



Wise Quarter

first class IT courses

Team : Salesforce T118

Course : Salesforce Developer

Date : 10/18/2023

Subject: Trigger Concept
-2

+1 912 888 1630

www.wisequarter.com



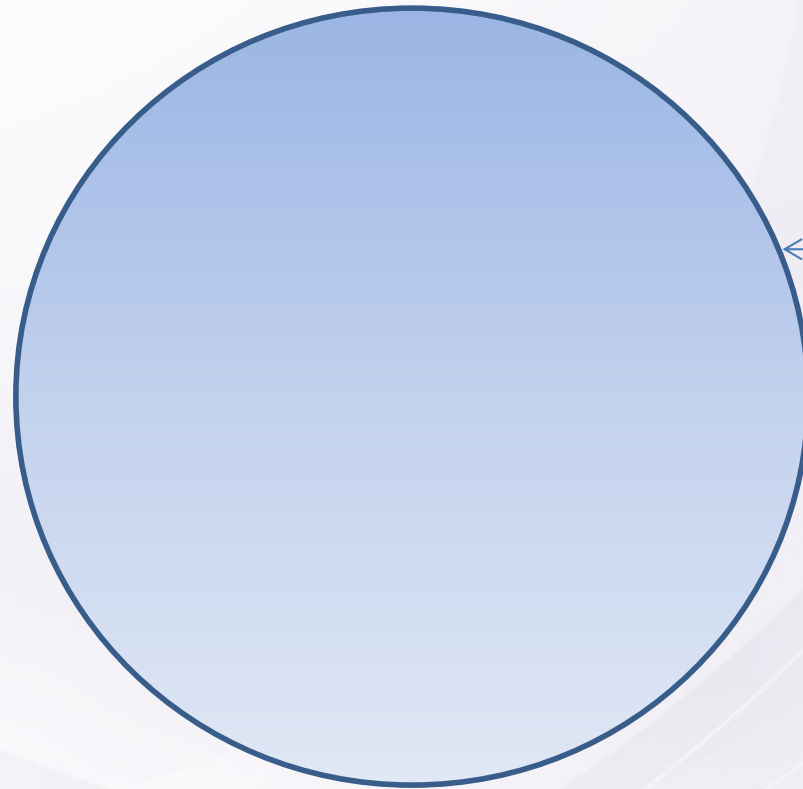
/wisequarter



The future at your fingertips

     /wisequarter

www.wisequarter.com

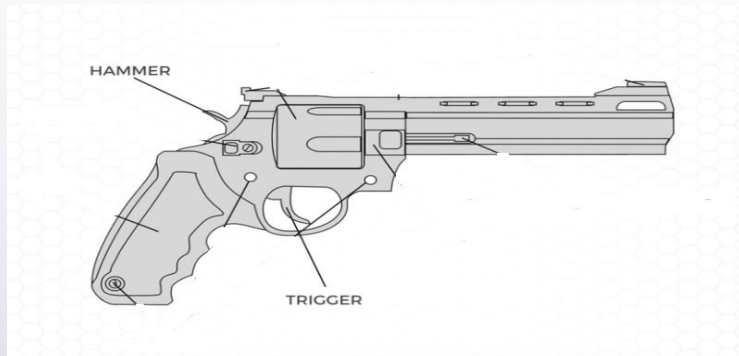


← Developer

AGENDA

- **Introduction To Apex Trigger**
- **Types of Triggers**
- **Trigger Context Variables**
- **Best Practices for Triggers**

What is Trigger?



Flow \approx Trigger

Select Object

Select the object whose records trigger the flow when they're created, updated, or deleted.

* Object



A value is required.

Configure Trigger

* Trigger the Flow When:

- ☒ A record is created
- ☐ A record is updated
- ☐ A record is created or updated
- ☐ A record is deleted

Set Entry Conditions

Specify entry conditions to reduce the number of records that trigger the flow and the number of times the flow is executed. Minimizing unnecessary flow executions helps to conserve your org's resources.

If you create a flow that's triggered when a record is updated, we recommend first defining entry conditions. Then select the **Only when a record is updated to meet the condition requirements** option for When to Run the Flow for Updated Records.

Condition Requirements

None

New Formulas for Conditions ⓘ

* Optimize the Flow for:

Fast Field Updates

Update fields on the record that triggers the flow to run. This high-performance flow runs *before* the record is saved to the database.

Actions and Related Records

Update any record and perform actions, like send an email. This more flexible flow runs *after* the record is saved to the database.

Why do we need trigger ?

- ✓ **To automate our tasks**
- ✓ **Triggers work extremely fast**
- ✓ **Triggers can be used to implement complex validations**
- ✓ **To communicate with third party systems**

Trigger Concepts

- Before && After
- Trigger.new && Trigger.old

Recap

- **DML Operations** (Data Manipulation Language)
 - **Insert, Update, Delete, Undelete**
 - insert accList
 - Database.insert (accList)
 - Database.insert(accList, true)
 - Database.insert(accList,false)
- } All or nothing
- individual

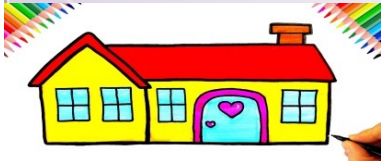
Concept 1

- Before

Your House



Parents' House



- After



BEFORE leaving my **own house**, I visited my parents' house



AFTER leaving my **own house**, I visited my parents' house



AFTER leaving my **own house**, I turned off the lights



BEFORE leaving my **own house**, I turned off the lights



- If you want to do some changes **IN your house** , you have to do **BEFORE** leaving the house.
- If you want to do something **outside of your house**
 - You can call someone befor/after you are still in your house
 - If you want to visit your parents's house You have to leave your house.

Concept 1

Account

Configure Trigger

• **Trigger the Flow When:**

☒ A record is created
☐ A record is updated
☐ A record is created or updated
☐ A record is deleted

Set Entry Conditions

Specify entry conditions to reduce the number of records that trigger the flow and the number of times the flow is executed. Minimizing unnecessary flow executions helps to conserve your org's resources.

If you create a flow that's triggered when a record is updated, we recommend first defining entry conditions. Then select the **Only when a record is updated to meet the condition requirements** option for When to Run the Flow for Updated Records.

Condition Requirements

None

• **Before**

• **Optimize the Flow for:**

Fast Field Updates

Update fields on the record that triggers the flow to run. This high-performance flow runs *before* the record is saved to the database.

• **After**

Actions and Related Records

Update any record and perform actions, like send an email. This more flexible flow runs *after* the record is saved to the database.

**Interact with database
(Lock the door)**

TYPES OF TRIGGERS



Before Action


- There are two main types of trigger;
 - **Before triggers** are used to update or validate record values **before** they're **saved** to the database.
 - **After triggers** are used to access field values that are set by the system (like record's Id etc), and to **affect** changes in **other records**.
 - The **records** that **fire the after-trigger are read-only**.



