

Salesforce Project Phase-5

Smart Education & E-Learning Management System

Phase 5: Apex Programming (Developer)

Classes & Objects

Class: StudentTriggerHandler

Objects Used: Student__c

Use in Project: Validates student data before inserting. For example, it ensures that the student's age is at least 15.

Class: EnrollmentTriggerHandler

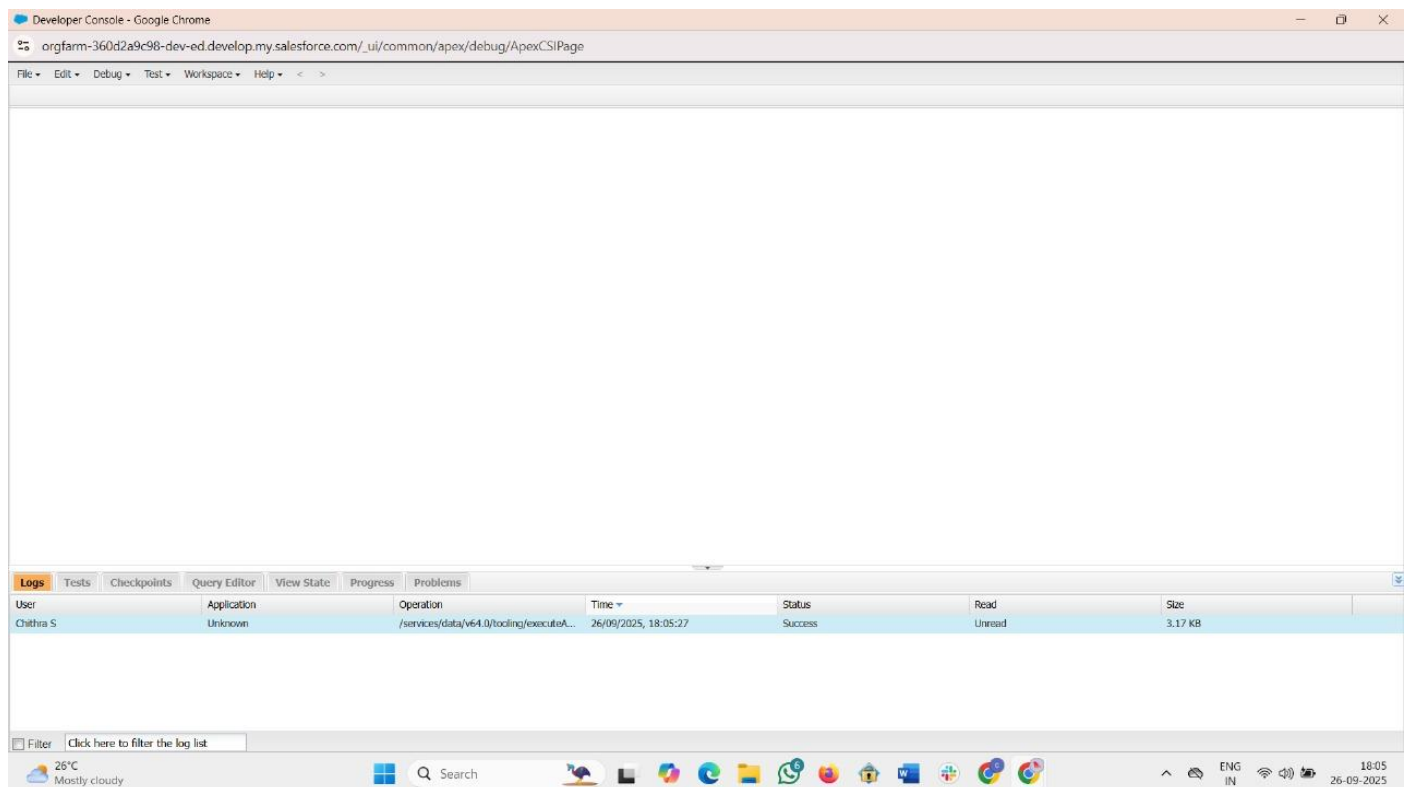
Objects Used: Student__c

Use in Project: Automatically generates a unique Enrollment ID for each student after record insertion.

