

Phase 5: Apex Programming (Developer)

Project: Visitor CheckIn system 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(

    'Jahnvi Gudapati',

    'jahnvi@example.com',

    '9876543211',

    'Interview',

    'Female',

    Date.newInstance(2003,8,15)

);

```

The screenshot displays the Salesforce IDE interface. At the top, the menu bar includes File, Edit, Debug, Test, Workspace, and Help. Below the menu, a log entry is visible: "Log executeAnonymous @9/24/2025, 7:21:31 PM". The main area shows the "Execution Log" with a table of events. The table has columns for Timestamp, Event, and Details. The events include USER_INFO, EXECUTION_ST..., CODE_UNIT_ST..., VARIABLE_SCO..., HEAP_ALLOCATE, and HEAP_ALLOCATE. An "Open" dialog box is overlaid on the log, showing a list of entity types (Entity Type, Classes, Triggers, Pages, Page Components, Objects, Static Resources, Packages) and a table of entities (Name, Namespace, Name, Extent, Direction). The "Triggers" entity type is selected. Below the dialog, there are checkboxes for "This Frame", "Executable", "Debug Only", and "Filter". At the bottom, there are tabs for "Logs", "Tests", "Checkpoints", "Query Editor", "View State", "Progress", and "Problems". The "Logs" tab is active, showing a table of log entries with columns for User, Application, Operation, Time, and Status.

| Timestamp | Event | Details |
|--------------|-----------------|--|
| 19:21:31:002 | USER_INFO | [EXTERNAL][005gK000007G5J]22691a0566732@agentforce.com[(GMT-07:00) Pacific Daylight Time (America/Los_Angeles)]GMT-07:00 |
| 19:21:31:002 | EXECUTION_ST... | |
| 19:21:31:002 | CODE_UNIT_ST... | [EXTERNAL][execute_anonymous_apex |
| 19:21:31:002 | VARIABLE_SCO... | [1] v Visitor__c true false |
| 19:21:31:003 | HEAP_ALLOCATE | [95] Bytes:3 |
| 19:21:31:003 | HEAP_ALLOCATE | [100] Bytes:152 |

| Entity Type | Entities | Related |
|-------------|----------------|-----------|
| Entity Type | Name | Namespace |
| Classes | VisitorTrigger | |

| User | Application | Operation | Time | Status |
|----------------------|-------------|--|-----------------------|---------|
| HARATHI ASWARTHAGARI | Unknown | /services/data/v64.0/tooling/executeA... | 9/24/2025, 7:21:31 PM | Success |
| HARATHI ASWARTHAGARI | Unknown | /services/data/v64.0/tooling/executeA... | 9/24/2025, 7:21:18 PM | Success |

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;
```

The screenshot shows the Salesforce Developer Console with a SOQL query executed. The query is: `SELECT Id, Name, Visitor_Email__c, Status__c, Purpose_of_visit__c FROM Visitor__c LIMIT 10`. The results table shows 10 records with columns: Id, Name, Visitor_Email__c, Status__c, and Purpose_of_visit__c. The records include visitors like Gunti Jahnavi, preethi, trisha, John Doe, anitha, and others, with statuses ranging from 'Cancelled' to 'Checked Out' and purposes like 'Delivery', 'Interview', and 'Meeting'.

| Id | Name | Visitor_Email__c | Status__c | Purpose_of_visit__c |
|--------------------|------------------|---------------------|------------------|---------------------|
| a00gk00000KcbuQAD | Gunti Jahnavi | jahnavi@gmail.com | | Delivery |
| a00gk00000K6eXQA1 | preethi | preethi@mits.ac.in | | Interview |
| a00gk00000Ktb1sQAB | trisha | trish@www.in | Cancelled | Delivery |
| a00gk00000LdKQQA3 | John Doe | john@gmail.com | pending approval | Meeting |
| a00gk00000LdY2QAN | anitha | anitha@kk.ac.in | pending approval | Delivery |
| a00gk00000LpBLUQAN | Jahnavi Gudapati | harathi@example.com | | Meeting |
| a00gk00000LpSUQAR | Jahnavi Gudapati | jahnavi@example.com | Checked Out | Interview |
| a00gk00000Lq147QAB | a00gk00000Lq147 | harathi@example.com | Checked Out | Meeting |
| a00gk00000Lq3cbQAB | a00gk00000Lq3cb | jahnavi@example.com | Checked Out | Interview |
| a00gk00000Lq5pvQAR | a00gk00000Lq5pv | jahnavi@example.com | Checked Out | Interview |

