Session 4
SOQL and SOSL

Quering data in our salesforce instance, to fetch the data which is present in our system.suppose if we have Standard objects like Account,Contacts,Oppurnity and custom objects,if we want to get the data with SOQL.

SOQL stands for Salesforce Object Query Language

- By using SOQL queries, we can retrieve the records of Salesforce objects.
- Upon fetching the record, we need to specify which fields to be retrieve.
- Upon fetching the records, we can apply user defined condition, to filter the records.
-All records of objects(Standard & Custom) are stored in "Database.com".
- It is nothing but a request to "Database.com", to retrieve required information from object.

You can query data from Salesforce using SOQL in:
Apex code
Developer Cosole
Salesforce REST and SOAP APIs
Workbench

- Syntax:
SELECT <columnNames>
FROM <objectName>
[WHERE <condition>]
[GROUP BY <columnNames>]
[HAVING <conditions>]
[ORDER BY <columnNames>]
[LIMIT <numberOfRecordsToReturn>]
[OFFSET <recordsToSkip>]

- Only "SELECT" clause and "FROM" clauses are manadatory clauses, all other clauses are optional.
- Field API name must be used in "SELECT" clause.

Examples:

1) SELECT Id, Name, BillingCity FROM Account Limit 5

2) SELECT Id,Name FROM Contact WHERE MailingCity = 'California'

3) Account[] accts = [SELECT Name,Phone FROM Account
WHERE (Name='SFDC Computing' AND NumberOfEmployees>25)
ORDER BY Name
LIMIT 10];

System.debug(accts.size() + ' account(s) returned.');
System.debug(accts);
4) `SELECT Name FROM Account` WHERE Name LIKE 'A%'
5) `SELECT Id FROM Contact` WHERE Name LIKE 'A%' AND MailingState='California'

You can use date or datetime values, or date literals. The format for date and dateTime fields are different.

- `SELECT Name FROM Account` WHERE CreatedDate > 2011-04-26T10:00:00-08:00
- `SELECT Name,Amount FROM Opportunity` WHERE CALENDAR_YEAR(CreatedDate) = 2021

Accessing Variables in SOQL Queries:

String targetDepartment = 'Wingo';
Contact[] conlist = [SELECT FirstName,LastName,department FROM Contact WHERE Department=:targetDepartment];
system.debug(conlist);

ORDER BY Clause:
- This clause is used to arrange resultset records in either ascending order or descending order.
- "ASC" is used for ascending order, whereas, "DESC" is used for descending order, used along with ORDER BY Clause.
- If user don't specify any sorting order, Salesforce arranges the records in ascending order by default.

1) SELECT Name,Phone FROM Account ORDER BY Name Limit 5

2) SELECT Name,Phone FROM Account ORDER BY Name ASC Limit 5

2) SELECT Name,Phone FROM Account ORDER BY Name DESC Limit 5

OFFSET Clause:

- This clause is used to specify number of columns to be get skipped from resultset.

SELECT Name FROM Merchandise__c WHERE Price__c > 5.0 ORDER BY Name LIMIT 100 OFFSET 10

The result set for the preceding example would be a subset of the full result set, returning rows 11 through 110 of the full set.

GROUP BY Clause:

- This clause is used to group the records based on one or more columns specified by user.
- This clause is used with "Aggregate Functions".

SELECT Industry,count(id) From Account GROUP BY Industry

Aggregate Functions:

- Aggregate functions takes set of values as an input, perform operation of it and return one value as a result.
- Apex supports below aggregate functions:

1. COUNT()
- This function returns number of records exist in an object.
- "Database.countQuery()" method is used, if this function is executed using dynamic query.
- COUNT(<columnName>) function counts the values exist in specified column, excluding "NULL" values.
- COUNT_DISTINCT(<columnName>) function counts the values exist in specified column, excluding "NULL" and "Duplicate" values.

Example:
select count() from student__c

2. SUM(<columnName>):
- This function returns sum of all values exist in specified column.
- This function can be used only with "numeric" values.
- Specified column type should be "Number", "Currency" and "Percent" type.

Example:
select sum(Amount__C) from vyom_labs_sales__c

3. AVG(<columnName>):
- This function returns average of all values exist in specified column.
- This function can be used only with "numeric" values.- Specified column type should be "Number", "Currency" and "Percent" type.

Example:
select AVG(Amount__C) from vyom_labs_sales__c

4. MIN(<columnName>):
- This function returns smallest value of all values exist in specified column.
- This function can be used only with "numeric" and "Date" values.
- Specified column type should be "Number", "Currency", "Percent" and "Date" type.

