



Nripesh Kumar Joshi (Salesforce Developer)

Important Topics in Salesforce

Admin

- Report
- Dashboard
- Data Modeling (Object, Field, Relationship)
- Validation Rule
- Record Type
- Page Layout
- Automation Tool (Flow, Approval Process, Process Builder and Workflow)
- **Data Security** - Object Level Security, Field Level Security, Record Level Security
- (Profile, Permission Set, OWD, Role Sharing Rule, Manual Sharing)
- Assignment Rule
- Auto Responsive Rule
- Web To Lead
- Email To Lead

Development

- Apex
- **Test Class** - Best Practices of Test Class, Some Annotation that are used in Test class

- **Aura Component** - Aura Bundle, Communication between Component (Parent to Child, Child to Parent), Aura Method, Attribute, Events (Application Event and Component Event)
-
- **LWC** -
 - ** Lifecycle hook
 - ** Aura Vs LWC
 - ** Communication between Component (Parent to Child, Child to Parent)
 - ** Way of fetching data from apex,
 - ** Decorators
 - ** LMS
 - ** Pub Sub Model
- **SOQL AND SOSL** - Parent to Child Relationship Query and Child to Parent Relationship Query, Query with date, filter
- Integration
- Remote Site Setting
- Named Credential
- Connected App
- Workbench/Postman
- Deployment (Through Vscode or Change Set)
- Governor Limits

Different Clouds in Salesforce -

- Sales Cloud
- Service Cloud

- Experience Cloud/ Community Cloud
- Financial Service Cloud
- Health Cloud
- Einstein Analytics Cloud
- Marketing Cloud
- Revenue Cloud
- Data Cloud
- Education Cloud
- Industry Cloud

Different Api that you can learn to grow -

- REST API
- SOAP API
- Metadata API
- Bulk API
- Tooling API
- GraphQL API

Various Technologies in Salesforce -

- Salesforce CPQ
- Vlocity
- Heroku
- Mulesoft
- Copado
- Omnistudio

Latest Technologies in AI -

- Sales GPT
- Service GPT
- Einstein GPT

Certificates for Admin →

- Salesforce Administrator
- Advanced Administrator
- Salesforce Business Analyst
- Salesforce CPQ Specialist
- Marketing Cloud Administrator

Certificates for Application Architect -

- Data Architect
- Sharing and Visibility Architect
- Platform Developer I
- Platform App Builder

Certificates for System Architect -

- Platform Developer I
- Development Lifecycle and Deployment Architect
- Identity and Access Management Architect
- Integration Architect

Certificates for Solution Architect -



Certificates for Salesforce Developer -

- Platform Developer I
- Platform Developer II
- JavaScript Developer I
- B2C Commerce Developer
- Industries CPQ Developer
- Marketing Cloud Developer
- OmniStudio Developer

Certificates for Salesforce Consultant -

- Business Analyst
- Data Cloud Consultant
- Education Cloud Consultant
- Experience Cloud Consultant
- Field Service Consultant

- Marketing Cloud Account Engagement Consultant
- Marketing Cloud Consultant
- Nonprofit Cloud Consultant
- OmniStudio Consultant
- Sales Cloud Consultant
- Service Cloud Consultant
- Tableau CRM & Einstein Discovery Consultant

Nripenesh Joshi



Nripesh Kumar Joshi (Salesforce Developer)

Some SOQL Query Examples in Salesforce (Part 1)

1. Fetch the child record from parent record (Contact is child and Account is Parent) – Standard Objects

```
SELECT Id, Name, (SELECT Id, LastName FROM Contacts) FROM Account.
```

Note- Contacts is a **Child Relationship Name** which is lookup field in child object

2. Fetch the child record from parent record (Student__c is child and Teacher__c is Parent) – Custom Objects

```
SELECT Id, Name, (SELECT Id, Name FROM Students__r) FROM Teacher__c
```

Note - Students is a **Child Relationship Name**(appended ' __r' in Students) which is lookup field in child object

3. Fetch the Parent record from Child Record (Contact is child and Account is Parent) – Standard Object

```
SELECT Id, Account.Name FROM Contact
```

4. Fetch the Parent record from Child Record (Student__c is child and Teacher__c is Parent) – Custom Object

```
SELECT Id, Teacher__r.Name FROM Student__c
```

Note- Here we don't need to add **s** in Relationship

5. Fetch the Account that has no contact record

```
SELECT Id, Name FROM Account WHERE Id NOT IN (SELECT AccountId FROM Contact)
```



Nripesh Kumar Joshi (Salesforce Developer)

Note- AccountId(Id of Account that is associated with contact) is lookup field

6. Fetch the Latest Account Record

```
SELECT Id, Name, CreatedDate FROM Account ORDER BY CreatedDate DESC
```

7. Fetch the Account record which is group by Name

```
SELECT Count(Id), Name FROM Account GROUP BY Name
```

Note- We can not use count with the field that is used in Group By.

For example we can not count name because we are using Name field in Group By

8. Determine how many leads are associated with each LeadSource value

```
SELECT LeadSource, COUNT(Name) FROM Lead GROUP BY LeadSource
```

9. Fetch the Lead Record that are associate with each LeadSource that generated more than 10 times

```
SELECT LeadSource, COUNT(Name) FROM Lead GROUP BY LeadSource Having Count(Name)>10
```

10. Fetch the Lead record where name end with 'abc'

```
SELECT LeadSource, Name FROM Lead WHERE Name LIKE '%abc'
```

Some Clauses in SOQL Query

In the above Queries we have used many clauses.

*** **NOT** - NOT keyword is used for negation.



Nripesh Kumar Joshi (Salesforce Developer)

*** **Group By** - GROUP BY is used with Aggregate functions to group the result set by single or multiple columns.

*** **Order By** - ORDER BY is used to sort the records in ascending(ASC) or descending(DESC) order. It is used after the WHERE clause.

*** **Like** - LIKE keyword allows selective queries using wildcards.

*** **Having** - HAVING is an optional clause that can be used in a SOQL query to filter results that aggregate functions return.

*** **WHERE** - We can add the condition with the help of WHERE Clause

Governor Limit for SOQL

Description	Synchronous Limit	Asynchronous Limit
Total number of SOQL queries issued ¹	100	200
Total number of records retrieved by SOQL queries	50,000	50,000



Nripesh Kumar Joshi (Salesforce Developer)

Some SOQL Query Examples in Salesforce (Part 2)

1. How to fetch all fields in SOQL Query?

Ans : Suppose you want to fetch all fields of **contact** object then you can use **FIELDS(ALL)**.

FIELDS(ALL) – This fetches all the fields of an object. This is similar like Select * from SQL.

SELECT FIELDS(ALL) FROM Contact Limit 10 (For Standard Object)

FIELDS(STANDARD) – This fetches all standard fields of an object.

SELECT FIELDS(STANDARD) FROM Contact Limit 10

FIELDS(CUSTOM) – This fetches all custom fields of an object.

SELECT FIELDS(CUSTOM) FROM Contact Limit 10

2. Fetch the records from the recycle bin using soql ?

If you try that in the developer console, though, you'll get an “**Unknown error parsing query**” message.

So, to execute the query, you need to open the anonymous code window from the developer console.



Nripesh Kumar Joshi (Salesforce Developer)

Some SOQL Query Examples in Salesforce (Part 2)

```
List<Account> acc = [SELECT Id, isDeleted FROM Account WHERE isDeleted =  
TRUE ALL ROWS];
```

```
system.debug('Deleted Account>>>' + acc);
```

3. For Update Clause in soql ?

FOR UPDATE to lock sObject records while they're being updated in order to prevent race conditions and other thread safety problems.

While an sObject record is locked, no other client or user is allowed to make updates either through code or the Salesforce user interface. The client locking the records can perform logic on the records and make updates with the guarantee that the locked records won't be changed by another client during the lock period.

```
SELECT Id, Name FROM Account FOR UPDATE
```

4. Different Operators in SOQL ?

AND - Use AND to return records that meet two conditions.

This query returns all records that have the first name **Nripesh** and the last name kum.

```
SELECT Name FROM Contact WHERE FirstName = 'Nripesh' AND LastName= 'Kum'
```



Nripesh Kumar Joshi (Salesforce Developer)

Some SOQL Query Examples in Salesforce (Part 2)

OR - Use **OR** to return records that meet one of two conditions. This query returns records with the last name Nripesh or the last name Kumar.

```
SELECT Name, Email FROM Contact WHERE FirstName = 'Nripesh' OR LastName = 'Kumar'
```

IN- Use **IN** to return records that meet at least one of three or more conditions. The **IN** clause is commonly used to return the values of a picklist, or values from a **LIST** or **SET**. This query returns all records that have the last name **James**, **Barr**, **Nedaerk**, or **Forbes**.

```
SELECT Name, Email FROM Contact WHERE LastName IN ('James', 'Barr', 'Nedaerk', 'Forbes')
```

ASC - Returns results in ascending order

```
SELECT Name, Email FROM Contact ORDER BY Name ASC LIMIT 5
```

DESC- Returns results in descending order

```
SELECT Name FROM Contact ORDER BY Name DESC LIMIT 5
```



Nripesh Kumar Joshi (Salesforce Developer)

Some SOQL Query Examples in Salesforce (Part 2)

NULLS FIRST | LAST - Returns null records at the beginning (NULLS FIRST) or end (NULLS LAST)

SELECT Name, Email FROM Contact ORDER BY Email NULLS LAST

SELECT Name, Email FROM Contact ORDER BY Email NULLS First

5. We want only accounts that have a related contact with the last name

Forbes ?

**SELECT Name, (SELECT Name FROM Contacts) FROM Account WHERE Id IN
(SELECT AccountId FROM Contact WHERE LastName = 'Kumar')**

Apex Trigger

salesforce

Nripesh Kumar Joshi
Salesforce Developer



Apex Trigger

- Apex triggers enable you to perform custom actions before or after events to records in Salesforce, such as insertions, updates, or deletions.
- Use Apex triggers if performance and scale is important, if your logic is too complex for the point-and-click tools.
- Salesforce automatically fires active triggers when the specified database events occur.

Trigger Syntax

- A trigger definition starts with the trigger keyword. It is then followed by the name of the trigger, the Salesforce object that the trigger is associated with, and the conditions under which it fires.

```
1 trigger TriggerName on ObjectName (trigger_events) {  
2     code_block  
3 }
```

[Copy](#)

Trigger Events

- To execute a trigger before or after insert, update, delete, and undelete operations, specify multiple trigger events in a comma-separated list. The events you can specify are:
 - before insert
 - before update
 - before delete
 - after insert
 - after update
 - after delete
 - after undelete

Types of Apex Trigger

- There are two types of trigger.

Before Trigger

Before trigger is used to update or validate the record values before save to the database

After Trigger

After trigger is used to access the field values that are set by the system such as record id and to affect changes in other record

Context Variables

- To access the records that caused the trigger to fire.

isExecuting

- Returns true if the current context for the Apex code is a trigger, not a Visualforce page, a Web service, or an executeanonymous() API call.

isBefore

- Returns true if this trigger was fired before any record was saved.

isAfter

- Returns true if this trigger was fired after all records were saved.

isInsert

- Returns true if this trigger was fired due to an insert operation, from the Salesforce user interface, Apex, or the API.

isUpdate

- Returns true if this trigger was fired due to an update operation, from the Salesforce user interface, Apex, or the API.

isDelete

- Returns true if this trigger was fired due to a delete operation, from the Salesforce user interface, Apex, or the API.



Nripesh

Context Variables

isUndelete

- Returns true if this trigger was fired after a record is recovered from the Recycle Bin (that is, after an undelete operation from the Salesforce user interface, Apex, or the API.)

new

- Returns a list of the new versions of the sObject records. Note that this sObject list is only available in insert and update triggers, and the records can only be modified in before triggers.

newMap

- A map of IDs to the new versions of the sObject records. Note that this map is only available in before update, after insert, and after update triggers.

old

- Returns a list of the old versions of the sObject records. Note that this sObject list is only available in update and delete triggers.

oldMap

- A map of IDs to the old versions of the sObject records. Note that this map is only available in update and delete triggers.

size

- The total number of records in a trigger invocation, both old and new.

How we can write apex Trigger ?

```
trigger ContextExampleTrigger on Account (before insert, after insert, after delete) {  
    if (Trigger.isInsert) {  
        if (Trigger.isBefore) {  
            // Process before insert  
        } else if (Trigger.isAfter) {  
            // Process after insert  
        }  
    }  
    else if (Trigger.isDelete) {  
        // Process after delete  
    }  
}
```

Nripesh

Best Practices of Apex Trigger

- One trigger per object
- Logic Less Trigger
- Context-specific handler methods
- Avoid SOQL Query inside for loop
- Avoid hardcoding IDs
- Avoid nested for loop
- Avoid DML inside for loop
- Bulkify Your Code
- Enforced Sharing in Salesforce
- Use @future Appropriately

One Trigger Per Object

- Can control the order of execution.
- If you create more than one trigger per object then you can not control which trigger logic will execute first. So always use One trigger per object.

Logic Less Trigger

- We should create a handler class and create a method inside that class and then write logic there.

Logic Less Trigger

1. Create an Apex Class
2. Create a method inside the apex class
3. Call the Apex Class method from Trigger

```
1 public class ContactTriggerHandler {  
2     public static void handleBeforeInsert(List<Contact> contactList){  
3         for(Contact con: contactList){  
4             con.Description = 'Login-Less Apex Triggers Rocks!';  
5         }  
6     }  
7 }
```

```
1 trigger ContactTrigger on Contact (before insert) {  
2     ContactTriggerHandler.handleBeforeInsert(Trigger.New);  
3 }
```


Avoid SOQL Query inside for loop

- As we all know that Salesforce works in Multi-Tenant Environment it is important that we keep all the limits in mind.
- There is a SOQL Limit of 100 in every Transaction.
- If you insert 101 contact records at once you will get the SOQL error.

```
1  public class ContactTriggerHandler {  
2  
3      public static void handleBeforeInsert(List<Contact> contactList){  
4          for(Contact con: contactList){  
5              con.Description = 'Login-Less Apex Triggers Rocks!';  
6              if(con.AccountId <> null){  
7                  Account acc = [SELECT Id, BillingCity FROM ACCOUNT WHERE ID =: con.AccountId];  
8                  con.MailingCity = acc.BillingCity;  
9                  // Add all other address fields  
10                 .....  
11             }  
12         }  
13     }  
14  
15     public static void handleAfterInsert(List<Contact> contactList){  
16         // validate the contact address using the API  
17         .....  
18     }
```

- Instead of using the SOQL Query, we should be taking help from Salesforce Collection and then reducing the error.

```
1 public class ContactTriggerHandler {
2     public static void handleBeforeInsert(List<Contact> contactList){
3         Set<String> accountIdsSet = new Set<String>();
4         for(Contact con: contactList){
5             if(con.AccountId <> null){
6                 accountIdsSet.add(con.AccountId);
7             }
8         }
9         Map<String, Account> accountMap = new Map<String, Account>();
10        for(Account acc : [SELECT Id, BillingCity FROM ACCOUNT WHERE ID IN: accountIdsSet] ){
11            accountMap.put(acc.Id, acc);
12        }
13        for(Contact con: contactList){
14            con.Description = 'Login-Less Apex Triggers Rocks!';
15            if(con.AccountId <> null){
16                Account acc = accountMap.get(con.AccountId);
17                con.MailingCity = acc.BillingCity;
18                // Add all other address fields
19            }
20        }
21    }
22    public static void handleAfterInsert(List<Contact> contactList){
23        // validate the contact address using the API
24    }
25
26 }
```

Avoid hardcoding IDs

- We should always avoid using hardcoding IDs in the Apex Class, Apex trigger. For example, if you wanted to check if the account record type is a business account then only process the records.

```
1 public static void handleAccount(List<Account> accountList){
2     String recordTypeId = '0125e000000P2xSAAS';
3     for(Account acc: accountList){
4         if(acc.RecordTypeId == recordTypeId){
5             // Process further Logic
6         } -----
7     }
8 }
```

Instead of using the hardcoding IDs, we should always use Custom labels. Store the record type name there in the custom label and then use that custom Label in the apex code to query the record type.