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 = '90000000001',
```

```

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 = '90000000002',
    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)
));

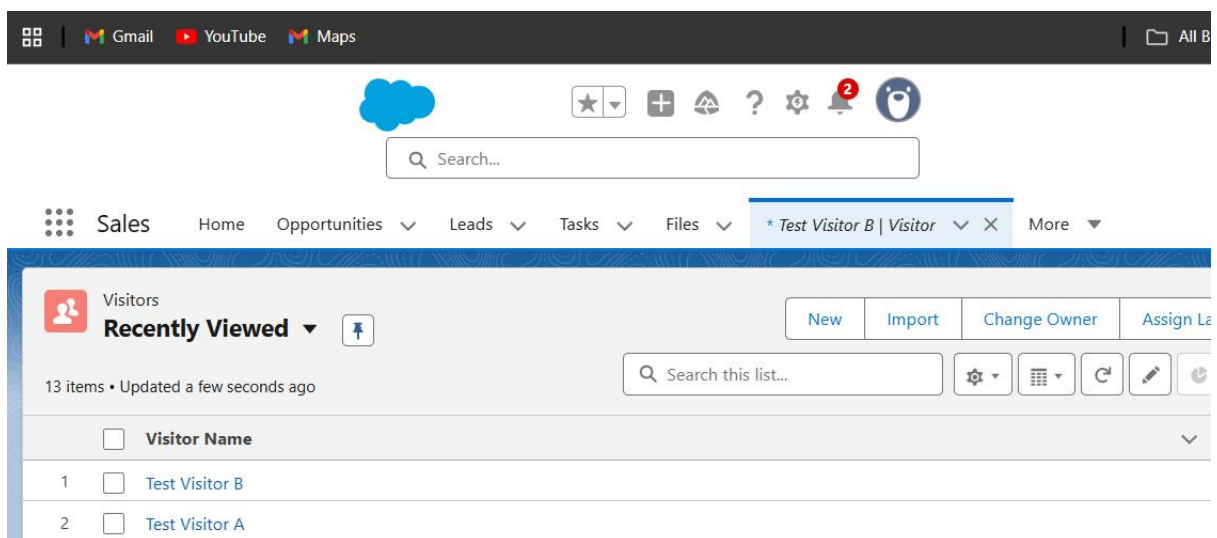
insert visitors;

System.debug('Inserted visitors count: ' + visitors.size());

