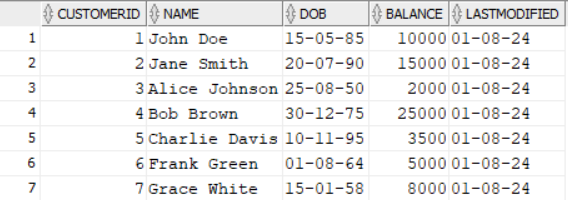
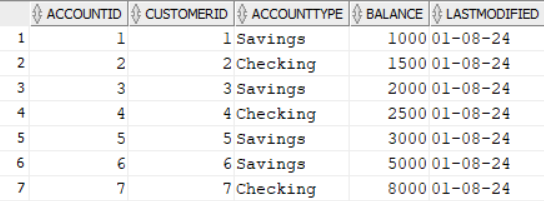
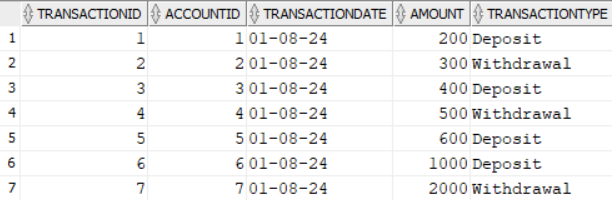
**NAME: SUBARNA PAUL**

**SUPERSET ID: 5008628**

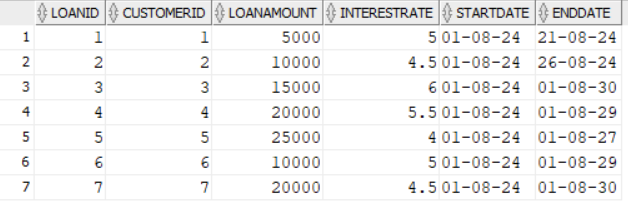
**Tables:** Initial Table creation & data insertion

** **

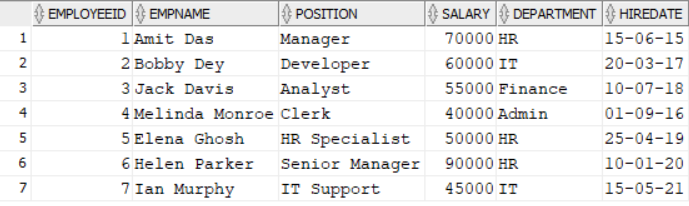
*Customers Accounts*

****

*Transactions*

****

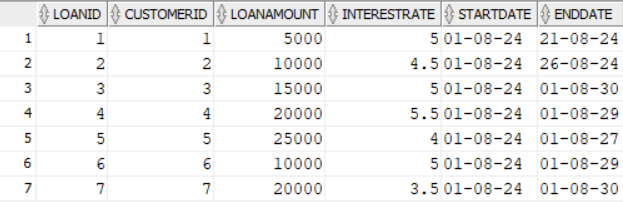
*Loans*

****

*Employees*

**Exercise 1: Control Structures**

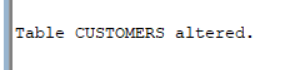
**Scenario 1: The bank wants to apply a discount to loan interest rates for customers above 60 years old.**

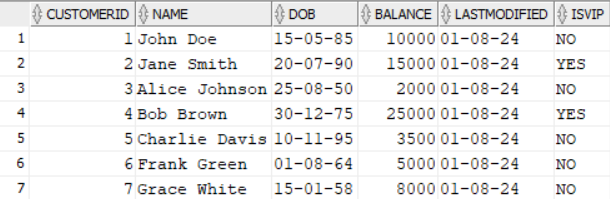
****

The Loans table will have a 1% reduced interest rate for loans belonging to customers over 60 years old.

Customer Id 7 is above 60 years old that’s why interest rate has become 3.5% from 4.5%.