Example:
select MIN(Amount__C) from vyom_labs_sales__c

5. MAX(<columnName>):
- This function returns largest value of all values exist in specified column.
- This function can be used only with "numeric" and "Date" values.
- Specified column type should be "Number", "Currency", "Percent" and "Date" type

Example:
select MAX(Amount__C) from vyom_labs_sales__c

HAVING Clause:

- This clause is used to apply filter, after grouping, using "GROUP BY" clause.
- "HAVING" clause should be used along with "GROUP BY" clause.
- "WHERE" clause is used to apply filter before grouping, whereas, "HAVING" clause is used to apply filter after grouping.
- "WHERE" and "HAVING", both clauses can be used in single SOQL query.

Example:

SELECT Industry, COUNT(Id) From Account GROUP BY Industry HAVING Industry='Construction'

Conditional Expressions:

Conditional Operator	Description	Query
=	Equals	select Name,closedate from opportunity where name='New Opportunity created'
!=	Not Equals	select Name,closedate from opportunity where name!='New Opportunity created'Select salary__c from employee__c where salary__c<=20000
< , <=	Less Than, Less Than or Equal To	Select salary__ c from employee__c where salary__c<=20000
> , >=	Greater Than, Greater Than or Equal To	Select salary__ from employee__c where salary>=20000
INCLUDES, EXCLUDES	Includes or Excludes Values. Applies only to Multi-Select Picklists.	Select name, country__c, Student_skills__c from Student__C where student_skills__C includes (‘salesforce’)
LIKE	Returns the records where the field values match the pattern specified after the LIKE operator	select Name,closedate from opportunity where name like '%united%'
IN	Selects only the records where the value of the field matches any of the values specified after IN keyword	select FirstName,LastName from contact where FirstName in ('rose','sean')
NOT IN	Selects only the records where the value of the DOES NOT MATCH any of the values specified after IN keyword	select FirstName,LastName from contact where FirstName not in ('rose','sean')

ALL ROWS Clause:

- This clause is used to fetch all the records existing in object, along with all "deleted" records.
- This clause should be the "last clause" in SOQL query.
- For each object, Salesforce maintains a hidden field, "isDeleted", which is a "Checkbox" field.

- "isDeleted" checkbox is checked only when record is deleted from object.

List<student__c> deletedstud = [select id, name from student__c where isDeleted = true ALL ROWS];

system.debug(deletedstud);

Relationship Queries:

- By using relationship queries, we can fetch records from multiple objects which are associated with each other by either "Lookup" association or "Master-Detail" association.
- By using relationship queries, we can reduce number of SOQL queries inside an application.

- Approaches:

1. Parent to Child (Standard Objects):
- Parent to Child relationship queries will always be "Inner Queries".
- Inside the inner query, child object name should always be referenced with "Plural Name" of an object.
- Outer query will fetch parent object records and inner queries fetch child objects records.

Example:
SELECT Name, (SELECT LastName FROM Contacts) FROM Account WHERE Name = 'SFDC Computing'

2. Child to Parent (Standard Objects):

- Upon fetching child object record, we can fetch its associated parent, even grandparent records also.
- To fetch parent object record details, parent object field should be refenced with, "ParentObjectName.fieldName".
- We can refer upto 10 reference objects in a single SOQL query.ie, upto 10th grandparent record details.

Example:
select Name,Account.Name,Account.Type from Opportunity

3. Parent to Child (Custom Objects):
- Parent to child relationship query will always be inner query.
- Outer query will fetch parent object records and inner queries fetch child objects records.
- Inside inner query, child object name should always be "ChildObjectPluralName__r" of child object.

Example:
select name,(select name,age__c,salary__c from employees__r) from department__c

4. Child to Parent (Custom Objects):
- Upon fetching child object record, we can fetch its associated parent, even grandparent records also.
- To fetch parent object record details, parent object field should be refenced with relationship name, ie,"ParentObjectName__r.fieldName".

Example:

1) select name, professor__r.name from student__c

What is "Too Many SOQL Queries: 101" exception:

- Upon executing the application, if more than 100 SOQL queries have been executed inside a transaction,
then Salesforce aborts the operation and throws the exception, "Too Many SOQL Queries: 101".

DML Statements:

- Apex provides below DML Statements:
1. Insert
2. Update
3. Delete
4. Upsert
5. Undelete

Governor Limits for DML Statements:

1. We can have maximum 150 DML statements inside a transaction.
2. By using each DML statement, we can process maximum 10,000 records at a time.

Insert Statement:

- By using "Insert" statement, we can insert one or more records in an associated object.
- Object can be standard object or custom object.

