accepted

**Definition**

[SOQL (Salesforce Object Query Language)](http://www.salesforce.com/us/developer/docs/api/Content/sforce_api_calls_soql.htm)

Use the Salesforce Object Query Language (SOQL) to construct simple but powerful query strings in the queryString parameter in the query() call, in Apex statements, in Visualforce controllers and getter methods, or in the Schema Explorer of the Force.com IDE.

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. SOQL uses the SELECT statement combined with filtering statements to return sets of data, which may optionally be ordered.

[SOSL (Salesforce Object Search Language)](http://www.salesforce.com/us/developer/docs/soql_sosl/index_Left.htm#CSHID=sforce_api_calls_sosl.htm|StartTopic=Content%2Fsforce_api_calls_sosl.htm|SkinName=webhelp)

Use the Salesforce Object Search Language (SOSL) to construct text searches in the search() call, in Apex statements, in Visualforce controllers and getter methods, or the Schema Explorer of the Eclipse Toolkit.

Unlike SOQL, which can only query one object at a time, SOSL enables you to search text, email, and phone fields for multiple objects simultaneously.

**Speed**

SOQL and SOSL have different indexes. An [index](http://en.wikipedia.org/wiki/Database_index) makes it much faster to filter queries.

[SOQL indexes](http://www.salesforce.com/us/developer/docs/apexcode/Content/langCon_apex_SOQL_VLSQ.htm#EffSOQLSection) are:

* Primary keys (Id, Name and Owner fields)
* Foreign keys (lookup or master-detail relationship fields)
* Audit dates (such as LastModifiedDate)
* Custom fields marked as External ID or Unique.

Fields that can't be indexed in SOQL are:

* Multi-select picklists
* Currency fields in a multicurrency organization
* Long text fields
* Some formula fields
* Binary fields (fields of type blob, file, or encrypted text.)

Note that new data types, typically complex ones, may be added to Salesforce and fields of these types may not allow custom indexing.

SOSL indexes are:

*This is the one point where my discussion is weak. I simply can't seem to find Salesforce documentation on the SOSL indexes. I know there are standard fields like Name that are indexed, but I can't find the documentation for all of it. If anyone can post a comment and/or edit the post here to include that info, I would really appreciate it.*

**Limits**

[SOQL](http://www.salesforce.com/us/developer/docs/soql_sosl/Content/sforce_api_calls_soql_limits.htm) and [SOSL](http://www.salesforce.com/us/developer/docs/soql_sosl/Content/sforce_api_calls_sosl_limits.htm) generally have the same limitations, however according to the [Governer Limit documentation](http://www.salesforce.com/us/developer/docs/apexcode/Content/apex_gov_limits.htm):

\*\*Description\*\* \*\*Limit\*\*

Total number of SOQL queries issued 100

Total number of SOQL queries issued for Batch Apex and future methods 200

Total number of records retrieved by SOQL queries 50,000

Total number of records retrieved by Database.getQueryLocator 10,000

Total number of SOSL queries issued 20

Total number of records retrieved by a single SOSL query 200

In addition:

* If a SOQL query runs more than 120 seconds, the request can be canceled by Salesforce.

# Write SOQL Queries

## Learning Objectives

After completing this unit, you'll be able to:

* Write SOQL queries in Apex.
* Execute SOQL queries by using the Query Editor in the Developer Console.
* Execute SOQL queries embedded in Apex by using Anonymous Apex.
* Query related records.

**Syntax**

[SOQL syntax](http://www.salesforce.com/us/developer/docs/soql_sosl/Content/sforce_api_calls_soql_select.htm) and [SOSL syntax](http://www.salesforce.com/us/developer/docs/soql_sosl/Content/sforce_api_calls_sosl_syntax.htm) differ greatly. For a truly in depth break, please refer to their respective documentation. However, a simple example of each is:

*SOQL*

SELECT Id, Name FROM Account WHERE Name = 'Acme'

Return all Accounts where the Name is exactly Acme.

