Advanced Salesforce Development - Interview Questions Guide

Comprehensive Interview Questions on Governor Limits, Asynchronous Apex, and Testing

This guide covers advanced Salesforce development topics including Governor Limits, Future Methods, Batch Apex, Queueable Apex, Apex Scheduler, and Test Classes with detailed examples.

Table of Contents

Governor Limits

Question 1: What are Governor Limits in Salesforce?

Answer: Governor limits are runtime limits enforced by Salesforce to ensure efficient use of resources on the Force.com multitenant platform. These limits prevent any single tenant from monopolizing shared resources.

Basic Example:

```
public class GovernorLimitExample {
    public static void demonstrateLimits() {
        // Check current limits
        System.debug('SOQL Queries Used: ' + Limits.getQueries());
```

```
        System.debug('SOQL Queries Limit: ' + Limits.getLimitQueries());

        System.debug('DML Statements Used: ' + Limits.getDmlStatements())
        System.debug('DML Statements Limit: ' + Limits.getLimitDmlStateme

        System.debug('Heap Size Used: ' + Limits.getHeapSize());
        System.debug('Heap Size Limit: ' + Limits.getLimitHeapSize());

        System.debug('CPU Time Used: ' + Limits.getCpuTime());
        System.debug('CPU Time Limit: ' + Limits.getLimitCpuTime());
    }
}
```

Question 2: What are the different types of Governor Limits?

Answer: Governor limits are categorized into:

- **Per-Transaction Limits**: Apply to each Apex transaction
- **Static Apex Limits**: Fixed across all transactions
- **Size-Specific Limits**: Related to data and code size
- **Platform Apex Limits**: Apply at platform level
- **Certified Managed Package Limits**: Separate limits for ISV packages

Basic Example:

```
public class LimitTypes {
    public void showLimitTypes() {
        // Per-Transaction Limit Example
        List<Account> accounts = [SELECT Id FROM Account LIMIT 100]; // S
        insert accounts; // DML statement

        // Static Limit Example
        Integer cpuTime = Limits.getCpuTime(); // CPU time in milliseconc

        // Size-Specific Limit Example
        String largeString = 'A'.repeat(1000000); // Heap size considerat
    }
}
```

Question 3: What are the key per-transaction synchronous and asynchronous limits?

Answer:

Limit Type	Synchronous Limit	Asynchronous Limit
Total SOQL queries	100	200
Total records retrieved by SOQL	50,000	50,000
Total SOSL queries	20	20
Total DML statements	150	150
Total records processed by DML	10,000	10,000
Maximum CPU time	10,000 ms (10 sec)	60,000 ms (60 sec)
Total heap size	6 MB	12 MB
Maximum callouts	100	100
Total email invocations	10	10

Basic Example:

```
public class SyncVsAsync {
    // Synchronous - 10 seconds CPU, 6 MB heap
    public void syncMethod() {
        System.debug('Sync CPU Limit: ' + Limits.getLimitCpuTime()); // 1
        System.debug('Sync Heap Limit: ' + Limits.getLimitHeapSize()); //
    }

    // Asynchronous - 60 seconds CPU, 12 MB heap
    @future
    public static void asyncMethod() {
        System.debug('Async CPU Limit: ' + Limits.getLimitCpuTime()); //
        System.debug('Async Heap Limit: ' + Limits.getLimitHeapSize()); /
    }
}
```

Question 4: What is the most common governor limit error and how do you avoid it?

Answer: The most common error is "Too many SOQL queries: 101" caused by queries inside loops.

Basic Example:

BAD - Query in Loop:

```
public class BadPractice {
    public void updateContacts(List<Account> accounts) {
        for(Account acc : accounts) {
            // Query inside loop - will hit limit after 100 iterations!
            List<Contact> contacts = [SELECT Id FROM Contact WHERE Accour
            // Process contacts
        }
    }
}
```

GOOD - Bulkified:

```
public class GoodPractice {
    public void updateContacts(List<Account> accounts) {
        // Collect all account IDs
        Set<Id> accountIds = new Set<Id>();
        for(Account acc : accounts) {
            accountIds.add(acc.Id);
        }

        // Single query outside loop
        List<Contact> contacts = [SELECT Id, AccountId FROM Contact WHERE

        // Create map for efficient lookup
        Map<Id, List<Contact>> accountToContacts = new Map<Id, List<Conta
        for(Contact con : contacts) {
            if(!accountToContacts.containsKey(con.AccountId)) {
                accountToContacts.put(con.AccountId, new List<Contact>())
            }
            accountToContacts.get(con.AccountId).add(con);
        }

        // Process without additional queries
        for(Account acc : accounts) {
            List<Contact> relatedContacts = accountToContacts.get(acc.Id)
            if(relatedContacts != null) {
                // Process contacts
            }
        }
    }
}
```

Question 5: How do you avoid DML statement governor limits?

Answer: Avoid performing DML operations inside loops. Instead, collect records in a list and perform a single DML operation.

Basic Example:

BAD - DML in Loop:

```
public class BadDML {
    public void updateAccounts(List<Account> accounts) {
        for(Account acc : accounts) {
            acc.Industry = 'Technology';
            update acc; // DML in loop - will hit limit after 150 iterati
        }
    }
}
```

GOOD - Bulkified DML:

```
public class GoodDML {
    public void updateAccounts(List<Account> accounts) {
        // Modify all records
        for(Account acc : accounts) {
            acc.Industry = 'Technology';
        }

        // Single DML statement
        update accounts; // Can handle up to 10,000 records
    }
}
```

Question 6: What is the Heap Size limit and how do you avoid exceeding it?

Answer: Heap size is the memory used by your code during execution. Synchronous: 6 MB, Asynchronous: 12 MB. Avoid by limiting data in memory.

Basic Example:

BAD - Loading Too Much Data:

```
public class HeapIssue {
    public void processRecords() {
        // Loading all records into memory at once
        List<Account> allAccounts = [SELECT Id, Name, Description, Indust
                                     BillingStreet, BillingCity, Billir
                                     (SELECT Id, FirstName, LastName, E
                                     FROM Account];

        // If too many records, will hit heap limit!
        System.debug('Heap Used: ' + Limits.getHeapSize());
    }
}
```

GOOD - Query Only Needed Fields:

```
public class HeapOptimized {
    public void processRecords() {
        // Query only required fields
        List<Account> accounts = [SELECT Id, Name FROM Account LIMIT 1000

        System.debug('Heap Used: ' + Limits.getHeapSize());

        // Process in batches if needed
        Integer batchSize = 200;
        for(Integer i = 0; i < accounts.size(); i += batchSize) {
            List<Account> batch = new List<Account>();
            for(Integer j = i; j < Math.min(i + batchSize, accounts.size(
                batch.add(accounts[j]);
            }
            // Process batch
            batch.clear(); // Clear to free memory
        }
    }
}
```

Question 7: What is CPU Time limit and how do you handle it?

Answer: CPU time is the actual time spent executing Apex code on Salesforce servers. Synchronous: 10 seconds, Asynchronous: 60 seconds.

Basic Example:

BAD - Complex Nested Loops:

```
public class CPUIntensive {
    public void complexOperation(List<Account> accounts) {
        for(Account acc : accounts) {
            for(Integer i = 0; i < 10000; i++) {
                for(Integer j = 0; j < 1000; j++) {
                    // Complex calculations
                    Decimal result = Math.pow(i, j);
                }
            }
        }
        // Will likely hit CPU timeout!
    }
}
```

GOOD - Optimized Logic:

```
public class CPUOptimized {
    public void optimizedOperation(List<Account> accounts) {
        // Check CPU time periodically
        Long startTime = System.currentTimeMillis();

        for(Account acc : accounts) {
            // Simplified logic
            acc.Rating = 'Hot';

            // Monitor CPU time
            if(Limits.getCpuTime() > 8000) { // 8 seconds
                System.debug('Approaching CPU limit, consider async proce
                break;
            }
        }

        Long endTime = System.currentTimeMillis();
        System.debug('Execution time: ' + (endTime - startTime) + ' ms');
    }
}
```

Question 8: How do you monitor and check governor limits in your code?

Answer: Use the Limits class to monitor current usage and remaining limits.

Basic Example:

```
public class LimitsMonitor {
    public static void checkLimits() {
        System.debug('=== SOQL Limits ===');
        System.debug('SOQL Queries: ' + Limits.getQueries() + '/' + Limit
        System.debug('SOQL Rows: ' + Limits.getQueryRows() + '/' + Limits

        System.debug('=== DML Limits ===');
        System.debug('DML Statements: ' + Limits.getDmlStatements() + '/'
        System.debug('DML Rows: ' + Limits.getDmlRows() + '/' + Limits.ge

        System.debug('=== CPU & Heap ===');
        System.debug('CPU Time: ' + Limits.getCpuTime() + '/' + Limits.ge
        System.debug('Heap Size: ' + Limits.getHeapSize() + '/' + Limits.

        System.debug('=== Other Limits ===');
        System.debug('Callouts: ' + Limits.getCallouts() + '/' + Limits.g
        System.debug('Email Invocations: ' + Limits.getEmailInvocations()
        System.debug('Queueable Jobs: ' + Limits.getQueueableJobs() + '/'
    }

    public static Boolean isApproachingLimits() {
        // Check if approaching 80% of limits
        return (Limits.getQueries() > Limits.getLimitQueries() * 0.8) ||
               (Limits.getDmlStatements() > Limits.getLimitDmlStatements(
               (Limits.getCpuTime() > Limits.getLimitCpuTime() * 0.8);
    }
}
```

Question 9: What happens when you exceed a governor limit?

