**Control statements**

1.

DECLARE cust\_id customers.customerid % type;

cust\_interest\_rate loans.interestrate % type;

cust\_age integer := 0;

cursor customer\_cursor is

select

   customers.customerid,

   loans.interestrate,

   TRUNC(Months\_Between(sysdate, customers.DOB) / 12)

from

   customers,

   loans

where

   (

      customers.customerid = loans.customerid

   );

BEGIN

   open customer\_cursor;

loop fetch customer\_cursor into cust\_id,

cust\_interest\_rate,

cust\_age;

exit

when

   customer\_cursor % notfound;

update

   loans

set

   interestrate = interestrate - (0.01\*interestrate)

where

   cust\_id = customerid

   and interestrate = cust\_interest\_rate

   and cust\_age > 60;

End loop;

close customer\_cursor;

end;

/

………………………………………………………………………………………………………………………

**2.**

alter table customers add IsVIP char(1);

DECLARE cust\_balance customers.balance % TYPE;

cust\_name customers.name % TYPE;

CURSOR customerCursor IS

SELECT

   balance,

   name

FROM

   customers;

BEGIN

   OPEN customerCursor;

LOOP FETCH customerCursor INTO cust\_balance,

cust\_name;

EXIT

WHEN

   customerCursor % NOTFOUND;

IF cust\_balance > 10000

THEN

   UPDATE

      customers

   SET

      IsVIP = TRUE

   WHERE

      name = cust\_name;

ELSE

   UPDATE

      customers

   SET

      IsVIP = FALSE

   WHERE

      name = cust\_name;

END IF;

END

LOOP;

CLOSE customerCursor;

END;

/

…………………………………………………………………………………………………………

**3.**

DECLARE cust\_name customers.name % type;

cust\_end\_date loans.enddate % type;

CURSOR remainderCursor IS

SELECT

   loans.enddate,

   customers.name

from

   loans,

   customers

where

   loans.customerid = customers.customerid;

BEGIN

   OPEN remainderCursor;

LOOP FETCH remainderCursor into cust\_end\_date,

cust\_name;

EXIT

WHEN

   remainderCursor % notfound;

IF cust\_end\_date BETWEEN SYSDATE AND SYSDATE + 30

THEN

   dbms\_output.put\_line('Remainder: Dear Customer ' || cust\_name || ', loan due deadline is within next 30 days');

END

IF;

END

LOOP;

END

;

/

—-----------------------------------------------------------------------------------------------------------------------------------------

**2.ERROR HANDLING**

**1.**

CREATE

OR REPLACE PROCEDURE SafeTransferFunds ( p\_from\_customer\_id IN NUMBER, p\_to\_customer\_id IN NUMBER, p\_amount IN NUMBER ) AS v\_from\_balance Customers.Balance % TYPE;

v\_to\_balance Customers.Balance % TYPE;

v\_transaction\_id NUMBER;

BEGIN

*-- Validate transfer amount*

   IF p\_amount <= 0

THEN

   RAISE\_APPLICATION\_ERROR( - 20001, 'Transfer amount must be greater than zero.');

END

IF;

-- Fetch and lock source account balance

SELECT

   Balance INTO v\_from\_balance

FROM

   Customers

WHERE

   CustomerID = p\_from\_customer\_id FOR

   UPDATE

;

-- Check for sufficient funds

IF v\_from\_balance < p\_amount

THEN

   RAISE\_APPLICATION\_ERROR( - 20002, 'Insufficient funds.');

END

IF;

-- Fetch and lock destination account balance

SELECT

   Balance INTO v\_to\_balance

FROM

   Customers

WHERE

   CustomerID = p\_to\_customer\_id FOR

   UPDATE

;

-- Perform fund transfer

UPDATE

   Customers

SET

   Balance = Balance - p\_amount,

   LastModified = SYSDATE

WHERE

   CustomerID = p\_from\_customer\_id;

UPDATE

   Customers

SET

   Balance = Balance + p\_amount,

   LastModified = SYSDATE

WHERE

   CustomerID = p\_to\_customer\_id;

-- Generate a unique transaction ID

SELECT

   NVL(MAX(TransactionID), 0) + 1 INTO v\_transaction\_id

FROM

   Transactions;

