SOQL QUERIES

SOQL ----> Salesforce Object Query Language

By using SOQL Queries we can fetch either one/more/all records from the salesforce objects.

We can fetch the data from one/more salesforce objects.

While fetching the records from the object we can add one or more user defined conditions to be used to filter the records from the result set.

Query: Query is a request to the Database.com to fetch the required records from Salesforce Objects.

SOQL Queries can be prepared by using "SELECT" statement.

Syntax:

SELECT <FieldNames> from <ObjectName>

[WHERE <Conditions>]

[GROUP BY <ColumnNames>]

[HAVING <Conditions>]

[ORDER BY <ColumnNames>]

[LIMIT <Number of Records to Restrict>]

[OFFSET <Number of Records to Skip>]

[ALL ROWS]

[FOR UPDATE]

Governor Limits

1. We can use maximum of 100 SOQL Queries within a Transaction. When the user tries to use more than 100 SOQL Queries, then Salesforce will raise an exception "System.LimitException. Too Many SOQL Queries: 101".

2. Each SOQL Query can return maximum of 50,000 records.

Best Practices

1. It is always recommended to avoid the usage of SOQL Queries inside the "FOR Loop".

2. While fetching the records from the object don't fetch the unnecessary data from the object, which causes the Performance issues.

Ways to Invoke SOQL Queries

Salesforce provides the below 2 ways to invoke the SOQL Queries.

1. By using "Query Editor" from "Developer Console".

Click on "Setup" menu.

1. Click on "Developer Console" menu item.

2. Go to the "Developer Console" Editor.

3. Expand the Tab Bar, and Click on "Query Editor" tab.

4. Write the SOQL Query inside the "Query Window".

5. Click on "Execute Button".

6. View the results inside the "Result Window".

2. By using "Apex Programming".

Use Case: Write an SOQL Query, to fetch the Account Record Id, Name, Rating, Industry, Annual Revenue, Phone, and Fax Field values.

Select id, name, rating, industry, annual revenue, phone, fax from Account

Use Case: Write an SOQL Query, to fetch all the Lead Records from Lead Object.

Select id, first Name, last Name, Company, Status, email, phone, fax, title, industry, rating, annual revenue from Lead

Use Case: Write an SOQL Query, to fetch all the Case Records from Case Object.

Select id, Case Number, Status, Priority, Type, Origin, Reason, Subject from Case

Use Case: Write an SOQL Query, to fetch all the Hiring Manager Records.

Select id, name, location__C, designation__C, Contact_Number__c, Email_Address__c

from Hiring_Manager__C

Use Case: Write an SOQL Query to fetch all the User Details from User Object.

Select id, firstName, lastname, companyname, title, department, username, email, isActive

from User

Use Case: Write an SOQL Query, to fetch all the Email Templates from the object.

Select id, name, templateType, isActive, description, subject, body from EmailTemplate

We can execute the SOQL Queries by using "Apex Programming". Apex sub-divides the SOQL Queries into 2 types.

1. Static SOQL Queries

2. Dynamic SOQL Queries

Static SOQL Query: In static SOQL Query, the query will be "fixed", which cannot be edited at runtime i.e., while preparing the SOQL Query we have to specify all the required fields and required conditions. We cannot add any more fields / conditions to the query at runtime.

Static SOQL Query should be enclosed in "Square Braces".

Syntax: [SOQL Query]

Ex: [Select id, name, rating, industry, annual revenue from Account]

Static SOQL query will get executed automatically by default. Once the Query executes, it will return the results.

SOQL Query returns Only One Record: If the SOQL Query returns only one record as the result, then we have to store the record by defining the variable of the associated data type.

Syntax: <SObjectDataType> <refName> = [SOQL Query - Returns One Record];

Ex: Account accRecord = [Select id, name, rating, industry, annualrevenue, active__C from Account];

```
System.debug('Account Record is....: '+ accRecord);

if(accRecord.id != Null)

        {

                System.debug('Account Record Id is....: '+ accRecord.Id);

                System.debug('Account Name is.......: '+ accRecord.name);

                system.debug('Industry Name is......: '+ accRecord.Industry);

                System.debug('Active Status is......: '+ accRecord.Active__C);

        }
```

SOQL Query returns Multiple Records: If the SOQL Query returns multiple records as the results, store the records in the "List Collection" of the associated type.

