**Cognizant DN 4.0 Deep Skilling – Java FSE**

**Week 2: PL/SQL**

**Exercise 1: Control Structures**

**Scenario 1**

|  |
| --- |
| BEGIN  FOR rec IN (SELECT \* FROM Loans L JOIN Customers C ON L.CustomerID = C.CustomerID) LOOP  IF MONTHS\_BETWEEN(SYSDATE, rec.DOB)/12 > 60 THEN  UPDATE Loans SET InterestRate = InterestRate - 1 WHERE LoanID = rec.LoanID;  END IF;  END LOOP;  END; |

**Scenario 2**

|  |
| --- |
| BEGIN  FOR rec IN (SELECT \* FROM Customers) LOOP  IF rec.Balance > 10000 THEN  UPDATE Customers SET IsVIP = 'TRUE' WHERE CustomerID = rec.CustomerID;  END IF;  END LOOP;  END; |

**Scenario 3**

|  |
| --- |
| BEGIN  FOR rec IN (SELECT \* FROM Loans WHERE EndDate <= SYSDATE + 30) LOOP  DBMS\_OUTPUT.PUT\_LINE('Reminder: Loan ' || rec.LoanID || ' for Customer ' || rec.CustomerID || ' is due soon.');  END LOOP;  END; |

**Exercise 2: Error Handling**

**Scenario 1**

|  |
| --- |
| CREATE OR REPLACE PROCEDURE SafeTransferFunds(p\_from NUMBER, p\_to NUMBER, p\_amount NUMBER) AS  BEGIN  UPDATE Accounts SET Balance = Balance - p\_amount WHERE AccountID = p\_from;  UPDATE Accounts SET Balance = Balance + p\_amount WHERE AccountID = p\_to;  COMMIT;  EXCEPTION  WHEN OTHERS THEN  ROLLBACK;  DBMS\_OUTPUT.PUT\_LINE('Transfer failed: ' || SQLERRM);  END; |

**Scenario 2**

|  |
| --- |
| CREATE OR REPLACE PROCEDURE UpdateSalary(p\_empid NUMBER, p\_percent NUMBER) AS  BEGIN  UPDATE Employees SET Salary = Salary \* (1 + p\_percent/100) WHERE EmployeeID = p\_empid;  IF SQL%ROWCOUNT = 0 THEN  RAISE\_APPLICATION\_ERROR(-20001, 'Employee not found');  END IF;  EXCEPTION  WHEN OTHERS THEN  DBMS\_OUTPUT.PUT\_LINE('Error updating salary: ' || SQLERRM);  END; |

**Scenario 3**

|  |
| --- |
| CREATE OR REPLACE PROCEDURE AddNewCustomer(p\_id NUMBER, p\_name VARCHAR2, p\_dob DATE, p\_balance NUMBER) AS  BEGIN  INSERT INTO Customers(CustomerID, Name, DOB, Balance, LastModified) VALUES(p\_id, p\_name, p\_dob, p\_balance, SYSDATE);  EXCEPTION  WHEN DUP\_VAL\_ON\_INDEX THEN  DBMS\_OUTPUT.PUT\_LINE('Customer ID already exists');  WHEN OTHERS THEN  DBMS\_OUTPUT.PUT\_LINE('Error: ' || SQLERRM);  END; |

**Exercise 3: Stored Procedures**

**Scenario 1**

|  |
| --- |
| CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest AS  BEGIN  UPDATE Accounts SET Balance = Balance \* 1.01 WHERE AccountType = 'Savings';  END; |

**Scenario 2**

|  |
| --- |
| CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus(p\_dept VARCHAR2, p\_bonus\_percent NUMBER) AS  BEGIN  UPDATE Employees SET Salary = Salary + (Salary \* p\_bonus\_percent / 100) WHERE Department = p\_dept;  END; |

**Scenario 3**

|  |
| --- |
| CREATE OR REPLACE PROCEDURE TransferFunds(p\_from NUMBER, p\_to NUMBER, p\_amount NUMBER) AS  v\_balance NUMBER;  BEGIN  SELECT Balance INTO v\_balance FROM Accounts WHERE AccountID = p\_from;  IF v\_balance >= p\_amount THEN  UPDATE Accounts SET Balance = Balance - p\_amount WHERE AccountID = p\_from;  UPDATE Accounts SET Balance = Balance + p\_amount WHERE AccountID = p\_to;  COMMIT;  ELSE  DBMS\_OUTPUT.PUT\_LINE('Insufficient funds');  END IF;  END; |