-- Log transactions

INSERT INTO

   Transactions (TransactionID, AccountID, TransactionDate, Amount, TransactionType)

VALUES

   (

      v\_transaction\_id,

      p\_from\_customer\_id,

      SYSDATE,

      - p\_amount,

      'debit'

   )

;

INSERT INTO

   Transactions (TransactionID, AccountID, TransactionDate, Amount, TransactionType)

VALUES

   (

      v\_transaction\_id + 1,

      p\_to\_customer\_id,

      SYSDATE,

      p\_amount,

      'credit'

   )

;

-- Commit changes

COMMIT;

-- Success message

dbms\_output.put\_line('Transfer successful.');

EXCEPTION

WHEN

   NO\_DATA\_FOUND

THEN

   ROLLBACK;

dbms\_output.put\_line('Error: Customer not found.');

WHEN

   OTHERS

THEN

   ROLLBACK;

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

END

SafeTransferFunds;

/

BEGIN

   SafeTransferFunds( p\_from\_customer\_id => 1, p\_to\_customer\_id => 2, p\_amount => 1000 );

END;

/

………………………………………………………………………………………………………………………

**2.**

CREATE

OR REPLACE PROCEDURE UpdateSalary ( p\_employeeid IN NUMBER, p\_percentageincrease IN NUMBER )AS v\_employeeid Employees.Employeeid % TYPE;

v\_salary Employees.Salary % TYPE;

Emp\_available INTEGER := 0;

BEGIN

   SELECT

      count(\*) INTO Emp\_available

   FROM

      Employees

   WHERE

      employeeid = p\_employeeid;

IF Emp\_available = 0

THEN

   RAISE\_APPLICATION\_ERROR( - 20001, 'No Employee with the provided ID exists!');

END

IF;

UPDATE

   Employees

SET

   SALARY = SALARY + (p\_percentageincrease / 100)\*SALARY

WHERE

   EMPLOYEEID = p\_employeeid;

EXCEPTION

WHEN

   NO\_DATA\_FOUND

THEN

   dbms\_output.put\_line('Data not found');

WHEN

   OTHERS

THEN

   dbms\_output.put\_line('Some error occured');

END

;

/

BEGIN

   UpdateSalary( p\_employeeid => 1, p\_percentageincrease => 50 );

END

;

/

……………………………………………………………………………………………………………………..

**3.**

CREATE

OR REPLACE PROCEDURE AddNewCustomer( p\_customerid IN NUMBER, p\_name IN VARCHAR, p\_DOB IN DATE, p\_balance IN NUMBER, p\_lastmodified IN DATE )AS Customer\_available INTEGER := 0;

IsVIP VARCHAR(5):= 'FALSE';

BEGIN

   SELECT

      Count(\*) INTO customer\_available

   FROM

      CUSTOMERS

   WHERE

      CUSTOMERID = p\_customerid;

IF customer\_available != 0

THEN

   RAISE\_APPLICATION\_ERROR( - 20001, 'Can''t add new customers as customer with the same ID already exists!');

END

IF;

IF p\_balance > 10000

THEN

   IsVIP:= 'TRUE';

END

IF;

INSERT INTO

   CUSTOMERS

VALUES

   (

      p\_customerid, p\_name, p\_DOB, p\_balance, p\_lastmodified, IsVIP

   )

;

dbms\_output.put\_line('Added new customer successfully!');

EXCEPTION

WHEN

   NO\_DATA\_FOUND

THEN

   dbms\_output.put\_line('Data not found');

WHEN

   OTHERS

THEN

   dbms\_output.put\_line('Can''t add new customers as customer with the same ID already exists!');

END

;

/

BEGIN

   AddNewCustomer( p\_customerid => 3, p\_name => 'Jenny', p\_balance => 20000, p\_DOB => SYSDATE - 30, p\_lastmodified => SYSDATE );

END

;

/

—-----------------------------------------------------------------------------------------------------------------------------------------

**3.PROCEDURES**

**1.**

CREATE

OR REPLACE PROCEDURE ProcessMonthlyInterest IS

BEGIN

   UPDATE

      ACCOUNTS

   SET

      BALANCE = BALANCE + BALANCE\*0.01

   WHERE

      ACCOUNTTYPE = 'SAVINGS';

