Common Trigger Errors and Solutions

1. Recursive Trigger Execution When a trigger causes itself to run repeatedly, creating an infinite loop.

Solution: Use a static variable to prevent the trigger from running multiple times.

apex
```
public class TriggerHelper {
    public static Boolean isExecuting = false;
}

trigger PreventRecursiveTrigger on ObjectName (before insert, before update) {
    if (TriggerHelper.isExecuting) return;

    TriggerHelper.isExecuting = true;
    try {
        // Trigger logic
    } finally {
        TriggerHelper.isExecuting = false;
    }
}
```

2. Too Many SOQL Queries: Limit Exception This error occurs when the trigger exceeds the maximum number of SOQL queries per transaction.

Solution: Use collections like lists or maps to bulkify your code and perform SOQL queries outside of loops.

apex
```
trigger BulkSOQL on ObjectName (before insert, before update) {
    Set<Id> recordIds = new Set<Id>();
    for (ObjectName obj : Trigger.new) {
        recordIds.add(obj.Id);
    }

    List<ObjectName> relatedRecords = [SELECT Id, Field FROM ObjectName WHERE Id IN
:recordIds];
    // Process related records
}
```

3. Too Many DML Statements: Limit Exception This error occurs when the trigger exceeds the maximum number of DML statements per transaction.

Solution: Group DML operations into a single statement outside of loops.

```
apex
trigger BulkDML on ObjectName (after insert, after update) {
    List<ObjectName> recordsToUpdate = new List<ObjectName>();
    for (ObjectName obj : Trigger.new) {
        // Modify records
        recordsToUpdate.add(obj);
    }
    update recordsToUpdate;
}
```

4. Mixed DML Operation Error This error occurs when performing DML operations on both setup and non-setup objects in the same transaction.

Solution: Use asynchronous methods like @future or Queueable Apex to separate DML operations.

```
apex
@future
public static void updateUser(User userToUpdate) {
    update userToUpdate;
}
```

5. Null Pointer Exception This error occurs when the trigger tries to access a null value.

Solution: Add null checks to ensure that the code does not attempt to operate on null values.

```
apex
trigger NullPointerCheck on ObjectName (before insert, before update) {
    for (ObjectName obj : Trigger.new) {
        if (obj.Field__c != null) {
            // Process field
        }
    }
}
```

6. Trigger Validation Errors Errors occur when validation rules prevent the trigger from saving records.

Solution: Make sure the trigger logic does not violate validation rules, or temporarily disable validation rules during trigger execution.

```
apex
trigger ValidateRecords on ObjectName (before insert, before update) {
    for (ObjectName obj : Trigger.new) {
        if (/* your validation condition */) {
            obj.addError('Validation error message.');
        }
    }
}
```

7. Missing Required Fields This error occurs when required fields are not provided in the trigger logic.

Solution: Ensure all required fields are populated before performing DML operations.

```
apex
trigger CheckRequiredFields on ObjectName (before insert, before update) {
    for (ObjectName obj : Trigger.new) {
        if (String.isBlank(obj.RequiredField__c)) {
            obj.addError('RequiredField__c cannot be blank.');
        }
    }
}
```

8. System.LimitException: Apex CPU Time Limit Exceeded This error occurs when the trigger exceeds the maximum CPU time limit for execution.

Solution: Optimize the trigger logic to reduce processing time and avoid inefficient code.

```
apex
trigger OptimizeCPUTime on ObjectName (before insert, before update) {
    // Use efficient data structures and algorithms
    Map<Id, ObjectName> objectMap = new Map<Id, ObjectName>();
    for (ObjectName obj : Trigger.new) {
        objectMap.put(obj.Id, obj);
    }
    // Process objects in bulk
}
```

9. Query Selector Limit Exceeded This error occurs when the number of query rows returned exceeds the governor limit.

Solution: Use `LIMIT` clauses in SOQL queries and filter data effectively.

```
apex
trigger LimitQueryRows on ObjectName (after insert) {
    List<ObjectName> records = [SELECT Id, Name FROM ObjectName LIMIT 100];
    // Process limited records
}
```

10. Uncommitted Work Pending Error This error occurs when a DML operation is performed after a callout without using `@future`.