Syntax:
insert sObject
insert sObject[]
Example:

```
Student__c s=new Student__C(Name='Gajanan',
                    Age__C=35,City__c='Pune'
                    ,Class__c='TY',Professor__c='a0B5g000000j35s');
try{
insert s;
}
catch(DmlException e){}
```

Update Statement:

- By using "Update" statement, we can modify one or more records of an object.
- Upon upating the records, we have to fetch the records from an object using SOQL query,assign new values to desired fields of the records and commit the statement.

Syntax:
update sObject

update sObject[]

Example:

1) Opportunity op=[select id from Opportunity where Name = 'New Opportunity created'];
op.Amount=1000;
update op;

2) Student__c s=[select id from student__C where name='shashi'];
s.age__c=35;
s.city__c='Ahmednagar';
update s;

Delete Statement:
- By using "Delete" statement, we can delete one or multiple records from specified object.
- Upon deleting the records from an object, "Record ID" should be used.
- All deleted records are stored in "RecycleBin" for 15 days.

Syntax:
delete sObject

delete sObject[]

1) List<Student__c> studlist=[select id from student__c where Name='shashi'];
delete studlist;
if(!studlist.isEmpty()){
  System.debug('student record is deleted');
}
2) Student__c[] deleteStudent=[select id from student__c where Name='gajanan'];

try{
delete deleteStudent;
}
catch(DmlException e){}

Undelete Statement:

- By using "Undelete" statement, we can bring back deleted records from "Recycle Bin" to actual object.
- We can restore one or more records from "Recycle Bin" to actual object.

Syntax:
undelete sObject | ID
undelete sObject[] | ID

Example:
The following example undeletes an account named 'Universal Containers'. The `ALL ROWS` keyword queries all rows for both top level and aggregate relationships, including deleted records and archived activities.

```
Account a = new Account(Name='Dummy Test Account01');

insert(a);

insert(new Contact(LastName='Carter',AccountId=a.Id));

delete a;

Account[] savedAccts = [SELECT Id, Name FROM Account WHERE Name ='Dummy Test Account01' ALL ROWS];


undelete savedAccts;

system.debug(savedAccts);
```

Upsert Statement:

- It is a combination of "Insert" and "Update" statements.
- We can perform "Insert" and "Update" operation by using single DML statement.
- It will "Update" the records if "Record ID" is exist, else, it will "Insert" the record in an object.

Syntax:

```
upsert sObject | sObject[]

upsert sObject | sObject[] field
```

Example:

```
Contact josh = new Contact(FirstName='Josh',LastName='Kaplan',Department='Finance');
insert josh;

josh.Description = 'Josh's record has been updated by the upsert operation.';

Contact kathy = new Contact(FirstName='Kathy',LastName='Brown',Department='Technology');

List<Contact> contacts = new List<Contact> { josh, kathy };

upsert contacts;
```

Example:

```
Account[] acctsList = [SELECT Id, Name, BillingCity FROM Account WHERE BillingCity = 'Kolkata'];
for (Account a : acctsList){
      a.BillingCity = 'Mumbai';
}

Account newAcct = new Account(Name = 'Vyom Labs', BillingCity = 'Hyderabad');
```

```
acctsList.add(newAcct);

try {
        upsert acctsList;
}
catch (DmlException e)
```

DML 151 Exception:

"System.LimitException: Too many DML statements: 151" Exception:

- When we try to perform more than 150 DML statements inside single transction, then Salesforce aborts the operation and throw "Too many DML statements: 151" exception.

Bulkification Process / Bulkify:

- By using this feature, we can process bulk of records at a time.
- Instead of inserting each record one by one using seperate DML statement, we can group all records into a collection and perform DML operation directly on collection.

```
List<Contact> conList = new List<Contact> {
    new Contact(FirstName='suresh',LastName='Shinde',Department='Finance'),
        new Contact(FirstName='rajesh',LastName='Varma',Department='Technology'),
        new Contact(FirstName='Sachin',LastName='Sharma',Department='Finance'),
        new Contact(FirstName='Ganesh',LastName='Mishra',Department='Finance')};

insert conList;

List<Contact> listToUpdate = new List<Contact>();

for(Contact con : conList) {
    if (con.Department == 'Finance') {
        con.Title = 'Financial analyst';

        listToUpdate.add(con);
    }
}
update listToUpdate;
```

Limitations of DML Statements:

1. Each DML statement is used implicit transaction by default, which allows to commit transaction only if, all operations are completed successfully.If any operation gets failed, then entire transaction gets rolled back.
2. There is no tracking mechanism to track which records got failed in DML transaction.
3. DML statements do not allow "Partial Processing" of records. ie, if any of record got failed, it doesn't allow to process other remaining records.

