Phase 5: Apex Programming (Developer)

Project:Visitor CheckIn sytem for offices

Implement Apex Programming concepts in the Visitor Management System to handle advanced business logic, automation, and asynchronous processes.

1. Classes & Objects

Use Case:

Created a VisitorHandler Apex class to manage visitor check-in and check-out logic. This class contains methods to validate visitor data, update statuses, and send notifications automatically.

Program:

```
public class Visitor {
  // Method to create a new Visitor record
  public static Visitor__c createVisitor(
    String name,
    String email,
    String phone,
    String purpose,
    String gender,
    Date age
  ){
    Visitor__c v = new Visitor__c();
    v.Visitor_Name__c = name;
    v.Visitor_Email__c = email;
    v.Visitor_Phone_Number__c = phone;
    v.Purpose_of_visit__c = purpose; // Picklist: Meeting, Interview, Delivery, Other
    v.Status__c = 'Checked In';
    v.Gender__c = gender; // Picklist: Male, Female, Other
    v.Age__c = age;
    v.Visitor_Check_In_Time__c = System.now();
    insert v; // Save record in Salesforce
    System.debug('Visitor created with Id: ' + v.Id);
    return v;
```

```
    }
}
```

Test Apex in Execute Anonymous

Code:

```
Visitor__c newVisitor = Visitor.createVisitor(
    'Harathi Aswarthagari',
    'harathi@example.com',
    '9876543210',
    'Meeting',
    'Female',
    Date.newInstance(2003,9,24)
);
System.debug('Visitor ID: ' + newVisitor.Id);
```

Explanation:

- Each method in the class is modular and reusable.
- Used object-oriented principles to organize business logic efficiently.

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

Use Case:
A trigger runs automatically when a record is inserted, updated, deleted, or undeleted.

In our case, we want to run logic automatically after a Visitor is checked in

Created a **before insert** trigger on Visitor__c to validate email and phone number before saving records.
Created an **after update** trigger to send email alerts when the Status__c field changes to Checked Out.

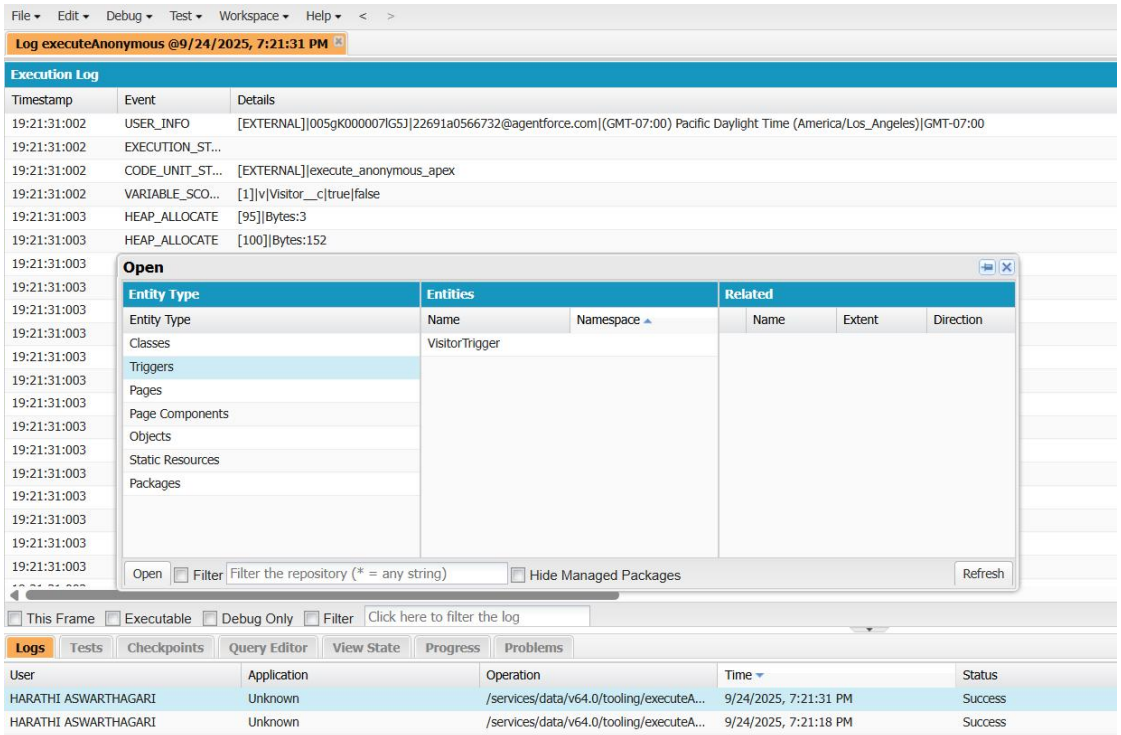
Code:

```
trigger VisitorTrigger on Visitor__c (after insert, after update) {
    // Loop through all Visitor records that triggered this event
    for(Visitor__c v : Trigger.new){
        // Check if the status is "Checked In"
        if(v.Status__c == 'Checked In'){
            // This is where you can run your logic
```

```
        System.debug('Visitor Checked In: ' + v.Visitor_Name__c);
        // Example: send email or enqueue queueable job here
      }
    }
}
```

Testing the trigger:

```
Visitor__c v = Visitor.createVisitor(
   'Jahnavi Gudapati',
   'jahnavi@example.com',
   '9876543211',
   'Interview',
   'Female',
   Date.newInstance(2003,8,15)
);
```



Explanation:

- Before Insert triggers are used for data validation.
- After Update triggers are used for actions that require the record ID or sending emails.