Answer: When a governor limit is exceeded, Salesforce throws a LimitException and the entire transaction is rolled back. No partial commits occur.

Basic Example:

```
public class LimitException {
    public void exceedLimit() {
        try {
            // Intentionally exceed SOQL limit
            for(Integer i = 0; i < 101; i++) {
```

```
                List<Account> accounts = [SELECT Id FROM Account LIMIT 1]
            }
        } catch(System.LimitException e) {
            System.debug('Governor Limit Exceeded!');
            System.debug('Error Message: ' + e.getMessage());
            System.debug('Stack Trace: ' + e.getStackTraceString());
            // Transaction is rolled back
        }
    }
}
```

Question 10: What are best practices to avoid governor limits?

Answer:

1. Bulkify your code (no SOQL/DML in loops)
2. Use collections (List, Set, Map) efficiently
3. Query only necessary fields and records
4. Use asynchronous processing for large operations
5. Monitor limits using Limits class
6. Use selective queries with WHERE clauses
7. Limit trigger recursion
8. Use Platform Events for cross-org integration

Basic Example:

```
public class BestPractices {
    // 1. Bulkified code
    public void bulkifiedUpdate(List<Account> accounts) {
        List<Account> accountsToUpdate = new List<Account>();

        for(Account acc : accounts) {
            acc.Industry = 'Technology';
            accountsToUpdate.add(acc);
        }

        update accountsToUpdate; // Single DML
    }

    // 2. Efficient collections
    public void efficientLookup(Set<Id> accountIds) {
        // Use Map for O(1) lookup instead of nested loops
```

```
        Map<Id, Account> accountMap = new Map<Id, Account>(
            [SELECT Id, Name FROM Account WHERE Id IN :accountIds]
        );

        // Quick lookup
        Account specificAccount = accountMap.get(accountIds.iterator().ne
    }

    // 3. Selective queries
    public void selectiveQuery(Date startDate) {
        // Query only what you need with filters
        List<Opportunity> recentOpps = [
            SELECT Id, Name, Amount
            FROM Opportunity
            WHERE CreatedDate >= :startDate
            AND Amount > 10000
            LIMIT 1000
        ];
    }

    // 4. Prevent recursion
    private static Boolean hasRun = false;

    public void preventRecursion() {
        if(!hasRun) {
            hasRun = true;
            // Execute logic once
        }
    }
}
```

Asynchronous Apex Overview

Question 11: What is Asynchronous Apex?

Answer: Asynchronous Apex allows you to run processes in the background without blocking the main execution thread. It provides higher governor limits and is used for long-running operations.

Types of Asynchronous Apex:

1. **Future Methods** - Simple async operations
2. **Batch Apex** - Process large volumes of records

3. **Queueable Apex** - Enhanced async with job chaining
4. **Scheduled Apex** - Run code at specific times

Basic Example:

```
public class AsyncOverview {
    // Synchronous - runs immediately
    public void syncMethod() {
        System.debug('Running synchronously');
        System.debug('CPU Limit: ' + Limits.getLimitCpuTime()); // 10000
    }

    // Asynchronous - runs in background
    @future
    public static void asyncMethod() {
        System.debug('Running asynchronously');
        System.debug('CPU Limit: ' + Limits.getLimitCpuTime()); // 60000
    }
}
```

Question 12: When should you use Asynchronous Apex?

Answer: Use asynchronous Apex when:

- Processing large data volumes
- Making external callouts (web services)
- Performing long-running operations
- Need to avoid governor limits
- Processing can be delayed
- Chain multiple operations

Basic Example:

```
public class AsyncUseCases {
    // Use Case 1: External Callout
    @future(callout=true)
    public static void makeCallout() {
        HttpRequest req = new HttpRequest();
        req.setEndpoint('https://api.example.com/data');
        req.setMethod('GET');

        Http http = new Http();
```

```
        HttpResponse res = http.send(req);
        System.debug('Response: ' + res.getBody());
    }

    // Use Case 2: Large data processing
    public static void processLargeDataset(List<Id> recordIds) {
        if(recordIds.size() > 200) {
            // Use Batch Apex for large datasets
            BatchProcessor batch = new BatchProcessor();
            Database.executeBatch(batch, 200);
        }
    }

    // Use Case 3: Mixed DML operations
    @future
    public static void mixedDML() {
        // Can perform DML on Setup and non-Setup objects
        User u = new User(Id = UserInfo.getUserId(), Title = 'Developer')
        update u; // Setup object

        Account acc = new Account(Name = 'Test');
        insert acc; // Non-Setup object
    }
}
```

Question 13: What are the differences between synchronous and asynchronous Apex?

Answer:

Aspect	Synchronous	Asynchronous
Execution	Immediate	Queued/Delayed
CPU Time	10 seconds	60 seconds
Heap Size	6 MB	12 MB
SOQL Queries	100	200
User waits	Yes	No
Callouts	Limited	Better support
Use Case	Quick operations	Long-running tasks

Basic Example:

```
public class SyncAsyncComparison {
    // Synchronous - user waits
    public void syncOperation() {
        List<Account> accounts = new List<Account>();
        for(Integer i = 0; i < 100; i++) {
            accounts.add(new Account(Name = 'Account ' + i));
        }
        insert accounts;
        System.debug('Sync complete - user waited');
    }

    // Asynchronous - user doesn't wait
    @future
    public static void asyncOperation() {
        List<Account> accounts = new List<Account>();
        for(Integer i = 0; i < 100; i++) {
            accounts.add(new Account(Name = 'Account ' + i));
        }
        insert accounts;
        System.debug('Async complete - user didn\'t wait');
    }

    // Calling async
    public void executeAsync() {
        asyncOperation(); // Returns immediately
        System.debug('This runs before asyncOperation completes');
    }
}
```

Future Methods

Question 14: What is a Future Method?

Answer: A Future method is an asynchronous method that runs in the background when Salesforce has available resources. It's marked with the @future annotation.

Characteristics:

- Must be static
- Must return void

- Can only have primitive parameters
- Cannot call another future method
- Provides higher governor limits

Basic Example:

```
public class FutureMethodExample {
    // Future method
    @future
    public static void processRecords(List<Id> accountIds) {
        List<Account> accounts = [SELECT Id, Name FROM Account WHERE Id I

        for(Account acc : accounts) {
            acc.Description = 'Processed by future method';
        }

        update accounts;
        System.debug('Future method completed');
    }

    // Calling future method
    public void callFuture() {
        List<Id> ids = new List<Id>{'001xx000003DGb2AAG', '001xx000003DGk
        processRecords(ids); // Runs asynchronously
        System.debug('Future method queued');
    }
}
```

Question 15: How do you make HTTP callouts using Future methods?

Answer: Use @future(callout=true) annotation to enable HTTP callouts in future methods.

Basic Example:

```
public class FutureCallout {
    @future(callout=true)
    public static void makeCallout(String endpoint) {
        HttpRequest req = new HttpRequest();
        req.setEndpoint(endpoint);
        req.setMethod('GET');
        req.setTimeout(12000);
```

```
        Http http = new Http();
        HttpResponse res = http.send(req);

        if(res.getStatusCode() == 200) {
            System.debug('Success: ' + res.getBody());

            // Process response
            Map<String, Object> jsonResponse = (Map<String, Object>)JSON.
            System.debug('Data: ' + jsonResponse);
        } else {
            System.debug('Error: ' + res.getStatus());
        }
    }

    // Trigger example
    public static void handleAccountUpdate(List<Account> accounts) {
        for(Account acc : accounts) {
            // Call external system asynchronously
            makeCallout('https://api.example.com/accounts/' + acc.Id);
        }
    }
}
```

Question 16: What are the limitations of Future methods?

Answer:

1. Cannot call another future method
2. Cannot take non-primitive parameters (no sObjects, objects)
3. Cannot track job status easily
4. No chaining of jobs
5. Limited to 50 per transaction (0 in batch/future context)

Basic Example:

```
public class FutureLimitations {
    // WRONG - Cannot pass sObject
    // @future
    // public static void processAccount(Account acc) { } // Compilation

    // CORRECT - Pass primitive (Id)
    @future
    public static void processAccount(Id accountId) {
```

```
        Account acc = [SELECT Id, Name FROM Account WHERE Id = :accountIc
        acc.Rating = 'Hot';
        update acc;
    }

    // WRONG - Cannot call another future from future
    @future
    public static void method1() {
        // method2(); // Runtime error!
    }

    @future
    public static void method2() {
        System.debug('Method 2');
    }

    // CORRECT - Use Queueable for chaining
    public class ChainableJob implements Queueable {
        public void execute(QueueableContext context) {
            // Process records
            System.enqueueJob(new NextJob()); // Can chain
        }
    }

    public class NextJob implements Queueable {
        public void execute(QueueableContext context) {
            System.debug('Next job');
        }
    }
}
```

Question 17: How do you test Future methods?

