

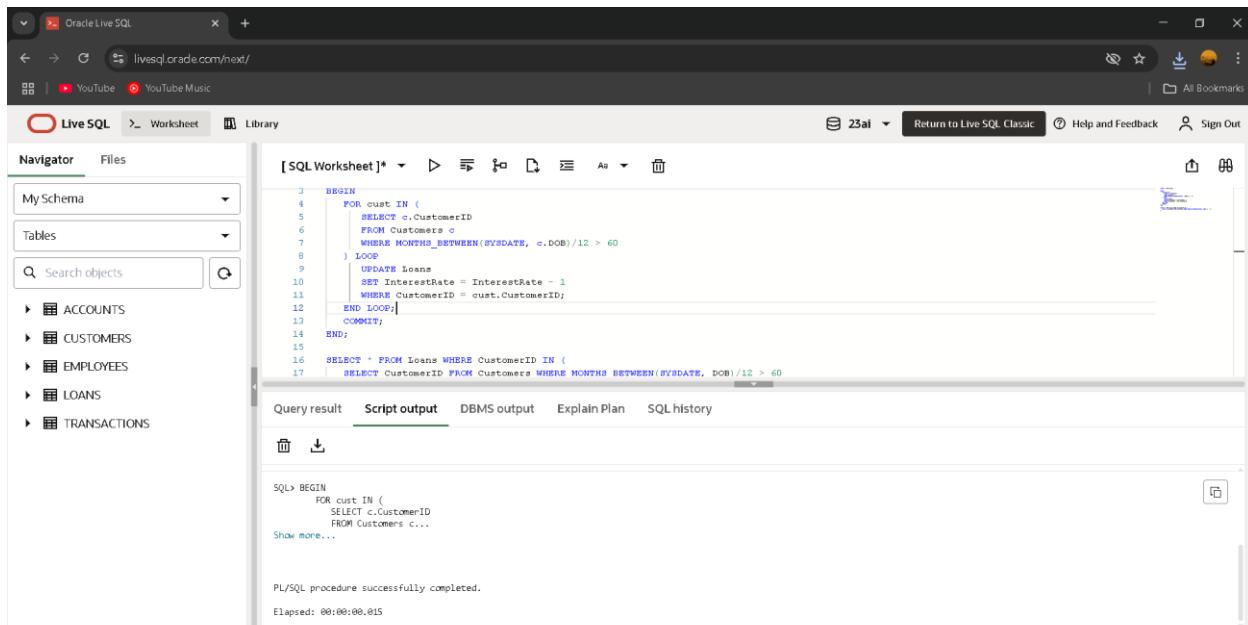
## Exercise 1: Control Structures

**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.

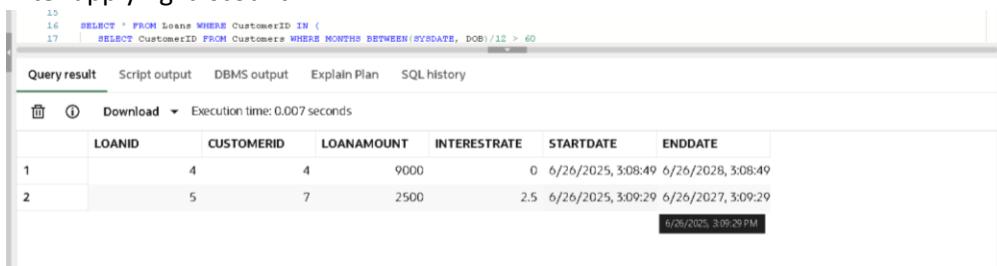
**Procedure:**

```
BEGIN
FOR cust IN (
SELECT c.CustomerID
FROM Customers c
WHERE MONTHS_BETWEEN(SYSDATE, c.DOB)/12 > 60
) LOOP
UPDATE Loans
SET InterestRate = InterestRate - 1
WHERE CustomerID = cust.CustomerID;
END LOOP;
COMMIT;
END;
```



The screenshot shows the Oracle Live SQL interface. The left sidebar displays the Navigator with tables like ACCOUNTS, CUSTOMERS, EMPLOYEES, LOANS, and TRANSACTIONS. The main area is titled [SQL Worksheet] and contains the PL/SQL code provided above. The 'Script output' tab is selected, showing the execution results:  
SQL> BEGIN  
FOR cust IN (  
SELECT c.CustomerID  
FROM Customers c  
WHERE MONTHS\_BETWEEN(SYSDATE, c.DOB)/12 > 60  
) LOOP  
UPDATE Loans  
SET InterestRate = InterestRate - 1  
WHERE CustomerID = cust.CustomerID;  
END LOOP;  
COMMIT;  
END;  
PL/SQL procedure successfully completed.  
Elapsed: 00:00:00.015

After applying discount:



The screenshot shows the Oracle Live SQL interface with the 'Query result' tab selected. The previous PL/SQL block has been run, and now a new query is being run:  
16 SELECT \* FROM Loans WHERE CustomerID IN (  
17 | SELECT CustomerID FROM Customers WHERE MONTHS\_BETWEEN(SYSDATE, DOB)/12 > 60  
The results table shows two rows of data:

LOANID	CUSTOMERID	LOANAMOUNT	INTERESTRATE	STARTDATE	ENDDATE
1	4	9000	0	6/26/2025, 3:08:49	6/26/2028, 3:08:49
2	5	2500	2.5	6/26/2025, 3:09:29	6/26/2027, 3:09:29

**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.

**Procedure:**

```
ALTER TABLE Customers ADD IsVIP VARCHAR2(5);
BEGIN
FOR cust IN (
    SELECT CustomerID, Balance FROM Customers
) LOOP
IF cust.Balance > 10000 THEN
    UPDATE Customers
    SET IsVIP = 'TRUE'
    WHERE CustomerID = cust.CustomerID;
END IF;
END LOOP;
COMMIT;
END;
```

The screenshot shows the Oracle Live SQL interface. On the left, the Navigator pane lists schemas, tables, and objects like ACCOUNTS, CUSTOMERS, EMPLOYEES, LOANS, and TRANSACTIONS. The main area displays the PL/SQL code. The 'Script output' tab is selected, showing the execution of the code. The output shows the procedure successfully completed. The 'Query result' tab at the bottom shows the updated data in the CUSTOMERS table, where two rows have their IsVIP column set to 'TRUE'.

```
4   SELECT CustomerID, Name, Balance, IsVIP FROM Customers;
5
6 BEGIN
7   FOR cust IN (
8     SELECT CustomerID, Balance FROM Customers
9   ) LOOP
10    IF cust.Balance > 10000 THEN
11      UPDATE Customers
12      SET IsVIP = 'TRUE'
13      WHERE CustomerID = cust.CustomerID;
14    END IF;
15  END LOOP;
16  COMMIT;
17 END;
18
19
20
21
22
```

```
SQL> BEGIN
  2  FOR cust IN (
  3    SELECT CustomerID, Balance FROM Customers
  4  ) LOOP...
  5
  6
  7
  8
  9
 10
 11
 12
 13
 14
 15
 16
 17
 18
 19
 20
 21
 22
```

PL/SQL procedure successfully completed.  
Elapsed: 00:00:00.016

After procedure:

The screenshot shows the Oracle Live SQL interface. The 'Query result' tab is selected, displaying the results of a query on the CUSTOMERS table. The results show two rows: David Wright and Emily Clark, both of whom have a balance over \$10,000 and have their IsVIP status set to 'TRUE'.

CUSTOMERID	NAME	BALANCE	ISVIP
6	David Wright	20000	TRUE
3	Emily Clark	12000	TRUE

**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.

**Procedure:**

```
BEGIN
FOR rec IN (
    SELECT l.LoanID, l.EndDate, c.Name
    FROM Loans l
    JOIN Customers c ON l.CustomerID = c.CustomerID
    WHERE l.EndDate BETWEEN SYSDATE AND SYSDATE + 30
) LOOP
    DBMS_OUTPUT.PUT_LINE('Reminder: Loan ID ' || rec.LoanID ||
        ' for customer ' || rec.Name ||
        ' is due on ' || TO_CHAR(rec.EndDate, 'DD-MON-YYYY'));
END LOOP;
END
```

The screenshot shows the Oracle SQL Developer interface. The top navigation bar includes 'Live SQL', 'Worksheet', 'Library', and various status indicators like '23ai' and 'Help and Feedback'. The main area has tabs for 'Navigator', 'Files', and 'SQL Worksheet'. The 'SQL Worksheet' tab is active, displaying the PL/SQL code. Below the code, there are tabs for 'Query result', 'Script output', 'DBMS output', 'Explain Plan', and 'SQL history'. The 'Script output' tab is selected, showing the executed code and its output. The output pane displays the reminder messages for each loan due within the next 30 days, such as 'Reminder: Loan ID 1 for customer John Doe is due on 15-JUN-2023'. At the bottom of the output pane, it says 'PL/SQL procedure successfully completed.' and 'Elapsed: 00:00:00.006'.

