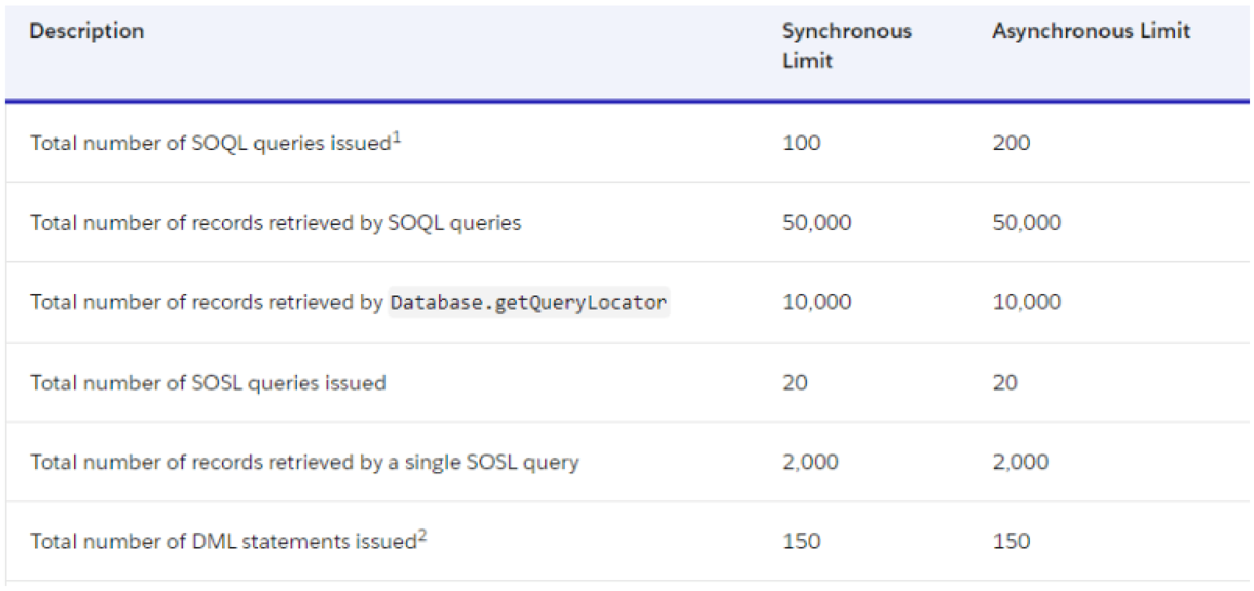
Comprehensive Guide of 150 Asynchronous Apex Interview Questions.

From Beginner to Advance

(Covering Detailed Scenario Questions)

1.What are the governor limit in Asynchronous Apex?



Description	Synchronous Limit	Asynchronous Limit
Total number of SOQL queries issued[1]	100	200
Total number of records retrieved by SOQL queries	50,000	50,000
Total number of records retrieved by `Database.getQueryLocator`	10,000	10,000
Total number of SOSL queries issued	20	20
Total number of records retrieved by a single SOSL query	2,000	2,000
Total number of DML statements issued[2]	150	150

2. Explain methods used in batch Apex?

Start: This method is called at the starting of a batch job to collect the data on which the batch job will be operating. It breaks the data or record into batches

Execute: This method executes after the Start method, and it does the actual processing for each batch, separately.

global void execute(Database.BatchableContext BC, list<sobject<) {}

Finish: This method will be called at last. Since this method is called in the end, it is responsible to perform post-processing operations such as sending an email. When this process is called all batches are already executed.

```
global void finish(Database.BatchableContext BC) {}
```

3. How to Invoke a Batch Class?

```
Database.executeBatch(new BatchApexExample(),100);
```

4. How to Monitoring Batch Apex?

To monitor or stop the execution of the batch Apex job, from Setup, enter Apex Jobs in the Quick Find box, then select Apex Jobs.

5. Explain Chaining in Batch Apex?

At maximum, only 5 Batch jobs can be chained to one another.

```
global database.querylocator start(Database.BatchableContext BC)
{
//start method logic here
}
global void execute(Database.BatchableContext BC, List<sObject> scope)
{
//start method logic here
}
global void finish(Database.BatchableContext BC)
{
//Batch Chaining Step Starts here
AccountBatch accBatch = new AccountBatch ();
Id batchProcessId = Database.executeBatch(accBatch);
//finish method logic here
}
```

6. What interface will you use for batch apex?

It is Database.Batchable interface

7.Why use Batch Apex in Salesforce instead of the normal Apex?

There are various reasons why Batch Apex is better than normal Apex.

- SOQL queries: Normal Apex uses 100 records per cycle to execute SOQL queries. Whereas, Batch Apex does the same in 200 records per cycle.
- Retrieval of SOQL queries: Normal Apex can retrieve 50,000 SOQL queries but, in Batch Apex, 50,000,000 SOQL queries can be retrieved.
- Heap size: Normal Apex has a heap size of 6 MB; whereas, Batch Apex has a heap size of 12 MB.

8. How many active batches(running parallel) can be allowed at a time?

Salesforce by default allows 5 active batches running at a time and other batches will be in queue for running

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

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.

Examples:

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

10.How many times the execute method will be executed to process the 1234 records.

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 – 1234/200 = 6.17 = 7 times execute method will be called

11.What is the maximum size of a batch that we can set up ?

2000

12.What is the minimum batch size we can set up is?

1

13.What is the default size of batch if we haven't configured at the time of execution?

200

14. What is Apex Flex Queue?

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.

15. What is the maximum number of batch classes allowed in Apex Flex Queue for execution?

100

16. Why to call only from the finish method why not from execute?

It is because the execute method will gets invoked multiple times based on the volume of the records and batch size.So,if you're calling it from execute method then the chaining class will get called multiple times which is not an suggested way of doing.

17.What is the difference between queryLocator object and Iterable used in batch apex?

QueryLocator object, the governor limit for the total number of records retrieved by SOQL queries is bypassed and you can query up to 50 million records. However, with an Iterable, the governor limit for the total number of records retrieved by SOQL queries is still enforced.

18. What is the state of batch apex?

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.

19.What is the use of Database.Stateful?

If you specify Database.Stateful in the class definition, you can maintain state across all transactions. When using Database.Stateful, only instance member variables retain their values 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.

20.When to use batch apex instead of Queueable Apex?

Only you should use Batch Apex if you have more than one batch of records. If you don't have enough records to run more than one batch, you should use Queueable Apex.

21.How to use HTTP Callouts in batch class?

To use HTTP Callouts in batch class we need to use Database.allowcallouts in interface.

22. If a batch is having 200 records and 1 record fails what will happen?

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

23. Can we call the batch into another batch apex?

Yes, we can call from the finish method.

24. Can we call batch apex into another batch in execute method?

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.

25.Can we call the batch apex from triggers in salesforce?

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.