Answer: Use Test.startTest() and Test.stopTest() to force future methods to execute synchronously in tests.

Basic Example:

```
// Future Method Class
public class AccountProcessor {
    @future
    public static void updateIndustry(List<Id> accountIds, String industr
        List<Account> accounts = [SELECT Id FROM Account WHERE Id IN :acc
```

```
        for(Account acc : accounts) {
            acc.Industry = industry;
        }
        update accounts;
    }
}

// Test Class
@isTest
public class AccountProcessorTest {
    @isTest
    static void testFutureMethod() {
        // Setup test data
        Account acc = new Account(Name = 'Test Account');
        insert acc;

        // Test future method
        Test.startTest();
        AccountProcessor.updateIndustry(new List<Id>{acc.Id}, 'Technology
        Test.stopTest(); // Future method executes here

        // Verify results
        Account updatedAcc = [SELECT Industry FROM Account WHERE Id = :ac
        System.assertEquals('Technology', updatedAcc.Industry, 'Industry
    }
}
```

Question 18: When should you use Future methods?

Answer: Use Future methods when:

- Making web service callouts
- Performing operations that take time
- Mixed DML operations (Setup and non-Setup objects)
- Simple async processing (no chaining needed)
- Don't need to pass complex objects

Basic Example:

```
public class FutureUseCases {
    // Use Case 1: Mixed DML
    @future
    public static void createUserAndAccount() {
```

```
    // Can mix Setup (User) and non-Setup (Account) objects
    User u = new User(
        FirstName = 'Test',
        LastName = 'User',
        Email = 'test@example.com',
        Username = 'testuser' + DateTime.now().getTime() + '@example.
        Alias = 'tuser',
        TimeZoneSidKey = 'America/Los_Angeles',
        LocaleSidKey = 'en_US',
        EmailEncodingKey = 'UTF-8',
        ProfileId = [SELECT Id FROM Profile WHERE Name = 'Standard Us
        LanguageLocaleKey = 'en_US'
    );
    insert u;

    Account acc = new Account(Name = 'Test Account');
    insert acc;
}

// Use Case 2: Time-consuming operation
@future
public static void sendEmailNotifications(List<Id> contactIds) {
    List<Contact> contacts = [SELECT Email FROM Contact WHERE Id IN :

    List<Messaging.SingleEmailMessage> emails = new List<Messaging.Si
    for(Contact con : contacts) {
        Messaging.SingleEmailMessage email = new Messaging.SingleEmai
        email.setToAddresses(new String[]{con.Email});
        email.setSubject('Welcome');
        email.setPlainTextBody('Thank you for registering!');
        emails.add(email);
    }

    Messaging.sendEmail(emails);
}

// Use Case 3: Avoiding mixed DML in trigger
public static void handleTrigger(List<Account> accounts) {
    List<Id> accountIds = new List<Id>();
    for(Account acc : accounts) {
        accountIds.add(acc.Id);
    }
    updateAccountsAsync(accountIds);
}
```

```
    @future
    public static void updateAccountsAsync(List<Id> accountIds) {
        // Process accounts asynchronously
    }
}
```

Batch Apex

Question 19: What is Batch Apex?

Answer: Batch Apex allows processing of large numbers of records asynchronously in chunks (batches). It can process up to 50 million records while respecting governor limits.

Three Required Methods:

1. **start()** - Collects records to process
2. **execute()** - Processes each batch
3. **finish()** - Post-processing operations

Basic Example:

```
public class AccountBatch implements Database.Batchable<sObject> {
    // 1. Start method - collect records
    public Database.QueryLocator start(Database.BatchableContext bc) {
        String query = 'SELECT Id, Name, Industry FROM Account';
        return Database.getQueryLocator(query);
    }

    // 2. Execute method - process each batch
    public void execute(Database.BatchableContext bc, List<Account> scope
        System.debug('Processing batch of ' + scope.size() + ' accounts')

        for(Account acc : scope) {
            acc.Description = 'Processed on ' + System.today();
        }

        update scope;
    }

    // 3. Finish method - post-processing
    public void finish(Database.BatchableContext bc) {
```

```
        System.debug('Batch job completed');

        // Get job status
        AsyncApexJob job = [
            SELECT Id, Status, NumberOfErrors, JobItemsProcessed, TotalJc
            FROM AsyncApexJob
            WHERE Id = :bc.getJobId()
        ];

        System.debug('Job Status: ' + job.Status);
        System.debug('Records Processed: ' + job.JobItemsProcessed);
        System.debug('Errors: ' + job.NumberOfErrors);
    }
}

// Execute batch
AccountBatch batch = new AccountBatch();
Id jobId = Database.executeBatch(batch, 200); // Batch size: 200
System.debug('Batch Job ID: ' + jobId);
```

Question 20: What is the difference between Database.QueryLocator and Iterable in start method?

Answer:

Database.QueryLocator:

- Can process up to 50 million records
- Uses SOQL query
- Best for simple queries on single object

Iterable:

- Limited to 50,000 records
- More flexible (custom logic, multiple queries, external data)
- Used for complex scenarios

Basic Example:

```
// Using Database.QueryLocator
public class QueryLocatorBatch implements Database.Batchable<sObject> {
    public Database.QueryLocator start(Database.BatchableContext bc) {
        // Can process up to 50 million records
```

```
        return Database.getQueryLocator('SELECT Id, Name FROM Account');
    }

    public void execute(Database.BatchableContext bc, List<Account> scope
        // Process batch
    }

    public void finish(Database.BatchableContext bc) {
        // Finish processing
    }
}

// Using Iterable
public class IterableBatch implements Database.Batchable<Account> {
    public Iterable<Account> start(Database.BatchableContext bc) {
        // Custom logic - limited to 50,000 records
        List<Account> accounts = new List<Account>();

        // Can add complex logic
        for(Account acc : [SELECT Id, Name FROM Account WHERE Industry =
            if(acc.Name.startsWith('A')) {
                accounts.add(acc);
            }
        }

        return accounts;
    }

    public void execute(Database.BatchableContext bc, List<Account> scope
        // Process batch
    }

    public void finish(Database.BatchableContext bc) {
        // Finish processing
    }
}
```

Question 21: How do you execute and monitor a Batch Apex job?

Answer: Execute using Database.executeBatch() and monitor using AsyncApexJob object or Salesforce UI.

Basic Example:

```
// Batch Class
public class DataCleanupBatch implements Database.Batchable<sObject> {
    public Database.QueryLocator start(Database.BatchableContext bc) {
        return Database.getQueryLocator('SELECT Id FROM Account WHERE Cre
    }

    public void execute(Database.BatchableContext bc, List<Account> scope
        delete scope;
    }

    public void finish(Database.BatchableContext bc) {
        System.debug('Cleanup completed');
    }
}

// Execute Batch
public class BatchExecutor {
    public static void runBatch() {
        DataCleanupBatch batch = new DataCleanupBatch();

        // Execute with batch size 200 (default)
        Id jobId = Database.executeBatch(batch);
        System.debug('Job ID: ' + jobId);

        // Execute with custom batch size
        Id jobId2 = Database.executeBatch(batch, 100);
    }

    // Monitor batch job
    public static void monitorJob(Id jobId) {
        AsyncApexJob job = [
            SELECT Id, Status, JobType, NumberOfErrors,
                   JobItemsProcessed, TotalJobItems, CreatedDate,
                   CompletedDate, ExtendedStatus
            FROM AsyncApexJob
            WHERE Id = :jobId
        ];

        System.debug('=== Batch Job Status ===');
        System.debug('Status: ' + job.Status);
        System.debug('Total Batches: ' + job.TotalJobItems);
        System.debug('Processed: ' + job.JobItemsProcessed);
        System.debug('Errors: ' + job.NumberOfErrors);
        System.debug('Started: ' + job.CreatedDate);
```

```
        System.debug('Completed: ' + job.CompletedDate);

        // Check if job completed
        if(job.Status == 'Completed') {
            System.debug('Batch completed successfully!');
        } else if(job.Status == 'Failed') {
            System.debug('Batch failed: ' + job.ExtendedStatus);
        }
    }
}

// Query all running batch jobs
List<AsyncApexJob> runningJobs = [
    SELECT Id, Status, ApexClass.Name, CreatedDate
    FROM AsyncApexJob
    WHERE JobType = 'BatchApex'
    AND Status IN ('Processing', 'Queued', 'Preparing')
    ORDER BY CreatedDate DESC
];

for(AsyncApexJob job : runningJobs) {
    System.debug('Batch: ' + job.ApexClass.Name + ' - Status: ' + job.Sta
}
```

Question 22: What is Database.Stateful in Batch Apex?

Answer: Database.Stateful interface maintains state across batch transactions. Without it, instance variables are reset after each execute() call.

