FREE eBook

LEARNING Salesforce

Free unaffiliated eBook created from
Stack Overflow contributors.

#salesforce

Table of Contents

About ... 1

Chapter 1: Getting started with Salesforce ... 2

- Remarks ... 2
- Examples ... 2
  - Installation or Setup ... 2
  - Salesforce Products ... 2
- Sales Cloud ... 2
- Service Cloud ... 2
- Marketing Cloud ... 2
- Community Cloud ... 2
- Analytics Cloud aka Wave Analytics ... 2
- App Cloud ... 3
- IoT Cloud ... 3
- Industry Specific Products ... 3
- Financial Services Cloud ... 3
- Health Cloud ... 3
- Heroku ... 3

Chapter 2: Apex Testing ... 4

- Examples ... 4
  - Assert Methods ... 4
  - Basic Test Class ... 4
  - Using testSetup ... 5
  - Using static blocks ... 5
  - Assertion Methods ... 6

Chapter 3: Apex Triggers ... 7

- Syntax ... 7
- Parameters ... 7
- Examples ... 7
  - Basic trigger ... 7
  - Trigger context variables ... 7

Manipulating records that fired the trigger ... 8
Chapter 4: Approval Process Objects ... 10
Remarks ... 10
Examples ... 11
ProcessDefinition ... 11
ProcessNode ... 12
ProcessInstance ... 12
ProcessInstanceStep & ProcessInstanceWorkitem ... 12
ProcessInstanceHistory* ... 13
Chapter 5: Custom Settings ... 14
Remarks ... 14
Introduction ... 14
List Custom Settings ... 14
Examples ... 15
Creating & Managing Custom Settings ... 15
Creation ... 15
Management ... 15
Using Hierarchy Custom Settings To Disable Workflow / Validation Rules ... 16
Custom Setting ... 16
Custom Setting Field ... 17
Custom Setting Field Value ... 17
Validation Rule ... 18
Workflow Rules ... 19
Using Hierarchy Custom Settings To Disable Apex Code ... 19
Explanation ... 19
Apex Class ... 19
Unit Test ... 19
Updating Hierarchy Custom Settings in Apex Code ... 21
Chapter 6: Date Time Manipulation ... 26
Examples ... 26
Easily Find Last Day of a Month ... 26

Chapter 7: Global Variables in classes 27
Introduction 27
Examples 27
UserInfo 27
Chapter 8: Global Variables on Visualforce pages 28
Examples 28
$Resource 28
$Label 28
$User 28
Chapter 9: Page Navigation with help of list wrapper class in sales force. 29
Introduction 29
Examples 29
Pagination Controller 29
Chapter 10: SalesForce CI Integration 33
Introduction 33
Examples 33
How to configure Jenkins to deploy code on Development or Production org ? 33
Jenkins CI tools which can be used for SalesForce Automation 33
Chapter 11: Salesforce Object Query Language (SOQL) 34
Syntax 34
Examples 34
Basic SOQL Query 34
SOQL Query With Filtering 34
SOQL Query With Ordering 35
Using SOQL to Construct a Map 35
SOQL Query to Reference Parent Object's Fields 35
SOQL Queries in Apex 36
Variable References in Apex SOQL Queries 36
Potential Exceptions in Apex SOQL Queries 36
Using a Semi-Join 37
Dynamic SOQL 37
Chapter 12: Salesforce REST API 38

Introduction ... 38
Examples ... 38
OAuth2 access_token and list of services ... 38
Chapter 13: Tools for Development ... 39
Examples ... 39
IDEs ... 39
Browser extensions ... 39
Debuggers ... 39
Salesforce ETL tools ... 40
Static Analysis Tools ... 40
Chapter 14: Trigger Bulkification ... 41
Examples ... 41
Bulkification ... 41
Chapter 15: Visualforce Page Development ... 42
Examples ... 42
Basic page ... 42
Using Standard Controllers ... 42
Chapter 16: Working with External Systems ... 43
Examples ... 43
Making an outbound callout ... 43
Credits ... 44

About

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: salesforce

It is an unofficial and free Salesforce ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official Salesforce.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

Chapter 1: Getting started with Salesforce

Remarks

This section provides an overview of what salesforce is, and why a developer might want to use it.

It should also mention any large subjects within salesforce, and link out to the related topics. Since the Documentation for salesforce is new, you may need to create initial versions of those related topics.

Examples

Installation or Setup

The best way to get started with Salesforce is to get your own Developer Edition.

Developer Edition (often referred to as a "DE org") is a fully-featured development environment with limits on data and users. Developer Edition is where you can get familiar with the environment, try out stuff and mess around in Salesforce.com. Developer edition is an environment lets you instantly start developing, testing and deploying your app in the cloud.

Salesforce Products

The Salesforce application consists of several products which can be integrated with each other:

Sales Cloud

Marketing Page, Salesforce Documentation

Service Cloud

Marketing Page, Salesforce Documentation, Trailhead

Marketing Cloud

Marketing Page

Community Cloud

Marketing Page, Salesforce Documentation

Analytics Cloud aka Wave Analytics

Marketing Page, Salesforce Documentation, Trailhead

App Cloud

Marketing Page

IoT Cloud

Marketing Page

Industry Specific Products

Financial Services Cloud

Marketing Page, Salesforce Documentation

Health Cloud

Marketing Page, Salesforce Documentation

Heroku

Marketing Page

Read Getting started with Salesforce online: https://riptutorial.com/salesforce/topic/3768/getting-started-with-salesforce

Chapter 2: Apex Testing

Examples

Assert Methods

```
static TestMethod void DmlTest() {
    List<Contact> allContacts = [SELECT Id FROM Contact];
    System.assert(allContacts.isEmpty());

    Contact testContact = new Contact(FirstName = 'John', LastName = 'Doe');
    insert testContact;
    allContacts = [SELECT Id FROM Contact];
    System.assertNotEquals(0, allContacts.size(), 'Optional message in case of failure');

    delete allContacts;
    allContacts = [SELECT Id FROM Contact];
    System.assertEquals(0, allContacts.size());
}
```

Basic Test Class

This test class will test the `IsBlank(...)` method of `SomeClass`. Below is the example `SomeClass`. This class has only the one, basic `static` method, but you will be unable to deploy it to a production instance for use until you have reached the code coverage threshold.

```
public class SomeClass {

    public static Boolean IsBlank(String someData) {
        if (someData == null) {
            return true;
        } else if (someData == '') {
            return true;
        } else {
            return false;
        }
    }

}
```

As one can see, this method is simply a `if` statement with three branches. To write an effective test class, we must cover each branch with code, and use `System.assertEquals(...)` statements to verify that the proper data was received from `IsBlank(...)`.

```
@isTest
public class SomeClass_test {

    @isTest
    public static void SomeClass_IsBlank_test() {

        String testData;
```

```
        // SomeClass.IsBlank() returns true for Null values
        System.assertEquals(true, SomeClass.IsBlank(testData));

        testData = '';

        // SomeClass.IsBlank() returns true for empty strings
        System.assertEquals(true, SomeClass.IsBlank(testData));

        testData = 'someData';

        // SomeClass.IsBlank() returns false when testData is neither
        // an empty string nor Null
        System.assertEquals(false, SomeClass.IsBlank(testData));

    }

}
```

Using testSetup

You can use a method annotated with `@testSetup` to write code that will have been executed before each test run:

```
public class AccountService {
  public static Account fetchAccount() {
    return [ SELECT Id, Name FROM Account LIMIT 1 ];
  }
}
```

```
@isTest
public class AccountServiceTest {
  private static final String TEST_ACCOUNT_NAME = 'My Test Account';

  @testSetup
  public static void setUpAccountData() {
    Account a = new Account(Name = TEST_ACCOUNT_NAME);
  }

  @isTest
  public static void testFetchAccount() {
    Account a = AccountService.fetchAccount();
    System.assertNotEquals(null, a, 'Account should not be null');
    System.assertEquals(TEST_ACCOUNT_NAME, a.Name, 'Account name should be correct');
  }
}
```

Using static blocks

While you can use the `@testSetup` annotation to designate a method to be run before tests are executed, this method will usually only be run once. If you need code to be run before each test, you can use a `static` block:

```
@isTest
public class MyTest {
  static {
```

```
    // code here will be run before each test is executed
  }
}
```

Assertion Methods

`System.assert` can be used to check that a boolean expression evaluates to true:

```
System.assert(Service.isActive());
System.assert(!Service.getItems().isEmpty(), 'items should not be empty');
```

`System.assertEquals` and `System.assertNotEquals` can be used to check equality of two values. The expected value is passed as the first parameter, and the value under test is passed as the second.

```
System.assertEquals(4, Service.getItems().size());
System.assertNotEquals(null, Service.getItems());

// failure messages are optional:
System.assertEquals(true, Service.doWork(), 'doWork should return true');
System.assertNotEquals(null, Service.doWork(), 'doWork should not be null');
```

Read Apex Testing online: https://riptutorial.com/salesforce/topic/4930/apex-testing

Chapter 3: Apex Triggers

Syntax

- trigger <*name*> on <*object-api-name*> (<*events*>) { // your trigger logic }

Parameters

parameter	description
name	Name of the trigger
object-api-name	Object on which the trigger will fire. Can be any standard or custom object.
events	Events which will fire the trigger. Are a combination of either `before`/`after` with any of `insert`/`update`/`delete`. There's also `after undelete` without `before` counterpart.