3. Trigger Design Pattern

Implemented **One Trigger per Object** pattern to keep all triggers on Visitor__c centralized.
Trigger logic delegates to a handler class (VisitorHandler) to separate logic from trigger context.

Explanation:

- Reduces code duplication.
- Makes future maintenance easier and ensures bulkification.

Code:

```
trigger VisitorHandlerTrigger on Visitor__c (before insert, before update) {
  // Example: Prevent duplicate emails
  if(Trigger.isBefore && Trigger.isInsert){
    Set<String> emails = new Set<String>();
    for(Visitor__c v : Trigger.new){
      if(v.Visitor_Email__c != null) emails.add(v.Visitor_Email__c);
    }

    if(!emails.isEmpty()){
      Map<String, Visitor__c> existing = new Map<String, Visitor__c>();
      for(Visitor__c e : [
        SELECT Visitor_Email__c FROM Visitor__c WHERE Visitor_Email__c IN :emails
      ]){
        existing.put(e.Visitor_Email__c, e);
      }

      for(Visitor__c v : Trigger.new){
        if(existing.containsKey(v.Visitor_Email__c)){
          v.addError('Duplicate email found: ' + v.Visitor_Email__c);
        }
      }
    }
  } }
```

4. SOQL & SOSL

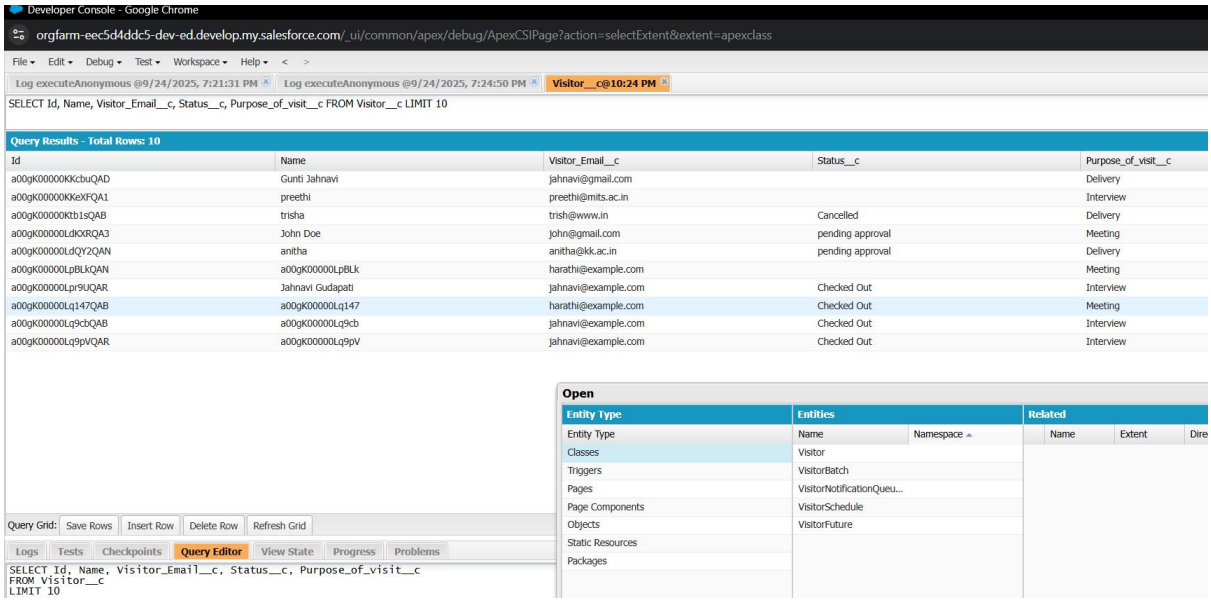
Used SOQL queries to fetch visitor records with Status__c = Checked In to generate daily reports.
Used SOSL to search visitors by name or email across multiple fields.

Code:

SELECT Id, Name, Visitor_Email__c, Status__c, Purpose_of_visit__c

FROM Visitor__c

LIMIT 10;



Explanation:

- SOQL used for retrieving records from a single object.
- SOSL used for global search across multiple objects.

5. Collections: List, Set, Map

Using List : Creating or inserting multiple visitor records

Code:

// Example: insert multiple Visitor__c records using a List

List<Visitor__c> visitors = new List<Visitor__c>();

