Advanced DataBases C4 Group

Bank Management System

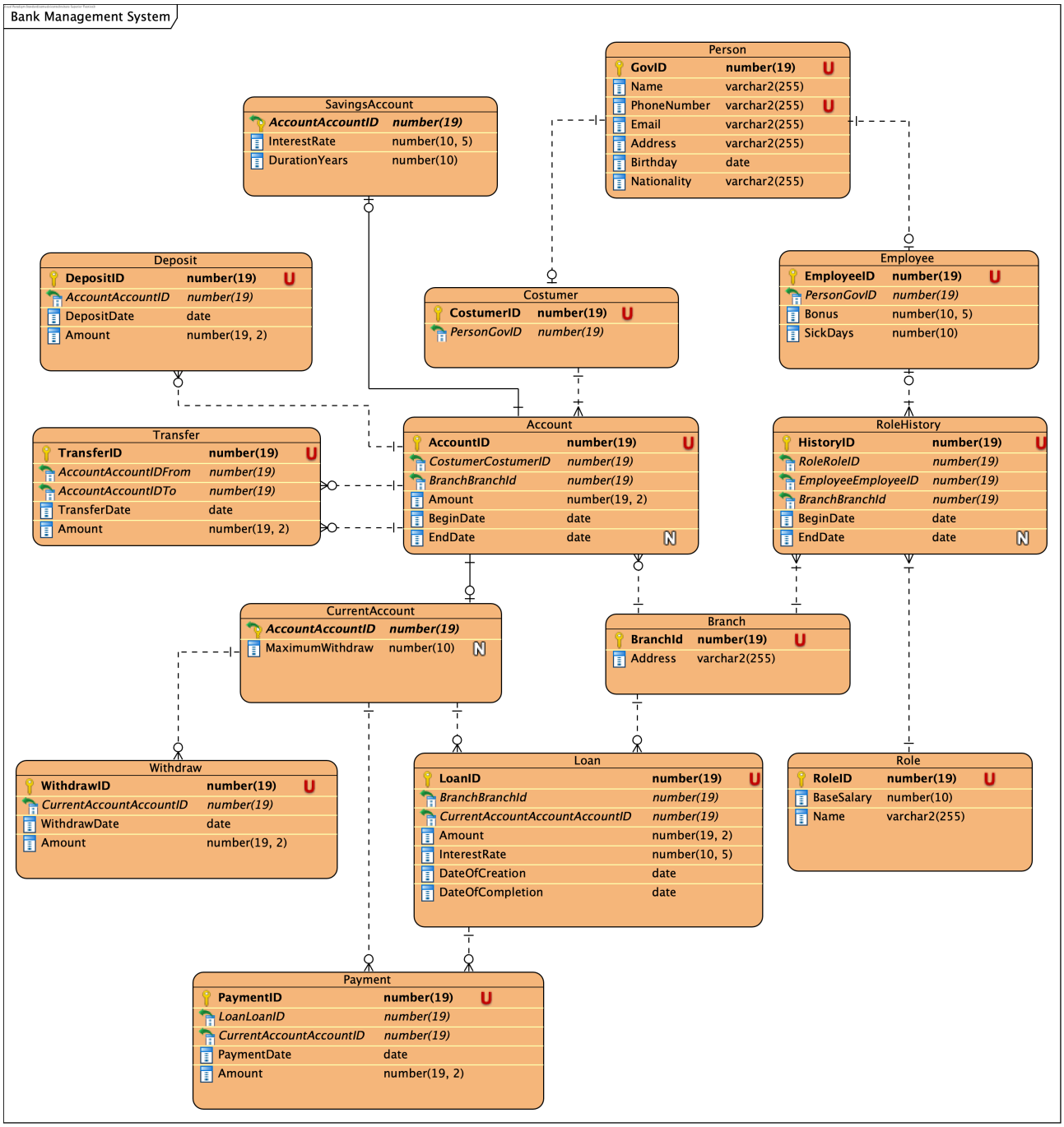
Samuel Lúcio Vicente, 251720 Daniel Silva, 251702
Jorge Marrero Camiruaga, 251438

Politechnika Wrocławska
today

Short Description

This database models a bank with Accounts that can either be a SavingsAccount or a CurrentAccount. There are some operations that are permitted, Withdraw from the CurrentAccount, Transfer between accounts, Loan, Pay the loan, Deposit on Account. There are employees that work in a branch that have roles, this database tracks the history of all the employees on every branch, and the previous roles that some particular employee has had and which branch he has working at.

