```
DROP DATABASE lab5;
CREATE DATABASE lab5;
USE lab5;

# Ser till att vi har samma inställningar
SET sql_mode='ONLY_FULL_GROUP_BY,STRICT_TRANS_TABLES';
SET FOREIGN_KEY_CHECKS = 1;

# 1. Skapa tabeller
# a) Users (id, username)
DROP TABLE IF EXISTS Users;
CREATE TABLE Users
(
      id       int PRIMARY KEY AUTO_INCREMENT,
      username varchar(50) NOT NULL UNIQUE
);

# b) Accounts (id, amount)
DROP TABLE IF EXISTS Accounts;
CREATE TABLE Accounts
(
      id     int PRIMARY KEY AUTO_INCREMENT,
      amount decimal(13, 2) NOT NULL
);

# c) Transfers (id, account_id, amount, note, t_datetime)
DROP TABLE IF EXISTS Transfers;
CREATE TABLE Transfers
(
      id         int PRIMARY KEY AUTO_INCREMENT,
      account_id int                    NOT NULL REFERENCES Accounts (id),
      amount     decimal(13, 2)          NOT NULL,
      note       text,
      t_datetime datetime DEFAULT NOW() NOT NULL
);

# d) Owners (user_id, account_id)
DROP TABLE IF EXISTS Owners;
CREATE TABLE Owners
(
      user_id    int NOT NULL REFERENCES Users (id),
      account_id int NOT NULL REFERENCES Accounts (id)
);

ALTER TABLE Owners
      ADD CONSTRAINT fk_user_id FOREIGN KEY (user_id) REFERENCES Users (id)
    ON UPDATE CASCADE
    ON DELETE CASCADE;

ALTER TABLE Owners
      ADD CONSTRAINT fk_account_id FOREIGN KEY (account_id) REFERENCES Accounts
(id)
    ON UPDATE CASCADE
    ON DELETE CASCADE;


# 2. Skapa innehåll
INSERT INTO Users (id, username)
VALUES (1, 'aha12'),
```

```
       (2, 'fjk34'),
       (3, 'wth334'),
       (4, 'sddt004'),
       (5, 'lots456'),
       (6, 'zcvs348');

INSERT INTO Accounts (id, amount)
VALUES (1, 305123.32),
       (2, 1002314.66),
       (3, 757341.51),
       (4, 463841.01),
       (5, 554545.43),
       (6, 687890.12),
       (7, 305030.10),
       (8, 1003040.00),
       (9, 7770231.75),
       (10, 467890.41),
       (11, 557001.30),
       (12, 688890.49);

INSERT INTO Owners (user_id, account_id)
VALUES (1, 1),
       (2, 3),
       (3, 4),
       (4, 6),
       (5, 5),
       (6, 2),
       (1, 7),
       (2, 9),
       (3, 12),
       (4, 11),
       (5, 8),
       (6, 10),
       (4, 1),
       (6, 3),
       (2, 4),
       (3, 6),
       (3, 5),
       (6, 7),
       (4, 1);


# 3. users_accounts
DROP VIEW IF EXISTS users_accounts;
CREATE VIEW users_accounts AS
SELECT users.id AS user_id, username AS user_name, accounts.id AS account_id,
accounts.amount AS account_amount
FROM Users
      JOIN Owners ON Owners.user_id = Users.id
      JOIN Accounts ON Accounts.id = Owners.account_id;

SELECT * FROM users_accounts WHERE user_id = 3 ORDER BY account_id ASC;
SELECT user_id FROM users_accounts WHERE account_id = 7;


# 4. deposit(account_id, amount)
DROP PROCEDURE IF EXISTS add_transfer;
DELIMITER //
CREATE PROCEDURE add_transfer(IN account_id INT, IN amount decimal(13, 2))
```

```
BEGIN
    START TRANSACTION;

    SET @existing_account_id := (SELECT id FROM Accounts WHERE id = account_id);
    SET @current_amount := (SELECT Accounts.amount FROM Accounts WHERE id =
account_id);
    SET @new_amount := @current_amount + amount;

    IF @existing_account_id IS NULL THEN
        SELECT 'Error: account does not exist' AS status;
    ELSEIF amount <= 0 THEN
        SELECT 'Error: amount is not > 0' AS status;
    ELSE

        UPDATE Accounts SET amount = @new_amount WHERE id = account_id;

        INSERT INTO Transfers (account_id, amount, note, t_datetime)
        VALUES (account_id, amount, CONCAT('(', NOW(), ') DEPOSIT: ', amount,
'kr'), NOW());

            SELECT * FROM Transfers WHERE id = LAST_INSERT_ID();
      END IF;

    COMMIT;
END//
DELIMITER ;

CALL add_transfer(3,5000);      # Ska fungera
CALL add_transfer(3, 0);        # Inte större än 0
CALL add_transfer(3, -30);      # Inte större än 0
CALL add_transfer(192873,5000); # Kontot finns ej