for(Visitor__c v : visitors) System.debug('Inserted Id: ' + v.Id + ' Name: ' + v.Visitor_Name__c);

```

output:



The screenshot shows the Salesforce user interface. At the top, there's a navigation bar with links to Gmail, YouTube, and Maps. Below that is a search bar. The main navigation menu includes 'Sales', 'Home', 'Opportunities', 'Leads', 'Tasks', 'Files', and a tab for '* Test Visitor B | Visitor'. The 'Visitors' section is active, showing a 'Recently Viewed' list with 13 items. The list contains two entries: 'Test Visitor B' and 'Test Visitor A'. The interface includes standard Salesforce controls like 'New', 'Import', 'Change Owner', and 'Assign Lead' buttons, as well as a search bar for the list and various view and action icons.

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 = '90000000' +
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);

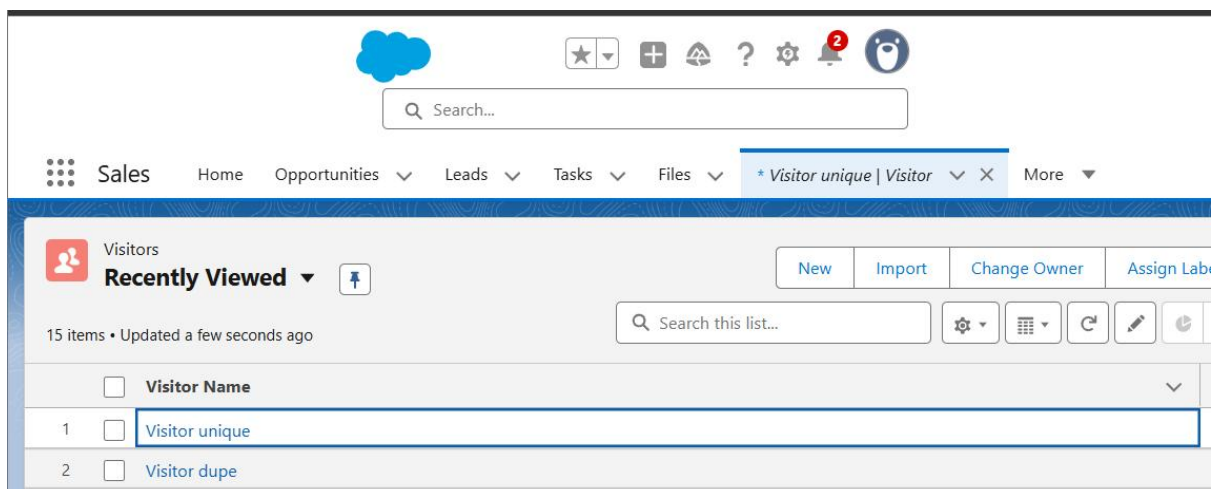
List<Visitor__c> finalInsert = new List<Visitor__c>();
for(Visitor__c v : toInsert){
    if(!existingEmails.contains(v.Visitor_Email__c)) finalInsert.add(v);
}

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:

The screenshot displays a Salesforce record page for a Visitor. The page features a navigation bar with tabs for Sales, Home, Opportunities, Leads, Tasks, and Files. Below the navigation bar, there is a search bar and a list of fields for the Visitor record. The fields include ID Proof Number, Gender (set to Female), Age (set to 1/1/2000), Visitor Duration (set to 0.05), Status (set to Checked Out), Visitor Name, and Visitor unique. The Status field is highlighted in blue. The page also shows a 'Last Modified By' section indicating the record was modified by HARATHI ASWARTHAGARI on 9/24/2025 at 9:40 AM.

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') {
```

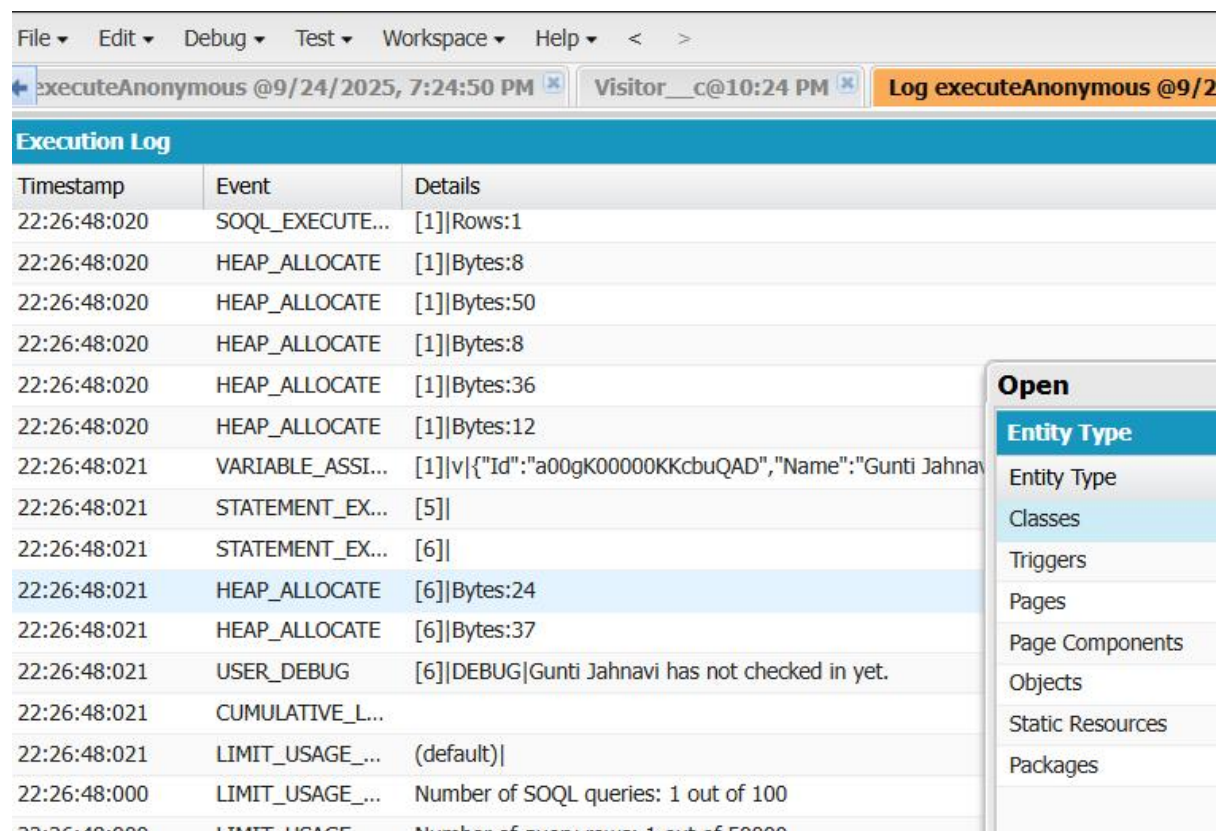
```
    System.debug(v.Name + ' is currently inside.');
```

```
} else {
```