Solution: Use `@future` or `Queueable` Apex to separate DML operations from callouts.

```
apex
@future
public static void futureMethod(Id recordId) {
    // DML operations
    update [SELECT Id FROM ObjectName WHERE Id = :recordId];
}
```

11. Too Many Child Relationships in a Query This error occurs when the number of child relationships queried exceeds the limit.

Solution: Reduce the number of child relationships in a single query or use separate queries.

```
apex
trigger ReduceChildRelationships on ParentObject (after insert) {
    List<ChildObject> children = [SELECT Id, ParentId FROM ChildObject WHERE ParentId IN
:Trigger.newMap.keySet()];
    // Process child records separately
}
```

12. Mixed DML Operation with Non-SObject Types This error occurs when DML operations are mixed with non-SObject types like UserRole.

Solution: Use `System.runAs()` to perform mixed DML operations within a test context.

```
apex
trigger MixedDML on User (after insert) {
    if (Test.isRunningTest()) {
        System.runAs(new User(Id = UserInfo.getUserId())) {
            // DML operations with UserRole
```

```
        }
    }
}
```

13. Apex Heap Size Limit Exceeded This error occurs when the trigger exceeds the maximum heap size.

Solution: Optimize data structures and avoid storing large amounts of data in memory.

```
apex
trigger OptimizeHeapSize on ObjectName (before insert, before update) {
    List<ObjectName> largeData = new List<ObjectName>();
    for (ObjectName obj : Trigger.new) {
        if (obj.Condition__c) {
            largeData.add(obj);
        }
    }
    // Process largeData
}
```

14. Too Many Future Method Invocations This error occurs when the number of future method invocations exceeds the governor limit.

Solution: Batch operations and reduce the need for multiple future method calls.

```
apex
@future
public static void futureMethod(Set<Id> recordIds) {
    List<ObjectName> records = [SELECT Id FROM ObjectName WHERE Id IN :recordIds];
    update records;
}
```

15. Invalid Cross-Reference Id This error occurs when a cross-reference ID does not match any record in the system.

Solution: Ensure cross-reference IDs are valid and exist in the system before performing operations.

```
apex
trigger ValidateCrossReferenceId on ObjectName (before insert) {
    Set<Id> referenceIds = new Set<Id>();
    for (ObjectName obj : Trigger.new) {
        referenceIds.add(obj.ReferenceId__c);
```

```
    }

    List<ObjectName> validReferences = [SELECT Id FROM ObjectName WHERE Id IN
:referenceIds];
    Set<Id> validIds = new Set<Id>();
    for (ObjectName ref : validReferences) {
        validIds.add(ref.Id);
    }

    for (ObjectName obj : Trigger.new) {
        if (!validIds.contains(obj.ReferenceId__c)) {
            obj.addError('Invalid cross-reference ID.');
        }
    }
}
```

16. Invalid Field for Insert Update This error occurs when a field cannot be inserted or updated due to field-level security or permissions.

Solution: Check field-level security and permissions before performing DML operations.

```
apex
trigger CheckFieldPermissions on ObjectName (before insert, before update) {
    for (ObjectName obj : Trigger.new) {
        if (!Schema.sObjectType.ObjectName.fields.FieldName__c.isAccessible()) {
            obj.addError('FieldName__c is not accessible.');
        }
    }
}
```

17. Aggregate Query Too Many Rows This error occurs when an aggregate query returns too many rows.

Solution: Use filters to reduce the number of rows returned by aggregate queries.

```
apex
trigger LimitAggregateRows on ObjectName (after insert) {
    List<AggregateResult> results = [SELECT COUNT(Id) FROM ObjectName WHERE
Condition__c = true GROUP BY Field__c LIMIT 100];
    // Process limited aggregate results
}
```

18. Unsupported External Object Operation This error occurs when an unsupported operation is performed on an external object.

