What is SOQL (Salesforce Object Query Language)?

Salesforce Object Query Language. It is used to retrieve data from the Salesforce database according to the specified conditions and objects. Similar to all Apex code, SOQL is also case insensitive.

Use SOQL when you know which objects the data resides in, and you want to:

- Retrieve data from a single object or from multiple objects that are related to one another.
- Count the number of records which meet the specified criteria.
- Sort results as part of the query.
- Retrieve data from number, date, or checkbox fields.

A SOQL query is equivalent to a SELECT SQL statement and searches the org database.

Example:

```
SELECT Name FROM Account // standard object
SELECT Name, Student_Name FROM Student__c // custom object
```

But we cannot use asterisk (*) in SOQL queries which we can use in sql.

OR, AND and WHERE Clause:

`Select Student_name from student__c` where couse_opted__c = ‘Salesforce Admin’;

`Select Student_name from student__c` where couse_opted__c = ‘Salesforce Admin and App Builder’ OR couse_opted__c = ‘Salesforce Developer’;

`Select Student_name from student__c` where couse_opted__c = ‘Salesforce Admin and App Builder’ OR graduated__c = false;

Example:

```
public class Soql
{
	public static void main()
	{
		List<Account> accList = [Select Name, NumberOfEmployees FROM Account];
			for(integer i=0; i<accList.size(); i++)
			{
				System.debug(accList[i].numberOfEmployees);
			}

			// *** FOR EACH LOOP ***

			for(Account a:acclist)
			{
				System.debug('Acc	Name	=	'+a.Name+'NumOfEmp	=
		'+a.numberOfEmployees);
			}
	}
}
```

Note: It is mandatory to mention the field in the SOQL query which you want to access in the Apex Code.

Note: Comparison of strings are case sensitive using '=' operator in SOQL.

SOQL Variable Binding:

SOQL queries support Apex variable binding. You can use an Apex variable and filter SOQL records against the value of that variable.

```
String strName = 'John Snow';
List<Position__c> positionList = [SELECT Name
                          FROM Position__c
                          WHERE Name =: strName];
```

SOQL Aggregate Functions:

"SOQL Aggregate Functions" shows a calculated result from the query and returns `AggregateResult`. Any query that includes an aggregate function returns its results in an array of AggregateResult objects. AggregateResult is a read-only sObject and is only used for query results.

1. Sum():

```
AggregateResult ar = [SELECT SUM(Max_Salary__c)
                      FROM Position__c WHERE Max_Salary__c > 10000];
```

2. Max():

```
AggregateResult ar = [SELECT MAX(Max_Salary__c)
                      FROM Position__c
                      WHERE Max_Salary__c > 10000];
```

3. Min():

```
AggregateResult ar = [SELECT MIN(Max_Salary__c)
                      FROM Position__c
                      WHERE Max_Salary__c];
```

4. Count() and Count(fieldName):

```
Integer i = [SELECT count() FROM Account] ;
AggregateResult ar = [SELECT count(Id) FROM Account] ;
```

5. Avg():

```
AggregateResult ar = [SELECT AVG(Min_Salary__c)
                      FROM Position__c
                      WHERE Max_Salary__c < 5000];
```

6. Count_Distinct():

```
AggregateResult ar = [SELECT COUNT_DISTINCT(Name)
                      FROM Position__c];
```

SOQL Keywords:

Some of the major keywords used in SOQL queries are:

1. IN: IN keyword is used to query on bulk data, which means the WHERE condition involves a collection of records. The collection of records can be a Set, List of IDs, or sObjects which have populated ID fields. Ex:

```
Set<Id> positionIdSet =
      new   Set<Id>{'a056000000WZEpq',   'a056000000WZYey'};   //   Note:
Hardcoding Ids is not a best practice.
List<Position__c> positionList = [SELECT Name
                           FROM Position__c
                           WHERE Id IN : positionIdSet];
```

   Note: Map cannot be used with IN keyword because it is not an iterable collection.