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

```
}
```

Output:



The screenshot shows the Salesforce IDE interface. At the top, there are tabs for 'executeAnonymous @9/24/2025, 7:24:50 PM', 'Visitor__c@10:24 PM', and 'Log executeAnonymous @9/24/2025, 7:24:50 PM'. Below the tabs is the 'Execution Log' table, which contains the following data:

| Timestamp | Event | Details |
|--------------|------------------|--|
| 22:26:48:020 | SOQL_EXECUTE... | [1] Rows:1 |
| 22:26:48:020 | HEAP_ALLOCATE | [1] Bytes:8 |
| 22:26:48:020 | HEAP_ALLOCATE | [1] Bytes:50 |
| 22:26:48:020 | HEAP_ALLOCATE | [1] Bytes:8 |
| 22:26:48:020 | HEAP_ALLOCATE | [1] Bytes:36 |
| 22:26:48:020 | HEAP_ALLOCATE | [1] Bytes:12 |
| 22:26:48:021 | VARIABLE_ASSI... | [1] v {"Id":"a00gK00000KKcbuQAD","Name":"Gunti Jahnavi"} |
| 22:26:48:021 | STATEMENT_EX... | [5] |
| 22:26:48:021 | STATEMENT_EX... | [6] |
| 22:26:48:021 | HEAP_ALLOCATE | [6] Bytes:24 |
| 22:26:48:021 | HEAP_ALLOCATE | [6] Bytes:37 |
| 22:26:48:021 | USER_DEBUG | [6] DEBUG Gunti Jahnavi has not checked in yet. |
| 22:26:48:021 | CUMULATIVE_L... | |
| 22:26:48:021 | LIMIT_USAGE_... | (default) |
| 22:26:48:000 | LIMIT_USAGE_... | Number of SOQL queries: 1 out of 100 |
| 22:26:48:000 | LIMIT_USAGE_... | Number of queries: 1 out of 50000 |

On the right side of the screen, there is a sidebar menu with the following items:

- Open
- Entity Type
- Entity Type
- Classes
- Triggers
- Pages
- Page Components
- Objects
- Static Resources
- Packages

For loop:

```
List<Visitor__c> visitors = [
```

```
    SELECT Name, Purpose_of_visit__c
```

```
    FROM Visitor__c
```

```
    WHERE Purpose_of_visit__c = 'Meeting']
```