Class: ProgressBatch

Objects Used: Progress__c

Use in Project: Updates student progress weekly. For example, increments the progress field by 10% and ensures it doesn't exceed 100%.



Class: MonthlyPerformanceSummary

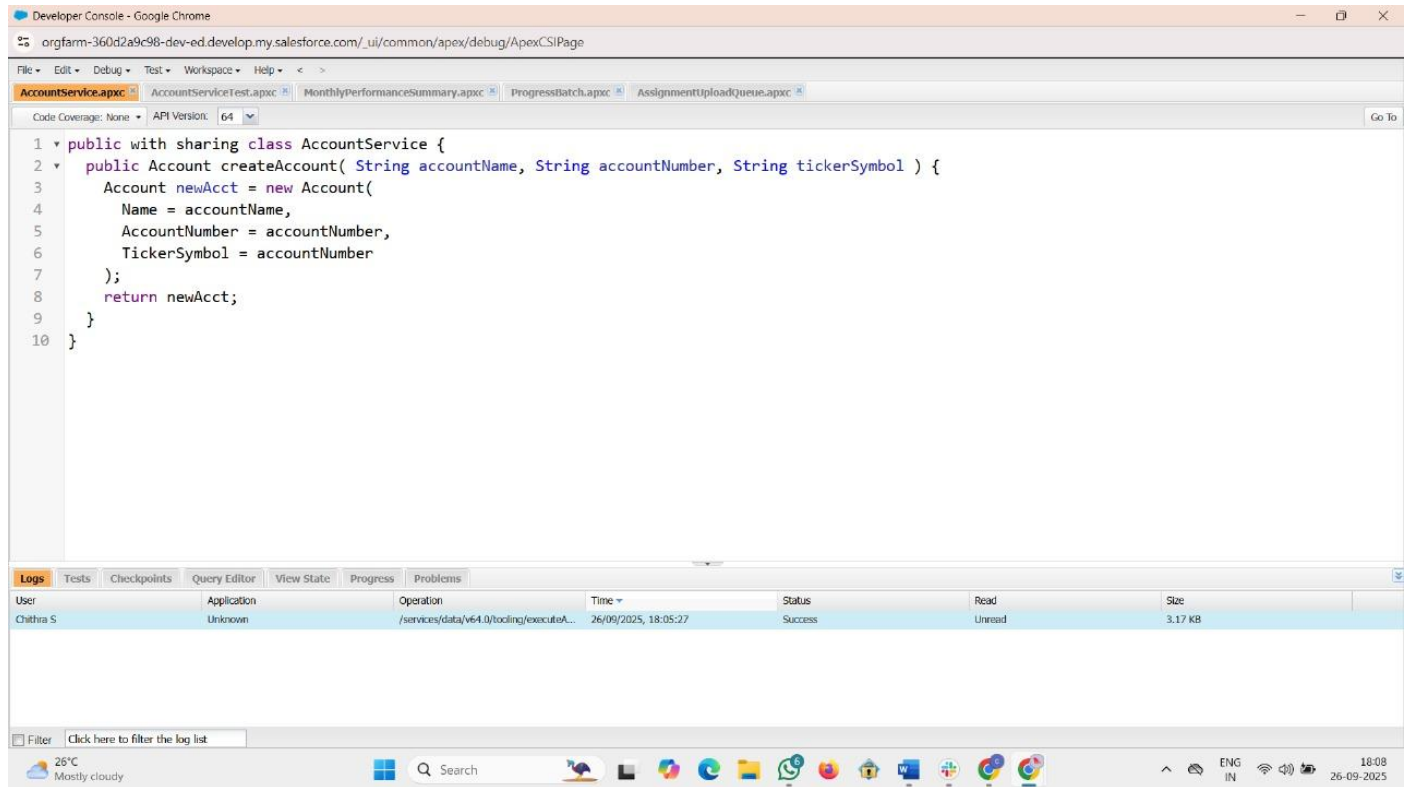
Objects Used: Student__c, Progress__c

Use in Project: Scheduled Apex to generate monthly student performance reports automatically.