See the below-modified code 📌

```
1 public static void handleAccount(List<Account> accountList){
2     String recordTypeId = Schema.SObjectType.Account.getRecordTypeInfoByName()
3         .get(System.Label.BusinessAccountRecordTypeId)
4         .getRecordTypeId();
5
6     for(Account acc: accountList){
7         if(acc.RecordTypeId == recordTypeId){
8             // Process further Logic
9         } -----
10    }
11 }
```

Avoid nested for loop

- We should always try to avoid the nested for loop in the apex code which will impact the performance of our apex class.
- **Requirement** – When the contact is created, update the contact address same as the Account Address.

```
1 public static void handleBeforeInsert(List<Contact> contactList){
2     Set<String> accountIdsSet = new Set<String>();
3     for(Contact con: contactList){
4         if(con.AccountId != null){
5             accountIdsSet.add(con.AccountId);
6         }
7     }
8     List<Account> accountList = [SELECT Id, BillingCity FROM ACCOUNT WHERE ID IN: accountIdsSet];
9
10    // Level 1 For Loop
11    for(Contact con: contactList){
12        con.Description = 'Login-Less Apex Triggers Rocks!';
13        if(con.AccountId != null){
14            // Level 2 For Loop
15            for(Account acc: accountList){
16                if(con.AccountId == acc.Id){
17                    con.MailingCity = acc.BillingCity;
18                    // Add all other address fields
19                    .....
20                }
21            }
22        }
23    }
```

Avoid nested for loop

- Instead of using nested for loops, we should use collection variables especially Map in apex class and get rid of for loops.

```
1  public class ContactTriggerHandler {
2      public static void handleBeforeInsert(List<Contact> contactList){
3          Set<String> accountIdsSet = new Set<String>();
4          for(Contact con: contactList){
5              if(con.AccountId <> null){
6                  accountIdsSet.add(con.AccountId);
7              }
8          }
9          Map<String, Account> accountMap = new Map<String, Account>();
10         for(Account acc : [SELECT Id, BillingCity FROM ACCOUNT WHERE ID IN: accountIdsSet] ){
11             accountMap.put(acc.Id, acc);
12         }
13         for(Contact con: contactList){
14             con.Description = 'Login-Less Apex Triggers Rocks!';
15             if(con.AccountId <> null){
16                 Account acc = accountMap.get(con.AccountId);
17                 con.MailingCity = acc.BillingCity;
18                 // Add all other address fields
19             }
20         }
21     }
22     public static void handleAfterInsert(List<Contact> contactList){
23         // validate the contact address using the API
24     }
25 }
26 }
```

Nripesh

Avoid DML inside for loop

- we should avoid the SOQL statement inside for loop. In the similar fashion we should avoid making DML inside the for loop. We should use collection (List) to store all the records and then do the DML outside of for loop.
- For Example, You need to create a child case when the Contact is created.

Avoid DML inside for loop

```
1 public static void handleAfterInsert(List<Contact> contactList){
2     // validate the contact address using the API
3     List<Case> caseList = new List<Case>();
4     for(Contact con : contactList){
5         Case caseRecord = new Case();
6         caseRecord.Subject = 'Sample Case';
7         caseRecord.Origin = 'Email';
8         caseRecord.ContactId = con.Id;
9         caseRecord.Description = 'Sample Case';
10        insert caseRecord;
11    }
12 }
```

If we use DML inside for loop then we will get into the governor limit of 151 DML. That means we can only make 150 DML in one transaction.

Here is how we should use collection to avoid the DML inside for loop.

```
1 public static void handleAfterInsert(List<Contact> contactList){
2     // validate the contact address using the API
3     List<Case> caseList = new List<Case>();
4     for(Contact con : contactList){
5         Case caseRecord = new Case();
6         caseRecord.Subject = 'Sample Case';
7         caseRecord.Origin = 'Email';
8         caseRecord.ContactId = con.Id;
9         caseRecord.Description = 'Sample Case';
10        caseList.add(caseRecord);
11    }
12    insert caseList;
13 }
```

Bulkify Your Code

- Bulkifying Apex code refers to the concept of making sure the code properly handles more than one record at a time.

Example -->

```
trigger testTrigger on Acount__c(before insert) {  
    integer i = 1;  
    for (Acount__c acc: Trigger.new) {  
        acc.Address__c = 'Test Address ' + i;  
        i++;  
    }  
}
```


Use @future Appropriately

- Sometimes there are some scenarios where we want to run some logic in asynchronous mode or sometimes we get into some errors like Mixed DML operations.
- We need to keep the @future or Queueable class and use it wherever we can use it.
- Below are a couple of scenarios where we can use either @future or Queueable apex
 - We are getting mixed DML operation Error
 - We need to make a callout from Apex Trigger.

Recursive Trigger

- Recursion is the process of executing the same Trigger multiple times to update the record again and again due to automation.
- There is a way to avoid recursive trigger.
 - Use Static Boolean Variable.

- This error is occurred in below code

```
trigger TestTrigger on Test__c (before insert) {  
    insert new Test__c();  
}
```

How to avoid Recursive Trigger ?

- Create a class with a static Boolean variable with a default value of true.

Apex Class with Static Variable

```
public class ContactTriggerHandler{  
    public static Boolean isFirstTime = true;  
}
```

Trigger Code

```
Trigger ContactTriggers on Contact (after update){  
    Set<String> accIdSet = new Set<String>();  
    if(ContactTriggerHandler.isFirstTime){  
        ContactTriggerHandler.isFirstTime = false;  
        System.debug('---- Trigger run ---->'+Trigger.New.size() );  
        for(Contact conObj : Trigger.New){  
            if(conObj.name != 'Test') {  
                accIdSet.add(conObj.accountId);  
            }  
        }  
        // any code here  
    }  
}
```



Nripesh

References

- <https://www.pantherschools.com/apex-trigger-best-practices-in-salesforce/>
- <https://jayakrishnasfdc.wordpress.com/2020/02/23/recursive-trigger-in-salesforce/>



Thank You





SALESFORCE PRODIGIES

Bringing people together & sharing knowledge in wider scope

Nripesh Kumar Joshi

Batch Class in Salesforce

The Batch class is used to process millions of records within normal processing limits. If you have a lot of records to process then you should go with Batch class.

Batch class has three methods that are following.

1. Start Method
2. Execute Method
3. Finish Method

1. Start Method

This method will collect records or objects on which the operation should be performed.

Syntax of Start Method →

```
global Database.QueryLocator start(Database.BatchableContext BC){}
```

Database.Batchable interface require a reference to a Database.BatchableContext object. Use this object to track the progress of the batch job.

2. Execute Method

This method processes a subset of the scoped records and performs operations which we want to perform on the records fetched from the start method.

Syntax of Execute Method —>

```
global void execute(Database.BatchableContext BC, list<sobject>) {  
  
}
```

3. Finish Method

This method executes after all batches are processed. This method is used for any post job or wrap-up work like sending confirmation email notifications.

Syntax of Finish Method —>

```
global void finish(Database.BatchableContext BC) {  
  
}
```

Batch class should be implemented by Database.Batchable interface.

Example —>

```
global class AccountBatch implements Database.Batchable<sObject>  
{  
    global Database.QueryLocator start(Database.BatchableContext  
BC)  
    {
```

```

String query = 'SELECT Id,Name FROM Account';

return Database.getQueryLocator(query);
}
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) {
}
}

```

Database.Stateful Interface

Database.Stateful, you can maintain state across these transactions.

For example, if your batch job is executing one logic and you need to send an email at the end of the batch job with all successful records and failed records. For that, you can use Database.Stateful in the class definition.

```

public class MyBatchJob implements
    Database.Batchable<sObject>, Database.Stateful{

    public integer summary;

    public MyBatchJob(String q){

```



```

        Summary = 0;
    }

    public Database.QueryLocator start(Database.BatchableContext
BC){
        return Database.getQueryLocator(query);
    }

    public void execute(
        Database.BatchableContext BC,
        List<sObject> scope){
        for(sObject s : scope){
            Summary ++;
        }
    }

    public void finish(Database.BatchableContext BC){
    }
}

```

Scheduler Class For Batch Apex

Schedulable Class is a global class that implements the Schedulable interface. That includes one execute method. Here is example of a scheduler class.

```

global class AccountBatchJobscheduled implements Schedulable {
    global void execute(SchedulableContext sc) {
        AccountBatch b = new AccountBatch();
        database.executebatch(b);
    }
}

```

}

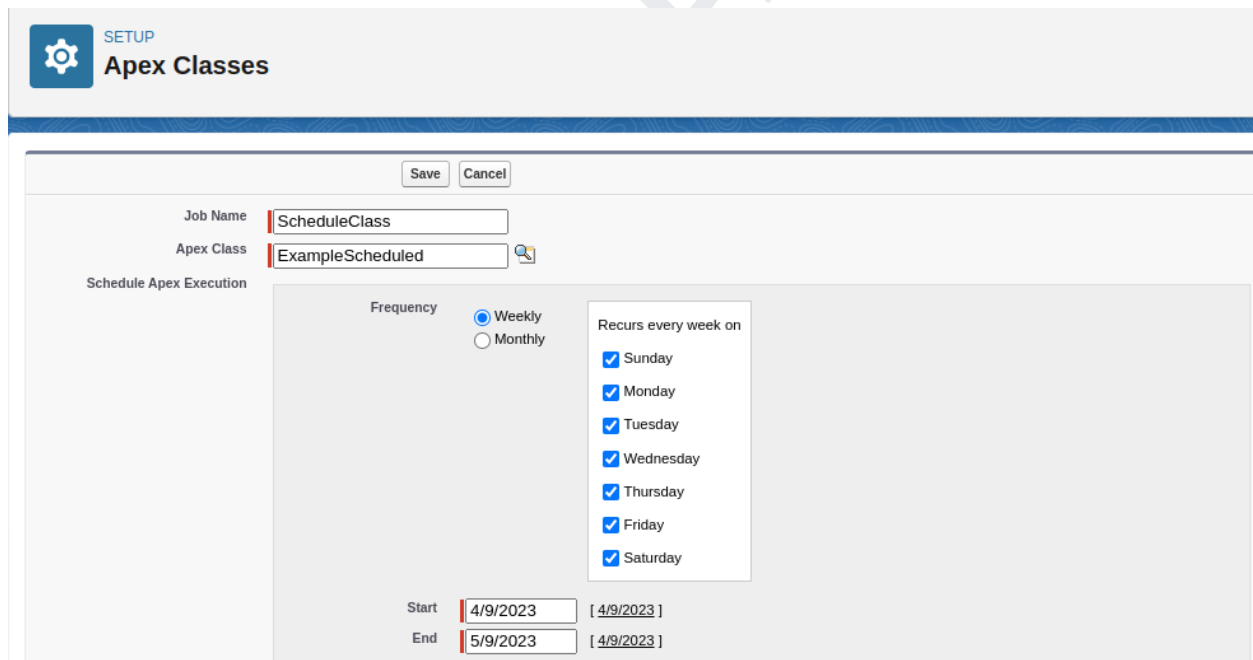
How to Schedule scheduler class

There are two options: we have scheduled the scheduler classes.

- 1) Declarative Approach
- 2) By Developer console

1. By Declarative Approach →

Step 1) Click on Setup->Apex class. Then search the Schedule Apex button.



The screenshot shows the Salesforce Setup page for scheduling an Apex class. The page title is "Apex Classes" under the "SETUP" menu. The "Job Name" field is set to "ScheduleClass" and the "Apex Class" field is set to "ExampleScheduled". The "Frequency" is set to "Weekly". A dropdown menu is open showing the days of the week, with all days (Sunday through Saturday) selected. The "Start" date is "4/9/2023" and the "End" date is "5/9/2023".

Job Name: ScheduleClass

Apex Class: ExampleScheduled

Frequency: Weekly

Recurs every week on:

- ☒ Sunday
- ☒ Monday
- ☒ Tuesday
- ☒ Wednesday
- ☒ Thursday
- ☒ Friday
- ☒ Saturday

Start: 4/9/2023 [4/9/2023]

End: 5/9/2023 [4/9/2023]

Step 2) Execute below code from developer console.

```
AccountBatchJobscheduled m = new AccountBatchJobscheduled();
```

```
String sch = '0 0 0 ? * * *';  
String jobId = system.schedule('Merge Job', sch, m);
```

What is CRON?

CRON is a software utility that is a time-based job scheduler in Unix-like computer operating systems. Developers who want to set up and maintain software environments, use this CRON to schedule jobs (commands or shell scripts) to run periodically at fixed times, dates, or intervals.

What is a CRON expression?

A CRON expression is basically a string of five or six fields separated by white spaces that represents a set of times, normally as a schedule to execute some routine.

Use in Salesforce

Use schedule with an Apex class that implements the Schedulable interface to schedule the class to run at the time specified by a Cron expression.

```
System.Schedule(JobName, CronExpression, SchedulableClass);
```

The System.Schedule method takes three arguments: a name for the job, an expression used to represent the time and date the job is scheduled to run, and the name of the class.

CRON expression has the following syntax:

```
0 0 5 ? * 1,2,3,4,5,6,7  
{1} {2} {3} {4} {5} {6}
```

{1} **Seconds** - so 0 here i.e. start of the minute.