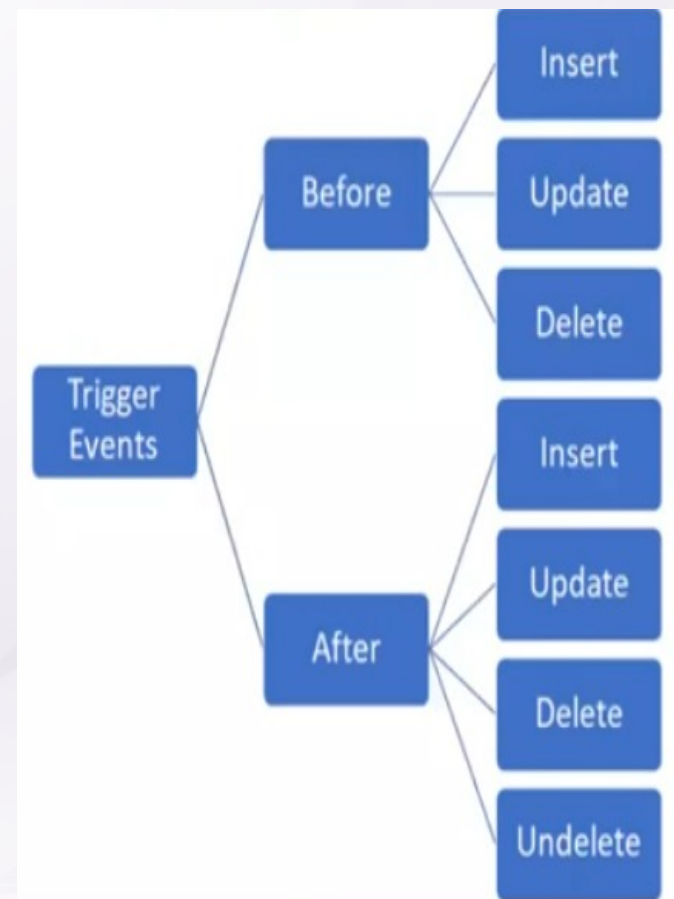
After Action

Concept 2

DML Operations (Data Manipulation Language)

- | | | |
|------------|--|------------------|
| • Insert | • Before Insert | • After Insert |
| • Update | • Before update | • After Update |
| • Delete | • Before Delete | • After Delete |
| • Undelete | • Before  ete | • After Undelete |

- upsert: fire each before and once insert or before and after update triggers as applicable
- merge : fire each before and after delete for the losing records, and both before and after update



How many records will the DML be applied to?

- Single  record
- Collections (List) • Always
 - Trigger.New • Trigger.Old
 - Trigger.newMap • Trigger.oldMap

Concept 3

- **Trigger.new:** Simply a list of the records and holds **LATEST Values** -----> List<sObject>List<Account>
- **Trigger.newMap:** A map of IDs to the new versions of the sObject records. -----> Map<Id,sObject>..... Map<Id,Lead>
- **Trigger.old:** will hold **OLD COPY** of current modifying record. -----> List<sObject>
- **Trigger.oldMap:** A map of IDs to the **OLD VERSIONS** of the sObject records. -----> Map<Id,sObject>
You can not make changes on old values (read-only)

<https://acesalesforce.com/trigger-new-vs-trigger-newmap-apex/>

Concept 3

Account / Accounts

Id : 0018Z00002f24ARQAY
Name: ABC
Rating: Hot
Phone: 345 987 99 88
.
.

Id	0018Z00002f24ARQAY
Name	ABC
Rating	Hot
Phone	345 987 99 88
.	.

Trigger.old
Trigger.oldMap

Account / Accounts

Id : 0018Z00002f24ARQAY
Name: ABC
Rating: **Warm**
Phone: **111 000 00 00**
.
.

Id	0018Z00002f24ARQAY
Name	ABC
Rating	Hot
Phone	345 987 99 88
.	.

Trigger.new
Trigger.newMap

**Required field
Validation
Flow
BEFORE Trigger
Etc.
Etc.**

**AFTER
Trigger**

DML (**Update**)

Start to **update**

Click Save

Concept 3

Account / Accounts

Id :
Name: ABC
Rating:
Phone:
.
.

Trigger.new

*Required field
Validation
Etc.
Etc.*

*BEFORE
Trigger*

*AFTER
Trigger*

Start to **create**

Click Save

DML (**Insert**)

Concept 3

Account / Accounts

Id : 0018Z00002f24ARQAY
Name: ABC
Rating: Hot
Phone: 345 987 99 88
.
.

Trigger.old
Trigger.oldMap



Required field
Validation
Trigger
Etc.
Etc.

DML (**Delete**)

Click **Delete**

Concept 3

lead

Id : 0018Z000002f24ARQAY
Name: zzzz
.
.

*Before
Trigger*

*After
Trigger*

Click **UnDelete**

DML (**unDelete**)

Concept 3

	BEFORE				AFTER			
	insert	update	delete	undelete	insert	update	delete	undelete
Trigger.New								
Trigger.newMap								
Trigger.Old								
Trigger.oldMap								

Returns a list of the new versions of the sObject records.

`new`

This sObject list is only available in `insert`, `update`, and `undelete` triggers, and the records can only be modified in `before` triggers.

A map of IDs to the new versions of the sObject records.

`newMap`

This map is only available in `before update`, `after insert`, `after update`, and `after undelete` triggers.

Returns a list of the old versions of the sObject records.

`old`

This sObject list is only available in `update` and `delete` triggers.

A map of IDs to the old versions of the sObject records.

`oldMap`

This map is only available in `update` and `delete` triggers.

TRIGGER CONTEXT VARIABLES

Boolean returning variables;

- ✓ `Trigger.isInsert`
(if trigger is insert trigger this will return **true**)
- ✓ `Trigger.isUpdate`
- ✓ `Trigger.isDelete`
- ✓ `Trigger.isUndelete`
- ✓ `Trigger.isBefore`
- ✓ `Trigger.isAfter`
- ✓ `Trigger.isExecuting`
(If the code is running in a trigger this will return **TRUE**)

Collection <List> returning variables;

- ✓ `Trigger.new`
- ✓ `Trigger.newMap`
- ✓ `Trigger.old`
- ✓ `Trigger.oldMap`

ENUM returning variable; (switch statement)

- ✓ `Trigger.operationType`

Integer returning variable;

- ✓ `Trigger.size`
(size of list)

https://developer.salesforce.com/docs/atlas.en-us.apexcode.meta/apexcode/apex_triggers_context_variables.htm

TRIGGER.OPERATIONTYPE (ENUM)

Boolean returning variables;

✓ Trigger.isBefore

✓ Trigger.isAfter

✓ Trigger.isInsert
(if trigger is insert trigger this will return **true**)

✓ Trigger.isUpdate

✓ Trigger.isDelete

✓ Trigger.isUndelete

✓ Trigger.isExecuting
(If the code is running in a trigger this will return **TRUE**)

