**Time triggers :**

Salesforce evaluates time-based workflow on the organization’s time zone, not the user’s. Users in different time zones might see differences in behavior.

Salesforce doesn’t necessarily execute time triggers in the order they appear on the workflow rule detail page. Workflow rules list time triggers that use the Before field first, followed by time triggers that use the After field **.**

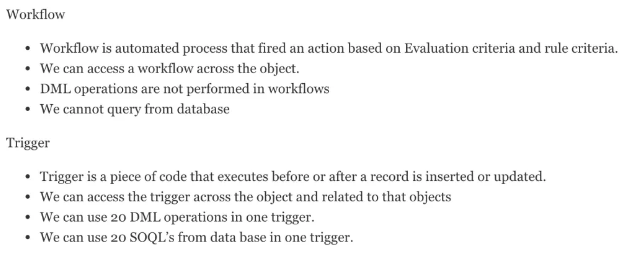
If you change a date field that is referenced by an unfired time trigger in a workflow rule that has been evaluated, Salesforce recalculates the unfired time triggers associated with the rule. For example, if a workflow rule is scheduled to alert the opportunity owner 7 days before the opportunity close date and the close date is set to 2/20/2011, Salesforce sends the alert on 2/13/2011. If the close date is updated to 2/10/2011 and the time trigger hasn't fired, Salesforce reschedules the alert for 2/3/2011. If Salesforce recalculates the time triggers to a date in the past, Salesforce triggers the associated actions shortly after you save the record.

Time-dependent actions remain in the workflow queue only as long as the workflow rule criteria are still valid. If a record no longer matches the rule criteria, Salesforce removes the time-dependent actions queued for that record.

How to clear the Time based workflow action queue?

Yes, all pending actions to be triggered on a future date appear in the Workflow Queue. Here's how System administrators can view and manage the queue: Go to Setup | Customize | Monitor | Time-Based Workflow and delete the action which you don't want to fire  
  
**To view pending actions:**  
  
1. From **Setup**, click **Monitoring** | **Time-Based Workflow**.  
2. Click Search to view all pending actions for any active workflow rules, or set the filter criteria and click Search to view only the pending actions that match the criteria. The filter options are:  
  
**Workflow Rule Name:** The name of the workflow rule.  
**Object**: The object that triggered the workflow rule. Enter the object name in the singular form.  
**Scheduled Date**: The date the pending actions are scheduled to occur.  
**Create Date**: The date the record that triggered the workflow was created.  
**Created By**: The user who created the record that triggered the workflow rule.  
**Record Name**: The name of the record that triggered the workflow rule.  
  
**To cancel pending actions:**  
Select the box next to the pending actions you want to cancel.  
Click Delete.

Difference between Trigger and Workflow?



**What is the difference between Created by and Owner ?**

CreatedBy : It is the user who has created the record. This can not be changed. It is just a System Information to track who created the record.

OwnerId : ID of the User who has been assigned to work this record.

If you update this field, the previous owner's access becomes Read Only or the access specified in your organization-wide default for opportunities, whichever is greater.

OwnerId defines ownership and is used to give access to the user on the record in case that object access is set to private in your organization.

On process builder how to invoke an apex class?

For calling a class from process Builder just follow the following steps:  
1.Click on the immediate action button once you have written the condition  
2.Then in action type Select Apex  
3.Then provide a Name to your action that you want to perform  
4.Now From the Apex Class Picklist Select the name of the class that you want to call  
5.In the last step click save.

Difference between action function and action support, in detail?

ActionFunction : provides support for invoking controller action methods directly from JavaScript code using an AJAXrequest

Used when we need to perform similar action on various events. Even though you can use it in place of actionSupport as well where only event is related to only one control.

<apex:actionFunction name="sayHello" action="{!sayHello}" rerender="out"

status="myStatus"/>

ActionSupport : A component that adds AJAX support to another component, allowing the component to be refreshed asynchronously by the server when a particular event occurs, such as a button click or mouseover.?

Used when we want to perform an action on a particular event of any control like onchange of any text box or picklist.

<apex:actionSupport event="onclick"

action="{!incrementCounter}"

rerender="counter" status="counterStatus"/>

Action poller, action region, action status?

**1. <apex:actionFunction> -**Provides support for invoking controller action methods directly from JavaScript code using an AJAX request. An <apex:actionFunction> component must be a child of an <apex:form> component.

**2. <apex:actionPoller> -** A timer that sends an AJAX update request to the server according to a time interval that you specify. Update request can then result in a full or partial page update. You should avoid using this component with enhanced lists.

**3. <apex:actionStatus> -** A component that displays the status of an AJAX update request. An AJAX request can either be in progress or complete

**apex:selectList**

A list of options that allows users to select only one value or multiple values at a time, depending on the value of its multiselect attribute.

|  |
| --- |
| <apex:selectList value="{!countries}" multiselect="true"> |

|  |  |
| --- | --- |
| 05 | <apex:selectOptions value="{!items}"/> |

|  |  |
| --- | --- |
| 06 | </apex:selectList> |

**Apex primitive data types include :**

String  
– Blob (for storing binary data)  
– Boolean  
– Date, DateTime and Time  
– Integer, Long, Decimal, Double  
– ID (Force.com database record identifier)

**Apex sObject Types**  
– Sobject (object representing a Force.com standard or custom object)  
**– Example:**  
• Account acct = new Account(); //Sobject example

**Apex has the following types of collections**  
– Lists  
– Maps  
– Sets

**Which SOQL statement can be used to get all records even from recycle bin or Achieved Activities?**

Ans : We will need “ALL Rows” clause of SOQL.

