**Use Case 1: Automatic Field Update**

**Scenario:**  
When an **Opportunity** is marked as *Closed Won*, the related **Account Status** should automatically update to *Active*.

**Recommended Solution:**

* Create an **after update trigger** on Opportunity.
* Update related Account records inside a **bulkified Apex trigger**.
* Use a **helper Apex class** to handle the logic.

trigger OpportunityTrigger on Opportunity (after update) {

if(Trigger.isAfter && Trigger.isUpdate){

OpportunityTriggerHandler.updateAccountStatus(Trigger.new, Trigger.oldMap);

}

}

public class OpportunityTriggerHandler {

public static void updateAccountStatus(List<Opportunity> newOpps, Map<Id, Opportunity> oldOppMap){

Set<Id> accountIdsToUpdate = new Set<Id>();

// Collect Accounts where Opportunity moved to Closed Won

for(Opportunity opp : newOpps){

Opportunity oldOpp = oldOppMap.get(opp.Id);

if(opp.StageName == 'Closed Won' && oldOpp.StageName != 'Closed Won'){

if(opp.AccountId != null){

accountIdsToUpdate.add(opp.AccountId);

}

}

}

if(!accountIdsToUpdate.isEmpty()){

List<Account> accList = [SELECT Id, Status\_\_c FROM Account WHERE Id IN :accountIdsToUpdate];

for(Account acc : accList){

acc.Status\_\_c = 'Active'; // Custom field on Account

}

update accList;

}

}

}

@isTest

public class OpportunityTriggerHandlerTest {

@isTest

static void testUpdateAccountStatus(){

// Create Account

Account acc = new Account(Name = 'Test Account', Status\_\_c = 'Inactive');

insert acc;

// Create Opportunity (Not Closed Won initially)

Opportunity opp = new Opportunity(

Name = 'Test Opportunity',

StageName = 'Prospecting',

CloseDate = Date.today().addDays(30),

AccountId = acc.Id

);

insert opp;

// Update Opportunity to Closed Won

opp.StageName = 'Closed Won';

update opp;

// Fetch updated Account

Account updatedAcc = [SELECT Id, Status\_\_c FROM Account WHERE Id = :acc.Id];

System.assertEquals('Active', updatedAcc.Status\_\_c, 'Account should be updated to Active');

}

}

## ****Use Case 2: Prevent Duplicate Contacts****

**Scenario:**  
Cool Air wants to prevent creating **duplicate Contacts** with the same **email address** under the same Account.

**Recommended Solution:**

* Write a **before insert trigger** on Contact.
* Query existing Contacts for the same email + AccountId.
* Add an **addError()** message if duplicate found.

trigger ContactTrigger on Contact (before insert, before update) {

if(Trigger.isBefore){

if(Trigger.isInsert || Trigger.isUpdate){

ContactTriggerHandler.preventDuplicateContacts(Trigger.new);

}

}

}

public class ContactTriggerHandler {

public static void preventDuplicateContacts(List<Contact> newContacts){

// Collect AccountIds and Emails from incoming records

Map<Id, Set<String>> accountEmailMap = new Map<Id, Set<String>>();

for(Contact con : newContacts){

if(con.AccountId != null && String.isNotBlank(con.Email)){

if(!accountEmailMap.containsKey(con.AccountId)){

accountEmailMap.put(con.AccountId, new Set<String>());

}

accountEmailMap.get(con.AccountId).add(con.Email.toLowerCase());

}

}

// Query existing Contacts for those Accounts and Emails

List<Contact> existingContacts = [

SELECT Id, Email, AccountId

FROM Contact

WHERE AccountId IN :accountEmailMap.keySet()

AND Email != null

];

// Build lookup for existing emails

Map<Id, Set<String>> existingEmailMap = new Map<Id, Set<String>>();

for(Contact ec : existingContacts){

if(!existingEmailMap.containsKey(ec.AccountId)){

existingEmailMap.put(ec.AccountId, new Set<String>());

}

existingEmailMap.get(ec.AccountId).add(ec.Email.toLowerCase());

}

// Validation: Check duplicates

for(Contact con : newContacts){

if(con.AccountId != null && String.isNotBlank(con.Email)){

if(existingEmailMap.containsKey(con.AccountId) &&

existingEmailMap.get(con.AccountId).contains(con.Email.toLowerCase())){

con.addError('Duplicate Contact: A Contact with this Email already exists under this Account.');

