Phase 5
Apex Programming (Developer)

1. Objective

Implement Apex logic to:

- Prevent deletion of paid fee payments.

- Calculate outstanding fees for a student.

- Ensure code is tested and achieves ≥75% coverage.

Deliverables:

- Apex Trigger

- Apex Classes

- Apex Test Class

- Screenshot of test results and coverage

2. Apex Trigger

Step1: Open Developer Console in a new tab

Step2:Click on new→Apex Trigger

Name: FeePaymentTrigger
Object: Fee_Payment__c

Step3:Click on “OK”
Purpose: Prevent deletion of Fee Payment if status = 'Paid'.

Trigger Code

```
trigger FeePaymentPreventDelete on Fee_Payment__c (before delete) {
    for (Fee_Payment__c fp : Trigger.old) {
        // Check if Status = 'Paid'
        if (fp.Status__c != null &&
String.valueOf(fp.Status__c).trim().equalsIgnoreCase('Paid')) {
            fp.addError('Paid Fee Payments cannot be deleted.');
        }
    }
}
```

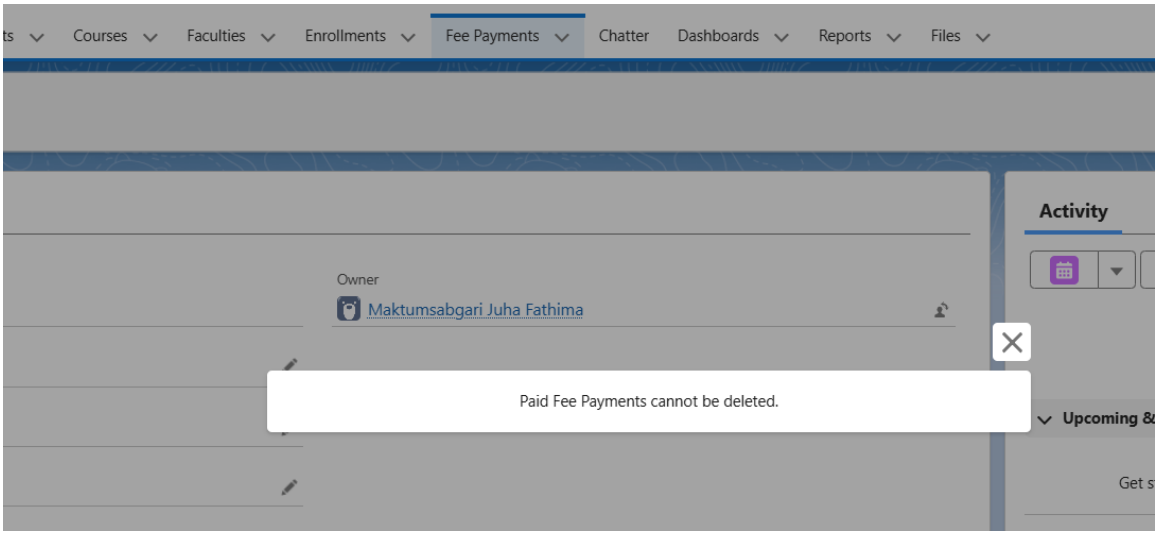

Fig1:Fee payment record detection

FeeCalculator

Step1: Open Developer Console in a new tab

Step2:Click on new→Apex Class

Name: FeeCalculator

Step3:Click on “OK”

Purpose: Calculate outstanding fees for a student.

FeeCalculator Code of Apex class:

```
public class FeeCalculator {

    // Update balance and status for a Fee Payment
    public static void updateFeePayment(Fee_Payment__c fp) {
        if(fp.Course__c == null) {
            throw new AuraHandledException('Course not linked for this Fee Payment.');
        }

        // Fetch course fee
        Course__c course = [SELECT Course_Fee__c FROM Course__c WHERE Id =
:fp.Course__c LIMIT 1];

        // Calculate balance
        Decimal balance = course.Course_Fee__c - fp.Paid_Amount__c;

        // Update Amount field (balance)
        fp.Amount__c = balance;

        // Update Status
        if(balance == 0) {
            fp.Status__c = 'Paid';
        } else {
            fp.Status__c = 'Pending)';
        }

        update fp; // Save changes
    }

    // Method to get outstanding fees for a student
    public static Decimal getOutstandingFees(Id studentId) {
        Decimal totalOutstanding = 0;

        // Fetch all fee payments of the student
        List<Fee_Payment__c> payments = [
            SELECT Amount__c
            FROM Fee_Payment__c
            WHERE Student__c = :studentId
        ];

        for(Fee_Payment__c fp : payments){
```

```
            totalOutstanding += fp.Amount__c; // Amount__c is balance
        }

        return totalOutstanding;
    }
}
```