*SOSL*

FIND {Joe Smith} IN Name Fields RETURNING lead(name, phone)

Look for the name Joe Smith in the name field of a lead and return the name and phone number.

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

## 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.

## 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];

// Query Using Apex variable

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

## Write SOQL Queries

To read a record from Salesforce, you need to write a query. Salesforce provides the Salesforce Object Query Language, or SOQL in short, that you can use to read saved records. SOQL is similar to the standard SQL language but is customized for the Lightning Platform.

Because Apex has direct access to Salesforce records that are stored in the database, you can embed SOQL queries in your Apex code and get results in a straightforward fashion. When SOQL is embedded in Apex, it is referred to as **inline SOQL**.

To include SOQL queries within your Apex code, wrap the SOQL statement within square brackets and assign the return value to an array of sObjects. For example, the following retrieves all account records with two fields, Name and Phone, and returns an array of Account sObjects.

Account[] accts = [SELECT Name,Phone FROM Account];

Copy

### Prerequisites

Some queries in this unit expect the org to have accounts and contacts. Before you run the queries, create some sample data.

1. In the Developer Console, open the Execute Anonymous window from the **Debug** menu.
2. Insert the below snippet in the window and click **Execute**.

// Add account and related contact

Account acct = new Account(

Name=**'SFDC Computing'**,

Phone=**'(415)555-1212'**,

NumberOfEmployees=50,

BillingCity=**'San Francisco'**);

insert acct;

// Once the account is inserted, the sObject will be

// populated with an ID.

// Get this ID.

ID acctID = acct.ID;

// Add a contact to this account.

Contact con = new Contact(

FirstName=**'Carol'**,

LastName=**'Ruiz'**,

Phone=**'(415)555-1212'**,

Department=**'Wingo'**,

AccountId=acctID);

insert con;

// Add account with no contact

Account acct2 = new Account(

Name=**'The SFDC Query Man'**,

Phone=**'(310)555-1213'**,

NumberOfEmployees=50,

BillingCity=**'Los Angeles'**,

Description=**'Expert in wing technologies.'**);

insert acct2;

Copy

### Using the Query Editor

The Developer Console provides the Query Editor console, which enables you to run your SOQL queries and view results. The Query Editor provides a quick way to inspect the database. It is a good way to test your SOQL queries before adding them to your Apex code. When you use the Query Editor, you need to supply only the SOQL statement without the Apex code that surrounds it.

Let’s try running the following SOQL example:

1. In the Developer Console, click the Query Editor tab.
2. Copy and paste the following into the first box under Query Editor, and then click **Execute**.

SELECT Name,Phone FROM Account

Copy

All account records in your org appear in the Query Results section as rows with fields.

### Basic SOQL Syntax

This is the syntax of a basic SOQL query:

SELECT fields FROM ObjectName [WHERE Condition]

Copy

The WHERE clause is optional. Let’s start with a very simple query. For example, the following query retrieves accounts and gets two fields for each account: the ID and the Phone.

SELECT Name,Phone FROM Account

Copy

The query has two parts:

1. SELECT Name,Phone: This part lists the fields that you would like to retrieve. The fields are specified after the SELECT keyword in a comma-delimited list. Or you can specify only one field, in which case no comma is necessary (e.g. SELECT Phone).
2. FROM Account: This part specifies the standard or custom object that you want to retrieve. In this example, it’s Account. For a custom object called Invoice\_Statement, it is Invoice\_Statement\_\_c.

#### Beyond the Basics

Unlike other SQL languages, you can’t specify \* for all fields. You must specify every field you want to get explicitly. If you try to access a field you haven’t specified in the SELECT clause, you’ll get an error because the field hasn’t been retrieved.

You don’t need to specify the Id field in the query as it is always returned in Apex queries, whether it is specified in the query or not. For example: SELECT Id,Phone FROM Account and SELECT Phone FROM Account are equivalent statements. The only time you may want to specify the Id field if it is the only field you’re retrieving because you have to list at least one field: SELECT Id FROM Account. You may want to specify the Id field also when running a query in the Query Editor as the ID field won’t be displayed unless specified.