2. LIKE: LIKE keyword allows selective queries using wildcards. For example, to query all the position records where Name starts with 'John', you can use a query such as the one given in the code here:

```
List<Position__c> positionList = [SELECT Name
                           FROM Position__c
                           WHERE Name LIKE 'John%'];
```

3. AND/OR: With AND/OR keywords you can apply AND/OR logic to your query conditions. For example, to query all the position records whose name starts with John and whose Status is open, use query syntax as given here:

```
List<Position__c> positionList = [SELECT name
                     FROM Position__c
                     WHERE Name LIKE 'John%' AND Status__c = 'Open'];
```

4. NOT: NOT keyword is used for negation. For example, if you want to query all the Position records apart from the two Position Salesforce IDs in a given set, use NOT syntax as given here.

```
Set<Id> positionIdSet =
      new Set<Id>{'a056000000WZEpq', 'a056000000WZYey'};    //      Note:
Hardcoding Ids is not a best practice.
List<Position__c> positionList = [SELECT name
                           FROM Position__c
                           WHERE Id NOT IN : positionIdSet];
```

5. ORDER BY: ORDER BY is used to sort the records in ascending(ASC) or descending(DESC) order. It is used after the WHERE clause. Also, you can specify if null records should be ordered at the beginning (NULLS FIRST) or end (NULLS LAST) of the results. By default, null values are sorted first.

```
List<Position__c> positionList = [SELECT Type__c, Comments__c
               FROM Position__c
               WHERE Status__c = 'Open' ORDER BY Type__c ASC NULLS LAST];
```

6. GROUP BY: GROUP BY is used with Aggregate functions to group the result set by single or multiple columns. This keyword also groups results for a particular field with minimal code.

```
SELECT Status__c, COUNT(Name)
FROM Position GROUP BY Status__c;
```

Position Status	Number of Records
Open	24
Close	13
On Hold	7

7. HAVING: HAVING is an optional clause that can be used in a SOQL query to filter results that Aggregate functions return. You can use a HAVING clause with a GROUP BY clause to filter the results returned by aggregate functions.

```
List<AggregateResult> agrRes = [SELECT LeadSource, COUNT(Name)
                    FROM Lead
                    GROUP BY LeadSource HAVING COUNT(Name) > 100]
```

8. LIMIT: LIMIT keyword puts a cap on the number of records a SOQL query should return. This keyword should always be used at the end of the SOQL query.

```
Student__c st1 = [SELECT Name FROM Student__c WHERE Student_Name__c =
'shrey' LIMIT 1];
```

9. FOR UPDATE: FOR UPDATE keyword locks the queried records from being updated by any other Apex Code or Transaction. When the records are locked, other Users cannot update them.

```
List<Position__c> positionList = [SELECT Name, Comments__c
                            FROM Position__c
                            WHERE Status__c = 'Open' FOR UPDATE];
```

10. ALL ROWS: ALL ROWS keyword in a SOQL query retrieves all the records from the database, including those that have been deleted. This keyword is mostly used to undelete records from the Recycle Bin.

```
List<Position__c> positionList = [SELECT Name
                            FROM Position__c
                            WHERE isDeleted = true ALL ROWS];
```

Query Functions:

1. convertCurrency(): Use convertCurrency() in the SELECT statement of a SOQL query to convert currency fields to the user's currency. This action requires that the org has multiple currencies enabled. Ex:

```
SELECT Id, convertCurrency(AnnualRevenue)
FROM Account
```

2. convertTimezone(): SOQL queries in a client application return dateTime field values as Coordinated Universal Time (UTC) values. You can use convertTimezone() in a date function to convert dateTime fields to the user's time zone. Ex:

```
SELECT convertTimezone(CreatedDate)
FROM Opportunity
```

3. Date Function: SOQL queries in a client application return dateTime field values as Coordinated Universal Time (UTC) values. You can use convertTimezone() in a date function to convert dateTime fields to the user's time zone. Ex:

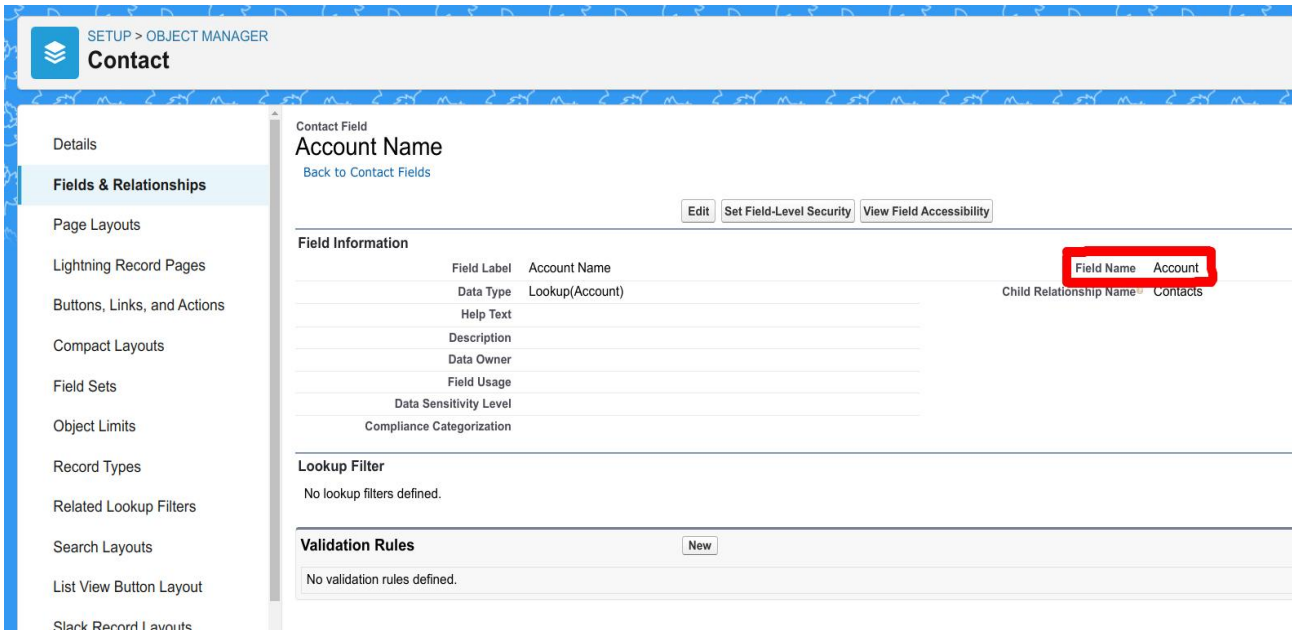
Relationship Queries:

Relationship queries involve at least two objects, a parent and a child. These queries are used to fetch data either from the parent object, when SOQL query is written on child, or from child object when SOQL query is written on parent.

1. Child to Parent:

   For Standard Objects:

```
SELECT Field_Name.Parent_Obj_Field FROM Child_Object;
List<Opportunity> oppList = [SELECT Name, Amount, CloseDate, Account.Name
                                    FROM Opportunity];
  for(Opportunity o: oppList)
  {
       System.debug('Opp: ' + o.Name  + ' Acc: '+o.Account.Name);
  }
```



   For Custom Objects:

```
SELECT (Field_Name)__r.Parent_Obj_Field FROM child_object;
List<Branch__c> branchList = [SELECT Branch_ID__c, State__c,
                          Branch_of_Bank__r.Bank_Name__c from Branch__c];
  for(Branch__c bl: branchList)
  {
       System.debug('Bank Name: '+bl.Branch_of_Bank__r.Bank_Name__c
                          + 'Branch: '+bl.State__c);
```

```
}
```

2. Parent to Child Relationship Query:

   For Standard Objects:

```
SELECT (SELECT Child_Obj_Field FROM Child_Relationship_Name) FROM
Parent_Object;
List<Account> accList = [SELECT Name, NumberOfEmployees, (SELECT Name, Amount,
                          CloseDate FROM Opportunities) FROM Account];
  for(Account acc: accList)
  {
       System.debug('ACC: ' + acc.Name + ' NoEmpl: '+acc.numberOfEmployees.;
       List<Opportunity> opList = acc.Opportunities;
       for(opportunity o: oplist)
       {
              System.debug('Opportunity Name:' + o.Name + 'Amount: '+o.Amount);
       }
  }
```

   For Custom Objects:

```
SELECT (SELECT Child_Obj_Field FROM (Child_Relationship_Name)__r)
FROM Parent_Object;
List<Bank__c> bankList = [SELECT Bank_Name__c, (SELECT Branch_Id__c, Name,
                                 State__c FROM Branches__r) FROM Bank__c];
for(Bank__c ba: bankList)
{
       List<Branch__c> branchList = ba.Branches__r;
       for(Branch__c br: branchList)
       {
              System.debug('Bank:'+ba.Bank_Name__c+'|Branch ID:'+br.Branch_ID__c
                     +'|Branch Name:'+br.Name+'|Branch State:'+br.State__c);
       }
}
```

Note: We can not write a subquery inside a subquery in SOQL.

