Understanding BUnderstanding Batch Apex Terminology

There is a governor limit on the total number of records retrieved by a SOQL query. We can only retrieve 50,000 records. But what if we want to retrieve more than 50,000 records. Here Batch Apex comes in as a savior. Using Batch Apex class we can process records in batches asynchronously. Each execution of a batch Apex job is considered a discrete transaction.

When we use batch apex we implements Database.batchable() interface

The Database.batchable() contains three methods that we need to defined inside batch class and they are,

1) start

2) execute

3) Finish

1. Start Method - This method is called once at the beginning of a Batch Apex job and returns either a Database.QueryLocator object or an Iterable that contains the records or objects passed to the job.

2. Execute Method - This method is called for each batch of records.

Execute method is called after the start method, it receives the list of record which can be processed as required.

The default batch size is 200 records. Batches of records are not guaranteed to execute in the order they are received from the start method.

3. Finish Method - The finish method is called after all batches are processed. Use this method to send confirmation emails or execute post-processing operations.

Syntax:

```
global class batch implements Database.Batchable <sObject> {

   global (Database.QueryLocator or  Iterable<sobject>)
      start(Database.BatchableContext bc) {

         //query on object;
         //return Database.getQueryLocator(query);

      }

   global void execute(Database.batchableContext bc, List <SObject> scope) {

      //some processing.
   }

   global void finish(Database.BatchableContext bc) {
      //job such as sending email or calling another batch class

   }
```

}

POINTS TO REMEMBER

• use Database.getQueryLocator when you are using simple query to create scope for object in batch job.

• use Iterable when you have complex criterion to process the record.

• If we use query locator object governor limit is bypassed.

• If we use Iterable governor limit is enforced.

• The order of execution is not guaranteed in execute method.

• If the batch is executed without the optional scope parameter and If in batch apex suppose we have 600 jobs and there are total 200 records to be proceeds in each transaction, so in total we have 3 transaction. If first transaction succeeds but second fails so in this case the records update done in first transaction are not rolled back.

Limits in Batch Apex

• Up to five queued or active batch jobs are allowed for Apex

• A maximum of 50 million records can be returned to the Database.QueryLocator object.

• If the start method returns a QueryLocator, the optional scope parameter of Database.executeBatch can have a maximum value of 2,000. If set to a higher value, Salesforce chunks the records returned by the QueryLocator into smaller batches of up to 2,000 records.

• If the start method returns an Iterable, the scope parameter value has no upper limit; however, if you use a very high number, you may run into other limits.

• The start, execute, and finish methods can implement up to 10 callouts each

• The maximum number of batch executions is 250,000 per 24 hours

• Only one batch Apex job's start method can run at a time in an organization. Batch jobs that haven't started yet remain in the queue until they're started.

Interview Series

Hey guys, from this post i am starting interview series on Asynchronous Apex (i.e. Batch Apex, Apex Scheduler, Future Methods and Queueable Apex) which are very helpful in interviews.

• Interview Series: Batch Apex

• Interview Series: Apex Scheduler

• Interview Series: Future Methods

• Interview Series: Queueable Apex

MIND IT !

Need	Batch Apex	Scheduled Apex	Future Methods	Queueable Apex
Process large data volumes	Yes			
Execute logic at specific time intervals		Yes		
Perform Web Service Calls	Yes	Yes	Yes	
Sequential Processing (Chaining)			Yes	
Support for Non-Primitive (sObject, custom Apex types)			Yes	Yes	Yes

In this blog series, I have tried to cover Batch Apex based questions that are often asked from a Salesforce developer in an interview.

Let start the first interview series on Batch Apex (Between Interviewer & Interviewee).

Interview Series: Batch Apex

Interviewer: Why do we use Batches?

Interviewee: If we have a use case to process millions of records, it would be difficult to do this manually as well as in a synchronous context (the time limit is 10 seconds) so we use batches to process bulk data or records together, with a new set of Governor limits per transaction.

Interviewer: Why to use Batch class as we already having data loader to process the bulk data.

Interviewee: Agree with this point if and only if the data needs to be updated is static or predefined or which can be done through excel.

We will choose batch class if we have to perform some custom calculations at run time or if you want to run some complex logic which can't be driven by excel sheets in those cases we have to go with batch classes.

For Examples:

- Do some relationship quires to update the data.
- Make a call out to get the some information related to each record.

Interviewer: What interface will you use for batch apex?

Interviewee: It is Database.Batchable interface.

Interviewer: What did you mean by “a new set of governor limits per transaction”?

Interviewee: Batch Jobs can be operated on any size of records, with a maximum of 200 records per batch. Each batch is a transaction. The Apex governor limits are reset for each transaction.

Interviewer: Can you write a batch class blueprint?

Interviewee:

```
global class batch implements Database.Batchable <sObject> {

    global (Database.QueryLocator or  Iterable<sobject>)
        start(Database.BatchableContext bc) {

            //query on object;
            //return Database.getQueryLocator(query);

        }
```

```
    global void execute(Database.batchableContext bc, List <SObject> scope) {

        //some processing.
    }

    global void finish(Database.BatchableContext bc) {
        //job such as sending email or calling another batch class

    }
}
```

Interviewer: What is the difference between Database.QueryLocator & Iterable<sobject> used in batch apex?

Interviewee: In Database.QueryLocator, the governor limit for the total number of records retrieved by SOQL queries is bypassed and you can query up to 50 million records.