{2} **Minutes** - 0 again so start of the hour.

{3} Hours - 5 so 5 am. Uses 24 hour notation so 21 = 9pm

{4} Day_of_month - ? means no specific value, only available for day of the month and day of the week.

{5} Month - * indicates all values, i.e. every month. (if we only want to run on 1st Jan say, this would be 1)

{6} Day_of_week - 1,2,3,4,5,6,7 here specifies days 1,2,3,4,5,6,7 in the week. We could also write this string as MON-FRI or preferably as * to indicate all values.

So this job reads to run at "0 seconds past 0 minutes of the 5th hour on no specific day of the month for every month of the year for every day of the week".

The following are the values for the expression:

Name	Values	Special Characters
<i>{1} Seconds</i>	0–59	None
<i>{2} Minutes</i>	0–59	None
<i>{3} Hours</i>	0–23	None
<i>{4} Day_of_month</i>	1–31	, - * ? / L W
<i>{5} Month</i>	1–12 or the following: ★ JAN ★ FEB ★ MAR ★ APR ★ MAY ★ JUN ★ JUL ★ AUG ★ SEP ★ OCT ★ NOV ★ DEC	, - * /
<i>{6} Day_of_week</i>	1–7 or the following: ★ SUN ★ MON ★ TUE ★ WED ★ THU ★ FRI ★ SAT	, - * ? / L #
<i>optional_year</i>	null or 1970–2099	, - * /

The special characters are defined as follows:

Special Character	Description
,	Delimits values. For example, use JAN, MAR, APR to specify more than one month.
-	Specifies a range. For example, use JAN-MAR to specify more than one month.
*	Specifies all values. For example, if Month is specified as *, the job is scheduled for every month.
?	Specifies no specific value. This is only available for Day_of_month and Day_of_week, and is generally used when specifying a value for one and not the other.
/	Specifies increments. The number before the slash specifies when the intervals will begin, and the number after the slash is the interval amount. For example, if you specify 1/5 for Day_of_month, the Apex class runs every fifth day of the month, starting on the first of the month.
L	Specifies the end of a range (last). This is only available for Day_of_month and Day_of_week. When used with Day of month, L always means the last day of the month, such as January 31, February 29 for leap years, and so on. When used with Day_of_week by itself, it always means 7 or SAT. When used with a Day_of_week value, it means the last of that type of day in the month. For example, if you specify 2L, you are specifying the last Monday of the month. Do not use a range of values with L as the results might be unexpected.
W	Specifies the nearest weekday (Monday-Friday) of the given day. This is only available for Day_of_month. For example, if you specify 20W, and the 20th is a Saturday, the class runs on the 19th. If you specify 1W, and the first is a Saturday, the class does not run in the previous month, but on the third, which is the following Monday.
#	Specifies the nth day of the month, in the format weekday#day_of_month. This is only available for Day_of_week. The number before the # specifies weekday (SUN-SAT). The number after the # specifies the day of the month. For example, specifying 2#2 means the class runs on the second Monday of every month.

NOTE: Use the L and W together to specify the last weekday of the month.

Cron Expression Examples

Expression	Description
0 0 0 ? * * *	at 12:00 AM every day
0 0 12 * * ?	at 12:00 PM every day
0 0 10 ? * *	at 10.00 AM every day
0 0 10 * * ?	at 10.00 AM every day
0 0 10 * * ? *	at 10.00 AM every day
0 0 15 ? * * *	at 3:00 PM every day
0 0-5 15 * * ?	Every minute starting at 3:00 PM and ending at 3:05 PM, every day
0 15 17 ? * MON-FRI	at 5:15 PM every Monday, Tuesday, Wednesday, Thursday and Friday
0 25 5 15 * ?	at 5:25 AM on the 15th day of every month
0 15 17 ? * 6#3	at 5:15 PM on the third Friday of every month
0 0 17 ? * 6L	runs the last Friday of every month at 5:00 PM.
'0 30 * * * *';	every 30 minutes
0 0 23 * * ? 2018	runs every day at 11:00 PM during the year 2016.

Example to schedule a class for every 5 min

System.schedule(Schedule Job Name 1', '0 00 * * * ?', new
testScheduleFiveMinutes());

System.schedule(Schedule Job Name 2', '0 05 * * * ?', new
testScheduleFiveMinutes());

System.schedule(Schedule Job Name 3', '0 10 * * * ?', new
testScheduleFiveMinutes());

System.schedule(Schedule Job Name 4', '0 15 * * * ?', new
testScheduleFiveMinutes());

System.schedule(Schedule Job Name 5', '0 20 * * * ?', new
testScheduleFiveMinutes());

System.schedule(Schedule Job Name 6', '0 25 * * * ?', new
testScheduleFiveMinutes());

System.schedule(Schedule Job Name 7', '0 30 * * * ?', new
testScheduleFiveMinutes());

System.schedule(Schedule Job Name 8', '0 35 * * * ?', new
testScheduleFiveMinutes());

System.schedule(Schedule Job Name 9', '0 40 * * * ?', new
testScheduleFiveMinutes());

System.schedule(Schedule Job Name 10', '0 45 * * * ?', new
testScheduleFiveMinutes());

System.schedule(Schedule Job Name 11', '0 50 * * * ?', new
testScheduleFiveMinutes());

System.schedule(Schedule Job Name 12', '0 55 * * * ?', new
testScheduleFiveMinutes());

Test Class For Batch Job

How to write a test class for Batch Job.

@isTest

```
public class AccountBatchTest {  
    static testMethod void testMethod1() {  
        List<Account> lstAccount= new List<Account>();  
        for(Integer i=0 ;i <200;i++) {  
            Account acc = new Account();  
            acc.Name ='Name'+i;  
            lstLead.add(acc);  
        }  
        insert lstAccount;  
        Test.startTest();  
        AccountBatch obj = new AccountBatch();  
        DataBase.executeBatch(obj);  
        Test.stopTest();  
    }  
}
```

Batch Class Interview Questions

- What is a Batch Apex?
- What is a Schedule apex?
- Where do we need to use Batch apex? What are the Features it offers?
- What access modifier should we use in Batch Class?
- What are the Methods we should Implement in Batch apex?
- What is the Use of Start method?
- What is the importance of the Execute method? and how many times it runs?

- What is the purpose of the finish method?
- What is a Database.Batchable interface?
- What is a Database.QueryLocator?
- What is the iterable<Sobject>?
- What is a Database.BatchableContext?
- What is the Difference between Synchronous and Asynchronous?
- Is Batch apex Synchronous or Asynchronous?
- What is the Difference between Stateless Or Stateful?
- Is Batch apex Stateless Or Stateful?
- How to Implement Stateful in Batch class?
- How can we schedule a batch class? How?
- Can we schedule Batch classes in Minutes or Hours? If yes, How?
- Can we call one Batch class to another Batch class?
- Can we schedule a batch apex class with in that batch class?
- Can we call future method from a batch class?
- Can we write batch class and schedulable class in a same class?
- Can we call callouts from batch apex?
- What is the Default batch size? and what is the max batch size?
- How to Execute a Batch Classes?
- How to track the details of the current running Batch job?
- What is the AsyncApexJob object?
- How many batch Jobs Active at a time?
- What are the Batch apex governor limits?
- What are the batch apex Best Practices?
- How to stop the Batch apex jobs?
- How to abort a batch jobs?
- What is BatchApexworker record type?



Salesforce Developer Interview Questions
(Admin + Development)

Service Based Companies
+
Product Based Companies



(TCS – 2 Yr Exp)

Admin

1. Difference between Custom Metadata and Custom Settings.
2. Difference between Role and Profile
3. Difference between dataloader and Data Import Wizard
4. Different Types of Flows
5. What are the following-
 - Sharing Settings
 - Owd
 - Role Hierarchy
 - Login Hours
 - Session Setting
 - Partner Community
 - Licenses

Development

6. What is Order of Execution in Salesforce.
7. Explain Governor Limits
8. Why do we write test classes in salesforce and what are the best practices of test class ?
9. What is lockout Error ?
10. What is the deployment process in Salesforce ?
11. Why do we use batch class in salesforce and write down the syntax of batch class ?
12. Types of exceptions in detail
13. Explain the Lifecycle Hooks in lwc
14. What are the decorators available in lwc and its use
15. Difference between wire and imperative wire in lwc
16. Do you know Some SLDS Classes ?
17. What is lightning data service (LDS) in lwc ?

19. What is the Lightning Message Service in lwc ?

20. **Program** – Write a trigger to create contacts based on Number of Contacts field on Account object while Account record save

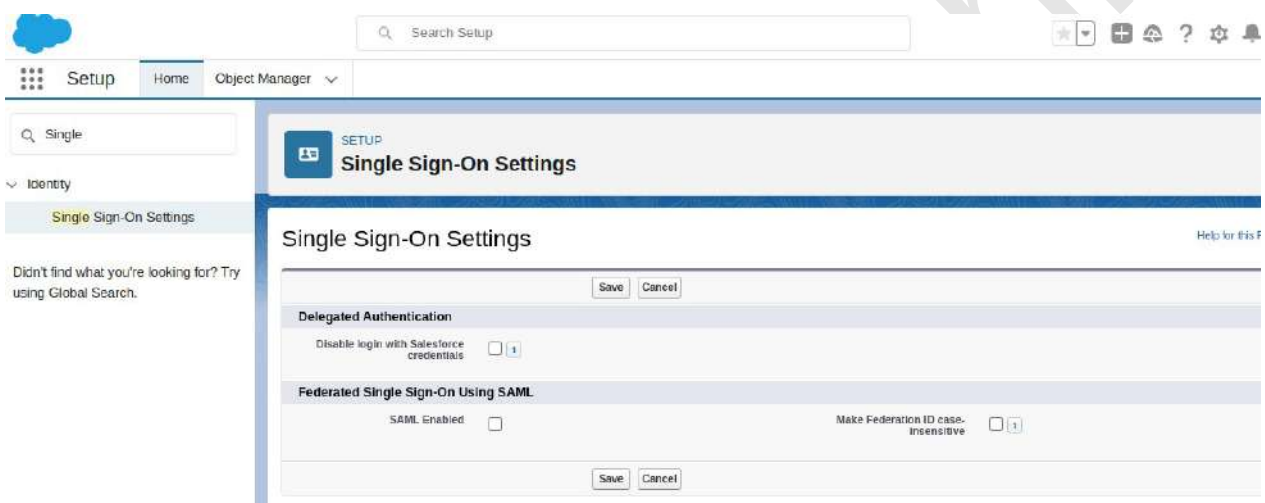


Integration

Q. What is Single Sign On ?

Ans -- > Single sign-on (SSO) allows users to access multiple applications with a single set of credentials. They do not need to remember separate user ID/password for each application. Salesforce offers various options for configuring single sign-on. This also includes:

- Federated Authentication using SAML
- Delegated Authentication
- OpenID Connect



We need to enable SAML in single sign on setting

Q. What is Identity Provider ?

Ans--> IdP stands for Identity Provider and SP stands for Service Provider. IdP is the system that authenticates user by validating the username and password and then subsequently all other applications trust IdP and allow user to access the application if the IdP asserts that the user is a valid user. IdP is the system that stores user's login name and password.

Q. What is Service Provider ?

Ans --> A service provider is a website that hosts apps. A Service Provider (SP) is the entity providing the service, typically in the form of an application. Let suppose we are accessing the google's credential to login salesforce org so Salesforce will be the Service Provider and Google will be the Identity provider because Google will validate the user and salesforce will give the service for doing work.

Q. What is Salesforce Integration ?

Ans--> Salesforce Integration is a process of connecting two or more applications.

Q. What are the Salesforce Integration Direction ?

Ans--> Integration can be two direction inbound integration or outbound Integration.

Inbound Integration: An external system initiates contact with Salesforce.

Outbound Integration: Salesforce initiates contact with an external system.

Q. What is outbound and inbound Integration in Salesforce ?

Ans - >

Inbound Web Service:

Inbound web service is when Salesforce exposes SOAP/REST web service, and any external/third party application consume it to get data from your Salesforce org. It is an Inbound call to Salesforce, but outbound call to the external system. Here, Salesforce is the publisher and external system is the consumer of web services.

Outbound Web Service:

Outbound web service is when Salesforce consume any external/third party application web service, a call needs to send to the external system. It is an Inbound call to the external system, but outbound call to Salesforce. Here, external system is the publisher of web services and Salesforce is the consumer.

Q. Different APIs in Salesforce and when we can use these Api?

Ans -

REST API

REST API is a simple and powerful web service based on RESTful principles. It exposes all sorts of Salesforce functionality via REST resources and HTTP methods. For example, you can create, read, update, and delete (CRUD) records, search or query your data, retrieve object metadata, and access information about limits in your org. REST API supports both XML and JSON.

Because REST API has a lightweight request and response framework and is easy to use, it's great for writing mobile and web apps.

SOAP API

SOAP API is a robust and powerful web service based on the industry-standard protocol of the same name. It uses a Web Services Description Language (WSDL) file to rigorously define the parameters for accessing data through the API. SOAP API supports XML only. Most of the SOAP API functionality is also available through REST API. It just depends on which standard better meets your needs.

Because SOAP API uses the WSDL file as a formal contract between the API and consumer, it's great for writing server-to-server integrations.

Bulk API

Bulk API is a specialized RESTful API for loading and querying lots of data at once. By lots, we mean 50,000 records or more. Bulk API is asynchronous, meaning that you can submit a request and come back later for the results. This approach is the preferred one when dealing with large amounts of data. There are two versions of Bulk API (1.0 and 2.0). Both versions handle large amounts of data, but we use Bulk API 2.0 in this module because it's a bit easier to use.

Bulk API is great for performing tasks that involve lots of records, such as loading data into your org for the first time.

Pub/Sub API

Use Pub/Sub API for integrating external systems with real-time events. You can subscribe to real-time events that trigger when changes are made to your data or subscribe to custom events. The APIs use a publish-subscribe, or pub/sub, model in which users can subscribe to channels that broadcast data changes or custom notifications.

The pub/sub model reduces the number of API requests by eliminating the need for making frequent API requests to get data. Pub/Sub API is great for writing apps that would otherwise need to frequently poll for changes.

When to Use Pub/Sub API

You can use Pub/Sub API to integrate external systems with real-time events. Streams of data are based on custom payloads through platform events or changes in Salesforce records through Change Data Capture. Within Salesforce, you can publish and subscribe to events with Apex triggers, Process Builder, and Flow Builder.

