# Apex -2

# Sharing using Apex

* Records can be shared using apex
* Each object has Objectshare entity which can be used for sharing

# Apex sharing code

```
List<ContactShare> sharesToCreate = new List<ContactShare>();

for (Contact contact : Trigger.new) {
  if (contact.Make_Public__c == true) {

    // create the new share for group
    ContactShare cs = new ContactShare();
    cs.ContactAccessLevel = 'Edit';
    cs.ContactId = contact.Id;
    cs.UserOrGroupId =  groupId;
    sharesToCreate.add(cs);

  }
}

// do the DML to create shares
if (!sharesToCreate.isEmpty())
  insert sharesToCreate;
```

Use case: Create edit sharing at run time user who has write access to the record,use parentid to update id of record.

# Checking access at runtime

* SELECT RecordId, HasReadAccess, HasTransferAccess, MaxAccessLevel FROM UserRecordAccess WHERE UserId = [single ID] AND RecordId = [single ID]
* Properties :HasReadAccess, HasEditAccess, HasDeleteAccess, HasTransferAccess, and HasAllAccess. You may includeMaxAccessLevel

* Labs: Create sharing at run time using other user provided by salesforce in dev org
* Check access at run time
* Use role heirarchy and OWD private settings for the same

# Some Key Classes

* Userinfo- used frequently in test classes

* getProfileId()
* Getuserid()
* Can be used to fetch logged in user details for run time

* Schema Class – Gets details of objects ,fields ,properties, all information about object metadata.
* Savepoint sp = Database.setSavepoint();
* Database.rollback(sp);
* Database.query(dynamic sql)
* @future(Callout= true) will make the method run in a separate thread(asych)

# Using Schema Class to get sobject details

* Schema.getGlobalDescribe(): Returns Sobject map from org
* :public Map <String, Schema.SObjectType> schemaMap = Schema.getGlobalDescribe();

* Loop through schemamap.keyset you will get name
* For (String x: schemamap.keyset())
* System.debug(x);

* Pass the schemmap key which is an object to another methods which will be
* Map <String, Schema.SObjectField> fieldMap = schemaMap.get('name of sobject).getDescribe().fields.getMap();
* Returns a map of fields

* Loop through fieldmap.values not keyset
* For each of these fieldmap: schema.describefieldresult dfield = sfield.getDescribe();

# Using Schema class to get field level details

* Map <String, Schema.SObjectField> fieldMap = schemaMap.get(object name).getDescribe().fields.getMap();

* Map <String, Schema.SObjectField> fieldMap = schemaMap.get(selectedObject).getDescribe().fields.getMap(); for(Schema.SObjectField sfield : fieldMap.Values()) { schema.describefieldresult dfield = sfield.getDescribe(); FieldWrapper wObj = new FieldWrapper(); wObj.fieldName = dfield.getLabel (); wObj.fieldAPIName = dfield.getname(); listField.add(wObj); }

# Classess

* Json Class – serialise and deserialise data
* Limits Class to check number of limits approaching in realtime
* Lab prevent hitting dml limits in realtime when updating one record at a time in a loop

# Asynch Apex

# Asynch Apex

| Asynchronous Apex Feature | When to Use |
| --- | --- |
| Future Methods | • When you have a long-running method and need to prevent delaying an Apex transaction<br>• When you make callouts to external Web services<br>• To segregate DML operations and bypass the mixed save DML error |
| Queueable Apex | • To start a long-running operation and get an ID for it<br>• To pass complex types to a job<br>• To chain jobs |
| Batch Apex | • For long-running jobs with large data volumes that need to be performed in batches, such as database maintenance jobs<br>• For jobs that need larger query results than regular transactions allow |
| Scheduled Apex | • To schedule an Apex class to run on a specific schedule |

# Key Features

* Bulk processing
* Implements batchable interface
* Start(),execute(),Finish methods

* Execute runs in batches of 200
* The initial query locator that is defined in your start() method can return up to 50,000,000 (yes, 50 million) records.

* each time that the your Batch Apex class's execute() method is called constitutes a new transaction, so that Limit is reset every time that method is called. This means that, if you needed to, you could query for up to 50,000 records each time that the execute method is called!