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

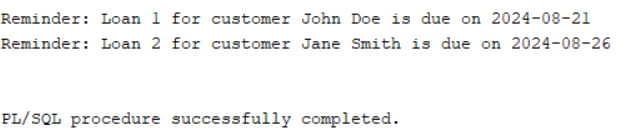




The Customers table will have the IsVIP flag set to 'YES' for customers with a balance over $10,000.

Customer Id 2 and 4 has bank balance over $10,000 that’s why IsVIP flag is YES and rest NO.

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

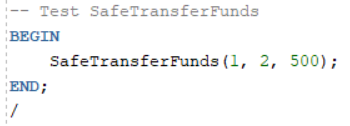
****

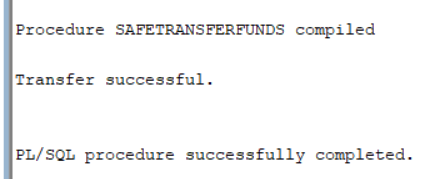
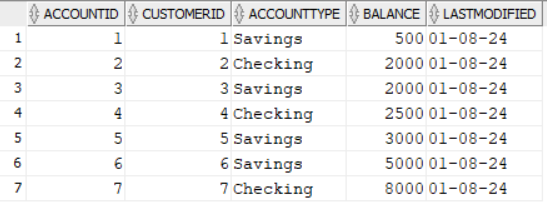
Reminder for the loans for customers 1 and 2 with due dates within the next 30 days.

**Exercise 2: Error Handling**

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

‘SafeTransferFunds’: This procedure transfers funds between two accounts. It checks if the accounts exist and if there are sufficient funds before performing the transfer. Errors are handled and logged appropriately.

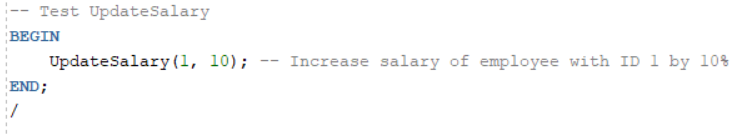


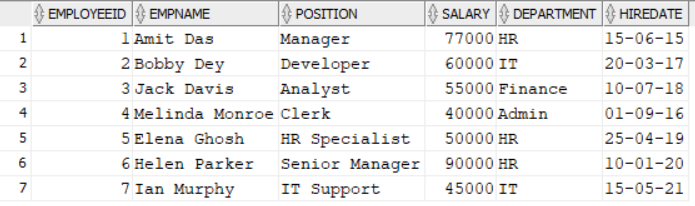
 

1. **Both Accounts Exist and Sufficient Funds Are Available:**
   1. Account 1 has a balance of at least 500.
   2. Account 2 exists.
   3. The procedure will transfer 500 from account 1 to account 2, commit the changes, and print: Transfer successful.
2. **Account Does Not Exist:**
   1. If either account 1 or 2 does not exist, the procedure will print: Error: Account does not exist. and roll back any changes.
3. **Insufficient Funds:**
   1. If account 1 has a balance less than 500, the procedure will print: Error: Insufficient funds in the "from" account. and roll back any changes.
4. **Other Errors:**
   1. If an unexpected error occurs, the procedure will print a generic error message: Error: <SQLERRM>

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

‘UpdateSalary’: This procedure increases an employee's salary by a given percentage. It handles cases where the employee ID does not exist and logs errors accordingly.





Previously Amit Das’s had a salary of 70,000. Now after updating salary of Amit Das has become 77,000.

**1. Employee Exists and Salary Update is Successful:**

* + Employee with EmployeeID 1 exists in the Employees table.
  + The salary increase is successfully applied.
  + The procedure will update the salary of the employee by the specified percentage, commit the changes and print: Salary updated successfully.

**2. Employee Does Not Exist:**

* + No employee with EmployeeID 1 exists in the Employees table.
  + The procedure will print: Error: Employee ID does not exist.

