Exercise 1:

**1.**

DECLARE

BEGIN

for customer in (select CustomerID, Name, DOB from Customers) loop

if(MONTHS\_BETWEEN(SYSDATE, customer.DOB)/12>60) then

update Loans

set InterestRate=InterestRate-1

where CustomerID=customer.CustomerID;

dbms\_output.put\_line('Customer Name : '||customer.name);

end if;

end loop;

END;

**2.**

alter table Customers add (IsVIP varchar2(3) default 'No');

BEGIN

for customer in (select CustomerID, Name, Balance from Customers) loop

if customer.Balance>10000 then

update Customers

set IsVIP='YES'

where CustomerID=customer.CustomerID;

end if;

end loop;

END;

**3.**

BEGIN

for loan in (select C.CustomerID, C.Name, L.LoanID, L.StartDate, L.EndDate

from Customers C join Loans L on C.CustomerID=L.CustomerID

where L.EndDate between SYSDATE and SYSDATE+30) loop

dbms\_output.put\_line('Reminder : Loan '||loan.LoanID||' for customer '||loan.Name||' is due on '||TO\_CHAR(loan.EndDate,'YYYY-MM-DD'));

end loop;

END;

Exercise 2:

**1.**

create or replace procedure SafeTransferFunds(

fromAccountID IN number, toAccountID IN number, amount IN number)

as

fromBalance number;

toBalance number;

BEGIN

select Balance into fromBalance from Accounts where AccountID=toAccountID;

select Balance into toBalance from Accounts where AccountID=fromAccountID;

if fromBalance<amount then

RAISE\_APPLICATION\_ERROR(-20001,'Insufficient funds in sender''s account. ');

end if;

update Accounts

set Balance=Balance-amount, LastModified=SYSDATE

where AccountID=fromAccountID;

update Accounts

set Balance=Balance+amount, LastModified=SYSDATE

where AccountID=toAccountID;

COMMIT;

EXCEPTION

when OTHERS then

ROLLBACK;

dbms\_output.put\_line('Error during fund transfer..'||SQLERRM);

END SafeTransferFunds;

**2.**

create or replace procedure UpdateSalary(

empID in number, percentage in number)

as

salary number;

BEGIN

select Salary into salary from Employees where EmployeeID=empID;

update Employees

set Salary=salary\*(1+percentage/100), HireDate=SYSDATE

where EmployeeID=empID;

COMMIT;

EXCEPTION

when NO\_DATA\_FOUND then

dbms\_output.put\_line('Employee with id '||empID||' does not exist.');

when OTHERS then

dbms\_output.put\_line('Error updating salary. '||SQLERRM);

ROLLBACK;

END UpdateSalary;

**3.**

create or replace procedure AddNewCustomer(

custID in number, custName in varchar2,

custDOB in date, balance in number)

is

BEGIN

insert into Customers(CustomerID, Name, DOB, Balance, LastModified)

values(custID,custName,custDOB,balance,SYSDATE);

COMMIT;

EXCEPTION

when DUP\_VAL\_ON\_INDEX then

dbms\_output.put\_line('Customer with id '||custID||' already exists.');

ROLLBACK;

when OTHERS then

dbms\_output.put\_line('Error adding new customer. '||SQLERRM);

ROLLBACK;

END AddNewCustomer;

Exercise 3:

**1.**

create or replace procedure ProcessMonthlyInterest as

BEGIN

for account in (select AccountID, Balance from Accounts where AccountType='Savings') loop

update Accounts

set Balance=Balance\*1.01, LastModified=SYSDATE

where AccountID=account.AccountID;

end loop;

commit;

END ProcessMonthlyInterest;

**2.**

create or replace procedure UpdateEmployeeBonus(

dept in number,

percentage in number

) as

BEGIN

for employee in (select EmployeeID, Salary from Employees where Department=dept) loop

update Employees

set Salary=Salary\*(1+percentage/100)

where EmployeeID=employee.EmployeeID;

end loop;

commit;

END UpdateEmployeeBonus;

**3.**

create or replace procedure TransferFunds(

fromAcc in number,

toAcc in number,

amount in number

) as

fromBal number;

BEGIN

select Balance into fromBal from Accounts where AccountID=fromAcc;

if fromBal<amount then

RAISE\_APPLICATION\_ERROR(-20001,'Insufficient Funds in sender account.');

end if;

update Accounts

set Balance=Balance-amount, LastModified=SYSDATE

where AccountID=fromAcc;

update Accounts

set Balance=Balance+amount, LastModified=SYSDATE

where AccountID=toAcc;

commit;

