Phase 5: Apex Programming (Developer)

AgriConnect

Salesforce-Based Farmer Support & Marketplace System

Step 1: Classes & Objects

Apex classes are reusable code units that implement business logic. Objects (standard/custom) hold data; classes operate on object records.

➢ **What to create:**

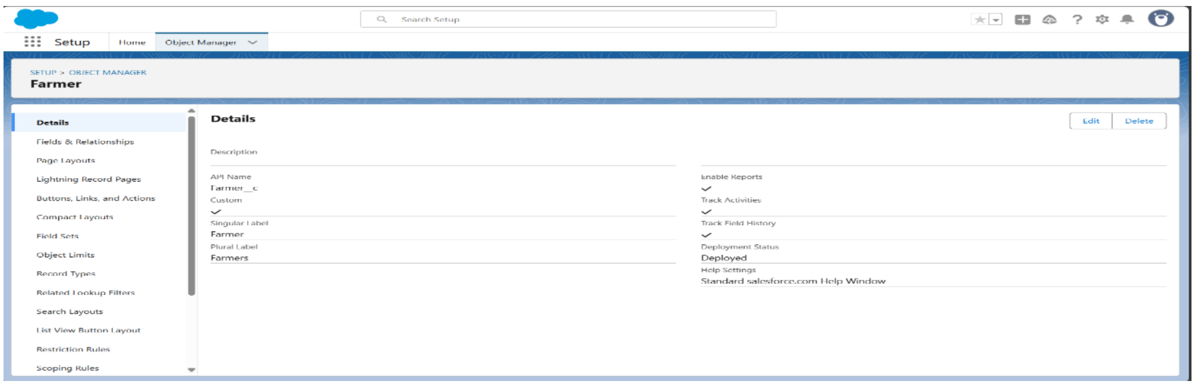
- Apex helper classes (e.g., DataQueryService, OrderHelper).
- Utility classes for shared logic.

➢ **Where to create:**

- Developer Console → *File* → *New* → *Apex Class* (name: DataQueryService).
- Setup → Object Manager → *[Your Object]* → *Fields & Relationships* to create custom fields.

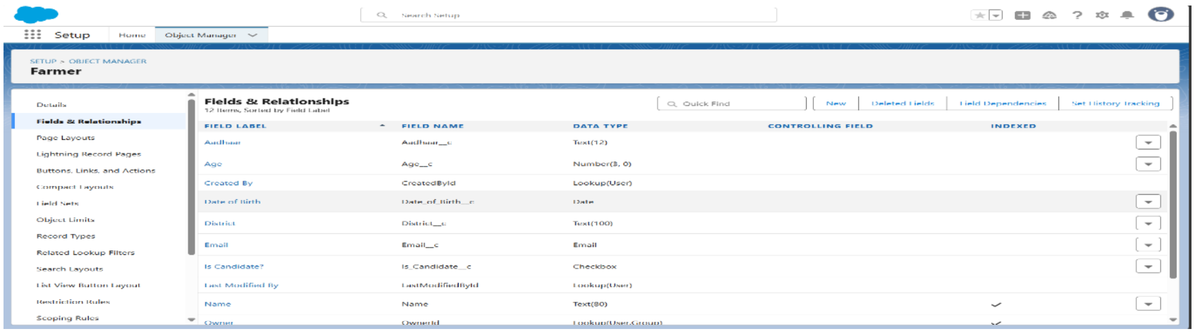
➢ **Steps:**

- Decide object(s) you will act on (e.g., Order__c).
- Create any missing custom fields needed (Status__c, Description__c).
- Create Apex classes with small focused methods (query, update, helper methods).



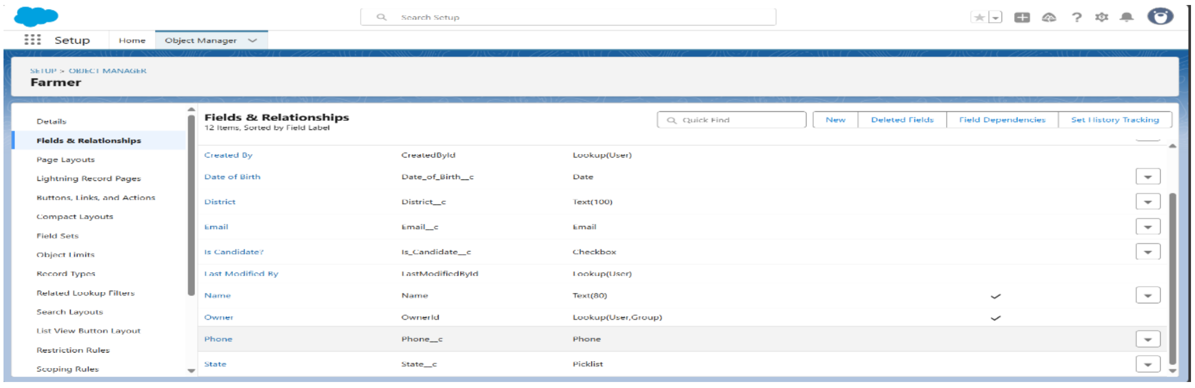
Details

Description	
API Name	Farmer__c
Custom	✓
Singular Label	Farmer
Plural Label	Farmers
Enable Reports	✓
Track Activities	✓
Track Field History	✓
Deployment Status	Deployed
Help Settings	Standard salesforce.com Help Window



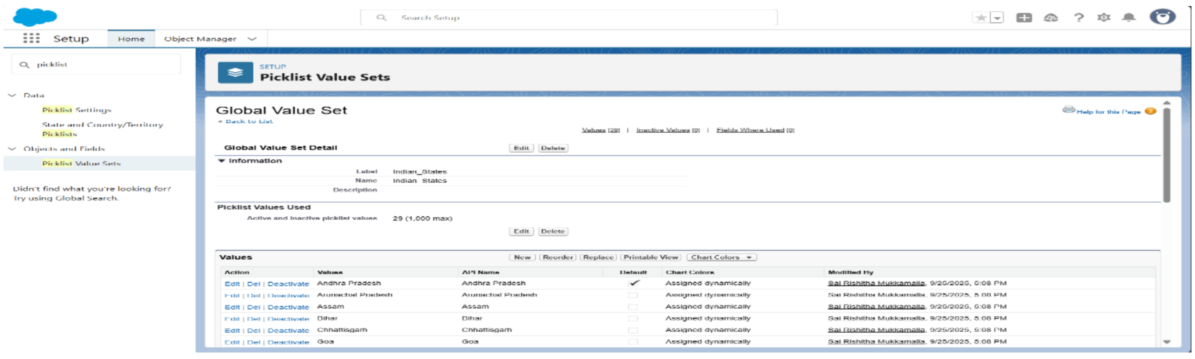
Fields & Relationships
12 Items, Sorted by Field Label

FIELD LABEL	FIELD NAME	DATA TYPE	CONTROLLING FIELD	INDEXED
Aadhaar	Aadhaar__c	Text(12)		
Age	Age__c	Number(3, 0)		
Created By	CreatedById	Lookup(User)		
Date of Birth	Date_of_Birth__c	Date		
District	District__c	Text(100)		
Email	Email__c	Email		
Is Candidate?	Is_Candidate__c	Checkbox		
Last Modified By	LastModifiedById	Lookup(User)		
Name	Name	Text(80)		✓
Owner	OwnerId	Lookup(User,Group)		✓



Fields & Relationships
12 Items, Sorted by Field Label

Created By	CreatedById	Lookup(User)	
Date of Birth	Date_of_Birth__c	Date	
District	District__c	Text(100)	
Email	Email__c	Email	
Is Candidate?	Is_Candidate__c	Checkbox	
Last Modified By	LastModifiedById	Lookup(User)	
Name	Name	Text(80)	✓
Owner	OwnerId	Lookup(User,Group)	✓
Phone	Phone__c	Phone	
State	State__c	Picklist	



Global Value Set

Global Value Set Detail

Information

Label	Indian_States
Name	Indian_States
Description	

Picklist Values Used

Active and inactive picklist values	29 (1,000 max)

Values

Action	Values	API Name	Default	Chart Colors	Modified By
Edit | Del | Deactivate	Andhra Pradesh	Andhra Pradesh	✓	Assigned dynamically	Sai Rishitha Mukkamalla, 9/25/2025, 5:08 PM
Edit | Del | Deactivate	Arunachal Pradesh	Arunachal Pradesh		Assigned dynamically	Sai Rishitha Mukkamalla, 9/25/2025, 5:08 PM
Edit | Del | Deactivate	Assam	Assam		Assigned dynamically	Sai Rishitha Mukkamalla, 9/25/2025, 5:08 PM
Edit | Del | Deactivate	Bihar	Bihar		Assigned dynamically	Sai Rishitha Mukkamalla, 9/25/2025, 5:08 PM
Edit | Del | Deactivate	Chhattisgarh	Chhattisgarh		Assigned dynamically	Sai Rishitha Mukkamalla, 9/25/2025, 5:08 PM
Edit | Del | Deactivate	Goa	Goa		Assigned dynamically	Sai Rishitha Mukkamalla, 9/25/2025, 5:08 PM

Step 2: Apex Triggers (before/after insert/update/delete)

Trigger code that runs on record DML events. Use them to enforce rules, set defaults, or call handler logic.