ERD



Generator Script

```
import random
import math

def newDate(year, month, numeroMeses):
    newMonth=(month+numeroMeses)%12 if (month+numeroMeses)%12 != 0 else 12
    newYear=year + (math.floor(numeroMeses/12 +1) if month>newMonth else math.floor(numeroMeses/12))
    return (newYear, newMonth)
phoneNumber=900000000
govId=1111111111
```

```
10
11 loanID=1
12 costumerID=1
13 employeeID=1
14 branchID=1
15 accountID=1
16 with open("./resources/citiesCountries.txt", 'r', encoding='utf-8') as cities:
17     places=cities.read().splitlines()
18     with open("./resources/costumerNames.txt", 'r', encoding='utf-8') as costumerNames:
19         with open("./resources/employeeName.txt", 'r', encoding='utf-8') as employeeNames:
20             with open("./resources/Roles.txt", 'r', encoding='utf-8') as roles:
21                 #Create Roles
22                 for role in roles:
23                     baseSalary=role.split(",")[1].strip()
24                     name=role.split(",")[0].strip()
25                     print(f"INSERT␣INTO␣Role(BaseSalary,␣Name)␣VALUES␣({baseSalary},␣'{name}');")
26                 #Finish Roles
27
28                 #create branch
29                 for place in places:
30                     print(f"INSERT␣INTO␣Branch(Address)␣VALUES␣('{place}');")
31                     i=1
32                     #create 8 employees per branch
33                     beginDate=f"TO_DATE('{random.randint(1,28)}-{random.randint(1,12)}-{random.randint
                           ↪ (2001,2018)}','DD-MM-YYYY')"
34                     for name in employeeNames:
35                         name=name.strip()
36                         email=f"{name.strip().replace('␣','␣'_')}{random.randint(0,100)}@sapo.pt"
37                         address=place
38                         birthday=f"TO_DATE('{random.randint(1,28)}-{random.randint(1,12)}-{random.
                               ↪ randint(1950,2000)}','DD-MM-YYYY')"
39                         nationality=address.split(",")[1].strip()
40                         print(f"INSERT␣INTO␣Person(GovID,Name,␣PhoneNumber,␣Email,␣Address,␣Birthday,␣
                               ↪ Nationality)␣VALUES␣({govId},'{name}','{phoneNumber}',␣'{email}',␣'{
                               ↪ address}',␣{birthday},␣'{nationality}');")
41                         bonus=random.uniform(0,1)
42                         sickDays=random.randint(0,9)
43                         print(f"INSERT␣INTO␣Employee(PersonGovID,␣Bonus,␣SickDays)␣VALUES␣({govId},{
                               ↪ format(bonus,␣'.4f')},{sickDays});")
44                         #create RoleHistory
45                         print(f"INSERT␣INTO␣RoleHistory(RoleRoleID,␣EmployeeEmployeeID,␣BranchBranchId,␣
                               ↪ BeginDate,␣EndDate)␣VALUES␣({i},{employeeID},{branchID},{beginDate},NULL)
                               ↪ ;")
46                         #Finish RoleHistory
47                         govId=govId+1
48                         phoneNumber=phoneNumber+1
49                         employeeID=employeeID+1
50                         i=1+i
51                         if(i==9):
52                             break
53                     branchID=branchID+1
54                     #Finish Employee
55                     #Finish Branch
56
57                 #Create Costumer
58                 for name in costumerNames:
59                     govId=govId+1
60                     name=name.strip()
61                     phoneNumber=phoneNumber+1
62                     email=f"{name.strip().replace('␣','␣'_')}{random.randint(0,100)}@sapo.pt"
63                     address=random.choice(places)
64                     day=random.randint(1,28)
```

```
65                  month=random.randint(1,12)
66                  year=random.randint(1950,2008)
67                  birthday=f"TO_DATE('{day}-{month}-{year}','DD-MM-YYYY')"
68                  nationality=address.split(",")[1].strip()
69                  print(f"INSERT␣INTO␣Person(GovID,Name,␣PhoneNumber,␣Email,␣Address,␣Birthday,␣
                        ↪ Nationality)␣VALUES␣({govId},'{name}','{phoneNumber}',␣'{email}',␣'{address
                        ↪ }',␣{birthday},␣'{nationality}');")
70
71                  print(f"INSERT␣INTO␣Costumer(PersonGovID)␣VALUES␣({govId});")
72                  #Create Accounts
73                  #Create the first CurrentAccount
74                  branchID=random.randint(1,1252)
75                  amount=random.randint(50000, 1000000)
76                  beginDateDay=random.randint(1,28)