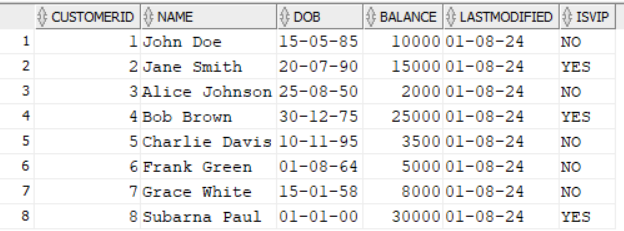
**3. Other Errors:**

* + Any unexpected errors occur (e.g., database issues, constraint violations).
  + The procedure will print a generic error message: Error: <SQLERRM>

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

‘AddNewCustomer’: This procedure adds a new customer to the database. It checks for existing customer IDs to avoid duplication and logs any errors encountered.





A new customer is added i.e. customer id 8

**1. CustomerID does not Exists and Addition is Successful:**

* + Customer with CustomerID 8 does not exist in the Customers table.
  + The procedure will add the Customer by the specified values, commit the changes and print: Customer added successfully.

**2. Customer Id already Exists:**

* + Employee with EmployeeID exists in the Employees table.
  + The procedure will print: Error: Error: Customer with this ID already exists.

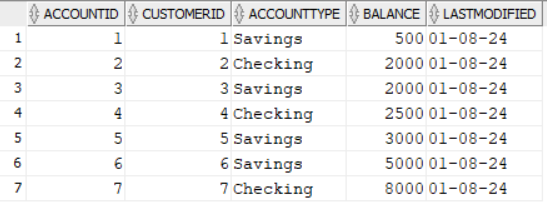
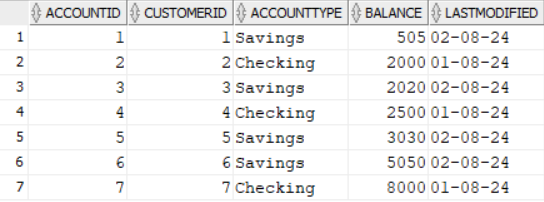
**3. Other Errors:**

* + For unexpected errors procedure will print a generic error message: Error: <SQLERRM>

**Exercise 3: Stored Procedures**

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

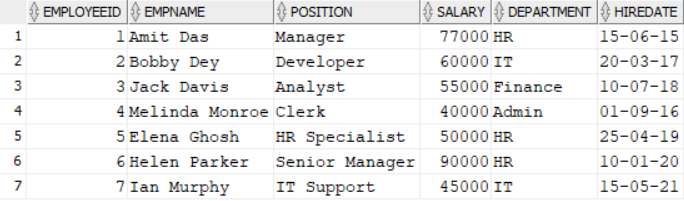
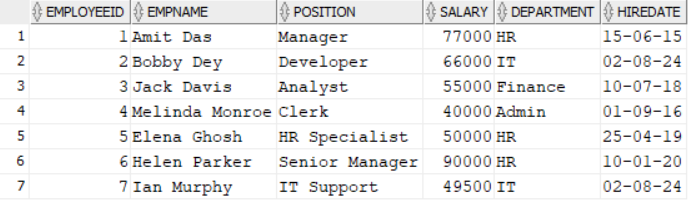
ProcessMonthlyInterest: Apply a 1% interest rate to all savings accounts and update their balances.

 ****

*Before Updating After Updating*

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

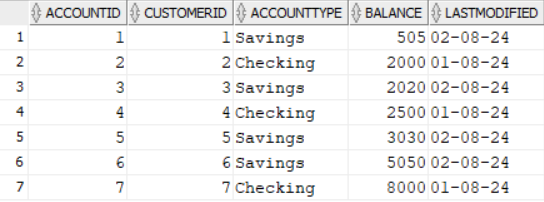
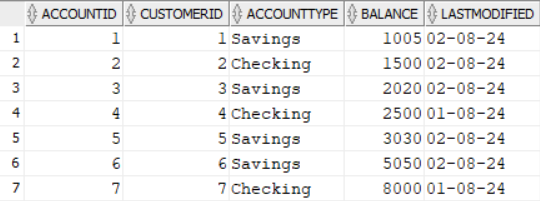
UpdateEmployeeBonus: Update the salaries of employees in a specified department by adding a bonus percentage to their current salaries.

