Apex DML Operations

You can perform DML operations using the Apex DML statements or the methods of the `Database` class. For lead conversion, use the `convertLead` method of the `Database` class. There is no DML counterpart for it.

See Also

- *Apex Developer Guide*: Working with Data in Apex
- Database Class

Apex DML Statements

Use Data Manipulation Language (DML) statements to insert, update, merge, delete, and restore data in Salesforce.

The following Apex DML statements are available:

Insert Statement

The `insert` DML operation adds one or more sObjects, such as individual accounts or contacts, to your organization's data. `insert` is analogous to the INSERT statement in SQL.

Syntax

```
insert sObject

insert sObject[]
```

Example

The following example inserts an account named 'Acme':

```
Account newAcct = new Account(name = 'Acme');
try {
   insert newAcct;
} catch (DmlException e) {
// Process exception here
}
```

> **Note**
>
> For more information on processing `DmlException`s, see Bulk DML Exception Handling.

Update Statement

The `update` DML operation modifies one or more existing sObject records, such as individual accounts or contacts, in your organization's data. `update` is analogous to the UPDATE statement in SQL.

Example

The following example updates the `BillingCity` field on a single account named 'Acme':

```
Account a = new Account(Name='Acme2');
insert(a);

Account myAcct = [SELECT Id, Name, BillingCity FROM Account WHERE Id = :a.Id];
myAcct.BillingCity = 'San Francisco';

try {
    update myAcct;
} catch (DmlException e) {
    // Process exception here
}
```

> **Note**
>
> For more information on processing `DmlException`s, see Bulk DML Exception Handling.

Upsert Statement

The `upsert` DML operation creates new records and updates sObject records within a single statement, using a specified field to determine the presence of existing objects, or the ID field if no field is specified.

Syntax

```
upsert sObject[opt_field]

upsert sObject[][opt_field]
```

The `upsert` statement matches the sObjects with existing records by comparing values of one field. If you don't specify a field when calling this statement, the `upsert` statement uses the sObject's ID to match the sObject with existing records in Salesforce. Alternatively, you can specify a field to use for matching. For custom objects, specify a custom field marked as external ID. For standard objects, you can specify any field that has the `idLookup` attribute set to true. For example, the Email field of Contact or User has the `idLookup` attribute set. To check a field's attribute, see the Object Reference for Salesforce.

Also, you can use foreign keys to upsert sObject records if they have been set as reference fields. For more information, see Field Types in the *Object Reference for Salesforce*.

The optional field parameter, *opt_field*, is a field token (of type `Schema.SObjectField`). For example, to specify the MyExternalID custom field, the statement is:

```
upsert sObjectList Account.Fields.MyExternalId__c;
```

If the field used for matching doesn't have the `Unique` attribute set, the context user must have the "View All Records" object-level permission for the target object or the "View All Data" permission so that `upsert` doesn't accidentally insert a duplicate record.

> **Note**
>
> Custom field matching is case-insensitive only if the custom field has the **Unique** and **Treat "ABC" and "abc" as duplicate values (case insensitive)** attributes selected as part

Upsert uses the sObject record's primary key (the ID), an idLookup field, or an external ID field to determine whether it should create a record or update an existing one:

- If the key isn't matched, a new object record is created.
- If the key is matched once, the existing object record is updated.
- If the key is matched multiple times, an error is generated and the object record isn't inserted or updated.

Example

This example performs an upsert of a list of accounts.

```
List<Account> acctList = new List<Account>();
// Fill the accounts list with some accounts

try {
    upsert acctList;
} catch (DmlException e) {

}
```

This next example performs an upsert of a list of accounts using a foreign key for matching existing records, if any.

```
List<Account> acctList = new List<Account>();
// Fill the accounts list with some accounts

try {
    // Upsert using an external ID field
    upsert acctList myExtIDField__c;
} catch (DmlException e) {

}
```

Delete Statement

The `delete` DML operation deletes one or more existing sObject records, such as individual accounts or contacts, from your organization's data. `delete` is analogous to the `delete()` statement in the SOAP API.

Syntax

```
delete sObject

delete sObject[]
```

Example

The following example deletes all accounts that are named 'DotCom':

```
Account[] doomedAccts = [SELECT Id, Name FROM Account
                         WHERE Name = 'DotCom'];
try {
    delete doomedAccts;
} catch (DmlException e) {
    // Process exception here
}
```

Undelete Statement

The `undelete` DML operation restores one or more existing sObject records, such as individual accounts or contacts, from your organization's Recycle Bin. `undelete` is analogous to the UNDELETE statement in SQL.

Syntax

```
undelete sObject | ID

undelete sObject[] | ID[]
```

Example

The following example undeletes an account named 'Universal Containers'. The `ALL ROWS` keyword queries all rows for both top level and aggregate relationships, including deleted records and archived activities.

```
Account[] savedAccts = [SELECT Id, Name FROM Account WHERE Name = 'Universal Containers' A
try {
    undelete savedAccts;
} catch (DmlException e) {
    // Process exception here
}
```

> **Note**
>
> For more information on processing `DmlException`s, see Bulk DML Exception Handling.

Merge Statement

The `merge` statement merges up to three records of the same sObject type into one of the records, deleting the others, and re-parenting any related records.

> **Note**
>
> This DML operation does not have a matching Database system method.

Syntax

```
merge sObject sObject

merge sObject sObject[]

merge sObject ID

merge sObject ID[]
```

The first parameter represents the master record into which the other records are to be merged. The second parameter represents the one or two other records that should be merged and then deleted. You can pass these other records into the `merge` statement as a single sObject record or ID, or as a list of two sObject records or IDs.

Example

The following example merges two accounts named 'Acme Inc.' and 'Acme' into a single record:

```
try {
    merge masterAcct mergeAcct;
} catch (DmlException e) {
    // Process exception here
}
```

Note

For more information on processing `DmlException`s, see Bulk DML Exception Handling.

DID THIS ARTICLE SOLVE YOUR ISSUE?
Let us know so we can improve!

Share your feedback