<u>**DML OPERATIONS**</u>

DML ----> Data Manipulation Language.

Using DML operations we can manage one or more records inside the salesforce objects.

We can perform the operations on both "Parent and Child Records" at a time.

Apex provides below 2 ways to perform the DML Operations.

1. By using DML Statements

Apex provides the below 6 DML Statements to perform the operations.

1. INSERT

2. UPDATE

3. DELETE

4. UNDELETE

5. UPSERT

6. MERGE

2. By using database class methods

Database Class provides a set of readymade methods to perform the DML Operations on the records.

1. Database.Insert()

2. Database.Update()

3. Database.Delete()

4. Database.UnDelete()

5. Database.Upsert()

6. Database.Merge()

7. Database.EmptyRecycleBin()

Governor Limits

1. We can use max. of 150 DML Statements per Transaction.

When the user tries to use more than 150 DML Statements within the transaction, Salesforce will raise an exception "System.LimitException. Too Many DML Statements: 151".

2. Each DML Statement can process maximum of 10,000 Records.

Best Practices

1. It is always recommended to avoid the usage of DML Statements inside the "FOR Loop".

2. Always follow the bulkification process to perform the operations on bulk records.

INSERT Operation:

By using this Statement, we can insert either one/more records inside the object.

Syntax:

<u>Insert Only One Record:</u>

Insert <objectName / referenceName>;

<u>Insert Multiple Records:</u>

Insert <collectionObjectName>;

<u>By using Database Class Method:</u>

Database.SaveResult[] Database.Insert(<collectionObjectName>, Boolean);

Use Case: Write an apex Class to insert a lead record inside the object.

Class Code:

```
public class DMLOperationsHelper
{
        Public static void CreateLeadRecord()
   {
       Lead ldRecord = new Lead ();

        ldrecord.FirstName = 'Vishnu';
        ldRecord.LastName = 'Prasad';
        ldRecord.Company = 'Cognizant Inc.';
        ldrecord.Status = 'Open - Not Contacted';
        ldRecord.Title = 'Test Engineer';
        ldRecord.Rating = 'Hot';
        ldRecord.Industry = 'Technology';
        ldRecord.AnnualRevenue = 1900000;
        ldRecord.Phone = '9900445577';
        ldrecord.Fax = '9900445577';
        ldRecord.LeadSource = 'Web';
        ldRecord.Email = 'vishnu@gmail.com';
        ldRecord.Website = 'www.gmail.com';
        ldrecord.City = 'Bangalore';
        ldRecord.State = 'Karnataka';
```

```
           ldrecord.Country = 'India';
           Insert ldRecord;
            if(ldRecord.Id != Null)
            System.debug('Lead Record Inserted Successfully with Id ....: '+ ldRecord.Id);
    }
}
```

Execution:

```
DMLOperationsHelper.CreateLeadRecord();
```

Use Case: Write an apex program, to insert 200 Hiring Manager Records inside the Object.

Class Code:

```
public class DMLOperationsHelper
{
    Public static void InsertBulkHRRecords()
    {
        List<Hiring_Manager__c> lstHRRecords = new List<Hiring_Manager__c>();

        for (Integer counter = 1; counter <= 200; counter++)
        {
            // Prepare the Record...
            Hiring_Manager__C hrRecord = new Hiring_Manager__C();

                    hrRecord.Name = 'Test HR Record - '+ counter.
                    hrRecord.Location__c = 'Chennai';
                    hrRecord.Contact_Number__c = '9900445599';
                    hrRecord.Email_Address__c = 'testhr'+counter+'@gmail.com';
                    hrRecord.Designation__c = 'HR Executive';
                    hrRecord.Current_CTC__c = 1200000;
            // Add the record to the Collection.
                    lstHRRecords.Add(hrRecord);
        }
        if (! lstHRRecords.isEmpty())
```

```
            Insert lstHRRecords;
    }
}
```

Execution:

DMLOperationsHelper.InsertBulkHRRecords();

<u>DELETE STATEMENT</u>

By using DELETE Statement, we can remove either one/more records from the Salesforce Object.

Note: To remove the records, we need the "Record ID". Deleted records will get resides inside the RecycleBin for next 15 Days.

Syntax:

<u>Delete Only One Record</u>

Delete <objectName / referenceName>;

<u>Delete Multiple Records</u>

Delete <collectionObjectName>;

<u>By using Database Class Method</u>

Database.DeleteResult[]   Database.Delete(<collectionObjectName>, Boolean);

Use Case: Write an apex program, to remove all the Hiring Manager Records from the object, whose names start with the specified characters at runtime.

