47. Schedulable Batch Class - 13 June 2022

**Definition:** When a class implements Schedulable interface, then that class is known as “**Schedulable**” Class.

This interface contain a method :

public void execute(SchedulableContext SC){

}

**=============================**

**User Story 1] LIC Policy**

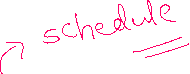
13 June 2022 🡪 LIC Policy 🡪 Status “Active”.

31st March 2023 🡪 LIC Policies 🡪 Status “Expired”.

**User Story 2] Diwali Greetings Mail 🡺 24 Oct 2022**

**=================================**

**User Story 3] Update Opp Amount = 0, if Stage is “Prospecting”. Run this Batch class on 13th June 2022 at 08:30 PM.**



global BatchClass{

start(){

}

Execute(){

}

finish(){

}

}

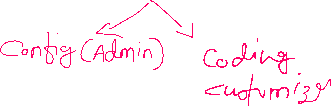
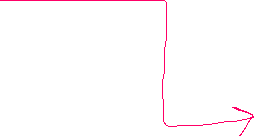
public ScheduleBatchClass implements Schedulable {

public void execute(SchedulableContext SC){

Database.executeBatch(new BatchClass());

}

}



global class BatchOpportunityAmount implements Database.Batchable<sObject> {

global Database.QueryLocator start(Database.BatchableContext BC){

return Database.getQueryLocator('select Id, Amount, StageName from Opportunity where StageName=\'Prospecting\' ');

}

global void execute(Database.BatchableContext BC, List<Opportunity> oppList){

if(!oppList.isEmpty()){

for(Opportunity objOpp : oppList){

objOpp.Amount = 0;

}

Database.update(oppList,false);

}

}

global void finish(Database.BatchableContext BC){

}

}

public class SchedulableBatchOpportunityAmount implements Schedulable {

public void execute(SchedulableContext SC){

Database.executeBatch(new BatchOpportunityAmount(),100);