SELECT * FROM Transfers; # Endast den första har lagts till


# 5. withdraw(account_id, amount)
DROP PROCEDURE IF EXISTS withdraw;
DELIMITER //
CREATE PROCEDURE withdraw(IN account_id INT, IN amount decimal(13, 2))
BEGIN
    START TRANSACTION;

    SET @existing_account_id := (SELECT id FROM Accounts WHERE id = account_id);
    SET @current_amount := (SELECT Accounts.amount FROM Accounts WHERE id =
account_id);
    SET @new_amount := @current_amount - amount;

    IF @existing_account_id IS NULL THEN
        SELECT 'Error: account does not exist' AS status;
    ELSEIF amount <= 0 THEN
        SELECT 'Error: amount is not > 0' AS status;
    ELSEIF @new_amount < 0 THEN
        SELECT 'Error: amount is too large' AS status;
    ELSE
        UPDATE Accounts SET amount = @new_amount WHERE id = account_id;

        INSERT INTO Transfers (account_id, amount, note, t_datetime)
        VALUES (account_id, -amount, CONCAT('(', NOW(), ') WITHDRAW: ', -amount,
'kr'), NOW());
```

```
            SELECT * FROM Transfers WHERE id = LAST_INSERT_ID();
      END IF;

    COMMIT;
END//
DELIMITER ;

CALL withdraw(3,1000);       # Ska fungera
CALL withdraw(3, 0);         # Inte större än 0
CALL withdraw(3, -30);       # Inte större än 0
CALL withdraw(3, 348957983); # Kan inte ta mer pengar än man har
CALL withdraw(192873,5000);  # Kontot finns ej

SELECT * FROM Transfers; # Endast den första har lagts till


# 6. show_transfers(account_id)
DROP PROCEDURE IF EXISTS show_transfers;
DELIMITER //
CREATE PROCEDURE show_transfers(IN account_id INT)
BEGIN
      SELECT * FROM Transfers
      WHERE Transfers.account_id = account_id
      ORDER BY t_datetime DESC;
END//
DELIMITER ;

CALL show_transfers(3);


# 7. no_of_owners(account_id)
DROP FUNCTION IF EXISTS no_of_owners;
DELIMITER //
CREATE FUNCTION no_of_owners(account_id int)
RETURNS int
READS SQL DATA
BEGIN
      RETURN (
          SELECT COUNT(*)
            FROM Owners
            WHERE Owners.account_id = account_id
            GROUP BY account_id);
END//
DELIMITER ;

SELECT id, no_of_owners(id) AS no_of_owners, amount
FROM Accounts
ORDER BY no_of_owners DESC;


# 8. Egen procedure eller function

# Skicka pengar från ett konto till ett användarnamn (typ som Swish)
DROP PROCEDURE IF EXISTS pay_user;
DELIMITER //
CREATE PROCEDURE pay_user(amount decimal(13, 2), sender_account_id int,
receiver_user_name varchar(50))
BEGIN
```

```
    START TRANSACTION;

    SET @receiver_id := (SELECT id FROM Users WHERE username = receiver_user_name);
    # If the user has more than one account we just select the first one
    SET @receiver_account := (SELECT account_id FROM Owners WHERE user_id =
@receiver_id LIMIT 1);

    SET @sender_current_amount := (SELECT Accounts.amount FROM Accounts WHERE id =
sender_account_id);
    SET @sender_new_amount := @sender_current_amount - amount;

    IF @receiver_id IS NULL THEN
        SELECT 'Error: user does not exist' AS status;
    ELSEIF @receiver_account IS NULL THEN
        SELECT 'Error: user does not have any accounts' AS status;
    ELSEIF @receiver_account = sender_account_id THEN
        SELECT 'Error: cannot send money to yourself' AS status;
    ELSEIF amount <= 0 THEN
        SELECT 'Error: amount is not > 0' AS status;
    ELSEIF @sender_new_amount < 0 THEN
        SELECT 'Error: you do not have this much money to send' AS status;
    ELSE
        # Använder tidigare procedures :)
            CALL withdraw(sender_account_id, amount);
            CALL add_transfer(@receiver_account, amount);
      END IF;

    COMMIT;
END//
DELIMITER ;

# User does not exist
CALL pay_user(300.31, 1, 'ajksdhkfkjashdfk');

# Cannot send money to yourself
CALL pay_user(300.31, 5, 'lots456');

# Amount is not > 0
CALL pay_user(0, 1, 'fjk34');

# Amount is not > 0
CALL pay_user(-100, 1, 'fjk34');

# You do not have this much money to send
CALL pay_user(99999999999, 1, 'fjk34');