**Exercise 4: Functions**

**Scenario 1**

|  |
| --- |
| CREATE OR REPLACE FUNCTION CalculateAge(p\_dob DATE) RETURN NUMBER IS  BEGIN  RETURN FLOOR(MONTHS\_BETWEEN(SYSDATE, p\_dob)/12);  END; |

**Scenario 2**

|  |
| --- |
| CREATE OR REPLACE FUNCTION CalculateMonthlyInstallment(p\_amount NUMBER, p\_rate NUMBER, p\_years NUMBER) RETURN NUMBER IS  v\_monthly\_rate NUMBER;  v\_months NUMBER;  BEGIN  v\_monthly\_rate := p\_rate / (12 \* 100);  v\_months := p\_years \* 12;  RETURN (p\_amount \* v\_monthly\_rate) / (1 - POWER(1 + v\_monthly\_rate, -v\_months));  END; |

**Scenario 3**

|  |
| --- |
| CREATE OR REPLACE FUNCTION HasSufficientBalance(p\_accid NUMBER, p\_amount NUMBER) RETURN BOOLEAN IS  v\_balance NUMBER;  BEGIN  SELECT Balance INTO v\_balance FROM Accounts WHERE AccountID = p\_accid;  RETURN v\_balance >= p\_amount;  EXCEPTION  WHEN NO\_DATA\_FOUND THEN  RETURN FALSE;  END; |

**Exercise 5: Triggers**

**Scenario 1**

|  |
| --- |
| CREATE OR REPLACE TRIGGER UpdateCustomerLastModified  BEFORE UPDATE ON Customers  FOR EACH ROW  BEGIN  :NEW.LastModified := SYSDATE;  END; |

**Scenario 2**

|  |
| --- |
| CREATE TABLE AuditLog (  LogID NUMBER GENERATED BY DEFAULT AS IDENTITY,  TransactionID NUMBER,  ActionDate DATE,  Details VARCHAR2(200)  );  CREATE OR REPLACE TRIGGER LogTransaction  AFTER INSERT ON Transactions  FOR EACH ROW  BEGIN  INSERT INTO AuditLog(TransactionID, ActionDate, Details)  VALUES(:NEW.TransactionID, SYSDATE, 'Transaction recorded.');  END; |

**Scenario 3**

|  |
| --- |
| CREATE OR REPLACE TRIGGER CheckTransactionRules  BEFORE INSERT ON Transactions  FOR EACH ROW  DECLARE  v\_balance NUMBER;  BEGIN  IF :NEW.TransactionType = 'Withdrawal' THEN  SELECT Balance INTO v\_balance FROM Accounts WHERE AccountID = :NEW.AccountID;  IF :NEW.Amount > v\_balance THEN  RAISE\_APPLICATION\_ERROR(-20002, 'Insufficient balance for withdrawal');  END IF;  ELSIF :NEW.TransactionType = 'Deposit' AND :NEW.Amount <= 0 THEN  RAISE\_APPLICATION\_ERROR(-20003, 'Deposit must be positive');  END IF;  END; |

**Exercise 6: Cursors**

**Scenario 1**

|  |
| --- |
| DECLARE  CURSOR cur\_txn IS  SELECT \* FROM Transactions WHERE EXTRACT(MONTH FROM TransactionDate) = EXTRACT(MONTH FROM SYSDATE);  BEGIN  FOR rec IN cur\_txn LOOP  DBMS\_OUTPUT.PUT\_LINE('Customer Transaction: Account ' || rec.AccountID || ', Amount: ' || rec.Amount);  END LOOP;  END; |

**Scenario 2**

|  |
| --- |
| DECLARE  CURSOR cur\_acc IS SELECT \* FROM Accounts;  BEGIN  FOR rec IN cur\_acc LOOP  UPDATE Accounts SET Balance = Balance - 100 WHERE AccountID = rec.AccountID;  END LOOP;  END; |

