Apex - DML

### DML Statements

DML are the actions which are performed in order to perform insert, update, delete, upsert, restoring records, merging records, or converting leads operation.

DML is one of the most important part in Apex as almost every business case involves the changes and modifications to database.

### Database Methods

All operations which you can perform using DML statements can be performed using Database methods as well. Database methods are the system methods which you can use to perform DML operations. Database methods provide more flexibility as compared to DML Statements.

In this chapter, we will be looking at the first approach using DML Statements. We will look at the Database Methods in a subsequent chapter.

## DML Statements

Let us now consider the instance of the Chemical supplier company again. Our Invoice records have fields as Status, Amount Paid, Amount Remaining, Next Pay Date and Invoice Number. Invoices which have been created today and have their status as 'Pending', should be updated to 'Paid'.

## 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');

# Database Methods

## 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());

}

}

}

We will be looking at only the Insert and Update operations in this tutorial. The other operations are quite similar to these operations and what we did in the last chapter.

# Apex - 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);

## 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.

We will explore SOQL in the next chapter.

# Apex - SOQL

## 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 =

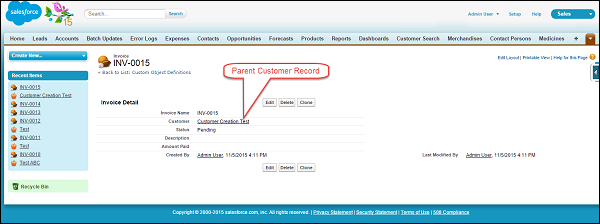
'Customer Creation Test' LIMIT 1].id;

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.

## 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