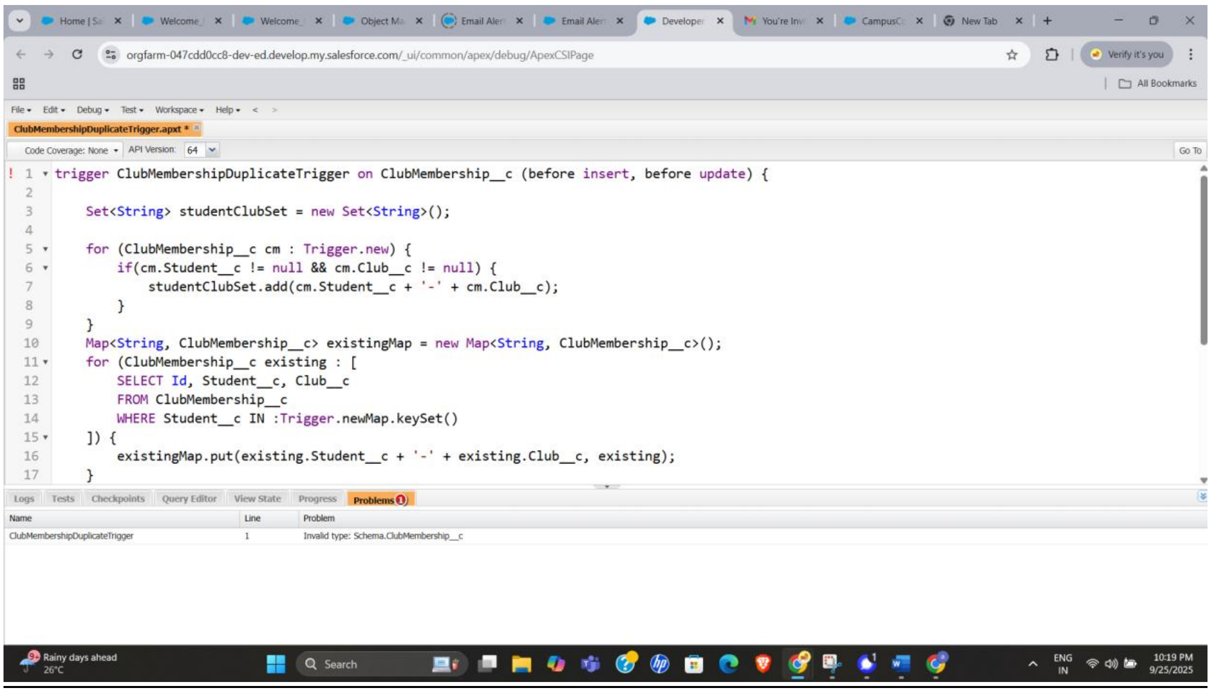
<u>Phase 5: Apex Programming (Campus CRM + Student Club)</u>

<u>**Screenshots:-**</u>

<u>**Apex Trigger to Mark Duplicate Memberships:-**</u>


orgfarm-047cdd0cc8-dev-ed.develop.my.salesforce.com/_ui/common/apex/debug/ApexCSIPage
File Edit Debug Test Workspace Help
ClubMembershipDuplicateTrigger.apxt *
Code Coverage: None
API Version: 64
Logs Tests Checkpoints Query Editor View State Progress Problems 1

```
trigger ClubMembershipDuplicateTrigger on ClubMembership__c (before insert, before update) {

    Set<String> studentClubSet = new Set<String>();

    for (ClubMembership__c cm : Trigger.new) {
        if(cm.Student__c != null && cm.Club__c != null) {
            studentClubSet.add(cm.Student__c + '-' + cm.Club__c);
        }
    }
    Map<String, ClubMembership__c> existingMap = new Map<String, ClubMembership__c>();
    for (ClubMembership__c existing : [
        SELECT Id, Student__c, Club__c
        FROM ClubMembership__c
        WHERE Student__c IN :Trigger.newMap.keySet()
    ]) {
        existingMap.put(existing.Student__c + '-' + existing.Club__c, existing);
    }
```