** **

*Before Updating After Updating*

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

TransferFunds: Transfer funds between two accounts, ensuring the source account has sufficient balance before making the transfer.

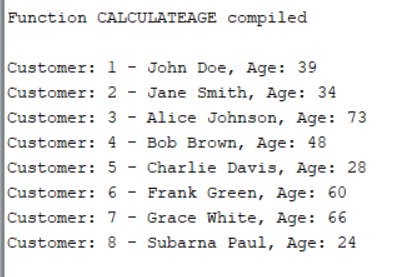
** **

*Before Updating After Updating*

**Exercise 4: Functions**

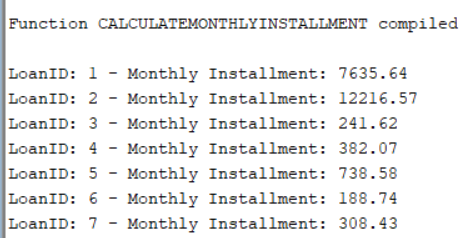
**Scenario 1: Calculate the age of customers for eligibility checks.**

CalculateAge: computes the age of a customer based on their date of birth.

****

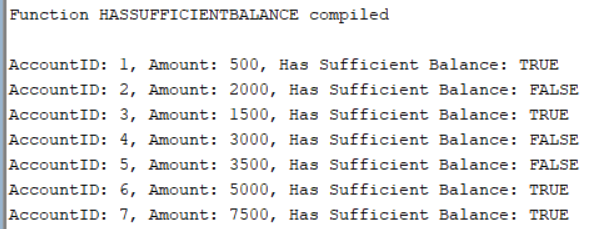
**Scenario 2: The bank needs to compute the monthly installment for a loan.**

CalculateMonthlyInstallment: calculates the monthly payment for a loan given the loan amount, interest rate, and duration.

****

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

HasSufficientBalance: checks if an account has enough balance to cover a specified amount.

****

**Exercise 5: Triggers**

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

UpdateCustomerLastModified: Updates the LastModified column of the Customers table to the current date whenever a customer's record is updated.

****

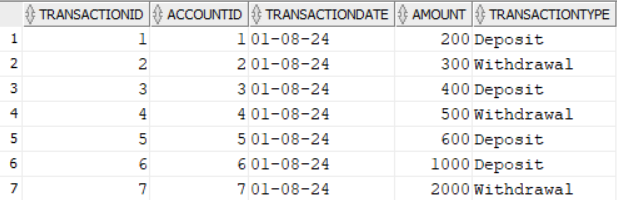
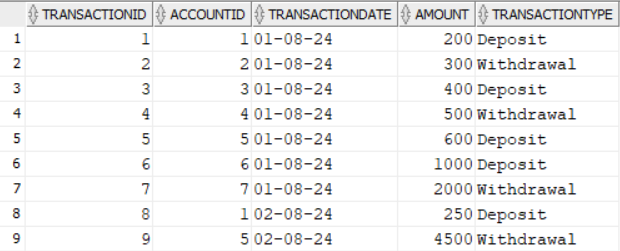
*Before Updating*

****

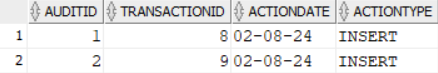
*After Updating*

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

LogTransaction: Inserts a record into an AuditLog table whenever a transaction is inserted into the Transactions table.

** **

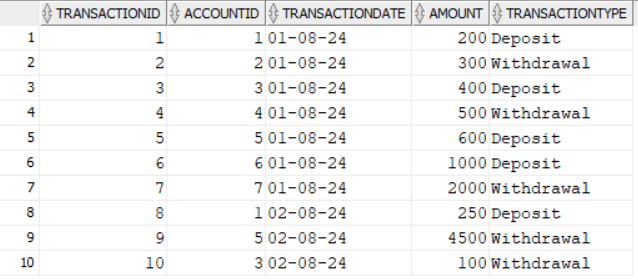
*Before Updating After Updating*

****

*AuditLog Table*

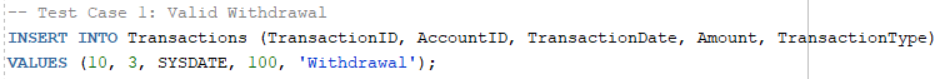
**Scenario 3: Enforce business rules on deposits and withdrawals.**

CheckTransactionRules: Ensures withdrawals do not exceed the balance and deposits are positive before inserting a record into the Transactions table.

