1. Control Structure:

Scenario 1:

DECLARE

CURSOR customer\_cursor IS

SELECT customer\_id, age, loan\_interest\_rate

FROM customers

WHERE age > 60;

v\_loan\_interest\_rate customers.loan\_interest\_rate%TYPE;

BEGIN

FOR rec IN customer\_cursor LOOP

v\_loan\_interest\_rate: = rec.loan\_interest\_rate - 0.01;

UPDATE customers

SET loan\_interest\_rate = v\_loan\_interest\_rate

WHERE customer\_id = rec.customer\_id;

END LOOP;

COMMIT;

END;

/

Scenario 2:

DECLARE

CURSOR customer\_cursor IS

SELECT customer\_id, balance

FROM customers;

BEGIN

FOR rec IN customer\_cursor LOOP

IF rec.balance > 10000 THEN

UPDATE customers

SET IsVIP = TRUE

WHERE customer\_id = rec.customer\_id;

ELSE

UPDATE customers

SET IsVIP = FALSE

WHERE customer\_id = rec.customer\_id;

END IF;

END LOOP;

COMMIT;

END;

/

Scenario 3:

DECLARE

CURSOR loan\_cursor IS

SELECT customer\_id, loan\_due\_date

FROM loans

WHERE loan\_due\_date BETWEEN SYSDATE AND SYSDATE + 30;

BEGIN

FOR rec IN loan\_cursor LOOP

DBMS\_OUTPUT.PUT\_LINE ('Reminder: Loan for customer ' || rec.customer\_id ||

' is due on ' || rec.loan\_due\_date ||

'. Please ensure payment is made in time.');

END LOOP;

END;

/

Exercise 2: Error Handling

Scenario 1:

CREATE OR REPLACE PROCEDURE SafeTransferFunds (

p\_from\_account IN NUMBER,

p\_to\_account IN NUMBER,

p\_amount IN NUMBER

) IS

insufficient\_funds EXCEPTION;

v\_from\_balance NUMBER;

BEGIN

SELECT balance INTO v\_from\_balance

FROM accounts

WHERE account\_id = p\_from\_account;

IF v\_from\_balance < p\_amount THEN

RAISE insufficient\_funds;

END IF;

SAVEPOINT before\_transfer;

UPDATE accounts

SET balance = balance - p\_amount

WHERE account\_id = p\_from\_account;

UPDATE accounts

SET balance = balance + p\_amount

WHERE account\_id = p\_to\_account;

COMMIT;

EXCEPTION

WHEN insufficient\_funds THEN

DBMS\_OUTPUT.PUT\_LINE ('Error: Insufficient funds for account ' || p\_from\_account);

ROLLBACK TO before\_transfer;

WHEN OTHERS THEN

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

ROLLBACK TO before\_transfer;

END SafeTransferFunds;

/

Scenario 2:

CREATE OR REPLACE PROCEDURE UpdateSalary (

p\_employee\_id IN NUMBER,

p\_increase\_percentage IN NUMBER

) IS

employee\_not\_found EXCEPTION;

BEGIN

UPDATE employees

SET salary = salary \* (1 + p\_increase\_percentage / 100)

WHERE employee\_id = p\_employee\_id;

IF SQL%ROWCOUNT = 0 THEN

RAISE employee\_not\_found;

END IF;

COMMIT;

EXCEPTION

WHEN employee\_not\_found THE

DBMS\_OUTPUT.PUT\_LINE ('Error: Employee with ID ' || p\_employee\_id || ' does not exist.');

WHEN OTHERS THEN

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

ROLLBACK;

END UpdateSalary;

/

Scenario 3:

CREATE OR REPLACE PROCEDURE AddNewCustomer (

p\_customer\_id IN NUMBER,

p\_name IN VARCHAR2,

p\_age IN NUMBER,

p\_balance IN NUMBER

) IS

duplicate\_customer EXCEPTION;

BEGIN

INSERT INTO customers (customer\_id, name, age, balance)

VALUES (p\_customer\_id, p\_name, p\_age, p\_balance);

COMMIT;

EXCEPTION

WHEN DUP\_VAL\_ON\_INDEX THEN

DBMS\_OUTPUT.PUT\_LINE ('Error: Customer with ID ' || p\_customer\_id || ' already exists.');

WHEN OTHERS THEN

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

ROLLBACK;

END AddNewCustomer;

/

Exercise 3: Stored Procedures

Scenario 1:

CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest IS

BEGIN

UPDATE accounts

SET balance = balance \* 1.01

WHERE account\_type = 'Savings';