- What to create:

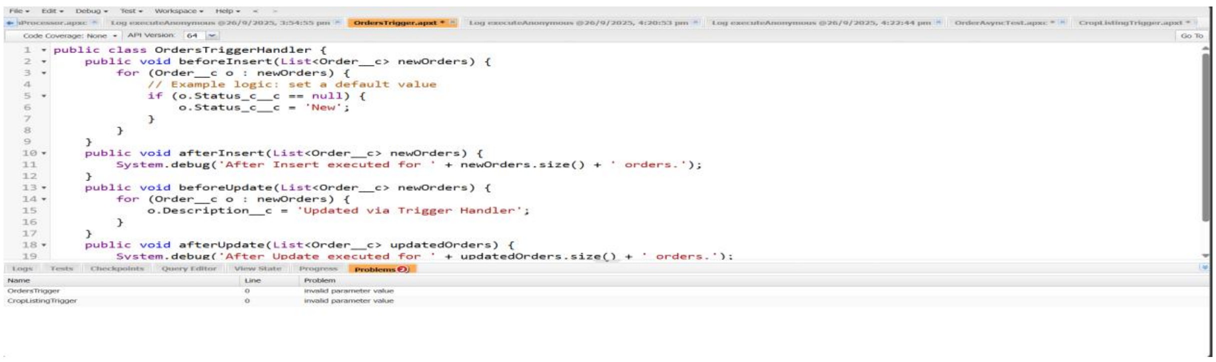
- One trigger per object with minimal logic (e.g., OrdersTrigger).
- Triggers should *delegate* to a handler class.

➢ Where to create:

- Developer Console → *File* → *New* → *Apex Trigger* (select sObject e.g., Order__c).

➢ Steps:

- Create trigger file OrdersTrigger specifying events: before insert, after insert, before update, after update, before delete, after delete.
- Keep trigger lean — only create OrdersTriggerHandler instance and call appropriate methods.
- Save and test by inserting/updating/deleting sample records.



```
public class OrdersTriggerHandler {
    public void beforeInsert(List<Order__c> newOrders) {
        for (Order__c o : newOrders) {
            // Example logic: set a default value
            if (o.Status_c__c == null) {
                o.Status_c__c = 'New';
            }
        }
    }
    public void afterInsert(List<Order__c> newOrders) {
        System.debug('After Insert executed for ' + newOrders.size() + ' orders.');
    }
    public void beforeUpdate(List<Order__c> newOrders) {
        for (Order__c o : newOrders) {
            o.Description__c = 'Updated via Trigger Handler';
        }
    }
    public void afterUpdate(List<Order__c> updatedOrders) {
        System.debug('After Update executed for ' + updatedOrders.size() + ' orders.');
```

Name	Line	Problem
OrdersTrigger	0	invalid parameter value
CropListingTrigger	0	invalid parameter value

Step 3: Trigger Design Pattern

A design pattern that separates the trigger (entry point) from business logic (handler class) and supports bulk operations.

➢ **What to create:**

- Trigger Handler class for each object (e.g., OrdersTriggerHandler). Methods to include:
    - beforeInsert(List<Order__c>)
    - afterInsert(List<Order__c>)
    - beforeUpdate(List<Order__c>, Map<Id,Order__c>)
    - afterUpdate(List<Order__c>, Map<Id,Order__c>)
    - beforeDelete(Map<Id,Order__c>)

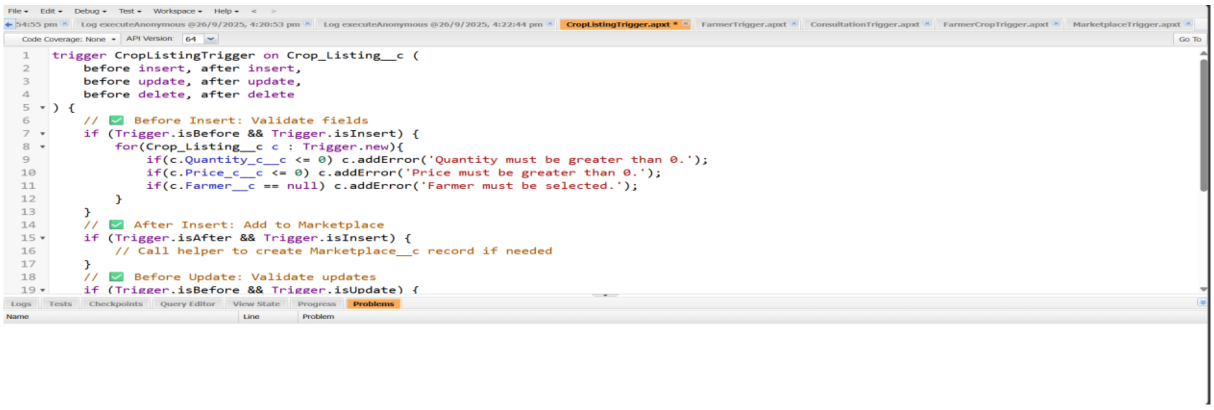
- afterDelete(Map<Id,Order__c>)

➢ **Where to create:**

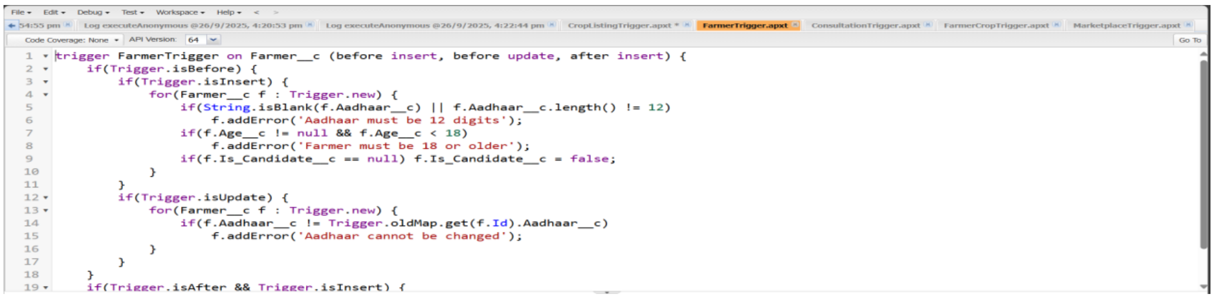
- Developer Console → *File* → *New* → *Apex Class* (name: OrdersTriggerHandler).

➢ **Steps:**

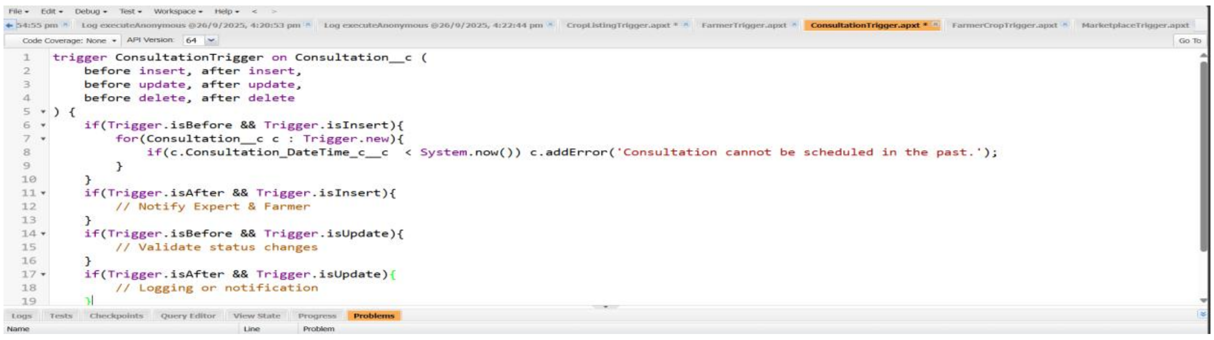
- Create handler class first, then create trigger that calls handler.
- Implement bulk-safe logic (collect Ids, single SOQL, use Maps for lookups).
- Add System.debug statements to help log which event ran during tests.