Solution: Ensure operations are supported for external objects or handle them differently.

```
apex
trigger HandleExternalObject on ExternalObject__x (before insert) {
    // Validate supported operations
    for (ExternalObject__x obj : Trigger.new) {
        if (obj.UnsupportedOperation__c) {
            obj.addError('This operation is not supported for external objects.');
        }
    }
}
```

19. Invalid Type Exception This error occurs when the wrong data type is used in a trigger.

Solution: Ensure data types are correctly handled and cast appropriately.

```
apex
trigger ValidateDataType on ObjectName (before insert, before update) {
    for (ObjectName obj : Trigger.new) {
        if (!(obj.Field__c instanceof String)) {
            obj.addError('Field__c must be a string.');
        }
    }
}
```

20. System.ListException: List Index Out of Bounds This error occurs when accessing an invalid index in a list.

Solution: Check list size and validate indices before accessing list elements.

```
apex
trigger CheckListIndex on ObjectName (before insert, before update) {
    List<ObjectName> records = [SELECT Id FROM ObjectName WHERE Condition__c = true];
    if (records.size() > 0) {
        ObjectName firstRecord = records[0];
        // Process firstRecord
    }
}
```

Basic Trigger Questions

1. **What are the different events in a Salesforce trigger, and how would you decide which one to use?**
    - *Expect the candidate to mention before/after insert, update, delete, undelete events and explain their appropriate usage.*
2. **How do you bulkify a trigger? Can you provide an example?**

Intermediate Trigger Questions

3. **What is the difference between `Trigger.New` and `Trigger.NewMap`? When would you use each?**
    - *Looking for clarity on their purpose, with examples.*
4. **How do you prevent recursive trigger execution? Can you show an example?**
5. **What are some common reasons for CPU time limit exceptions in triggers, and how do you avoid them?**
6. **Can you explain the difference between a trigger and a process builder/flow, and when you would prefer one over the other?**
7. **How do you handle relationships (parent-child or child-parent) in a trigger? For instance, how would you update all child records when a parent record is updated?**

Advanced Trigger Questions

8. **Have you worked with trigger frameworks? Which framework did you use, and why?**
9. **How would you implement a scenario where a trigger needs to call an asynchronous process (like a Future or Queueable method)? Provide an example.**
10. **Suppose you have multiple triggers on the same object. How would you ensure they execute in the correct order?**
11. **Can you discuss a scenario where you had to write a trigger to solve a complex business requirement? What were the challenges, and how did you overcome them?**
12. **How do you test triggers in Salesforce, and what are your best practices for writing test classes for them?**

Scenario-Based Questions

13. **Scenario 1:**

- A company wants to automatically create a "Task" whenever an `Opportunity` is closed. The task should include details about the opportunity. How would you implement this in a trigger?

14. **Scenario 2:**
- When a `Case` is closed, all related `Task` records should be updated with a status of "Completed." Write the pseudo-code for this trigger and explain your thought process.

15. **Scenario 3:**
- You are tasked with creating a trigger to ensure that when an `Account` is deleted, all associated `Contacts` are also deleted. How would you approach this?

16. **Scenario 4:**
- A `Lead` is converted, and you need to copy certain custom field values from the `Lead` to the `Opportunity` and `Account`. How would you design this functionality in a trigger?

Best Practices and Troubleshooting

17. **What are some best practices you follow when writing triggers?**
18. **How do you debug a trigger that isn't working as expected or is throwing exceptions?**
19. **How do you ensure triggers don't conflict with each other when multiple triggers exist on the same object?**
20. **What tools or methods do you use to monitor the performance of triggers in production?**

Bonus:

21. **What changes have you noticed in trigger development over the past few years with Salesforce's push for declarative tools (like Flows)? How do you see the role of Apex triggers evolving?**
22. **How would you design and implement a trigger to handle millions of records efficiently, ensuring no governor limits are hit?**

Basic Questions

1. What is a trigger in Salesforce? A trigger is an Apex script that executes before or after specific events on Salesforce records, such as insertions, updates, or deletions.

2. How do you create a simple trigger in Salesforce? You can create a trigger in Salesforce using the following syntax:

```
apex
trigger TriggerName on ObjectName (before insert) {
    // trigger logic
}
```

3. What is the difference between a before trigger and an after trigger?

- **Before triggers** are used to update or validate record values before they're saved to the database.
- **After triggers** are used to access field values that are set by the system and to make changes in other records.

