```
USE lab2;


# all_users
# Skapa en vy med en kolumn för username och en kolumn med no_of_friends där det är
uträknat hur många vänner varje användare har. Ska kunna köras med SELECT * FROM
all_users och då visa en rad för varje användare.
DROP VIEW IF EXISTS all_users;
CREATE VIEW all_users AS
      SELECT username, COUNT(*) AS no_of_friends
      FROM Friends
      JOIN users ON users.id = Friends.u_id
      GROUP BY username;

SELECT * FROM all_users;


# friends_list
# Skapa en vy med användarnamn (username) och namnen på deras kompisar
(friendname). Ska t ex gå att göra SELECT friendname FROM friends_list WHERE
username = "arya" för att få ut lista med vänner till Arya.
DROP VIEW IF EXISTS friends_list;
CREATE VIEW friends_list AS
    SELECT User.username, Friend.username AS friendname
      FROM Friends
      JOIN Users AS User ON User.id = Friends.u_id
      JOIN Users AS Friend ON Friend.id = Friends.f_id;

SELECT friendname
FROM friends_list
WHERE username = 'arya';


# user_email()
# Skapa en procedure som när den anropas med CALL user_email(); ger en lista med
epost, fname och lname för alla användare i DB.
DROP PROCEDURE IF EXISTS user_email;
DELIMITER //
CREATE PROCEDURE user_email()
BEGIN
      SELECT email, fname, lname FROM users;
END//
DELIMITER ;

CALL user_email();


# add_hobby(hobby_name, description)
# Skapa en procedure som lägger till en hobby i tabellen hobbies.
DROP PROCEDURE IF EXISTS add_hobby;
DELIMITER //
CREATE PROCEDURE add_hobby(IN hobby_name TEXT, IN description TEXT)
BEGIN
      INSERT INTO Hobbies (name, description) VALUES (hobby_name, description);
END//
DELIMITER ;

DELETE FROM Hobbies WHERE name = 'Coding';
CALL add_hobby('Coding', 'Writing code');
```

```
# add_user(username, pass, fname, lname, email, age)
# Skapa en procedure som lägger till en user. För alla nya användare som läggs till
ska även läggas till en koppling så att de får hobbyn "Swords". Proceduren ska
returnera/svara med det id som nya användaren lagts till på.
DROP PROCEDURE IF EXISTS add_user;
DELIMITER //
CREATE PROCEDURE add_user(IN username TEXT, IN pass TEXT, IN fname TEXT, IN lname
TEXT, IN email TEXT, IN age INT)
BEGIN
      INSERT INTO Users (username, pass, fname, lname, email, age) VALUES
(username, pass, fname, lname, email, age);
      SET @user_id = LAST_INSERT_ID();
      INSERT INTO Users_hobbies (u_id, h_id) VALUES (@user_id, 1);
      SELECT @user_id;
END//
DELIMITER ;

DELETE FROM Users_hobbies WHERE u_id = (SELECT id FROM Users WHERE username =
'Admin');
DELETE FROM Users WHERE username = 'Admin';

CALL add_user('Admin', '1234', 'Admin', 'Adminsson', 'admin@example.com', 44);


# add_friendship(id_a, id_b)
# Vid lagring av nya vänner så måste det ligga två rader för att kopplingarna ska
fungera.  Skriv en procedure som tar två användar-id och lagrar (id_a, id_b) och
(id_b, id_a) i tabellen för Friends.
DROP PROCEDURE IF EXISTS add_friendship;
DELIMITER //
CREATE PROCEDURE add_friendship(IN id_a INT, IN id_b INT)
BEGIN
      INSERT INTO Friends (u_id, f_id) VALUES (id_a, id_b);
      INSERT INTO Friends (u_id, f_id) VALUES (id_b, id_a);
END//
DELIMITER ;

CALL add_friendship(2, 8);


# agecheck
# Gör triggers som innan insert och update kontrollerar att ålder på användare är
15 eller mer än 15. Om ålder är angiven till mindre än 15 så ska ett meddelande
skrivas ut och ingen insert eller update köras.
