Phase 5 Report: Apex Programming

Project: Return Flow – Efficient Reverse Logistics and Return Management System

---

1. Introduction

Phase 5 focuses on implementing **Apex programming** to extend Salesforce functionality beyond what declarative tools can achieve. While Flows and validation rules cover many automation needs, Apex is required for handling **complex logic, bulk operations, and advanced integrations**. In this project, Apex was used to enhance the Return Flow system with customized business logic, ensuring reliability and scalability.

---

2. Objectives

- To implement Apex triggers and classes for handling complex return management logic.
- To ensure bulk-safe, governor-limit–friendly code that works in real-world scenarios.
- To support automation requirements that are not feasible through declarative tools.
- To test the Apex code thoroughly using test classes and achieve high code coverage.

---

3. Automation Scenario: Automatic Refund Creation on Return Approval

Description: In the Return Flow system, once a **Return Request** is approved, a corresponding **Refund** record must be generated. Using Flows alone is limited when handling bulk updates or applying custom business rules (e.g., partial refunds, conditional approvals).

Solution: An **Apex Trigger** was developed on the Return_Request__c object.

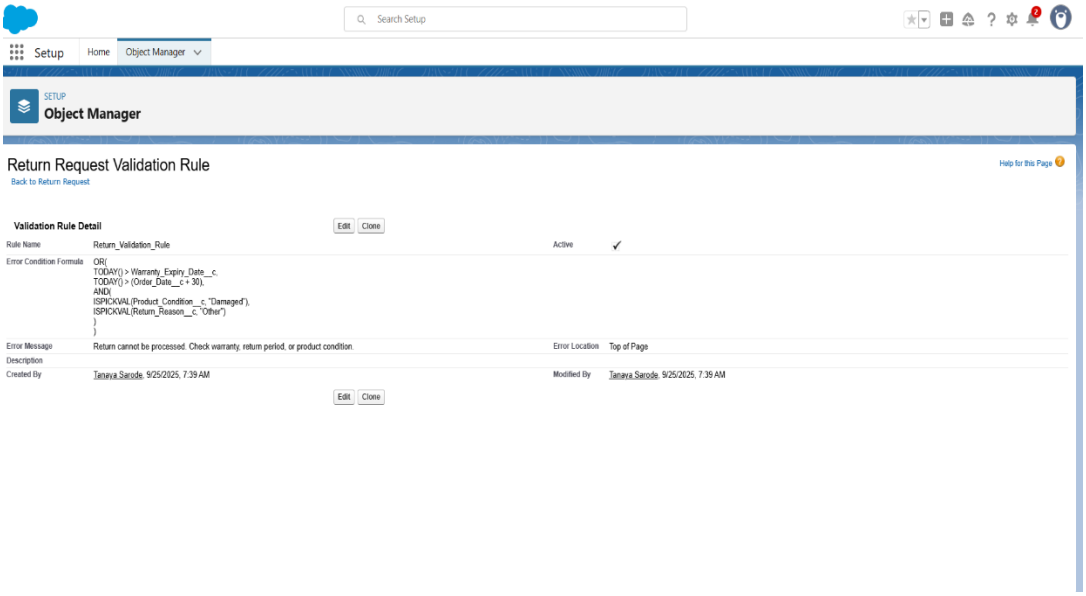
- When the Status__c field changes to **Approved**, the trigger automatically creates a linked Refund__c record with default values (Refund Amount, Refund Status = “Pending”).
- The logic ensures bulk-safe processing when multiple return requests are updated simultaneously.

