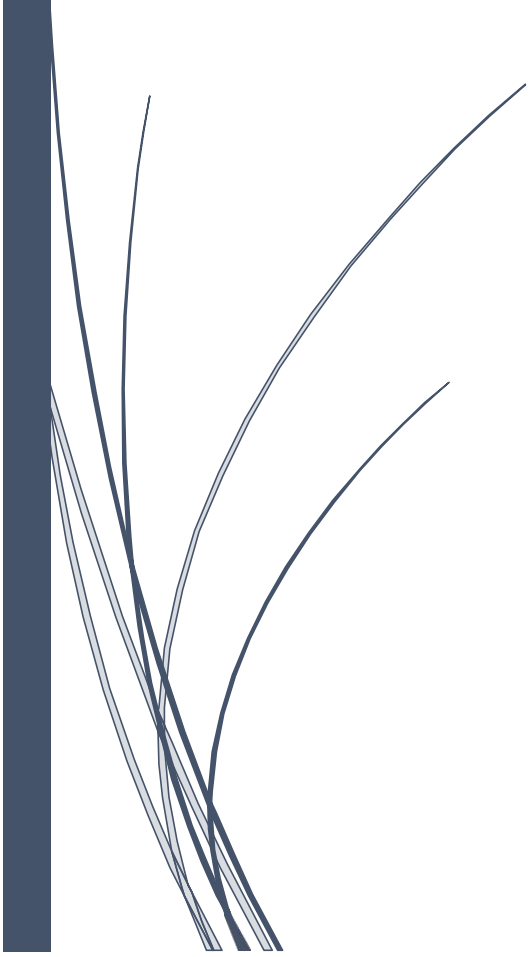
LearnFrenzy

Salesforce Interview Questions



Basic Salesforce Admin Questions

1. What is Salesforce?

Answer:
Salesforce is a cloud-based **Customer Relationship Management (CRM)** platform that helps businesses manage sales, customer support, marketing, and automation. It offers tools like Sales Cloud, Service Cloud, Marketing Cloud, and more.

---

2. What are Objects in Salesforce?

Answer:
Objects are database tables in Salesforce where data is stored. There are two types:

1. **Standard Objects** – Predefined by Salesforce (e.g., Account, Contact, Opportunity).
2. **Custom Objects** – Created by users to store unique business data.

---

3. What is a Profile in Salesforce?

Answer:
A **Profile** controls user permissions, field access, and object access. It defines what a user can **view, create, edit, and delete**. Example profiles: **System Administrator, Standard User.**

---

4. What is a Permission Set?

Answer:
A **Permission Set** is a collection of settings and permissions that can be assigned to users **in addition to their profile**. It helps provide extra permissions without changing profiles.

---

5. What is the difference between Role and Profile?

Answer:

Feature	Profile	Role
Controls	Object & field access	Record visibility (Sharing Rules)
Users Assigned	One profile per user	One role per user (optional)
Example	Read-Only Profile	Sales Manager Role

---

6. What are Record Types in Salesforce?

Answer:
Record Types allow different **page layouts and picklist values** for different users within the same object. Example: In the **Opportunity object**, different record types for **Small Business Sales** and **Enterprise Sales**.

---

7. What is a Page Layout?

Answer:
A **Page Layout** controls how fields, sections, buttons, and related lists appear on a record's detail page.

Security & Access Control Questions

8. What are Organization-Wide Defaults (OWD)?

Answer:
OWD defines the **baseline level of access** for records within Salesforce. It sets the default record visibility for users who don't own the record. Levels:

- **Private** – Only the owner can see the record.
- **Public Read-Only** – Anyone can view, but only the owner can edit.
- **Public Read/Write** – Anyone can view and edit.
- **Controlled by Parent** – Access is determined by the parent record.

---

9. What are Sharing Rules?

Answer:
Sharing Rules extend record access to users based on roles, public groups, or territories. They are used when OWD is set to Private but we need to share specific records.

---

10. What is a Role Hierarchy?

Answer:
A **Role Hierarchy** allows record access to be automatically granted to users higher up in the hierarchy. Example: A **Sales Manager** can see records of their **Sales Reps**.

Automation & Workflow Questions

11. What is a Workflow Rule?

Answer:
A **Workflow Rule** is an automation tool that performs actions like **sending emails, updating fields, creating tasks, or sending outbound messages** when a certain condition is met.

---

12. What is the difference between Workflow and Process Builder?

Answer:

Feature	Workflow Rule	Process Builder
Actions	Field updates, Email alerts, Tasks, Outbound messages	Everything in Workflow + Create records, Post to Chatter, Call Apex
Complexity	Simple	More advanced logic
Future Scope	Being replaced by Flow	Recommended

---

13. What is a Flow in Salesforce?

Answer:
Flow is a powerful automation tool that **collects data, updates records, creates new records, and makes complex decisions** with clicks instead of code.

Data Management Questions

14. What is Data Loader in Salesforce?

Answer:
Data Loader is a tool used to **import, update, delete, and export** data in bulk (up to 5 million records). It is used when working with large data sets.