Types of SOQL Queries:

1. Static SOQL queries
2. Dynamic SOQL queries

Static SOQL queries:

- Query columns and condition are static. ie, upon preparing the query we hava to specify column names and required condition. We cannot include columns or condition at runtime.
- All SOQL queries written inside Apex class, should be placed inside square brackets "[....]".
- All static SOQL queries will be executed automatically.
- If SOQL query returns only one record, then we need to store the result in the object of associated type.
- If SOQL query returns multiple records, then we need to store the result in "List" collection of associated type.

Governor Limits for SOQL queries:

1. Each SOQL query can return maximum 50,000 records.
2. In single transaction, we can have maximum 100 SOQL queries.

Dynamic SOQL Query:

- By using dynamic SOQL query, we can prepare a query by adding columns and conditions at runtime.
- Dynamic SOQL query must be stored in "String" variable, and once query is prepared, we have to execute that query "manually".
- Dynamic SOQL query is executed manually, by using "Database.query(String ‘dynamicQuery’) method.
- Dynamic SOQL query always returns "List" collection of associated type, though query return single record.
- The variable passed as a parameter, must also be enclosed in single quotation mark.

Example:

```
String myTestString = 'Test';
List<sObject> sobjList = Database.query('SELECT Id,Name FROM Account WHERE Name
= :myTestString');
system.debug(sobjList);
```

Example:

```
public class DatabaseHandler {
    public static void getContact(String searchText){
        if(searchText!=NULL && searchText!=''){
            string queryString='select Id,FirstName,LastName,Email from Contact';

            queryString=queryString + ' ' + 'where Lastname=:searchText';
```

```
        List<Contact> contactList=Database.query(queryString);
        if(!contactList.isEmpty()){
           for(contact con:contactlist){
           system.debug('id'+con.id);
           system.debug('First Name'+con.FirstName);
           system.debug('Last Name '+con.LastName);
           system.debug('Email'+con.Email);

           }
        }
        else{
           system.debug('search text is empty');
        }
     }
  }
}
```

DatabaseHandler.getContact('Godrej Company');

Example:

```
public class Dynamic_SOQL{

public static void SOQLMethod(String rec){
List<sObject> acc=Database.query(rec);
System.debug(acc);
}
}
```

Execution:

```
String query='select id,name from account';
Dynamic_SOQL.SOQLMethod(query);
```

Difference between Static Query and Dynamic Query:

1. Static query should be placed in "Square bracket", whereas dynamic query should be stored in String variable.
2. Static query executes automatically, whereas, dynamic query has to be executed manually by using, "Database.query()" method.
3. If static query returns single record, we can store it in associated object, whereas, if dynamic query returns single record, still, we have to store it in "List" collection only.

Database Class Methods For DML Operations:

Unlike DML statements, Database methods have an optional allOrNone parameter that allows you to specify whether the operation should partially succeed. When this parameter is set to false, if errors occur on a partial set of records, the successful records will be committed and errors will be returned for the failed records. Also, no exceptions are thrown with the partial success option.

- Salesforce provides below methods for DML operations:

1. Database.insert()

2. Database.update()
3. Database.delete()
4. Database.undelete()
5. Database.upsert()
6. Database.merge()

- These methods allow us to perform "Partial Processing" of records.
- These methods also provide tracking mechanism to track which records get failed.

1) Database.insert():

- This method is used to insert one or more records in an associated object.
- Syntax:

Database.SaveResult[] result = Database.insert(<collection>, boolean allOrNone);

- The second argument of this method is optional. If we dont provide second argument,or we provide "TRUE" value, then this method works like normal DML statement.
- If we provide second argument value as "FALSE", then this method allows "Partial Processing" of records.
- This method returns array of "Database.SaveResult" class. This class is used to track result of each operation.

Example:

```
List<Account> accounts = new List<Account>();
 accounts.add(new Account(Name = 'Test Account 1001'));
 accounts.add(new Account(Name = ''));
 System.debug('Account List Size : ' +accounts.size());
 try {
 Database.SaveResult[] results = Database.insert(accounts, false);
 } catch(DMLException e) {
//system.debug(e.getMessage());
 throw new DMLException('Unable to Perform the DML Operation on Account : '
+e.getMessage());
 }
```

Example2:

```
Opportunity opp1 =  new Opportunity(
Name='First Opportunity',
    CloseDate=Date.today(),
        Amount=7000,
            StageName='Prospecting'
);
Opportunity opp2 = new Opportunity(
Name='New Opportunity1',
CloseDate=Date.today(),
    Amount=6000,
    StageName='Prospecting'
    );
List<Opportunity> lsr=new List<Opportunity>{opp1,opp2};
    Database.SaveResult[] sr=Database.insert(lsr,false);
```

- Methods of "Database.SaveResult" class:
1. Boolean isSuccess():This method returns "TRUE" if operation is completed successfully, else it returns "FALSE".

