Assignment No.6

Title of Assignment:

Write a PL/SQL block of code using parameterized Cursor, that will merge the data available in the newly created table N_RollCall with the data available in the table O_RollCall. If the data in the first table already exist in the second table then that data should be skipped.

Objective:

To learn how to use **MySQL cursor** in stored procedures to iterate through a result set returned by a SELECT statement.

Relevant Theory

Introduction to MySQL cursor

To handle a result set inside a stored procedure, you use a cursor. A cursor allows you to iterate set of rows returned by a query and process each row accordingly.
MySQL cursor is read only, non-scrollable and asensitive.

- **Read only**: you cannot update data in the underlying table through the cursor.
- **Non-scrollable**: you can only fetch rows in the order determined by the SELECT statement. You cannot fetch rows in the reversed order. In addition, you cannot skip rows or jump to a specific row in the result set.
- **Asensitive**: there are two kinds of cursors: asensitive cursor and insensitive cursor. An asensitive cursor points to the actual data, whereas an insensitive cursor uses a temporary copy of the data. An asensitive cursor performs faster than an insensitive cursor because it does not have to make a temporary copy of data. However, any change that made to the data from other connections will affect the data that is being used by an asensitive cursor, therefore it is safer if you don't update the data that is being used by an asensitive cursor. MySQL cursor is asensitive.

You can use MySQL cursors in stored procedures, stored functions and triggers.

Working with MySQL cursor

First, you have to declare a cursor by using the DECLARE statement:

```
DECLARE cursor_name CURSOR FOR SELECT_statement;
```

The cursor declaration must be after any variable declaration. If you declare a cursor before variables declaration, MySQL will issue an error. A cursor must always be associated with a SELECT statement.
Next, you open the cursor by using the OPEN statement. The OPEN statement initializes the result set for the cursor therefore you must call the OPEN statement before fetching rows from the result set.

```
OPEN cursor_name;
```

Then, you use the FETCH statement to retrieve the next row pointed by the cursor and move the cursor to the next row in the result set.

```
FETCH cursor_name INTO variables list;
```

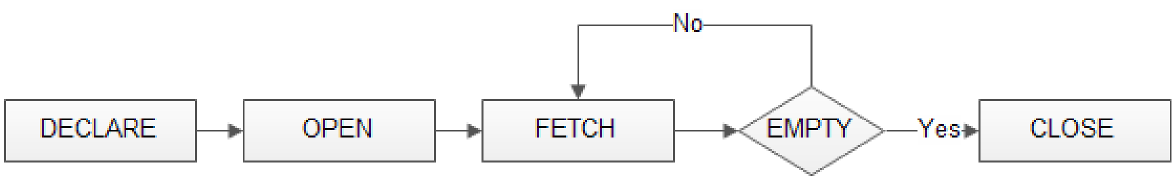
After that, you can check to see if there is any row available before fetching it.
Finally, you call the CLOSE statement to deactivate the cursor and release the memory associated with it as follows:

```
CLOSE cursor_name;
```

When the cursor is no longer used, you should close it.
When working with MySQL cursor, you must also declare a NOT FOUND handler to handle the situation when the cursor could not find any row.
Because each time you call the FETCH statement, the cursor attempts to read the next row in the result set. When the cursor reaches the end of the result set, it will not be able to get the data, and a condition is raised. The handler is used to handle this condition.
To declare a NOT FOUND handler, you use the following syntax:

```
DECLARE CONTINUE HANDLER FOR NOT FOUND SET finished = 1;
```

The following diagram illustrates how MySQL cursor works.



MySQL Cursor Example
We are going to develop a stored procedure that builds an email list of all employees in the employees table in the company sample database provided with this assignment

First, we declare some variables, a cursor for looping over the emails of employees, and a NOT FOUND handler:

```
DECLARE finished INTEGER DEFAULT 0;
DECLARE email varchar(255) DEFAULT "";

-- declare cursor for employee email
DEClARE email_cursor CURSOR FOR
    SELECT email FROM employees;

-- declare NOT FOUND handler
DECLARE CONTINUE HANDLER
FOR NOT FOUND SET finished = 1;
```

Next, we open the email_cursor by using the OPEN statement:
OPEN email_cursor;
Then, we iterate the email list, and concatenate all emails where each email is separated by a semicolon(;):

```
get_email: LOOP
    FETCH email_cursor INTO v_email;
```

```
    IF v_finished = 1 THEN
        LEAVE get_email;
    END IF;
    -- build email list
    SET email_list = CONCAT(v_email,";",email_list);
END LOOP get_email;
```

After that, inside the loop we used the v_finished variable to check if there is any email in the list to terminate the loop.
Finally, we close the cursor using the CLOSE statement:

```
CLOSE email_cursor;
```

The build_email_list stored procedure is as follows:

```
DELIMITER $$

CREATE PROCEDURE build_email_list (INOUT email_list varchar(4000))
BEGIN

    DECLARE v_finished INTEGER DEFAULT 0;
        DECLARE v_email varchar(100) DEFAULT "";

    -- declare cursor for employee email
    DEClARE email_cursor CURSOR FOR
        SELECT email FROM employees;

    -- declare NOT FOUND handler
    DECLARE CONTINUE HANDLER
        FOR NOT FOUND SET v_finished = 1;

    OPEN email_cursor;

    get_email: LOOP

        FETCH email_cursor INTO v_email;

        IF v_finished = 1 THEN
            LEAVE get_email;
        END IF;

        -- build email list
        SET email_list = CONCAT(v_email,";",email_list);

    END LOOP get_email;

    CLOSE email_cursor;

END$$

DELIMITER ;
```

You can test the build_email_list stored procedure using the following script:

```
SET @email_list = "";
CALL build_email_list(@email_list);
SELECT @email_list;
```

Exercises:

1. Write a PL/SQL block to calculate the grade of minimum 10 students using database cursor on student table congaing roll no and marks for three subjects.

Conclusion:

We have shown you how to use MySQL cursor to iterate a result set and process each row accordingly.