visitors.add(new Visitor__c(

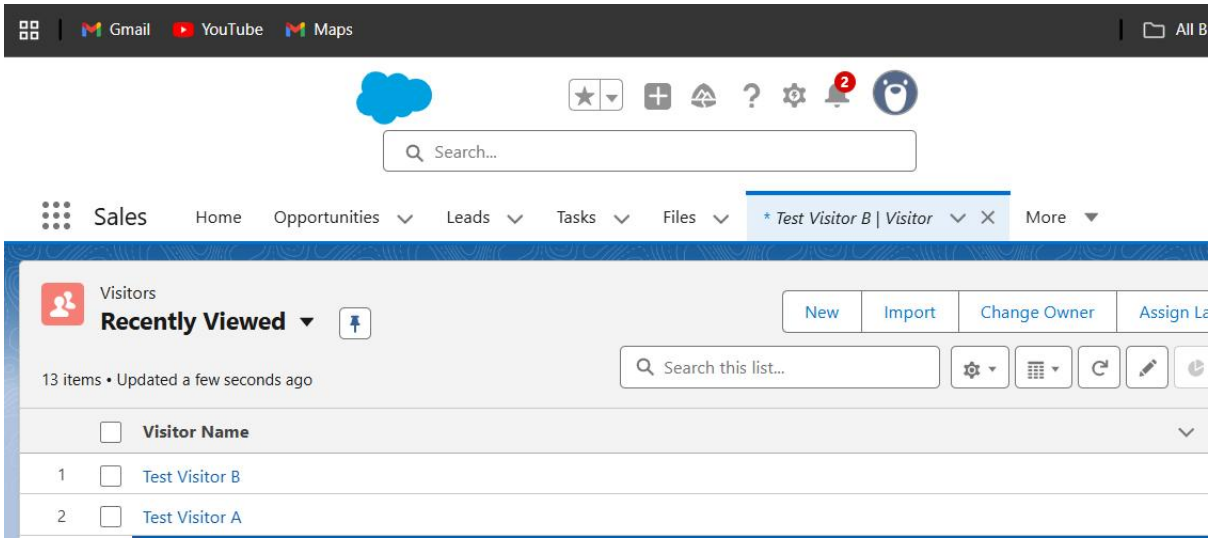
Visitor_Name__c = 'Test Visitor A',

Visitor_Email__c = 'testA@example.com',

Visitor_Phone_Number__c = '9000000001',

```
    Purpose_of_visit__c = 'Meeting',
    Gender__c = 'Female',
    Status__c = 'Checked In',
    Visitor_Check_In_Time__c = System.now(),
    Age__c = Date.newInstance(2000,1,1)
));

visitors.add(new Visitor__c(
    Visitor_Name__c = 'Test Visitor B',
    Visitor_Email__c = 'testB@example.com',
    Visitor_Phone_Number__c = '9000000002',
    Purpose_of_visit__c = 'Interview',
    Gender__c = 'Female',
    Status__c = 'Checked In',
    Visitor_Check_In_Time__c = System.now(),
    Age__c = Date.newInstance(1998,5,10)
));
insert visitors;
System.debug('Inserted visitors count: ' + visitors.size());
for(Visitor__c v : visitors) System.debug('Inserted Id: ' + v.Id + ' Name: ' + v.Visitor_Name__c);
```

output:



Using set:*Used Set<String> to remove duplicate visitor emails before sending notifications.*

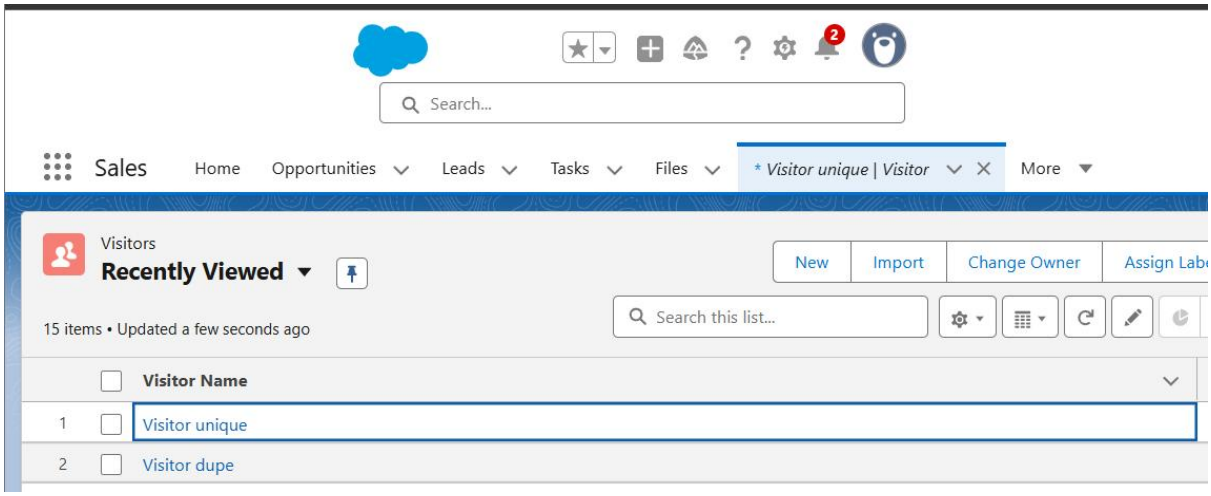
use a Set to de-duplicate emails

code:

```
// Example: use Set to check duplicates in anonymous script

Set<String> newEmails = new
Set<String>{'dupe@example.com','unique@example.com','dupe@example.com'};

System.debug('Emails to insert (unique): ' + newEmails);


List<Visitor__c> toInsert = new List<Visitor__c>();

for(String e : newEmails){

   toInsert.add(new Visitor__c(

      Visitor_Name__c = 'Visitor ' + e.substringBefore('@'),

      Visitor_Email__c = e,

      Visitor_Phone_Number__c = '9000000' +
String.valueOf(Math.mod(Math.abs(Crypto.getRandomInteger()),10000)),

      Purpose_of_visit__c = 'Other',

      Gender__c = 'Female',

      Status__c = 'Checked In',

      Visitor_Check_In_Time__c = System.now(),

      Age__c = Date.newInstance(2000,1,1)

   ));

}


// Before inserting, check if any of these emails already exist in Salesforce

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

for(Visitor__c v : toInsert) emailsToCheck.add(v.Visitor_Email__c);


List<Visitor__c> existing = [SELECT Id, Visitor_Email__c FROM Visitor__c WHERE Visitor_Email__c
IN :emailsToCheck];

if(!existing.isEmpty()){

   for(Visitor__c e : existing) System.debug('Existing email found: ' + e.Visitor_Email__c);
```

```
    // Option: remove existing emails from toInsert or stop insert
}

// Now insert only the new ones (remove existing)
Set<String> existingEmails = new Set<String>();
for(Visitor__c ex : existing) existingEmails.add(ex.Visitor_Email__c);

List<Visitor__c> finalInsert = new List<Visitor__c>();
for(Visitor__c v : toInsert){
    if(!existingEmails.contains(v.Visitor_Email__c)) finalInsert.add(v);
}

insert finalInsert;
System.debug('Inserted new visitors: ' + finalInsert.size());
```