77                  beginDateMonth=random.randint(1,12)
78                  beginDateYear=random.randint(2000,2018)
79                  beginDate=f"TO_DATE('{beginDateDay}-{beginDateMonth}-{beginDateYear}','DD-MM-YYYY')"
80                  endDate=f"NULL"
81                  print(f"INSERT␣INTO␣Account(CostumerCostumerID,␣BranchBranchID,␣Amount,␣BeginDate,␣
                        ↪ EndDate)␣VALUES␣({costumerID},␣{branchID},␣{amount/4},␣{beginDate},␣{endDate
                        ↪ });")
82                  maximumWithdraw=400
83                  print(f"INSERT␣INTO␣CurrentAccount(AccountAccountID,␣MaximumWithdraw)␣VALUES␣({
                        ↪ accountID},␣{maximumWithdraw});")
84                  print(f"INSERT␣INTO␣Deposit(AccountAccountID,␣DepositDate,␣Amount)␣VALUES␣({
                        ↪ accountID},␣{beginDate},{amount});")
85                  print(f"INSERT␣INTO␣Withdraw(CurrentAccountAccountID,␣WithdrawDate,␣Amount)␣VALUES␣
                        ↪ ({accountID},␣{beginDate},␣{amount/4});")
86                  accountID=accountID+1
87                  #Create the second SavingsAccount
88                  print(f"INSERT␣INTO␣Account(CostumerCostumerID,␣BranchBranchID,␣Amount,␣BeginDate,␣
                        ↪ EndDate)␣VALUES␣({costumerID},␣{branchID},␣{amount/2},␣{beginDate},␣{endDate
                        ↪ });")
89                  interestRate=0.05
90                  duration=random.randint(3,10)
91                  print(f"INSERT␣INTO␣SavingsAccount(AccountAccountID,␣InterestRate,␣DurationYears)␣
                        ↪ VALUES␣({accountID},␣{interestRate},␣{duration});")
92                  print(f"INSERT␣INTO␣Transfer(AccountAccountIDFrom,␣AccountAccountIDTo,␣TransferDate,
                        ↪ ␣Amount)␣VALUES␣({accountID-1},␣{accountID},␣{beginDate},␣{amount/2});")
93                  accountID=accountID+1
94
95                  if(random.randint(0,100)>97):
96                      #Create Loan
97                      loanAmount=random.randint(5000, 1000000)
98                      loanInterestRate=0.07
99                      loanCompletionYear=beginDateYear+random.randint(1,2)
100                     loanDateOfCompletion=f"TO_DATE('{beginDateDay}-{beginDateMonth}-{
                            ↪ loanCompletionYear}','DD-MM-YYYY')"
101                     loanDateOfCreation=beginDate
102
103                     print(f"INSERT␣INTO␣Account(CostumerCostumerID,␣BranchBranchID,␣Amount,␣
                            ↪ BeginDate,␣EndDate)␣VALUES␣({costumerID},␣{branchID},␣{loanAmount},␣{
                            ↪ beginDate},␣{endDate});")
104
105                     print(f"INSERT␣INTO␣CurrentAccount(AccountAccountID,␣MaximumWithdraw)␣VALUES␣({
                            ↪ accountID},␣{maximumWithdraw});")
106
107                     currentYear=2019
108                     currentMonth=11
109                     currentDay=6
110
```

```
                    loanMonthsToCurrentDate=(currentYear-beginDateYear)*12+(currentMonth-
                        ↪ beginDateMonth)+(-1 if currentDay < beginDateDay else (0))

                    loanMonthsToCompletion=(loanCompletionYear-beginDateYear)*12

                    paymentAmountPerMonth=math.ceil((loanAmount/loanMonthsToCompletion)*1+
                        ↪ loanInterestRate)

                    print(f"INSERT␣INTO␣Loan(BranchBranchID,␣CurrentAccountAccountAccountID,␣Amount,
                        ↪ ␣InterestRate,␣DateOfCreation,␣DateOfCompletion)␣VALUES␣({branchID},␣{
                        ↪ accountID},␣{loanAmount},␣{loanInterestRate},␣{loanDateOfCreation},␣{
                        ↪ loanDateOfCompletion});")

                    for j in range(1,min(loanMonthsToCompletion, loanMonthsToCurrentDate)+1):
                        date=newDate(beginDateYear, beginDateMonth, j)
                        paymentDate=f"TO_DATE('{beginDateDay}-{date[1]}-{date[0]}','DD-MM-YYYY')"

                        print(f"INSERT␣INTO␣Payment(LoanLoanID,␣CurrentAccountAccountID,␣PaymentDate,
                            ↪ ␣Amount)␣VALUES␣({loanID},␣{accountID},␣{paymentDate},␣{
                            ↪ paymentAmountPerMonth});")
                        print(f"INSERT␣INTO␣Deposit(AccountAccountID,␣DepositDate,␣Amount)␣VALUES␣({
                            ↪ accountID},␣{paymentDate},{paymentAmountPerMonth});")

                    loanID=loanID+1
                    accountID=accountID+1
                #Finish Accounts
                costumerID=costumerID+1
            #Finish Costumer
```

