Salesforce Development Course - Interview Questions Guide

Complete Interview Questions with Basic Examples

This comprehensive guide contains interview questions for each video/topic in the Salesforce Development Course Premium Playlist by Salesforce Hulk (S2 Labs).

Module 1: Introduction to Apex

Video 1: What is Apex?

Interview Questions:

1. **Q: What is Apex in Salesforce?**
    - **A:** Apex is a strongly-typed, object-oriented programming language developed by Salesforce that allows developers to execute flow and transaction control statements on the Salesforce platform server. It has Java-like syntax and is used to add custom business logic to applications.

    Basic Example:

    ```
    // Simple Apex class
    public class HelloWorld {
        public static void sayHello() {
            System.debug('Hello, Salesforce!');
        }
    }
    ```

2. **Q: Where is Apex used in Salesforce?**
    - **A:** Apex is used in:
        - Triggers (to perform operations before/after DML operations)
        - Custom controllers and extensions for Visualforce pages
        - Web services (REST/SOAP APIs)

        - Batch processing
        - Scheduled jobs
        - Email services

Basic Example:

```
// Apex in a trigger
trigger AccountTrigger on Account (before insert) {
    for(Account acc : Trigger.new) {
        acc.Description = 'Created via trigger';
    }
}
```

3. **Q: What are the key features of Apex?**
    - **A:** Key features include:
        - Strongly typed
        - Case-insensitive
        - Multitenant aware
        - Integrated with database
        - Easy to use and test
        - Versioned

Basic Example:

```
// Demonstrating strong typing
Integer count = 10;  // Must declare type
String name = 'John';  // Case-insensitive: STRING, string, String a
```

4. **Q: How does Apex differ from Java?**
    - **A:** While Apex syntax is similar to Java:
        - Apex runs on Salesforce servers only
        - Apex is case-insensitive
        - Apex has built-in governor limits
        - Apex has native database integration
        - Apex doesn't support file I/O operations
5. **Q: What is the Apex class in Salesforce?**
    - **A:** An Apex class is a template or blueprint from which objects are created. It contains methods and variables.

Basic Example:

```
public class Calculator {
    // Variable
    Integer result;

    // Method
    public Integer add(Integer a, Integer b) {
        result = a + b;
        return result;
    }
}
```

Video 2: Features of Apex

Interview Questions:

1. **Q: Explain why Apex is a strongly-typed language.**
   - **A:** In Apex, every variable must be declared with a specific data type before use. The compiler checks type compatibility at compile time.

   Basic Example:

   ```
   Integer age = 25;   // Correct
   age = 'Twenty';     // Error: Cannot assign String to Integer
   ```

2. **Q: What does it mean that Apex is case-insensitive?**
   - **A:** Apex keywords, variable names, and method names are case-insensitive, meaning `String`, `STRING`, and `string` are treated the same.

   Basic Example:

   ```
   String name = 'John';
   STRING name2 = 'Jane';   // Both are valid
   system.debug(name);      // system, System, SYSTEM all work
   ```

3. **Q: How does Apex support DML operations?**
   - **A:** Apex provides built-in DML statements (insert, update, delete, undelete, upsert, merge) to manipulate database records.

   Basic Example:

```
// Create and insert an account
Account acc = new Account(Name = 'Acme Corp');
insert acc;  // DML insert operation
System.debug('Account created with ID: ' + acc.Id);
```

4. **Q: What is multitenant architecture in Apex?**
   - **A:** Salesforce runs on a shared infrastructure where multiple organizations share the same resources. Apex enforces governor limits to ensure fair resource usage.

   Basic Example:

```
// Governor limits example
List<Account> accounts = [SELECT Id FROM Account LIMIT 50000];
// Limited to prevent one org from using all resources
```

5. **Q: Can Apex execute multiple queries concurrently?**
   - **A:** Yes, Apex can execute multiple SOQL queries and DML statements in a single transaction, subject to governor limits.

   Basic Example:

```
// Multiple operations in one transaction
List<Account> accounts = [SELECT Id, Name FROM Account LIMIT 10];
List<Contact> contacts = [SELECT Id, Name FROM Contact LIMIT 10];

// Both queries execute in the same transaction
System.debug('Accounts: ' + accounts.size());
System.debug('Contacts: ' + contacts.size());
```

Video 3: When to Use Apex

Interview Questions:

1. **Q: When should you use Apex instead of declarative tools?**
   - **A:** Use Apex when:
     - Complex business logic is required
     - Declarative tools (workflows, process builder) can't meet requirements
     - Need to integrate with external systems

- Need custom transaction control
- Require complex data processing

Basic Example:

```
// Complex logic example: Calculate commission based on multiple fact
public class CommissionCalculator {
    public Decimal calculateCommission(Decimal sales, String region,
        Decimal commission = 0;

        if(region == 'North' && experience > 5) {
            commission = sales * 0.15;
        } else if(region == 'South' && experience > 3) {
            commission = sales * 0.12;
        } else {
            commission = sales * 0.10;
        }

        return commission;
    }
}
```

2. **Q: Can you give an example where Apex is necessary?**
   - **A:** Creating complex validation rules, integrating with third-party APIs, or implementing custom batch processing.

Basic Example:

```
// Custom validation that can't be done declaratively
public class AccountValidator {
    public static Boolean validateAccount(Account acc) {
        if(acc.AnnualRevenue > 1000000 && acc.NumberOfEmployees < 10)
            return false; // High revenue with few employees - suspic
        }
        return true;
    }
}
```

3. **Q: What scenarios require Apex triggers?**
   - **A:** Complex before/after logic, updating related records across objects, complex validations, calling external services on record changes.

Basic Example:

```
// Update related contacts when account phone changes
trigger AccountPhoneUpdate on Account (after update) {
    List<Contact> contactsToUpdate = new List<Contact>();

    for(Account acc : Trigger.new) {
        if(acc.Phone != Trigger.oldMap.get(acc.Id).Phone) {
            for(Contact con : [SELECT Id FROM Contact WHERE AccountI
                con.Phone = acc.Phone;
                contactsToUpdate.add(con);
            }
        }
    }

    if(!contactsToUpdate.isEmpty()) {
        update contactsToUpdate;
    }
}
```

4. **Q: When should you NOT use Apex?**
    - **A:** When declarative tools can achieve the same result, as they're easier to maintain and don't require code deployment.
5. **Q: What are the advantages of using Apex for custom business logic?**
    - **A:** Full programming capabilities, version control, ability to handle complex scenarios, integration capabilities, and transaction control.

Video 4: Limitations of Apex

Interview Questions:

1. **Q: What features does Apex NOT support?**
    - **A:** Apex doesn't support:
        - File I/O operations
        - Creating temporary files
        - Multi-threading
        - Access to native OS features
        - Direct UI manipulation (must use Visualforce/Lightning)

Basic Example:

```
// This is NOT possible in Apex:
// File myFile = new File("temp.txt"); // Error: File class not suppo

// Alternative: Use Attachment/ContentVersion for file operations
ContentVersion cv = new ContentVersion();
cv.Title = 'My Document';
cv.PathOnClient = 'document.txt';
cv.VersionData = Blob.valueOf('File content here');
insert cv;
```

2. **Q: What are Governor Limits in Apex?**
    - **A:** Governor limits are runtime limits enforced by Salesforce to ensure efficient use of shared resources in the multitenant environment.

Basic Example:

```
// Governor limit example
// Maximum 100 SOQL queries per transaction

// BAD - Will hit governor limits
for(Integer i = 0; i < 150; i++) {
    Account acc = [SELECT Id FROM Account LIMIT 1]; // Query in loop
}

// GOOD - Single query
List<Account> accounts = [SELECT Id FROM Account LIMIT 150];
for(Account acc : accounts) {
    // Process accounts
}
```

3. **Q: Can you perform file operations in Apex?**
    - **A:** No, direct file I/O is not supported. You must use ContentVersion, ContentDocument, or Attachment objects.
4. **Q: Does Apex support multithreading?**
    - **A:** No, Apex doesn't support creating multiple threads. However, you can use asynchronous Apex (Future, Batch, Queueable) for parallel processing.

Basic Example:

```
// Asynchronous processing using @future
public class AsyncProcessor {
    @future
    public static void processRecords(List<Id> recordIds) {
        // This runs asynchronously
        List<Account> accounts = [SELECT Id, Name FROM Account WHERE
        // Process accounts
    }
}
```

5. **Q: How do you work around Apex limitations?**
    - **A:** Use alternatives like:
        - Asynchronous Apex for heavy processing
        - Platform Events for near real-time processing
        - External services for operations Apex can't perform
        - Static resources for file storage

Video 5: Flow of Action (Compilation and Execution Process)

Interview Questions:

1. **Q: Explain the Apex compilation process.**
    - **A:** When Apex code is saved:
        1. Code is compiled into abstract code
        2. Stored as metadata
        3. On execution, runtime interpreter executes the abstract code
        4. Errors are caught during compilation

    Basic Example:

```
// During save/deployment
public class MyClass {
    public void myMethod() {
        Integer x = 10;  // Compiled to abstract code
        String s = 'Hello';  // Syntax checked here
    }
}
```

2. **Q: What happens during Apex code execution?**

- **A:** The Apex runtime interpreter retrieves compiled code from metadata and executes it on the server. Governor limits are enforced during execution.

3. **Q: What is the role of the Apex runtime interpreter?**
    - **A:** It fetches compiled Apex code and executes it, enforcing governor limits and handling database operations.
4. **Q: How does Salesforce handle Apex errors?**
    - **A:** Compilation errors are caught when saving. Runtime errors are caught during execution and can be handled using try-catch blocks.

Basic Example:

```
public class ErrorHandler {
    public void processAccount() {
        try {
            Account acc = [SELECT Id FROM Account WHERE Name = 'Test'
            acc.Name = 'Updated';
            update acc;
        } catch(QueryException e) {
            System.debug('Query error: ' + e.getMessage());
        } catch(DmlException e) {
            System.debug('DML error: ' + e.getMessage());
        }
    }
}
```

5. **Q: What are the components of the force.com platform in Apex execution?**
    - **A:** Application Server (runs Apex runtime), Database (stores data and metadata), and Client (browser/mobile app).

Module 2: Apex Development Environment

Video 6: Different Tools for Writing Apex

Interview Questions:

1. **Q: What is the Developer Console in Salesforce?**

- **A:** Developer Console is an integrated development environment (IDE) within Salesforce for writing, testing, and debugging Apex code.

Basic Example:

```
// Open Developer Console: Setup → Developer Console
// Create new Apex class
public class DeveloperConsoleExample {
    public static void testMethod() {
        System.debug('Testing in Developer Console');
    }
}
// Execute Anonymous: testMethod();
```

2. **Q: What are the advantages of using Visual Studio Code with Salesforce Extension?**
    - **A:** Benefits include:
        - Advanced code editing features
        - Integration with source control
        - Work with scratch orgs and sandboxes
        - Better code completion and syntax highlighting
        - Command palette for Salesforce operations
3. **Q: How do you use the code editor in Salesforce interface?**
    - **A:** Navigate to Setup → Apex Classes → New, or edit existing classes directly in the browser.