26.How many times start,execute,finish methods will execute in batch apex?

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

27.Can we call the future method in batch class?

No,we can't call.

28.Can I call Queueable from a batch?

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.

29.How to test batch apex?

Code is run between test.startTest and test.stopTest. Any asynchronous code included within Test.startTest and Test.stopTest is executed synchronously after Test.stopTest.

30.How many records we can insert while testing batch apex?

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.

31. What is apex Flex Queue?
The Apex Flex queue enables you to submit up to 100 batch jobs for execution. Any jobs that are submitted for execution are in holding status and are placed in the Apex Flex queue. Up to 100 batch jobs can be in the holding status.

32. Can you change order of job in Apex Flex Queue?

Jobs are processed first-in first-out—in the order in which they're submitted. You can look at the current queue order and shuffle the queue, so that you could move an important job to the front, or less important ones to the back.

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

33. **Explain status of jobs in Apex Flex Queue?**
Holding : Job has been submitted and is held in the Apex flex queue until system resources become available to queue the job for processing.
Queued : Job is awaiting execution.
Preparing : The start method of the job has been invoked. This status can last a few minutes depending on the size of the batch of records.
Processing: Job is being processed.
Aborted : Job aborted by a user.
Completed : Job completed with or without failures.
Failed : Job experienced a system failure.

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

35. Can I Use FOR UPDATE in SOQL using Database.QueryLocator?
No, We can't. It will throw an exception stating that "Locking is implied for each batch execution and therefore FOR UPDATE should not be specified"

List<Account> accounts = [SELECT Id, Name FROM Account WHERE Name = 'SpecificAccountName' FOR UPDATE];

36.**Can I query related records using Database.QueryLocator?**
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 faster.

36. Can you write a batch class blueprint?

global class batchExample implements Database.Batchable<sObject> {

global (Database.QueryLocator | Iterable<sObject>) start(Database.BatchableContext bc) {

// collect the batches of records or objects to be passed to execute

}

global void execute(Database.BatchableContext bc, List<sObject> records){

// process each batch of records

}

global void finish(Database.BatchableContext bc){

// execute any post-processing operations

}

}

37.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 395th record, an error occurred, What will happen in that case?

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.

38. How can you stop a Batch job?
The Database.executeBatch and System.scheduleBatch method returns an ID that can be used in System.abortJob method.

39. Call we call future method in batch class?

Methods declared as future can't be called from Batch Apex class.

Batch class and future method are designed for a different kind of task. Batch Apex is like a heavy-duty used for big tasks (processing lots of data), while future methods are more like a screwdriver, good for smaller, quick fixes (simple, one-time operations- a method). They work differently, so they can't always be used together.

40.How to use Aggregate queries in Batch Apex

Aggregate queries don't work in Batch Apex because aggregate queries doesn't support queryMore(). They run into the error '**Aggregate query does not support queryMore(), use LIMIT to restrict the results to a single batch'**

41.**What is the difference between database.batchable & database.batchablecontext bc?**

Database.Batchable (Interface):

- Think of this as a blueprint that your Apex class needs to follow.
- When you implement Database.Batchable, you're telling Salesforce that your class is ready to be used for batch processing.

- It defines the methods your class must have, like start(), execute(), and finish().

Database.BatchableContext (Context Variable):

- It stores runtime information about the batch job, like the job ID and other details.
- You can use this to access and work with information related to the current batch job.

42.What if you change the name of Execute method to Execute1 in the batch class? Will the batch job still run?

Go ahead and change Execute to Excute1 and try saving the class.

Output

Class batchUpdateAccountsContacts must implement the method: void Database.Batchable<SObject>.execute(Database.BatchableContext, List<SObject>)

Finding

It won let you save the batch class as it says class must implement execute method.

43.Is there a way in which I can call a future method from a batch Job?

Calling a future method is not allowed in the Execute method, but a web service can be called. A web service can also call an @future method. So, we can define a web service having a future method invocation and call the web service from the execute method of Batch Job.

44.What is batchable context? why we need it?

1. database.batchable is an interface.
2. database.batchableContext is a context variable which store the runtime information eg jobId

It gives the context of the batch class. Represents the parameter type of a batch job method and contains the batch job ID.

45.Can I write method other than start, execute, finish?
No we cannot write other method. Whatever method is there is Database.batchable interface only that we can call.

46.Why batch class is global?
global makes a class usable by code outside of a managed package.

47.insert vs database.insert?
If we use the DML statement (insert), then in bulk operation if error occurs, the execution will stop and Apex code throws an error which can be handled in try catch block.
If DML database methods (Database.insert) used, then if error occurs the remaining records will be inserted / updated means partial DML operation will be done.

48.database.insert vs Stateful?
Database.insert – It show failure for records where complete batch is failed and wont pick the records where some records are passed for a particular batch and therefore we need to use database.stateful

49.Is finish asynchronous?
Finish method is sync because it does not go in queue. Only execute method is async.

Execute method: The execute method is asynchronous. It processes records in batches and may make use of the Salesforce job queue. Records are processed in the background.

Finish method: The finish method, on the other hand, is synchronous. It doesn’t go into a queue, and it runs immediately after the execute method has completed processing all records.

50.What we do to do callout from batch?
1. Use Database.allowcallout interface on class level
2. Do callout from execute method.

51.Can we do callout from start method?
We can do from start, but it will hit the limit so best is to do from execute.
The execute method processes records in batches, which means you can make callouts for each batch of records, allowing you to stay within Salesforce’s governor limits.

Additionally, if a callout fails in the execute method, you can implement error handling and retry logic for that specific batch of records, which is more challenging to do in the start method.

52.Can we call queuable from batch?
We can call because it is a job and can be queued. It can be called from finish.

53.Can I call a batch from another batch?
We can call from finish.

54.Can we get records without querying a batch?
We will do using iterator
from Batch A we will call batch B
In Batch B constructor will pass success records
In this case we dont need to query but we will use iterator

55. How to run Future Method?

If we want any method to run in future, put @future annotation. This way Salesforce understands that this method needs to run asynchronously.

```
@future

public static void clothesInLaundry(){

}
```

Void – As it runs in future, it does not return any value to you currently.

The specified parameters must be primitive data types, arrays of primitive data types, or collections of primitive data types; future methods can’t take objects as arguments.

56. What are Primitive Values and why to pass in future?

– Primitive data types, such as Integer, Boolean, or String, represent simple values with no internal complexity.

– These values are typically immutable, meaning they can’t change after they are assigned. For example, once you set an Integer to 5, it will always be 5.

– Because of their immutability and simplicity, Salesforce can reliably pass primitive values to a future method without concerns about changes occurring between the time of invocation and execution.

Objects and Complexity:

– Objects in Salesforce can be complex structures with various fields and relationships.

– An object's data can change at any time due to user interactions or other processes. For example, a record's fields can be updated, related records can be created or deleted, and triggers can fire, all of which might modify the object's state.