## 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.

**Procedure:**

```
CREATE OR REPLACE PROCEDURE SafeTransferFunds (
    p_from_account_id IN NUMBER,
    p_to_account_id IN NUMBER,
    p_amount IN NUMBER
) AS
```

```

v_balance NUMBER;
BEGIN
    -- Check balance of sender
    SELECT Balance INTO v_balance
    FROM Accounts
    WHERE AccountID = p_from_account_id;
    IF v_balance < p_amount THEN
        RAISE_APPLICATION_ERROR(-20001, 'Insufficient funds in source account.');
    END IF;
    -- Deduct from sender
    UPDATE Accounts
    SET Balance = Balance - p_amount,
        LastModified = SYSDATE
    WHERE AccountID = p_from_account_id;
    -- Credit to receiver
    UPDATE Accounts
    SET Balance = Balance + p_amount,
        LastModified = SYSDATE
    WHERE AccountID = p_to_account_id;
    COMMIT;
    DBMS_OUTPUT.PUT_LINE('Transfer completed successfully.');
EXCEPTION
    WHEN OTHERS THEN
        ROLLBACK;
        DBMS_OUTPUT.PUT_LINE('Error during fund transfer: ' || SQLERRM);
END;

```

The screenshot shows the Oracle Live SQL interface. The main area displays the PL/SQL code for the `SafeTransferFunds` procedure. Below the code, the results of the execution are shown, including the SQL command, the compiled procedure name, and the elapsed time (00:00:00.018). The bottom of the screen shows the Windows taskbar with various application icons.

```

CREATE OR REPLACE PROCEDURE SafeTransferFunds (
    p_from_account_id IN NUMBER,
    p_to_account_id IN NUMBER,
    p_amount IN NUMBER
) AS
    v_balance NUMBER;
BEGIN
    -- Check balance of sender
    SELECT Balance INTO v_balance
    FROM Accounts
    WHERE AccountID = p_from_account_id;
    IF v_balance < p_amount THEN
        RAISE_APPLICATION_ERROR(-20001, 'Insufficient funds in source account.');
    END IF;
    -- Deduct from sender
    UPDATE Accounts
    SET Balance = Balance - p_amount,
        LastModified = SYSDATE
    WHERE AccountID = p_from_account_id;
    -- Credit to receiver
    UPDATE Accounts
    SET Balance = Balance + p_amount,
        LastModified = SYSDATE
    WHERE AccountID = p_to_account_id;
    COMMIT;
    DBMS_OUTPUT.PUT_LINE('Transfer completed successfully.');
EXCEPTION
    WHEN OTHERS THEN
        ROLLBACK;
        DBMS_OUTPUT.PUT_LINE('Error during fund transfer: ' || SQLERRM);
END;

```

Procedure SAFETRANSFERFUNDS compiled  
Elapsed: 00:00:00.018

**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.

**Procedure**

```
CREATE OR REPLACE PROCEDURE UpdateSalary (
    p_employee_id IN NUMBER,
    p_percentage  IN NUMBER
) AS
    v_count NUMBER;
BEGIN
    -- Check if the employee exists
    SELECT COUNT(*) INTO v_count
    FROM Employees
    WHERE EmployeeID = p_employee_id;

    IF v_count = 0 THEN
        DBMS_OUTPUT.PUT_LINE('Error: Employee with ID ' || p_employee_id || ' does not exist.');
        RETURN;
    END IF;

    -- Update salary
    UPDATE Employees
    SET Salary = Salary + (Salary * p_percentage / 100)
    WHERE EmployeeID = p_employee_id;

    COMMIT;
    DBMS_OUTPUT.PUT_LINE('Salary updated successfully for Employee ID ' || p_employee_id);
EXCEPTION
    WHEN OTHERS THEN
        ROLLBACK;
        DBMS_OUTPUT.PUT_LINE('Unexpected error: ' || SQLERRM);
END;

EXEC UpdateSalary(99, 10);
```

The screenshot shows the Oracle Live SQL interface. On the left, the Navigator pane lists schemas, tables, and objects like ACCOUNTS, CUSTOMERS, EMPLOYEES, LOANS, and TRANSACTIONS. The main area displays a SQL Worksheet with the following PL/SQL code:

```

14    RETURN;
15  END IF;
16
17  -- Update salary
18  UPDATE Employees
19    SET Salary = Salary + (Salary * p_percentage / 100)
20   WHERE EmployeeID = p_employee_id;
21
22  COMMIT;
23  DBMS_OUTPUT.PUT_LINE('Salary updated successfully for Employee ID ' || p_employee_id);
24 EXCEPTION
25   WHEN OTHERS THEN
26     ROLLBACK;
27     DBMS_OUTPUT.PUT_LINE('Unexpected error: ' || SQLERRM);

```

The "Script output" tab is selected, showing the results of the execution:

```

Procedure UPDATESALARY compiled
Elapsed: 00:00:00.024

SQL> EXEC UpdateSalary(99, 10)

Error: Employee with ID 99 does not exist.

```

At the bottom, the status bar shows "r31.1".

### 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.

#### Procedure:

```

CREATE OR REPLACE PROCEDURE AddNewCustomer (
  p_customer_id IN NUMBER,
  p_name      IN VARCHAR2,
  p_dob       IN DATE,
  p_balance   IN NUMBER
) AS
BEGIN
  INSERT INTO Customers (CustomerID, Name, DOB, Balance, LastModified)
  VALUES (p_customer_id, p_name, p_dob, p_balance, SYSDATE);

  COMMIT;
  DBMS_OUTPUT.PUT_LINE('Customer inserted successfully. ID: ' || p_customer_id);
EXCEPTION
  WHEN DUP_VAL_ON_INDEX THEN
    DBMS_OUTPUT.PUT_LINE('Error: Customer with ID ' || p_customer_id || ' already exists. Insertion aborted.');
  WHEN OTHERS THEN
    DBMS_OUTPUT.PUT_LINE('Unexpected error occurred: ' || SQLERRM);
END;

```

```
EXEC AddNewCustomer(5, 'Duplicate', TO_DATE('1990-01-01', 'YYYY-MM-DD'), 3000);
```

The screenshot shows the Oracle Live SQL interface. On the left, there is a sidebar with database schema navigation: ACCOUNTS, CUSTOMERS, EMPLOYEES, LOANS, and TRANSACTIONS. The main area has tabs at the top: Query result, Script output (which is selected), DBMS output, Explain Plan, and SQL history. Below the tabs, there is a toolbar with icons for copy, paste, and refresh. The script output pane contains the following text:

```
21
22
Procedure ADDNEWCUSTOMER compiled
Elapsed: 00:00:00.019
SQL> EXEC AddNewCustomer(5, 'Duplicate', TO_DATE('1990-01-01', 'YYYY-MM-DD'), 3000)
Error: Customer with ID 5 already exists. Insertion aborted.
```

At the bottom of the interface, there are links for About Oracle, Contact Us, Legal Notices, Terms and Conditions, Your Privacy Rights, Delete Your Live SQL Account, and Cookie Preferences. A copyright notice from 2014-2025 is also present. The system status bar at the bottom right shows the date (6/27/2025), time (8:35 PM), battery level (ENG IN), and temperature (33°C Haze).

## 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.

### Procedure

```
CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest IS
BEGIN
    -- Update balance for all 'Savings' accounts by adding 1% interest
    UPDATE Accounts
    SET Balance = Balance + (Balance * 0.01),
        LastModified = SYSDATE
    WHERE AccountType = 'Savings';

    COMMIT;
    DBMS_OUTPUT.PUT_LINE('Monthly interest of 1% applied to all savings accounts.');
EXCEPTION
    WHEN OTHERS THEN
        ROLLBACK;
        DBMS_OUTPUT.PUT_LINE('Error applying interest: ' || SQLERRM);
END;

EXEC ProcessMonthlyInterest;

--to see whether it has been applied
SELECT AccountID, Balance FROM Accounts WHERE AccountType = 'Savings';
```

```

9   COMMIT;
10  DBMS_OUTPUT.PUT_LINE('Monthly interest of 1% applied to all savings accounts.');
11  EXCEPTION
12    WHEN OTHERS THEN
13      ROLLBACK;
14      DBMS_OUTPUT.PUT_LINE('Error applying interest: ' || SQLERRM);
15  END;
16
17  EXEC ProcessMonthlyInterest;
18

```

Procedure PROCESSTHREEMONTHLYINTEREST compiled  
Elapsed: 00:00:00.012

SQL> EXEC ProcessMonthlyInterest

Monthly interest of 1% applied to all savings accounts.

PL/SQL procedure successfully completed.