---

15. What are Validation Rules?

Answer:
Validation Rules ensure data quality by enforcing specific **business rules** before saving a record. Example: A rule to ensure that the **Phone Number** field is always entered.

Reports & Dashboards Questions

16. What is a Report in Salesforce?

Answer:
A **Report** is a collection of **filtered data** presented in rows and columns. Types of reports:

1. **Tabular Report** – Simple table format.
2. **Summary Report** – Grouped data with subtotals.
3. **Matrix Report** – Data grouped by rows & columns.
4. **Joined Report** – Multiple reports in one.

---

17. What is a Dashboard?

Answer:
A **Dashboard** is a visual representation of multiple reports using **charts, graphs, and metrics**. It helps monitor key performance indicators (KPIs).

Other Important Questions

18. What are Queues in Salesforce?

Answer:
A **Queue** is a group of users who share responsibility for managing records. Example: A **Case Queue** where support agents can pick cases to work on.

---

19. What are Public Groups?

Answer:
A **Public Group** is a collection of users, roles, and other public groups used to simplify **sharing rules and email alerts**.

---

20. What are Approval Processes in Salesforce?

Answer:
An **Approval Process** automates **record approval** workflows. Example: A manager must approve a **discount request** before finalizing a deal.

Bonus: Scenario-Based Questions

21. A sales team needs different picklist values than the support team. How would you configure this?

Answer: Use **Record Types** with different **picklist values** for Sales and Support.

22. A user cannot see an Opportunity record. What will you check?

Answer:

1. Check **OWD (Private/Public)** settings.
2. Check if the user has proper **Role & Profile** permissions.
3. Check **Sharing Rules** and **Manual Sharing**.

Final Tips for the Interview:

- **Understand the concepts** instead of memorizing answers.
- **Explain with examples** when possible.
- **Practice hands-on** in a Salesforce Developer Org.
- **Be confident and structured in your answers.**

Scenario-Based Salesforce Administrator Questions & Answers (Using Flow)

1. Scenario: Profile vs. Permission Set

Question: A user needs **edit access to Contacts**, but their Profile only allows **read access**. How will you fix this without changing their Profile?

🟩 Solution:

- I will create a **Permission Set** that grants **edit access** to the Contact object.
- I will assign this **Permission Set** to the user.
- This allows additional permissions without modifying the user's Profile.

🔹 **Why Not Change the Profile?**
Profiles are **fixed** and **not flexible**. Using **Permission Sets** allows us to grant extra permissions to specific users **without affecting others**.

---

2. Scenario: Missing Record Access

Question: A sales representative cannot see an **Opportunity record** owned by a different team. How will you troubleshoot?

🟩 Solution:
I will check:
1. **OWD (Organization-Wide Defaults)** – If set to **Private**, users can't see others' records. 2. **Role Hierarchy** – If the sales rep is below in the hierarchy, they may not have access.
3. **Sharing Rules** – I will create a **Sharing Rule** to grant access to the required group. 4. **Manual Sharing** – The owner can manually share the record if needed.

🔹 **Best Practice: Use Role Hierarchy & Sharing Rules** instead of manual sharing.

---

3. Scenario: Restricting Field Editing

Question: A manager wants to ensure that only certain users can update the **Discount field** on the Opportunity. How will you achieve this?

🟩 Solution:

- I will use **Field-Level Security (FLS)** to **hide** the Discount field from unauthorized profiles.
- If certain users should see but not edit the field, I will create a **Validation Rule**:

🔹 Validation Rule Example:

```
AND(ISCHANGED(Discount__c), NOT($Profile.Name = "Sales Manager"))
```

This prevents changes unless the user is a **Sales Manager**.

---

4. Scenario: Automating Follow-Ups

Question: When a new **lead** is created, the system should send an **email reminder** to the assigned sales rep after **3 days** if the lead is not contacted. How will you set this up?

🟩 Solution (Using Flow):

1. **Create a Record-Triggered Flow** on the **Lead** object. 2.
Set Entry Conditions:

- Lead Status = 'New'
- Lead Owner is assigned
  Add a Scheduled Path:
- Delay the action for **3 days**
  Add an Action:
- Send an **Email Alert** to the Lead Owner.
  Save & Activate the Flow.

🟠 Why Use Flow Instead of Workflow?

- Workflows **cannot** handle **complex logic** or create records.
- Salesforce **announced** that Workflow Rules and Process Builder will be **deprecated**.
- Flow is the **future-proof solution** that allows **scheduled automation and decision-making**.

---

5. Scenario: Report Access Issue

Question: A user reports that they **cannot see a report** in a shared folder. How will you fix this?

🟩 Solution:

I will check:

1. **Report Folder Permissions** – Ensure the user has **Viewer** or **Editor** access. 2. **Object Permissions** – Check if the user has access to the objects in the report. 3. **Field-Level Security** – Ensure the fields in the report are visible to the user.

---

6. Scenario: Preventing Duplicate Records

Question: Users are creating duplicate **Accounts** with the same name. How will you prevent this?

🟩 Solution:

1. Create a **Matching Rule** to detect duplicate Account Names.
2. Create a **Duplicate Rule** to **block duplicates** or **allow but alert users**.
3. Optionally, use **Validation Rules** to enforce specific naming conventions.

