Phase 5: Apex Programming (Developer)

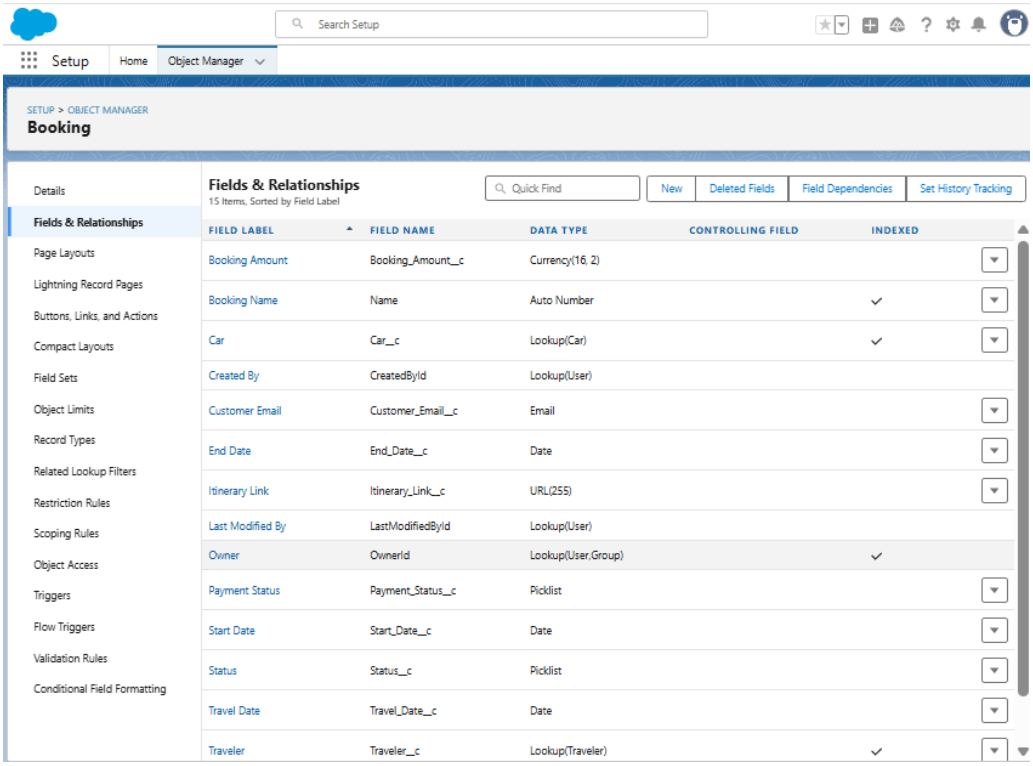
1. Classes & Objects:

Booking__c — booking object

- Car__c (Lookup to Car__c)
- Start_Date__c (Date)
- End_Date__c (Date)
- Status__c (Picklist: e.g. 'Available','Confirmed','Rented','Overdue')

Car__c — car object

- Status__c Picklist:(('Available','Unavailable')



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

- In Salesforce, click the gear icon → Developer Console.
- In Developer Console: File → New → Apex Class.
- In the popup, type BookingService → click OK.