In Iterable<sobject>, we can create a custom scope for processing, which would be not possible to create using SOQL where clauses. The governor limit for the total number of records retrieved by SOQL queries is still enforced. For example,

```
global class CustomIterable implements Iterator<account>{
    List<account> acc {get; set;}
    Integer i {get; set;}
    public CustomIterable(){
        acc = [SELECT Id, Name, NumberOfEmployees FROM Account WHERE Name = 'false'];
        i = 0;
```

```
    }
    global boolean hasNext(){
        if(i >= acc.size()) {
            return false;
        } else {
            return true;
        }
    }
    global Account next(){
        // 7 is an arbitrary constant in this example that represents the maximum size of the list.
        if(i == 7){return null;}
        i++;
        return acc[i-1];
    }
}
```

So we can return our custom scope by simply calling as below...

```
1
2
3
global Iterable<account> start(Database.batchableContext info){
      return new CustomIterable();
  }
```

Also, If the code accesses external objects and is used in batch Apex, use Iterable<sObject> else Database.QueryLocator.

Interviewer: Can you explain more about interable interface. What you mean by custom scope?

Interviewee: When you have a use case in which SQOL can’t be constructed, we can use custom code to create the scope of a batch.

For example, A batch accepts account records, where each account is having more than 2 contacts and each contact’s mailing address should not be the same.

Now, you can’t create a SOQL for this use case, so use the iterable interface, and create a list of accounts using queries and loops, and return the final processed account list, to the batch. This final list is called a custom scope.

Interviewer: If We use Itereable<sObject>, will We be able to process Batches in a specific order?

Interviewee: Batches of records tend to execute in the order in which they’re received from the start method. However, the order in which batches of records execute depends on various factors. The order of execution isn’t guaranteed. So even if you do ORDER BY, there is no guarantee of processing in the same order.

Interviewer: In a batch context, can I process Data in a specific order?

Interviewee: Yes, You can order the scope of that batch, however, you want and process it.

Interviewer: Can I query related records using Database.QueryLocator ?

Interviewee: Yes, You can do subquery for related records, but with a relationship subquery, the batch job processing becomes slower. A better strategy is to perform the subquery separately, from within the execute method, which allows the batch job to run using the faster.

Interviewer: Can I Use FOR UPDATE in SOQL using Database.QueryLocator?

Interviewee: No, We can't. It will through an exception stating that

"Locking is implied for each batch execution and therefore FOR UPDATE should not be specified"

The reason is, If we query the entire database of an org using SELECT.

FOR UPDATE, we would lock the entire database as long as the batch was active.

Interviewer: What is the state of batch apex?

Interviewee: Batch Apex is typically stateless. Each execution of a batch Apex job is considered a discrete transaction.

For example, a batch Apex job that contains 1,000 records and uses the default batch size is considered five transactions of 200 records each.

Interviewer: Let's say Record A has to be processed before Record B, but Record B came in the first Batch and Record A came in the second batch. The batch picks records which are unprocessed every time it runs. How will you control the processing Order?

Interviewee: The Processing order can't be controlled, but we can bypass the Record B processing before Record A. We can implement Database.STATEFUL and use one class variable to track whether Record A has processed or not. If not processed and Record B has come, don't process Record B. After all the execution completes, Record A has already been processed so Run the batch again, to process Record B.

Interviewer: What is Database.Stateful?

Interviewee: Database.Stateful is a Marker interface helps in maintaining state across transactions. When using Database.Stateful, only instance member variables retain their values between transactions.

Static member variables don't retain their values and are reset between transactions.

Maintaining state is useful for counting or summarizing records as they're processed.

For example, we'll be updating contact records in our batch job and want to keep track of the total records affected so we can include it in the notification email.

Interviewer: Is there any other interface that can be implemented by a Batch apex?

Interviewee: Database.AllowsCallouts is another Marker Interface, which is implemented to make HTTP callouts through Batch.

Interviewer: How to monitor Batch job?

Interviewee: To monitor the status of the jobs

1. From Setup, enter Jobs in Quick Find box
2. Select Apex Jobs

–> Go to Setup --->Apex Job page :

Interviewer: What is the Limit of Size of scope in a batch?

Interviewee: If the start method of the batch class returns a QueryLocator, the optional scope parameter of Database.executeBatch can have a maximum value of 2,000. If set to a higher value, Salesforce chunks the records returned by the QueryLocator into smaller batches of up to 2,000 records. If the start method of the batch class returns an iterable, the scope parameter value has no upper limit.

Interviewer: If a batch is having 200 records and 1 record fails what will happen?

Interviewee: If any record fails all 200 record will fail but next batch will get executed.

Interviewer: How to calculate the batch size if we are making any call out from the batch class execute method?

Interviewee: Basically in salesforce we have limitation of 100 call outs for a transaction. When it comes to batch class each execute method call will be considered as one transaction. So, in this case your batch size must be calculated in such way

Batch size = (No of callouts allowed for single transaction /total number of call outs for each record) - 1;

Batch size = (100 /total number of call outs for each record) - 1;

Interviewer: How many times the execute method will be executed to process the 1236 records.

Interviewee: It depends on your batch size what you have configured at the time of calling the batch class from schedule class.

Execution method count = Total No Of Records/Batch Size (Any decimal ceil it to upper value)

If you haven't set any batch size then :

1236 = 6.18 = 7 times execute method will be called

200

Interviewer: Can we call the batch into another batch apex?

Interviewee: Yes, we can call from the finish method.

Interviewer: Can we call batch apex into another batch in execute method?

Interviewee: Only in batch class finish method, We can call another batch class. If you will call another batch class from batch class execute and start method, then Salesforce will throw below runtime error.

System.AsyncException: Database.executeBatch cannot be called from a batch start, batch execute, or future method.

Interviewer: Why to call only from the finish method why not from execute?

Interviewee: Because the execute method will gets invoked multiple times based on the volume of the records and batch size. So, if your calling it from execute method then the chaining class will get called multiple times which is not an suggested way of doing.

We can also use the Queueable Apex for chaining of the jobs.

Interviewer: Let's say, we have run an apex batch to process 1000 records, and It is running with batch size 200. Now, while doing DML on 398th record, an error occurred, What will happen in that case?