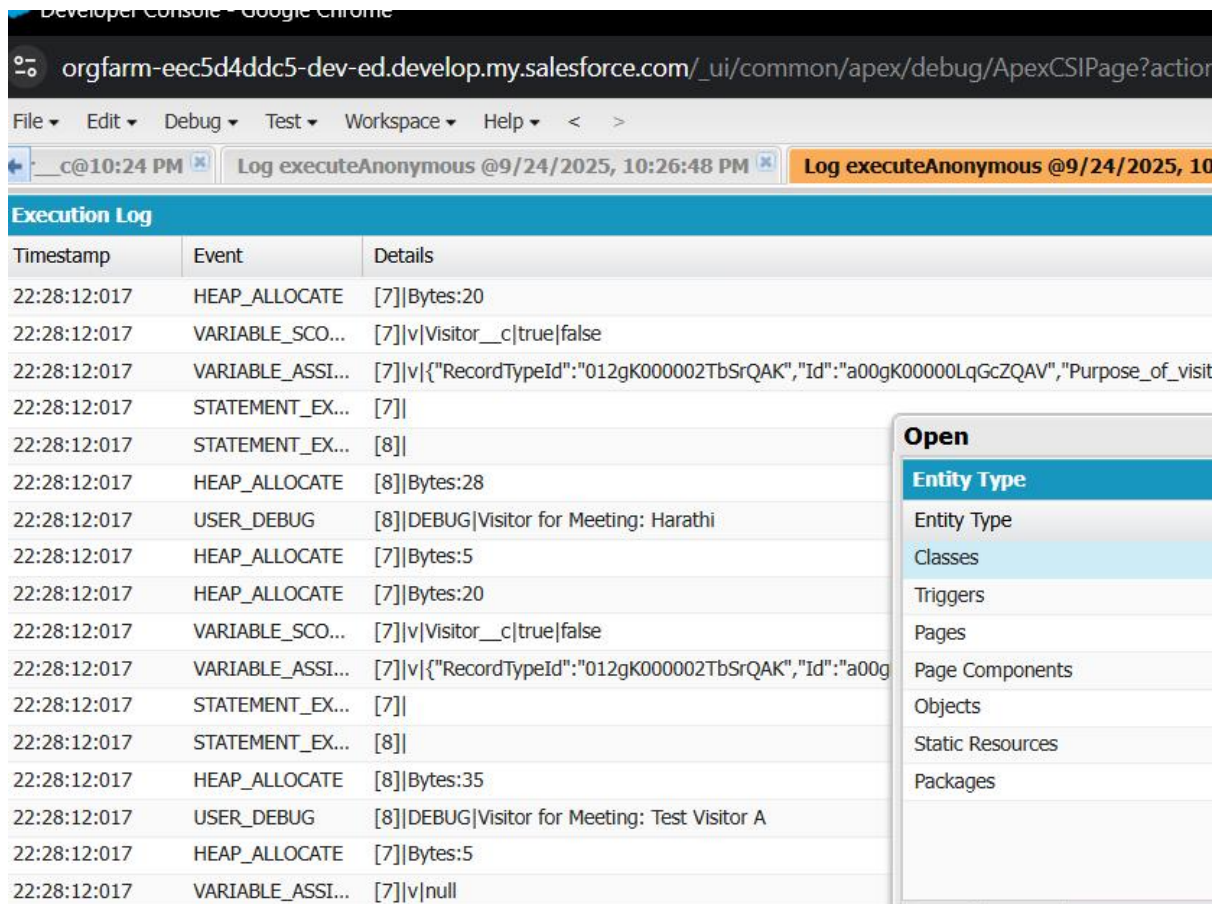
```
];

for (Visitor__c v : visitors) {

    System.debug('Visitor for Meeting: ' + v.Name);

}
```

Output:



| Execution Log | | |
|---------------|----------------------|--|
| Timestamp | Event | Details |
| 22:28:12:017 | HEAP_ALLOCATE | [7] Bytes:20 |
| 22:28:12:017 | VARIABLE_SCOPE... | [7] v Visitor__c true false |
| 22:28:12:017 | VARIABLE_ASSIGN... | [7] v {"RecordTypeId":"012gK000002TbSrQAK","Id":"a00gK00000LqGcZQAV","Purpose_of_visit |
| 22:28:12:017 | STATEMENT_EXECUTE... | [7] |
| 22:28:12:017 | STATEMENT_EXECUTE... | [8] |
| 22:28:12:017 | HEAP_ALLOCATE | [8] Bytes:28 |
| 22:28:12:017 | USER_DEBUG | [8] DEBUG Visitor for Meeting: Harathi |
| 22:28:12:017 | HEAP_ALLOCATE | [7] Bytes:5 |
| 22:28:12:017 | HEAP_ALLOCATE | [7] Bytes:20 |
| 22:28:12:017 | VARIABLE_SCOPE... | [7] v Visitor__c true false |
| 22:28:12:017 | VARIABLE_ASSIGN... | [7] v {"RecordTypeId":"012gK000002TbSrQAK","Id":"a00g |
| 22:28:12:017 | STATEMENT_EXECUTE... | [7] |
| 22:28:12:017 | STATEMENT_EXECUTE... | [8] |
| 22:28:12:017 | HEAP_ALLOCATE | [8] Bytes:35 |
| 22:28:12:017 | USER_DEBUG | [8] DEBUG Visitor for Meeting: Test Visitor A |
| 22:28:12:017 | HEAP_ALLOCATE | [7] Bytes:5 |
| 22:28:12:017 | VARIABLE_ASSIGN... | [7] v null |