– When you pass an object to a future method, it's possible that the object's data may change before the future method runs. This can lead to unexpected behavior or inconsistencies if the future method relies on specific data within the object.

In essence, allowing objects as arguments in future methods would introduce potential data integrity and consistency issues**. By restricting future methods to primitive data types, Salesforce ensures that the data passed to future methods is stable and won't change unexpectedly** between the time of invocation and execution.

57.How to call future method?

ClassName.clothesInLaundry();

58.Scenarios when we use future method?

Scenario 1: Whenever we want our code to be run in background.

Scenario 2: You must have faced DML exception. This exception occurs when we insert setup and non-setup objects in one transaction.

Setup: User, Group, Profile,Layout, Email Template

Non Setup: All other standard and custom objects (Account, Custom Objects)

In same context, we cannot insert User/Profile and Account in same trasanction. Future method helps in this situation.

Scenario 3: We cannot perform callout from trigger. Use future method to put callout in future method to make a callout.

59.What are Limitations of Future method?

1. You cannot call one future method from another method
2. Can have only primate as parameters
3. Order of Invocation is not respected

60.Can you write the syntax of a future method?

Methods with the future annotation must be static methods and can only return a void type.

```
global class FutureClass

{

@future

public static void myFutureMethod()

{

// Perform some operations

}

}
```

61.What could be the workaround for sobject types?

To work with sObjects, pass the sObject ID instead (or collection of IDs) and use the ID to perform a query for the most up-to-date record.

```
global class FutureMethodRecordProcessing

{

@future

public static void processRecords(List<ID> recordIds)

{
```

```
// Get those records based on the IDs
// Process records
}
}
```

62.How can I perform Callouts from Future methods?

We need to add a parameter **callout=true** in **@future.**

```
global class FutureMethodExample
{
@future(callout=true)
public static void doCallouts(String name)
{
// Perform a callout to an external service
}
}
```

63.Can I write a future call in Trigger?

Yes, you can.

64. How can avoid this Exception condition, without using try-catch?

We can update the trigger logic to leverage
the System.isFuture() and System.isBatch() calls so that the future method invocation is not made if the current execution context is future or batch.

65.How Many Future methods can be defined in a Class?

Any number of. There are no restrictions as such.

66.If I want to call a future method from a future method, what could be the solution?

Workaround could be calling **a web service** that has future invocation.

66.How Future method helps in avoiding Mixed DML errors?

We can shift DML operations of a particular kind in the Future scope. Since both the DML operations are isolated from each other, the transaction doesn’t fail.

For example:

```
public class MixedDMLFuture {
public static void useFutureMethod() {
// First DML operation
Account a = new Account(Name=’Acme’);
insert a;
// This next operation (insert a user with a role)
// can’t be mixed with the previous insert unless
// it is within a future method.
// Call future method to insert a user with a role.
Util.insertUserWithRole(
‘abc.com’, ‘Test’,
‘acb.com’, ‘Test2’);
}
}
```

67.Once I call a future method, how can I trace its execution?

Salesforce uses a queue-based framework to handle asynchronous processes from such sources as future methods and batch Apex. So, we can check the apex jobs if it has run or not.

However, **future methods don’t return an ID,** so We can’t trace it directly. We can use another filter such as MethodName, or JobType, to find the required job.

68.How to test a future method?

To test methods defined with the future annotation, call the class containing the method in a startTest(), stopTest() code block. All asynchronous calls made after the startTest method are collected by the system. When stopTest is executed, all asynchronous processes are run synchronously.

69.Is it possible to call future method from apex scheduler or not?

Yes, it is possible to call future method from apex scheduler

70.**Why future method is static and void?**

Future methods will run in the future. You don’t want your synchronous code waiting an unknown period for an asynchronous bit of code to finish working. By only returning void, you can’t have code that waits for a result.

The future method is static so that variables with this method is associated to the class and not the instance and **you can access them without instantiating the class.**

71.**From which places we can call future method?**

- Trigger
- Apex Class
- Schedulable Class

72.Can we pass the wrapper to future method?

You can pass wrapper, but for that, you'll need to serialize/deserialize that parameter. You can convert the Wrapper to a String which is a primitive.

Once converted into a String you can them pass that string as a parameter to the future method in consideration.

73. **What is the advantage of Queueable Apex over future methods?**

Queueable Apex allows you to submit jobs for asynchronous processing like future methods.

- Non-primitive types: Your Queueable class can contain member variables of non-primitive data types, such as sObjects or custom Apex types.
- Monitoring: When you submit your job by invoking the System.enqueueJob method, the method returns the ID of the AsyncApexJob record. You can use this ID to identify your job and monitor its progress, either through the Salesforce user interface in the Apex Jobs page, or programmatically by querying your record from AsyncApexJob.

// You can use jobId to monitor the progress of your job

AsyncApexJob jobInfo = [SELECT Id, Status, NumberOfErrors

FROM AsyncApexJob

WHERE Id = :jobId];

- Chaining jobs: You can chain one job to another job by starting a second job from a running job. Chaining jobs is useful if you need to do some sequential processing.

74.What is the interface used for Queueable Apex?

Queueable interface

75.What are methods used in Queueable Apex Class?

Execute method.

76.How many jobs can you chain from executing a job?

You can add only one job from an executing job, which means that only one child's job can exist for each parent job.

76.How many jobs can you queue in a single transaction?

You can add up to 50 jobs to the queue with System.enqueueJob in a single transaction.

77.How can I use this Job Id to trace the Job?

Just perform a SOQL query on AsyncApexJob by filtering on the job ID.

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

78.Can I do callouts from a Queueable Job?

Yes, you have to implement the Database.AllowsCallouts interface to do callouts from Queueable Jobs.

78.If I have written more than one System.enqueueJob call, what will happen?

System will throw LimitException stating "Too many queueable jobs added to the queue: N"

79.What are the Limitations of Queueable Jobs?

50 jobs can be added to the queue with System.enqueueJob() method in a single transaction

Maximum depth of chain job is 5 i.e., 4 child jobs and initial parent jobs for Developer and Trail organizations but there is no limit in other editions

80.Can you write a blueprint of Queueable Job?

Create a class, implement the Queueable interface, and override the execute method.

```
public class QueueableApexExample implements Queueable {

public void execute(QueueableContext context) {

//some process

}

}
```

81. What is QueueableContext?

It is an interface that is implemented internally by Apex, and contains the job ID. Once you queue for the Queueable Job, the Job Id will be returned to you, by apex through QueueableContext's getJobId() method**.**

82.How can I queue Queueable Job?

Using System.enqueueJob Method.

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

83.I have 200 records to be processed using Queueable Apex, How Can I divide the execution Context for every 100 records?

Similar to future jobs, queueable jobs don't process batches, so you can't divide the execution Context. It will process all 200 records, in a single execution Context.