****

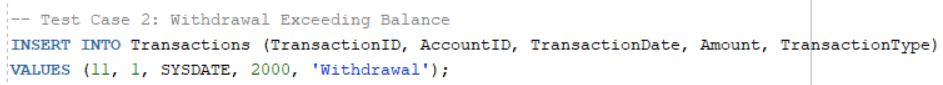
*Before Updating*

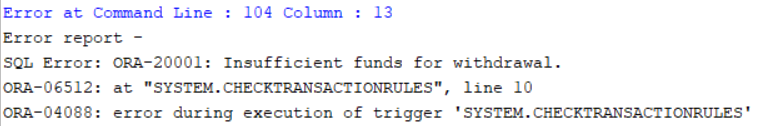
**Test case 1:**

****

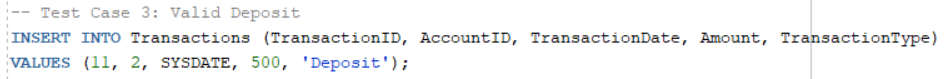
****

**Test case 2:**

****

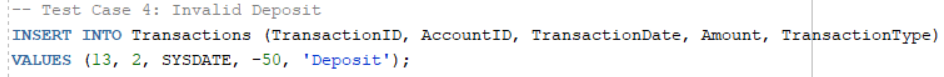
****

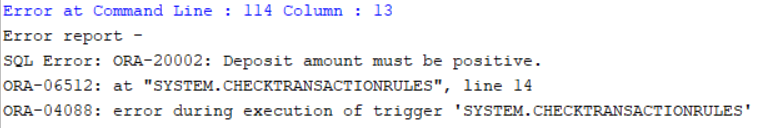
**Test case 3:**

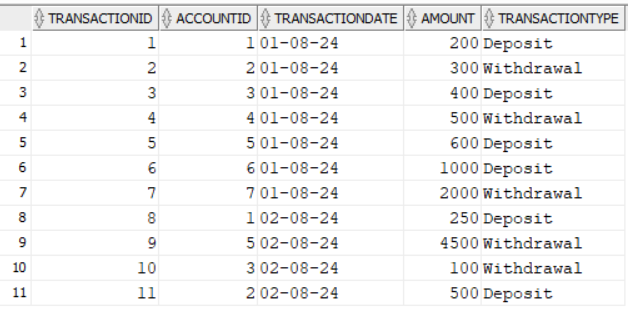
****

****

**Test case 4:**

****

****

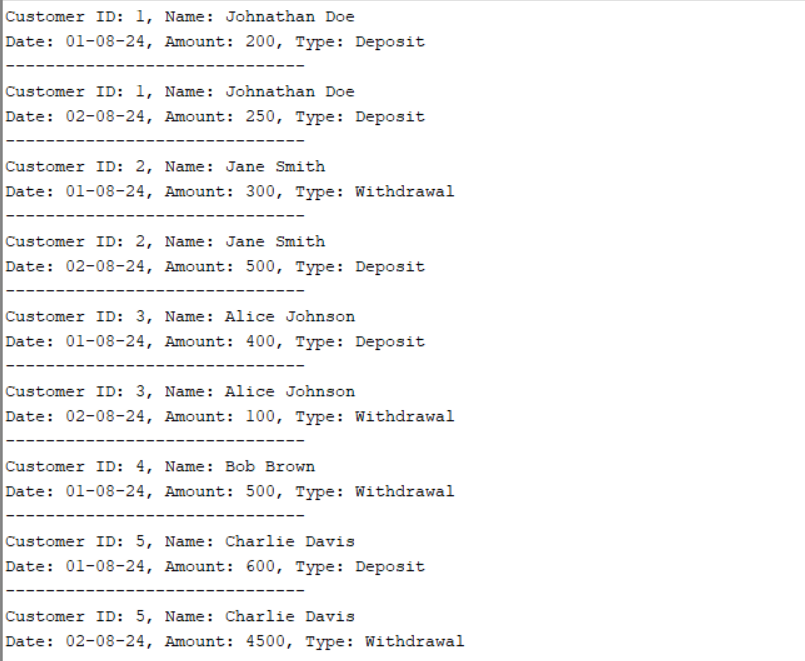
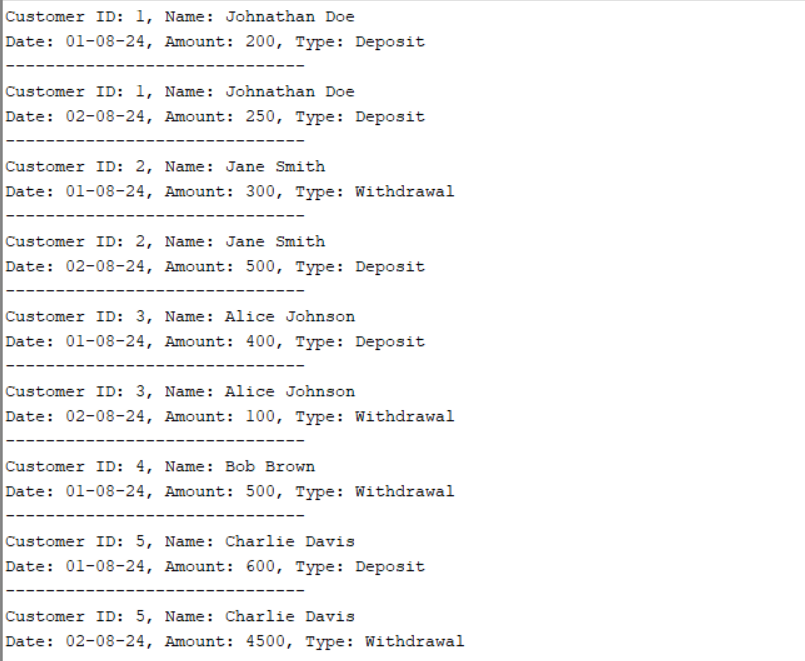
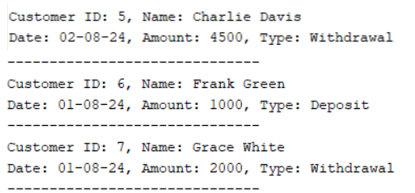
****

*After Updating*

**Exercise 6: Cursors**

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

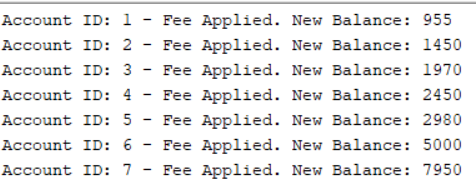
GenerateMonthlyStatements: Retrieves all transactions for the current month and prints a statement for each customer.

*Monthly Statements of each Customer*

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

ApplyAnnualFee: Deducts an annual maintenance fee from the balance of all accounts.

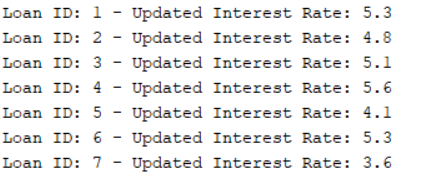


*Updated Balances*

We are applying a fixed annual maintenance fee of 50 to all accounts. This fee is deducted from the current balance of each account using an explicit cursor to iterate through each account and update its balance accordingly.

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

UpdateLoanInterestRates: Fetches all loans and updates their interest rates based on the new policy.



