Name: Jeyakumar N K
Superset ID: 5008540

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
);
```

```
Table created.
```

```
CREATE TABLE Accounts (
   AccountID NUMBER PRIMARY KEY,
   CustomerID NUMBER,
   AccountType VARCHAR2(20),
   Balance NUMBER,
   LastModified DATE,
   FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)
);
```

```
Table created.
```

```
CREATE TABLE Transactions (
   TransactionID NUMBER PRIMARY KEY,
   AccountID NUMBER,
   TransactionDate DATE,
   Amount NUMBER,
```

```
    TransactionType VARCHAR2(10),
    FOREIGN KEY (AccountID) REFERENCES Accounts(AccountID)
);
```

```
Table created.
```

```
CREATE TABLE Loans (
    LoanID NUMBER PRIMARY KEY,
    CustomerID NUMBER,
    LoanAmount NUMBER,
    InterestRate NUMBER,
    StartDate DATE,
    EndDate DATE,
    FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)
);
```

```
Table created.
```

```
CREATE TABLE Employees (
    EmployeeID NUMBER PRIMARY KEY,
    Name VARCHAR2(100),
    Position VARCHAR2(50),
    Salary NUMBER,
    Department VARCHAR2(50),
    HireDate DATE
);
```

```
Table created.
```

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;
/
```

```
1 row(s) inserted.
```

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;
/
```

```
1 row(s) inserted.
```

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;
/
```

```
1 row(s) inserted.
```

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;
/
```

```
1 row(s) inserted.
```

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;
/
```

```
1 row(s) inserted.
```

Retrieving Records from the tables:

Select * from Customers;

CUSTOMERID	NAME	DOB	BALANCE	LASTMODIFIED
1	AKCHARA TD	01-JAN-62	5000	07-AUG-24
2	ANAMIKA M	22-JUL-90	15000	07-AUG-24
3	HARINSOWMIYAM	15-MAY-61	7500	07-AUG-24
4	KEERTHI S	25-NOV-85	12000	07-AUG-24
5	NAGLAKSHMI G	10-MAR-82	3000	07-AUG-24
6	POORNIMA K	08-SEP-88	4000	07-AUG-24
7	RESHMIKA K S	17-JAN-93	6000	07-AUG-24

Select * from Accounts;

ACCOUNTID	CUSTOMERID	ACCOUNTTYPE	BALANCE	LASTMODIFIED
1	1	Checking	2000	07-AUG-24
2	2	Savings	5000	07-AUG-24
3	3	Checking	1500	07-AUG-24
4	4	Savings	8000	07-AUG-24
5	5	Checking	1200	07-AUG-24
6	6	Savings	2500	07-AUG-24
7	7	Checking	3000	07-AUG-24

Select * from Transactions;

TRANSACTIONID	ACCOUNTID	TRANSACTIONDATE	AMOUNT	TRANSACTIONTYPE
1	1	15-JUL-24	200	Deposit
2	1	16-JUL-24	100	Withdrawal
3	2	17-JUL-24	500	Deposit
4	2	18-JUL-24	200	Withdrawal
5	3	19-JUL-24	300	Deposit
6	3	20-JUL-24	150	Withdrawal
7	4	21-JUL-24	1000	Deposit

Select * from Loans;

LOANID	CUSTOMERID	LOANAMOUNT	INTERESTRATE	STARTDATE	ENDDATE
1	1	50000	.05	01-JAN-23	01-JAN-24
2	2	75000	.04	01-MAY-22	01-MAY-24
3	3	30000	.06	01-AUG-23	01-AUG-25
4	4	45000	.05	01-MAR-23	01-MAR-24
5	5	20000	.07	01-NOV-22	01-NOV-24
6	6	60000	.05	01-JUL-23	01-JUL-25
7	7	35000	.06	15-FEB-23	15-FEB-24

Select * from Employees;

