CONTROL STRUCTURES

QUESTION 1:

DECLARE

CURSOR cust\_cursor IS

SELECT c.CustomerID, l.LoanID, l.InterestRate

FROM Customers c

JOIN Loans l ON c.CustomerID = l.CustomerID

WHERE MONTHS\_BETWEEN(SYSDATE, c.DOB)/12 > 60;

v\_new\_rate NUMBER;

BEGIN

FOR cust\_rec IN cust\_cursor LOOP

v\_new\_rate := cust\_rec.InterestRate - 1; -- 1% discount

UPDATE Loans

SET InterestRate = v\_new\_rate

WHERE LoanID = cust\_rec.LoanID;

DBMS\_OUTPUT.PUT\_LINE('Applied discount for customer ' || cust\_rec.CustomerID ||

'. New rate: ' || v\_new\_rate || '%');

END LOOP;

COMMIT;

EXCEPTION

WHEN OTHERS THEN

DBMS\_OUTPUT.PUT\_LINE('Error applying discounts: ' || SQLERRM);

END;

/

-- Sample Output:

-- Applied discount for customer 101. New rate: 4.5%

-- Applied discount for customer 205. New rate: 3.75%

-- Error applying discounts: ORA-01722: invalid number

QUESTION 2:

BEGIN

FOR cust IN (SELECT CustomerID, Name, Balance FROM Customers)

LOOP

IF cust.Balance > 10000 THEN

DBMS\_OUTPUT.PUT\_LINE('Promoting ' || cust.Name || ' to VIP status');

END IF

END LOOP;

COMMIT;

END;

/

-- Sample Output:

-- Promoting John Smith to VIP status

-- Promoting Maria Garcia to VIP status

QUESTION 3:

DECLARE

CURSOR due\_loans IS

SELECT c.Name, l.LoanID, l.EndDate

FROM Customers c JOIN Loans l ON c.CustomerID = l.CustomerID

WHERE l.EndDate BETWEEN SYSDATE AND SYSDATE+30;

BEGIN

FOR loan IN due\_loans

LOOP

DBMS\_OUTPUT.PUT\_LINE('Reminder for ' || loan.Name ||

': Loan #' || loan.LoanID ||

' due on ' || TO\_CHAR(loan.EndDate, 'DD-MON-YYYY'));

END LOOP

END;

/

-- Sample Output:

-- Reminder for Robert Johnson: Loan #1025 due on 15-JUL-2023

-- Reminder for Sarah Williams: Loan #2041 due on 22-JUL-2023

ERROR HANDLING

QUESTION 1:

CREATE OR REPLACE PROCEDURE SafeTransferFunds(

p\_from\_acc NUMBER,

p\_to\_acc NUMBER,

p\_amount NUMBER

) AS

v\_from\_bal NUMBER;

BEGIN

SELECT Balance INTO v\_from\_bal

FROM Accounts

WHERE AccountID = p\_from\_acc FOR UPDATE;

IF v\_from\_bal < p\_amount THEN

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

END IF;

UPDATE Accounts SET Balance = Balance - p\_amount

WHERE AccountID = p\_from\_acc;

UPDATE Accounts SET Balance = Balance + p\_amount

WHERE AccountID = p\_to\_acc;

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('Transfer successful');

EXCEPTION

WHEN OTHERS THEN

ROLLBACK;

DBMS\_OUTPUT.PUT\_LINE('Transfer failed: ' || SQLERRM);

INSERT INTO TransferErrors VALUES(SYSDATE, p\_from\_acc, p\_to\_acc, p\_amount, SQLERRM);

END;

/

-- Test calls:

-- EXEC SafeTransferFunds(101, 202, 500);

-- EXEC SafeTransferFunds(101, 202, 5000);

-- Sample Output:

-- Transfer successful

-- Transfer failed: ORA-20001: Insufficient funds

QUESTION 2:

CREATE OR REPLACE PROCEDURE UpdateSalary(

p\_emp\_id NUMBER,

p\_percent NUMBER

) AS

v\_count NUMBER;

BEGIN

SELECT COUNT(\*) INTO v\_count FROM Employees

WHERE EmployeeID = p\_emp\_id;

IF v\_count = 0 THEN

RAISE\_APPLICATION\_ERROR(-20002, 'Employee not found');

END IF;

UPDATE Employees

SET Salary = Salary \* (1 + p\_percent/100)

WHERE EmployeeID = p\_emp\_id;

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('Salary updated successfully');

EXCEPTION

WHEN OTHERS THEN

DBMS\_OUTPUT.PUT\_LINE('Error updating salary: ' || SQLERRM);

ROLLBACK;

END;

/

-- Sample Output:

-- Salary updated successfully

-- Error updating salary: ORA-20002: Employee not found

STORED PROCEDURES

QUESTION 3:

CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest AS

CURSOR savings\_accts IS

SELECT AccountID, Balance

FROM Accounts

WHERE AccountType = 'Savings'

FOR UPDATE;

v\_interest\_rate CONSTANT NUMBER := 0.01; -- 1%

BEGIN

FOR acct IN savings\_accts LOOP

UPDATE Accounts

SET Balance = Balance \* (1 + v\_interest\_rate)

WHERE AccountID = acct.AccountID;

DBMS\_OUTPUT.PUT\_LINE('Applied interest to account ' || acct.AccountID ||

'. New balance: ' || acct.Balance \* (1 + v\_interest\_rate));

END LOOP;

COMMIT;

END;

/

-- Sample Output:

-- Applied interest to account 101. New balance: 5050.25

-- Applied interest to account 205. New balance: 10201.00

FUNCTIONS

QUESTION 1:

