**Digital Nurture 3.0 – Deep Skilling**

**Assessment Solutions**

Oracle Online SQL Editor was used to perform this assessment: <https://livesql.oracle.com/>

**Schemas Created:**

CREATE TABLE Customers (

CustomerID NUMBER PRIMARY KEY,

Name VARCHAR2(100),

DOB DATE,

Balance NUMBER,

LastModified DATE

);



CREATE TABLE Accounts (

AccountID NUMBER PRIMARY KEY,

CustomerID NUMBER,

AccountType VARCHAR2(20),

Balance NUMBER,

LastModified DATE,

FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)

);



CREATE TABLE Transactions (

TransactionID NUMBER PRIMARY KEY,

AccountID NUMBER,

TransactionDate DATE,

Amount NUMBER,

TransactionType VARCHAR2(10),

FOREIGN KEY (AccountID) REFERENCES Accounts(AccountID)

);



CREATE TABLE Loans (

LoanID NUMBER PRIMARY KEY,

CustomerID NUMBER,

LoanAmount NUMBER,

InterestRate NUMBER,

StartDate DATE,

EndDate DATE,

FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)

);



CREATE TABLE Employees (

EmployeeID NUMBER PRIMARY KEY,

Name VARCHAR2(100),

Position VARCHAR2(50),

Salary NUMBER,

Department VARCHAR2(50),

HireDate DATE

);

****

**Let me try to insert values into the created tables:**

**Inserting values into Customers table**

BEGIN

INSERT INTO Customers (CustomerID, Name, DOB, Balance, LastModified) VALUES (1, 'AKCHARA TD', TO\_DATE('1962-01-01', 'YYYY-MM-DD'), 5000, SYSDATE);

INSERT INTO Customers (CustomerID, Name, DOB, Balance, LastModified) VALUES (2, 'ANAMIKA M', TO\_DATE('1990-07-22', 'YYYY-MM-DD'), 15000, SYSDATE);

INSERT INTO Customers (CustomerID, Name, DOB, Balance, LastModified) VALUES (3, 'HARINSOWMIYAM', TO\_DATE('1961-05-15', 'YYYY-MM-DD'), 7500, SYSDATE);

INSERT INTO Customers (CustomerID, Name, DOB, Balance, LastModified) VALUES (4, 'KEERTHI S', TO\_DATE('1985-11-25', 'YYYY-MM-DD'), 12000, SYSDATE);

INSERT INTO Customers (CustomerID, Name, DOB, Balance, LastModified) VALUES (5, 'NAGLAKSHMI G', TO\_DATE('1982-03-10', 'YYYY-MM-DD'), 3000, SYSDATE);

INSERT INTO Customers (CustomerID, Name, DOB, Balance, LastModified) VALUES (6, 'POORNIMA K', TO\_DATE('1988-09-08', 'YYYY-MM-DD'), 4000, SYSDATE);

INSERT INTO Customers (CustomerID, Name, DOB, Balance, LastModified) VALUES (7, 'RESHMIKA K S', TO\_DATE('1993-01-17', 'YYYY-MM-DD'), 6000, SYSDATE);

INSERT INTO Customers (CustomerID, Name, DOB, Balance, LastModified) VALUES (8, 'SNEHAA S', TO\_DATE('1987-10-02', 'YYYY-MM-DD'), 7000, SYSDATE);

INSERT INTO Customers (CustomerID, Name, DOB, Balance, LastModified) VALUES (9, 'ABIKANNAN P R', TO\_DATE('1984-06-20', 'YYYY-MM-DD'), 8000, SYSDATE);

INSERT INTO Customers (CustomerID, Name, DOB, Balance, LastModified) VALUES (10, 'ARRCHIT RAMANA M S', TO\_DATE('1991-08-15', 'YYYY-MM-DD'), 9500, SYSDATE);

INSERT INTO Customers (CustomerID, Name, DOB, Balance, LastModified) VALUES (11, 'BALASUBRAMANI T', TO\_DATE('1978-02-10', 'YYYY-MM-DD'), 11000, SYSDATE);

INSERT INTO Customers (CustomerID, Name, DOB, Balance, LastModified) VALUES (12, 'GOURAV PRITHAM G R', TO\_DATE('1989-04-25', 'YYYY-MM-DD'), 3500, SYSDATE);

INSERT INTO Customers (CustomerID, Name, DOB, Balance, LastModified) VALUES (13, 'GOWTHAM S P', TO\_DATE('1986-07-30', 'YYYY-MM-DD'), 5000, SYSDATE);

INSERT INTO Customers (CustomerID, Name, DOB, Balance, LastModified) VALUES (14, 'HARIPRASATH N', TO\_DATE('1981-11-11', 'YYYY-MM-DD'), 2500, SYSDATE);

INSERT INTO Customers (CustomerID, Name, DOB, Balance, LastModified) VALUES (15, 'MANIVELAN K', TO\_DATE('1994-09-03', 'YYYY-MM-DD'), 9000, SYSDATE);

INSERT INTO Customers (CustomerID, Name, DOB, Balance, LastModified) VALUES (16, 'MOHAMED KANI H', TO\_DATE('1983-12-22', 'YYYY-MM-DD'), 10000, SYSDATE);

INSERT INTO Customers (CustomerID, Name, DOB, Balance, LastModified) VALUES (17, 'SANTHOSH L', TO\_DATE('1987-05-13', 'YYYY-MM-DD'), 6000, SYSDATE);

INSERT INTO Customers (CustomerID, Name, DOB, Balance, LastModified) VALUES (18, 'SHYAM SUNDAR P S', TO\_DATE('1990-06-08', 'YYYY-MM-DD'), 12000, SYSDATE);

INSERT INTO Customers (CustomerID, Name, DOB, Balance, LastModified) VALUES (19, 'VIGHRANTH T', TO\_DATE('1985-01-25', 'YYYY-MM-DD'), 3000, SYSDATE);

INSERT INTO Customers (CustomerID, Name, DOB, Balance, LastModified) VALUES (20, 'PRITHIVI RAJ S D', TO\_DATE('1992-07-12', 'YYYY-MM-DD'), 8500, SYSDATE);

INSERT INTO Customers (CustomerID, Name, DOB, Balance, LastModified) VALUES (21, 'SANTHOSH KUMAR V', TO\_DATE('1989-10-30', 'YYYY-MM-DD'), 11500, SYSDATE);

COMMIT;

END;

/

****

**Inserting values into Accounts table**

BEGIN

INSERT INTO Accounts (AccountID, CustomerID, AccountType, Balance, LastModified) VALUES (1, 1, 'Checking', 2000, SYSDATE);

INSERT INTO Accounts (AccountID, CustomerID, AccountType, Balance, LastModified) VALUES (2, 2, 'Savings', 5000, SYSDATE);

INSERT INTO Accounts (AccountID, CustomerID, AccountType, Balance, LastModified) VALUES (3, 3, 'Checking', 1500, SYSDATE);

INSERT INTO Accounts (AccountID, CustomerID, AccountType, Balance, LastModified) VALUES (4, 4, 'Savings', 8000, SYSDATE);

INSERT INTO Accounts (AccountID, CustomerID, AccountType, Balance, LastModified) VALUES (5, 5, 'Checking', 1200, SYSDATE);

INSERT INTO Accounts (AccountID, CustomerID, AccountType, Balance, LastModified) VALUES (6, 6, 'Savings', 2500, SYSDATE);

INSERT INTO Accounts (AccountID, CustomerID, AccountType, Balance, LastModified) VALUES (7, 7, 'Checking', 3000, SYSDATE);

INSERT INTO Accounts (AccountID, CustomerID, AccountType, Balance, LastModified) VALUES (8, 8, 'Savings', 4200, SYSDATE);

INSERT INTO Accounts (AccountID, CustomerID, AccountType, Balance, LastModified) VALUES (9, 9, 'Checking', 1800, SYSDATE);

INSERT INTO Accounts (AccountID, CustomerID, AccountType, Balance, LastModified) VALUES (10, 10, 'Savings', 7000, SYSDATE);

INSERT INTO Accounts (AccountID, CustomerID, AccountType, Balance, LastModified) VALUES (11, 11, 'Checking', 3500, SYSDATE);

INSERT INTO Accounts (AccountID, CustomerID, AccountType, Balance, LastModified) VALUES (12, 12, 'Savings', 4000, SYSDATE);

INSERT INTO Accounts (AccountID, CustomerID, AccountType, Balance, LastModified) VALUES (13, 13, 'Checking', 5000, SYSDATE);

INSERT INTO Accounts (AccountID, CustomerID, AccountType, Balance, LastModified) VALUES (14, 14, 'Savings', 2200, SYSDATE);

INSERT INTO Accounts (AccountID, CustomerID, AccountType, Balance, LastModified) VALUES (15, 15, 'Checking', 6000, SYSDATE);

INSERT INTO Accounts (AccountID, CustomerID, AccountType, Balance, LastModified) VALUES (16, 16, 'Savings', 3000, SYSDATE);

INSERT INTO Accounts (AccountID, CustomerID, AccountType, Balance, LastModified) VALUES (17, 17, 'Checking', 2800, SYSDATE);

INSERT INTO Accounts (AccountID, CustomerID, AccountType, Balance, LastModified) VALUES (18, 18, 'Savings', 5500, SYSDATE);

INSERT INTO Accounts (AccountID, CustomerID, AccountType, Balance, LastModified) VALUES (19, 19, 'Checking', 3500, SYSDATE);

INSERT INTO Accounts (AccountID, CustomerID, AccountType, Balance, LastModified) VALUES (20, 20, 'Savings', 6200, SYSDATE);

COMMIT;

END;

/

****

**Inserting values into Transactions table**

BEGIN

INSERT INTO Transactions (TransactionID, AccountID, TransactionDate, Amount, TransactionType) VALUES (1, 1, TO\_DATE('2024-07-15', 'YYYY-MM-DD'), 200, 'Deposit');

INSERT INTO Transactions (TransactionID, AccountID, TransactionDate, Amount, TransactionType) VALUES (2, 1, TO\_DATE('2024-07-16', 'YYYY-MM-DD'), 100, 'Withdrawal');

INSERT INTO Transactions (TransactionID, AccountID, TransactionDate, Amount, TransactionType) VALUES (3, 2, TO\_DATE('2024-07-17', 'YYYY-MM-DD'), 500, 'Deposit');

INSERT INTO Transactions (TransactionID, AccountID, TransactionDate, Amount, TransactionType) VALUES (4, 2, TO\_DATE('2024-07-18', 'YYYY-MM-DD'), 200, 'Withdrawal');

INSERT INTO Transactions (TransactionID, AccountID, TransactionDate, Amount, TransactionType) VALUES (5, 3, TO\_DATE('2024-07-19', 'YYYY-MM-DD'), 300, 'Deposit');

INSERT INTO Transactions (TransactionID, AccountID, TransactionDate, Amount, TransactionType) VALUES (6, 3, TO\_DATE('2024-07-20', 'YYYY-MM-DD'), 150, 'Withdrawal');

INSERT INTO Transactions (TransactionID, AccountID, TransactionDate, Amount, TransactionType) VALUES (7, 4, TO\_DATE('2024-07-21', 'YYYY-MM-DD'), 1000, 'Deposit');