🔹 **Best Practice: Use Duplicate Rules instead of Apex Triggers** for standard duplication checks.

---

7. Scenario: Conditional Picklist Values

Question: The **Stage** picklist in Opportunities should show different values for **Sales Reps** and **Sales Managers**. How will you configure this?

🟩 Solution:

- Create **two Record Types** for Sales Reps and Sales Managers.
- Assign different **picklist values** to each Record Type.
- Ensure Profiles are assigned to the correct Record Type.

---

8. Scenario: Automating Approval Process

Question: When a deal's discount exceeds **20%**, a manager must approve it before closing. How will you configure this?

🟩 **Solution (Using Approval Process):**

1. **Entry Criteria:** Discount > 20%.
2. **Approval Step:** Assign to the Sales Manager.
3. **Final Action:** If approved, status updates to 'Approved'; if rejected, notify the sales rep.

---

9. Scenario: User Cannot Log In

Question: A user reports that they **cannot log in** to Salesforce. How will you troubleshoot?

🟩 **Solution:**

1. **Check Login History** – Look for errors like **Incorrect Password** or **IP Restriction**. 2. **Check Profile Login Hours** – Ensure login is allowed at that time.
3. **Check IP Whitelisting** – Ensure they are logging in from an allowed IP range.
4. **Check Multi-Factor Authentication (MFA)** – Ensure they have completed MFA setup.

---

10. Scenario: Data Importing Issue

Question: You need to import **10,000 Contacts** from an Excel file into Salesforce. How will you do it?

🟩 **Solution:**

- I will use **Data Loader** because:
  - 🟩 It supports bulk data import (up to 5 million records).
  - 🟩 I can map fields correctly before import.
  - 🟩 It allows data validation before inserting records.

🔹 **Alternative:** If under **50,000 records**, use **Data Import Wizard** for a simpler UI.

🟠 Basic Apex Interview Questions & Answers

1. What is Apex in Salesforce?

Answer:
Apex is **a strongly-typed, object-oriented programming language** used in Salesforce to execute business logic. It is similar to Java and is designed to run **on the Salesforce platform**. Apex allows developers to write **triggers, controllers, batch jobs, integrations, and test classes** to customize and automate processes.

---

2. What are the key features of Apex?

Answer:
Apex provides:
✓ **Object-Oriented Programming (OOP)** - Supports classes, interfaces, and inheritance.
✓ **Database Integration** - Works seamlessly with SOQL and DML statements.
✓ **Governor Limits** - Ensures efficient code execution to maintain multi-tenancy.
✓ **Asynchronous Processing** - Supports Batch Apex, Queueable Apex, and Future methods.
✓ **Built-in Exception Handling** - Provides robust error-handling mechanisms.
✓ **Trigger-based Execution** - Automates logic before or after DML operations.

---

3. What are different data types in Apex?

Answer:
Apex supports:
✓ **Primitive Data Types** - `Integer`, `Double`, `Boolean`, `String`, `Date`, `Datetime`, `Long`, `Decimal`.
✓ **Collections** - `List`, `Set`, and `Map`.
✓ **Objects** - Instances of standard/custom Salesforce objects (`Account`, `Contact`, etc.).
✓ **Enums** - Used to define a set of constant values.

---

4. What are Apex classes and methods?

Answer:
An **Apex class** is a blueprint that defines variables and methods. A **method** is a function inside a class that performs a specific action.

— Example:

```
apex

public class MyClass {
    public String greet(String name)
        { return 'Hello, ' + name;
    }
}
```

This class has a method `greet()` that takes a name and returns a greeting message.

---

5. What is the difference between Apex classes and triggers?

Answer:

Feature	Apex Class	Apex Trigger
Purpose	Defines business logic	Automates processes on data changes
Execution	Called explicitly in code	Executes automatically on DML events
Methods	Supports multiple methods	Does not support methods directly
Bulk Processing	Needs manual bulkification	Automatically processes bulk records

---

6. What are different types of Apex collections?

Answer:
✓ **List** – Ordered collection, allows duplicates.
✓ **Set** – Unordered collection, does not allow duplicates.
✓ **Map** – Key-value pairs for fast lookup.

-- Example:

```
apex

List<String> myList = new List<String>{'A', 'B', 'C'};
Set<Integer> mySet = new Set<Integer>{1, 2, 3};
Map<Integer, String> myMap = new Map<Integer, String>{1 => 'A', 2 => 'B'};
```

---

7. What is SOQL, and how is it different from SQL?

Answer:
SOQL (Salesforce Object Query Language) is used to **retrieve data** from Salesforce objects. Unlike SQL, it does not support `JOIN` but uses **relationship queries**.

-- Example:

```
apex

SELECT Name, Email FROM Contact WHERE LastName = 'Smith'
```

This SOQL query fetches **Name and Email** from **Contact** where `LastName = 'Smith'`.