Note: We can write up to 20 subquery in a SOQL query.

3. Multi-level Relationships: In SOQL, you can traverse up to a maximum of five levels when querying data from child object to parent.

```
       SELECT Name, Position__r.Contact.Account.Owner.FirstName
              FROM Job_Application__c;
       SELECT Name, (SELECT Name FROM Job_Applications__r)
              FROM Position__c
Five Levels:
```

Job Application → Position → Contact → Account → User

SOQL 'for' Loops:

SOQL 'for' Loops are SOQL statements that can be used inline along with FOR Loops. This means that instead of iterating over a list of records, it is done directly over a SOQL query written inline inside the 'for' Loop iteration brackets.

```
FOR(Position__c pos : [SELECT Name, Max_Salary__c
                       FROM Position__c
```

```
WHERE Status__c = 'Open']){ //code block }
```

There are two formats to this type of SOQL query:

1. Single Variable Format:
   Single Variable Format executes the loop once per list of sObject records. The syntax of this format is given here:

```
List<Position__c> positionRecordsList = [SELECT Name, Max_Salary__c
                          FROM Position__c
                          WHERE Status__c = 'Open'];
for(Position__c pos : positionRecordsList) {
      // code block
}
```

2. List Variable Format:
   List Variable Format executes the loop code once per 200 sObject records.

```
a.    for(Position__c pos : [SELECT Name, Max_Salary__c
                      FROM Position__c
                      WHERE Status__c = 'Open']){
            //code block
      }
b.    for(List<Position__c> positionList : [SELECT Name, Max_Salary__c
                                    FROM Position__c]){
            for(Position pos: positionList){
                  //code block
            }
      }
```

Dynamic SOQL:

Dynamic SOQL means creation of SOQL string at runtime with Apex code. It is basically used to create more flexible queries based on user's input.

Use Database.query() to create dynamic SOQL.

```
public static void main(String objectName)
{
      String query = 'SELECT Name FROM '+objectName;
      List<sObject> sList = Database.query(query);
      for(sObject s: sList)
      {
            System.debug(s);
      }
}
```

Note: If the query string is wrong, this method returns the QueryException.

```
public static void main(String table, String field)
{
      String query = 'SELECT Name,';
      query = query + field;
      query = query + ' FROM';
      query = query + table;
      String str = ' WHERE Name LIKE \'Acme\'';
      query = query + str;
}
```

Note: We can bind variables (like ':i') in dynamic SOQL but we can't bind variable fields in a dynamic SOQL (like ':500').

```
public static void main()
{
	String s = 'Test%';
	Database.query('SELECT Name FROM Account WHERE Name LIKE :s');
}
```

Note: The above query will work in dynamic as well as static SOQL.

```
public static void main()
{
	Account a = new Account(Name= 'Test', Phone='12345');
	Database.query('SELECT Name FROM Account WHERE Phone= :a.Phone');
}
```

Note: The above query will not work in dynamic SOQL and will result in a 'variable does not exist' error in it. But it will work perfectly in static SOQL.

There is a hack to bind variable fields in SOQL.

```
public static void main()
{
	Account a = new Account(Name='abcd', Phone='12345');
	String str = a.Phone;
	String s = 'SELECT Name FROM Account WHERE phone=:str';
}
```

Note: Use string escape singleQuotes(String str) on the string used for creating the query on dynamic SOQL, just to prevent SOQL injection.

```
String finalString = String.escapeSingleQuotes('SELECT Name FROM Account WHERE phone=:str');
Database.query(finalString);
```

This method replaces all single quotes(') by (\') which ensures that all single quotation marks are treated as enclosing strings instead of database commands.

```
String fieldString = 'Name';
	String sObjectString = 'Position__c';
	List<Position__c> positionList = Database.query('SELECT ' + fieldString + 'FROM ' + sObjectString);
```

What is SOSL (Salesforce Object Search Language)?