FeeCalculator Code of Apex Trigger:

Step1: Open Developer Console in a new tab

Step2:Click on new→Apex Trigger

Name: FeePaymentTrigger
Object: Fee_Payment__c

Step3:Click on “OK”

```
trigger FeePaymentTrigger on Fee_Payment__c (before insert, before update) {

    for (Fee_Payment__c fp : Trigger.new) {
        if (fp.Course__c != null) {

            // Fetch course fee
            Course__c course = [
                SELECT Course_Fee__c
                FROM Course__c
                WHERE Id = :fp.Course__c
```

```
            LIMIT 1
        ];

        // If Paid_Amount__c is provided
        if (fp.Paid_Amount__c != null) {
            // Calculate remaining balance
            Decimal balance = course.Course_Fee__c - fp.Paid_Amount__c;
            fp.Amount__c = balance;

            // Update status
            if (balance == 0) {
                fp.Status__c = 'Paid';
            } else {
                fp.Status__c = 'Pending)';
            }

            // Auto-fill today's date
            fp.Payment_Date__c = Date.today();

        } else {
            // If no Paid Amount entered
            fp.Amount__c = course.Course_Fee__c;
            fp.Status__c = 'None';
            fp.Payment_Date__c = null;
        }
    }
  }
}
```

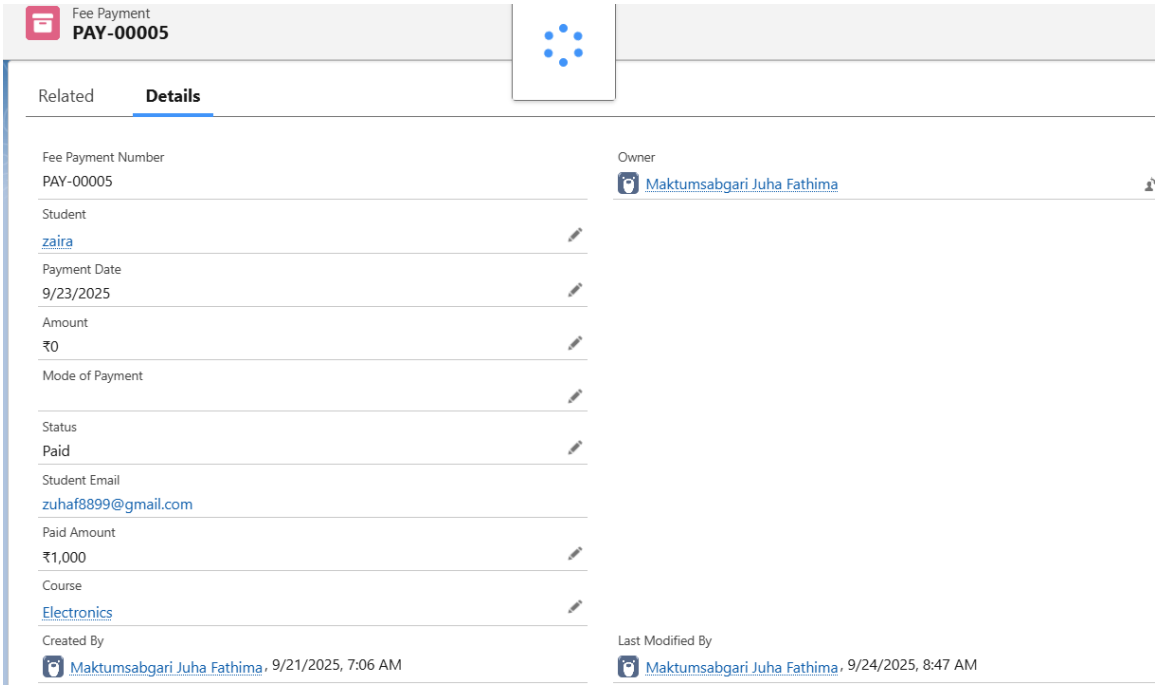

Fig2:Calcuate amount by code

4. Test Class

Step1: Open Developer Console in a new tab

Step2:Click on new→Apex Class

Name: FeeCalculatorTest

Step3:Click on "OK"

Purpose:Validate trigger and class logic; achieve ≥75% code coverage.