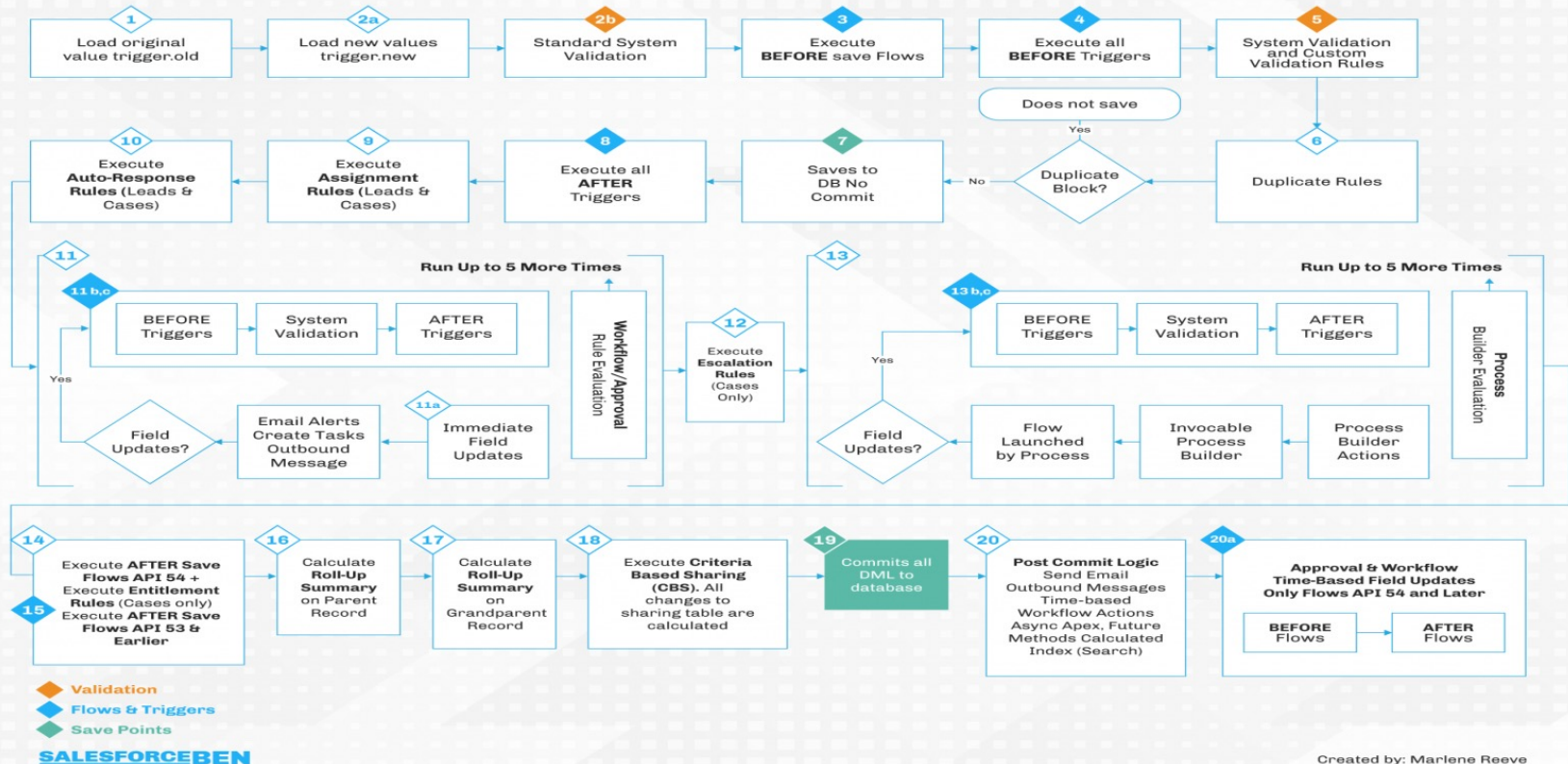
ENUM returning variable; (switch statement)

✓ Trigger.operationType

- BEFORE_INSERT
- BEFORE_UPDATE
- BEFORE_DELETE
- AFTER_INSERT
- AFTER_UPDATE
- AFTER_DELETE
- AFTER_UNDELETE

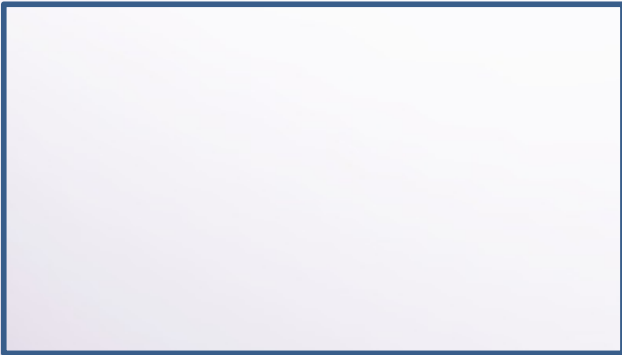
SALESFORCE ORDER OF EXECUTION

Spring '22 - API v54



- <https://www.salesforceben.com/learn-salesforce-order-of-execution/>

EGG FARM

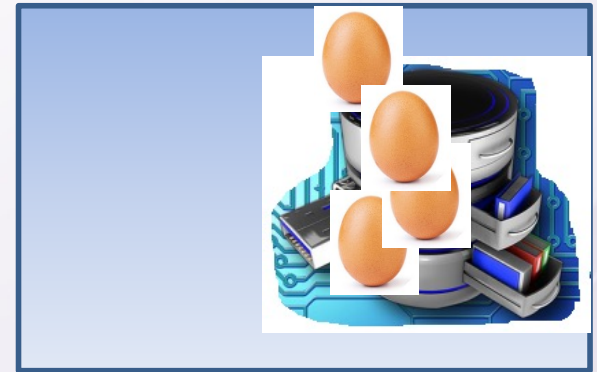


SOMETHING DONE HERE

TRANSPORT



FACTORY



FIRE TRIGGER
AND
DO SOMETHING

Let's see some code

Steps for Trigger Creation

* Before starting please check all validations, flows etc.

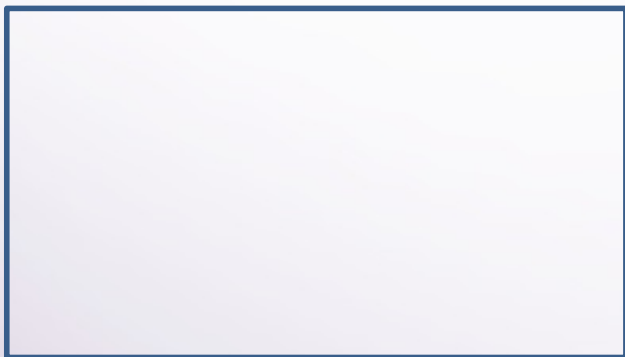
Task

If an account is created

No matter what The rating of account should be 'Hot'
and BillingState should be 'ARKANSAS'

- Determine on which object trigger will be set (Account , Opportunity, Student__c)
- Determine what type of trigger is needed (Before or After)
- Determine what kind of DML operations to use (insert, update, delete, undelete)

EGG FARM



Create a new Account

The rating of account should be 'Hot'
and BillingState should be 'ARKANSAS'

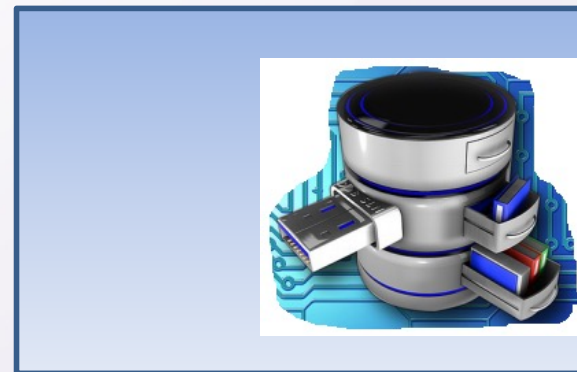
```
Account acc = new Account(Name='aaaaaaa');  
database.insert(acc);
```

TRANSPORT



Trigger.new /old

SALESFORCE CLOUD DATABASE

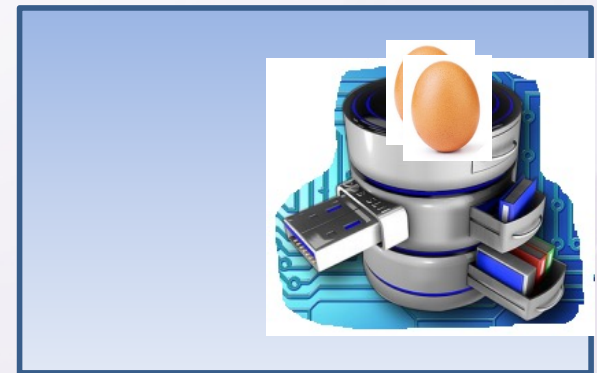


```
for ( Account acc: Trigger.new){  
    acc.Rating = 'Hot';  
    acc. BillingState = 'ARKANSAS';  
    System.debug(Trigger.new);  
    System.debug(Trigger.operationType);  
}
```

EGG FARM

TRANSPORT

SALESFORCE CLOUD DATABASE



Create Two new Accounts on Anonymous window
The rating of account should be 'Hot' and BillingState should be 'ARKANSAS'