CREATE OR REPLACE FUNCTION CalculateAge(p\_dob DATE)

RETURN NUMBER IS

v\_age NUMBER;

BEGIN

v\_age := FLOOR(MONTHS\_BETWEEN(SYSDATE, p\_dob)/12);

RETURN v\_age;

EXCEPTION

WHEN OTHERS THEN

RETURN NULL;

END;

/

-- Test:

-- SELECT Name, CalculateAge(DOB) AS Age FROM Customers;

/\* Sample Output:

NAME AGE

-------- ---

John Doe 38

Jane Smith 33

(Note: Age calculated based on current date)

\*/

QUESTION 2:

CREATE OR REPLACE FUNCTION CalculateMonthlyInstallment(

p\_amount NUMBER,

p\_rate NUMBER,

p\_years NUMBER

) RETURN NUMBER IS

v\_monthly\_rate NUMBER := p\_rate/12/100;

v\_months NUMBER := p\_years\*12;

BEGIN

RETURN ROUND((p\_amount\*v\_monthly\_rate\*POWER(1+v\_monthly\_rate,v\_months))

/(POWER(1+v\_monthly\_rate,v\_months)-1), 2);

END;

/

-- Test:

-- SELECT CalculateMonthlyInstallment(10000, 5, 5) FROM dual;

/\* Sample Output:

CALCULATEMONTHLYINSTALLMENT(10000,5,5)

---------------------------------------

188.71

\*/

TRIGGERS

CREATE OR REPLACE TRIGGER UpdateCustomerLastModified

BEFORE UPDATE ON Customers

FOR EACH ROW

BEGIN

:NEW.LastModified := SYSDATE;

DBMS\_OUTPUT.PUT\_LINE('Updated last modified for customer ' || :OLD.CustomerID);

EXCEPTION

WHEN OTHERS THEN

NULL;

END;

/

/\* Sample Output (when customer record updates):

Updated last modified for customer 101

\*/

QUESTION 2:

CREATE OR REPLACE TRIGGER LogTransaction

AFTER INSERT ON Transactions

FOR EACH ROW

DECLARE

v\_customer\_id NUMBER;

BEGIN

SELECT CustomerID INTO v\_customer\_id FROM Accounts

WHERE AccountID = :NEW.AccountID;

INSERT INTO AuditLog(LogDate, CustomerID, TransactionID, Action)

VALUES(SYSDATE, v\_customer\_id, :NEW.TransactionID, 'New ' || :NEW.TransactionType);

DBMS\_OUTPUT.PUT\_LINE('Logged transaction ' || :NEW.TransactionID);

END;

/

/\* Sample Output:

Logged transaction 1025

Logged transaction 1026

\*/

CURSORS

DECLARE

CURSOR stmt\_cursor IS

SELECT c.CustomerID, c.Name, t.TransactionDate, t.Amount, 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, 'MONTH') AND LAST\_DAY(SYSDATE);

v\_current\_cust NUMBER := 0;

BEGIN

FOR stmt\_rec IN stmt\_cursor LOOP

IF stmt\_rec.CustomerID != v\_current\_cust THEN

DBMS\_OUTPUT.PUT\_LINE(CHR(10) || 'Statement for: ' || stmt\_rec.Name);

v\_current\_cust := stmt\_rec.CustomerID;

END IF;

DBMS\_OUTPUT.PUT\_LINE(TO\_CHAR(stmt\_rec.TransactionDate, 'DD-MON') || ' ' ||

RPAD(stmt\_rec.TransactionType, 10) || ' ' ||

TO\_CHAR(stmt\_rec.Amount, '99990.99'));

END LOOP;

END;

/

/\* Sample Output:

Statement for: John Doe

15-JUL Deposit 200.00

20-JUL Withdrawal 50.00

Statement for: Jane Smith

18-JUL Deposit 500.00

\*/

PACKAGES

CREATE OR REPLACE PACKAGE CustomerManagement AS

PROCEDURE AddCustomer(

p\_id NUMBER,

p\_name VARCHAR2,

p\_dob DATE,

p\_initial\_bal NUMBER

);

PROCEDURE UpdateCustomer(

p\_id NUMBER,

p\_new\_name VARCHAR2,

p\_new\_dob DATE

);

FUNCTION GetCustomerBalance(p\_id NUMBER) RETURN NUMBER;

END CustomerManagement;

/

CREATE OR REPLACE PACKAGE BODY CustomerManagement AS

PROCEDURE AddCustomer(

p\_id NUMBER,

p\_name VARCHAR2,

p\_dob DATE,

p\_initial\_bal NUMBER

) IS

BEGIN

INSERT INTO Customers VALUES(p\_id, p\_name, p\_dob, p\_initial\_bal, SYSDATE);

COMMIT;

EXCEPTION

WHEN OTHERS THEN

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

END AddCustomer;

FUNCTION GetCustomerBalance(p\_id NUMBER) RETURN NUMBER IS

v\_balance NUMBER;

BEGIN

SELECT Balance INTO v\_balance FROM Customers WHERE CustomerID = p\_id;

RETURN v\_balance;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

RETURN -1;

END GetCustomerBalance;

END CustomerManagement;

/

-- Test:

-- BEGIN

-- CustomerManagement.AddCustomer(3, 'Bob Wilson', TO\_DATE('1988-11-25'), 2000);

-- DBMS\_OUTPUT.PUT\_LINE('Balance: ' || CustomerManagement.GetCustomerBalance(3));

-- END;

/

/\* Sample Output:

Balance: 2000

\*/