END

/

BEGIN

   ProcessMonthlyInterest();

END;

/

**…………………………………………………………………………………………………………………………**

**2.**

CREATE

OR REPLACE PROCEDURE UpdateSalary ( p\_employeedepartment IN VARCHAR2, p\_percentageincrease IN INTEGER := 0 ) IS v\_department\_count INTEGER;

BEGIN

   SELECT

      COUNT(\*) INTO v\_department\_count

   FROM

      Employees

   WHERE

      DEPARTMENT = p\_employeedepartment;

IF v\_department\_count = 0

THEN

   RAISE\_APPLICATION\_ERROR( - 20001, 'Department ' || p\_employeedepartment || ' does not exist.');

END

IF;

UPDATE

   Employees

SET

   SALARY = SALARY + (p\_percentageincrease / 100) \* SALARY

WHERE

   DEPARTMENT = p\_employeedepartment;

DBMS\_OUTPUT.PUT\_LINE('Salaries updated for department: ' || p\_employeedepartment);

EXCEPTION

WHEN

   OTHERS

THEN

   DBMS\_OUTPUT.PUT\_LINE('An error occurred: ' || SQLERRM);

END;

/

………………………………………………………………………………………………………………………….

**3.**

CREATE

OR REPLACE PROCEDURE TransferFunds( p\_sender\_id IN NUMBER, p\_amount IN NUMBER, p\_receiver\_id IN NUMBER )AS p\_sender\_available INTEGER := 0;

p\_receiver\_available INTEGER := 0;

p\_balance INTEGER := 0;

BEGIN

   SELECT

      COUNT(\*),

      SUM(balance) INTO p\_sender\_available,

      p\_balance

   FROM

      CUSTOMERS

   WHERE

      CUSTOMERID = p\_sender\_id;

IF p\_sender\_available = 0

THEN

   RAISE\_APPLICATION\_ERROR( - 20001, 'No such sender exists');

END

IF;

IF p\_balance < p\_amount

THEN

   RAISE\_APPLICATION\_ERROR( - 20002, 'Balance is not sufficient');

END

IF;

SELECT

   COUNT(\*) INTO p\_receiver\_available

FROM

   CUSTOMERS

WHERE

   CUSTOMERID = p\_receiver\_id;

IF p\_receiver\_available = 0

THEN

   RAISE\_APPLICATION\_ERROR( - 20003, 'No such receiver exists');

END

IF;

UPDATE

   CUSTOMERS

SET

   balance = balance - p\_amount

where

   CUSTOMERID = p\_sender\_id;

UPDATE

   CUSTOMERS

SET

   balance = balance + p\_amount

where

   CUSTOMERID = p\_receiver\_id;

dbms\_output.put\_line('Funds transferred successfully');

EXCEPTION

WHEN

   OTHERS

THEN

   dbms\_output.put\_line('Transaction failed:' || SQLERRM);

END;

/

—-----------------------------------------------------------------------------------------------------------------------------------------

**4.FUNCTIONS**

**1**.

CREATE

OR REPLACE FUNCTION CalculateAge( dob IN DATE )RETURN NUMBER IS age NUMBER;

BEGIN

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

RETURN age;

EXCEPTION

WHEN

   OTHERS

THEN

   dbms\_output.put\_line('Invalid input ');

END;

/

………………………………………………………………………………………………………………………

**2.**

CREATE

OR REPLACE FUNCTION CalculateAge( dob IN DATECREATE

OR REPLACE FUNCTION CalculateMonthlyInstallment( p\_loanAmount IN NUMBER, p\_interestRate IN NUMBER, p\_loanDuration IN NUMBER ) RETURN NUMBER IS v\_monthlyRate NUMBER;

v\_totalInstallments NUMBER;

v\_monthlyInstallment NUMBER;

BEGIN

   v\_monthlyRate := p\_interestRate / 100 / 12;

v\_totalInstallments := p\_loanDuration \* 12;

v\_monthlyInstallment :=

(

   p\_loanAmount \* v\_monthlyRate \* POWER(1 + v\_monthlyRate, v\_totalInstallments)

)

