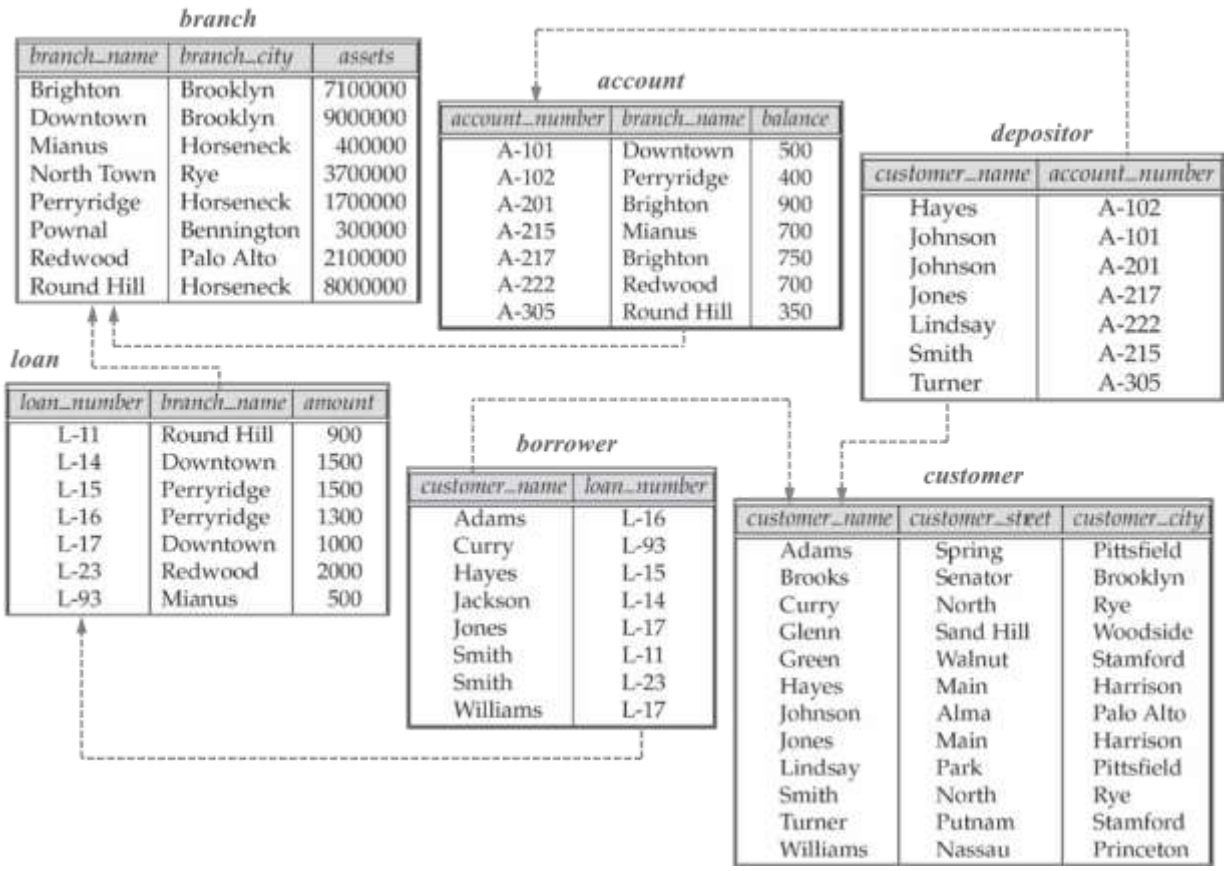
Lab Session 7: Functions, Stored Procedures and Triggers

We will be using the bank database of previous labs.



branch

branch_name	branch_city	assets
Brighton	Brooklyn	7100000
Downtown	Brooklyn	9000000
Mianus	Horseneck	400000
North Town	Rye	3700000
Perryridge	Horseneck	1700000
Pownal	Bennington	300000
Redwood	Palo Alto	2100000
Round Hill	Horseneck	8000000

account

account_number	branch_name	balance
A-101	Downtown	500
A-102	Perryridge	400
A-201	Brighton	900
A-215	Mianus	700
A-217	Brighton	750
A-222	Redwood	700
A-305	Round Hill	350

depositor

customer_name	account_number
Hayes	A-102
Johnson	A-101
Johnson	A-201
Jones	A-217
Lindsay	A-222
Smith	A-215
Turner	A-305