Interviewee: In batches, If the first transaction succeeds but the second fails, the database updates made in the first transaction are not rolled back.

Since the batch size is 200, so the first batch will be processed completely and all data will be committed to DB. In seconds batch, if we are committing records using normal DML statements like insert, update than the whole batch will be rollbacked. So records 201 to 400 will not be processed.

If we use the Database DML operations like Database.insert with AllOrNone as false, then partial commit can happen and only 398th record will not be processed in that batch and total 199 records will be processed. Also, the other batch execution will not gonna be hampered.

Interviewer: Can I call any method from Batch Apex?

Interviewee: Methods declared as future can’t be called from a batch Apex class.

Interviewer: Is there is any way through which we can call future method from batch apex?

Interviewee: As we know that a webservice can be called from batch class and webservice can call @future method. So in your batch class call web service and which can call your @future method. Also, you can call future methods from the finish method in the batch class.

Interviewer: How many can concurrent batch jobs be added to the queue?

Interviewee: At most, there can be 5 batch jobs queued at a time.

Interviewer: Can we call the batch apex from triggers in salesforce?

Interviewee: Yes, it is possible. We can call a batch apex from trigger but we should always keep in mind that we should not call batch apex from trigger each time as this will exceeds the governor limit this is because of the reason that we can only have 5 apex jobs queued or executing at a time.

Interviewer: Can we call webservice callout from batch apex?

Interviewee: To make a Webservice callout in batch Apex, we have to implement Database.AllowsCallouts interface.

Interviewer: How many times start, execute, finish methods will execute in batch apex?

Interviewee: Start method, finish method one time, execute method it depends on requirement. Based on the batch size and data retrieved in Start method.

Interviewer: What is the Batch executions limit per day?

Interviewee: The maximum number of batch executions is 250,000 per 24 hours.

Interviewer: How can we track the status of the current running batch job?

Interviewee: The job Id returned by the batchable context variable helps us in finding the status of a batch through AsyncApexJob object. For example,

1

2

3

4

5

6

7

8

9

10

```
11
12
13
14
15
16
17
18
19
20
21
22
global void finish(Database.BatchableContext info){

    // Get the ID of the AsyncApexJob representing this batch job
    // from Database.BatchableContext.
    // Query the AsyncApexJob object to retrieve the current job's information.

    AsyncApexJob a = [SELECT Id, Status, NumberOfErrors, JobItemsProcessed,
                      TotalJobItems, CreatedBy.Email FROM AsyncApexJob WHERE Id =
:info.getJobId()];

    // Send an email to the Apex job's submitter notifying of job completion.

    Messaging.SingleEmailMessage mail = new Messaging.SingleEmailMessage();

    String[] toAddresses = new String[] {a.CreatedBy.Email};
    mail.setToAddresses(toAddresses);
    mail.setSubject('Account and contact update' + a.Status);
    mail.setPlainTextBody
```

```
        ('The batch Apex job processed ' + a.TotalJobItems +
         ' batches with '+ a.NumberOfErrors + ' failures.'+successRecord+'successRecordids: '+
'failRecordids: '+ failRecord);
    Messaging.sendEmail(new Messaging.SingleEmailMessage[] { mail });
}
```

Interviewer: How can you stop a Batch job?

Interviewee: The Database.executeBatch and System.scheduleBatch method returns an ID that can be used in System.abortJob method.

Interviewer: How to stop all the running batches related to a particular batch classes before scheduling the same batch class from the schedule class.

Interviewee:

1

2

```
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
global class CaseCreationonSchedular implements Schedulable
{
   global void execute(SchedulableContext SC)
   {


      for(AsyncApexJob ap: [SELECT Id, ApexClass.Name,createddate, JobType, Status,
JobItemsProcessed, MethodName, ParentJobId
                            FROM AsyncApexJob
                            Where ParentJobId!=Null
                            AND JobType IN('BatchApex','BatchApexWorker')
                            And ApexClass.Name='YourBatchClassName'
                            And Status NOT IN('Completed','Aborted')])
      {
         System.abortJob(ap.ParentJobId);
```

```
        }
        YourBatchClassName cls = new YourBatchClassName();
        DataBase.executeBatch(cls);
    }
}
```

Interviewer: What is Apex Flex Queue?

Interviewee: At a time salesforce allows 5 batches to be running or to be in queued state. So, if you have consumed all these 5 limits and if system has received one or more batch execution request all these waiting batch will be stored in this Apex Flex Queue.

The maximum number of batch classes allowed in Apex Flex Queue for execution is 100.

Interviewer: Let's say, I have 150 Batch jobs to execute, Will I be able to queue them in one go?

Interviewee: Once you run Database.executeBatch, the Batch jobs will be placed in the Apex flex queue and its status becomes Holding. The Apex flex queue has the maximum number of 100 jobs, Database.executeBatch throws a LimitException and doesn't add the job to the queue. So atmost 100 jobs can be added in one go.

Also, if Apex flex queue is not enabled, the Job status becomes Queued, Since the concurrent limit of the queued or active batch is 5, so atmost 5 batch jobs can be added in one go.

Interviewer: Can I change the order of already queued Batch Jobs?

Interviewee: Only jobs having status as Holding can be moved. It can be done through UI of Apex Flex queue or we can use Apex Flex Queue Methods.

1

Boolean isSuccess = System.FlexQueue.moveBeforeJob(jobToMoveId, jobInQueueId);

Interviewer: How Can I schedule a Batch Apex?

Interviewee: Through System.scheduleBatch Method, we can schedule batch for once at a future time. This method returns the scheduled job ID also called CronTrigger ID.

1

2

3

4

```
String cronID = System.scheduleBatch(reassign, 'job example', 1);


CronTrigger ct = [SELECT Id, TimesTriggered, NextFireTime
                  FROM CronTrigger WHERE Id = :cronID];
```

Interviewer: Is it possible to do Synchronous Web service callouts from scheduled apex?