/ (POWER(1 + v\_monthlyRate, v\_totalInstallments) - 1);

RETURN v\_monthlyInstallment;

EXCEPTION

WHEN

   OTHERS

THEN

   DBMS\_output.put\_line('Some error occured!');

END;

/

SELECT

   CalculateMonthlyInstallment(25000, 4.5, 5) AS monthlyInstallment

FROM

   dual;

)RETURN NUMBER IS age NUMBER;

BEGIN

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

RETURN age;

EXCEPTION

WHEN

   OTHERS

THEN

   dbms\_output.put\_line('Invalid input ');

END;

/

……………………………………………………………………………………………………………………

**3.**

CREATE

OR REPLACE FUNCTION CalculateAge( dob IN DATECREATE

OR REPLACE FUNCTION CalculateMonthlyInstallment( p\_loanAmount IN NUMBER, p\_interestRate IN NUMBER, p\_loanDuration IN NUMBER ) RETURN NUMBER IS v\_monthlyRate NUMBER;

v\_totalInstallments NUMBER;

v\_monthlyInstallment NUMBER;

BEGIN

   v\_monthlyRate := p\_interestRate / 100 / 12;

v\_totalInstallments := p\_loanDuration \* 12;

v\_monthlyInstallment :=

(

   p\_loanAmount \* v\_monthlyRate \* POWER(1 + v\_monthlyRate, v\_totalInstallments)

)

/ (POWER(1 + v\_monthlyRate, v\_totalInstallments) - 1);

RETURN v\_monthlyInstallment;

EXCEPTION

WHEN

   OTHERS

THEN

   DBMS\_output.put\_line('Some error occured!');

END;

/

SELECT

   CalculateMonthlyInstallment(25000, 4.5, 5) AS monthlyInstallment

FROM

   dual;

)RETURN NUMBER IS age NUMBER;

BEGIN

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

RETURN age;

EXCEPTION

WHEN

   OTHERS

THEN

   dbms\_output.put\_line('Invalid input ');

END;

/

—----------------------------------------------------------------------------------------------------------------------------------------

**5.TRIGGER**

**1.**

CREATE

OR REPLACE TRIGGER UpdateCustomerLastModified BEFORE

UPDATE

   ON Customers FOR EACH ROW

   BEGIN

      :NEW.LastModified:= SYSDATE;

   END;

/

………………………………………………………………………………………………………………………….

**2**.

CREATE TABLE AuditLog ( TRANSACTIONID INT, ACCOUNTID INT, TRANSACTIONDATE DATE, AMOUNT INT, TRANSACTIONTYPE VARCHAR2(25) );

CREATE

OR REPLACE TRIGGER LogTransaction BEFORE INSERT

ON Transactions FOR EACH ROW

BEGIN

   INSERT INTO

      AuditLog (TRANSACTIONID, ACCOUNTID, TRANSACTIONDATE, AMOUNT, TRANSACTIONTYPE)

   VALUES

      (

         :NEW.TRANSACTIONID, :NEW.ACCOUNTID, :NEW.TRANSACTIONDATE, :NEW.AMOUNT, :NEW.TRANSACTIONTYPE

      );

END;

/**3.**

CREATE

OR REPLACE TRIGGER CheckTransactionRules BEFORE INSERT

ON Transactions FOR EACH ROW

DECLARE p\_balance Accounts.balance % TYPE;

BEGIN

   SELECT

      balance INTO p\_balance

   FROM

      ACCOUNTS

   WHERE

      ACCOUNTID = :NEW.ACCOUNTID;

IF (:NEW.AMOUNT > p\_balance

AND :NEW.TransactionType = 'Withdrawal')

OR

(

   :NEW.AMOUNT < 0

   AND :NEW.TransactionType = 'Deposit'

)

THEN

   RAISE\_APPLICATION\_ERROR( - 20003, 'Invalid transaction');

END IF;

END;

/

—-----------------------------------------------------------------------------------------------------------------------------------------

