PL/SQL

# Exercise-1

### Scenario 1: Apply a 1% Discount to Loan Interest Rates for Customers Above 60 Years Old

DECLARE

CURSOR cur\_customers IS

SELECT customer\_id, age, loan\_interest\_rate FROM customers;

v\_customer\_id customers.customer\_id%TYPE; v\_age customers.age%TYPE;

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

OPEN cur\_customers;

LOOP

FETCH cur\_customers INTO v\_customer\_id, v\_age, v\_loan\_interest\_rate; EXIT WHEN cur\_customers%NOTFOUND;

IF v\_age > 60 THEN UPDATE customers

SET loan\_interest\_rate = v\_loan\_interest\_rate \* 0.99 WHERE customer\_id = v\_customer\_id;

END IF; END LOOP;

CLOSE cur\_customers; COMMIT;

END;

/

### Scenario 2: Set IsVIP Flag for Customers with Balance Over $10,000

DECLARE

CURSOR cur\_customers IS SELECT customer\_id, balance FROM customers;

v\_customer\_id customers.customer\_id%TYPE; v\_balance customers.balance%TYPE;

BEGIN

OPEN cur\_customers;

LOOP

FETCH cur\_customers INTO v\_customer\_id, v\_balance; EXIT WHEN cur\_customers%NOTFOUND;

IF v\_balance > 10000 THEN UPDATE customers

SET IsVIP = TRUE

WHERE customer\_id = v\_customer\_id; END IF;

END LOOP;

CLOSE cur\_customers; COMMIT;

END;

/

### Scenario 3: Send Reminders for Loans Due in the Next 30 Days

DECLARE

CURSOR cur\_loans IS

SELECT l.customer\_id, l.due\_date, c.name FROM loans l

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

WHERE l.due\_date BETWEEN SYSDATE AND SYSDATE + 30;

v\_customer\_id loans.customer\_id%TYPE; v\_due\_date loans.due\_date%TYPE; v\_name customers.name%TYPE;

BEGIN

OPEN cur\_loans;

LOOP

FETCH cur\_loans INTO v\_customer\_id, v\_due\_date, v\_name; EXIT WHEN cur\_loans%NOTFOUND;

DBMS\_OUTPUT.PUT\_LINE('Reminder: ' || v\_name || ' (Customer ID: ' || v\_customer\_id || ') has a loan due on ' || TO\_CHAR(v\_due\_date, 'DD-MON-YYYY'));

END LOOP;

CLOSE cur\_loans; END;

/

# Exercise-2

### Scenario 1: SafeTransferFunds Stored Procedure

CREATE OR REPLACE PROCEDURE SafeTransferFunds( p\_from\_account\_id IN accounts.account\_id%TYPE, p\_to\_account\_id IN accounts.account\_id%TYPE, p\_amount IN accounts.balance%TYPE

) IS

e\_insufficient\_funds EXCEPTION; v\_from\_balance accounts.balance%TYPE;

BEGIN

-- Check if the source account has sufficient funds

SELECT balance INTO v\_from\_balance FROM accounts WHERE account\_id = p\_from\_account\_id;

IF v\_from\_balance < p\_amount THEN RAISE e\_insufficient\_funds;

END IF;

-- Transfer funds UPDATE accounts

SET balance = balance - p\_amount WHERE account\_id = p\_from\_account\_id;

UPDATE accounts

SET balance = balance + p\_amount WHERE account\_id = p\_to\_account\_id;

-- Commit transaction COMMIT;

EXCEPTION

WHEN e\_insufficient\_funds THEN ROLLBACK;

DBMS\_OUTPUT.PUT\_LINE('Error: Insufficient funds in account ' || p\_from\_account\_id); WHEN OTHERS THEN

ROLLBACK;

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

/

### Scenario 2: UpdateSalary Stored Procedure

CREATE OR REPLACE PROCEDURE UpdateSalary(

p\_employee\_id IN employees.employee\_id%TYPE, p\_percentage IN NUMBER

) IS

e\_employee\_not\_found EXCEPTION; v\_current\_salary employees.salary%TYPE;

BEGIN

-- Check if employee exists

SELECT salary INTO v\_current\_salary FROM employees WHERE employee\_id = p\_employee\_id;