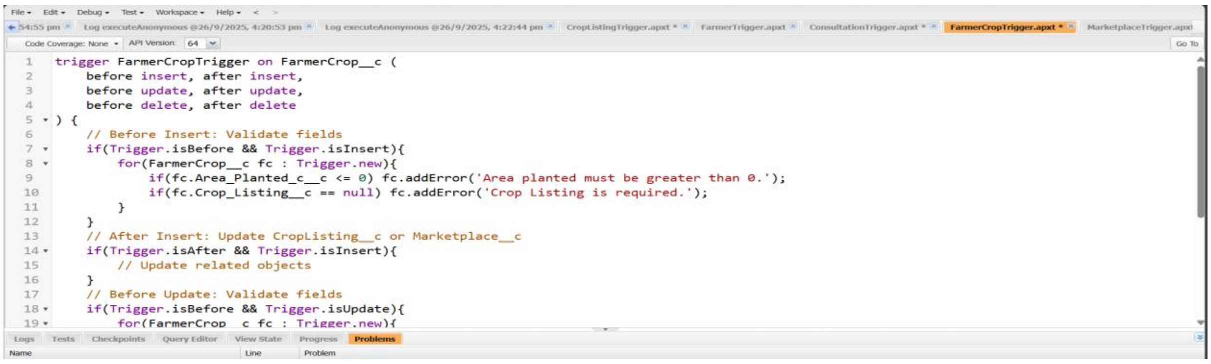
```
trigger CropListingTrigger on Crop_Listing__c (
    before insert, after insert,
    before update, after update,
    before delete, after delete
) {
    // ✅ Before Insert: Validate fields
    if (Trigger.isBefore && Trigger.isInsert) {
        for(Crop_Listing__c c : Trigger.new){
            if(c.Quantity_c__c <= 0) c.addError('Quantity must be greater than 0.');
            if(c.Price_c__c <= 0) c.addError('Price must be greater than 0.');
            if(c.Farmer__c == null) c.addError('Farmer must be selected.');
        }
    }
    // ✅ After Insert: Add to Marketplace
    if (Trigger.isAfter && Trigger.isInsert) {
        // Call helper to create Marketplace__c record if needed
    }
    // ✅ Before Update: Validate updates
    if (Trigger.isBefore && Trigger.isUpdate) {
```



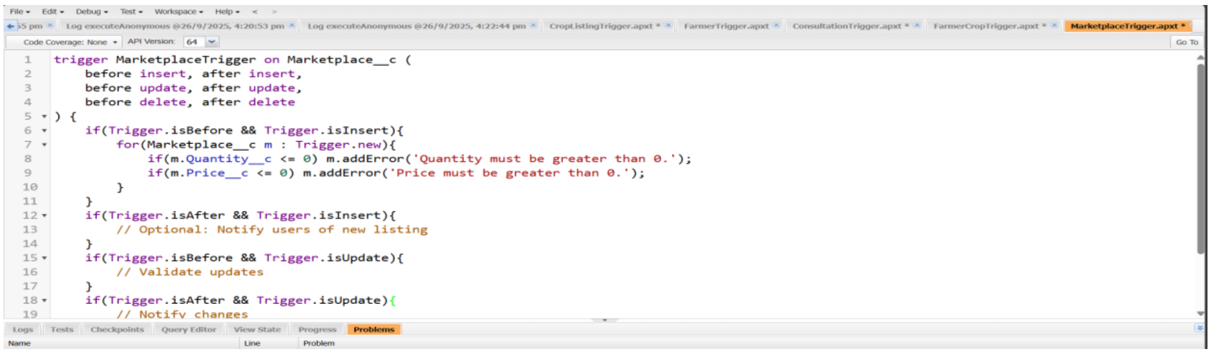
```
trigger FarmerTrigger on Farmer__c (before insert, before update, after insert) {
    if(Trigger.isBefore) {
        if(Trigger.isInsert) {
            for(Farmer__c f : Trigger.new) {
                if(String.isBlank(f.Aadhaar__c) || f.Aadhaar__c.length() != 12)
                    f.addError('Aadhaar must be 12 digits');
                if(f.Age__c != null && f.Age__c < 18)
                    f.addError('Farmer must be 18 or older');
                if(f.Is_Candidate__c == null) f.Is_Candidate__c = false;
            }
        }
        if(Trigger.isUpdate) {
            for(Farmer__c f : Trigger.new) {
                if(f.Aadhaar__c != Trigger.oldMap.get(f.Id).Aadhaar__c)
                    f.addError('Aadhaar cannot be changed');
            }
        }
    }
    if(Trigger.isAfter && Trigger.isInsert) {
```



```
trigger ConsultationTrigger on Consultation__c (
    before insert, after insert,
    before update, after update,
    before delete, after delete
) {
    if(Trigger.isBefore && Trigger.isInsert){
        for(Consultation__c c : Trigger.new){
            if(c.Consultation_DateTime_c__c  < System.now()) c.addError('Consultation cannot be scheduled in the past.');
        }
    }
    if(Trigger.isAfter && Trigger.isInsert){
        // Notify Expert & Farmer
    }
    if(Trigger.isBefore && Trigger.isUpdate){
        // Validate status changes
    }
    if(Trigger.isAfter && Trigger.isUpdate){
        // Logging or notification
    }
```



```
trigger FarmerCropTrigger on FarmerCrop__c (
    before insert, after insert,
    before update, after update,
    before delete, after delete
) {
    // Before Insert: Validate fields
    if(Trigger.isBefore && Trigger.isInsert){
        for(FarmerCrop__c fc : Trigger.new){
            if(fc.Area_Planted_c__c <= 0) fc.addError('Area planted must be greater than 0.');
            if(fc.Crop_Listing__c == null) fc.addError('Crop Listing is required.');
        }
    }
    // After Insert: Update CropListing__c or Marketplace__c
    if(Trigger.isAfter && Trigger.isInsert){
        // Update related objects
    }
    // Before Update: Validate fields
    if(Trigger.isBefore && Trigger.isUpdate){
        for(FarmerCrop__c fc : Trigger.new){
```

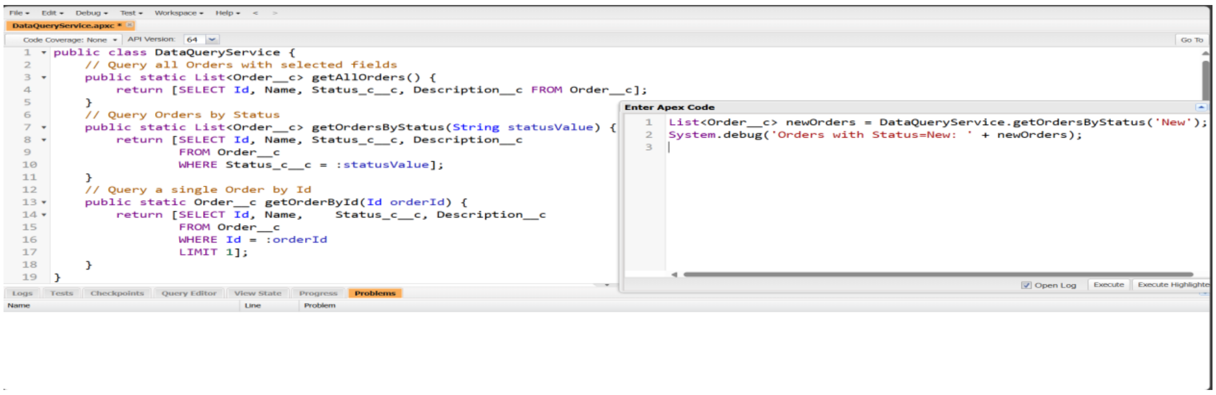


```
trigger MarketplaceTrigger on Marketplace__c (
    before insert, after insert,
    before update, after update,
    before delete, after delete
) {
    if(Trigger.isBefore && Trigger.isInsert){
        for(Marketplace__c m : Trigger.new){
            if(m.Quantity__c <= 0) m.addError('Quantity must be greater than 0.');
            if(m.Price__c <= 0) m.addError('Price must be greater than 0.');
        }
    }
    if(Trigger.isAfter && Trigger.isInsert){
        // Optional: Notify users of new listing
    }
    if(Trigger.isBefore && Trigger.isUpdate){
        // Validate updates
    }
    if(Trigger.isAfter && Trigger.isUpdate){
        // Notify changes
```

Step 4: SOQL & SOSL

SOQL queries records from one object; SOSL searches across multiple objects/fields.

- **What to create:**
  - Reusable query methods inside DataQueryService.
- **Where to create / run:**
  - Queries inside Apex Classes or Execute Anonymous (Developer Console → Debug → Execute Anonymous). Quick SOQL in Developer Console → *Query Editor*.
- **Steps:**
  - Verify API names (Object Manager) before writing queries.
  - Start with simple queries: SELECT Id, Name FROM Order__c LIMIT 10.
  - Use bind variables in Apex (WHERE Status__c = :statusVar).
  - Use SOSL only for text search; ensure object has “Allow Search” enabled for SOSL.



```
public class DataQueryService {
    // Query all Orders with selected fields
    public static List<Order__c> getAllOrders() {
        return [SELECT Id, Name, Status_c__c, Description__c FROM Order__c];
    }
    // Query Orders by Status
    public static List<Order__c> getOrdersByStatus(String statusValue) {
        return [SELECT Id, Name, Status_c__c, Description__c
                FROM Order__c
                WHERE Status_c__c = :statusValue];
    }
    // Query a single Order by Id
    public static Order__c getOrderById(Id orderId) {
        return [SELECT Id, Name,    Status_c__c, Description__c
                FROM Order__c
                WHERE Id = :orderId
                LIMIT 1];
    }
}
```