Basic Example:

```
// Without Stateful - count resets each batch
public class BatchWithoutStateful implements Database.Batchable<sObject>
    Integer recordCount = 0;

    public Database.QueryLocator start(Database.BatchableContext bc) {
        return Database.getQueryLocator('SELECT Id FROM Account');
    }

    public void execute(Database.BatchableContext bc, List<Account> scope
        recordCount += scope.size();
        System.debug('Count in execute: ' + recordCount); // Correct in t
    }
```

```
    public void finish(Database.BatchableContext bc) {
        System.debug('Total count: ' + recordCount); // WRONG - Shows onl
    }
}

// With Stateful - count persists across batches
public class BatchWithStateful implements Database.Batchable<sObject>, Da
    Integer totalRecords = 0;
    Integer processedRecords = 0;
    Integer errorRecords = 0;

    public Database.QueryLocator start(Database.BatchableContext bc) {
        return Database.getQueryLocator('SELECT Id, Name FROM Account');
    }

    public void execute(Database.BatchableContext bc, List<Account> scope
        totalRecords += scope.size();

        for(Account acc : scope) {
            try {
                acc.Rating = 'Hot';
                processedRecords++;
            } catch(Exception e) {
                errorRecords++;
                System.debug('Error: ' + e.getMessage());
            }
        }

        update scope;
    }

    public void finish(Database.BatchableContext bc) {
        System.debug('=== Batch Summary ===');
        System.debug('Total Records: ' + totalRecords); // Correct total!
        System.debug('Processed: ' + processedRecords);
        System.debug('Errors: ' + errorRecords);

        // Send summary email
        Messaging.SingleEmailMessage email = new Messaging.SingleEmailMes
        email.setToAddresses(new String[]{'admin@example.com'});
        email.setSubject('Batch Job Completed');
        email.setPlainTextBody('Total: ' + totalRecords + '\nProcessed: '
        Messaging.sendEmail(new Messaging.SingleEmailMessage[]{email});
```

```
    }
}
```

Question 23: How do you handle errors in Batch Apex?

Answer: Use try-catch blocks, Database methods with partial success, and Database.Stateful to track errors.

Basic Example:

```
public class BatchWithErrorHandling implements Database.Batchable<sObject
    List<String> errorMessages = new List<String>();
    Integer successCount = 0;
    Integer errorCount = 0;

    public Database.QueryLocator start(Database.BatchableContext bc) {
        return Database.getQueryLocator('SELECT Id, Name FROM Account');
    }

    public void execute(Database.BatchableContext bc, List<Account> scope
        List<Account> accountsToUpdate = new List<Account>();

        for(Account acc : scope) {
            try {
                // Business logic that might fail
                if(acc.Name != null) {
                    acc.Industry = 'Technology';
                    accountsToUpdate.add(acc);
                }
            } catch(Exception e) {
                errorCount++;
                errorMessages.add('Account ' + acc.Id + ': ' + e.getMessa
            }
        }

        // Use Database method for partial success
        if(!accountsToUpdate.isEmpty()) {
            Database.SaveResult[] results = Database.update(accountsToUpc

            for(Integer i = 0; i < results.size(); i++) {
                if(results[i].isSuccess()) {
                    successCount++;
                } else {
```

```
                    errorCount++;
                    String errorMsg = 'Account ' + accountsToUpdate[i].Ic
                    for(Database.Error err : results[i].getErrors()) {
                        errorMsg += err.getMessage() + '; ';
                    }
                    errorMessages.add(errorMsg);
                }
            }
        }
    }

    public void finish(Database.BatchableContext bc) {
        // Log errors
        System.debug('=== Batch Completed ===');
        System.debug('Success: ' + successCount);
        System.debug('Errors: ' + errorCount);

        if(!errorMessages.isEmpty()) {
            System.debug('Error Details:');
            for(String msg : errorMessages) {
                System.debug(msg);
            }

            // Send error report
            sendErrorReport(errorMessages);
        }
    }

    private void sendErrorReport(List<String> errors) {
        String emailBody = 'Batch job encountered ' + errors.size() + ' e
        for(String error : errors) {
            emailBody += error + '\n';
        }

        Messaging.SingleEmailMessage email = new Messaging.SingleEmailMes
        email.setToAddresses(new String[]{'admin@example.com'});
        email.setSubject('Batch Job Errors');
        email.setPlainTextBody(emailBody);
        Messaging.sendEmail(new Messaging.SingleEmailMessage[]{email});
    }
}
```

Question 24: What is the batch size and its limits?

Answer:

- **Default batch size:** 200 records
- **Minimum:** 1 record
- **Maximum:** 2,000 records
- **Recommendation:** Use 200 for most cases

Basic Example:

```
public class BatchSizeExample implements Database.Batchable<sObject> {
    public Database.QueryLocator start(Database.BatchableContext bc) {
        return Database.getQueryLocator('SELECT Id FROM Account');
    }

    public void execute(Database.BatchableContext bc, List<Account> scope
        System.debug('Batch size: ' + scope.size());
        // Process records
    }

    public void finish(Database.BatchableContext bc) {
        System.debug('Complete');
    }
}

public class BatchExecutionExamples {
    public static void executeBatches() {
        BatchSizeExample batch = new BatchSizeExample();

        // Default batch size (200)
        Database.executeBatch(batch);

        // Small batch size - more batches, slower but safer
        Database.executeBatch(batch, 50);

        // Large batch size - fewer batches, faster but may hit limits
        Database.executeBatch(batch, 2000);

        // Minimum batch size - one record at a time
        Database.executeBatch(batch, 1);
    }

    // Choosing batch size based on complexity
    public static void chooseBatchSize() {
        BatchSizeExample batch = new BatchSizeExample();
```

```
        // Simple processing - use larger batch
        Database.executeBatch(batch, 2000);

        // Complex processing with queries/DML - use smaller batch
        Database.executeBatch(batch, 50);

        // Very complex with external callouts - use very small batch
        Database.executeBatch(batch, 10);
    }
}
```

Question 25: How do you test Batch Apex?

Answer: Use Test.startTest() and Test.stopTest() to execute batch synchronously in test context.

Basic Example:

```
// Batch Class
public class AccountBatchProcessor implements Database.Batchable<sObject>
    public Database.QueryLocator start(Database.BatchableContext bc) {
        return Database.getQueryLocator('SELECT Id, Industry FROM Account
    }

    public void execute(Database.BatchableContext bc, List<Account> scope
        for(Account acc : scope) {
            acc.Industry = 'Technology';
        }
        update scope;
    }

    public void finish(Database.BatchableContext bc) {
        System.debug('Batch finished');
    }
}

// Test Class
@isTest
public class AccountBatchProcessorTest {
    @isTest
    static void testBatch() {
        // Setup test data
        List<Account> testAccounts = new List<Account>();
```

```
        for(Integer i = 0; i < 200; i++) {
            testAccounts.add(new Account(Name = 'Test Account ' + i));
        }
        insert testAccounts;

        // Test batch execution
        Test.startTest();
        AccountBatchProcessor batch = new AccountBatchProcessor();
        Id jobId = Database.executeBatch(batch, 100);
        Test.stopTest(); // Batch executes synchronously here

        // Verify results
        List<Account> updatedAccounts = [SELECT Industry FROM Account];
        for(Account acc : updatedAccounts) {
            System.assertEquals('Technology', acc.Industry, 'Industry shc
        }

        // Verify job status
        AsyncApexJob job = [
            SELECT Status, NumberOfErrors
            FROM AsyncApexJob
            WHERE Id = :jobId
        ];
        System.assertEquals('Completed', job.Status);
        System.assertEquals(0, job.NumberOfErrors);
    }

    @testSetup
    static void setupTestData() {
        // Create common test data for all test methods
        List<Account> accounts = new List<Account>();
        for(Integer i = 0; i < 50; i++) {
            accounts.add(new Account(Name = 'Setup Account ' + i));
        }
        insert accounts;
    }
}
```

Queueable Apex

Question 26: What is Queueable Apex?

Answer: Queueable Apex is an enhanced way of running asynchronous Apex compared to future methods. It combines benefits of both future methods and batch apex.

Advantages over Future Methods:

1. Can pass non-primitive types (sObjects, custom types)
2. Returns a job ID for monitoring
3. Supports job chaining
4. Better error handling

Basic Example:

```
public class AccountQueueable implements Queueable {
    private List<Account> accounts;
    private String industry;

    // Constructor - can pass complex objects
    public AccountQueueable(List<Account> accs, String ind) {
        this.accounts = accs;
        this.industry = ind;
    }

    // Execute method
    public void execute(QueueableContext context) {
        System.debug('Job ID: ' + context.getJobId());

        for(Account acc : accounts) {
            acc.Industry = industry;
        }

        update accounts;
        System.debug('Processed ' + accounts.size() + ' accounts');
    }
}

// Execute queueable
List<Account> accountsToProcess = [SELECT Id FROM Account LIMIT 100];
AccountQueueable job = new AccountQueueable(accountsToProcess, 'Technolog
Id jobId = System.enqueueJob(job);
System.debug('Queued Job ID: ' + jobId);
```

Question 27: How does Queueable differ from Future methods?

