**Exercise 1:**

* + **Question:** Write a PL/SQL block that loops through all customers, checks their age, and if they are above 60, apply a 1% discount to their current loan interest rates.

Answer:

BEGIN

FOR rec IN (

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

FROM customers

WHERE age > 60

)

LOOP

UPDATE customers

SET loan\_interest\_rate = loan\_interest\_rate - 1

WHERE customer\_id = rec.customer\_id;

DBMS\_OUTPUT.PUT\_LINE('Discount applied for customer ID: ' || rec.customer\_id);

END LOOP;

COMMIT;

END;

/

* + **Question:** Write a PL/SQL block that iterates through all customers and sets a flag IsVIP to TRUE for those with a balance over $10,000.

Answer:

BEGIN

FOR rec IN (

SELECT customer\_id, balance

FROM customers

WHERE balance > 10000

)

LOOP

UPDATE customers

SET isvip = 'Y' -- or TRUE if using BOOLEAN type

WHERE customer\_id = rec.customer\_id;

DBMS\_OUTPUT.PUT\_LINE('Promoted to VIP: Customer ID ' || rec.customer\_id);

END LOOP;

COMMIT;

END;

/

**Question:** Write a PL/SQL block that fetches all loans due in the next 30 days and prints a reminder message for each customer

Answer:

BEGIN

FOR rec IN (

SELECT l.loan\_id, l.customer\_id, l.due\_date, l.amount, c.name

FROM loans l

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

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

)

LOOP

DBMS\_OUTPUT.PUT\_LINE('Reminder: Dear ' || rec.name ||

', your loan (ID: ' || rec.loan\_id ||

') of amount $' || rec.amount ||

' is due on ' || TO\_CHAR(rec.due\_date, 'DD-MON-YYYY') || '.');

END LOOP;

END;

/

**Exercise 2:**

* + **Question:** Write a stored procedure **SafeTransferFunds** that transfers funds between two accounts. Ensure that if any error occurs (e.g., insufficient funds), an appropriate error message is logged and the transaction is rolled back.

Answer:

CREATE OR REPLACE PROCEDURE SafeTransferFunds (

p\_from\_account\_id IN NUMBER,

p\_to\_account\_id IN NUMBER,

p\_amount IN NUMBER

)

IS

v\_from\_balance NUMBER;

BEGIN

-- Get balance of source account

SELECT balance INTO v\_from\_balance

FROM accounts

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

-- Check for sufficient funds

IF v\_from\_balance < p\_amount THEN

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

END IF;

-- Debit from source

UPDATE accounts

SET balance = balance - p\_amount

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

-- Credit to target

UPDATE accounts

SET balance = balance + p\_amount

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

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('Transfer successful: ₹' || p\_amount ||

' from account ' || p\_from\_account\_id ||

' to account ' || p\_to\_account\_id);

EXCEPTION

WHEN OTHERS THEN

-- Log error and rollback

ROLLBACK;

INSERT INTO error\_log (error\_message)

VALUES ('Transfer failed: ' || SQLERRM);

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

END;

/

* + **Question:** Write a stored procedure **UpdateSalary** that increases the salary of an employee by a given percentage. If the employee ID does not exist, handle the exception and log an error message.

**Answer:**

CREATE OR REPLACE PROCEDURE UpdateSalary (

p\_employee\_id IN NUMBER,

p\_percentage IN NUMBER

)

IS

v\_current\_salary NUMBER;

BEGIN

-- Get current salary

SELECT salary INTO v\_current\_salary

FROM employees

WHERE employee\_id = p\_employee\_id;

-- Update salary

UPDATE employees

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

WHERE employee\_id = p\_employee\_id;

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('Salary updated successfully for Employee ID ' || p\_employee\_id);

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

ROLLBACK;

INSERT INTO error\_log (error\_message)

VALUES ('Update failed: Employee ID ' || p\_employee\_id || ' not found.');

DBMS\_OUTPUT.PUT\_LINE('Error: Employee ID not found.');

WHEN OTHERS THEN

ROLLBACK;

INSERT INTO error\_log (error\_message)

VALUES ('Update failed for Employee ID ' || p\_employee\_id || ': ' || SQLERRM);

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

END;

/

* + **Question:** Write a stored procedure **AddNewCustomer** that inserts a new customer into the Customers table. If a customer with the same ID already exists, handle the exception by logging an error and preventing the insertion.

Answer:

CREATE OR REPLACE PROCEDURE AddNewCustomer (

p\_customer\_id IN NUMBER,

p\_name IN VARCHAR2,

p\_email IN VARCHAR2,

p\_balance IN NUMBER

)

IS

v\_exists NUMBER;

BEGIN

-- Check if customer already exists

SELECT COUNT(\*) INTO v\_exists

FROM customers

WHERE customer\_id = p\_customer\_id;

IF v\_exists > 0 THEN

RAISE\_APPLICATION\_ERROR(-20001, 'Customer with ID ' || p\_customer\_id || ' already exists.');

END IF;

-- Insert new customer

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

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

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('Customer added successfully: ' || p\_name);

EXCEPTION

WHEN OTHERS THEN

ROLLBACK;

INSERT INTO error\_log (error\_message)

VALUES ('Add customer failed for ID ' || p\_customer\_id || ': ' || SQLERRM);

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

END;

/