INSERT INTO Transactions (TransactionID, AccountID, TransactionDate, Amount, TransactionType) VALUES (8, 4, TO\_DATE('2024-07-22', 'YYYY-MM-DD'), 500, 'Withdrawal');

INSERT INTO Transactions (TransactionID, AccountID, TransactionDate, Amount, TransactionType) VALUES (9, 5, TO\_DATE('2024-07-23', 'YYYY-MM-DD'), 200, 'Deposit');

INSERT INTO Transactions (TransactionID, AccountID, TransactionDate, Amount, TransactionType) VALUES (10, 5, TO\_DATE('2024-07-24', 'YYYY-MM-DD'), 100, 'Withdrawal');

INSERT INTO Transactions (TransactionID, AccountID, TransactionDate, Amount, TransactionType) VALUES (11, 6, TO\_DATE('2024-07-25', 'YYYY-MM-DD'), 400, 'Deposit');

INSERT INTO Transactions (TransactionID, AccountID, TransactionDate, Amount, TransactionType) VALUES (12, 6, TO\_DATE('2024-07-26', 'YYYY-MM-DD'), 200, 'Withdrawal');

INSERT INTO Transactions (TransactionID, AccountID, TransactionDate, Amount, TransactionType) VALUES (13, 7, TO\_DATE('2024-07-27', 'YYYY-MM-DD'), 600, 'Deposit');

INSERT INTO Transactions (TransactionID, AccountID, TransactionDate, Amount, TransactionType) VALUES (14, 7, TO\_DATE('2024-07-28', 'YYYY-MM-DD'), 300, 'Withdrawal');

INSERT INTO Transactions (TransactionID, AccountID, TransactionDate, Amount, TransactionType) VALUES (15, 8, TO\_DATE('2024-07-29', 'YYYY-MM-DD'), 700, 'Deposit');

INSERT INTO Transactions (TransactionID, AccountID, TransactionDate, Amount, TransactionType) VALUES (16, 8, TO\_DATE('2024-07-30', 'YYYY-MM-DD'), 350, 'Withdrawal');

INSERT INTO Transactions (TransactionID, AccountID, TransactionDate, Amount, TransactionType) VALUES (17, 9, TO\_DATE('2024-07-31', 'YYYY-MM-DD'), 800, 'Deposit');

INSERT INTO Transactions (TransactionID, AccountID, TransactionDate, Amount, TransactionType) VALUES (18, 9, TO\_DATE('2024-08-01', 'YYYY-MM-DD'), 400, 'Withdrawal');

INSERT INTO Transactions (TransactionID, AccountID, TransactionDate, Amount, TransactionType) VALUES (19, 10, TO\_DATE('2024-08-02', 'YYYY-MM-DD'), 900, 'Deposit');

INSERT INTO Transactions (TransactionID, AccountID, TransactionDate, Amount, TransactionType) VALUES (20, 10, TO\_DATE('2024-08-03', 'YYYY-MM-DD'), 450, 'Withdrawal');

COMMIT;

END;

/

****

**Inserting values into Loans table**

BEGIN

INSERT INTO Loans (LoanID, CustomerID, LoanAmount, InterestRate, StartDate, EndDate) VALUES

(1, 1, 50000, 0.05, TO\_DATE('2023-01-01', 'YYYY-MM-DD'), TO\_DATE('2024-01-01', 'YYYY-MM-DD'));

INSERT INTO Loans (LoanID, CustomerID, LoanAmount, InterestRate, StartDate, EndDate) VALUES

(2, 2, 75000, 0.04, TO\_DATE('2022-05-01', 'YYYY-MM-DD'), TO\_DATE('2024-05-01', 'YYYY-MM-DD'));

INSERT INTO Loans (LoanID, CustomerID, LoanAmount, InterestRate, StartDate, EndDate) VALUES

(3, 3, 30000, 0.06, TO\_DATE('2023-08-01', 'YYYY-MM-DD'), TO\_DATE('2025-08-01', 'YYYY-MM-DD'));

INSERT INTO Loans (LoanID, CustomerID, LoanAmount, InterestRate, StartDate, EndDate) VALUES

(4, 4, 45000, 0.05, TO\_DATE('2023-03-01', 'YYYY-MM-DD'), TO\_DATE('2024-03-01', 'YYYY-MM-DD'));

INSERT INTO Loans (LoanID, CustomerID, LoanAmount, InterestRate, StartDate, EndDate) VALUES

(5, 5, 20000, 0.07, TO\_DATE('2022-11-01', 'YYYY-MM-DD'), TO\_DATE('2024-11-01', 'YYYY-MM-DD'));

INSERT INTO Loans (LoanID, CustomerID, LoanAmount, InterestRate, StartDate, EndDate) VALUES

(6, 6, 60000, 0.05, TO\_DATE('2023-07-01', 'YYYY-MM-DD'), TO\_DATE('2025-07-01', 'YYYY-MM-DD'));

INSERT INTO Loans (LoanID, CustomerID, LoanAmount, InterestRate, StartDate, EndDate) VALUES

(7, 7, 35000, 0.06, TO\_DATE('2023-02-15', 'YYYY-MM-DD'), TO\_DATE('2024-02-15', 'YYYY-MM-DD'));

INSERT INTO Loans (LoanID, CustomerID, LoanAmount, InterestRate, StartDate, EndDate) VALUES

(8, 8, 55000, 0.05, TO\_DATE('2023-04-20', 'YYYY-MM-DD'), TO\_DATE('2025-04-20', 'YYYY-MM-DD'));

INSERT INTO Loans (LoanID, CustomerID, LoanAmount, InterestRate, StartDate, EndDate) VALUES

(9, 9, 25000, 0.07, TO\_DATE('2022-12-10', 'YYYY-MM-DD'), TO\_DATE('2024-12-10', 'YYYY-MM-DD'));

INSERT INTO Loans (LoanID, CustomerID, LoanAmount, InterestRate, StartDate, EndDate) VALUES

(10, 10, 40000, 0.06, TO\_DATE('2023-06-05', 'YYYY-MM-DD'), TO\_DATE('2024-06-05', 'YYYY-MM-DD'));

INSERT INTO Loans (LoanID, CustomerID, LoanAmount, InterestRate, StartDate, EndDate) VALUES

(11, 11, 70000, 0.04, TO\_DATE('2023-01-15', 'YYYY-MM-DD'), TO\_DATE('2024-01-15', 'YYYY-MM-DD'));

INSERT INTO Loans (LoanID, CustomerID, LoanAmount, InterestRate, StartDate, EndDate) VALUES

(12, 12, 30000, 0.05, TO\_DATE('2022-09-01', 'YYYY-MM-DD'), TO\_DATE('2024-09-01', 'YYYY-MM-DD'));

INSERT INTO Loans (LoanID, CustomerID, LoanAmount, InterestRate, StartDate, EndDate) VALUES

(13, 13, 45000, 0.06, TO\_DATE('2023-11-01', 'YYYY-MM-DD'), TO\_DATE('2025-11-01', 'YYYY-MM-DD'));

INSERT INTO Loans (LoanID, CustomerID, LoanAmount, InterestRate, StartDate, EndDate) VALUES

(14, 14, 20000, 0.07, TO\_DATE('2023-05-10', 'YYYY-MM-DD'), TO\_DATE('2024-05-10', 'YYYY-MM-DD'));

INSERT INTO Loans (LoanID, CustomerID, LoanAmount, InterestRate, StartDate, EndDate) VALUES

(15, 15, 50000, 0.06, TO\_DATE('2023-03-15', 'YYYY-MM-DD'), TO\_DATE('2024-03-15', 'YYYY-MM-DD'));

INSERT INTO Loans (LoanID, CustomerID, LoanAmount, InterestRate, StartDate, EndDate) VALUES

(16, 16, 65000, 0.05, TO\_DATE('2023-08-01', 'YYYY-MM-DD'), TO\_DATE('2025-08-01', 'YYYY-MM-DD'));

INSERT INTO Loans (LoanID, CustomerID, LoanAmount, InterestRate, StartDate, EndDate) VALUES

(17, 17, 30000, 0.06, TO\_DATE('2022-12-01', 'YYYY-MM-DD'), TO\_DATE('2024-12-01', 'YYYY-MM-DD'));

INSERT INTO Loans (LoanID, CustomerID, LoanAmount, InterestRate, StartDate, EndDate) VALUES

(18, 18, 40000, 0.05, TO\_DATE('2023-07-15', 'YYYY-MM-DD'), TO\_DATE('2024-07-15', 'YYYY-MM-DD'));

INSERT INTO Loans (LoanID, CustomerID, LoanAmount, InterestRate, StartDate, EndDate) VALUES

(19, 19, 25000, 0.07, TO\_DATE('2023-04-05', 'YYYY-MM-DD'), TO\_DATE('2024-04-05', 'YYYY-MM-DD'));

INSERT INTO Loans (LoanID, CustomerID, LoanAmount, InterestRate, StartDate, EndDate) VALUES

(20, 20, 55000, 0.06, TO\_DATE('2023-09-10', 'YYYY-MM-DD'), TO\_DATE('2025-09-10', 'YYYY-MM-DD'));

INSERT INTO Loans (LoanID, CustomerID, LoanAmount, InterestRate, StartDate, EndDate) VALUES

(21, 21, 50000, 0.05, TO\_DATE('2023-02-20', 'YYYY-MM-DD'), TO\_DATE('2024-02-20', 'YYYY-MM-DD'));

COMMIT;

END;

/

****

**Inserting values into Employees table**

BEGIN

INSERT INTO Employees (EmployeeID, Name, Position, Salary, Department, HireDate) VALUES (1, 'BARANI SRI K', 'Manager', 75000, 'Sales', TO\_DATE('2020-01-15', 'YYYY-MM-DD'));

INSERT INTO Employees (EmployeeID, Name, Position, Salary, Department, HireDate) VALUES (2, 'EARLENE MELBA J', 'Assistant Manager', 60000, 'Marketing', TO\_DATE('2019-03-22', 'YYYY-MM-DD'));

INSERT INTO Employees (EmployeeID, Name, Position, Salary, Department, HireDate) VALUES (3, 'HARINI SRI T R', 'Senior Developer', 85000, 'IT', TO\_DATE('2018-07-10', 'YYYY-MM-DD'));

INSERT INTO Employees (EmployeeID, Name, Position, Salary, Department, HireDate) VALUES (4, 'HARIVARSHINI M', 'HR Specialist', 55000, 'Human Resources', TO\_DATE('2021-02-01', 'YYYY-MM-DD'));

INSERT INTO Employees (EmployeeID, Name, Position, Salary, Department, HireDate) VALUES (5, 'KARTHIYAIYINI G', 'Financial Analyst', 72000, 'Finance', TO\_DATE('2017-11-30', 'YYYY-MM-DD'));

INSERT INTO Employees (EmployeeID, Name, Position, Salary, Department, HireDate) VALUES (6, 'KEERTHA DHARSHINI S', 'Marketing Coordinator', 65000, 'Marketing', TO\_DATE('2022-04-25', 'YYYY-MM-DD'));

INSERT INTO Employees (EmployeeID, Name, Position, Salary, Department, HireDate) VALUES (7, 'NIVEATHA N', 'Customer Service Representative', 50000, 'Customer Service', TO\_DATE('2021-08-15', 'YYYY-MM-DD'));