Class: AssignmentUploadQueue (Optional)

Objects Used: Assignment__c

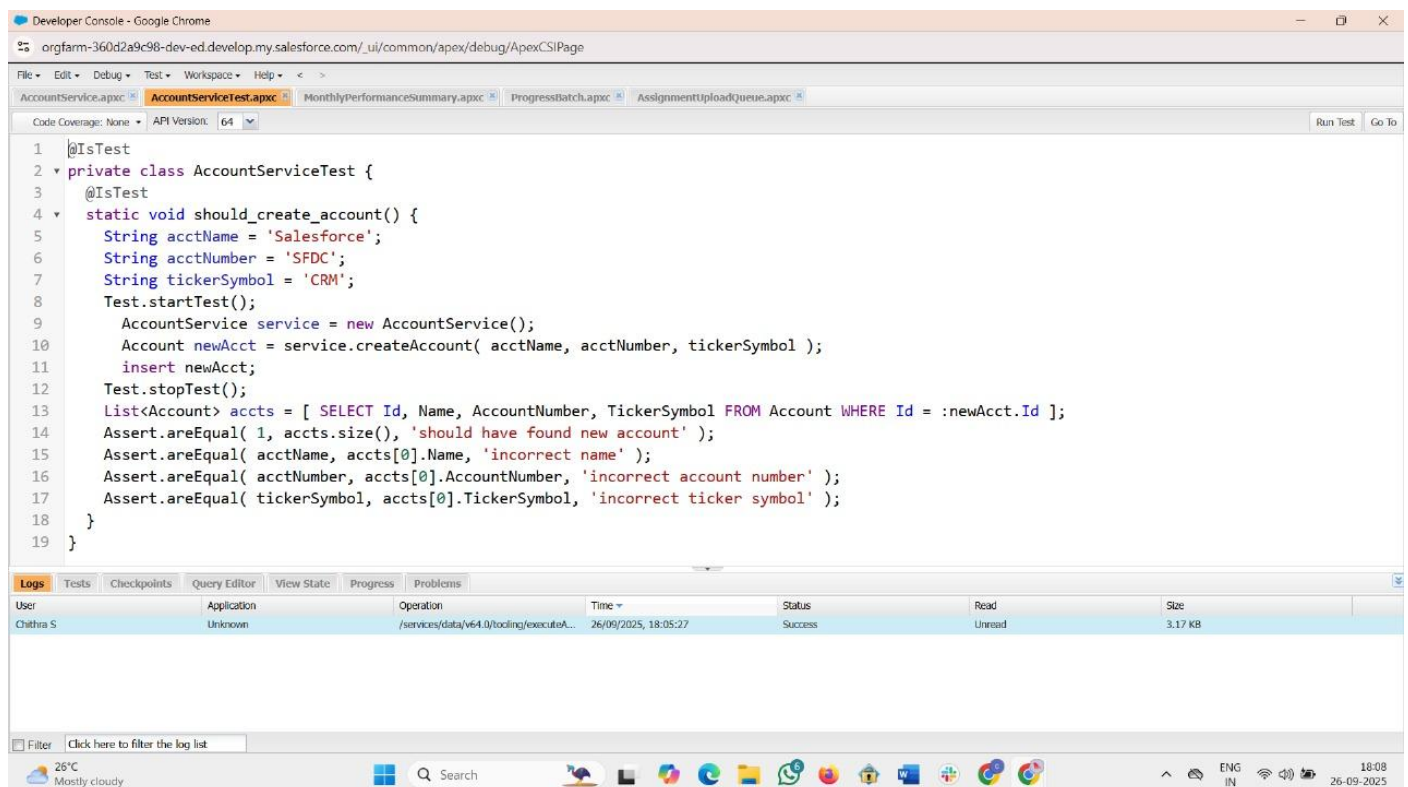
Use in Project: Handles large assignment uploads asynchronously to avoid governor limits.