Syntax: List<SObjectDataType> <refName> = [SOQL Query - Multiple Records];

Ex: List<Account> lstAccounts = [Select id, name, rating, industry, annualrevenue, active__C, phone from Account];

```
System.debug('Collection Size is......: '+ lstAccounts.Size());

if(! lstAccounts.isEmpty())

        {

        for(Account acc : lstAccounts)

        {

                System.debug('Account Id is....: '+ acc.id);

                System.debug('Account Name is...: '+ acc.Name);

                System.debug('Industry Name is...: '+ acc.Industry);

                System.debug('Active Status is....: '+ acc.active__C);

        }

        }
```

Use Case: Write an apex program, to represent all the account records on the log File.

Class Code:

```
public class DatabaseHelper
{
        Public static void GetAllAccounts()
   {
      List<Account> lstAccounts =    [Select id, name, rating, industry, annualrevenue,
                                      active__C, customerpriority__C from Account];

      System.debug('Collection Size is....: '+ lstAccounts.size());

      if(! lstAccounts.isEmpty())
      {
         for(Account acc : lstAccounts)
         {
            System.debug('Account Record Id is....: '+ acc.id);
            System.debug('Account Record Name is...: '+ acc.name);
            System.debug('Rating Value is.....: '+ acc.rating);
            System.debug('Annual Revenue is....: '+ acc.AnnualRevenue);
            System.debug('Active Status is....: '+ acc.Active__c);
            System.debug('Customer Priority is...: '+ acc.CustomerPriority__c);
            System.debug('-----------------------------------------------');
         }
      }
   }
}
```

Execution: DatabaseHelper.GetAllAccounts();

Use Case: Write an apex program to fetch all the case records from the object and represent them on the Debug Log File.

Class Code:

```
public class DatabaseHelper
{
   Public static void GetAllCaseRecords()
```

```
    {
        for(Case csRecord : [ Select id, CaseNumber, Status, Priority, Type, Origin, Reason, Subject from Case ])
        {
            System.debug('Case Number is....: '+ csRecord.CaseNumber);
            System.debug('Case Status is....: '+ csRecord.Status);
            System.debug('Case Priority is..: '+ csRecord.priority);
            System.debug('Case Origin is....: '+ csRecord.Origin);
            System.debug('Case Reason is....: '+ csRecord.Reason);
            System.debug('Case Subject is...: '+ csRecord.Subject);
            System.debug('---------------------------------------');
        }
        System.debug('Number of Records Returned....: '+ System.Limits.getQueryRows());
    }
}
```

Execution: DatabaseHelper.GetAllCaseRecords();

Use Case: Write an apex program to fetch all the Hiring Manager records from the object and store them to a Map Collection. Represent the records on the Debug Log File.

Class Code:

```
public class DatabaseHelper
{
    Public static void GetAllHRRecords()
    {
        /*
        List<Hiring_Manager__C> lstHRRecords = [Select id, name, location__C, Contact_Number_del_del__c,  Email_Address_del__c, designation__C from Hiring_Manager__c];

        if(! lstHRRecords.isEmpty())
        {
            Map<Id, Hiring_Manager__C> mapHRRecords = new Map<Id, Hiring_Manager__C>();

            for(Hiring_Manager__C hrRecord : lstHRRecords)
```

```
        {
            mapHRRecords.Put(hrRecord.Id, hrRecord);
        }
    }
            */

    Map<Id, Hiring_Manager__C> mapHRRecords = new Map<ID, Hiring_Manager__C>([Select
id, name, location__C, Contact_Number_del_del__c,Email_Address_del__c, designation__C
                    from Hiring_Manager__c]);
    System.debug('Number of Records.....: '+ mapHRRecords.Size());
    if(! mapHRRecords.isEmpty())
    {
        for(Hiring_Manager__C hr : mapHRRecords.values())
        {
            System.debug('Hiring Manager Record is...: '+ hr);
        }
    }
  }
}
```

Execution: DatabaseHelper.GetAllHRRecords();

LIMIT Clause:Limit Clause is used to restrict the Number of Records to be returned by the SOQL Query.