Pub/Sub API is built for high scale, bi-directional event integration with Salesforce. Use Pub/Sub API to efficiently publish and subscribe to binary event messages in the Apache Avro format. Pub/Sub API is based on gRPC and HTTP/2 and uses a pull-based model so you can control the subscription flow. With Pub/Sub API, you can use one of the 11 programming languages that gRPC supports.

When to Use Apex REST API

Use Apex REST API when you want to expose your Apex classes and methods so that external applications can access your code through REST architecture. Apex REST API supports both OAuth 2.0 and Session ID for authentication.

When to Use Apex SOAP API

Use Apex SOAP API when you want to expose Apex methods as SOAP web service APIs so that external applications can access your code through SOAP. Apex SOAP API supports both OAuth 2.0 and Session ID for authentication.

When to Use Tooling API

Use Tooling API to build custom development tools or apps for Platform applications. For example, you can use Tooling API to add features and functionality to your existing Platform tools and build dynamic modules into your enterprise integration tools. You can also use Tooling API to build specialized development tools for a specific application or service.

Tooling API's SOQL capabilities for many metadata types allow you to retrieve smaller pieces of metadata. Smaller retrieves improve performance, making Tooling API a good fit for developing interactive applications. Tooling API provides SOAP and REST interfaces.

When to Use GraphQL API

Build highly responsive and scalable apps by returning only the data a client needs, all in a single request. GraphQL API overcomes the challenges posed by traditional REST APIs through field selection, resource aggregation, and schema introspection. Field selection reduces the size of the payload, sending back only fields that were included in the query. Aggregations reduce round trips between the client and server, returning a set of related resources within a single response. Schema introspection enables a user to see the types, fields, and objects that the user has access to.

Q. What is connected App in Salesforce and how can we create connected app in Salesforce ?

Ans - A connected app is a framework that enables an external application to integrate with Salesforce using APIs and standard protocols, such as **Security Assertion Markup Language (SAML), OAuth, and OpenID Connect**. Connected apps use these protocols to authorize, authenticate, and provide single sign-on (SSO) for external apps.

Q. Some Scenarios for using connected app in Salesforce ?

Ans- Integration with a Custom Marketing Automation Tool

Implementation Steps:

Step 1: Create a Connected App in Salesforce:

1. Log in to Salesforce as an administrator.
2. Click on the gear icon in the top-right corner to access "Setup."
3. In the Quick Find box, type "App Manager" and select "App Manager."
4. Click the "New Connected App" button.
5. Configure the basic information:

- **Connected App Name**: Give your app a name (e.g., "MarketingAutomationIntegration").
- **API Name**: It will be automatically generated based on the app name.
- **Contact Email**: Provide a contact email address.
- **Enable OAuth Settings**: Check this box to enable OAuth authentication.

Step 2: Configure OAuth Settings:

6. In the "API (Enable OAuth Settings)" section, configure the OAuth settings:

- **Callback URL**: Provide the callback URL where your marketing automation tool will redirect users after authentication. It should be a URL within your marketing automation tool's settings.
- **Selected OAuth Scopes**: Choose the necessary OAuth scopes based on the permissions your integration needs. For example, you might need "Access and manage your data (api)" and "Perform requests on your behalf at any time (refresh_token, offline_access)."

- **Require Secret for Web Server Flow:** If your integration will use the Web Server OAuth flow, check this box.

7. Optionally, you can configure other settings, such as digital signatures or IP Relaxation settings, depending on your security requirements.

Step 3: Save the Connected App:

8. Click the "Save" button to save your connected app configuration.

Step 4: Note the Consumer Key and Secret:

9. After saving, Salesforce will generate a "Consumer Key" and "Consumer Secret." Keep these values secure as they'll be used for authentication.

Step 5: Configure Permitted Users:

10. In the "Profiles" related list, specify which user profiles or permission sets are allowed to use the connected app. Ensure that the appropriate users have access to the connected app.

Step 6: Implement Integration with the Marketing Automation Tool:

11. In your custom marketing automation tool, implement OAuth 2.0 authentication using the Salesforce connected app's Consumer Key and Secret. Follow Salesforce's OAuth 2.0 authentication flow to obtain access tokens.

12. Once you obtain an access token, use it to make authorized API requests to Salesforce. You can create and update leads, contacts, and other Salesforce objects as needed from your marketing automation tool.

Step 7: Monitor and Maintain:

13. Regularly monitor the integration's usage, and ensure that you handle token expiration and token refresh logic in your integration to maintain seamless connectivity.

Scenario 2- External Identity Providers:

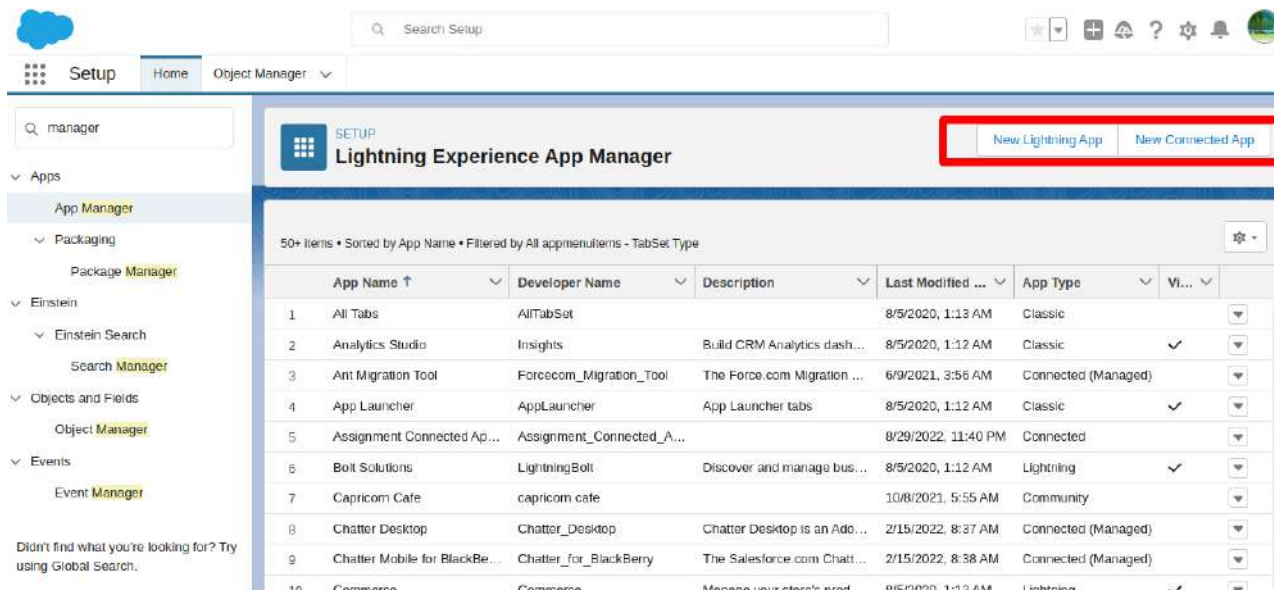
Your organization uses an external identity provider (IdP) for user authentication, and you want to connect Salesforce with this IdP.

- **Solution:** Create a connected app in Salesforce and configure it to use the IdP's authentication services, such as SAML or OpenID Connect, for user authentication and single sign-on.

Scenario 3 - Single Sign-On (SSO):

Scenario: Your company uses Salesforce for customer relationship management (CRM) and Google Workspace (formerly G Suite) for email and collaboration. You want your Salesforce users to access Google Workspace seamlessly without having to log in separately.

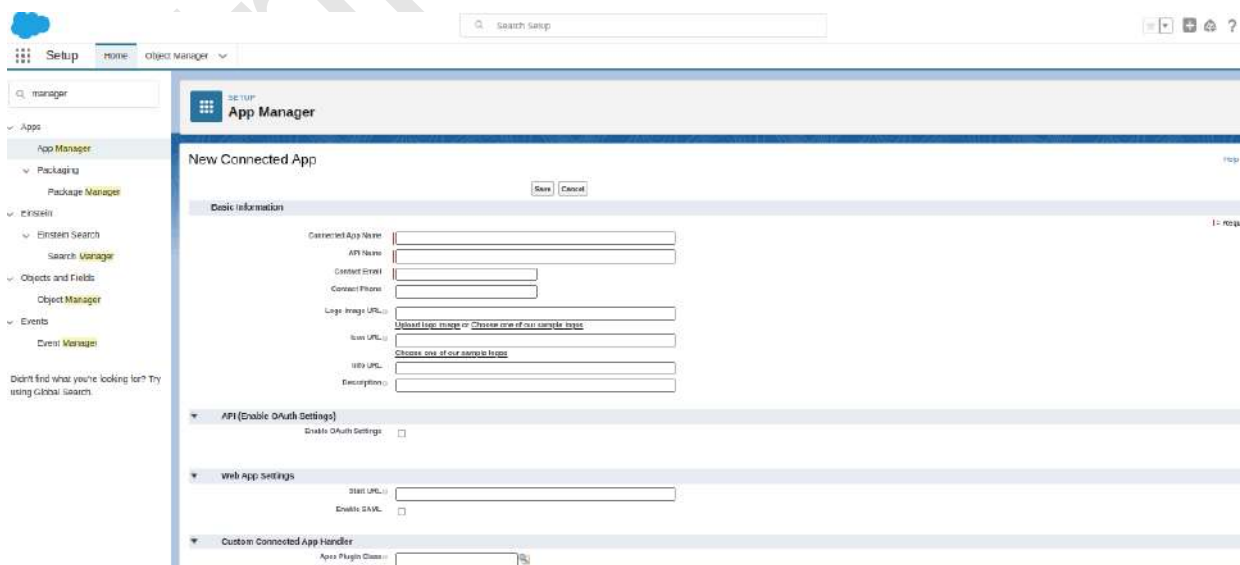
Solution: Create a connected app in Salesforce and configure it to use OAuth 2.0 for SSO with Google Workspace. This way, users can log in to Salesforce and access Google Workspace resources without additional login steps.



The screenshot shows the Salesforce Setup interface. In the left sidebar, the 'App Manager' link is selected under the 'Apps' section. The main content area displays the 'Lightning Experience App Manager' with a table of installed apps. In the top right corner of the app manager, there are two buttons: 'New Lightning App' and 'New Connected App'. The 'New Connected App' button is highlighted with a red rectangular box.

	App Name ↑	Developer Name	Description	Last Modified ...	App Type	Vi...
1	All Tabs	AllTabSet		8/5/2020, 1:13 AM	Classic	
2	Analytics Studio	Insights	Build CRM Analytics dash...	8/5/2020, 1:12 AM	Classic	✓
3	Ant Migration Tool	Forcecom_Migration_Tool	The Force.com Migration ...	6/9/2021, 3:56 AM	Connected (Managed)	
4	App Launcher	AppLauncher	App Launcher tabs	8/5/2020, 1:12 AM	Classic	✓
5	Assignment Connected Ap...	Assignment_Connected_A...		8/29/2022, 11:40 PM	Connected	
6	Bolt Solutions	LightningBolt	Discover and manage bus...	8/5/2020, 1:12 AM	Lightning	✓
7	Capricorn Cafe	capricorn cafe		10/8/2021, 5:55 AM	Community	
8	Chatter Desktop	Chatter_Desktop	Chatter Desktop is an Ado...	2/15/2022, 8:37 AM	Connected (Managed)	
9	Chatter Mobile for BlackBe...	Chatter_for_BlackBerry	The Salesforce.com Chatti...	2/15/2022, 8:38 AM	Connected (Managed)	
10	Commerce	Commerce	Manage your store's prod...	8/5/2020, 1:13 AM	Lightning	✓

If you want to create a connected app in Salesforce first you search on Quick find box 'App Manager' and select this after that you can click on **New Connected App**



The screenshot shows the 'New Connected App' form in Salesforce Setup. The form is divided into several sections:

- Basic information:** Contains fields for Connected App Name, API Name, Contact Email, Contact Phone, Logo Image URL, and icons (Upload logo image or Choose one of our sample logos).
- API (Enable OAuth Settings):** Includes a checkbox for 'Enable OAuth Settings'.
- Web app settings:** Includes a field for 'Start URL' and a checkbox for 'Enable SAML'.
- Custom Connected App Handler:** Includes a field for 'App Plugin Class'.

You need to fill the details as per your requirement

You can create connected app and try to implement in your org and refer this module for practice

https://trailhead.salesforce.com/content/learn/modules/mobile_sdk_introduction/mobilesdk_intro_security

Q. OAuth Terminologies

OAuth (Open Authorization): OAuth is an open standard protocol that enables secure authorization and authentication for granting access to resources, such as APIs, without exposing user credentials.

Client Application: The software application that wants to access protected resources on behalf of the user. In Salesforce integration, this could be a third-party app or system.

Resource Owner: The user or entity that owns the protected resource. In Salesforce, this is typically a Salesforce user whose data or resources are being accessed.

Authorization Server: In Salesforce, the authorization server is the Salesforce Identity and Access Management system. It handles user authentication and issues access tokens after the user grants permission.

Access Token: An access token is a credential used by the client application to access protected resources. In Salesforce, access tokens are short-lived and grant access to specific resources for a limited time.

Refresh Token: A refresh token is a long-lived credential that can be used to obtain a new access token when the current one expires. It is often used to maintain a long-term connection between the client application and Salesforce.

Authorization Code: In the Web Server OAuth Authentication Flow, after the user is authenticated, the authorization server issues an authorization code. This code is exchanged for an access token and a refresh token by the client application.

Consumer Key: It is value used by the consumer—in this case, the Mobile SDK app—to identify itself to Salesforce. Referred to as `client_id`.

Scopes: Scopes define the specific permissions and access rights requested by the client application. In Salesforce, scopes can control the level of access to objects and data.

Redirect URI (Callback URL): When the user is authenticated and grants permissions, the authorization server redirects the user's browser to a specific URL (the redirect URI) with the authorization code. In Salesforce integrations, this URL is often provided by the client application.

JWT (JSON Web Token): JWT is a compact, URL-safe means of representing claims to be transferred between two parties. In Salesforce, JWTs are used in the JWT Bearer Token OAuth Authentication Flow for secure communication.

Connected App: In Salesforce, a connected app represents the client application that wants to integrate with Salesforce using OAuth. Connected apps define various settings, including OAuth settings, to control the integration.

User-Agent Flow: Also known as the Implicit Flow, this OAuth flow is used for single-page applications and mobile apps where the access token is returned directly to the user's browser or app.

Username-Password Flow: This OAuth flow allows the client application to directly exchange the user's Salesforce credentials for an access token. It's generally discouraged due to security concerns.