SOSL is a highly optimized way of searching records in Salesforce across multiple Objects that are specified in the query. A SOSL query returns a list of list of sObjects and it can be performed on multiple objects.

Whether you use SOQL or SOSL depends on whether you know which objects or fields you want to search, plus other considerations.

Use SOSL when you don't know which object or field the data resides in, and you want to:

- Retrieve data for a specific term that you know exists within a field. Because SOSL can tokenize multiple terms within a field and build a search index from this, SOSL searches are faster and can return more relevant results.
- Retrieve multiple objects and fields efficiently where the objects might or might not be related to one another.
- Retrieve data for a particular division in an organization using the divisions feature.
- Retrieve data that's in Chinese, Japanese, Korean, or Thai. Morphological tokenization for CJKT terms helps ensure accurate results.

The basic difference between SOQL and SOSL is that SOSL allows you to query on multiple objects simultaneously whereas in SOQL we can only query a single object at a time.

Some key point:

- ➔ In the query editor, `FIND {Test}`
- ➔ `FIND {Test} RETURNING Account(name), Contact(Name).`
- ➔ Text searches are case-insensitive.
- ➔ `FIND {Test*}` where asterisk(*) is a wildcard.
- ➔ `FIND {Test} IN NAME FIELDS`
- ➔ Whenever referring to ALL FIELDS, use * for emails and when specifying EMAIL FIELDS, no need for *.
- ➔ `FIND {What?} [What type?] [Where?].`
- ➔ `FIND {Upsert} RETURNING Account(Name,Phone), Contact(LastName), Student__c(Name, Name__c).`
- ➔ `FIND {test OR upsert}.`
- ➔ `FIND {MyProspect AND MyCompany}.`
- ➔ In Apex FIND clause differs, `FIND 'Test'`
- ➔ In order to limit characters
- ➔ Find {Test??} ? => single character.
  * => zero or multiple characters.
- ➔ SOSL does not have to use a wildcard (*) as it already matches the fields based on the string match.

SOSL Syntax:

SOSL queries start with the FIND keyword, and then the word or characters that need to be searched is written in the form of a String. Wildcards can also be used with SOSL queries which are not supported by SOQL queries. A wildcard is a keyboard character such as an asterisk (*) or a question mark (?) that is used to represent one or more characters when you are searching for records.

SOSL supports two wildcards:

```
List<List<sObject>> results = [FIND 'Univ*'....];
List<List<sObject>> results = [FIND 'Jo?n'....];
List<List<sObject>> results = [FIND 'Univ*' IN NAME FIELDS .....];
```

1. Returning: If we want to return texts from a particular object then we use returning keyword.
   Ex:

```
List<List<sObject>> results = [FIND 'Univ*' IN NAME FIELDS RETURNING Account,
Contact];
List<Account> accounts = (List<Account>)results[0];
List<Contact> contacts = (List<Contact>)results[1];
```

2. Return specified fields: If we want to search text in a particular field, we are going to use a search group.
   Find {contact} IN (searchgroup)

SearchGroup	**Description**
ALL FIELDS	Search all searchable fields. If the IN clause is unspecified, then ALL FIELDS is the default setting.
EMAIL FIELDS	Search only email fields.
NAME FIELDS	Search only name fields for standard objects.In addition to the Name field, Salesforce searches the following fields when using IN NAME FIELDS with these standard objects: Account: Website, Site, NameLocal Asset: SerialNumber Case: SuppliedName, SuppliedCompany, Subject Contact: AssistantName, FirstNameLocal, LastNameLocal, AccountName Event: Subject Lead: Company, CompanyLocal, FirstNameLocal, LastNameLocal Note: Title PermissionSet: Label Report: Description TagDefinition: NormName Task: Subject User: CommunityNickname In custom objects, fields that are defined as “Name Field” are searched. In standard and custom objects, name fields have the nameField property set to true.
PHONE FIELDS	Search only phone number fields.
SIDEBAR FIELDS	Search for valid records as listed in the Sidebar dropdown list. Unlike search in the application, the asterisk (*) wildcard isn’t appended to the end of a search string.