Enter Apex Code

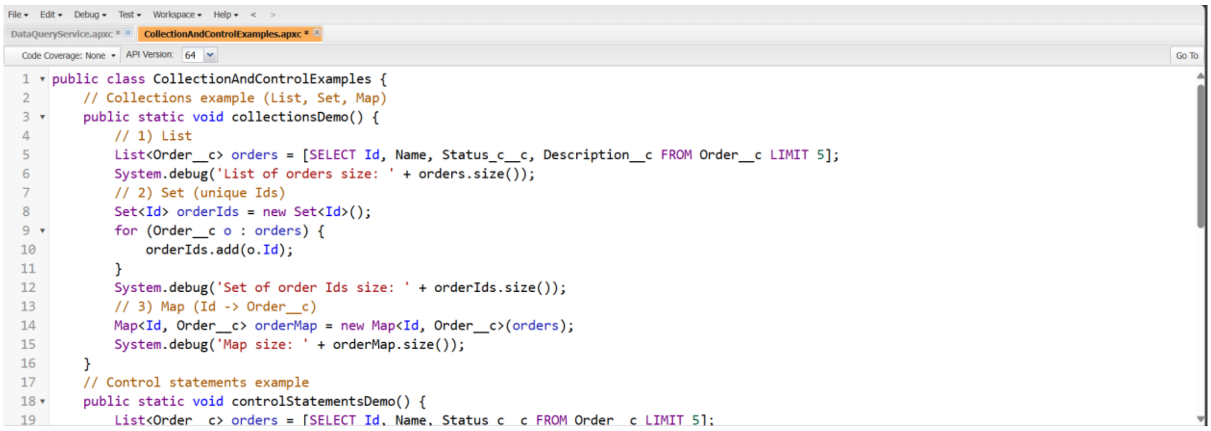
```
List<Order__c> newOrders = DataQueryService.getOrdersByStatus('New');
System.debug('Orders with Status=New: ' + newOrders);
```

Open Log
Execute
Execute Highlighte
Logs Tests Checkpoints Query Editor View State Progress Problems
Name Line Problem

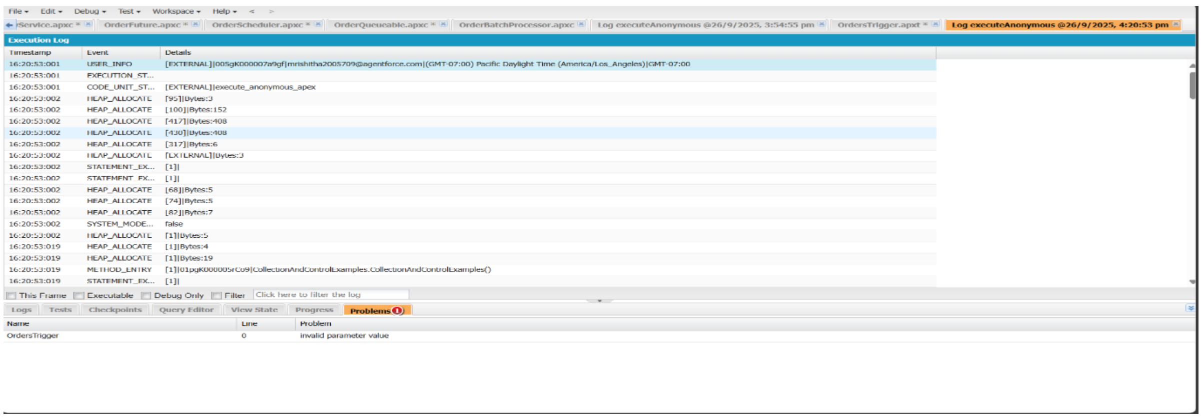
Step 5: Collections: List, Set, Map

Collections store multiple values: Lists (ordered), Sets (unique), Maps (key-value). They are essential for bulk-safe Apex.

- **What to create:**
  - Use Lists to hold sObjects, Sets to collect Ids, Maps for fast lookups.
- **Where to create:**
  - Inside Apex classes and trigger handlers.
- **Steps:**
  - In any handler method, collect parent Ids into a Set<Id>.
  - Query parent objects with single SOQL WHERE Id IN :set.
  - Store results in Map<Id, SObject> for quick access.


File Edit Debug Test Workspace Help
DataQueryService.apxc *
CollectionAndControlExamples.apxc *
Code Coverage: None
API Version: 64
Go To

```
public class CollectionAndControlExamples {
    // Collections example (List, Set, Map)
    public static void collectionsDemo() {
        // 1) List
        List<Order__c> orders = [SELECT Id, Name, Status_c__c, Description__c FROM Order__c LIMIT 5];
        System.debug('List of orders size: ' + orders.size());
        // 2) Set (unique Ids)
        Set<Id> orderIds = new Set<Id>();
        for (Order__c o : orders) {
            orderIds.add(o.Id);
        }
        System.debug('Set of order Ids size: ' + orderIds.size());
        // 3) Map (Id -> Order__c)
        Map<Id, Order__c> orderMap = new Map<Id, Order__c>(orders);
        System.debug('Map size: ' + orderMap.size());
    }
    // Control statements example
    public static void controlStatementsDemo() {
        List<Order__c> orders = [SELECT Id, Name, Status_c__c FROM Order__c LIMIT 5];
```



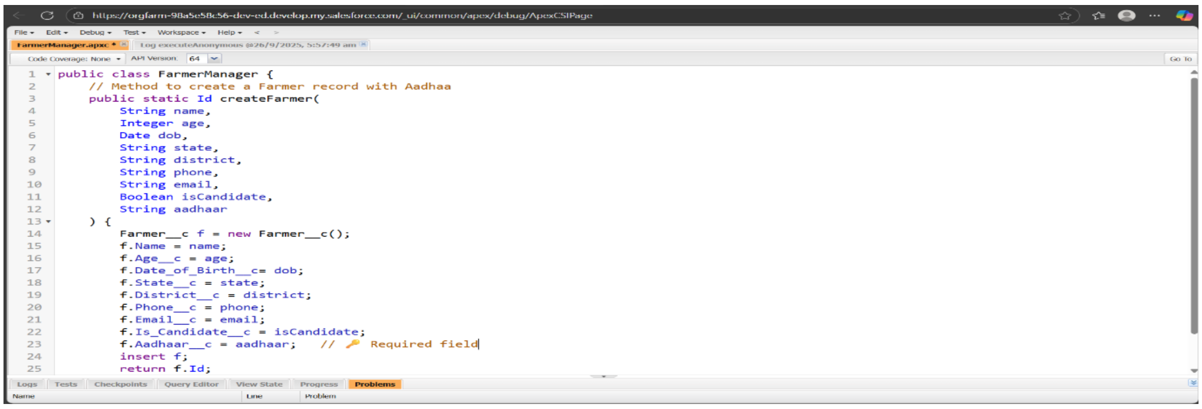
```
Timestamp     Event            Details
16:20:53:001  USER_INFO        [EXTERNAL]|005gK000007a9gf|mrishitha2005709@agentforce.com|(GMT-07:00) Pacific Daylight Time (America/Los_Angeles)|GMT-07:00
16:20:53:001  EXECUTION_ST...
16:20:53:001  CODE_UNIT_ST...  [EXTERNAL]|execute_anonymous_apex
16:20:53:002  HEAP_ALLOCATE    [95]|Bytes:3
16:20:53:002  HEAP_ALLOCATE    [100]|Bytes:152
16:20:53:002  HEAP_ALLOCATE    [417]|Bytes:408
16:20:53:002  HEAP_ALLOCATE    [430]|Bytes:408
16:20:53:002  HEAP_ALLOCATE    [317]|Bytes:6
16:20:53:002  HEAP_ALLOCATE    [EXTERNAL]|Bytes:3
16:20:53:002  STATEMENT_EX...  [1]|
16:20:53:002  STATEMENT_EX...  [1]|
16:20:53:002  HEAP_ALLOCATE    [68]|Bytes:5
16:20:53:002  HEAP_ALLOCATE    [74]|Bytes:5
16:20:53:002  HEAP_ALLOCATE    [82]|Bytes:7
16:20:53:002  SYSTEM_MODE...   false
16:20:53:002  HEAP_ALLOCATE    [1]|Bytes:5
16:20:53:019  HEAP_ALLOCATE    [1]|Bytes:4
16:20:53:019  HEAP_ALLOCATE    [1]|Bytes:19
16:20:53:019  METHOD_ENTRY     [1]|01pgK000005rCo9|CollectionAndControlExamples.CollectionAndControlExamples()
16:20:53:019  STATEMENT_EX...  [1]|
```

Name	Line	Problem
OrdersTrigger	0	invalid parameter value

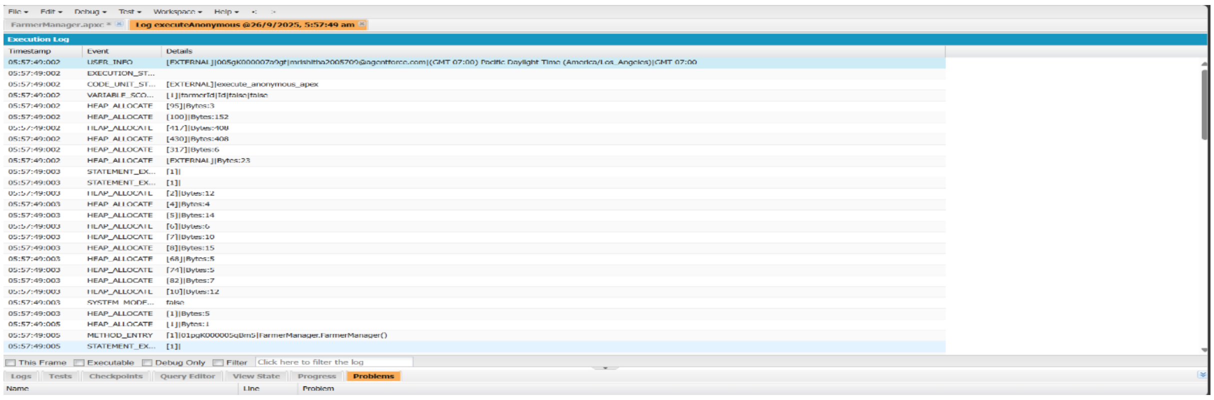
Step 6: Control Statements

if/else, for, while, switch used to implement decision logic.