After procedure schema looks like

```

9   COMMIT;
10  DBMS_OUTPUT.PUT_LINE('Monthly interest of 1% applied to all savings accounts.');
11  EXCEPTION
12    WHEN OTHERS THEN
13      ROLLBACK;
14      DBMS_OUTPUT.PUT_LINE('Error applying interest: ' || SQLERRM);
15  END;
16
17  EXEC ProcessMonthlyInterest;
18
19
20  SELECT AccountID, Balance FROM Accounts WHERE AccountType = 'Savings';
21

```

Execution time: 0.005 seconds

ACCOUNTID	BALANCE
1	5 505
2	3 12120
3	6 20200

**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.

**Procedure:**

CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus (

```

p_department IN VARCHAR2,
p_bonus_percent IN NUMBER
) AS
BEGIN
-- Update salaries for employees in the specified department
UPDATE Employees
SET Salary = Salary + (Salary * p_bonus_percent / 100)
WHERE Department = p_department;
IF SQL%ROWCOUNT = 0 THEN
    DBMS_OUTPUT.PUT_LINE('No employees found in department: ' || p_department);
ELSE
    DBMS_OUTPUT.PUT_LINE('Bonus of ' || p_bonus_percent || '%' applied to department: ' || p_department);
END IF;

COMMIT;
EXCEPTION
WHEN OTHERS THEN
    ROLLBACK;
    DBMS_OUTPUT.PUT_LINE('Error applying bonus: ' || SQLERRM);
END;
EXEC UpdateEmployeeBonus('IT', 10);
--After applying bonus
SELECT EmployeeID, Name, Department, Salary FROM Employees WHERE Department = 'IT';

```

The screenshot shows the Oracle SQL Worksheet interface. On the left, the Navigator pane displays the schema structure with tables: ACCOUNTS, CUSTOMERS, EMPLOYEES, LOANS, and TRANSACTIONS. The main workspace contains the PL/SQL code provided above. The 'Script output' tab is selected, showing the execution results:

```

Elapsed: 00:00:00.060
SQL> EXEC UpdateEmployeeBonus('IT', 10)

Bonus of 10% applied to department: IT

PL/SQL procedure successfully completed.
Elapsed: 00:00:00.013

```

At the bottom, the Windows taskbar shows the system status: 33°C Light rain, ENG, 9:02 PM, 6/27/2025.

Schema after execution looks like

The screenshot shows a SQL developer interface. On the left, there's a sidebar with icons for CUSTOMERS, EMPLOYEES, LOANS, and TRANSACTIONS. The main area has a query editor with the following code:

```

26  --After applying bonus
27  SELECT EmployeeID, Name, Department, Salary FROM Employees WHERE Department = 'IT';
28

```

Below the code is a table titled "Query result" with the following data:

EMPLOYEEID	NAME	DEPARTMENT	SALARY
1	5 Anita Roy	IT	68200

At the bottom of the interface, there are links for About Oracle, Contact Us, Legal Notices, Terms and Conditions, Your Privacy Rights, Delete Your Live SQL Account, and Cookie Preferences. The status bar at the bottom right shows "r31.1", "33°C Light rain", "ENG 9:04 PM IN 6/27/2025", and a battery icon.

**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.

**Procedure:**

```

CREATE OR REPLACE PROCEDURE TransferFunds (
    p_from_account_id IN NUMBER,
    p_to_account_id  IN NUMBER,
    p_amount        IN NUMBER
) AS
    v_from_balance NUMBER;
    v_from_customer_id NUMBER;
    v_to_customer_id NUMBER;
BEGIN
    -- Get source account balance and customer
    SELECT Balance, CustomerID INTO v_from_balance, v_from_customer_id
    FROM Accounts
    WHERE AccountID = p_from_account_id;
    -- Get destination account's customer
    SELECT CustomerID INTO v_to_customer_id
    FROM Accounts
    WHERE AccountID = p_to_account_id;
    -- Check if both accounts belong to the same customer
    IF v_from_customer_id != v_to_customer_id THEN
        DBMS_OUTPUT.PUT_LINE('Error: Cannot transfer between accounts of different customers.');
        RETURN;
    END IF;
    -- Check for sufficient funds
    IF v_from_balance < p_amount THEN
        DBMS_OUTPUT.PUT_LINE('Error: Insufficient funds in source account.');
        RETURN;
    END IF;
    -- Perform transfer

```

```

UPDATE Accounts
SET Balance = Balance - p_amount,
    LastModified = SYSDATE
WHERE AccountID = p_from_account_id;
UPDATE Accounts
SET Balance = Balance + p_amount,
    LastModified = SYSDATE
WHERE AccountID = p_to_account_id;
COMMIT;
DBMS_OUTPUT.PUT_LINE('Transfer of ' || p_amount || ' successful between accounts ' ||
                     p_from_account_id || ' and ' || p_to_account_id);

EXCEPTION
    WHEN NO_DATA_FOUND THEN
        DBMS_OUTPUT.PUT_LINE('Error: One or both account IDs not found.');
    WHEN OTHERS THEN
        ROLLBACK;
        DBMS_OUTPUT.PUT_LINE('Unexpected error: ' || SQLERRM);
END;

```

EXEC TransferFunds(3, 5, 500); -- Assuming both accounts belong to CustomerID = 3

The screenshot shows the Oracle Live SQL interface. The top navigation bar includes 'Live SQL', 'Worksheet', 'Library', '23al', 'Return to Live SQL Classic', 'Help and Feedback', and 'Sign Out'. The main area has tabs for 'Navigator' (selected), 'Files', and 'SQL Worksheet'. The Navigator pane shows schema objects: My Schema (Tables: ACCOUNTS, CUSTOMERS, EMPLOYEES, LOANS, TRANSACTIONS). The SQL Worksheet pane contains the PL/SQL code provided above. The 'Script output' tab is selected, showing the following results:

```

Elapsed: 00:00:00.016
SQL> EXEC TransferFunds(3, 5, 500)

Error: Cannot transfer between accounts of different customers.

PL/SQL procedure successfully completed.

Elapsed: 00:00:00.075

```

At the bottom, there are links for 'About Oracle', 'Contact Us', 'Legal Notices', 'Terms and Conditions', 'Your Privacy Rights', 'Delete Your Live SQL Account', and 'Cookie Preferences'. A copyright notice states 'Copyright © 2014, 2025 Oracle and/or its affiliates. All rights reserved.' The status bar at the bottom right shows 'r31.1', 'ENG', and '9:10 PM'.

## 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

## Procedure

```
CREATE OR REPLACE FUNCTION CalculateAge (
    p_dob IN DATE
) RETURN NUMBER IS
    v_age NUMBER;
BEGIN
    -- Calculate age in years
    v_age := FLOOR(MONTHS_BETWEEN(SYSDATE, p_dob) / 12);
    RETURN v_age;
END;
```

--to use function

```
SELECT Name, DOB, CalculateAge(DOB) AS Age
FROM Customers;
```

The screenshot shows the Oracle Live SQL interface. In the top navigation bar, 'LOANS' and 'TRANSACTIONS' are visible. The main area has tabs for 'Query result', 'Script output' (which is selected), 'DBMS output', 'Explain Plan', and 'SQL history'. The 'Script output' tab displays the SQL code for creating the function. Below the code, it says 'Function CALCULATEAGE compiled' and shows an elapsed time of 'Elapsed: 00:00:00.013'. At the bottom, there's a toolbar with various icons and a status bar showing 'r31.1', '32°C Mostly cloudy', and the date '6/27/2025'.

This screenshot shows the Oracle Live SQL interface after executing the function. The 'Navigator' sidebar on the left lists 'My Schema', 'Tables', and other tables like 'ACCOUNTS', 'CUSTOMERS', 'EMPLOYEES', 'LOANS', and 'TRANSACTIONS'. The 'SQL Worksheet' tab is active, showing the same SQL code as the previous screenshot. The 'Query result' tab is selected, displaying a table with columns 'NAME', 'DOB', and 'AGE'. The data is as follows:

	NAME	DOB	AGE
1	Nina Patel	9/30/1955, 12:00:00	69
2	Michael Lee	8/25/1962, 12:00:00	62
3	David Wright	11/11/1988, 12:00:00	36
4	Sara Khan	1/5/2000, 12:00:00	25
5	Emily Clark	3/10/1970, 12:00:00	55

At the bottom, the status bar includes links to 'About Oracle', 'Contact Us', 'Legal Notices', 'Terms and Conditions', 'Your Privacy Rights', 'Delete Your Live SQL Account', and 'Cookie Preferences'. It also shows the date '6/27/2025' and the version 'r31.1'.

**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

**Procedure:**

```
CREATE OR REPLACE FUNCTION CalculateMonthlyInstallment (
    p_loan_amount    IN NUMBER,
    p_annual_rate    IN NUMBER,
    p_years          IN NUMBER
) RETURN NUMBER IS
    v_monthly_rate   NUMBER;
    v_months         NUMBER;
    v_emi             NUMBER;
BEGIN
    -- Convert annual rate to monthly decimal rate
    v_monthly_rate := p_annual_rate / 12 / 100;
    v_months := p_years * 12;

    -- EMI formula: P * r * (1 + r)^n / ((1 + r)^n - 1)
    IF v_monthly_rate = 0 THEN
        v_emi := p_loan_amount / v_months; -- No interest case
    ELSE
        v_emi := p_loan_amount * v_monthly_rate * POWER(1 + v_monthly_rate, v_months)
            / (POWER(1 + v_monthly_rate, v_months) - 1);
    END IF;

    RETURN ROUND(v_emi, 2);
END;

--to use fuction in schema
SELECT LoanID,
       LoanAmount,
       InterestRate,
       CalculateMonthlyInstallment(LoanAmount, InterestRate, 5) AS MonthlyEMI
FROM Loans;
```

```

1 -- EMI formula: P * r * (1 + r)^n / ((1 + r)^n - 1)
2 IF v_monthly_rate = 0 THEN
3   v_eml := p_loan_amount / v_months; -- No interest case
4 ELSE
5   v_eml := p_loan_amount * v_monthly_rate * POWER(1 + v_monthly_rate, v_months)
6   / (POWER(1 + v_monthly_rate, v_months) - 1);
7 END IF;
8
9 RETURN ROUND(v_eml, 2);
10
11 END;

```

Query result   Script output   DBMS output   Explain Plan   SQL history

SQL> CREATE OR REPLACE FUNCTION CalculateMonthlyInstallment (
 p\_loan\_amount IN NUMBER,
 p\_annual\_rate IN NUMBER,
 p\_years IN NUMBER...
Show more...

