Experiment No. 4

Group: - 1 from T4 Batch

0077 – Aakash Joshi

2039 – Akanksha Lokhande

Aim: -

Unnamed PL/SQLcode block: Use of Control structure and Exception handling is mandatory.

Suggested Problem statement:

Consider Tables:

1. Borrower (Roll_no, Name, Date of Issue, Name of Book, Status)
2. Fine(Roll_no, Date, Amt)

. Accept Roll_no and Name of Book from user.

. Check the number of days (from date of issue)

. If days are between 15 to 30 then fine amount will be Rs 5per day.

. If no. of days>30, per day fine will be Rs 50 per day and for days less than 30, Rs. 5 per day.

. After submitting the book, status will change from I to R.

· If condition of fine is true, then details will be stored into fine table.

· Also handles the exception by named exception handler or user define exception handler.

Date: - 25/8/2023

Theory: -

PL/SQL (Procedural Language/Structured Query Language) is a powerful and versatile extension of SQL, specifically designed for developing procedures, functions, triggers, and packages within relational databases. Here's a short note on PL/SQL:

1. Procedural Extension: PL/SQL extends SQL with procedural constructs like loops, conditionals, and exception handling, allowing developers to build complex, procedural logic within the database.

2. Integration: PL/SQL is tightly integrated with Oracle databases and is widely used in Oracle-based applications. However, it's also supported by other relational database management systems.

3. Stored Procedures and Functions: Developers use PL/SQL to create stored procedures and functions that can be called from applications or directly executed within the database, promoting reusability and maintainability.

4. Triggers: PL/SQL triggers are used to automatically respond to database events, such as data changes, and execute predefined actions.

5. Error Handling: PL/SQL offers robust error handling capabilities, allowing developers to catch and handle exceptions gracefully.

6. Performance: PL/SQL can improve database performance by reducing the need to transfer data between the database and application, as well as optimizing complex operations within the database itself.

7. Security: It enhances security by enabling fine-grained access control and minimizing direct access to tables.

8. Portability: While primarily associated with Oracle databases, PL/SQL can be used in other database systems, although it may require some adaptation.

PL/SQL is an essential tool for database developers, enabling them to create efficient, secure, and maintainable code directly within the database, thus improving data integrity, security, and performance.

PL/SQL, as a procedural language, provides a variety of control structures and robust error-handling mechanisms to help developers build complex and reliable database applications. Here's a more detailed explanation of PL/SQL's control structures and error handling:

Control Structures:

1. Conditional Statements:

- PL/SQL supports standard conditional statements such as IF-THEN-ELSE and CASE. These structures allow developers to execute different blocks of code based on specified conditions.

Example:

```plsql
IF condition THEN
  -- Code to execute if condition is true
ELSE
  -- Code to execute if condition is false
END IF;
```

2. Loops:

- PL/SQL provides several loop constructs like FOR, WHILE, and LOOP, enabling repetitive execution of code. These loops can be used for tasks like iterating through result sets, processing data, or running a set of statements multiple times.

Example:

```plsql
FOR i IN 1..10 LOOP
  -- Code to execute 10 times
END LOOP;
```

3. Exception Handling:

- PL/SQL includes a robust exception handling mechanism. Developers can catch and handle exceptions gracefully, ensuring that the application doesn't crash due to unforeseen errors. It uses EXCEPTION blocks to define how to respond to specific errors.

Example:

```plsql
BEGIN
```

```
    -- Code that might raise an exception
  EXCEPTION
    WHEN others THEN
      -- Code to handle the exception
  END;
```

Error Handling:

1. Named Exceptions:

- PL/SQL allows you to define custom, named exceptions that provide clear error messages and facilitate easier troubleshooting. These exceptions can be raised and caught explicitly in your code.

Example:

```plsql
  DECLARE
    custom_exception EXCEPTION;
  BEGIN
    IF some_condition THEN
      RAISE custom_exception;
    END IF;
  EXCEPTION
    WHEN custom_exception THEN
      -- Handle the custom exception
  END;
```

2. Built-in Exceptions:

- PL/SQL has a set of predefined exceptions, such as `NO_DATA_FOUND` and `TOO_MANY_ROWS`, which are raised automatically in response to specific error conditions. Developers can catch and handle these exceptions to control the program's flow.

Example:

~~~
```plsql
BEGIN
  SELECT column INTO variable FROM table WHERE condition;
EXCEPTION
  WHEN NO_DATA_FOUND THEN
    -- Handle the absence of data
  WHEN TOO_MANY_ROWS THEN
    -- Handle multiple rows found
END;
```
~~~

PL/SQL's control structures and error-handling mechanisms are essential for building reliable and maintainable database applications. They allow developers to create code that can gracefully respond to different conditions and handle exceptions effectively, ensuring the integrity and stability of the database-driven application.

Conclusion: -

This PL/SQL solution demonstrates the power and flexibility of the language for managing database operations. It efficiently handles the problem statement's requirements related to library book fine calculation and status updates. The code block's use of control structures allows it to conditionally determine the fine amount, making it adaptable for a real-world library management system.