Interviewee: No, Synchronous Web service callouts are not supported by scheduled Apex. However, if your scheduled Apex executes a batch job, callouts are supported by the batch class.

Interviewer: Can we call the scheduler from the future method?

Interviewee: Yes..!! We can call the scheduler from the future method.

Interviewer: Can we modify the scheduler class or classes referenced by this scheduler class if any scheduled job is pending?

Interviewee: No, If there are one or more active scheduled jobs for an Apex class, you cannot update the class or any classes referenced by this class through the Salesforce user interface.

Interviewer: Can I call Queueable from a batch?

Interviewee: Yes, But you're limited to just one System.enqueueJob call per execute in the Database.Batchable class. Salesforce has imposed this limitation to prevent explosive execution.

Interviewer: How Can I Test a Batch Apex?

Interviewee: We can test it by calling the Batch job between Test.startTest() and Test.stopTest(). Using the Test methods startTest and stopTest around the executeBatch method to ensure that it finishes before continuing your test. All asynchronous calls made after the startTest method are collected by the system. When stopTest is executed, all asynchronous processes are run synchronously.

Also we can test only one execution of the execute method. So we need to set the scope as per the governor limits.

Interviewer: How many records we can insert while testing Batch Apex?

Interviewee: We have to make sure that the number of records inserted is less than or equal to the batch size of 200 because test methods can execute only one batch. We must also ensure that the Iterable returned by the start method matches the batch size.

Batch Apex Real Time Scenarios Based Questions

Batch Apex Scenario 1 :

Write a Batch apex class to automatically deactivate users when the user's Last Login Date is greater than 30 days.

Batch Apex Code: batchDeactivateUsers.apxc

```
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
global class batchDeactivateUsers implements Database.Batchable<sobject> {

    global Database.QueryLocator start(Database.BatchableContext bc){
```

```
        string query = 'SELECT Name, isActive, LastLoginDate FROM User WHERE
LastLoginDate < LAST_N_DAYS:30';
        return database.getQueryLocator(query);
    }

    global void execute(Database.BatchableContext bc, List<user> users){

        List<user> userlist = new List<user>();

        for(User u : users){
            u.IsActive = false;
            userlist.add(u);
        }

        update userlist;

    }

    global void finish(Database.BatchableContext bc){

    }

}
```

Batch Apex Scenario 2 :

Update account description, number of employees, contact last name using batch apex. Get the failure record ids in the email. Also, schedule the job for every Monday.

Batch Apex Code: batchUpdateAccountsContacts.apxc

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26

```
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
global class batchUpdateAccountsContacts implements Database.Batchable
<sObject>,Database.Stateful,Schedulable {

    global batchUpdateAccountsContacts(){
```

```
    }
    Set<id> successRecord = new Set<id>();
    Set<id> failRecord = new Set<id>();

    global Database.QueryLocator start(Database.BatchableContext info){
        String SOQL='Select id, name, NumberOfEmployees, description,(select id, name from
contacts) from Account';
        return Database.getQueryLocator(SOQL);
    }
    global void execute(Database.BatchableContext info, List<account> scope){
        List<account> accsToUpdate = new List<account>();
        List<contact> contUpdate = new List<contact>();
        for(Account a : scope)
        {
            a.description ='Welcome to Learnfrenzy - Best Place to Learn';
            a.NumberOfEmployees = 70;
            accsToUpdate.add(a);
            for (Contact c:a.contacts){
                c.lastname = 'test+a';
                contUpdate.add(c);
            }
        }
        Database.SaveResult[] srList = Database.update(accsToUpdate, false);
        Database.SaveResult[] srList1 = Database.update(contUpdate, false);


        for (Database.SaveResult sr : srList) {
            if (sr.isSuccess()) {
                // Operation was successful, so get the ID of the record that was processed
                successRecord.add(sr.getId());
            }
```

```
            else {
                for(Database.Error err : sr.getErrors()) {
                }
                failRecord.add(sr.getId());
            }
        }

        for (Database.SaveResult sr : srList1) {
            if (sr.isSuccess()) {
                successRecord.add(sr.getId());
            }
            else {
                for(Database.Error err : sr.getErrors()) {
                }
                failRecord.add(sr.getId());
            }
        }



    }
    global void finish(Database.BatchableContext info){


        // Get the ID of the AsyncApexJob representing this batch job
        // from Database.BatchableContext.
        // Query the AsyncApexJob object to retrieve the current job's information.
        AsyncApexJob a = [SELECT Id, Status, NumberOfErrors, JobItemsProcessed,
                          TotalJobItems, CreatedBy.Email FROM AsyncApexJob WHERE Id =
:info.getJobId()];
```

```
        // Send an email to the Apex job's submitter notifying of job completion.
        Messaging.SingleEmailMessage mail = new Messaging.SingleEmailMessage();

        String[] toAddresses = new String[] {a.CreatedBy.Email};
            mail.setToAddresses(toAddresses);
        mail.setSubject('Account and contact update' + a.Status);
        mail.setPlainTextBody

            ('The batch Apex job processed ' + a.TotalJobItems +
             ' batches with '+ a.NumberOfErrors + ' failures.'+successRecord+'successRecordids: '+
'failRecordids: '+ failRecord);
        Messaging.sendEmail(new Messaging.SingleEmailMessage[] { mail });
    }



    global void execute(SchedulableContext SC){
        database.executeBatch(new batchUpdateAccountsContacts(),100);
        //for cron expression
        // String cronexpression = '0 0 0 ? * * *'
        // System.schedule('Testing', cronexpression, testobj);
    }

}
```

Running Batch Job

1. Go to developer console

2. Open execute anonymous window.

3. Run the below code

1

database.executeBatch(new batchUpdateAccountsContacts(),100);

–> Open Execute Anonymous Window :

atch Apex Terminology

There is a governor limit on the total number of records retrieved by a SOQL query. We can only retrieve 50,000 records. But what if we want to retrieve more than 50,000 records. Here Batch Apex comes in as a savior. Using Batch Apex class we can process records in batches asynchronously. Each execution of a batch Apex job is considered a discrete transaction.