INSERT INTO Employees (EmployeeID, Name, Position, Salary, Department, HireDate) VALUES (8, 'YOGITA C M', 'Project Manager', 78000, 'Project Management', TO\_DATE('2019-06-20', 'YYYY-MM-DD'));

INSERT INTO Employees (EmployeeID, Name, Position, Salary, Department, HireDate) VALUES (9, 'CHAKKARAVARTHI N', 'Business Analyst', 67000, 'Business Analysis', TO\_DATE('2018-12-05', 'YYYY-MM-DD'));

INSERT INTO Employees (EmployeeID, Name, Position, Salary, Department, HireDate) VALUES (10, 'JAI NITHISH N', 'IT Support Specialist', 54000, 'IT', TO\_DATE('2020-10-12', 'YYYY-MM-DD'));

INSERT INTO Employees (EmployeeID, Name, Position, Salary, Department, HireDate) VALUES (11, 'JEYAKUMAR N K', 'Senior Accountant', 74000, 'Finance', TO\_DATE('2017-09-01', 'YYYY-MM-DD'));

INSERT INTO Employees (EmployeeID, Name, Position, Salary, Department, HireDate) VALUES (12, 'KABISH M', 'Data Scientist', 82000, 'Data Science', TO\_DATE('2021-01-10', 'YYYY-MM-DD'));

INSERT INTO Employees (EmployeeID, Name, Position, Salary, Department, HireDate) VALUES (13, 'LINGESH KUMAR K', 'Operations Manager', 77000, 'Operations', TO\_DATE('2019-11-25', 'YYYY-MM-DD'));

INSERT INTO Employees (EmployeeID, Name, Position, Salary, Department, HireDate) VALUES (14, 'MOHAMMED HAMZA M', 'Graphic Designer', 56000, 'Design', TO\_DATE('2022-03-05', 'YYYY-MM-DD'));

INSERT INTO Employees (EmployeeID, Name, Position, Salary, Department, HireDate) VALUES (15, 'PAWAN PRASATH SM', 'Sales Executive', 62000, 'Sales', TO\_DATE('2020-07-30', 'YYYY-MM-DD'));

INSERT INTO Employees (EmployeeID, Name, Position, Salary, Department, HireDate) VALUES (16, 'PRIYADHARSAN S', 'Supply Chain Analyst', 70000, 'Supply Chain', TO\_DATE('2018-06-15', 'YYYY-MM-DD'));

INSERT INTO Employees (EmployeeID, Name, Position, Salary, Department, HireDate) VALUES (17, 'RAMANATHAN R', 'Legal Advisor', 68000, 'Legal', TO\_DATE('2021-05-20', 'YYYY-MM-DD'));

INSERT INTO Employees (EmployeeID, Name, Position, Salary, Department, HireDate) VALUES (18, 'VENKATESH PRABU R', 'Chief Technology Officer', 90000, 'Technology', TO\_DATE('2017-04-10', 'YYYY-MM-DD'));

INSERT INTO Employees (EmployeeID, Name, Position, Salary, Department, HireDate) VALUES (19, 'SRIRAM BALAJI R', 'Network Engineer', 62000, 'IT', TO\_DATE('2019-12-01', 'YYYY-MM-DD'));

COMMIT;

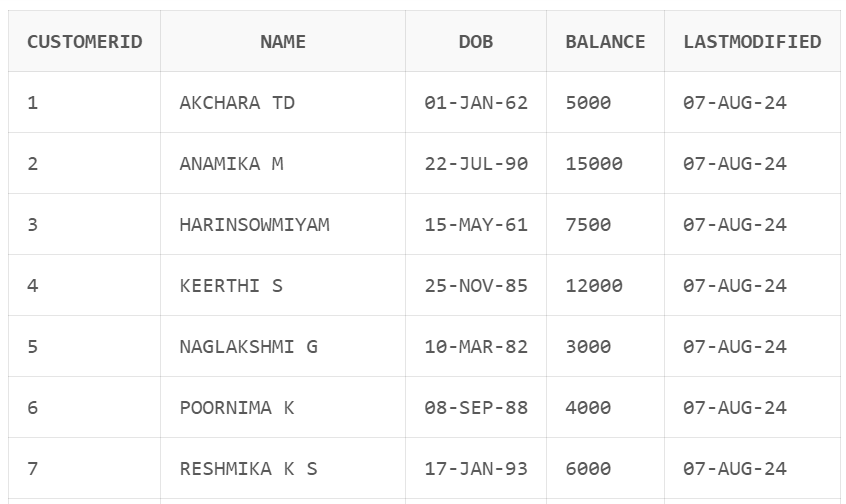
END;

/

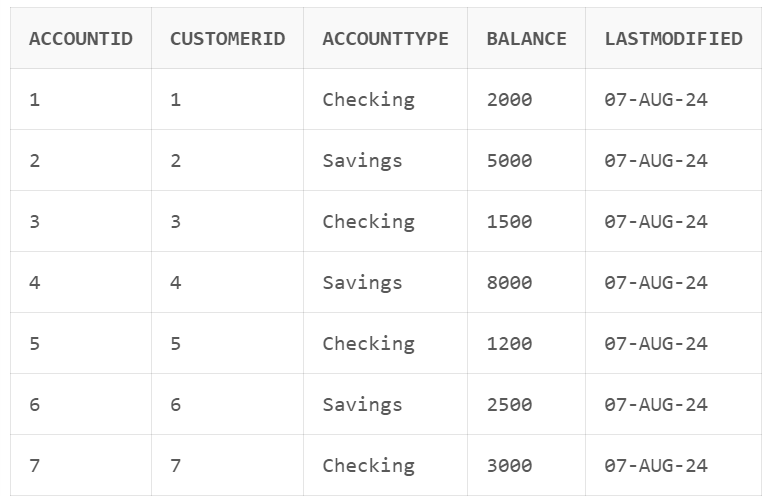
****

**Retrieving Records from the tables:**

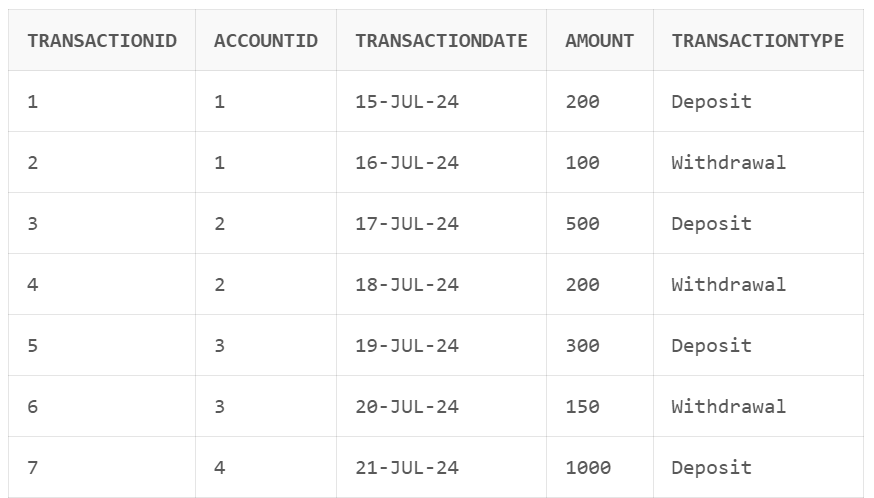
**Select \* from Customers;**

****

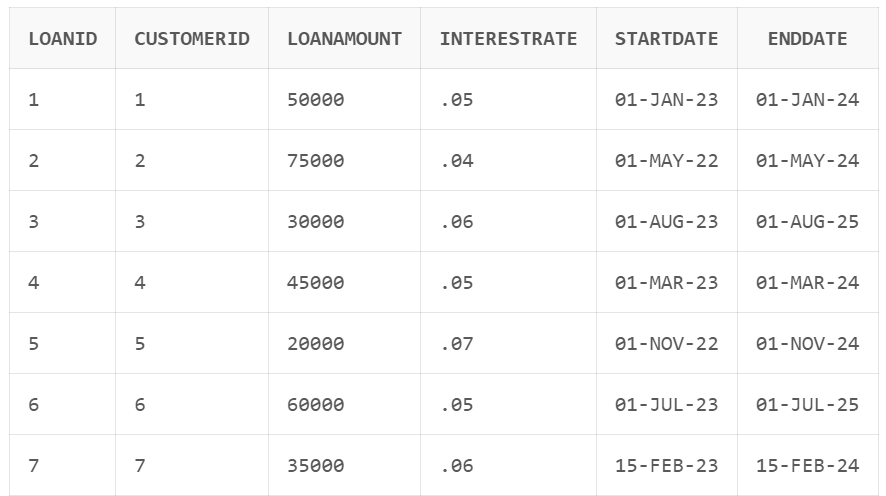
**Select \* from Accounts;**

****

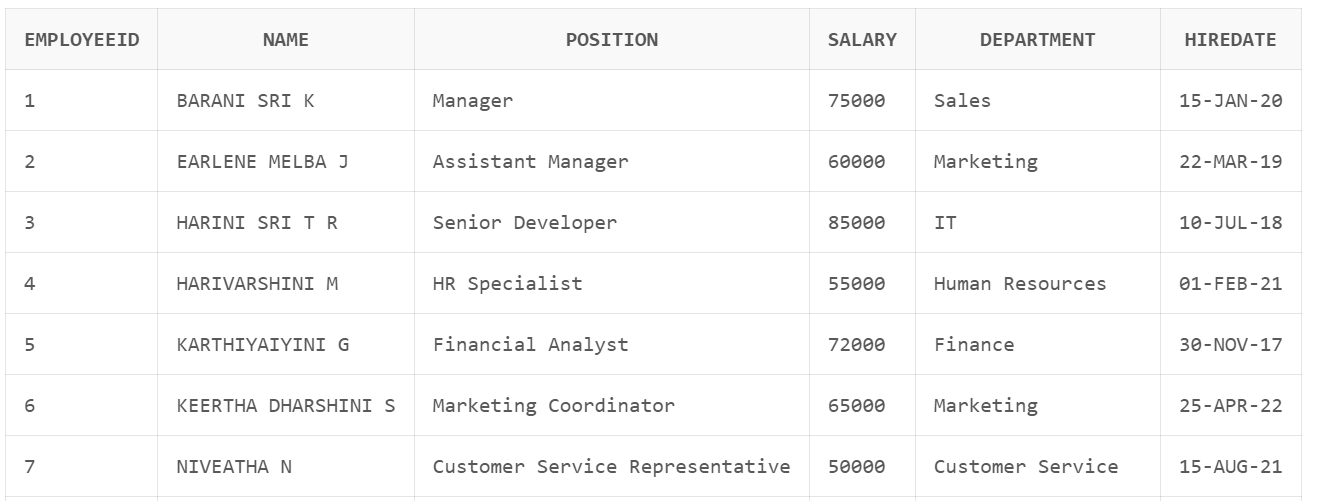
**Select \* from Transactions;**

****

**Select \* from Loans;**

****

**Select \* from Employees;**

****

**Exercise 1: Control Structures**

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

* + **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.