-- Update salary UPDATE employees

SET salary = salary \* (1 + p\_percentage / 100) WHERE employee\_id = p\_employee\_id;

-- Commit transaction COMMIT;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

RAISE e\_employee\_not\_found; WHEN e\_employee\_not\_found THEN

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

WHEN OTHERS THEN DBMS\_OUTPUT.PUT\_LINE('Error: ' || SQLERRM);

END;

/

### Scenario 3: AddNewCustomer Stored Procedure

CREATE OR REPLACE PROCEDURE AddNewCustomer(

p\_customer\_id IN customers.customer\_id%TYPE, p\_name IN customers.name%TYPE,

p\_age IN customers.age%TYPE, p\_balance IN customers.balance%TYPE

) IS

e\_customer\_exists EXCEPTION; BEGIN

-- Insert new customer

INSERT INTO customers (customer\_id, name, age, balance) VALUES (p\_customer\_id, p\_name, p\_age, p\_balance);

-- Commit transaction COMMIT;

EXCEPTION

WHEN DUP\_VAL\_ON\_INDEX THEN ROLLBACK;

RAISE e\_customer\_exists; WHEN e\_customer\_exists THEN

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

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

/

# Exercise-3

### Scenario 1: ProcessMonthlyInterest Stored Procedure

CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest IS v\_account\_id accounts.account\_id%TYPE;

v\_balance accounts.balance%TYPE; v\_interest\_rate CONSTANT NUMBER := 0.01; CURSOR cur\_savings\_accounts IS

SELECT account\_id, balance FROM accounts

WHERE account\_type = 'SAVINGS'; BEGIN

OPEN cur\_savings\_accounts;

LOOP

FETCH cur\_savings\_accounts INTO v\_account\_id, v\_balance; EXIT WHEN cur\_savings\_accounts%NOTFOUND;

-- Calculate and update the balance with interest

UPDATE accounts

SET balance = balance + (v\_balance \* v\_interest\_rate) WHERE account\_id = v\_account\_id;

END LOOP;

CLOSE cur\_savings\_accounts;

-- Commit the changes COMMIT;

END;

/

### Scenario 2: UpdateEmployeeBonus Stored Procedure

CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus( p\_department\_id IN employees.department\_id%TYPE, p\_bonus\_percentage IN NUMBER

) IS BEGIN

-- Update salary with the bonus percentage UPDATE employees

SET salary = salary \* (1 + p\_bonus\_percentage / 100) WHERE department\_id = p\_department\_id;

-- Commit the changes COMMIT;

END;

/

### Scenario 3: TransferFunds Stored Procedure

CREATE OR REPLACE PROCEDURE TransferFunds(

p\_from\_account\_id IN accounts.account\_id%TYPE, p\_to\_account\_id IN accounts.account\_id%TYPE, p\_amount IN accounts.balance%TYPE

) IS

v\_from\_balance accounts.balance%TYPE; BEGIN

-- Check if the source account has sufficient balance

SELECT balance INTO v\_from\_balance FROM accounts WHERE account\_id = p\_from\_account\_id;

IF v\_from\_balance < p\_amount THEN

RAISE\_APPLICATION\_ERROR(-20001, 'Insufficient balance in source account.'); END IF;

-- Deduct the amount from the source account UPDATE accounts

SET balance = balance - p\_amount WHERE account\_id = p\_from\_account\_id;

-- Add the amount to the destination account UPDATE accounts

SET balance = balance + p\_amount

WHERE account\_id = p\_to\_account\_id;

-- Commit the transaction COMMIT;

EXCEPTION

WHEN OTHERS THEN

-- Rollback in case of any error ROLLBACK;

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

/

**Scenario 1: CalculateAge Function**

# Exercise-4

CREATE OR REPLACE FUNCTION CalculateAge(p\_date\_of\_birth DATE) RETURN NUMBER IS

v\_age NUMBER;

BEGIN

SELECT TRUNC(MONTHS\_BETWEEN(SYSDATE, p\_date\_of\_birth) / 12) INTO v\_age FROM DUAL;

RETURN v\_age; END;

/

### Scenario 2: CalculateMonthlyInstallment Function

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 / 12 / 100; v\_number\_of\_payments := p\_loan\_duration\_years \* 12;

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

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

RETURN v\_monthly\_installment; END;