}

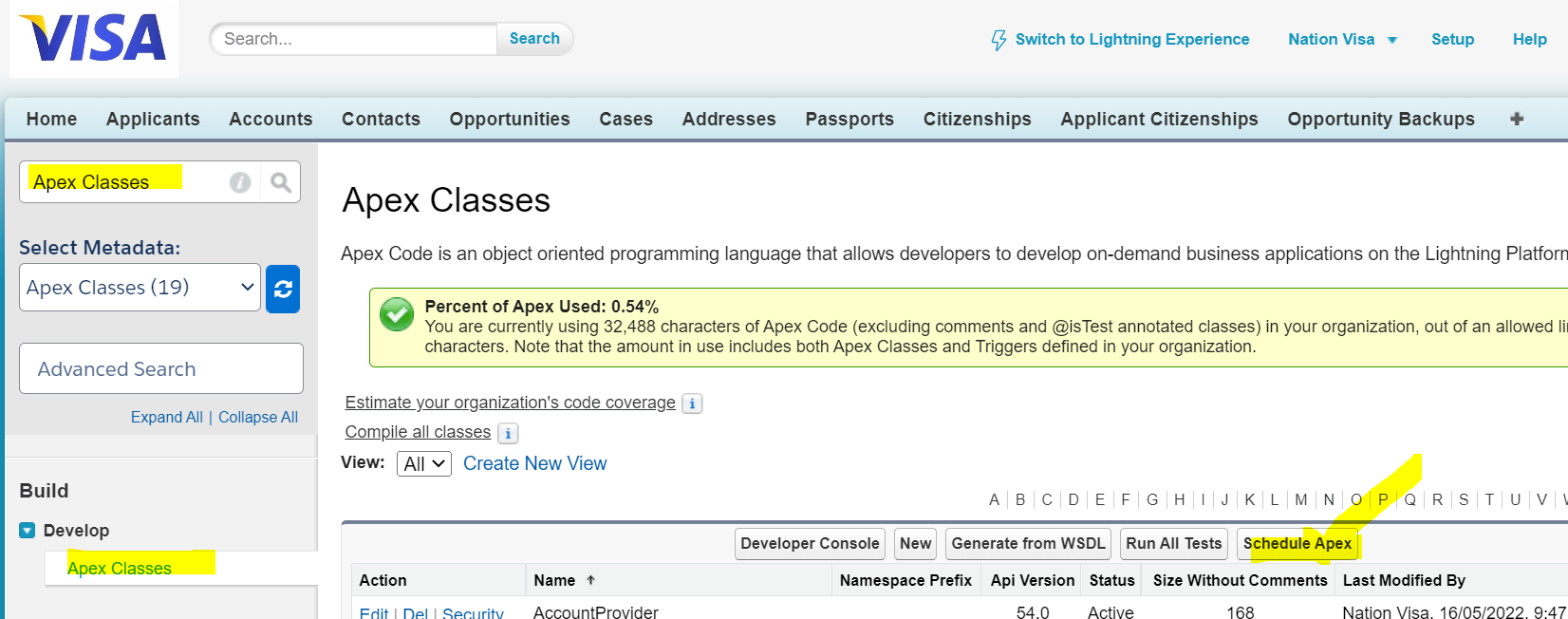
}

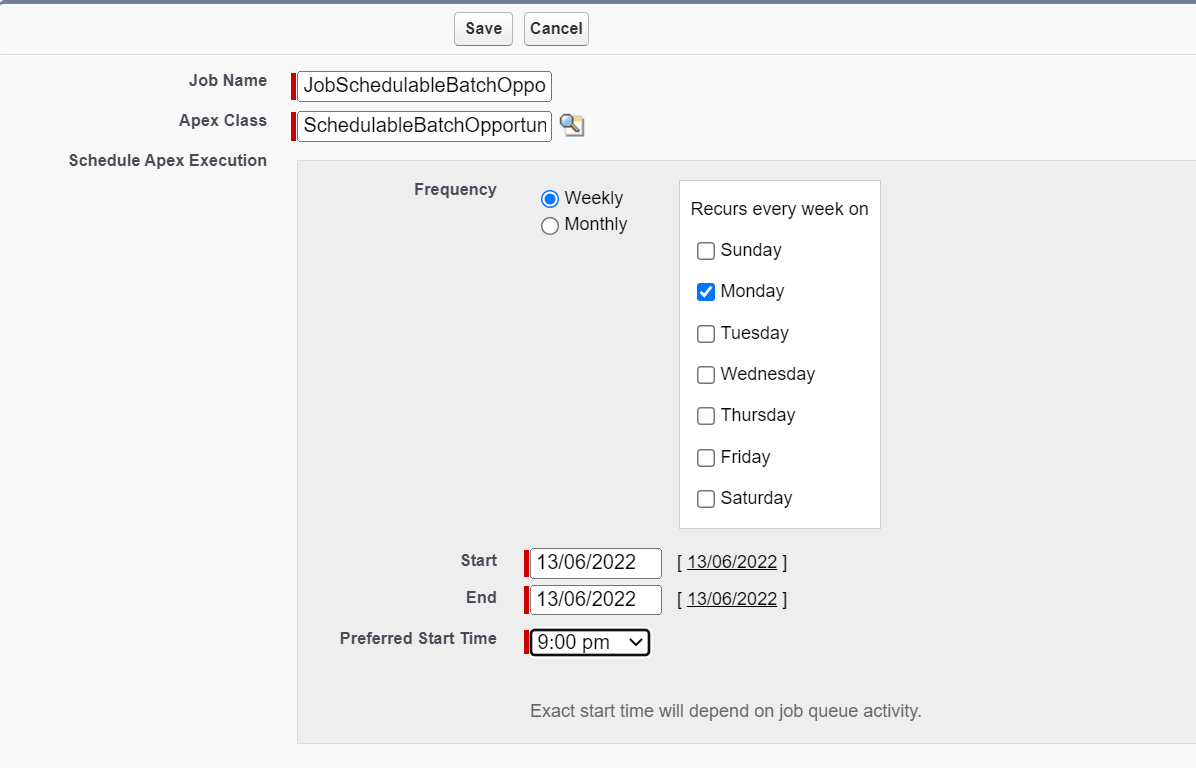
How to Schedule:

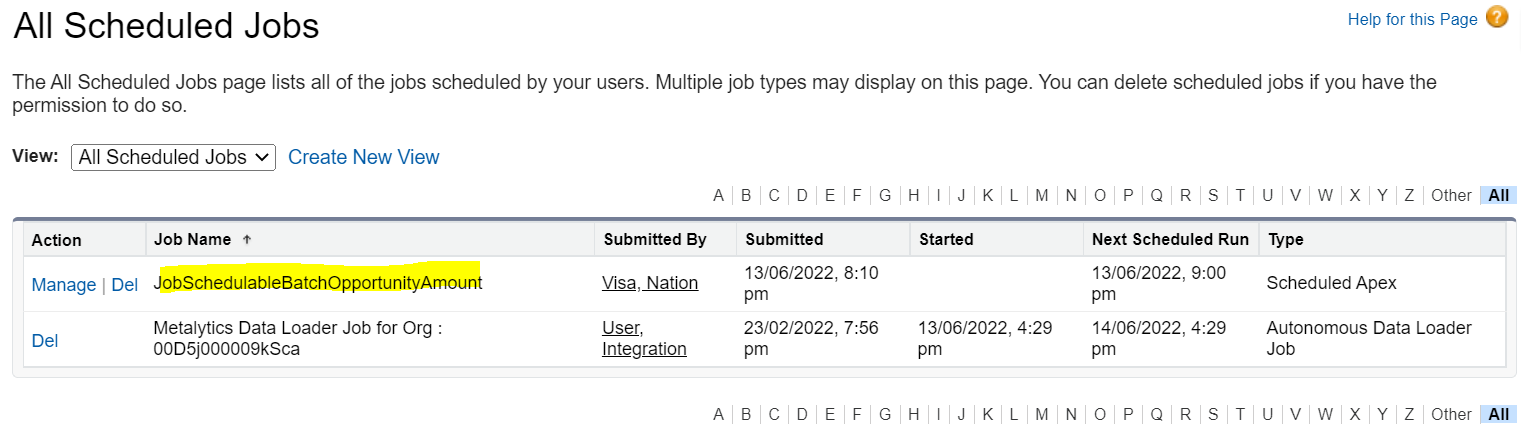
1] Admin (Config)