Syntax: [SOQL Query Limit <Integer>]

Ex: Limit 1

Limit 10

Use Case: Write an SOQL Query to fetch 5 Lead Records from the Object.

```
List<Lead> lstLeads = [Select id, firstName, lastName, company, title,
                        status, email, phone from Lead Limit 5];
```

Use Case: Write an SOQL Query to fetch only 2 Position Records from the Object.

```
List<Position__C> lstPositions = [Select id, name, location__C,
    position_status__C, maximum_budget__C, milestone_date__C, open_date__C
```

```
from Position__C Limit 2];
```

ALL ROWS Clause: All Rows Clause is used to fetch the records inside the object along with the deleted records waiting inside the Recycle Bin.

Note: We can differentiate the records by using "IsDeleted" field, which is a checkbox datatype field.

If IsDeleted == TRUE ---> It is a Deleted Record.

If IsDeleted == FALSE ---> It is not a Deleted Record.

Syntax: [SOQL Query ALL ROWS]

Use Case: Write an apex program to fetch all the account records from the object including deleted records.

Class Code:

```
public class DatabaseHelper
{
   Public static void ExportAllAccounts()
   {
      List<Account> lstAccounts = [Select id, name, rating, industry, annualrevenue,
                        active__C, phone, fax, isDeleted from Account ALL ROWS];
System.debug('Number of Account Records.....: '+ lstAccounts.size());
if(! lstAccounts.isEmpty())
      {
         for(Account acc : lstAccounts)
         {
            System.debug('Account Record is....: '+ acc);
         }
      }
   }
}
```

Execution: DatabaseHelper.ExportAllAccounts();

ORDER BY Clause: By using this Clause, we can arrange the records either in "Ascending / Descending" order based on one or more fields inside the object.

Syntax: [SOQL Query ORDER BY <FieldName> [ASC(Default) / DESC]]

UseCase: Write an SOQL Query to fetch all the Contact Records from the Object and arrange them in Ascending Order based on "First Name".

```
List<Contact> lstContacts = [Select id, firstname, lastname, email, title, birthdate, phone
                             from Contact order by firstname ];
```

Use Case: Write an SOQL Query, to fetch all the case records from the object and arrange them in Chronological Order based on "CreatedDate".

```
List<Case> lstCases = [Select id, caseNumber, status, priority, origin, reason, createddate
                       from Case Order by Created Date desc];
```

Use Case: Write an SOQL Query, to fetch Top 4 Highest Annual Revenue Account Records from the Object.

```
List<Account> lstAccounts = [Select id, name, rating, industry, annual revenue, phone, fax
                             from Account Order by Annual Revenue desc Limit 4];
```

OFFSET Clause: This Clause is used to skip the specified number of records from the result set.

Note: Both "Limit and OFFSET" clauses will be used upon implementing the "Pagination".

Always "OFFSET" clause should be followed by the "Limit Clause".

Syntax: [SOQL Query OFFSET <Integer>]

Use Case: Write an apex program, to represent both "Limit and OFFSET" clauses.

Class Code:

```
public class DatabaseHelper
{
   Public static void GetAccountsByOFFSET()
   {
      for (Account accRecord : [Select id, name, rating, industry, annualrevenue, active__C
                                from Account  Order by name Limit 50 Offset 10])
      {
         System.debug('Account Record is....: '+ accRecord);
      }
   }
}
```

Execution: DatabaseHelper.GetAccountsByOFFSET();

FOR UPDATE Clause: This Clause is used to apply the lock on the records upon updation. Till the operation is completed, the lock will be applied by Salesforce automatically, so that the other users can't access the records.