loan

loan_number	branch_name	amount
L-11	Round Hill	900
L-14	Downtown	1500
L-15	Perryridge	1500
L-16	Perryridge	1300
L-17	Downtown	1000
L-23	Redwood	2000
L-93	Mianus	500

borrower

customer_name	loan_number
Adams	L-16
Curry	L-93
Hayes	L-15
Jackson	L-14
Jones	L-17
Smith	L-11
Smith	L-23
Williams	L-17

customer

customer_name	customer_street	customer_city
Adams	Spring	Pittsfield
Brooks	Senator	Brooklyn
Curry	North	Rye
Glenn	Sand Hill	Woodside
Green	Walnut	Stamford
Hayes	Main	Harrison
Johnson	Alma	Palo Alto
Jones	Main	Harrison
Lindsay	Park	Pittsfield
Smith	North	Rye
Turner	Putnam	Stamford
Williams	Nassau	Princeton

Part I: Functions

We have seen how to create a function that counts the accounts of a given customer, where the customer name is passed as input parameter to the function:

```
delimiter $$

create function count_accounts(c_name varchar(255))
returns integer
begin
        declare a_count integer;
        select count(account_number) into a_count
        from depositor
        where customer_name = c_name;
        return a_count;
end$$

delimiter ;
```

1. Write an SQL function that returns the "absolute balance" of a customer. The "absolute balance" is the difference between all the money that the customer has (in accounts), and all the money that the customer owes to the bank (in loans).

   Hint: the following queries return the balance for customer 'Smith'. You can combine these queries in your function:

```
select sum(balance)
from account natural join depositor
where customer_name = 'Smith';

select sum(amount)
from loan natural join borrower
where customer_name = 'Smith';
```

2. Check that the function that you have created yields the same result as the query above when 'Smith' is passed as input parameter.

3. Using your function, find the customer who has the highest absolute balance in the bank.

Note: when determining the absolute balance of each customer, consider only those customers who appear both in table depositor and in table borrower.

Part II: Stored Procedures

We have seen how to create a stored procedure to execute an arbitrary SQL statement. Here is a simple example:

```
delimiter $$

create procedure reward_accounts(in interest_rate float)
begin
        update account
        set balance = balance + interest_rate * balance;
end$$

delimiter ;

call reward_accounts(0.01);
```

4. Write a new stored procedure that returns, for a given branch, the list of customers who have accounts in that branch.

   Hint: the following query returns the list of customers who have accounts in 'Brighton'. You can use a similar query in your procedure:

```
select customer_name
from depositor as d, account as a
where d.account_number = a.account_number
   and a.branch_name = 'Brighton';
```

5. Call your stored procedure and check that it produces the same result as the query above when 'Brighton' is passed as input parameter.

Part III: Triggers

We have seen how to create a trigger to automatically create a new loan when the balance of an account becomes negative:

```
delimiter $$

create trigger check_balance before update on account
for each row
begin
    if new.balance < 0 then
        insert into loan values (new.account_number,
                                 new.branch_name,
                                 (-1)*new.balance);
        insert into borrower (
            select customer_name, account_number
            from depositor as d
            where d.account_number = new.account_number);
        set new.balance = 0;
    end if;
end$$

delimiter ;
```

The trigger above will be called every time there is an update to table account. If the balance of an account is changed to a negative value, the trigger creates a new loan and also associates the depositors of the account as borrowers for the new loan. Finally, it sets the account balance to zero.

6. Write a new trigger to be called every time there is an update on table loan. If the amount of a loan is changed to a negative value, the trigger should create a new account and also associate the borrowers of the loan as depositors for the new account. Finally, it should set the loan amount to zero.

7. Test the trigger by subtracting 1200 from the amount of 'L-17'. Confirm that a new account has been created, that the depositors for that account have been inserted correctly, and that the amount of 'L-11' has been set to zero.

account_number	branch_name	balance
A-101	Downtown	500.00
A-102	Perryridge	400.00
A-201	Brighton	900.00
A-215	Mianus	700.00
A-217	Brighton	750.00
A-222	Redwood	700.00
A-305	Round Hill	350.00
L-17	Downtown	200.00

customer_name	account_number
Johnson	A-101
Hayes	A-102
Johnson	A-201
Smith	A-215
Jones	A-217
Lindsay	A-222
Turner	A-305
Jones	L-17
Williams	L-17

loan_number	branch_name	amount
L-11	Round Hill	900.00
L-14	Downtown	1500.00
L-15	Perryridge	1500.00
L-16	Perryridge	1300.00
L-17	Downtown	0.00
L-23	Redwood	2000.00
L-93	Mianus	500.00