Output:

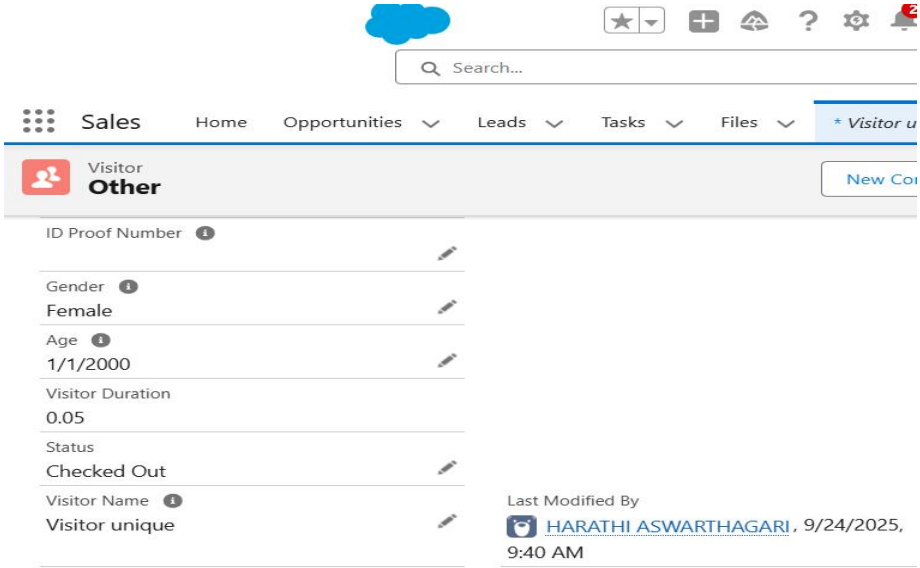


Using map: Used a Map<Id, Visitor__c> to efficiently update multiple visitor records in bulk.

```
// Example: query records and build a map
List<Visitor__c> visitors = [SELECT Id, Visitor_Name__c, Status__c FROM Visitor__c WHERE Status__c = 'Checked In' LIMIT 50];

Map<Id, Visitor__c> visitorMap = new Map<Id, Visitor__c>(visitors);
System.debug('Visitor map size: ' + visitorMap.size());
```

```
// Example: update a field for all visitors in the map
List<Visitor__c> toUpdate = new List<Visitor__c>();
for(Id vid : visitorMap.keySet()){
    Visitor__c v = visitorMap.get(vid);
    // put a test flag or change status (example: set to Checked Out for demo)
    v.Status__c = 'Checked Out';
    v.Visitor_Check_Out_Time__c = System.now();
    toUpdate.add(v);
}

if(!toUpdate.isEmpty()){
    update toUpdate;
    System.debug('Updated visitors count: ' + toUpdate.size());
}
```

Output:



Explanation:

- Collections improve performance and reduce governor limits.

6. Control Statements

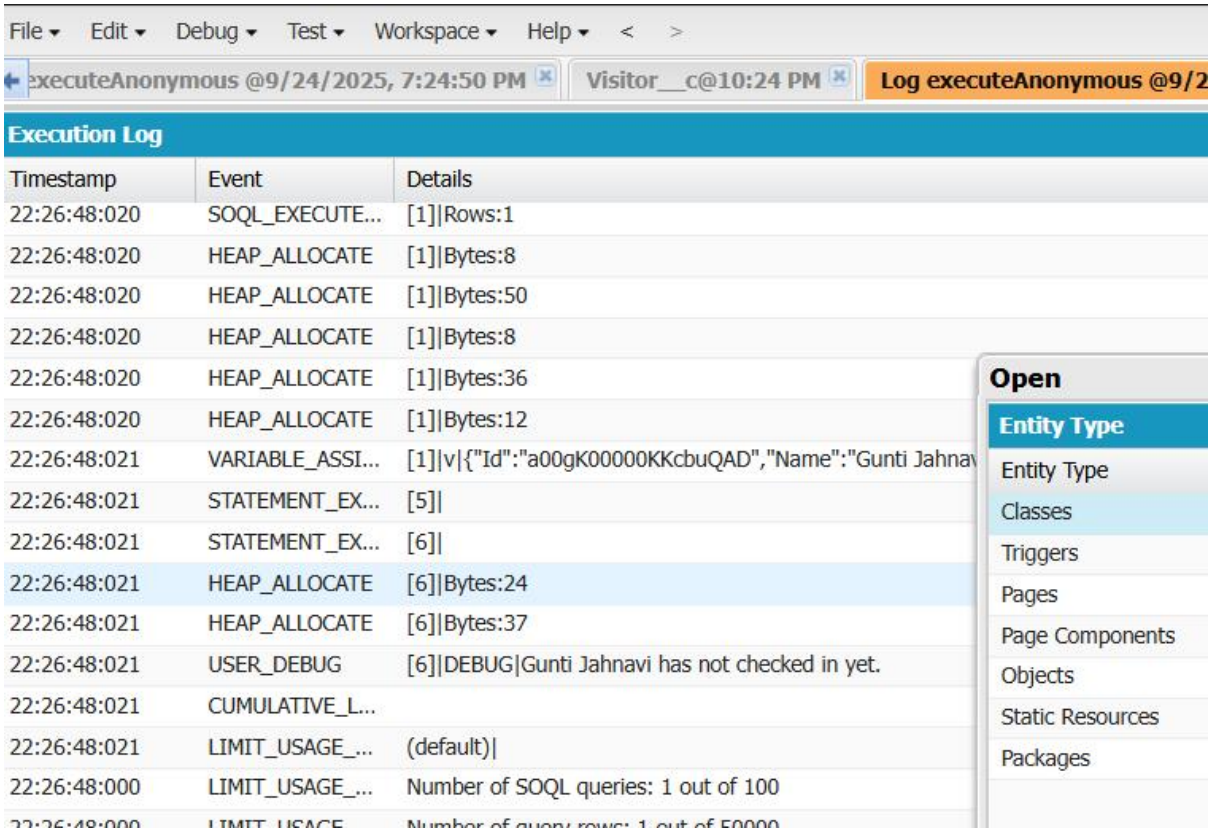
Used if-else, for, and while statements to handle conditional checks and loops in visitor record processing.