### Filtering Query Results with Conditions

If you have more than one account in the org, they will all be returned. If you want to limit the accounts returned to accounts that fulfill a certain condition, you can add this condition inside the WHERE clause. The following example retrieves only the accounts whose names are SFDC Computing. Note that comparisons on strings are case-insensitive.

SELECT Name,Phone FROM Account WHERE Name=**'SFDC Computing'**

Copy

The WHERE clause can contain multiple conditions that are grouped by using logical operators (AND, OR) and parentheses. For example, this query returns all accounts whose name is SFDC Computing that have more than 25 employees:

SELECT Name,Phone FROM Account WHERE (Name=**'SFDC Computing'** AND NumberOfEmployees>25)

Copy

This is another example with a more complex condition. This query returns all SFDC Computing accounts, or all accounts with more than 25 employees whose billing city is Los Angeles.

SELECT Name,Phone FROM Account WHERE (Name=**'SFDC Computing'** OR (NumberOfEmployees>25 AND BillingCity=**'Los Angeles'**))

Copy

#### Beyond the Basics

Instead of using the equal operator (=) for comparison, you can perform fuzzy matches by using the LIKEoperator. For example, you can retrieve all accounts whose names start with SFDC by using this condition: WHERE Name LIKE 'SFDC%'. The % wildcard character matches any or no character. The \_ character in contrast can be used to match just one character.

### Ordering Query Results

When a query executes, it returns records from Salesforce in no particular order, so you can’t rely on the order of records in the returned array to be the same each time the query is run. You can however choose to sort the returned record set by adding an ORDER BY clause and specifying the field by which the record set should be sorted. This example sorts all retrieved accounts based on the Name field.

SELECT Name,Phone FROM Account ORDER BY Name

Copy

The default sort order is in alphabetical order, specified as ASC. The previous statement is equivalent to:

SELECT Name,Phone FROM Account ORDER BY Name ASC

Copy

To reverse the order, use the DESC keyword for descending order:

SELECT Name,Phone FROM Account ORDER BY Name DESC

Copy

You can sort on most fields, including numeric and text fields. You can’t sort on fields like rich text and multi-select picklists.

Try these SOQL statements in the Query Editor and see how the order of the returned record changes based on the Name field.

### Limiting the Number of Records Returned

You can limit the number of records returned to an arbitrary number by adding the LIMIT n clause where n is the number of records you want returned. Limiting the result set is handy when you don’t care which records are returned, but you just want to work with a subset of records. For example, this query retrieves the first account that is returned. Notice that the returned value is one account and not an array when using LIMIT 1.

Account oneAccountOnly = [SELECT Name,Phone FROM Account LIMIT 1];

Copy

### Combining All Pieces Together

You can combine the optional clauses in one query, in the following order:

SELECT Name,Phone FROM Account

WHERE (Name = **'SFDC Computing'** AND NumberOfEmployees>25)

ORDER BY Name

LIMIT 10

Copy

Execute the following SOQL query in Apex by using the Execute Anonymous window in the Developer Console. Then inspect the debug statements in the debug log. One sample account should be returned.

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.'**);

// Write all account array info

System.**debug**(accts);

Copy

### Accessing Variables in SOQL Queries

SOQL statements in Apex can reference Apex code variables and expressions if they are preceded by a colon (:). The use of a local variable within a SOQL statement is called a **bind**.

This example shows how to use the targetDepartment variable in the WHERE clause.

String targetDepartment = **'Wingo'**;

Contact[] techContacts = [SELECT FirstName,LastName

FROM Contact WHERE Department=:targetDepartment];

Copy

### Querying Related Records

Records in Salesforce can be linked to each other through relationships: lookup relationships or master-detail relationships. For example, the Contact has a lookup relationship to Account. When you create or update a contact, you can associate it with an account. The contacts that are associated with the same account appear in a related list on the account’s page. In the same way you can view related records in the Salesforce user interface, you can query related records in SOQL.