/

### Scenario 3: HasSufficientBalance Function

CREATE OR REPLACE FUNCTION HasSufficientBalance( p\_account\_id IN accounts.account\_id%TYPE,

p\_amount IN accounts.balance%TYPE

) RETURN BOOLEAN IS

v\_balance accounts.balance%TYPE; BEGIN

-- Fetch the current balance of the account

SELECT balance INTO v\_balance FROM accounts WHERE account\_id = p\_account\_id;

-- Check if the balance is sufficient IF v\_balance >= p\_amount THEN

RETURN TRUE; ELSE

RETURN FALSE; END IF;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN RETURN FALSE;

END;

/

# Exercise-5

### Scenario 1: UpdateCustomerLastModified Trigger

CREATE OR REPLACE TRIGGER UpdateCustomerLastModified BEFORE UPDATE ON customers

FOR EACH ROW BEGIN

:NEW.LastModified := SYSDATE; END;

/

### Scenario 2: LogTransaction Trigger

CREATE OR REPLACE TRIGGER LogTransaction AFTER INSERT ON transactions

FOR EACH ROW BEGIN

INSERT INTO AuditLog (transaction\_id, account\_id, transaction\_type, amount, transaction\_date, log\_date)

VALUES

(:NEW.transaction\_id, :NEW.account\_id, :NEW.transaction\_type, :NEW.amount, :NEW.transaction

\_date, SYSDATE); END;

/

### Scenario 3: CheckTransactionRules Trigger

CREATE OR REPLACE TRIGGER CheckTransactionRules BEFORE INSERT ON transactions

FOR EACH ROW DECLARE

v\_balance accounts.balance%TYPE; BEGIN

-- Check if the transaction is a withdrawal

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

-- Fetch the current balance of the account

SELECT balance INTO v\_balance FROM accounts WHERE account\_id = :NEW.account\_id;

-- Ensure the withdrawal does not exceed the balance IF :NEW.amount > v\_balance THEN

RAISE\_APPLICATION\_ERROR(-20001, 'Insufficient balance for withdrawal.'); END IF;

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

-- Ensure the deposit amount is positive IF :NEW.amount <= 0 THEN

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

END IF; END;

/

# Exercise-6

### Scenario 1: GenerateMonthlyStatements

DECLARE

CURSOR cur\_customers IS SELECT DISTINCT customer\_id FROM transactions

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

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

v\_customer\_id transactions.customer\_id%TYPE; BEGIN

OPEN cur\_customers;

LOOP

FETCH cur\_customers INTO v\_customer\_id; EXIT WHEN cur\_customers%NOTFOUND;

DBMS\_OUTPUT.PUT\_LINE('Monthly Statement for Customer ID: ' || v\_customer\_id); FOR cur\_transactions IN (

SELECT transaction\_date, transaction\_type, amount

FROM transactions

WHERE customer\_id = v\_customer\_id

AND EXTRACT(MONTH FROM transaction\_date) = EXTRACT(MONTH FROM SYSDATE)

AND EXTRACT(YEAR FROM transaction\_date) = EXTRACT(YEAR FROM SYSDATE)

) LOOP

DBMS\_OUTPUT.PUT\_LINE(

'Date: ' || TO\_CHAR(cur\_transactions.transaction\_date, 'DD-MON-YYYY') || ', Type: ' || cur\_transactions.transaction\_type ||

', Amount: ' || cur\_transactions.amount

);

END LOOP;

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

END LOOP;

CLOSE cur\_customers; END;

/

### Scenario 2: ApplyAnnualFee

DECLARE

CURSOR cur\_accounts IS SELECT account\_id, balance FROM accounts;

v\_account\_id accounts.account\_id%TYPE; v\_balance accounts.balance%TYPE;

v\_annual\_fee CONSTANT NUMBER := 50; -- Define the annual fee amount here BEGIN

OPEN cur\_accounts;

LOOP

FETCH cur\_accounts INTO v\_account\_id, v\_balance; EXIT WHEN cur\_accounts%NOTFOUND;

-- Deduct the annual fee from the account balance UPDATE accounts

SET balance = balance - v\_annual\_fee WHERE account\_id = v\_account\_id;