**6.CURSORS**

**1.**

DECLARE CURSOR GenerateMonthlyStatements IS

SELECT

   t.TRANSACTIONID,

   t.ACCOUNTID,

   t.AMOUNT,

   t.TRANSACTIONDATE,

   c.NAME

FROM

   TRANSACTIONS t

   JOIN

      ACCOUNTS a

      ON t.ACCOUNTID = a.ACCOUNTID

   JOIN

      CUSTOMERS c

      ON a.CUSTOMERID = c.CUSTOMERID

WHERE

   EXTRACT(MONTH

FROM

   t.TRANSACTIONDATE) = EXTRACT(MONTH

FROM

   SYSDATE)

   AND EXTRACT(YEAR

FROM

   t.TRANSACTIONDATE) = EXTRACT(YEAR

FROM

   SYSDATE);

v\_TransactionID Transactions.TransactionID % TYPE;

v\_AccountID Transactions.AccountID % TYPE;

v\_Amount Transactions.Amount % TYPE;

v\_TransactionDate Transactions.TransactionDate % TYPE;

v\_CustomerName Customers.Name % TYPE;

BEGIN

   OPEN GenerateMonthlyStatements;

LOOP FETCH GenerateMonthlyStatements INTO v\_TransactionID,

v\_AccountID,

v\_Amount,

v\_TransactionDate,

v\_CustomerName;

EXIT

WHEN

   GenerateMonthlyStatements % NOTFOUND;

DBMS\_OUTPUT.PUT\_LINE('Transaction ID: ' || v\_TransactionID);

DBMS\_OUTPUT.PUT\_LINE('Customer Name: ' || v\_CustomerName);

DBMS\_OUTPUT.PUT\_LINE('Account ID: ' || v\_AccountID);

DBMS\_OUTPUT.PUT\_LINE('Amount: ' || v\_Amount);

DBMS\_OUTPUT.PUT\_LINE('Transaction Date: ' || v\_TransactionDate);

DBMS\_OUTPUT.PUT\_LINE('-----------------------------');

END

LOOP;

CLOSE GenerateMonthlyStatements;

END;

/

………………………………………………………………………………………………………………………

**2.**

DECLARE ANNUALFEE NUMBER := 250;

CURSOR ApplyAnnualFee IS

SELECT

   a.BALANCE,

   a.ACCOUNTID,

   c.NAME

FROM

   ACCOUNTS a

   JOIN

      CUSTOMERS c

      ON c.CUSTOMERID = a.CUSTOMERID;

v\_balance ACCOUNTS.BALANCE % TYPE;

v\_accountid ACCOUNTS.ACCOUNTID % TYPE;

v\_customername CUSTOMERS.NAME % TYPE;

BEGIN

   OPEN ApplyAnnualFee;

LOOP FETCH ApplyAnnualFee INTO v\_balance,

v\_accountid,

v\_customername;

EXIT

WHEN

   ApplyAnnualFee % NOTFOUND;

IF v\_balance < ANNUALFEE

THEN

   RAISE\_APPLICATION\_ERROR( - 20003, 'INSUFFICIENT BALANCE TO DEDUCT ANNUAL FEE');

END

IF;

UPDATE

   ACCOUNTS

SET

   BALANCE = BALANCE - ANNUALFEE

WHERE

   ACCOUNTID = v\_accountid;

DBMS\_OUTPUT.PUT\_LINE('DEAR ' || v\_customername || 'Annual Fee $250 got deducted from your account.Thank you!');

END

LOOP;

CLOSE ApplyAnnualFee;

EXCEPTION

WHEN

   OTHERS

THEN

   DBMS\_OUTPUT.PUT\_LINE(SQLERRM);

END;

/

……………………………………………………………………………………………………………………

**3.**

DECLARE NewPolicyInterestRate NUMBER := 10;

CURSOR UpdateLoanInterestRates IS

SELECT

   INTERESTRATE,

   c.NAME,

   c.CUSTOMERID

FROM

   LOANS l

   JOIN

      CUSTOMERS c

      ON c.CUSTOMERID = l.CUSTOMERID;

v\_interestrate LOANS.INTERESTRATE % TYPE;

v\_customername CUSTOMERS.NAME % TYPE;

v\_customerid CUSTOMERS.CUSTOMERID % TYPE;

BEGIN

   OPEN UpdateLoanInterestRates;

LOOP FETCH UpdateLoanInterestRates INTO v\_interestrate,

v\_customername,

v\_customerid;