Basic Example:

```
// In Setup → Apex Classes → New
public class SetupEditorExample {
    public void simpleMethod() {
        System.debug('Created in Setup editor');
    }
}
```

4. **Q: What's the difference between Developer Console and VS Code?**
    - **A:** Developer Console is browser-based and simpler, while VS Code offers advanced features like Git integration, better debugging, and works offline with local files.
5. **Q: Which tool is best for debugging Apex?**

- **A:** Developer Console has built-in debugging with checkpoints and logs. VS Code offers Apex Replay Debugger for step-through debugging.

Basic Example:

```
public class DebugExample {
    public static void debugMethod() {
        Integer x = 10;
        System.debug('Value of x: ' + x);  // View in Developer Conso

        String name = 'Test';
        System.debug('Name: ' + name);
    }
}
```

Module 3: Apex Basics

Video 7: Apex Variables

Interview Questions:

1. **Q: What is a variable in Apex?**
    - **A:** A variable is a named storage location that holds a value. Variables must be declared with a data type before use.

Basic Example:

```
public class VariableExample {
    // Instance variable
    String companyName;

    // Static variable
    static Integer count = 0;

    // Local variable in method
    public void setName() {
        String localName = 'Salesforce';
        companyName = localName;
    }
}
```

2. **Q: What are the different types of variables in Apex?**
   - **A:**
     - Local variables (declared in methods)
     - Instance variables (class member variables)
     - Static variables (shared across all instances)

Basic Example:

```
public class VariableTypes {
    // Instance variable - unique to each object
    Integer instanceVar = 100;

    // Static variable - shared by all instances
    static Integer staticVar = 200;

    public void exampleMethod() {
        // Local variable - exists only in this method
        Integer localVar = 300;

        System.debug('Instance: ' + instanceVar);
        System.debug('Static: ' + staticVar);
        System.debug('Local: ' + localVar);
    }
}
```

3. **Q: How do you declare a variable in Apex?**
   - **A:** Syntax: `DataType variableName = value;`

Basic Example:

```
// Variable declarations
Integer age = 25;
String name = 'John';
Boolean isActive = true;
Decimal price = 99.99;
Date today = Date.today();
Account acc = new Account();
```

4. **Q: What is variable initialization?**
   - **A:** Initialization is assigning a value to a variable when it's declared. Uninitialized variables have null value by default.

Basic Example:

```
// Initialized variables
String name = 'Alice';  // Initialized
Integer count;  // Not initialized (null)

System.debug('Name: ' + name);  // Output: Alice
System.debug('Count: ' + count);  // Output: null
```

5. **Q: Can you change the value of a variable after initialization?**
   - **A:** Yes, unless the variable is declared as `final`.

Basic Example:

```
Integer age = 25;
age = 30;  // Value changed

final Integer MAX_SIZE = 100;
// MAX_SIZE = 200;  // Error: Cannot change final variable
```

Video 8: Keywords in Apex

Interview Questions:

1. **Q: What are keywords in Apex?**
   - **A:** Keywords are reserved words with special meaning in the Apex language. They cannot be used as identifiers (variable/method names).

Basic Example:

```
// Common keywords: public, private, class, static, void, return
public class KeywordExample {
    private static final Integer MAX = 100;

    public void myMethod() {
        return;
    }
}
```

2. **Q: List some important Apex keywords.**

- **A:** Common keywords include:
    - Access modifiers: public, private, protected, global
    - Class/method: class, interface, extends, implements
    - Control: if, else, for, while, do, break, continue
    - Other: static, final, virtual, abstract, override

Basic Example:

```
public class KeywordDemo {
    // 'public' and 'static' are keywords
    public static void method1() {
        // 'if', 'else' are keywords
        if(true) {
            System.debug('True');
        } else {
            System.debug('False');
        }
    }
}
```

3. **Q: What is the 'final' keyword in Apex?**

    - **A:** 'final' prevents modification of variables, methods from being overridden, or classes from being extended.

Basic Example:

```
public class FinalExample {
    // Final variable - cannot be changed
    final Integer MAX_VALUE = 1000;

    // Final method - cannot be overridden
    public final void finalMethod() {
        System.debug('This cannot be overridden');
    }
}

// final class CannotExtend { }  // Cannot be extended
```

4. **Q: What is the 'static' keyword?**

    - **A:** 'static' makes variables/methods belong to the class rather than instances. Static members are shared across all instances.

Basic Example:

```
public class StaticExample {
    static Integer counter = 0;

    public StaticExample() {
        counter++;  // Shared by all instances
    }

    public static void displayCounter() {
        System.debug('Counter: ' + counter);
    }
}

// Usage
StaticExample obj1 = new StaticExample();
StaticExample obj2 = new StaticExample();
StaticExample.displayCounter();  // Output: Counter: 2
```

5. **Q: What is the difference between 'public' and 'global' keywords?**
    - **A:** 'public' makes a class/method accessible within the same namespace. 'global' makes it accessible across all namespaces and from managed packages.

Basic Example:

```
// Public class - accessible in same namespace
public class PublicClass {
    public void publicMethod() { }
}

// Global class - accessible everywhere
global class GlobalClass {
    global static void globalMethod() { }
}
```

Video 9: Literals in Apex

Interview Questions:

1. **Q: What are literals in Apex?**

- **A:** Literals are fixed values written directly in code, such as numbers, strings, or boolean values.

Basic Example:

```
Integer num = 100;  // 100 is an integer literal
String text = 'Hello';  // 'Hello' is a string literal
Boolean flag = true;  // true is a boolean literal
Decimal price = 99.99;  // 99.99 is a decimal literal
```

2. **Q: What are the types of literals in Apex?**

    - **A:** Types include:
        - Integer literals: 42
        - Long literals: 42L
        - Decimal/Double literals: 3.14
        - String literals: 'text'
        - Boolean literals: true, false
        - Null literal: null

Basic Example:

```
Integer intLit = 42;
Long longLit = 1000000000L;
Decimal decLit = 3.14159;
String strLit = 'Salesforce';
Boolean boolLit = true;
Account accLit = null;
```

3. **Q: How do you represent string literals in Apex?**

    - **A:** String literals are enclosed in single quotes ('text').

Basic Example:

```
String name = 'John Doe';
String quote = 'He said, \'Hello!\'';  // Escape single quote with ba
String multiline = 'Line 1\nLine 2';  // \n for new line
```

4. **Q: What is the null literal?**

    - **A:** 'null' represents the absence of a value. All variables are initialized to null by default if not assigned a value.

Basic Example:

```
String name;  // null by default
Account acc = null;  // Explicitly set to null

if(name == null) {
    System.debug('Name is null');
}
```

5. **Q: How do you escape special characters in string literals?**
   - **A:** Use backslash () to escape characters like quotes, newlines, tabs.

   Basic Example:

```
String escaped = 'She said, \'Hi!\'';  // Single quote
String newLine = 'First line\nSecond line';  // New line
String tab = 'Column1\tColumn2';  // Tab
String backslash = 'C:\\Users\\File';  // Backslash
```

Video 10: Data Types in Apex

Interview Questions:

1. **Q: What is a data type in Apex?**
   - **A:** A data type defines the kind of value a variable can hold and the operations that can be performed on it.

   Basic Example:

```
Integer count = 10;  // Can store whole numbers
String name = 'Alice';  // Can store text
Boolean isActive = true;  // Can store true/false
```

2. **Q: What are primitive data types in Apex?**
   - **A:** Primitive types are basic data types: Integer, Long, Decimal, Double, Boolean, String, Date, Datetime, Time, ID, Blob.

   Basic Example:

```
// Primitive data types
Integer age = 30;
Long population = 8000000000L;
Decimal price = 199.99;
Double percentage = 75.5;
Boolean isValid = true;
String message = 'Hello World';
Date today = Date.today();
Datetime now = Datetime.now();
Time currentTime = Time.newInstance(14, 30, 0, 0);
ID recordId = '001xx000003DGb2AAG';
```

3. **Q: What is the difference between Integer and Long?**
    - **A:** Integer is a 32-bit number (-2,147,483,648 to 2,147,483,647). Long is a 64-bit number for larger values.

   Basic Example:

```
Integer smallNum = 1000;
Long largeNum = 9000000000L;  // Too large for Integer

System.debug('Small: ' + smallNum);
System.debug('Large: ' + largeNum);
```

4. **Q: What is the difference between Decimal and Double?**
    - **A:** Decimal is for precise calculations (currency), Double has less precision but larger range. Use Decimal for financial calculations.

   Basic Example:

```
Decimal price = 19.99;  // Precise for currency
Double percentage = 33.333333;  // Approximation OK

// Financial calculation - use Decimal
Decimal total = price * 3;
System.debug('Total: ' + total);  // 59.97
```

5. **Q: What are sObject data types?**
    - **A:** sObject types represent Salesforce objects (standard or custom) like Account, Contact, or custom objects.

Basic Example:

```
// sObject data types
Account acc = new Account();
acc.Name = 'Acme Corp';
acc.Industry = 'Technology';

Contact con = new Contact();
con.FirstName = 'John';
con.LastName = 'Doe';

// Generic sObject
sObject obj = new Account(Name = 'Test');
```

Video 11: Operators in Apex

Interview Questions:

1. **Q: What are operators in Apex?**
    - **A:** Operators are symbols that perform operations on variables and values (arithmetic, comparison, logical, etc.).

    Basic Example:

    ```
    Integer a = 10;
    Integer b = 5;

    Integer sum = a + b;  // + is an operator
    Boolean result = a > b;  // > is an operator
    ```

2. **Q: What are the types of operators in Apex?**
    - **A:** Types include:
        - Arithmetic: +, -, *, /, ++, —
        - Comparison: ==, !=, >, <, >=, <=
        - Logical: &&, ||, !
        - Assignment: =, +=, -=, *=, /=

    Basic Example:

```
// Arithmetic operators
Integer sum = 10 + 5;  // 15
Integer diff = 10 - 5;  // 5
Integer product = 10 * 5;  // 50
Integer quotient = 10 / 5;  // 2

// Comparison operators
Boolean isEqual = (10 == 10);  // true
Boolean isGreater = (10 > 5);  // true

// Logical operators
Boolean result = (10 > 5) && (5 < 20);  // true
```

3. **Q: Explain arithmetic operators with examples.**
    - **A:** Arithmetic operators perform mathematical operations.

Basic Example:

```
Integer a = 10;
Integer b = 3;

Integer sum = a + b;  // 13
Integer diff = a - b;  // 7
Integer product = a * b;  // 30
Integer quotient = a / b;  // 3 (integer division)

// Increment/Decrement
a++;  // a becomes 11
b--;  // b becomes 2

System.debug('Sum: ' + sum);
```

4. **Q: What are comparison operators?**
    - **A:** Comparison operators compare two values and return Boolean (true/false).

