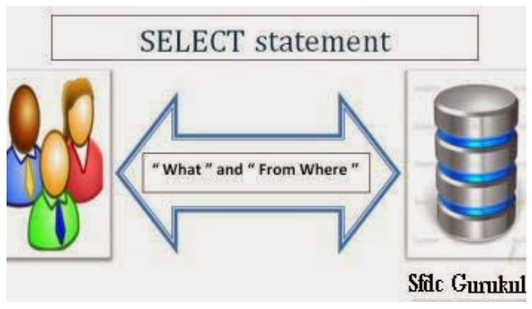
How to use SELECT Query in SOQL (SALESFORCE OBJECT QUERY LANGUAGE)?

0 Unknown Sunday, 26 April 2015

SELECT Query in SOQL

The Salesforce Object Query Language (SOQL) to construct simple but powerful query strings to select records from salesforce objects. Similar to the SELECT command in Structured Query Language (SQL), SOQL allows you to specify the source object (such as Account), a list of fields to retrieve, and conditions for selecting rows in the source object.



Syntax:
SELECT <fields> FROM <object>

Example:
The following SOQL query returns the value of the Id and Name field for all Account records.

```
SELECT Id, Name FROM Account
```

Now we will see the Each one with example.
LIMIT:
Use LIMIT to specify the maximum number of rows to return:

Syntax:

```
SELECT fieldList
FROM objectType
[WHERE conditionExpression]
LIMIT number_of_rows
```

Example: This Query returns the first 100 Account records whose Industry is IT.

```
SELECT Name
FROM Account
WHERE Industry = 'It' LIMIT 100
```

Syntax	Description
fieldList	Specifies a list of one or more fields, separated by commas, that you want to retrieve from the specified object. ▪ SELECT **Id, Name, BillingCity** FROM Account ▪ SELECT **count()** FROM Contact ▪ SELECT **Contact.Firstname, Contact.Account.Name** FROM Contact
objectType	Specifies the object on which you are writing the query. You must specify a valid object, such as Account, and must have read-level permissions to that object.
conditionExpression	If WHERE is specified, determines which rows and values in the specified object (objectType) to filter against. If unspecified, the query() retrieves all the rows in the object that are visible to the user.
filteringExpression	If WITH DATA CATEGORY is specified, the query() only returns matching records that are associated with the specified data categories and are visible to the user. If unspecified, the query() returns the matching records that are visible to the user.
fieldGroupByList	Specifies a list of one or more fields, separated by commas, that are used to group the query results. A GROUP BY clause is used with aggregate functions to summarize the data and enable you to roll up query results rather than having to process the individual records in your code.
fieldSubtotalGroupByList	Specifies a list of up to three fields, separated by commas, that are used to group the query results. The results include extra subtotal rows for the grouped data.
havingConditionExpression	If the query includes a GROUP BY clause, this conditional expression filters the records that the GROUP BY returns.
fieldOrderByList	Specifies a list of one or more fields, separated by commas, that are used to order the query results. For example, you can query for contacts and order the results by last name, and then by first name: SELECT Id, LastName, FirstName FROM Contact ORDER BY LastName, FirstName

The below query selects the Id and Name fields of all records from the standard object Account and stores them in the list acclist.

```
List<Account> acclist = [SELECT Id, Name FROM Account];
```

The above query return more than one record, and it is moved into a collection and not a single variable.

Below are the lists the comparison Operator values that can be used in WHERE Clause:

Operator	Name	Description
=	Equals	Expression is true if the value in the specified *fieldName* equals the specified *value* in the expression. String comparisons using the equals operator are case-insensitive.
!=	Not equals	Expression is true if the value in the specified *fieldName* does not equal the specified *value*.
<	Less than	Expression is true if the value in the specified *fieldName* is less than the specified *value*.
<=	Less or equal	Expression is true if the value in the specified *fieldName* is less than, or equals, the specified *value*.
>	Greater than	Expression is true if the value in the specified *fieldName* is greater than the specified *value*.