Examples

Basic trigger

```
trigger AccountTrigger on Account (before insert) {
    System.debug('Account(s) are about to be inserted');
}
```

Trigger context variables

```
trigger ContactTrigger on Contact (before insert, after insert,
                                   before update, after update,
                                   before delete, after delete,
                                   after undelete) {
    /** Before or After trigger execution**/
    //Returns true if trigger is before
    System.debug('Trigger:Time:Before : ' + Trigger.isBefore);
    //Returns true if trigger is after
    System.debug('Trigger:Time:After  : ' + Trigger.isAfter);

    /**DML Operation trigger execution **/
    //Returns true if trigger is insert
    System.debug('Trigger:DML:Insert  : ' + Trigger.isInsert);
    //Returns true if trigger is update
    System.debug('Trigger:DML:Update  : ' + Trigger.isUpdate);
    //Returns true if trigger is delete
    System.debug('Trigger:DML:Delete  : ' + Trigger.isDelete);
    //Returns true if trigger is undelete
    System.debug('Trigger:DML:Undelete: ' + Trigger.isUndelete);
```

```
    /** Records on Trigger execution **/
    //Returns data in state before DML. Records are read only
    //Not available for Insert Operation
    //Format: List<sObject>
    List<Contact> old_contacts = Trigger.old;
    System.debug('Trigger:Data:Old    : ' + old_contacts);
    //Returns data in state before DML. Records are read only
    //Not available for Insert Operation
    //Format: Map<Id, sObject>
    Map<Id, Contact> old_contacts_map = Trigger.oldMap;
    System.debug('Trigger:Data:OldMap : ' + old_contacts_map);
    //Returns data in state after DML.
    //Allowed for modifications in before context only
    //Not available for Delete Operation
    //Format: List<sObject>
    List<Contact> new_contacts = Trigger.new;
    System.debug('Trigger:Data:New    : ' + new_contacts);
    //Returns data in after before DML.
    //Allowed for modifications in before context only
    //Not available for InsertOperation
    //Format: Map<Id, sObject>
    Map<Id, Contact> new_contacts_map = Trigger.newMap;
    System.debug('Trigger:Data:NewMap : ' + new_contacts_map);

    /** Another context variables **/
    //Returns amount of record in DML for trigger execution
    System.debug('Trigger:Size        :' + Trigger.size);
    //Returns true if the current context for the Apex code
    //is a trigger, not VF, web service or anonymous apex
    System.debug('Trigger:isExecuting :' + Trigger.isExecuting);

    //Simple example how to use above context variables
    //for different scenarios in combination
    if (Trigger.isBefore && Trigger.isUpdate) {
        // actions for before update
    } else if (Trigger.isAfter) {
        if (Trigger.isUpdate) {
            // actions for after update
        } else if (Trigger.isInsert) {
            // actions for after insert
        }
    }
}
```

Manipulating records that fired the trigger

```
trigger MyTrigger on SomeObject__c (after insert, after update) {
    if (Trigger.isAfter && Trigger.isInsert) {
        System.debug('The following records were inserted: ');
        for (SomeObject__c o : Trigger.new) {
            System.debug(o.Name);
        }
    } else if (Trigger.isAfter && Trigger.isUpdate) {
        for (Id key : Trigger.newMap) {
            SomeObject__c theOldOne = Trigger.newMap.get(key);
            SomeObject__c theNewOne = Trigger.oldMap.get(key);
            if (theNewOne.Name != theOldOne.Name) {
                System.debug('The name of ' + key + ' has been changed');
```

```
                }
            }
        }
    }
```

Read Apex Triggers online: https://riptutorial.com/salesforce/topic/4864/apex-triggers

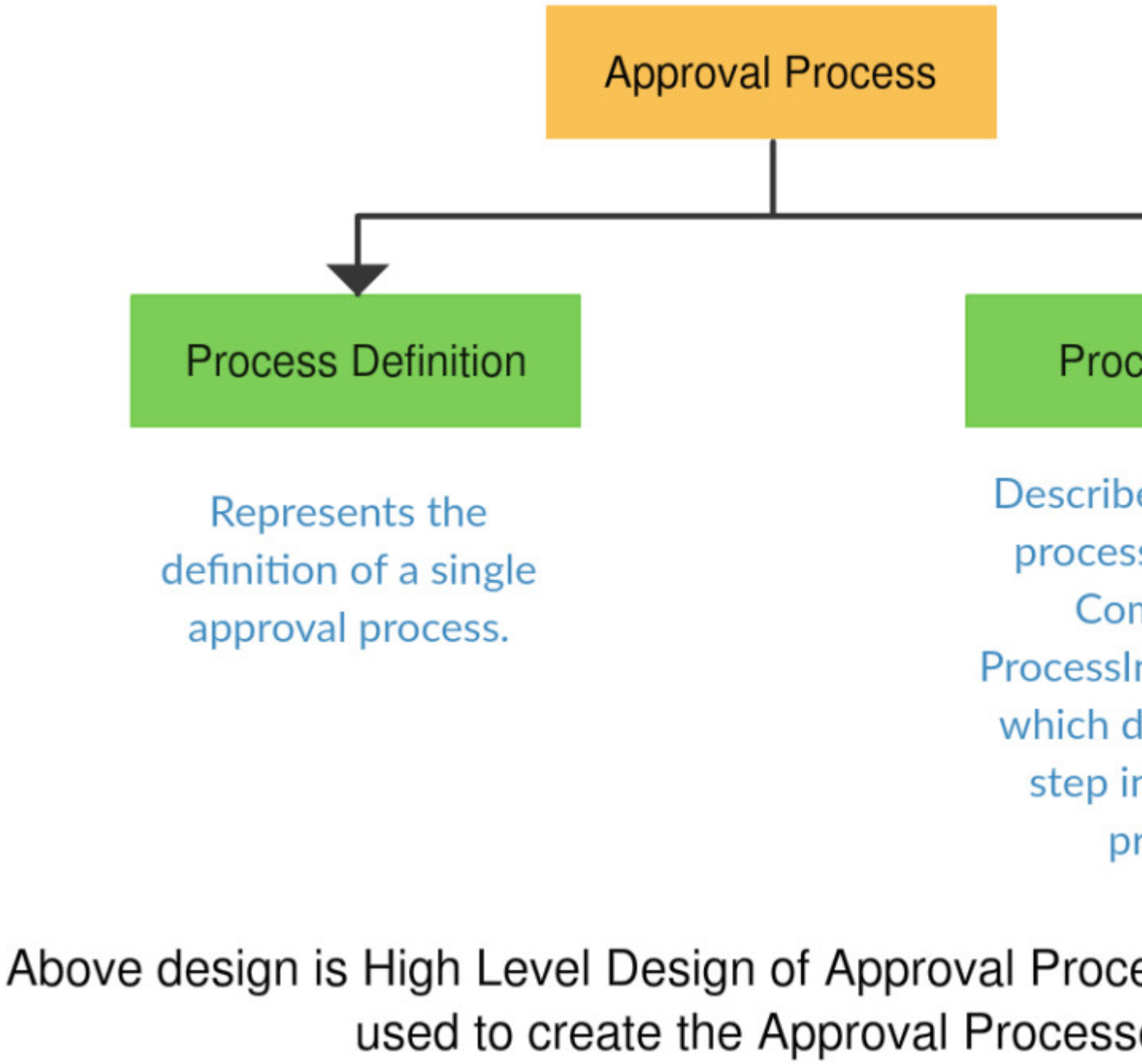
Chapter 4: Approval Process Objects

Remarks

Approval Process is a very amazing feature in Salesforce to automate the business process. An approval process is a set of the steps necessary for a particular record to be approved or rejected by approver or set of approvers.

A step can apply to all records included in the process, or just records that meet certain administrator-defined criteria. An approval process also specifies the actions to take when a record is approved, rejected, recalled, or first submitted for approval.

> ProcessDefinition and ProcessNode objects act as a template and store the master configurations for Approval Process itself.



Above design is High Level Design of Approval Proce
used to create the Approval Process

Examples

ProcessDefinition

Represents the definition of a single approval process. Use this object to read the description of an approval process. The definition is read-only. We can not modify the record created in ProcessDefinition Object. But we can describe, query, search and retrieve the approval processes information.

~ Query ~

```
SELECT CreatedById,CreatedDate,Description,DeveloperName,LastModifiedById,
LastModifiedDate,LockType,Name,State,SystemModstamp,TableEnumOrId,Type,Id
```

```
FROM ProcessDefinition
```

The records are created when we create a new approval process using Salesforce user interface of Approval Process.