Tables

Table Name	Number of rows
Person	30000
Costumer	19984
Employee	10016
Account	40533
CurrentAccount	20549
SavingsAccount	19984
Role	8
RoleHistory	10016
Transfer	19984
Withdraw	19984
Payment	10079
Loan	565
Deposit	30063
Branch	1252

Schema

```
CREATE TABLE Person (
  GovID number(19) NOT NULL,
  Name varchar2(255) NOT NULL,
  PhoneNumber varchar2(255) NOT NULL UNIQUE,
  Email varchar2(255) NOT NULL,
  Address varchar2(255) NOT NULL,
  Birthday date NOT NULL,
```

```
  Nationality varchar2(255) NOT NULL,
  PRIMARY KEY (GovID));

CREATE TABLE Costumer (
  CostumerID number(19) GENERATED AS IDENTITY,
  PersonGovID number(19) NOT NULL,
  PRIMARY KEY (CostumerID));

CREATE TABLE Employee (
  EmployeeID number(19) GENERATED AS IDENTITY,
  PersonGovID number(19) NOT NULL,
  Bonus number(10, 5) NOT NULL CHECK(Bonus>=0),
  SickDays number(10) NOT NULL CHECK(SickDays<10),
  PRIMARY KEY (EmployeeID));

CREATE TABLE Account (
  AccountID number(19) GENERATED AS IDENTITY,
  CostumerCostumerID number(19) NOT NULL,
  BranchBranchId number(19) NOT NULL,
  Amount number(19, 2) NOT NULL CHECK(Amount>=0),
  BeginDate date NOT NULL,
  EndDate date,
  PRIMARY KEY (AccountID));

CREATE TABLE Branch (
  BranchId number(19) GENERATED AS IDENTITY,
  Address varchar2(255) NOT NULL,
  PRIMARY KEY (BranchId));

CREATE TABLE RoleHistory (
  HistoryID number(19) GENERATED AS IDENTITY,
  RoleRoleID number(19) NOT NULL,
  EmployeeEmployeeID number(19) NOT NULL,
  BranchBranchId number(19) NOT NULL,
  BeginDate date NOT NULL,
  EndDate date,
  PRIMARY KEY (HistoryID));

CREATE TABLE Role (
  RoleID number(19) GENERATED AS IDENTITY,
  BaseSalary number(10) NOT NULL CHECK(BaseSalary>0),
  Name varchar2(255) NOT NULL,
  PRIMARY KEY (RoleID));

CREATE TABLE Loan (
  LoanID number(19) GENERATED AS IDENTITY,
  BranchBranchId number(19) NOT NULL,
  CurrentAccountAccountAccountID number(19) NOT NULL,
  Amount number(19, 2) NOT NULL CHECK(Amount>0),
  InterestRate number(10, 5) NOT NULL,
  DateOfCreation date NOT NULL,
  DateOfCompletion date NOT NULL,
  PRIMARY KEY (LoanID));

CREATE TABLE Payment (
  PaymentID number(19) GENERATED AS IDENTITY,
  LoanLoanID number(19) NOT NULL,
  CurrentAccountAccountID number(19) NOT NULL,
  PaymentDate date NOT NULL,
  Amount number(19, 2) NOT NULL CHECK(Amount>0),
  PRIMARY KEY (PaymentID));