DECLARE

v\_age NUMBER;

BEGIN

FOR cust\_rec IN (SELECT CustomerID, Name, DOB FROM Customers) LOOP

v\_age := FLOOR(MONTHS\_BETWEEN(SYSDATE, cust\_rec.DOB) / 12);

-- Checking if the customer is older than 60 years

IF v\_age > 60 THEN

UPDATE Loans

SET InterestRate = InterestRate - 0.01

WHERE CustomerID = cust\_rec.CustomerID;

DBMS\_OUTPUT.PUT\_LINE('Applied 1% discount to loan interest rate for : ' || cust\_rec.Name);

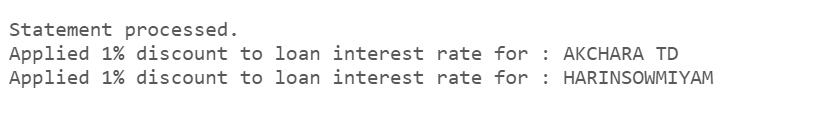
END IF;

END LOOP;

COMMIT;

END;

/

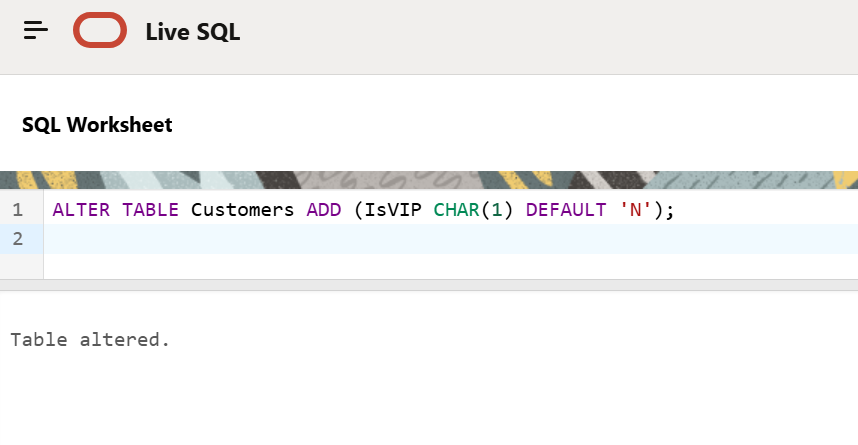


**Figure 1.1: Scenario 1 – Applying 1% discount for Customers whose age is above 60**

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

* + **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.

Before writing PL/SQL block for the given scenario let me alter the table to have an additional attribute called “IsVIP”.



**Figure 1.2: Altering Customer Table by adding IsVIP attribute**

BEGIN

FOR cust\_rec IN (SELECT CustomerID, Name, Balance FROM Customers) LOOP

-- Checking if the customer's balance is over $10,000

IF cust\_rec.Balance > 10000 THEN

-- Update the IsVIP flag to 'Y' for this customer

UPDATE Customers

SET IsVIP = 'Y'

WHERE CustomerID = cust\_rec.CustomerID;

-- Output the change

DBMS\_OUTPUT.PUT\_LINE(cust\_rec.Name ||' - Promoted to VIP status.');

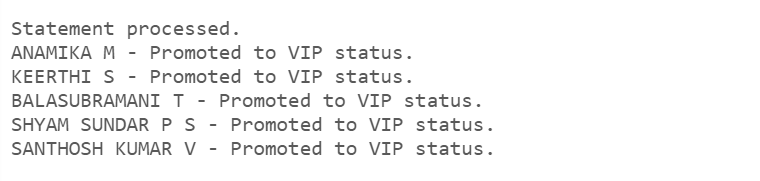
END IF;

END LOOP;

COMMIT;

END;

/



**Figure 1.3: Scenario 2 – Promoting Customers to VIP Status**

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

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

DECLARE

CURSOR loan\_cursor IS

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

FROM Loans l

JOIN Customers c ON l.CustomerID = c.CustomerID

WHERE l.EndDate BETWEEN SYSDATE AND SYSDATE + 30;

l\_loan\_id Loans.LoanID%TYPE;

l\_customer\_id Customers.CustomerID%TYPE;

l\_end\_date Loans.EndDate%TYPE;

l\_customer\_name Customers.Name%TYPE;

BEGIN

FOR loan\_rec IN loan\_cursor LOOP

l\_loan\_id := loan\_rec.LoanID;

l\_customer\_id := loan\_rec.CustomerID;

l\_end\_date := loan\_rec.EndDate;

l\_customer\_name := loan\_rec.Name;

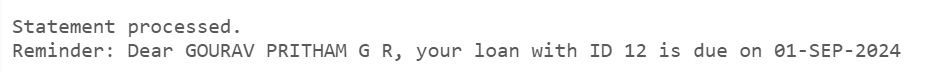
-- Print reminder message

DBMS\_OUTPUT.PUT\_LINE('Reminder: Dear ' || l\_customer\_name || ', your loan with ID ' || l\_loan\_id || ' is due on ' || TO\_CHAR(l\_end\_date, 'DD-MON-YYYY'));

END LOOP;

END;

/

****

**Figure 1.4: Scenario 3 – Sending Reminders to Customers**

**Exercise 2: Error Handling**

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

* + **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.

CREATE OR REPLACE PROCEDURE SafeTransferFunds (

p\_source\_account\_id IN Accounts.AccountID%TYPE,

p\_target\_account\_id IN Accounts.AccountID%TYPE,

p\_amount IN Accounts.Balance%TYPE

)

IS

insufficient\_funds EXCEPTION;

l\_source\_balance Accounts.Balance%TYPE;

l\_target\_balance Accounts.Balance%TYPE;

BEGIN

-- Start the transaction

SAVEPOINT start\_transaction;

-- Fetch the source account balance

SELECT Balance INTO l\_source\_balance

FROM Accounts

WHERE AccountID = p\_source\_account\_id

FOR UPDATE;

-- Check if the source account has sufficient funds

IF l\_source\_balance < p\_amount THEN

RAISE insufficient\_funds;

END IF;

-- Deduct the amount from the source account

UPDATE Accounts

SET Balance = Balance - p\_amount

WHERE AccountID = p\_source\_account\_id;

-- Add the amount to the target account

UPDATE Accounts

SET Balance = Balance + p\_amount

WHERE AccountID = p\_target\_account\_id;

-- Commit the transaction

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('Transfer successful from Account ' || p\_source\_account\_id || ' to Account ' || p\_target\_account\_id || ' for amount ' || p\_amount);

EXCEPTION

WHEN insufficient\_funds THEN

-- Log the error message

DBMS\_OUTPUT.PUT\_LINE('Error: Insufficient funds in Account ' || p\_source\_account\_id);

-- Rollback to the savepoint

ROLLBACK TO start\_transaction;

WHEN OTHERS THEN

-- Handle other exceptions

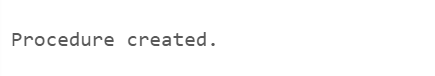
DBMS\_OUTPUT.PUT\_LINE('Error: ' || SQLERRM);

-- Rollback to the savepoint

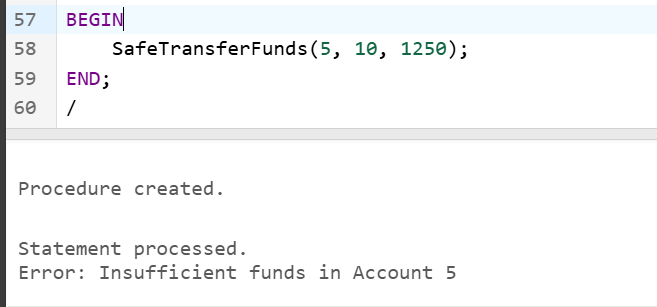
ROLLBACK TO start\_transaction;

END SafeTransferFunds;

/



**Figure 2.1: Scenario 1 – Ensuring Safe Transfer Fund**



**Figure 2.2: Scenario 1 – Trying to transfer amount from Account 5 to 10**

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

* + **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.

CREATE OR REPLACE PROCEDURE UpdateSalary (

p\_emp\_id IN EMPLOYEES.EMPLOYEEID%TYPE,

p\_percentage IN NUMBER

) AS

v\_old\_salary EMPLOYEES.SALARY%TYPE;

BEGIN

-- Select the current salary of the employee

BEGIN

SELECT SALARY INTO v\_old\_salary

FROM EMPLOYEES

WHERE EMPLOYEEID = p\_emp\_id;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

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

RETURN;

END;

-- Update the employee's salary by the given percentage

UPDATE EMPLOYEES

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

WHERE EMPLOYEEID = p\_emp\_id;

COMMIT;

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

EXCEPTION

WHEN OTHERS THEN

DBMS\_OUTPUT.PUT\_LINE('An unexpected error occurred: ' || SQLERRM);

END;

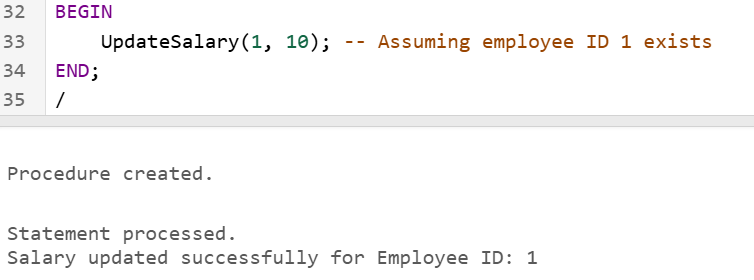
/

BEGIN

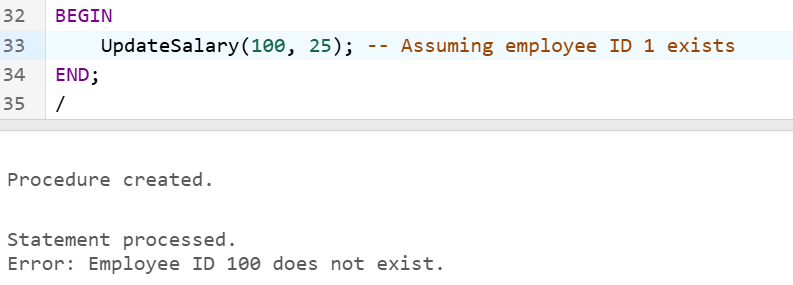
UpdateSalary(100, 25); -- Assuming employee ID 1 exists

END;

/



**Figure 2.3: Scenario 2 – Updating Salary of an existing Employee ID**



**Figure 2.4: Scenario 2 – Trying to Update Salary of a non-existing Employee ID**

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

* + **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.