ProcessNode

Represents the Process Steps created for particular approval process(ProcessDefinition). This object is used to read the description of process step. In simple words ProcessNode records describes a step in a process definition. We can describe, query, search and retrieve the approval processes Steps.

~ Query ~

```
SELECT Description,DeveloperName,Name,ProcessDefinitionId,SystemModstamp
,Id,FROM ProcessNode
```

As we can see ProcessDefinitionId field is acting like a foreign key which is referring ProcessDefinition Object or Table for which steps or process nodes are created. This object is also read only as ProcessDefinition Object.

ProcessInstance

Represents an instance of a single, complete approval process. ProcessInstance record is created every time for particular object record which is submitted for approval. Its is also read-only object. We can describe, query and retrieve the approval processes Instance.

~ Query ~

```
SELECT CompletedDate,CreatedById,CreatedDate,ElapsedTimeInDays,
ElapsedTimeInHours,ElapsedTimeInMinutes,Id,IsDeleted,LastActorId,
LastModifiedById,LastModifiedDate,ProcessDefinitionId,Status,
SubmittedById,SystemModstamp,TargetObjectId FROM ProcessInstance
```

All ProcessInstance fields are automatically populated once the record is submitted for approval, with two exceptions fields: CompletedDate and LastActorId that are populated only after the approval process instance is complete. The ProcessDefinitionId field is the reference or foreign key ID of the ProcessDefinition Object.

ProcessInstanceStep & ProcessInstanceWorkitem

Both objects ProcessInstanceStep & ProcessInstanceWorkItem are instances of process steps that are created for particular ProcessInstance. ProcessInstanceStep represents a step instance in an approval process (ProcessInstance) on which users has already acted and ProcessInstanceWorkItem represents a step instance in an approval process(ProcessInstance) on which is pending and users has to perform some action next on it. We can describe, query and retrieve the approval processes steps and workItems.

~ Query ~

```
SELECT CreatedById,CreatedDate,ElapsedTimeInDays,ElapsedTimeInHours,
ElapsedTimeInMinutes,Id,IsDeleted,OriginalActorId,ProcessInstanceId,
ActorId,SystemModstamp FROM ProcessInstanceWorkitem

SELECT ActorId,Comments,CreatedById,CreatedDate,ElapsedTimeInDays,Id,
ElapsedTimeInHours,ElapsedTimeInMinutes,OriginalActorId,ProcessInstanceId
,StepNodeId,StepStatus,SystemModstamp FROM ProcessInstanceStep
```

ProcessInstanceHistory*

The ***ProcessInstanceHistory*** is the object which is neither searchable nor queryable & this is the read-only object which shows all steps and pending approval requests associated with an approval process (ProcessInstance). *But we can use this object to replicate the related list functionality of the Salesforce user interface for approval processes which will be shown in my next blog post soon.* We can use ProcessInstanceHistory for a single read-only view of the both ProcessInstanceStep and ProcessInstanceWorkitem objects. We can query ProcessInstanceHistory by querying it in a nested soql query on the parent ProcessInstance object. The nested soql query references ***StepsAndWorkitems***, which is the child relationship name for ProcessInstanceHistory in the ProcessInstance object. This is very useful object to solve various business problems.

~ Query ~

```
SELECT CompletedDate, CreatedById, CreatedDate,Id,IsDeleted,LastActorId,
LastModifiedById,LastModifiedDate,ProcessDefinitionId,Status,SubmittedById
,SystemModstamp,TargetObjectId, (SELECT ID, ProcessNodeId, StepStatus,
Comments,TargetObjectId,ActorId,CreatedById,IsDeleted,IsPending,
OriginalActorId,ProcessInstanceId,RemindersSent,CreatedDate
FROM StepsAndWorkitems ) FROM ProcessInstance
```

Read Approval Process Objects online: https://riptutorial.com/salesforce/topic/6387/approval-process-objects

Chapter 5: Custom Settings

Remarks

Introduction

Unlike custom objects which have records based on them, custom settings let you utilize custom data sets across your org, or distinguish particular users or profiles based on custom criteria. This means, for example, that admins can edit hierarchy custom settings to deactivate Workflow / Validation Rules for single users or profiles, without having to switch them off for the whole org (see the Using Hierarchy Custom Settings To Disable Workflow / Validation Rules example above).

Validation rules commonly need to be disabled temporarily when:

- Code is updating old records, which were last edited before a validation rule was activated & therefore don't meet the newer rule's criteria.
- Code is inserting new records without the values required by a validation rule's criteria.

Workflow rules commonly need to be disabled temporarily when:

- They would trigger an Email Alert or Field Update which would overwrite or interfere the changes you are making to the record.

Use of a custom setting grants admins some declarative control over code so one of the many use cases is that when utilized, they can make it unnecessary to deploy code in order to disable triggers (see the Using Hierarchy Custom Settings To Disable Apex Code example above).

A key benefit for developers is that custom setting's data is exposed in the application cache, which enables efficient access without the cost of repeated queries to the database. This data can then be used by formula fields, validation rules, flows, Apex, and the SOAP API - see the Salesforce documentation.

The limits & considerations for custom settings are documented here.

List Custom Settings

It is possible to create List Custom Settings too, common use cases include storing two-letter state abbreviations, international dialing prefixes, and catalog numbers for products. However Salesforce is now promoting the use Custom Metadata Types, instead of List Custom Settings.

When you go to create a new Custom Setting, the following message will be displayed

> **Tip: Use Custom Metadata Types for App Configuration**
> If you're thinking of using list custom settings, consider using custom metadata types instead. Unlike list custom settings, you can migrate the records of custom metadata

types using using packages or Metadata API tools.

Custom Metadata Types have additional benefits vs List Custom Settings as described in this answer. And according to the lead developer of CMDs "There's a lot more planned for custom metadata types than custom settings on steroids."

Examples

Creating & Managing Custom Settings

Creation

To create a Custom Setting, go to:

Classic
Setup > Develop > Custom Settings > New

Lightning
Setup > Custom Code > Custom Settings > New

Create your setting (see the Remarks later in this document for the differences between Hierarchy & List custom settings). You can ignore the Visibility picklist, unless you plan to deploy your setting in a managed package.

To create your setting fields click the New button and follow the usual process for creating a custom field.

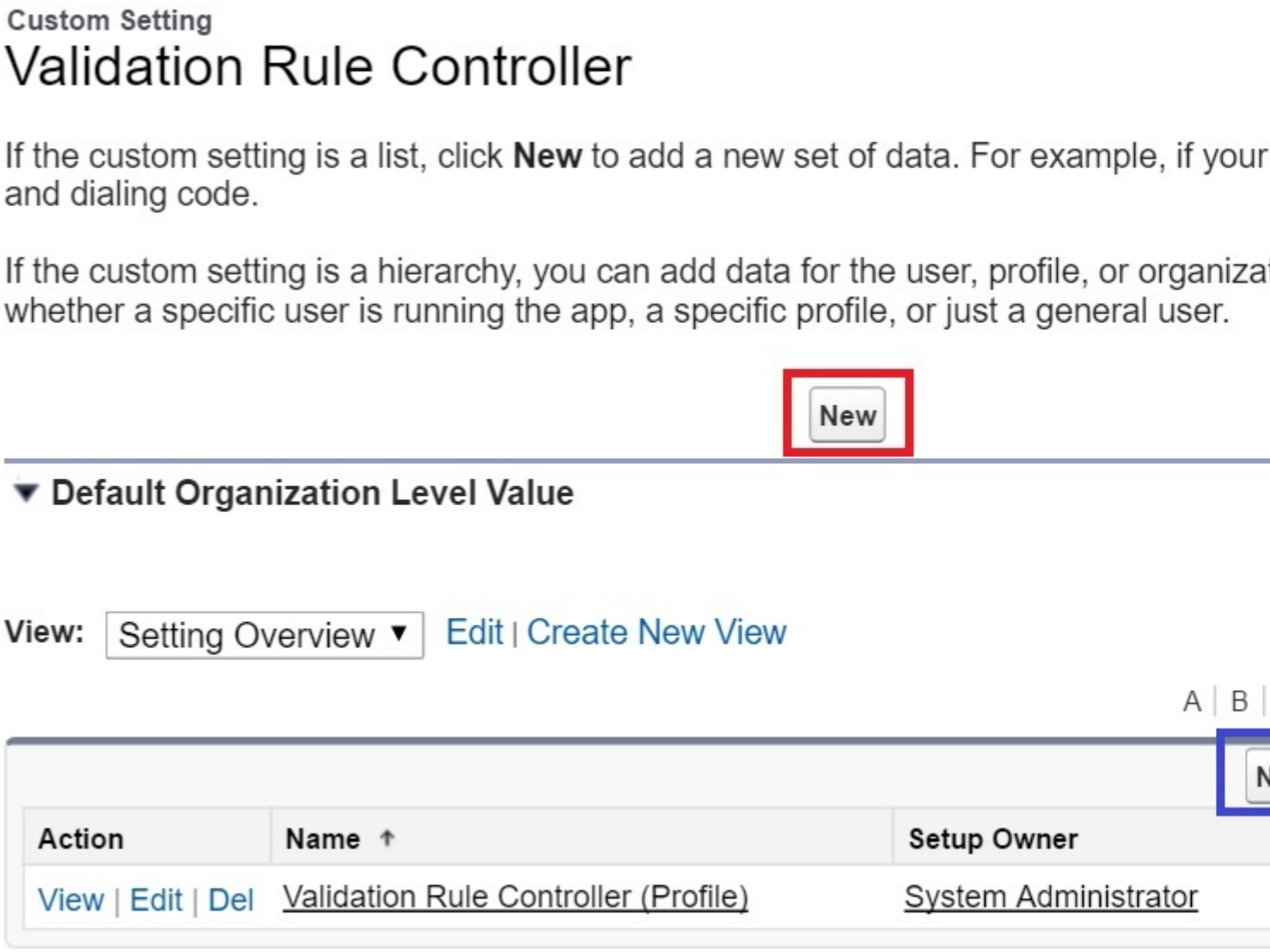
Management

Once you have created your field(s) you can start configuring the setting by clicking the Manage button.

