SOQL

Salesforce Object Query Language designed to work with SFDC Database. It can search a record on a given criterion only in single sObject.

Like SOSL, it cannot search across multiple objects but it does support nested queries.

SOQL Example

Consider our ongoing example of Chemical Company. Suppose, we need a list of records which are created today and whose customer name is not 'test'. In this case, we will have to use the SOQL query as given below −

// fetching the Records via SOQL

List<apex\_invoice\_\_c> InvoiceList = new List<apex\_invoice\_\_c>();

InvoiceList = [SELECT Id, Name, APEX\_Customer\_\_r.Name, APEX\_Status\_\_c FROM

APEX\_Invoice\_\_c WHERE createdDate = today AND APEX\_Customer\_\_r.Name != 'Test'];

// SOQL query for given criteria

// Printing the fetched records

System.debug('We have total '+InvoiceList.size()+' Records in List');

for (APEX\_Invoice\_\_c objInvoice: InvoiceList) {

System.debug('Record Value is '+objInvoice);

// Printing the Record fetched

}

You can run the SOQL query via the Query Editor in the Developer console as shown below.

Run the query given below in the Developer Console. Search for the Invoice records created today.

SELECT Id, Name, APEX\_Customer\_\_r.Name, APEX\_Status\_\_c FROM APEX\_Invoice\_\_c

WHERE createdDate = today

You must select the fields for which you need the values, otherwise, it can throw run time errors.

Traversing Relationship Fields

This is one of the most important parts in SFDC as many times we need to traverse through the parent child object relationship

Also, there may be cases when you need to insert two associated objects records in Database. For example, Invoice object has relationship with the Customer object and hence one Customer can have many invoices.

Suppose, you are creating the invoice and then you need to relate this invoice to Customer. You can use the following code for this functionality −

// Now create the invoice record and relate it with the Customer object

// Before executing this, please create a Customer Records with Name 'Customer

// Creation Test'

APEX\_Invoice\_\_c objInvoice = new APEX\_Invoice\_\_c();

// Relating Invoice to customer via id field of Customer object

objInvoice.APEX\_Customer\_\_c = [SELECT id FROM APEX\_Customer\_\_c WHERE Name =

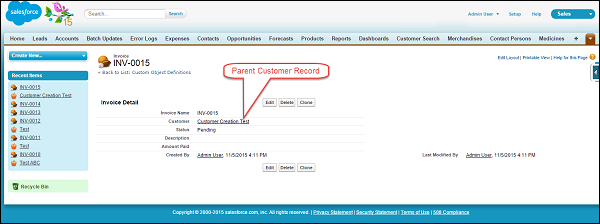
'Customer Creation Test' LIMIT 1].id;

objInvoice.APEX\_Status\_\_c = 'Pending';

insert objInvoice; //Creating Invoice

System.debug('Newly Created Invoice'+objInvoice); //Newly created invoice

Execute this code snippet in the Developer Console. Once executed, copy the Id of invoice from the Developer console and then open the same in SFDC as shown below. You can see that the Parent record has already been assigned to Invoice record as shown below.



Fetching Child Records

Let us now consider an example wherein, all the invoices related to particular customer record need to be in one place. For this, you must know the child relationship name. To see the child relationship name, go to the field detail page on the child object and check the "Child Relationship" value. In our example, it is invoices appended by \_\_r at the end.

Example

In this example, we will need to set up data, create a customer with name as 'ABC Customer' record and then add 3 invoices to that customer.

Now, we will fetch the invoices the Customer 'ABC Customer' has. Following is the query for the same −

// Fetching Child Records using SOQL

List<apex\_customer\_\_c> ListCustomers = [SELECT Name, Id,

(SELECT id, Name FROM Invoices\_\_r) FROM APEX\_Customer\_\_c WHERE Name = 'ABC Customer'];

// Query for fetching the Child records along with Parent

System.debug('ListCustomers '+ListCustomers); // Parent Record

List<apex\_invoice\_\_c> ListOfInvoices = ListCustomers[0].Invoices\_\_r;

// By this notation, you could fetch the child records and save it in List

System.debug('ListOfInvoices values of Child '+ListOfInvoices);

// Child records

You can see the Record values in the Debug logs.