>=	Greater or equal	Expression is true if the value in the specified *fieldName* is greater than or equal to the specified *value*.
LIKE	Like	Expression is true if the value in the specified *fieldName* matches the characters of the text string in the specified *value*. The LIKE operator in SOQL and SOSL is similar to the LIKE operator in SQL; it provides a mechanism for matching partial text strings and includes support for wildcards. • The % and _ wildcards are supported for the LIKE operator. • The % wildcard matches zero or more characters. ▪ The _ wildcard matches exactly one character. ▪ The text string in the specified *value* must be enclosed in single quotes. ▪ The LIKE operator is supported for string fields only. ▪ The LIKE operator performs a case-insensitive match, unlike the case-sensitive matching in SQL. ▪ The LIKE operator in SOQL and SOSL supports escaping of special characters % or _. ▪ Do not use the backslash character in a search except to escape a character. For example, the following query matches Appleton, Apple, and Bappl , but not Appl: `SELECT AccountId, FirstName, lastname FROM Contact WHERE lastname LIKE 'appl_%'`
IN	IN	If the value equals any one of the specified values in a WHERE clause. For example: `SELECT Name FROM Account WHERE BillingState IN ('California', 'New York')` Note that the values for IN must be in parentheses. String values must be surrounded by single quotes. IN and NOT IN can also be used for semi-joins and anti-joins when querying on ID (primary key) or reference (foreign key) fields.
NOT IN	NOT IN	If the value does not equal any of the specified values in a WHERE clause. For example: `SELECT Name FROM Account WHERE BillingState NOT IN ('California', 'New York')` Note that the values for NOT IN must be in parentheses, and string values must be surrounded by single quotes. There is also a logical operator NOT, which is unrelated to this comparison operator.

Condition Expression:

The conditionExpression in the WHERE clause in a SOQL statement uses the following syntax:

```
fieldExpression [ logicalOperator fieldExpression2 ... ]
```

You can add multiple field expressions to a condition expression by using logical operators.

Using LIKE:

```
SELECT Name FROM Account WHERE Name like 'A%'
```

Using AND:

```
SELECT Id, Name FROM Account WHERE Name = 'Sandy' AND Industry = â��Bankingâ��
```

Using OR:

```
SELECT Id, Name FROM Account WHERE Name = 'Sandy' OR Name = â��Jackyâ��
```

Using both AND & OR condition in the query:

```
SELECT Id, Name FROM Account
WHERE (Name = 'Sandy' OR Name = â��Jackyâ��) AND Type = â��Prospectâ��
```

Using Greater then or Less then:

```
SELECT Name FROM Account WHERE CreatedDate > 2011-04-26T10:00:00-08:00
```

Using Equal:

```
SELECT Amount FROM Opportunity WHERE CALENDAR_YEAR(CreatedDate) = 2011
```

Using null:

Use the value null to represent null values in SOQL queries.

For example, the following statement would return the account IDs of all events with a non-null activity date:

```
SELECT AccountId FROM Event WHERE ActivityDate != null
```

Using tolabel():

A client application can have results from a query returned that are translated into the user’s language, Using toLabel():

```
SELECT Company, toLabel(Recordtype.Name) FROM Lead
```

Using Multi-Select Picklists:

The following operators are supported for querying multi-select picklists:

Operators	Description
=	Equals the specified string.
!=	Does not equal the specified string.
includes	Includes (contains) the specified string.
excludes	Excludes (does not contain) the specified string.
;	Specifies AND for two or more strings. Use ; for multi-select picklists when two or more items must be selected. For example: 'AAA;BBB'

Using Multi-select picklist in the query:

Examples:

The following query filters on values in the MSP1__c field that are equal to AAA and BBB selected (exact match):

```
SELECT Id, MSP1__c FROM CustObj__c WHERE MSP1__c = 'AAA;BBB'
```

The following query filters on values in the MSP1__c field that contains either of these values:
• AAA and BBB selected.
• CCC selected.

```
SELECT Id, MSP1__c from CustObj__c WHERE MSP1__c includes ('AAA;BBB','CCC')
```

Semi-Joins with IN and Anti-Joins with NOT IN:

You can query values in a field where another field on the same object has a specified set of values, using IN.
In addition, you can create more complex queries by replacing the list of values in the IN or NOT IN clause with a subquery. The subquery can filter by ID (primary key) or reference (foreign key) fields. A semi-join is a subquery on another object in an IN clause to restrict the records returned. An anti-join is a subquery on another object in a NOT IN clause to restrict the records returned.
Sample uses of semi-joins and anti-joins include:
• Get all contacts for accounts that have an opportunity with a particular record type.
• Get all open opportunities for accounts that have active contracts.
• Get all open cases for contacts that are the decision maker on an opportunity.
• Get all accounts that do not have any open opportunities.
Example:

```
SELECT Name FROM Account WHERE BillingState IN ('California', 'New York')
```