2] Coding (Cust)

1] Admin (Config)





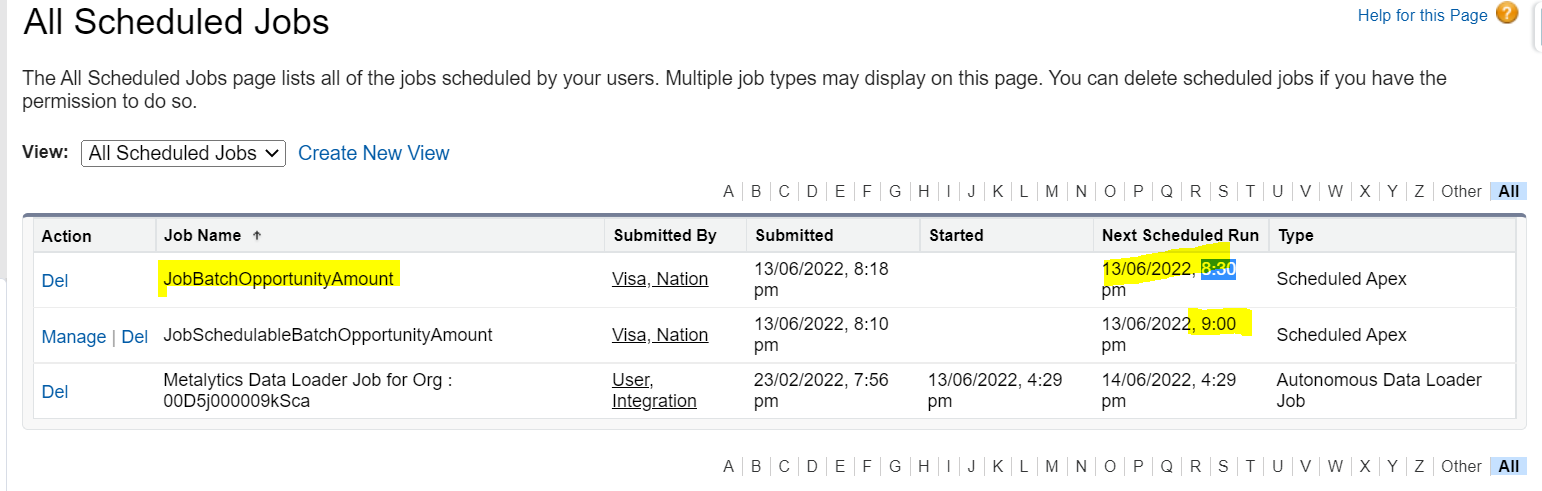


**2] Coding (Customization)**

SchedulableBatchOpportunityAmount objSchBatch = new SchedulableBatchOpportunityAmount();

String cronExpression = '0 30 20 13 6 ? 2022' ;//SS:MM:HH DD MM ? YYYY

System.schedule('JobBatchOpportunityAmount', cronExpression, objSchBatch);