COMMIT;

DBMS\_OUTPUT.PUT\_LINE ('Monthly interest processed for all savings accounts.');

END ProcessMonthlyInterest;

/

Scenario 2:

CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus (

p\_department\_id IN NUMBER,

p\_bonus\_percentage IN NUMBER

) IS

BEGIN

UPDATE employees

SET salary = salary \* (1 + p\_bonus\_percentage / 100)

WHERE department\_id = p\_department\_id;

COMMIT;

DBMS\_OUTPUT.PUT\_LINE ('Employee bonuses updated for department ' || p\_department\_id || '.');

END UpdateEmployeeBonus;

/

Scenario 3:

CREATE OR REPLACE PROCEDURE TransferFunds (

p\_from\_account IN NUMBER,

p\_to\_account IN NUMBER,

p\_amount IN NUMBER

) IS

insufficient\_funds EXCEPTION;

v\_from\_balance NUMBER;

BEGIN

SELECT balance INTO v\_from\_balance

FROM accounts

WHERE account\_id = p\_from\_account;

IF v\_from\_balance < p\_amount THEN

RAISE insufficient\_funds;

END IF;

SAVEPOINT before\_transfer;

UPDATE accounts

SET balance = balance - p\_amount

WHERE account\_id = p\_from\_account;

UPDATE accounts

SET balance = balance + p\_amount

WHERE account\_id = p\_to\_account;

COMMIT;

DBMS\_OUTPUT.PUT\_LINE ('Transferred ' || p\_amount || ' from account ' || p\_from\_account || ' to account ' || p\_to\_account || '.');

EXCEPTION

WHEN insufficient\_funds THEN

-- Log the error message for insufficient funds

DBMS\_OUTPUT.PUT\_LINE ('Error: Insufficient funds for account ' || p\_from\_account);

ROLLBACK TO before\_transfer;

WHEN OTHERS THEN

-- Handle any other unexpected errors

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

ROLLBACK TO before\_transfer;

END TransferFunds;

/

Exercise 4: Functions

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;

END CalculateAge;

/

Scenario 2:

CREATE OR REPLACE FUNCTION CalculateMonthlyInstallment (

p\_loan\_amount NUMBER,

p\_annual\_interest\_rate NUMBER,

p\_loan\_duration\_years NUMBER

) RETURN NUMBER IS

v\_monthly\_interest\_rate NUMBER;

v\_number\_of\_payments NUMBER;

v\_monthly\_installment NUMBER;

BEGIN

v\_monthly\_interest\_rate: = p\_annual\_interest\_rate / 1200;

v\_number\_of\_payments: = p\_loan\_duration\_years \* 12;

IF v\_monthly\_interest\_rate > 0 THEN

v\_monthly\_installment: = (p\_loan\_amount \* v\_monthly\_interest\_rate) /

(1 - POWER (1 + v\_monthly\_interest\_rate, -v\_number\_of\_payments));

ELSE

v\_monthly\_installment: = p\_loan\_amount / v\_number\_of\_payments;

END IF;

RETURN v\_monthly\_installment;

END CalculateMonthlyInstallment;

/

Scenario 3:

CREATE OR REPLACE FUNCTION HasSufficientBalance (

p\_account\_id NUMBER,

p\_amount NUMBER

) RETURN BOOLEAN IS

v\_balance NUMBER;

BEGIN

SELECT balance INTO v\_balance

FROM accounts

WHERE account\_id = p\_account\_id;

RETURN v\_balance >= p\_amount;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

RETURN FALSE;

WHEN OTHERS THEN

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

RETURN FALSE;

END HasSufficientBalance;

/

Exercise 5: Triggers

Scenario 1:

CREATE OR REPLACE TRIGGER UpdateCustomerLastModified

BEFORE UPDATE ON customers

FOR EACH ROW

BEGIN

:NEW.LastModified := SYSDATE;

END UpdateCustomerLastModified;

/

Scenario 2:

CREATE OR REPLACE TRIGGER LogTransaction

AFTER INSERT ON transactions

FOR EACH ROW

BEGIN

INSERT INTO AuditLog (

log\_id,

transaction\_id,

action\_type,

action\_timestamp,

action\_details

) VALUES (

AuditLog\_seq.NEXTVAL,

:NEW.transaction\_id,

'INSERT',

SYSDATE,

'Transaction inserted: Amount ' ||: NEW.amount || ', Account ' ||: NEW.account\_id

);

END LogTransaction;

/

Scenario 3:

