CREATE ACCOUNT

Working✅

```
DELIMITER //
CREATE PROCEDURE create_user(
    IN p_username VARCHAR(20)
  )
  BEGIN
    DECLARE user_exists INT;

    -- Check if the user already exists
    SELECT COUNT(*) INTO user_exists FROM new_users WHERE username =
p_username;

    IF user_exists = 0 THEN
      -- User does not exist, so create a new user
      INSERT INTO new_users (username) VALUES (p_username);

      -- Get the ID of the newly created user
      SET @user_id = LAST_INSERT_ID();

      -- Assign four random Pokemon to the new user
      SET @counter = 0;
      WHILE @counter < 4 DO
        -- Get a random available Pokemon ID
        SET @random_pokemon_id = (SELECT pokeId FROM new_pokemon
WHERE pokeId NOT IN (SELECT pokeId FROM new_owns) ORDER BY RAND()
LIMIT 1);

        -- Check if the user already owns this Pokemon
        SET @already_owned = (SELECT COUNT(*) FROM new_owns WHERE
pokeId = @random_pokemon_id);

        IF @already_owned = 0 THEN
          -- User does not already own this Pokemon, so assign it
          INSERT INTO new_owns (userId, pokeId) VALUES (@user_id,
@random_pokemon_id);
          SET @counter = @counter + 1;
        END IF;
      END WHILE;
    END IF;
    END //
DELIMITER ;
```

GET USER 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 new_pokemon.PokeId, new_pokemon.name
   FROM new_owns
   INNER JOIN new_pokemon ON new_owns.pokeId = new_pokemon.PokeId
   WHERE new_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 ;
CALL GetUserCards(8);
```

GETRANK

```
DELIMITER //

CREATE PROCEDURE GetRank(IN p_userId INT)
BEGIN
    DECLARE user_rank INT;

    -- Get the rank of the user
    SELECT COUNT(*) + 1 INTO user_rank
    FROM new_users
    WHERE points > (SELECT points FROM new_users WHERE userId = p_userId);

    -- If no one has more points than the user, their rank is 1
    IF user_rank IS NULL THEN
        SET user_rank = 1;
    END IF;

    -- Display the rank
    SELECT user_rank;
END //

DELIMITER ;
```

```
CALL GetRank(9);
```

GETALLRANKS

```
DELIMITER //

CREATE PROCEDURE GetAllRanks()
BEGIN

   SELECT
      userid,
      username,
      points,
      (SELECT COUNT(*) + 1 FROM new_users AS u WHERE u.points > nu.points) AS
user_rank
   FROM new_users AS nu
   ORDER BY points DESC;
END //

DELIMITER ;
call getallranks;
```

CheckExpiry

```
DELIMITER //

CREATE FUNCTION CheckPokemonExpiry(
   p_userId INT,
   p_pokeId INT
)
RETURNS VARCHAR(100)
DETERMINISTIC
BEGIN
   DECLARE matches_left INT;

   -- Get the remaining matches of the Pokémon for the user
   SELECT MatchesRemaining 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 ;
SELECT CheckPokemonExpiry(8, 104);
```

Purchase card

```
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 new_users WHERE userId = p_userId;

    SELECT PokeId INTO p_pokeId FROM new_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 new_users SET COINS = user_coins - card_cost WHERE userId =
p_userId;
 UPDATE new_users SET no_of_cards = no_of_cards +1 WHERE userId = p_userId;
```

```
        -- Insert the purchased card into the owns table with 3 remaining matches
        INSERT INTO new_owns (userId, pokeId, MatchesRemaining) 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 ;
CALL PurchaseCard(8, 'Charmander');
```

AVAILABLE POKE

```
DELIMITER //
CREATE PROCEDURE availablepoke()
BEGIN
select pokeid, name from new_pokemon where pokeId not in (select pokeId from
new_owns);
END //
DELIMITER ;
call availablepoke;
```

INSERT CARD(into u1 u2 battle table)

```
DELIMITER //
CREATE PROCEDURE insertbattlecard( IN poke_id INT)
BEGIN
DECLARE uid INT;
DECLARE pokecount INT;
DECLARE HP,AP FLOAT;
DECLARE cat VARCHAR(50);
DECLARE pname VARCHAR(50);
SELECT COUNT(*) INTO pokecount FROM battle_cards WHERE pokeid=poke_id;
IF pokecount=0 THEN
                    SELECT userid,name,health_points,attack_points, category INTO
          uid,pname ,hp,ap,cat FROM pokemon natural join owns WHERE
          pokeid=poke_id;
```

