```
-- Authors = Benjamin Appelberg & Arvid Bryntesson
drop database if exists DB2_VT22_L5;
create database DB2_VT22_L5;
use DB2_VT22_L5;

/* 1. Skapa tabeller
Skriv queries för att skapa en databas och tabeller för att hantera personer som
har konton med pengar.
Konton ska kunna ha en eller flera innehavare, pengar på konto får aldrig hamna på
mindre än noll.
Använd lämpliga datatyper, nycklar, index, constraints och foreign keys som passar.

Tabellerna och kolumner ska vara:
a) Users (id, username)
b) Accounts (id, amount)
c) Transfers (id, account_id, amount, note, t_datetime)
d) Owners (user_id, account_id) */
create table users (
      id int not null auto_increment,
    username varchar(32) not null,
    primary key (id)
);
create table accounts (
      id int not null auto_increment,
    amount int unsigned not null default 0,
    primary key (id)
);
create table transfers (
      id int not null auto_increment,
    account_id int not null,
    amount int not null,
    note text,
    t_datetime datetime not null default now(),
    index (t_datetime),
    primary key (id),
    foreign key (account_id) references accounts(id)
            on delete cascade
);
create table owners (
      user_id int not null,
    account_id int not null,
    foreign key (user_id) references users(id)
            on delete cascade,
    foreign key (account_id) references accounts(id)
            on delete cascade
);

/* 2. Skapa innehåll
Skriv och kör queries för att fylla på med minst 5 användare med minst två konton
vardera.
Välj godtyckliga rimliga värden för belopp som finns för konton.
Gör så att minst 2 konton har mer än en ägare. */
-- 5 användare
insert into users (username)
      values ("Stefan"), ("Jörgen"), ("Torbjörn"), ("Helen"), ("jessica");

-- 11 konton
insert into accounts (amount)
      values (20500),(750),(10000),(7597),(16847),
```

```
    (rand()*100000),(rand()*100000),(rand()*100000),
    (rand()*100000),(rand()*100000),(rand()*100000);

insert into owners
      values      (1,1),(1,2),
                  (2,1),(2,2),
            (3,3),(3,6),
            (4,4),(4,7),
            (5,5),(5,8),(5,9);

/* 3. users_accounts
Skapa en vy som innehållller kolumer med data från users och accounts som visar
vilka konton som hör
till en user och hur mycket pengar som finns på kontot. Vi vill visa user_id,
user_name, account_id och account_amount.
Ska kunna användas med t ex
SELECT * FROM users_accounts WHERE user_id = 3 ORDER BY account_id ASC;
eller SELECT user_id FROM users_accounts WHERE account_id = 7; */

create or replace view users_accounts as
select user_id, username, account_id, amount
from accounts
join owners on account_id = accounts.id
join users on users.id = user_id;

-- query ex:
SELECT * FROM users_accounts WHERE user_id = 3 ORDER BY account_id ASC;
SELECT user_id FROM users_accounts WHERE account_id = 7;

/* deposit(account_id, amount)
Skriv en procedure som lägger till pengar till ett konto.
Procedure ska ändra pengar på konto samt också lägga till en rad i transfers med
notering om att det är en insättning,
på hur mycket och tid och datum för insättningen. Det ska endast gå att göra
insättningar med positiva tal och till konton som redan finns.
Endast insättningar som är godkända ska loggas i transfers. Använd TRANSACTION och
COMMIT så att det inte kan bli fel vid drift.
Avsluta med ett SELECT som visar på aktuell rad i transfers om allt gick igenom
annars gör SELECT med ett status som är ett meddelande om
"Error: acoount does not exist" eller "Error: amount is not > 0" som sätts i
procedure. */
drop procedure if exists deposit;
delimiter //
create procedure deposit(in param_account_id int, in param_amount int)
begin
start transaction;

if param_account_id not in (select id from accounts) then
select 'Account does not exist' as 'Errormessage: 1';
elseif param_amount < 1 then
select 'Amount is less than one' as 'Errormessage: 2';
else
      update accounts
      set amount = param_amount + amount
      where id = param_account_id;

      insert into transfers(account_id, amount, note, t_datetime) values
      (param_account_id, param_amount, 'deposit', now());
      select * from transfers
```

```
      where id = last_insert_id();
end if;
commit;
end //
delimiter ;

-- procedure ex:
call deposit(1,1000);
call deposit(50,1000);
call deposit(1,-100);

/* 5. withdraw(account_id, amount)
Skriv en procedure som tar ut pengar från ett konto. Procedure ska ändra mängd
pengar på konto samt också lägga till en rad i transfers
med notering om att det är ett uttag, på hur mycket och tid och datum för uttaget.