Once the transaction completes, the lock will be released by Salesforce.

(Maximum Time Duration for the Transaction is 10,000 Milliseconds)

Syntax: [SOQL Query FOR UPDATE]

Ex: List<Account> lstAccounts = [Select id, name, rating, industry, active__C

from Account FOR UPDATE];

WHERE Clause: By using WHERE clause, we can add one or more user defined conditions along with the SOQL Query, used to filter the records from the result set.

Based on the need we can add one or more filters based on the object fields.

Syntax: [SOQL Query WHERE <Conditions>]

Ex: Where Rating = 'Hot'

Where AnnualRevenue < 4500000

Where Phone != Null

Where Email__C = 'ram@gmail.com'

Where isActive = true

We can add multiple conditions along with the Query, by using "Logical Operators".

Where Rating = 'Warm' && AnnualRevenue > 5000000

Where Location__C = 'Bangalore' || Location__C = 'Chennai'

Use Case: Write an apex program, to fetch all the Energy Industry Accounts from Account Object.

Class Code:

```
public class DatabaseHelper
{
   Public static void GetEnergyIndustryAccounts()
   {
        List<Account> lstAccounts = [Select id, name, rating, industry, annualrevenue,
                                    active__C, customerPriority__C
                                        from Account  Where Industry = 'Energy'];
     System.debug('Number of Records Count....: '+ lstAccounts.Size());
```

```
        if(! lstAccounts.isEmpty())
        {
            for(Account accRecord : lstAccounts)
            {
                System.debug('Account Record is....: '+ accrecord);
            }
        }
    }
}
```

Execution: DatabaseHelper.GetEnergyIndustryAccounts();

Use Case: Write an SOQL Query, to fetch all the High Priority Cases.

```
List<Case> lstCases = [Select id, caseNumber, priority, status, origin, reason
                                                    from Case Where Priority = 'High'];
```

Use Case: Write an SOQL Query to fetch all the active accounts from the object which are associated with "Finance Industry".

```
List<Account> lstAccounts = [Select id, name, rating, industry, annualrevenue, active__C
                                    from Account Where industry = 'Finance' and Active__C = 'Yes'];
```

Use Case: Write an SOQL Query, to fetch all the Active Users.

```
List<User> activeUsers = [Select id, firstname, lastname, email, username, isActive
                                                    from User Where isActive = true];
```

Use Case: Write an SOQL Query to fetch all the deleted opportunity records from the object, whose Amount value is more than 50000.

```
List<Opportunity> oppty = [Select id, name, amount, isDeleted rom Opportunity
                                    Where isDeleted = true and Amount > 50000 ALL ROWS];
```

IN Operator: By using IN Operator, we can compare a field value with a collection of elements so that we can fetch the matching records.

IN Operator should be always used along with the "WHERE Clause".

Syntax: [SOQL Query where <FieldName> IN: <collectionName/ arrayName>]

```
Ex: Where industry = 'Finance' OR
                    industry = 'Banking' OR
                    industry = 'Manufacturing' OR
```

industry = 'Education'

(OR)

Where Industry IN ('Finance','Banking','Manufacturing','Education')

(OR)

String[] industryNames = new String[]{'Finance','Banking','Manufacturing','Education'}

Where Industry IN: industryNames

Use Case: Write an Apex Program to fetch all the related cases for the Energy Industry Accounts.

Class Code:

```
public class DatabaseHelper
{
   Public static void GetEnergyIndustryCases()
   {
      // Get the Account Records based on Industry
        Map<ID,Account> mapAccounts = new Map<ID, Account>([Select id, name, industry
                                                  from Account  Where industry = 'Energy']);
      // Get the related cases
        If (! mapAccounts.isEmpty())
         {
            List<Case> lstCases = [Select id, caseNumber, status, priority, origin, accountId
                                    from Case Where AccountID IN: mapAccounts.keySet()];
            System.debug('Number of Related Cases....: '+ lstCases.Size());
            for(Case cs : lstCases)
            {
               System.debug('Case Record is....: '+ cs);
            }
         }
   }
}
```

Execution: DatabaseHelper.GetEnergyIndustryCases();

Use Case: Write an apex program, to fetch all the Accounts based on the specified Industry Name at runtime.