DROP PROCEDURE IF EXISTS validate_age;
DROP TRIGGER IF EXISTS age_check_insert;
DROP TRIGGER IF EXISTS age_check_update;

DELIMITER //
CREATE PROCEDURE validate_age(IN age INT)
BEGIN
      IF age < 15 THEN
          SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Måste vara 15 eller äldre för
att registrera sig';
      END IF;
END //
```

```
DELIMITER ;

CREATE TRIGGER age_check_insert
BEFORE INSERT ON Users FOR EACH ROW CALL validate_age(NEW.age);

CREATE TRIGGER age_check_update
BEFORE UPDATE ON Users FOR EACH ROW CALL validate_age(NEW.age);

DELETE FROM Users WHERE username = 'a';

# Numbers below 15 in the INSERT or the UPDATE will show error
INSERT INTO Users (username, pass, email, fname, lname, age) VALUES ('a', 'b', 'c',
'd', 'e', 14);
UPDATE Users SET age = 14 WHERE username = 'a';


# greeting
# Gör en funktion som returnerar en valfri hälsningsfras tillsammans med
användarens förnamn (att använda vid t ex vid mailutskick). Ska kunna köras i
exempelvis SELECT email, greeting(fname) FROM users;
DROP FUNCTION IF EXISTS greeting;

DELIMITER //
CREATE FUNCTION greeting(name TEXT)
RETURNS TEXT
NO SQL
BEGIN
      RETURN concat('Hello, ', name, '!');
END //
DELIMITER ;

SELECT email, greeting(fname) FROM Users;


# suggest_friends(username)
# gör en procedure som för en användare ger en lista med som mest 3 st användare
(username) som den inte är kompis men som någon av den användarens kompisar är
kompis med. I listan ska det även så med vem som är gemensam kompis. Om det inte
finns finns några kompisförslag så ska det skrivas ut ett meddelande om det
istället.
DROP PROCEDURE IF EXISTS suggest_friends;

DELIMITER //

CREATE PROCEDURE suggest_friends(IN username TEXT)
BEGIN
      SET @user_id := (SELECT id FROM Users WHERE Users.username = username);

      SELECT
            GROUP_CONCAT(DISTINCT CommonFriends.username SEPARATOR ', ') AS
common_friends,
            FriendSuggestions.username AS friend_suggestion
      FROM Friends AS UserFriends
            JOIN Friends AS FriendsOfFriends ON UserFriends.f_id =
FriendsOfFriends.u_id
          # Join user tables to get the names of the friends instead of the IDs
            JOIN Users AS CommonFriends ON UserFriends.f_id = CommonFriends.id
            JOIN Users AS FriendSuggestions ON FriendsOfFriends.f_id =
FriendSuggestions.id
```

```
      WHERE UserFriends.u_id = @user_id
            AND FriendsOFFriends.f_id != @user_id # Do not suggest the user
themselves
            AND FriendsOFFriends.f_id NOT IN # Do not suggest users that are
already friends with the user
        (
              SELECT f_id
              FROM Friends
              WHERE u_id = @user_id
        )
      GROUP BY friend_suggestion # Do not suggest the same user twice
      LIMIT 3;

      IF FOUND_ROWS() = 0 THEN
          SELECT 'Inga vänförslag' AS Status;
      END IF;
END//

CALL suggest_friends('jamie');


# delete_user(id)
# Skapa en stored procedure för att ta bort en användare. (Den ska i princip göra
samma sak som om vi hade haft ON DELETE CASCADE på lämpliga ställen (men nu har vi
inte det och måste lösa det med en procedure istället).)
DROP PROCEDURE IF EXISTS delete_user;

DELIMITER //
CREATE PROCEDURE delete_user(id int)
BEGIN
      DELETE FROM Friends WHERE Friends.f_id = id;
      DELETE FROM Friends WHERE Friends.u_id = id;
      DELETE FROM Users_hobbies WHERE Users_hobbies.u_id = id;
      DELETE FROM Users WHERE Users.id = id;
END//
DELIMITER ;

CALL delete_user(1);


# egen procedure
# Skapa en egen procedure som gör något som är meningsfullt och användbart med
databasen. Hitta på något eget som kan passa och vara intressant. Skriv tydliga
kommentarer och queries som visar hur den används.
DROP PROCEDURE IF EXISTS update_user;

DELIMITER //
CREATE PROCEDURE update_user(id INT, age INT) # create procedure to update the age
of a user
BEGIN
      UPDATE Users SET Users.age = age WHERE Users.id = id; # set the new age on
the user
END//
DELIMITER ;

CALL update_user(2, 18); # set age to 18 for user with id 2


# egen function
```

```
# Skapa en egen function som gör något som är meningsfullt och användbart med databasen. Hitta på något eget som kan passa och vara intressant. Skriv tydliga kommentarer och queries som visar hur den används.
DROP FUNCTION IF EXISTS passcheck;

DELIMITER //
CREATE FUNCTION passcheck(id int, value text) # create function that checks if password matches for a user
RETURNS BOOLEAN
READS SQL DATA # will read data from database
BEGIN
    SET @user_pass := (SELECT pass FROM Users WHERE Users.id = id); # store the correct password in @user_pass variable

    IF @user_pass = value THEN RETURN TRUE; # correct password returns true
    Else
    RETURN FALSE; # incorrect password returns false
    END IF;
END//
DELIMITER ;

SELECT username, passcheck(id,'VkDvqBIeq') FROM Users; # Check which user has the password 'VkDvqBIeq'
```