Q. OAuth 1.0 vs OAuth 2.0 ?

Better support for non-browser applications

OAuth 1.0 has been designed focusing on the interactions of inbound and outbound messages in web client applications. Therefore, it is inefficient for non-browser clients. OAuth 2.0 has addressed this issue by introducing more authorization flows for different client needs that do not use web UIs.

Reduced complexity in signing requests

OAuth 1.0 needs to generate a signature on every API call to the server resource and that should be matched with the signature generated at the receiving endpoint in order to have access for the client. OAuth 2.0 do not need to generate signatures. It uses TLS/SSL (HTTPS) for communication.

The separation of roles

Handling resource requests and handling user authorization can be decoupled in OAuth 2.0. It has clearly defined the roles involved in communication which are client, resource owner, resource server, and authorization server.

The short-lived access token and the refresh token

In OAuth 1.0, access tokens can be stored for a year or more. But in OAuth 2.0, access tokens can contain an expiration time, which improves the security and reduces the chances of illegal access. And it offers a refresh token which can be used to get a new access token at the access token expiration without reauthorizing.

Q. What is Rest API ?

Ans -> The Salesforce REST API lets you integrate with Salesforce applications using simple HTTP methods, in either JSON or XML formats, making this an ideal API for developing mobile applications or external clients.

HTTP Method		Description
GET	>>>>	Retrieve data identified by a URL.
POST	>>>>	Create a resource or post data to the server.
DELETE	>>>>	Delete a resource identified by a URL.
PUT	>>>>	Create or replace the resource sent in the request body.
PATCH	>>>>	Partial update to an existing resource,

Rest Api Callout ---> Code

Piece of code --> Retrieve the data from third party (Get the Data)

```
Http http = new Http();
HttpRequest request = new HttpRequest();
request.setEndpoint('https://th-apex-http-callout.herokuapp.com/animals');
request.setMethod('GET');
HttpResponse response = http.send(request);
// If the request is successful, parse the JSON response.
if(response.getStatusCode() == 200) {
    // Deserialize the JSON string into collections of primitive data types.
    Map<String, Object> results = (Map<String, Object>)
JSON.deserializeUntyped(response.getBody());
    // Cast the values in the 'animals' key as a list
    List<Object> animals = (List<Object>) results.get('animals');
    System.debug('Received the following animals:');
    for(Object animal: animals) {
        System.debug(animal);
    }
}
```

Piece of code --> Send Data to a Service (Post)

```
Http http = new Http();
HttpRequest request = new HttpRequest();
request.setEndpoint('https://th-apex-http-callout.herokuapp.com/animals');
request.setMethod('POST');
request.setHeader('Content-Type', 'application/json; charset=UTF-8');
// Set the body as a JSON object
request.setBody('{"name": "mighty moose"}');
HttpResponse response = http.send(request);
// Parse the JSON response
if(response.getStatusCode() != 201) {
    System.debug('The status code returned was not expected: ' + response.getStatusCode() + ' +
response.getStatus());
} else {
    System.debug(response.getBody());
}
```

now you can test callout with the help of HttpCalloutMock

Test a Callout with HttpCalloutMock

To test your POST callout, we provide an implementation of the HttpCalloutMock interface. This interface enables you to specify the response that's sent in the respond method. Your test class instructs the Apex runtime to send this fake response by calling Test.setMock again. For the first argument, pass HttpCalloutMock.class. For the second argument, pass a new instance of

AnimalsHttpCalloutMock, which is your interface implementation of HttpCalloutMock. (We'll write AnimalsHttpCalloutMock in the example after this one.)

```
Test.setMock(HttpCalloutMock.class, new AnimalsHttpCalloutMock());
```

Now add the class that implements the HttpCalloutMock interface to intercept the callout. If an HTTP callout is invoked in test context, the callout is not made. Instead, you receive the mock response that you specify in the respond method implementation in AnimalsHttpCalloutMock.

Apex Class

AnimalsCallouts

```
public class AnimalsCallouts {
    public static HttpResponse makeGetCallout() {
        Http http = new Http();
        HttpRequest request = new HttpRequest();
        request.setEndpoint('https://th-apex-http-callout.herokuapp.com/animals');
        request.setMethod('GET');
        HttpResponse response = http.send(request);
        // If the request is successful, parse the JSON response.
        if(response.getStatusCode() == 200) {
            // Deserializes the JSON string into collections of primitive data types.
            Map<String, Object> results = (Map<String, Object>)
JSON.deserializeUntyped(response.getBody());
            // Cast the values in the 'animals' key as a list
            List<Object> animals = (List<Object>) results.get('animals');
            System.debug('Received the following animals:');
            for(Object animal: animals) {
                System.debug(animal);
            }
        }
        return response;
    }
    public static HttpResponse makePostCallout() {
        Http http = new Http();
        HttpRequest request = new HttpRequest();
        request.setEndpoint('https://th-apex-http-callout.herokuapp.com/animals');
        request.setMethod('POST');
        request.setHeader('Content-Type', 'application/json;charset=UTF-8');
        request.setBody('{"name":"mighty moose"}');
        HttpResponse response = http.send(request);
        // Parse the JSON response
        if(response.getStatusCode() != 201) {
            System.debug('The status code returned was not expected: ' +
                response.getStatusCode() + ' ' + response.getStatus());
        } else {
            System.debug(response.getBody());
        }
        return response;
    }
}
```

```
}
```

AnimalsCalloutsTest

```
@isTest
private class AnimalsCalloutsTest {
    @isTest static void testGetCallout() {
        // Create the mock response based on a static resource
        StaticResourceCalloutMock mock = new StaticResourceCalloutMock();
        mock.setStaticResource('GetAnimalResource');
        mock.setStatusCode(200);
        mock.setHeader('Content-Type', 'application/json;charset=UTF-8');
        // Associate the callout with a mock response
        Test.setMock(HttpCalloutMock.class, mock);
        // Call method to test
        HttpResponse result = AnimalsCallouts.makeGetCallout();
        // Verify mock response is not null
        System.assertNotEquals(null, result, 'The callout returned a null response.');
```

Trishanth Joshi

```
        // Verify status code
        System.assertEquals(200, result.getStatusCode(), 'The status code is not 200.');
```

Trishanth Joshi

```
        // Verify content type
        System.assertEquals('application/json;charset=UTF-8',
            result.getHeader('Content-Type'),
            'The content type value is not expected.');
```

Trishanth Joshi

```
        // Verify the array contains 3 items
        Map<String, Object> results = (Map<String, Object>)
            JSON.deserializeUntyped(result.getBody());
        List<Object> animals = (List<Object>) results.get('animals');
        System.assertEquals(3, animals.size(), 'The array should only contain 3 items.');
```

Trishanth Joshi

```
    }
}
```

HttpCalloutMock

```
@isTest
global class AnimalsHttpCalloutMock implements HttpCalloutMock {
    // Implement this interface method
    global HTTPResponse respond(HTTPRequest request) {
        // Create a fake response
        HttpResponse response = new HttpResponse();
        response.setHeader('Content-Type', 'application/json');
        response.setBody('{ "animals": ["majestic badger", "fluffy bunny", "scary bear", "chicken",
"mighty moose"]}');
        response.setStatusCode(200);
        return response;
    }
}
```

Create Custom Rest API In Salesforce

Sometimes we need to do some customization in OOB REST API for some complex implementation.

	Use	Action
@RestResource(urlMapping=“url”)	Defines the class as a custom Apex endpoint	
@HttpGet	Defines the function to be called via Http Get- Used to retrieve a record	Read
@HttpDelete	Used to delete a record	Delete
@HttpPost	Used to create a record	Create
@HttpPatch	Used to partially update a record	Upsert
@HttpPut	Used to fully update a record	Update

MyFirstRestAPIClass

RestContext --> To access the RestRequest and RestResponse objects in an Apex REST method.

```
@RestResource(urlMapping='/api/Account/*')
global with sharing class MyFirstRestAPIClass
{
    @HttpGet
    global static Account doGet() {
        RestRequest req = RestContext.request;
        RestResponse res = RestContext.response;
        String AccNumber = req.requestURI.substring(req.requestURI.lastIndexOf('/')+1);
        Account result = [SELECT Id, Name, Phone, Website FROM Account WHERE
AccountNumber = :AccNumber ];
        return result;
    }

    @HttpDelete
    global static void doDelete() {
        RestRequest req = RestContext.request;
        RestResponse res = RestContext.response;
        String AccNumber = req.requestURI.substring(req.requestURI.lastIndexOf('/')+1);
        Account result = [SELECT Id, Name, Phone, Website FROM Account WHERE
AccountNumber = :AccNumber ];
        delete result;
    }

    @HttpPost
    global static String doPost(String name,String phone,String AccountNumber ) {
        Account acc = new Account();
        acc.name= name;
        acc.phone=phone;
        acc.AccountNumber =AccountNumber ;
        insert acc;
    }
}
```



```

        return acc.id;
    }
}

```

Test Class for REST API

MyFirstRestAPIClassTest

@IsTest

```
private class MyFirstRestAPIClassTest {
```

```
    static testMethod void testGetMethod(){
```

```
        Account acc = new Account();
        acc.Name='Test';
        acc.AccountNumber ='12345';
        insert acc;
```

```
        RestRequest request = new RestRequest();
        request.requestUri ='/services/apexrest/api/Account/12345';
        request.httpMethod = 'GET';
        RestContext.request = request;
        Account acct = MyFirstRestAPIClass.doGet();
        System.assert(acct != null);
        System.assertEquals('Test', acct.Name);
```

```
    }
```

```
    static testMethod void testPostMethod(){
```

```
        RestRequest request = new RestRequest();
        request.requestUri ='/services/apexrest/api/Account/12345';
        request.httpMethod = 'POST';
        RestContext.request = request;
        String strId = MyFirstRestAPIClass.doPost('Amit','2345678','12345');
        System.assert(strId !=null );
```

```
    }
```

```
    static testMethod void testDeleteMethod(){
```

```
        Account acc = new Account();
        acc.Name='Test';
        acc.AccountNumber ='12345';
        insert acc;
```

```
        RestRequest request = new RestRequest();
        request.requestUri ='/services/apexrest/api/Account/12345';
        request.httpMethod = 'DELETE';
        RestContext.request = request;
        MyFirstRestAPIClass.doDelete();
```

```
        List<Account> ListAcct = [SELECT Id FROM Account WHERE Id=:acc.id];
```

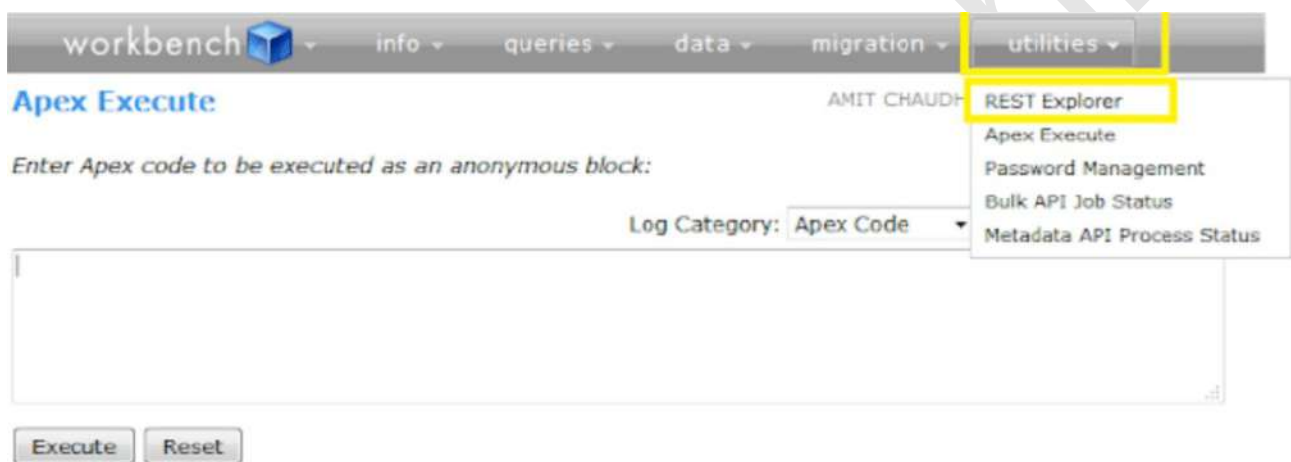
```
    System.assert(ListAcct.size() ==0 );  
}  
  
}
```

Execute Your Apex REST Class In Workbench

Step 1:- Open and log in.

Step 2:- Select Environment as Production and select the checkbox to agree on the terms and conditions then select log in with Salesforce

Step 3:- In the Workbench tool select Utilities > REST Explorer



Step 4:- In the REST Explorer window paste the following URL in the box

Method:- Get

URL:- /services/apexrest/api/Account/12345

REST Explorer

AMIT CHAUDHARY AT FORBLOG ON API 36.0

Choose an HTTP method to perform on the REST API service URI below:

☒ GET ☐ POST ☐ PUT ☐ PATCH ☐ DELETE ☐ HEAD Headers Reset Up

/services/apexrest/api/Account/12345

Execute

Expand All Collapse All Hide Raw Response

attributes
Id: 0019000001gJduwAAC
Name: Test Rest
Phone: 123456789

Requested in 0.727 sec
Workbench 36.0.1

Raw Response

```
HTTP/1.1 200 OK
Date: Wed, 13 Apr 2016 16:17:44 GMT
Content-Security-Policy-Report-Only: default-src https:; script-src https:
'unsafe-inline' 'unsafe-eval'; style-src https: 'unsafe-inline'; img-src
https: data:; font-src https: data:; report-uri
/_/ContentDomainCSPNoAuth?type=mydomain
Set-Cookie: BrowserId=StpbD_7_Iq2TintMfKv5A; Path=/; Domain=.salesforce.com;
Expires=Sun, 12-Jun-2016 16:17:44 GMT
Expires: Thu, 01 Jan 1970 00:00:00 GMT
Content-Type: application/json; charset=UTF-8
Content-Encoding: gzip
Transfer-Encoding: chunked
```

```
{
  "attributes" : {
    "type" : "Account",
    "url" : "/services/data/v36.0/sobjects/Account/0019000001gJduwAAC"
  },
  "Id" : "0019000001gJduwAAC",
  "Name" : "Test Rest",
  "Phone" : "123456789"
}
```

Nripesh Kumar Joshi (Salesforce Developer)

Collection(List, Set, Map) in Apex

List --> It is basically ordered list and contain duplicate values and we can access the element using index.