Code:

If-else

```
Visitor__c v = [SELECT Id, Name, Status__c FROM Visitor__c LIMIT 1];

if (v.Status__c == 'Checked In') {

   System.debug(v.Name + ' is currently inside.');

} else {

   System.debug(v.Name + ' has not checked in yet.');

}
```

Output:

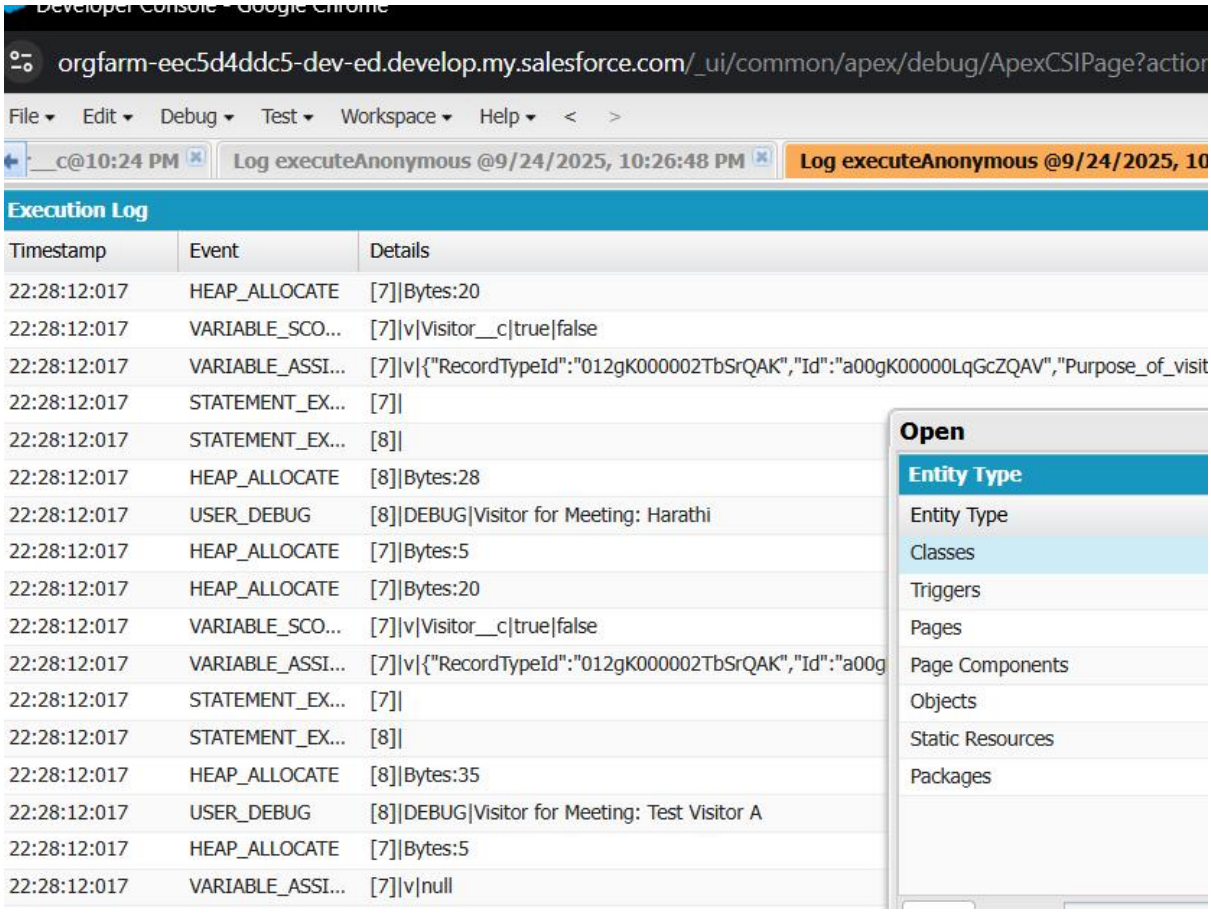


For loop:

```
List<Visitor__c> visitors = [

   SELECT Name, Purpose_of_visit__c

   FROM Visitor__c

   WHERE Purpose_of_visit__c = 'Meeting'
```

```
];
for (Visitor__c v : visitors) {
    System.debug('Visitor for Meeting: ' + v.Name);
}
```