Class Code:

```
public class DatabaseHelper
{
   Public static void GetAccountsByIndustry(String industryName
   {
      if(industryName != Null && industryName != '')
      {
         for(Account accRecord : [Select id, name, rating, industry, annualrevenue
                                                  from Account
                                                      Where industry =: industryName ])
         {
            System.debug('Account Record is....: '+ accRecord);
         }
      }
   }
}
```

Execution: DatabaseHelper.GetAccountsByIndustry('Banking');

LIKE Operator: LIKE operator is used to fetch the similar kind of information from the object based on the specified expression.

LIKE Operator should be always used along with the "WHERE Clause".

Syntax: [SOQL Query Where <FieldName> LIKE <Expression>]

To use LIKE operator, prepare the expression which includes the "Wild Card Characters". SOQL query supports the below 2 wild card characters.

"%" ----> It represents zero or more characters.

Ex: 'Ram%' ----> Represent all the Words starting with "Ram".

Ex: Ram, Ramana, Ramesh, Ram Kumar etc.

'%Kumar' ----> Represents all the words which ends with "Kumar".

Ex: Ram Kumar, Kumar, Suresh Kumar, Pawan Kumar...etc.

'%am%' ----> Represents all the words, which contains "am".

Ex: Ram, aman, Ramana, Ramesh, Paramesh...etc.

"_" (Underscrore) ----> It represents only one character.

Ex: '_u%' ----> Represents only those words, whose Second Character is "u".

Ex: Suresh, Sukesh, Murali, Murugan etc.

Use Case: Write an SOQL Query to fetch all the account records whose name starts with "Ed".

```
List<Account> lstAccounts = [Select id, name, rating, industry, annualrevenue
                             from Account Where name like 'Ed%'];
```

Use Case: Write an SOQL Query, to fetch all the Hiring Manager Records from the object, whose name is ending with the characters "Kumar".

```
List<Hiring_Manager__C> lstHRRecords = [Select id, name, location__C, email_address__C
                                        from Hiring_Manager__C Where name like '%Kumar'];
```

GROUP BY Clause: Used to segregate the records into various groups based on one or more columns inside the object.

Should be used always with aggregate functions.

Aggregate functions take collection of values as input, perform operations on the records and returns only one value as the result.

The following aggregate functions can be used in SOQL queries.

i**. Avg():** Returns the average value of the numerical field.

Ex: Select Campaignid, Avg (Amount) from opportunity group by campaignid

ii. **Count(fieldname):** Returns the number of rows that matches the filter conditions and have a non-null value for the field name.

We can include multiple Count(fieldname) items in a SELECT statement.

Select count (id) from Account where name like 'a%'.

iii. **Count_distinct()**: Returns the number of distinct non null field values matching the query criteria.
Ex: Select count_distinct(company) from lead

iv. **Count()**: Returns an integer value, the number of records existing inside the object.

iv.**Min():** Returns minimum value of a field. Can take number/percent/currency/date/time/date time as the input.

Ex: Select Min(Createddate), firstname, lastname from contact group by firstname, lastname

iv. **Max():** Returns maximum value of a field.

Select name, max(budgetedcost) from campaign group by name

v. **Sum():** Returns total sum of a numerical field.

Ex: Select sum(amount) from opportunity where Isclosed=false and probability>60

Upon using aggregate functions inside the query, the result should be of type Aggregate Result class.

Ex: Aggregate Result[] results= [Soql query with group by clause];

We can fetch each record inside the result set by iterating the collection. We can get the value of the field in the record using get function.

Syntax: <object name>. get (field name/alias name);

Upon using aggregate functions in the query, we should provide an alias name which is a temporary name allocated to the column to fetch the values.

Having Clause: Optional clause used in SOQL query to filter the records that aggregate functions return.

