**Exercise 1: Control Structures**

# Exercise 1 - Scenario 1: Age-based Loan Interest Discount

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.

## PL/SQL Code:

BEGIN  
 FOR rec IN (SELECT customer\_id, age, interest\_rate FROM customers) LOOP  
 IF rec.age > 60 THEN  
 UPDATE customers  
 SET interest\_rate = interest\_rate - 0.01  
 WHERE customer\_id = rec.customer\_id;  
 END IF;  
 END LOOP;  
 COMMIT;  
END;  
/

## Before Execution:

|  |  |  |
| --- | --- | --- |
| customer\_id | age | interest\_rate |
| 101 | 65 | 7.5% |
| 102 | 55 | 8.0% |
| 103 | 70 | 6.5% |

## After Execution:

|  |  |  |
| --- | --- | --- |
| customer\_id | age | interest\_rate |
| 101 | 65 | 6.5% |
| 102 | 55 | 8.0% |
| 103 | 70 | 5.5% |

## Actual Output:

Interest rates for customers aged over 60 have been reduced by 1%.

# Exercise 1 - Scenario 2: VIP Status Based on Balance

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.

## PL/SQL Code:

BEGIN  
 FOR rec IN (SELECT customer\_id, balance FROM customers) LOOP  
 IF rec.balance > 10000 THEN  
 UPDATE customers  
 SET IsVIP = 'TRUE'  
 WHERE customer\_id = rec.customer\_id;  
 END IF;  
 END LOOP;  
 COMMIT;  
END;  
/

## Before Execution:

|  |  |  |
| --- | --- | --- |
| customer\_id | balance | IsVIP |
| 201 | 15000 | FALSE |
| 202 | 8000 | FALSE |
| 203 | 11000 | FALSE |

## After Execution:

|  |  |  |
| --- | --- | --- |
| customer\_id | balance | IsVIP |
| 201 | 15000 | TRUE |
| 202 | 8000 | FALSE |
| 203 | 11000 | TRUE |

## Actual Output:

Customers with balance over $10,000 have been marked as VIP.

# Exercise 1 - Scenario 3: Loan Due Reminders

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

## PL/SQL Code:

BEGIN  
 FOR rec IN (SELECT customer\_id, loan\_id, due\_date FROM loans WHERE due\_date <= SYSDATE + 30) LOOP  
 DBMS\_OUTPUT.PUT\_LINE('Reminder: Loan ' || rec.loan\_id || ' for Customer ' || rec.customer\_id || ' is due on ' || TO\_CHAR(rec.due\_date, 'DD-MON-YYYY'));  
 END LOOP;  
END;  
/

## Before Execution:

|  |  |  |
| --- | --- | --- |
| loan\_id | customer\_id | due\_date |
| 301 | 201 | 10-JUL-25 |
| 302 | 202 | 25-AUG-25 |
| 303 | 203 | 01-JUL-25 |

## Actual Output:

Reminder: Loan 301 for Customer 201 is due on 10-JUL-2025  
Reminder: Loan 303 for Customer 203 is due on 01-JUL-2025

**-------------------------------------------------**

**Exercise 3: Stored Procedures**

# Exercise 3 - Scenario 1: Monthly Interest Processing

Write a stored procedure ProcessMonthlyInterest that calculates and updates the balance of all savings accounts by applying an interest rate of 1%.

## PL/SQL Code:

CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest IS  
BEGIN  
 FOR rec IN (SELECT account\_id, balance FROM savings\_accounts) LOOP  
 UPDATE savings\_accounts  
 SET balance = balance + (balance \* 0.01)  
 WHERE account\_id = rec.account\_id;  
 END LOOP;  
 COMMIT;  
END;  
/

## Before Execution:

|  |  |
| --- | --- |
| account\_id | balance |
| 501 | 1000.00 |
| 502 | 2000.00 |

## After Execution:

|  |  |
| --- | --- |
| account\_id | balance |
| 501 | 1010.00 |
| 502 | 2020.00 |

## Actual Output:

Balances of all savings accounts increased by 1%.

# Exercise 3 - Scenario 2: Employee Bonus Update

Write a stored procedure UpdateEmployeeBonus that updates the salary of employees in a given department by adding a bonus percentage passed as a parameter.

## PL/SQL Code:

CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus(dept\_id IN NUMBER, bonus\_percent IN NUMBER) IS  
BEGIN  
 UPDATE employees  
 SET salary = salary + (salary \* bonus\_percent / 100)  
 WHERE department\_id = dept\_id;  
 COMMIT;  
END;  
/

## Before Execution:

|  |  |  |
| --- | --- | --- |
| employee\_id | dept\_id | salary |
| 601 | 10 | 50000 |
| 602 | 10 | 55000 |

## After Execution:

|  |  |  |
| --- | --- | --- |
| employee\_id | dept\_id | salary |
| 601 | 10 | 55000 |
| 602 | 10 | 60500 |

## Actual Output:

Salaries of employees in the specified department increased by the given bonus percentage.

# Exercise 3 - Scenario 3: Transfer Funds Between Accounts

Write a stored procedure TransferFunds that transfers a specified amount from one account to another, checking that the source account has sufficient balance.

## PL/SQL Code:

CREATE OR REPLACE PROCEDURE TransferFunds(from\_acc IN NUMBER, to\_acc IN NUMBER, amt IN NUMBER) IS  
 v\_balance NUMBER;  
BEGIN  
 SELECT balance INTO v\_balance FROM accounts WHERE account\_id = from\_acc;  
 IF v\_balance >= amt THEN  
 UPDATE accounts SET balance = balance - amt WHERE account\_id = from\_acc;  
 UPDATE accounts SET balance = balance + amt WHERE account\_id = to\_acc;  
 COMMIT;  
 ELSE  
 RAISE\_APPLICATION\_ERROR(-20001, 'Insufficient Balance');  
 END IF;  
END;  
/

## Before Execution:

|  |  |
| --- | --- |
| account\_id | balance |
| 701 | 3000 |
| 702 | 2000 |

## After Execution:

|  |  |
| --- | --- |
| account\_id | balance |
| 701 | 2000 |
| 702 | 3000 |

## Actual Output:

If sufficient balance:  
Transfer completed successfully.  
  
If not:  
ORA-20001: Insufficient Balance