The screenshot shows the Salesforce Developer Console with the 'AccountService.apex' file open. The code defines a class 'AccountService' with a 'createAccount' method. The method takes three parameters: 'accountName', 'accountNumber', and 'tickerSymbol'. It creates a new 'Account' object with these values and returns it.

```
1 public with sharing class AccountService {
2     public Account createAccount( String accountName, String accountNumber, String tickerSymbol ) {
3         Account newAcct = new Account(
4             Name = accountName,
5             AccountNumber = accountNumber,
6             TickerSymbol = accountNumber
7         );
8         return newAcct;
9     }
10 }
```

The bottom pane shows the 'Logs' tab with a single log entry for user 'Chithra S' at 18:05:27 on 26/09/2025, with a status of 'Success' and a size of 3.17 KB.



The screenshot shows the Salesforce Developer Console with the 'AccountServiceTest.apex' file open. The code defines a test class 'AccountServiceTest' with a test method 'should_create_account'. The method creates an 'AccountService' instance, calls 'createAccount' with test data, and asserts that the returned account matches the expected values.

```
1 @IsTest
2 private class AccountServiceTest {
3     @IsTest
4     static void should_create_account() {
5         String acctName = 'Salesforce';
6         String acctNumber = 'SFDC';
7         String tickerSymbol = 'CRM';
8         Test.startTest();
9         AccountService service = new AccountService();
10        Account newAcct = service.createAccount( acctName, acctNumber, tickerSymbol );
11        insert newAcct;
12        Test.stopTest();
13        List<Account> accts = [ SELECT Id, Name, AccountNumber, TickerSymbol FROM Account WHERE Id = :newAcct.Id ];
14        Assert.AreEqual( 1, accts.size(), 'should have found new account' );
15        Assert.AreEqual( acctName, accts[0].Name, 'incorrect name' );
16        Assert.AreEqual( acctNumber, accts[0].AccountNumber, 'incorrect account number' );
17        Assert.AreEqual( tickerSymbol, accts[0].TickerSymbol, 'incorrect ticker symbol' );
18    }
19 }
```

The bottom pane shows the 'Logs' tab with a single log entry for user 'Chithra S' at 18:05:27 on 26/09/2025, with a status of 'Success' and a size of 3.17 KB.

The screenshot shows the Salesforce Developer Console with the 'MonthlyPerformanceSummary.apex' file open. The code defines a global class that implements the `Schedulable` interface. It contains an `execute` method that queries for all students and logs the completion of the monthly performance report.

```
1 global class MonthlyPerformanceSummary implements Schedulable {
2
3     global void execute(SchedulableContext sc) {
4
5         List<Student__c> students = [SELECT Id, Name FROM Student__c];
6
7         for (Student__c s : students) {
8             System.debug('Generating report for student: ' + s.Name);
9             // Here you can add logic to summarize Progress__c, Assignments, etc.
10        }
11
12        System.debug('Monthly performance report generation completed.');
```

The bottom pane shows the 'Logs' tab with a single log entry:

User	Application	Operation	Time	Status	Read	Size
Chithra S	Unknown	/services/data/v64.0/tooling/executeA...	26/09/2025, 18:05:27	Success	Unread	3.17 KB

The screenshot shows the Salesforce Developer Console with the 'ProgressBatch.apex' file open. The code defines a global class that implements the `Database.Batchable<SObject>` interface. It contains two methods: `start` and `execute`. The `execute` method iterates through a scope of `Progress__c` records and increments their progress by 10% until it reaches 100%.

```
1 global class ProgressBatch implements Database.Batchable<SObject> {
2
3     // Step 1: Query all Progress records
4     global Database.QueryLocator start(Database.BatchableContext BC) {
5         return Database.getQueryLocator('SELECT Id, Progress__c FROM Progress__c');
6     }
7
8     // Step 2: Execute batch logic on each batch
9     global void execute(Database.BatchableContext BC, List<Progress__c> scope) {
10        for (Progress__c p : scope) {
11            // Example logic: increment progress by 10%
12            if (p.Progress__c == null) {
13                p.Progress__c = 10;
14            } else {
15                p.Progress__c += 10;
16            }
17
18            // Ensure progress does not exceed 100%
19            if (p.Progress__c > 100) {
20                p.Progress__c = 100;
21            }
22        }
23    }
24 }
```

