DML and Triggers

DML in salesforce

→We have seen how to retrieve data using SOQL and SOSL and store them in collection objects like Lists, Map etc.

→Now let's look at how we can add new records, update existing record or if required delete records

→ Insert, Update , Delete are some of the commonly used DML (Data Manipulation Language) operations

→ Insert a new student master record ,assuming Name is a text field and rest of the fields are optional

» Studentmaster_c stud = new studentmaster_c(Name='Studen001',studentemail__c='test@test.com');

» Insert stud;

→ You can also update records that are already present in the database

→ You can use SOQL to retrieve to store them in List object and then use Update std;

→You can use Delete to delete records (soft delete)

Using Sobjects in apex (anonymous blocks)

* Account ac= new account();
* Account[] acarray = new account[];
* Sobject variables can be used to create records or list of records
* List<Account> aclist = new List<account>();
* Aclist.add(ac);
* Use the debug log to view data system.debug(‘’);
* Labs Create data using structures,
* Loop through records

Contd.

→Sample to perform updates on student master records

» student_master__c studentlist = [SELECT Id, Name, Department__c FROM student_master__c where Department__c = 'CIVIL'];

» studentlist.Department__c = 'MECH';

» update studentlist;

→The above code updates department field for all students in CIVIL department to MECH

→Just ensure you have at least 1 record so that this works successfully as we have not done any exception handling here

→Run this from Execute Anonymous block from Developer console or Force.com IDE

→DML transactions work within anonymous block or trigger or class method and this is a transaction boundary

→ If there are 10 lines of code in the boundary and the transaction (INSERT/UPDATE) is committed to the database only if all lines do not have any errors

Governor limits

Why governor limits?

Force.com is built on a multi-tenant Architecture. In order to ensure that Apex code of a single tenant does not consume significant amount of shared resources on Force.com platform and to ensure every tenant gets equal share of the platform resources.

Per-transaction Apex Limits

Total # of SOQL Queries: 100 (Sync) | 200 (Async)
Total # of records retrieved: 50K
Total # of DML statements: 150
Total # of records processed by DML: 10K
Total heap size: 6MB (Sync) | 12MB (Async)
Maximum SOQL query runtime before Salesforce cancels the transaction: 120 seconds

https://developer.salesforce.com/docs/atlas.en-us.apexcode.meta/apexcode/apex_gov_limits.htm Labs:test out dml for inserts

Limits and DML

* TO buypass governor limits use data collections like lists ,sets,maps
* Create a list of sobjects and then insert and update the list in one go instead of multiple insert statement's which might hit the governor limit
* Example:
* List<account> acclist = new List<account>();
* Account accobj ;

While{accobj = new Account();accobj.name='test'+x; acclist.add(accobj);

}

* Insert acclist;//single insert statement

Database.class

→Let's see an example of using Database class instead of DML

→Using Database class method helps in handling errors in specified records in a given list of records

→In DML even if there is one record has some issue all records are rolled back whereas in Database class you can analyze how many records have succeeded and failed

→You can look at all error records and take corrective action

→You can decide if you want to use DML and exception handling combinations or go for Database class and do review of records that have failed DML

→DB class has certain methods /options that are not available in DML

- » transaction control and rollback
- » empty recycle bin
- » methods related to SOQL queries

DatabaseClass

```
List<studentmaster__c> studentlist = new List<studentmaster__c>();
studentlist.add(new studentmaster__c(customermailid__c='student1@stud.com'));
studentlist.add(new studentmaster__c(customermailid__c='student2@stud.com'));
Database.SaveResult[] srList = Database.insert(studentlist, false);
for (Database.SaveResult sresult : srList){
    if (sresult.isSuccess()){
    System.debug('student successly created: ' + sresult.getId()); }
 else  {  for(Database.Error err : sresult.getErrors())
     {
      System.debug(err.getStatusCode() + ': ' + err.getMessage());
      System.debug('Fields that have errors ' + err.getFields());
     }
 }
}
```

Database.saveresults

* If databaseinsert or update is not given with false ,then flow will not
* Return db result and move into exception

* Database.update(listobj,false);
* Database.update(listobj); all will fail even if one record errors out

* Lab:Try bulk update of student master records and process for any validation errors or other issues using saveresults

Transaction Database Savepoint

* Create transaction control using database.savepoint
* Rollback to savepoint will reverse any trasaction created between savepoint
* Savepoint sp = Database.setSavepoint();
* All Transaction in between this statements form a transaction unit explicitly.
* Database.rollback(sp);

- *Apex code must provide proper exception handling.*
- *When querying large data sets, use a SOQL "for" loop.*
- *We should not use SOQL or SOSL queries inside loops.*
- *Apex code must provide proper exception handling.*
- *We should prevent SOQL and SOSL injection attacks by using static queries, binding variables or escape single quotes method.*
- *Should use SOSL rather than SOQL where possible.*

Where is the Apex in trigger?