- **What to create:**
  - Methods demonstrating logic in ControlStatementsExample class.
- **Where to create:**
  - Developer Console → *File* → *New* → *Apex Class* (name: ControlStatementsExample).
- **Steps:**
  - Use for to iterate Trigger.new.
  - Use if/else to validate fields.
  - Use switch for multi-branch conditions where supported.



```
public class FarmerManager {
    // Method to create a Farmer record with Aadhaa
    public static Id createFarmer(
        String name,
        Integer age,
        Date dob,
        String state,
        String district,
        String phone,
        String email,
        Boolean isCandidate,
        String aadhaar
    ) {
        Farmer__c f = new Farmer__c();
        f.Name = name;
        f.Age__c = age;
        f.Date_of_Birth__c= dob;
        f.State__c = state;
        f.District__c = district;
        f.Phone__c = phone;
        f.Email__c = email;
        f.Is_Candidate__c = isCandidate;
        f.Aadhaar__c = aadhaar;   // 🔑 Required field
        insert f;
        return f.Id;
```

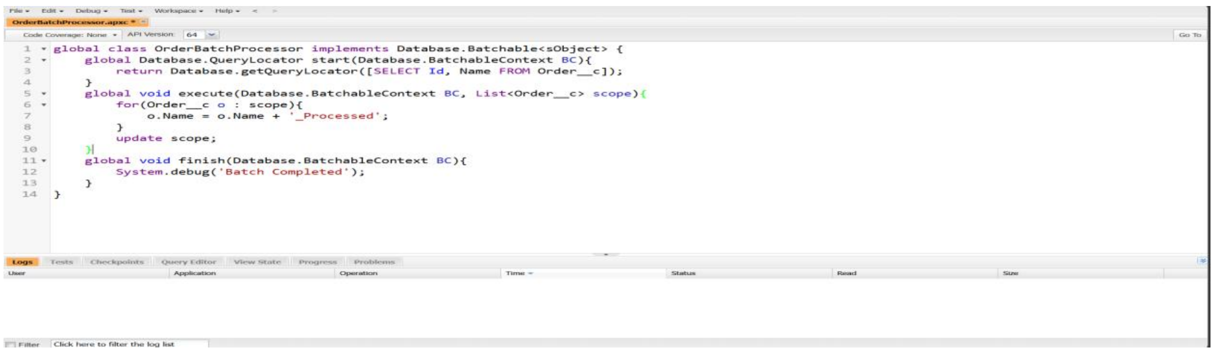


```
Timestamp     Event            Details
05:57:49:002  USER_INFO        [EXTERNAL]|005gK000007a9gf|mrishitha2005709@agentforce.com|(GMT 07:00) Pacific Daylight Time (America/Los_Angeles)|GMT 07:00
05:57:49:002  EXECUTION_ST...
05:57:49:002  CODE_UNIT_ST...  [EXTERNAL]|execute_anonymous_apex
05:57:49:002  VARIABLE_SCO...  [1]|farmerId|Id|false|false
05:57:49:002  HEAP_ALLOCATE    [95]|Bytes:3
05:57:49:002  HEAP_ALLOCATE    [100]|Bytes:152
05:57:49:002  HEAP_ALLOCATE    [417]|Bytes:408
05:57:49:002  HEAP_ALLOCATE    [430]|Bytes:408
05:57:49:002  HEAP_ALLOCATE    [317]|Bytes:6
05:57:49:002  HEAP_ALLOCATE    [EXTERNAL]|Bytes:23
05:57:49:003  STATEMENT_EX...  [1]|
05:57:49:003  STATEMENT_EX...  [1]|
05:57:49:003  HEAP_ALLOCATE    [2]|Bytes:12
05:57:49:003  HEAP_ALLOCATE    [4]|Bytes:4
05:57:49:003  HEAP_ALLOCATE    [5]|Bytes:14
05:57:49:003  HEAP_ALLOCATE    [6]|Bytes:6
05:57:49:003  HEAP_ALLOCATE    [7]|Bytes:10
05:57:49:003  HEAP_ALLOCATE    [8]|Bytes:15
05:57:49:003  HEAP_ALLOCATE    [68]|Bytes:5
05:57:49:003  HEAP_ALLOCATE    [74]|Bytes:5
05:57:49:003  HEAP_ALLOCATE    [82]|Bytes:7
05:57:49:003  HEAP_ALLOCATE    [10]|Bytes:12
05:57:49:003  SYSTEM_MODE...   false
05:57:49:003  HEAP_ALLOCATE    [1]|Bytes:5
05:57:49:005  HEAP_ALLOCATE    [1]|Bytes:1
05:57:49:005  METHOD_ENTRY     [1]|01pgK000005qBm5|FarmerManager.FarmerManager()
05:57:49:005  STATEMENT_EX...  [1]|
```

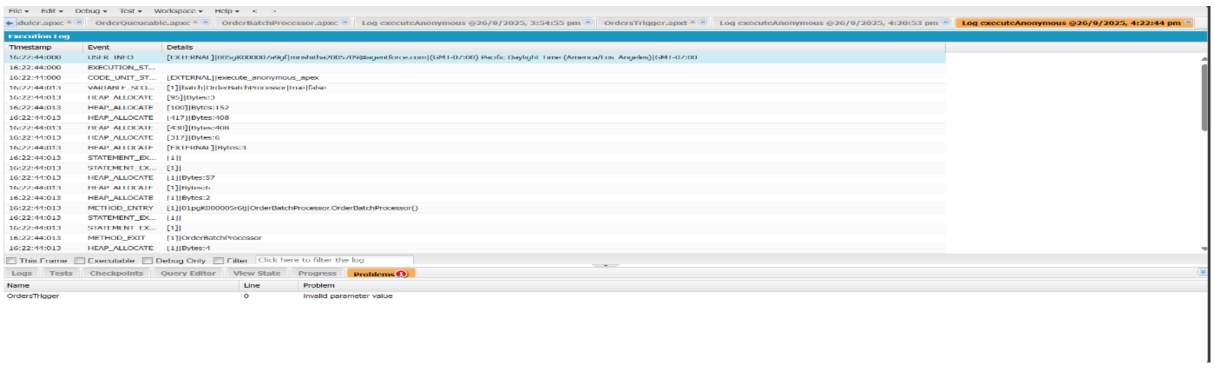
Step 7: Batch Apex

Process large volumes of records asynchronously in chunks. Use when records > 50,000 or heavy processing required.

- What to create:
  - Global batch class: OrderBatchProcessor implements Database.Batchable<sObject> with start, execute, finish.
- Where to create:
  - Developer Console → *File* → *New* → *Apex Class* (name: OrderBatchProcessor).
- Steps:
  - Implement start to return Database.getQueryLocator([SELECT ... FROM Order__c WHERE ...]).
  - Implement execute(Database.BatchableContext, List<Order__c> scope) to process and update scope.
  - Use Database.executeBatch(new OrderBatchProcessor(), 200) in Execute Anonymous.
  - Use Test.startTest()/Test.stopTest() in test class to run batch in tests.



```
global class OrderBatchProcessor implements Database.Batchable<sObject> {
    global Database.QueryLocator start(Database.BatchableContext BC){
        return Database.getQueryLocator([SELECT Id, Name FROM Order__c]);
    }
    global void execute(Database.BatchableContext BC, List<Order__c> scope){
        for(Order__c o : scope){
            o.Name = o.Name + '_Processed';
        }
        update scope;
    }
    global void finish(Database.BatchableContext BC){
        System.debug('Batch Completed');
    }
}
```