CREATE OR REPLACE PROCEDURE AddNewCustomer (

p\_customer\_id IN Customers.CustomerID%TYPE,

p\_name IN Customers.Name%TYPE,

p\_dob IN Customers.DOB%TYPE,

p\_balance IN Customers.Balance%TYPE

)

IS

customer\_exists EXCEPTION;

PRAGMA EXCEPTION\_INIT(customer\_exists, -00001); -- Initialize exception for duplicate key

BEGIN

-- Attempt to insert a new customer

BEGIN

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

VALUES (p\_customer\_id, p\_name, p\_dob, p\_balance, SYSDATE);

DBMS\_OUTPUT.PUT\_LINE('Customer added successfully with ID ' || p\_customer\_id);

-- Commit the transaction

COMMIT;

EXCEPTION

WHEN customer\_exists THEN

-- Handle the case where the customer ID already exists

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

-- Rollback the transaction

ROLLBACK;

END;

EXCEPTION

WHEN OTHERS THEN

-- Handle other exceptions

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

-- Rollback the transaction

ROLLBACK;

END AddNewCustomer;

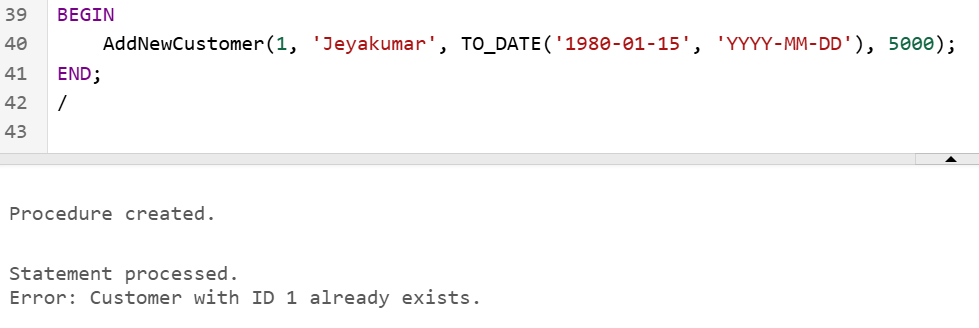
/

BEGIN

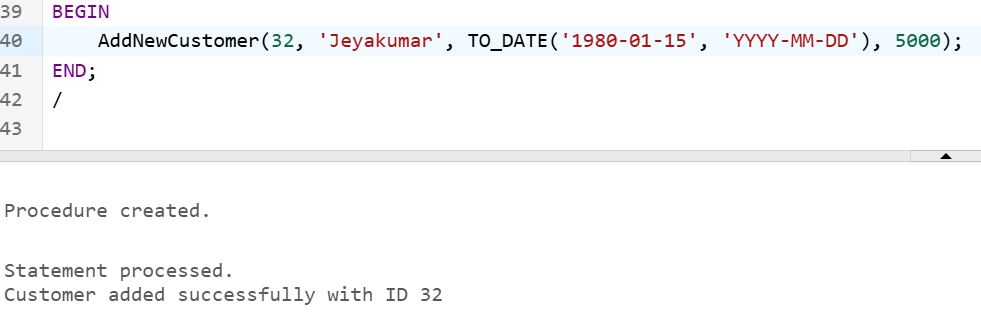
AddNewCustomer(31, 'John', TO\_DATE('1980-01-15', 'YYYY-MM-DD'), 5000);

END;

/



**Figure 2.5: Scenario 3 – Adding a Customer with Duplicate CustomerID**



**Figure 2.6: Scenario 3 – Adding a Customer with Unique CustomerID**

**Exercise 3: Stored Procedures**

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

* + **Question:** Write a stored procedure **ProcessMonthlyInterest** that calculates and updates the balance of all savings accounts by applying an interest rate of 1% to the current balance.

CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest

IS

l\_account\_id Accounts.AccountID%TYPE;

l\_current\_balance Accounts.Balance%TYPE;

l\_new\_balance Accounts.Balance%TYPE;

l\_interest\_rate CONSTANT NUMBER := 0.01; -- 1% interest rate

BEGIN

-- Cursor to select all savings accounts

FOR account\_rec IN (SELECT AccountID, Balance FROM Accounts WHERE AccountType = 'Savings' FOR UPDATE)

LOOP

l\_account\_id := account\_rec.AccountID;

l\_current\_balance := account\_rec.Balance;

-- Calculate the new balance with interest

l\_new\_balance := l\_current\_balance + (l\_current\_balance \* l\_interest\_rate);

-- Update the account balance

UPDATE Accounts

SET Balance = l\_new\_balance,

LastModified = SYSDATE

WHERE AccountID = l\_account\_id;

-- Print a message for each account processed

DBMS\_OUTPUT.PUT\_LINE('Account ID ' || l\_account\_id || ' updated. New Balance: ' || l\_new\_balance);

END LOOP;

-- Commit the transaction

COMMIT;

EXCEPTION

WHEN OTHERS THEN

-- Handle other exceptions

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

-- Rollback the transaction

ROLLBACK;

END ProcessMonthlyInterest;

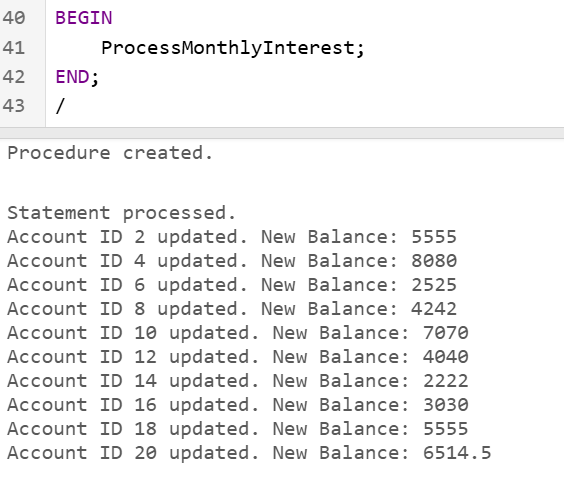
/

BEGIN

ProcessMonthlyInterest;

END;

/



**Figure 3.1: Scenario 1 – Updating Monthly Interest to Account Holders**

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

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

CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus (

p\_department IN Employees.Department%TYPE,

p\_bonus\_percentage IN NUMBER

)

IS

l\_bonus\_amount Employees.Salary%TYPE;

BEGIN

-- Cursor to select all employees in the specified department

FOR employee\_rec IN (SELECT EmployeeID, Salary FROM Employees WHERE Department = p\_department FOR UPDATE)

LOOP

-- Calculate the bonus amount

l\_bonus\_amount := employee\_rec.Salary \* p\_bonus\_percentage / 100;

-- Update the employee's salary with the bonus

UPDATE Employees

SET Salary = Salary + l\_bonus\_amount

WHERE EmployeeID = employee\_rec.EmployeeID;

-- Print a message for each employee processed

DBMS\_OUTPUT.PUT\_LINE('Employee ID ' || employee\_rec.EmployeeID || ' updated. New Salary: ' || (employee\_rec.Salary + l\_bonus\_amount));

END LOOP;

-- Commit the transaction

COMMIT;

EXCEPTION

WHEN OTHERS THEN

-- Handle other exceptions

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

-- Rollback the transaction

ROLLBACK;

END UpdateEmployeeBonus;

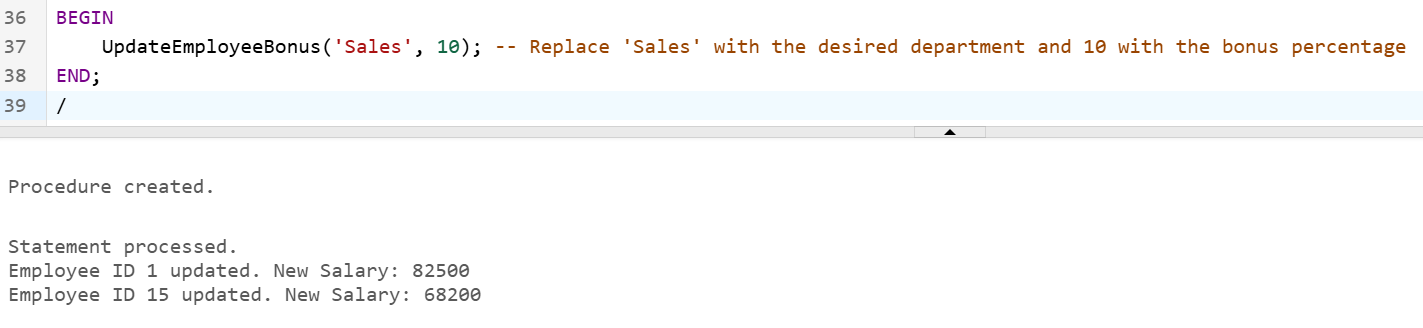
/

BEGIN

UpdateEmployeeBonus('Sales', 10); -- Replace 'Sales' with the desired department and 10 with the bonus percentage

END;

/



**Figure 3.2: Scenario 2 – Updating Employees Salary with Bonus Amount**

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

* + **Question:** Write a stored procedure **TransferFunds** that transfers a specified amount from one account to another, checking that the source account has sufficient balance before making the transfer.

CREATE OR REPLACE PROCEDURE TransferFunds (

p\_source\_account\_id IN Accounts.AccountID%TYPE,

p\_target\_account\_id IN Accounts.AccountID%TYPE,

p\_amount IN Accounts.Balance%TYPE

)

IS

insufficient\_funds EXCEPTION;

l\_source\_balance Accounts.Balance%TYPE;

l\_target\_balance Accounts.Balance%TYPE;

BEGIN

-- Start the transaction

SAVEPOINT start\_transaction;

-- Fetch the source account balance

SELECT Balance INTO l\_source\_balance

FROM Accounts

WHERE AccountID = p\_source\_account\_id

FOR UPDATE;

-- Check if the source account has sufficient funds

IF l\_source\_balance < p\_amount THEN

RAISE insufficient\_funds;

END IF;

-- Deduct the amount from the source account

UPDATE Accounts

SET Balance = Balance - p\_amount

WHERE AccountID = p\_source\_account\_id;

-- Add the amount to the target account

UPDATE Accounts

SET Balance = Balance + p\_amount

WHERE AccountID = p\_target\_account\_id;

-- Commit the transaction

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('Transfer successful from Account ' || p\_source\_account\_id || ' to Account ' || p\_target\_account\_id || ' for amount ' || p\_amount);

EXCEPTION

WHEN insufficient\_funds THEN

-- Log the error message

DBMS\_OUTPUT.PUT\_LINE('Error: Insufficient funds in Account ' || p\_source\_account\_id);

-- Rollback to the savepoint

ROLLBACK TO start\_transaction;

WHEN OTHERS THEN

-- Handle other exceptions

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

-- Rollback to the savepoint

ROLLBACK TO start\_transaction;

END SafeTransferFunds;

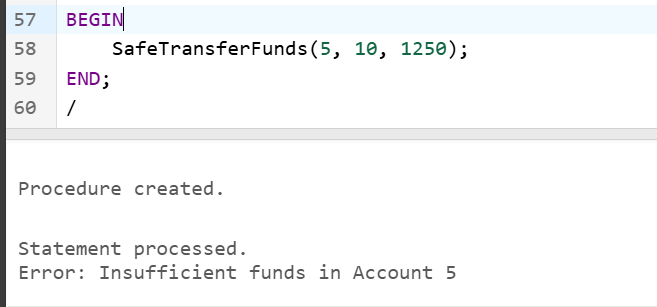
/

BEGIN

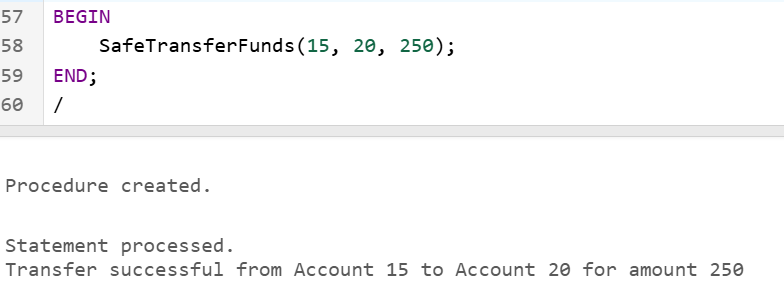
TransferFunds(5, 10, 1250);

END;

/



**Figure 3.3: Scenario 3 – Trying to transfer amount from Account 5 to 10**

****

**Figure 3.4: Scenario 3 – Trying to transfer amount from Account 15 to 20**

**Exercise 4: Functions**

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

* + **Question:** Write a function CalculateAge that takes a customer's date of birth as input and returns their age in years.