---

8. What is the difference between SOQL and SOSL?

Answer:

Feature	SOQL (Salesforce Object Query Language)	SOSL (Salesforce Object Search Language)
Used For	Querying specific objects	Searching across multiple objects
Search Type	Structured queries	Full-text search
Data Scope	Fields of specified objects	Any field (including unstructured text)

-- Example:

```
apex

// SOQL: Retrieves contacts named 'John'
SELECT Name FROM Contact WHERE Name = 'John'

// SOSL: Searches 'John' across multiple objects
FIND 'John' IN ALL FIELDS RETURNING Contact(Name), Account(Name)
```

---

9. What are Governor Limits in Apex?

Answer:
Governor Limits are **restrictions imposed by Salesforce** to ensure efficient use of resources in a multi-tenant environment.

Key limits:
✓ **SOQL Queries** – Max **100 queries per transaction**
✓ **DML Statements** – Max **150 DML operations per transaction**
✓ **CPU Time Limit** – Max **10,000ms per transaction**
✓ **Heap Size** – Max **6 MB for synchronous, 12 MB for async Apex**

🔹 **To avoid hitting limits, we must use bulkification, efficient SOQL, and optimized DML operations.**

---

10. What is the 'with sharing' and 'without sharing' keyword in Apex?

Answer:
✓ **With Sharing** – Enforces **user's security settings** (CRUD, FLS, Sharing Rules).
✓ **Without Sharing** – Ignores security rules and runs with **system-level access**.

🔸 **Example:**

```
apex

public with sharing class MyClass { } // Enforces sharing rules
public without sharing class MyClass { } // Ignores sharing rules
```

Intermediate Apex Interview Questions & Answers

11. What are Apex Triggers? Explain the different types.

Answer:
An **Apex Trigger** is code that runs **before or after DML operations** (insert, update, delete).

🟩 **Types of Triggers:**
✓ **Before Trigger** – Runs **before** the record is saved to the database.
✓ **After Trigger** – Runs **after** the record is saved.

-- **Example:**

```
apex

trigger AccountTrigger on Account (before insert, after update)
    { if (Trigger.isBefore) {
        for (Account acc : Trigger.new)
            { acc.Name = 'Updated Name';
        }
    }
}
```

---

12. What is the difference between 'before' and 'after' triggers?

Answer:

Trigger Type	Runs	Use Case
Before Trigger	Before saving the record to DB	Modify values before saving
After Trigger	After saving the record to DB	Access record ID, perform DML on related records

---

13. What is a Trigger Context Variable?

Answer:
Trigger Context Variables provide metadata about the DML event.

🟩 **Important Context Variables:**
✓ `Trigger.new` – List of **new records** being inserted/updated.
✓ `Trigger.old` – List of **old records** before update/delete.
✓ `Trigger.isInsert` – Checks if the trigger fired on insert.
✓ `Trigger.isUpdate` – Checks if the trigger fired on update.

---

14. What is a Recursive Trigger? How do you avoid recursion in Apex?

Answer:
A **Recursive Trigger** occurs when a trigger calls itself indefinitely.

🟩 **Solution:** Use a **Static Boolean Variable**

```
apex
```

```
public class TriggerHandler {
    public static Boolean isTriggerExecuted = false;
}

trigger AccountTrigger on Account (before update)
    { if (!TriggerHandler.isTriggerExecuted) {
        TriggerHandler.isTriggerExecuted = true;
        // Perform logic
    }
}
```

15. What are Future Methods in Apex? Why are they used?

Answer:
A **Future Method** in Apex runs **asynchronously** to perform long-running tasks, such as **callouts to external systems or heavy processing**, without blocking the main execution.

🟩 **Key Features:**
✓ Runs in a **separate thread**
✓ Supports **callouts** (if `callout=true` is specified)
✓ Cannot return values or be called from another future method

🔸 **Example:**

```
apex

public class FutureExample
    { @future(callout=true)
    public static void sendRequest(String name)
        { System.debug('Processing: ' + name);
    }
}
```

Use Case: If you need to **call an external API**, but Salesforce doesn't allow callouts inside triggers, you can use a **future method** to bypass this limitation.

---

16. What is a Queueable Apex? How is it different from Future Methods?

Answer:
Queueable Apex is used for **asynchronous processing**, but it is **more flexible** than future methods.

🟩 Key Differences from Future Methods:

✓ Supports **chaining** (one queueable job can trigger another).
✓ Allows **complex objects** like `sObjects` and `Custom Objects`.
✓ Better **debugging & monitoring** in Apex Jobs.

🔸 Example:

```
apex

public class MyQueueableJob implements Queueable
    { public void execute(QueueableContext context)
    {
        System.debug('Queueable Job Running...');
    }
}
```

To run it:

```
apex

ID jobId = System.enqueueJob(new MyQueueableJob());
```

---

17. What is Batch Apex? When should we use it?

Answer:
Batch Apex is used when we need to process **large amounts of data** asynchronously in chunks to **avoid governor limits**.

🟩 Key Features:

✓ Divides records into **small batches** for processing.
✓ Can process **millions of records**.
✓ Uses `Database.QueryLocator` for efficient querying.