EMPLOYEEID	NAME	POSITION	SALARY	DEPARTMENT	HIREDATE
1	BARANI SRI K	Manager	75000	Sales	15-JAN-20
2	EARLENE MELBA J	Assistant Manager	60000	Marketing	22-MAR-19
3	HARINI SRI T R	Senior Developer	85000	IT	10-JUL-18
4	HARIVARSHINI M	HR Specialist	55000	Human Resources	01-FEB-21
5	KARTHIYAIYINI G	Financial Analyst	72000	Finance	30-NOV-17
6	KEERTHA DHARSHINI S	Marketing Coordinator	65000	Marketing	25-APR-22
7	NIVEATHA N	Customer Service Representative	50000	Customer Service	15-AUG-21

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;
/
```

```
Statement processed.
Applied 1% discount to loan interest rate for : AKCHARA TD
Applied 1% discount to loan interest rate for : HARINSOWMIYAM
```

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”.

Live SQL

SQL Worksheet

```
ALTER TABLE Customers ADD (IsVIP CHAR(1) DEFAULT 'N');

```

```
Table altered.
```

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;
/
```

```
Statement processed.
ANAMIKA M - Promoted to VIP status.
KEERTHI S - Promoted to VIP status.
BALASUBRAMANI T - Promoted to VIP status.
SHYAM SUNDAR P S - Promoted to VIP status.
SANTHOSH KUMAR V - Promoted to VIP status.
```

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;
/
```

```
Statement processed.
Reminder: Dear GOURAV PRITHAM G R, your loan with ID 12 is due on 01-SEP-2024
```

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;
/
```

```
Procedure created.
```

Figure 2.1: Scenario 1 – Ensuring Safe Transfer Fund

```
BEGIN
    SafeTransferFunds(5, 10, 1250);
END;
/
```

```
Procedure created.

Statement processed.
Error: Insufficient funds in Account 5
```

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;
/
```

```
BEGIN
    UpdateSalary(1, 10); -- Assuming employee ID 1 exists
END;
/
```

```
Procedure created.

Statement processed.
Salary updated successfully for Employee ID: 1
```

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

```
BEGIN
    UpdateSalary(100, 25); -- Assuming employee ID 1 exists
END;
/
```

```
Procedure created.

Statement processed.
Error: Employee ID 100 does not exist.
```

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;
/
```

```
BEGIN
    AddNewCustomer(1, 'Jeyakumar', TO_DATE('1980-01-15', 'YYYY-MM-DD'), 5000);
END;
/
```

```
Procedure created.

Statement processed.
Error: Customer with ID 1 already exists.
```

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

```
BEGIN
    AddNewCustomer(32, 'Jeyakumar', TO_DATE('1980-01-15', 'YYYY-MM-DD'), 5000);
END;
/
```

```
Procedure created.

Statement processed.
Customer added successfully with ID 32
```

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;
/
```

```
BEGIN
    ProcessMonthlyInterest;
END;
/
```

```
Procedure created.

Statement processed.
Account ID 2 updated. New Balance: 5555
Account ID 4 updated. New Balance: 8080
Account ID 6 updated. New Balance: 2525
Account ID 8 updated. New Balance: 4242
Account ID 10 updated. New Balance: 7070
Account ID 12 updated. New Balance: 4040
Account ID 14 updated. New Balance: 2222
Account ID 16 updated. New Balance: 3030
Account ID 18 updated. New Balance: 5555
Account ID 20 updated. New Balance: 6514.5
```

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;
/
```

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

```
Procedure created.

Statement processed.
Employee ID 1 updated. New Salary: 82500
Employee ID 15 updated. New Salary: 68200
```

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;
/
```

```
BEGIN
    SafeTransferFunds(5, 10, 1250);
END;
/
```

```
Procedure created.

Statement processed.
Error: Insufficient funds in Account 5
```

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

```
BEGIN
    SafeTransferFunds(15, 20, 250);
END;
/
```

```
Procedure created.