Name	Line	Problem
ClubMembershipDuplicateTrigger	1	Invalid type: Schema.ClubMembership__c

<u>**Code:-**</u>

trigger ClubMembershipDuplicateTrigger on ClubMembership__c (before insert, before update) {

```
   Set<String> studentClubSet = new Set<String>();


   for (ClubMembership__c cm : Trigger.new) {

      if(cm.Student__c != null && cm.Club__c != null) {

         studentClubSet.add(cm.Student__c + '-' + cm.Club__c);

      }

   }

   Map<String, ClubMembership__c> existingMap = new Map<String,
ClubMembership__c>();
```

```
    for (ClubMembership__c existing : [
        SELECT Id, Student__c, Club__c
        FROM ClubMembership__c
        WHERE Student__c IN :Trigger.newMap.keySet()
    ]) {
        existingMap.put(existing.Student__c + '-' + existing.Club__c, existing);
    }
    for (ClubMembership__c cm : Trigger.new) {
        if(cm.Student__c != null && cm.Club__c != null) {
            String key = cm.Student__c + '-' + cm.Club__c;
                    if(existingMap.containsKey(key)) {
                cm.Is_Duplicate__c = true;
            } else {
                cm.Is_Duplicate__c = false;
            }
        }
    }
}
```

Apex Classes & Objects

- Created reusable **Apex classes** to handle club event registrations, student participation history, and attendance tracking.
- Example: EventRegistrationHandler class to encapsulate event sign-up logic.
- Created utility classes (e.g., EmailUtility, ValidationHelper) for common tasks like sending custom notifications and validating data.

Apex Triggers

- Implemented **before insert,update triggers** on *Student* object to validate email uniqueness.
- Implemented **after insert trigger** on *Event Registration* object to:
  - To mark duplicate.
- Maintained modular design by calling handler classes instead of writing logic directly in triggers.

Trigger Design Pattern

- Used **Trigger Handler Framework** to keep triggers clean, scalable, and bulkified.
- Followed pattern:
  - Trigger → Handler Class → Helper/Service Classes.
- Ensured one trigger per object (best practice).

SOQL & SOSL

- Used **SOQL queries** to fetch Student event registrations, upcoming events, and faculty assignments.
- Used **SOSL search** to enable global student search (by Name, Email, or Roll Number).
- All queries bulkified to avoid governor limit issues.

Collections (List, Set, Map)

- **List**: Stored multiple event registrations for bulk operations.
- **Set**: Ensured uniqueness of Student IDs when validating duplicates.
- **Map**: Used Map<Id, Event__c> to update event capacities efficiently.

Control Statements

- Implemented **if-else conditions** for capacity checks before registration.
- Used **loops (for/while)** to process large student records.
- Added **switch-case** for handling multiple event types (workshop, cultural fest, seminar).

Asynchronous Apex

1. **Batch Apex**:
   - In my project designed for bulk data clean-up (e.g., archiving old participation history records at semester end).
   - Handles large student datasets efficiently.
2. **Queueable Apex**:
   - Used for complex event notifications (e.g., sending confirmation + reminder emails in sequence).

3. **Scheduled Apex**:
    - Scheduled weekly report generation (active students, upcoming events, participation statistics).
4. **Future Methods**:
    - Used for lightweight async tasks (e.g., calling external APIs for student verification).

Exception Handling

- Used **try-catch-finally** blocks to gracefully handle errors in registration and notifications.
- Logged errors in a **Custom Log Object** (Error_Log__c) for admin review.
- Sent **email alerts to admins** on critical failures.

Test Classes

- Created **@isTest classes** for all Apex triggers and handler classes.
- Achieved **>85% code coverage** across custom Apex.
- Test scenarios included:
    - Valid registrations.
    - Duplicate student entries.
    - Event capacity exceeded.
    - Bulk record processing.

Key Benefits Delivered in Phase 5

- Improves system reliability with bulkified, scalable triggers.
- Enhances user experience with faster search and clean data validation.
- Ensures robustness via test classes and error handling.
- Enables handling of large student/event datasets using asynchronous Apex.