CREATE OR REPLACE TRIGGER CheckTransactionRules

BEFORE INSERT ON transactions

FOR EACH ROW

DECLARE

v\_balance NUMBER;

BEGIN

IF:NEW.transaction\_type = 'Withdrawal' THEN

SELECT balance INTO v\_balance

FROM accounts

WHERE account\_id =: NEW.account\_id;

IF:NEW.amount > v\_balance THEN

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

END IF;

ELSIF:NEW.transaction\_type = 'Deposit' THEN

IF:NEW.amount <= 0 THEN

RAISE\_APPLICATION\_ERROR (-20002, 'Deposit amount must be positive.');

END IF;

END IF;

END CheckTransactionRules;

/

Exercise 6: Cursors

Scenario 1:

DECLARE

CURSOR transaction\_cursor IS

SELECT t.transaction\_id, t.customer\_id, t.transaction\_date, t.amount, c.name

FROM transactions t

JOIN customers c ON t.customer\_id = c.customer\_id

WHERE EXTRACT (MONTH FROM t.transaction\_date) = EXTRACT(MONTH FROM SYSDATE)

AND EXTRACT (YEAR FROM t.transaction\_date) = EXTRACT(YEAR FROM SYSDATE);

v\_transaction\_record transaction\_cursor%ROWTYPE;

BEGIN

OPEN transaction\_cursor;

LOOP

FETCH transaction\_cursor INTO v\_transaction\_record;

EXIT WHEN transaction\_cursor%NOTFOUND;

DBMS\_OUTPUT.PUT\_LINE ('Customer: ' || v\_transaction\_record.name);

DBMS\_OUTPUT.PUT\_LINE ('Transaction ID: ' || v\_transaction\_record.transaction\_id);

DBMS\_OUTPUT.PUT\_LINE ('Date: ' || TO\_CHAR (v\_transaction\_record.transaction\_date, 'YYYY-MM-DD'));

DBMS\_OUTPUT.PUT\_LINE ('Amount: ' || v\_transaction\_record.amount);

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

END LOOP;

CLOSE transaction\_cursor;

END;

/

Scenario 2:

DECLARE

v\_annual\_fee NUMBER: = 50; -- Example fee amount

CURSOR account\_cursor IS

SELECT account\_id, balance

FROM accounts;

v\_account\_record account\_cursor%ROWTYPE;

BEGIN

OPEN account\_cursor;

LOOP

FETCH account\_cursor INTO v\_account\_record;

EXIT WHEN account\_cursor%NOTFOUND;

UPDATE accounts

SET balance = balance - v\_annual\_fee

WHERE account\_id = v\_account\_record.account\_id;

DBMS\_OUTPUT.PUT\_LINE ('Account ID: ' || v\_account\_record.account\_id);

DBMS\_OUTPUT.PUT\_LINE ('New Balance: ' || (v\_account\_record.balance - v\_annual\_fee));

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

END LOOP;

CLOSE account\_cursor;

COMMIT;

END;

/

Scenario 3:

DECLARE

v\_new\_interest\_rate NUMBER: = 0.05; -- Example new interest rate (5%)

CURSOR loan\_cursor IS

SELECT loan\_id, interest\_rate

FROM loans;

v\_loan\_record loan\_cursor%ROWTYPE;

BEGIN

OPEN loan\_cursor;

LOOP

FETCH loan\_cursor INTO v\_loan\_record;

EXIT WHEN loan\_cursor%NOTFOUND;

UPDATE loans

SET interest\_rate = v\_new\_interest\_rate

WHERE loan\_id = v\_loan\_record.loan\_id;

DBMS\_OUTPUT.PUT\_LINE ('Loan ID: ' || v\_loan\_record.loan\_id);

DBMS\_OUTPUT.PUT\_LINE ('New Interest Rate: ' || v\_new\_interest\_rate);

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

END LOOP;

CLOSE loan\_cursor;

COMMIT;

END;

/

Exercise 7: Packages

Scenario 1:

CREATE OR REPLACE PACKAGE CustomerManagement AS

PROCEDURE AddNewCustomer (

p\_customer\_id IN NUMBER,

p\_name IN VARCHAR2,

p\_dob IN DATE,

p\_balance IN NUMBER

);

PROCEDURE UpdateCustomerDetails (

p\_customer\_id IN NUMBER,

p\_name IN VARCHAR2,

p\_dob IN DATE

);

FUNCTION GetCustomerBalance (

p\_customer\_id IN NUMBER

) RETURN NUMBER;

END CustomerManagement;