**Scenario 3**

|  |
| --- |
| DECLARE  CURSOR cur\_loan IS SELECT \* FROM Loans;  BEGIN  FOR rec IN cur\_loan LOOP  UPDATE Loans SET InterestRate = rec.InterestRate + 0.5 WHERE LoanID = rec.LoanID;  END LOOP;  END; |

**Exercise 7: Packages**

**Scenario 1**

|  |
| --- |
| CREATE OR REPLACE PACKAGE CustomerManagement AS  PROCEDURE AddCustomer(p\_id NUMBER, p\_name VARCHAR2, p\_dob DATE, p\_balance NUMBER);  PROCEDURE UpdateCustomer(p\_id NUMBER, p\_name VARCHAR2);  FUNCTION GetCustomerBalance(p\_id NUMBER) RETURN NUMBER;  END;  CREATE OR REPLACE PACKAGE BODY CustomerManagement AS  PROCEDURE AddCustomer(p\_id NUMBER, p\_name VARCHAR2, p\_dob DATE, p\_balance NUMBER) IS  BEGIN  INSERT INTO Customers VALUES(p\_id, p\_name, p\_dob, p\_balance, SYSDATE);  END;  PROCEDURE UpdateCustomer(p\_id NUMBER, p\_name VARCHAR2) IS  BEGIN  UPDATE Customers SET Name = p\_name WHERE CustomerID = p\_id;  END;  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;  END;  END; |

**Scenario 2**

|  |
| --- |
| CREATE OR REPLACE PACKAGE EmployeeManagement AS  PROCEDURE HireEmployee(p\_id NUMBER, p\_name VARCHAR2, p\_position VARCHAR2, p\_salary NUMBER, p\_dept VARCHAR2);  PROCEDURE UpdateEmployee(p\_id NUMBER, p\_salary NUMBER);  FUNCTION GetAnnualSalary(p\_id NUMBER) RETURN NUMBER;  END;  CREATE OR REPLACE PACKAGE BODY EmployeeManagement AS  PROCEDURE HireEmployee(p\_id NUMBER, p\_name VARCHAR2, p\_position VARCHAR2, p\_salary NUMBER, p\_dept VARCHAR2) IS  BEGIN  INSERT INTO Employees VALUES(p\_id, p\_name, p\_position, p\_salary, p\_dept, SYSDATE);  END;  PROCEDURE UpdateEmployee(p\_id NUMBER, p\_salary NUMBER) IS  BEGIN  UPDATE Employees SET Salary = p\_salary WHERE EmployeeID = p\_id;  END;  FUNCTION GetAnnualSalary(p\_id NUMBER) RETURN NUMBER IS  v\_salary NUMBER;  BEGIN  SELECT Salary INTO v\_salary FROM Employees WHERE EmployeeID = p\_id;  RETURN v\_salary \* 12;  END;  END; |

**Scenario 3**

|  |
| --- |
| CREATE OR REPLACE PACKAGE AccountOperations AS  PROCEDURE OpenAccount(p\_accid NUMBER, p\_custid NUMBER, p\_type VARCHAR2, p\_balance NUMBER);  PROCEDURE CloseAccount(p\_accid NUMBER);  FUNCTION GetTotalBalance(p\_custid NUMBER) RETURN NUMBER;  END;  CREATE OR REPLACE PACKAGE BODY AccountOperations AS  PROCEDURE OpenAccount(p\_accid NUMBER, p\_custid NUMBER, p\_type VARCHAR2, p\_balance NUMBER) IS  BEGIN  INSERT INTO Accounts VALUES(p\_accid, p\_custid, p\_type, p\_balance, SYSDATE);  END;  PROCEDURE CloseAccount(p\_accid NUMBER) IS  BEGIN  DELETE FROM Accounts WHERE AccountID = p\_accid;  END;  FUNCTION GetTotalBalance(p\_custid NUMBER) RETURN NUMBER IS  v\_total NUMBER;  BEGIN  SELECT SUM(Balance) INTO v\_total FROM Accounts WHERE CustomerID = p\_custid;  RETURN v\_total;  END;  END; |