- Apex Code that executes before or after a DML event
  - Work in same way as database triggers
  - DML events include: insert, update, delete, undelete
  - Before and After events
    - Before triggers can update or validate values before they are saved
    - After triggers can access field values to affect changes in other records
  - Access to “old” & “new” lists of records

Exception handling

→What is an exception? Exceptions happen at the time of execution of a program and the normal flow/execution is stopped

→Exceptions needs to be handled else the programs stop abruptly and cause trouble to the end users resulting in broken process flows

→Exception can occur due to various reasons like invalid data entry by users that is not handled properly or poor coding practices etc.

→Exception handling statements are available in Apex like try, catch, finally that help in gracefully handling them

→Throw statement helps in creating an exception and handing it over to the Apex engine

→Exception records the following details, Error that happened when a method was executed, Type of error, state of the program when error occurred

→System defined exceptions

- » DML exceptions
- » Query exception – Can occur if you get zero row or more than one row but you are expecting only one r your code and assign the SOQL result to a sObject variable
- » Null pointer exceptions
- » sObject exception
- » String exception
- » Type exception
- » Math exception

→User-defined exceptions

Exception handling lab

```
public  class Exceptionhandlero {
   public static void excpmethod(){
      try {

       // Account acc =[select id , name from account  where name ='pop'];
          account acc = [select id from account limit 2];
         // system.debug(acc.name);
          //Account    acc = new Account(); //acc.AccountNumber ='0'; // acc.name='Test';          insert acc;
          // Causes an SObjectException because we didn't retrieve
          // the Total_Inventory__c field.

      }
        catch(QueryException e){
          System.debug('Query Exception caught: ' + e.getMessage() +e.getTypeName());
      }catch(DmlException e) {
          System.debug('DmlException caught: ' + e.getMessage()+e.getTypeName());
      } catch(SObjectException e) {
```

Custom Exception

* Apex allows user to raise custom exception similar to java.

```
public class Exceptionhandler1Exception extends Exception{
    public void excpmethod(){
        try{
            throw new  Exceptionhandler1Exception('raisecustomexception');
        }catch(Exceptionhandler1Exception e){
        }
    }
}
```

Triggers

→Triggers are similar to Workflow rules in the way they are defined on a specific object and execute when a record is created or updated but triggers are written using Apex

→Triggers also work when delete, undelete and few other operations like merge, upsert

→Triggers help you perform any field updates on the record that is inserted or updated before it is committed to the database

→Simple example can be updating a field on the object, but usually it is recommended to use Process builder or Workflow field update but for complex validations , say querying all related records or sibling records and do validation and update them

→Triggers are bulk by default which means you can get one or more records (say bulk updates)and apex code should be able to handle this in the trigger code

→You should also be aware of the Governor limits as you may run into issues if not handled properly

Trigger context

→Trigger.new and Trigger.old provides a list of records that have a old copy of the record and new version of the records

→Refer to the apex reference guide for more details on triggers

→It's advisable to write one trigger per object and define classes to handle business logic. Avoid writing business lc in triggers

→Be aware that if workflow field updates and trigger is defined on the same object , you need to be aware that the trigger will fire twice once for insert , then once after the field update event

→Trigger should be set to Active

→Merge statements and upsert also can be handled. Merge events fires one update and delete (one for each recor If there are 3 duplicates and you merge them , 1 insert and 2 updates will be fired

Trigger Context variables

* **1) isExecuting** Returns true if the current context for the Apex code is a trigger, not a Visualforce page, a Web service, or anexecuteanonymous() API call.
  2) isInsert Returns true if this trigger was fired due to an insert operation, from the Salesforce user interface, Apex, or theAPI.
  3) isUpdate Returns true if this trigger was fired due to an update operation, from the Salesforce user interface, Apex, or theAPI.
  4) isDelete Returns true if this trigger was fired due to a delete operation, from the Salesforce user interface, Apex, or theAPI.
  5) isBefore Returns true if this trigger was fired before any record was saved.
  6) isAfter Returns true if this trigger was fired after all records were saved.

Trigger Context variables

* **7) isUndelete** Returns true if this trigger was fired after a record is recovered from the Recycle Bin (that is, after an undelete operation from the Salesforce user interface, Apex, or the API.)
  8) new Returns a list of the new versions of the sObject records.Note that this sObject list is only available in insert and update triggers, and the records can only be modified in before triggers.
  9) newMap A map of IDs to the new versions of the sObject records. Note that this map is only available in before update, after insert, and after update triggers.
  10) old Returns a list of the old versions of the sObject records.Note that this sObject list is only available in update and delete triggers.
  11) oldMap A map of IDs to the old versions of the sObject records.Note that this map is only available in update and delete triggers.
  12) size The total number of records in a trigger invocation, both old and new.

