Apex Triggers

Monday, August 08, 2016 10:27 AM

Trigger Context Variables

The following table is a comprehensive list of all context variables available for triggers.

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 (that is, 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.
oldMap	A map of IDs to the old versions of the sObject records. This map is only available in update and delete triggers.
size	The total number of records in a trigger invocation, both old and new.

Trigger Context Variables

With triggers, you're able to add and edit related objects.

Beyond the Basics

The trigger you've added iterates over all records that are part of the trigger context—the for loop iterates over Trigger.New. However, the loop in this trigger could be more efficient. We don't really need to access every account in this trigger context, but only a subset—the accounts without opportunities. The next unit shows how to make this trigger more efficient. In the Bulk Trigger Design Patterns unit, learn how to modify the SOQL query to get only the accounts with no

opportunities. Then, learn to iterate only over those records.

Using Trigger Exceptions

To prevent saving records in a trigger, call the addError() method on the sObject you're currently working. The addError() method throws a fatal error inside a trigger. Error shows in the UI and is logged.

Beyond the Basics

Calling addError() in a trigger causes the entire set of operations to roll back, except when bulk DML is called with partial success.
If a DML statement in Apex spawned the trigger, any error rolls back the entire operation. However, the runtime engine still processes every record in the operation to compile a comprehensive list of errors.
If a bulk DML call in the Force.com API spawned the trigger, the runtime engine sets the bad records aside. The runtime engine then attempts a partial save of the records that did not generate errors.

Integrating Apex code with external Web services (Callouts)

When making a callout from a trigger, the callout must be done asynchronously so that the trigger process doesn't block you from working while waiting for the external service's response.

Bulk Apex Triggers

When you use bulk design patterns, your triggers have better performance, consume less server resources, and are likely to exceed platform limits. You can process a large number of records efficiently and run within governor limits on the Force.com platform.

Operating on Record Sets

If the origin of the action was bulk DML or the API, triggers operate on a record set rather than one record. Always assume that the trigger operates on a collection of records so that it works in all circumstances.

Bulk SOQL

Querying the SOQL code outside the context loop helps avoid hitting query limits - 100 SOQL queries for synchronous Apex or 200 for asynchronous Apex. Use inner queries to pull related records and use the IN clause on the Trigger.New context variable.
Avoid calling queries within the context loop
For each statements that contain queries only run the query once and store the query results into the cache.

Beyond the Basics

Triggers execute on batches of 200 records at a time. So if 400 records cause a trigger to fire, the trigger fires twice, once for each 200 records. For this reason, you don’t get the benefit of SOQL for loop record batching in triggers, because triggers batch up records as well. The SOQL for loop is called twice in this example, but a standalone SOQL query would also be called twice. However, the SOQL for loop still looks more elegant than iterating over a collection variable!

Bulk DML

Apex runtime allows up to 150 DML calls in one transaction. Instead of updating each sObject, update a list of sObjects.