```

```
70  CREATE TABLE SavingsAccount (
71    AccountAccountID number(19) NOT NULL,
72    InterestRate number(10, 5) NOT NULL CHECK(InterestRate>0),
73    DurationYears number(10) NOT NULL CHECK(DurationYears>0),
74    PRIMARY KEY (AccountAccountID));
75
76  CREATE TABLE Deposit (
77    DepositID number(19) GENERATED AS IDENTITY,
78    AccountAccountID number(19) NOT NULL,
79    DepositDate date NOT NULL,
80    Amount number(19, 2) NOT NULL CHECK(Amount>0),
81    PRIMARY KEY (DepositID));
82
83  CREATE TABLE Transfer (
84    TransferID number(19) GENERATED AS IDENTITY,
85    AccountAccountIDFrom number(19) NOT NULL,
86    AccountAccountIDTo number(19) NOT NULL,
87    TransferDate date NOT NULL,
88    Amount number(19, 2) NOT NULL CHECK(Amount>0),
89    PRIMARY KEY (TransferID));
90
91  CREATE TABLE Withdraw (
92    WithdrawID number(19) GENERATED AS IDENTITY,
93    CurrentAccountAccountID number(19) NOT NULL,
94    WithdrawDate date NOT NULL,
95    Amount number(19, 2) NOT NULL CHECK(Amount>0),
96    PRIMARY KEY (WithdrawID));
97
98  CREATE TABLE CurrentAccount (
99    AccountAccountID number(19) NOT NULL,
100   MaximumWithdraw number(10),
101   PRIMARY KEY (AccountAccountID));
102 ALTER TABLE Costumer ADD CONSTRAINT FKCostumer923053 FOREIGN KEY (PersonGovID) REFERENCES Person (
        ↪ GovID);
103
104 ALTER TABLE Employee ADD CONSTRAINT FKEmployee249023 FOREIGN KEY (PersonGovID) REFERENCES Person (
        ↪ GovID);
105
106 ALTER TABLE Account ADD CONSTRAINT FKAccount895601 FOREIGN KEY (CostumerCostumerID) REFERENCES
        ↪ Costumer (CostumerID);
107
108 ALTER TABLE RoleHistory ADD CONSTRAINT FKRoleHistor647811 FOREIGN KEY (RoleRoleID) REFERENCES Role (
        ↪ RoleID);
109
110 ALTER TABLE RoleHistory ADD CONSTRAINT FKRoleHistor516821 FOREIGN KEY (EmployeeEmployeeID) REFERENCES
        ↪ Employee (EmployeeID);
111
112 ALTER TABLE RoleHistory ADD CONSTRAINT FKRoleHistor171832 FOREIGN KEY (BranchBranchId) REFERENCES
        ↪ Branch (BranchId);
113
114 ALTER TABLE Loan ADD CONSTRAINT FKLoan357293 FOREIGN KEY (BranchBranchId) REFERENCES Branch (BranchId)
        ↪ ;
115
116 ALTER TABLE SavingsAccount ADD CONSTRAINT FKSavingsAcc25288 FOREIGN KEY (AccountAccountID) REFERENCES
        ↪ Account (AccountID);
117
118 ALTER TABLE Payment ADD CONSTRAINT FKPayment955503 FOREIGN KEY (LoanLoanID) REFERENCES Loan (LoanID);
119
120 ALTER TABLE Deposit ADD CONSTRAINT FKDeposit626030 FOREIGN KEY (AccountAccountID) REFERENCES Account (
        ↪ AccountID);
121
```

```
122 ALTER TABLE Transfer ADD CONSTRAINT FKTransfer731892 FOREIGN KEY (AccountAccountIDFrom) REFERENCES
        ↪ Account (AccountID);
123
124 ALTER TABLE Transfer ADD CONSTRAINT FKTransfer432158 FOREIGN KEY (AccountAccountIDTo) REFERENCES
        ↪ Account (AccountID);
125
126 ALTER TABLE Payment ADD CONSTRAINT FKPayment25568 FOREIGN KEY (CurrentAccountAccountID) REFERENCES
        ↪ CurrentAccount (AccountAccountID);
127
128 ALTER TABLE Withdraw ADD CONSTRAINT FKWithdraw546165 FOREIGN KEY (CurrentAccountAccountID) REFERENCES
        ↪ CurrentAccount (AccountAccountID);
129
130 ALTER TABLE CurrentAccount ADD CONSTRAINT FKCurrentAcc16041 FOREIGN KEY (AccountAccountID) REFERENCES
        ↪ Account (AccountID);
131
132 ALTER TABLE Account ADD CONSTRAINT FKAccount396299 FOREIGN KEY (BranchBranchId) REFERENCES Branch (
        ↪ BranchId);
133
134 ALTER TABLE Loan ADD CONSTRAINT FKLoan522632 FOREIGN KEY (CurrentAccountAccountAccountID) REFERENCES
        ↪ CurrentAccount (AccountAccountID);
```

