**Hands-on Week2(PL SQL Commands)**

**Exercise 1: Control Structures**

**Scenario 1:** The bank wants to apply a discount to loan interest rates for customers above 60 years old.

BEGIN

FOR rec IN (

SELECT CustomerID

FROM Customers

WHERE (TRUNC(MONTHS\_BETWEEN(SYSDATE, DOB) / 12)) > 60

) LOOP

UPDATE Loans

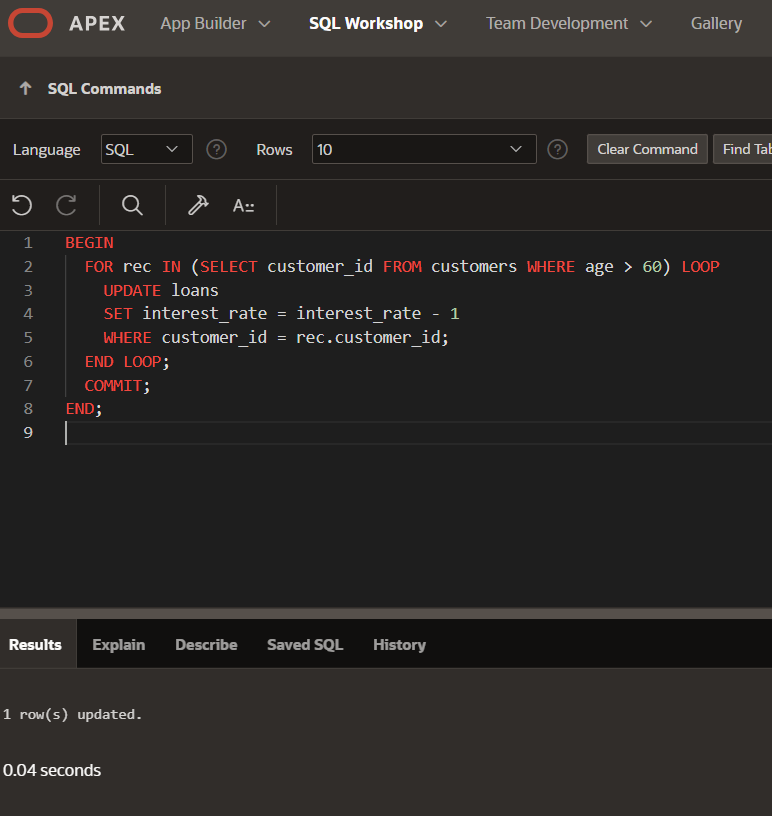
SET InterestRate = InterestRate - 1

WHERE CustomerID = rec.CustomerID;

END LOOP;

COMMIT;

END;



**Scenario 2:** **:** A customer can be promoted to VIP status based on their balance..

BEGIN

FOR rec IN (

SELECT CustomerID

FROM Customers

WHERE Balance > 10000

) LOOP

UPDATE Customers

SET IsVIP = 'Y'

WHERE CustomerID = rec.CustomerID;

END LOOP;

COMMIT;

END;

A screenshot of a computer

AI-generated content may be incorrect.

**Scenario 3:** The bank wants to send reminders to customers whose loans are due within the next 30 days.

CREATE OR REPLACE PROCEDURE SendLoanReminders IS

BEGIN

FOR rec IN (

SELECT l.LoanID, c.Name, l.EndDate

FROM Loans l

JOIN Customers c ON l.CustomerID = c.CustomerID

WHERE l.EndDate BETWEEN SYSDATE AND SYSDATE + 30

) LOOP

DBMS\_OUTPUT.PUT\_LINE('🔔 Reminder: ' || rec.Name ||

', your loan (ID: ' || rec.LoanID ||

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

END LOOP;

END;

A screenshot of a computer

AI-generated content may be incorrect.

**Exercise 2: Error Handling**

**Scenario 1:** Handle exceptions during fund transfers between accounts.

CREATE OR REPLACE PROCEDURE SafeTransferFunds (

p\_from\_id IN NUMBER,

p\_to\_id IN NUMBER,

p\_amount IN NUMBER

)

IS

v\_balance NUMBER;

BEGIN

SELECT Balance INTO v\_balance FROM Accounts WHERE AccountID = p\_from\_id;

IF v\_balance < p\_amount THEN

RAISE\_APPLICATION\_ERROR(-20001, 'Not enough funds.');

END IF;

UPDATE Accounts SET Balance = Balance - p\_amount WHERE AccountID = p\_from\_id;

UPDATE Accounts SET Balance = Balance + p\_amount WHERE AccountID = p\_to\_id;

COMMIT;