CODE:

```
public with sharing class BookingService {
```

```
    public static Boolean rangesOverlap(Date aStart, Date aEnd, Date
bStart, Date bEnd) {

        if (aStart == null || aEnd == null || bStart == null || bEnd == null)
return false;

        return !(aStart > bEnd || aEnd < bStart);

    }

   private static Map<Id, List<Booking__c>>
getExistingBookingsByCar(

        Set<Id> carIds,

        Date windowStart,

        Date windowEnd,

        Set<Id> excludeIds

    ) {

        Map<Id, List<Booking__c>> byCar = new Map<Id,
List<Booking__c>>();

        if (carIds.isEmpty()) return byCar;



        List<Booking__c> existing = [

            SELECT Id, Car__c, Start_Date__c, End_Date__c, Status__c

            FROM Booking__c

            WHERE Car__c IN :carIds

              AND Start_Date__c <= :windowEnd

              AND End_Date__c >= :windowStart

              AND Status__c IN ('Confirmed','Rented')  // adjust values to
your picklist

              AND Id NOT IN :excludeIds

        ];
```

```
        for (Booking__c b : existing) {
            if (!byCar.containsKey(b.Car__c)) {
                byCar.put(b.Car__c, new List<Booking__c>());
            }
            byCar.get(b.Car__c).add(b);
        }
        return byCar;
    }
    public static Map<Integer, List<Booking__c>>
findOverlaps(List<Booking__c> incoming, Boolean isUpdate) {
        Map<Integer, List<Booking__c>> result = new Map<Integer,
List<Booking__c>>();
        if (incoming == null || incoming.isEmpty()) return result;
        Set<Id> carIds = new Set<Id>();
        Date minStart = incoming[0].Start_Date__c;
        Date maxEnd = incoming[0].End_Date__c;
        Set<Id> excludeIds = new Set<Id>();
        for (Integer i = 0; i < incoming.size(); i++) {
            Booking__c b = incoming[i];
            if (b.Car__c != null) carIds.add(b.Car__c);
            if (b.Start_Date__c != null && (minStart == null ||
b.Start_Date__c < minStart)) {
                minStart = b.Start_Date__c;
            }
```

```
        if (b.End_Date__c != null && (maxEnd == null ||
b.End_Date__c > maxEnd)) {

            maxEnd = b.End_Date__c;

        }

        if (isUpdate && b.Id != null) excludeIds.add(b.Id);

        result.put(i, new List<Booking__c>());

    }



    Map<Id, List<Booking__c>> existingByCar =
getExistingBookingsByCar(carIds, minStart, maxEnd, excludeIds);



    for (Integer i = 0; i < incoming.size(); i++) {

        Booking__c newB = incoming[i];

        if (newB.Car__c == null || newB.Start_Date__c == null ||
newB.End_Date__c == null) continue;



        List<Booking__c> candidates =
existingByCar.get(newB.Car__c);

        if (candidates == null) continue;



        for (Booking__c ex : candidates) {

            if (rangesOverlap(newB.Start_Date__c, newB.End_Date__c,
ex.Start_Date__c, ex.End_Date__c)) {

                result.get(i).add(ex);

            }

        }

    }
```

```
        return result;
    }
    public static List<Car__c> getAvailableCars(Date startDate, Date
endDate) {
        return [
            SELECT Id, Name, Status__c
            FROM Car__c
            WHERE Status__c = 'Available'
              AND Id NOT IN (
                SELECT Car__c FROM Booking__c
                WHERE Start_Date__c <= :endDate
                  AND End_Date__c >= :startDate
                  AND Status__c IN ('Confirmed','Rented')
              )
        ];
    }
}
```

3.Trigger Design Pattern:

1. Create BookingTriggerHandler Class:

- Go to Developer Console.
- File → New → Apex Class.
- Name it: BookingTriggerHandler → OK

CODE:

```
public with sharing class BookingTriggerHandler {
```

```
    public static void beforeInsertOrUpdate(List<Booking__c> newList,
    Map<Id, Booking__c> oldMap) {

    Boolean isUpdate = (oldMap != null);

        Map<Integer, List<Booking__c>> conflicts =
        BookingService.findOverlaps(newList, isUpdate);

            for (Integer i : conflicts.keySet()) {

          if (!conflicts.get(i).isEmpty()) {

           newList[i].addError('This booking overlaps with an existing booking
          for the same car.');

           }

        }

    }

}
```

2. Creating Actual booking Trigger:

- In **Developer Console → File → New → Apex Trigger**.
- Name: BookingTrigger
- SObject: Booking__c → **OK**.