Answer:

Feature	Future Method	Queueable Apex
Parameters	Primitives only	Any type (sObjects, custom)
Job ID	No	Yes
Chaining	No	Yes
Monitoring	Difficult	Easy with job ID
Implementation	@future	Implements Queueable

Basic Example:

```
// Future Method Limitations
public class FutureExample {
    // Can only pass primitives
    @future
    public static void processAccount(Id accountId, String name) {
        Account acc = new Account(Id = accountId, Name = name);
        update acc;
    }

    // WRONG - Cannot pass sObject
    // @future
    // public static void process(Account acc) { } // Error!
}

// Queueable Advantages
public class QueueableExample implements Queueable {
    private Account account;
    private Contact contact;
    private List<Opportunity> opportunities;

    // Can pass any type!
    public QueueableExample(Account acc, Contact con, List<Opportunity> c
        this.account = acc;
        this.contact = con;
        this.opportunities = opps;
    }

    public void execute(QueueableContext context) {
        // Get job ID
        Id jobId = context.getJobId();
```

```
        System.debug('Processing job: ' + jobId);

        // Process complex objects
        account.Industry = 'Technology';
        update account;

        contact.Title = 'Manager';
        update contact;

        update opportunities;
    }
}

// Usage
Account acc = new Account(Id = '001xx000003DGb2AAG');
Contact con = new Contact(Id = '003xx000003DGb2AAG');
List<Opportunity> opps = [SELECT Id FROM Opportunity WHERE AccountId = :a

QueueableExample job = new QueueableExample(acc, con, opps);
Id jobId = System.enqueueJob(job);
```

Question 28: How do you chain Queueable jobs?

Answer: Chain jobs by calling System.enqueueJob() from within the execute() method. Only one child job can be enqueued per parent job.

Basic Example:

```
// First Job
public class FirstJob implements Queueable {
    public void execute(QueueableContext context) {
        System.debug('First job executing');

        // Process accounts
        List<Account> accounts = [SELECT Id FROM Account LIMIT 10];
        for(Account acc : accounts) {
            acc.Rating = 'Hot';
        }
        update accounts;

        // Chain to next job
        System.enqueueJob(new SecondJob());
    }
```

```
}

// Second Job
public class SecondJob implements Queueable {
    public void execute(QueueableContext context) {
        System.debug('Second job executing');

        // Process contacts
        List<Contact> contacts = [SELECT Id FROM Contact LIMIT 10];
        for(Contact con : contacts) {
            con.LeadSource = 'Web';
        }
        update contacts;

        // Chain to third job
        System.enqueueJob(new ThirdJob());
    }
}

// Third Job
public class ThirdJob implements Queueable {
    public void execute(QueueableContext context) {
        System.debug('Third job executing');

        // Process opportunities
        List<Opportunity> opps = [SELECT Id FROM Opportunity LIMIT 10];
        for(Opportunity opp : opps) {
            opp.StageName = 'Qualification';
        }
        update opps;

        System.debug('All jobs completed');
    }
}

// Start the chain
System.enqueueJob(new FirstJob());
```

Question 29: How do you implement dynamic job chaining in Queueable?

Answer: Pass the next job class as a parameter to make chaining flexible and configurable.

Basic Example:

```
// Generic Queueable with dynamic chaining
public class DynamicQueueable implements Queueable {
    private String processType;
    private Type nextJobType;

    public DynamicQueueable(String type, Type nextJob) {
        this.processType = type;
        this.nextJobType = nextJob;
    }

    public void execute(QueueableContext context) {
        System.debug('Processing: ' + processType);

        // Process based on type
        if(processType == 'Accounts') {
            processAccounts();
        } else if(processType == 'Contacts') {
            processContacts();
        } else if(processType == 'Opportunities') {
            processOpportunities();
        }

        // Chain to next job if specified
        if(nextJobType != null) {
            Queueable nextJob = (Queueable)nextJobType.newInstance();
            System.enqueueJob(nextJob);
        }
    }

    private void processAccounts() {
        List<Account> accounts = [SELECT Id FROM Account LIMIT 50];
        for(Account acc : accounts) {
            acc.Rating = 'Hot';
        }
        update accounts;
    }

    private void processContacts() {
        List<Contact> contacts = [SELECT Id FROM Contact LIMIT 50];
        for(Contact con : contacts) {
            con.LeadSource = 'Web';
        }
        update contacts;
    }
```

```
    private void processOpportunities() {
        List<Opportunity> opps = [SELECT Id FROM Opportunity LIMIT 50];
        for(Opportunity opp : opps) {
            opp.StageName = 'Qualification';
        }
        update opps;
    }
}

// Specific job classes
public class AccountJob extends DynamicQueueable {
    public AccountJob() {
        super('Accounts', ContactJob.class);
    }
}

public class ContactJob extends DynamicQueueable {
    public ContactJob() {
        super('Contacts', OpportunityJob.class);
    }
}

public class OpportunityJob extends DynamicQueueable {
    public OpportunityJob() {
        super('Opportunities', null); // Last in chain
    }
}

// Start dynamic chain
System.enqueueJob(new AccountJob());
```

Question 30: What are the limits for Queueable Apex?

Answer:

- Maximum 50 jobs can be enqueued per transaction
- Only 1 child job can be chained from parent
- Default chain depth: 5 (can be increased)
- Shares the 250,000 daily async limit with other async methods

Basic Example:

```
public class QueueableLimits {
    // Maximum 50 enqueued in one transaction
    public static void enqueueMultiple() {
        for(Integer i = 0; i < 50; i++) {
            System.enqueueJob(new ProcessJob(i));
        }
        // System.enqueueJob(new ProcessJob(51)); // Would fail!
    }

    // Can only chain one job at a time
    public class ChainJob implements Queueable {
        public void execute(QueueableContext context) {
            // Process records

            // Can enqueue only ONE child job
            System.enqueueJob(new NextJob());

            // System.enqueueJob(new AnotherJob()); // Would fail!
        }
    }

    public class NextJob implements Queueable {
        public void execute(QueueableContext context) {
            System.debug('Next job');
        }
    }

    // Check limits
    public static void checkLimits() {
        System.debug('Queueable Jobs Used: ' + Limits.getQueueableJobs())
        System.debug('Queueable Jobs Limit: ' + Limits.getLimitQueueable
    }
}

public class ProcessJob implements Queueable {
    private Integer jobNumber;

    public ProcessJob(Integer num) {
        this.jobNumber = num;
    }

    public void execute(QueueableContext context) {
        System.debug('Processing job ' + jobNumber);
```

```
    }
}
```

Question 31: How do you test Queueable Apex?

Answer: Use Test.startTest() and Test.stopTest() to execute queueable jobs synchronously in tests.

Basic Example:

```
// Queueable Class
public class AccountUpdateQueueable implements Queueable {
    private List<Id> accountIds;
    private String industry;

    public AccountUpdateQueueable(List<Id> ids, String ind) {
        this.accountIds = ids;
        this.industry = ind;
    }

    public void execute(QueueableContext context) {
        List<Account> accounts = [SELECT Id FROM Account WHERE Id IN :acc
        for(Account acc : accounts) {
            acc.Industry = industry;
        }
        update accounts;
    }
}

// Test Class
@isTest
public class AccountUpdateQueueableTest {
    @testSetup
    static void setup() {
        List<Account> accounts = new List<Account>();
        for(Integer i = 0; i < 10; i++) {
            accounts.add(new Account(Name = 'Test Account ' + i));
        }
        insert accounts;
    }

    @isTest
    static void testQueueable() {
        List<Account> accounts = [SELECT Id FROM Account];
```

```
        List<Id> accountIds = new List<Id>();
        for(Account acc : accounts) {
            accountIds.add(acc.Id);
        }

        Test.startTest();
        Id jobId = System.enqueueJob(new AccountUpdateQueueable(accountIc
        System.assertNotEquals(null, jobId, 'Job ID should not be null');
        Test.stopTest(); // Queueable executes synchronously here

        // Verify results
        List<Account> updatedAccounts = [SELECT Industry FROM Account];
        for(Account acc : updatedAccounts) {
            System.assertEquals('Technology', acc.Industry, 'Industry shc
        }
    }

    @isTest
    static void testChaining() {
        List<Account> accounts = [SELECT Id FROM Account];

        Test.startTest();
        System.enqueueJob(new ChainedJob1());
        Test.stopTest();

        // Verify all chained jobs executed
        List<Account> results = [SELECT Industry, Rating FROM Account];
        System.assertEquals('Technology', results[0].Industry);
        System.assertEquals('Hot', results[0].Rating);
    }
}

// Chained Jobs for testing
public class ChainedJob1 implements Queueable {
    public void execute(QueueableContext context) {
        List<Account> accounts = [SELECT Id FROM Account LIMIT 1];
        accounts[0].Industry = 'Technology';
        update accounts;

        if(!Test.isRunningTest() || Test.isRunningTest()) {
            System.enqueueJob(new ChainedJob2());
        }
    }
}
```

```
public class ChainedJob2 implements Queueable {
    public void execute(QueueableContext context) {
        List<Account> accounts = [SELECT Id FROM Account LIMIT 1];
        accounts[0].Rating = 'Hot';
        update accounts;
    }
}
```

Apex Scheduler

Question 32: What is Apex Scheduler?

Answer: Apex Scheduler allows you to schedule Apex classes to run at specific times. It uses the Schedulable interface and cron expressions for scheduling.