DBMS\_OUTPUT.PUT\_LINE('Applied annual fee to Account ID: ' || v\_account\_id); END LOOP;

CLOSE cur\_accounts;

-- Commit the changes COMMIT;

END;

/

### Scenario 3: UpdateLoanInterestRates

DECLARE

CURSOR cur\_loans IS

SELECT loan\_id, interest\_rate FROM loans;

v\_loan\_id loans.loan\_id%TYPE; v\_interest\_rate loans.interest\_rate%TYPE; v\_new\_interest\_rate loans.interest\_rate%TYPE;

BEGIN

OPEN cur\_loans;

LOOP

FETCH cur\_loans INTO v\_loan\_id, v\_interest\_rate; EXIT WHEN cur\_loans%NOTFOUND;

-- Define the new interest rate policy here

v\_new\_interest\_rate := v\_interest\_rate \* 1.05; -- Example: increase by 5%

-- Update the loan's interest rate UPDATE loans

SET interest\_rate = v\_new\_interest\_rate WHERE loan\_id = v\_loan\_id;

DBMS\_OUTPUT.PUT\_LINE('Updated interest rate for Loan ID: ' || v\_loan\_id); END LOOP;

CLOSE cur\_loans;

-- Commit the changes COMMIT;

END;

/

# Exercise-7

### Scenario 1: CustomerManagement Package

#### Package Specification

CREATE OR REPLACE PACKAGE CustomerManagement IS

PROCEDURE AddCustomer(p\_customer\_id IN customers.customer\_id%TYPE, p\_name IN customers.name%TYPE,

p\_age IN customers.age%TYPE, p\_balance IN customers.balance%TYPE);

PROCEDURE UpdateCustomerDetails(p\_customer\_id IN customers.customer\_id%TYPE, p\_name IN customers.name%TYPE,

p\_age IN customers.age%TYPE);

FUNCTION GetCustomerBalance(p\_customer\_id IN customers.customer\_id%TYPE) RETURN customers.balance%TYPE;

END CustomerManagement;

/

#### Package Body

CREATE OR REPLACE PACKAGE BODY CustomerManagement IS PROCEDURE AddCustomer(p\_customer\_id IN customers.customer\_id%TYPE,

BEGIN

p\_name IN customers.name%TYPE, p\_age IN customers.age%TYPE,

p\_balance IN customers.balance%TYPE) IS

INSERT INTO customers (customer\_id, name, age, balance) VALUES (p\_customer\_id, p\_name, p\_age, p\_balance);

COMMIT; EXCEPTION

WHEN OTHERS THEN DBMS\_OUTPUT.PUT\_LINE('Error: ' || SQLERRM);

END AddCustomer;

PROCEDURE UpdateCustomerDetails(p\_customer\_id IN customers.customer\_id%TYPE, p\_name IN customers.name%TYPE,

p\_age IN customers.age%TYPE) IS

BEGIN

UPDATE customers

SET name = p\_name, age = p\_age WHERE customer\_id = p\_customer\_id;

COMMIT; EXCEPTION

WHEN OTHERS THEN DBMS\_OUTPUT.PUT\_LINE('Error: ' || SQLERRM);

END UpdateCustomerDetails;

FUNCTION GetCustomerBalance(p\_customer\_id IN customers.customer\_id%TYPE)

RETURN customers.balance%TYPE IS v\_balance customers.balance%TYPE;

BEGIN

SELECT balance INTO v\_balance FROM customers

WHERE customer\_id = p\_customer\_id;

RETURN v\_balance; EXCEPTION

WHEN NO\_DATA\_FOUND THEN RETURN NULL;

WHEN OTHERS THEN DBMS\_OUTPUT.PUT\_LINE('Error: ' || SQLERRM); RETURN NULL;

END GetCustomerBalance; END CustomerManagement;

/

**Scenario 2: EmployeeManagement Package**

#### Package Specification

CREATE OR REPLACE PACKAGE EmployeeManagement IS

PROCEDURE HireEmployee(p\_employee\_id IN employees.employee\_id%TYPE, p\_name IN employees.name%TYPE,

p\_department\_id IN employees.department\_id%TYPE, p\_salary IN employees.salary%TYPE);

PROCEDURE UpdateEmployeeDetails(p\_employee\_id IN employees.employee\_id%TYPE, p\_name IN employees.name%TYPE,

p\_department\_id IN employees.department\_id%TYPE);