EXCEPTION

when OTHERS then

rollback;

dbms\_output.put\_line('Error transfering funds. '||SQLERRM);

END TransferFunds;

Exercise 4:

**1.**

create or replace function CalculateAge(dob date)

return number is

age number;

BEGIN

age:=FLOOR(MONTHS\_BETWEEN(SYSDATE,dob)/12);

return age;

END CalculateAge;

**2.**

create or replace function CalculateMonthlyInstallment(

loanAmount in number, interestRate in number,

loanDuration in number) return number is

monthlyInstallment number;

monthlyInterest number;

numberOfPayments number;

BEGIN

monthlyInterest:=interestRate/12/100;

numberOfPayments:=loanDuration\*12;

if monthlyInterest=0 then

monthlyInstallment:=loanAmount/numberOfPayments;

else

monthlyInstallment:=loanAmount\*monthlyInterest/(1-POWER(1+monthlyInterest,-numberOfPayments));

end if;

return monthlyInstallment;

END CalculateMonthlyInstallment;

**3.**

create or replace function HasSufficientBalance(

accID number,

amount number

) return boolean as

balance number;

BEGIN

select Balance into balance from Accounts where AccountID=accID;

if balance>=amount then

return true;

else

return false;

end if;

EXCEPTION

when NO\_DATA\_FOUND then

return false;

END HasSufficientBalance;

Exercise 5:

**1.**

create or replace trigger UpdateCustomerLastModified

before update on Customers

for each row

BEGIN

:NEW.LastModified:=SYSDATE;

END UpdateCustomerLastModified;

**2.**

create table AuditLog(

LogID number primary key, TransactionID number, AccountID number, TransactionDate date, Amount number, TransactionType varchar2(10), LogDate date

);

create sequence AuditLog\_SEQ

start with 1

increment by 1

nocache;

create or replace trigger LogTransaction

after insert on Transactions

for each row

BEGIN

insert into AuditLog(LogID, TransactionID, AccountID, TransactionDate, Amount, TransactionType, LogDate)

values(AuditLog\_SEQ.NEXTVAL,:NEW.TransactionID,:NEW.AccountID,:NEW.TransactionDate,:NEW.Amount,:NEW.TransactionType,SYSDATE);

END LogTransaction;

**3.**

create or replace trigger CheckTransactionRules

before insert on Transactions

for each row

DECLARE

balance number;

BEGIN

select Balance into balance from Accounts where AccountID=:NEW.AccountID;

if :NEW.TransactionType='Withdrawal' then

if :NEW.Amount>balance then

RAISE\_APPLICATION\_ERROR(-20001,'Insufficient funds for withdrawal.');

end if;

end if;

if :NEW.TransactionType='Deposit' then

if :NEW.Amount<0 then

RAISE\_APPLICATION\_ERROR(-20001,'Deposits must be positive.');

end if;

end if;

END CheckTransactionRules;

Exercise 6:

**1.**

DECLARE

cursor GenerateMonthlyStatement is

select C.CustomerID, C.Name, T.Amount, T.TransactionDate, T.TransactionType

from Customers C

join Accounts A on C.CustomerID=A.CustomerID

join Transactions T on A.AccountID=T.AccountID

where T.TransactionDate between TRUNC(SYSDATE,'MM') and LAST\_DAY(SYSDATE);

custID Customers.CustomerID%type;

name Customers.Name%type;

amount Transactions.Amount%type;

tDate Transactions.TransactionDate%type;

tType Transactions.TransactionType%type;

BEGIN

OPEN GenerateMonthlyStatement;

loop

fetch GenerateMonthlyStatement into custID,name,amount,tDate,tType;

exit when GenerateMonthlyStatement%NOTFOUND;

dbms\_output.put\_line('Customer ID: '||custID||', Name: '||name||', Amount: '||amount||', Date: '||TO\_CHAR(tDate,'YYYY-MM-DD')||', Type: '||tType);

end loop;

CLOSE GenerateMonthlyStatement;

END;

**2.**

DECLARE

cursor ApplyAnnualFee is

select AccountID, Balance

from Accounts;

accID Accounts.AccountID%type;

balance Accounts.Balance%type;

annualFee number:=50;

BEGIN

open ApplyAnnualFee;

loop

fetch ApplyAnnualFee into accID, balance;

exit when ApplyAnnualFee%NOTFOUND;

update Accounts

set Balance=Balance-annualFee, LastModified=SYSDATE

where AccountID=accID;

dbms\_output.put\_line('Account ID: '||accID||', New Balance: '||(balance-annualFee));

end loop;

close ApplyAnnualFee;

commit;

