interview questions

1.can we call the batch in to another batch apex?

yes we can call finish method

2.can we call batch apex in to another batch in excute 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.

3.can we call the batch apex in 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.

4.webservices we can call in batch apex?

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

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

start method,finsih method one time, excute method it depends on you.- Based on the batch size and data retrieved in Start method.

6. batch excutions perday?

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

7.can we call the future method in batch class?

No,we cant call

8.future method supports primitive data types why sobject parameters not supported?

The reason why sObjects can't be passed as arguments to future methods is that the sObject might change between the time you call the method and the time it executes. In this case, the future method will get the old sObject values and might overwrite them.

9.How can I perform Callouts from Future methods?

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

10.Can I write a future call in Trigger?

yes

11.What are some limitations of future methods?

It is not a good option to process large numbers of records.

Only primitive data types supported.

Tracing a future job is also typical.

Can't call future from batch and future contexts, 1 call from queueable context is allowed.

12.future method?

use of future methods to isolate DML operations on different sObject types to prevent the mixed DML error. Each future method is queued and executes when system resources become available. That way, the execution of your code doesn't have to wait for the completion of a long-running operation. A benefit of using future methods is that some governor limits are higher, such as SOQL query limits and heap size limits

NOTE :-

1) Methods with the future annotation must be static methods

2) can only return a void type

3) The specified parameters must be primitive data types, arrays of primitive data types, or collections of primitive data types

4) Methods with the future annotation cannot take sObjects or objects as arguments.

5) You can invoke future methods the same way you invoke any other method. However, a future method can't invoke another future method

6) No more than 50 method calls per Apex invocation

7) Asynchronous calls, such as @future or executeBatch, called in a startTest, stopTest block, do not count against your limits for the number of queued jobs

8) The maximum number of future method invocations per a 24-hour period is 250,000 or the number of user licenses in your organization multiplied by 200, whichever is greater

9) 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

13. How does Queueable Apex differ from Future methods?

Queueable Apex is similar to future methods in that they're both queued for execution, but they provide us these additional benefits.

When you queue a Queueable Apex, you get a job ID, that can be used to trace it easily, which is not possible in case of future methods.

You can use non-primitive datatypes in Queueable Apex, like objects and sObjects, which is not possible in case of future methods, because it supports only primitive data types as params.

You can chain jobs, by calling another starting a second job from a running job, which is not possible in case of future methods, because we can't call another future method from a future context.

14.Can you write a sample 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
   }
}
```

15.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];
```

16.Can I do callouts from a Queueable Job?

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

17.How many numbers of jobs, I can queue using System.enqueueJob() at a time?

You can add up to 50 jobs to the queue with System.enqueueJob in a single transaction in Synchronous apex. In asynchronous transactions, you can add only one job to the queue.

18.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.

19.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".

20. I have a use case to call more than one Queueable Jobs from a Batch apex, how can I achieve it?

Since we can't call more than one Queueable Job from each execution Context, We can go for scheduling the Queueable Jobs.

The approach is we need to first check how many queueable jobs are added in the queue in the current transaction by making use of Limits class. If the number has reached the limit, then call a schedulable class and enqueue the queueable class from the execute method of a schedulable class.