It's easier to manage the setting if you create a new view and include any fields that you've created to give yourself a comprehensive overview of the setting, at a glance. The Setup Owner is the user or profile that the setting applies to.

To manage the setting at the org level, click the New button above the Default Organization Level Value header (in red box below).

To manage the setting at the user or profile level, click the New button in the blue box below.



Using Hierarchy Custom Settings To Disable Workflow / Validation Rules

Custom Setting

Custom Setting Definition

Validation Rule Controller

Create the fields for your custom setting. The data in these fields are cached with the appli

Custom Setting Definition Detail Edit Delete Manage

Label	Validation Rule Controller
API Name	Val_Rule_Cntrlr__c
Visibility	Public
Namespace Prefix	
Last Modified Date	28/07/2016 14:00

Custom Setting Field

Validation Rule Controller Custom Field

All Opportunity Disabled

Back to Validation Rule Controller

Custom Field Definition Detail Edit

Field Information

Field Label	All Opportunity Disabled
Field Name	All_Opportunity_Disabled
API Name	All_Opportunity_Disabled__c
Description	When checked, all validation rules on the Opportunity object wi
Help Text	
Created By	Alex Sherwood, 28/07/2016 14:01

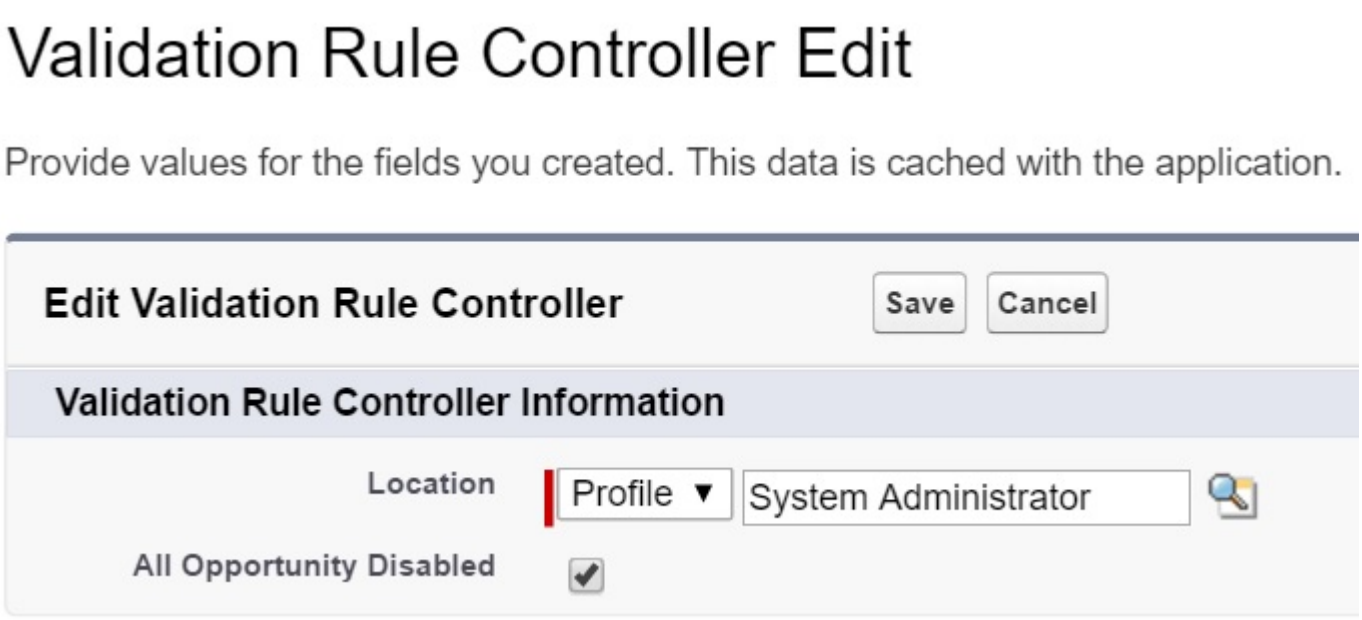
General Options

Default Value	Unchecked

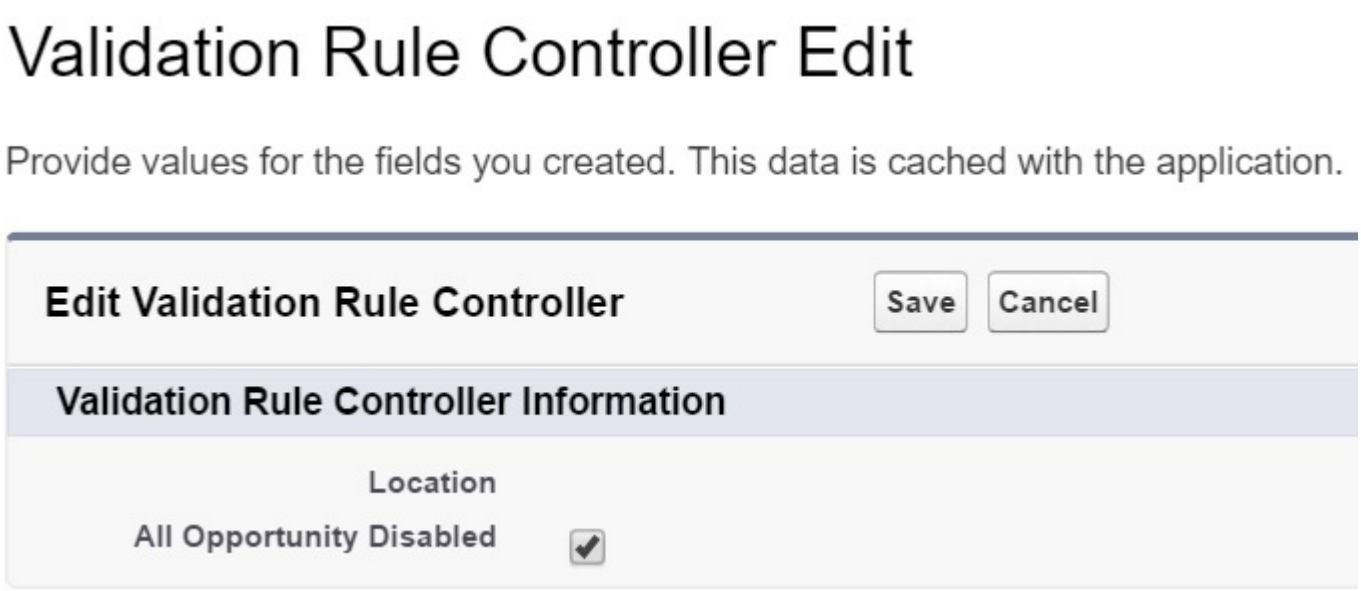
Custom Setting Field Value

When the field is checked the validation rule will be disabled, for the running user or in this

example, their profile -



The rule can also be disabled for a whole Salesforce org -



Validation Rule

```
AND(
  /* the below is the reference to the Val_Rule_Cntrlr__c custom setting's checkbox field
All_Opportunity_Disabled__c
  */
  $Setup.Val_Rule_Cntrlr__c.All_Opportunity_Disabled__c = FALSE,

  /* the below is the remainder of the validation rule's formula
  */
  CloseDate < TODAY()
)
```

In the above rule, both pieces of criteria must evaluate to `TRUE` in order for the rule to be triggered.

Since `All_Opportunity_Disabled__c` checkbox will evaluate to `TRUE` when the running user's profile is System Administrator, the rule will evaluate to `FALSE`.

Workflow Rules

The same approach can be applied in order to deactivate Workflow Rules.

Using Hierarchy Custom Settings To Disable Apex Code

Explanation

In this example a simple Trigger has been created to change the Close Date of an Opportunity, that's about to be inserted or updated, to a date 10 days in the future.

The Apex Controller custom setting's checkbox field enables the code to be disabled at the user / profile / org level.