Function CALCULATEMONTHLYINSTALLMENT compiled

Elapsed: 00:00:00.014

### Schema:

```

1 -- to use function in schema
2 BEGIN
3   SELECT LoanID,
4         LoanAmount,
5         InterestRate,
6         .....
7         CalculateMonthlyInstallment(LoanAmount, InterestRate, 5) AS MonthlyEMI
8   FROM Loans;
9
10
11 END;
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32

```

Query result   Script output   DBMS output   Explain Plan   SQL history

	LOANID	LOANAMOUNT	INTERESTRATE	MONTHLYEMI	
1		6	20000	3.9	36743
2		4	9000	0	150
3		5	2500	2.5	44.37
4		2	7000	4.5	130.5
5		3	15000	5.2	284.44

About Oracle | Contact Us | Legal Notices | Terms and Conditions | Your Privacy Rights | Delete Your Live SQL Account | Cookie Preferences  
Copyright © 2014, 2025 Oracle and/or its affiliates. All rights reserved.

**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.

### Procedure:

```

CREATE OR REPLACE FUNCTION HasSufficientBalance (
  p_account_id IN NUMBER,
  p_amount IN NUMBER
) RETURN BOOLEAN IS
  v_balance NUMBER;
BEGIN

```

```

SELECT Balance INTO v_balance
FROM Accounts
WHERE AccountID = p_account_id;

RETURN v_balance >= p_amount;
EXCEPTION
    WHEN NO_DATA_FOUND THEN
        DBMS_OUTPUT.PUT_LINE('Account ID not found: ' || p_account_id);
        RETURN FALSE;
    WHEN OTHERS THEN
        DBMS_OUTPUT.PUT_LINE('Error: ' || SQLERRM);
        RETURN FALSE;
END;

```

--using this func

```

DECLARE
    v_result BOOLEAN;
BEGIN
    v_result := HasSufficientBalance(3, 500);

    IF v_result THEN
        DBMS_OUTPUT.PUT_LINE('Sufficient balance available.');
    ELSE
        DBMS_OUTPUT.PUT_LINE('Insufficient balance or account not found.');
    END IF;
END;

```

The screenshot shows the Oracle Live SQL interface. The code area contains the provided PL/SQL block. The results section shows the output: "Sufficient balance available." followed by a message indicating the procedure was successfully completed.

```

22  --using function
23  DECLARE
24      v_result BOOLEAN;
25  BEGIN
26      v_result := HasSufficientBalance(3, 500);
27
28      IF v_result THEN
29          DBMS_OUTPUT.PUT_LINE('Sufficient balance available.');
30      ELSE
31          DBMS_OUTPUT.PUT_LINE('Insufficient balance or account not found.');
32      END IF;
33  END;

```

```

v_result BOOLEAN;
BEGIN
    v_result := HasSufficientBalance(3, 500);...

Show more...

Sufficient balance available.

PL/SQL procedure successfully completed.

Elapsed: 00:00:00.010

```

## 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.

### Procedure:

```
CREATE OR REPLACE TRIGGER UpdateCustomerLastModified
```

```
BEFORE UPDATE ON Customers
```

```
FOR EACH ROW
```

```
BEGIN
```

```
    :NEW.LastModified := SYSDATE;
```

```
END;
```

--to modify

```
UPDATE Customers
```

```
SET Balance = Balance + 1000
```

```
WHERE CustomerID = 6;
```

```
COMMIT;
```

--to check

```
SELECT CustomerID, Name, LastModified FROM Customers WHERE CustomerID = 6;
```

The screenshot shows the Oracle Live SQL interface. On the left, the Navigator pane lists tables: ACCOUNTS, CUSTOMERS, EMPLOYEES, LOANS, and TRANSACTIONS. The main area is a SQL Worksheet containing the following code:

```
1 CREATE OR REPLACE TRIGGER UpdateCustomerLastModified
2 BEFORE UPDATE ON Customers
3 FOR EACH ROW
4 BEGIN
5     :NEW.LastModified := SYSDATE;
6 END;
7
8 --to modify
9 UPDATE Customers
10 SET Balance = Balance + 1000
11 WHERE CustomerID = 1;
12
13 COMMIT;
14
```

Below the worksheet, the Query result tab shows the trigger definition and execution results:

```
SQL> CREATE OR REPLACE TRIGGER UpdateCustomerLastModified
  BEFORE UPDATE ON Customers
  FOR EACH ROW
  BEGIN...
Show more...
```

```
Trigger UPDATECUSTOMERLASTMODIFIED compiled
Elapsed: 00:00:00.016
```

## After modification

The screenshot shows the Oracle Live SQL interface. In the SQL Worksheet, a transaction is performed on the CUSTOMERS table:

```
5 :NEW.LastModified := SYSDATE;
6 END;
7
8 --to modify
9 UPDATE Customers
10 SET Balance = Balance + 1000
11 WHERE CustomerID = 6;
12
13 COMMIT;
14
15 --to check
16 SELECT CustomerID, Name, LastModified FROM Customers WHERE CustomerID = 6;
```

The Query result pane shows the updated record:

CUSTOMERID	NAME	LASTMODIFIED
6	David Wright	6/27/2025, 4:09:42

At the bottom, the Windows taskbar shows the date as 6/27/2025.

## 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.

### Procedure

--to create audit table

CREATE TABLE AuditLog (

```
LogID      NUMBER GENERATED ALWAYS AS IDENTITY PRIMARY KEY,
TransactionID NUMBER,
AccountID   NUMBER,
Amount       NUMBER,
TransactionType VARCHAR2(10),
TransactionDate DATE,
LoggedAt     DATE
);
```

CREATE OR REPLACE TRIGGER LogTransaction

AFTER INSERT ON Transactions

FOR EACH ROW

BEGIN

```
INSERT INTO AuditLog (
    TransactionID,
    AccountID,
    Amount,
    TransactionType,
```

```

        TransactionDate,
        LoggedAt
    )
VALUES (
    :NEW.TransactionID,
    :NEW.AccountID,
    :NEW.Amount,
    :NEW.TransactionType,
    :NEW.TransactionDate,
    SYSDATE
);
END;

```

--insert values

```

INSERT INTO Transactions (TransactionID, AccountID, TransactionDate, Amount, TransactionType)
VALUES (100, 3, SYSDATE, 1500, 'Deposit');

```

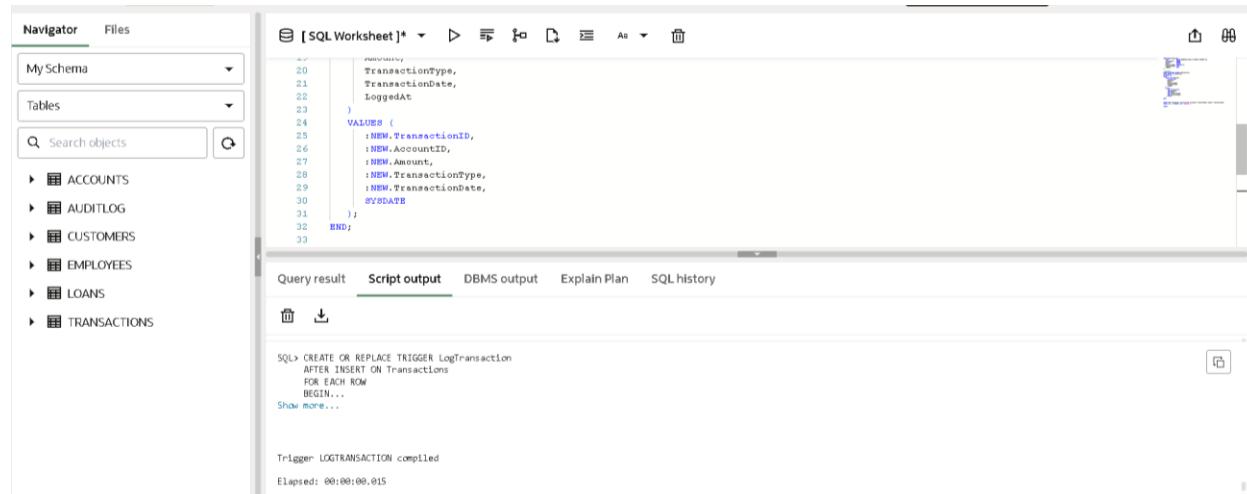
COMMIT;

--display audit

```

SELECT * FROM AuditLog WHERE TransactionID = 100;

```



The screenshot shows the Oracle SQL Developer interface. On the left, the Navigator pane displays the schema structure with tables: ACCOUNTS, AUDITLOG, CUSTOMERS, EMPLOYEES, LOANS, and TRANSACTIONS. The SQL Worksheet on the right contains the following PL/SQL code:

```

CREATE OR REPLACE TRIGGER LogTransaction
AFTER INSERT ON Transactions
FOR EACH ROW
BEGIN...