```
        INSERT INTO battle_cards VALUE
        (uid,poke_id,pname,ap,hp,cat);
        UPDATE owns
        SET MatchesRemaining=MatchesRemaining-1
        WHERE pokeid=poke_id AND userId=uid;
        END IF;
END//
Delimiter ;
call insertbattlecard(80);
```

CHECK ELIGIBILITY

```
DELIMITER //
CREATE FUNCTION check_eligilibilty( user_id INT)
RETURNS INT DETERMINISTIC
BEGIN
DECLARE card INT;
SELECT no_of_cards INTO card FROM users WHERE userid=user_id;
IF card<4 THEN
RETURN 0;
ELSE
RETURN 1;
END IF;
END //
DELIMITER ;
```

PLAYING

```
DELIMITER //
CREATE FUNCTION playing(p1 INT, p2 INT,match_no INT)
RETURNS INT DETERMINISTIC
BEGIN
   DECLARE u1, u2 INT;
   DECLARE cat1, cat2 VARCHAR(50);
   DECLARE effect1, effect2 FLOAT;
   DECLARE new_health1, new_health2 FLOAT;
   DECLARE ap1, ap2 INT;
```

```
    -- Get the user IDs of the players
    SELECT userid INTO u1 FROM battle_cards WHERE pokeid = p1;
    SELECT userid INTO u2 FROM battle_cards WHERE pokeid = p2;

    -- If one of the players is not found, award points and coins to the other player
    IF u1 IS NULL THEN
        UPDATE new_users
        SET coins = coins + 500,
            points = points + 10
        WHERE userId = u2;
UPDATE new_battle
SET winner=u2 WHERE matchId=match_no;
UPDATE new_users
SET Played_matches=Played_matches+1,won_matches=won_matches+1
WHERE userId=u2;
SET Played_matches=Played_matches+1
WHERE userId=u1;
        RETURN u2;

    ELSEIF u2 IS NULL THEN
        UPDATE new_users
        SET coins = coins + 500,
            points = points + 10
        WHERE userId = u1;
UPDATE new_battle
SET winner=u1 WHERE matchId=match_no;
        RETURN u1;

    ELSE
        -- Get the categories of the Pokémon
        SELECT category INTO cat1 FROM battle_cards WHERE pokeId = p1;
        SELECT category INTO cat2 FROM battle_cards WHERE pokeId = p2;

        -- Get the attack effectiveness
        SELECT
            CASE cat1
                WHEN 'Fire' THEN Fire
                WHEN 'Water' THEN water
                WHEN 'Grass' THEN Grass
                WHEN 'Lightning' THEN lightning
                WHEN 'Psychic' THEN psychic
                WHEN 'Metal' THEN metal
            END INTO effect1
        FROM attackingpower WHERE category = cat2;

        SELECT
            CASE cat2
                WHEN 'Fire' THEN Fire
                WHEN 'Water' THEN water
                WHEN 'Grass' THEN Grass
                WHEN 'Lightning' THEN lightning
                WHEN 'Psychic' THEN psychic
                WHEN 'Metal' THEN metal
            END INTO effect2
        FROM attackingpower WHERE category = cat1;

        -- Get the attack points of the Pokémon
        SELECT attack_points INTO ap1 FROM battle_cards WHERE pokeid = p1;
        SELECT attack_points INTO ap2 FROM battle_cards WHERE pokeid = p2;

        -- Update the health points of the Pokémon
        UPDATE battle_cards
        SET health_points = health_points - (effect1 * ap1)
        WHERE pokeid = p2;

        UPDATE battle_cards
        SET health_points = health_points - (effect2 * ap2)
        WHERE pokeid = p1;

          DELETE FROM battle_cards WHERE health_points<=0;

        RETURN 0;
    END IF;
END //
DELIMITER ;
```

UPDATE BATTLE TABLE

```
DELIMITER //
CREATE FUNCTION updatebattle( u1 INT,u2 INT)
```

```
RETURNS INT DETERMINISTIC
BEGIN
DECLARE mid INT;
INSERT INTO new_battle (user1_id,user2_id,winner) VALUES
(U1,U2,0);
SELECT MAX(matchId) INTO mid FROM new_battle;
RETURN mid;
END //
DELIMITER ;
```

REMOVE ROW FROM OWNS AFTER REMAINING MATCHES=0

```
DELIMITER //

CREATE TRIGGER remove_row_after_update
AFTER UPDATE ON owns
FOR EACH ROW
BEGIN
   IF NEW.MatchesRemaining = 0 THEN
      DELETE FROM owns WHERE userId = OLD.userId AND pokeId = OLD.pokeId;

      UPDATE users
      SET no_of_cards = no_of_cards - 1
      WHERE userId = OLD.userId;
   END IF;
END;
//

DELIMITER ;
```