When we use batch apex we implements Database.batchable() interface

The Database.batchable() contains three methods that we need to defined inside batch class and they are,

1) start

2) execute

3) Finish

1. Start Method - This method is called once at the beginning of a Batch Apex job and returns either a Database.QueryLocator object or an Iterable that contains the records or objects passed to the job.

2. Execute Method - This method is called for each batch of records.

Execute method is called after the start method, it receives the list of record which can be processed as required.

The default batch size is 200 records. Batches of records are not guaranteed to execute in the order they are received from the start method.

3. Finish Method - The finish method is called after all batches are processed. Use this method to send confirmation emails or execute post-processing operations.

Syntax:

1

2

3

4

5

6

7

8

```
9
10
11
12
13
14
15
16
17
18
19
20
21
global class batch implements Database.Batchable <sObject> {

   global (Database.QueryLocator or  Iterable<sobject>)
      start(Database.BatchableContext bc) {

         //query on object;
         //return Database.getQueryLocator(query);

      }


   global void execute(Database.batchableContext bc, List <SObject> scope) {

      //some processing.
   }

   global void finish(Database.BatchableContext bc) {
```

```
        //job such as sending email or calling another batch class

    }
}
```

POINTS TO REMEMBER

• use Database.getQueryLocator when you are using simple query to create scope for object in batch job.

• use Iterable when you have complex criterion to process the record.

• If we use query locator object governor limit is bypassed.

• If we use Iterable governor limit is enforced.

• The order of execution is not guaranteed in execute method.

• If the batch is executed without the optional scope parameter and If in batch apex suppose we have 600 jobs and there are total 200 records to be proceeds in each transaction, so in total we have 3 transaction. If first transaction succeeds but second fails so in this case the records update done in first transaction are not rolled back.

Limits in Batch Apex

• Up to five queued or active batch jobs are allowed for Apex

• A maximum of 50 million records can be returned to the Database.QueryLocator object.

• If the start method returns a QueryLocator, the optional scope parameter of Database.executeBatch can have a maximum value of 2,000. If set to a higher value, Salesforce chunks the records returned by the QueryLocator into smaller batches of up to 2,000 records.

• If the start method returns an Iterable, the scope parameter value has no upper limit; however, if you use a very high number, you may run into other limits.

• The start, execute, and finish methods can implement up to 10 callouts each

• The maximum number of batch executions is 250,000 per 24 hours

• Only one batch Apex job's start method can run at a time in an organization. Batch jobs that haven't started yet remain in the queue until they're started.

Interview Series

Hey guys, from this post i am starting interview series on Asynchronous Apex (i.e. Batch Apex, Apex Scheduler, Future Methods and Queueable Apex) which are very helpful in interviews.

• Interview Series: Batch Apex

• Interview Series: Apex Scheduler

• Interview Series: Future Methods

• Interview Series: Queueable Apex

MIND IT !

Need	Batch Apex	Scheduled Apex	Future Methods	Queueable Apex
Process large data volumes	Yes			
Execute logic at specific time intervals		Yes		
Perform Web Service Calls	Yes	Yes	Yes	
Sequential Processing (Chaining)			Yes	
Support for Non-Primitive (sObject, custom Apex types)		Yes	Yes	Yes

In this blog series, I have tried to cover Batch Apex based questions that are often asked from a Salesforce developer in an interview.

Let start the first interview series on Batch Apex (Between Interviewer & Interviewee).

Interview Series: Batch Apex

Interviewer: Why do we use Batches?

Interviewee: If we have a use case to process millions of records, it would be difficult to do this manually as well as in a synchronous context (the time limit is 10 seconds) so we use batches to process bulk data or records together, with a new set of Governor limits per transaction.

Interviewer: Why to use Batch class as we already having data loader to process the bulk data.

Interviewee: Agree with this point if and only if the data needs to be updated is static or predefined or which can be done through excel.

We will choose batch class if we have to perform some custom calculations at run time or if you want to run some complex logic which can't be driven by excel sheets in those cases we have to go with batch classes.

For Examples:

- Do some relationship quires to update the data.
- Make a call out to get the some information related to each record.

Interviewer: What interface will you use for batch apex?

Interviewee: It is Database.Batchable interface.

Interviewer: What did you mean by “a new set of governor limits per transaction”?

Interviewee: Batch Jobs can be operated on any size of records, with a maximum of 200 records per batch. Each batch is a transaction. The Apex governor limits are reset for each transaction.

Interviewer: Can you write a batch class blueprint?

Interviewee:

```
global class batch implements Database.Batchable <sObject> {

    global (Database.QueryLocator or  Iterable<sobject>)
        start(Database.BatchableContext bc) {

            //query on object;
            //return Database.getQueryLocator(query);
```

```
    }

    global void execute(Database.batchableContext bc, List <SObject> scope) {

        //some processing.
    }

    global void finish(Database.BatchableContext bc) {
        //job such as sending email or calling another batch class

    }
}
```

Interviewer: What is the difference between Database.QueryLocator & Iterable<sobject> used in batch apex?

Interviewee: In Database.QueryLocator, the governor limit for the total number of records retrieved by SOQL queries is bypassed and you can query up to 50 million records.