84.**Can I chain a job that has implemented *allowsCallouts* from a Job that doesn't have?**

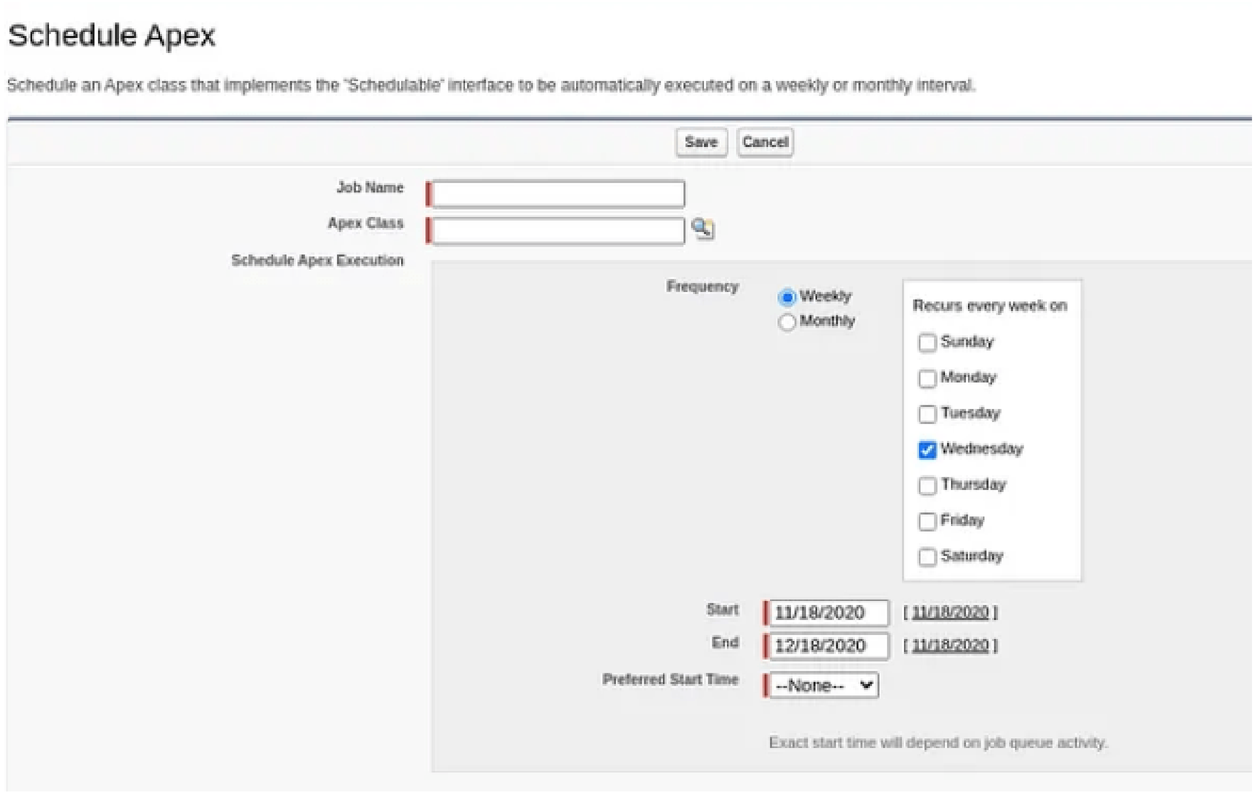
Yes**,** callouts are also allowed in chained queueable jobs.

85.How to test Chaining?

You can't chain queueable jobs in an Apex test. So you have to write separate test cases for each chained queueable job. Also, while chaining the jobs, add a check of Test.isRunningTest() before calling the enqueueJob.

86.What is an apex Scheduler?

The Apex Scheduler lets you delay execution so that you can run Apex classes at a specified time. This is ideal for daily or weekly maintenance tasks using Batch Apex.


Schedule Apex

Schedule an Apex class that implements the 'Schedulable' interface to be automatically executed on a weekly or monthly interval.

Job Name	
Apex Class	
Schedule Apex Execution	
Frequency	[✓ Weekly] Monthly
Recurs every week on	Sunday, Monday, Tuesday, [✓ Wednesday], Thursday, Friday, Saturday
Start	11/18/2020 [11/18/2020]
End	12/18/2020 [11/18/2020]
Preferred Start Time	--None--

Exact start time will depend on job queue activity.

87.What is the interface used for Schedulable Apex?

Schedulable interface

88.What are the methods used with a Schedulable interface?

The only method this interface contains is the execute method.

89.What is the parameter of the execute method ?

The parameter of this method is a SchedulableContext object.

90.What happens after the class is scheduled?

After a class has been scheduled, a CronTrigger object is created that represents the scheduled job. It provides a getTriggerId method that returns the ID of a CronTrigger API object.

91.What are the arguments of the System.Schedule method?

The System.Schedule method takes three arguments:

- Name for the job

- CRON expression used to represent the time and date the job is scheduled to run
- Name of the class.

92.How to Test Scheduled Apex?

To test Scheduled Apex you must ensure that the scheduled job is finished before testing against the results. To do this, use startTest and stopTest around the System.schedule method, to ensure processing finishes before continuing your test.

93. What is the governor limit of Scheduled Apex?

You can only have 100 scheduled Apex jobs at one time and there are a maximum number of scheduled Apex executions per a 24-hour period.

93.Can we make a callout from Scheduled Apex?

Synchronous Web service callouts are **not supported** from scheduled Apex.

94.How to monitor Scheduled Jobs?

After an Apex job has been scheduled, you can obtain more information about it by running a SOQL query on CronTrigger.

CronTrigger job = [SELECT Id, CronJobDetail.Id, CronJobDetail.Name, CronJobDetail.JobType FROM CronTrigger ORDER BY CreatedDate DESC LIMIT 1];

95. Explain code to schedule batch Apex to run at regular intervals?

global class SampleBatchScheduler implements Schedulable {

// Execute at regular intervals

global void execute(SchedulableContext ctx){

String soql = ‘SELECT Id, Name FROM Account’;

SampleBatch batch = new SampleBatch(soql);

Database.executebatch(batch, 200);

}

}

96. What is return type of system.schedule?

System.schedule method returns the job ID in string format.

String jobID = system.schedule('Merge Job', sch, m);

97.How to get count of Apex Scheduled Job programmatically?

You can programmatically query the CronTrigger and CronJobDetail objects to get the count of Apex scheduled jobs.

98.If there are one or more active scheduled jobs for an Apex class, can you update the class, or any classes referenced in Salesforce UI?

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. However, you can enable deployments to update the class with active scheduled jobs by using the Metadata API

99. Does Apex Scheduler run in system mode?

The scheduler runs as system—all classes are executed, whether or not the user has permission to execute the class..

100.Callout is not supported in Scheduled Apex so what is the alternative?