Basic Example:

```
Integer x = 10;
Integer y = 20;

Boolean isEqual = (x == y);  // false
Boolean notEqual = (x != y);  // true
Boolean greater = (x > y);  // false
Boolean less = (x < y);  // true
Boolean greaterEqual = (x >= 10);  // true
Boolean lessEqual = (y <= 20);  // true
```

5. **Q: Explain logical operators in Apex.**
   - **A:** Logical operators combine boolean expressions: && (AND), || (OR), ! (NOT).

   Basic Example:

```
Integer age = 25;
Boolean hasLicense = true;

// AND operator - both must be true
Boolean canDrive = (age >= 18) && hasLicense;  // true

// OR operator - at least one must be true
Boolean isAdult = (age >= 18) || (age >= 21);  // true

// NOT operator - reverses boolean
Boolean cannotDrive = !canDrive;  // false
```

Module 4: Collections in Apex

Video 12: What are Collections?

Interview Questions:

1. **Q: What are collections in Salesforce Apex?**
   - **A:** Collections are data structures that store multiple values in a single variable. Apex supports three types: List, Set, and Map.

   Basic Example:

```
// List - ordered collection
List<String> names = new List<String>{'Alice', 'Bob', 'Charlie'};

// Set - unordered, unique values
Set<Integer> numbers = new Set<Integer>{1, 2, 3};
```

```
// Map - key-value pairs
Map<String, Integer> ages = new Map<String, Integer>{'Alice' => 30, '
```

2. **Q: Why do we need collections in Apex?**
    - **A:** Collections allow storing and processing multiple values efficiently, essential for bulk operations and working with query results.

Basic Example:

```
// Without collection - inefficient
Account acc1 = new Account(Name = 'Account 1');
Account acc2 = new Account(Name = 'Account 2');
Account acc3 = new Account(Name = 'Account 3');
// ... more accounts

// With collection - efficient
List<Account> accounts = new List<Account>();
for(Integer i = 1; i <= 100; i++) {
    accounts.add(new Account(Name = 'Account ' + i));
}
insert accounts;  // Bulk insert
```

3. **Q: What are the three types of collections in Apex?**
    - **A:**
        - List: Ordered collection allowing duplicates
        - Set: Unordered collection of unique values
        - Map: Collection of key-value pairs

Basic Example:

```
// List example
List<String> fruits = new List<String>{'Apple', 'Banana', 'Apple'};
System.debug('List size: ' + fruits.size());  // 3 (allows duplicates

// Set example
Set<String> uniqueFruits = new Set<String>{'Apple', 'Banana', 'Apple'
System.debug('Set size: ' + uniqueFruits.size());  // 2 (unique value

// Map example
Map<Integer, String> studentMap = new Map<Integer, String>{1 => 'John
System.debug('Student 1: ' + studentMap.get(1));
```

4. **Q: When should you use each type of collection?**
   - **A:**
     - List: When order matters and duplicates are allowed (query results)
     - Set: When you need unique values only
     - Map: When you need key-value associations for quick lookups

   Basic Example:

```
// Use List for ordered data
List<Account> accounts = [SELECT Id, Name FROM Account ORDER BY Name

// Use Set for unique IDs
Set<Id> accountIds = new Set<Id>();
for(Account acc : accounts) {
    accountIds.add(acc.Id);
}

// Use Map for quick lookups
Map<Id, Account> accountMap = new Map<Id, Account>(accounts);
Account specificAccount = accountMap.get(someId);
```

5. **Q: Can collections be nested in Apex?**
   - **A:** Yes, you can have collections within collections (e.g., List of Lists, Map of Lists).

   Basic Example:

```
// List of Lists (nested)
List<List<Integer>> matrix = new List<List<Integer>>{
    new List<Integer>{1, 2, 3},
    new List<Integer>{4, 5, 6},
    new List<Integer>{7, 8, 9}
};

System.debug('Value at [1][1]: ' + matrix[1][1]);  // Output: 5
```

Video 13: List Class in Apex

Interview Questions:

1. **Q: What is a List in Apex?**
    - **A:** A List is an ordered collection that can store multiple values of the same data type. Lists are indexed starting from 0 and allow duplicate values.

Basic Example:

```
// Creating a list
List<String> colors = new List<String>();
colors.add('Red');
colors.add('Blue');
colors.add('Green');
colors.add('Red');  // Duplicate allowed

System.debug('First color: ' + colors[0]);  // Red
System.debug('List size: ' + colors.size());  // 4
```

2. **Q: How do you declare and initialize a List?**
    - **A:** Lists can be declared empty or with initial values.

Basic Example:

```
// Method 1: Empty list
List<Integer> numbers = new List<Integer>();
numbers.add(10);
numbers.add(20);

// Method 2: Initialize with values
List<String> names = new List<String>{'Alice', 'Bob', 'Charlie'};

// Method 3: From array
String[] cities = new String[]{'NYC', 'LA', 'Chicago'};
List<String> cityList = new List<String>(cities);
```

3. **Q: What are the properties of a List?**
    - **A:** Lists are:
        - Ordered (maintain insertion order)
        - Indexed (access by position)
        - Allow duplicates
        - Dynamic size (can grow/shrink)

Basic Example:

```
List<Integer> nums = new List<Integer>{10, 20, 10, 30};

// Ordered - maintains sequence
System.debug('First: ' + nums[0]);  // 10
System.debug('Second: ' + nums[1]);  // 20

// Allows duplicates
System.debug('Contains duplicate 10: ' + (nums[0] == nums[2]));  // t

// Dynamic size
nums.add(40);
System.debug('New size: ' + nums.size());  // 5
```

4. **Q: How do you access elements in a List?**

   - **A:** Use index notation (list[index]) or the get() method.

   Basic Example:

```
List<String> fruits = new List<String>{'Apple', 'Banana', 'Orange'};

// Access by index
String first = fruits[0];  // Apple
String second = fruits.get(1);  // Banana

// Modify element
fruits[0] = 'Mango';

// Loop through list
for(Integer i = 0; i < fruits.size(); i++) {
    System.debug('Fruit ' + i + ': ' + fruits[i]);
}
```

5. **Q: What are common List methods?**

   - **A:** add(), get(), set(), size(), isEmpty(), clear(), remove(), sort()

   Basic Example:

```
List<Integer> numbers = new List<Integer>();

// add() - add element
numbers.add(50);
numbers.add(30);
numbers.add(40);

// size() - get count
System.debug('Size: ' + numbers.size());  // 3

// sort() - sort list
numbers.sort();
System.debug('Sorted: ' + numbers);  // [30, 40, 50]

// isEmpty() - check if empty
System.debug('Is empty: ' + numbers.isEmpty());  // false

// clear() - remove all
numbers.clear();
System.debug('After clear: ' + numbers.size());  // 0
```

Video 14: List Properties and Methods

Interview Questions:

1. **Q: How do you add elements to a List?**
   - **A:** Use the add() method to append elements or add(index, element) to insert at a specific position.

   Basic Example:

```
List<String> names = new List<String>();

// Add at end
names.add('Alice');
names.add('Charlie');

// Add at specific index
names.add(1, 'Bob');  // Insert at position 1

System.debug('Names: ' + names);  // [Alice, Bob, Charlie]
```

2. **Q: How do you remove elements from a List?**
   - **A:** Use remove(index) or clear() to remove all elements.

Basic Example:

```
List<Integer> numbers = new List<Integer>{10, 20, 30, 40};

// Remove by index
numbers.remove(1);  // Removes 20
System.debug('After remove: ' + numbers);  // [10, 30, 40]

// Clear all
numbers.clear();
System.debug('After clear: ' + numbers.size());  // 0
```

3. **Q: How do you sort a List?**
   - **A:** Use the sort() method for primitive types. For custom objects, implement Comparable interface.

Basic Example:

```
// Sort primitive types
List<Integer> numbers = new List<Integer>{30, 10, 50, 20};
numbers.sort();
System.debug('Sorted numbers: ' + numbers);  // [10, 20, 30, 50]

List<String> names = new List<String>{'Charlie', 'Alice', 'Bob'};
names.sort();
System.debug('Sorted names: ' + names);  // [Alice, Bob, Charlie]
```

4. **Q: How do you check if a List contains a specific element?**
   - **A:** Use the contains() method (for Set) or loop through the List.

Basic Example:

```
List<String> fruits = new List<String>{'Apple', 'Banana', 'Orange'};

// Manual check
Boolean hasApple = false;
for(String fruit : fruits) {
    if(fruit == 'Apple') {
        hasApple = true;
        break;
    }
}
System.debug('Has Apple: ' + hasApple);  // true

// Convert to Set for contains()
Set<String> fruitSet = new Set<String>(fruits);
System.debug('Has Mango: ' + fruitSet.contains('Mango'));  // false
```

5. **Q: How do you create a List from a SOQL query?**
   - **A:** SOQL queries return Lists of sObjects.

   Basic Example:

```
// Query returns a List
List<Account> accounts = [SELECT Id, Name FROM Account LIMIT 10];

System.debug('Account count: ' + accounts.size());

// Loop through results
for(Account acc : accounts) {
    System.debug('Account: ' + acc.Name);
}
```

Video 15: Nested Lists

Interview Questions:

1. **Q: What is a nested List?**
   - **A:** A nested List is a List that contains other Lists as elements (List of Lists).

   Basic Example:

```
// List of Lists
List<List<Integer>> matrix = new List<List<Integer>>();

// Add rows
matrix.add(new List<Integer>{1, 2, 3});
matrix.add(new List<Integer>{4, 5, 6});
matrix.add(new List<Integer>{7, 8, 9});

// Access elements
```

```
System.debug('Row 0: ' + matrix[0]);  // [1, 2, 3]
System.debug('Element [1][2]: ' + matrix[1][2]);  // 6
```

2. **Q: How do you create a 2D array using nested Lists?**
   - **A:** Create a List where each element is another List.

   Basic Example:

```
// 2D array (3x3 matrix)
List<List<Integer>> grid = new List<List<Integer>>{
    new List<Integer>{1, 2, 3},
    new List<Integer>{4, 5, 6},
    new List<Integer>{7, 8, 9}
};

// Access and modify
Integer value = grid[1][1];  // 5
grid[0][0] = 10;  // Change first element

System.debug('Modified grid: ' + grid);
```

3. **Q: How do you iterate through a nested List?**
   - **A:** Use nested for loops.

   Basic Example:

```
List<List<String>> teams = new List<List<String>>{
    new List<String>{'Alice', 'Bob'},
    new List<String>{'Charlie', 'David'},
    new List<String>{'Eve', 'Frank'}
};

// Nested iteration
for(Integer i = 0; i < teams.size(); i++) {
    System.debug('Team ' + i + ':');
    for(Integer j = 0; j < teams[i].size(); j++) {
        System.debug('  Member: ' + teams[i][j]);
    }
}
```

4. **Q: Can you have Lists of different depths?**

- **A:** Yes, each element List can have different sizes.

Basic Example:

```
List<List<Integer>> irregularList = new List<List<Integer>>{
    new List<Integer>{1, 2},
    new List<Integer>{3, 4, 5, 6},
    new List<Integer>{7}
};

for(List<Integer> innerList : irregularList) {
    System.debug('Inner list size: ' + innerList.size());
    System.debug('Values: ' + innerList);
}
```

5. **Q: What are practical uses of nested Lists?**
    - **A:** Representing matrices, grouped data, hierarchical structures, or complex data relationships.