CREATE OR REPLACE FUNCTION CalculateAge (

p\_dob IN DATE

) RETURN NUMBER

IS

l\_age NUMBER;

BEGIN

-- Calculate the age in years

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

RETURN l\_age;

EXCEPTION

WHEN OTHERS THEN

-- Handle other exceptions

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

RETURN NULL;

END CalculateAge;

/

DECLARE

v\_dob DATE;

v\_age NUMBER;

BEGIN

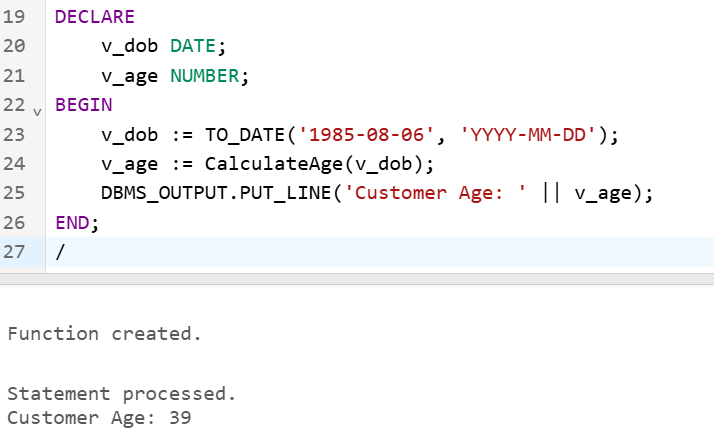
v\_dob := TO\_DATE('1985-08-06', 'YYYY-MM-DD');

v\_age := CalculateAge(v\_dob);

DBMS\_OUTPUT.PUT\_LINE('Customer Age: ' || v\_age);

END;

/



**Figure 4.1: Scenario 1 – Calculating Customer’s Age**

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

* + **Question:** Write a function **CalculateMonthlyInstallment** that takes the loan amount, interest rate, and loan duration in years as input and returns the monthly installment amount.

CREATE OR REPLACE FUNCTION CalculateMonthlyInstallment (

p\_loan\_amount IN NUMBER,

p\_annual\_interest\_rate IN NUMBER,

p\_loan\_duration\_years IN NUMBER

) RETURN NUMBER

IS

l\_monthly\_interest\_rate NUMBER;

l\_total\_payments NUMBER;

l\_monthly\_installment NUMBER;

BEGIN

-- Convert annual interest rate to monthly interest rate

l\_monthly\_interest\_rate := p\_annual\_interest\_rate / 12 / 100;

-- Calculate the total number of payments

l\_total\_payments := p\_loan\_duration\_years \* 12;

-- Calculate the monthly installment using the loan amortization formula

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

(1 - POWER(1 + l\_monthly\_interest\_rate, -l\_total\_payments));

RETURN l\_monthly\_installment;

EXCEPTION

WHEN OTHERS THEN

-- Handle other exceptions

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

RETURN NULL;

END CalculateMonthlyInstallment;

/

DECLARE

v\_loan\_amount NUMBER := 100000;

v\_annual\_interest\_rate NUMBER := 5;

v\_loan\_duration\_years NUMBER := 10;

v\_monthly\_installment NUMBER;

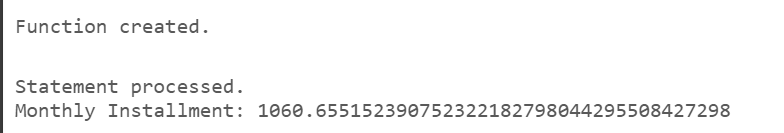
BEGIN

v\_monthly\_installment := CalculateMonthlyInstallment(v\_loan\_amount, v\_annual\_interest\_rate, v\_loan\_duration\_years);

DBMS\_OUTPUT.PUT\_LINE('Monthly Installment: ' || v\_monthly\_installment);

END;

/



**Figure 4.2: Scenario 2 – Calculating Monthly Installment**

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

* + **Question:** Write a function **HasSufficientBalance** that takes an account ID and an amount as input and returns a boolean indicating whether the account has at least the specified amount.

CREATE OR REPLACE FUNCTION HasSufficientBalance (

p\_account\_id IN Accounts.AccountID%TYPE,

p\_amount IN NUMBER

) RETURN BOOLEAN

IS

l\_balance Accounts.Balance%TYPE;

BEGIN

-- Fetch the balance of the specified account

SELECT Balance INTO l\_balance

FROM Accounts

WHERE AccountID = p\_account\_id;

-- Compare the balance with the specified amount

IF l\_balance >= p\_amount THEN

RETURN TRUE;

ELSE

RETURN FALSE;

END IF;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

-- Handle account not found case

DBMS\_OUTPUT.PUT\_LINE('Error: Account ID ' || p\_account\_id || ' not found.');

RETURN FALSE;

WHEN OTHERS THEN

-- Handle other exceptions

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

RETURN FALSE;

END HasSufficientBalance;

/

DECLARE

v\_account\_id NUMBER := 15;

v\_amount NUMBER := 2000;

v\_has\_sufficient\_balance BOOLEAN;

BEGIN

v\_has\_sufficient\_balance := HasSufficientBalance(v\_account\_id, v\_amount);

IF v\_has\_sufficient\_balance THEN

DBMS\_OUTPUT.PUT\_LINE('Account has sufficient balance.');

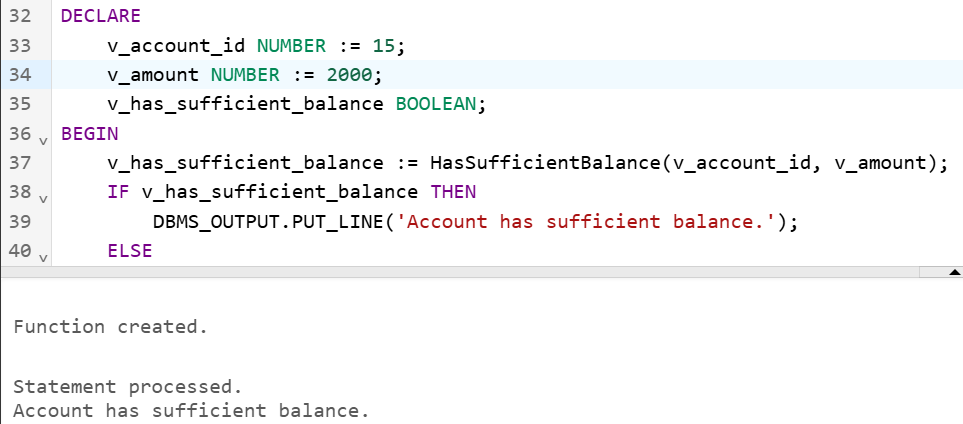
ELSE

DBMS\_OUTPUT.PUT\_LINE('Account does not have sufficient balance.');

END IF;

END;

/

****

**Figure 4.3: Scenario 3 – Checking Minimum Balance of an Account**

**Exercise 5: Triggers**

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

* + **Question:** Write a trigger **UpdateCustomerLastModified** that updates the LastModified column of the Customers table to the current date whenever a customer's record is updated.

CREATE OR REPLACE TRIGGER UpdateCustomerLastModified

BEFORE UPDATE ON Customers

FOR EACH ROW

BEGIN

:NEW.LastModified := SYSDATE;

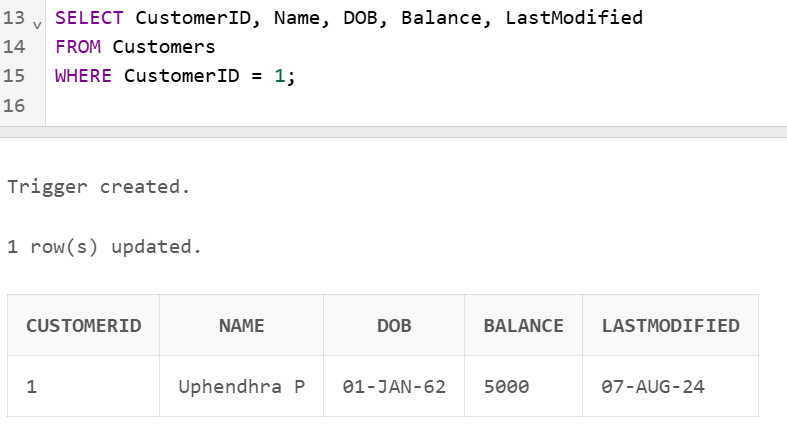
END;

/

UPDATE Customers

SET Name = 'Uphendhra P'

WHERE CustomerID = 1;



**Figure 5.1: Scenario 1 – Last Modified Updated to Current Date**

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

* + **Question:** Write a trigger **LogTransaction** that inserts a record into an AuditLog table whenever a transaction is inserted into the Transactions table.

-- AuditLog table

CREATE TABLE AuditLog (

AuditID INT GENERATED ALWAYS AS IDENTITY PRIMARY KEY,

TransactionID INT,

AccountID INT,

TransactionDate DATE,

Amount INT,

TransactionType VARCHAR(10),

AuditTimestamp TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

Action VARCHAR(10)

);

-- Creating the trigger

CREATE OR REPLACE TRIGGER LogTransaction

AFTER INSERT ON Transactions

FOR EACH ROW

BEGIN

INSERT INTO AuditLog (

TransactionID,

AccountID,

TransactionDate,

Amount,

TransactionType,

Action

)

VALUES (

:NEW.TransactionID,

:NEW.AccountID,

:NEW.TransactionDate,

:NEW.Amount,

:NEW.TransactionType,

'INSERT'

);

END;

/

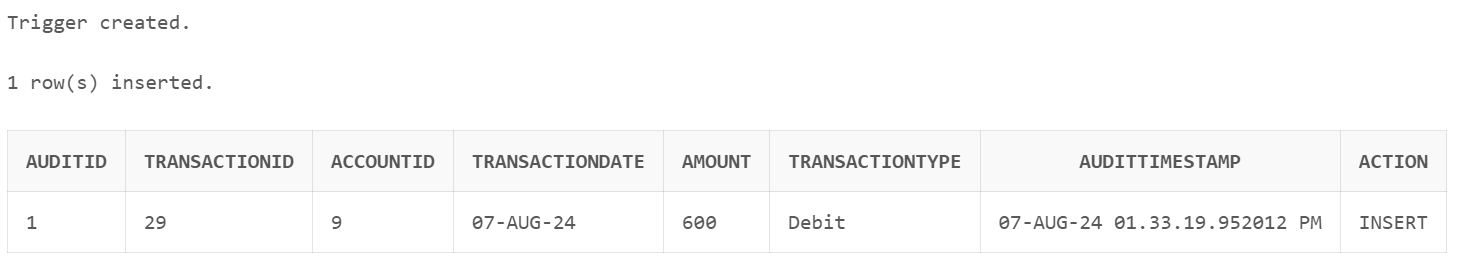
-- Inserting a transaction

INSERT INTO Transactions (TransactionID, AccountID, TransactionDate, Amount, TransactionType)

VALUES (29, 9, SYSDATE, 600, 'Debit');

-- Checking the AuditLog table

SELECT \* FROM AuditLog;

****

**Figure 5.2: Scenario 2 – LogTransaction Trigger to keep track of AuditLogs**

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

* + **Question:** Write a trigger **CheckTransactionRules** that ensures withdrawals do not exceed the balance and deposits are positive before inserting a record into the Transactions table.