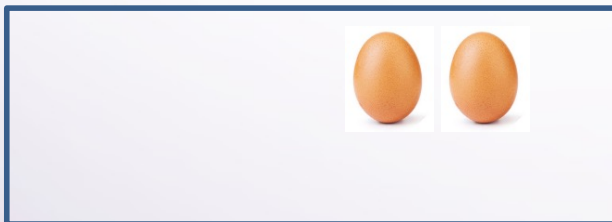
```
Account ac1 = new Account(Name='ac1');  
Account ac2 = new Account(Name='ac2');
```

```
database.insert(ac1 );  
database.insert(ac2 );
```

```
for ( Account acc: trigger.new ){
```

```
    acc.Rating = 'Hot';  
    acc.BillingState = 'ARKANSAS';  
    System.debug(trigger.new);
```

EGG FARM



TRANSPORT

Trigger.new /old



SALESFORCE CLOUD DATABASE



Create Two new Accounts on Anonymous window
The rating of account should be 'Hot' and BillingState should be 'ARKANSAS'

```
List<Account> accountList = new List<Account>();
```

```
Account ac1 = new Account(Name='ac1');
```

```
Account ac2 = new Account(Name='ac2');
```

```
accountList.add(ac1);
```

```
accountList.add(ac2);
```

```
database.insert(accountList);
```

```
for ( Account acc: trigger.new ){
```

```
    acc.Rating = 'Hot';
```

```
    acc.BillingState = 'ARKANSAS';
```

```
    System.debug(Trigger.new);
```

```
}
```

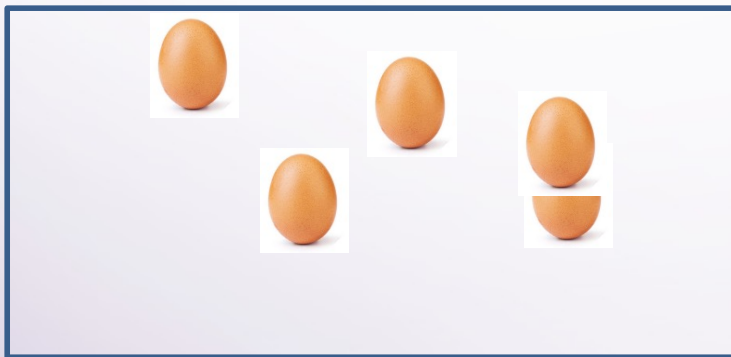
BEST PRACTICES FOR TRIGGERS PART 1

- **Bulkify your Code:** Bulkifying Apex code refers to the concept of making sure the code properly handles more than one record at a time.
- **Governor Limits (do not use DML/ SOQL inside the loop)**
 - DML operations ---- 150
 - SOQL----- Synchronous-100 / Asynchronous-200
 - https://developer.salesforce.com/docs/atlas.en-us.salesforce_app_limits_cheatsheet.meta/salesforce_app_limits_cheatsheet/salesforce_app_limits_platform_apexgov.htm

Please create 200 Accounts and update the Ratings to Hot

- How to deactivate a trigger?
- Object manager -> select object → (at the bottom) trigger

EGG FARM (Anonymous Window)



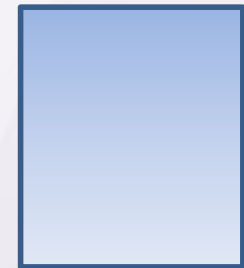
```
for ( integer i=0; i<200 ; i++ ){  
    Account newAcc = new Account();  
    Account(Name= 'WiseQuarter ' + i );  
    insert newAcc;  
}  
List<Account> acList= [SELECT id,name FROM Account];
```

TRANSPORT

Trigger.new /old

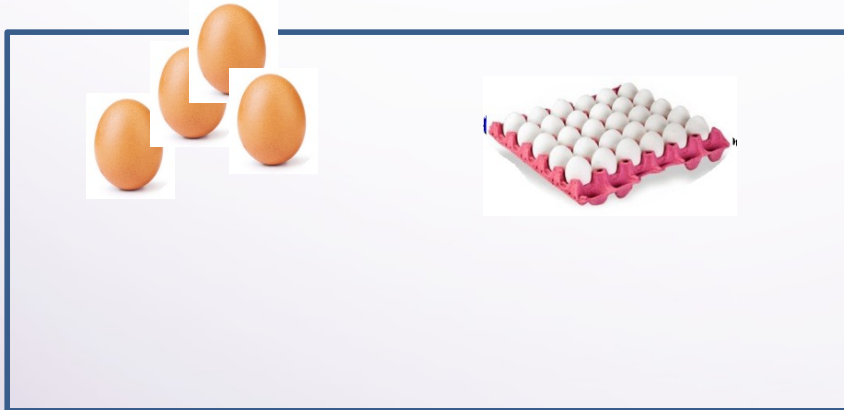


DATABASE



```
for ( Account acc: Trigger.new  
) {  
    acc.Rating = 'Hot';  
}
```

Anonymous Window

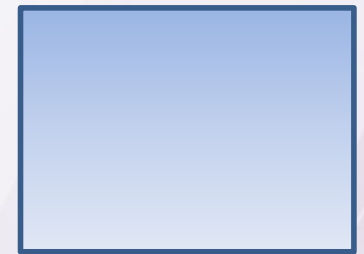


TRANSPORT

Trigger.new /old



DATABASE



```
List<Account> accList = new List<Account>();
for ( integer i=0; i<200 ; i++ ){
    Account newAcc = new Name='WiseQuarter--'+i);
    accList .add(newAcc);
}
insert accList;
```

```
for ( Account acc: Trigger.new ){
    acc.Rating = 'Hot';
}
```

Anonymous Window

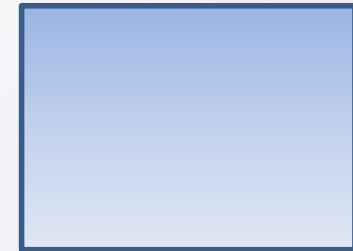


TRANSPORT

Trigger.new /old



DATABASE



```
List<Account> accList = new List<Account>();
for ( integer i=0; i<200 ; i++ ){
    Account newAcc = new Account( Name='WiseQuarter--'+i);
    accList .add(newAcc);
}
insert accList;
```

To Delete newly created Accounts :

```
List<Account> accList = [Select id FROM Account WHERE name LIKE 'Wise%'];
Database.delete(accList);
```

```
List<Account> updatedAccounts = new List< Account >();
```

```
for(Account acc:Trigger.new){
    acc.Rating = 'Hot';
    updatedAccounts.add(acc);
}
```

```
Database.insert(updatedAccounts);
```

After Insert

- After creating an Account
- Assign a contact to account like 'Contact of XXX'

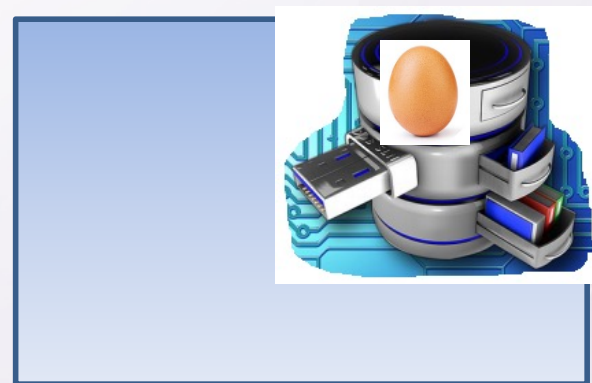
EGG FARM



TRANSPORT



SALESFORCE CLOUD DATABASE



```
Account newAcc = new Account(Name='WiseQuarter');  
insert newAcc;
```

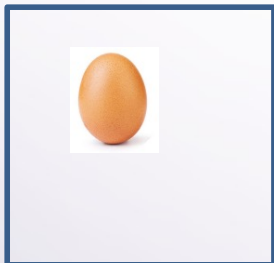
```
for (Account acc : Trigger.new){
```

```
    Contact newCnt =new Contact(LastName='Contact Of '+acc.Name);
```