To get child records related to a parent record, add an inner query for the child records. The FROM clause of the inner query runs against the relationship name, rather than a Salesforce object name. This example contains an inner query to get all contacts that are associated with each returned account. The FROM clause specifies the Contacts relationship, which is a default relationship on Account that links accounts and contacts.

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

Copy

This next example embeds the example SOQL query in Apex and shows how to get the child records from the SOQL result by using the Contacts relationship name on the sObject.

Account[] acctsWithContacts = [SELECT Name, (SELECT FirstName,LastName FROM Contacts)

FROM Account

WHERE Name = **'SFDC Computing'**];

// Get child records

Contact[] cts = acctsWithContacts[0].Contacts;

System.**debug**(**'Name of first associated contact: '**

+ cts[0].FirstName + **', '** + cts[0].LastName);

Copy

You can traverse a relationship from a child object (contact) to a field on its parent (Account.Name) by using dot notation. For example, the following Apex snippet queries contact records whose first name is Carol and is able to retrieve the name of Carol’s associated account by traversing the relationship between accounts and contacts.

Contact[] cts = [SELECT Account.Name FROM Contact

WHERE FirstName = **'Carol'** AND LastName=**'Ruiz'**];

Contact carol = cts[0];

String acctName = carol.Account.Name;

System.**debug**(**'Carol\'s account name is '** + acctName);

SOSL

## 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);

DML

## Insert Operation

Insert operation is used to create new records in Database. You can create records of any Standard or Custom object using the Insert DML statement.

**Example**

We can create new records in APEX\_Invoice\_\_c object as new invoices are being generated for new customer orders every day. We will create a Customer record first and then we can create an Invoice record for that new Customer record.

// fetch the invoices created today, Note, you must have at least one invoice

// created today

List<apex\_invoice\_\_c> invoiceList = [SELECT id, Name, APEX\_Status\_\_c,

createdDate FROM APEX\_Invoice\_\_c WHERE createdDate = today];

// create List to hold the updated invoice records

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

APEX\_Customer\_\_c objCust = new APEX\_Customer\_\_C();

objCust.Name = 'Test ABC';

//DML for Inserting the new Customer Records

insert objCust;

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

if (objInvoice.APEX\_Status\_\_c == 'Pending') {

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

updatedInvoiceList.add(objInvoice);

}

}

// DML Statement to update the invoice status

update updatedInvoiceList;

// Prints the value of updated invoices

System.debug('List has been updated and updated values are' + updatedInvoiceList);

// Inserting the New Records using insert DML statement

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

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

objNewInvoice.APEX\_Amount\_Paid\_\_c = 1000;

objNewInvoice.APEX\_Customer\_\_c = objCust.id;

// DML which is creating the new Invoice record which will be linked with newly

// created Customer record

insert objNewInvoice;

System.debug('New Invoice Id is '+objNewInvoice.id+' and the Invoice Number is'

+ objNewInvoice.Name);

## Update Operation

Update operation is to perform updates on existing records. In this example, we will be updating the Status field of an existing Invoice record to 'Paid'.

**Example**

// Update Statement Example for updating the invoice status. You have to create

and Invoice records before executing this code. This program is updating the

record which is at index 0th position of the List.

// First, fetch the invoice created today

List<apex\_invoice\_\_c> invoiceList = [SELECT id, Name, APEX\_Status\_\_c,

createdDate FROM APEX\_Invoice\_\_c];

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

// Update the first record in the List

invoiceList[0].APEX\_Status\_\_c = 'Pending';

updatedInvoiceList.add(invoiceList[0]);

// DML Statement to update the invoice status

update updatedInvoiceList;

// Prints the value of updated invoices

System.debug('List has been updated and updated values of records are'

+ updatedInvoiceList[0]);

## Upsert Operation

Upsert Operation is used to perform an update operation and if the records to be updated are not present in database, then create new records as well.

**Example**

Suppose, the customer records in Customer object need to be updated. We will update the existing Customer record if it is already present, else create a new one. This will be based on the value of field APEX\_External\_Id\_\_c. This field will be our field to identify if the records are already present or not.