Example --> There are different ways of adding the value in List

Syntax of creating a list in apex

```
List<Integer> exampleList = new List<Integer>();  
List<String> exampleList = new List<String>{'a','b','c'};
```

A). Add the element in list with simple way

Exmple of List of integer

```
List<Integer> intList = new List<Integer>{1, 2,3,4,5};  
system.debug('Integer List'+ intList);
```

Execution Log		
Timestamp	Event	Details
17:27:15:002	USER_DEBUG	[2][DEBUG]Integer List(1, 2, 3, 4, 5)

Exmple of List of String

```
List<String> strList = new List<String>{'Sales Cloud', 'Service Cloud', 'Experience Cloud', 'Data  
Cloud', 'Financial Service Cloud'};  
system.debug('String List'+ strList);
```

Execution Log		
Timestamp	Event	Details
17:32:29:002	USER_DEBUG	[2][DEBUG]String List(Sales Cloud, Service Cloud, Experience Cloud, Data Cloud, Financial Service Cloud)

B). Adding element in last using add function

Exmple of List of integer

```
List<Integer> intList = new List<Integer>();  
intList.add(1);  
intList.add(2);  
intList.add(3);
```

```
intList.add(4);
intList.add(5);

system.debug('Integer List'+ intList);
```

Output ---->

Timestamp	Event	Details
16:59:50:003	USER_DEBUG	[7]]DEBUG Integer List(1, 2, 3, 4, 5)

Some methods that can be used in list

1. add(listElement): Adds an element to the end of the list.

```
List<String> strList = new List<String>();
strList.add('Sales Cloud');
strList.add('Service Cloud');
strList.add('Data Cloud');
strList.add('AI Cloud');
strList.add('Experience Cloud');
system.debug('String List'+ strList);
```

Timestamp	Event	Details
17:06:39:003	USER_DEBUG	[7]]DEBUG String List(Sales Cloud, Service Cloud, Data Cloud, AI Cloud, Experience Cloud)

2. add(index, listElement): Inserts an element into the list at the specified index position.

```
List<Integer> intList = new List<Integer>{1,2,3,4,5};
intList.add(3,0);
system.debug('Integer List'+ intList);
```

Execution Log		
Timestamp	Event	Details
17:52:59:003	USER_DEBUG	[3]]DEBUG Integer List(1, 2, 3, 0, 4, 5)

3. contains(listElement): Returns true if the list contains the specified element.

```
List<String> strList = new List<String>{'Sales Cloud', 'Service Cloud', 'Experience Cloud', 'Data Cloud', 'Financial Service Cloud'};
Boolean strContain = strList.contains('Sales Cloud');
system.debug('String List'+ strContain);
```

Execution Log		
Timestamp	Event	Details
18:11:44:002	USER_DEBUG	[3]DEBUG String Listtrue

4. size(): Returns the number of elements in the list.

```
List<String> strList = new List<String>{'Sales Cloud', 'Service Cloud', 'Experience Cloud', 'Data Cloud', 'Financial Service Cloud'};
system.debug('Return List size >>>>' + strList.size());
```

Execution Log		
Timestamp	Event	Details
18:15:25:003	USER_DEBUG	[2]DEBUG Return List size >>>> 5

5. isEmpty(): Returns true if the list has zero elements.

```
List<String> strList = new List<String>{'Sales Cloud', 'Service Cloud', 'Experience Cloud', 'Data Cloud', 'Financial Service Cloud'};
system.debug('Return true of list is empty otherwise false >>>>' + strList.isEmpty());
```

6. indexOf(listElement): Returns the index of the first occurrence of the specified element in this list.

```
List<String> strList = new List<String>{'Sales Cloud', 'Service Cloud', 'Experience Cloud', 'Data Cloud', 'Financial Service Cloud'};
system.debug('String>>>>' + strList.indexOf('Data Cloud'));
```

Execution Log		
Timestamp	Event	Details
18:28:35:003	USER_DEBUG	[2]DEBUG String>>>> 3

If this list does not contain the element, returns -1.

```
List<String> strList = new List<String>{'Sales Cloud', 'Service Cloud', 'Experience Cloud', 'Data Cloud', 'Financial Service Cloud'};
system.debug('String>>>>' + strList.indexOf('Cloud'));
```

Execution Log		
Timestamp	Event	Details
18:29:53:003	USER_DEBUG	[2]DEBUG String>>>> -1

SET – It is an unordered collection and it does not contain duplicate value

```
SET<String> exampleSet = new SET<String>();
exampleSet.add('a');
exampleSet.add('b');
exampleSet.add('c');
system.debug('Set>>>> '+ exampleSet);
```

Execution Log		
Timestamp	Event	Details
18:39:00:002	USER_DEBUG	[5] DEBUG Set>>>> {a, b, c}

if you will try to set the duplicate value, it will not add the duplicate value in set. In the below example you can see that duplicate value is not adding in set.

```
SET<String> exampleSet = new SET<String>();
exampleSet.add('a');
exampleSet.add('b');
exampleSet.add('c');
exampleSet.add('c');
system.debug('Set>>>> '+ exampleSet);
```

Execution Log		
Timestamp	Event	Details
18:39:00:002	USER_DEBUG	[5] DEBUG Set>>>> {a, b, c}

Set does not support the indexof(element) but we can access by value. All methods are same as list in SET.

Map - A map is a more complex collection than either a list or a set.

Each item in a map has two parts: **a key and a value**, known as a key-value pair. Keys and values can be any data type. Although each key is unique, values can be repeated within a map

**** Declare a Map Using the Map Keyword ****

```
Map<String, Integer> myMap = new Map<String, Integer>();
myMap.put('One', 1);
myMap.put('Two', 2);
system.debug('Map>>>>'+myMap);
```

Execution Log		
Timestamp	Event	Details
16:43:06:006	USER_DEBUG	[5] DEBUG Map>>>>{One=1, Two=2}

In this map, Key is the String and Integer is Value.

**** Declare and Initialize a Map with Values ****

```
Map<String, String> countryCapitalMap = new Map<String, String>{
    'USA' => 'Washington, D.C.',
    'India' => 'New Delhi',
    'Japan' => 'Tokyo'
};
system.debug('countryCapitalMap>>>>'+countryCapitalMap);
```

Here, countryCapitalMap is a map with string keys and string values, and it is initialized with key-value pairs.

USA,India, Japan are key and Washington, D.C, New Delhi and Tokyo are values

Execution Log		
Timestamp	Event	Details
16:47:21:002	USER_DEBUG	[7]DEBUG countryCapitalMap>>>>{India=New Delhi, Japan=Tokyo, USA=Washington, D.C.}

**** Declare a Map with a Specific Type Using Subject ****

```
Map<Id, Account> accountMap = new Map<Id, Account>();
List<Account> accounts = [SELECT Id, Name FROM Account LIMIT 5];
for (Account acc : accounts) {
    accountMap.put(acc.Id, acc);
}
system.debug('accountMap>>>>'+accountMap);
```

Execution Log		
Timestamp	Event	Details
16:57:42:013	USER_DEBUG	[6]DEBUG accountMap>>>>{0012w00000WQdGAAW:Account{id=0012w00000WQdGAAW, Name=My Batch Class22}, 0012w00000u2nORAAY:Account{id=0012w00000u2nORAAY, Name=Laila Account}, 0012w00000

In this case, accountMap is a map with Salesforce record IDs as keys and Account objects as values.

Example of map and explain the different methods

1. get(key): Retrieves the value associated with the specified key.

```
Map<String, String> countryCapitalMap = new Map<String, String>{
    'USA' => 'Washington, D.C.',
    'India' => 'New Delhi',
    'Japan' => 'Tokyo'
};
```


// Get the New Delhi value so we have to get the value using India Key

```
String value = countryCapitalMap.get('India');  
system.debug('countryCapitalMap value>>>>'+value);
```

Execution Log		
Timestamp	Event	Details
7:04:17:002	USER_DEBUG	[8]DEBUG countryCapitalMap value>>>>New Delhi

2. containsKey(key): Checks if the map contains a specific key.

```
Boolean value = countryCapitalMap.containsKey('India');  
Boolean val = countryCapitalMap.containsKey('UK');  
system.debug('countryCapitalMap value 1>>>>'+value);  
system.debug('countryCapitalMap val>>>>'+val);
```

Execution Log		
Timestamp	Event	Details
17:18:53:002	USER_DEBUG	[9]DEBUG countryCapitalMap value 1>>>>true
17:18:53:002	USER_DEBUG	[10]DEBUG countryCapitalMap val>>>>>false

3. keySet(): Returns a set of all the keys in the map.

```
Set<String> keys = countryCapitalMap.keySet();  
system.debug('keys >>>>'+keys);
```

Execution Log		
Timestamp	Event	Details
17:24:08:003	USER_DEBUG	[8]DEBUG keys >>>>{India, Japan, USA}

India, Japan, USA are keys

4. values(): Returns a list of all the values in the map.

```
List<String> allValues = countryCapitalMap.values();  
system.debug('allValues >>>>'+allValues);
```

Execution Log		
Timestamp	Event	Details
17:29:23:002	USER_DEBUG	[9]DEBUG allValues >>>>(Washington, D.C., New Delhi, Tokyo)

How to iterate on map.values()

```
// Declare a Map with a Specific Type Using SObject:
Map<Id, Account> accountMap = new Map<Id, Account>();
List<Account> accounts = [SELECT Id, Name FROM Account LIMIT 5];
for(Account acc : accounts) {
    accountMap.put(acc.Id, acc);
}

for(Account acc: accountMap.values()){
    system.debug('acc >>>>' + acc.Name);
}
```

Execution Log		
Timestamp	Event	Details
17:39:34:026	USER_DEBUG	[9]DEBUG acc >>>>My Batch Class15
17:39:34:026	USER_DEBUG	[9]DEBUG acc >>>>Lala Account
17:39:34:026	USER_DEBUG	[9]DEBUG acc >>>>Salesforce Trigger
17:39:34:026	USER_DEBUG	[9]DEBUG acc >>>>Salesforce Uptated trigger
17:39:34:026	USER_DEBUG	[9]DEBUG acc >>>>My Batch Class22

In this example are iterating over the value of accountMap and In above pic We can see the values of AccountMap (Account Name)

How to iterate on map.keySet()

```
Map<Id, Account> accountMap = new Map<Id, Account>();
List<Account> accounts = [SELECT Id, Name FROM Account LIMIT 5];
for(Account acc : accounts) {
    accountMap.put(acc.Id, acc);
}
for(Id acc: accountMap.keySet()){
    system.debug('acc >>>>' + acc);
}
```

Execution Log		
Timestamp	Event	Details
17:47:11:016	USER_DEBUG	[9]DEBUG acc >>>>0012w0000152zf3AAA
17:47:11:016	USER_DEBUG	[9]DEBUG acc >>>>0012w00000uznORAAY
17:47:11:016	USER_DEBUG	[9]DEBUG acc >>>>0012w00000uznU3AAI
17:47:11:017	USER_DEBUG	[9]DEBUG acc >>>>0012w00000uznU4AAI
17:47:11:017	USER_DEBUG	[9]DEBUG acc >>>>0012w00000rWQdGAAW

In the above pic you can see the key means (Id) as per our map

Difference between List, Set and Map

Feature	Map	Set	List
Definition	Collection of key-value pairs	Unordered collection of unique elements	Ordered collection of elements
Syntax	<code>Map<KeyType, ValueType> mapName = new Map<KeyType, ValueType>();</code>	<code>Set<ElementType> setName = new Set<ElementType>();</code>	<code>List<ElementType> listName = new List<ElementType>();</code>
Access by Index/Key	Access by key	Not applicable	Access by index (zero-based)
Order	No guaranteed order	No guaranteed order	Maintains insertion order
Duplicates	Unique keys; values may be duplicates	Unique elements	May contain duplicate elements
Example	<code>Map<String, Integer> myMap = new Map<String, Integer>(); myMap.put('One', 1);</code>	<code>Set<String> mySet = new Set<String>{'apple', 'orange', 'banana'};</code>	<code>List<String> myList = new List<String>{'apple', 'orange', 'banana'};</code>

Nripesh Joshi

Salesforce Developer

Asynchronous Apex Questions

1. Can we call future from future ?

Ans- No, not possible (Chaining is possible in queueable)

```
public class futureMethodExample {

    @future
    public static void MyFutureMethod(){
        Map<Id,Account> mapacct = new Map<Id,Account>([SELECT ID,Name FROM ACCOUNT]);
        system.debug('test method'+mapacct);
        // MyFutureMethod1(); // You can try to call one future method in another future method
    }

    @future(callout=true)
    public static void MyFutureMethod1(){
        Http http = new Http();
        HttpRequest request = new HttpRequest();
        request.setEndpoint('https://th-apex-http-callout.herokuapp.com/animals/');
        request.setMethod('GET');
        // System.debug('>>>>>>' + id);
        HttpResponse response = http.send(request);
        Object animals;
        String returnValue;

        // If the request is successful, parse the JSON response
        if (response.getStatusCode() == 200) {
            // Deserialize the JSON string into collections of primitive data types.
            Map<String, Object> result = (Map<String, Object>) JSON.deserializeUntyped(response.getBody());
            System.debug('>>>>>' + response.getBody());
            animals = result.get('animal');
            System.debug(animals);
        }

        BatchClassPractice bcn = new BatchClassPractice(); // You can try to call batch class from future method
        ID batchprocessid = Database.executeBatch(bcn);
    }
}
```

2. Can we call batch from schedule apex ?

Ans - Yes, We can call batch from schedule apex.

```
public class ScheduledBatchClassPractice implements schedulable{

    public void execute(SchedulableContext sc){
        try{
            BatchClassPractice btch = new BatchClassPractice(); // You can call batch class from schedule apex
            database.executeBatch(btch,400);

            // Call the future method in schedulable apex
            futureMethodExample.MyFutureMethod1();
        }catch(exception e){
            System.debug('Exception in ScheduledBatchClassPractice: ' + e.getMessage());
        }
    }
}
```

3. Can we call batch from batch ?

Ans - Yes, we can call from finish method but you cannot call in start or excute method.

```
public with sharing class BatchExample2 implements Database.Batchable<sObject>{
    public Database.QueryLocator start(Database.BatchableContext context) {
        return Database.getQueryLocator('SELECT ID, Mark_For_Delete_c FROM Studentc WHERE
Mark_For_Delete_c= \'Yes\');
    }
    public void execute(Database.BatchableContext context, List<SObject> records)
    {
        delete records;
    }
    public void finish(Database.BatchableContext bc) {
        BatchClassPractice btch = new BatchClassPractice();
        database.executeBatch(btch,200);
    }
}
```

// You can call batch class from batch in finish Method

4. Can we call future from batch ?

Ans- No, we cannot call otherwise Fatal error(System.AsyncException) error will occur.