4. Can you name the events that can fire a trigger in Salesforce? Insert, update, delete, merge, upsert, undelete.

5. What is the `Trigger.new` context variable? `Trigger.new` is a list of the new versions of the sObject records.

6. What is the `Trigger.old` context variable? `Trigger.old` is a list of the old versions of the sObject records.

7. Can a single Apex trigger handle multiple events? Yes, a single trigger can handle multiple events like before insert, after insert, before update, etc.

8. What is recursion in triggers and how can it be avoided? Recursion occurs when a trigger calls itself repeatedly. It can be avoided by using static variables or a helper class.

9. What is a context variable in Salesforce triggers? Context variables are implicit variables that provide information about the trigger execution context.

10. Can we use DML operations inside a trigger? Yes, but it should be used judiciously to avoid hitting governor limits.

Intermediate Questions

11. How can you prevent triggers from executing recursively? By using a static boolean variable within an Apex class to control trigger execution.

12. What are the best practices for writing triggers in Salesforce?

- One trigger per object.
- Use context-specific handler methods.
- Avoid SOQL and DML inside loops.
- Write comprehensive test classes.

13. What is a trigger handler pattern? A design pattern where the business logic is separated from the trigger using a handler class.

14. How do you test triggers in Salesforce? By writing test classes that cover all possible scenarios and ensuring they meet the required code coverage.

15. What is a bulk trigger? A bulk trigger is designed to handle multiple records efficiently within the same execution context.

16. How can you ensure a trigger is bulk-safe? By using collections like lists or maps and avoiding SOQL and DML operations inside loops.

17. Can you disable a trigger in Salesforce? Yes, by using the `ApexTrigger` metadata and setting the status to 'Inactive.'

18. What are trigger exceptions and how do you handle them? Trigger exceptions occur when there is an issue during trigger execution. They can be handled using try-catch blocks.

19. Can triggers be called from batch Apex? Yes, triggers can be invoked from batch Apex when records are inserted, updated, or deleted.

20. What are trigger events? Trigger events are specific actions that cause a trigger to execute, such as insert, update, delete, etc.

Complex Scenarios

21. Scenario: Updating Parent Records When a `Child__c` record is inserted, update a field on the related `Parent__c` record.

Trigger Example:

```
apex
trigger UpdateParentField on Child__c (after insert) {
    Set<Id> parentIds = new Set<Id>();
    for (Child__c child : Trigger.new) {
        parentIds.add(child.Parent__c);
    }

    List<Parent__c> parents = [SELECT Id, FieldToUpdate__c FROM Parent__c WHERE Id IN
:parentIds];
    for (Parent__c parent : parents) {
        parent.FieldToUpdate__c = 'UpdatedValue';
    }
    update parents;
}
```

22. Scenario: Preventing Duplicate Records Prevent the creation of duplicate `Account` records based on a custom field.

Trigger Example:

```
apex
trigger PreventDuplicateAccounts on Account (before insert) {
    Set<String> customFieldSet = new Set<String>();
    for (Account acc : Trigger.new) {
        customFieldSet.add(acc.CustomField__c);
    }

    List<Account> existingAccounts = [SELECT Id, CustomField__c FROM Account WHERE
CustomField__c IN :customFieldSet];
    Map<String, Account> customFieldMap = new Map<String, Account>();
    for (Account acc : existingAccounts) {
        customFieldMap.put(acc.CustomField__c, acc);
    }

    for (Account acc : Trigger.new) {
        if (customFieldMap.containsKey(acc.CustomField__c)) {
            acc.addError('Duplicate record found.');
        }
    }
}
```

23. Scenario: Roll-Up Summary Trigger When a child record `OrderItem__c` is inserted, update the `TotalAmount__c` field on the parent `Order__c` object.

Trigger Example:

```
apex
trigger RollUpOrderAmount on OrderItem__c (after insert) {
    Map<Id, Decimal> orderAmountMap = new Map<Id, Decimal>();
    for (OrderItem__c item : Trigger.new) {
        if (orderAmountMap.containsKey(item.Order__c)) {
            orderAmountMap.put(item.Order__c, orderAmountMap.get(item.Order__c) +
item.Amount__c);
        } else {
            orderAmountMap.put(item.Order__c, item.Amount__c);
        }
    }
```

```
    List<Order__c> ordersToUpdate = new List<Order__c>();
    for (Id orderId : orderAmountMap.keySet()) {
        Order__c ord = new Order__c(Id = orderId, TotalAmount__c =
orderAmountMap.get(orderId));
        ordersToUpdate.add(ord);
    }
    update ordersToUpdate;
}
```

24. Scenario: Sending Email Notifications Send an email notification to the account owner when an opportunity's stage is updated to 'Closed Won'.