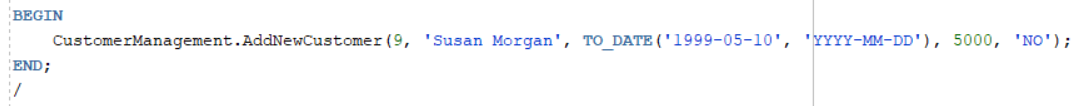
*Updated Interests*

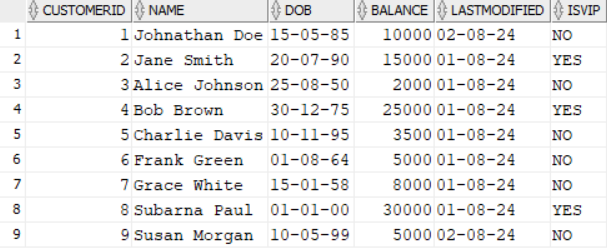
We are updating the interest rates for all loans based on the loan amount. Loans under 5,000 get a 0.5% increase, loans between 5,000 and 10,000 get a 0.3% increase, and loans over 10,000 get a 0.1% increase. This is done using an explicit cursor to update each loan.

**Exercise 7: Packages**

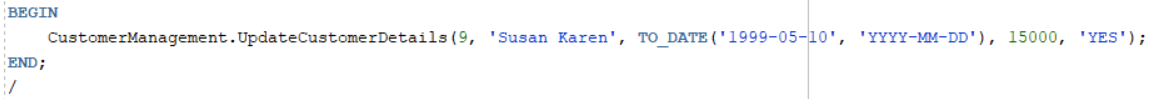
**Scenario 1: Group all customer-related procedures and functions into a package.**

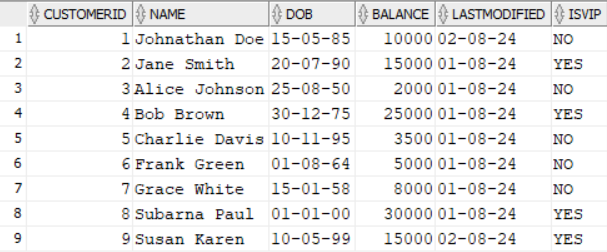
CustomerManagement: A package with procedures for adding a new customer, updating customer details, and a function to get customer balance.



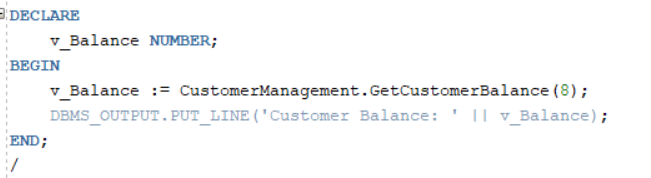


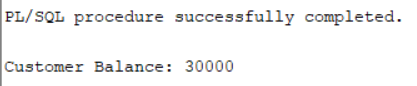
*Adding a new customer*





*Updating a customer’s details*

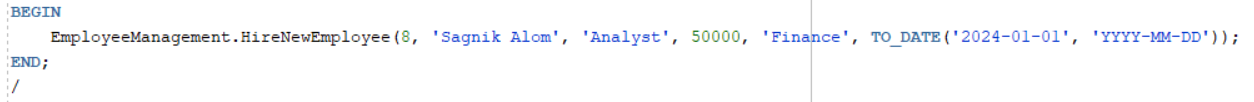


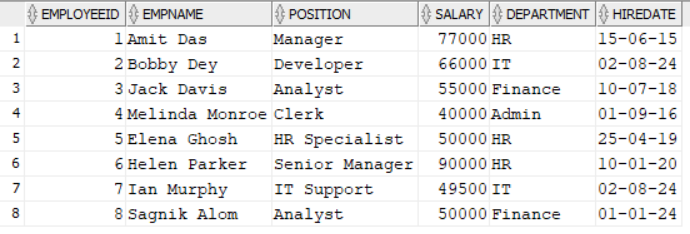


*Showing Balance of Customer Id 8*

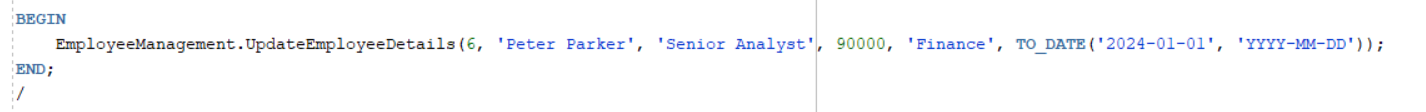
**Scenario 2: Create a package to manage employee data.**