}

}

}

}

}

@isTest

public class ContactTriggerHandlerTest {

@isTest

static void testPreventDuplicateContacts(){

// Create Account

Account acc = new Account(Name = 'Cool Air Test Account');

insert acc;

// Create first Contact

Contact c1 = new Contact(

FirstName = 'John',

LastName = 'Doe',

Email = 'john.doe@coolair.com',

AccountId = acc.Id

);

insert c1;

// Try inserting duplicate Contact with same Email under same Account

Contact c2 = new Contact(

FirstName = 'Jane',

LastName = 'Smith',

Email = 'john.doe@coolair.com',

AccountId = acc.Id

);

Test.startTest();

Database.SaveResult sr = Database.insert(c2, false); // allOrNone = false

Test.stopTest();

// Verify error

System.assertEquals(false, sr.isSuccess(), 'Duplicate should not be allowed');

System.assert(sr.getErrors()[0].getMessage().contains('Duplicate Contact'));

}

}

## ****Use Case 3: Roll-Up Summary for a Non-Master Detail Relationship****

**Scenario:**  
For each **Account**, management wants to store the **total number of open Cases** in a custom field. This needs to update whenever Cases are created, closed, or deleted.

**Recommended Solution:**

* Write an **after insert, after update, after delete trigger** on Case.
* Aggregate **COUNT()** of Cases grouped by AccountId.
* Update the parent Account’s custom field with the value.

trigger CaseTrigger on Case (after insert, after update, after delete, after undelete) {

if(Trigger.isAfter){

if(Trigger.isInsert || Trigger.isUpdate || Trigger.isDelete || Trigger.isUndelete){

CaseTriggerHandler.updateOpenCaseCount(Trigger.new, Trigger.oldMap, Trigger.isDelete);

}

}

}

public class CaseTriggerHandler {

public static void updateOpenCaseCount(List<Case> newCases, Map<Id, Case> oldCaseMap, Boolean isDelete){

Set<Id> accountIds = new Set<Id>();

// Collect AccountIds based on operation

if(isDelete){

for(Case c : oldCaseMap.values()){

if(c.AccountId != null){

accountIds.add(c.AccountId);

}

}

} else {

for(Case c : newCases){

if(c.AccountId != null){

accountIds.add(c.AccountId);

}

}

}

if(accountIds.isEmpty()) return;

// Aggregate open cases per Account

Map<Id, Integer> caseCountMap = new Map<Id, Integer>();

List<AggregateResult> aggResults = [

SELECT AccountId acctId, COUNT(Id) cnt

FROM Case

WHERE IsClosed = false AND AccountId IN :accountIds

GROUP BY AccountId

];

for(AggregateResult ar : aggResults){

caseCountMap.put((Id)ar.get('acctId'), (Integer)ar.get('cnt'));

}

// Prepare Accounts for update

List<Account> accList = new List<Account>();

for(Id accId : accountIds){

Account acc = new Account(Id = accId);

acc.Open\_Cases\_Count\_\_c = caseCountMap.containsKey(accId) ? caseCountMap.get(accId) : 0;

accList.add(acc);

}

if(!accList.isEmpty()){

update accList;

}

}

}

@isTest

public class CaseTriggerHandlerTest {

@isTest

static void testUpdateOpenCaseCount(){

// Create Account

Account acc = new Account(Name = 'Test Account', Open\_Cases\_Count\_\_c = 0);

insert acc;

// Create Cases

Case c1 = new Case(Subject = 'Issue 1', Status = 'New', AccountId = acc.Id);

Case c2 = new Case(Subject = 'Issue 2', Status = 'Working', AccountId = acc.Id);

insert new List<Case>{c1, c2};

// Check count after insert

Account accAfterInsert = [SELECT Open\_Cases\_Count\_\_c FROM Account WHERE Id = :acc.Id];

System.assertEquals(2, accAfterInsert.Open\_Cases\_Count\_\_c, 'Should have 2 open cases');

// Close one case

c1.Status = 'Closed';

update c1;

// Check count after update

Account accAfterUpdate = [SELECT Open\_Cases\_Count\_\_c FROM Account WHERE Id = :acc.Id];