Synchronous Web service callouts are not supported from scheduled Apex. To be able to make callouts, make an asynchronous callout by placing the callout in a method annotated with @future(callout=true) and call this method from scheduled Apex. However, if your scheduled Apex executes a batch job, callouts are supported from the batch class.

101.What are limitations of Scheduled Apex?

1. We can schedule only 100 jobs at a time.
2. Max no. of apex schedule jobs in 24 hours is 2,50,000 number of jobs (can change with salesforce updates).

102.How many callouts we can make from batch?

1. Number of Callouts per Batch Execution: Salesforce imposes a limit of 100 callouts to external services per Apex transaction. In the context of batch Apex, each execution of the execute method counts as a separate transaction. Therefore, each execution can have up to 100 callouts.

2. Handling Large Data Sets*: If your batch job needs to process more than 100 callouts, you need to design your execute method to handle the data in chunks, ensuring that each chunk does not exceed the limit of 100 callouts.

103. Consider a scenario of you need to make 200 callouts in a Salesforce batch Apex job?

We have to handle it in such a way that we don't exceed the Salesforce governor limit of 100 callouts per execution. This requires splitting the callouts across multiple executions. Here's a way to approach it:

1. **Use a Counter to Track Callouts**: Track the number of callouts made and stop the execution once you hit the limit.
2. **Chain Batch Jobs**: If we have more callouts to make than the limit allows, chain another batch job from the finish method of the current batch job.

```
global void finish(Database.BatchableContext BC) {
// Check if there are remaining records to process
if ([SELECT COUNT() FROM MyObject WHERE Status != 'Processed'] > 0)
{
// Chain the next batch job
 Database.executeBatch(new MyBatchClass(), 100);
// Adjust batch size as needed
} }
```

1. After the batch job completes, the finish method checks if there are more records left to process. If there are, it chains another batch job to continue processing.

104. How to pass parameters in queueable?

In Salesforce, when using the Queueabl` interface, you can pass parameters to the queueable class constructor.

Sample Code

```
public class MyQueueableClass implements Queueable {
    private String recordId;
    private String customMessage;


    // Constructor to pass parameters
    public MyQueueableClass(String recordId, String customMessage) {
        this.recordId = recordId;
        this.customMessage = customMessage;
    }
    public void execute(QueueableContext context) {
        // Use the parameters in the job logic
        Account acc = [SELECT Id, Name FROM Account WHERE Id =
:recordId];
        acc.Description = customMessage;
        update acc;
    }
}
```

105. How to pass parameters in future method?

There are some limitations when running future method, including the fact that only primitive data types can be passed as parameters.

Sample Code

```
public class FutureExample {
    // @future method with parameters
    @future
    public static void updateAccountDescription(String accountId, String description) {
        Account acc = [SELECT Id, Name FROM Account WHERE Id = :accountId LIMIT 1];
        acc.Description = description;
        update acc;
    }
}
```

107. How to schedule a queuable job?

To schedule a Queueable job in Salesforce, you need to use a combination of the Queueable interface and the System.schedule method. Here’s a simple guide on how to achieve this:

1. Implementing a Queueable Class

```
public class ExampleQueueableJob implements Queueable {
    public void execute(QueueableContext context) {
        // Your job logic here
        System.debug('Queueable job is running');
    }
```

}

2. Scheduling the Queueable Job

To schedule this Queueable job, use the **System.schedule method**. This method takes three parameters:

- The name of the scheduled job.

- A CRON expression defining the schedule.

- The instance of the Queueable class to be executed.

108. Give me a real time scenario where you have used batch apex?

Company needs to synchronize opportunity data from Salesforce with an external system ‘Marketo’, every night. This involves sending information about opportunities that have been updated during the day, including the opportunity name, amount, and stage. The batch job must accumulate results and handle the callouts in batches, ensuring all opportunities are processed efficiently.

Batch Job Flow:

1. **Start Method:**
   - Query opportunities that have been updated in the last 24 hours.
   - Store these opportunities in a list to be processed.
2. **Execute Method:**
   - Process each batch of opportunities.
   - For each opportunity, make a callout to the external system to update the record.
   - Track successful and failed callouts.
3. **Finish Method:**
   - Send a summary email report to the system administrator, detailing the number of successful and failed updates.

Governor Limits Considerations

Callout Limits: Salesforce imposes a limit of 100 callouts per transaction. This means that within a single execution of the execute method in a batch job, you cannot exceed 100 callouts.

Heap Size Limit: The heap size in a batch job can quickly be exceeded if handling large volumes of data.

109. When to use Batch apex instead of Data Loader?

1.Data Loader is primarily a tool for manual data import/export and requires human intervention to run the process. For a nightly synchronization task, automation is crucial to ensure timely and consistent updates.

While Data Loader can be scheduled using external tools or scripts, it doesn't provide the same level of integration and automation as a batch job within Salesforce. Managing and automating the scheduling, error handling, and notifications are more complex and less integrated.

110. When to use Batch Apex instead of Queueable Apex?

1. Queuable Apex is suitable for smaller, less complex asynchronous processing tasks. It doesn't support processing large datasets as efficiently as a batch job, which can handle millions of records by processing them in manageable chunks (batches).

2. Queueable Apex has similar governor limits to synchronous Apex in terms of SOQL queries, DML operations, and heap size, which can be restrictive when dealing with large volumes of data. A batch job, on the other hand, allows higher limits for certain resources per transaction.

111.Give a Scenario where you used Queueable Apex?

Company needs to synchronize order data from Salesforce to an external logistics system, 'ShipStation', in real-time as orders are placed or updated. The goal is to ensure that the logistics system is always up-to-date with the latest order information, including order items, quantities, and shipping addresses.

1. Trigger on Order Object:

   - A trigger on the Order object is used to detect changes such as creation or updates.

   - The trigger calls a Queueable Apex class to handle the integration.

2. Queueable Apex Execution:

   - The Queueable Apex class processes each order record, preparing the necessary data for the external API.

   - For each order, the class makes an API callout to 'ShipStation' to update the order details.

Now you might be thinking why we will use Queueable Apex Instead of Batch or Future Apex here?

- **Queueable Apex**: It allows near real-time processing since it can be triggered immediately after a database event (like order creation or update) occurs.
- **Batch Apex**: Designed for handling large volumes of data asynchronously, Batch Apex operates on scheduled intervals (e.g., nightly). This isn't suitable for real-time updates
- **Future Methods**: While future methods can be used for asynchronous processing, they do not support job chaining or more complex operations that Queueable Apex allows. They are also limited in the number of concurrent calls (50 calls per org at a time) Future methods cannot maintain state across different method calls,

unlike Queueable or Batch Apex where stateful processing is possible.