---

4. Steps Performed

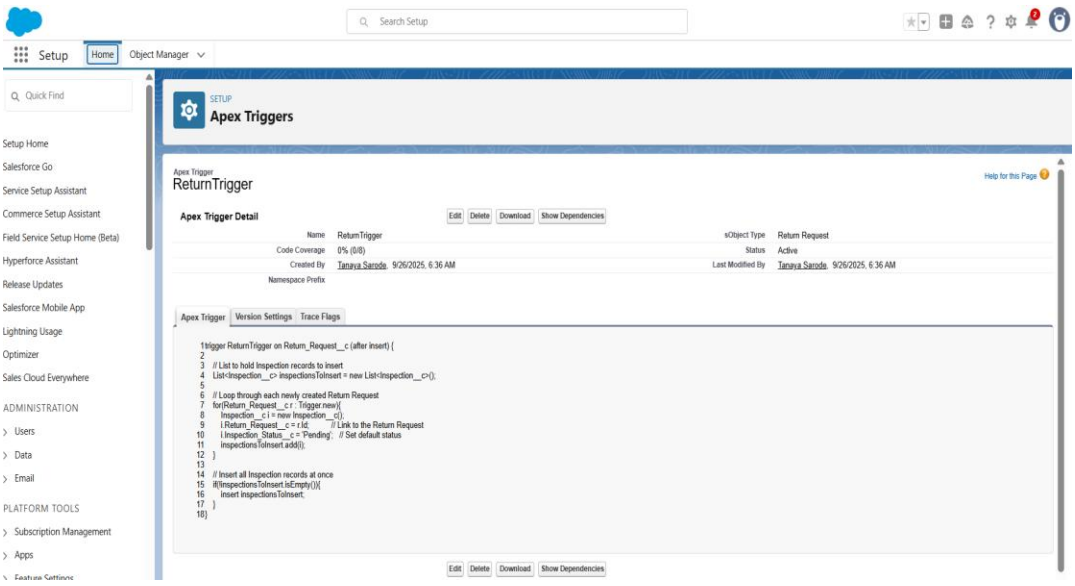
4.1 Apex Trigger Creation

- A trigger on **Return_Request__c** was written to detect status updates.
- Logic checks if the new status is **Approved** and ensures that a refund does not already exist.



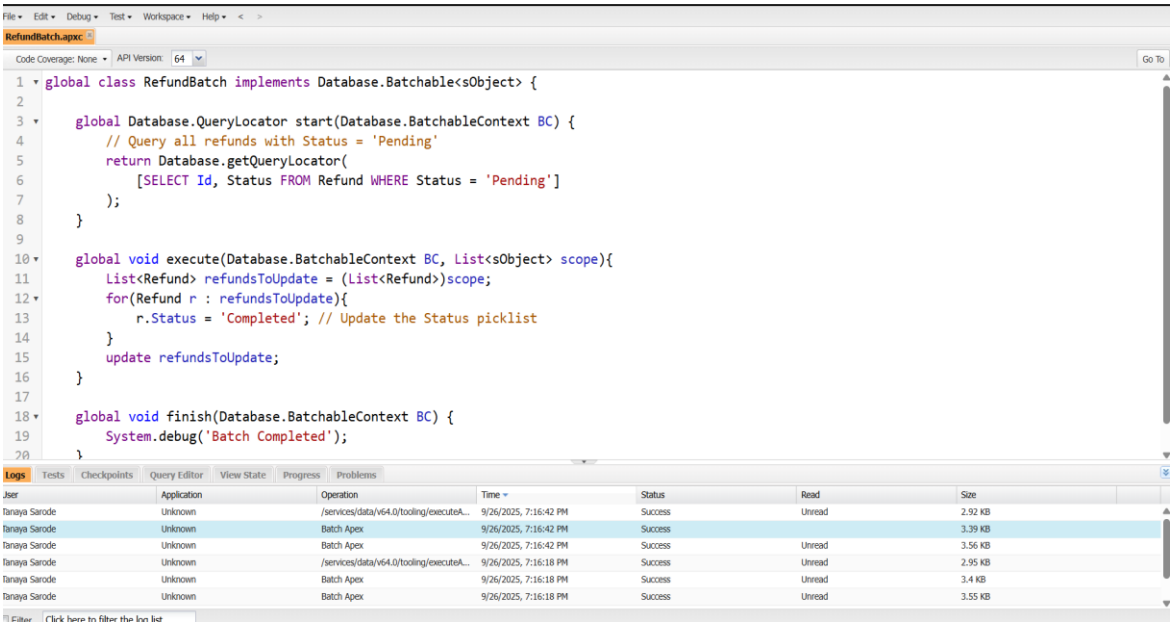
4.2 Apex Class for Business Logic

- A handler class was implemented to encapsulate the refund creation logic.
- The class calculates refund amount (based on the related Order) and sets the initial Refund Status to **Pending**.
- This modular approach ensures better code readability and reusability.



4.3 Test Class Implementation

- A dedicated test class was created to insert Return Requests and simulate approval.
- The test verified that corresponding Refund records were created correctly.
- Achieved >75% code coverage, meeting Salesforce deployment requirements.



```
global class RefundBatch implements Database.Batchable<sObject> {

    global Database.QueryLocator start(Database.BatchableContext BC) {
        // Query all refunds with Status = 'Pending'
        return Database.getQueryLocator(
            [SELECT Id, Status FROM Refund WHERE Status = 'Pending']
        );
    }

    global void execute(Database.BatchableContext BC, List<sObject> scope){
        List<Refund> refundsToUpdate = (List<Refund>)scope;
        for(Refund r : refundsToUpdate){
            r.Status = 'Completed'; // Update the Status picklist
        }
        update refundsToUpdate;
    }

    global void finish(Database.BatchableContext BC) {
        System.debug('Batch Completed');
    }
```