The bottom pane shows the 'Logs' tab with a single log entry:

User	Application	Operation	Time	Status	Read	Size
Chithra S	Unknown	/services/data/v64.0/tooling/executeA...	26/09/2025, 18:05:27	Success	Unread	3.17 KB

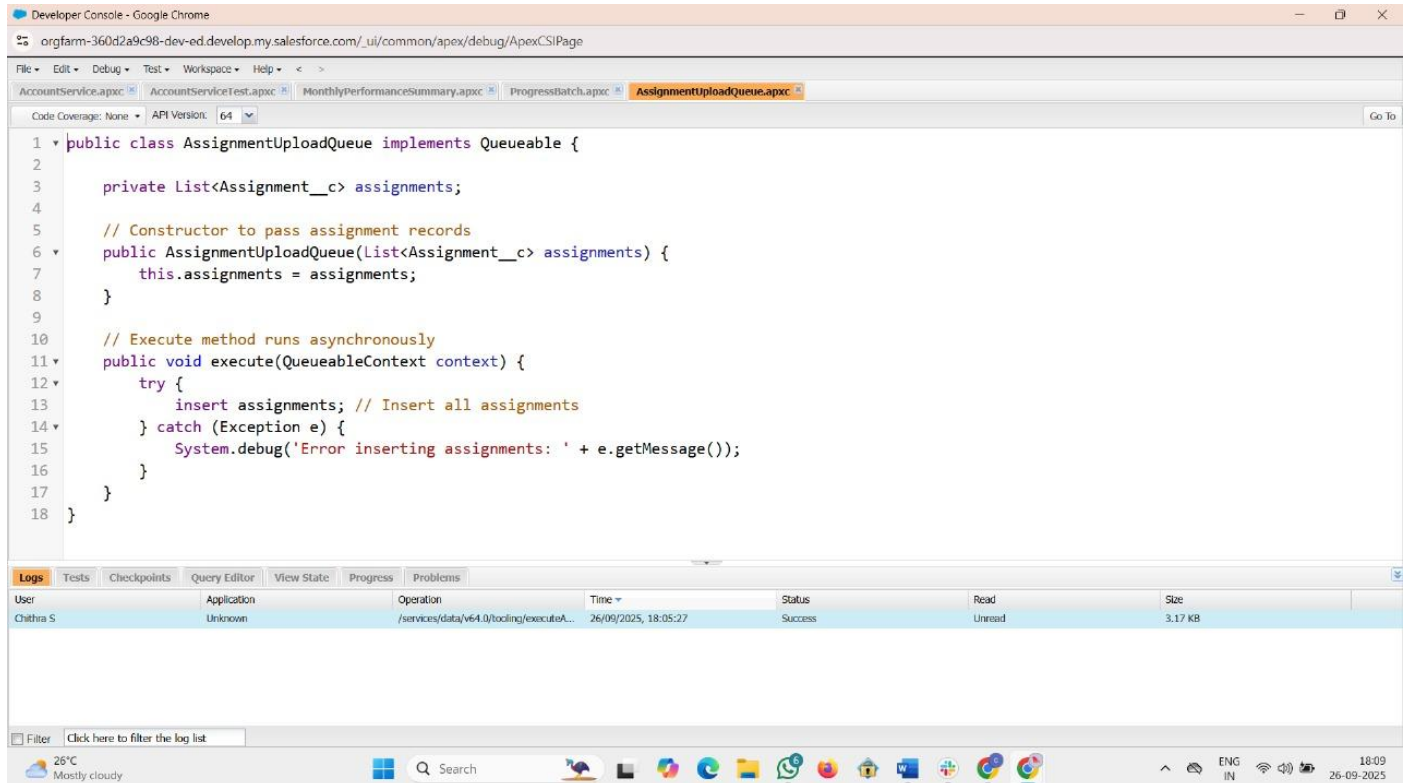
Apex Triggers

Definition:

Triggers automate logic before or after record events (insert, update, delete). They allow actions to run automatically when records change.