112. Difference Between Iterable and Database.QueryLocator in Salesforce?

Iterable is Suitable for scenarios where data comes from various sources, including non-Salesforce databases, external systems, or complex collections in memory. It offers flexibility in defining custom logic for data retrieval. It provides more control over the batch size and is limited to processing up to 50,000 records.

global Iterable<Account> start(Database.BatchableContext BC) {accounts = [SELECT Id, Name FROM Account WHERE CreatedDate = LAST_N_DAYS:30];

Database.QueryLocator is Ideal for handling large volumes of Salesforce records, particularly when performing SOQL queries. It can process upto 50 million records.

global Database.QueryLocator start(Database.BatchableContext BC) { String query = 'SELECT Id, Name FROM Account'; return Database.getQueryLocator(query); }

113. How do you schedule a Batch Apex job?

A Batch Apex job can be scheduled using the System.scheduleBatch method in Apex code or through the Salesforce user interface by navigating to the Scheduled Jobs section. The System.scheduleBatch method requires a cron expression to define the schedule.

114. What are the governor limits for Batch Apex?

  - Maximum number of batch jobs in the Apex flex queue: 100
  - Maximum number of batch jobs queued or active: 5
  - Maximum number of records processed per batch execution: 50,000
  - Maximum batch size: 2,000 records (default is 200)
  - Maximum number of callouts per transaction: 100
  - Maximum execution time for each transaction: 60 seconds

115. What are some real-world scenarios where you might use Batch Apex?

- **Data Synchronization**: Synchronizing large datasets between Salesforce and an external system, such as Marketo or SAP.
- **Data Cleanup**: Periodically cleaning up old records or performing data transformations on a large scale.
- **Mass Updates**: Updating millions of records in response to business process changes or data migrations.

116. Can we change the order of already queued batch jobs?

Yes, the order of batch jobs in the Apex Flex Queue can be changed, but only for jobs with the status "Holding". This can be done either through the UI of the Apex Flex Queue or using the Apex Flex Queue methods.

117.What is Database.RaisesPlatformEvents in Batch Apex?

Database.RaisesPlatformEvents is used to publish platform events from within a batch class. This allows batch jobs to trigger platform events as part of their execution, which can then be handled asynchronously by subscribers to those events.

118. What is the significance of the scope parameter in Batch Apex?

The scope parameter in Batch Apex defines the number of records that are passed to the execute method at one time. It helps in managing governor limits by breaking down large datasets into smaller, more manageable chunks for processing. The maximum value for the scope parameter is 2,000 records.

119. How to handle errors in Batch Apex?

Errors can be handled in Batch Apex by implementing try-catch blocks within the execute method. Additionally, you can use the Database.saveResult to track and log any errors encountered during DML operations within the batch job.

120. Can we call Schedulable Apex from Future Method?

It is possible to call Schedulable Apex from a Future Method, and you can schedule up to 100 jobs.

121. What happens if more than 50 future method calls are made in a transaction?

Salesforce enforces a limit of 50 future method invocations per transaction, beyond which an exception is thrown.

122. **Can you call a Batch Apex job from a Future Method?**

No, Batch Apex cannot be called from a Future Method due to restrictions on mixing asynchronous operations in Salesforce.

123. Can you call Queueable Apex from a Future Method?

Yes, but you are limited to enqueuing only one Queueable job per future method invocation.

124 .How to handle limitations in Future Methods regarding passing data:

Future Methods cannot directly pass complex data types like SObjects. Instead, pass IDs or primitive types and query the necessary data within the Future Method to ensure accuracy.

125. **Why is it not recommended to do heavy processing in Future Methods?**

Future Methods are designed for lightweight, asynchronous operations. Heavy processing can lead to longer execution times and potentially exhaust governor limits, for which Batch Apex is more suitable.

126. Give Scenarios where Future Methods are preferred over Batch Apex?

Future Methods are preferred for making asynchronous callouts, handling isolated tasks that do not require bulk processing, and when avoiding mixed DML errors.

127. What are the governor limits for Future Methods in Salesforce?

Governor limits for Future Methods include a maximum of 50 future calls per transaction and up to 250,000 future invocations per 24-hour period per Salesforce org.

128. You have a requirement to update the status of 100,000 account records based on certain conditions. Due to the large data volume, synchronous processing is not feasible. Which asynchronous method would you use and why?

For handling large volumes of data, Batch Apex is most suitable because it allows processing in smaller chunks (batches), avoids governor limits, and can handle up to 50 million records. It also provides methods to track the status and results of batch jobs.

129. Your Salesforce instance needs to send data to an external system (such as a third-party API) whenever an opportunity is closed. This system has varying response times, and the callout should not block other operations. Which asynchronous method should you use?

Future Methods or Queueable Apex should be used for this scenario. Future Methods are simpler but limited, while Queueable Apex allows for more complex logic, such as handling callouts, and offers better error handling and chaining capabilities.

130. Your organization requires a weekly cleanup job that deletes obsolete records from a custom object and archives them in an external system. What asynchronous Apex approach would you choose?

Schedulable Apex is appropriate here as it allows for scheduling jobs to run at specific intervals (weekly, in this case). The System.schedule method or Apex Scheduler can be used to automate the job execution

131. You need to deactivate user accounts and reassign their active cases to a new owner. These operations involve setup and non-setup objects, which could lead to a mixed DML error. How would you handle this scenario?

To avoid mixed DML errors, you can use Future Methods. The user deactivation can be handled asynchronously using a future method, while the reassignment of cases can be done in the synchronous part of the transaction.

132. Your company requires real-time synchronization of product inventory data between Salesforce and an external warehouse management system. The data needs to be updated as soon as changes are made in Salesforce. Which asynchronous Apex approach would you use?

For real-time synchronization, Platform Events or Queueable Apex are suitable. Platform Events can provide near real-time updates by publishing and subscribing to event messages, while Queueable Apex allows for immediate processing of inventory updates and can handle complex logic or chaining.

133. A new data integration project requires the insertion of large datasets into Salesforce on a nightly basis, exceeding the usual DML governor limits. How would you implement this using asynchronous Apex?

Batch Apex should be used for inserting large datasets as it allows breaking down the data into manageable chunks. The start method can query the data, execute can handle the insertion in batches, and finish be used for logging or sending notifications.

134. Your application requires executing complex business logic that involves multiple DML operations and potential callouts, but these need to happen sequentially to maintain data integrity. What approach should you take?

Queueable Apex is ideal for this scenario due to its support for complex business logic and sequential processing through job chaining. This allows each step of the process to complete before the next begins, maintaining data integrity.

135. A Salesforce organization has a process where users are created and assigned to specific roles. There is also a need to update related account records when a new user is assigned. Describe how you would use Future Methods to handle this scenario while avoiding mixed DML errors?

To handle this scenario without encountering mixed DML errors, which occur when DML operations on setup and non-setup objects are mixed in the same transaction, Future Methods can be used.

First, perform the DML operation on the setup object (user and role assignments) synchronously.