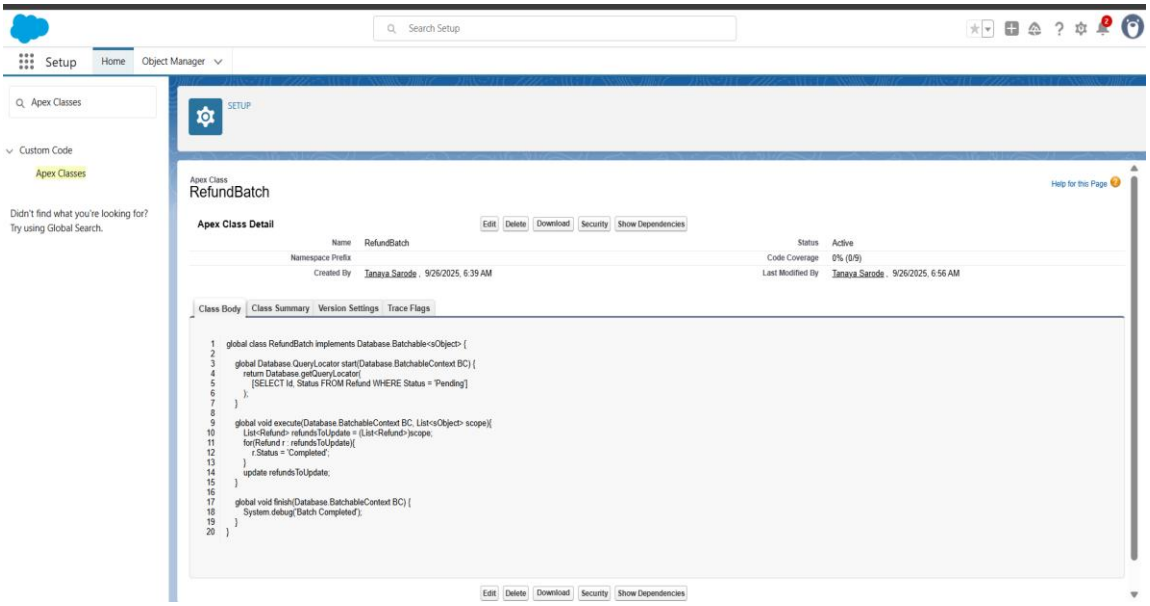
Logs | Tests | Checkpoints | Query Editor | View State | Progress | Problems

User	Application	Operation	Time	Status	Read	Size
Tanaya Sarode	Unknown	/services/data/v64.0/tooling/executeA...	9/26/2025, 7:16:42 PM	Success	Unread	2.92 KB
Tanaya Sarode	Unknown	Batch Apex	9/26/2025, 7:16:42 PM	Success		3.39 KB
Tanaya Sarode	Unknown	Batch Apex	9/26/2025, 7:16:42 PM	Success	Unread	3.56 KB
Tanaya Sarode	Unknown	/services/data/v64.0/tooling/executeA...	9/26/2025, 7:16:18 PM	Success	Unread	2.95 KB
Tanaya Sarode	Unknown	Batch Apex	9/26/2025, 7:16:18 PM	Success	Unread	3.4 KB
Tanaya Sarode	Unknown	Batch Apex	9/26/2025, 7:16:18 PM	Success	Unread	3.55 KB

---

5. Testing and Verification

- Test Return Requests were updated to **Approved**, and the trigger successfully generated Refund records.
- Bulk update tests (multiple Return Requests approved at once) confirmed that the trigger handled all records without hitting governor limits.
- Negative test cases (e.g., requests already linked to a refund) ensured that duplicate refunds were not created.



---

6. Expected Outcomes

- **Reduced Manual Work:** Refunds are created automatically without admin intervention.
- **Data Consistency:** Refunds are always tied to approved return requests, preventing missed cases.
- **Scalable Logic:** Bulk-safe coding ensures smooth handling of large datasets.
- **Stronger System Reliability:** Apex test classes validate logic and safeguard against regressions.

---

7. Conclusion

Phase 5 successfully demonstrated the use of **Apex programming** to extend Salesforce capabilities. By implementing triggers, classes, and test classes, the Return Flow system is now more robust and intelligent. This phase lays the foundation for **future enhancements**, including integration with external payment gateways and logistics APIs. The project is now ready to proceed to **Phase 6: Integration and Advanced Features**.