Components:

1. Implement Schedulable interface
2. Define execute method
3. Schedule using System.schedule() or UI
4. Use cron expressions for timing

Basic Example:

```
// Schedulable class
public class DailyAccountProcessor implements Schedulable {
    public void execute(SchedulableContext sc) {
        System.debug('Scheduled job running');

        // Business logic
        List<Account> accounts = [SELECT Id, LastModifiedDate FROM Accour
                                  WHERE LastModifiedDate = LAST_WEEK];

        for(Account acc : accounts) {
            acc.Description = 'Processed on ' + System.today();
        }

        update accounts;

        System.debug('Processed ' + accounts.size() + ' accounts');
    }
```

```
}

// Schedule the job programmatically
public class ScheduleJob {
    public static void scheduleDaily() {
        DailyAccountProcessor job = new DailyAccountProcessor();

        // Cron expression: "Seconds Minutes Hours Day_of_month Month Day
        // Run every day at 2:00 AM
        String cronExp = '0 0 2 * * ?';

        String jobName = 'Daily Account Processor';
        String jobId = System.schedule(jobName, cronExp, job);

        System.debug('Scheduled job ID: ' + jobId);
    }
}
```

Question 33: How do cron expressions work?

Answer: Cron expressions define when a scheduled job runs. Format: `Seconds Minutes Hours Day_of_Month Month Day_of_Week Optional_Year`

Cron Expression Components:

- **Seconds:** 0-59
- **Minutes:** 0-59
- **Hours:** 0-23
- **Day of Month:** 1-31
- **Month:** 1-12 or JAN-DEC
- **Day of Week:** 1-7 or SUN-SAT (1=Sunday)
- **Year:** Optional (1970-2099)

Basic Example:

```
public class CronExpressions {
    public static void scheduleExamples() {
        DailyAccountProcessor job = new DailyAccountProcessor();

        // Every day at 12:00 PM
        System.schedule('Daily at Noon', '0 0 12 * * ?', job);

        // Every Monday at 8:00 AM
```

```
        System.schedule('Monday Morning', '0 0 8 ? * MON', job);

        // Every hour
        System.schedule('Every Hour', '0 0 * * * ?', job);

        // Every 15 minutes
        System.schedule('Every 15 Min', '0 0,15,30,45 * * * ?', job);

        // First day of every month at 6:00 AM
        System.schedule('Monthly', '0 0 6 1 * ?', job);

        // Every weekday (Mon-Fri) at 9:00 AM
        System.schedule('Weekdays', '0 0 9 ? * MON-FRI', job);

        // Last day of every month
        System.schedule('Month End', '0 0 23 L * ?', job);

        // Every Sunday at 11:59 PM
        System.schedule('Sunday Night', '0 59 23 ? * SUN', job);
    }

    // Common cron patterns
    public static String getHourlyPattern() {
        return '0 0 * * * ?'; // Every hour
    }

    public static String getDailyPattern(Integer hour, Integer minute) {
        return '0 ' + minute + ' ' + hour + ' * * ?'; // Daily at specifi
    }

    public static String getWeeklyPattern(String dayOfWeek, Integer hour)
        return '0 0 ' + hour + ' ? * ' + dayOfWeek; // Weekly on specific
    }
}
```

Question 34: How do you schedule a job using the UI?

Answer: Setup → Apex Classes → Schedule Apex

Steps:

1. Go to Setup
2. Search for "Apex Classes"

3. Click "Schedule Apex"
4. Select class that implements Schedulable
5. Choose frequency (Weekly, Monthly, etc.)
6. Set time and dates
7. Save

Programmatic Alternative:

```
public class ScheduleFromUI {
    // This class can be scheduled from UI
    public class UISchedulableJob implements Schedulable {
        public void execute(SchedulableContext sc) {
            // Your logic here
            processRecords();
        }

        private void processRecords() {
            List<Account> accounts = [SELECT Id FROM Account WHERE Create
            for(Account acc : accounts) {
                acc.Rating = 'Hot';
            }
            update accounts;
        }
    }

    // Or schedule programmatically
    public static void scheduleJob() {
        UISchedulableJob job = new UISchedulableJob();

        // Schedule to run daily at 1:00 AM
        String cronExp = '0 0 1 * * ?';
        System.schedule('Daily Job', cronExp, job);
    }
}
```

Question 35: How do you monitor and abort scheduled jobs?

Answer: Query CronTrigger object to monitor, use System.abortJob() to abort.

Basic Example:

```
public class ScheduledJobManagement {
    // Monitor all scheduled jobs
    public static void listScheduledJobs() {
        List<CronTrigger> scheduledJobs = [
            SELECT Id, CronJobDetail.Name, CronJobDetail.JobType,
                   NextFireTime, PreviousFireTime, State,
                   CronExpression, CreatedDate
            FROM CronTrigger
            WHERE CronJobDetail.JobType = '7' // Scheduled Apex
            ORDER BY NextFireTime
        ];

        System.debug('=== Scheduled Jobs ===');
        for(CronTrigger job : scheduledJobs) {
            System.debug('Name: ' + job.CronJobDetail.Name);
            System.debug('Next Run: ' + job.NextFireTime);
            System.debug('Last Run: ' + job.PreviousFireTime);
            System.debug('State: ' + job.State);
            System.debug('Cron: ' + job.CronExpression);
            System.debug('---');
        }
    }

    // Abort a scheduled job by name
    public static void abortJobByName(String jobName) {
        List<CronTrigger> jobs = [
            SELECT Id
            FROM CronTrigger
            WHERE CronJobDetail.Name = :jobName
        ];

        if(!jobs.isEmpty()) {
            System.abortJob(jobs[0].Id);
            System.debug('Aborted job: ' + jobName);
        } else {
            System.debug('Job not found: ' + jobName);
        }
    }

    // Abort all scheduled jobs for a specific class
    public static void abortAllJobsForClass(String className) {
        List<CronTrigger> jobs = [
            SELECT Id, CronJobDetail.Name
            FROM CronTrigger
```

```
            WHERE CronJobDetail.Name LIKE :('%' + className + '%')
        ];

        for(CronTrigger job : jobs) {
            System.abortJob(job.Id);
            System.debug('Aborted: ' + job.CronJobDetail.Name);
        }

        System.debug('Total aborted: ' + jobs.size());
    }

    // Check if a job is already scheduled
    public static Boolean isJobScheduled(String jobName) {
        Integer count = [
            SELECT COUNT()
            FROM CronTrigger
            WHERE CronJobDetail.Name = :jobName
        ];

        return count > 0;
    }

    // Reschedule a job (abort old, schedule new)
    public static void rescheduleJob(String oldJobName, String newCronExp
        // Abort existing job
        abortJobByName(oldJobName);

        // Schedule new job
        DailyAccountProcessor job = new DailyAccountProcessor();
        System.schedule(oldJobName, newCronExp, job);

        System.debug('Job rescheduled: ' + oldJobName);
    }
}
```

Question 36: Can you schedule a Batch Apex job?

Answer: Yes, either by implementing Schedulable interface or using System.scheduleBatch().

Basic Example:

```
// Method 1: Schedulable class that calls Batch
public class ScheduledBatch implements Schedulable {
```

```
    public void execute(SchedulableContext sc) {
        // Execute batch from scheduled job
        AccountBatch batch = new AccountBatch();
        Database.executeBatch(batch, 200);
    }
}

// Schedule it
String cronExp = '0 0 2 * * ?'; // 2:00 AM daily
System.schedule('Scheduled Batch Job', cronExp, new ScheduledBatch());

// Method 2: Using System.scheduleBatch() directly
public class DirectScheduleBatch {
    public static void scheduleAccountBatch() {
        AccountBatch batch = new AccountBatch();

        // Schedule to run after 5 minutes
        Datetime scheduledTime = Datetime.now().addMinutes(5);
        String jobId = System.scheduleBatch(batch, 'Scheduled Account Bat

        System.debug('Batch scheduled with Job ID: ' + jobId);
    }

    public static void scheduleDailyBatch() {
        AccountBatch batch = new AccountBatch();

        // Calculate tomorrow at 3:00 AM
        DateTime tomorrow = DateTime.now().addDays(1);
        Time scheduledTime = Time.newInstance(3, 0, 0, 0);
        DateTime scheduledDateTime = DateTime.newInstance(
            tomorrow.date(),
            scheduledTime
        );

        Integer minutesFromNow = Integer.valueOf(
            (scheduledDateTime.getTime() - DateTime.now().getTime()) / (1
        );

        String jobId = System.scheduleBatch(batch, 'Daily Batch', minutes
        System.debug('Scheduled for: ' + scheduledDateTime);
    }
}

// Batch class
```

```
public class AccountBatch implements Database.Batchable<sObject> {
    public Database.QueryLocator start(Database.BatchableContext bc) {
        return Database.getQueryLocator('SELECT Id FROM Account');
    }

    public void execute(Database.BatchableContext bc, List<Account> scope
        for(Account acc : scope) {
            acc.Rating = 'Hot';
        }
        update scope;
    }

    public void finish(Database.BatchableContext bc) {
        System.debug('Batch completed');
    }
}
```

Question 37: What are the limits for Scheduled Apex?