Apex Class

```
trigger CloseDateUpdate on Opportunity (before insert, before update) {

    Id userId;
    Apx_Cntrlr__c userApexController;
    Boolean userSetting;

    userId = userinfo.getUserId();
    userApexController = Apx_Cntrlr__c.getInstance(userId);
    userSetting = userApexController.Close_Date_Update_Disabled__c;

    if (userSetting == false) {
        for(Opportunity opp : Trigger.new) {
            opp.CloseDate = date.today().addDays(10);
        }
    }

}
```

Unit Test

```
@isTest
public class CloseDateUpdateTest {

    @testSetup
    static void dataSetup() {

        Profile p = [SELECT Id FROM Profile WHERE Name = 'System Administrator' LIMIT 1];

        User u = new User(LastName = 'Test',Alias = 't1',Email = 'example@gmail.com',Username
= 'sotest@gmail.com',ProfileId = p.Id,TimeZoneSidKey = 'America/Denver',LocaleSidKey =
```

```
'en_US',EmailEncodingKey = 'UTF-8',LanguageLocaleKey = 'en_US');
        insert u;
    }

    static testMethod void testCloseDateUpdateEnabled() {

        User u = [SELECT Id FROM User WHERE Username = 'sotest@gmail.com'];
        // set the custom setting field to FALSE so that the trigger is not deactivated
        Apx_Cntrlr__c apexController = new Apx_Cntrlr__c(SetupOwnerId =
u.Id,Close_Date_Update_Disabled__c = false);
        upsert apexController;

        Opportunity[] opportunities1 = new Opportunity[]{};

        test.startTest();
        system.runAs(u){

                for(integer i = 0; i < 200; i++) {
                    opportunities1.add(new Opportunity(
                        Name          = 'Test Opp ' + i,
                        OwnerId       = u.Id,
                        StageName     = 'Prospecting',
                        CloseDate     = date.today().addDays(1),
                        Amount        = 100));
                }
            insert opportunities1;
        }
        test.stopTest();

        List<Opportunity> opportunities2 = [SELECT CloseDate FROM Opportunity];

        for(Opportunity o : opportunities2){
            system.assertEquals(date.today().addDays(10), o.closeDate, 'CloseDateUpdate
trigger should have changed the Opportunity close date as it was not disabled by the
apexController custom setting');
        }
    }

    static testMethod void testCloseDateUpdateDisabled() {

        User u = [SELECT Id FROM User WHERE Username = 'sotest@gmail.com'];
        // set the custom setting field to TRUE to deactivate the trigger
        Apx_Cntrlr__c apexController = new Apx_Cntrlr__c(SetupOwnerId =
u.Id,Close_Date_Update_Disabled__c = true);
        upsert apexController;

        Opportunity[] opportunities1 = new Opportunity[]{};

        test.startTest();
        system.runAs(u){

            for(integer i = 0; i < 200; i++) {
                opportunities1.add(new Opportunity(
                    Name          = 'Test Opp ' + i,
                    OwnerId       = u.Id,
                    StageName     = 'Prospecting',
                    CloseDate     = date.today().addDays(1),
                    Amount        = 100));
            }
            insert opportunities1;
        }
```

```
        test.stopTest();

        List<Opportunity> opportunities2 = [SELECT CloseDate FROM Opportunity];

        for(Opportunity o : opportunities2){
            system.assertEquals(date.today().addDays(1), o.closeDate, 'CloseDateUpdate trigger
should not have changed the Opportunity close date as it was disabled by the apexController
custom setting');
        }
    }

}
```

Updating Hierarchy Custom Settings in Apex Code

You may wish to update your custom setting's during the execution of your code, to switch off validation or workflow rules.

In the below code, I have created a Schedulable Apex Class which will update the Close Date of any Opportunities whose Close Date is less than or equal to 6 days from the current date, changing the date to 20 days in the future.

I will use my Custom Setting Val_Rule_Cntrlr__c to deactivate any validation rules which would prevent me from updating the Opportunities that meet my criteria.

```
global class Scheduled_OppCloseDateUpdate implements Schedulable {

    global void execute(SchedulableContext SC) {
        updOpportunityCloseDates();
    }

    global void updOpportunityCloseDates() {

        Id userId;
        Val_Rule_Cntrlr__c setting;
        Boolean validationRulesAlreadyDisabled;
        List<Opportunity> processedOpps = new List<Opportunity>();
        Date d;

        // get running user's Id
        userId = userinfo.getUserId();
        // retrieve Custom Setting status, for running user
        setting = Val_Rule_Cntrlr__c.getInstance(userId);

        // if the setting field is false, update it to disable validation rules
        if (setting.All_Opportunity_Disabled__c == false) {
            setting.All_Opportunity_Disabled__c = true;
            upsert setting;
        }
        // if the setting field was already true, there's no need to disable it
        // but it shouldn't be switched to false by this class once the process has been
completed
        else {
            validationRulesAlreadyDisabled = true;
        }

        // execute code to manage business process
```

```
        d = system.today().addDays(6);

        for(Opportunity o : [SELECT Id, CloseDate
                               FROM Opportunity
                              WHERE CloseDate <= :d
                                // class only updates open Opportunities
                                AND Probability > 0 AND Probability < 100])
        {
            o.CloseDate = System.today().addDays(20);
            processedOpps.add(o);
        }

        if (processedOpps.size() > 0) {
            update processedOpps;
        }

        // reactivate validation rules
        if (validationRulesAlreadyDisabled == false) {
            setting.All_Opportunity_Disabled__c = false;
            upsert setting;
        }

    }

}
```

To make sure that my validation rules are being deactivated by the changes to my custom setting in my class, I have created a checkbox field `Trigger_Validation_Rule__c` (which would not be visible to users or added to page layouts) & a validation rule with this criteria:

```
AND(
  $Setup.Val_Rule_Cntrlr__c.All_Opportunity_Disabled__c = FALSE,
  Trigger_Validation_Rule__c = TRUE,

  /* allow the above criteria to be met while inserting the Opportunities, without triggering
the rule, in the @testSetup portion of the test */
  NOT(ISNEW())
)
```

I then set the checkbox field to `true` when creating my Opportunities so that the rules criteria would be met, if the custom setting field is not edited by my code.

```
@isTest
private class WE_ScheduledCloseDateUpdateTest {

    @testSetup
    static void dataSetup() {

        Profile p = [SELECT Id FROM Profile WHERE Name = 'System Administrator' LIMIT 1];

        User u = new User(LastName = 'Test',Alias = 't1',Email = 'example@gmail.com',Username
= 'sotest@gmail.com',ProfileId = p.Id,TimeZoneSidKey = 'America/Denver',LocaleSidKey =
'en_US',EmailEncodingKey = 'UTF-8',LanguageLocaleKey = 'en_US');
        insert u;

        Val_Rule_Cntrlr__c valRuleCntrlr = new Val_Rule_Cntrlr__c(SetupOwnerId =
u.Id,All_Opportunity_Disabled__c = false);
```

```
        upsert valRuleCntrlr;

        List<Opportunity> testOpps = new List<Opportunity>();

        // create the Opportunities that will be updated by the class
        for(integer i = 0; i < 200; i++) {
            testOpps.add(new Opportunity(
                Name          = 'Test Opp Update' + i,
                OwnerId       = u.Id,
                StageName     = 'Prospecting',
                CloseDate     = date.today().addDays(1),
                Amount        = 100,
                // set checkbox field to true, to trigger validation rules if they've not been
deactivated by class
                Trigger_Validation_Rule__c = true));
        }
        // create the Opportunities that won't be updated by the class
        for(integer i = 0; i < 200; i++) {
            testOpps.add(new Opportunity(
                Name          = 'Test Opp Skip' + i,
                OwnerId       = u.Id,
                StageName     = 'Prospecting',
                CloseDate     = date.today().addDays(15),
                Amount        = 100,
                Trigger_Validation_Rule__c = true));
        }
        insert testOpps;

    }

    // code required to test a scheduled class, see
https://developer.salesforce.com/docs/atlas.en-us.apexcode.meta/apexcode/apex_scheduler.htm
for more details
    public static String CRON_EXP = '0 0 0 15 3 ? 2022';

    static testmethod void testCloseDateUpdates() {

        // execute scheduled class

        Test.startTest();

        String jobId = System.schedule('ScheduleApexClassTest',
                                       CRON_EXP,
                                       new Scheduled_OppCloseDateUpdate());

        CronTrigger ct = [SELECT Id, CronExpression, TimesTriggered, NextFireTime
                          FROM CronTrigger
                         WHERE id = :jobId];

        System.assertEquals(CRON_EXP, ct.CronExpression);
        System.assertEquals(0, ct.TimesTriggered);
        System.assertEquals('2022-03-15 00:00:00', String.valueOf(ct.NextFireTime));

        Test.stopTest();

        // test results

        Integer updateCount = 0;
        Integer skipCount   = 0;

        List <Opportunity> opportunitys = [SELECT Id, Name, CloseDate FROM Opportunity];
```

```
        for(Opportunity o : opportunitys) {
            if (o.Name.contains('Update') &&
                updateCount == 0)
            {
                System.assertEquals(date.today().addDays(20), o.CloseDate, 'Opportunity\'s
Close Date should have been updated as it was less than 7 days away');
                updateCount = 1;
            }
            if (o.Name.contains('Skip') &&
                skipCount == 0)
            {
                System.assertEquals(date.today().addDays(15), o.CloseDate, 'Opportunity should
not have been updated as it\'s Close Date is more than 7 days away');
                skipCount = 1;
            }
        }
        // check that both lists of Opportunities have been tested
        System.assertEquals(2, updateCount + skipCount, 'Count should be 2 once all assertions
have been executed');

    }

    // check that the class does not change the custom setting's field to false, if it was
true before class was executed
    static testmethod void testSettingUpdates() {

        User u = [SELECT Id FROM User WHERE UserName = 'sotest@gmail.com'];

        // switch the custom setting field to true before the scheduled job executes
        Val_Rule_Cntrlr__c setting;
        setting = Val_Rule_Cntrlr__c.getInstance(u.Id);
        setting.All_Opportunity_Disabled__c = true;
        upsert setting;

        System.runAs(u) {

            Test.startTest();

            String jobId = System.schedule('ScheduleApexClassTest',
                                        CRON_EXP,
                                        new Scheduled_OppCloseDateUpdate());

            CronTrigger ct = [SELECT Id, CronExpression, TimesTriggered, NextFireTime
                              FROM CronTrigger
                             WHERE id = :jobId];

            System.assertEquals(CRON_EXP, ct.CronExpression);
            System.assertEquals(0, ct.TimesTriggered);
            System.assertEquals('2022-03-15 00:00:00', String.valueOf(ct.NextFireTime));

            Test.stopTest();
        }
        setting = Val_Rule_Cntrlr__c.getInstance(u.Id);

        // check that the class did not change the All_Opportunity_Disabled__c field to false
        System.assertEquals(true, setting.All_Opportunity_Disabled__c);
    }

}
```