🔸 Example:

```
apex

global class MyBatchClass implements Database.Batchable<sObject>
    { global Database.QueryLocator start(Database.BatchableContext BC)
    {
        return Database.getQueryLocator('SELECT Id, Name FROM Account');
    }
    global void execute(Database.BatchableContext BC, List<Account> scope)
        { for (Account acc : scope) {
            acc.Name = 'Updated Name';
        }
        update scope;
    }
    global void finish(Database.BatchableContext BC)
        { System.debug('Batch Job Completed!');
```

```
    }
}
```

To run it:

```
apex

MyBatchClass batchJob = new MyBatchClass();
ID batchId = Database.executeBatch(batchJob, 100);
```

Use Case: If you need to **update 1 million records**, a normal Apex trigger will hit governor limits. Instead, you should use **Batch Apex**.

---

18. What is the difference between Future, Queueable, and Batch Apex?

Answer:

Feature	Future Method	Queueable Apex	Batch Apex
Processing Type	Asynchronous	Asynchronous	Asynchronous (Bulk)
Supports Chaining	+ No	✅ Yes (one job at a time)	✅ Yes (via finish method)
Can Process Large Data?	+ No	+ No	✅ Yes (millions of records)
Callouts Allowed?	✅ Yes	✅ Yes	✅ Yes
Supports Complex Objects?	+ No	✅ Yes	✅ Yes

---

19. What is the difference between Database.insert() and insert DML statement?

Answer:

Feature	`insert` DML Statement	`Database.insert()` Method
Exception Handling	Throws an error & stops execution	Can handle errors using `false` option
Partial Success	+ No (stops at first error)	✅ Yes (allows partial success)
Returns a Result	+ No	✅ Yes (returns `Database.SaveResult[]`)

-- **Example:**

```
apex

// Using insert (stops execution on error)
insert accList;

// Using Database.insert() (continues execution)
Database.SaveResult[] results = Database.insert(accList, false);
for (Database.SaveResult sr : results) {
    if (!sr.isSuccess()) {
        System.debug('Error: ' + sr.getErrors()[0].getMessage());
    }
}
```

---

20. What is the difference between Synchronous and Asynchronous Apex?

Answer:
✓ **Synchronous Apex** – Runs **immediately** in a single transaction.
✓ **Asynchronous Apex** – Runs **in the background** to avoid blocking execution.

Feature	Synchronous Apex	Asynchronous Apex
Execution	Runs in real-time	Runs in the background
Used For	Small operations	Heavy processing (callouts, bulk jobs)
Example	Normal Apex classes, triggers	Future, Queueable, Batch Apex

---

21. What are Governor Limits, and how do you avoid hitting them?

Answer:
Governor Limits ensure that no single Apex transaction **overuses resources** in Salesforce's multi-tenant environment.

🟩 **Ways to Avoid Governor Limits:**
✓ **Use Bulkified SOQL Queries** – Instead of running queries inside a loop, use lists.
✓ **Minimize DML Operations** – Use collections and update records in bulk.
✓ **Use Asynchronous Processing** – Move long-running tasks to Future, Queueable, or Batch Apex.
✓ **Use Limits Class** – Check `Limits.getQueries()` to track limits.

-- **Example:** (+ Bad – Query inside a loop)

```
apex
```

```
for (Contact c : contactList) {
    Account acc = [SELECT Name FROM Account WHERE Id = :c.AccountId]; // BAD!
}
```

✅ Fixed Version:

```
apex

Set<Id> accIds = new Set<Id>();
for (Contact c : contactList) {
    accIds.add(c.AccountId);
}
Map<Id, Account> accMap = new Map<Id, Account>([SELECT Id, Name FROM Account
WHERE Id IN :accIds]);
```

---

22. What is the @TestVisible annotation in Apex?

Answer:
`@TestVisible` is used to **expose private methods and variables** to test classes.

🔹 Example:

```
apex

public class MyClass {
    @TestVisible private static Integer counter = 0;
}
```

Without `@TestVisible`, private variables **cannot be accessed** in test classes.

---

23. What are Custom Metadata and Custom Settings in Apex?

Answer:
✓ **Custom Metadata** – Used to store **configuration data** that is deployable via metadata API.
✓ **Custom Settings** – Used for **static application settings** that can be accessed without SOQL queries.

---

24. What is the difference between @isTest and Test.startTest()/Test.stopTest()?

Answer:

✓ **@isTest** – Used to **define test classes and methods**.

✓ **Test.startTest() & Test.stopTest()** – Used to **isolate governor limits for testing asynchronous operations**.

-- Example:

```
apex

@isTest
private class MyTestClass
    { @isTest
    static void testMethod()
        { Test.startTest();
        // Call the method that runs a future/queueable job
        Test.stopTest();
    }
}
```

---

25. What is the best way to handle exceptions in Apex?

Answer:

✓ Use **Try-Catch Blocks**

✓ Use **Custom Exceptions**

✓ Use **Error Logging** in a Custom Object

-- Example:

```
apex

try {
    Account acc = [SELECT Id FROM Account WHERE Name = 'Test'];
} catch (Exception e) {
    System.debug('Error: ' + e.getMessage());


}
```

Advanced Apex Interview Questions & Answers

---

26. How does Apex enforce security (CRUD, FLS, Sharing Rules)?

Answer:
Apex does **NOT** automatically enforce **CRUD (Create, Read, Update, Delete)**, **FLS (Field-Level Security)**, or **Sharing Rules**. We must **manually enforce** them.