CODE:

```
        trigger BookingTrigger on Booking__c (before insert, before update) {

    if (Trigger.isBefore) {

        if (Trigger.isInsert) {

            BookingTriggerHandler.beforeInsertOrUpdate(Trigger.new, null);

        }

        if (Trigger.isUpdate) {
```

```
        BookingTriggerHandler.beforeInsertOrUpdate(Trigger.new,
Trigger.oldMap);

    }

  }

}
```

3. Create a Test Class for Booking Trigger:

1. In **Developer Console → File → New → Apex Class**.
2. Name it: BookingTriggerTest → click **OK**.

CODE:

```
@isTest

public class BookingTriggerTest {

  private static Car__c createCar(String name) {

    Car__c car = new Car__c(Name = name, Status__c = 'Available');

    insert car;

    return car;

  }

  @isTest

  static void testNoOverlapBooking() {

    Car__c car1 = createCar('Car A');

    Booking__c b1 = new Booking__c(

      Name = 'Booking 1',

      Car__c = car1.Id,

      Start_Date__c = Date.today(),

      End_Date__c = Date.today().addDays(2),
```

```
        Status__c = 'Confirmed'
    );
    insert b1;
    Booking__c b2 = new Booking__c(
        Name = 'Booking 2',
        Car__c = car1.Id,
        Start_Date__c = Date.today().addDays(3),
        End_Date__c = Date.today().addDays(5),
        Status__c = 'Confirmed'
    );
    Test.startTest();
    insert b2;
    Test.stopTest();
    System.assertNotEquals(null, b2.Id, 'Booking without overlap
should insert successfully');
  }
  @isTest
  static void testOverlapBooking() {
     Car__c car1 = createCar('Car B');
    Booking__c b1 = new Booking__c(
        Name = 'Booking 1',
        Car__c = car1.Id,
        Start_Date__c = Date.today(),
        End_Date__c = Date.today().addDays(5),
```

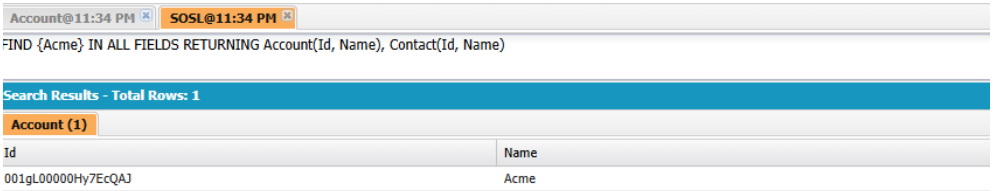
```
            Status__c = 'Confirmed'
        );
        insert b1;
        Booking__c b2 = new Booking__c(
            Name = 'Booking 2',
            Car__c = car1.Id,
            Start_Date__c = Date.today().addDays(2), // Overlaps with b1
            End_Date__c = Date.today().addDays(7),
            Status__c = 'Confirmed'
        );
        Test.startTest();
        try {
            insert b2;
            System.assert(false, 'Expected overlap error but insert
succeeded');
        } catch (DmlException e) {
            System.assert(e.getMessage().contains('overlaps'), 'Expected
overlap error message');
        }
        Test.stopTest();
    }
    @isTest
    static void testUpdateOverlapBooking() {
        Car__c car1 = createCar('Car C');
```

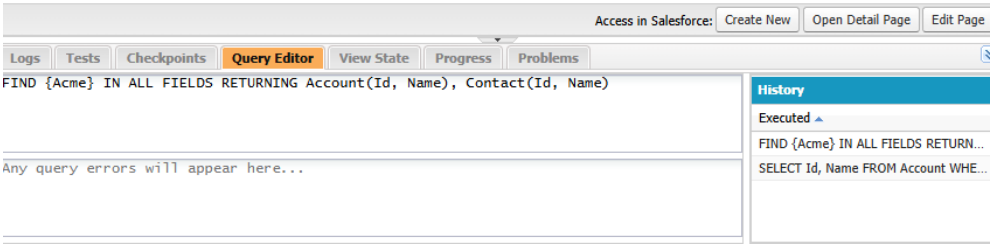
```
Booking__c b1 = new Booking__c(
    Name = 'Booking 1',
    Car__c = car1.Id,
    Start_Date__c = Date.today(),
    End_Date__c = Date.today().addDays(5),
    Status__c = 'Confirmed'
);
insert b1;
Booking__c b2 = new Booking__c(
    Name = 'Booking 2',
    Car__c = car1.Id,
    Start_Date__c = Date.today().addDays(6),
    End_Date__c = Date.today().addDays(8),
    Status__c = 'Confirmed'
);
insert b2;
b2.Start_Date__c = Date.today().addDays(3); // now overlaps
Test.startTest();
try {
    update b2;
    System.assert(false, 'Expected overlap error but update succeeded');
} catch (DmlException e) {
```

```
            System.assert(e.getMessage().contains('overlaps'), 'Expected
overlap error message');

        }

        Test.stopTest();

    }

}
```