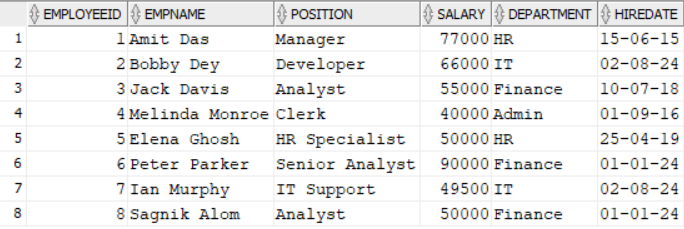
EmployeeManagement: A package with procedures to hire new employees, update employee details, and a function to calculate annual salary.



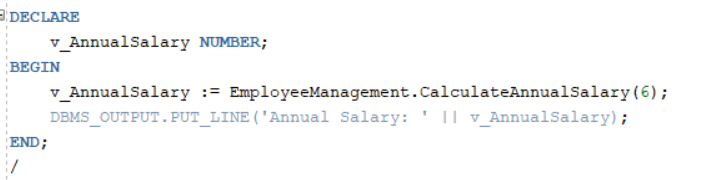


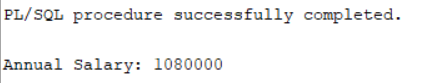
*Hiring a new employee*





*Updating employee details*

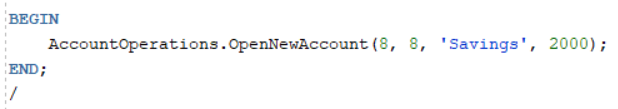


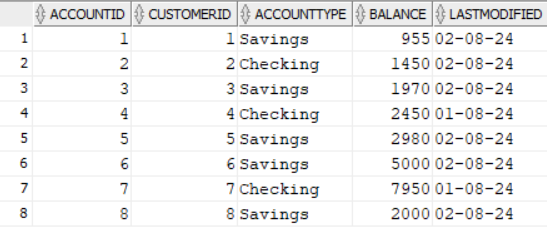


*Showing annual income of Employee whose ID is* *6*

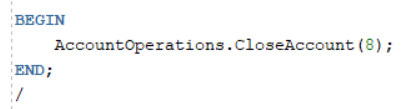
**Scenario 3: Group all account-related operations into a package.**

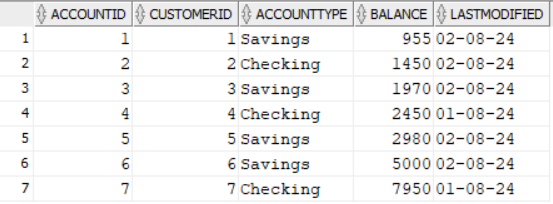
AccountOperations: A package with procedures for opening a new account, closing an account, and a function to get the total balance of a customer across all accounts.



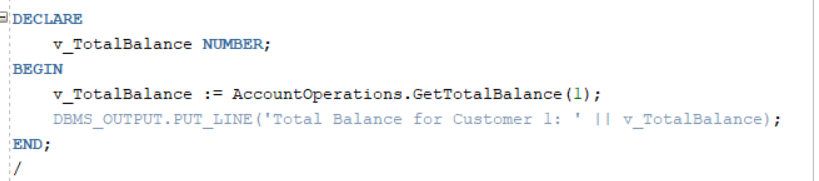


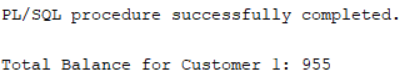
*Adding new account*





*Closing the created account*





*Showing total balance of customer whose ID is 1*