**System.AsyncException: Future method cannot be called from a future or batch method:
futureMethodExample.MyFutureMethod()**

```
/* Write a batch a class to update the Teacher's records whose specialization is math */

public class BatchClassPractice implements Database.Batchable<Subject>, Database.stateful{

    public integer count = 0;

    public Database.QueryLocator start(Database.BatchableContext BC){
        String teacherList = 'SELECT Id, Name,Specialization__c FROM Teacher__c WHERE Specialization__c
        =\'Math\' OR Specialization__c =\'Physics\'';

        return Database.getQueryLocator(teacherList);
    }

    public void execute(Database.BatchableContext BC, List<Teacher__c> tchList){
        futureMethodExample.MyFutureMethod();

        if(!tchList.isEmpty()){
            for(Teacher__c tch: tchList){
                if(tch.Specialization__c == 'Math'){
                    tch.Name = 'Sir/Madam'+ ' ' + tch.Name;
                }if(tch.Specialization__c == 'Physics'){
                    tch.Name = 'Respected Madam'+ ' ' + tch.Name;
                }
                count++;
            }
            try{
                update tchList;
                system.debug('Count of Teacher'+count);
            }
            catch(exception e){
                System.debug(e);
            }
        }
    }

    public void finish(Database.BatchableContext BC){
        futureMethodExample.MyFutureMethod();
    }
}
```

5. Can we call batch from future ?

Ans- No, We cannot call otherwise Fatal Error(System.AsyncException) error will occur.

FATAL_ERROR System.AsyncException: Database.executeBatch cannot be called from a batch start, batch execute, or future method.

```
public class futureMethodExample {

    @future
    public static void MyFutureMethod(){
        Map<Id,Account> mapacct = new Map<Id,Account>([SELECT ID,Name FROM ACCOUNT]);
        system.debug('test method'+mapacct);
        // MyFutureMethod1(); // You can try to call one future method in another future method
    }

    @future(callout=true)
    public static void MyFutureMethod1(){
        Http http = new Http();
        HttpRequest request = new HttpRequest();
        request.setEndpoint('https://th-apex-http-callout.herokuapp.com/animals/');
        request.setMethod('GET');
        // System.debug('>>>>>>'+id);
        HttpResponse response = http.send(request);
        Object animals;
        String returnValue;

        // If the request is successful, parse the JSON response
        if (response.getStatusCode() == 200) {
            // Deserialize the JSON string into collections of primitive data types.
            Map<String, Object> result = (Map<String, Object>) JSON.deserializeUntyped(response.getBody());
            System.debug('>>>>>'+response.getBody());
            animals = result.get('animal');
            System.debug(animals);
        }

        BatchClassPractice bcn = new BatchClassPractice() ;
        ID batchprocessid = Database.executeBatch(bcn);
    }
}
```

6. Can we call future and batch from Queueable ?

Ans: Yes we can call future and batch from Queueable and vice-versa.

```
public class QueueableApexExample implements Queueable {  
  
    public static void execute(QueueableContext QC){  
  
        Account act = new Account(Name = 'Tested Accctt 97', Phone='0523124578',  
isAddress__c=true);  
        insert act;  
  
        futureMethodExample.MyFutureMethod1();  
  
    }  
}
```

7. Which interfaces are important in batch ?

Ans: Database.Batchable - It is mandatory in Batch Class

Database.AllowsCallout - It is mandatory while calling callout

Database.Stateful - This Interfaces is used to maintain the state of variable. Let's suppose you are inserting account and you want to count the all account that are inserted then you can use this interfaces.

8. What are the methods in batch class?

Ans- Start Method >> Return Type ==> Database.QueryLocator or Iterable

Execute Method >> Return Type ==> Void

Finish Method >> Return Type ==> Void

9. Some Interfaces in Asynchronous Apex ?

Ans-

1. Queueable,
2. Schedulable,
3. Database.Batchable

10. Can we call future method from Trigger ?

Ans- Yes, We can call future method from trigger.

```
public class FutureMethodExample2 {  
  
    // DML Operation performs on Setup and Non Setup Method in a single transaction  
    public static void insertSetWithNonSetup(){  
        Account act = new Account();  
        act.Name = 'Non Setup Record';  
        insert act;  
  
        //  
        InsertUserSetup();  
        System.debug(act);  
    }  
    @future  
    public static void insertUserSetup(){  
  
        Profile p = [SELECT Id FROM Profile WHERE Name='Cross Org Data Proxy User'];  
        UserRole r = [SELECT Id FROM UserRole WHERE Name='COO'];  
        // Create new user with a non-null user role ID  
        User usr = new User();  
        usr.Lastname = 'SetupTest';  
        usr.Alias = 'stest';  
        usr.email = 'nripeshjoshi97@gmail.com';  
        usr.ProfileId = p.Id;  
        usr.UserRoleId = r.Id;  
        insert usr;  
        system.debug(usr);  
    }  
}
```

```
Trigger AccountTrigger on Account(Before delete, Before update){
```

```
    if(trigger.isBefore && trigger.isUpdate){
        AccountHandlerClass.updateRelatedContactPhoneField(trigger.new, trigger.oldMap,
            trigger.newMap);
        // FutureMethodExample2.insertUserSetup(); // Calling Future Method
    }
    if(trigger.isBefore && trigger.isdelete){
        AccountHandlerClass.accountDelete(trigger.old);
    }
}
```

11. What are the way of monitor the schedule job ?

Ans: There are two way of monitor the job

1. Apex Jobs Page:

- **Setup > Apex Jobs** to see a list of Apex jobs, including scheduled jobs, their status, and execution details.

2. Developer Console:

- You can use the Developer Console to monitor and debug scheduled jobs. In the Developer Console, go to the "Query Editor" tab and execute the following query to retrieve scheduled jobs:

```
SELECT Id, CronExpression, NextFireTime, PreviousFireTime, StartTime, EndTime, Status,
JobItemsProcessed, TotalJobItems, NumberOfErrors, ExtendedStatus FROM CronTrigger
WHERE CronJobDetail.JobType = '7'
```

12. How can we stop scheduling job ?

Ans: // Retrieve the CronTrigger Id of the scheduled job
String jobName = 'YourScheduledJobName';
CronTrigger jobTrigger = [SELECT Id FROM CronTrigger WHERE CronJobDetail.Name = :jobName
LIMIT 1];
// Abort the job using the CronTrigger Id
System.abortJob(jobTrigger.Id);

Some Exception in Apex

SWIPE



Nripesh Kumar Joshi

DmlException:

When we are performing dml such as insert, update, delete and we are not populating require fields during the insertion then dmlException error occur.

```
Public class ExceptionExamples {  
  
    public static void exceptionError(){  
  
        Contact cont = new Contact();  
        cont.FirstName = 'Test';  
        insert cont;  
  
    }  
}
```

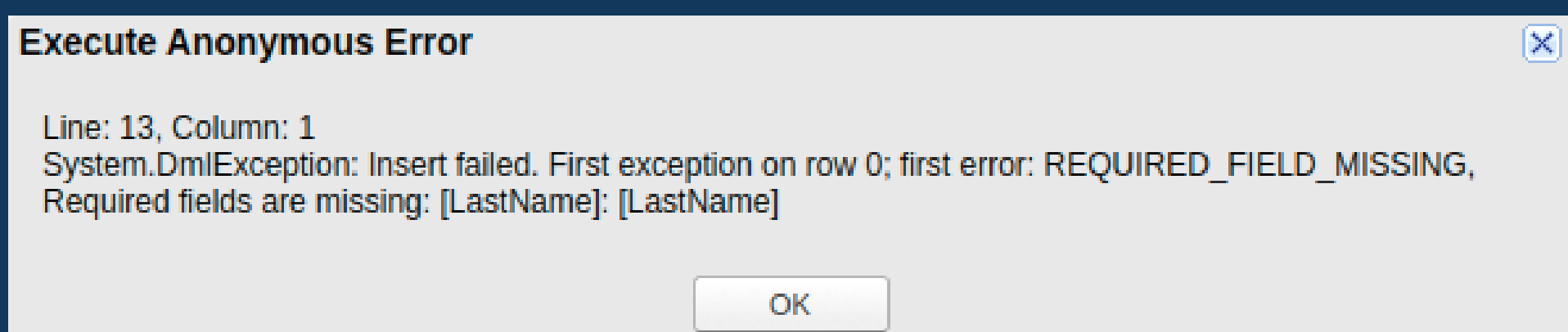


KEEP SWIPING

Nripesh Kumar Joshi

DmlException:

In above code dmlException error is occurred that is showing below.



To avoid this error, we should mention the required field during the dml operation in apex code.

KEEP SWIPING

Nripesh Kumar Joshi

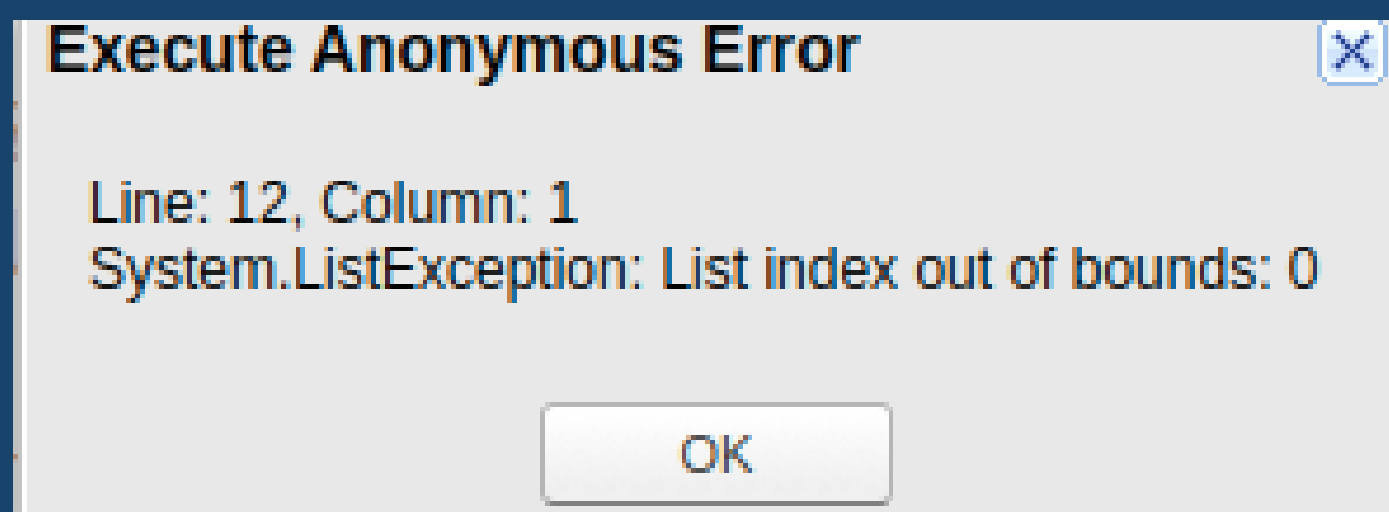
ListException

4

Whenever problem occurs in list in our apex code and accessing the element at an index that doesn't exist in the list then ListException is occurred.

```
Public class ExceptionExamples {  
  
    public static void exceptionError(){  
  
        List<Contact> contactList = [SELECT Id, Name FROM Contact  
WHERE Name ='SFDC Einstein'];  
        system.debug('List Exception' +contactList[0].Name);  
    }  
}
```

In above code ListException error is occurred that is showing below.



KEEP SWIPING

Nripesh Kumar Joshi

ListException

5

To avoid this error we should check the size of list

```
// How can we avoid this error ???  
// We should check the size of list  
/* if(contactList.size())>0){  
    system.debug('List Exception' +contactList[0].Name);  
  
} */
```

KEEP SWIPING

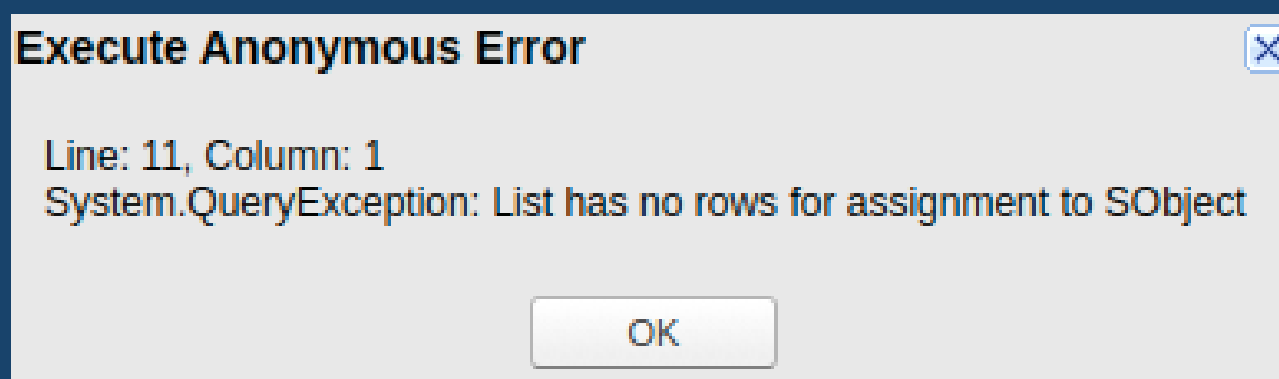
Nripesh Kumar Joshi

QueryException

This error is occurred when there's an issue with a soql. When soql query does not return any record and assigning that query in subject then this error is generally occurred.

```
Public class ExceptionExamples {  
  
    public static void exceptionError(){  
  
        Teacher__c teacherList = [SELECT Id, Name FROM Teacher__c  
WHERE Name ='Marc'];  
        system.debug('teacherList'+teacherList);  
  
    }  
}
```

In above code queryException error is occurred that is showing below.