Read Custom Settings online: https://riptutorial.com/salesforce/topic/4927/custom-settings

Chapter 6: Date Time Manipulation

Examples

Easily Find Last Day of a Month

If you need to find the last day of the month, you can do complicated DateTime gymnastics or you can use the following method.

Say you want to find the last day of February 2021. Do the following:

```
Integer month = 2;
Integer day = null;
Integer year = 2021;

// Create a new DateTime object for the first day of the month following
// the date you're looking for.
DateTime dtTarget = DateTime.newInstance(year, month, 1);
//In this case we would be sure that month would not be out of bound
dtTarget = dtTarget.addMonths(1);
// Then use the .addDays() method to add negative 1 days. This gives you
// the last day of the target month.
DateTime lastDayOfMonth = dtTarget.addDays(-1);
day = lastDayOfMonth.day();

System.debug(lastDayOfMonth);
System.debug(day);
```

This produces the following output in the Logs:

```
18:19:57:005 USER_DEBUG [15]|DEBUG|2021-02-28 08:00:00
18:21:10:003 USER_DEBUG [16]|DEBUG|28
```

This works for all the add methods, allowing you easily and quickly find DateTimes in the past.

Read Date Time Manipulation online: https://riptutorial.com/salesforce/topic/4928/date-time-manipulation

Chapter 7: Global Variables in classes

Introduction

In this topic I would like to mention all possible global variables which can be used in Apex code. Like [UserInfo Class][1]. I suggest we just list a global classes/variables and links. If you know about a global class/variable but can't find a documentation, please provide as much information as possible. [1]: https://developer.salesforce.com/docs/atlas.en-us.apexcode.meta/apexcode/apex_methods_system_userinfo.htm

Examples

UserInfo

getFirstName() - returns the context user's first name.
getLastName() - returns the context user's last name.

Read Global Variables in classes online: https://riptutorial.com/salesforce/topic/8174/global-variables-in-classes

Chapter 8: Global Variables on Visualforce pages

Examples

$Resource

Use the `$Resource` variable to reference static resources.

```
<img src="{!$Resource.myImage}" />
```

Can be used in conjunction with the `URLFOR` function to reference files inside of a zipped static resource:

```
<apex:includeScript value="{!URLFOR($Resource.myResources, 'js/app.js')}" />
```

$Label

The `$Label` variable can be used to display text defined in your custom labels.

```
<apex:outputText value="{!$Label.Welcome_Message}" />
```

You can do formatting with labels as well. Suppose you have a custom label named `Welcome_Message` defined as

```
Welcome to our site, {0}!
```

You could use the label to display a formatted message:

```
<apex:outputText value="{!$Label.Welcome_Message}"
rendered="{!NOT(ISBLANK($User.ContactId))}">
  <apex:param value="{!$User.Contact.FirstName}" />
</apex:outputText>
```

$User

The `$User` variable gives you access to all the standard and custom fields on the `User` object for the currently logged in user.

```
You are logged in as a user from {!$User.CompanyName}.
```

Read Global Variables on Visualforce pages online:
https://riptutorial.com/salesforce/topic/5184/global-variables-on-visualforce-pages

Chapter 9: Page Navigation with help of list wrapper class in sales force.

Introduction

Slaesforce StandardSetController hold only List of sObject, It will not holdslist of wrapper class object. Below source code demonstrates the usage of Paginating using Wrapper Class in sales-force.

Examples

Pagination Controller

Code example : Now Start with Creating the Pagination Controller

```
public with sharing class Pagination {

}
```

i am going to show all the contacts in the form of pagination and there should be one checkbox for each contact to selected or deselect contact to perform delete operation on contacts. So i need to be create a wrapper class to hold contact and Boolean Variable for selection. First create Wrapper class for controller Pagination Insert below code into pagination controller.

```
public class contactWrapper{
       public Contact cont {get;set;}
       public Boolean isSelected{get;set;}
       public contactWrapper(contact c,Boolean s)
       {
           cont=c;
           isSelected=s;
       }
   }
```

Now Retrieving Data in Apex to Paginate with the help of StandardSetController. The StandardSetController is an extremely powerful tool with built-in functionality that you can use to greatly simplify the custom code in your Visualforce pages. Because the server returns only the data for the page being requested, the StandardSetController can significantly reduce view state, especially compared to the view state you would get while using SOQL.

```
Public Integer noOfRecords{get; set;} // Future reference in Visual force Page
   Public Integer size{get;set;}  // Future reference in Visual force Page
   public final Integer Page_Size=10; // Number records in a Page should be displayed

public ApexPages.StandardSetController setCon {
       get{
           if(setCon == null){
```

```
                size=Page_Size;
                string queryString = 'Select Id,Name, Email, Birthdate, Phone, MobilePhone
from Contact order by Name';
                setCon = new
ApexPages.StandardSetController(Database.getQueryLocator(queryString));
                setCon.setPageSize(size);
                noOfRecords = setCon.getResultSize();
            }
            return setCon;
        }set;
    }
```

Now you have the contacts in the Variable setCon , whenever your requested for setCon.getRecords() it will retrieve the first 10 contact records from the setCon. Here i have create simple wrapper class to show you the demo. You can create your own wrapper class based on requirement. But always you must be aware that ApexPages.StandardSetController hold only List of sObject, it will not hold the list of wrapper class object . That's the reason i have written extra code below to accomplish this future in different way. Below code convert list of contacts into list of wrapper class objects, So that when ever you call contacts in visual force page it will receive the list of wrapper class object.

```
 public list<contactWrapper> contWpr{get;set;}
    public set<id> selectedContactIds{ get;private set;} // to maintain state of the selected
contact
                                                         // through out paginating

    public Pagination() {
       selectedContactIds=new  set<id>();
    }

      Public list<contactWrapper> getContacts(){
         contWpr =new list<contactWrapper>();
        for(Contact c: (List<Contact>)setCon.getRecords())
            if(selectedContactIds.contains(c.id))
                contWpr.add(new contactWrapper(c,true));
                else
                contWpr.add(new contactWrapper(c,false));
        return contWpr;
    }
```

Now you have written code for generating result But how to navigate across the page? This can be done with easy step with ApexPages.StandardSetController.Look at the below code beauty of the StandardSetController, No need to maintain page number,offset and limit etc.. Just use the StandardSetController methods. Copy the below code into Pagination controller.

```
public Boolean hasNext {
        get {
            return setCon.getHasNext();
        }
        set;
    }
    public Boolean hasPrevious {
        get {
            return setCon.getHasPrevious();
        }
```

```
        set;
    }

    public Integer pageNumber {
        get {
            return setCon.getPageNumber();
        }
        set;
    }

    public void first() {
        setCon.first();
        // do you operation here
    }

    public void last() {
        setCon.last();
            // do you operation here
    }

    public void previous() {
        setCon.previous();
            // do you operation here
    }

    public void next() {
        setCon.next();
            // do you operation here
    }
```

Your almost done with Paginating the contacts. Last few methods i have added to fulfill my entire page functionality. As i mention earlier we have additional checkbox for selecting contact and perform delete operation on selected contacts.

```
public void contactSelection()
    {
        Id id=(Id)ApexPages.currentPage().getParameters().get('cId');
        if(selectedContactIds.contains(id))
            selectedContactIds.remove(id);
            else
            selectedContactIds.add(id);
    }

 public void deleteContacts()
    {
        List<contact> contactToDelete=[select id from contact where id in
:selectedContactIds];
        if(contactToDelete.size()!=0)  //   if(!contactToDelete.isEmpty()) // Best Practice
           {
                try {  delete contactToDelete; }  // You may get Exception if you try to
delete the
                                                  // related contact ,include try block to avoid
error.
                catch(exception ex){ System.debug(ex); }
                refresh();
           }
    }

 public pageReference refresh() {
```

```
        setCon = null;
        selectedContactIds=new set<id>();
        getContacts();
        setCon.setPageNumber(1);
        return null;
    }
```

Read Page Navigation with help of list wrapper class in sales force. online: https://riptutorial.com/salesforce/topic/10744/page-navigation-with-help-of-list-wrapper-class-in-sales-force-

Chapter 10: SalesForce CI Integration

Introduction

Place to use Jenkins and Sonar for CI

Examples

How to configure Jenkins to deploy code on Development or Production org ?