Answer:

- Maximum 100 scheduled jobs at a time
- Jobs may not run at exact scheduled time (depends on resources)
- Cannot schedule from triggers (risk of exceeding limit)
- Count scheduled jobs before adding new ones

Basic Example:

```
public class ScheduleLimits {
    // Check current scheduled job count
    public static Integer getScheduledJobCount() {
        Integer count = [
            SELECT COUNT()
            FROM CronTrigger
            WHERE CronJobDetail.JobType = '7'
        ];

        System.debug('Current scheduled jobs: ' + count);
        return count;
    }

    // Safe scheduling - check limit first
    public static void safeSchedule(String jobName, String cronExp, Schec
```

```
        Integer currentCount = getScheduledJobCount();

        if(currentCount >= 100) {
            System.debug('ERROR: Cannot schedule - limit reached (100 job
            return;
        }

        // Check if job already exists
        Integer existingJobs = [
            SELECT COUNT()
            FROM CronTrigger
            WHERE CronJobDetail.Name = :jobName
        ];

        if(existingJobs > 0) {
            System.debug('Job already scheduled: ' + jobName);
            return;
        }

        // Schedule the job
        String jobId = System.schedule(jobName, cronExp, job);
        System.debug('Successfully scheduled: ' + jobName + ' (ID: ' + jo
    }

    // Clean up old scheduled jobs
    public static void cleanupOldJobs() {
        List<CronTrigger> oldJobs = [
            SELECT Id, CronJobDetail.Name
            FROM CronTrigger
            WHERE CreatedDate < LAST_MONTH
            ORDER BY CreatedDate
        ];

        Integer cleaned = 0;
        for(CronTrigger job : oldJobs) {
            System.abortJob(job.Id);
            cleaned++;
        }

        System.debug('Cleaned up ' + cleaned + ' old scheduled jobs');
    }
}
```

Question 38: How do you test Scheduled Apex?

Answer: Use Test.startTest() and Test.stopTest() to execute scheduled job synchronously in tests.

Basic Example:

```
// Schedulable Class
public class DailyProcessor implements Schedulable {
    public void execute(SchedulableContext sc) {
        List<Account> accounts = [SELECT Id FROM Account WHERE CreatedDat
        for(Account acc : accounts) {
            acc.Description = 'Processed by scheduler';
        }
        update accounts;
    }
}

// Test Class
@isTest
public class DailyProcessorTest {
    @testSetup
    static void setup() {
        List<Account> accounts = new List<Account>();
        for(Integer i = 0; i < 5; i++) {
            accounts.add(new Account(Name = 'Test Account ' + i));
        }
        insert accounts;
    }

    @isTest
    static void testScheduledJob() {
        // Get test data
        List<Account> accounts = [SELECT Id FROM Account];
        System.assertEquals(5, accounts.size(), 'Should have 5 test accou

        Test.startTest();
        // Schedule the job
        String cronExp = '0 0 0 15 3 ? 2025'; // Specific date in past/fu
        String jobId = System.schedule('Test Scheduled Job', cronExp, new

        // Verify job was scheduled
        CronTrigger ct = [
            SELECT Id, CronExpression, TimesTriggered, NextFireTime
            FROM CronTrigger
```

```
            WHERE Id = :jobId
        ];

        System.assertEquals(cronExp, ct.CronExpression);
        System.assertEquals(0, ct.TimesTriggered);

        Test.stopTest(); // Scheduled job executes here

        // Verify results
        List<Account> updatedAccounts = [SELECT Description FROM Account]
        for(Account acc : updatedAccounts) {
            System.assertEquals('Processed by scheduler', acc.Descriptior
        }
    }

    @isTest
    static void testScheduleAbort() {
        Test.startTest();
        String jobId = System.schedule('Abort Test Job', '0 0 0 15 3 ? 20

        // Abort the job
        System.abortJob(jobId);
        Test.stopTest();

        // Verify job was aborted
        List<CronTrigger> jobs = [SELECT Id FROM CronTrigger WHERE Id = :
        System.assertEquals(0, jobs.size(), 'Job should be aborted');
    }
}
```

Advanced Test Classes

Question 39: What is @testSetup annotation?

Answer: @testSetup creates test data once that's available to all test methods in the class, improving test performance.

Benefits:

- Creates test data once (not per method)
- Reduces test execution time
- Data is rolled back after each test method

- Each method gets fresh copy of data

Basic Example:

```
@isTest
public class TestSetupExample {
    // Create test data once for all methods
    @testSetup
    static void setupTestData() {
        // This runs once before all test methods
        List<Account> accounts = new List<Account>();
        for(Integer i = 0; i < 100; i++) {
            accounts.add(new Account(
                Name = 'Test Account ' + i,
                Industry = 'Technology',
                AnnualRevenue = 100000
            ));
        }
        insert accounts;

        List<Contact> contacts = new List<Contact>();
        for(Account acc : accounts) {
            contacts.add(new Contact(
                FirstName = 'Test',
                LastName = 'Contact',
                AccountId = acc.Id,
                Email = 'test@example.com'
            ));
        }
        insert contacts;

        System.debug('Test data created');
    }

    @isTest
    static void testMethod1() {
        // Access test data created in @testSetup
        List<Account> accounts = [SELECT Id, Name FROM Account];
        System.assertEquals(100, accounts.size());

        // Modify data
        accounts[0].Name = 'Modified';
        update accounts[0];
    }
```

```
    @isTest
    static void testMethod2() {
        // Gets fresh copy of data (changes from testMethod1 are rolled b
        List<Account> accounts = [SELECT Id, Name FROM Account];
        System.assertEquals(100, accounts.size());
        System.assertNotEquals('Modified', accounts[0].Name); // Reset to
    }

    @isTest
    static void testMethod3() {
        // Can also access contacts
        List<Contact> contacts = [SELECT Id FROM Contact];
        System.assertEquals(100, contacts.size());
    }
}
```

Question 40: What is a Test Data Factory?

Answer: A Test Data Factory is a centralized class that creates test data for multiple test classes, promoting code reuse and maintainability.

Basic Example:

```
@isTest
public class TestDataFactory {
    // Create single account
    public static Account createAccount(Boolean doInsert) {
        Account acc = new Account(
            Name = 'Test Account',
            Industry = 'Technology',
            AnnualRevenue = 1000000
        );

        if(doInsert) {
            insert acc;
        }

        return acc;
    }

    // Create multiple accounts
    public static List<Account> createAccounts(Integer count, Boolean doI
```

```
    List<Account> accounts = new List<Account>();

    for(Integer i = 0; i < count; i++) {
        accounts.add(new Account(
            Name = 'Test Account ' + i,
            Industry = 'Technology',
            Phone = '555-000' + i
        ));
    }

    if(doInsert) {
        insert accounts;
    }

    return accounts;
}

// Create contact with account
public static Contact createContact(Id accountId, Boolean doInsert) {
    Contact con = new Contact(
        FirstName = 'Test',
        LastName = 'Contact',
        AccountId = accountId,
        Email = 'test@example.com'
    );

    if(doInsert) {
        insert con;
    }

    return con;
}

// Create multiple contacts
public static List<Contact> createContacts(Id accountId, Integer cour
    List<Contact> contacts = new List<Contact>();

    for(Integer i = 0; i < count; i++) {
        contacts.add(new Contact(
            FirstName = 'Test' + i,
            LastName = 'Contact' + i,
            AccountId = accountId,
            Email = 'test' + i + '@example.com'
        ));
```

```
        }

        if(doInsert) {
            insert contacts;
        }

        return contacts;
    }

    // Create opportunity
    public static Opportunity createOpportunity(Id accountId, Boolean doI
        Opportunity opp = new Opportunity(
            Name = 'Test Opportunity',
            AccountId = accountId,
            StageName = 'Prospecting',
            CloseDate = Date.today().addDays(30),
            Amount = 50000
        );

        if(doInsert) {
            insert opp;
        }

        return opp;
    }
}

// Using Test Data Factory
@isTest
public class MyTestClass {
    @isTest
    static void testBusinessLogic() {
        // Use factory to create test data
        Account acc = TestDataFactory.createAccount(true);
        List<Contact> contacts = TestDataFactory.createContacts(acc.Id, 5
        Opportunity opp = TestDataFactory.createOpportunity(acc.Id, true)

        // Test logic
        Test.startTest();
        // Your test logic here
        Test.stopTest();

        // Assertions
        System.assertEquals(5, [SELECT COUNT() FROM Contact WHERE Account
```

```
    }
}
```

Question 41: What are best practices for writing test classes?