CREATE OR REPLACE TRIGGER CheckTransactionRules

BEFORE INSERT ON Transactions

FOR EACH ROW

DECLARE

v\_balance NUMBER;

BEGIN

-- Fetch the current balance of the account

SELECT Balance INTO v\_balance

FROM Accounts

WHERE AccountID = :NEW.AccountID;

-- Check the transaction type and validate accordingly

IF :NEW.TransactionType = 'Withdrawal' THEN

IF :NEW.Amount > v\_balance THEN

RAISE\_APPLICATION\_ERROR(-20001, 'Withdrawal amount exceeds the current balance.');

END IF;

ELSIF :NEW.TransactionType = 'Deposit' THEN

IF :NEW.Amount <= 0 THEN

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

END IF;

ELSE

RAISE\_APPLICATION\_ERROR(-20003, 'Invalid transaction type.');

END IF;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

RAISE\_APPLICATION\_ERROR(-20004, 'Account does not exist.');

WHEN OTHERS THEN

RAISE\_APPLICATION\_ERROR(-20005, 'An unexpected error occurred: ' || SQLERRM);

END;

/

-- Insert valid transactions

INSERT INTO Transactions (TransactionID, AccountID, TransactionDate, Amount, TransactionType)

VALUES (39, 1, SYSDATE, 100, 'Deposit');

INSERT INTO Transactions (TransactionID, AccountID, TransactionDate, Amount, TransactionType)

VALUES (40, 2, SYSDATE, 50, 'Withdrawal');

-- Insert invalid transactions

-- This should raise an error: 'Withdrawal amount exceeds the current balance.'

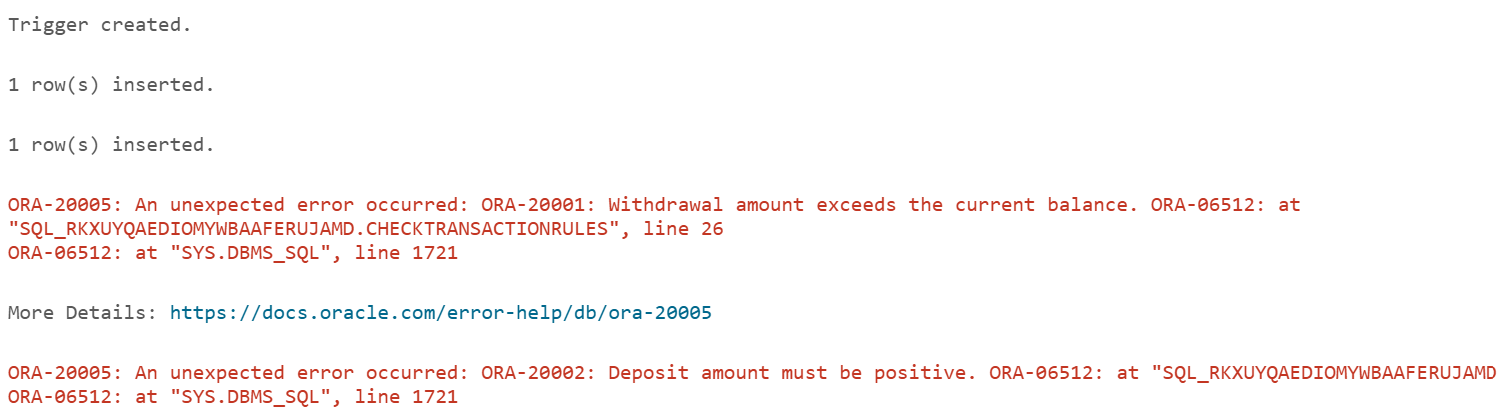
INSERT INTO Transactions (TransactionID, AccountID, TransactionDate, Amount, TransactionType)

VALUES (41, 3, SYSDATE, 10000, 'Withdrawal');

-- This should raise an error: 'Deposit amount must be positive.'

INSERT INTO Transactions (TransactionID, AccountID, TransactionDate, Amount, TransactionType)

VALUES (42, 4, SYSDATE, -50, 'Deposit');

****

**Figure 5.3: Scenario 3 – Validity of Deposits and Withdrawals**

**Exercise 6: Cursors**

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

* + **Question:** Write a PL/SQL block using an explicit cursor **GenerateMonthlyStatements** that retrieves all transactions for the current month and prints a statement for each customer.

DECLARE

CURSOR customer\_cursor IS

SELECT

c.CustomerID,

c.Name,

a.AccountID,

t.TransactionDate,

t.Amount,

t.TransactionType

FROM

Customers c

JOIN Accounts a ON c.CustomerID = a.CustomerID

JOIN Transactions t ON a.AccountID = t.AccountID

WHERE

t.TransactionDate >= TRUNC(SYSDATE, 'MM')

AND t.TransactionDate < TRUNC(SYSDATE, 'MM') + INTERVAL '1' MONTH;

customer\_record customer\_cursor%ROWTYPE;

BEGIN

OPEN customer\_cursor;

LOOP

FETCH customer\_cursor INTO customer\_record;

EXIT WHEN customer\_cursor%NOTFOUND;

DBMS\_OUTPUT.PUT\_LINE('Customer ID: ' || customer\_record.CustomerID);

DBMS\_OUTPUT.PUT\_LINE('Customer Name: ' || customer\_record.Name);

DBMS\_OUTPUT.PUT\_LINE('Account ID: ' || customer\_record.AccountID);

DBMS\_OUTPUT.PUT\_LINE('Transaction Date: ' || customer\_record.TransactionDate);

DBMS\_OUTPUT.PUT\_LINE('Amount: ' || customer\_record.Amount);

DBMS\_OUTPUT.PUT\_LINE('Transaction Type: ' || customer\_record.TransactionType);

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

END LOOP;

CLOSE customer\_cursor;

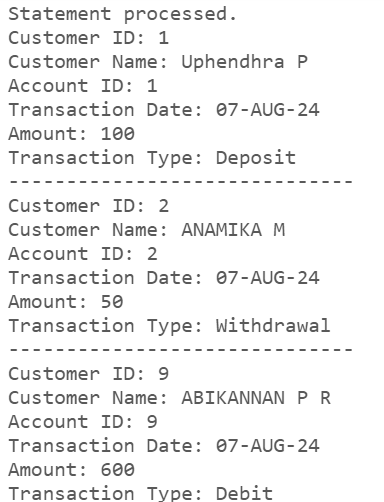
EXCEPTION

WHEN OTHERS THEN

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

END;

/



**Figure 6.1: Scenario 1 – Generating Monthly Statements**

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

* + **Question:** Write a PL/SQL block using an explicit cursor **ApplyAnnualFee** that deducts an annual maintenance fee from the balance of all accounts.

DECLARE

-- Define the annual fee amount

annual\_fee NUMBER := 50;

-- Cursor to fetch all accounts

CURSOR account\_cursor IS

SELECT

AccountID,

Balance

FROM

Accounts;

-- Record type for the cursor

account\_record account\_cursor%ROWTYPE;

BEGIN

OPEN account\_cursor;

LOOP

FETCH account\_cursor INTO account\_record;

EXIT WHEN account\_cursor%NOTFOUND;

UPDATE Accounts

SET Balance = Balance - annual\_fee

WHERE AccountID = account\_record.AccountID;

DBMS\_OUTPUT.PUT\_LINE('Account ID: ' || account\_record.AccountID);

DBMS\_OUTPUT.PUT\_LINE('New Balance: ' || (account\_record.Balance - annual\_fee));

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

END LOOP;

CLOSE account\_cursor;

COMMIT;

EXCEPTION

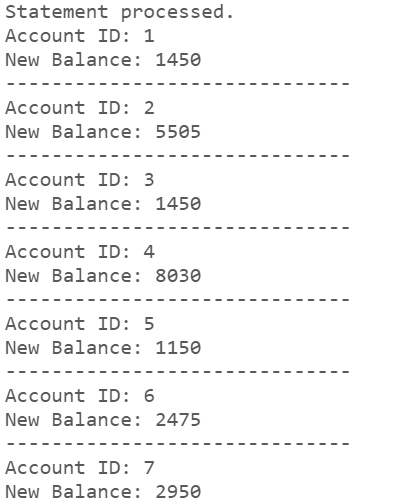
WHEN OTHERS THEN

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

ROLLBACK;

END;

/



**Figure 6.2: Scenario 2 – Deducting Annual Maintenance Fee**

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

* + **Question:** Write a PL/SQL block using an explicit cursor **UpdateLoanInterestRates** that fetches all loans and updates their interest rates based on the new policy.

DECLARE

percentage\_increase NUMBER := 0.02;

-- Cursor to fetch all loans

CURSOR loan\_cursor IS

SELECT

LoanID,

InterestRate

FROM

Loans;

loan\_record loan\_cursor%ROWTYPE;

OPEN loan\_cursor;

LOOP

FETCH loan\_cursor INTO loan\_record;

EXIT WHEN loan\_cursor%NOTFOUND;

DECLARE

new\_interest\_rate NUMBER;

BEGIN

new\_interest\_rate := loan\_record.InterestRate \* (1 + percentage\_increase);

UPDATE Loans

SET InterestRate = new\_interest\_rate

WHERE LoanID = loan\_record.LoanID;

DBMS\_OUTPUT.PUT\_LINE('Loan ID: ' || loan\_record.LoanID);

DBMS\_OUTPUT.PUT\_LINE('New Interest Rate: ' || new\_interest\_rate);

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

END;

END LOOP;

CLOSE loan\_cursor;

COMMIT;

EXCEPTION

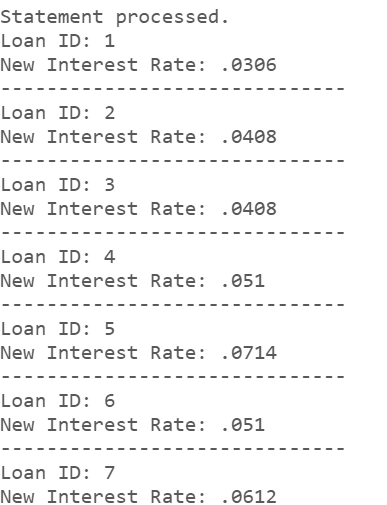
WHEN OTHERS THEN

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

ROLLBACK; -- Rollback changes in case of error

END;

/



**Figure 6.3: Scenario 3 – Updating Loan Interest Rates**

**Exercise 7: Packages**

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

* + **Question:** Create a package **CustomerManagement** with procedures for adding a new customer, updating customer details, and a function to get customer balance.