How we can use jenkins in our SalesForce product development. What are the tools plugins are available for Jenkins Integration How to solve CI configuration issue.....etc

Jenkins CI tools which can be used for SalesForce Automation

1. Jenkins: The leading open source automation server, Jenkins provides hundreds of plugins to support building, deploying and automating any project.
2. Sonar Qube: SonarQube provides the capability to not only show health of an application but also to highlight issues newly introduced.
3. Apache Ant: Apache Ant is a Java library and command-line tool whose mission is to drive processes described in build files as targets and extension points dependent upon each other.
4. Apache Maven: Apache Maven is a software project management and comprehension tool. Based on the concept of a project object model (POM), Maven can manage a project's build, reporting and documentation from a central piece of information.
5. SfApexDoc: Support for JavaDoc like documentation creation tool. Can be used by Ant/Jenkins to create Documents.
6. JUnit format Report for APEX: Extends the Force.com com.salesforce.ant.DeployTask to accept an optional junitreportdir argument that defines the folder that a JUnitReport XML file is output into. This file can be consumed directly by the Jenkins continuous integration tool to produce trend graphs and test result details or by the JUnitReport Ant task.
7. Version Control System: Can use **GIT**, **SVN** or any other Version Control system
8. PMD Apex: Contains the PMD implementation to support the Apex programming language.
9. Sonar for Apex(enforce-sonarqube-plugin): The plugin has support for the Apex language grammar, the current list of checks is focused mainly on test components. The support for more SFDC components is in progress.

Read SalesForce CI Integration online: https://riptutorial.com/salesforce/topic/10044/salesforce-ci-integration

Chapter 11: Salesforce Object Query Language (SOQL)

Syntax

- SELECT Id FROM Account
- SELECT Id, Name FROM Account
- SELECT Id FROM Account WHERE Name = 'SomeAccountName'
- SELECT Id, Name, (SELECT Id, Name FROM Contacts) FROM Account
- SELECT Id, Name FROM Account WHERE Id = :apexVariableName

Examples

Basic SOQL Query

```
SELECT Id, Name FROM Account
```

This will return the Id and Name fields from the Account table. No filtering or sorting will be applied.

SOQL Query With Filtering

```
SELECT Name FROM User WHERE IsActive = true
```

This will return the name of all active Users.

```
SELECT Name, Phone FROM Contact WHERE CreatedDate >= 2016-01-01T00:00:00.000Z
```

This will return Contacts created on or after January 1st, 2016.

```
SELECT Id, Name FROM Account LIMIT 100
```

This will return the first 100 Accounts from an unordered list.

```
SELECT Id, Name, Phone FROM Lead WHERE Phone LIKE '(%) %-%'
```

This will return Leads with a phone number matching the specified format. "%" acts as a wild card character.

Using `LIKE '% %'` also enables a developer to replicate a `CONTAINS( )` formula.

```
SELECT Email FROM Lead WHERE LeadSource LIKE '%Google%'
```

Will return Leads with a lead source that contains Google i.e. 'Google AdWords' & 'Google Natural Search'.

SOQL Query With Ordering

```
SELECT Id, Name FROM User ORDER BY LastName

SELECT Id, Name FROM Contact ORDER BY LastModifiedDate DESC

SELECT Name, Title FROM User ORDER BY Title ASC NULLS FIRST

SELECT Id FROM Contact ORDER BY LastName ASC NULLS LAST, FirstName ASC NULLS FIRST
```

Using SOQL to Construct a Map

A very useful feature many people overlook is the ability to construct a Map using a SOQL query.

```
Map<Id, Account> accounts = new Map<Id, Account>([SELECT Id, Name FROM Account]);
System.debug(accounts);
```

When you run this code, `accounts` then contains a Map of your Account objects, keyed on Id. The output to the debug log would look similar to this:

```
11:15:10:025 USER_DEBUG [13]|DEBUG|{
    XXXXXXXXXXXXXXXXXX=Account:{Id=XXXXXXXXXXXXXXXXXX, Name=Account 1},
    YYYYYYYYYYYYYYYYYY=Account:{Id=YYYYYYYYYYYYYYYYYY, Name=Account 2},
    ZZZZZZZZZZZZZZZZZZ=Account:{Id=ZZZZZZZZZZZZZZZZZZ, Name=Account 3},
    ...
}
```

You are now able to look up the Account objects using their Id. Furthermore, if you want a collection of unique IDs, you can call the `keySet()` function of the Map class, like so:

```
System.debug(accounts.keySet());
```

which looks something like this in the debug log:

```
11:23:21:010 USER_DEBUG [15]|DEBUG|{XXXXXXXXXXXXXXXXXX, YYYYYYYYYYYYYYYYYY,
ZZZZZZZZZZZZZZZZZZ, ...}
```

This is very useful when you need to query to get records and access them repeatedly in your code.

SOQL Query to Reference Parent Object's Fields

When object's are linked by a lookup or master-detail relationship, the parent records field's can be referenced from the child record or 'base object' in a query. This is also known as upwards traversal.

```
SELECT FirstName, Account.Name, Account.Category__c FROM Contact
```

It's possible to traverse five records upwards.

```
SELECT Account.Owner.Profile.CreatedBy.Name FROM Contact
```

When the base object is a custom lookup field, the __c in field's name Primary_Influencer__c, for example, will be changed to __r.

```
SELECT Primary_Influencer__r.Nickname__c FROM Contact
```

SOQL Query to get child Records

```
SELECT Id, Name, (SELECT Id, FirstName, LastName FROM Contacts) FROM Account
```

SOQL Queries in Apex

To perform a query in Apex, surround the query with square brackets. The result can be assigned to a list, or to a single object.

```
List<Account> allAccounts = [SELECT Id, Name FROM Account];
Account oldestAccount = [SELECT Id, Name FROM Account ORDER BY CreatedDate LIMIT 1];
```

Variable References in Apex SOQL Queries

To reference a variable in a query, add a colon (:) before the variable name.

```
Datetime targetDate = Datetime.now().addDays(-7);
List<Lead> recentLeads = [SELECT Id FROM Lead WHERE CreatedDate > :targetDate];

string targetName = 'Unknown';
List<Contact> incompleteContacts = [SELECT Id FROM Contact WHERE FirstName = :targetName];
```

Potential Exceptions in Apex SOQL Queries

When assigning to a single object, a query that returns anything other than a single row will throw a `QueryException`.

```
try {
    Account a = [SELECT Id FROM Account WHERE Name = 'Non-existent Account'];
} catch (QueryException e) {
    // List has no rows for assignment to SObject
}

try {
    Account a = [SELECT Id FROM Account];
} catch (QueryException e) {
    // List has more than 1 row for assignment to SObject
}
```

Attempting to use a field that you did not include in the query will throw a `SObjectException`

```
Account a = [SELECT Id FROM Account LIMIT 1];
try {
    System.debug( a.Name );
} catch (SObjectException e) {
    // SObject row was retrieved via SOQL without querying the requested field: Name
}
```

Using a Semi-Join

Selecting all accounts that have open opportunity records under them

```
SELECT Id, Name FROM Account WHERE AccountId IN
    (SELECT Id FROM Opportunity WHERE IsClosed = false)
```

Dynamic SOQL

You can execute a database query from a String rather than a regular SOQL expression:

```
String tableName = 'Account';
String queryString = 'SELECT Id FROM ' + tableName + ' WHERE CreatedDate >= YESTERDAY';
List<SObject> objects = Database.query(queryString);
```

Since dynamic SOQL queries are not compiled, their schema references are not validated, so it is preferable to use Apex variable interpolation using the `:variable` syntax where possible.