Transactions

1:Changing Query

Description

This transaction doubles the amount of all the accounts under the average amount and then selects the name of the costumer that has the biggest amount in an Account.

Times

#	1	2	3	4	5
readings	5.533	3.778	3.312	4.045	3.55
average	4,0436				

SQL

```
1  ALTER SYSTEM FLUSH BUFFER_CACHE;
2
3  SET TIMING ON;
4
5  UPDATE Account
6  SET amount = amount*2
7  WHERE amount <= ALL(
8          SELECT Avg(amount)
9          FROM Account
10 );
11
12 SELECT Name
13     FROM Costumer
14     INNER JOIN Person
15         ON GovID = PersonGovID
16     INNER JOIN Account
17         ON CostumerID = CostumerCostumerID
18     WHERE amount >= ALL(
19         SELECT MAX(amount)
```

```
        FROM Account
);
```

2:Changing Query

Description

This transaction counts the number of SavingsAccount that have a duration bigger than 7 years and increases their interestRate by 0.03. If its less than 7 then it only increases by 0.01.

Times

#	1	2	3	4	5
readings	3.133	3.236	3.368	3.41	3.126
average	3,2546				

SQL

```
ALTER SYSTEM FLUSH BUFFER_CACHE;

SET TIMING ON;

select count(*)
from savingsaccount
where durationyears>7;

update SavingsAccount
set interestRate =
CASE
    WHEN durationYears>7 THEN interestrate + 0.03
    ELSE interestrate + 0.01
END;
```

3:Changing Query

Description

This transaction preforms a transfer between the SavingsAccount 2 and the CurrentAccount 1, it checks if the period of the SavingsAccount has passed and if so transfers the amount plus interest, if not only the amount. To do this we first add the amount to the CurrentAccount then we add a entry to the Transaction ledger and then we update the amount on the SavingsAccount.

Times

#	1	2	3	4	5
readings	11.997	12.286	11.737	13.495	11.975
average	12.298				

SQL

```
ALTER SYSTEM FLUSH BUFFER_CACHE;

SET TIMING ON;