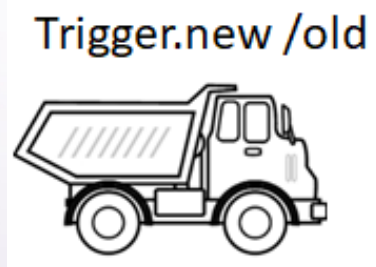
```
    Database.insert newCnt;
```

```
}
```

Anonymous Window



TRANSPORT



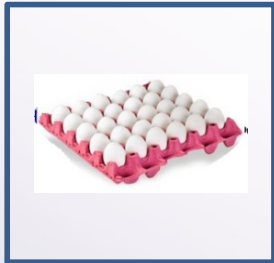
SALESFORCE CLOUD DATABASE



DONE

```
for(Account acc:Trigger.new){  
  
    Contact newCnt =new Contact(LastName='Contact Of '+acc.Name, AccountId=acc.Id);  
  
    insert newCnt;  
  
}
```

Anonymous Window



TRANSPORT

Trigger.new /old



DATABASE



DONE

```
List<Contact> relatedContactList=new List<Contact>();  
  
for(Account acc:Trigger.new){  
    Contact newcnt=new Contact(LastName='Contact Of '+acc.Name, AccountId=acc.Id);  
    relatedContactList.add(newcnt);  
}  
  
Database.insert(relatedContactList);
```


Before Insert and Before Update

- Before creating an Account if the industry filed is null assign 'Banking' to Industry field
AND
- Before updating an Account assign 'Education' to Industry field

```
trigger AccountTrigger on Account (before insert , before update) {  
  
    List<Account> updatedAccounts = new List< Account >();  
  
    for(Account acc:Trigger.new){  
  
        if( String.isBlank(acc.Industry) ){  
            // acc.Industry == null  
            // If field data type is number    acc.AnnualRevenue==null  
            acc.Industry = ' Banking';  
            updatedAccounts.add(acc);  
        }  
  
    }  
  
}
```

BEST PRACTICES FOR TRIGGERS PART 2

- **One Trigger Per Object** : A single Apex Trigger is all you need for one object. If you develop multiple Triggers for a single object, Salesforce doesn't guaranty execution of triggers.

```
trigger XXXXXXTrigger on XXXXXX (before insert , before update , before delete , after insert , after update, after delete, after undelete) { }
```

Option 1

```
SWITCH on Trigger.operationType {  
  
    WHEN BEFORE_INSERT {}  
    WHEN BEFORE_UPDATE {}  
    WHEN BEFORE_DELETE {}  
    WHEN AFTER_INSERT {}  
    WHEN AFTER_UPDATE {}  
    WHEN AFTER_DELETE {}  
    WHEN AFTER_UNDELETE {}  
  
}
```

Option 2

```
if ( Trigger.IsBefore && Trigger.isInsert) { }  
if ( Trigger.IsBefore && Trigger.isUpdate) { }  
if ( Trigger.IsBefore && Trigger.isDelete) { }  
if ( Trigger.IsAfter && Trigger.isInsert) { }  
if ( Trigger.IsAfter && Trigger.isUpdate) { }  
if ( Trigger.IsAfter && Trigger.isDelete) { }  
if ( Trigger.IsAfter && Trigger.isUndelete) { }
```

Option 3

```
if ( Trigger.IsBefore ) {  
    if ( Trigger.isInsert) { }  
    if ( Trigger.isUpdate) { }  
    if ( Trigger.isDelete) { }  
}  
  
if ( Trigger.IsAfter ) {  
    if ( Trigger.isInsert) { }  
    if ( Trigger.isUpdate) { }  
    if ( Trigger.isDelete) { }  
    if ( Trigger.isUndelete) { }  
}  
  
}
```

Lead Trigger (combined)

- When a new Lead is created assign 'Education' if Industry field is null
- if Lead is updated, assign 'Banking' if Industry field is null

```
if (Trigger.isBefore && Trigger.isInsert) {  
    for( Lead lead:Trigger.new){  
        if( String.isBlank(lead.Industry )){  
            lead.Industry = 'Education';  
        }  
    }  
}
```

```
if (Trigger.isBefore && Trigger.isUpdate) {  
    for( Lead lead:Trigger.new){  
        if( String.isBlank(lead.Industry )){  
            lead.Industry = 'Education';  
        }  
    }  
}
```

BEST PRACTICES FOR TRIGGERS PART 3

- **Trigger handler Class :** You should put trigger logic in handler classes and call that class and method from trigger. By doing so,
 - ✓ You can keep trigger simple and readable
 - ✓ You can expose logic to be re-used anywhere else in your org.

Handler Class

- Generate an Apex class and name as XXXTriggerHandler
- Generate a function
- Provide a list to function as argument/parameter instead of Trigger.new
- Call this method from Trigger

```
trigger LeadTrigger on Lead (xxxxxxxxxxx) {  
  
    if ( Trigger.IsBefore && Trigger.isInsert) {  
        LeadTriggerHandler. industryFieldCheck(Trigger.new);  
    }  
    if ( Trigger.IsBefore && Trigger.isUpdate) {  
        LeadTriggerHandler. industryFieldCheck(Trigger.new);  
    }  
    if ( Trigger.IsBefore && Trigger.isDelete) { }  
    if ( Trigger.IsAfter && Trigger.isInsert) { }  
    if ( Trigger.IsAfter && Trigger.isUpdate) { }  
    if ( Trigger.IsAfter && Trigger.isDelete) { }  
    if ( Trigger.IsAfter && Trigger.isUndelete) { }  
  
}
```

```
public class LeadTriggerHandler {  
  
    public static void industryFieldCheck( List<Lead> leadList ){  
        for(Lead lead:leadList){  
            if( String.isBlank(lead.Industry )){  
                lead.Industry = 'Education';  
            }  
        }  
    }  
}
```

Lead Trigger (Validation)

- If a new Lead is created or updated and if 'Industry' filed is null; Give a warning

```
for ( Lead Id: Trigger.new){  
  
    if(String.isBlank(Id.Industry)){  
        Id.addError('Sisst noluyo industry bos kardess!!!');  
    }  
}
```

- When ever an account is restored, state restore information like restore time and who restored the record on the description field of RELATED CONTACT

Lead Trigger (After undelete)

- When a lead record is Restored from a recycle bin, type 'Restored' in the console

```
if(Trigger.isUndelete){  
    for (Lead lead:Trigger.new){  
  
        system.debug('Restored --> ' + lead.FirstName);  
  
    }  
  
}
```

Lead Trigger

- Create a field on Lead named Recycled and update the field as 'restored' when a lead record is recycled
- WHY ?

Account Trigger

- When ever an account is restored, state restore information like restore time and who restored the record on the description field of RELATED CONTACT

Before undelete ???