<u>Class Code</u>

```
public class DMLOperationsHelper
{
    Public static void DeleteHRRecords(String startingChars)
    {
        /*
        if(startingChars != Null && startingChars != '')
        {
            startingChars += '%';

            List<Hiring_Manager__C> lstHRRecords = [Select id, name from hiring_manager__C
                                                                    Where name like: startingChars];
            if(! lstHRRecords.isEmpty())
                Delete lstHRRecords;
        }
                    */


        Delete [Select id, name from Hiring_Manager__C
                                             Where name like : startingChars+'%'];
    }
}
```

<u>Execution</u>

DMLOperationsHelper.DeleteHRRecords('te');

UNDELETE STATEMENT

By using this feature, we can re-store the deleted records back to the actual object.

We can restore either One / Multiple / All the deleted records back to the object by adding the user defined conditions.

Syntax:

<u>Re-Store Only One Record</u>

UnDelete <objectName / referenceName>;

<u>Re-Store Multiple Records</u>

UnDelete <collectionObjectName>;

<u>By using Database Class Methods:</u>

Database.UnDeleteResult[] Database.UnDelete(<collectionObjectName>, Boolean);

Use Case: Write an apex program, to re-store the Hiring Manager Records back to the object, based on the specified characters at runtime.

<u>Class Code:</u>

```
public class DMLOperationsHelper
{
    Public static void RestoreHRRecords(String startingChars)
    {
        if(startingChars != Null && startingChars != '')
        {
            UnDelete [Select id, Name, isDeleted  from Hiring_Manager__C
```

```
                 Where isDeleted = true and name like: startingChars+'%'    ALL ROWS];
      }
   }
}
```

Execution:

```
DMLOperationsHelper.RestoreHRRecords('Test HR');
```

UPDATE STATEMENT

By using Update Statement, we can Update either one or more records inside the object. i.e., We can assign the new values for the required fields in the records.

Note: While updating the records, the record should have the "Record Id".

Syntax:

Update Only One Record

Update <objectName / referenceName>;

Update Multiple Records

Update <collectionObjectName>;

By using Database Class Methods

Database.SaveResult[] Database.Update(<collectionObjectName>, Boolean);

Note: While updating the records, we must follow the below steps.

Step 1: Get the Required records to be get updated by using "SOQL Query" and store them to a Collection.

Ex:

```
List<Account> lstAccounts = [Select id, name, rating, industry, annualrevenue, active__C

                         from Account  Where industry = 'Insurance'];
```

Step 2: Assign the New Values for the required fields, by iterating the collection by using "FOR Loop".

```
                 Ex:   if(! lstAccounts.isEmpty())
                         {
                                 for(Account accRecord : lstAccounts)
                                 {
                                         accRecord.Rating = 'Hot';
```

```
                                        accRecord.AnnualRevenue = 5000000;
                              }
                    }
```

Step 3: Once all the records has been assigned with New Values, then update the collection back to the Database by using "Update Statement".

Ex: Update lstAccounts;

Use Case: Write an Apex Program, to Update the Banking Industry Customer Records as below

Account:Rating = 'Hot'

Account:AnnualRevenue = 6000000

Account:Type = 'Customer - Direct'

Account:CustomerPriority__C = 'High'

Class Code:

```
public class DMLOperationsHelper
{
   Public static void UpdateCustomerDetails()
   {
      // Get the records to be get updated.
      List<Account> lstAccounts = [Select id, name, rating, industry, annualrevenue,
                        type, customerpriority__c from Account Where Industry = 'Banking'];
      if (! lstAccounts.isEmpty())
      {
         // Iterate the Collection and Assign New Values for the fields.
        for (Account accRecord : lstAccounts)
        {
           accRecord.Rating = 'Hot';
           accRecord.AnnualRevenue = 6000000;
           accRecord.Type = 'Customer - Direct';
           accRecord.CustomerPriority__c = 'High';
        }
         // Commit the Changes to Database.
         Update lstAccounts;
        }
    }
}
```

Execution:

DMLOperationsHelper.UpdateCustomerDetails();

Use Case: Write an apex program, to de-activate the user based on the specified username at runtime.

<u>Class Code:</u>

```
public class DMLOperationsHelper
{
    Public static void DeActivateUser(String uName)
    {
        if(uName != Null && uName != '')
        {
            User userToDeActivate =     [Select id, firstName, lastName, username, isActive
                                        from User Where userName =: uName and isActive = true];
            if(userToDeActivate.id != Null)
            {
                userToDeActivate.IsActive = false;
                Update userToDeActivate;
            }
        }
    }
}
```

Execution

DMLOperationsHelper.DeActivateUser('manager@dl.com');

Use Case: Write an apex program, to Transfer All the Accounts to the "Development User" based on the specified industry name at runtime.