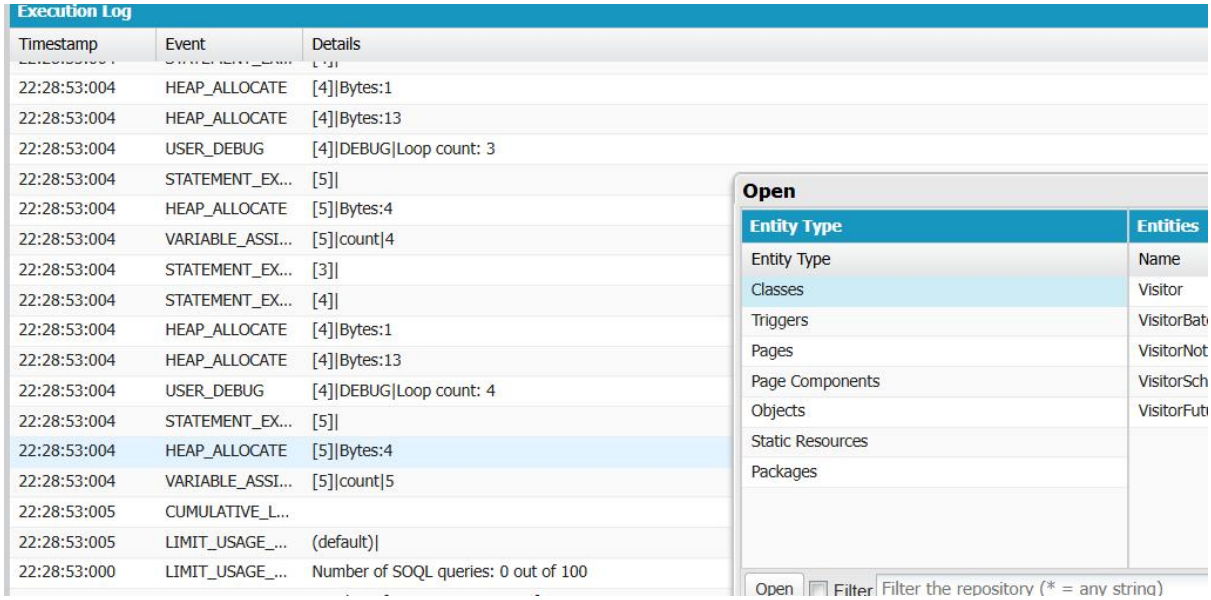
Output:



While Loop:

```
Integer count = 0;

while (count < 5) {
    System.debug('Loop count: ' + count);
    count++;
}
```

Output:



Explanation:

- Ensures proper decision-making and iteration in Apex logic.

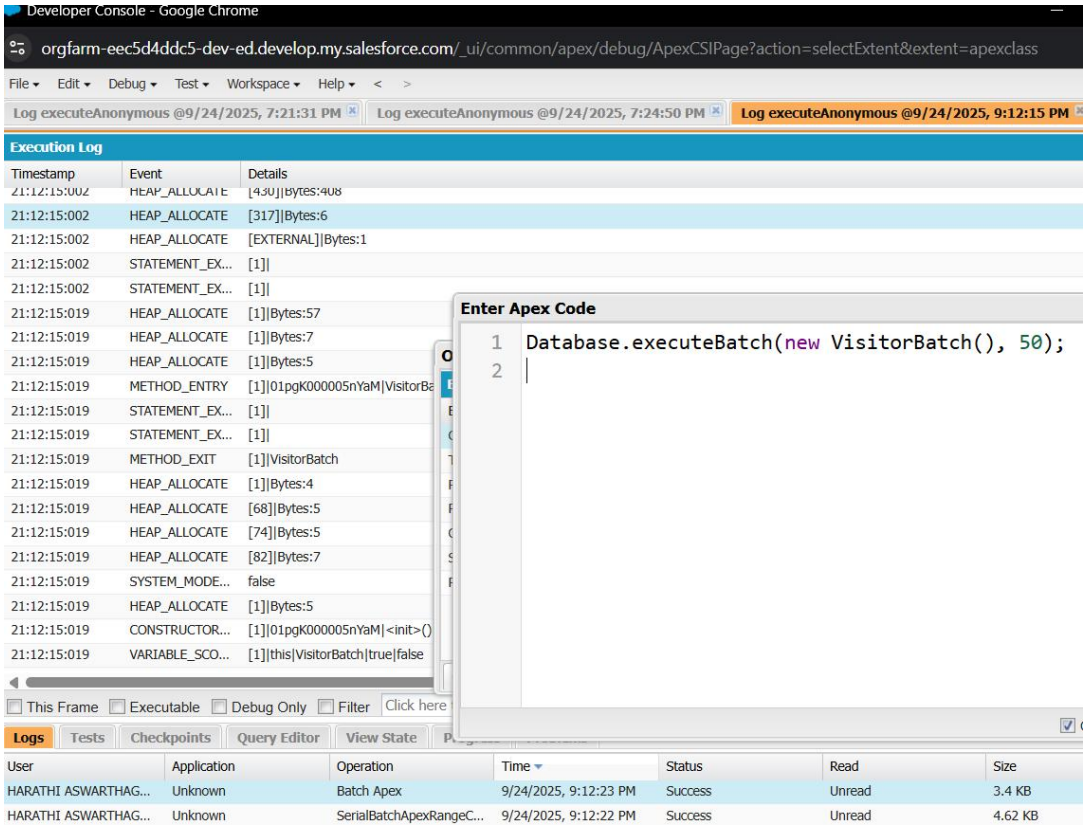
7. Batch Apex

Purpose: Process large datasets in chunks to avoid governor limits

Code:

```
global class VisitorBatch implements Database.Batchable<sObject> {
  global Database.QueryLocator start(Database.BatchableContext BC){
    // Select all checked-in visitors
    return Database.getQueryLocator(
      'SELECT Id, Visitor_Name__c, Status__c, Visitor_Email__c FROM Visitor__c WHERE Status__c = \'Checked In\''
    );
  }
  global void execute(Database.BatchableContext BC, List<Visitor__c> scope){
    for(Visitor__c v : scope){
      System.debug('Batch processing: ' + v.Visitor_Name__c);
    }
```

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



Explanation:

- Batch Apex handles large datasets asynchronously without hitting governor limits.