Trigger Example:

```
apex
trigger SendEmailNotification on Opportunity (after update) {
    List<Messaging.SingleEmailMessage> emails = new
List<Messaging.SingleEmailMessage>();
    for (Opportunity opp : Trigger.new) {
        if (opp.StageName == 'Closed Won' && opp.StageName !=
Trigger.oldMap.get(opp.Id).StageName) {
            Messaging.SingleEmailMessage email = new Messaging.SingleEmailMessage();
            email.setToAddresses(new String[] { opp.Owner.Email });
            email.setSubject('Opportunity Closed Won');
            email.setPlainTextBody('Congratulations! The opportunity ' + opp.Name + ' has been
closed won.');
            emails.add(email);
        }
    }
    Messaging.sendEmail(emails);
}
```

25. Scenario: Auto-Number Generation Generate a custom auto-number for a custom object `Invoice__c` on insert.

Trigger Example:

```
apex
trigger GenerateInvoiceNumber on Invoice__c (before insert) {
    for (Invoice__c inv : Trigger.new) {
        inv.InvoiceNumber__c = 'INV-' + String.valueOf(System.currentTimeMillis());
    }
```

```
}
```

26. Scenario: Updating Related Records When an `Account`'s status is updated to 'Inactive', update all related `Contacts`' statuses to 'Inactive'.

Trigger Example:

```
apex
trigger UpdateContactStatus on Account (after update) {
    List<Contact> contactsToUpdate = new List<Contact>();
    for (Account acc : Trigger.new) {
        if (acc.Status__c == 'Inactive' && acc.Status__c != Trigger.oldMap.get(acc.Id).Status__c) {
            contactsToUpdate.addAll([SELECT Id, Status__c FROM Contact WHERE AccountId =
:acc.Id]);
        }
    }

    for (Contact con : contactsToUpdate) {
        con.Status__c = 'Inactive';
    }
    update contactsToUpdate;
}
```

27. Scenario: Calculating Discounts Calculate and set a discount field on an `Opportunity` record based on a custom field value.

Trigger Example:

```
apex
trigger CalculateDiscount on Opportunity (before insert, before update) {
    for (Opportunity opp : Trigger.new) {
        opp.Discount__c = opp.Amount * (opp.DiscountRate__c / 100);
    }
}
```

28. Scenario: Archiving Records Move records from an `Order__c` object to an `OrderArchive__c` object after they are closed for 30 days.

Trigger Example:

```
apex
trigger ArchiveClosedOrders on Order__c (after update) {
```

```
    List<OrderArchive__c> ordersToArchive = new List<OrderArchive__c>();
    for (Order__c ord : Trigger.new) {
        if (ord.Status__c == 'Closed' && Date.today().addDays(-30) == ord.CloseDate) {
            OrderArchive__c archive = new OrderArchive__c();
            archive.OrderId__c = ord.Id;
            archive.Amount__c = ord.Amount;
            archive.CloseDate__c = ord.CloseDate;
            // add other fields as needed
            ordersToArchive.add(archive);
        }
    }
    insert ordersToArchive;
}
```

.

30. Scenario: Updating Fields on Dependent Objects When an `Account`'s industry is updated, update the `Industry__c` field on related `Contact` records.

Trigger Example:

```
apex
trigger UpdateContactIndustry on Account (after update) {
    Map<Id, String> accountIndustryMap = new Map<Id, String>();
    for (Account acc : Trigger.new) {
        if (acc.Industry != Trigger.oldMap.get(acc.Id).Industry) {
            accountIndustryMap.put(acc.Id, acc.Industry);
        }
    }

    List<Contact> contactsToUpdate = [SELECT Id, Industry__c, AccountId FROM Contact
WHERE AccountId IN :accountIndustryMap.keySet()];
    for (Contact con : contactsToUpdate) {
        con.Industry__c = accountIndustryMap.get(con.AccountId);
    }
    update contactsToUpdate;
}
```

31. Scenario: Ensuring Data Consistency Ensure the `CloseDate` of an `Opportunity` is always set to a future date.

Trigger Example:

```apex
trigger ValidateOpportunityCloseDate on Opportunity (before insert, before update) {
    for (Opportunity opp : Trigger.new) {
        if (opp.CloseDate <= Date.today()) {
            opp.addError('Close Date must be a future date.');
        }
    }
}
```

32. Scenario: Automatically Linking Related Records When a `Contact` is inserted, link it to an `Account` based on the `AccountName__c` custom field.

