

## Phase 5: Apex Programming (Developer)

### 1. Introduction

While admin automation (Flows and Validation Rules) handles simple scenarios, Apex programming allows us to implement custom logic that cannot be achieved through standard Salesforce tools. In this project, Apex is used to prevent overlapping leave requests for the same employee, ensuring data integrity and realistic leave tracking.

### 2. Apex Trigger: Prevent Overlapping Leaves

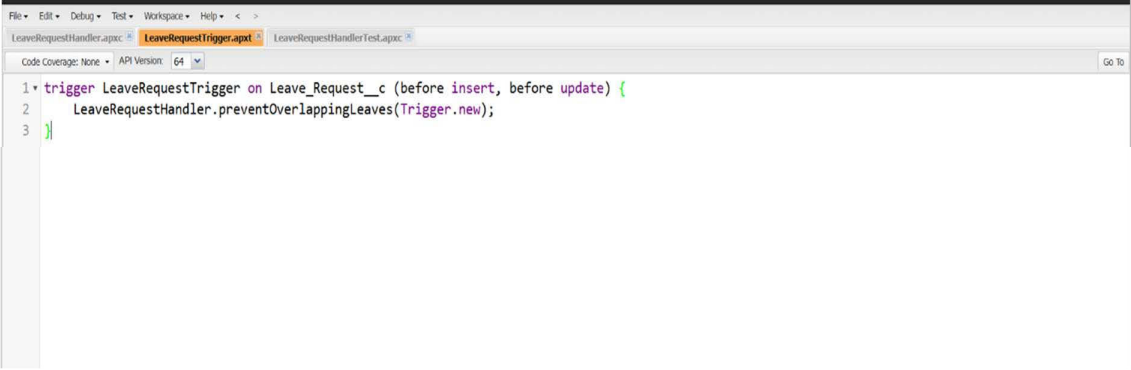
#### Purpose:

- Ensure that an employee cannot submit a new leave request that overlaps with an existing approved or pending leave.
- Prevents conflicts and errors in leave planning.

#### Implementation Steps:

1. Go to Setup → Object Manager → Leave\_Request\_\_c → Triggers → New
2. Write a trigger that calls a Handler Class for logic separation:

#### Trigger (LeaveRequestTrigger.cls)



```
File • Edit • Debug • Test • Workspace • Help • < • >
LeaveRequestHandler.apex • LeaveRequestTrigger.apex • LeaveRequestHandlerTest.apex
Code Coverage: None • API Version: 64 • Go To

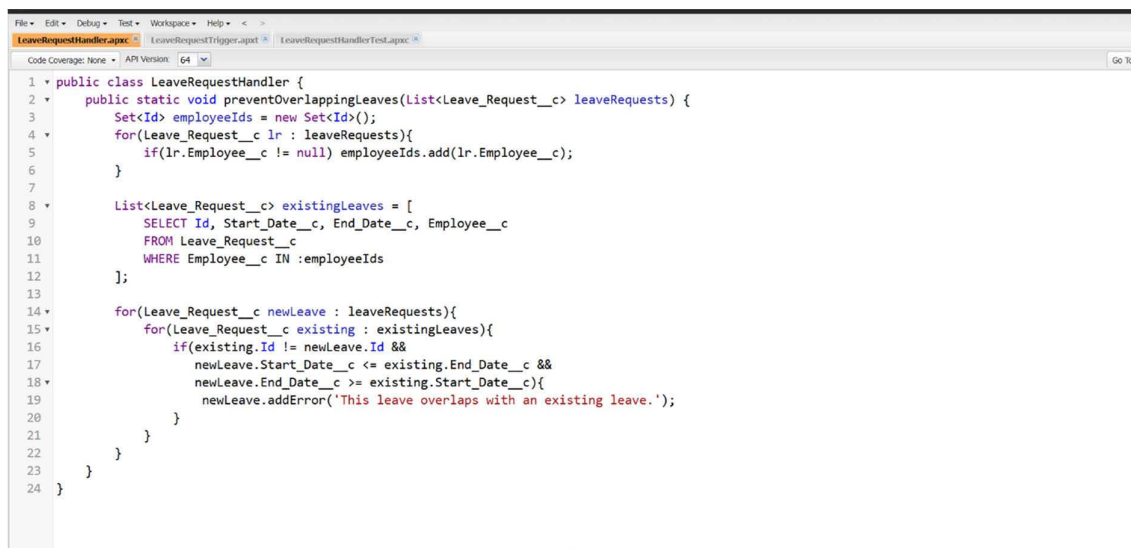
1 • trigger LeaveRequestTrigger on Leave_Request__c (before insert, before update) {
2     LeaveRequestHandler.preventOverlappingLeaves(trigger.new);
3 }
```

### 3. Apex Handler Class

#### Purpose:

- Encapsulates the logic to check for overlapping leave requests.
- Keeps the trigger clean and maintainable.