4. SOQL & SOSL

- Go to Setup → Developer Console.
- Click Query Editor at the bottom.
- SELECT Id, Name FROM Account WHERE Industry = 'Technology'
- Click **Execute**.
- For SOSL: FIND {Acme} IN ALL FIELDS RETURNING Account(Id, Name), Contact(Id, Name)





5. Collections: List, Set, Map

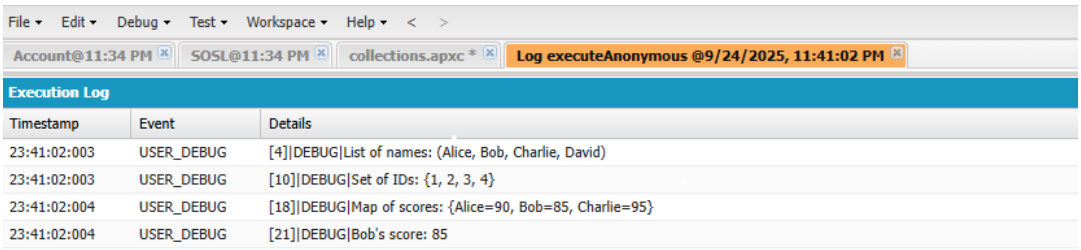
Purpose: Store multiple records efficiently.

Steps in Apex:

```
List<String> names = new List<String>{'Alice', 'Bob'};
```

Set<Integer> ids = new Set<Integer>{1, 2, 3};

Map<String, Account> accountsMap = new Map<String, Account>([SELECT Id, Name FROM Account]).



6. Control Statements

Purpose: Add logic (if/else, loops).

Steps in Apex:

for(Account acc : [SELECT Id, Name FROM Account LIMIT 10]){

if(acc.Name == 'Acme') {

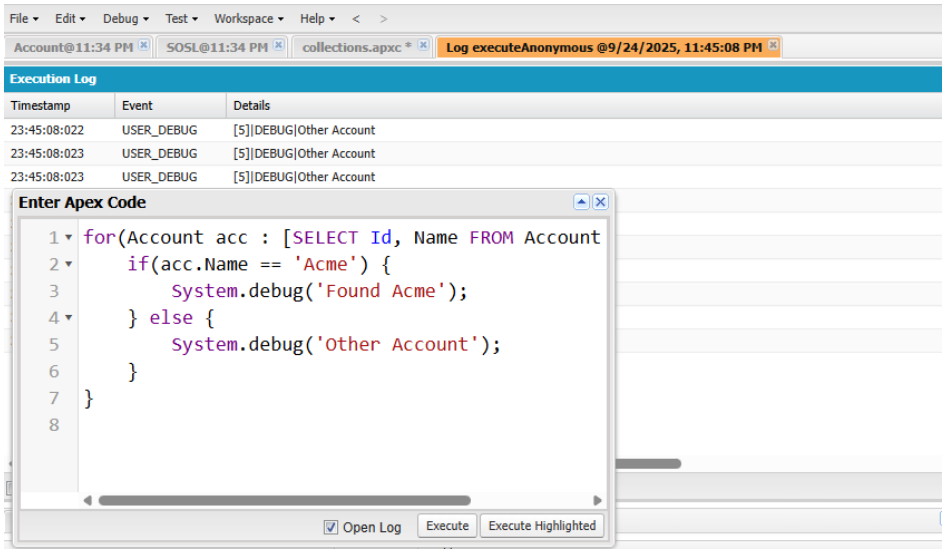
System.debug('Found Acme');

} else {

System.debug('Other Account');

}

}



7. Batch Apex:

Go to Developer Console → File → New → Apex Class.