After Undelete

When AFTER_UNDELETE{

```
List<Contact> cntList = [SELECT Id, AccountId, Description FROM Contact WHERE AccountId IN:Trigger.newMap.keySet();
```

```
For(Contact cnt : cntList ) {
```

```
  Cnt.Description = 'Account restored by ' + System.UserInfo.getName() + ', on ' + system.today();
```

```
}
```

```
Database.update(cntList );
```

```
}
```

- Generate a trigger that After creating a new lead, Creates another lead (FirstName = 'AAA new Lead Created')

Account (Before Delete)

- Before deleting an account write 'xxxx company is deleted' on the console

```
if(Trigger.isBefore && Trigger.isDelete){  
    for(Account acc:Trigger.new){  
  
        system.debug( acc.Name + ' is Deleted');  
  
    }  
  
}
```

- WHY ? old

* Lead Trigger *

- ```
If(Trigger.isAfter && Trigger.isUpdate) {
 For(Lead lead:Trigger.new) {
 lead.Industry = 'Education';
 }
}
```

Note. Why not working ? Because read Only

# Lead Trigger (validation)

- Create an error (Warning) trigger when a Lead is updated from 'Open - Not Contacted' status to 'Closed - Converted' or 'Closed - Not Converted' status. (Validation )

```
if (Trigger.IsBefore && Trigger.isUpdate) {

 LeadTriggerHandler.statusFieldCheck(Trigger.new , Trigger.oldMap);

}

Public static void statusFieldCheck (List<Lead> leadList , Map<Id, Lead> leadOldMap) {

 for(Lead lead: leadList){

 Lead oldLead = leadOldMap.get(lead.Id);

 if(oldLead.Status == 'Open - Not Contacted' && (lead.Status=='Closed - Converted' || lead.Status=='Closed - Not Converted')){

 lead.Status.addError('No No No! You can not do this.');
 }

 }

}
```



# Lead Trigger (Combined)

- While creating a Lead assign 'Other' if Lead Source is null. (Before insert)
- After creating Lead record , create a Task. (After insert)

```
if (Trigger.IsBefore && Trigger.IsInsert) {
 for (Lead lead:Trigger.new){
 if(String.isBlank(lead.LeadSource)){
 lead.LeadSource ='Other';
 }
 }
}
```

## Anonymous window

```
List<Lead> leadList = new List<Lead>();
for (Integer i =0 ; i<200 ; i++){
 Lead l = new Lead(LastName='Test ' + i , Company ='Wise');
 leadList.add(l);
}
```

```
Database.insert(leadList);
```

```
if (Trigger.IsAfter && Trigger.IsInsert) {

 List<Task> taskList = new List<Task>();

 for(Lead lead:trigger.new){
 // Task newTask = new Task(Subject = 'new lead Created, Status = 'Not Started', Whold = lead.Id , OwnerId=lead.OwnerId);
 Task newTask = new Task();

 newTask.Subject = 'new Lead Created';
 newTask.Status = 'Not Started';
 newTask.Whold = lead.Id;
 newTask.OwnerId=lead.OwnerId;

 taskList.add(newTask);
 }

 Database.insert(taskList);
}
```

# Lead Trigger (validation)

- Generate a trigger that
- After creating a new lead,
- Creates another lead ( FirstName = 'new Lead Created' )

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

```
 Lead secondLead = new Lead();
 secondLead.FirstName='new Lead Created' ;
 secondLead.Company = 'Wise';
 secondLead.Industry='Education';
```

```
 Database.insert(secondLead);
}
```

## Why?

LeadTrigger: execution of AfterInsert caused by: System.DmlException: Insert failed. First exception on row 0; first error: CANNOT\_INSERT\_UPDATE\_ACTIVATE\_ENTITY, LeadTrigger: maximum trigger depth exceeded Lead trigger event AfterInsert Lead trigger event AfterInsert Lead trigger event AfterInsert Lead trigger event AfterInsert Lead trigger event AfterInsert Lead trigger event AfterInsert Lead trigger event AfterInsert Lead trigger event AfterInsert Lead trigger event AfterInsert Lead trigger event AfterInsert Lead trigger event AfterInsert Lead trigger event AfterInsert Lead trigger event AfterInsert: [] Trigger.LeadTrigger: line 49, column 1

## BEST PRACTICES FOR TRIGGERS PART 4

- **Avoid Trigger Recursion :** You can avoid recursion of a trigger by calling static boolean variable from another class.

# Lead Trigger (validation)

- Generate a trigger that
- After creating a new lead,
- Creates another lead ( FirstName = 'AAA new Lead Created' )

\* Handler class

```
public static void afterInsertHandler(List<Lead> leadList){

 Lead secondLead = new Lead();
 secondLead.FirstName='AAA new Lead Created' ;
 secondLead.Company = 'Wise';
 // secondLead.Industry='Education';
 secondLead.Status='Open - Not Contacted';

 Database.insert(secondLead);

}
```

Why this does not work ?

# Lead Trigger (validation)

- Generate a trigger that
- After creating a new lead,
- Creates another lead ( FirstName = 'new Lead Created' )

\* Handler class

```
public static void afterInsertHandler(List<Lead> leadList){
```

```
 Lead secondLead = new Lead();
 secondLead.FirstName='AAA new Lead Created' ;
 secondLead.Company = 'Wise';
 // secondLead.Industry='Education';
 secondLead.Status='Open - Not Contacted';
```

```
 Database.insert(secondLead);
```

}  
**Why this does not work ?**

\*Handler Class

**public static Boolean recursion= true;**

```
public static void afterInsertHandler(List<Lead> leadList){
```

```
 if(recursion) {
```

**recursion= false;**

```
 Lead secondLead = new Lead();
 secondLead.lastName='AAA new Lead Created' ;
 secondLead.Company = 'Wise';
 // secondLead.Industry='Education';
 secondLead.Status='Open - Not Contacted';
```

```
 Database.insert(secondLead);
```

```
 }
```

## BEST PRACTICES FOR TRIGGERS TOTAL

- **Bulkify your Code:** Bulkifying Apex code refers to the concept of making sure the code properly handles more than one record at a time.
  - Governor Limits ( do not use DML/ SOQL inside the loop)
    - DML operations ---- 150
    - SOQL----- Synchronous-100 / Asynchronous-200
    - [https://developer.salesforce.com/docs/atlas.en-us.salesforce\\_app\\_limits\\_cheatsheet.meta/salesforce\\_app\\_limits\\_cheatsheet/salesforce\\_app\\_limits\\_platform\\_apexgov.htm](https://developer.salesforce.com/docs/atlas.en-us.salesforce_app_limits_cheatsheet.meta/salesforce_app_limits_cheatsheet/salesforce_app_limits_platform_apexgov.htm)
- **One Trigger Per Object :** A single Apex Trigger is all you need for one object. If you develop multiple Triggers for a single object, Salesforce doesn't guaranty execution of triggers.
- **Trigger handler Class :** You should put trigger logic in handler classes and call that class and method from trigger. By doing so,
  - ✓ You can keep trigger simple and readable
  - ✓ You can expose logic to be re-used anywhere else in your org.
- **Avoid Trigger Recursion :** You can avoid recursion of a trigger by calling static boolean variable from another class.

# TASK 1

- Populate Opportunity description (‘Test purpose Opportunity ‘ ) when user creates Opportunity
- While creating a Lead assign ‘Other’ if Lead Source is null. (Before insert)

## Before Insert

```
for (Opportunity opp:Trigger.new){
 opp.Description = 'Test purpose Opportunity' ;
}
```



# Lead Trigger (validation)

- Create an error (Warning) trigger when a Lead is updated from 'Open - Not Contacted' status to 'Closed - Converted' or 'Closed - Not Converted' status. (Validation )

```
if (Trigger.IsBefore && Trigger.isUpdate) {

 LeadTriggerHandler.statusFieldCheck(Trigger.new , Trigger.oldMap);

}

Public static void statusFieldCheck (List<Lead> leadList , Map<Id, Lead> leadOldMap) {

 for(Lead lead: leadList){

 Lead oldLead = leadOldMap.get(lead.Id);

 if(oldLead.Status == 'Open - Not Contacted' && (lead.Status=='Closed - Converted' || lead.Status=='Closed - Not Converted')){

 lead.Status.addError('No No No! You can not do this.');
 }

 }

}
```

# Lead Trigger (Combined)

- While creating a Lead assign 'Other' if Lead Source is null. (Before insert)
- After creating Lead record , create a Task. (After insert)

```
if (Trigger.IsBefore && Trigger.IsInsert) {
 for (Lead lead:Trigger.new){
 if(String.isBlank(lead.LeadSource)){
 lead.LeadSource ='Other';
 }
 }
}
```