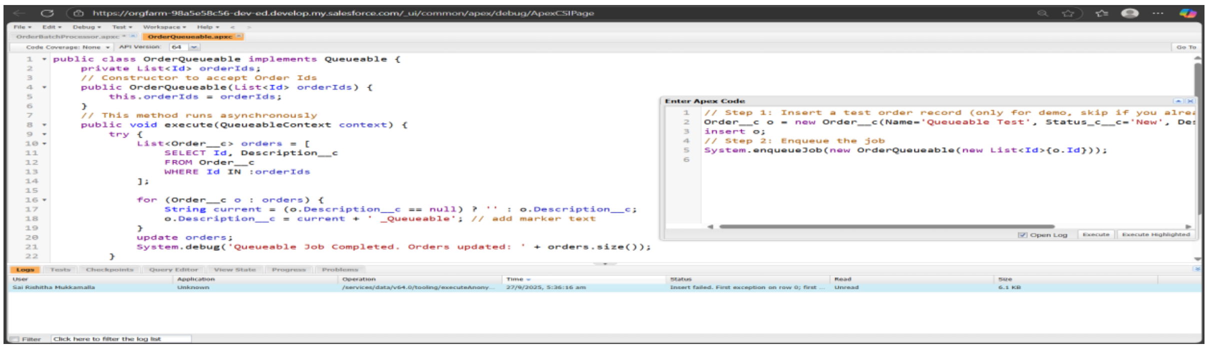
```
Execution Log
Timestamp       Event               Details
16:22:44:000    USER_INFO           [EXTERNAL]|005gK000007a9gf|rnishitha2005709@agentforce.com|(GMT-07:00) Pacific Daylight Time (America/Los_Angeles)|GMT-07:00
16:22:44:000    EXECUTION_ST...
16:22:44:000    CODE_UNIT_ST...     [EXTERNAL]|execute_anonymous_apex
16:22:44:013    VARIABLE_SCO...     [1]|batch|OrderBatchProcessor|true|false
16:22:44:013    HEAP_ALLOCATE       [95]|Bytes:3
16:22:44:013    HEAP_ALLOCATE       [100]|Bytes:152
16:22:44:013    HEAP_ALLOCATE       [417]|Bytes:408
16:22:44:013    HEAP_ALLOCATE       [430]|Bytes:408
16:22:44:013    HEAP_ALLOCATE       [317]|Bytes:6
16:22:44:013    HEAP_ALLOCATE       [EXTERNAL]|Bytes:3
16:22:44:013    STATEMENT_EX...     [1]|
16:22:44:013    STATEMENT_EX...     [1]|
16:22:44:013    HEAP_ALLOCATE       [1]|Bytes:57
16:22:44:013    HEAP_ALLOCATE       [1]|Bytes:6
16:22:44:013    HEAP_ALLOCATE       [1]|Bytes:2
16:22:44:013    METHOD_ENTRY        [1]|01pgK000005rGIJ|OrderBatchProcessor.OrderBatchProcessor()
16:22:44:013    STATEMENT_EX...     [1]|
16:22:44:013    STATEMENT_EX...     [1]|
16:22:44:013    METHOD_EXIT         [1]|OrderBatchProcessor
16:22:44:013    HEAP_ALLOCATE       [1]|Bytes:4

Name            Line    Problem
OrdersTrigger   0       Invalid parameter value
```

Step 8: Queueable Apex

Lighter-weight async jobs that can be chained and accept complex types.

- What to create:
    - OrderQueueable implements Queueable with execute(QueueableContext).
- Where to create:
    - Developer Console → *File* → *New* → *Apex Class* (name: OrderQueueable).
- Steps:
    - Implement execute to perform query/update and System.enqueueJob(new OrderQueueable()) to run.
    - Use for medium-sized asynchronous tasks.
    - Test with Test.startTest() / Test.stopTest()



```
https://orgfarm-98a5e58c56-dev-ed.develop.my.salesforce.com/_ui/common/apex/debug/ApexCSIPage
```

```
public class OrderQueueable implements Queueable {
    private List<Id> orderIds;
    // Constructor to accept Order Ids
    public OrderQueueable(List<Id> orderIds) {
        this.orderIds = orderIds;
    }
    // This method runs asynchronously
    public void execute(QueueableContext context) {
        try {
            List<Order__c> orders = [
                SELECT Id, Description__c
                FROM Order__c
                WHERE Id IN :orderIds
            ];

            for (Order__c o : orders) {
                String current = (o.Description__c == null) ? '' : o.Description__c;
                o.Description__c = current + ' _Queueable'; // add marker text
            }
            update orders;
            System.debug('Queueable Job Completed. Orders updated: ' + orders.size());
        }
```

Enter Apex Code

```
// Step 1: Insert a test order record (only for demo, skip if you alrea
Order__c o = new Order__c(Name='Queueable Test', Status_c__c='New', Des
insert o;
// Step 2: Enqueue the job
System.enqueueJob(new OrderQueueable(new List<Id>{o.Id}));
```

```
User                    Application  Operation                                         Time                     Status                                          Read    Size
Sai Rishitha Mukkamalla Unknown      /services/data/v64.0/tooling/executeAnony...      27/9/2025, 5:36:16 am    Insert failed. First exception on row 0; first ...  Unread  6.1 KB
```

Step 9: Scheduled Apex

Run Apex classes at defined times using cron expressions.

➢ **What to create:**

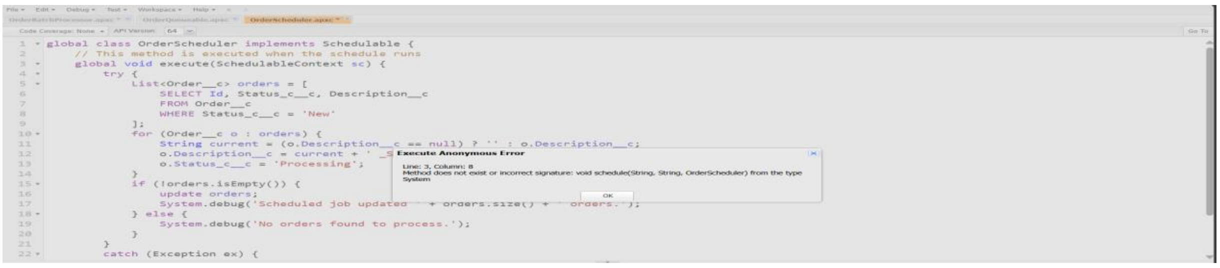
- OrderScheduler implements Schedulable and calls a Queueable or Batch job.

➢ **Where to create:**

- Developer Console → *File* → *New* → *Apex Class* (name: OrderScheduler).
- Schedule via Execute Anonymous or Setup → Apex Classes → Schedule Apex.

➢ **Steps:**

- Implement execute(SchedulableContext) to System.enqueueJob(new OrderQueueable()).
- Schedule with: System.schedule('DailyJob','0 0 12 * * ?', new OrderScheduler());.



```
global class OrderScheduler implements Schedulable {
    // This method is executed when the schedule runs
    global void execute(SchedulableContext sc) {
        try {
            List<Order__c> orders = [
                SELECT Id, Status_c__c, Description__c
                FROM Order__c
                WHERE Status_c__c = 'New'
            ];
            for (Order__c o : orders) {
                String current = (o.Description__c == null) ? '' : o.Description__c;
                o.Description__c = current + ' _S
                o.Status_c__c = 'Processing';
            }
            if (!orders.isEmpty()) {
                update orders;
                System.debug('Scheduled job updated ' + orders.size() + ' orders.');
            } else {
                System.debug('No orders found to process.');
            }
        }
        catch (Exception ex) {
```

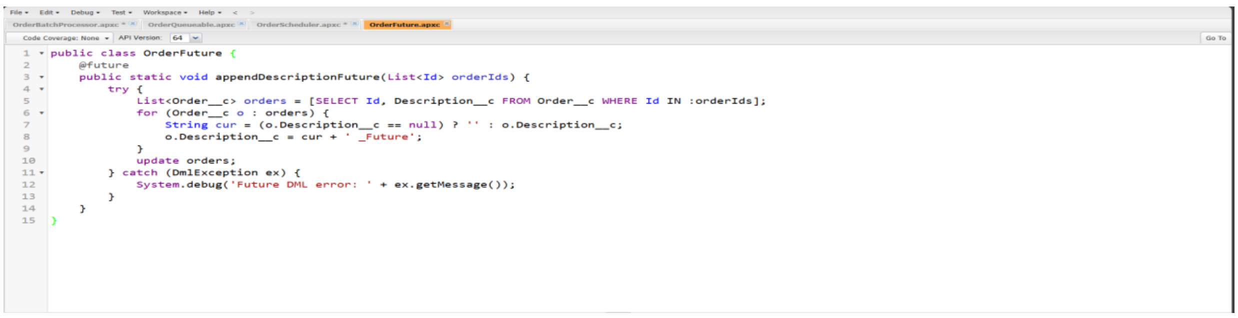
Execute Anonymous Error

```
Line: 3, Column: 8
Method does not exist or incorrect signature: void schedule(String, String, OrderScheduler) from the type
System
```

Step 10: Future Methods

Simple async methods annotated with @future. Use for callouts or fire-and-forget tasks.

- **What to create:**
  - OrderFuture class with @future public static void appendDescriptionFuture(List<Id> orderIds).
- **Where to create:**
  - Developer Console → *File* → *New* → *Apex Class* (name: OrderFuture).
- **Steps:**
  - Method must accept primitives or collections of primitives (e.g., List<Id>).
  - Call via OrderFuture.appendDescriptionFuture(new List<Id>{id1,id2}).
  - Test inside Test.startTest() / Test.stopTest().