Test Class Code

```
@isTest
public class FeeModuleTest {

  @testSetup
  static void createData() {

    // Create Course
    Course__c course = new Course__c(
      Name = 'Test Course',
```

```
        Course_Fee__c = 1000
    );
    insert course;

    // Create Student
    Student__c stu = new Student__c(
        Name = 'Test Student'
    );
    insert stu;

    // Create Paid Fee Payment
    Fee_Payment__c fee1 = new Fee_Payment__c(
        Student__c = stu.Id,
        Course__c = course.Id,
        Paid_Amount__c = 400,
        Status__c = 'Paid'
    );
    insert fee1;

    // Create Pending Fee Payment
    Fee_Payment__c fee2 = new Fee_Payment__c(
        Student__c = stu.Id,
        Course__c = course.Id,
        Paid_Amount__c = 200,
        Status__c = 'Pending)'
    );
    insert fee2;

    // Update Amount__c using FeeCalculator
    FeeCalculator.updateFeePayment(fee1);
    FeeCalculator.updateFeePayment(fee2);
  }

  // Test 1: Outstanding Fee Calculation
  @isTest
  static void testOutstandingFees() {
    Student__c stu = [SELECT Id FROM Student__c LIMIT 1];
    Decimal outstanding = FeeCalculator.getOutstandingFees(stu.Id);

    // Assuming each Fee Payment has its own balance stored in Amount__c
    // Fee1: 1000 - 400 = 600, Fee2: 1000 - 200 = 800 → total = 1400
    System.assertEquals(1400, outstanding, 'Outstanding Fee should be 1400');
  }

  // Test 2: Prevent Deletion of Paid Fee (Trigger)
  @isTest
  static void testTriggerPreventDeletePaid() {
    Fee_Payment__c paidFee = [SELECT Id FROM Fee_Payment__c WHERE Status__c = 'Paid'
LIMIT 1];
```

```
        try {
            delete paidFee;
            System.assert(false, 'Deletion should not be allowed for Paid fee');
        } catch (DmlException e) {
            System.assert(e.getMessage().contains('cannot be deleted'),
                'Expected error message not found');
        }
    }

    // Test 3: UpdateFeePayment Method
    @isTest
    static void testUpdateFeePayment() {
        Fee_Payment__c pendingFee = [SELECT Id, Amount__c FROM Fee_Payment__c WHERE
Status__c = 'Pending)' LIMIT 1];

        // Call update method again to ensure it updates Amount__c correctly
        FeeCalculator.updateFeePayment(pendingFee);

        Fee_Payment__c updatedFee = [SELECT Amount__c FROM Fee_Payment__c WHERE Id =
:pendingFee.Id];
        System.assertEquals(800, updatedFee.Amount__c, 'Pending Fee Amount__c should be
updated correctly');
    }
}
```

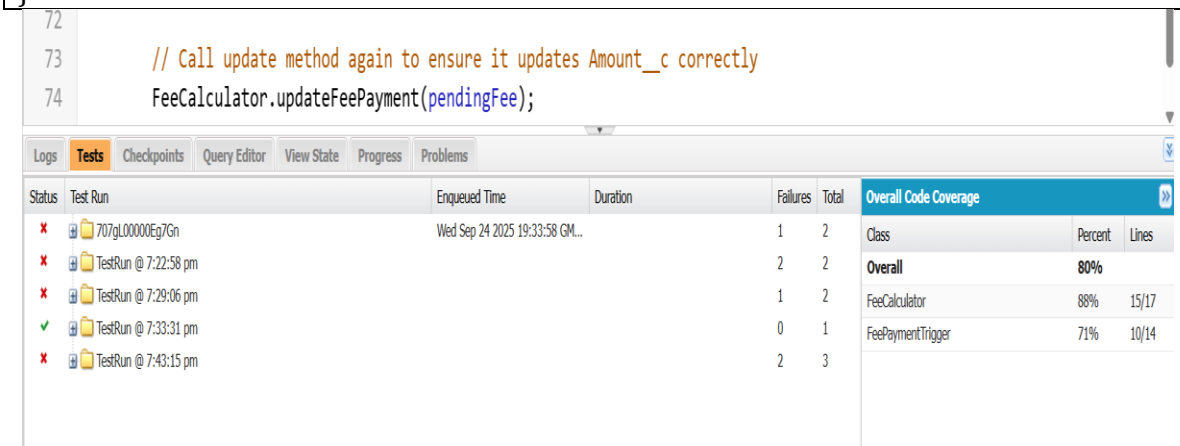

Fig 3:Test class

5. Summary

Component	Purpose
Apex Trigger	Prevent deletion of paid fee payments
FeeHandler	Centralized logic for trigger
FeeCalculator	Calculate outstanding fees for a student
Test Class	Validate logic, test triggers, ensure ≥75% coverage

Deliverables:
- Apex Trigger code (FeePaymentTrigger)
- Apex Classes (FeeHandler, FeeCalculator)
- Apex Test Class (FeeCalculatorTest)
- Screenshot of test results showing code coverage