Statement processed.
Transfer successful from Account 15 to Account 20 for amount 250
```

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;
/
```

```
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;
/
```

```
Function created.

Statement processed.
Customer Age: 39
```

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;
/
```

```
Function created.

Statement processed.
Monthly Installment: 1060.6551523907523221827980442955084272 98
```

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;
/
```

```
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
```

```
Function created.

Statement processed.
Account has sufficient balance.
```

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;
```

```
SELECT CustomerID, Name, DOB, Balance, LastModified
FROM Customers
WHERE CustomerID = 1;
```

```
Trigger created.

1 row(s) updated.
```

CUSTOMERID	NAME	DOB	BALANCE	LASTMODIFIED
1	Uphendhra P	01-JAN-62	5000	07-AUG-24

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;
```

```
Trigger created.

1 row(s) inserted.
```

AUDITID	TRANSACTIONID	ACCOUNTID	TRANSACTIONDATE	AMOUNT	TRANSACTIONTYPE	AUDITTIMESTAMP	ACTION
1	29	9	07-AUG-24	600	Debit	07-AUG-24 01.33.19.952012 PM	INSERT

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');
```

```
Trigger created.

1 row(s) inserted.

1 row(s) inserted.

ORA-20005: An unexpected error occurred: ORA-20001: Withdrawal amount exceeds the current balance. ORA-06512: at
"SQL_RKXUYQAEDIOMYWBAAFERUJAMD.CHECKTRANSACTIONRULES", line 26
ORA-06512: at "SYS.DBMS_SQL", line 1721

More Details: https://docs.oracle.com/error-help/db/ora-20005

ORA-20005: An unexpected error occurred: ORA-20002: Deposit amount must be positive. ORA-06512: at "SQL_RKXUYQAEDIOMYWBAAFERUJAMD
ORA-06512: at "SYS.DBMS_SQL", line 1721
```

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;
/
```

```
Statement processed.
Customer ID: 1
Customer Name: Uphendhra P
Account ID: 1
Transaction Date: 07-AUG-24
Amount: 100
Transaction Type: Deposit
------------------------------
Customer ID: 2
Customer Name: ANAMIKA M
Account ID: 2
Transaction Date: 07-AUG-24
Amount: 50
Transaction Type: Withdrawal
------------------------------
Customer ID: 9
Customer Name: ABIKANNAN P R
Account ID: 9
Transaction Date: 07-AUG-24
Amount: 600
Transaction Type: Debit
```

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;
/
```

```
Statement processed.
Account ID: 1
New Balance: 1450
------------------------------
Account ID: 2
New Balance: 5505
------------------------------
Account ID: 3
New Balance: 1450
------------------------------
Account ID: 4
New Balance: 8030
------------------------------
Account ID: 5
New Balance: 1150
------------------------------
Account ID: 6
New Balance: 2475
------------------------------
Account ID: 7
New Balance: 2950
```

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;
/
```

```
Statement processed.
Loan ID: 1
New Interest Rate: .0306
------------------------------
Loan ID: 2
New Interest Rate: .0408
------------------------------
Loan ID: 3
New Interest Rate: .0408
------------------------------
Loan ID: 4
New Interest Rate: .051
------------------------------
Loan ID: 5
New Interest Rate: .0714
------------------------------
Loan ID: 6
New Interest Rate: .051
------------------------------
Loan ID: 7
New Interest Rate: .0612
```

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;
/
```

```
Package created.

Package Body created.

Statement processed.
Customer Balance: 5500
```

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;
/
```

```
Package created.

Package Body created.

Statement processed.
Employee ID 201 already exists.
Annual Salary: 84000
```

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;
   /
```

```
Package created.

Package Body created.

Statement processed.
An error occurred: ORA-02291: integrity constraint (SQL_RKXUYQAEDIOMYWBAAFERUJAMD.SYS_C00163968282) violated - parent key not found
No account found with ID 1001
Total Balance: 0
```

Figure 7.3: Scenario 3 – Account Management using Packages