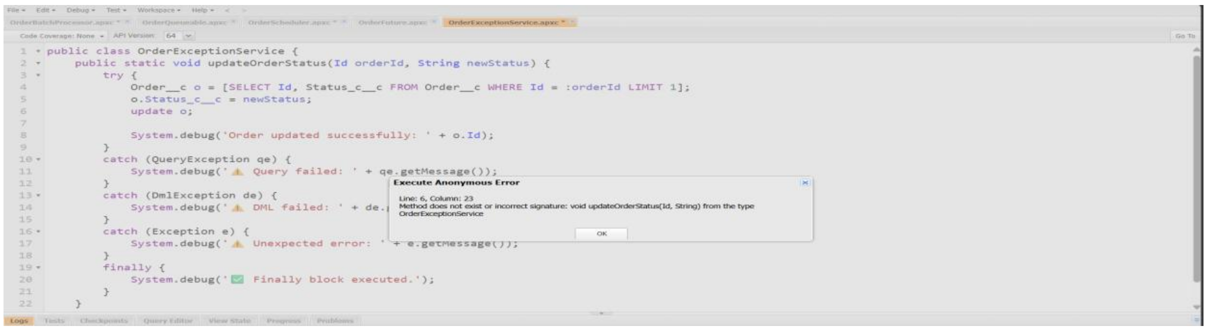
```
public class OrderFuture {
    @future
    public static void appendDescriptionFuture(List<Id> orderIds) {
        try {
            List<Order__c> orders = [SELECT Id, Description__c FROM Order__c WHERE Id IN :orderIds];
            for (Order__c o : orders) {
                String cur = (o.Description__c == null) ? '' : o.Description__c;
                o.Description__c = cur + ' _Future';
            }
            update orders;
        } catch (DmlException ex) {
            System.debug('Future DML error: ' + ex.getMessage());
        }
    }
}
```

Step 11: Exception Handling

Use try/catch/finally to catch DmlException and other Exception types and avoid unhandled runtime errors.

- **What to create:**
  - Wrap DML and callouts in try/catch blocks inside all classes (Batch/Queueable/Future/Handlers).
- **Where to create:**
  - Inside all Apex classes; ensure meaningful logging and graceful fallbacks.
- **Steps:**

- Catch DmlException specifically to examine getDmlMessage(i) if needed.
- Always include finally for cleanup logic where required.



```
public class OrderExceptionService {
    public static void updateOrderStatus(Id orderId, String newStatus) {
        try {
            Order__c o = [SELECT Id, Status_c__c FROM Order__c WHERE Id = :orderId LIMIT 1];
            o.Status_c__c = newStatus;
            update o;

            System.debug('Order updated successfully: ' + o.Id);
        }
        catch (QueryException qe) {
            System.debug('⚠ Query failed: ' + qe.getMessage());
        }
        catch (DmlException de) {
            System.debug('⚠ DML failed: ' + de.
        }
        catch (Exception e) {
            System.debug('⚠ Unexpected error: ' + e.getMessage());
        }
        finally {
            System.debug('✅ Finally block executed.');
        }
    }
```

```
Execute Anonymous Error

Line: 6, Column: 23
Method does not exist or incorrect signature: void updateOrderStatus(Id, String) from the type
OrderExceptionService
```

Step 12: Test Classes

@isTest classes and methods that create test data and assert results; required for deployment (≥75% coverage).

➢ **What to create:**

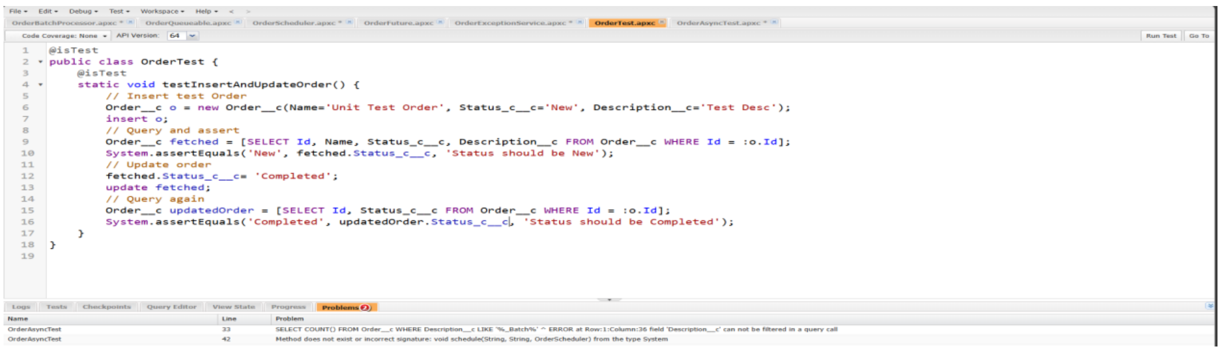
- Test classes for each component: OrderBatchTest, OrderQueueableTest, OrderFutureTest, OrdersTriggerTest (or single OrderAsyncTest).

➢ **Where to create:**

- Developer Console → *File* → *New* → *Apex Class* and annotate with @isTest.

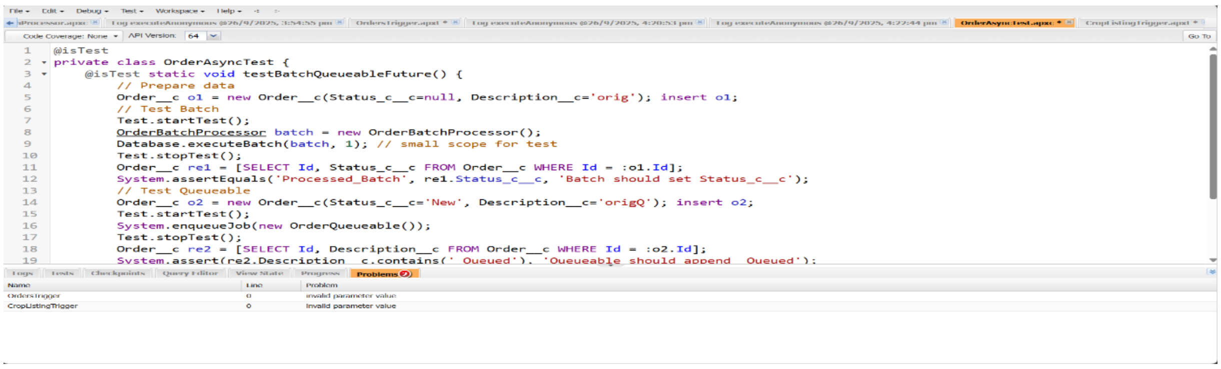
➢ **Steps:**

- Create test records (do not use SeeAllData=true).
- Use Test.startTest() / Test.stopTest() for async execution.
- Assert expected state changes with System.assertEquals or System.assert.



```
@isTest
public class OrderTest {
    @isTest
    static void testInsertAndUpdateOrder() {
        // Insert test Order
        Order__c o = new Order__c(Name='Unit Test Order', Status_c__c='New', Description__c='Test Desc');
        insert o;
        // Query and assert
        Order__c fetched = [SELECT Id, Name, Status_c__c, Description__c FROM Order__c WHERE Id = :o.Id];
        System.assertEquals('New', fetched.Status_c__c, 'Status should be New');
        // Update order
        fetched.Status_c__c= 'Completed';
        update fetched;
        // Query again
        Order__c updatedOrder = [SELECT Id, Status_c__c FROM Order__c WHERE Id = :o.Id];
        System.assertEquals('Completed', updatedOrder.Status_c__c, 'Status should be Completed');
    }
}
```

Name	Line	Problem
OrderAsyncTest	33	SELECT COUNT() FROM Order__c WHERE Description__c LIKE '%_Batch%' ^ ERROR at Row:1:Column:36 field 'Description__c' can not be filtered in a query call
OrderAsyncTest	42	Method does not exist or incorrect signature: void schedule(String, String, OrderScheduler) from the type System



```
@isTest
private class OrderAsyncTest {
    @isTest static void testBatchQueueableFuture() {
        // Prepare data
        Order__c o1 = new Order__c(Status_c__c=null, Description__c='orig'); insert o1;
        // Test Batch
        Test.startTest();
        OrderBatchProcessor batch = new OrderBatchProcessor();
        Database.executeBatch(batch, 1); // small scope for test
        Test.stopTest();
        Order__c re1 = [SELECT Id, Status_c__c FROM Order__c WHERE Id = :o1.Id];
        System.assertEquals('Processed_Batch', re1.Status_c__c, 'Batch should set Status_c__c');
        // Test Queueable
        Order__c o2 = new Order__c(Status_c__c='New', Description__c='origQ'); insert o2;
        Test.startTest();
        System.enqueueJob(new OrderQueueable());
        Test.stopTest();
        Order__c re2 = [SELECT Id, Description__c FROM Order__c WHERE Id = :o2.Id];
        System.assert(re2.Description__c.contains(' Queued'), 'Queueable should append  Queued');
```