CREATE OR REPLACE PACKAGE CustomerManagement AS

PROCEDURE AddNewCustomer(

p\_CustomerID IN NUMBER,

p\_Name IN VARCHAR2,

p\_DOB IN DATE,

p\_Balance IN NUMBER

);

PROCEDURE UpdateCustomerDetails(

p\_CustomerID IN NUMBER,

p\_Name IN VARCHAR2,

p\_DOB IN DATE,

p\_Balance IN NUMBER

);

FUNCTION GetCustomerBalance(

p\_CustomerID IN NUMBER

) RETURN NUMBER;

END CustomerManagement;

/

CREATE OR REPLACE PACKAGE BODY CustomerManagement AS

PROCEDURE AddNewCustomer(

p\_CustomerID IN NUMBER,

p\_Name IN VARCHAR2,

p\_DOB IN DATE,

p\_Balance IN NUMBER

) IS

BEGIN

BEGIN

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

VALUES (p\_CustomerID, p\_Name, p\_DOB, p\_Balance, SYSDATE);

COMMIT;

EXCEPTION

WHEN DUP\_VAL\_ON\_INDEX THEN

DBMS\_OUTPUT.PUT\_LINE('Customer ID ' || p\_CustomerID || ' already exists.');

WHEN OTHERS THEN

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

END;

END AddNewCustomer;

-- Implementation of UpdateCustomerDetails procedure

PROCEDURE UpdateCustomerDetails(

p\_CustomerID IN NUMBER,

p\_Name IN VARCHAR2,

p\_DOB IN DATE,

p\_Balance IN NUMBER

) IS

BEGIN

BEGIN

UPDATE Customers

SET Name = p\_Name,

DOB = p\_DOB,

Balance = p\_Balance,

LastModified = SYSDATE

WHERE CustomerID = p\_CustomerID;

IF SQL%ROWCOUNT = 0 THEN

DBMS\_OUTPUT.PUT\_LINE('No customer found with ID ' || p\_CustomerID);

ELSE

COMMIT;

END IF;

EXCEPTION

WHEN OTHERS THEN

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

END;

END UpdateCustomerDetails;

-- Implementation of GetCustomerBalance function

FUNCTION GetCustomerBalance(

p\_CustomerID IN NUMBER

) RETURN NUMBER IS

v\_Balance NUMBER;

BEGIN

BEGIN

SELECT Balance INTO v\_Balance

FROM Customers

WHERE CustomerID = p\_CustomerID;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

DBMS\_OUTPUT.PUT\_LINE('Customer ID ' || p\_CustomerID || ' not found.');

RETURN NULL;

WHEN OTHERS THEN

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

RETURN NULL;

END;

RETURN v\_Balance;

END GetCustomerBalance;

END CustomerManagement;

/

BEGIN

-- Add a new customer

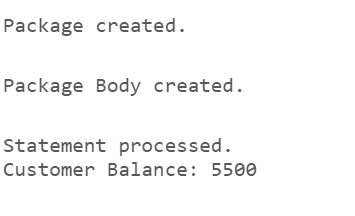
CustomerManagement.AddNewCustomer(50, 'Kumar', DATE '1931-08-15', 5000);

CustomerManagement.UpdateCustomerDetails(50, Ajaykumar, DATE '1931-08-15', 5500);

DBMS\_OUTPUT.PUT\_LINE('Customer Balance: ' || CustomerManagement.GetCustomerBalance(50));

END;

/



**Figure 7.1: Scenario 1 – Customer Management using Packages**

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

* + **Question:** Write a package **EmployeeManagement** with procedures to hire new employees, update employee details, and a function to calculate annual salary.

CREATE OR REPLACE PACKAGE EmployeeManagement AS

PROCEDURE HireEmployee(

p\_EmployeeID IN NUMBER,

p\_Name IN VARCHAR2,

p\_Position IN VARCHAR2,

p\_Salary IN NUMBER,

p\_Department IN VARCHAR2,

p\_HireDate IN DATE

);

PROCEDURE UpdateEmployeeDetails(

p\_EmployeeID IN NUMBER,

p\_Name IN VARCHAR2,

p\_Position IN VARCHAR2,

p\_Salary IN NUMBER,

p\_Department IN VARCHAR2

);

FUNCTION CalculateAnnualSalary(

p\_EmployeeID IN NUMBER

) RETURN NUMBER;

END EmployeeManagement;

/

CREATE OR REPLACE PACKAGE BODY EmployeeManagement AS

PROCEDURE HireEmployee(

p\_EmployeeID IN NUMBER,

p\_Name IN VARCHAR2,

p\_Position IN VARCHAR2,

p\_Salary IN NUMBER,

p\_Department IN VARCHAR2,

p\_HireDate IN DATE

) IS

BEGIN

BEGIN

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

VALUES (p\_EmployeeID, p\_Name, p\_Position, p\_Salary, p\_Department, p\_HireDate);

COMMIT;

EXCEPTION

WHEN DUP\_VAL\_ON\_INDEX THEN

DBMS\_OUTPUT.PUT\_LINE('Employee ID ' || p\_EmployeeID || ' already exists.');

WHEN OTHERS THEN

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

END;

END HireEmployee;

PROCEDURE UpdateEmployeeDetails(

p\_EmployeeID IN NUMBER,

p\_Name IN VARCHAR2,

p\_Position IN VARCHAR2,

p\_Salary IN NUMBER,

p\_Department IN VARCHAR2

) IS

BEGIN

BEGIN

UPDATE Employees

SET Name = p\_Name,

Position = p\_Position,

Salary = p\_Salary,

Department = p\_Department

WHERE EmployeeID = p\_EmployeeID;

IF SQL%ROWCOUNT = 0 THEN

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

ELSE

COMMIT;

END IF;

EXCEPTION

WHEN OTHERS THEN

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

END;

END UpdateEmployeeDetails;

FUNCTION CalculateAnnualSalary(

p\_EmployeeID IN NUMBER

) RETURN NUMBER IS

v\_Salary NUMBER;

BEGIN

BEGIN

SELECT Salary INTO v\_Salary

FROM Employees

WHERE EmployeeID = p\_EmployeeID;

RETURN v\_Salary \* 12;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

DBMS\_OUTPUT.PUT\_LINE('Employee ID ' || p\_EmployeeID || ' not found.');

RETURN NULL;

WHEN OTHERS THEN

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

RETURN NULL;

END;

END CalculateAnnualSalary;

END EmployeeManagement;

/

BEGIN

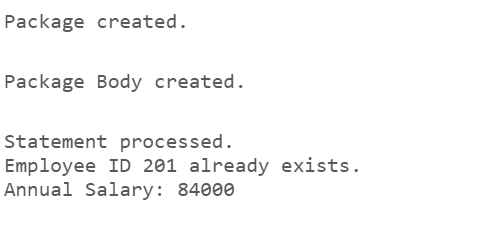
EmployeeManagement.HireEmployee(201, 'Jeyakumar', 'Developer', 6000, 'IT', DATE '2024-08-01');

EmployeeManagement.UpdateEmployeeDetails(201, 'Jeyakumar', 'Senior Developer', 7000, 'IT');

DBMS\_OUTPUT.PUT\_LINE('Annual Salary: ' || EmployeeManagement.CalculateAnnualSalary(201));

END;

/



**Figure 7.2: Scenario 2 – Employee Management using Packages**

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

* + **Question:** Create a package **AccountOperations** with procedures for opening a new account, closing an account, and a function to get the total balance of a customer across all accounts.

CREATE OR REPLACE PACKAGE AccountOperations AS

PROCEDURE OpenAccount(

p\_AccountID IN NUMBER,

p\_CustomerID IN NUMBER,

p\_AccountType IN VARCHAR2,

p\_Balance IN NUMBER

);

PROCEDURE CloseAccount(

p\_AccountID IN NUMBER

);

FUNCTION GetTotalBalance(

p\_CustomerID IN NUMBER

) RETURN NUMBER;

END AccountOperations;

/

CREATE OR REPLACE PACKAGE BODY AccountOperations AS

PROCEDURE OpenAccount(

p\_AccountID IN NUMBER,

p\_CustomerID IN NUMBER,

p\_AccountType IN VARCHAR2,

p\_Balance IN NUMBER

) IS

BEGIN

BEGIN

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

VALUES (p\_AccountID, p\_CustomerID, p\_AccountType, p\_Balance, SYSDATE);

COMMIT;

EXCEPTION

WHEN DUP\_VAL\_ON\_INDEX THEN

DBMS\_OUTPUT.PUT\_LINE('Account ID ' || p\_AccountID || ' already exists.');

WHEN OTHERS THEN

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

END;

END OpenAccount;

PROCEDURE CloseAccount(

p\_AccountID IN NUMBER

) IS

BEGIN

BEGIN

DELETE FROM Accounts

WHERE AccountID = p\_AccountID;

IF SQL%ROWCOUNT = 0 THEN

DBMS\_OUTPUT.PUT\_LINE('No account found with ID ' || p\_AccountID);

ELSE

COMMIT;

END IF;

EXCEPTION

WHEN OTHERS THEN

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

END;

END CloseAccount;

FUNCTION GetTotalBalance(

p\_CustomerID IN NUMBER

) RETURN NUMBER IS

v\_TotalBalance NUMBER;

BEGIN

BEGIN

SELECT SUM(Balance) INTO v\_TotalBalance

FROM Accounts

WHERE CustomerID = p\_CustomerID;

IF v\_TotalBalance IS NULL THEN

RETURN 0;

ELSE

RETURN v\_TotalBalance;

END IF;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

RETURN 0;

WHEN OTHERS THEN

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

RETURN NULL;

END;

END GetTotalBalance;

END AccountOperations;

/

BEGIN

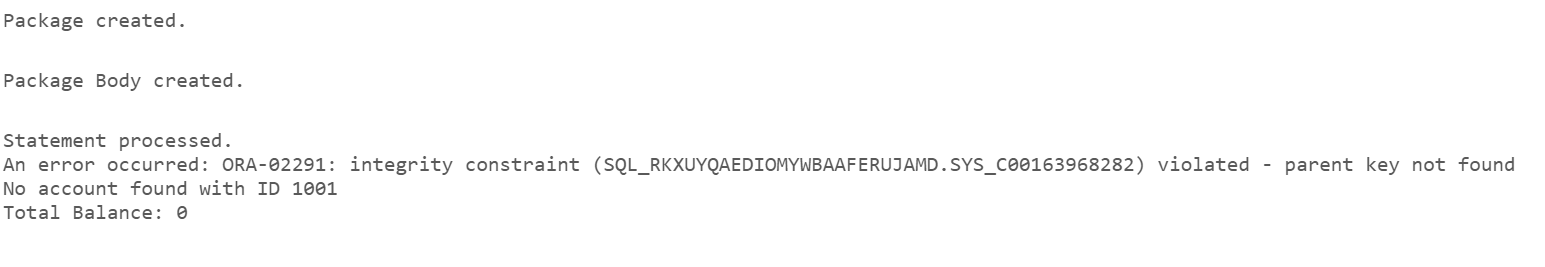
AccountOperations.OpenAccount(1001, 101, 'Savings', 2000);

AccountOperations.CloseAccount(1001);

DBMS\_OUTPUT.PUT\_LINE('Total Balance: ' || AccountOperations.GetTotalBalance(101));

END;

/



**Figure 7.3: Scenario 3 – Account Management using Packages**