Fetching Parent Record

Suppose, you need to fetch the Customer Name of Invoice the creation date of which is today, then you can use the query given below for the same −

Example

Fetch the Parent record's value along with the child object.

// Fetching Parent Record Field value using SOQL

List<apex\_invoice\_\_c> ListOfInvoicesWithCustomerName = new List<apex\_invoice\_\_c>();

ListOfInvoicesWithCustomerName = [SELECT Name, id, APEX\_Customer\_\_r.Name

FROM APEX\_Invoice\_\_c LIMIT 10];

// Fetching the Parent record's values

for (APEX\_Invoice\_\_c objInv: ListOfInvoicesWithCustomerName) {

System.debug('Invoice Customer Name is '+objInv.APEX\_Customer\_\_r.Name);

// Will print the values, all the Customer Records will be printed

}

Here we have used the notation APEX\_Customer\_\_r.Name, where APEX\_Customer\_\_r is parent relationship name, here you have to append the \_\_r at the end of the Parent field and then you can fetch the parent field value.

Aggregate Functions

SOQL does have aggregate function as we have in SQL. Aggregate functions allow us to roll up and summarize the data. Let us now understand the function in detail.

Suppose, you wanted to know that what is the average revenue we are getting from Customer 'ABC Customer', then you can use this function to take up the average.

Example

// Getting Average of all the invoices for a Perticular Customer

AggregateResult[] groupedResults = [SELECT

AVG(APEX\_Amount\_Paid\_\_c)averageAmount FROM APEX\_Invoice\_\_c WHERE

APEX\_Customer\_\_r.Name = 'ABC Customer'];

Object avgPaidAmount = groupedResults[0].get('averageAmount');

System.debug('Total Average Amount Received From Customer ABC is '+avgPaidAmount);

Check the output in Debug logs. Note that any query that includes an aggregate function returns its results in an array of **AggregateResult** objects. AggregateResult is a readonly sObject and is only used for query results. It is useful when we need to generate the Report on Large data.

There are other aggregate functions as well which you can be used to perform data summary.

**MIN()** − This can be used to find the minimum value

**MAX()** − This can be used to find the maximum value.

**COUNT() –**

**SUM() -**

Binding Apex Variables

You can use the Apex variable in SOQL query to fetch the desired results. Apex variables can be referenced by the Colon (:) notation.

Example

// Apex Variable Reference

String CustomerName = 'ABC Customer';

List<apex\_customer\_\_c> ListCustomer = [SELECT Id, Name FROM APEX\_Customer\_\_c

WHERE Name = :CustomerName];

List<apex\_customer\_\_c> ListCustomer = [SELECT MAX(ID) FROM APEX\_Customer\_\_c

WHERE Name = :CustomerName];

// Query Using Apex variable

System.debug('ListCustomer Name'+ListCustomer); // Customer Name

**SOQL - COUNT()**

Getting the "Count" of results being returned in a SOQL data set is pretty simple as well. For example, if I wanted to know how many Leads were going to be returned in my SELECT statement above, I can use the COUNT() function below:

**SELECT COUNT() from Lead WHERE email = 'john.doe@somecompany.com'**

**SOQL Comparison Operators**

|  |  |
| --- | --- |
| **Operator** | **Common name** |
| = | Equals |
| != | Not equals |
| < | Less than |
| <= | Less than or equal |
| > | Greater than |
| >= | Greater than or equal |
| IN | In |
| NOT IN | Not in (WHERE clause) |
| INCLUDES EXCLUDES | Applies to multi-select picklists |
| LIKE | Like (see section below) |

For a full chart of all the comparison operators, check out [this Salesforce.com developer documentation page](http://www.salesforce.com/us/developer/docs/api/index_Left.htm#CSHID=sforce_api_calls_soql_select.htm|StartTopic=Content%2Fsforce_api_calls_soql_select.htm|SkinName=webhelp).