Name	Line	Problem
OrdersTrigger	0	invalid parameter value
CropListingTrigger	0	Invalid parameter value

Step 13: Asynchronous Processing

➢ **What to create:**

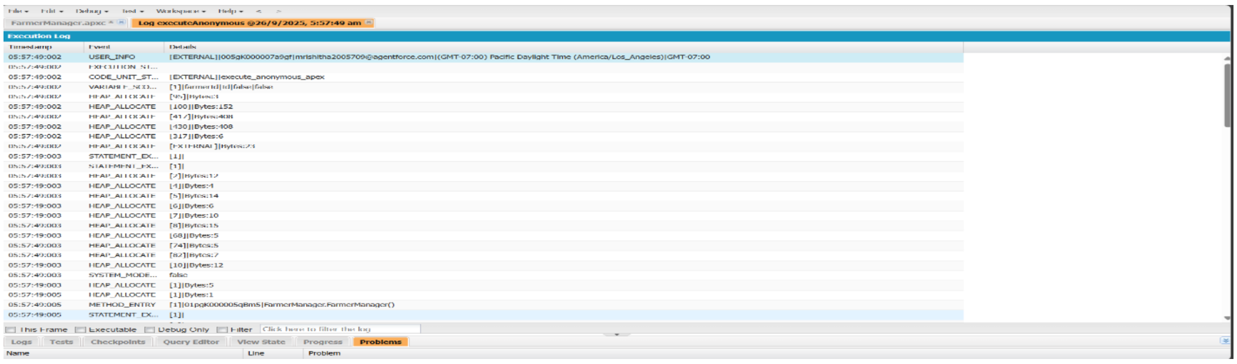
- Implement at least one example of each: Batch (OrderBatchProcessor), Queueable (OrderQueueable), Scheduled (OrderScheduler), Future (OrderFuture).

➢ **Where to create / run:**

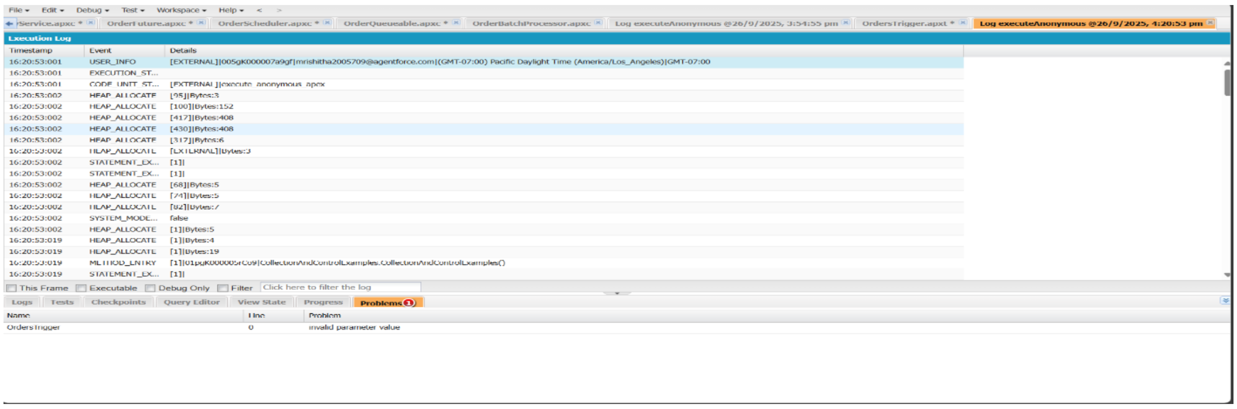
- Developer Console → Apex Class for each implementation. Use Execute Anonymous to run/schedule, and Developer Console Test runner to execute tests.

➢ **Steps:**

- Implement classes, add exception handling and test classes.
- Run each async job manually in Execute Anonymous to verify behavior.
- Run tests and fix any issues shown in logs.


File Edit Debug Test Workspace Help < >
FarmerManager.apxc *
Log executeAnonymous @26/9/2025, 5:57:49 am
Execution Log

```
Timestamp       Event             Details
05:57:49:002    USER_INFO         [EXTERNAL]|005gK000007a9gf|mrishitha2005709@agentforce.com|(GMT-07:00) Pacific Daylight Time (America/Los_Angeles)|GMT-07:00
05:57:49:002    EXECUTION_ST...
05:57:49:002    CODE_UNIT_ST...   [EXTERNAL]|execute_anonymous_apex
05:57:49:002    VARIABLE_SCO...   [1]|farmerId|Id|false|false
05:57:49:002    HEAP_ALLOCATE     [95]|Bytes:3
05:57:49:002    HEAP_ALLOCATE     [100]|Bytes:152
05:57:49:002    HEAP_ALLOCATE     [417]|Bytes:408
05:57:49:002    HEAP_ALLOCATE     [430]|Bytes:408
05:57:49:002    HEAP_ALLOCATE     [317]|Bytes:6
05:57:49:002    HEAP_ALLOCATE     [EXTERNAL]|Bytes:23
05:57:49:003    STATEMENT_EX...   [1]|
05:57:49:003    STATEMENT_EX...   [1]|
05:57:49:003    HEAP_ALLOCATE     [2]|Bytes:12
05:57:49:003    HEAP_ALLOCATE     [4]|Bytes:4
05:57:49:003    HEAP_ALLOCATE     [5]|Bytes:14
05:57:49:003    HEAP_ALLOCATE     [6]|Bytes:6
05:57:49:003    HEAP_ALLOCATE     [7]|Bytes:10
05:57:49:003    HEAP_ALLOCATE     [8]|Bytes:15
05:57:49:003    HEAP_ALLOCATE     [68]|Bytes:5
05:57:49:003    HEAP_ALLOCATE     [74]|Bytes:5
05:57:49:003    HEAP_ALLOCATE     [82]|Bytes:7
05:57:49:003    HEAP_ALLOCATE     [10]|Bytes:12
05:57:49:003    SYSTEM_MODE...    false
05:57:49:003    HEAP_ALLOCATE     [1]|Bytes:5
05:57:49:005    HEAP_ALLOCATE     [1]|Bytes:1
05:57:49:005    METHOD_ENTRY      [1]|01pgK000005qBm5|FarmerManager.FarmerManager()
05:57:49:005    STATEMENT_EX...   [1]|
```



```
Timestamp       Event             Details
16:20:53:001    USER_INFO         [EXTERNAL]|005gK000007a9gf|mrishitha2005709@agentforce.com|(GMT-07:00) Pacific Daylight Time (America/Los_Angeles)|GMT-07:00
16:20:53:001    EXECUTION_ST...
16:20:53:001    CODE_UNIT_ST...   [EXTERNAL]|execute_anonymous_apex
16:20:53:002    HEAP_ALLOCATE     [95]|Bytes:3
16:20:53:002    HEAP_ALLOCATE     [100]|Bytes:152
16:20:53:002    HEAP_ALLOCATE     [417]|Bytes:408
16:20:53:002    HEAP_ALLOCATE     [430]|Bytes:408
16:20:53:002    HEAP_ALLOCATE     [317]|Bytes:6
16:20:53:002    HEAP_ALLOCATE     [EXTERNAL]|Bytes:3
16:20:53:002    STATEMENT_EX...   [1]|
16:20:53:002    STATEMENT_EX...   [1]|
16:20:53:002    HEAP_ALLOCATE     [68]|Bytes:5
16:20:53:002    HEAP_ALLOCATE     [74]|Bytes:5
16:20:53:002    HEAP_ALLOCATE     [82]|Bytes:7
16:20:53:002    SYSTEM_MODE...    false
16:20:53:002    HEAP_ALLOCATE     [1]|Bytes:5
16:20:53:019    HEAP_ALLOCATE     [1]|Bytes:4
16:20:53:019    HEAP_ALLOCATE     [1]|Bytes:19
16:20:53:019    METHOD_ENTRY      [1]|01pgK000005rC09|CollectionAndControlExamples.CollectionAndControlExamples()
16:20:53:019    STATEMENT_EX...   [1]|
```

```
Name                 Line    Problem
OrdersTrigger        0       invalid parameter value
```

Conclusion

In Phase 5 of Apex Programming, we covered the complete development lifecycle for the Order__c object, starting with classes and objects where reusable logic was placed into service classes, and Apex triggers designed with the trigger handler pattern for clean, scalable code. Using SOQL and SOSL, we queried and searched records efficiently, while collections such as List, Set, and Map helped in managing data without duplication. Control statements like if-else and loops enforced business logic. For asynchronous processing, Batch Apex (OrderBatchProcessor) handled large volumes of data, Queueable Apex (OrderQueueable) provided flexible job chaining, Scheduled Apex (OrderScheduler) automated recurring tasks, and Future methods supported lightweight background operations. Exception handling ensured system stability by catching and logging errors through OrderExceptionService. Finally, robust test classes (OrderTest, OrderAsyncTest) were built with Test.startTest() and Test.stopTest() to validate logic and guarantee coverage, ensuring governor limit compliance. Together, these components created a production-ready, scalable, and efficient Salesforce solution that demonstrates best practices in Apex programming with strong focus on maintainability, reliability, and performance.