**Note** − Before executing this code, please create a record in Customer object with the external Id field value as '12341' and then execute the code given below −

// Example for upserting the Customer records

List<apex\_customer\_\_c> CustomerList = new List<apex\_customer\_\_c>();

for (Integer i = 0; i < 10; i++) {

apex\_customer\_\_c objcust=new apex\_customer\_\_c(name = 'Test' +i,

apex\_external\_id\_\_c='1234' +i);

customerlist.add(objcust);

} //Upserting the Customer Records

upsert CustomerList;

System.debug('Code iterated for 10 times and created 9 records as one record with

External Id 12341 is already present');

for (APEX\_Customer\_c objCustomer: CustomerList) {

if (objCustomer.APEX\_External\_Id\_c == '12341') {

system.debug('The Record which is already present is '+objCustomer);

}

}

## Delete Operation

You can perform the delete operation using the Delete DML.

**Example**

In this case, we will delete the invoices which have been created for the testing purpose, that is the ones which contain the name as 'Test'.

You can execute this snippet from the Developer console as well without creating the class.

// fetch the invoice created today

List<apex\_invoice\_\_c> invoiceList = [SELECT id, Name, APEX\_Status\_\_c,

createdDate FROM APEX\_Invoice\_\_c WHERE createdDate = today];

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

APEX\_Customer\_\_c objCust = new APEX\_Customer\_\_C();

objCust.Name = 'Test';

// Inserting the Customer Records

insert objCust;

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

if (objInvoice.APEX\_Status\_\_c == 'Pending') {

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

updatedInvoiceList.add(objInvoice);

}

}

// DML Statement to update the invoice status

update updatedInvoiceList;

// Prints the value of updated invoices

System.debug('List has been updated and updated values are' + updatedInvoiceList);

// Inserting the New Records using insert DML statement

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

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

objNewInvoice.APEX\_Amount\_Paid\_\_c = 1000;

objNewInvoice.APEX\_Customer\_\_c = objCust.id;

// DML which is creating the new record

insert objNewInvoice;

System.debug('New Invoice Id is' + objNewInvoice.id);

// Deleting the Test invoices from Database

// fetch the invoices which are created for Testing, Select name which Customer Name

// is Test.

List<apex\_invoice\_\_c> invoiceListToDelete = [SELECT id FROM APEX\_Invoice\_\_c

WHERE APEX\_Customer\_\_r.Name = 'Test'];

// DML Statement to delete the Invoices

delete invoiceListToDelete;

System.debug('Success, '+invoiceListToDelete.size()+' Records has been deleted');

## Undelete Operation

You can undelete the record which has been deleted and is present in Recycle bin. All the relationships which the deleted record has, will also be restored.

**Example**

Suppose, the Records deleted in the previous example need to be restored. This can be achieved using the following example. The code in the previous example has been modified for this example.

// fetch the invoice created today

List<apex\_invoice\_\_c> invoiceList = [SELECT id, Name, APEX\_Status\_\_c,

createdDate FROM APEX\_Invoice\_\_c WHERE createdDate = today];

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

APEX\_Customer\_\_c objCust = new APEX\_Customer\_\_C();

objCust.Name = 'Test';

// Inserting the Customer Records

insert objCust;

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

if (objInvoice.APEX\_Status\_\_c == 'Pending') {

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

updatedInvoiceList.add(objInvoice);

}

}

// DML Statement to update the invoice status

update updatedInvoiceList;

// Prints the value of updated invoices

System.debug('List has been updated and updated values are' + updatedInvoiceList);

// Inserting the New Records using insert DML statement

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

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

objNewInvoice.APEX\_Amount\_Paid\_\_c = 1000;

objNewInvoice.APEX\_Customer\_\_c = objCust.id;

// DML which is creating the new record

insert objNewInvoice;

System.debug('New Invoice Id is '+objNewInvoice.id);

// Deleting the Test invoices from Database

// fetch the invoices which are created for Testing, Select name which Customer Name

// is Test.