8. Queueable Apex

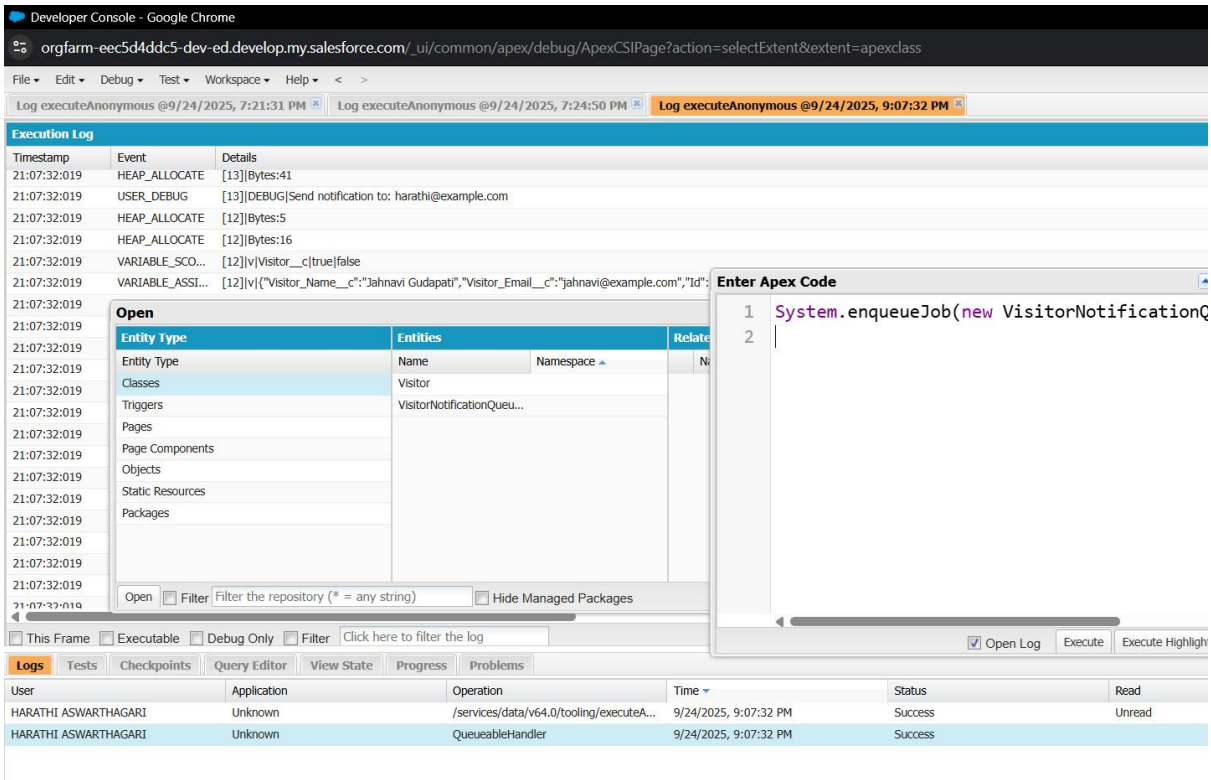
Use Case:
*Useful for sending **emails, updating records, or calling external systems** without slowing down the user's action.*

Queueable Apex is used to send visitor check-out notifications asynchronously after the record is updated.

Code:

```
public class VisitorNotificationQueueable implements Queueable {


    public void execute(QueueableContext context){
```

```
        // Query all visitors who are checked in
        List<Visitor__c> checkedInVisitors = [
            SELECT Id, Visitor_Name__c, Visitor_Email__c
            FROM Visitor__c
            WHERE Status__c = 'Checked In'
        ];
        for(Visitor__c v : checkedInVisitors){
            System.debug('Send notification to: ' + v.Visitor_Email__c);
        }
    }
}
```



Explanation:

- Queueable Apex allows chaining and asynchronous processing for complex tasks.

9. Scheduled Apex

Purpose: Run jobs automatically on a schedule (e.g., daily notification for checked-in visitors).

Scheduled Apex is used to run a nightly job that archives old visitor records older than 1 year.