Answer:

Best Practices:

1. Use @testSetup for common data
2. Use Test Data Factory
3. Test both positive and negative scenarios
4. Test bulk operations (200+ records)
5. Use System.assert() statements
6. Don't use @isTest(SeeAllData=true)
7. Use Test.startTest() and Test.stopTest()
8. Test with different user permissions
9. Aim for 90%+ code coverage (minimum 75%)
10. Test exception handling

Basic Example:

```
// Test Class with Best Practices
@isTest
public class BestPracticesTest {
    // 1. Use @testSetup
    @testSetup
    static void setup() {
        TestDataFactory.createAccounts(200, true);
    }

    // 2. Test positive scenario
    @isTest
    static void testPositiveScenario() {
        List<Account> accounts = [SELECT Id FROM Account];

        Test.startTest();
        for(Account acc : accounts) {
            acc.Rating = 'Hot';
        }
        update accounts;
        Test.stopTest();
```

```
    // Assert success
    List<Account> updated = [SELECT Rating FROM Account];
    for(Account acc : updated) {
        System.assertEquals('Hot', acc.Rating);
    }
}

// 3. Test negative scenario
@isTest
static void testNegativeScenario() {
    Account acc = new Account(); // No name - should fail

    Test.startTest();
    try {
        insert acc;
        System.assert(false, 'Should have thrown exception');
    } catch(DmlException e) {
        System.assert(e.getMessage().contains('REQUIRED_FIELD_MISSING
    }
    Test.stopTest();
}

// 4. Test bulk operations
@isTest
static void testBulkOperation() {
    List<Account> accounts = [SELECT Id FROM Account];
    System.assertEquals(200, accounts.size(), 'Should have 200 accour

    Test.startTest();
    // Process all 200 records
    for(Account acc : accounts) {
        acc.Industry = 'Technology';
    }
    update accounts;
    Test.stopTest();

    // Verify bulk update
    List<Account> updated = [SELECT Industry FROM Account];
    System.assertEquals(200, updated.size());
}

// 5. Test with different user (System.runAs)
@isTest
static void testWithDifferentUser() {
```

```
    // Create test user
    Profile p = [SELECT Id FROM Profile WHERE Name = 'Standard User'
    User testUser = new User(
        FirstName = 'Test',
        LastName = 'User',
        Email = 'testuser@example.com',
        Username = 'testuser' + DateTime.now().getTime() + '@example.
        Alias = 'tuser',
        TimeZoneSidKey = 'America/Los_Angeles',
        LocaleSidKey = 'en_US',
        EmailEncodingKey = 'UTF-8',
        ProfileId = p.Id,
        LanguageLocaleKey = 'en_US'
    );
    insert testUser;

    System.runAs(testUser) {
        // Test as different user
        Account acc = new Account(Name = 'Test');
        insert acc;
        System.assertEquals(testUser.Id, [SELECT CreatedById FROM Acc
    }
}

// 6. Test exception handling
@isTest
static void testExceptionHandling() {
    Test.startTest();
    try {
        // Cause exception
        List<Account> accounts = [SELECT Id FROM Account WHERE Name =
        Account acc = accounts[0]; // Will throw exception
        System.assert(false, 'Should have thrown exception');
    } catch(ListException e) {
        System.debug('Caught expected exception');
        System.assert(true);
    }
    Test.stopTest();
}

// 7. Test with governor limits
@isTest
static void testGovernorLimits() {
    Test.startTest();
```

```
        // Check initial limits
        System.debug('Initial queries: ' + Limits.getQueries());

        List<Account> accounts = [SELECT Id FROM Account LIMIT 100];

        // Verify we haven't exceeded limits
        System.assert(Limits.getQueries() < Limits.getLimitQueries());
        System.assert(Limits.getDmlStatements() < Limits.getLimitDmlState

        Test.stopTest();
    }
}
```

Question 42: How do you test callouts in Apex?

Answer: Use HttpCalloutMock interface to create mock responses for testing HTTP callouts.

Basic Example:

```
// Callout Class
public class ExternalService {
    @future(callout=true)
    public static void makeCallout(String endpoint) {
        HttpRequest req = new HttpRequest();
        req.setEndpoint(endpoint);
        req.setMethod('GET');

        Http http = new Http();
        HttpResponse res = http.send(req);

        if(res.getStatusCode() == 200) {
            System.debug('Success: ' + res.getBody());
        }
    }
}

// Mock Callout Class
@isTest
global class MockHttpResponse implements HttpCalloutMock {
    global HttpResponse respond(HttpRequest req) {
        // Create mock response
        HttpResponse res = new HttpResponse();
```

```
        res.setHeader('Content-Type', 'application/json');
        res.setBody('{"status":"success","data":"test"}');
        res.setStatusCode(200);
        return res;
    }
}

// Test Class
@isTest
public class ExternalServiceTest {
    @isTest
    static void testCallout() {
        // Set mock callout
        Test.setMock(HttpCalloutMock.class, new MockHttpResponse());

        Test.startTest();
        ExternalService.makeCallout('https://api.example.com/data');
        Test.stopTest();

        // Verify callout was made (check debug logs or results)
    }

    @isTest
    static void testCalloutError() {
        // Mock error response
        Test.setMock(HttpCalloutMock.class, new MockHttpErrorResponse());

        Test.startTest();
        ExternalService.makeCallout('https://api.example.com/data');
        Test.stopTest();
    }
}

// Mock Error Response
@isTest
global class MockHttpErrorResponse implements HttpCalloutMock {
    global HttpResponse respond(HttpRequest req) {
        HttpResponse res = new HttpResponse();
        res.setStatusCode(500);
        res.setStatus('Internal Server Error');
        return res;
    }
}
```

Question 43: How do you achieve maximum code coverage?

Answer:

Strategies:

1. Test all methods and branches
2. Test positive and negative scenarios
3. Test exception handling
4. Test bulk operations
5. Test different user contexts
6. Use Test.startTest() and Test.stopTest()

Basic Example:

```
// Class to test
public class AccountService {
    public static void updateIndustry(List<Account> accounts, String indu
        if(accounts == null || accounts.isEmpty()) {
            throw new AccountException('No accounts provided');
        }

        for(Account acc : accounts) {
            if(String.isBlank(industry)) {
                acc.Industry = 'Other';
            } else {
                acc.Industry = industry;
            }
        }

        try {
            update accounts;
        } catch(DmlException e) {
            System.debug('Error updating accounts: ' + e.getMessage());
            throw new AccountException('Failed to update accounts: ' + e.
        }
    }

    public class AccountException extends Exception {}
}

// Comprehensive Test Class
@isTest
public class AccountServiceTest {
```

```
@testSetup
static void setup() {
    TestDataFactory.createAccounts(200, true);
}

// Test positive scenario
@isTest
static void testUpdateIndustry_Success() {
    List<Account> accounts = [SELECT Id FROM Account];

    Test.startTest();
    AccountService.updateIndustry(accounts, 'Technology');
    Test.stopTest();

    List<Account> updated = [SELECT Industry FROM Account];
    for(Account acc : updated) {
        System.assertEquals('Technology', acc.Industry);
    }
}

// Test null industry (branch coverage)
@isTest
static void testUpdateIndustry_NullIndustry() {
    List<Account> accounts = [SELECT Id FROM Account LIMIT 10];

    Test.startTest();
    AccountService.updateIndustry(accounts, null);
    Test.stopTest();

    List<Account> updated = [SELECT Industry FROM Account WHERE Id IN
    for(Account acc : updated) {
        System.assertEquals('Other', acc.Industry);
    }
}

// Test empty list exception
@isTest
static void testUpdateIndustry_EmptyList() {
    Test.startTest();
    try {
        AccountService.updateIndustry(new List<Account>(), 'Tech');
        System.assert(false, 'Should have thrown exception');
    } catch(AccountService.AccountException e) {
        System.assert(e.getMessage().contains('No accounts'));
```

```
        }
        Test.stopTest();
    }

    // Test null list exception
    @isTest
    static void testUpdateIndustry_NullList() {
        Test.startTest();
        try {
            AccountService.updateIndustry(null, 'Tech');
            System.assert(false, 'Should have thrown exception');
        } catch(AccountService.AccountException e) {
            System.assert(true);
        }
        Test.stopTest();
    }

    // Test bulk operation
    @isTest
    static void testUpdateIndustry_Bulk() {
        List<Account> accounts = [SELECT Id FROM Account];
        System.assertEquals(200, accounts.size());

        Test.startTest();
        AccountService.updateIndustry(accounts, 'Finance');
        Test.stopTest();

        Integer count = [SELECT COUNT() FROM Account WHERE Industry = 'Fi
        System.assertEquals(200, count);
    }
}
```

Summary

This guide covered:

✅ **Governor Limits** - Types, monitoring, and best practices
✅ **Asynchronous Apex** - Overview and when to use
✅ **Future Methods** - Simple async processing with callouts
✅ **Batch Apex** - Processing millions of records
✅ **Queueable Apex** - Enhanced async with chaining

✅ **Apex Scheduler** - Running jobs at specific times
✅ **Advanced Testing** - @testSetup, factories, best practices

Key Takeaways:

1. Always bulkify your code to avoid governor limits
2. Use appropriate async method based on requirements
3. Monitor limits using Limits class
4. Test thoroughly with positive, negative, and bulk scenarios
5. Use Test Data Factory for reusable test data
6. Aim for 90%+ code coverage
7. Chain queueable jobs for sequential processing
8. Use cron expressions for flexible scheduling
9. Handle errors gracefully in async methods
10. Test async methods using Test.startTest/stopTest

Document Information:

- Topics Covered: 43 detailed questions
- Interview Questions: 43+ with comprehensive answers
- Code Examples: 100+ practical examples
- Coverage: Governor Limits, Future, Batch, Queueable, Scheduler, Testing

Good luck with your Salesforce interviews! 🚀