<u>Class Code:</u>

```
public class DMLOperationsHelper
{
    Public static void TransferAccountsByIndustry(String industryName)
    {
        if (industryName != Null && industryName != '')
        {
           List<Account> lstAccounts = [Select id, name, rating, industry, ownerid
                                               from Account  Where industry =: industryName ];

            User devUser = [Select id, firstname, lastname, username, isActive
                        from User Where username = 'developmentuser@dl.com' and  isActive = true];
            if((! lstAccounts.isEmpty()) && devUser.Id != Null)
            {
                    // Transfer the Records.
                    for (Account acc : lstAccounts)
                {
                    acc.OwnerId = devUser.Id;
                }

                Update lstAccounts;
            }
        }
    }
}
```

Execution:

DMLOperationsHelper.TransferAccountsByIndustry('Finance');

Use Case: Write an apex program to synchronize the Account Records Phone, Fax, and Address Details to the Related Contact Records based on the Account Name supplied at runtime.

Class Code:

```
public class DMLOperationsHelper
{
    Public static void SyncAccountDetailsToContacts(String accountRecordName)
    {
        if (accountRecordName != Null && accountRecordName != '')
    {
Map<ID, Account> mapAccounts = new Map<ID, Account>([Select id, name, phone, fax,
                            billingCity, billingState, billingCountry, billingPostalCode, billingStreet
                                    from Account Where name =: accountRecordName]);
        if(! mapAccounts.isEmpty())
        {
            // Get the Related Contacts.
            List<Contact> lstContacts = [Select id, firstname, lastname, phone, fax,
                            mailingStreet, mailingCity, mailingState,
                        mailingCountry, mailingPostalCode, accountid
                        from Contact    Where accountid IN : mapAccounts.KeySet()];
            if (! lstContacts.isEmpty())
            {
                for (Contact con: lstContacts)
                {
                    // Get the Related Parent Account for contact
                  Account accRecord = mapAccounts.Get(con.AccountID);
                    // Sync the Data from Account to Contact
                        con.Phone = accRecord.Phone;
                        con.Fax = accRecord.Fax;
                        con.MailingStreet = accRecord.BillingStreet;
                        con.MailingCity = accRecord.BillingCity;
                        con.MailingState = accrecord.BillingState;
                        con.MailingCountry = accRecord.BillingCountry;
                        con.MailingPostalCode = accRecord.BillingPostalCode;
                }

                if (! lstContacts.isEmpty())
                    Update lstContacts;
            }
        }
        }
    }
}
```

Execution:

DMLOperationsHelper.SyncAccountDetailsToContacts('United Oil & Gas Corp.');

Upsert Statement: By using Upsert Statement, we can perform both Insert and Update operations on the records at a time, by using a single DML Statement.

1. It will differentiate the records based on the "Record ID".

2. If the Record contains the ID, it will update to database.

3. If the record doesn't have the ID, it will insert the record in the database.

Syntax:

Upsert Only One Record

Upsert <objectName / referenceName>;

Upsert Multiple Records

Upsert <collectionObjectName>;

By using Database Class Methods

Database.UpsertResult[]   Database.Upsert(<collectionObjectName>, Boolean);

Use Case: Write an apex program to Update all the New Cases Priority as "High" and create a new case record inside the object.

Class Code:

```
public class DMLOperationsHelper
{
    Public static void UpsertCaseRecords()
    {
        List<Case> casesToUpsert = new List<Case>();
        // Update the Existing Case Records
          List<Case> lstCases = [Select id, caseNumber, status, priority
                                          from Case Where status = 'New'];
          if(! lstCases.isEmpty())
           {
              for(Case csRecord : lstCases)
              {
                csRecord.Priority = 'High';
                // Add the Case to Collection
                casesToUpsert.Add(csrecord);
            }
        }
     // Write the Code to Insert a Case Record
       Case csRecord = new Case();
                csRecord.Status = 'Working';
                csRecord.Priority = 'High';
                csRecord.Origin = 'Web';
                csRecord.Type = 'Mechanical';
                csRecord.Reason = 'Performance';
                csRecord.Subject = 'My Newly Purchased Tab Device is unable to connect to Wi-
fi.';
                csRecord.Description = 'Dear Customer Support, '+
                                'My Newly purchased Tab Device is unable to connect to wi-fi.'+
                                'Please have a look into the issue and close it ASAP. '+
                                'Thanks & Regards, Ram Kumar.';
       // add the record to collection
                casesToUpsert.Add(csRecord);
       // Perform the Upsert Operation
       if(! casesToUpsert.isEmpty())
       {
           Upsert casesToUpsert;
       }
   }
}
```

Execution:

DMLOperationsHelper.UpsertCaseRecords();

Merge Statement: Using this statement, we can merge up to three records of same Sobject type.