#### LeaveRequestHandler.cls

The image shows a screenshot of an IDE window displaying the code for the 'LeaveRequestHandler' class. The code is written in Apex and is designed to prevent overlapping leave requests. It includes a static method 'preventOverlappingLeaves' that takes a list of 'Leave\_Request\_\_c' objects. The method first creates a set of employee IDs from the input requests. Then, it queries the database for existing leave requests for those employees. Finally, it iterates through the input requests and checks if any of them overlap with the existing requests based on their start and end dates. If an overlap is found, an error is added to the request.

```
1 public class LeaveRequestHandler {
2     public static void preventOverlappingLeaves(List<Leave_Request__c> leaveRequests) {
3         Set<Id> employeeIds = new Set<Id>();
4         for(Leave_Request__c lr : leaveRequests){
5             if(lr.Employee__c != null) employeeIds.add(lr.Employee__c);
6         }
7
8         List<Leave_Request__c> existingLeaves = [
9             SELECT Id, Start_Date__c, End_Date__c, Employee__c
10            FROM Leave_Request__c
11            WHERE Employee__c IN :employeeIds
12        ];
13
14        for(Leave_Request__c newLeave : leaveRequests){
15            for(Leave_Request__c existing : existingLeaves){
16                if(existing.Id != newLeave.Id &&
17                    newLeave.Start_Date__c <= existing.End_Date__c &&
18                    newLeave.End_Date__c >= existing.Start_Date__c){
19                    newLeave.addError('This leave overlaps with an existing leave.');
```

### 4. Test Class

#### Purpose:

- Ensure the trigger works correctly.
- Covers valid and invalid scenarios to pass deployment requirements.

## Test Class (LeaveRequestHandlerTest.cls):

```
File • Edit • Debug • Test • Workspace • Help • < • >
LeaveRequestHandler.apex • LeaveRequestTrigger.apex • LeaveRequestHandlerTest.apex
Code Coverage: None • API Version: 64 • Go To

1  @isTest
2  public class LeaveRequestHandlerTest {
3      @isTest static void testOverlappingLeave() {
4          Employee__c emp = new Employee__c(Name='Sathvika', E_Mail__c='sathvika@test.com', Department__c='IT');
5          insert emp;
6
7          Leave_Request__c leave1 = new Leave_Request__c(
8              Employee__c = emp.Id,
9              Start_Date__c = Date.today(),
10             End_Date__c = Date.today().addDays(2),
11             Leave_Type__c = 'Casual',
12             Status__c = 'New'
13         );
14         insert leave1;
15
16         Leave_Request__c leave2 = new Leave_Request__c(
17             Employee__c = emp.Id,
18             Start_Date__c = Date.today().addDays(1),
19             End_Date__c = Date.today().addDays(3),
20             Leave_Type__c = 'Casual',
21             Status__c = 'New'
22         );
23
24         Test.startTest();
25         try {
26             insert leave2;
27         } catch (DmlException e) {
28
29         }
30     }
31 }
32
33
Logs, Tests, and Problems
```

```
File • Edit • Debug • Test • Workspace • Help • < • >
LeaveRequestHandler.apex • LeaveRequestTrigger.apex • LeaveRequestHandlerTest.apex
Code Coverage: None • API Version: 64 • Go To

7      Leave_Request__c leave1 = new Leave_Request__c(
8          Employee__c = emp.Id,
9          Start_Date__c = Date.today(),
10         End_Date__c = Date.today().addDays(2),
11         Leave_Type__c = 'Casual',
12         Status__c = 'New'
13     );
14     insert leave1;
15
16     Leave_Request__c leave2 = new Leave_Request__c(
17         Employee__c = emp.Id,
18         Start_Date__c = Date.today().addDays(1),
19         End_Date__c = Date.today().addDays(3),
20         Leave_Type__c = 'Casual',
21         Status__c = 'New'
22     );
23
24     Test.startTest();
25     try {
26         insert leave2;
27     } catch (DmlException e) {
28         System.assert(e.getMessage().contains('overlaps'));
29     }
30     Test.stopTest();
31 }
32
33
Logs, Tests, and Problems
```

## 5. Outcome of Phase 5

- The Apex trigger and handler prevent overlapping leaves.
- Test class ensures deployment readiness by covering positive and negative cases.
- Employees cannot create conflicting leave requests, maintaining data integrity.

## 6. Conclusion

- Apex programming in Phase 5 adds **developer-level functionality** that cannot be achieved through standard Salesforce tools. This ensures the Leave Management System is robust, accurate, and aligned with real-world business requirements.