Similar to where clause, we can include aggregate functions in Having clause, but not in a Where clause.

Ex: [Select leadsource, count(name) from lead group by leadsource

having count(name)>100];

[Select name, count(id) recordcount from Account

group by name, having count(id)>1];

Dynamic SOQL Query: We can prepare the SOQL Query dynamically at runtime and we can add the required fields/conditions to the query at runtime.

Dynamic SOQL Query should be always enclosed with single quote and should be stored inside the string variable.

Syntax: String <variableName> = 'Dynamic SOQL Query';

Dynamic SOQL Queries should be invoked with the help of Database.Query() method, which always returns the results in the form of a list collection.

Syntax: List<SObjectDataType> <refName> = Database.Query('SOQL Query');

Use Case: Write an apex program to fetch all the Hiring Manager Records from the object based on the specified location at runtime.

Class Code:

```
public class DatabaseHelper
{
        Public static void GetHRRecordsByCity(String cityName)
   {
      String hrRecordsQuery = 'Select id, name, location__C, Contact_Number__c, Email_Address__c, '+'designation__C from Hiring_Manager__C '+
                                        Where location__C =: cityName';
      List<Hiring_Manager__C> lstHRRecords =       Database.query(hrRecordsQuery);
      System.debug('Number of Records Count....: '+ lstHRRecords.size());
      If (! lstHRRecords.isEmpty())
      {
```

```
            For (Hiring_Manager__C hrRecord : lstHRRecords)
            {
                System.debug('Hiring Manager Record...: '+ hrRecord);
            }
        }
    }
}
```

Execution: DatabaseHelper.GetHRRecordsByCity('Pune');

Use Case: Write an apex program to fetch the Case Records based on the specified "Priority and Status" at runtime.

Class Code

```
public class DatabaseHelper
{
    Public static void GetCaseRecordsByStatus(String caseStatus, String casePriority)
    {
        if(caseStatus != '' && caseStatus != Null && casePriority != Null && casePriority != '')
        {
            List<Case> lstCases = [Select id, caseNumber, Status, priority, origin, reason, subject
                                                    from Case
                                    Where Status =: caseStatus and Priority =: casePriority];
            System.debug('Number of Cases Count.....: '+ lstCases.Size());
            if(! lstCases.isEmpty())
            {
                for(Case csRecord : lstCases)
                {
                    System.debug('Case Record is....: '+ csRecord);
                }
            }
        }
    }
}
```

Execution: DatabaseHelper.GetCaseRecordsByStatus('New', 'High');

Use Case: Write an apex program to count the number of records exist in the Account Object.

Class Code:

```
public class DatabaseHelper
{
   Public static void GetAccountsCount()
   {
      // Static SOQL Query
         Integer recordsCount = [Select count() from Account];
         System.debug('Account Records Count - Static SOQL is....: '+ recordsCount);
      // Dynamic SOQL Query.
         String accountsQuery = 'Select count() from Account';
         Integer accountsCount = Database.countQuery(accountsQuery);
         System.debug('Account Record Count - Dynamic SOQL is....: '+ accountsCount);
   }
}
```

Execution: DatabaseHelper.GetAccountsCount();

Use Case: Write an apex program to fetch all the account records based on the specified starting character at runtime.

Class Code

```
public class DatabaseHelper
{
   Public static void SearchAccountsByStartingChars(String startingChars)
   {
      if(startingChars != Null && startingChars != '')
      {
         startingChars += '%';
 List<Account> lstAccounts = [Select id, name, rating, industry, annualrevenue, active__C
      from Account Where name like : startingChars];
//Dynamic SOQL
String accountsQuery = 'Select id, name, rating, industry, active__C from Account Where name
like : startingChars';
System.debug('Number of Account Records.....: '+ lstAccounts.size());
```

```
if(! lstAccounts.isEmpty())
        {
 for(Account accRecord : lstAccounts)
        {
System.debug('Account Record is.....: '+ accRecord);
         }
         }
}
}
```