Simple trigger

```
trigger simpleTrigger on Account (after insert) {
    for (Account a : Trigger.new) {
        // Iterate over each sObject
    }

    // This single query finds every contact that is associated with any of t
    // triggering accounts. Note that although Trigger.new is a collection of
    // records, when used as a bind variable in a SOQL query, Apex automatica
    // transforms the list of records into a list of corresponding Ids.
    Contact[] cons = [SELECT LastName FROM Contact
                      WHERE AccountId IN :Trigger.new];
}
```

Trigger context usage

```
trigger myAccountTrigger on Account(before delete, before insert, before update,
                                    after delete, after insert, after update) {
if (Trigger.isBefore) {
     if (Trigger.isDelete) {

          // In a before delete trigger, the trigger accesses the records that will be
          // deleted with the Trigger.old list.
          for (Account a : Trigger.old) {
               if (a.name != 'okToDelete') {
                    a.addError('You can\'t delete this record!');
               }
          }
     } else {

     // In before insert or before update triggers, the trigger accesses the new records
     // with the Trigger.new list.
          for (Account a : Trigger.new) {
               if (a.name == 'bad') {
                    a.name.addError('Bad name');
               }
     }
     if (Trigger.isInsert) {
          for (Account a : Trigger.new) {
               System.assertEquals('xxx', a.accountNumber);
               System.assertEquals('industry', a.industry);
               System.assertEquals(100, a.numberofemployees);
               System.assertEquals(100.0, a.annualrevenue);
               a.accountNumber = 'yyy';
          }

// If the trigger is not a before trigger, it must be an after trigger.
} else {
     if (Trigger.isInsert) {
          List<Contact> contacts = new List<Contact>();
          for (Account a : Trigger.new) {
               if(a.Name == 'makeContact') {
                    contacts.add(new Contact (LastName = a.Name,
                                              AccountId = a.Id));
               }
          }
```

Labs

Labs:Create trigger to avoid deletion of Accounts if Contacts are linked to it.
Labs:Do not Allow deletion of Positions with active job applications related to the position

Trigger based on profile lab

* system.debug(userinfo.getProfileId());
* String Name = [select name from Profile where id =:userinfo.getProfileId()].name;
* system.debug(Name);

Avoid soql inside loops

BAD CODE:

```
trigger GetContacts on Accounts (after insert, after update) {
for(Account a : Trigger.new){
List c = [SELECT Id FROM Contact WHERE AccountId = a.Id];
}
}
```

GOOD CODE:

```
trigger GetContacts on Accounts (after insert, after update) {
Set ids = Trigger.newMap.keySet();
List c = [SELECT Id FROM Contact WHERE AccountId in :ids];
}
```

Avoid DML inside loops

BAD CODE:

```
trigger UpdateContacts on Accounts (after insert, after update) {
for(Account a : Trigger.new){
List cl = [SELECT Id, LastName FROM Contact
WHERE AccountId = a.Id];
for (Contact c: cl){
c.LastName = c.LastName.toUpperCase();
}

UPDATE cl;
```

GOOD CODE:

```
trigger UpdateContacts on Accounts (after insert, after update) {
Set ids = Trigger.newMap.keySet();
List cl = [SELECT Id, LastName FROM Contact
WHERE AccountId in :ids];
for(Contact c: cl){
c.LastName = c.LastName.toUpperCase();
}

UPDATE cl;

}
```

Order of execution

→When a record is insert or updated the following things are executed in order

→Loads the original record from the database or initializes a new record

→Old values are over written with new values

→All before trigger events

→System validations and user defined validations followed by duplicate rules

→Saves the record to the database without commit (Record ID and other system fields are not ready)

→Execute all after triggers

→Execute assignment and auto - response rules

→Workflow rules(field updates and others) resulting in firing before and after update triggers only once

→Escalation rules

→Roll - up summary fields on the parent object and goes up to grand parent object

→Criteria based sharing rules

→Commit all DML ops to the database

→Execute post commit logic such as sending email

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

Trigger handler factory class implementation

* Realtime implementation of trigger factory template

* Create Interface with all the bulk and for update methods
* Create specifichandler class for each object ,which implements this interface
* Create common trigger handler class to invoke each of these handlers based on the sobject type being passed by the Trigger

* Labs:Create trigger to avoid deletion of Accounts if Contacts are linked to it.
* Bypass this trigger using concept of static variable