* Each batch Apex invocation creates an AsyncApexJob record. To construct a SOQL query to retrieve the job's status, number of errors, progress, and submitter, use the AsyncApexJob record's ID.

# Batch Apex

```
Batch Schedule Class
global class batchContactUpdate implements Database.Batchable<sObject>
{
   global Database.QueryLocator start(Database.BatchableContext BC)
   {
      String query = 'SELECT Id, FirstName,LastName FROM Contact';
      return Database.getQueryLocator(query);
   }

   global void execute(Database.BatchableContext BC, List<Contact> scope)
   {
      for(Contact a : scope)
      {
         a.FirstName=a.FirstName+'FirstName is Updated';
         a.LastName = a.LastName +'LastName is updated';
      }
      update scope;
   }
   global void finish(Database.BatchableContext BC)
   {
   }
}
```

# Schedule

* **Schedule Class**
  **----------**

```
global class BatchScheduleUpdate implements Schedulable
{
   global void execute(SchedulableContext sc)
   {
      // Implement any logic to be scheduled

      // We now call the batch class to be scheduled
      BatchContactUpdate b = new BatchContactUpdate ();

      //Parameters of ExecuteBatch(context,BatchSize)
      database.executebatch(b,200);
   }

}
```

* **In order to schedule a batch apex class,implement a schedulable interface and instantiate the batch from within execute**

# Final scheduling

* **Schedule from Developer Console**
  **----------**
* **BatchScheduleUpdate batchSch=new BatchScheduleUpdate();**
* **String sch='0 5 2 * * ?';**
* **//System.schedule(String jobName, String cronExp, APEX_OBJECT schedulable);**
* **System.schedule('Batch Schedule', sch , batchSch);**

## Queueable Example:

```
public class AsyncExecutionExample implements Queueable {
   public void execute( QueueableContext context) {
      Account a = new Account(Name='Acme',Phone='(415) 555-1212');
      insert a;
   }
}
```

**To add this class as a job on the queue, call this method:**

```
ID jobID = System.enqueueJob(new AsyncExecutionExample());
```

To add this class as a job on the queue, call this method:

```
ID jobID = System.enqueueJob(new AsyncExecutionExample());
```

To query information about your submitted job, perform a SOQL query on AsyncApexJob by filtering on the job ID that the System.enqueueJob method returns. This example uses the jobID variable that was obtained in the previous example.

```
AsyncApexJob jobInfo = [SELECT Status,NumberOfErrors FROM AsyncApexJob WHERE Id=:jobID];
```

# Queueable Example:

Chaining Jobs

If we need to run a job after some other processing is done first by another job, we can chain queueable jobs. To chain a job to another job, submit the second job from the execute() method of your queueable class. You can add only one job from an executing job.

```
public class AsyncExecutionExample implements Queueable {
   public void execute( QueueableContext context) {
      // Your processing logic here

      // Chain this job to next job by submitting the next job
      System.enqueueJob(new SecondJob());
   }
}
```

# Stopping a job

* System.AbortJob(aJob.Id);

# Batch Apex

- Advantages

1. It can process up to 50m records
2. It can be scheduled to run at a particular time
3. Asynchronous processing

- Disadvantages

1. Only 5 concurrent batch jobs running at a time
2. It's difficult to troubleshoot
3. Execution may be delayed based on server availability
4. @future methods are not allowed

| Batchable | @future | Queueable |
| --- | --- | --- |
| - Good at processing large number of records (50m).<br>- Can be scheduled to run at a certain time<br>- Maximum of 5 concurrent jobs running at a time<br>- You need good error handling for troubleshooting | - Quick async processing (typically 1 record at a time) e.g. avoid mixed DML or a web service callout<br>- Faster than a Batch<br>- Easy to implement<br>- Only accepts primitive type arguments<br>- Can't chain jobs<br>- Hard to monitor | - Quick async processing that supports primitive types<br>- Faster than a batch<br>- Ability to chain jobs<br>- Can't have more than 1 job doing callouts within the chain<br>- Can be monitored |

# Test Class

- Invoke asynchronous Apex between the *Test.startTest()* and *Test.stopTest()* methods.
- The *Test.stopTest()* method will execute all asynchronous and scheduled Apex synchronously.

# Deployment options

* Changeset
* IDE
* ANT