Trigger Example:

```apex
trigger LinkContactToAccount on Contact (before insert) {
    Map<String, Id> accountNameIdMap = new Map<String, Id>();
    for (Account acc : [SELECT Id, Name FROM Account]) {
        accountNameIdMap.put(acc.Name, acc.Id);
    }

    for (Contact con : Trigger.new) {
        if (accountNameIdMap.containsKey(con.AccountName__c)) {
            con.AccountId = accountNameIdMap.get(con.AccountName__c);
        }
    }
}
```

33. Scenario: Creating Related Records Automatically Automatically create a `Task` when a new `Lead` is inserted.

Trigger Example:

```apex
trigger CreateTaskOnLeadInsert on Lead (after insert) {
    List<Task> tasksToCreate = new List<Task>();
    for (Lead lead : Trigger.new) {
        Task newTask = new Task(
            WhatId = lead.Id,
            Subject = 'Follow up with lead',
            ActivityDate = Date.today().addDays(7)
        );
        tasksToCreate.add(newTask);
```

```
    }
    insert tasksToCreate;
}
```

34. Scenario: Restricting Updates to Specific Fields Prevent updates to the `AccountNumber` field on the `Account` object.

Trigger Example:

```
apex
trigger PreventAccountNumberUpdate on Account (before update) {
    for (Account acc : Trigger.new) {
        if (acc.AccountNumber != Trigger.oldMap.get(acc.Id).AccountNumber) {
            acc.AccountNumber.addError('Account Number cannot be updated.');
        }
    }
}
```

35. Scenario: Calculating and Setting a Custom Field Calculate a custom score based on the values of multiple fields on the `Contact` object.

Trigger Example:

```
apex
trigger CalculateContactScore on Contact (before insert, before update) {
    for (Contact con : Trigger.new) {
        con.Score__c = (con.Field1__c * 0.3) + (con.Field2__c * 0.5) + (con.Field3__c * 0.2);
    }
}
```

36. Scenario: Sending Notifications on Record Deletion Send an email notification when a `Case` record is deleted.

Trigger Example:

```
apex
trigger NotifyOnCaseDeletion on Case (after delete) {
    List<Messaging.SingleEmailMessage> emails = new
List<Messaging.SingleEmailMessage>();
    for (Case caseRecord : Trigger.old) {
        Messaging.SingleEmailMessage email = new Messaging.SingleEmailMessage();
        email.setToAddresses(new String[] { caseRecord.Owner.Email });
```

```
        email.setSubject('Case Deleted');
        email.setPlainTextBody('The case ' + caseRecord.CaseNumber + ' has been deleted.');
        emails.add(email);
    }
    Messaging.sendEmail(emails);
}
```

37. Scenario: Archiving Records Based on Criteria Archive `Event` records that are older than a year.

Trigger Example:

```
apex
trigger ArchiveOldEvents on Event (after insert, after update) {
    List<EventArchive__c> eventsToArchive = new List<EventArchive__c>();
    for (Event evt : [SELECT Id, Subject, StartDateTime FROM Event WHERE StartDateTime <
:Date.today().addYears(-1)]) {
        EventArchive__c archive = new EventArchive__c();
        archive.EventId__c = evt.Id;
        archive.Subject__c = evt.Subject;
        archive.StartDateTime__c = evt.StartDateTime;
        // add other fields as needed
        eventsToArchive.add(archive);
    }
    insert eventsToArchive;
}
```

38. Scenario: Validating Related Records on Insert Ensure that `Opportunity` records are not created if the related `Account` is inactive.