```

```
DECLARE
    costID NUMBER:=1;

BEGIN

WHILE costID<700
LOOP
  UPDATE Account
SET amount =
  CASE
      WHEN (
      SELECT EXTRACT(YEAR FROM CURRENT_DATE) - EXTRACT(YEAR FROM (
          SELECT BeginDate
          FROM Account INNER JOIN SavingsAccount
            ON AccountID=AccountAccountID
          WHERE AccountAccountID=(
            select min(AccountID)
            from Account inner join SavingsAccount
                on AccountAccountID=AccountID
            where CostumerCostumerID=costID)))
        AS year FROM dual) > (
          SELECT DurationYears
          FROM SavingsAccount
          WHERE AccountAccountID=(
            select min(AccountID)
            from Account inner join SavingsAccount
                on AccountAccountID=AccountID
            where CostumerCostumerID=costID))
      THEN amount + (
          SELECT (amount+1)*12*DurationYears*InterestRate
          FROM SavingsAccount INNER JOIN Account
          ON AccountID=AccountAccountID
          WHERE AccountAccountID=(
            select min(AccountID)
            from Account inner join SavingsAccount
                on AccountAccountID=AccountID
            where CostumerCostumerID=costID))
      ELSE amount + (
      SELECT amount
      FROM SavingsAccount INNER JOIN Account
      ON AccountID=AccountAccountID
      WHERE AccountAccountID=(
            select min(AccountID)
            from Account inner join SavingsAccount
                on AccountAccountID=AccountID
            where CostumerCostumerID=costID))
  END
WHERE AccountID = (
    SELECT AccountID
    FROM CurrentAccount INNER JOIN Account
        ON AccountID=AccountAccountID
    WHERE AccountAccountID=(
            select min(AccountID)
            from Account inner join CurrentAccount
                on AccountAccountID=AccountID
            where CostumerCostumerID=costID));

INSERT INTO Transfer(TransferDate, Amount, AccountAccountIDFrom, AccountAccountIDTo) VALUES (
    ↪ CURRENT_DATE, (
Select
  CASE
      WHEN (
```

```
         SELECT EXTRACT(YEAR FROM CURRENT_DATE) - EXTRACT(YEAR FROM (
            SELECT BeginDate
            FROM Account INNER JOIN SavingsAccount
              ON AccountID=AccountAccountID WHERE AccountAccountID=(
           select min(AccountID)
           from Account inner join SavingsAccount
              on AccountAccountID=AccountID
           where CostumerCostumerID=costID)))
         AS year FROM dual) > (
            SELECT DurationYears
            FROM SavingsAccount
            WHERE AccountAccountID=(
           select min(AccountID)
           from Account inner join SavingsAccount
              on AccountAccountID=AccountID
           where CostumerCostumerID=costID))
      THEN amount*12*DurationYears*InterestRate
      ELSE amount
  END
FROM SavingsAccount INNER JOIN Account
       ON AccountID=AccountAccountID
    WHERE AccountAccountID=(
           select min(AccountID)
           from Account inner join SavingsAccount
              on AccountAccountID=AccountID
           where CostumerCostumerID=costID)),(
           select min(AccountID)
           from Account inner join SavingsAccount
              on AccountAccountID=AccountID
           where CostumerCostumerID=costID), (
           select min(AccountID)
           from Account inner join CurrentAccount
              on AccountAccountID=AccountID
           where CostumerCostumerID=costID));

UPDATE Account
SET amount = 0
WHERE AccountID = (
    SELECT AccountID
    FROM SavingsAccount INNER JOIN Account
        ON AccountID=AccountAccountID
    WHERE AccountAccountID=(
           select min(AccountID)
           from Account inner join SavingsAccount
              on AccountAccountID=AccountID
           where CostumerCostumerID=costID));
costID:=costID+1;
END LOOP;
END;
```

4:Changing Query

Description

This transaction upgrades the role of the first 5000 employees to role 1.

Times

#	1	2	3	4	5
readings	10.939	11.303	10.408	10.729	11.476
average	10,971				

SQL

```
ALTER SYSTEM FLUSH BUFFER_CACHE;

SET TIMING ON;

DECLARE
    empID NUMBER := 1;

BEGIN
WHILE empID<5000
LOOP
    INSERT INTO RoleHistory(RoleRoleID, EmployeeEmployeeID, BranchBranchId, BeginDate, EndDate) VALUES
        ↪ (
    1,
    empID,
    (SELECT BranchBranchID FROM RoleHistory WHERE EmployeeEmployeeID = empID AND EndDate is NULL),
    CURRENT_DATE,
    NULL);

    UPDATE RoleHistory
        SET EndDate = CURRENT_DATE
    WHERE HistoryID = (SELECT min(HistoryID) FROM RoleHistory WHERE EmployeeEmployeeID = empID AND
        ↪ EndDate is NULL);

empID:=empID+1;
END LOOP;
END;
```

5:Selecting Query

Description

This query selects the names of the persons that have money movements bigger than the average amount of movements.