In Iterable<sobject>, we can create a custom scope for processing, which would be not possible to create using SOQL where clauses. The governor limit for the total number of records retrieved by SOQL queries is still enforced. For example,

```
global class CustomIterable implements Iterator<account>{
    List<account> acc {get; set;}
    Integer i {get; set;}
    public CustomIterable(){
```

```
        acc = [SELECT Id, Name, NumberOfEmployees FROM Account WHERE Name = 'false'];
        i = 0;
    }
    global boolean hasNext(){
        if(i >= acc.size()) {
            return false;
        } else {
            return true;
        }
    }
    global Account next(){
        // 7 is an arbitrary constant in this example that represents the maximum size of the list.
        if(i == 7){return null;}
        i++;
        return acc[i-1];
    }
}
```

So we can return our custom scope by simply calling as below...

```
global Iterable<account> start(Database.batchableContext info){
      return new CustomIterable();
  }
```

Also, If the code accesses external objects and is used in batch Apex, use Iterable<sObject> else Database.QueryLocator.

Interviewer: Can you explain more about interable interface. What you mean by custom scope?

Interviewee: When you have a use case in which SQOL can’t be constructed, we can use custom code to create the scope of a batch.

For example, A batch accepts account records, where each account is having more than 2 contacts and each contact’s mailing address should not be the same.

Now, you can’t create a SOQL for this use case, so use the iterable interface, and create a list of accounts using queries and loops, and return the final processed account list, to the batch. This final list is called a custom scope.

Interviewer: If We use Itereable<sObject>, will We be able to process Batches in a specific order?

Interviewee: Batches of records tend to execute in the order in which they’re received from the start method. However, the order in which batches of records execute depends on various factors. The order of execution isn’t guaranteed. So even if you do ORDER BY, there is no guarantee of processing in the same order.

Interviewer: In a batch context, can I process Data in a specific order?

Interviewee: Yes, You can order the scope of that batch, however, you want and process it.

Interviewer: Can I query related records using Database.QueryLocator ?

Interviewee: Yes, You can do subquery for related records, but with a relationship subquery, the batch job processing becomes slower. A better strategy is to perform the subquery separately, from within the execute method, which allows the batch job to run using the faster.

Interviewer: Can I Use FOR UPDATE in SOQL using Database.QueryLocator?

Interviewee: No, We can't. It will through an exception stating that

"Locking is implied for each batch execution and therefore FOR UPDATE should not be specified"

The reason is, If we query the entire database of an org using SELECT.

FOR UPDATE, we would lock the entire database as long as the batch was active.

Interviewer: What is the state of batch apex?

Interviewee: Batch Apex is typically stateless. Each execution of a batch Apex job is considered a discrete transaction.

For example, a batch Apex job that contains 1,000 records and uses the default batch size is considered five transactions of 200 records each.

Interviewer: Let's say Record A has to be processed before Record B, but Record B came in the first Batch and Record A came in the second batch. The batch picks records which are unprocessed every time it runs. How will you control the processing Order?

Interviewee: The Processing order can't be controlled, but we can bypass the Record B processing before Record A. We can implement Database.STATEFUL and use one class variable to track whether Record A has processed or not. If not processed and Record B has come, don't process Record B. After all the execution completes, Record A has already been processed so Run the batch again, to process Record B.

Interviewer: What is Database.Stateful?

Interviewee: Database.Stateful is a Marker interface helps in maintaining state across transactions. When using Database.Stateful, only instance member variables retain their values between transactions.

Static member variables don't retain their values and are reset between transactions.

Maintaining state is useful for counting or summarizing records as they're processed.

For example, we'll be updating contact records in our batch job and want to keep track of the total records affected so we can include it in the notification email.

Interviewer: Is there any other interface that can be implemented by a Batch apex?

Interviewee: Database.AllowsCallouts is another Marker Interface, which is implemented to make HTTP callouts through Batch.

Interviewer: How to monitor Batch job?

Interviewee: To monitor the status of the jobs

1. From Setup, enter Jobs in Quick Find box
2. Select Apex Jobs

–> Go to Setup --->Apex Job page :

Interviewer: What is the Limit of Size of scope in a batch?

Interviewee: If the start method of the batch class returns a QueryLocator, the optional scope parameter of Database.executeBatch can have a maximum value of 2,000. If set to a higher value, Salesforce chunks the records returned by the QueryLocator into smaller batches of up to 2,000 records. If the start method of the batch class returns an iterable, the scope parameter value has no upper limit.

Interviewer: If a batch is having 200 records and 1 record fails what will happen?

Interviewee: If any record fails all 200 record will fail but next batch will get executed.

Interviewer: How to calculate the batch size if we are making any call out from the batch class execute method?

Interviewee: Basically in salesforce we have limitation of 100 call outs for a transaction. When it comes to batch class each execute method call will be considered as one transaction. So, in this case your batch size must be calculated in such way

Batch size = (No of callouts allowed for single transaction /total number of call outs for each record) - 1;

Batch size = (100 /total number of call outs for each record) - 1;

Interviewer: How many times the execute method will be executed to process the 1236 records.

Interviewee: It depends on your batch size what you have configured at the time of calling the batch class from schedule class.

Execution method count = Total No Of Records/Batch Size (Any decimal ceil it to upper value)

If you haven't set any batch size then :

1236 = 6.18 = 7 times execute method will be called

200

Interviewer: Can we call the batch into another batch apex?

Interviewee: Yes, we can call from the finish method.

Interviewer: Can we call batch apex into another batch in execute method?

Interviewee: Only in batch class finish method, We can call another batch class. If you will call another batch class from batch class execute and start method, then Salesforce will throw below runtime error.

System.AsyncException: Database.executeBatch cannot be called from a batch start, batch execute, or future method.

Interviewer: Why to call only from the finish method why not from execute?

Interviewee: Because the execute method will gets invoked multiple times based on the volume of the records and batch size. So, if your calling it from execute method then the chaining class will get called multiple times which is not an suggested way of doing.

We can also use the Queueable Apex for chaining of the jobs.