Trigger Example:

```
apex
trigger ValidateOpportunityAccountStatus on Opportunity (before insert) {
    Set<Id> accountIds = new Set<Id>();
    for (Opportunity opp : Trigger.new) {
        accountIds.add(opp.AccountId);
    }

    Map<Id, Account> accountMap = new Map<Id, Account>([SELECT Id, IsActive__c FROM
Account WHERE Id IN :accountIds]);
    for (Opportunity opp : Trigger.new) {
        if (accountMap.get(opp.AccountId).IsActive__c == false) {
```

```
            opp.addError('Cannot create an Opportunity for an inactive Account.');
        }
    }
}
```

39. Scenario: Automatically Calculating and Setting a Date Field Set the `FollowUpDate__c` field on a `Lead` record to 7 days after the `CreatedDate`.

Trigger Example:

```
apex
trigger SetFollowUpDate on Lead (before insert) {
    for (Lead lead : Trigger.new) {
        lead.FollowUpDate__c = lead.CreatedDate.addDays(7);
    }
}
```

40. Scenario: Validating Record Data Before Insert Ensure that the `Phone` field on `Contact` records is not empty.

Trigger Example:

```
apex
trigger ValidateContactPhone on Contact (before insert, before update) {
    for (Contact con : Trigger.new) {
        if (String.isBlank(con.Phone)) {
            con.addError('Phone number cannot be empty.');
        }
    }
}
```

41. Scenario: Creating Related Records on Condition Create a `Task` for the owner when an `Opportunity` is updated to the 'Negotiation' stage.

Trigger Example:

```
apex
trigger CreateTaskOnOpportunityUpdate on Opportunity (after update) {
    List<Task> tasksToCreate = new List<Task>();
    for (Opportunity opp : Trigger.new) {
        if (opp.StageName == 'Negotiation' && opp.StageName !=
Trigger.oldMap.get(opp.Id).StageName) {
```

```
            Task newTask = new Task(
                WhatId = opp.Id,
                Subject = 'Follow up on Negotiation',
                ActivityDate = Date.today().addDays(3)
            );
            tasksToCreate.add(newTask);
        }
    }
    insert tasksToCreate;
}
```

42. Scenario: Preventing Insert of Duplicate Records Prevent the insertion of duplicate `Contact` records based on email.

Trigger Example:

apex

```
trigger PreventDuplicateContacts on Contact (before insert) {
    Set<String> emailSet = new Set<String>();
    for (Contact con : Trigger.new) {
        emailSet.add(con.Email);
    }

    List<Contact> existingContacts = [SELECT Id, Email FROM Contact WHERE Email IN
:emailSet];
    Map<String, Contact> emailMap = new Map<String, Contact>();
    for (Contact con : existingContacts) {
        emailMap.put(con.Email, con);
    }

    for (Contact con : Trigger.new) {
        if (emailMap.containsKey(con.Email)) {
            con.addError('Duplicate contact found.');
        }
    }
}
```

43. Scenario: Calculating and Setting a Custom Score Calculate and set a custom score on an `Opportunity` record based on several fields.

Trigger Example:

apex

```
trigger CalculateOpportunityScore on Opportunity (before insert, before update) {
    for (Opportunity opp : Trigger.new) {
        opp.Score__c = (opp.Amount * 0.3) + (opp.Probability * 0.5) + (opp.StageOrder__c * 0.2);
    }
}
```

44. Scenario: Archiving Records Based on Status Archive `Case` records when their status is set to 'Closed' for more than 90 days.

Trigger Example:

```
apex
trigger ArchiveClosedCases on Case (after update) {
    List<CaseArchive__c> casesToArchive = new List<CaseArchive__c>();
    for (Case caseRecord : [SELECT Id, CaseNumber, Status, ClosedDate FROM Case WHERE
Status = 'Closed' AND ClosedDate < :Date.today().addDays(-90)]) {
        CaseArchive__c archive = new CaseArchive__c();
        archive.CaseId__c = caseRecord.Id;
        archive.CaseNumber
```

1. System.LimitException: Too many SOQL queries: 101

Description: This error occurs when the number of SOQL queries executed in a single transaction exceeds the limit (100 queries).