```

The code is partially visible, ending with a BEGIN... statement. Below the code, the Script output tab shows the command:

```

SQL> CREATE OR REPLACE TRIGGER LogTransaction
      AFTER INSERT ON Transactions
      FOR EACH ROW
      BEGIN...

```

Followed by the message:

```

Trigger LOGTRANSACTION compiled
Elapsed: 00:00:00.015

```

The screenshot shows the Oracle Live SQL interface. On the left, the Navigator pane lists schemas (My Schema), tables (ACCOUNTS, AUDITLOG, CUSTOMERS, EMPLOYEES, LOANS, TRANSACTIONS), and a search bar. The main area displays a SQL worksheet with the following code:

```

30    );
31  );
32  END;
33
34
35  INSERT INTO Transactions (TransactionID, AccountID, TransactionDate, Amount, TransactionType)
36  VALUES (100, 3, SYSDATE, 1500, 'Deposit');
37
38  COMMIT;
39
40  SELECT * FROM AuditLog WHERE TransactionID = 100;
41

```

Below the code, the Query result tab is selected, showing the following output:

LOGID	TRANSACTIONID	ACCOUNTID	AMOUNT	TRANSACTIONTYP	TRANSACTIONDAT	LOGGEDAT
1	1	100	3	1500	Deposit	6/27/2025, 4:16:52

At the bottom, there are links for About Oracle, Contact Us, Legal Notices, Terms and Conditions, Your Privacy Rights, Delete Your Live SQL Account, and Cookie Preferences. The system status bar shows the date (Copyright © 2014, 2025 Oracle and/or its affiliates All rights reserved.), time (r31.1), and system information (32°C Mostly cloudy, ENG IN 9:47 PM 6/27/2025).

### 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.

#### Procedure

```

CREATE OR REPLACE TRIGGER CheckTransactionRules
BEFORE INSERT ON Transactions
FOR EACH ROW
DECLARE
  v_balance NUMBER;
BEGIN
  -- Get the current balance of the account
  SELECT Balance INTO v_balance
  FROM Accounts
  WHERE AccountID = :NEW.AccountID;

  -- Rule: Deposit must be positive
  IF :NEW.TransactionType = 'Deposit' THEN
    IF :NEW.Amount <= 0 THEN
      RAISE_APPLICATION_ERROR(-20001, 'Deposit amount must be positive.');
    END IF;

    -- Rule: Withdrawal must not exceed balance
    ELSIF :NEW.TransactionType = 'Withdrawal' THEN
      IF :NEW.Amount <= 0 THEN
        RAISE_APPLICATION_ERROR(-20002, 'Withdrawal amount must be positive.');
      ELSIF :NEW.Amount > v_balance THEN
        RAISE_APPLICATION_ERROR(-20003, 'Insufficient balance for withdrawal.');
      END IF;
    END IF;
  END IF;

```

```

END IF;
END IF;
END;

```

--to check

```

INSERT INTO Transactions (TransactionID, AccountID, TransactionDate, Amount, TransactionType)
VALUES (102, 3, SYSDATE, -500, 'Deposit');

```

The screenshot shows two windows of the Oracle Live SQL interface. The top window displays the creation of a trigger:

```

SQL> CREATE OR REPLACE TRIGGER CheckTransactionRules
  BEFORE INSERT ON Transactions
  FOR EACH ROW
  DECLARE...
  Show more...

```

The trigger is successfully compiled, and the elapsed time is 00:00:00.015.

The bottom window shows the SQL Worksheet with the following code:

```

21 | RAISE_APPLICATION_ERROR(-20002, 'Withdrawal amount must be positive.');
22 | ELSEIF :NEW.Amount > v_balance THEN
23 |   RAISE_APPLICATION_ERROR(-20003, 'Insufficient balance for withdrawal.');
24 |
25 | END IF;
26 |
27 | END;
28 |
29 | ---to check
30 | INSERT INTO Transactions (TransactionID, AccountID, TransactionDate, Amount, TransactionType)
31 | VALUES (102, 3, SYSDATE, -500, 'Deposit');
32 |

```

An attempt to execute the insertion statement results in an error:

```

ORA-20001: Deposit amount must be positive.
ORA-06512: at "SQL_MINZSHNU4DVLX6B97ZZCFIHVS4.CHECKTRANSACTIONRULES", line 12
ORA-06500: error during execution of trigger 'SQL_MINZSHNU4DVLX6B97ZZCFIHVS4.CHECKTRANSACTIONRULES'
https://docs.oracle.com/error-help/db/ora-20001/
Error at Line: 4 Column: 6

```

## 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.

```

procedure.
DECLARE
  CURSOR txn_cursor IS
    SELECT c.CustomerID, c.Name, a.AccountID, t.TransactionDate,
           t.TransactionType, t.Amount
      FROM Customers c
     JOIN Accounts a ON c.CustomerID = a.CustomerID
     JOIN Transactions t ON a.AccountID = t.AccountID
    WHERE TRUNC(t.TransactionDate, 'MM') = TRUNC(SYSDATE, 'MM') -- Current month
  ORDER BY c.CustomerID, t.TransactionDate;

  v_cust_id      Customers.CustomerID%TYPE;
  v_name         Customers.Name%TYPE;
  v_acc_id       Accounts.AccountID%TYPE;
  v_txn_date     Transactions.TransactionDate%TYPE;
  v_txn_type     Transactions.TransactionType%TYPE;
  v_amount       Transactions.Amount%TYPE;
  v_last_id      NUMBER := -1;

BEGIN
  DBMS_OUTPUT.PUT_LINE('--- Monthly Statements ---');

  OPEN txn_cursor;
  LOOP
    FETCH txn_cursor INTO v_cust_id, v_name, v_acc_id, v_txn_date, v_txn_type, v_amount;
    EXIT WHEN txn_cursor%NOTFOUND;

    IF v_cust_id != v_last_id THEN
      DBMS_OUTPUT.PUT_LINE(CHR(10) || 'Customer: ' || v_name || ' (ID: ' || v_cust_id || ')');
      DBMS_OUTPUT.PUT_LINE('-----');
      v_last_id := v_cust_id;
    END IF;

    DBMS_OUTPUT.PUT_LINE('Account ID: ' || v_acc_id ||
                         ', Date: ' || TO_CHAR(v_txn_date, 'DD-MON-YYYY') ||
                         ', Type: ' || v_txn_type ||
                         ', Amount: ₹' || TO_CHAR(v_amount, '99999.99'));

  END LOOP;
  CLOSE txn_cursor;
END;
--to check OUTPUT
SELECT * FROM Transactions
WHERE TRUNC(TransactionDate, 'MM') = TRUNC(SYSDATE, 'MM');

```

```

37      END LOOP;
38      CLOSE txn_cursor;
39  END;
40

```

Customer: David Wright (ID: 6)  
Account ID: 6, Date: 26-JUN-2025, Type: Deposit, Amount: ₹ 3000.00  
Customer: Nina Patel (ID: 7)  
Account ID: 7, Date: 26-JUN-2025, Type: Withdrawal, Amount: ₹ 1500.00  
PL/SQL procedure successfully completed.  
Elapsed: 00:00:00.088

About Oracle Contact Us Legal Notices Terms and Conditions Your Privacy Rights Delete Your Live SQL Account Cookie Preferences Copyright © 2014, 2025 Oracle and/or its affiliates. All rights reserved. r31.1

```

31      END IF;
32
33      DBMS_OUTPUT.PUT_LINE('Account ID: ' || v_acc_id || 
34          ', Date: ' || TO_CHAR(v_txn_date, 'DD-MON-YYYY') || 
35          ', Type: ' || v_txn_type || 
36          ', Amount: ' || TO_CHAR(v_amount, '99999.99'));
37
38      END LOOP;
39  END;
40  --to check OUTPUT
41  SELECT * FROM Transactions
42 WHERE TRUNC(TransactionDate, 'MM') = TRUNC(SYSDATE, 'MM');
43
44

```

... Monthly Statements ...  
Customer: Emily Clark (ID: 3)  
Account ID: 3, Date: 26-JUN-2025, Type: Deposit, Amount: ₹ 1000.00  
Account ID: 3, Date: 27-JUN-2025, Type: Deposit, Amount: ₹ 1500.00  
Customer: Michael Lee (ID: 4)  
Account ID: 4, Date: 26-JUN-2025, Type: Withdrawal, Amount: ₹ 500.00  
Customer: Sara Khan (ID: 5)  
Account ID: 5, Date: 26-JUN-2025, Type: Deposit, Amount: ₹ 200.00