Basic Example:

```
// Practical example: Student grades by subject
List<List<Integer>> studentGrades = new List<List<Integer>>();

// Student 1 grades: Math, Science, English
studentGrades.add(new List<Integer>{85, 90, 88});

// Student 2 grades
studentGrades.add(new List<Integer>{92, 87, 95});

// Calculate average for student 0
Integer sum = 0;
for(Integer grade : studentGrades[0]) {
    sum += grade;
}
Decimal average = Decimal.valueOf(sum) / studentGrades[0].size();
System.debug('Student 1 average: ' + average);  // 87.67
```

Video 16: Set in Apex

Interview Questions:

1. **Q: What is a Set in Apex?**
    - **A:** A Set is an unordered collection that stores only unique values (no duplicates allowed).

   Basic Example:

```
Set<String> colors = new Set<String>();
colors.add('Red');
colors.add('Blue');
colors.add('Red');  // Duplicate - will be ignored

System.debug('Set size: ' + colors.size());  // 2 (only unique values
System.debug('Colors: ' + colors);  // {Blue, Red} (unordered)
```

2. **Q: How is a Set different from a List?**
    - **A:** Set: unordered, no duplicates. List: ordered, allows duplicates.

   Basic Example:

```
// List allows duplicates
List<Integer> numList = new List<Integer>{1, 2, 2, 3};
System.debug('List size: ' + numList.size());  // 4

// Set removes duplicates
Set<Integer> numSet = new Set<Integer>{1, 2, 2, 3};
System.debug('Set size: ' + numSet.size());  // 3

// Set is unordered - no index access
// numSet[0];  // Error: Sets don't support indexing
```

3. **Q: How do you create and initialize a Set?**
    - **A:** Use the Set constructor.

   Basic Example:

```
// Empty Set
Set<String> names = new Set<String>();
names.add('Alice');
names.add('Bob');

// Initialize with values
Set<Integer> numbers = new Set<Integer>{10, 20, 30};
```

```
// Create from List (removes duplicates)
List<String> cityList = new List<String>{'NYC', 'LA', 'NYC'};
Set<String> citySet = new Set<String>(cityList);
System.debug('Cities: ' + citySet);  // {LA, NYC}
```

4. **Q: What are common Set methods?**
    - **A:** add(), addAll(), contains(), remove(), size(), isEmpty(), clear()

Basic Example:

```
Set<Integer> numbers = new Set<Integer>();

// add() - add element
numbers.add(10);
numbers.add(20);
numbers.add(10);  // Duplicate ignored

// contains() - check if exists
Boolean has10 = numbers.contains(10);  // true
System.debug('Contains 10: ' + has10);

// size() - get count
System.debug('Size: ' + numbers.size());  // 2

// remove() - remove element
numbers.remove(10);
System.debug('After remove: ' + numbers);  // {20}
```

5. **Q: When should you use a Set instead of a List?**
    - **A:** Use Set when:
        - You need unique values only
        - Order doesn't matter
        - You need fast contains() checks
        - Removing duplicates from data

Basic Example:

```
// Example: Get unique account IDs from contacts
List<Contact> contacts = [SELECT Id, AccountId FROM Contact LIMIT 10(

Set<Id> uniqueAccountIds = new Set<Id>();
```

```
    for(Contact con : contacts) {
        if(con.AccountId != null) {
            uniqueAccountIds.add(con.AccountId);  // Automatically ensure
        }
    }

    System.debug('Unique accounts: ' + uniqueAccountIds.size());
```

Video 17: Map in Apex

Interview Questions:

1. **Q: What is a Map in Apex?**

   - **A:** A Map is a collection of key-value pairs where each key is unique and maps to exactly one value.

   Basic Example:

   ```
   // Map of student ID to name
   Map<Integer, String> students = new Map<Integer, String>();
   students.put(1, 'Alice');
   students.put(2, 'Bob');
   students.put(3, 'Charlie');

   // Get value by key
   String studentName = students.get(2);
   System.debug('Student 2: ' + studentName);  // Bob
   ```

2. **Q: How do you declare and initialize a Map?**

   - **A:** Use Map<KeyType, ValueType> syntax.

   Basic Example:

   ```
   // Empty Map
   Map<String, Integer> scores = new Map<String, Integer>();
   scores.put('Alice', 95);
   scores.put('Bob', 87);

   // Initialize with values
   Map<String, String> capitals = new Map<String, String>{
       'USA' => 'Washington DC',
   ```

```
        'India' => 'New Delhi',
        'Japan' => 'Tokyo'
    };

    // Create from List (Id to sObject)
    List<Account> accounts = [SELECT Id, Name FROM Account LIMIT 10];
    Map<Id, Account> accountMap = new Map<Id, Account>(accounts);
```

3. **Q: What are the key methods of Map?**
   - **A:** put(), get(), containsKey(), remove(), keySet(), values(), size()

   Basic Example:

```
Map<String, Integer> ages = new Map<String, Integer>();

// put() - add/update entry
ages.put('Alice', 30);
ages.put('Bob', 25);

// get() - retrieve value
Integer aliceAge = ages.get('Alice');  // 30

// containsKey() - check if key exists
Boolean hasBob = ages.containsKey('Bob');  // true

// keySet() - get all keys
Set<String> names = ages.keySet();
System.debug('Names: ' + names);

// values() - get all values
List<Integer> ageList = ages.values();
System.debug('Ages: ' + ageList);

// remove() - remove entry
ages.remove('Bob');
```

4. **Q: How do you iterate through a Map?**
   - **A:** Use keySet() or values() to loop through keys or values.

   Basic Example:

```
Map<String, Integer> scores = new Map<String, Integer>{
    'Math' => 85,
    'Science' => 90,
    'English' => 88
};

// Iterate through keys
for(String subject : scores.keySet()) {
    Integer score = scores.get(subject);
    System.debug(subject + ': ' + score);
}

// Iterate through values
for(Integer score : scores.values()) {
    System.debug('Score: ' + score);
}
```

5. **Q: When should you use a Map?**
    - **A:** Use Map when:
        - Need quick lookups by key
        - Associating related data (ID to record)
        - Avoiding nested loops
        - Tracking unique pairs

Basic Example:

```
// Practical example: Update contacts based on account changes
Set<Id> accountIds = new Set<Id>{acc1.Id, acc2.Id, acc3.Id};

// Query accounts into Map for quick lookup
Map<Id, Account> accountMap = new Map<Id, Account>(
    [SELECT Id, Phone FROM Account WHERE Id IN :accountIds]
);

List<Contact> contactsToUpdate = new List<Contact>();
for(Contact con : [SELECT Id, AccountId FROM Contact WHERE AccountId
    Account acc = accountMap.get(con.AccountId);  // O(1) lookup inst
    if(acc != null) {
        con.Phone = acc.Phone;
        contactsToUpdate.add(con);
    }
}
```

```
update contactsToUpdate;
```

Module 5: sObjects

Video 18: Understanding sObjects

Interview Questions:

1. **Q: What is an sObject in Salesforce?**
   - **A:** sObject is a generic abstract type representing any Salesforce object (standard or custom). It's the base type for all records.

   Basic Example:

   ```
   // Specific sObject types
   Account acc = new Account(Name = 'Acme Corp');
   Contact con = new Contact(FirstName = 'John', LastName = 'Doe');

   // Generic sObject
   sObject obj = new Account(Name = 'Generic Account');

   // Must cast to access specific fields
   Account specificAcc = (Account)obj;
   System.debug('Name: ' + specificAcc.Name);
   ```

2. **Q: What's the difference between sObject and a specific object type?**
   - **A:** sObject is generic and can hold any object type. Specific types (Account, Contact) have defined fields.

   Basic Example:

   ```
   // Generic sObject - flexible but limited
   sObject genericObj = new Account();
   // genericObj.Name = 'Test';  // Error: can't access fields directly

   // Specific type - full field access
   Account acc = new Account();
   acc.Name = 'Test';  // Works fine
   ```

```
acc.Industry = 'Technology';

// Cast generic to specific
sObject gen = acc;
Account specific = (Account)gen;
System.debug('Name: ' + specific.Name);
```

3. **Q: How do you create an sObject record?**
    - **A:** Use the new keyword with field assignments.

Basic Example:

```
// Method 1: Using constructor
Account acc1 = new Account(
    Name = 'Acme Corp',
    Industry = 'Technology',
    Phone = '555-1234'
);

// Method 2: Step by step
Account acc2 = new Account();
acc2.Name = 'Wayne Enterprises';
acc2.Industry = 'Manufacturing';
acc2.AnnualRevenue = 5000000;

// Insert to save
insert acc1;
System.debug('Created account with ID: ' + acc1.Id);
```

4. **Q: How do you access sObject fields?**
    - **A:** Use dot notation: object.FieldName

Basic Example:

```
// Create account
Account acc = new Account(Name = 'Test Company');
insert acc;

// Access standard fields
String accountName = acc.Name;
Id accountId = acc.Id;
```

```
// Access custom fields (end with __c)
// acc.Custom_Field__c = 'Value';

// Query and access
Account queriedAcc = [SELECT Id, Name, Industry FROM Account WHERE Id
System.debug('Name: ' + queriedAcc.Name);
System.debug('Industry: ' + queriedAcc.Industry);
```

5. **Q: What are the types of sObjects?**
    - **A:** Standard sObjects (Account, Contact, etc.) and Custom sObjects (MyObject__c).

   Basic Example:

```
// Standard sObjects
Account acc = new Account(Name = 'Standard Object');
Contact con = new Contact(LastName = 'Doe');
Opportunity opp = new Opportunity(Name = 'Deal', StageName = 'Prospec

// Custom sObject (example - ends with __c)
// Custom_Student__c student = new Custom_Student__c(Name = 'John', (

// Generic sObject can hold any
sObject obj1 = acc;
sObject obj2 = con;
```

Module 6: SOQL (Salesforce Object Query Language)

Video 19: Introduction to SOQL

Interview Questions:

1. **Q: What is SOQL?**
    - **A:** SOQL (Salesforce Object Query Language) is used to query records from the Salesforce database. Similar to SQL but designed for Salesforce.

   Basic Example:

```
// Basic SOQL query
List<Account> accounts = [SELECT Id, Name FROM Account];
System.debug('Found ' + accounts.size() + ' accounts');
```

```
for(Account acc : accounts) {
    System.debug('Account: ' + acc.Name);
}
```

2. **Q: What is the basic syntax of SOQL?**
    - **A:** SELECT [fields] FROM [object] WHERE [conditions]

Basic Example:

```
// SELECT specific fields
List<Contact> contacts = [SELECT Id, FirstName, LastName, Email FROM

// SELECT with WHERE clause
List<Account> techAccounts = [SELECT Id, Name FROM Account WHERE Indu

// SELECT with multiple conditions
List<Opportunity> hotOpps = [
    SELECT Id, Name, Amount
    FROM Opportunity
    WHERE StageName = 'Negotiation' AND Amount > 100000
];
```

3. **Q: How does SOQL differ from SQL?**
    - **A:** SOQL:
        - No SELECT *
        - No JOIN (uses relationships)
        - Case-insensitive
        - Returns typed Lists
        - No INSERT/UPDATE/DELETE (uses DML)

Basic Example:

```
// SOQL query
List<Account> accounts = [SELECT Id, Name, Industry FROM Account LIM

// Relationship query (instead of JOIN)
List<Contact> contacts = [
    SELECT Id, FirstName, Account.Name, Account.Industry
    FROM Contact
    WHERE Account.Industry = 'Technology'
];
```

4. **Q: What types of relationships can SOQL query?**
    - **A:** Parent-to-child (subquery) and child-to-parent (dot notation).