🟩 **How to Enforce Security in Apex:**
✓ **CRUD Security (Check User Permissions on Object)**

```
apex

if (Schema.sObjectType.Account.isAccessible()) {
    Account acc = [SELECT Id, Name FROM Account LIMIT 1];
}
```

✓ **FLS Security (Check Field Access Before Querying)**

```
apex

if (Schema.sObjectType.Account.fields.Name.isAccessible())
    { Account acc = [SELECT Name FROM Account LIMIT 1];
}
```

✓ **Sharing Rules (Use `with sharing` Keyword)**

```
apex

public with sharing class AccountHandler
    { public static void getAccounts() {
        List<Account> accList = [SELECT Id, Name FROM Account];
    }
}
```

---

27. What are the different types of Apex Triggers?

Answer:
✓ **Before Triggers** – Used to **update records before they are saved** to the database.
✓ **After Triggers** – Used when we need to **use record Ids** (e.g., inserting related records).

🟩 Example: Before Insert Trigger (Prevent Duplicate Accounts)

```
apex

trigger AccountBeforeInsert on Account (before insert)
    { for (Account acc : Trigger.new) {
        if ([SELECT Count() FROM Account WHERE Name = :acc.Name] > 0)
            { acc.addError('Duplicate Account Name not allowed');
        }
    }
}
```

🟩 Example: After Insert Trigger (Create Contact After Account Creation)

```
apex

trigger AccountAfterInsert on Account (after insert)
    { List<Contact> contacts = new List<Contact>();
    for (Account acc : Trigger.new) {
        contacts.add(new Contact(LastName = acc.Name, AccountId = acc.Id));
    }
    insert contacts;
}
```

---

28. What are Apex Trigger Context Variables?

Answer:
Context variables help us determine **when and how the trigger runs**.

🟩 Important Context Variables:

✓ `Trigger.new` – Contains **new records** in Insert/Update.
✓ `Trigger.old` – Contains **old records** in Update/Delete.
✓ `Trigger.isInsert` – Returns `true` if it is an Insert event.
✓ `Trigger.isUpdate` – Returns `true` if it is an Update event.
✓ `Trigger.isDelete` – Returns `true` if it is a Delete event.
✓ `Trigger.isBefore` – Runs **before DML operation**.
✓ `Trigger.isAfter` – Runs **after DML operation**.

---

29. What is a Custom Exception in Apex? Why do we use it?

Answer:
A Custom Exception is a **user-defined exception** used to handle specific error cases.

🟩 Example:

```
apex

public class CustomException extends Exception {}

public class AccountHandler {
    public static void createAccount(String accName)
        { if (accName == null) {
            throw new CustomException('Account Name cannot be null');
        }
    }
}
```

---

30. What is a Wrapper Class in Apex?

Answer:
A **Wrapper Class** is a **custom class that wraps multiple objects** together in a single unit.

🟩 Example:

```
apex

public class AccountWrapper {
    public Account acc { get; set; }
    public Boolean isSelected { get; set; }

    public AccountWrapper(Account acc, Boolean isSelected)
        { this.acc = acc;
        this.isSelected = isSelected;
    }
}
```

Use Case: Used in LWC or Visualforce pages to send multiple types of data.

---

31. What are Platform Events in Apex? How do they work?

Answer:
Platform Events enable real-time event-driven communication between systems.

🟩 Steps to Use Platform Events:

1. **Create a Platform Event** (via Setup → Platform Events). 2.
Publish the Event in Apex

```
MyEvent__e eventMessage = new MyEvent__e(Message__c='Test Event');
EventBus.publish(eventMessage);
```

3. **Subscribe to Events using Apex Triggers or LWC**

---

32. What is the difference between Queueable and Batch Apex?

Answer:

Feature	Queueable Apex	Batch Apex
Execution	Runs Once	Runs in Batches
Chaining	✅ Yes	✅ Yes (via finish method)
Large Data Processing	❌ No	✅ Yes (millions of records)
SOQL Queries	1 Query	Multiple Queries Allowed

---

33. What is the @AuraEnabled annotation in Apex?

Answer:
The `@AuraEnabled` annotation exposes **Apex methods** to **LWC or Aura Components**.