Then, use a Future Method to handle the DML operation on the related account records asynchronously.

This separation ensures that the two different types of DML operations do not conflict. When using Future Methods, only pass the IDs of the records, not the entire objects.

136. Your company is migrating data from a legacy system into Salesforce. The data includes over 10 million records, which need to be cleansed and transformed during the import process. Explain how Batch Apex would be implemented in this scenario?

Batch Apex is ideal for processing large datasets in Salesforce. To implement this:

1. Start Method: Use Database.QueryLocator to efficiently query the data to be processed, as it supports handling up to 50 million records.

2. Execute Method: Ensure that the batch size is set appropriately (up to 2000 records).

3. Finish Method:*Finalize any post-processing tasks, such as sending summary emails or updating a status field.

Use Database.Stateful if you need to maintain state across transaction boundaries, for example, to track the total number of records processed.

137. A process requires calculating commissions for sales reps, updating records, and sending notifications sequentially. Explain how Queueable Apex can be utilized for this scenario?

Queueable Apex is suitable for sequential processing due to its ability to chain jobs. In this scenario:

1. First Queueable Job: Calculate the commissions for sales reps and update their records.

2. Chaining Jobs: After the first job completes, chain another Queueable job to send notifications based on the updated records.

138. The marketing department requires a weekly report on campaign performance metrics. How would you automate this?

To automate the weekly report generation:

1. Schedulable Apex Class: Implement a class that implements the Schedulable interface and the execute method. In this method, include the logic to query campaign performance data, aggregate metrics, and format the report.

2. Scheduling the Job: Use System.schedule to schedule the class to run at a specific time weekly, such as every Monday at 8 AM.

3. Sending the Report: After generating the report, send it via email to the marketing team or store it in a designated location.

139. A company needs to process customer data in a series of steps: validating the data, calculating customer scores, and sending notifications. Each step depends on the completion of the previous one. How would you implement this using Queueable Apex?

Queueable Apex is ideal for this scenario because it supports job chaining, which allows one job to enqueue another upon completion.

1. First Job (Data Validation): Implement a Queueable Apex class for data validation. At the end of the execute method, enqueue the next job.

2. Second Job (Score Calculation): The next Queueable Apex job calculates the customer scores. This job is enqueued at the end of the first job. Once done, it enqueues the final job.

3. Third Job (Notification Sending): The final Queueable job sends notifications based on the results of the score calculations.

Sample Code:

```
public class ValidateDataJob implements Queueable {
    public void execute(QueueableContext context) {
        // Data validation logic
        // Chain the next job
        System.enqueueJob(new CalculateScoreJob());
    }
}

public class CalculateScoreJob implements Queueable {
    public void execute(QueueableContext context) {
        // Score calculation logic
        // Chain the final job
        System.enqueueJob(new SendNotificationJob());
    }
}
public class SendNotificationJob implements Queueable {
    public void execute(QueueableContext context) {
        // Notification logic
    }
}
```

140. After processing records in a Batch Apex job, there is a need to perform a final aggregation step, but only for records that were successfully processed. How can you use Queueable Apex to handle this?

To handle post-processing using Queueable Apex from a Batch Apex job:

1. Batch Job: The execute method processes records in batches. Track processed records that require further aggregation.

2. Finish Method: In the finish method of the Batch Apex, enqueue a Queueable job to handle the final aggregation step based on the records processed during the batch job.

```
    public void finish(Database.BatchableContext bc) {
        // Enqueue Queueable job for aggregation
        System.enqueueJob(new AggregationJob(processedRecordIds));
    }
}
public class AggregationJob implements Queueable {
    private List<Id> recordIds;

    public AggregationJob(List<Id> recordIds) {
        this.recordIds = recordIds;
    }

    public void execute(QueueableContext context) {
        // Aggregation logic
    }
}
```

141. A system processes two distinct sets of data in batches: Orders and Payments. Once the Orders batch job completes, a Payments batch job should run. How can this be implemented?

Chaining Batch Apex jobs should be done in the finish method to prevent hitting governor limits and to ensure that jobs run sequentially.

```
    public void finish(Database.BatchableContext bc) {
        // Chain the next batch job
        Database.executeBatch(new PaymentBatchJob(), 200);
    }
}
```

142. A future method is used to update records asynchronously, but additional processing is required afterward. How can Queueable Apex be used in conjunction with Future Methods?

Future methods can invoke Queueable Apex, allowing for extended processing. This might be useful if you want to process a large number of records or chain further actions.

```
public class MyFutureClass {
    @future
    public static void myFutureMethod(Set<Id> recordIds) {
        // Perform initial updates
        // Queue further processing
        System.enqueueJob(new MyQueueableJob(recordIds));
    }
}


public class MyQueueableJob implements Queueable {
    private Set<Id> recordIds;
    public MyQueueableJob(Set<Id> recordIds) {
        this.recordIds = recordIds;
    }

    public void execute(QueueableContext context) {
        // Further processing logic
    }
}
```

143. A company requires a daily refresh of certain data records at midnight. How can this be scheduled using Queueable Apex?

Use the Schedulable Apex interface to schedule a job that enqueues a Queueable Apex job.

```
public class NightlyJobScheduler implements Schedulable {
    public void execute(SchedulableContext sc) {
        // Enqueue the Queueable job
        System.enqueueJob(new NightlyDataRefreshJob());
    }
}
```

Queueable Apex

```
public class NightlyDataRefreshJob implements Queueable {
    public void execute(QueueableContext context) {
        // Data refresh logic
    }
}
```

Scheduling the Job:

```
String cronExp = '0 0 0 * * ?'; // Midnight daily
System.schedule('Nightly Data Refresh', cronExp, new
NightlyJobScheduler());
```

144. A company needs to send notifications to customers at specific times based on their subscription plans. How to implement this scenario?

Use Schedulable Apex to schedule the initial job that calculates the exact times for notifications and Queueable Apex for sending the notifications

1. Schedulable Apex for Scheduling:

   - This class runs daily, calculates when notifications should be sent, and enqueues Queueable jobs for those specific times.

2. Queueable Apex for Notifications

   - These jobs handle the actual sending of notifications, which might involve interacting with external systems or services.

```
public class NotificationScheduler implements Schedulable {
    public void execute(SchedulableContext sc) {
        // Logic to determine notification times
        List<DateTime> notificationTimes = calculateNotificationTimes();

        // Schedule Queueable jobs for each notification time
        for (DateTime notificationTime : notificationTimes) {
            System.schedule('Send Notification', notificationTime.format('ss mm
HH dd MM ? yyyy'),
                    new SendNotificationJob());
        }
    }

    private List<DateTime> calculateNotificationTimes() {
        // Logic to calculate times
        return new List<DateTime>(); // Placeholder logic
    }
}

public class SendNotificationJob implements Queueable {
    public void execute(QueueableContext context) {
        // Logic to send notifications
    }
}
```

145. A system requires real-time updates to Salesforce records based on data changes in an external system. However, these updates should not delay the main transaction flow. How can this be achieved using Future Methods?