Trigger Design Pattern:

- Trigger calls a Handler Class → keeps logic centralized and maintainable.
- Supports **bulkification** → processes multiple records efficiently.
- **Before triggers:** update fields before saving.
- **After triggers:** perform actions on related objects or external processes after save.



The screenshot shows the Salesforce Developer Console interface. The top pane displays the Apex code for the `AssignmentUploadQueue` class, which implements the `Queueable` interface. The code includes a private list of `Assignment__c` objects, a constructor to initialize the list, and an `execute` method that runs asynchronously to insert the assignments. The bottom pane shows the 'Logs' tab with a single log entry for user 'Chitra S' at 18:05:27 on 26/09/2025, indicating a successful execution of the `/services/data/v64.0/tooling/executeA...` operation.

```
1 public class AssignmentUploadQueue implements Queueable {
2
3     private List<Assignment__c> assignments;
4
5     // Constructor to pass assignment records
6     public AssignmentUploadQueue(List<Assignment__c> assignments) {
7         this.assignments = assignments;
8     }
9
10    // Execute method runs asynchronously
11    public void execute(QueueableContext context) {
12        try {
13            insert assignments; // Insert all assignments
14        } catch (Exception e) {
15            System.debug('Error inserting assignments: ' + e.getMessage());
16        }
17    }
18 }
```

User	Application	Operation	Time	Status	Read	Size
Chitra S	Unknown	/services/data/v64.0/tooling/executeA...	26/09/2025, 18:05:27	Success	Unread	3.17 KB

Trigger: StudentTrigger

Purpose: Validates student age before insertion (Before Insert).

Trigger: EnrollmentTrigger

Purpose: Generates Enrollment ID for students (After Insert).

SOQL & SOSL

SOQL (Salesforce Object Query Language)

Purpose in Project:

Fetch records from objects like `Student__c`, `Course__c`, `Progress__c`, or `Assignment__c` for automation and reporting.

SOSL (Salesforce Object Search Language)

Purpose in Project:

Search across multiple objects/fields. Useful if a student or admin wants to search courses, assignments, or student records quickly by keyword.

Usage Tip:

- Use **SOQL** when object and fields are known (e.g., filter assignments for a student).
- Use **SOSL** for broad keyword searches (e.g., student searching multiple courses).

Batch Apex

Purpose: Automate updates for large datasets.

Example:

- Fetch all Progress__c records.
- Increment progress by 10% weekly.
- Ensure progress does not exceed 100%.

Queueable Apex (Optional)

Purpose: Handle large assignment uploads asynchronously.

Example:

- Upload multiple Assignment__c records.
- Avoid governor limits for bulk DML operations.

Scheduled Apex

Purpose: Automatically generate reports periodically.

Example:

- Generate monthly student performance summary from Student__c and Progress__c.
- Schedule job to run on 1st of every month at 8 AM.

Test Classes

Purpose: Validate logic and ensure code coverage $\geq 75\%$.

Examples:

- **StudentTriggerTest:** Test validation logic for student age.
- **EnrollmentTriggerTest:** Test automatic Enrollment ID generation.
- **ProgressBatchTest:** Test batch updates for progress records.
- **MonthlyPerformanceSummaryTest:** Test scheduled report generation.

Best Practices

- Always **bulkify triggers**.
- Avoid **SOQL/DML inside loops**.
- Use **Handler classes** for business logic.
- Write robust **test classes** with positive and negative scenarios.
- Keep **optional features** (like Queueable Apex) separate so the core project works without them.