✅ **Example:**

```
apex

public with sharing class AccountController
    { @AuraEnabled(cacheable=true)
    public static List<Account> getAccounts()
        { return [SELECT Id, Name FROM Account];
    }
}
```

---

34. How do you call a REST API from Apex?

Answer:
✓ **Using `Http` and `HttpRequest` Classes**

```
apex

public class APIClient {
    public static void callExternalAPI()
        { Http http = new Http();
        HttpRequest request = new HttpRequest();
        request.setEndpoint('https://api.example.com/data');
        request.setMethod('GET');
        HttpResponse response = http.send(request);
        System.debug(response.getBody());
    }
```

```
}
```

---

35. What is the Difference Between Database.Stateful and Database.Batchable?

Answer:

Feature	`Database.Stateful`	`Database.Batchable`
Purpose	Maintains state between batches	Runs independently for each batch
Use Case	Track changes across batches	Process bulk records efficiently

---

36. What are the different types of Collections in Apex?

Answer:
✓ **List** – Ordered collection of elements
✓ **Set** – Unique, unordered collection
✓ **Map** – Key-value pair storage

Example:

```
apex

List<String> names = new List<String>{'John', 'Doe'};
Set<String> uniqueNames = new Set<String>{'John', 'Doe'};
Map<String, String> countryCodes = new Map<String, String>{'US' => 'United
States'};
```

---

37. How do you handle limits in Apex callouts?

Answer:
✓ **Use Asynchronous Apex (Future/Queueable Methods)**
✓ **Limit Concurrent Callouts**
✓ **Batch Requests to Reduce API Calls**

---

38. What is Apex Email Service?

Answer:
Apex Email Service allows Salesforce to **process incoming emails** and **send automated email notifications**.

🟩 Example: Inbound Email Handler

```
apex

global class EmailHandler implements Messaging.InboundEmailHandler
    { global Messaging.InboundEmailResult
handleInboundEmail(Messaging.InboundEmail email,
 Messaging.InboundEnvelope env) {
        Messaging.InboundEmailResult result =
 new Messaging.InboundEmailResult();
        System.debug('Email Received: ' + email.subject);
        return result;
    }
}
```

---

39. What is a Dynamic SOQL Query in Apex?

Answer:
A **Dynamic SOQL Query** is built **at runtime** instead of being hardcoded.

🟩 Example:

```
apex

String query = 'SELECT Id, Name FROM Account WHERE Name LIKE \'%' +
searchTerm + '%\'';
List<Account> accList = Database.query(query);
```

---

40. What is a Polymorphic SOQL Query?

Answer:
A **Polymorphic SOQL Query** is used when a field can refer to multiple objects.

🟩 **Example:** Querying the `WhatId` field on the Task object (which can be related to different objects).

```
apex

SELECT Id, WhatId, What.Name FROM Task
```

Expert-Level Apex Interview Questions & Answers

---

41. What is the Apex Transactions model?

Answer:
Apex follows the **transaction model**, where a group of **DML operations** are executed as a **single unit of work**.

✓ If all operations succeed → 🟩 Changes are committed.
✓ If any operation fails → + Entire transaction is rolled back.

🟩 Example (Single Transaction with Rollback)

```
apex

Savepoint sp = Database.setSavepoint();
try {
    Account acc = new Account(Name='Test Account');
    insert acc;

    Contact con = new Contact(FirstName='John', AccountId=acc.Id);
    insert con;

    // Simulating an error
    con.LastName = null;
    update con;
} catch (Exception e)
    { Database.rollback(sp);
    System.debug('Transaction Rolled Back: ' + e.getMessage());
}
```

---

42. What is the difference between `with sharing`, `without sharing`, and `inherited sharing`?

Answer:

Keyword	Description
`with sharing`	**Respects** the user's sharing rules.
`without sharing`	**Ignores** the user's sharing rules.
`inherited sharing`	**Inherits** sharing settings from the caller class.

🟩 Example:

```
apex

public with sharing class AccountHandler
    { public static List<Account> getAccounts()
    {
        return [SELECT Id, Name FROM Account];
    }
}
```

Best Practice: Use `inherited sharing` for security best practices.

---

43. How do you prevent concurrent updates in Apex?

Answer:
To **prevent record conflicts**, we use:

1. **FOR UPDATE Keyword** (Locks the record until the transaction completes)

```
apex

Account acc = [SELECT Id, Name FROM Account WHERE Id = '001...' FOR UPDATE];
acc.Name = 'Updated Name';
update acc;
```

2. **Optimistic Locking** (System automatically checks for changes) 3.
Platform Events (Real-time updates to avoid conflicts)

---

44. How do you handle large data volumes (LDV) in Apex?

Answer:
Salesforce imposes **governor limits**, so handling **millions of records** requires:

✓ **Batch Apex** (Processes 200 records per batch)
✓ **Asynchronous Processing** (`@future`, Queueable)
✓ **SOQL Best Practices** (Avoid `SELECT *`, Use **indexed fields**)
✓ **Data Skew Handling** (Avoid single-owner record skew)

🟩 Example: Batch Apex for Large Data Processing

```
apex

global class AccountBatch implements Database.Batchable<sObject> {
```

```
    global Database.QueryLocator start(Database.BatchableContext BC)
        { return Database.getQueryLocator('SELECT Id, Name FROM Account
        WHERE
CreatedDate = LAST_N_DAYS:30');
    }

    global void execute(Database.BatchableContext BC, List<Account> scope)
        { for (Account acc : scope) {
            acc.Name = 'Updated Name';
        }
        update scope;
    }

    global void finish(Database.BatchableContext BC)
        { System.debug('Batch Completed!');
    }
}
```

45.What is the difference between Future Methods, Queueable, and Batch Apex?

