Trigger Class

Use the `Trigger` class to access run-time context information in a trigger, such as the type of trigger or the list of sObject records that the trigger operates on.

Namespace

System

Trigger Context Variables

The `Trigger` class provides the following context variables.

Variable	Usage
`isExecuting`	Returns true if the current context for the Apex code is a trigger, not a Visualforce page, a Web service, or an `executeanonymous()` API call.
`isInsert`	Returns `true` if this trigger was fired due to an insert operation, from the Salesforce user interface, Apex, or the API.
`isUpdate`	Returns `true` if this trigger was fired due to an update operation, from the Salesforce user interface, Apex, or the API.
`isDelete`	Returns `true` if this trigger was fired due to a delete operation, from the Salesforce user interface, Apex, or the API.
`isBefore`	Returns `true` if this trigger was fired before any record was saved.
`isAfter`	Returns `true` if this trigger was fired after all records were saved.
`isUndelete`	Returns `true` if this trigger was fired after a record is recovered from the Recycle Bin. This recovery can occur after an undelete operation from the Salesforce user interface, Apex, or the API.
`new`	Returns a list of the new versions of the sObject records. This sObject list is only available in `insert`, `update`, and `undelete` triggers, and the records can only be modified in `before` triggers.
`newMap`	A map of IDs to the new versions of the sObject records. This map is only available in `before update`, `after insert`, `after update`, and `after undelete` triggers.
`old`	Returns a list of the old versions of the sObject records. This sObject list is only available in `update` and `delete` triggers.

	This map is only available in `update` and `delete` triggers.
`operationType`	Returns an enum of type System.TriggerOperation corresponding to the current operation. Possible values of the System.TriggerOperation enum are: `BEFORE_INSERT`, `BEFORE_UPDATE`, `BEFORE_DELETE`, `AFTER_INSERT`, `AFTER_UPDATE`, `AFTER_DELETE`, and `AFTER_UNDELETE`. If you vary your programming logic based on different trigger types, consider using the `switch` statement with different permutations of unique trigger execution enum states.
`size`	The number of records processed in a trigger invocation. DML operations that include over 200 records are processed in batches, and the trigger is invoked for each batch. `Trigger.size` includes only the number of records in the current batch, not the total number of records in the DML operation.

Note

The record firing a trigger can include an invalid field value, such as a formula that divides by zero. In this case, the field value is set to `null` in these variables:

- `new`
- `newMap`
- `old`
- `oldMap`

Example

For example, in this simple trigger, `Trigger.new` is a list of sObjects and can be iterated over in a `for` loop. It can also be used as a bind variable in the `IN` clause of a SOQL query.

```
Trigger simpleTrigger on Account (after insert) {
    for (Account a : Trigger.new) {
        // Iterate over each sObject
    }

    // This single query finds every contact that is associated with any of the
    // triggering accounts. Note that although Trigger.new is a collection of
    // records, when used as a bind variable in a SOQL query, Apex automatically
    // transforms the list of records into a list of corresponding Ids.
    Contact[] cons = [SELECT LastName FROM Contact
                      WHERE AccountId IN :Trigger.new];
}
```

This trigger uses Boolean context variables like `Trigger.isBefore` and `Trigger.isDelete` to define code that only executes for specific trigger conditions:

```
trigger myAccountTrigger on Account(before delete, before insert, before update,
                                    after delete, after insert, after update) {
if (Trigger.isBefore) {
    if (Trigger.isDelete) {

        // In a before delete trigger, the trigger accesses the records that will be
        // deleted with the Trigger.old list.
        for (Account a : Trigger.old) {
            if (a.name != 'okToDelete') {
```

```
        // with the Trigger.new list.
            for (Account a : Trigger.new) {
                if (a.name == 'bad') {
                    a.name.addError('Bad name');
                }
            }
            if (Trigger.isInsert) {
                for (Account a : Trigger.new) {
                    System.assertEquals('xxx', a.accountNumber);
                    System.assertEquals('industry', a.industry);
                    System.assertEquals(100, a.numberofemployees);
                    System.assertEquals(100.0, a.annualrevenue);
                    a.accountNumber = 'yyy';
                }

    // If the trigger is not a before trigger, it must be an after trigger.
    } else {
        if (Trigger.isInsert) {
            List<Contact> contacts = new List<Contact>();
            for (Account a : Trigger.new) {
                if(a.Name == 'makeContact') {
                    contacts.add(new Contact (LastName = a.Name,
                                              AccountId = a.Id));
                }
            }
          insert contacts;
        }
      }
    }}}
```