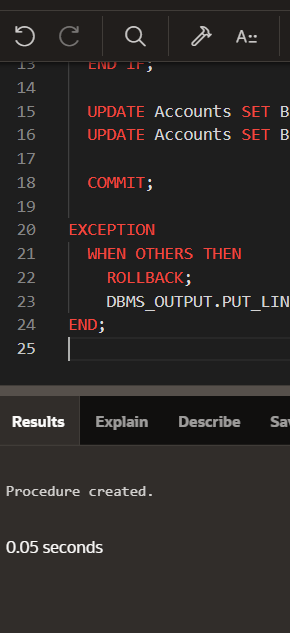
EXCEPTION

WHEN OTHERS THEN

ROLLBACK;

DBMS\_OUTPUT.PUT\_LINE('❌ Error in transfer: ' || SQLERRM);

END;



**Scenario 2:** Manage errors when updating employee salaries.

CREATE OR REPLACE PROCEDURE UpdateSalary (

p\_emp\_id IN NUMBER,

p\_percent IN NUMBER

)

IS

BEGIN

UPDATE Employees

SET Salary = Salary + (Salary \* p\_percent / 100)

WHERE EmployeeID = p\_emp\_id;

IF SQL%ROWCOUNT = 0 THEN

DBMS\_OUTPUT.PUT\_LINE('❌ No employee found with ID ' || p\_emp\_id);

ELSE

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('✅ Salary updated.');

END IF;

EXCEPTION

WHEN OTHERS THEN

ROLLBACK;

DBMS\_OUTPUT.PUT\_LINE('❌ Error updating salary: ' || SQLERRM);

END;

A screenshot of a computer program

AI-generated content may be incorrect.

**Scenario 3:** Ensure data integrity when adding a new customer.

CREATE OR REPLACE PROCEDURE AddNewCustomer (

p\_id IN NUMBER,

p\_name IN VARCHAR2,

p\_dob IN DATE,

p\_bal IN NUMBER

)

IS

BEGIN

INSERT INTO Customers (CustomerID, Name, DOB, Balance, LastModified)

VALUES (p\_id, p\_name, p\_dob, p\_bal, SYSDATE);

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('✅ Customer added.');

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;

A screenshot of a computer program

AI-generated content may be incorrect.

**Exercise 3: Stored Procedures**

**Scenario 1:** The bank needs to process monthly interest for all savings accounts.

CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest

IS

BEGIN

UPDATE Accounts

SET Balance = Balance + (Balance \* 0.01)

WHERE AccountType = 'Savings';

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('✅ Interest processed for all savings accounts.');

EXCEPTION

WHEN OTHERS THEN

ROLLBACK;

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

END;

To run-

BEGIN

ProcessMonthlyInterest;

END;

A screenshot of a computer

AI-generated content may be incorrect.

**Scenario 2:** The bank wants to implement a bonus scheme for employees based on their performance.

CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus (

p\_department IN VARCHAR2,

p\_bonus\_pct IN NUMBER

)

IS

BEGIN

UPDATE Employees

SET Salary = Salary + (Salary \* p\_bonus\_pct / 100)

WHERE Department = p\_department;

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('✅ Bonus applied to department: ' || p\_department);

EXCEPTION

WHEN OTHERS THEN

ROLLBACK;

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

END;

**To run:**

sql

Copy code

BEGIN

UpdateEmployeeBonus('Finance', 5);

END;

A screenshot of a computer

AI-generated content may be incorrect.

**Scenario 3:** Customers should be able to transfer funds between their accounts.

CREATE OR REPLACE PROCEDURE TransferFunds (

p\_from\_id IN NUMBER,

p\_to\_id IN NUMBER,

p\_amount IN NUMBER

)

IS

v\_balance NUMBER;

BEGIN

SELECT Balance INTO v\_balance

FROM Accounts

WHERE AccountID = p\_from\_id

FOR UPDATE;

IF v\_balance < p\_amount THEN

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

END IF;

-- Deduct and Add

UPDATE Accounts SET Balance = Balance - p\_amount WHERE AccountID = p\_from\_id;

UPDATE Accounts SET Balance = Balance + p\_amount WHERE AccountID = p\_to\_id;

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('✅ Transfer complete.');

EXCEPTION

WHEN OTHERS THEN

ROLLBACK;

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

END;

**To run:**

sql

Copy code

BEGIN

TransferFunds(1, 2, 1000);

END;

A screenshot of a computer

AI-generated content may be incorrect.

**Exercise 4: Functions**

**Scenario 1:** Calculate the age of customers for eligibility checks.

CREATE OR REPLACE FUNCTION CalculateAge (

p\_dob IN DATE

) RETURN NUMBER

IS

v\_age NUMBER;

BEGIN

v\_age := TRUNC(MONTHS\_BETWEEN(SYSDATE, p\_dob) / 12);

RETURN v\_age;

END;

SELECT CalculateAge(DOB) AS Age FROM Customers;

A screenshot of a computer

AI-generated content may be incorrect.