# It works!
CALL pay_user(300.31, 1, 'fjk34');


# 9. Egen vy
DROP VIEW IF EXISTS richest_users;
CREATE VIEW richest_users AS
SELECT username AS user_name, accounts.amount AS account_amount
FROM Users
    JOIN Owners ON Owners.user_id = Users.id
    JOIN Accounts ON Accounts.id = Owners.account_id;

SELECT * FROM richest_users ORDER BY account_amount DESC;
```

```
# 10. TRANSACTION
# (1) Startar en transaktion
START TRANSACTION;

# Visar oförändrat användarnamn för användare 1 (aha12)
SELECT username FROM Users WHERE id = 1;

# (2) Gör en update som sätter username till Hej1234 för användare 1
UPDATE Users SET username = 'Hej1234' WHERE id = 1;

# Visar nya användarnamnet för användare 1 (Hej1234)
SELECT username FROM Users WHERE id = 1;

# (3) Gör en save point vid namn first_update
SAVEPOINT first_update;

# (4) Gör en update som sätter username till Hej5678 för användare 1
UPDATE Users SET username = 'Hej5678' WHERE id = 1;

# Visar nya användarnamnet för användare 1 (Hej5678)
SELECT username FROM Users WHERE id = 1;

# (5) Gör en rollback till innan vi ändrade användarnamnet till Hej5678
ROLLBACK TO first_update;

# (6) Gör en commit vilket sparar nuvarande ändringarna
COMMIT;

# Visar användarnamnet vi hade vid save point "first_update" för användare 1
(Hej1234)
SELECT username FROM Users WHERE id = 1;


# 11. USERS
# a)
CREATE USER IF NOT EXISTS 'kim'@'localhost'  IDENTIFIED BY '1';
CREATE USER IF NOT EXISTS 'alex'@'localhost' IDENTIFIED BY '2';
CREATE USER IF NOT EXISTS 'app'@'localhost'  IDENTIFIED BY '3';

# b)
GRANT SELECT, UPDATE
ON lab5.*
TO
      'kim'@'localhost',
      'alex'@'localhost',
      'app'@'localhost';

# c)
REVOKE UPDATE
ON lab5.*
FROM
      'alex'@'localhost',
    'app'@'localhost';

# d)
ALTER USER 'alex'@'localhost' WITH MAX_QUERIES_PER_HOUR 200;
```

```
# 12. ROLES
# a)
CREATE ROLE IF NOT EXISTS db_read;
CREATE ROLE IF NOT EXISTS db_write;
CREATE ROLE IF NOT EXISTS db_sproc;

GRANT SELECT ON lab5.* TO db_read;
GRANT INSERT, UPDATE ON lab5.* TO db_write;
GRANT EXECUTE ON PROCEDURE lab5.add_transfer TO db_sproc;
GRANT EXECUTE ON PROCEDURE lab5.show_transfers TO db_sproc;
GRANT EXECUTE ON PROCEDURE lab5.withdraw TO db_sproc;

# b)
GRANT 'db_read' TO 'alex'@'localhost';
GRANT 'db_write' TO 'kim'@'localhost';
GRANT 'db_sproc' TO 'app'@'localhost';

# c)
SHOW GRANTS FOR 'kim'@'localhost';

# d)
SHOW GRANTS FOR 'db_write';
```