## Anonymous window

```
List<Lead> leadList = new List<Lead>();
for (Integer i =0 ; i<200 ; i++){
 Lead l = new Lead(LastName='Test ' + i , Company ='Wise');
 leadList.add(l);
}
```

```
Database.insert(leadList);
```

```
if (Trigger.IsAfter && Trigger.IsInsert) {

 List<Task> taskList = new List<Task>();

 for(Lead lead:trigger.new){
 // Task newTask = new Task(Subject = 'new lead Created, Status = 'Not Started', Whold = lead.Id , OwnerId=lead.OwnerId);
 Task newTask = new Task();

 newTask.Subject = 'new Lead Created';
 newTask.Status = 'Not Started';
 newTask.Whold = lead.Id;
 newTask.OwnerId=lead.OwnerId;

 taskList.add(newTask);
 }

 Database.insert(taskList);
}
```

# Homework 1

- When the Opportunity stage is updated to "Negotiation/Review" Develop a trigger that creates a TASK for the owner

## After Update

```
List<Task> taskList = new List<task>();
```

```
for (Opportunity opp:Trigger.new){
```

```
 If(opp.stageName == 'Negotiation/Review' && Trigger.oldMap.get(opp.Id). stageName != 'Negotiation/Review') {
```

```
 Task taskForOwner = new Task();
```

```
 taskForOwner.Subject = 'Stage is updated';
```

```
 taskForOwner.OwnerId = opp.OwnerId;
```

```
 taskForOwner.WhatId= opp.Id;
```

```
 taskList.add(taskForOwner);
```

```
 }
```

```
}
```

```
Database.Insert(taskForOwner);
```

# Homework 2

- Prevent deleting of an Opportunity if Opportunity is 'Closed Won'

## Before Delete

```
for (Opportunity opp:Trigger.old){
 if(opp.stageName == 'Closed Won') {
 opp.addError(' You can not delete won deals');
 }
}
```

# Concept 3

Account / Accounts

Id : 0018Z00002f24ARQAY  
Name: ABC  
Rating: Hot  
Phone: 345 987 99 88  
.  
.

***Trigger.old***  
***Trigger.oldMap***

|                                                                                          |                                                                                          |
|------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------|
| Id :<br>0018Z00002f24ARQAY<br>Name: ABC<br>Rating: Hot<br>Phone: 345 987 99 88<br>.<br>. | Id :<br>0018Z00002f24ARQAY<br>Name: ABC<br>Rating: Hot<br>Phone: 345 987 99 88<br>.<br>. |
|------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------|

***Required field***  
***Validation***  
***Trigger***  
***Etc.***  
***Etc.***

Click Delete

DML ( Update , Delete)

# Homework 3

- Populate Opportunity description (‘Test purpose Opportunity ‘ ) when user creates Opportunity

Before Insert

```
for (Opportunity opp:Trigger.new){
 opp.Description = 'Test purpose Opportunity' ;
}
```

# Homework 4

- Whenever Opportunity is updated show name of user who updates opportunity on description of Opportunity

Before Update

```
for (Opportunity opp:Trigger.new){
 opp.Description = 'This opp is updated by ' + System.userInfo.getFirstName() + ' ' + System.userInfo.getLastName() ;
}
```



# Homework 5

- Create a field named 'Old Phone' on account object
- Whenever phone field is updated on Account record
- Then Old Phone field should also get updated with old version of phone number on account

## Before Update

```
for (Account acc :Trigger.new){

 Account oldver = Trigger.oldMap.get(acc.Id);
 acc. Old_Phone__c = oldVer.Phone ;

}
```

# Homework 6

- When the account's billing city is updated, assign the billing City field of Account to the mailing city field in the contacts of that account.

## After Update

```
// List<Id> accIds = new List<Id>();
Set<Id> accIds = new Set<Id>();

for (Account acc :Trigger.new){
 if (acc.BillingCity != Trigger.oldMap.get(acc.Id).BillingCity){
 accIds.add(acc.Id);
 }
}

List<Contact> conList = [SELECT id, MailingCity , Account.BillingCity FROM Contact WHERE AccountId IN :accIds];

For(Contact con:conList){
 con.MailingCity = con.Account.BillingCity;
}
Database.update(conList);
```

# Homework 7 ( interview)

- Create a number of contacts field on the account and develop a trigger that updates this field on the account every time a new contact is created

## After Insert

```
// List<Id> accIds = new List<Id>();
Set<Id> accIds = new Set<Id>();

For(Contact con: Trigger.new){
 if(!String.isBlank(con.AccountId)) { // con.AccountId != null

 accIds.add(con.AccountId);

 }
}
If(accIds.size() >0) { // if accIds size is zero this may cause error
 List<Account> accList = [SELECT id, (SELECT Id FROM Contacts) FROM Account WHERE AccountId IN :accIds]; //parent to child query

 for (Account acc :accList){
 acc.Number_of_Contacts__c = acc.Contacts.size();

 }
 Database.update(accList);
}
```

# Homework 8 ( interview)

- In this Trigger scenario we are going to create a field called “Sales Repr” with data type (Text) on the account Object.
- When we create the Account record, the Account Owner will be automatically added to Sales Rep field. When we update the Account owner of the record, then also the Sales Rep will be automatically updated.

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

```
 Set<Id>setAccOwner=new Set<Id>();
```

```
 for(Account Acc: trigger.new) {
 setAccOwner.add(Acc.OwnerId);
 }
```

```
 Map<Id,User>User_map = new Map<Id,User>([select Name from User where id in:setAccOwner]);
```

```
 User_map = [select Name from User where id in:setAccOwner];
```

```
 for(Account Acc: Trigger.new) {
```

```
 User usr=User_map.get(Acc.OwnerId);
```

```
 Acc.Sales_Rep__c=usr.Name;
```

```
 }
```

```
}
```

## Homework 9 ( Sending email)

- You need to create an apex trigger on contact object which send email when contact will inserted.

```
// Send a Email to each new Contact
trigger ContactEmailTrigger on contact (before insert) {
//get all email in the list
List<Messaging.SingleEmailMessage> mails = new List<Messaging.SingleEmailMessage>();
for (Contact myContact : Trigger.new) {
if (myContact.Email != null && myContact.FirstName != null) {
// Step 1: Create a new Email
Messaging.SingleEmailMessage mail = new Messaging.SingleEmailMessage();
// Step 2: Set list of people who should get the email
List<String> sendTo = new List<String>();
sendTo.add(myContact.Email);
mail.setToAddresses(sendTo);
// Step 3. Set email contents - you can use variables!
mail.setSubject('Your contact detail are added'); //Subject of the mail and the body of the mail
String body = 'Dear ' + myContact.FirstName + ', ';
body += 'According to the ContactEmailTrigger trigger';
body += 'your contact details were added successful';
body += 'For more details you can visit ';
body += ' https://salesforceforfresher.wordpress.com/ ';
mail.setHtmlBody(body);
// Step 4. Add your email to the master list
mails.add(mail);
}}
```

