

# Exercise 4: SQL Programming

Björn Þór Jónsson

---

## Setup and Preparation

Use the E04-bills-schema.sql script to create a new database called bills (this is the same script as used in lecture in week 3). The script creates four tables: People (P), Bills (B), Accounts (A), and AccountRecords (R). The script then inserts 200 individuals into the People table, as well as many records into Accounts, AccountRecords and Bills.

The intention is that A.aBalance is the current balance of the account and that A.aDate is the date of the last change to the account. Then, for each record, R.rAmount should be the amount of the transaction (positive for a deposit, negative for a withdrawal), R.rDate should be the date of that transaction, and R.rBalance should be the state of the account immediately following the transaction. Thus, the following two queries should return the same data for any account X:

```
SELECT A.aDate, A.aBalance
FROM Accounts A
WHERE A.AID = X;

SELECT R.rDate, R.rBalance
FROM AccountRecords R
WHERE R.AID = X
AND R.RID = (SELECT MAX(R1.RID)
             FROM AccountRecords R1
             WHERE R1.AID = R.AID)
```

Accounts can be overdrawn to the amount A.aOver (i.e.,  $A.aBalance \geq -A.aOver$ ).

AccountRecords contains all transactions made on each account. Each record includes an R.aType attribute, which denotes the type of the record:

- B for Bills
- T for Transfers (Deposit or Withdrawal).
- L for loans.
- O for other.

All records due to bills will therefore have R.aType value of B. When a loan is granted and deposited into the account, the account record will be of R.aType L.

The Bills table contains both paid and unpaid bills for a person. The B.bIsPaid attribute indicates if the person has paid the bill or not. A bill always has a positive amount, but when paid, the same negative amount should be deducted from an account. A bill also has a B.bDueDate, which is the date it should be paid.

## The Exercise

Write SQL code according to the following description. You are supplied with a test script that tests many aspects of the exercise, but you may also implement your own tests for more thorough coverage.

1. Create a view AllAccountRecords that joins the Accounts and AccountRecords and shows one entry for each record for each account. The view should show all columns from both tables, first Accounts and then AccountRecords, except the AccountRecords.AID column. Accounts with no entry in AccountRecords should be shown with NULLs in the columns for AccountRecords.
2. Create a trigger CheckBills for insertions to Bills that enforces a) the rule that bills cannot have a negative amount, and b) that the due date must be in the future (tomorrow or later).
3. Create a trigger on the AccountRecords table that does the following for each insertion: a) checks whether the amount is available (taking the overdraft into account) on the account being withdrawn from, b) updates the balance on the account and c) installs the correct balance into the AccountRecords entry.
4. Create a procedure Transfer, which takes three parameters, iToAID, iFromAID, and iAmount, and transfers the given amount between the two given accounts in a single transaction. If the amount is not available in the iFromAID account, then the operation should be aborted (this should happen via the trigger from #3). No return value.
5. Create a view DebtorStatus that shows, for each person whose total balance is less than 0, the PID and pName of the person, the total balance of all their accounts, and the total overdraft of all their accounts.
6. Create a trigger NewPerson on People that creates a new account when a new person is inserted to the table. The bank has a rule that each new customer gets an initial overdraft of 10000.
7. Create a procedure InsertPerson that takes four IN parameters iName, iGender, iHeight, and iAmount. The function should insert a new person to the People table, and then deposit the amount iAmount to the account created by the trigger of #6. If you were unable to write the trigger of #6, you can create the account in the function as well. No return value.
8. Create a procedure PayOneBill that takes one parameter, iBID, and pays the bill in question. The payment should come from the account with the highest balance (including the overdraft) of the person's accounts (if two accounts have the same balance + overdraft, either one can be chosen). The operations should of course rely on the trigger from #5 to maintain consistency. If there are insufficient funds in a single account (including the overdraft) to pay the bill, then the operation (transaction) should be aborted. Of course, the bIsPaid field must be set to "true" for the bill. No return value.

9. Create a procedure LoanMoney, which takes three parameters iAID, iAmount and iDueDate. The procedure should give the account a loan of iAmount and also create a bill with the same lAmount and due date on iDueDate. No return value.
10. Create a view FinancialStatus that shows, for each person that has one or more accounts, the PID and pName of the person, the total amount they have on all their accounts, and the total amount of all unpaid bills.

*Note: This is a difficult view, and in fact a good SQL exercise! I suggest to create two helper views. The first helper view should sum up all accounts of each person, having one entry for each person that has any accounts. The second helper view should sum up all unpaid bills of each person, with one entry for each person that has any unpaid bills. The final view should join the People table with the two helper views, using an outer join for the second helper view.*