**Scenario 2:** The bank needs to compute the monthly installment for a loan.

CREATE OR REPLACE FUNCTION CalculateMonthlyInstallment (

p\_amount IN NUMBER,

p\_annual\_rate IN NUMBER,

p\_years IN NUMBER

) RETURN NUMBER

IS

v\_monthly\_rate NUMBER;

v\_months NUMBER;

v\_emi NUMBER;

BEGIN

v\_monthly\_rate := p\_annual\_rate / 12 / 100;

v\_months := p\_years \* 12;

v\_emi := (p\_amount \* v\_monthly\_rate \* POWER(1 + v\_monthly\_rate, v\_months)) /

(POWER(1 + v\_monthly\_rate, v\_months) - 1);

RETURN ROUND(v\_emi, 2);

END;

SELECT CalculateMonthlyInstallment(100000, 6.5, 5) AS EMI FROM dual;

A screenshot of a computer

AI-generated content may be incorrect.

**Scenario 3:** Check if a customer has sufficient balance before making a transaction.

CREATE OR REPLACE FUNCTION HasSufficientBalance (

p\_account\_id IN NUMBER,

p\_amount IN NUMBER

) RETURN BOOLEAN

IS

v\_balance NUMBER;

BEGIN

SELECT Balance INTO v\_balance

FROM Accounts

WHERE AccountID = p\_account\_id;

RETURN v\_balance >= p\_amount;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

RETURN FALSE;

END;

Run:

DECLARE

result BOOLEAN;

BEGIN

result := HasSufficientBalance(101, 2000);

IF result THEN

DBMS\_OUTPUT.PUT\_LINE('✅ Enough balance');

ELSE

DBMS\_OUTPUT.PUT\_LINE('❌ Not enough balance');

END IF;

END;

A screenshot of a computer

AI-generated content may be incorrect.

**Exercise 5: Triggers**

**Scenario 1:** Automatically update the last modified date when a customer's record is updated.

CREATE OR REPLACE TRIGGER UpdateCustomerLastModified

BEFORE UPDATE ON Customers

FOR EACH ROW

BEGIN

:NEW.LastModified := SYSDATE;

END;

A screenshot of a computer

AI-generated content may be incorrect.

**Scenario 2:** Maintain an audit log for all transactions.

CREATE TABLE AuditLog (

LogID NUMBER GENERATED ALWAYS AS IDENTITY,

TransactionID NUMBER,

LogTime DATE

);

CREATE OR REPLACE TRIGGER LogTransaction

AFTER INSERT ON Transactions

FOR EACH ROW

BEGIN

INSERT INTO AuditLog (TransactionID, LogTime)

VALUES (:NEW.TransactionID, SYSDATE);

END;

A screenshot of a computer

AI-generated content may be incorrect.

**Scenario 3:** Enforce business rules on deposits and withdrawals.

CREATE OR REPLACE TRIGGER CheckTransactionRules

BEFORE INSERT ON Transactions

FOR EACH ROW

DECLARE

v\_balance NUMBER;

BEGIN

SELECT Balance INTO v\_balance FROM Accounts WHERE AccountID = :NEW.AccountID;

IF :NEW.TransactionType = 'Withdrawal' AND :NEW.Amount > v\_balance THEN

RAISE\_APPLICATION\_ERROR(-20001, '❌ Not enough balance');

ELSIF :NEW.TransactionType = 'Deposit' AND :NEW.Amount <= 0 THEN

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

END IF;

END;

UPDATE Customers

SET Name = 'Test Update'

WHERE CustomerID = 1;

SELECT Name, LastModified FROM Customers WHERE CustomerID = 1;

A screenshot of a computer

AI-generated content may be incorrect.

**Exercise 6: Cursors**

**Scenario 1:** Generate monthly statements for all customers.

DECLARE

CURSOR txn\_cur IS

SELECT c.Name, t.\*

FROM Transactions t

JOIN Accounts a ON t.AccountID = a.AccountID

JOIN Customers c ON a.CustomerID = c.CustomerID

WHERE EXTRACT(MONTH FROM t.TransactionDate) = EXTRACT(MONTH FROM SYSDATE)

AND EXTRACT(YEAR FROM t.TransactionDate) = EXTRACT(YEAR FROM SYSDATE);