**SOQL - Like Operator**

The LIKE operator provides a way to match partial text strings and includes support for wildcards. Let's say for a moment we want to find all the Leads where the email domain is the same. For this, we can use a "LIKE" operator.  He is an example of a LIKE statement with the % wildcard.

**SELECT Id, Name from Lead WHERE email  LIKE '%somecompany.com'**

 The placement of the percent sign '%' is key here. I am basically saying, bring me back all the Leads where the email ends with "somecompany.com". Therefore I place the '%' at the beginning of whatever I am looking for." Anything to the left of the % sign is ignored in the search. If I didn't know the full domain I could use the following statement:

**SELECT Id, Name from Lead WHERE email  LIKE '%somecomp%'**

This is going to return all the leads where the email contains "somecomp".

Other wildcard is the underscore "\_". Thing is used to match exactly one character.

Note: Unlike with SQL, the LIKE operator in SOQL performs a case-insensitive match.

**SOQL - WHERE/OR**

 If you want to extend the WHERE clause to include multiple values, you can OR. See the example statement below:

**SELECT ProductCode FROM PricebookEntry WHERE CurrencyIsoCode = 'USD' or CurrencyIsoCode = 'GBP'**

Taking it a step further, you can evaludate multiple things in the WHERE clause:

**SELECT ProductCode,UnitPrice FROM PricebookEntry   
WHERE (UnitPrice >= 10 and CurrencyIsoCode='USD')   
OR (UnitPrice >= 5.47 and CurrencyIsoCode='EUR')**

SOSL

Searching the text string across the object and across the field will be done by using SOSL. This is Salesforce Object Search Language. It has the capability of searching a particular string across multiple objects.

SOSL statements evaluate to a list of sObjects, wherein, 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.

SOSL Query Example

Consider a business case wherein, we need to develop a program which can search a specified string. Suppose, we need to search for string 'ABC' in the Customer Name field of Invoice object. The code goes as follows −

First, you have to create a single record in Invoice object with Customer name as 'ABC' so that we can get valid result when searched.

// Program To Search the given string in all Object

// List to hold the returned results of sObject generic type

List<list<SObject>> invoiceSearchList = new List<List<SObject>>();

// SOSL query which will search for 'ABC' string in Customer Name field of Invoice Object

invoiceSearchList = [FIND 'ABC\*' IN ALL FIELDS RETURNING APEX\_Invoice\_c

(Id,APEX\_Customer\_r.Name)];

// Returned result will be printed

System.debug('Search Result '+invoiceSearchList);

// Now suppose, you would like to search string 'ABC' in two objects,

// that is Invoice and Account. Then for this query goes like this:

// Program To Search the given string in Invoice and Account object,

// you could specify more objects if you want, create an Account with Name as ABC.

// List to hold the returned results of sObject generic type

List<List<SObject>> invoiceAndSearchList = new List<List<SObject>>();

// SOSL query which will search for 'ABC' string in Invoice and in Account object's fields

invoiceAndSearchList = [FIND 'ABC\*' IN ALL FIELDS RETURNING APEX\_Invoice\_\_c

(Id,APEX\_Customer\_\_r.Name), Account];

// Returned result will be printed

System.debug('Search Result '+invoiceAndSearchList);

// This list will hold the returned results for Invoice Object

APEX\_Invoice\_\_c [] searchedInvoice = ((List<APEX\_Invoice\_c>)invoiceAndSearchList[0]);

// This list will hold the returned results for Account Object

Account [] searchedAccount = ((List<Account>)invoiceAndSearchList[1]);

System.debug('Value of searchedInvoice'+searchedInvoice+'Value of searchedAccount'

+ searchedAccount);

SOQL

This is almost the same as SOQL. You can use this to fetch the object records from one object only at a time. You can write nested queries and also fetch the records from parent or child object on which you are querying now.