SQL

Times

#	1	2	3	4	5
readings	1.331	1.499	1.539	1.389	1.857
average	1.523				

```
ALTER SYSTEM FLUSH BUFFER_CACHE;

set timing on;

select Name from (
```

```
select CostumerID as costID, sum(amount) as amount from (
select CostumerID, amount from (
select CostumerID, Sum(deposit.amount) as amount
from Deposit inner join Account
        on AccountID=AccountAccountID
    inner join Costumer
        on CostumerCostumerID=CostumerID
    GROUP BY CostumerID) union (
select CostumerID, Sum(transfer.amount) as amount
from Transfer inner join Account
        on AccountID=AccountAccountIDFrom
    inner join Costumer
        on CostumerCostumerID=CostumerID
    GROUP BY CostumerID) union (
select CostumerID, Sum(withdraw.amount) as amount
from withdraw inner join Account
        on AccountID=CurrentAccountAccountID
    inner join Costumer
        on CostumerCostumerID=CostumerID
    GROUP BY CostumerID) union (
select CostumerID, Sum(payment.amount) as amount
from Payment inner join Account
        on AccountID=CurrentAccountAccountID
    inner join Costumer
        on CostumerCostumerID=CostumerID
    GROUP BY CostumerID)
)
group by CostumerID
order by CostumerID) inner join Costumer
        on costID=CostumerID
    inner join Person
        on GovId=PersonGovID
where amount > (
select avg(amount) from (
select CostumerID, sum(amount) as amount from (
select CostumerID, amount from (
select CostumerID, Sum(deposit.amount) as amount
from Deposit inner join Account
        on AccountID=AccountAccountID
    inner join Costumer
        on CostumerCostumerID=CostumerID
    GROUP BY CostumerID) union (
select CostumerID, Sum(transfer.amount) as amount
from Transfer inner join Account
        on AccountID=AccountAccountIDFrom
    inner join Costumer
        on CostumerCostumerID=CostumerID
    GROUP BY CostumerID) union (
select CostumerID, Sum(withdraw.amount) as amount
from withdraw inner join Account
        on AccountID=CurrentAccountAccountID
    inner join Costumer
        on CostumerCostumerID=CostumerID
    GROUP BY CostumerID) union (
select CostumerID, Sum(payment.amount) as amount
from Payment inner join Account
        on AccountID=CurrentAccountAccountID
    inner join Costumer
        on CostumerCostumerID=CostumerID
    GROUP BY CostumerID)
)
group by CostumerID
```

```
order by CostumerID
));
```

6:Selecting Query

Description

This query shows the amount that was deposited on each day of each month of each year, each month of each year, each year, each day of each month, each month and each day.

Times

#	1	2	3	4	5
readings	6.369	3.765	3.305	3.276	2.939
average	3,9308				

SQL

```
ALTER SYSTEM FLUSH BUFFER_CACHE;

SET TIMING ON;

SELECT EXTRACT(YEAR FROM DepositDate) AS year, EXTRACT(MONTH FROM DepositDate) AS month, EXTRACT(DAY
    ↪ FROM DepositDate) AS day, SUM(amount)
FROM Deposit
GROUP BY CUBE(EXTRACT(YEAR FROM DepositDate), EXTRACT(MONTH FROM DepositDate), EXTRACT(DAY FROM
    ↪ DepositDate))
ORDER BY year, month, day;
```

7:Selecting Query

Description

This query shows the number of loans given by each branch by year, by month in year and by day in month in year.

Times

#	1	2	3	4	5
readings	16.528	13.040	17.354	16.977	17.264
average	16,2326				

SQL

```
ALTER SYSTEM FLUSH BUFFER_CACHE;

SET TIMING ON;

SELECT Address, EXTRACT(YEAR FROM DateOfCreation) AS year, EXTRACT(MONTH FROM DateOfCreation) AS month
    ↪ , EXTRACT(DAY FROM DateOfCreation) AS day, Count(*) AS total
FROM Loan INNER JOIN Branch
    on BranchBranchID = BranchBranchID
GROUP BY Rollup(branch.address, EXTRACT(YEAR FROM DateOfCreation), EXTRACT(MONTH FROM DateOfCreation),
    ↪  Extract(DAY FROM DateOfCreation))
ORDER BY Address, year, month, day;
```