Interviewer: Let's say, we have run an apex batch to process 1000 records, and It is running with batch size 200. Now, while doing DML on 398th record, an error occurred, What will happen in that case?

Interviewee: In batches, If the first transaction succeeds but the second fails, the database updates made in the first transaction are not rolled back.

Since the batch size is 200, so the first batch will be processed completely and all data will be committed to DB. In seconds batch, if we are committing records using normal DML statements like insert, update than the whole batch will be rollbacked. So records 201 to 400 will not be processed.

If we use the Database DML operations like Database.insert with AllOrNone as false, then partial commit can happen and only 398th record will not be processed in that batch and total 199 records will be processed. Also, the other batch execution will not gonna be hampered.

Interviewer: Can I call any method from Batch Apex?

Interviewee: Methods declared as future can't be called from a batch Apex class.

Interviewer: Is there is any way through which we can call future method from batch apex?

Interviewee: As we know that a webservice can be called from batch class and webservice can call @future method. So in your batch class call web service and which can call your @future method. Also, you can call future methods from the finish method in the batch class.

Interviewer: How many can concurrent batch jobs be added to the queue?

Interviewee: At most, there can be 5 batch jobs queued at a time.

Interviewer: Can we call the batch apex from triggers in salesforce?

Interviewee: Yes, it is possible. We can call a batch apex from trigger but we should always keep in mind that we should not call batch apex from trigger each time as this will exceeds the governor limit this is because of the reason that we can only have 5 apex jobs queued or executing at a time.

Interviewer: Can we call webservice callout from batch apex?

Interviewee: To make a Webservice callout in batch Apex, we have to implement Database.AllowsCallouts interface.

Interviewer: How many times start, execute, finish methods will execute in batch apex?

Interviewee: Start method, finish method one time, execute method it depends on requirement. Based on the batch size and data retrieved in Start method.

Interviewer: What is the Batch executions limit per day?

Interviewee: The maximum number of batch executions is 250,000 per 24 hours.

Interviewer: How can we track the status of the current running batch job?

Interviewee: The job Id returned by the batchable context variable helps us in finding the status of a batch through AsyncApexJob object. For example,

```
global void finish(Database.BatchableContext info){

    // Get the ID of the AsyncApexJob representing this batch job
    // from Database.BatchableContext.
    // Query the AsyncApexJob object to retrieve the current job's information.

    AsyncApexJob a = [SELECT Id, Status, NumberOfErrors, JobItemsProcessed,
                      TotalJobItems, CreatedBy.Email FROM AsyncApexJob WHERE Id =
:info.getJobId()];
```

```
    // Send an email to the Apex job's submitter notifying of job completion.

    Messaging.SingleEmailMessage mail = new Messaging.SingleEmailMessage();

    String[] toAddresses = new String[] {a.CreatedBy.Email};
    mail.setToAddresses(toAddresses);
    mail.setSubject('Account and contact update' + a.Status);
    mail.setPlainTextBody

        ('The batch Apex job processed ' + a.TotalJobItems +
         ' batches with '+ a.NumberOfErrors + ' failures.'+successRecord+'successRecordids: '+
'failRecordids: '+ failRecord);
    Messaging.sendEmail(new Messaging.SingleEmailMessage[] { mail });
}
```

Interviewer: How can you stop a Batch job?

Interviewee: The Database.executeBatch and System.scheduleBatch method returns an ID that can be used in System.abortJob method.

Interviewer: How to stop all the running batches related to a particular batch classes before scheduling the same batch class from the schedule class.

Interviewee:

```
global class CaseCreationonSchedular implements Schedulable
{
   global void execute(SchedulableContext SC)
   {

      for(AsyncApexJob ap: [SELECT Id, ApexClass.Name,createddate, JobType, Status,
JobItemsProcessed, MethodName, ParentJobId
                            FROM AsyncApexJob
                            Where ParentJobId!=Null
                            AND JobType IN('BatchApex','BatchApexWorker')
                            And ApexClass.Name='YourBatchClassName'
                            And Status NOT IN('Completed','Aborted')])
      {
         System.abortJob(ap.ParentJobId);

      }
      YourBatchClassName cls = new YourBatchClassName();
      DataBase.executeBatch(cls);
   }
}
```

Interviewer: What is Apex Flex Queue?

Interviewee: At a time salesforce allows 5 batches to be running or to be in queued state. So, if you have consumed all these 5 limits and if system has received one or more batch execution request all these waiting batch will be stored in this Apex Flex Queue.

The maximum number of batch classes allowed in Apex Flex Queue for execution is 100.

Interviewer: Let's say, I have 150 Batch jobs to execute, Will I be able to queue them in one go?

Interviewee: Once you run Database.executeBatch, the Batch jobs will be placed in the Apex flex queue and its status becomes Holding. The Apex flex queue has the maximum number of 100 jobs, Database.executeBatch throws a LimitException and doesn't add the job to the queue. So atmost 100 jobs can be added in one go.

Also, if Apex flex queue is not enabled, the Job status becomes Queued, Since the concurrent limit of the queued or active batch is 5, so atmost 5 batch jobs can be added in one go.

Interviewer: Can I change the order of already queued Batch Jobs?

Interviewee: Only jobs having status as Holding can be moved. It can be done through UI of Apex Flex queue or we can use Apex Flex Queue Methods.

1

```
Boolean isSuccess = System.FlexQueue.moveBeforeJob(jobToMoveId, jobInQueueId);
```

Interviewer: How Can I schedule a Batch Apex?

Interviewee: Through System.scheduleBatch Method, we can schedule batch for once at a future time. This method returns the scheduled job ID also called CronTrigger ID.

1

2

3

4

```
String cronID = System.scheduleBatch(reassign, 'job example', 1);

CronTrigger ct = [SELECT Id, TimesTriggered, NextFireTime
                  FROM CronTrigger WHERE Id = :cronID];
```

Interviewer: Is it possible to do Synchronous Web service callouts from scheduled apex?