EXIT

WHEN

   UpdateLoanInterestRates % NOTFOUND;

UPDATE

   LOANS

SET

   INTERESTRATE = NewPolicyInterestRate

WHERE

   CUSTOMERID = v\_customerid;

DBMS\_OUTPUT.PUT\_LINE('DEAR ' || v\_customername || ', Interest rate of your loan got updated to 20% based on company''s new policy.Thank you!');

END

LOOP;

CLOSE UpdateLoanInterestRates;

END;

/

—-----------------------------------------------------------------------------------------------------------------------------------------

**7.PACKAGES**

**1.**

CREATE

OR REPLACE PACKAGE CustomerManagement AS PROCEDURE add\_customers( P\_CUSTOMERID NUMBER, P\_NAME VARCHAR2, P\_DOB DATE, P\_BALANCE NUMBER, P\_LASTMODIFIED DATE );

PROCEDURE update\_customer( P\_CUSTOMERID NUMBER, P\_BALANCE NUMBER );

FUNCTION get\_customerbalance( P\_CUSTOMERID NUMBER ) RETURN NUMBER;

END

CustomerManagement;

/ CREATE

OR REPLACE PACKAGE BODY CustomerManagement AS PROCEDURE add\_customers( P\_CUSTOMERID NUMBER, P\_NAME VARCHAR2, P\_DOB DATE, P\_BALANCE NUMBER, P\_LASTMODIFIED DATE ) IS

BEGIN

   INSERT INTO

      CUSTOMERS (CUSTOMERID, NAME, DOB, BALANCE, LASTMODIFIED)

   VALUES

      (

         P\_CUSTOMERID, P\_NAME, P\_DOB, P\_BALANCE, P\_LASTMODIFIED

      )

;

DBMS\_OUTPUT.PUT\_LINE('CUSTOMER ADDED SUCCESSFULLY!');

END

add\_customers;

PROCEDURE update\_customer( P\_CUSTOMERID NUMBER, P\_BALANCE NUMBER ) IS

BEGIN

   UPDATE

      CUSTOMERS

   SET

      BALANCE = P\_BALANCE

   WHERE

      CUSTOMERID = P\_CUSTOMERID;

DBMS\_OUTPUT.PUT\_LINE('CUSTOMER''S BALANCE UPDATED SUCCESSFULLY!');

END

update\_customer;

FUNCTION get\_customerbalance( P\_CUSTOMERID NUMBER ) RETURN NUMBER IS v\_balance NUMBER;

BEGIN

   SELECT

      BALANCE INTO v\_balance

   FROM

      CUSTOMERS

   WHERE

      CUSTOMERID = P\_CUSTOMERID;

RETURN v\_balance;

EXCEPTION

WHEN

   NO\_DATA\_FOUND

THEN

   RETURN NULL;

WHEN

   OTHERS

THEN

   DBMS\_OUTPUT.PUT\_LINE('Error retrieving customer balance: ' || SQLERRM);

RETURN NULL;

END

get\_customerbalance;

END

CustomerManagement;

/

DECLARE v\_balance NUMBER;

BEGIN

   CustomerManagement.add\_customers(12, 'Tejas', TO\_DATE('2003-03-02', 'YYYY-MM-DD'), 50000, SYSDATE);

CustomerManagement.update\_customer(12, 70000);

v\_balance := CustomerManagement.get\_customerbalance(12);

DBMS\_OUTPUT.PUT\_LINE(v\_balance);

END;

/

…………………………………………………………………………………………………………………………..

**2.**

.CREATE

OR REPLACE PACKAGE EmployeeManagement AS PROCEDURE hire\_employee( P\_EMPLOYEEID NUMBER, P\_NAME VARCHAR2, P\_POSITION VARCHAR2, P\_SALARY NUMBER, P\_DEPARTMENT VARCHAR2, P\_HIREDATE DATE );

PROCEDURE update\_employee( P\_EMPLOYEEID NUMBER, P\_SALARY NUMBER );

FUNCTION get\_annualsalary( P\_EMPLOYEEID NUMBER ) RETURN NUMBER;

END

EmployeeManagement;