Read Salesforce Object Query Language (SOQL) online:
https://riptutorial.com/salesforce/topic/4217/salesforce-object-query-language--soql-

Chapter 12: Salesforce REST API

Introduction

Force.com REST API Documentation. Full list of API's is here

Examples

OAuth2 access_token and list of services

To get OAuth2 access token simply do

```
curl https://login.salesforce.com/services/oauth2/token -d "grant_type=password" -d
"client_id=myclientid" -d "client_secret=myclientsecret" -d "username=mylogin@salesforce.com" -d
"password=mypassword123456"
```

You should get response something like

```
{
  "access_token":
"00D6F0xxx001g1qs!ARsAQL7BRiQQ0lgTW7zXu3kILJBxxxxxHvDnChF2ETBFJpX0T2LsBsm8MVABhAvINAyZqgDIAHhJDp6QjuF6Z

  "instance_url": "https://ap4.salesforce.com",
  "id": "https://login.salesforce.com/id/00D6F000001xxxxAA/0056F000006DMcxxxx",
  "token_type": "Bearer",
  "issued_at": "14878401xxxxx",
  "signature": "Ra5Sdm6gq4xxxeZYk3H2yBIVpZ6hBUDgkQ4Tjp9Q="
}
```

Then do

```
{
  "tooling": "/services/data/v20.0/tooling",
  "eclair": "/services/data/v20.0/eclair",
  "prechatForms": "/services/data/v20.0/prechatForms",
  "async-queries": "/services/data/v20.0/async-queries",
  "query": "/services/data/v20.0/query",
  "chatter": "/services/data/v20.0/chatter",
  "wave": "/services/data/v20.0/wave",
  "search": "/services/data/v20.0/search",
  "identity": "https://login.salesforce.com/id/00D6F000001g1qsUAA/0056F000006DMcMQAW",
  "sobjects": "/services/data/v20.0/sobjects",
  "serviceTemplates": "/services/data/v20.0/serviceTemplates",
  "recent": "/services/data/v20.0/recent",
  "connect": "/services/data/v20.0/connect",
  "licensing": "/services/data/v20.0/licensing"
}
```

Read Salesforce REST API online: https://riptutorial.com/salesforce/topic/9210/salesforce-rest-api

Chapter 13: Tools for Development

Examples

IDEs

A list of available IDEs to create classes, triggers, Visualforce/Lightning pages/components.

- Force.com IDE – plugin for Eclipse
- JedIDE – plugin for IntelliJ IDEA & standalone Force.com IDE
- MavensMate – plugin for Sublime Text and Atom and VS Code
- FuseIT SFDC Explorer - This is a standalone tool
- Welkin Suite – This is a standalone tool;
- Illuminated Cloud - plugin for IntelliJ IDE
- Aside.io - Web-based IDE
- Cloud 9 - Web-based IDE
- VimAwesome - VIM plugin for Force.com
- HaoIDE - Sublime Text plugin for Force.com
- Metaforce - A lightweight Chrome app for Salesforce develoment

Browser extensions

- Salesforce Navigator (Google Chrome)
- Force.com Logins (Google Chrome, Firefox)
- Salesforce Developer Tool Suite (Google Chrome)
- Salesforce Lighting Components Inspector (Google Chrome)
- Salesforce Developer Tool Suite (Google Chrome)
- Salesforce Schema Builder Expander (Google Chrome)
- Boostr for Salesforce (Google Chrome)
- Salesforce API Fieldnames (GoogleChrome)
- Changeset Helper (Google Chrome)
- Salesforce Inspector (Google Chrome, Firefox)
- Salesforce Mass Editor (Google Chrome)
- Space (Google Chrome)

Debuggers

- Official Apex Debugger
    - Realtime debugging
    - Force.com IDE
    - requires special licensing from salesforce
- Welkin Suit
    - Log replay debugging

    - Stand alone IDE
    - Subscription required

- Illuminated Cloud:

    - Log replay debugging
    - JetBrains Extension
    - Subscription required

- Salesforce Apex Debug

    - Log replay debugging
    - VS Code Extension
    - Free and open source
    - Currently still in alpha

Salesforce ETL tools

- Salesforce DataLoader
- Dataloader.io
- Jitterbit
- SFXOrgData
- DreamFactory Monarch
- Pentaho Kettle
- Talend

Static Analysis Tools

- CodeClimate: Cloud Service
- CodeScan: Cloud Service
- Clayton.io: Cloud Service
- VSCode Apex PMD: VS Code extension for realtime static analysis as you code
- Apex PMD: command line core that runs most the above tools

Read Tools for Development online: https://riptutorial.com/salesforce/topic/4095/tools-for-development

Chapter 14: Trigger Bulkification

Examples

Bulkification

If you do row-by-row processing in Salesforce, you'll probably reach the governor limit quickly. This is especially true with triggers and things that fire when you don't expect them. One documented method of escaping the governor limit is bulkification.

Note: The following information is based on the official Salesforce docs.

Bulkifying Apex code means making sure that the code properly handles more than one record at a time. When a batch of records initiate Apex, a single instance of that Apex code is executed, but that instance needs to handle all of the records in that given batch.

Not Bulkified:

```
trigger accountTestTrggr on Account (before insert, before update)
{

   //This only handles the first record in the Trigger.new collection
   //But if more than 1 Account initiated this trigger, those additional records
   //will not be processed
   Account acct = Trigger.new[0];
   List<Contact> contacts = [select id, salutation, firstname, lastname, email
              from Contact where accountId =&amp;nbsp;:acct.Id];

}
```

Bulkified:

```
trigger accountTestTrggr on Account (before insert, before update)
{
    List<String> accountNames = new List<String>{};
   //Loop through all records in the Trigger.new collection
   for(Account a: Trigger.new){
      //Concatenate the Name and billingState into the Description field
      a.Description = a.Name + ':' + a.BillingState
   }
}
```

Read Trigger Bulkification online: https://riptutorial.com/salesforce/topic/4272/trigger-bulkification

Chapter 15: Visualforce Page Development

Examples

Basic page

A basic VisualForce page can be created like this:

```
<apex:page>
  <h1>Hello, world!</h1>
</apex:page>
```

Using Standard Controllers

If your page is for displaying or editing information about a particular type of record, it may be helpful to use a standard controller to reduce the amount of boilerplate code you need to write.

By using a standard controller, your page will be displayed with an `?id=SALESFORCE_ID` parameter, and you automatically get access to all merge fields on the record.

Add a standard controller to your page by specifying the `standardController` attribute on `<apex:page>`:

```
<apex:page standardController="Account">
  This is a page for {!Account.Name}
</apex:page>
```

You also get the standard controller methods for free:

- `cancel()` - returns the `PageReference` for the cancel page (usually navigates back to a list view)
- `delete()` - deletes the record and returns the `PageReference` for the delete page
- `edit()` - returns the `PageReference` for the standard edit page
- `save()` - saves the record and returns the `PageReference` to the updated record
- `view()` - returns the `PageReference` for the standard view page

You can use them like this:

```
<apex:page standardController="Account">
  Name: <apex:inputField value="{!Account.Name}" />
  <apex:commandButton value="Update record" action="{!save}" />
</apex:page>
```

Read Visualforce Page Development online:
https://riptutorial.com/salesforce/topic/6372/visualforce-page-development

Chapter 16: Working with External Systems

Examples

Making an outbound callout

This is an example on how to call a web service from salesforce. The code below is calling a REST based service hosted on data.gov to find farmers markets close to the zipcode.

Please remember in order to invoke a HTTP callout from your org, you need to tweak the remote settings for the org.

```
string url= 'http://search.ams.usda.gov/farmersmarkets/v1/data.svc/zipSearch?zip=10017';
Http h = new Http();
HttpRequest req = new HttpRequest();
HttpResponse res = new HttpResponse();
req.setEndpoint(url);
req.setMethod('GET');
res = h.send(req);
System.Debug('response body '+res.getBody());
```

Read Working with External Systems online: https://riptutorial.com/salesforce/topic/4926/working-with-external-systems

Credits

S. No	Chapters	Contributors
1	Getting started with Salesforce	abhi, Alex S, Andrii Muzychuk, Ashwani, Community, Reshma
2	Apex Testing	Andree Wille, battery.cord, Ben, Eric Dobbs
3	Apex Triggers	kurunve, Pedro Otero
4	Approval Process Objects	Ajay Gupta
5	Custom Settings	Alex S
6	Date Time Manipulation	kurunve, LDP, RamenChef
7	Global Variables in classes	Andrii Muzychuk
8	Global Variables on Visualforce pages	Ben
9	Page Navigation with help of list wrapper class in sales force.	NITHESH K
10	SalesForce CI Integration	Sanjay Kharwar
11	Salesforce Object Query Language (SOQL)	abhi, Alex S, Andrii Muzychuk, battery.cord, Ben, ca_peterson, Doug B, Eric Dobbs, LDP, Ratan Paul
12	Salesforce REST API	radbrawler
13	Tools for Development	abhi, Andrii Muzychuk, Daniel Ballinger, Force2b, Gres, hleb not bread, itzmukeshy7, Mahmood, NSjonas, Pavel Slepiankou, pchittum, Ratan Paul, sorenkrabbe, wintermute
14	Trigger Bulkification	abhi, hillary.fraley, LDP
15	Visualforce Page Development	Ben

16	Working with External Systems	abhi, mnoronha