**1] 14 June 2022 , 03:15 PM**

**CronExp : ‘0 15 15 14 6 ? 2022’;**

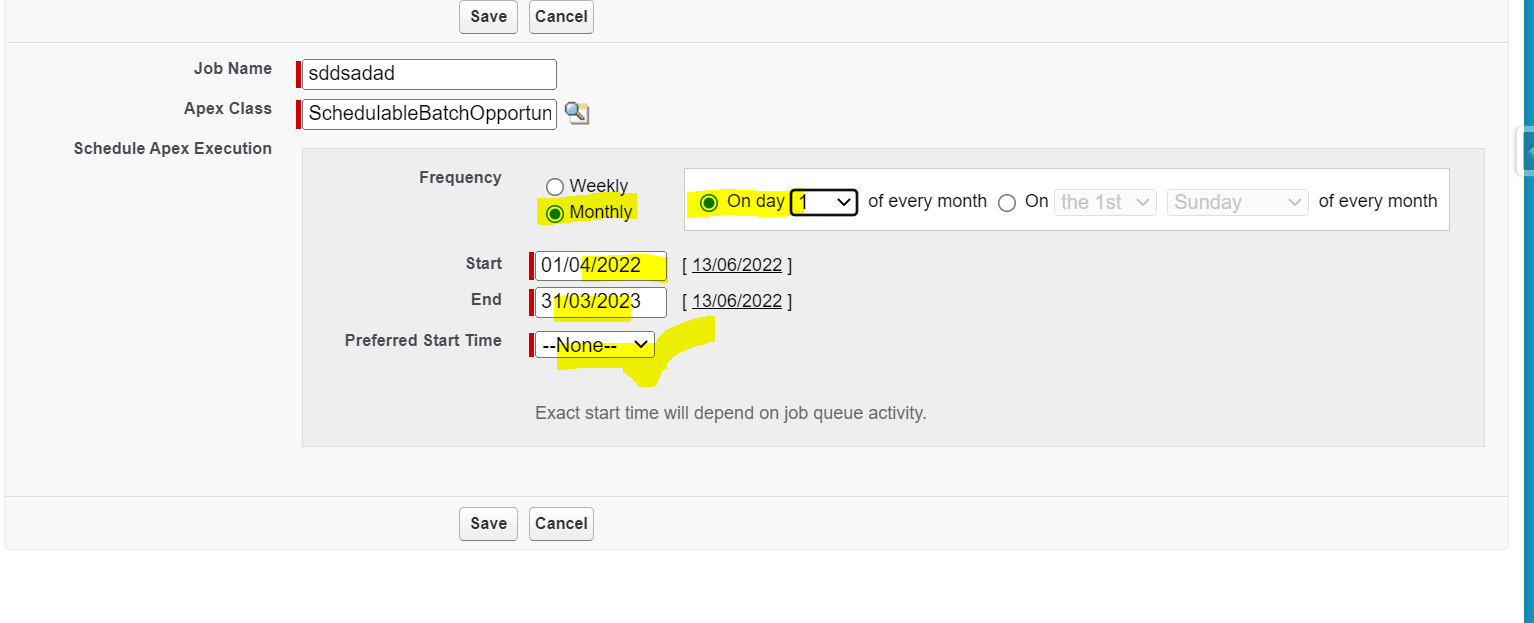
**2] Daily 12:00 AM**

**CronExp = ‘0 0 0 \* \* ? \*’;**

**Admin : Every Month : 01 Days**

**Start : 01 April 2022**

**End : 31st March 2023**



**Trigger:**

**Scenario 8]**

|  |  |
| --- | --- |
| **Contact** | **Lead** |
| [a@gmail.com](mailto:a@gmail.com) | c@gmail.com |
| b@gmail.com | [b@gmail.com](mailto:b@gmail.com) 🡺 Error |

trigger LeadTrigger on Lead (before Insert , before update){

Set<String> emailSet = new Set<String>();

for(Lead objLead : trigger.new){

if(String.isNotBlank(objLead.Email))

emailSet.add(objLead.Email);

}

Map<String, Contact> conMap = new Map<String,Contact>();

for(Contact objCon : [select Id, Email from Contact where Email IN : emailSet]){

conMap.put(objCon.Email, objCon);

}

for(Lead objLead : trigger.new){

if(conMap.containsKey(objLead.Email)){

objLead.Email.addError('This Email already exists on Contact');

}

}

}

=============

TeamMember:

trigger OppTriggerTeam on Opportunity (after insert) {

List<AccountTeamMember> accListMemberList = new List<AccountTeamMember>();

for(Opportunity objOpp : trigger.new){

if(objOpp.Probability == 50){

AccountTeamMember ca = new AccountTeamMember();

ca.AccountId = objOpp.AccountId;

ca.TeamMemberRole = 'Sales Rep';

ca.UserId = objOpp.OwnerId;

ca.AccountAccessLevel = 'Read';

ca.CaseAccessLevel = 'Read';

ca.OpportunityAccessLevel = 'Read';

accListMemberList.add(ca);

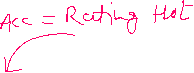
}

}

Database.insert(accListMemberList,false);

}

**Scenario 14]**



|  |  |
| --- | --- |
| **AccountTrigger1** | **AccountTrigger2** |
| After Update  If Rating = Hot 🡺 SLA = Gold | After Update  If SLA = Gold 🡺 Rating =Hot |