Uttaget ska anges som ett positivt tal när vi anropar SP men pengar på kontot ska
minskas och i transfers-loggen ska talet skrivas som negativt.
Det ska inte gå att ta ut mer pengar än vad som finns på kontot.
Använd TRANSACTION och COMMIT så att det inte kan bli fel vid drift till exempel
att det görs flera uttag samtidigt).
Avsluta med ett SELECT  som visar på aktuell rad i transfers om allt gick igenom
annars gör
SELECT med ett status som är ett meddelande om "Error: acoount does not exist"
eller
"Error: amount is too large" eller "ERROR: amount is not > 0" som sätts i
procedure. */

drop procedure if exists withdraw;
delimiter //
create procedure withdraw(in param_account_id int, in param_amount int)
begin
      start transaction;
            if param_account_id not in (select id from accounts) then
select 'Account does not exist' as 'Errormessage: 1';
elseif param_amount < 1 then
select 'Amount is less than one' as 'Errormessage: 2';
elseif param_amount > (select amount from accounts where id = param_account_id)
then
select 'Amount is too large' as 'Errormessage: 3';
else
      update accounts
      set amount = amount - param_amount
      where id = param_account_id;

      insert into transfers(account_id, amount, note, t_datetime) values
      (param_account_id, (param_amount * -1), 'withdrawal', now());
      select * from transfers
      where id = last_insert_id();
end if;
      commit;
end //
delimiter ;

-- SP ex:
select * from accounts
      where id = 1;
call withdraw(1,1000);
select * from accounts
      where id = 1;
```

```
call withdraw(50,1000);
call withdraw(1,-100);
call withdraw(1,1000000);

/* 6. show_transfers(account_id)
Skriv queries för att skapa en procedure, show_transfers(account_id),
som listar alla transfers för ett angivet konto sorterade på datum och tid. Senaste
transfer överst. */

drop procedure if exists show_transfers;
delimiter //
create procedure show_transfers(in param_account_id int)
begin
select * from transfers
where account_id = param_account_id
order by t_datetime
desc;

end //

delimiter ;

call show_transfers(1);

/* 7. no_of_owners(account_id)
Skapa en funktion som returnerar antal ägare för ett kontonummer (account_id).
Ska kunna användas i t ex
SELECT id, no_of_owners(id) AS no_of_owners, amount FROM Accounts
ORDER BY no_of_owners DESC; */

drop function if exists no_of_owners;
delimiter //
create function no_of_owners(param_account_id int)
returns int
deterministic
begin
      declare total_owners int;
      set total_owners = (select count(user_id) from owners where account_id =
param_account_id);
      return total_owners;
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
Skapa en egen procedure eller 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 och fungerar.
*/

/* own stored procedure: transfer(in param_from_account int, in param_to_account
int, in param_amount int)
first argument is the account to transfer amount from,
second argument is the account to transfer the amount to and
last argument is the desired amount to transfer.
```

```
transfer will be recorded in transfers table and the note column value will be
transfer*/
drop procedure if exists transfer;
delimiter //
create procedure transfer(in param_from_account int, in param_to_account int, in
param_amount int)
begin
      start transaction;
            if param_from_account not in (select id from accounts) or
param_to_account not in (select id from accounts) then
                  select 'Account does not exist' as 'Errormessage: 1';

            elseif param_amount < 1 then
                  select 'Amount is less than one' as 'Errormessage: 2';

            elseif param_amount > (select amount from accounts where id =
param_from_account) then
                  select 'Amount is too large' as 'Errormessage: 3';

            else
                  call withdraw(param_from_account, param_amount);
            update transfers set note = 'transfer'
            where id = last_insert_id();
                  call deposit(param_to_account,param_amount);
                  update transfers set note = 'transfer'
            where id = last_insert_id();
            end if;

    commit;
end //
delimiter ;

call transfer(5, 6, 250);

select * from transfers;

/* 9. Egen vy
Skapa en egen vy 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 och fungerar.