/ CREATE

OR REPLACE PACKAGE BODY EmployeeManagement AS PROCEDURE hire\_employee( P\_EMPLOYEEID NUMBER, P\_NAME VARCHAR2, P\_POSITION VARCHAR2, P\_SALARY NUMBER, P\_DEPARTMENT VARCHAR2, P\_HIREDATE DATE ) IS

BEGIN

   INSERT INTO

      EMPLOYEES (EMPLOYEEID, NAME, POSITION, SALARY, DEPARTMENT, HIREDATE)

   VALUES

      (

         P\_EMPLOYEEID, P\_NAME, P\_POSITION, P\_SALARY, P\_DEPARTMENT, P\_HIREDATE

      )

;

DBMS\_OUTPUT.PUT\_LINE('HIRED EMPLOYEE DETAILS ADDED SUCCESSFULLY!');

END

hire\_employee;

PROCEDURE update\_employee( P\_EMPLOYEEID NUMBER, P\_SALARY NUMBER ) IS

BEGIN

   UPDATE

      EMPLOYEES

   SET

      SALARY = P\_SALARY

   WHERE

      EMPLOYEEID = P\_EMPLOYEEID;

DBMS\_OUTPUT.PUT\_LINE('EMPLOYEE''S SALARY  UPDATED SUCCESSFULLY!');

END

update\_employee;

FUNCTION get\_annualsalary( P\_EMPLOYEEID NUMBER ) RETURN NUMBER IS v\_salary NUMBER;

BEGIN

   SELECT

      SALARY INTO v\_salary

   FROM

      EMPLOYEES

   WHERE

      EMPLOYEEID = P\_EMPLOYEEID;

RETURN v\_salary\*12;

EXCEPTION

WHEN

   NO\_DATA\_FOUND

THEN

   RETURN NULL;

WHEN

   OTHERS

THEN

   DBMS\_OUTPUT.PUT\_LINE('Error retrieving customer balance: ' || SQLERRM);

RETURN NULL;

END

get\_annualsalary;

END

EmployeeManagement;

/

DECLARE v\_annual\_salary NUMBER;

BEGIN

   EmployeeManagement.hire\_employee(10, 'Tejas', 'Developer', 60000, 'IT', SYSDATE);

EmployeeManagement.update\_employee(10, 70000);

v\_annual\_salary := EmployeeManagement.get\_annualsalary(101);

DBMS\_OUTPUT.PUT\_LINE(v\_annual\_salary);

END;

/

……………………………………………………………………………………………………………………

**3.**

CREATE

OR REPLACE PACKAGE AccountOperations AS PROCEDURE open\_account(p\_accountid NUMBER, p\_customerid NUMBER, p\_balance NUMBER);

PROCEDURE close\_account(p\_accountid NUMBER);

FUNCTION get\_total\_balance(p\_customerid NUMBER) RETURN NUMBER;

END

AccountOperations;

/ CREATE

OR REPLACE PACKAGE BODY AccountOperations AS PROCEDURE open\_account(p\_accountid NUMBER, p\_customerid NUMBER, p\_balance NUMBER) IS

BEGIN

   INSERT INTO

      ACCOUNTS (ACCOUNTID, CUSTOMERID, BALANCE)

   VALUES

      (

         p\_accountid, p\_customerid, p\_balance

      )

;

END

open\_account;

PROCEDURE close\_account(p\_accountid NUMBER) IS

BEGIN

   DELETE

   FROM

      ACCOUNTS

   WHERE

      ACCOUNTID = p\_accountid;

END

close\_account;

FUNCTION get\_total\_balance(p\_customerid NUMBER) RETURN NUMBER IS v\_total\_balance NUMBER;

BEGIN

   SELECT

      SUM(BALANCE) INTO v\_total\_balance

   FROM

      ACCOUNTS

   WHERE

      CUSTOMERID = p\_customerid;

RETURN NVL(v\_total\_balance, 0);

END

get\_total\_balance;

END

AccountOperations;

/

DECLARE v\_total\_balance NUMBER;

BEGIN

   AccountOperations.open\_account(101, 1, 5000);

AccountOperations.close\_account(101);

v\_total\_balance := AccountOperations.get\_total\_balance(1);

DBMS\_OUTPUT.PUT\_LINE(v\_total\_balance);

END

;

/