Future methods are well-suited for this scenario because they allow operations like callouts to be performed asynchronously, without impacting the main transaction flow.

1. Trigger to Invoke Future Method:

A trigger on the relevant object (e.g., Account) detects changes and invokes a future method to handle the external callout.

2. Future Method for Integration:The future method makes a callout to the external system, retrieves the necessary data, and updates the Salesforce record asynchronously.

```
public class AccountTriggerHandler {

    @future(callout=true)

    public static void updateFromExternalSystem(Set<Id> accountIds) {

        // Callout logic to external system

        for (Id accountId : accountIds) {

            // Perform callout and update Account records

        }

    }

}
```

Considerations: Future methods do not return values and are not guaranteed to execute immediately. They are also subject to certain limits, such as the number of future calls per 24-hour period.

146. A system requires weekly cleanup of old or irrelevant records to maintain data quality. The cleanup process involves checking multiple conditions and potentially deleting records. How can this be automated using Schedulable and Batch Apex?

Schedulable Apex can be used to schedule the job weekly, and Batch Apex can handle the actual cleanup process, leveraging its ability to process large data volumes.

1. Schedulable Apex for Scheduling: Schedules the Batch Apex job to run weekly.

2. Batch Apex for Data Cleanup: Queries records based on specified conditions and processes them in manageable batches, performing deletions or updates as needed.

Sample Code

```
public class DataCleanupScheduler implements Schedulable {
    public void execute(SchedulableContext sc) {
        Database.executeBatch(new DataCleanupBatchJob(), 200);
    }
}
```

Batch Apex

```
public class DataCleanupBatchJob implements
Database.Batchable<sObject> {
    public Database.QueryLocator start(Database.BatchableContext bc) {
        // Query records for cleanup
        return Database.getQueryLocator('SELECT Id FROM MyObject__c
WHERE Condition__c = \'Old\'');
    }
    public void execute(Database.BatchableContext bc, List<sObject>
scope) {
        // Perform cleanup logic
        for (MyObject__c record : (List<MyObject__c>)scope) {
            // Example: Delete or update records
        }
    }
    public void finish(Database.BatchableContext bc) {
        // Post-processing logic if needed
    }
}
```

147. A Salesforce org requires a series of related processes to be executed in a specific order, such as updating account information, sending notification emails, and logging the actions. How to implement this scenario?

Queueable Apex allows for chaining jobs, enabling one job to start another once it completes. This is useful for executing dependent processes sequentially.

1. First Queueable Job:

   - Updates account information and, upon completion, enqueues the next Queueable job.

2. Second Queueable Job:

   - Sends notification emails based on the updated information and then enqueues the final job.

3. Third Queueable Job:

   - Logs the actions taken, providing a record of the processes that occurred.

Sample Code

```
public class UpdateAccountJob implements Queueable {
    public void execute(QueueableContext context) {
        // Update account information
        // Chain the next job
        System.enqueueJob(new SendNotificationJob());
    }
}


public class SendNotificationJob implements Queueable {
    public void execute(QueueableContext context) {
        // Send notification emails
        // Chain the final job
        System.enqueueJob(new LogActionsJob());
    }
}
public class LogActionsJob implements Queueable {
    public void execute(QueueableContext context) {
        // Log the actions taken
    }
}
```

148. You need to deactivate a Salesforce user and transfer ownership of their records to another user. How can you avoid Mixed DML errors?

Mixed DML errors occur when DML operations on setup objects (like User) and non-setup objects (like Account) are combined in the same transaction. Using a Future Method allows you to separate these operations into different transactions.

1. Initial Synchronous Transaction: In the synchronous transaction (e.g., a trigger or class), update non-setup objects like Account.

2. Future Method: Use a future method to perform operations on setup objects, such as deactivating a user.

Sample Code

```
public class UserManagement {
    public void transferOwnershipAndDeactivate(User user) {
        // Transfer ownership of non-setup objects (e.g., Account)
        transferAccountOwnership(user.Id);
        // Call future method to deactivate user
        deactivateUserAsync(user.Id);
    }

    @future
    public static void deactivateUserAsync(Id userId) {
        User user = [SELECT Id, IsActive FROM User WHERE Id = :userId];
        user.IsActive = false;
        update user;
    }

    private void transferAccountOwnership(Id userId) {
        // Logic to transfer account ownership
    }
}
```

149. How can Salesforce ensure real-time data synchronization with an external system when records are created or updated?

Queueable Apex can be used to handle near real-time data synchronization by enqueuing jobs to process and send data to an external system asynchronously.

1. Trigger to Detect Changes: A trigger on the relevant object detects changes and enqueues a Queueable job.

2. Queueable Job for Synchronization: The job handles the logic for sending data to the external system via a callout.

Sample Code

```
trigger ContactTrigger on Contact (after insert, after update) {
    for (Contact con : Trigger.new) {
        System.enqueueJob(new SyncContactWithExternalSystem(con.Id));
    }
}

public class SyncContactWithExternalSystem implements Queueable,
Database.AllowsCallouts {
    private Id contactId;

    public SyncContactWithExternalSystem(Id contactId) {
        this.contactId = contactId;
    }
    public void execute(QueueableContext context) {
        // Retrieve the contact data
        Contact con = [SELECT Id, Name, Email FROM Contact WHERE Id =
:contactId];
        // Perform callout to external system
        syncWithExternalSystem(con);
    }
    private void syncWithExternalSystem(Contact con) {
        // Callout logic
    }
}
```

150. A company needs to export Salesforce data to an external system every night. The data set can be large, and the export needs to be handled efficiently. How can this be achieved ?

Use Schedulable Apex to trigger the Batch Apex job nightly. The Batch Apex job will handle exporting data in manageable chunks.

1. Schedulable Apex for Scheduling: Schedule the Batch Apex job to run at a specific time each night.

2. Batch Apex for Data Export: Query the relevant records, process them in chunks, and perform the data export.

```
public class DataExportScheduler implements Schedulable {
    public void execute(SchedulableContext sc) {
        Database.executeBatch(new DataExportBatch(), 200);
    }
}

public class DataExportBatch implements Database.Batchable<sObject>,
Database.AllowsCallouts {
    public Database.QueryLocator start(Database.BatchableContext bc) {
        // Query records to export
        return Database.getQueryLocator('SELECT Id, Name,
LastModifiedDate FROM MyObject__c');
    }

    public void execute(Database.BatchableContext bc, List<sObject>
scope) {
        // Export logic for each batch of records
        for (MyObject__c record : (List<MyObject__c>)scope) {
            // Perform export logic (e.g., callout to external system)
        }
    }

    public void finish(Database.BatchableContext bc) {
        // Finalize export, send notifications if needed
    }
}
```