CODE:

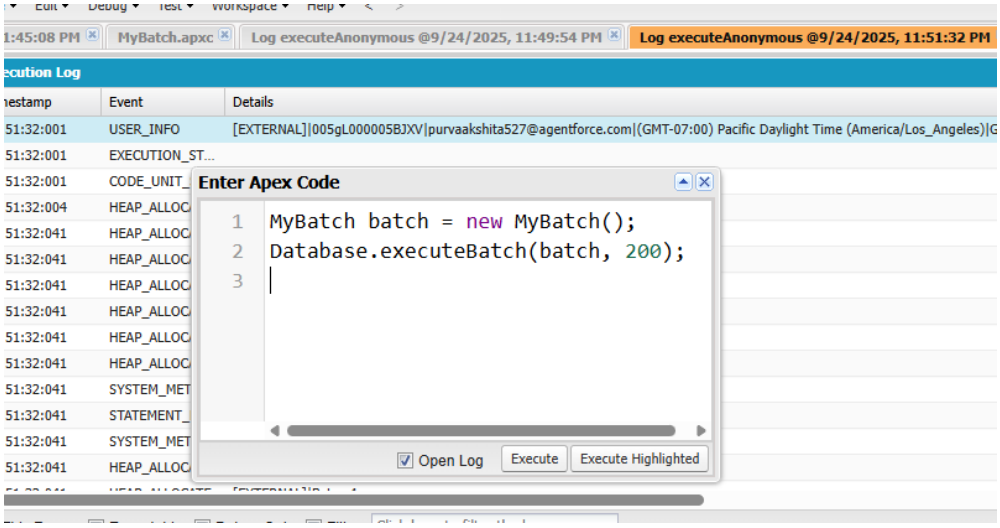
```
global class MyBatch implements Database.Batchable<SObject> {
global Database.QueryLocator start(Database.BatchableContext BC){
 return Database.getQueryLocator('SELECT Id, Name FROM Account');
}
global void execute(Database.BatchableContext BC, List<Account> scope){
for(Account a : scope){
        a.Name = a.Name + ' Updated';
}
update scope;
}
 global void finish(Database.BatchableContext BC){
 System.debug('Batch Finished');
```

}

}

To run:

- MyBatch batch = new MyBatch();
- Database.executeBatch(batch, 200);



8. Queueable Apex:

public class MyQueueable implements Queueable {

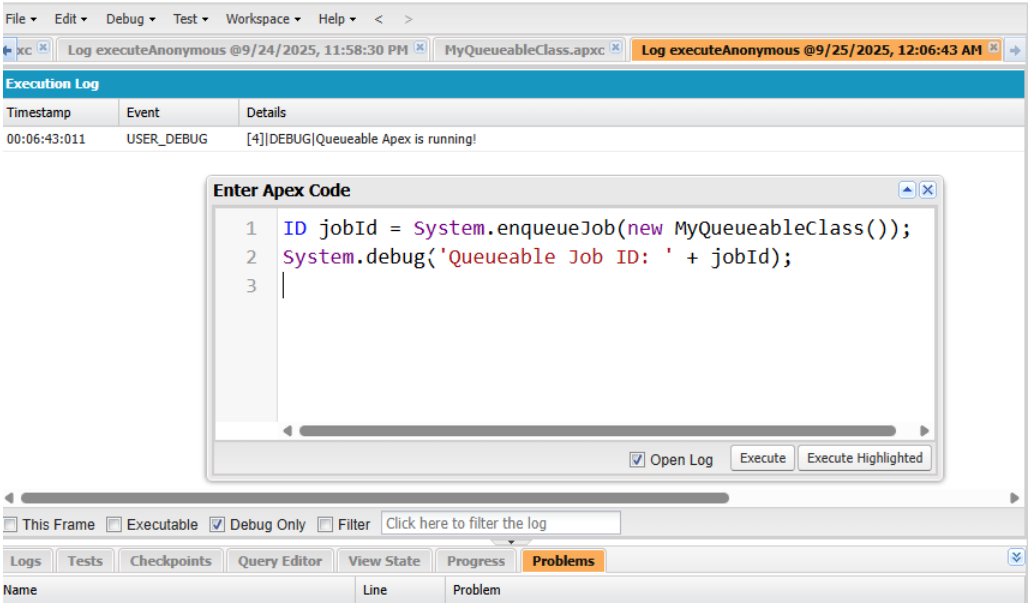
public void execute(QueueableContext context) {