   Basic Example:

```
// Child-to-parent (dot notation)
List<Contact> contacts = [
    SELECT Id, FirstName, Account.Name, Account.Owner.Name
    FROM Contact
    LIMIT 10
];
System.debug('Contact: ' + contacts[0].FirstName + ', Account: ' + co

// Parent-to-child (subquery)
List<Account> accountsWithContacts = [
    SELECT Id, Name, (SELECT Id, FirstName, LastName FROM Contacts)
    FROM Account
    LIMIT 10
];
for(Account acc : accountsWithContacts) {
    System.debug('Account: ' + acc.Name);
    for(Contact con : acc.Contacts) {
        System.debug('  Contact: ' + con.FirstName + ' ' + con.LastNa
    }
}
```

5. **Q: What is the purpose of LIMIT clause in SOQL?**
    - **A:** LIMIT restricts the number of records returned, helping avoid governor limits.

   Basic Example:

```
// Without LIMIT - might return too many records
// List<Account> accounts = [SELECT Id FROM Account];  // Risky!

// With LIMIT - safe
List<Account> accounts = [SELECT Id, Name FROM Account LIMIT 100];
System.debug('Retrieved ' + accounts.size() + ' accounts');

// LIMIT with OFFSET for pagination
```

```
    List<Account> page1 = [SELECT Id, Name FROM Account ORDER BY Name LIM
    List<Account> page2 = [SELECT Id, Name FROM Account ORDER BY Name LIM
```

Video 20: LIKE Keyword in SOQL

Interview Questions:

1. **Q: What is the LIKE operator in SOQL?**
    - **A:** LIKE is used for pattern matching in string fields, similar to SQL LIKE.

    Basic Example:

    ```
    // Find accounts starting with 'A'
    List<Account> accounts = [SELECT Id, Name FROM Account WHERE Name LIK

    for(Account acc : accounts) {
        System.debug('Account: ' + acc.Name);
    }
    ```

2. **Q: What wildcards can be used with LIKE?**
    - **A:**
        - % (percent) - matches zero or more characters
        - _ (underscore) - matches exactly one character

    Basic Example:

    ```
    // % wildcard - multiple characters
    List<Account> startsWithA = [SELECT Name FROM Account WHERE Name LIKE
    List<Account> endsWithInc = [SELECT Name FROM Account WHERE Name LIKE
    List<Account> containsCorp = [SELECT Name FROM Account WHERE Name LIK

    // _ wildcard - single character
    List<Account> threeLetters = [SELECT Name FROM Account WHERE Name LIK
    ```

3. **Q: Is LIKE case-sensitive in SOQL?**
    - **A:** No, LIKE in SOQL is case-insensitive.

    Basic Example:

```
// All of these will match 'Acme Corp', 'ACME CORP', 'acme corp'
List<Account> result1 = [SELECT Name FROM Account WHERE Name LIKE 'ac
List<Account> result2 = [SELECT Name FROM Account WHERE Name LIKE 'AC
List<Account> result3 = [SELECT Name FROM Account WHERE Name LIKE 'Ac

System.debug('All queries return the same results');
```

4. **Q: How do you escape special characters in LIKE?**
   - **A:** Use backslash () to escape % and _.

Basic Example:

```
// Find accounts with actual % in name
List<Account> withPercent = [SELECT Name FROM Account WHERE Name LIKE

// Find accounts with actual _ in name
List<Account> withUnderscore = [SELECT Name FROM Account WHERE Name I

// Example: Finding '100% Corp' or 'Test_Account'
```

5. **Q: Can you use LIKE with other operators?**
   - **A:** Yes, combine LIKE with AND, OR, NOT.

Basic Example:

```
// LIKE with AND
List<Account> techStartingWithA = [
    SELECT Name, Industry
    FROM Account
    WHERE Name LIKE 'A%' AND Industry = 'Technology'
];

// LIKE with OR
List<Account> multiplePatterns = [
    SELECT Name
    FROM Account
    WHERE Name LIKE 'A%' OR Name LIKE 'B%'
];

// NOT LIKE
List<Account> notStartingWithA = [
    SELECT Name
    FROM Account
    WHERE Name != null AND NOT (Name LIKE 'A%')
];
```

Video 21: IN Keyword in SOQL

Interview Questions:

1. **Q: What is the IN operator in SOQL?**
   - **A:** IN checks if a field value matches any value in a list.

   Basic Example:

```
// IN with literal values
List<Account> accounts = [
    SELECT Id, Name, Industry
    FROM Account
    WHERE Industry IN ('Technology', 'Finance', 'Healthcare')
];

for(Account acc : accounts) {
    System.debug(acc.Name + ' - ' + acc.Industry);
}
```

2. **Q: How do you use IN with a collection variable?**
   - **A:** Use :variableName to bind a List or Set.

   Basic Example:

```
// Create a Set of industries
Set<String> industries = new Set<String>{'Technology', 'Finance', 'Re

// Use IN with variable
List<Account> accounts = [
    SELECT Id, Name, Industry
    FROM Account
    WHERE Industry IN :industries
];

System.debug('Found ' + accounts.size() + ' accounts');
```

3. **Q: Can you use IN with IDs?**
    - **A:** Yes, commonly used to query specific records by ID.

Basic Example:

```
// Create Set of IDs
Set<Id> accountIds = new Set<Id>();
accountIds.add('001xx000003DGb2AAG');
accountIds.add('001xx000003DGb3AAG');
accountIds.add('001xx000003DGb4AAG');

// Query using IN
List<Account> accounts = [
    SELECT Id, Name
    FROM Account
    WHERE Id IN :accountIds
];

// Practical example: Get contacts for specific accounts
List<Contact> contacts = [
    SELECT Id, FirstName, LastName, AccountId
    FROM Contact
    WHERE AccountId IN :accountIds
];
```

4. **Q: What's the difference between IN and = (equals)?**
    - **A:** = checks for one value, IN checks for multiple values.

Basic Example:

```
// Using = (single value)
List<Account> techAccounts = [
    SELECT Id, Name
    FROM Account
    WHERE Industry = 'Technology'
];

// Using IN (multiple values)
List<Account> multipleIndustries = [
    SELECT Id, Name
    FROM Account
    WHERE Industry IN ('Technology', 'Finance', 'Healthcare')
];
```

5. **Q: Can you use NOT IN?**
    - **A:** Yes, NOT IN excludes records matching the list.

   Basic Example:

```
Set<String> excludeIndustries = new Set<String>{'Banking', 'Insurance

// NOT IN - exclude specific industries
List<Account> accounts = [
    SELECT Id, Name, Industry
    FROM Account
    WHERE Industry NOT IN :excludeIndustries
];

// Practical: Get accounts without opportunities
Set<Id> accountsWithOpps = new Set<Id>();
for(Opportunity opp : [SELECT AccountId FROM Opportunity]) {
    accountsWithOpps.add(opp.AccountId);
}

List<Account> accountsWithoutOpps = [
    SELECT Id, Name
    FROM Account
    WHERE Id NOT IN :accountsWithOpps
];
```

Module 7: DML (Data Manipulation Language)

Video 22: DML Basics in Apex

Interview Questions:

1. **Q: What is DML in Salesforce?**
    - **A:** DML (Data Manipulation Language) operations are used to insert, update, delete, and undelete records in the database.

   Basic Example:

```
// INSERT - create new record
Account acc = new Account(Name = 'New Account');
insert acc;
System.debug('Created account with ID: ' + acc.Id);

// UPDATE - modify existing record
acc.Industry = 'Technology';
update acc;

// DELETE - remove record
delete acc;

// UNDELETE - restore deleted record
undelete acc;
```

2. **Q: What are the types of DML operations?**
   - **A:** insert, update, upsert, delete, undelete, merge

   Basic Example:

```
// INSERT
Contact con = new Contact(LastName = 'Smith');
insert con;

// UPDATE
con.FirstName = 'John';
update con;

// UPSERT (insert or update based on external ID)
Account acc = new Account(Name = 'Test', ExternalId__c = '12345');
upsert acc ExternalId__c;

// DELETE
delete con;

// UNDELETE (restore from recycle bin)
undelete con;
```

3. **Q: Can you perform DML on multiple records at once?**
   - **A:** Yes, DML operations can be performed on Lists (bulk operations).

Basic Example:

```
// Bulk INSERT
List<Account> accounts = new List<Account>();
for(Integer i = 1; i <= 100; i++) {
    accounts.add(new Account(Name = 'Account ' + i));
}
insert accounts;  // Single DML statement for 100 records

// Bulk UPDATE
List<Account> accountsToUpdate = [SELECT Id, Industry FROM Account WH
for(Account acc : accountsToUpdate) {
    acc.Industry = 'Technology';
}
update accountsToUpdate;
```

4. **Q: What happens if a DML operation fails?**
    - **A:** By default, the entire transaction rolls back. Use Database methods for partial success.

Basic Example:

```
// Standard DML - all or nothing
try {
    List<Account> accounts = new List<Account>{
        new Account(Name = 'Valid Account'),
        new Account()  // Invalid - no name
    };
    insert accounts;  // This will fail and rollback both
} catch(DmlException e) {
    System.debug('Error: ' + e.getMessage());
}

// Database method - partial success
List<Account> accounts = new List<Account>{
    new Account(Name = 'Valid Account'),
    new Account()  // Invalid
};
Database.SaveResult[] results = Database.insert(accounts, false);  /

for(Database.SaveResult sr : results) {
    if(sr.isSuccess()) {
        System.debug('Success: ' + sr.getId());
    } else {
        System.debug('Error: ' + sr.getErrors()[0].getMessage());
    }
}
```

5. **Q: What are DML governor limits?**
    - **A:** Maximum 150 DML statements per transaction. Each DML can affect up to 10,000 records.

   Basic Example:

```
// BAD - Multiple DML statements
for(Integer i = 0; i < 200; i++) {
    Account acc = new Account(Name = 'Account ' + i);
    insert acc;  // 200 DML statements - will hit limit!
}

// GOOD - Single DML statement
List<Account> accounts = new List<Account>();
for(Integer i = 0; i < 200; i++) {
    accounts.add(new Account(Name = 'Account ' + i));
}
insert accounts;  // 1 DML statement for 200 records
```

Module 8: OOP Concepts in Apex

Video 23: Classes and Objects

Interview Questions:

1. **Q: What is a class in Apex?**
    - **A:** A class is a blueprint or template that defines the structure and behavior of objects. It contains variables and methods.