About Oracle Contact Us Legal Notices Terms and Conditions Your Privacy Rights Delete Your Live SQL Account Cookie Preferences https://livesql.oracle.com/next/# 2025 Oracle and/or its affiliates. All rights reserved. r31.1

## 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

### Procedure

SET SERVEROUTPUT ON;

```

DECLARE
  CURSOR acc_cursor IS
    SELECT AccountID, Balance
    FROM Accounts;

  v_acc_id Accounts.AccountID%TYPE;
  v_balance Accounts.Balance%TYPE;

```

```

v_fee CONSTANT NUMBER := 250; -- Set annual maintenance fee
BEGIN
  DBMS_OUTPUT.PUT_LINE('--- Applying Annual Maintenance Fee ---');

  OPEN acc_cursor;
  LOOP
    FETCH acc_cursor INTO v_acc_id, v_balance;
    EXIT WHEN acc_cursor%NOTFOUND;

    -- Check if balance is sufficient before deducting
    IF v_balance >= v_fee THEN
      UPDATE Accounts
      SET Balance = Balance - v_fee,
          LastModified = SYSDATE
      WHERE AccountID = v_acc_id;

      DBMS_OUTPUT.PUT_LINE('Fee of ₹' || v_fee || ' applied to Account ID: ' || v_acc_id);
    ELSE
      DBMS_OUTPUT.PUT_LINE('Skipped Account ID ' || v_acc_id || ' (Insufficient balance)');
    END IF;
  END LOOP;
  CLOSE acc_cursor;

  COMMIT;
  DBMS_OUTPUT.PUT_LINE('Annual fee processing complete.');
END;

```

The screenshot shows the Oracle SQL Developer interface. The Navigator pane on the left lists the schema and tables. The SQL Worksheet pane in the center contains the provided PL/SQL code. The Query result pane at the bottom shows the execution results, including the fees applied to each account and the final completion message.

```

11  v_fee CONSTANT NUMBER := 250; -- Set annual maintenance fee
12  BEGIN
13    DBMS_OUTPUT.PUT_LINE('--- Applying Annual Maintenance Fee ---');
14
15    OPEN acc_cursor;
16
17    |  FETCH acc_cursor INTO v_acc_id, v_balance;
18    |  EXIT WHEN acc_cursor%NOTFOUND;

... Applying Annual Maintenance Fee ...
Fee of ₹250 applied to Account ID: 5
Fee of ₹250 applied to Account ID: 7
Fee of ₹250 applied to Account ID: 3
Fee of ₹250 applied to Account ID: 4
Fee of ₹250 applied to Account ID: 6
Annual fee processing complete.

PL/SQL procedure successfully completed.
Elapsed: 00:00:00.012

```

**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.

**Procedure:**

DECLARE

```
CURSOR loan_cursor IS
  SELECT LoanID, LoanAmount, InterestRate
    FROM Loans;
```

```
v_loan_id  Loans.LoanID%TYPE;
v_amount    Loans.LoanAmount%TYPE;
v_old_rate  Loans.InterestRate%TYPE;
v_new_rate  NUMBER;
```

BEGIN

```
  DBMS_OUTPUT.PUT_LINE('--- Updating Loan Interest Rates Based on New Policy ---');
```

```
  OPEN loan_cursor;
```

```
  LOOP
```

```
    FETCH loan_cursor INTO v_loan_id, v_amount, v_old_rate;
```

```
    EXIT WHEN loan_cursor%NOTFOUND;
```

```
-- Apply new interest rate policy
```

```
  IF v_amount < 10000 THEN
```

```
    v_new_rate := 6;
```

```
  ELSIF v_amount BETWEEN 10000 AND 50000 THEN
```

```
    v_new_rate := 5;
```

```
  ELSE
```

```
    v_new_rate := 4.5;
```

```
  END IF;
```

```
-- Update only if rate has changed
```

```
  IF v_new_rate != v_old_rate THEN
```

```
    UPDATE Loans
```

```
      SET InterestRate = v_new_rate
```

```
      WHERE LoanID = v_loan_id;
```

```
    DBMS_OUTPUT.PUT_LINE('Loan ID: ' || v_loan_id ||
```

```
      ' | Old Rate: ' || v_old_rate || '%' ||
```

```
      ' → New Rate: ' || v_new_rate || '%');
```

```
  END IF;
```

```
END LOOP;
```

```
CLOSE loan_cursor;
```

```
COMMIT;
```

```

DBMS_OUTPUT.PUT_LINE('Interest rate update complete.');
END;

```

--test procedure and display updated loan

```

SELECT LoanID, LoanAmount, InterestRate FROM Loans;

```

The screenshot shows the Oracle Live SQL interface. In the top navigation bar, 'Live SQL' is selected. The main area contains a SQL Worksheet with the following code:

```

34    END IF;
35    END LOOP;
36    CLOSE loan_cursor;
37
38    COMMIT;
39
40
41
42
43
44
45 --test procedure and display updated loan
46 SELECT LoanID, LoanAmount, InterestRate
47   FROM Loans;

```

The 'Script output' tab is selected, showing the results of the execution:

```

SQL> DECLARE
  CURSOR loan_cursor IS
    SELECT LoanID, LoanAmount, InterestRate
    FROM Loans;...
  Show more...

--- Updating Loan Interest Rates Based on New Policy ---
Loan ID: 6 | Old Rate: 3.0% → New Rate: 5%
Loan ID: 4 | Old Rate: 0% → New Rate: 6%
Loan ID: 5 | Old Rate: 2.5% → New Rate: 6%
Loan ID: 2 | Old Rate: 4.5% → New Rate: 6%
Loan ID: 3 | Old Rate: 5.0% → New Rate: 5%
Interest rate update complete.

PL/SQL procedure successfully completed.

Elapsed: 00:00:00.016

```

The bottom status bar shows the session ID 'r31.1' and the system date/time '6/28/2025 8:49 AM'.

Table:

The screenshot shows the Oracle Live SQL interface. The SQL Worksheet contains the same code as before, but the 'Query result' tab is selected, showing the following table:

LOANID	LOANAMOUNT	INTERESTRATE	
1	6	20000	5
2	4	9000	6
3	5	2500	6
4	2	7000	6
5	3	15000	5

The bottom status bar shows the session ID 'r31.1' and the system date/time '6/28/2025 8:54 AM'.

## 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.

**Procedure:**

```
CREATE OR REPLACE PACKAGE CustomerManagement AS
```

```
    PROCEDURE AddCustomer(  
        p_customer_id IN NUMBER,  
        p_name      IN VARCHAR2,  
        p_dob       IN DATE,  
        p_balance   IN NUMBER  
    );
```

```
    PROCEDURE UpdateCustomerDetails(  
        p_customer_id IN NUMBER,  
        p_name      IN VARCHAR2,  
        p_dob       IN DATE  
    );
```

```
    FUNCTION GetCustomerBalance(  
        p_customer_id IN NUMBER  
    ) RETURN NUMBER;  
END CustomerManagement;
```

```
CREATE OR REPLACE PACKAGE BODY CustomerManagement AS
```

```
    PROCEDURE AddCustomer (  
        p_customer_id IN NUMBER,  
        p_name      IN VARCHAR2,  
        p_dob       IN DATE,  
        p_balance   IN NUMBER  
    ) IS  
        BEGIN  
            INSERT INTO Customers (CustomerID, Name, DOB, Balance, LastModified)  
            VALUES (p_customer_id, p_name, p_dob, p_balance, SYSDATE);  
  
            COMMIT;  
            DBMS_OUTPUT.PUT_LINE('Customer added successfully. ID: ' || p_customer_id);  
        EXCEPTION  
            WHEN DUP_VAL_ON_INDEX THEN  
                DBMS_OUTPUT.PUT_LINE('Error: Customer with ID ' || p_customer_id || ' already exists.');
```

```

WHEN OTHERS THEN
    DBMS_OUTPUT.PUT_LINE('Unexpected error: ' || SQLERRM);
END;

PROCEDURE UpdateCustomerDetails (
    p_customer_id IN NUMBER,
    p_name      IN VARCHAR2,
    p_dob       IN DATE
) IS
BEGIN
    UPDATE Customers
    SET Name = p_name,
        DOB = p_dob,
        LastModified = SYSDATE
    WHERE CustomerID = p_customer_id;

    IF SQL%ROWCOUNT = 0 THEN
        DBMS_OUTPUT.PUT_LINE('Customer ID ' || p_customer_id || ' not found.');
    ELSE
        COMMIT;
        DBMS_OUTPUT.PUT_LINE('Customer details updated for ID: ' || p_customer_id);
    END IF;
END;

FUNCTION GetCustomerBalance (
    p_customer_id IN NUMBER
) RETURN NUMBER IS
    v_balance NUMBER;
BEGIN
    SELECT Balance INTO v_balance
    FROM Customers
    WHERE CustomerID = p_customer_id;

    RETURN v_balance;
EXCEPTION
    WHEN NO_DATA_FOUND THEN
        DBMS_OUTPUT.PUT_LINE('Customer not found.');
        RETURN NULL;
    WHEN OTHERS THEN
        DBMS_OUTPUT.PUT_LINE('Error: ' || SQLERRM);
        RETURN NULL;
END;

END CustomerManagement;

```

```

8   PROCEDURE UpdateCustomerDetails(
9     p_customer_id IN NUMBER,
10    p_name ..... IN VARCHAR2,
11    p_dob ..... IN DATE
12  );
13
14
15  FUNCTION GetCustomerBalance(
16    p_customer_id IN NUMBER
17  ) RETURN NUMBER;
18 END CustomerManagement;
19

```

Query result    Script output    DBMS output    Explain Plan    SQL history

Elapsed: 00:00:00.016

```

SQL> CREATE OR REPLACE PACKAGE CustomerManagement AS
  PROCEDURE AddCustomer(
    p_customer_id IN NUMBER,
    p_name ..... IN VARCHAR2,...);
  Show more...