2. ID getID():
This method returns "ID" of successfully inserted record.

3. Database.Error[] getErrors():
This method returns array of "Database.Error" class. This class is used to get error information, that has been generated during insertion operation.

- Methods of "Database.Error" class:
1. String getMessage():
This method returns "Error Message" of actual error, which occured during insertion.
2. String getStatusCode():
This method returns "Status Code" of error, which occurred during insertion.
3. Field[] getFields():
This method returns the "Fields", which causes actual exception.

2) Database.update():

- This method is used to update one or more records of an object.
- Syntax:

Database.SaveResult[] Database.update(<collection>, Boolean allOrNone);

3) Database.delete():

- This method is used to delete one or more records of an object.
- Syntax:
Database.DeleteResult[] = Database.delete(<collection>, Boolean allOrNone);

4) Database.undelete():

- This method is used to undelete one or more records from "Recycle Bin" to actual object.
- Syntax:

Database.UndeleteResult[] = Database.undelete(<collection>,Boolean allOrNone);

5) Database.upsert():

- This method is updates a record if "Record ID" is exist, or it inserts new record.
- Syntax:

Database.UpsertResult[] = Database.upsert(<collection>, Boolean allOrNone);

6) Database.emptyRecycleBin():

- This method is used to delete all records from "Recycle Bin" permanently.

- Syntax:

Database.EmptyRecycleBinResult[] = Database.emptyRecycleBin(<collection>, Boolean allOrNone);

Difference Between DML Statements and Database Statements

DML Statements	**Database Methods**
Partial Update is not allowed. For example, if you have 100 records in list, then either all the records will be updated or none.	Partial update is allowed. You can specify the Parameter in Database method as true or false, true to allow the partial update and false for not allowing the same.
You cannot get the list of success and failed records.	You can get the list of success and failed records
ConvertLead Not Available in DML Statements	ConvertLead Operation is only available in Database class.
Example − insert accounts;	**Example** − Database.insert(listName, False), where false indicate that partial update is not allowed.

SOSL Statements:

Salesforce Object Search Language (SOSL) is a Salesforce search language that is used to perform text searches in records. Use SOSL to search fields across multiple standard and custom object records in Salesforce.

The Syntax of Basic SOSL Query
FIND 'SearchQuery' [IN SearchGroup] [RETURNING ObjectsAndFields]

Example:
1) find {sfdc} in all fields returning Account(Name)

2) find {sfdc)} in all fields returning account(Id,name),Lead(Id,Name)

This is an example of a SOSL query that searches for accounts and contacts that have any fields with the word 'SFDC'.

Adding SOSL queries to Apex is simple—you can embed SOSL queries directly in your Apex code. When SOSL is embedded in Apex, it is referred to as inline SOSL.

SOSL statements evaluate to a list of lists of sObjects, where each list contains the search results for a particular sObject type. The result lists are always returned in the same order as they were specified in the SOSL query. If a SOSL query does not return any records for a specified sObject type, the search results include an empty list for that sObject.

Example:

you can return a list of Account and leads that begins with sfdc.

```
List<List<SObject>> searchList=[find 'sfdc*'in all fields returning
Account(Id,Name),Lead(Id,Name)];
Account[] accs=searchList[0];
Lead[] leads=searchList[1];
System.debug('AccountList'+accs);
System.debug('Leads List'+leads);
```

For example, you can return a list of accounts, contacts, opportunities, and leads that begin with the phrase map:

```
List<List<SObject>> searchList = [FIND 'map*' IN ALL FIELDS RETURNING Account (Id,
Name), Contact, Opportunity, Lead];
```

from searchlist, you can create arrays for each object returned:

```
Account [] accounts = ((List<Account>)searchList[0]);
    1.Contact [] contacts = ((List<Contact>)searchList[1]);
    2.Opportunity [] opportunities = ((List<Opportunity>)searchList[2]);
    3.Lead [] leads = ((List<Lead>)searchList[3]);
        System.debug(‘Contact List + contacts);
        System.debug(‘Opportunity List + opportunities);
        System.debug(‘Account List + accounts);
        System.debug(‘Leads list + leads
```