Interviewee: No, Synchronous Web service callouts are not supported by scheduled Apex. However, if your scheduled Apex executes a batch job, callouts are supported by the batch class.

Interviewer: Can we call the scheduler from the future method?

Interviewee: Yes..!! We can call the scheduler from the future method.

Interviewer: Can we modify the scheduler class or classes referenced by this scheduler class if any scheduled job is pending?

Interviewee: No, If there are one or more active scheduled jobs for an Apex class, you cannot update the class or any classes referenced by this class through the Salesforce user interface.

Interviewer: Can I call Queueable from a batch?

Interviewee: Yes, But you're limited to just one System.enqueueJob call per execute in the Database.Batchable class. Salesforce has imposed this limitation to prevent explosive execution.

Interviewer: How Can I Test a Batch Apex?

Interviewee: We can test it by calling the Batch job between Test.startTest() and Test.stopTest(). Using the Test methods startTest and stopTest around the executeBatch method to ensure that it finishes before continuing your test. All asynchronous calls made after the startTest method are collected by the system. When stopTest is executed, all asynchronous processes are run synchronously.

Also we can test only one execution of the execute method. So we need to set the scope as per the governor limits.

Interviewer: How many records we can insert while testing Batch Apex?

Interviewee: We have to make sure that the number of records inserted is less than or equal to the batch size of 200 because test methods can execute only one batch. We must also ensure that the Iterable returned by the start method matches the batch size.

Batch Apex Real Time Scenarios Based Questions

Batch Apex Scenario 1 :

Write a Batch apex class to automatically deactivate users when the user's Last Login Date is greater than 30 days.

Batch Apex Code: batchDeactivateUsers.apxc

```
global class batchDeactivateUsers implements Database.Batchable<sobject> {

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

      string query = 'SELECT Name, isActive, LastLoginDate FROM User WHERE
LastLoginDate < LAST_N_DAYS:30';
      return database.getQueryLocator(query);
   }

   global void execute(Database.BatchableContext bc, List<user> users){
```

```
        List<user> userlist = new List<user>();

        for(User u : users){
            u.IsActive = false;
            userlist.add(u);
        }

        update userlist;

    }

    global void finish(Database.BatchableContext bc){

    }

}
```

Batch Apex Scenario 2 :

Update account description, number of employees, contact last name using batch apex. Get the failure record ids in the email. Also, schedule the job for every Monday.

Batch Apex Code: batchUpdateAccountsContacts.apxc

```
global class batchUpdateAccountsContacts implements Database.Batchable
<sObject>,Database.Stateful,Schedulable {
    global batchUpdateAccountsContacts(){
    }
    Set<id> successRecord = new Set<id>();
```

```
    Set<id> failRecord = new Set<id>();


    global Database.QueryLocator start(Database.BatchableContext info){
        String SOQL='Select id, name, NumberOfEmployees, description,(select id, name from
contacts) from Account';
        return Database.getQueryLocator(SOQL);
    }
    global void execute(Database.BatchableContext info, List<account> scope){
        List<account> accsToUpdate = new List<account>();
        List<contact> contUpdate = new List<contact>();
        for(Account a : scope)
        {
            a.description ='Welcome to Learnfrenzy - Best Place to Learn';
            a.NumberOfEmployees = 70;
            accsToUpdate.add(a);
            for (Contact c:a.contacts){
                c.lastname = 'test+a';
                contUpdate.add(c);
            }
        }
        Database.SaveResult[] srList = Database.update(accsToUpdate, false);
        Database.SaveResult[] srList1 = Database.update(contUpdate, false);



        for (Database.SaveResult sr : srList) {
            if (sr.isSuccess()) {
                // Operation was successful, so get the ID of the record that was processed
                successRecord.add(sr.getId());
            }


            else {
```

```
            for(Database.Error err : sr.getErrors()) {
            }
            failRecord.add(sr.getId());
        }
    }

    for (Database.SaveResult sr : srList1) {
        if (sr.isSuccess()) {
            successRecord.add(sr.getId());
        }
        else {
            for(Database.Error err : sr.getErrors()) {
            }
            failRecord.add(sr.getId());
        }
    }



  }
  global void finish(Database.BatchableContext info){

    // Get the ID of the AsyncApexJob representing this batch job
    // from Database.BatchableContext.
    // Query the AsyncApexJob object to retrieve the current job's information.
    AsyncApexJob a = [SELECT Id, Status, NumberOfErrors, JobItemsProcessed,
                      TotalJobItems, CreatedBy.Email FROM AsyncApexJob WHERE Id =
:info.getJobId()];

    // Send an email to the Apex job's submitter notifying of job completion.
    Messaging.SingleEmailMessage mail = new Messaging.SingleEmailMessage();
```

```
        String[] toAddresses = new String[] {a.CreatedBy.Email};
            mail.setToAddresses(toAddresses);
        mail.setSubject('Account and contact update' + a.Status);
        mail.setPlainTextBody

            ('The batch Apex job processed ' + a.TotalJobItems +
             ' batches with '+ a.NumberOfErrors + ' failures.'+successRecord+'successRecordids: '+
'failRecordids: '+ failRecord);
        Messaging.sendEmail(new Messaging.SingleEmailMessage[] { mail });
    }




    global void execute(SchedulableContext SC){
        database.executeBatch(new batchUpdateAccountsContacts(),100);
        //for cron expression
        // String cronexpression = '0 0 0 ? * * *'
        // System.schedule('Testing', cronexpression, testobj);
    }

}
```

Running Batch Job

1. Go to developer console
2. Open execute anonymous window.
3. Run the below code

1

database.executeBatch(new batchUpdateAccountsContacts(),100);

–> Open Execute Anonymous Window :