List<apex\_invoice\_\_c> invoiceListToDelete = [SELECT id FROM APEX\_Invoice\_\_c

WHERE APEX\_Customer\_\_r.Name = 'Test'];

// DML Statement to delete the Invoices

delete invoiceListToDelete;

system.debug('Deleted Record Count is ' + invoiceListToDelete.size());

System.debug('Success, '+invoiceListToDelete.size() + 'Records has been deleted');

// Restore the deleted records using undelete statement

undelete invoiceListToDelete;

System.debug('Undeleted Record count is '+invoiceListToDelete.size()+'. This should

be same as Deleted Record count');

## Differences between Database Methods and DML Statements

|  |  |
| --- | --- |
| **DML Statements** | **Database Methods** |
| Partial Update is not allowed. For example, if you have 20 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 as we have seen in the example. |
| **Example** − insert listName | **Example** − Database.insert(listName, False), where false indicate that partial update is not allowed. |

## Insert Operation

Inserting new records via database methods is also quite simple and flexible. Let us consider the previous scenario wherein, we have inserted new records using the DML statements. We will be inserting the same using Database methods.

### Example

// Insert Operation Using Database methods

// Insert Customer Records First using simple DML Statement. This Customer Record will be

// used when we will create Invoice Records

APEX\_Customer\_\_c objCust = new APEX\_Customer\_\_C();

objCust.Name = 'Test';

insert objCust; // Inserting the Customer Records

// Insert Operation Using Database methods

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

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

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

objNewInvoice.APEX\_Customer\_\_c = objCust.id;

objNewInvoice.APEX\_Amount\_Paid\_\_c = 1000;

InvoiceListToInsert.add(objNewInvoice);

Database.SaveResult[] srList = Database.insert(InvoiceListToInsert, false);

// Database method to insert the records in List

// Iterate through each returned result by the method

for (Database.SaveResult sr : srList) {

if (sr.isSuccess()) {

// This condition will be executed for successful records and will fetch the ids

// of successful records

System.debug('Successfully inserted Invoice. Invoice ID: ' + sr.getId());

// Get the invoice id of inserted Account

} else {

// This condition will be executed for failed records

for(Database.Error objErr : sr.getErrors()) {

System.debug('The following error has occurred.');

// Printing error message in Debug log

System.debug(objErr.getStatusCode() + ': ' + objErr.getMessage());

System.debug('Invoice oject field which are affected by the error:'

+ objErr.getFields());

}

}

}

## Update Operation

Let us now consider our business case example using the database methods. Suppose we need to update the status field of Invoice object but at the same time, we also require information like status of records, failed record ids, success count, etc. This is not possible by using DML Statements, hence we must use Database methods to get the status of our operation.

### Example

We will be updating the Invoice's 'Status' field if it is in status 'Pending' and date of creation is today.

The code given below will help in updating the Invoice records using the Database.update method. Also, create an Invoice record before executing this code.

// Code to update the records using the Database methods

List<apex\_invoice\_\_c> invoiceList = [SELECT id, Name, APEX\_Status\_\_c,

createdDate FROM APEX\_Invoice\_\_c WHERE createdDate = today];

// fetch the invoice created today

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

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

if (objInvoice.APEX\_Status\_\_c == 'Pending') {

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

updatedInvoiceList.add(objInvoice); //Adding records to the list

}

}

Database.SaveResult[] srList = Database.update(updatedInvoiceList, false);

// Database method to update the records in List

// Iterate through each returned result by the method

for (Database.SaveResult sr : srList) {

if (sr.isSuccess()) {

// This condition will be executed for successful records and will fetch

// the ids of successful records

System.debug('Successfully updated Invoice. Invoice ID is : ' + sr.getId());

} else {

// This condition will be executed for failed records

for(Database.Error objErr : sr.getErrors()) {

System.debug('The following error has occurred.');

// Printing error message in Debug log

System.debug(objErr.getStatusCode() + ': ' + objErr.getMessage());

System.debug('Invoice oject field which are affected by the error:'

+ objErr.getFields());

}

}

}