Asynchronous Apex

Future :

- A future method runs in the background asynchronously.
- Can call a future method for executing long running operations such as callout to extend web services or any operation that would like to run in its own thread, on its own time.
- Can use future method to isolate DML Operations on different Sobject types to prevent mixed DML error.
- Each future method is queued and executes when System resources become available. That way, the execution of the code doesn't have to wait for the completion of a long running operation.
- Benefit of using future method is that some governor limits are higher, such a SOQL, Query limits and heap Size.

Syntax :

```
@future

Public static void method2(){

}
```

Considerations :

- Method should be static and void.
- Parameters passed can be primitive, arrays of primitive, or collection of primitive dataTypes.
- Methods annotated with @future can't take subject/object as parameter.

Note : The reason why Sobject can't be passes as parameter is Sobject changes between the tie it calls the method and executes the method. In case, the future method gets the Sobject values(old) and can override them. To work with Sobject that already exist in the database, Pass the Sobject Id instead (or collection of Ids) and use the Id to perform a query for the most up-to-date record.

- To call an external Service use callout = true in the annotation.
  Syntax :

  ```
  @future(callout = true)
  Public static void method1(){

  }
  ```

- For mixed DML Operation.
- For a non null user roleId must be invoked in future.

Account acc = new Account(Name = 'Test Account'); User user = new User(name , email) ⇨ Role null won't throw any mixed DML error.	⇨ If the role is mentioned then use @future to else it will throw mixed DML error.

- A future method can't be invoked from another future method.

Method with @future has the following Limitations:

- No more than 0 in batch and future contexts ; 50 in queueable context method calls per transaction/apex invocation.
- Asynchronous calls such as @future or executeBatch, called In a startTest and stopTest block in test class, doesn't count against the limit for the number of queued jobs.

Note :

Having multiple future methods fan out from a queueable jobs isn't recommended practice as it can rapidly add a large number of future methods to the asynchronous queue. Request processing can be delayed and can quickly hits the daily limit for asynchronous Apex method execution.

- The maximum no of future method invocation per day is 2,50,000 or no of licenses in the org multiplied by 200 whichever is greater.
- The limit for entire org is shared with all asynchronous apex . Batch apex , queueable apex , schedulable apex , future methods.
- To check how many asynchronous apex limits are available execute REST API limits resource.

  GET : {InstanceURL}/services/data/v{version}/limits

- If the no of asynchronous apex executions needed by a job exceeds the available number that's calculated using the 24-hour rolling limit, an exception is thrown.

  EX: If async job requires 10,000 method executions and the available 24-hour rolling limit is 9,500. Can get AsynApexExecutions limit exceed exception.
  The license type that count towards the limit includes full salesforce & salesforce platform user license , App Subscription User license , chatter only users, Identity users, and any company communites user.

Note :

- Future jobs queued by a transaction aren't processed if the transaction roll back.
- Future method job queued by salesforce service maintenance downtime remain in queue. After Service downtime ends by and when System resources becomes available, the queued future method jobs are executed . If a future method was running when downtime occurred, the future method execution is rolled back and restarted after the service comes back up.

Testing Future Methods:

Call the future method in Test.startTest() and Test.stopTest() code block.

All asynchronous calls made after startTest methods are collected by the system. When stopTest is executed all asynchronous process are run Synchronously.

Best Practices :

SalesForce uses queue-based framework to handle asynchronous process from such sources as future methods and batch apex. This queue is used to balance request workload across organizations.

Avoid adding large number of future methods to the asynchronous queue. If possible if more than 2000 unprocessed requests from a single organizations are in the queue, any additional resources from the same organization will be delayed while the queue handles request from other organizations.

Ensure that future methods executes as fast as possible. To ensure fast execution of batch jobs, minimize web service callout times and tune queries used in future methods. The longer the future method executes, the more likely other queued requests are delayed when there are a large number of requests in the queue.

Test your future methods at scale. To help determine if delays can occur. Test using an environment that generates the maximum number of future methods you'd expect to handle.

Consider using batch apex instead of future methods to process large numbers of records.