Open

- Entity Type
- Entity Type
- Classes
- Triggers
- Pages
- Page Components
- Objects
- Static Resources
- Packages

While Loop:

Integer count = 0;

```
while (count < 5) {

    System.debug('Loop count: ' + count);

    count++;

}
```

Output:

| Execution Log | | |
|---------------|------------------|--------------------------------------|
| Timestamp | Event | Details |
| 22:28:53:004 | HEAP_ALLOCATE | [4] Bytes:1 |
| 22:28:53:004 | HEAP_ALLOCATE | [4] Bytes:13 |
| 22:28:53:004 | USER_DEBUG | [4] DEBUG Loop count: 3 |
| 22:28:53:004 | STATEMENT_EX... | [5] |
| 22:28:53:004 | HEAP_ALLOCATE | [5] Bytes:4 |
| 22:28:53:004 | VARIABLE_ASSI... | [5] count 4 |
| 22:28:53:004 | STATEMENT_EX... | [3] |
| 22:28:53:004 | STATEMENT_EX... | [4] |
| 22:28:53:004 | HEAP_ALLOCATE | [4] Bytes:1 |
| 22:28:53:004 | HEAP_ALLOCATE | [4] Bytes:13 |
| 22:28:53:004 | USER_DEBUG | [4] DEBUG Loop count: 4 |
| 22:28:53:004 | STATEMENT_EX... | [5] |
| 22:28:53:004 | HEAP_ALLOCATE | [5] Bytes:4 |
| 22:28:53:004 | VARIABLE_ASSI... | [5] count 5 |
| 22:28:53:005 | CUMULATIVE_L... | |
| 22:28:53:005 | LIMIT_USAGE_... | (default) |
| 22:28:53:000 | LIMIT_USAGE_... | Number of SOQL queries: 0 out of 100 |

Open

| Entity Type | Entities |
|------------------|------------|
| Entity Type | Name |
| Classes | Visitor |
| Triggers | VisitorBat |
| Pages | VisitorNot |
| Page Components | VisitorSch |
| Objects | VisitorFut |
| Static Resources | |
| Packages | |

☐ Filter
 Filter the repository (* = any string)

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.');
```

```

}

}
```

The screenshot shows the Salesforce Developer Console interface. The top part displays the 'Execution Log' with a table of events. Below this, a code editor window titled 'Enter Apex Code' is open, showing the following code:

```

1 Database.executeBatch(new VisitorBatch(), 50);
2
```

At the bottom, there is a table showing the execution results:

| User | Application | Operation | Time | Status | Read | Size |
|----------------------|-------------|--------------------------|-----------------------|---------|--------|---------|
| HARATHI ASWARTHAG... | Unknown | Batch Apex | 9/24/2025, 9:12:23 PM | Success | Unread | 3.4 KB |
| HARATHI ASWARTHAG... | Unknown | SerialBatchApexRangeC... | 9/24/2025, 9:12:22 PM | Success | Unread | 4.62 KB |

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);

}

}
```