System.debug('Running Queueable job');

}

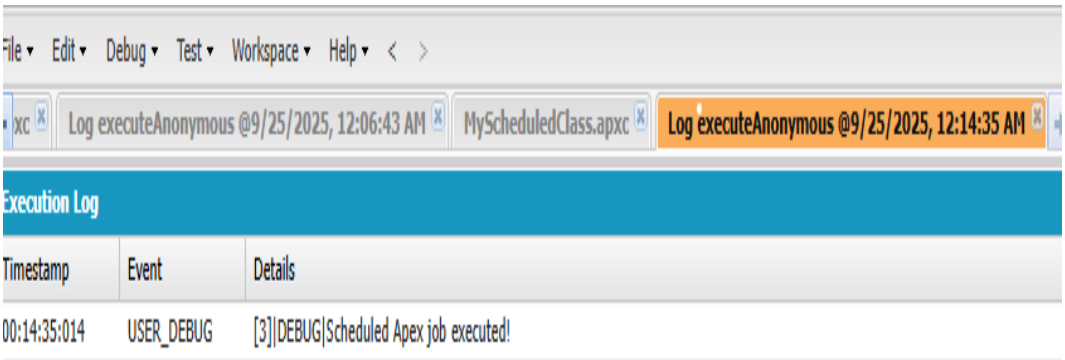
}

Run: ID jobId = System.enqueueJob(new MyQueueable());



9. Scheduled Apex:

```
global class MyScheduler implements Schedulable {

global void execute(SchedulableContext sc){

System.debug('Scheduled job executed');

}

}
```



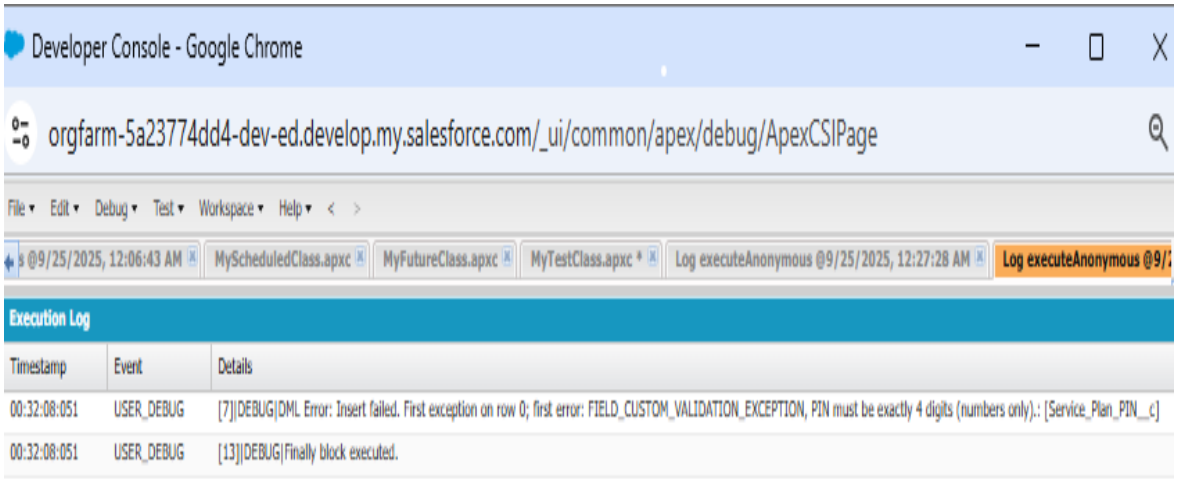
10. Future Methods:

```
public class MyFutureClass {

@future

public static void doAsyncWork(List<Id> accountIds){

        List<Account> accs = [SELECT Id, Name FROM Account WHERE Id
IN :accountIds];

}

}
```