```

Package CUSTOMERMANAGEMENT compiled

Elapsed: 00:00:00.012

[About Oracle](#) | [Contact Us](#) | [Legal Notices](#) | [Terms and Conditions](#) | [Your Privacy Rights](#) | [Delete Your Live SQL Account](#) | [Cookie Preferences](#)

Copyright © 2014, 2025 Oracle and/or its affiliates. All rights reserved.

r31.1

--to add customer

BEGIN

CustomerManagement.AddCustomer(12, 'Ravi Verma', TO\_DATE('1990-05-10', 'YYYY-MM-DD'), 7000);  
END;

--to update customer

BEGIN

CustomerManagement.UpdateCustomerDetails(12, 'R. Verma', TO\_DATE('1990-05-10', 'YYYY-MM- DD'));

END;

--to get balance

DECLARE

  v\_balance NUMBER;

BEGIN

  v\_balance := CustomerManagement.GetCustomerBalance(12);

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

END;

The screenshot shows the Oracle Live SQL interface. In the Navigator pane, 'My Schema' is selected. The SQL Worksheet contains the following PL/SQL code:

```

1 --to add customer
2 BEGIN
3   CustomerManagement.AddCustomer(12, 'Ravi Verma', TO_DATE('1990-05-10', 'YYYY-MM-DD'), 7000);
4 END;
5 --to update customer
6 BEGIN

```

The output window shows:

- Error: Customer with ID 12 already exists.
- PL/SQL procedure successfully completed.
- Elapsed: 00:00:00.015

The screenshot shows the Oracle Live SQL interface. In the Navigator pane, 'My Schema' is selected. The SQL Worksheet contains the following PL/SQL code:

```

8   END;
9   --to get balance
10
11  DECLARE
12    v_balance NUMBER;
13  BEGIN

```

The output window shows:

- Customer details updated for ID: 12
- PL/SQL procedure successfully completed.
- Elapsed: 00:00:00.012

Below the code, there is a continuation of the procedure:

```

SQL> DECLARE
      v_balance NUMBER;
      BEGIN
        v_balance := CustomerManagement.GetCustomerBalance(12);
      ...
Show more...

```

Output:

- Balance: ₹7000
- PL/SQL procedure successfully completed.
- Elapsed: 00:00:00.077

## 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.

### Procedure:

CREATE OR REPLACE PACKAGE EmployeeManagement AS

-- Procedure to hire a new employee

PROCEDURE HireEmployee(

```

p_emp_id IN NUMBER,
p_name IN VARCHAR2,
p_position IN VARCHAR2,
p_salary IN NUMBER,
p_department IN VARCHAR2,
p_hire_date IN DATE
);

-- Procedure to update existing employee details
PROCEDURE UpdateEmployeeDetails(
    p_emp_id IN NUMBER,
    p_name IN VARCHAR2,
    p_position IN VARCHAR2,
    p_salary IN NUMBER,
    p_department IN VARCHAR2
);

-- Function to calculate annual salary
FUNCTION GetAnnualSalary(
    p_emp_id IN NUMBER
) RETURN NUMBER;
END EmployeeManagement;

```

CREATE OR REPLACE PACKAGE BODY EmployeeManagement AS

```

PROCEDURE HireEmployee (
    p_emp_id IN NUMBER,
    p_name IN VARCHAR2,
    p_position IN VARCHAR2,
    p_salary IN NUMBER,
    p_department IN VARCHAR2,
    p_hire_date IN DATE
) IS
BEGIN
    INSERT INTO Employees (
        EmployeeID, Name, Position, Salary, Department, HireDate
    ) VALUES (
        p_emp_id, p_name, p_position, p_salary, p_department, p_hire_date
    );
    COMMIT;
    DBMS_OUTPUT.PUT_LINE('Hired employee: ' || p_name);
EXCEPTION

```

```

WHEN DUP_VAL_ON_INDEX THEN
    DBMS_OUTPUT.PUT_LINE('Error: Employee with ID ' || p_emp_id || ' already exists.');
WHEN OTHERS THEN
    DBMS_OUTPUT.PUT_LINE('Unexpected error: ' || SQLERRM);
END;

PROCEDURE UpdateEmployeeDetails (
    p_emp_id    IN NUMBER,
    p_name      IN VARCHAR2,
    p_position   IN VARCHAR2,
    p_salary     IN NUMBER,
    p_department IN VARCHAR2
) IS
BEGIN
    UPDATE Employees
    SET Name = p_name,
        Position = p_position,
        Salary = p_salary,
        Department = p_department
    WHERE EmployeeID = p_emp_id;

    IF SQL%ROWCOUNT = 0 THEN
        DBMS_OUTPUT.PUT_LINE('Employee ID ' || p_emp_id || ' not found.');
    ELSE
        COMMIT;
        DBMS_OUTPUT.PUT_LINE('Employee details updated for ID: ' || p_emp_id);
    END IF;
END;

FUNCTION GetAnnualSalary (
    p_emp_id IN NUMBER
) RETURN NUMBER IS
    v_salary Employees.Salary%TYPE;
BEGIN
    SELECT Salary INTO v_salary
    FROM Employees
    WHERE EmployeeID = p_emp_id;

    RETURN v_salary * 12;
EXCEPTION
    WHEN NO_DATA_FOUND THEN
        DBMS_OUTPUT.PUT_LINE('Employee not found.');
        RETURN NULL;
    WHEN OTHERS THEN

```

```

    DBMS_OUTPUT.PUT_LINE('Error: ' || SQLERRM);
    RETURN NULL;
END;

END EmployeeManagement;

```

The screenshot shows the Oracle Live SQL interface. The code in the editor is:

```

42: IF :SQL%ROWCOUNT = 0 THEN
43:   DBMS_OUTPUT.PUT_LINE('Employee ID: ' || p_emp_id || ' not found.');
44: ELSE
45:   COMMIT;
46:   DBMS_OUTPUT.PUT_LINE('Employee details updated for ID: ' || p_emp_id);
47: END IF;

```

The results pane shows the package and its body being compiled. The elapsed time for compilation is 00:00:00.016.

Below the results, there is a note about the package body being compiled with an elapsed time of 00:00:00.018.

At the bottom, the Windows taskbar shows the system status: 31°C Sunny, ENG, 10:44 AM, 6/29/2025.

## Testing package

--to hire an employee

BEGIN

```

EmployeeManagement.HireEmployee(
  101, 'Neha Sharma', 'Analyst', 45000, 'Finance', TO_DATE('2022-08-01', 'YYYY-MM-DD')
);

```

END;

--to update employee

BEGIN

```

EmployeeManagement.UpdateEmployeeDetails(
  101, 'Neha S.', 'Senior Analyst', 50000, 'Finance'
);

```

END;

--to get salary

DECLARE

```

  v_annual_salary NUMBER;

```

```

BEGIN
  v_annual_salary := EmployeeManagement.GetAnnualSalary(101);
  DBMS_OUTPUT.PUT_LINE('Annual Salary: ₹' || v_annual_salary);
END;

```

The screenshot shows the Oracle Live SQL interface. In the SQL Worksheet, a PL/SQL block is run:

```

5   );
6   END;
7   --to update employee
8   BEGIN
9     EmployeeManagement.UpdateEmployeeDetails(
10    | 101, 'Neha S.', 'Senior Analyst', 50000, 'Finance'

```

The output pane shows:

- Hired employee: Neha Sharma
- PL/SQL procedure successfully completed.
- Elapsed: 00:00:00.013

Another execution of the same block is shown below:

```

SQL> BEGIN
      EmployeeManagement.UpdateEmployeeDetails(
        101, 'Neha S.', 'Senior Analyst', 50000, 'Finance'
      );
    ...
Show more...

```

The output is identical to the first execution.