Sample :

SELECT COUNT() FROM Contact WHERE AccountId = a.Id ALL ROWS

**How can you lock record using SOQL so that it cannot be modified by other user.?**

Ans : we will need “FOR UPDATE” clause of SOQL.

Sample :

Account [] accts = [SELECT Id FROM Account LIMIT 2 FOR UPDATE];

**Declare Visualforce page as HTML5.**

<apex:page docType="html-5.0" />

**Visualforce page output as JSON.**

<apex:page controller="ControllerName" contentType="application/x-JavaScript; charset=utf-8" showHeader="false" standardStylesheets="false" sidebar="false">

{!jsonString}

</apex:page>

**What’s the maximum batch size in a single trigger execution?**

Default batch size is 200 ,However maximum batch size is 2000.

**Give an example of a standard object that’s also junction object.?**

Quote is junction between Contract and Opportunity.

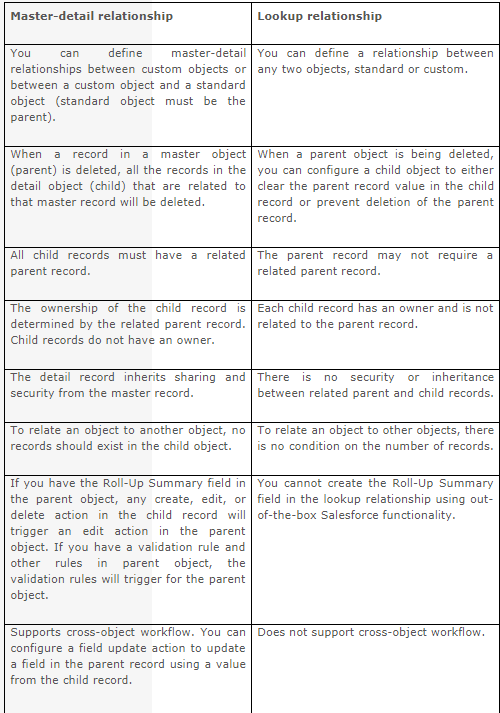
**what is the difference between "Trigger.New" and "Trigger.old"?**

Trigger.new : Returns a list of the new versions of the sObject records. Note that this sObject list is only available in insert and update triggers, and the records can only be modified in before triggers.

Trigger.old : Returns a list of the old versions of the sObject records. Note that this sObject list is only available in update and delete triggers.

trigger.new is holding your object record data which is currently set by user and trigger.old contains history object records. like if we have Opportunity trigger. and in this Opportunity.trigger we have even after update and if you need to compute if there is any change in Opportunity stage then you can figure out this changes using trigger.new and trigger.old values.

**Difference between master-detail relationships and lookup relationships**



**Batch Apex :**

Apex class that implements the Database.Batchable Interface is known as batch class .

We are basically using batch apex to avoid some governor limits .

There are two interface in Database Namespace .

1.Batchable

Below are methods in Batchable interface

Public System.Iterable start(Database.BatchabelContext bcon)

Public Database.QueryLocator start(Database.BatchabelContext bcon)

Public void execute(Database.BatchableContext bc,List<Sobject> scope)

Public void finish(Database.BatchableContext bc)

2.BatchableContext

Below are the methods in the interface

Public Id getJobId()

public Id getChildJobId()

Basic thing is that once you will implement Batchable interface in your class you have to implement three methods in that interface .

**Batch Apex Governor Limits:-**

All methods in the class must be defined as global or public.

Up to 5 batch jobs can be queued or active concurrently.

In a running test, you can submit a maximum of 5 batch jobs.

The maximum number of batch Apex method executions per a 24-hour period is 250,000 or the number of user licenses in your organization multiplied by 200, whichever is greater.

A maximum of 50 million records can be returned in the Database.QueryLocator object. If more than 50 million records are returned, the batch job is immediately terminated and marked as Failed.

Start method can have up to 15 query cursors open at a time per user.

Execute and finish methods each have a limit of five open query cursors per user.

One example is if you need to make a field update to every Account in your organization. If you have 10,001 Account records in your org, this is impossible without some way of breaking it up. So in the start() method, you define the query you're going to use in this batch context: 'select Id from Account'. Then the execute() method runs, but only receives a relatively short list of records (default 200). Within the execute(), everything runs in its own transactional context, which means almost all of the governor limits only apply to that block. Thus each time execute() is run, you are allowed 150 queries and 50,000 DML rows and so on. When that execute() is complete, a new one is instantiated with the next group of 200 Accounts, with a brand new set of governor limits. Finally the finish() method wraps up any loose ends as necessary, like sending a status email.

So your batch that runs against 10,000 Accounts will actually be run in 50 separate execute() transactions, each of which only has to deal with 200 Accounts. Governor limits still apply, but only to each transaction, along with a separate set of limits for the batch as a whole.

Disadvantages of batch processing:

--> It runs asynchronously, which can make it hard to troubleshoot without some coded debugging, logging, and persistent stateful reporting. It also means that it's queued to run, which may cause delays in starting.

--> There's a limit of 5 batches in play at any time, which makes it tricky to start batches from triggers unless you are checking limits.