KEEP SWIPING

QueryException

7

To avoid queryException we should use the List

We can update the line in above code.

```
LIST<TEACHER_C> TEACHERLIST = [SELECT ID, NAME FROM  
TEACHER_C WHERE NAME ='MARC'];
```



KEEP SWIPING

Nripesh Kumar Joshi

SubjectException

This occurs when you attempt to access a field on a queried SObject record, but that field was not included in the SELECT clause of the SOQL query.

```
Public class ExceptionExamples {  
  
    public static void exceptionError(){  
  
        List<Teacher__c> teacherList = [SELECT Id FROM Teacher__c];  
  
        if(teacherList.size()>0){  
            for(Teacher__c tch: teacherList){  
                system.debug('teacherList'+tch.Name);  
            }  
        }  
    }  
}
```



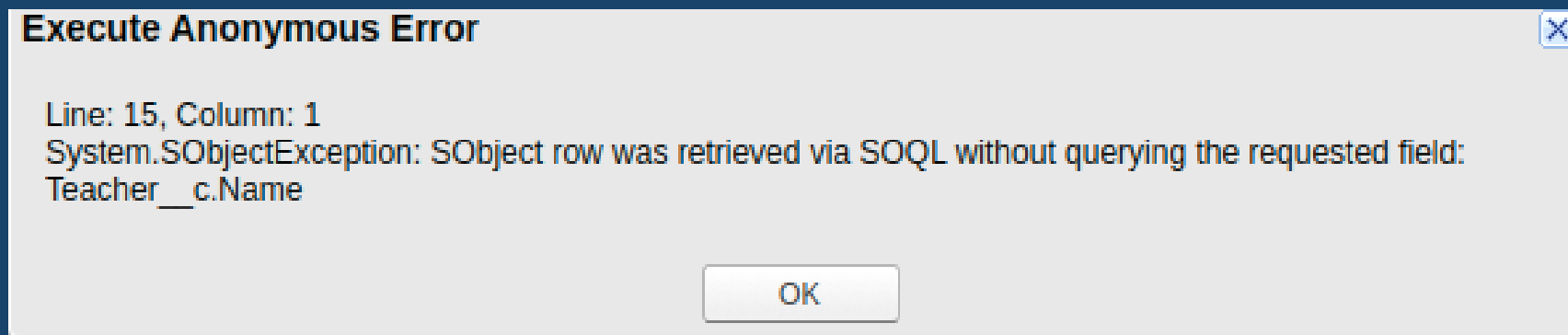
KEEP SWIPING

Nripesh Kumar Joshi

SubjectException

9

In above code subjectException error is occurred that is showing below.



To avoid this error we need to mention the field in soql query which we want to access in apex code.

```
List<Teacher__c> teacherList = [SELECT Id, Name FROM Teacher__c];
```

In above line we need to retrieve the field Name.

KEEP SWIPING

Nripesh Kumar Joshi

Limit Exception:

Salesforce provides the governor limit. If the governor limit is exceeded then this limit exception is occurred.

1. TOO MANY DML STATEMENTS: 151:

The error occurs when your Apex code exceeds the maximum number of allowed DML statements in a single transaction. Salesforce enforces governor limits to ensure fair resource usage among all the executing transactions. The limit for DML statements is 150.

```
Public class ExceptionExamples {  
    public static void exceptionError(){  
        for(integer i= 0; i<151; i++){  
            Contact con = new Contact();  
            con.LastName = 'SFDC'+ i;  
            insert con;  
        }  
    }  
}
```

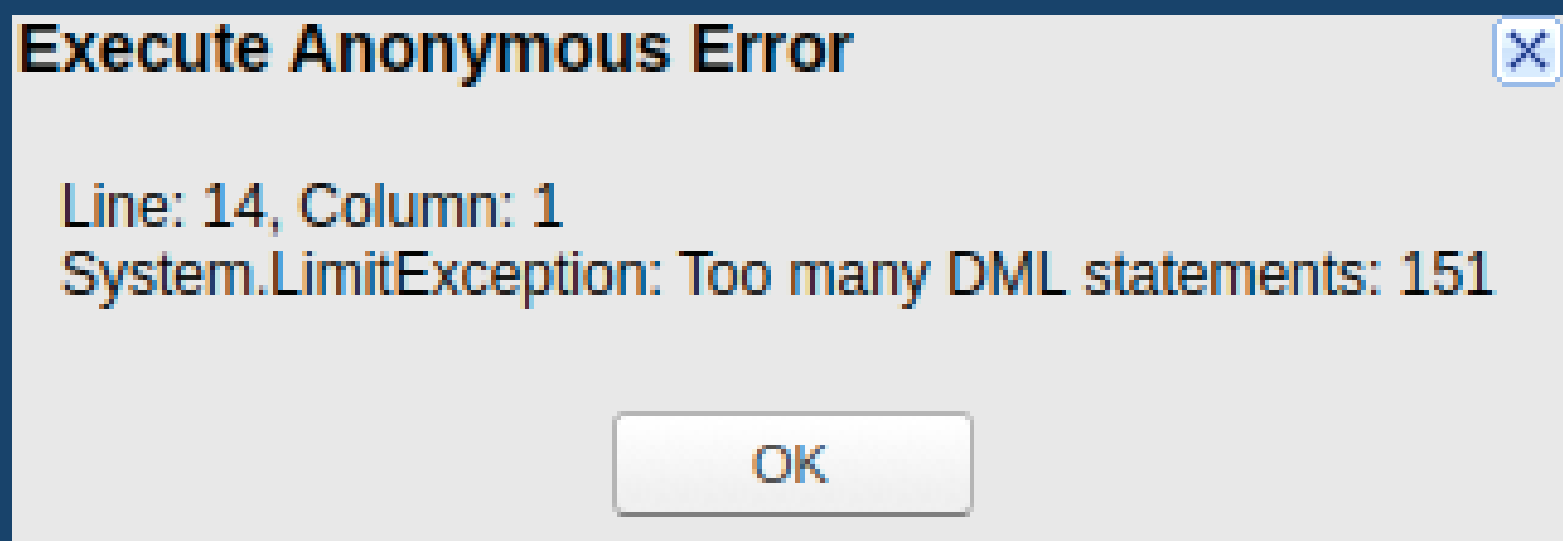
KEEP SWIPING

Nripesh Kumar Joshi

LimitException:

1. TOO MANY DML STATEMENTS: 151:

In above code limitException error is occurred that is showing below.



To avoid the "Too many DML statements: 151" error, we can bulkify your code by inserting the contacts outside of the loop.

KEEP SWIPING

Nripesh Kumar Joshi

Limit Exception:

1. TOO MANY DML STATEMENTS: 151:

```
Public class ExceptionExamples {  
  
    public static void exceptionError(){  
        List<Contact> contactList = new List<Contact>();  
        for(integer i= 0; i<151; i++){  
            Contact con = new Contact();  
            con.LastName = 'SFDC'+ i;  
            contactList.add(con);  
  
        }  
        if(contactList.size()>0){  
            insert contactList;  
        }  
  
    }  
}
```

In above code error will not occur.

KEEP SWIPING

Limit Exception:

1. TOO MANY SOQL QUERIES: 101:

This error in Salesforce occurs when your Apex code or triggers execute more than the maximum allowed number of SOQL queries in a single transaction. The governor limit for SOQL queries in a transaction is 100.

```
Public class ExceptionExamples {  
  
    public static void exceptionError(){  
        List<Contact> contactList = new List<Contact>();  
        for(integer i= 0; i<101; i++){  
            List<Contact> con = [SELECT Id, Name FROM  
Contact WHERE Name Like '%contact%'];  
            system.debug('Exception'+con[0].Name);  
        }  
    }  
}
```



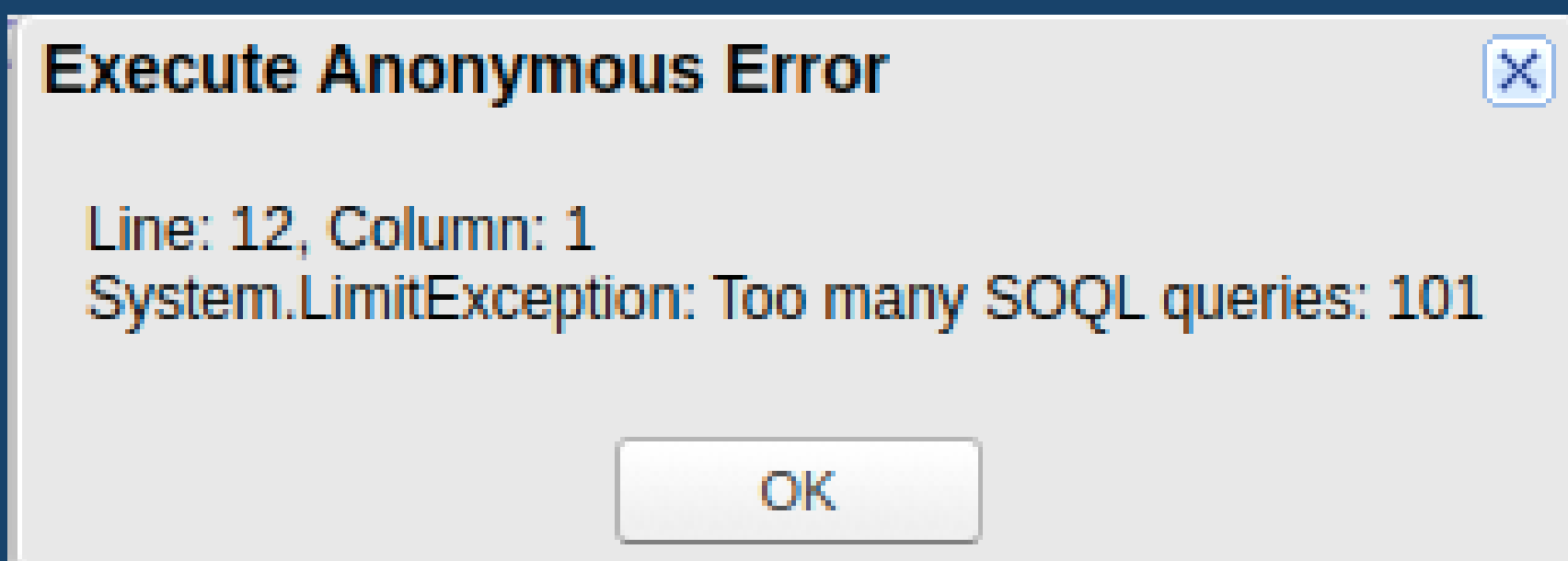
KEEP SWIPING

Nripesh Kumar Joshi

Limit Exception:

1. TOO MANY SOQL QUERIES: 101:

In above code limitException error is occurred that is showing below.



To avoid this error we need to move the soql query outside the for loop to avoid the too many soql query in apex code.

ONE MORE

Nripesh Kumar Joshi

LimitException:

1. TOO MANY SOQL QUERIES: 101:

```
Public class ExceptionExamples {  
  
    public static void exceptionError(){  
        // Move the SOQL query outside the loop  
        List<Contact> con = [SELECT Id, Name FROM  
Contact WHERE Name Like '%contact%'];  
        // Process the results in the loop  
        for (integer i = 0; i < 101; i++){  
            // You can use the con variable to access  
the query results  
            system.debug('Exception ' + con[0].Name);  
        }  
    }  
}
```

In above code error will not occur.



KEEP SWIPING

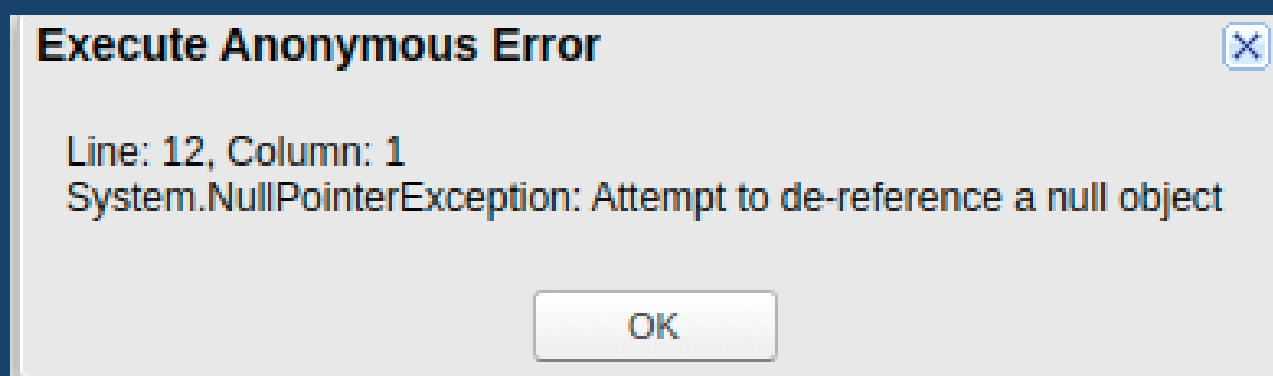
Nripesh Kumar Joshi

NullPointerException:

A NullPointerException in Apex occurs when we attempt to access a null object reference. This error indicates that our code is trying to perform an operation on an object that is not instantiated

```
Public class ExceptionExamples {  
  
    public static void exceptionError(){  
  
        Account acc;  
        System.debug('NullPointerException'+acc.Name);  
    }  
}
```

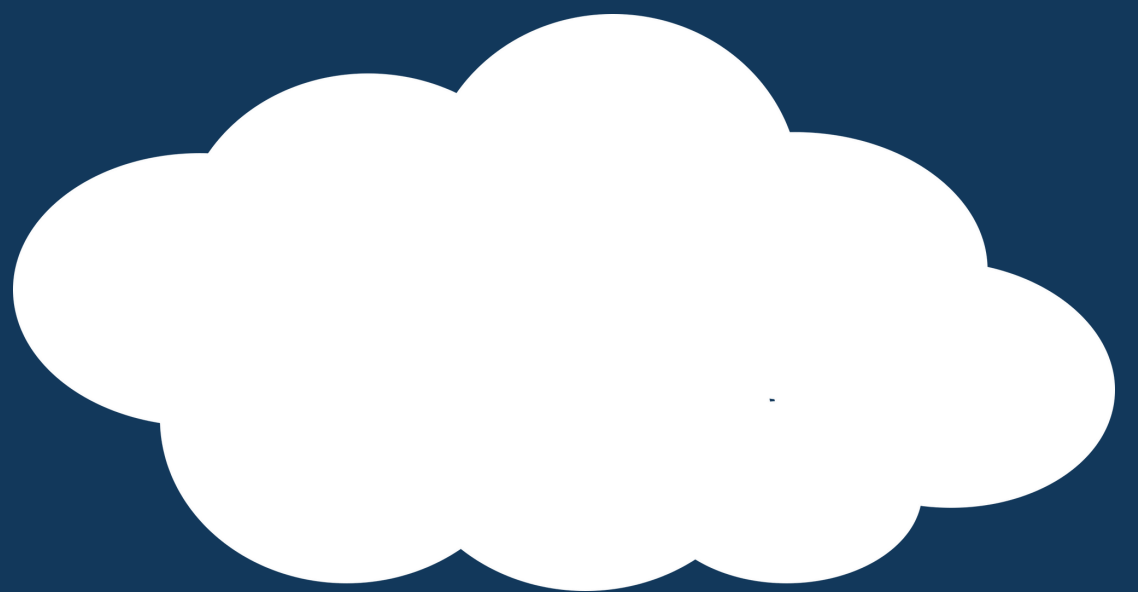
In above code NullPointerException error is occurred that is showing below.



To avoid this error, we need to assign the value before accessing in our code.

KEEP SWIPING

Stay tuned for more updates



COMMENT BELOW

Nripesh Kumar Joshi