   Basic Example:

```
// Define a class
public class Student {
    // Variables (properties)
    public String name;
    public Integer age;
    public String grade;

    // Method (behavior)
    public void displayInfo() {
        System.debug('Name: ' + name + ', Age: ' + age + ', Grade: '
    }
}

// Create object (instance) of class
Student student1 = new Student();
student1.name = 'Alice';
student1.age = 15;
student1.grade = 'A';
student1.displayInfo();
```

2. **Q: What is an object in Apex?**
   - **A:** An object is an instance of a class. It's a concrete entity created from the class blueprint.

Basic Example:

```
public class Car {
    public String brand;
    public String model;
    public Integer year;

    public void start() {
        System.debug(brand + ' ' + model + ' started!');
    }
}

// Create multiple objects from same class
Car car1 = new Car();
car1.brand = 'Toyota';
car1.model = 'Camry';
car1.year = 2020;

Car car2 = new Car();
car2.brand = 'Honda';
car2.model = 'Civic';
car2.year = 2021;
```

```
car1.start();  // Toyota Camry started!
car2.start();  // Honda Civic started!
```

3. **Q: What's the difference between a class and an object?**
    - **A:** Class is a template/blueprint. Object is a real instance created from the class.

Basic Example:

```
// Class - Blueprint
public class BankAccount {
    public Decimal balance;

    public void deposit(Decimal amount) {
        balance += amount;
    }
}

// Objects - Real instances
BankAccount account1 = new BankAccount();
account1.balance = 1000;

BankAccount account2 = new BankAccount();
account2.balance = 5000;

// Different objects, independent data
account1.deposit(500);
System.debug('Account 1: ' + account1.balance);  // 1500
System.debug('Account 2: ' + account2.balance);  // 5000
```

4. **Q: How do you create an object from a class?**
    - **A:** Use the 'new' keyword followed by the class name and parentheses.

Basic Example:

```
public class Calculator {
    public Integer add(Integer a, Integer b) {
        return a + b;
    }
}

// Create object
Calculator calc = new Calculator();
```

```
// Use object's methods
Integer result = calc.add(10, 20);
System.debug('Result: ' + result);  // 30
```

5. **Q: Can a class have both variables and methods?**
    - **A:** Yes, classes contain both member variables (data) and methods (behavior).

   Basic Example:

```
public class Employee {
    // Variables (data)
    public String name;
    public Decimal salary;
    public String department;

    // Methods (behavior)
    public void giveRaise(Decimal percentage) {
        salary = salary + (salary * percentage / 100);
    }

    public void displayDetails() {
        System.debug('Employee: ' + name);
        System.debug('Department: ' + department);
        System.debug('Salary: ' + salary);
    }
}

// Usage
Employee emp = new Employee();
emp.name = 'John Doe';
emp.salary = 50000;
emp.department = 'Sales';
emp.giveRaise(10);  // 10% raise
emp.displayDetails();
```

Video 24: Methods in Apex

Interview Questions:

1. **Q: What is a method in Apex?**

- **A:** A method is a block of code that performs a specific task. Methods can accept parameters and return values.

Basic Example:

```
public class MathOperations {
    // Method with parameters and return value
    public Integer add(Integer a, Integer b) {
        return a + b;
    }

    // Method with no return value (void)
    public void printMessage(String message) {
        System.debug(message);
    }
}

// Usage
MathOperations math = new MathOperations();
Integer sum = math.add(10, 20);
math.printMessage('Sum is: ' + sum);
```

2. **Q: What are the components of a method?**
    - **A:** Access modifier, return type, method name, parameters, method body.

Basic Example:

```
// public = access modifier
// Integer = return type
// multiply = method name
// (Integer x, Integer y) = parameters
public Integer multiply(Integer x, Integer y) {
    Integer result = x * y;  // method body
    return result;
}
```

3. **Q: What's the difference between static and instance methods?**
    - **A:** Static methods belong to the class. Instance methods belong to objects.

Basic Example:

```
public class Utility {
    // Static method - called on class
    public static Integer add(Integer a, Integer b) {
        return a + b;
    }

    // Instance method - called on object
    public void displayMessage() {
        System.debug('Instance method called');
    }
}

// Call static method - no object needed
Integer sum = Utility.add(10, 20);

// Call instance method - need object
Utility util = new Utility();
util.displayMessage();
```

4. **Q: Can methods have parameters?**
    - **A:** Yes, methods can have zero or more parameters of any data type.

Basic Example:

```
public class Calculator {
    // No parameters
    public void sayHello() {
        System.debug('Hello!');
    }

    // One parameter
    public Integer square(Integer num) {
        return num * num;
    }

    // Multiple parameters
    public Decimal calculateTax(Decimal amount, Decimal taxRate) {
        return amount * taxRate / 100;
    }
}

// Usage
```

```
Calculator calc = new Calculator();
calc.sayHello();
Integer sq = calc.square(5);  // 25
Decimal tax = calc.calculateTax(1000, 5);  // 50
```

5. **Q: What is method overloading?**
   - **A:** Having multiple methods with the same name but different parameters.

   Basic Example:

```
public class Printer {
    // Print integer
    public void print(Integer num) {
        System.debug('Integer: ' + num);
    }

    // Print string
    public void print(String text) {
        System.debug('String: ' + text);
    }

    // Print two integers
    public void print(Integer a, Integer b) {
        System.debug('Two integers: ' + a + ', ' + b);
    }
}

// Usage - same method name, different parameters
Printer p = new Printer();
p.print(42);           // Calls first method
p.print('Hello');      // Calls second method
p.print(10, 20);       // Calls third method
```

Video 25: Constructors

Interview Questions:

1. **Q: What is a constructor in Apex?**
   - **A:** A constructor is a special method that's automatically called when an object is created. It has the same name as the class and no return type.

Basic Example:

```
public class Student {
    public String name;
    public Integer age;

    // Constructor
    public Student(String studentName, Integer studentAge) {
        name = studentName;
        age = studentAge;
        System.debug('Student object created: ' + name);
    }
}

// When you create object, constructor is called
Student s1 = new Student('Alice', 20);  // "Student object created: A
```

2. **Q: What's the difference between default and parameterized constructors?**
    - **A:** Default constructor has no parameters. Parameterized constructor accepts arguments.

Basic Example:

```
public class Account {
    public String accountName;
    public Decimal balance;

    // Default constructor
    public Account() {
        accountName = 'New Account';
        balance = 0;
    }

    // Parameterized constructor
    public Account(String name, Decimal initialBalance) {
        accountName = name;
        balance = initialBalance;
    }
}

// Using default constructor
Account acc1 = new Account();  // name = 'New Account', balance = 0
```

```
// Using parameterized constructor
Account acc2 = new Account('Savings', 5000);  // name = 'Savings', ba
```

3. **Q: Can you have multiple constructors in a class?**
    - **A:** Yes, this is called constructor overloading.

Basic Example:

```
public class Product {
    public String name;
    public Decimal price;
    public String category;

    // Constructor 1 - name only
    public Product(String productName) {
        name = productName;
        price = 0;
        category = 'General';
    }

    // Constructor 2 - name and price
    public Product(String productName, Decimal productPrice) {
        name = productName;
        price = productPrice;
        category = 'General';
    }

    // Constructor 3 - all parameters
    public Product(String productName, Decimal productPrice, String p
        name = productName;
        price = productPrice;
        category = productCategory;
    }
}

// Different ways to create object
Product p1 = new Product('Laptop');
Product p2 = new Product('Mouse', 25.99);
Product p3 = new Product('Keyboard', 75.50, 'Electronics');
```

4. **Q: What happens if you don't define a constructor?**
    - **A:** Apex provides a default no-argument constructor automatically.

Basic Example:

```
public class SimpleClass {
    public String value;
    // No constructor defined
}

// Still works - Apex provides default constructor
SimpleClass obj = new SimpleClass();
obj.value = 'Test';
```

5. **Q: Can constructors call other constructors?**
   - **A:** Yes, using this() to call another constructor in the same class.

Basic Example:

```
public class Employee {
    public String name;
    public Decimal salary;
    public String department;

    // Constructor 1 - all parameters
    public Employee(String empName, Decimal empSalary, String dept) 
        name = empName;
        salary = empSalary;
        department = dept;
    }

    // Constructor 2 - calls constructor 1 with default department
    public Employee(String empName, Decimal empSalary) {
        this(empName, empSalary, 'General');  // Calls constructor 1
    }
}

// Usage
Employee emp1 = new Employee('John', 50000, 'Sales');
Employee emp2 = new Employee('Jane', 60000);  // Department = 'Genera
```

Video 26: Access Modifiers

Interview Questions:

1. **Q: What are access modifiers in Apex?**
   - **A:** Access modifiers control the visibility and accessibility of classes, methods, and variables. Types: private, public, protected, global.

Basic Example:

```
public class AccessExample {
    private String privateVar = 'Private';  // Only within this class
    public String publicVar = 'Public';     // Within namespace
    protected String protectedVar = 'Protected';  // Within class anc
    global String globalVar = 'Global';     // Everywhere

    private void privateMethod() {
        System.debug('Private method');
    }

    public void publicMethod() {
        System.debug('Public method');
        privateMethod();  // Can call private from within class
    }
}
```

2. **Q: What's the difference between private and public?**
   - **A:** Private: accessible only within the class. Public: accessible within the namespace.

Basic Example:

```
public class MyClass {
    private Integer privateNum = 10;
    public Integer publicNum = 20;

    private void privateMethod() {
        System.debug('Private');
    }

    public void publicMethod() {
        System.debug('Public');
        privateMethod();  // OK - within same class
        System.debug(privateNum);  // OK
    }
}
```

```
// From another class
MyClass obj = new MyClass();
obj.publicMethod();  // OK
// obj.privateMethod();  // Error - not accessible
System.debug(obj.publicNum);  // OK
// System.debug(obj.privateNum);  // Error - not accessible
```

3. **Q: When should you use 'global' access modifier?**
    - **A:** Use 'global' when you want the class/method accessible from managed packages or anywhere across organizations.

   Basic Example:

```
// Global class - accessible everywhere
global class GlobalUtility {
    global static Integer add(Integer a, Integer b) {
        return a + b;
    }
}

// Public class - accessible in same namespace
public class PublicUtility {
    public static Integer multiply(Integer a, Integer b) {
        return a * b;
    }
}
```

4. **Q: What is the 'protected' access modifier?**
    - **A:** Protected makes members accessible within the class and its subclasses.

   Basic Example:

```
public virtual class Parent {
    protected String protectedField = 'Protected';

    protected void protectedMethod() {
        System.debug('Protected method in parent');
    }
}

public class Child extends Parent {
    public void testAccess() {
        System.debug(protectedField);  // OK - accessible in subclass
        protectedMethod();  // OK
    }
}

// From outside
Child c = new Child();
c.testAccess();  // OK
// c.protectedMethod();  // Error - not accessible outside
```

5. **Q: What happens if you don't specify an access modifier?**

   - **A:** In Apex, you must specify an access modifier for top-level classes. For members, default is private.