Merges up to three records into one of the records, deletes the others and reparents any related records.

Can be applied only on account, lead, contact and case object records in salesforce.

Syntax: Merge<target/parent record><source/child record>;

Merge<target record> List<source records>;

Use Case: Write an Apex Program to merge two account records.

```
public class DMLOperationshelper
{
Public static void MergeRecords()
 {
Account facc= [Select id, name, rating, industry, annualrevenue from account where name=
'New account1' limit 1];
Account sacc= [Select id, name, rating, industry, annualrevenue from account where name=
'New account2' limit 1];
System.debug('First Account record is...'+ facc);
System.debug('First Account record is..'+ sacc);
If(facc!= null && sacc!=null)
{
Merge facc sacc;
}
}
```

Execution:

```
DMLOperationsHelper.MergeRecords();
```

<u>Database Class Methods</u>

1. DML Statements are Atomic Statements (i.e., Each DML Statement uses Implicit Transaction by default. Hence if any of the record operation fails, the whole transaction will get Rolled back.)

2. DML Statements won't support Partial Processing i.e., if any of the record operation fails, it will ignore the record and will continue processing the rest of the records.

3. DML Statements won't provide the tracking mechanism to track each record operation result.

All the above drawbacks can be overcome using Database class methods.

Database.Insert(): By using this method, we can insert either one/more records inside the object. While inserting the records, we can allow Partial Processing.

It provides the tracking mechanism, to track the record operation result.

Syntax: Database.SaveResult[] Database.Insert(<collectionName>, Boolean);

We can enable the partial processing.

TRUE ---> Maintains the Transaction

FALSE ---> Allows the Partial Processing

Database.SaveResult Class: This Class is used to store the result of a record upon insertion. To store the multiple records results we have to create either Array/List collection of "Database.SaveResult" class.

Ex: Database.SaveResult ----> One Record Result.

List<Database.SaveResult> / Database.SaveResult[] ----> Multiple Records Result

To track each record result, use the below methods.

1. Boolean IsSuccess(): It returns TRUE, if the record is inserted successfully, else it returns FALSE.

2. GetID(): It returns the Record Id generated by Salesforce once the record inserted.

3. Database.Error GetErrors(): It returns the error information, if the record operation fails. It returns the error in the form of "Database.Error" class.

Database.Error Class: This Class is used to store the errors has been occurred, while processing the records inside the object.

We have the below methods, to track the Error details.

1. GetMessage(): It returns the Error Message generated by Apex runtime.

2. GetStatusCode(): It returns the Error Status Code.

REQUIRED_FIELDS_MISSING, CUSTOM_VALIDATION_RULES, FIELD_FILTER_VALIDATION, DUPLICATES_DETECTED, MIXED_DML_OPERATION etc.

3. GetFields(): It returns the fields information, which causes the error.

Use Case: Write an apex program to insert multiple hiring manager records inside the object.

1. Allow the Partial Processing, while processing the records.

2. Track the record operation results and represent on debug log file.

Class Code:

```
public class DMLOperationsHelper

{

   Public static void CreateBulkHRRecords()

   {

      List<Hiring_Manager__c> lstHRRecords = new List<Hiring_Manager__c>();

      for(Integer counter = 1; counter <= 50; counter++)

      {
```

```
    // Prepare the Record
    Hiring_Manager__C hrRecord = new Hiring_Manager__C();
            hrRecord.Name = 'Bulk HR - '+ counter;
            hrRecord.Location__c = 'Bangalore';
            hrRecord.Contact_Number__c = '9955667766';
            if(counter != 45)
       {
            hrRecord.Email_Address__c = 'bulkhr'+counter+'@gmail.com';
       }
            hrRecord.Designation__c = 'Recruitment Specialist';
            hrRecord.Current_CTC__c = 1500000;
    // Add the record to the Collection
            lstHRRecords.Add(hrRecord);
}
if(! lstHRRecords.isEmpty())
{
   Database.SaveResult[] results = Database.insert(lstHRRecords, false);
    for( Database.SaveResult recResult : results)
    {
       if(recResult.isSuccess())
       {
          System.debug('Record has been Inserted Successfully.');
          System.debug('Record Id is: '+ recResult.getId());
       }
       else
       {
          Database.Error[] errors = recResult.getErrors();
          for(Database.Error err : errors)
          {
             System.debug('Error Message is....: '+ err.getMessage());
             System.debug('Error Status Code is...: '+ err.getStatusCode());
```

```
                        System.debug('Effected Fields are....: '+ err.getFields());
                    }
                }
            }
        }
    }
}
```

Execution: DMLOperationsHelper.CreateBulkHRRecords();