RUN: MyFutureClass.doAsyncWork(newList<Id>{'001xx000003DGbEAAW'});

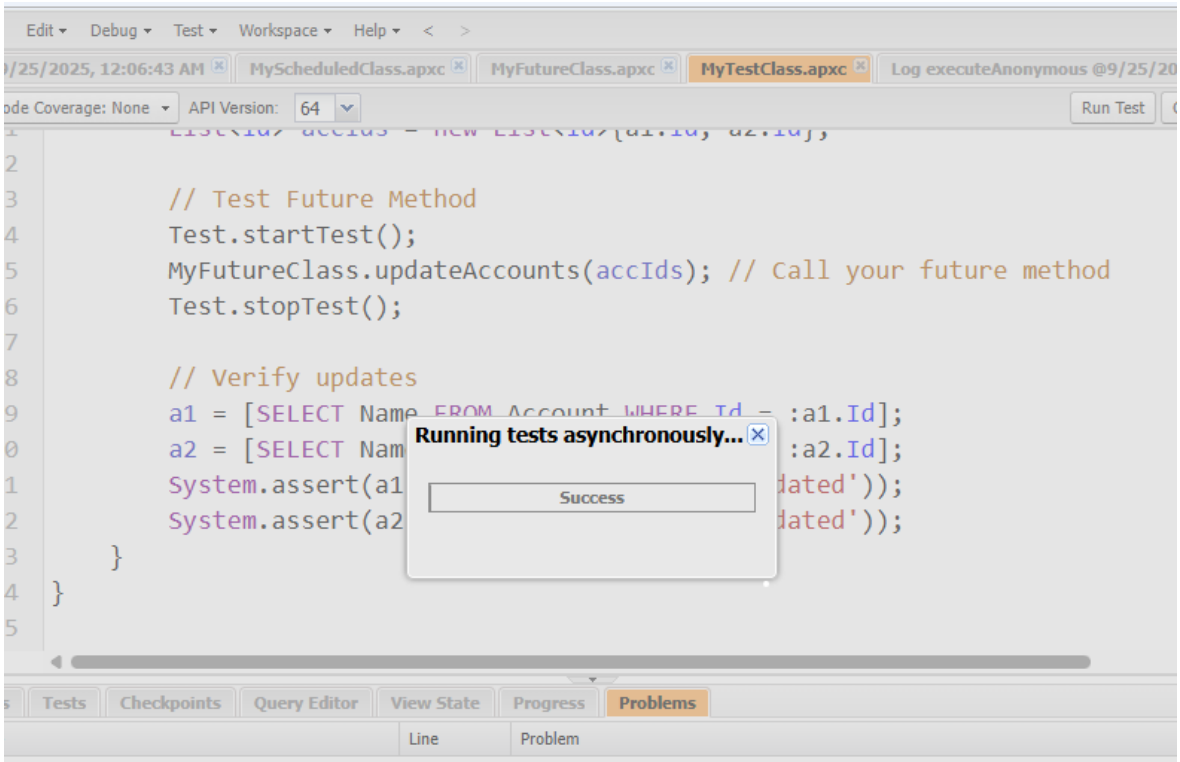
11. Exception Handling

```
try {

Account a = new Account (Name='Test');

 insert a;

} catch(DmlException e) {

System.debug('Error: ' + e.getMessage());

}
```



12. Test Classes:

```
@IsTest
private class MyTestClass {
    @IsTest static void testBatch() {
        Account a = new Account(Name='Test');
         insert a;
         Test.startTest();
        MyBatch batch = new MyBatch();
        Database.executeBatch(batch);
        Test.stopTest();
        System.assert([SELECT COUNT() FROM Account] > 0);
        }
    }
```



13. Asynchronous Processing:

- Always use Test.startTest() and Test.stopTest() when testing async code.