The screenshot shows the Oracle Live SQL interface. In the SQL Worksheet, a PL/SQL block is run:

```

16  v_annual_salary NUMBER;
17  BEGIN
18    v_annual_salary := EmployeeManagement.GetAnnualSalary(101);
19    DBMS_OUTPUT.PUT_LINE('Annual Salary: ₹'||v_annual_salary);
20  END;

```

The output pane shows:

- PL/SQL procedure successfully completed.
- Elapsed: 00:00:00.013

Another execution of the same block is shown below:

```

SQL> DECLARE
      v_annual_salary NUMBER;
    BEGIN
      v_annual_salary := EmployeeManagement.GetAnnualSalary(101);
    ...
Show more...

```

The output shows the variable being assigned the value \$600000.

**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.

**Procedure:**

```
CREATE OR REPLACE PACKAGE AccountOperations AS  
    -- Procedure to open a new account  
    PROCEDURE OpenAccount(  
        p_account_id IN NUMBER,  
        p_customer_id IN NUMBER,  
        p_account_type IN VARCHAR2,  
        p_balance IN NUMBER  
    );  
  
    -- Procedure to close an account  
    PROCEDURE CloseAccount(  
        p_account_id IN NUMBER  
    );  
  
    -- Function to get total balance for a customer across all accounts  
    FUNCTION GetTotalBalance(  
        p_customer_id IN NUMBER  
    ) RETURN NUMBER;  
END AccountOperations;
```

```
CREATE OR REPLACE PACKAGE BODY AccountOperations AS
```

```
    PROCEDURE OpenAccount (  
        p_account_id IN NUMBER,  
        p_customer_id IN NUMBER,  
        p_account_type IN VARCHAR2,  
        p_balance IN NUMBER  
    ) IS  
        BEGIN  
            INSERT INTO Accounts (  
                AccountID, CustomerID, AccountType, Balance, LastModified  
            ) VALUES (  
                p_account_id, p_customer_id, p_account_type, p_balance, SYSDATE  
            );  
  
            COMMIT;  
            DBMS_OUTPUT.PUT_LINE('Account opened successfully: ' || p_account_id);
```

```

EXCEPTION
WHEN DUP_VAL_ON_INDEX THEN
  DBMS_OUTPUT.PUT_LINE('Error: Account ID ' || p_account_id || ' already exists.');
WHEN OTHERS THEN
  DBMS_OUTPUT.PUT_LINE('Unexpected error: ' || SQLERRM);
END;

PROCEDURE CloseAccount (
  p_account_id IN NUMBER
) IS
BEGIN
  DELETE FROM Accounts
  WHERE AccountID = p_account_id;

  IF SQL%ROWCOUNT = 0 THEN
    DBMS_OUTPUT.PUT_LINE('Account ID ' || p_account_id || ' not found.');
  ELSE
    COMMIT;
    DBMS_OUTPUT.PUT_LINE('Account ID ' || p_account_id || ' closed.');
  END IF;
END;

FUNCTION GetTotalBalance (
  p_customer_id IN NUMBER
) RETURN NUMBER IS
  v_total NUMBER := 0;
BEGIN
  SELECT NVL(SUM(Balance), 0)
  INTO v_total
  FROM Accounts
  WHERE CustomerID = p_customer_id;

  RETURN v_total;
EXCEPTION
  WHEN OTHERS THEN
    DBMS_OUTPUT.PUT_LINE('Error: ' || SQLERRM);
    RETURN NULL;
END;

END AccountOperations;

```

```

47   FROM Accounts
48   WHERE CustomerID = p_customer_id;
49
50   RETURN v_total;
51
52  EXCEPTION
      WHEN OTHERS THEN

```

Query result    Script output    DBMS output    Explain Plan    SQL history

```

1 Procedure OpenAccount(
2     p_account_id IN NUMBER, ...
3
4 Show more...
5
6 Package ACCOUNTOPERATIONS compiled
7 Elapsed: 00:00:00.015
8
9 SQL> CREATE OR REPLACE PACKAGE BODY AccountOperations AS
10
11     PROCEDURE OpenAccount (
12         p_account_id IN NUMBER, ...
13
14     Show more...
15
16     Package Body ACCOUNTOPERATIONS compiled
17
18     Elapsed: 00:00:00.018

```

About Oracle | Contact Us | Legal Notices | Terms and Conditions | Your Privacy Rights | Delete Your Live SQL Account | Cookie Preferences  
Copyright © 2014, 2025 Oracle and/or its affiliates. All rights reserved.

## Testing Account Operation

--to open account

BEGIN

    AccountOperations.OpenAccount(10, 4, 'Savings', 3000);

END;

--to close account

BEGIN

    AccountOperations.CloseAccount(10);

END;

--to get balance

DECLARE

    v\_balance NUMBER;

BEGIN

    v\_balance := AccountOperations.GetTotalBalance(3);

    DBMS\_OUTPUT.PUT\_LINE('Total Balance for Customer 3: #' || v\_balance);

END;

The screenshot shows the Oracle Live SQL interface. On the left, the Navigator pane lists objects: ACCOUNTS, AUDITLOG, CUSTOMERS, EMPLOYEES, LOANS, and TRANSACTIONS. The main area is a SQL Worksheet titled '[SQL Worksheet]\*'. It contains the following PL/SQL script:

```
1  AccountOperations.OpenAccount(10, 4, "Savings", 3000);
2  END;
3  --
4  --to close account
5  BEGIN
6      AccountOperations.CloseAccount(10);
7  END;
```

The output shows the account was opened successfully with ID 10, and then closed successfully.

```
Account opened successfully: 10
PL/SQL procedure successfully completed.
Elapsed: 00:00:00.007

SQL> BEGIN
   2      AccountOperations.CloseAccount(10);
   3  END;

Account ID 10 closed.

PL/SQL procedure successfully completed.
Elapsed: 00:00:00.010
```

About Oracle | Contact Us | Legal Notices | Terms and Conditions | Your Privacy Rights | Delete Your Live SQL Account | Cookie Preferences  
Copyright © 2014, 2025 Oracle and/or its affiliates. All rights reserved.

r31.1

The screenshot shows the Oracle Live SQL interface. The Navigator pane lists the same objects as the first screenshot. The main area is a SQL Worksheet titled '[SQL Worksheet]\*'. It contains the following PL/SQL script:

```
15  v_balance := AccountOperations.GetTotalBalance(3);
16  DEMO_OUTPUT.PUT_LINE('Total Balance for Customer 3: '|| v_balance);
17  END;
18
19
```

The output shows the total balance for customer 3.

```
Account ID 10 closed.

PL/SQL procedure successfully completed.
Elapsed: 00:00:00.010

SQL> DECLARE
   2      v_balance NUMBER;
   3  BEGIN
   4      v_balance := AccountOperations.GetTotalBalance(3);
   5  END;
   6
   7  DEMO_OUTPUT.PUT_LINE('Total Balance for Customer 3: '|| v_balance);
   8
   9 Show more...

Total Balance for Customer 3: ₹11878

PL/SQL procedure successfully completed.
Elapsed: 00:00:00.008
```

About Oracle | Contact Us | Legal Notices | Terms and Conditions | Your Privacy Rights | Delete Your Live SQL Account | Cookie Preferences  
Copyright © 2014, 2025 Oracle and/or its affiliates. All rights reserved.

r31.1

## Schema to be 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  
    );
```

### **Example Scripts for Sample Data Insertion**

```
INSERT INTO Customers (CustomerID, Name, DOB, Balance, LastModified)  
VALUES (1, 'John Doe', TO_DATE('1985-05-15', 'YYYY-MM-DD'), 1000, SYSDATE);
```

```
INSERT INTO Customers (CustomerID, Name, DOB, Balance, LastModified)  
VALUES (2, 'Jane Smith', TO_DATE('1990-07-20', 'YYYY-MM-DD'), 1500, SYSDATE);
```

```
INSERT INTO Accounts (AccountID, CustomerID, AccountType, Balance, LastModified)  
VALUES (1, 1, 'Savings', 1000, SYSDATE);
```

```
INSERT INTO Accounts (AccountID, CustomerID, AccountType, Balance, LastModified)  
VALUES (2, 2, 'Checking', 1500, SYSDATE);
```

```
INSERT INTO Transactions (TransactionID, AccountID, TransactionDate, Amount, TransactionType)  
VALUES (1, 1, SYSDATE, 200, 'Deposit');
```

```
INSERT INTO Transactions (TransactionID, AccountID, TransactionDate, Amount, TransactionType)  
VALUES (2, 2, SYSDATE, 300, 'Withdrawal');
```

```
INSERT INTO Loans (LoanID, CustomerID, LoanAmount, InterestRate, StartDate, EndDate)  
VALUES (1, 1, 5000, 5, SYSDATE, ADD_MONTHS(SYSDATE, 60));
```

```
INSERT INTO Employees (EmployeeID, Name, Position, Salary, Department, HireDate)  
VALUES (1, 'Alice Johnson', 'Manager', 70000, 'HR', TO_DATE('2015-06-15', 'YYYY-MM-DD'));
```

```
INSERT INTO Employees (EmployeeID, Name, Position, Salary, Department, HireDate)  
VALUES (2, 'Bob Brown', 'Developer', 60000, 'IT', TO_DATE('2017-03-20', 'YYYY-MM-DD'));
```