   Basic Example:

```
public class MyClass {
    String name;  // Default is private for member variables

    void display() {  // Default is private for methods
        System.debug(name);
    }
}
```

Video 27: Inheritance

Interview Questions:

1. **Q: What is inheritance in Apex?**

   - **A:** Inheritance is when a class (child) inherits properties and methods from another class (parent). Uses 'extends' keyword.

   Basic Example:

```
// Parent class
public virtual class Animal {
    public String name;

    public virtual void makeSound() {
        System.debug('Some generic sound');
    }
}

// Child class inherits from Animal
public class Dog extends Animal {
    public override void makeSound() {
        System.debug('Woof! Woof!');
    }
}

// Usage
Dog myDog = new Dog();
myDog.name = 'Buddy';  // Inherited from Animal
myDog.makeSound();  // "Woof! Woof!"
```

2. **Q: Why do we use inheritance?**
    - **A:** Code reusability, creating hierarchical relationships, polymorphism.

Basic Example:

```
// Parent - common functionality
public virtual class Employee {
    public String name;
    public Decimal baseSalary;

    public virtual Decimal calculateSalary() {
        return baseSalary;
    }
}

// Child 1 - specific type
public class Manager extends Employee {
    public Decimal bonus;

    public override Decimal calculateSalary() {
        return baseSalary + bonus;
    }
}

// Child 2 - specific type
public class Developer extends Employee {
    public Integer hoursWorked;
    public override Decimal calculateSalary() {
        return baseSalary + (hoursWorked * 50);
    }
}
```

3. **Q: What is the 'virtual' keyword?**
    - **A:** 'virtual' allows a method or class to be overridden by child classes.

   Basic Example:

```
public virtual class Vehicle {
    public virtual void start() {
        System.debug('Vehicle starting...');
    }
}

public class Car extends Vehicle {
    public override void start() {
        System.debug('Car engine starting...');
    }
}

Vehicle v = new Car();
v.start();  // "Car engine starting..." - overridden method called
```

4. **Q: What is the 'override' keyword?**
    - **A:** 'override' is used in child class to provide a new implementation of a parent's virtual method.

   Basic Example:

```
public virtual class Shape {
    public virtual Decimal area() {
        return 0;
    }
}

public class Circle extends Shape {
    public Decimal radius;

    public Circle(Decimal r) {
        radius = r;
    }

    public override Decimal area() {
        return 3.14 * radius * radius;
    }
}

Circle c = new Circle(5);
System.debug('Circle area: ' + c.area());  // 78.5
```

5. **Q: Can you call parent class methods from child class?**
    - **A:** Yes, using 'super' keyword.

   Basic Example:

```
    public virtual class Parent {
        public virtual void display() {
            System.debug('Parent display');
        }
    }

    public class Child extends Parent {
        public override void display() {
            super.display();  // Call parent's method
            System.debug('Child display');
        }
    }

    // Usage
    Child c = new Child();
    c.display();
    // Output:
    // Parent display
    // Child display
```

Video 28: Interfaces

Interview Questions:

1. **Q: What is an interface in Apex?**

- **A:** An interface is a contract that defines method signatures without implementation. Classes that implement the interface must provide the implementation.

Basic Example:

```
// Interface definition
public interface Drawable {
    void draw();
}

// Class implementing interface
public class Circle implements Drawable {
    public void draw() {
        System.debug('Drawing a circle');
    }
}

// Usage
Drawable shape = new Circle();
shape.draw();  // "Drawing a circle"
```

2. **Q: How is an interface different from a class?**

   - **A:** Interface: only method signatures, no implementation. Class: has both method signatures and implementation.

Basic Example:

```
// Interface - no implementation
public interface Calculator {
    Integer add(Integer a, Integer b);
    Integer subtract(Integer a, Integer b);
}

// Class - must implement all methods
public class BasicCalculator implements Calculator {
    public Integer add(Integer a, Integer b) {
        return a + b;  // Implementation
    }

    public Integer subtract(Integer a, Integer b) {
        return a - b;  // Implementation
    }
}
```

3. **Q: Can a class implement multiple interfaces?**
    - **A:** Yes, Apex supports multiple interface implementation (but not multiple inheritance).

   Basic Example:

```
public interface Flyable {
    void fly();
}

public interface Swimmable {
    void swim();
}

// Class implementing both interfaces
public class Duck implements Flyable, Swimmable {
    public void fly() {
        System.debug('Duck is flying');
    }

    public void swim() {
        System.debug('Duck is swimming');
    }
}

// Usage
Duck duck = new Duck();
duck.fly();
duck.swim();
```

4. **Q: What happens if you don't implement all interface methods?**
    - **A:** Compilation error. All methods declared in the interface must be implemented.

   Basic Example:

```
public interface Animal {
    void eat();
    void sleep();
}

// This will cause error if sleep() is not implemented
public class Dog implements Animal {
    public void eat() {
        System.debug('Dog is eating');
    }

    // Must implement sleep() too
    public void sleep() {
        System.debug('Dog is sleeping');
    }
}
```

5. **Q: Why use interfaces?**
    - **A:** Standardization, loose coupling, polymorphism, defining contracts for classes.

Basic Example:

```
// Interface defines contract
public interface PaymentProcessor {
    Boolean processPayment(Decimal amount);
}

// Different implementations
public class CreditCardProcessor implements PaymentProcessor {
    public Boolean processPayment(Decimal amount) {
        System.debug('Processing credit card payment: ' + amount);
        return true;
    }
}

public class PayPalProcessor implements PaymentProcessor {
    public Boolean processPayment(Decimal amount) {
        System.debug('Processing PayPal payment: ' + amount);
        return true;
    }
}

// Use polymorphism
public void makePayment(PaymentProcessor processor, Decimal amount)
    processor.processPayment(amount);  // Works with any implementat
}
```

Module 9: Apex Triggers

Video 29: Introduction to Triggers

Interview Questions:

1. **Q: What is an Apex Trigger?**
    - **A:** A trigger is code that executes automatically before or after DML operations (insert, update, delete, undelete) on a specific object.

   Basic Example:

```
// Trigger on Account object
trigger AccountTrigger on Account (before insert, after insert) {
    if(Trigger.isBefore && Trigger.isInsert) {
        System.debug('Before inserting accounts');
        for(Account acc : Trigger.new) {
            if(acc.Industry == null) {
                acc.Industry = 'Other';  // Set default value
            }
        }
    }

    if(Trigger.isAfter && Trigger.isInsert) {
        System.debug('After inserting accounts');
        System.debug('Number of accounts inserted: ' + Trigger.new.s
    }
}
```

2. **Q: When should you use a trigger?**
    - **A:** Use triggers when:
        - Complex validation needed
        - Updating related records
        - Cannot be done with declarative tools
        - Need to perform operations before/after DML

   Basic Example:

```
// Use case: Prevent account deletion if it has opportunities
trigger AccountDeletionPrevention on Account (before delete) {
    Set<Id> accountIds = new Set<Id>();
    for(Account acc : Trigger.old) {
        accountIds.add(acc.Id);
    }
```

```
        List<Opportunity> relatedOpps = [
            SELECT Id, AccountId
            FROM Opportunity
            WHERE AccountId IN :accountIds
        ];

        Map<Id, Integer> accountOppCount = new Map<Id, Integer>();
        for(Opportunity opp : relatedOpps) {
            if(!accountOppCount.containsKey(opp.AccountId)) {
                accountOppCount.put(opp.AccountId, 0);
            }
            accountOppCount.put(opp.AccountId, accountOppCount.get(opp.Ac
        }

        for(Account acc : Trigger.old) {
            if(accountOppCount.containsKey(acc.Id)) {
                acc.addError('Cannot delete account with related opportu
            }
        }
    }
```

3. **Q: What are trigger events?**
    - **A:** Events define when the trigger fires: before insert, before update, before delete, after insert, after update, after delete, after undelete.

Basic Example:

```
trigger ContactTrigger on Contact (before insert, before update, afte
    System.debug('Trigger event: ' + Trigger.operationType);

    if(Trigger.isBefore && Trigger.isInsert) {
        System.debug('Before Insert');
    }

    if(Trigger.isBefore && Trigger.isUpdate) {
        System.debug('Before Update');
    }

    if(Trigger.isAfter && Trigger.isInsert) {
        System.debug('After Insert');
    }
    if(Trigger.isAfter && Trigger.isUpdate) {
        System.debug('After Update');
    }

    if(Trigger.isAfter && Trigger.isDelete) {
        System.debug('After Delete');
    }
}
```

4. **Q: Can you have multiple triggers on the same object?**
   - **A:** Yes, but it's bad practice. Use one trigger per object with trigger handler pattern.

Basic Example:

```
// BAD - Multiple triggers (hard to maintain)
// trigger AccountTrigger1 on Account (before insert) { }
// trigger AccountTrigger2 on Account (after update) { }

// GOOD - One trigger with handler
trigger AccountTrigger on Account (before insert, before update, afte
    AccountTriggerHandler.handleTrigger();
}

// Handler class
public class AccountTriggerHandler {
    public static void handleTrigger() {
        if(Trigger.isBefore) {
            if(Trigger.isInsert) {
                handleBeforeInsert();
            }
            if(Trigger.isUpdate) {
                handleBeforeUpdate();
            }
        }
    }

    private static void handleBeforeInsert() {
        // Logic here
    }

    private static void handleBeforeUpdate() {
        // Logic here
        }
    }
```

5. **Q: What is the syntax to create a trigger?**
   - **A:** trigger TriggerName on ObjectName (events) { // code }

   Basic Example:

```
    // Basic syntax
    trigger OpportunityTrigger on Opportunity (before insert, after updat
        // Trigger logic here

        if(Trigger.isBefore && Trigger.isInsert) {
            for(Opportunity opp : Trigger.new) {
                // Validate or modify before insert
                if(opp.Amount == null) {
                    opp.Amount = 0;
                }
            }
        }
    }
```

Video 30: Trigger Context Variables

Interview Questions:

1. **Q: What are trigger context variables?**
   - **A:** Context variables provide information about the trigger execution context: Trigger.new, Trigger.old, Trigger.newMap, Trigger.oldMap, Trigger.size, etc.

   Basic Example:

```
    trigger AccountTrigger on Account (before insert, after update) {
        // Trigger.new - List of new records
        System.debug('New records count: ' + Trigger.new.size());

        // Trigger.old - List of old records (before changes)
        if(Trigger.isUpdate) {
            System.debug('Old records count: ' + Trigger.old.size());
        }
        // Trigger.newMap - Map of new records by Id
        if(Trigger.isUpdate) {
            System.debug('New Map: ' + Trigger.newMap);
        }

        // Trigger.oldMap - Map of old records by Id
        if(Trigger.isUpdate) {
            System.debug('Old Map: ' + Trigger.oldMap);
        }
    }
```

2. **Q: What is Trigger.new?**

   - **A:** Trigger.new is a List of records being inserted or the new versions of records being updated.