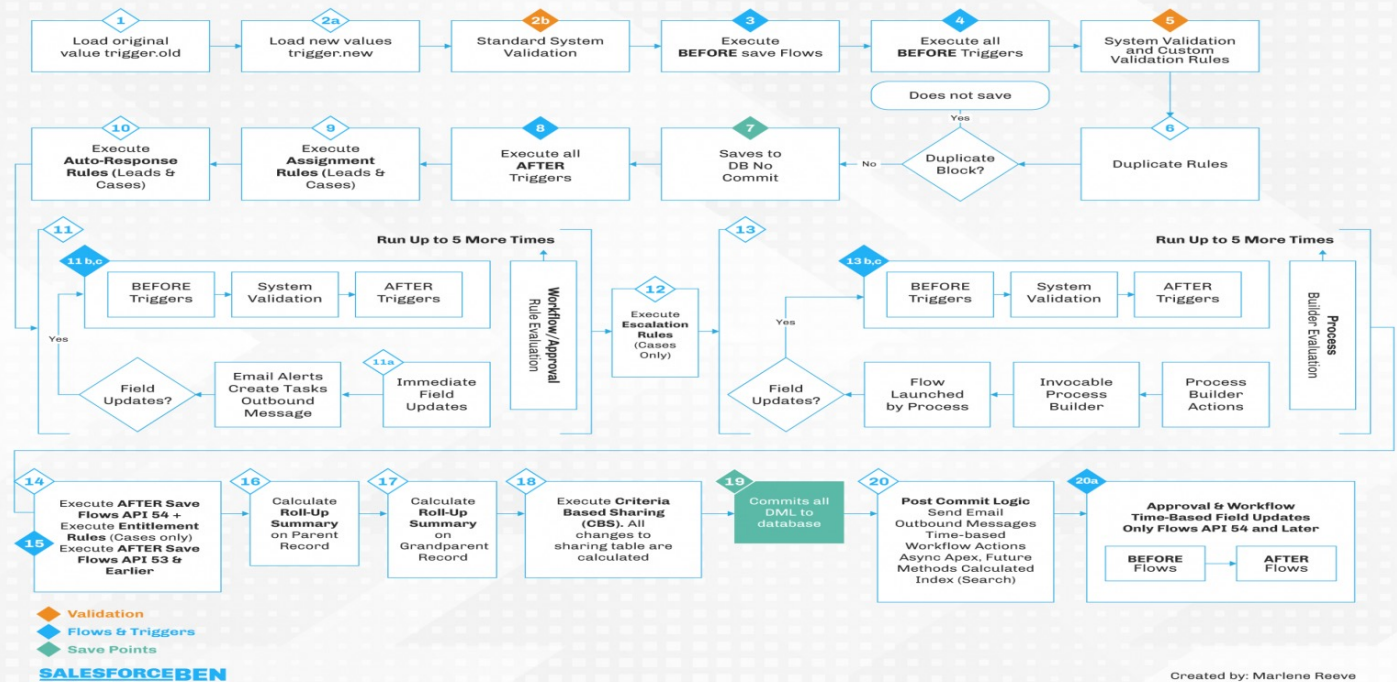
# Sample Triggers

- <https://salesforceforfresher.wordpress.com/tag/salesforce-trigger-exercises/>
- <https://salesforceforfresher.wordpress.com/2021/07/08/top-5-most-used-apex-trigger-scenarios-examples/>
- <https://salesforceforfresher.wordpress.com/2021/07/08/sending-email-from-apex-trigger/>
- <https://www.salesforcetutorial.com/trigger-scenarios-in-salesforce/>
- <https://mindmajix.com/trigger-scenarios-in-salesforce>
- <https://tekslate.com/15-sample-triggers-different-scenarios>



## SALESFORCE ORDER OF EXECUTION

Spring '22 - API v54



- <https://www.salesforceben.com/learn-salesforce-order-of-execution/>





## Order of Execution Overview

When you save a record with an insert, update, or upsert statement, Salesforce performs the following events in order. Before Salesforce executes these events on the server, the browser runs JavaScript validation if the record contains any dependent picklist fields. The validation limits each dependent picklist field to its available values. No other validation occurs on the client side.

VERSION:

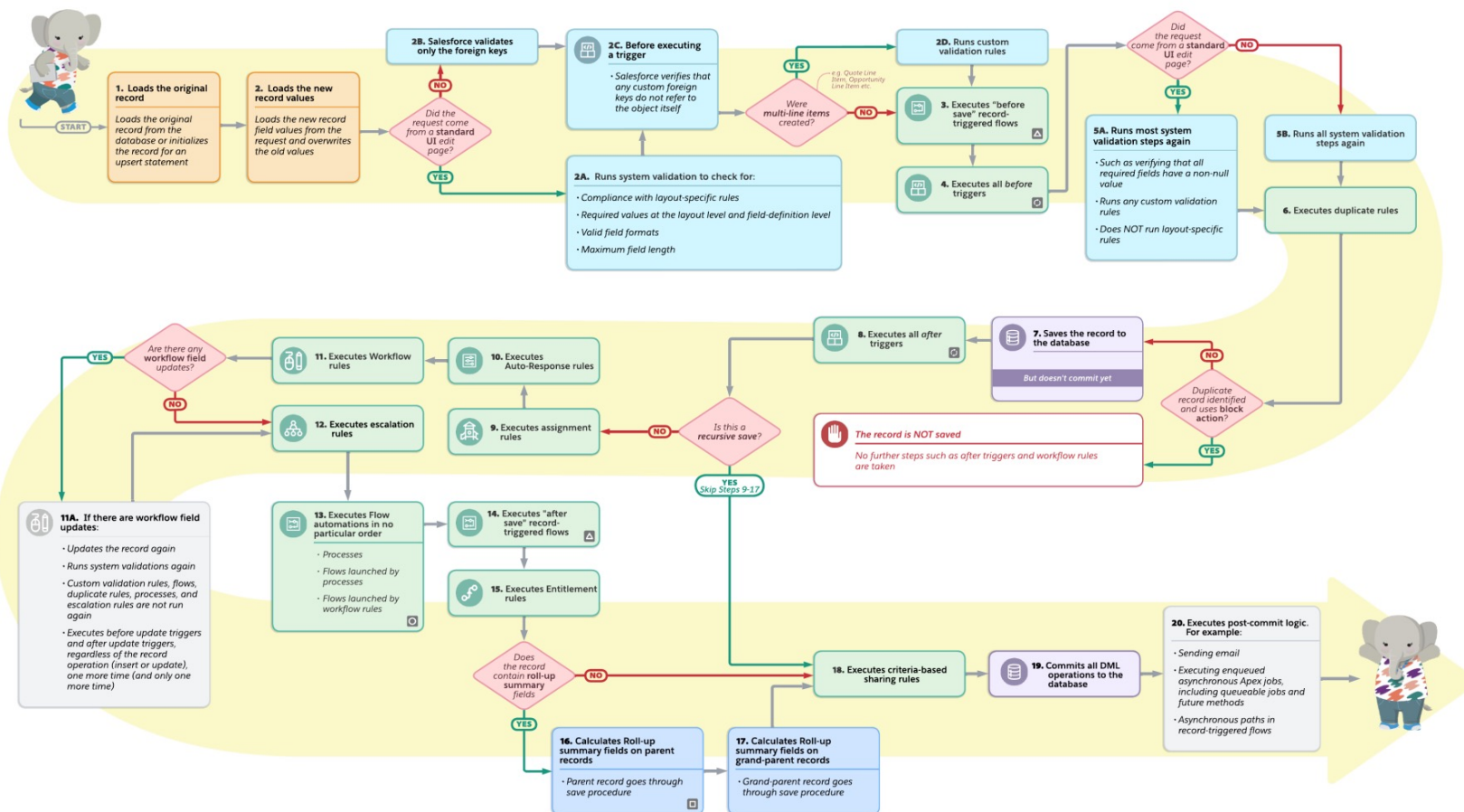


LEGEND:

- Entitlement rules
- Workflow
- Flow
- Trigger
- Escalation rules
- Auto-response rules
- Assignment Rules

- Execute
- Calculate
- Load
- Save/Commit
- Validate

- If multiple active record-triggered flows are configured, the order in which those flows are executed isn't guaranteed.
- The order of the trigger execution isn't guaranteed. Consider using a trigger handler framework based on best practice.
- Only relevant if the record contains a roll-up summary field or is part of a cross-object workflow.
- When a process or flow executes a DML operation, the affected record goes through the save procedure.



*Thank  
you*