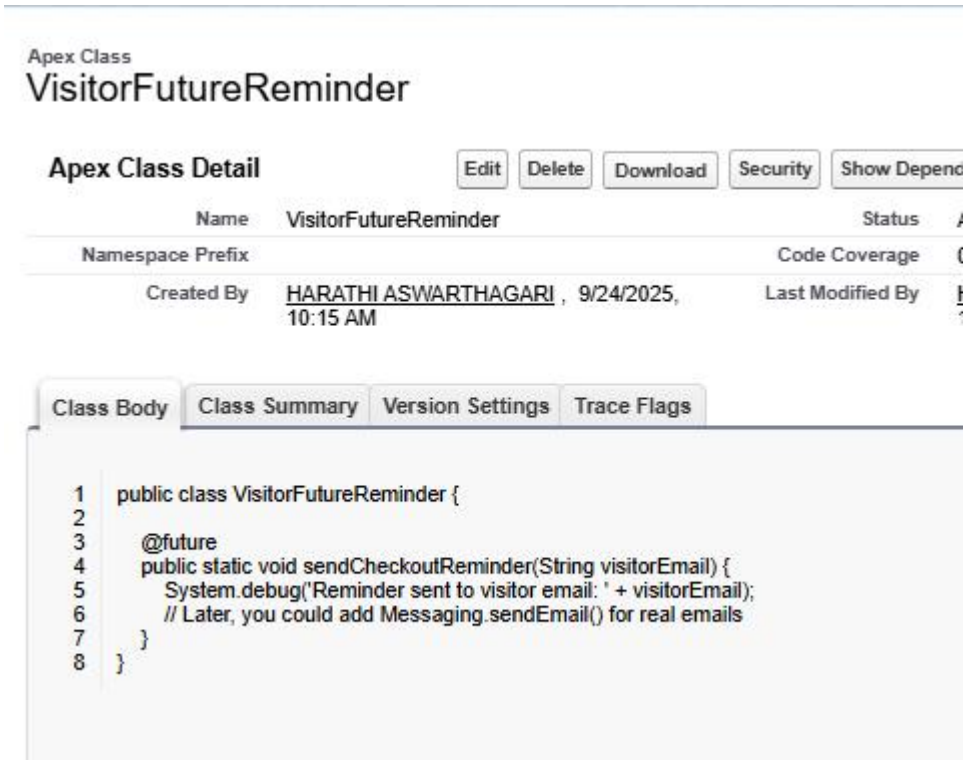
Code:

```
global class VisitorSchedule implements Schedulable {
   global void execute(SchedulableContext sc){
      System.enqueueJob(new VisitorNotificationQueueable());
   }
}
```

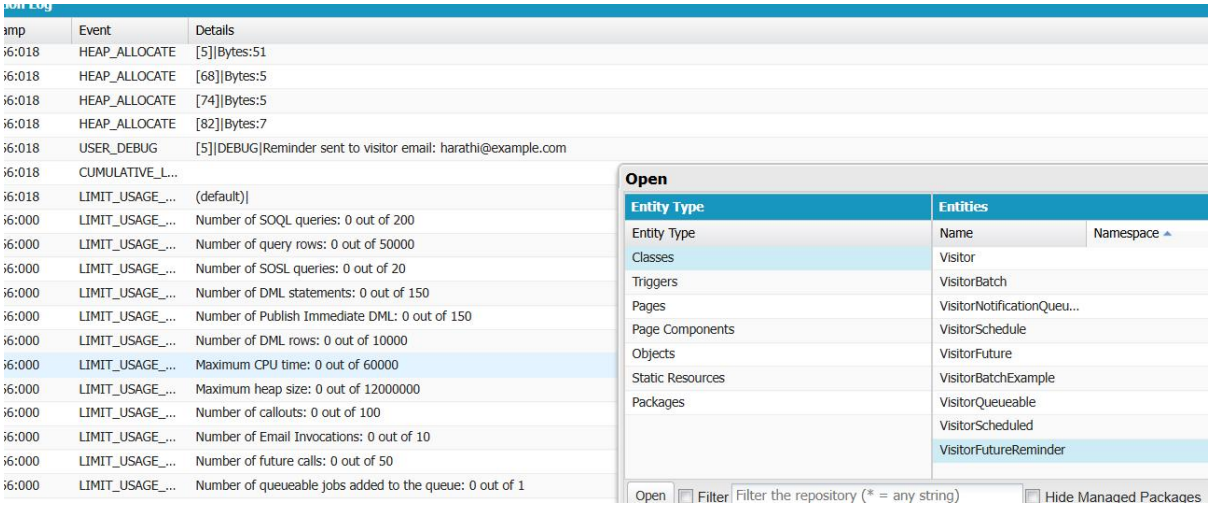
Explanation:

- Scheduled Apex automates recurring tasks without manual intervention.

10. Future Methods



Used @future method to send external API notifications when visitors check out, so it does not block the main transaction.



amp	Event	Details
56:018	HEAP_ALLOCATE	[5]|Bytes:51
56:018	HEAP_ALLOCATE	[68]|Bytes:5
56:018	HEAP_ALLOCATE	[74]|Bytes:5
56:018	HEAP_ALLOCATE	[82]|Bytes:7
56:018	USER_DEBUG	[5]|DEBUG|Reminder sent to visitor email: harathi@example.com
56:018	CUMULATIVE_L...	
56:018	LIMIT_USAGE_...	(default)|
56:000	LIMIT_USAGE_...	Number of SOQL queries: 0 out of 200
56:000	LIMIT_USAGE_...	Number of query rows: 0 out of 50000
56:000	LIMIT_USAGE_...	Number of SOSL queries: 0 out of 20
56:000	LIMIT_USAGE_...	Number of DML statements: 0 out of 150
56:000	LIMIT_USAGE_...	Number of Publish Immediate DML: 0 out of 150
56:000	LIMIT_USAGE_...	Number of DML rows: 0 out of 10000
56:000	LIMIT_USAGE_...	Maximum CPU time: 0 out of 60000
56:000	LIMIT_USAGE_...	Maximum heap size: 0 out of 12000000
56:000	LIMIT_USAGE_...	Number of callouts: 0 out of 100
56:000	LIMIT_USAGE_...	Number of Email Invocations: 0 out of 10
56:000	LIMIT_USAGE_...	Number of future calls: 0 out of 50
56:000	LIMIT_USAGE_...	Number of queueable jobs added to the queue: 0 out of 1

Explanation:

- Future methods allow callouts or processing asynchronously to avoid governor limits.

11. External Objects

Integrated visitor-related external data (like host employee information) via External Objects to display in Salesforce without storing in native objects.

Explanation:

- External objects let you access external data directly from Salesforce without duplication.