CREATE TEMPORARY TABLE temp\_user2\_cards (

PokeId INT,

name VARCHAR(50),

Attack\_Points FLOAT,

Health\_Points FLOAT,

Category VARCHAR(50)

);

-- User 1 selects 4 cards for the battle

INSERT INTO temp\_user1\_cards (PokeId, name, Attack\_Points, Health\_Points, Category)

SELECT p.PokeId, p.name, p.Attack\_Points, p.Health\_Points, p.Category

FROM pokemon p

INNER JOIN owns o ON p.PokeId = o.pokeId

WHERE o.userId = p\_userId1

LIMIT 4;

-- User 2 selects 4 cards for the battle

INSERT INTO temp\_user2\_cards (PokeId, name, Attack\_Points, Health\_Points, Category)

SELECT p.PokeId, p.name, p.Attack\_Points, p.Health\_Points, p.Category

FROM pokemon p

INNER JOIN owns o ON p.PokeId = o.pokeId

WHERE o.userId = p\_userId2

LIMIT 4;

DECLARE user1\_selected\_pokeid INT;

DECLARE user2\_selected\_pokeid INT;

-- Battle Loop

WHILE (1) DO

-- User 1's turn

SELECT PokeId INTO user1\_selected\_pokeid FROM temp\_user1\_cards LIMIT 1;

-- Check if User 1 has any cards left

IF user1\_selected\_pokeid IS NULL THEN

-- User 2 wins, update points and coins

UPDATE users SET points = points + 10, COINS = COINS + 500 WHERE userId = p\_userId2;

LEAVE;

END IF;

-- User 2's turn

SELECT PokeId INTO user2\_selected\_pokeid FROM temp\_user2\_cards WHERE Health\_Points > 0 LIMIT 1;

-- Check if User 2 has any cards left

IF user2\_selected\_pokeid IS NULL THEN

-- User 1 wins, update points and coins

UPDATE users SET points = points + 10, COINS = COINS + 500 WHERE userId = p\_userId1;

LEAVE;

END IF;

DECLARE u1\_category,u2\_category VARCHAR(50);

SELECT category INTO u1\_category FROM temp\_user1\_cards WHERE pokeId= user1\_selected\_pokeid;

SELECT category INTO u2\_category FROM temp\_user2\_cards WHERE pokeId= user2\_selected\_pokeid;

-- Calculate attack effectiveness based on category table

DECLARE attack\_multiplier FLOAT;

SELECT u2\_category INTO attack\_multiplier FROM attackingpower WHERE category = u1\_category;

-- Update User 1's Pokémon's health points based on attack effectiveness

UPDATE temp\_user1\_cards

SET Health\_Points = Health\_Points - (attack\_multiplier \* (SELECT Attack\_Points FROM temp\_user2\_cards WHERE PokeId = user2\_selected\_pokeid))

WHERE PokeId = user1\_selected\_pokeid AND Health\_Points > 0;

UPDATE temp\_user2\_cards

SET Health\_Points = Health\_Points - (attack\_multiplier \* (SELECT Attack\_Points FROM temp\_user1\_cards WHERE PokeId = user1\_selected\_pokeid))

WHERE PokeId = user2\_selected\_pokeid AND Health\_Points > 0;

-- Check if User 1's Pokémon has fainted

IF (SELECT Health\_Points FROM temp\_user1\_cards WHERE PokeId = user1\_selected\_pokeid) <= 0 THEN

-- Remove the Pokémon from the battle and give 10 points to User 2

DELETE FROM temp\_user1\_cards WHERE PokeId = user1\_selected\_pokeid;

UPDATE users SET points = points + 10 WHERE userId = p\_userId2;

-- Check if User 2's Pokémon has fainted

IF (SELECT Health\_Points FROM temp\_user2\_cards WHERE PokeId = user2\_selected\_pokeid) <= 0 THEN

-- Remove the Pokémon from the battle and give 10 points to User 1

DELETE FROM temp\_user2\_cards WHERE PokeId = user2\_selected\_pokeid;

UPDATE users SET points = points + 10 WHERE userId = p\_userId1;

END IF;

END WHILE;

-- Drop temporary tables

DROP TEMPORARY TABLE IF EXISTS temp\_user1\_cards;

DROP TEMPORARY TABLE IF EXISTS temp\_user2\_cards;

END //

DELIMITER ;

***9)notify the user***

DELIMITER //

CREATE PROCEDURE GetUserSelectedCards(

IN p\_userId INT,

IN card1\_id INT,

IN card2\_id INT,

IN card3\_id INT,

IN card4\_id INT

)

BEGIN

DECLARE card\_id INT;

DECLARE card\_name VARCHAR(50);

DECLARE matches\_left INT;

-- Loop through each selected card

DECLARE card\_cursor CURSOR FOR

SELECT pokeId FROM owns WHERE userId = p\_userId AND pokeId IN (card1\_id, card2\_id, card3\_id, card4\_id);

OPEN card\_cursor;

card\_loop: LOOP

FETCH card\_cursor INTO card\_id;

IF card\_id IS NULL THEN

LEAVE card\_loop;

END IF;

-- Check if the card is not expired

SELECT matches\_left INTO matches\_left FROM owns WHERE userId = p\_userId AND pokeId = card\_id;

IF matches\_left = 0 THEN

SIGNAL SQLSTATE '45000' SET MESSAGE\_TEXT = CONCAT('Card with ID ', card\_id, ' has expired and cannot be selected.');

ELSEIF matches\_left = 1 THEN

-- Notify the user that the card has only 1 match left

INSERT INTO notifications (userId, message, notification\_date)

VALUES (p\_userId, CONCAT('Your Pokémon with ID ', card\_id, ' has only 1 match left.'), NOW());

ELSE

SELECT name INTO card\_name FROM pokemon WHERE PokeId = card\_id;

-- You can do something with the selected card, like inserting into a temporary table

END IF;

END LOOP;

CLOSE card\_cursor;

END //

DELIMITER ;

***10)eligible or not***

DELIMITER //

CREATE PROCEDURE StartBattleWithCategory(

IN p\_userId1 INT,

IN p\_userId2 INT

)

BEGIN

DECLARE user1\_card\_count INT;

DECLARE user2\_card\_count INT;

-- Check if User 1 has 4 selected cards

SELECT COUNT(\*) INTO user1\_card\_count FROM temp\_user1\_cards;

IF user1\_card\_count < 4 THEN

SIGNAL SQLSTATE '45000' SET MESSAGE\_TEXT = 'User 1 does not have 4 selected cards. Battle cannot start.';

END IF;

-- Check if User 2 has 4 selected cards

SELECT COUNT(\*) INTO user2\_card\_count FROM temp\_user2\_cards;

IF user2\_card\_count < 4 THEN

SIGNAL SQLSTATE '45000' SET MESSAGE\_TEXT = 'User 2 does not have 4 selected cards. Battle cannot start.';

END IF;

-- Battle logic

-- ...

END //

DELIMITER ;