   Basic Example:

```
trigger ContactTrigger on Contact (before insert, before update) {
    // Loop through new records
    for(Contact con : Trigger.new) {
        System.debug('Contact Name: ' + con.FirstName + ' ' + con.Las

        // Modify record before save
        if(con.Email == null) {
            con.Email = 'noemail@example.com';
        }
    }
}
```

3. **Q: What is Trigger.old?**

   - **A:** Trigger.old is a List of records before they were updated or deleted. Not available for insert.

   Basic Example:

```
trigger AccountTrigger on Account (before update, before delete) {
    if(Trigger.isUpdate) {
        // Compare old and new values
        for(Account acc : Trigger.old) {
            Account newAcc = Trigger.newMap.get(acc.Id);
            if(acc.Industry != newAcc.Industry) {
                System.debug('Industry changed from ' + acc.Industry
            }
        }
    }

    if(Trigger.isDelete) {
        // Access deleted records
        for(Account acc : Trigger.old) {
            System.debug('Deleting account: ' + acc.Name);
        }
    }
}
```

4. **Q: What are Trigger.newMap and Trigger.oldMap?**
   - **A:** Maps of records with Id as key. Provide quick lookup by record Id.

   Basic Example:

```
trigger OpportunityTrigger on Opportunity (after update) {
    // Trigger.newMap - Map<Id, SObject> of new records
    Map<Id, Opportunity> newOppsMap = Trigger.newMap;

    // Trigger.oldMap - Map<Id, SObject> of old records
    Map<Id, Opportunity> oldOppsMap = Trigger.oldMap;

    for(Id oppId : newOppsMap.keySet()) {
        Opportunity newOpp = newOppsMap.get(oppId);
        Opportunity oldOpp = oldOppsMap.get(oppId);

        // Check if stage changed
        if(newOpp.StageName != oldOpp.StageName) {
            System.debug('Opportunity ' + oppId + ' stage changed');
        }
    }
}
```

5. **Q: What other context variables are available?**
   - **A:** Trigger.isInsert, Trigger.isUpdate, Trigger.isDelete, Trigger.isBefore, Trigger.isAfter, Trigger.size, Trigger.isExecuting

   Basic Example:

```
trigger UniversalTrigger on Account (before insert, before update, a
    System.debug('Is Insert? ' + Trigger.isInsert);
    System.debug('Is Update? ' + Trigger.isUpdate);
    System.debug('Is Delete? ' + Trigger.isDelete);
    System.debug('Is Before? ' + Trigger.isBefore);
    System.debug('Is After? ' + Trigger.isAfter);
    System.debug('Record count: ' + Trigger.size);
    System.debug('Is Executing? ' + Trigger.isExecuting);
}
```

Video 31: Before and After Triggers

Interview Questions:

1. **Q: What's the difference between before and after triggers?**
    - **A:** Before triggers: execute before record is saved to database (can modify values). After triggers: execute after record is saved (have record IDs).

    Basic Example:

```
trigger AccountTrigger on Account (before insert, after insert) {
    if(Trigger.isBefore) {
        // Modify records before they're saved
        for(Account acc : Trigger.new) {
            acc.Description = 'Modified in before trigger';
            // No need to call update - changes are saved automatica
        }
    }

    if(Trigger.isAfter) {
        // Records are saved, have IDs
        for(Account acc : Trigger.new) {
            System.debug('Account created with ID: ' + acc.Id);
            // To modify, must use DML
        }
    }
}
```

2. **Q: When should you use before triggers?**
    - **A:** Use before triggers to:

        - Validate data
        - Set default values
        - Modify records before save
        - Implement complex validation

Basic Example:

```
trigger ContactValidation on Contact (before insert, before update) ·
    for(Contact con : Trigger.new) {
        // Validation
        if(con.Email != null && !con.Email.contains('@')) {
            con.Email.addError('Invalid email format');
        }

        // Set default value
        if(con.LeadSource == null) {
            con.LeadSource = 'Web';
        }

        // Modify record
        if(con.FirstName != null) {
            con.FirstName = con.FirstName.capitalize();
        }
    }
}
```

3. **Q: When should you use after triggers?**

    - **A:** Use after triggers to:
        - Create related records
        - Update other objects
        - Send notifications
        - Access record IDs

Basic Example:

```
trigger AccountAfterInsert on Account (after insert) {
    List<Contact> contactsToCreate = new List<Contact>();

    for(Account acc : Trigger.new) {
        // Create default contact for each account
        Contact con = new Contact(
            FirstName = 'Default',
            LastName = 'Contact',
            AccountId = acc.Id,  // ID is available in after trigger
            Email = 'default@' + acc.Name.toLowerCase().replace(' ',
        );
        contactsToCreate.add(con);
    }

    if(!contactsToCreate.isEmpty()) {
        insert contactsToCreate;
    }
}
```

4. **Q: Can you modify records in after triggers?**

   - **A:** Not directly. You must use DML statements to update records in after triggers.

Basic Example:

```
trigger OpportunityAfterUpdate on Opportunity (after update) {
    List<Opportunity> oppsToUpdate = new List<Opportunity>();

    for(Opportunity opp : Trigger.new) {
        if(opp.StageName == 'Closed Won') {
            // Can't modify directly: opp.Description = 'Won!';

            // Must create new instance and use DML
            Opportunity oppToUpdate = new Opportunity(
                Id = opp.Id,
                Description = 'Opportunity won on ' + System.today()
            );
            oppsToUpdate.add(oppToUpdate);
        }
    }

    if(!oppsToUpdate.isEmpty()) {
        update oppsToUpdate;  // Separate DML required
    }
}
```

5. **Q: Do record IDs exist in before triggers?**

   - **A:** No for insert (records don't have IDs yet). Yes for update/delete.

Basic Example:

```
trigger TestIdAvailability on Account (before insert, before update,
    if(Trigger.isBefore && Trigger.isInsert) {
        for(Account acc : Trigger.new) {
            System.debug('Before Insert - ID: ' + acc.Id);  // null
        }
    }

    if(Trigger.isBefore && Trigger.isUpdate) {
        for(Account acc : Trigger.new) {
            System.debug('Before Update - ID: ' + acc.Id);  // Has II
        }
    }

    if(Trigger.isAfter && Trigger.isInsert) {
        for(Account acc : Trigger.new) {
            System.debug('After Insert - ID: ' + acc.Id);  // Has ID
        }
    }
}
```

Module 10: Apex Testing

Video 32: Test Classes and Methods

Interview Questions:

1. **Q: What is a test class in Apex?**
   - **A:** A test class contains methods that test your Apex code to verify it works correctly. Marked with @isTest annotation.

   Basic Example:

```
// Class to test
public class Calculator {
    public static Integer add(Integer a, Integer b) {
        return a + b;
    }
}

// Test class
```

```
@isTest
public class CalculatorTest {
    @isTest
    static void testAdd() {
        Integer result = Calculator.add(5, 3);
        System.assertEquals(8, result, 'Addition should return 8');
    }
}
```

2. **Q: Why is testing required in Salesforce?**

    - **A:** Testing ensures:
        - Code works correctly
        - 75% code coverage for deployment
        - Prevents bugs
        - Documents expected behavior

Basic Example:

```
// Without test - cannot deploy to production
public class AccountService {
    public static void updateIndustry(Id accountId, String industry)
        Account acc = [SELECT Id FROM Account WHERE Id = :accountId];
        acc.Industry = industry;
        update acc;
    }
}

// With test - can deploy
@isTest
public class AccountServiceTest {
    @isTest
    static void testUpdateIndustry() {
        // Create test data
        Account acc = new Account(Name = 'Test Account');
        insert acc;

        // Test method
        Test.startTest();
        AccountService.updateIndustry(acc.Id, 'Technology');
        Test.stopTest();

        // Verify result
        Account updatedAcc = [SELECT Industry FROM Account WHERE Id =
        System.assertEquals('Technology', updatedAcc.Industry);
    }
}
```

3. **Q: What is @isTest annotation?**
    - **A:** @isTest marks a class or method as a test. Test methods don't count against org limits and don't need access modifiers.

Basic Example:

```
@isTest
public class MyTest {
    // Test method
    @isTest
    static void testMethod1() {
        // Test code
    }

    // Another test method
    @isTest
    static void testMethod2() {
        // Test code
    }
}
```

4. **Q: What is Test.startTest() and Test.stopTest()?**
    - **A:** They create a test context with fresh governor limits. Code between them gets new limits.

Basic Example:

```
@isTest
public class LimitTest {
    @isTest
    static void testWithLimits() {
        // Setup data - uses governor limits
        List<Account> accounts = new List<Account>();
        for(Integer i = 0; i < 100; i++) {
            accounts.add(new Account(Name = 'Test ' + i));
        }
        insert accounts;  // Uses 1 DML statement
        // Start test context - fresh limits
        Test.startTest();

        // This code gets new governor limits
        for(Integer i = 0; i < 100; i++) {
            Account acc = new Account(Name = 'Test2 ' + i);
            // Test code here
        }

        Test.stopTest();  // End test context

        // Assertions after stopTest
        List<Account> results = [SELECT Id FROM Account];
        System.assertEquals(100, results.size());
    }
}
```

5. **Q: What is System.assertEquals()?**
   - **A:** Assertion method to verify expected vs actual values. Test fails if they don't match.

Basic Example:

```
@isTest
public class AssertionTest {
    @isTest
    static void testAssertions() {
        Integer result = 5 + 3;

        // Assert equality
        System.assertEquals(8, result, 'Should equal 8');

        // Assert not null
        System.assertNotEquals(null, result, 'Should not be null');

        // Assert boolean
        Boolean isPositive = result > 0;
        System.assert(isPositive, 'Should be positive');
    }
}
```

Conclusion

This comprehensive guide covers the major topics from the Salesforce Development Course Premium Playlist. Each topic includes:

- Interview questions commonly asked in Salesforce Developer interviews
- Clear explanations with context
- Basic code examples demonstrating the concepts
- Practical use cases

Tips for Interview Success:

1. **Practice code examples** - Type them out and run them in Developer Console
2. **Understand concepts** - Don't just memorize, understand the "why"
3. **Build small projects** - Apply concepts in real scenarios
4. **Review governor limits** - Know the constraints
5. **Master trigger patterns** - Understand trigger handler pattern and bulkification
6. **Write test classes** - Get comfortable with testing
7. **Study SOQL thoroughly** - It's used extensively in development
8. **Understand OOP principles** - Classes, inheritance, interfaces are fundamental

Next Steps:

- Complete the Trailhead modules for hands-on practice
- Build sample applications using these concepts
- Join Salesforce developer communities
- Practice coding challenges
- Review Salesforce documentation

Good luck with your Salesforce Developer journey and interviews!

Document Information:

- Topics Covered: 32+ major topics
- Total Modules: 10
- Examples: 150+ code examples
- Interview Questions: 150+ questions with answers

Resources:

- Salesforce Developer Documentation: developer.salesforce.com
- Trailhead: trailhead.salesforce.com
- S2 Labs Website: s2-labs.com
- YouTube Channel: Salesforce Hulk