FUNCTION CalculateAnnualSalary(p\_employee\_id IN employees.employee\_id%TYPE) RETURN NUMBER;

END EmployeeManagement;

/

#### Package Body

CREATE OR REPLACE PACKAGE BODY EmployeeManagement IS PROCEDURE HireEmployee(p\_employee\_id IN employees.employee\_id%TYPE,

p\_name IN employees.name%TYPE,

p\_department\_id IN employees.department\_id%TYPE, p\_salary IN employees.salary%TYPE) IS

BEGIN

INSERT INTO employees (employee\_id, name, department\_id, salary) VALUES (p\_employee\_id, p\_name, p\_department\_id, p\_salary);

COMMIT; EXCEPTION

WHEN OTHERS THEN DBMS\_OUTPUT.PUT\_LINE('Error: ' || SQLERRM);

END HireEmployee;

PROCEDURE UpdateEmployeeDetails(p\_employee\_id IN employees.employee\_id%TYPE, p\_name IN employees.name%TYPE,

p\_department\_id IN employees.department\_id%TYPE) IS

BEGIN

UPDATE employees

SET name = p\_name, department\_id = p\_department\_id WHERE employee\_id = p\_employee\_id;

COMMIT; EXCEPTION

WHEN OTHERS THEN DBMS\_OUTPUT.PUT\_LINE('Error: ' || SQLERRM);

END UpdateEmployeeDetails;

FUNCTION CalculateAnnualSalary(p\_employee\_id IN employees.employee\_id%TYPE) RETURN NUMBER IS

v\_salary employees.salary%TYPE; 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 NULL;

WHEN OTHERS THEN DBMS\_OUTPUT.PUT\_LINE('Error: ' || SQLERRM); RETURN NULL;

END CalculateAnnualSalary; END EmployeeManagement;

/

## Scenario 3: AccountOperations Package

#### Package Specification

CREATE OR REPLACE PACKAGE AccountOperations IS

PROCEDURE OpenAccount(p\_account\_id IN accounts.account\_id%TYPE, p\_customer\_id IN accounts.customer\_id%TYPE, p\_account\_type IN accounts.account\_type%TYPE,

p\_balance IN accounts.balance%TYPE);

PROCEDURE CloseAccount(p\_account\_id IN accounts.account\_id%TYPE);

FUNCTION GetTotalBalance(p\_customer\_id IN accounts.customer\_id%TYPE) RETURN NUMBER;

END AccountOperations;

/

#### Package Body

CREATE OR REPLACE PACKAGE BODY AccountOperations IS PROCEDURE OpenAccount(p\_account\_id IN accounts.account\_id%TYPE,

p\_customer\_id IN accounts.customer\_id%TYPE, p\_account\_type IN accounts.account\_type%TYPE, p\_balance IN accounts.balance%TYPE) IS

BEGIN

INSERT INTO accounts (account\_id, customer\_id, account\_type, balance) VALUES (p\_account\_id, p\_customer\_id, p\_account\_type, p\_balance);

COMMIT; EXCEPTION

WHEN OTHERS THEN DBMS\_OUTPUT.PUT\_LINE('Error: ' || SQLERRM);

END OpenAccount;

PROCEDURE CloseAccount(p\_account\_id IN accounts.account\_id%TYPE) IS BEGIN

DELETE FROM accounts

WHERE account\_id = p\_account\_id;

COMMIT; EXCEPTION

WHEN OTHERS THEN DBMS\_OUTPUT.PUT\_LINE('Error: ' || SQLERRM);

END CloseAccount;

FUNCTION GetTotalBalance(p\_customer\_id IN accounts.customer\_id%TYPE) RETURN NUMBER IS

v\_total\_balance NUMBER; BEGIN

SELECT SUM(balance) INTO v\_total\_balance FROM accounts

WHERE customer\_id = p\_customer\_id;

RETURN v\_total\_balance; EXCEPTION

WHEN NO\_DATA\_FOUND THEN RETURN 0;

WHEN OTHERS THEN DBMS\_OUTPUT.PUT\_LINE('Error: ' || SQLERRM); RETURN 0;

END GetTotalBalance; END AccountOperations;

/