The screenshot shows the Salesforce Developer Console interface. The top bar indicates the URL: `orgfarm-ee5d4ddc5-dev-ed.develop.my.salesforce.com/_ui/common/apex/debug/ApexCSIPage?action=selectExtent&extent=apexclass`. The main area displays an 'Execution Log' with the following entries:

| Timestamp | Event | Details |
|--------------|---------------------|---|
| 21:07:32:019 | HEAP_ALLOCATE | [13]]Bytes:41 |
| 21:07:32:019 | USER_DEBUG | [13]]DEBUG Send notification to: harathi@example.com |
| 21:07:32:019 | HEAP_ALLOCATE | [12]]Bytes:5 |
| 21:07:32:019 | HEAP_ALLOCATE | [12]]Bytes:16 |
| 21:07:32:019 | VARIABLE_SCOPE | [12]]v Visitor__c true false |
| 21:07:32:019 | VARIABLE_ASSIGNMENT | [12]]v {"Visitor_Name__c":"Jahnnavi Gudapati","Visitor_Email__c":"jahnnavi@example.com","Id": |

An 'Enter Apex Code' dialog is open, showing the following code:

```
1 System.enqueueJob(new VisitorNotificationQ
2 |
```

Below the log, there is a table with the following data:

| User | Application | Operation | Time | Status | Read |
|----------------------|-------------|--|-----------------------|---------|--------|
| HARATHI ASWARTHAGARI | Unknown | /services/data/v64.0/tooling/executeA... | 9/24/2025, 9:07:32 PM | Success | Unread |
| HARATHI ASWARTHAGARI | Unknown | QueueableHandler | 9/24/2025, 9:07:32 PM | Success | |

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

The screenshot displays the Salesforce Developer Console interface for an Apex Class named **VisitorFutureReminder**. The class is public and uses the `@future` annotation for the `sendCheckoutReminder` method. The code is as follows:

```

1 public class VisitorFutureReminder {
2
3     @future
4     public static void sendCheckoutReminder(String visitorEmail) {
5         System.debug('Reminder sent to visitor email: ' + visitorEmail);
6         // Later, you could add Messaging.sendEmail() for real emails
7     }
8 }

```

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

| Time | Event | Details |
|--------|------------------------|---|
| i6:018 | HEAP_ALLOCATE | [5] Bytes:51 |
| i6:018 | HEAP_ALLOCATE | [68] Bytes:5 |
| i6:018 | HEAP_ALLOCATE | [74] Bytes:5 |
| i6:018 | HEAP_ALLOCATE | [82] Bytes:7 |
| i6:018 | USER_DEBUG | [5] DEBUG Reminder sent to visitor email: harathi@example.com |
| i6:018 | CUMULATIVE_LIMIT_USAGE | |
| i6:018 | LIMIT_USAGE__c | (default) |
| i6:000 | LIMIT_USAGE__c | Number of SOQL queries: 0 out of 200 |
| i6:000 | LIMIT_USAGE__c | Number of query rows: 0 out of 50000 |
| i6:000 | LIMIT_USAGE__c | Number of SOSL queries: 0 out of 20 |
| i6:000 | LIMIT_USAGE__c | Number of DML statements: 0 out of 150 |
| i6:000 | LIMIT_USAGE__c | Number of Publish Immediate DML: 0 out of 150 |
| i6:000 | LIMIT_USAGE__c | Number of DML rows: 0 out of 10000 |
| i6:000 | LIMIT_USAGE__c | Maximum CPU time: 0 out of 60000 |
| i6:000 | LIMIT_USAGE__c | Maximum heap size: 0 out of 12000000 |
| i6:000 | LIMIT_USAGE__c | Number of callouts: 0 out of 100 |
| i6:000 | LIMIT_USAGE__c | Number of Email Invocations: 0 out of 10 |
| i6:000 | LIMIT_USAGE__c | Number of future calls: 0 out of 50 |
| i6:000 | LIMIT_USAGE__c | Number of queueable jobs added to the queue: 0 out of 1 |

Open

| Entity Type | Entities |
|------------------|----------------------------|
| Entity Type | Name Namespace ▲ |
| Classes | Visitor |
| Triggers | VisitorBatch |
| Pages | VisitorNotificationQueu... |
| Page Components | VisitorSchedule |
| Objects | VisitorFuture |
| Static Resources | VisitorBatchExample |
| Packages | VisitorQueueable |
| | VisitorScheduled |
| | VisitorFutureReminder |

☐ Filter
 Filter the repository (* = any string)
 ☐ Hide Managed Packages

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.