BEGIN

FOR rec IN txn\_cur LOOP

DBMS\_OUTPUT.PUT\_LINE('Customer: ' || rec.Name ||

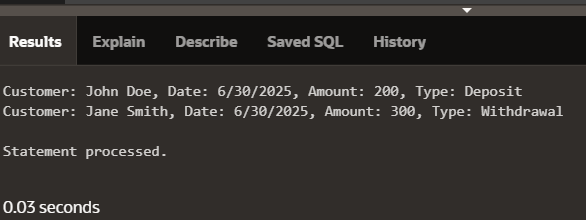
', Date: ' || rec.TransactionDate ||

', Amount: ' || rec.Amount ||

', Type: ' || rec.TransactionType);

END LOOP;

END;



**Scenario 2:** Apply annual fee to all accounts.

DECLARE

CURSOR acc\_cur IS SELECT AccountID, Balance FROM Accounts;

BEGIN

FOR acc IN acc\_cur LOOP

UPDATE Accounts

SET Balance = Balance - 100

WHERE AccountID = acc.AccountID;

END LOOP;

COMMIT;

END;

A screenshot of a computer

AI-generated content may be incorrect.

**Scenario 3:** Update the interest rate for all loans based on a new policy.

DECLARE

CURSOR loan\_cur IS SELECT LoanID, LoanAmount FROM Loans;

BEGIN

FOR rec IN loan\_cur LOOP

UPDATE Loans

SET InterestRate = CASE

WHEN rec.LoanAmount > 50000 THEN 6

ELSE 5

END

WHERE LoanID = rec.LoanID;

END LOOP;

COMMIT;

END;

A screenshot of a computer

AI-generated content may be incorrect.

**Exercise 7: Packages**

**Scenario 1:** Group all customer-related procedures and functions into a package.

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 CustomerManagement;

BODY:

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 (CustomerID, Name, DOB, Balance, LastModified)

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, LastModified = SYSDATE

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 CustomerManagement;

A screenshot of a computer screen

AI-generated content may be incorrect.

**Scenario 2:** Create a package to manage employee data.

Specification:

CREATE OR REPLACE PACKAGE EmployeeManagement AS

PROCEDURE HireEmployee(p\_id NUMBER, p\_name VARCHAR2, p\_pos VARCHAR2, p\_salary NUMBER, p\_dept VARCHAR2);

PROCEDURE UpdateEmployee(p\_id NUMBER, p\_salary NUMBER);

FUNCTION GetAnnualSalary(p\_id NUMBER) RETURN NUMBER;

END EmployeeManagement;

Body:

CREATE OR REPLACE PACKAGE BODY EmployeeManagement AS

PROCEDURE HireEmployee(p\_id NUMBER, p\_name VARCHAR2, p\_pos VARCHAR2, p\_salary NUMBER, p\_dept VARCHAR2) IS

BEGIN

INSERT INTO Employees (EmployeeID, Name, Position, Salary, Department, HireDate)

VALUES (p\_id, p\_name, p\_pos, 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 EmployeeManagement;

A screenshot of a computer

AI-generated content may be incorrect.

Ex:

Get Balance:

DECLARE

v\_balance NUMBER;

BEGIN

v\_balance := CustomerManagement.GetCustomerBalance(101);

DBMS\_OUTPUT.PUT\_LINE('Customer Balance: ' || v\_balance);

END;

A screenshot of a computer

AI-generated content may be incorrect.

**Scenario 3:** Group all account-related operations into a package.

Specification:

CREATE OR REPLACE PACKAGE AccountOperations AS

PROCEDURE OpenAccount(p\_id NUMBER, p\_custid NUMBER, p\_type VARCHAR2, p\_balance NUMBER);

PROCEDURE CloseAccount(p\_id NUMBER);

FUNCTION GetTotalBalance(p\_custid NUMBER) RETURN NUMBER;

END AccountOperations;

Body:

CREATE OR REPLACE PACKAGE BODY AccountOperations AS

PROCEDURE OpenAccount(p\_id NUMBER, p\_custid NUMBER, p\_type VARCHAR2, p\_balance NUMBER) IS

BEGIN

INSERT INTO Accounts (AccountID, CustomerID, AccountType, Balance, LastModified)

VALUES (p\_id, p\_custid, p\_type, p\_balance, SYSDATE);

END;

PROCEDURE CloseAccount(p\_id NUMBER) IS

BEGIN

DELETE FROM Accounts WHERE AccountID = p\_id;

END;

FUNCTION GetTotalBalance(p\_custid NUMBER) RETURN NUMBER IS

v\_total NUMBER := 0;

BEGIN

SELECT NVL(SUM(Balance), 0) INTO v\_total FROM Accounts WHERE CustomerID = p\_custid;

RETURN v\_total;

END;

END AccountOperations;

A screenshot of a computer screen

AI-generated content may be incorrect.