END;

**3.**

DECLARE

cursor UpdateLoanInterestRates is

select LoanID, InterestRate

from Loans;

loanID Loans.LoanID%type;

rate Loans.InterestRate%type;

newRate number;

BEGIN

open UpdateLoanInterestRates;

loop

fetch UpdateLoanInterestRates into loanID, rate;

exit when UpdateLoanInterestRates%NOTFOUND;

newRate:=rate+0.5;

update Loans

-- Suppose interest rate increase by 0.5

set InterestRate=newRate

where LoanID=loanID;

dbms\_output.put\_line('Loan ID: '||loanID||', Interest Rate: '||newRate);

end loop;

close UpdateLoanInterestRates;

commit;

END;

Exercise 7:

**1.**

create or replace package CustomerManagement as

procedure AddCustomer(

custID in number,

custName in varchar2,

dob in date,

balance in number

);

procedure UpdateCustomerDetails(

custID in number,

custName in varchar2,

dob in date,

balance in number

);

function GetCustomerBalance(

custID in number

) return number;

end CustomerManagement;

create or replace package body CustomerManagement as

procedure AddCustomer(

custID in number,

custName in varchar2,

dob in date,

balance in number

) is

BEGIN

insert into Customers(CustomerID,Name,DOB,Balance,LastModified)

values (custID,custName,dob,balance,SYSDATE);

END AddCustomer;

procedure UpdateCustomerDetails(

custID in number,

custName in varchar2,

dob in date,

balance in number

) is

BEGIN

update Customers

set Name=custName,

DOB=dob,

Balance=balance,

LastModified=SYSDATE

where CustomerID=custID;

END UpdateCustomerDetails;

function GetCustomerBalance(

custID in number

) return number is

balance number;

BEGIN

select Balance into balance

from Customers

where CustomerID=custID;

return balance;

EXCEPTION

when NO\_DATA\_FOUND then

return null;

END GetCustomerBalance;

END CustomerManagement;

**2.**

create or replace package EmployeeManagement as

procedure HireEmployee(

empID in number,

name in varchar2,

position in varchar2,

salary in number,

dept in varchar2,

hireDate in date

);

procedure UpdateEmployeeDetails(

empID in number,

name in varchar2,

position in varchar2,

salary in number,

dept in varchar2

);

function CalaculateAnnualSalary(

empID in number

) return number;

end EmployeeManagement;

create or replace package body EmployeeManagement as

procedure HireEmployee(

empID in number,

name in varchar2,

position in varchar2,

salary in number,

dept in varchar2,

hireDate in date

) is

BEGIN

insert into Employees (EmployeeID, Name, Position, Salary, Department, HireDate)

values (empID, name, position, salary, dept, hireDate);

END HireEmployee;

procedure UpdateEmployeeDetails(

empID in number,

name in varchar2,

position in varchar2,

salary in number,

dept in varchar2

) is

BEGIN

update Employees

set Name=name,

Position=position,

Salary=salary,

Department=dept

where EmployeeID=empID;

END UpdateEmployeeDetails;

function CalculateAnnualSalary(

empID in number

) return number is

salary number;

BEGIN

select Salary into salary

from Employees

where EmployeeID=empID;

return salary\*12;

EXCEPTION

when NO\_DATA\_FOUND then

return null;

END CalculateAnnualSalary;

END EmployeeManagement;

**3.**

create or replace package AccountOperations as

procedure OpenAccount(

accID in number,

custID in number,

accType in varchar2,

balance in number

);

procedure CloseAccount(

accID in number

);

function GetTotalCustomerBalance(

custID in number

) return number;

end AccountOperations;

create or replace package body AccountOperations as

procedure OpenAccount(

accID in number,

custID in number,

accType in varchar2,

balance in number

) is

BEGIN

insert into Accounts(AccountID,CustomerID,AccountType,Balance,LastModified)

values(accID, custID, accType, balance, SYSDATE);

END OpenAccount;

procedure CloseAccount(

accID in number

) is

BEGIN

delete from Accounts

where AccountID=accID;

END CloseAccount;

function GetTotalCustomerBalance(

custID in number

) return number is

totalBalance number;

BEGIN

select SUM(Balance) into totalBalance

from Accounts

where CustomerID=custID;

return totalBalance;

EXCEPTION

when NO\_DATA\_FOUND then

return 0;

END GetTotalCustomerBalance;

END AccountOperations;