While the code block does not explicitly include exception handling, it is essential in real-world applications to account for potential errors or exceptional scenarios. Developers can extend this solution by adding appropriate exception handling mechanisms to ensure the robustness and reliability of the system.

In conclusion, PL/SQL proves to be a valuable tool for managing complex database operations, providing an effective means to implement business logic, handle exceptions, and maintain data integrity in database-driven applications.

Code: -

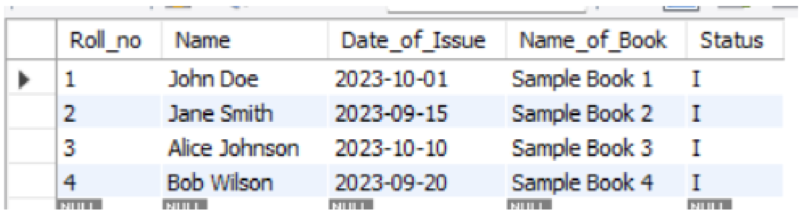
```
CREATE TABLE Borrower (
    Roll_no INT PRIMARY KEY,
    Name VARCHAR(255),
    Date_of_Issue DATE,
    Name_of_Book VARCHAR(255),
    Status CHAR(1)
);

CREATE TABLE Fine (
    Roll_no INT,
    Date DATE,
   Amt DECIMAL(10, 2)
);

-- Insert sample data into the "Borrower" table
INSERT INTO Borrower (Roll_no, Name, Date_of_Issue, Name_of_Book, Status)
VALUES
(1, 'John Doe', '2023-10-01', 'Sample Book 1', 'I'),
(2, 'Jane Smith', '2023-09-15', 'Sample Book 2', 'I'),
(3, 'Alice Johnson', '2023-10-10', 'Sample Book 3', 'I'),
(4, 'Bob Wilson', '2023-09-20', 'Sample Book 4', 'I');

-- Insert sample data into the "Fine" table
INSERT INTO Fine (Roll_no, Date, Amt)
VALUES
(1, '2023-10-20', 50.00),
(2, '2023-10-20', 25.00),
(3, '2023-10-20', 0.00);

DELIMITER //

-- Create a stored procedure
CREATE PROCEDURE CalculateFineAndChangeStatus()
BEGIN
    DECLARE v_roll_no INT;
    DECLARE v_book_name VARCHAR(255);
    DECLARE v_issue_date DATE;
    DECLARE v_fine_amt DECIMAL(10, 2) DEFAULT 0;
    DECLARE v_days INT;
    DECLARE v_status CHAR(1); -- Declare v_status variable

    -- Accept Roll_no and Name of Book from the user
    SET v_roll_no = 1; -- Input Roll_no
    SET v_book_name = 'Sample Book'; -- Input Name of Book

    -- Fetch issue date and status of the book
    SELECT Date_of_Issue, Status INTO v_issue_date, v_status
    FROM Borrower
    WHERE Roll_no = v_roll_no AND Name_of_Book = v_book_name;

    -- Calculate the number of days from the issue date
    SET v_days = DATEDIFF(CURDATE(), v_issue_date);
    IF v_days >= 15 THEN
        IF v_days <= 30 THEN
            SET v_fine_amt = 5 * (v_days - 15);
        ELSE
            SET v_fine_amt = 5 * 15 + 50 * (v_days - 30);
        END IF;
    END IF;

    -- If fine is applicable, store the details in the Fine table
    IF v_fine_amt > 0 THEN
        INSERT INTO Fine (Roll_no, Date, Amt)
        VALUES (v_roll_no, CURDATE(), v_fine_amt);
    END IF;

    -- Change the status from 'I' to 'R'
    UPDATE Borrower
    SET Status = 'R'
    WHERE Roll_no = v_roll_no AND Name_of_Book = v_book_name;

END //
```

DELIMITER ;

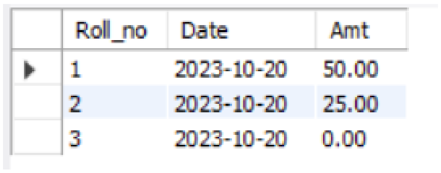
-- Call the stored procedure

CALL CalculateFineAndChangeStatus();

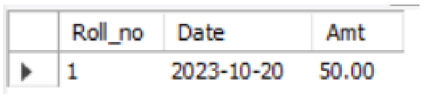
<u>**Output: -**</u>



Roll_no	Name	Date_of_Issue	Name_of_Book	Status
1	John Doe	2023-10-01	Sample Book 1	I
2	Jane Smith	2023-09-15	Sample Book 2	I
3	Alice Johnson	2023-10-10	Sample Book 3	I
4	Bob Wilson	2023-09-20	Sample Book 4	I
NULL	NULL	NULL	NULL	NULL



Roll_no	Date	Amt
1	2023-10-20	50.00
2	2023-10-20	25.00
3	2023-10-20	0.00



Roll_no	Date	Amt
1	2023-10-20	50.00