System.assertEquals(1, accAfterUpdate.Open\_Cases\_Count\_\_c, 'Should have 1 open case');

// Delete remaining open case

delete c2;

// Check count after delete

Account accAfterDelete = [SELECT Open\_Cases\_Count\_\_c FROM Account WHERE Id = :acc.Id];

System.assertEquals(0, accAfterDelete.Open\_Cases\_Count\_\_c, 'Should have 0 open cases');

}

}

## ****Use Case 4: Auto-Creation of Child Records****

**Scenario:**  
When a new **Account** is created, automatically create **two default Contacts** (Primary Contact & Support Contact).

**Recommended Solution:**

* Write an **after insert trigger** on Account.
* Insert default Contact records linked to the new Account.

trigger AccountTrigger on Account (after insert) {

if(Trigger.isAfter && Trigger.isInsert){

AccountTriggerHandler.createDefaultContacts(Trigger.new);

}

}

public class AccountTriggerHandler {

public static void createDefaultContacts(List<Account> newAccounts){

List<Contact> contactsToInsert = new List<Contact>();

for(Account acc : newAccounts){

// Primary Contact

Contact primaryCon = new Contact(

FirstName = 'Primary',

LastName = 'Contact',

Email = 'primary@' + acc.Name.replaceAll(' ', '').toLowerCase() + '.com',

AccountId = acc.Id

);

// Support Contact

Contact supportCon = new Contact(

FirstName = 'Support',

LastName = 'Contact',

Email = 'support@' + acc.Name.replaceAll(' ', '').toLowerCase() + '.com',

AccountId = acc.Id

);

contactsToInsert.add(primaryCon);

contactsToInsert.add(supportCon);

}

if(!contactsToInsert.isEmpty()){

insert contactsToInsert;

}

}

}

@isTest

public class AccountTriggerHandlerTest {

@isTest

static void testCreateDefaultContacts(){

// Create Account

Account acc = new Account(Name = 'Test Account');

Test.startTest();

insert acc;

Test.stopTest();

// Query related Contacts

List<Contact> contacts = [

SELECT FirstName, LastName, Email

FROM Contact

WHERE AccountId = :acc.Id

];

System.assertEquals(2, contacts.size(), 'Two default contacts should be created');

// Validate names

Set<String> expectedNames = new Set<String>{'Primary Contact', 'Support Contact'};

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

for(Contact con : contacts){

actualNames.add(con.FirstName + ' ' + con.LastName);

}

System.assertEquals(expectedNames, actualNames, 'Default contacts should be Primary and Support');

}

}

## ****Use Case 5: Restrict Deletion****

**Scenario:**  
Business requires that **Accounts with Open Opportunities** cannot be deleted.

**Recommended Solution:**

* Write a **before delete trigger** on Account.
* Check for related Open Opportunities.
* Use **addError()** to prevent deletion.

## ****Use Case 6: Send Notifications****

**Scenario:**  
Whenever a **High Priority Case** is created, the assigned agent should receive an **email alert**.

**Recommended Solution:**

* Write an **after insert trigger** on Case.
* Call an **Apex Class with Messaging.sendEmail()**.
* Ensure bulkification by batching email requests.

## ****Use Case 7: Scheduled Apex****

**Scenario:**  
Every night, management wants a report of **all Opportunities closing in the next 7 days** emailed to Sales Managers.

**Recommended Solution:**

* Write a **Scheduled Apex Class** (implements Schedulable).
* Query Opportunities with CloseDate in next 7 days.
* Send email summary using Messaging.SingleEmailMessage.

## ****Use Case 8: Batch Apex for Data Cleanup****

**Scenario:**  
Company has **1 million old Case records** that need to be archived by setting a custom flag.

**Recommended Solution:**

* Write a **Batch Apex Class** (Database.Batchable).
* Process records in manageable chunks.
* Optionally chain batch jobs for large volumes.

## ****Use Case 9: Future / Queueable Apex****

**Scenario:**  
When a new **Lead** is created, the system should make an **external callout** to validate email address with a third-party service.

**Recommended Solution:**

* Write a **Future Method with callout=true** OR a **Queueable Apex Class**.
* Trigger calls the async method after insert.