Find {contact} IN (searchgroup) returning objects & fields.

```
List<List<sObject>> results = [FIND 'Univ*' IN NAME FIELDS RETURNING
       Account(Name, BillingCountry), Contact(FirstName, LastName)];
List<Account> accounts = (List<Account>)results[0];
system.debug(accounts[0].Name);
```

Note: Return Type of SOSL is list of list of sObjects which is the parent class of all Objects.

Note: In Apex use single quotes ‘ ’ instead of curly braces { }.

WHERE Clause:

```
List<List<sObject>> results = [FIND 'Univ*' IN NAME FIELDS
                              RETURNING Account(Name, BillingCountry
                              WHERE CreatedDate < TODAY)];
```

IN Clause:

Specify which types of text fields to search for on a SOSL query by using the IN SearchGroup optional clause. For example you can search name, email, phone, sidebar, or all fields.

You can specify one of the following values (note that numeric fields are not searchable). If not specified the default behavior is to search all text fields in searchable objects.

Ex:

```
List<List<SObject>> soList = [Find ‘Test?’ IN ALL FIELDS RETURNING Account, Contact];
List<Account> aList = soList[0];
List<Contact> conList = soList[1];
for(Account a: aList)
{
       System.debug(a);
}
for(Contact c: conList)
{
       System.debug(c);
}
```

Note: In java, class casts exceptions.

Note: In standard and custom objects, name fields have the NAME FIELD property set to true, in order to make it searchable.

Some standard fields in standard objects already have this property set to true like:

1. Account: Website, Site, NameLocal
2. Asset: SerialNumber
3. Case: SuppliedName, SuppliedCompany, Subject
4. Contact: AssistantName, FirstNameLocal, LastNameLocal, AccountName
5. Event: Subject
6. Lead: Company, CompanyLocal, FirstNameLocal, LastNameLocal
7. Note: Title
8. PermissionSet: Label
9. Report: Description
10. TagDefinition: NormName
11. Task: Subject
12. User: CommunityNickname

Note: If you want to specify a WHERE clause, you must include a field list with at least one specific field.

```
FIND {upsert} RETURNING Account(Name), Contact, Student__c(Name WHERE
Registered_for_course__c= 'Salesforce' AND intitial_fees > 60)
```

ORDER BY Clause:

```
Find {Test} Returning Account(Name, Phone, WHERE website!=null ORDER
BY Name DESC), Contact(Name ORDER BY Name)
```

LIMIT Clause:

```
Find {Test} RETURNING Account(Name, Phone LIMIT 50)
```

Note: Limit should always be the last statement.
Note: Numeric fields are not searchable.
Note: It is recommended to specify the search scope unless you need to search all fields.

Dynamic SOSL:

Dynamic SOSL refers to the creation of a SOSL string at run time with Apex code. Dynamic SOSL enables you to create more flexible applications. For example, you can create a search based on input from an end user, or update records with varying field names.

Use Search.query() to create a dynamic SOSL query at run time.

```
public static void main(String objectName)
{
	String searchquery = 'FIND \'Edge*\'IN ALL FIELDS RETURNING Account(Id, Name),
	Contact, Lead';
	List<List<SObject>> searchList = Search.query(searchquery);
}
```

Note: Dynamic 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 dynamic SOSL query. From the example above, the results from Account are first, then Contact, then Lead.

The search query method can be used wherever an inline SOSL query can be used, such as in regular assignment statements and for loops. The results are processed in much the same way as static SOSL queries are processed.

SOQL vs. SOSL:

SOQL	SOSL
You know in which object the data you are searching for resides	You are not sure in which object the data might be.

The data needs to be retrieved from a single object or related objects.	You want to retrieve multiple objects and fields, which might not be related to each other.
We can perform DML operations on query results.	We can not perform DML operations.
It returns a list of sObject.	It returns a list of list of sObject.
You can count records using the count function.	You cannot count records using the count function.
Query all types of fields.	Query on specific scopes.