Possible Resolutions:

- Optimize your code to reduce the number of queries.
- Use collections (lists, maps, sets) to perform queries outside loops.
- Use a single query to retrieve related records.

```
apex
// Example of optimized SOQL
Map<Id, Account> accountMap = new Map<Id, Account>([SELECT Id, Name FROM Account
WHERE Id IN :accountIds]);
```

2. System.LimitException: Too many DML statements: 151

Description: This error occurs when the number of DML statements executed in a single transaction exceeds the limit (150 statements).

Possible Resolutions:

- Batch your DML operations into a single statement.
- Avoid DML operations inside loops.

```
apex
// Example of batched DML operations
List<Contact> contactsToUpdate = new List<Contact>();
for (Contact con : Trigger.new) {
    con.Phone = '123-456-7890';
    contactsToUpdate.add(con);
}
update contactsToUpdate;
```

3. System.LimitException: Apex CPU time limit exceeded

Description: This error occurs when the Apex code execution exceeds the maximum CPU time limit.

Possible Resolutions:

- Optimize your code for efficiency.
- Use maps to avoid nested loops.
- Move complex logic to asynchronous methods (e.g., @future, Queueable).

```
apex
// Example of using maps to optimize code
Map<Id, Contact> contactMap = new Map<Id, Contact>();
for (Contact con : Trigger.new) {
    contactMap.put(con.Id, con);
}
```

4. System.NullPointerException

Description: This error occurs when the code tries to access or dereference a null object.

Possible Resolutions:

- Add null checks before accessing objects or their fields.
- Ensure variables are properly initialized.

```
apex
// Example of null checks
if (contact.AccountId != null) {
```

```
    Account acc = [SELECT Id, Name FROM Account WHERE Id = :contact.AccountId];
}
```

5. DmlException: Insert failed. First exception on row 0; first error: FIELD_CUSTOM_VALIDATION_EXCEPTION, Custom Validation Error: []

Description: This error occurs when a record fails to meet custom validation rules during a DML operation.

Possible Resolutions:

- Review and adjust validation rules to align with business requirements.
- Ensure trigger logic complies with validation rules.

```
apex
// Ensure trigger logic meets validation rules
if (String.isBlank(contact.Email)) {
    contact.addError('Email cannot be blank.');
}
```

6. Mixed DML Operation Error

Description: This error occurs when DML operations are performed on both setup and non-setup objects in the same transaction.

Possible Resolutions:

- Use asynchronous methods (e.g., @future, Queueable) to separate setup and non-setup DML operations.

```
apex
@future
public static void updateUser(User userToUpdate) {
    update userToUpdate;
}
```

7. Too many query rows: 50001

Description: This error occurs when the number of query rows returned exceeds the governor limit (50,000 rows).

Possible Resolutions:

- Use query filters and `LIMIT` clauses to reduce the number of rows returned.
- Use pagination to process large datasets in smaller batches.

apex
// Example of limiting query rows
List<Account> accounts = [SELECT Id, Name FROM Account LIMIT 100];

8. System.AsyncException: Future method cannot be called from a future or batch method

Description: This error occurs when a future method tries to call another future or batch method.

Possible Resolutions:

- Avoid chaining future methods or batch jobs. Use Queueable instead for chaining asynchronous operations.

apex
public class FirstQueueableJob implements Queueable {
    public void execute(QueueableContext context) {
        // Perform logic
        System.enqueueJob(new SecondQueueableJob());
    }
}

9. Invalid Cross-Reference Id

Description: This error occurs when a cross-reference ID does not match any record in the system.

Possible Resolutions:

- Validate the existence of cross-reference IDs before performing operations.
- Use error handling to manage invalid references.

apex
if (contact.AccountId != null) {
    Account acc = [SELECT Id FROM Account WHERE Id = :contact.AccountId];
    if (acc == null) {
        contact.addError('Invalid Account reference.');
    }
}

10. System.SObjectException: DML operation is not allowed on User

Description: This error occurs when attempting unauthorized DML operations on specific system objects like User.

Possible Resolutions:

- Ensure the operation is allowed and the context user has the necessary permissions.
- Use `System.runAs` in tests to simulate a different user context.

apex
```
// Ensure user has necessary permissions
if (!UserInfo.getUserRoleId().isEmpty()) {
    update [SELECT Id FROM User WHERE Id = :userId];
}
```