/

CREATE OR REPLACE PACKAGE BODY CustomerManagement AS

PROCEDURE AddNewCustomer (

p\_customer\_id IN NUMBER,

p\_name IN VARCHAR2,

p\_dob IN DATE,

p\_balance IN NUMBER

) IS

BEGIN

INSERT INTO customers (customer\_id, name, dob, balance)

VALUES (p\_customer\_id, p\_name, p\_dob, p\_balance);

COMMIT;

END AddNewCustomer;

PROCEDURE UpdateCustomerDetails (

p\_customer\_id IN NUMBER,

p\_name IN VARCHAR2,

p\_dob IN DATE

) IS

BEGIN

UPDATE customers

SET name = p\_name,

dob = p\_dob

WHERE customer\_id = p\_customer\_id;

COMMIT;

END UpdateCustomerDetails;

FUNCTION GetCustomerBalance (

p\_customer\_id IN NUMBER

) RETURN NUMBER IS

v\_balance NUMBER;

BEGIN

SELECT balance INTO v\_balance

FROM customers

WHERE customer\_id = p\_customer\_id;

RETURN v\_balance;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

RETURN 0;

END GetCustomerBalance;

END CustomerManagement;

/

Scenario 2:

CREATE OR REPLACE PACKAGE EmployeeManagement AS

PROCEDURE HireEmployee (

p\_employee\_id IN NUMBER,

p\_name IN VARCHAR2,

p\_salary IN NUMBER,

p\_department\_id IN NUMBER

);

PROCEDURE UpdateEmployeeDetails (

p\_employee\_id IN NUMBER,

p\_name IN VARCHAR2,

p\_salary IN NUMBER

);

FUNCTION CalculateAnnualSalary (

p\_employee\_id IN NUMBER

) RETURN NUMBER;

END EmployeeManagement;

/

CREATE OR REPLACE PACKAGE BODY EmployeeManagement AS

PROCEDURE HireEmployee (

p\_employee\_id IN NUMBER,

p\_name IN VARCHAR2,

p\_salary IN NUMBER,

p\_department\_id IN NUMBER

) IS

BEGIN

INSERT INTO employees (employee\_id, name, salary, department\_id)

VALUES (p\_employee\_id, p\_name, p\_salary, p\_department\_id);

COMMIT;

END HireEmployee;

PROCEDURE UpdateEmployeeDetails (

p\_employee\_id IN NUMBER,

p\_name IN VARCHAR2,

p\_salary IN NUMBER

) IS

BEGIN

UPDATE employees

SET name = p\_name,

salary = p\_salary

WHERE employee\_id = p\_employee\_id;

COMMIT;

END UpdateEmployeeDetails;

FUNCTION CalculateAnnualSalary (

p\_employee\_id IN NUMBER

) RETURN NUMBER IS

v\_salary NUMBER;

BEGIN

SELECT salary INTO v\_salary

FROM employees

WHERE employee\_id = p\_employee\_id;

RETURN v\_salary \* 12;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

RETURN 0;

END CalculateAnnualSalary;

END EmployeeManagement;

/

Scenario 3:

CREATE OR REPLACE PACKAGE AccountOperations AS

PROCEDURE OpenNewAccount (

p\_account\_id IN NUMBER,

p\_customer\_id IN NUMBER,

p\_initial\_balance IN NUMBER

);

PROCEDURE CloseAccount (

p\_account\_id IN NUMBER

);

FUNCTION GetTotalBalance (

p\_customer\_id IN NUMBER

) RETURN NUMBER;

END AccountOperations;

/

CREATE OR REPLACE PACKAGE BODY AccountOperations AS

PROCEDURE OpenNewAccount (

p\_account\_id IN NUMBER,

p\_customer\_id IN NUMBER,

p\_initial\_balance IN NUMBER

) IS

BEGIN

INSERT INTO accounts (account\_id, customer\_id, balance)

VALUES (p\_account\_id, p\_customer\_id, p\_initial\_balance);

COMMIT;

END OpenNewAccount;

PROCEDURE CloseAccount (

p\_account\_id IN NUMBER

) IS

BEGIN

DELETE FROM accounts

WHERE account\_id = p\_account\_id;

COMMIT;

END CloseAccount;

FUNCTION GetTotalBalance (

p\_customer\_id IN NUMBER

) RETURN NUMBER IS

v\_total\_balance NUMBER;

BEGIN

SELECT SUM (balance) INTO v\_total\_balance

FROM accounts

WHERE customer\_id = p\_customer\_id;

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

END GetTotalBalance;

END AccountOperations;

/