(Vanlig vy räcker. Behöver ej vara en materialiserad vy med triggers etc.
Men för den intresserade som vill ha en liten utmaning går det även att bygga en
materialiserad vy och redovisa den.) */

-- show_users_transfer shows records of users transfers with users name.
create or replace view show_users_transfers as
select  t.id, t.account_id, group_concat(username) as users, note, amount,
t_datetime
from transfers t
join owners o
      on o.account_id = t.account_id
join users u
      on o.user_id = u.id
group by t.id;

-- result ex:
select * from show_users_transfers;

/* 10. TRANSACTION
```

```
Skriv queries för att:
(1) starta en transaktion,
(2) göra en UPDATE,
(3) sätta en SAVEPOINT,
(4) göra en UPDATE,
(5) göra en ROLLBACK till savepoint,
(6) en göra COMMIT.
Som exempel kan någon användare och/eller konton läggas till eller olika
insättningar och uttag.
Lägg in SELECT och kommentarer som tydligt visar och förklarar vad som händer och
inte händer och varför.*/

-- transaction show first username to update
-- updates the lowercase j to uppercase then creates a savepoint
-- after savepoint updates the 'c' to a 'k'
-- then rolls back to the declared savepoint before the update c -> k
start transaction;

      select username from users where id = 5; -- lowercase j 'jessica'

      update users set username = 'Jessica'
      where id = 5;
      savepoint change_name_savepoint;

      select username from users where id = 5;

      update users set username = 'Jessika'
      where id = 5;
      select username from users where id = 5;

      rollback to change_name_savepoint;
      select username from users where id = 5;

commit;

/* 11. USERS
Skriv queries för att skapa login för tre USERS till MySQL (inte users för
tabellerna med accounts):
a) De tre ska heta: kim, alex, app
b) Ge alla access (GRANT) så de kan göra SELECT och UPDATE på alla tabeller i DB
för denna laboration
c) Ta bort (REVOKE) så att alex och app inte får göra UPDATE
d) Begränsa så att alex inte får göra mer än 200 queries per timme */

-- a)

drop user if exists 'kim';
drop user if exists 'alex';
drop user if exists 'app';

create user 'kim' identified by 'password1';
create user 'alex' identified by 'password2';
create user 'app' identified by 'password3';

-- b)
grant update, select on db2_vt22_l5.* to 'kim';
grant update, select on db2_vt22_l5.* to 'alex';
grant update, select on db2_vt22_l5.* to 'app';
```

```
-- c)
revoke update on db2_vt22_l5.* from 'alex';

-- d)
alter user 'alex' with max_queries_per_hour 200;

/* 12. ROLES
Skriv queries för att skapa roller för att administrera databasen:
a) De ska heta: db_read (kan bara göra SELECT (alla tabeller i DB), db_write (kan
göra INSERT och UPDATE (alla tabeller i DB), db_sproc (får köra de sproc som vi
implementerat ovan)
b) Tilldela db_read till Alex, db_write till Kim och db_sproc till app.
c) Skriv query för att lista rättigheter för Kim.
d) Skriv query för att lista rättigheter för db_write */

-- a)

drop role if exists db_read, db_write, db_sproc;
create role db_read;
grant select on db2_vt22_l5.* to db_read;

create role db_write;
grant insert, update on db2_vt22_l5.* to db_write;

create role db_sproc;
grant execute on db2_vt22_l5.* to db_sproc;

-- b)
grant db_read to 'alex';
grant db_write to 'kim';
grant db_sproc to 'app';

-- c)
show grants for kim;

-- d)
show grants for db_write;
```