Answer:

Feature	Future Method	Queueable Apex	Batch Apex
Execution	Asynchronous	Asynchronous	Batch Processing
Supports Chaining	+ No	✅ Yes	✅ Yes (in finish method)
Supports Complex Processing	+ No	✅ Yes	✅ Yes
Transaction Control	+ No	✅ Yes	✅ Yes

✅ When to Use?

- **Future Method** → Simple callouts, lightweight processing.
- **Queueable Apex** → Advanced processing, supports chaining.
- **Batch Apex** → Large data processing in batches.

---

46. What is the `Continuation` class in Apex?

Answer:
The `Continuation` class is used for **long-running callouts** (up to 120 seconds).

✅ Example:

```
public class LongRunningCallout
    { @AuraEnabled
    public static Object makeCallout()
        { Continuation con = new Continuation(120);
```

```
        HttpRequest req = new HttpRequest();
        req.setEndpoint('https://api.example.com');
        req.setMethod('GET');
        con.addHttpRequest(req);
        return con;
    }
}
```

---

47. How do you monitor Apex logs and governor limits?

Answer:
✓ Use **Developer Console** → "Logs" Tab
✓ Use **`Limits` Class**

```
apex

System.debug('SOQL Queries Used: ' + Limits.getQueries());
System.debug('DML Statements Used: ' + Limits.getDMLStatements());
```

---

48. What is the Transaction Finalizer in Apex?

Answer:
Finalizers **execute code after Queueable Apex completes**, even if it fails.

Example:

```
apex

public class MyQueueable implements Queueable
    { public void execute(QueueableContext context)
    {
        System.debug('Processing Queueable Job...');
    }
}

public class MyFinalizer implements Finalizer
    { public void execute(FinalizerContext context)
    {
        System.debug('Queueable Job Completed.');
    }
}
```

---

49. How do you handle bulk API callouts in Apex?

Answer:
✓ Use **Batch Apex** for callouts (Set `Database.AllowsCallouts=true`)

✓ Use **Continuation Apex** for parallel processing
✓ Use **Platform Events** to avoid hitting callout limits

---

50. What is Custom Metadata and how is it different from Custom Settings?

Answer:

Feature	Custom Metadata	Custom Settings
Deployment	🟩 Can be deployed via metadata API	+ Cannot be deployed
Governor Limits	+ Does not count against SOQL limits	🟩 Counts against limits
Purpose	Store **configuration** data	Store **custom application** data

🟩 **Example: Querying Custom Metadata in Apex**

```
apex

List<My_Metadata__mdt> metaData = [SELECT DeveloperName, Field1__c FROM
My_Metadata__mdt];
```

---

51. How do you optimize Apex performance?

Answer:
✓ **Bulkify SOQL/DML** (Avoid loops)
✓ **Use `@AuraEnabled(cacheable=true)`** (for LWC caching)
✓ **Use Query Plan Tool** (For optimizing SOQL queries)
✓ **Avoid Recursive Triggers** (Use Static Variables)

---

52. How do you debug Apex code efficiently?

Answer:
✓ Use `System.debug()` for logging
✓ Use **Checkpoints** in Developer Console
✓ Use `Log Inspector` to analyze Apex Execution

---

53. What is a Skinny Table in Salesforce?

Answer:
A **Skinny Table** is a **performance optimization technique** used to improve SOQL query performance on **large objects**.

✓ Salesforce automatically creates them for standard objects.
✓ They contain frequently used fields to reduce joins.
✓ Cannot be modified manually.

---

54. How do you implement OAuth authentication in Apex?

Answer:
✓ Generate Access Token using `HttpRequest`
✓ Use `Authorization` Header for API Calls

🟩 Example:

```
apex

HttpRequest req = new HttpRequest();
req.setEndpoint('https://login.salesforce.com/services/oauth2/token');
req.setMethod('POST');
req.setBody('grant_type=password&client_id=xxx&client_secret=yyy');
HttpResponse res = new Http().send(req);
```

🟠 Basic LWC Interview Questions

1. What is LWC? How is it different from Aura Components?

Answer:
LWC (Lightning Web Component) is a modern **framework** built on web standards like JavaScript, HTML, and CSS to create **lightweight, fast, and efficient** UI components in Salesforce.

Differences between LWC and Aura:

- **Performance** → LWC is **faster** because it uses native browser capabilities, while Aura relies on a JavaScript framework.
- **Syntax** → LWC uses **modern JavaScript (ES6+)**, making it more aligned with industry standards.
- **Shadow DOM** → LWC uses **Shadow DOM**, providing better security and encapsulation.
- **Reusability** → LWC components can be easily reused inside **Aura** components, but not the other way around.

---

2. What are the advantages of using LWC?

Answer:
LWC offers **several advantages**, making it the preferred choice for Salesforce development:
🟩 **Performance** → Faster execution due to **native browser support**.
🟩 **Security** → Uses **Shadow DOM** for encapsulation and better security.
🟩 **Simplified Development** → Uses **modern JavaScript (ES6, ES7)** and follows **web standards**.
🟩 **Better Debugging** → Supports browser dev tools for easy debugging.
🟩 **Reusability** → Can be reused inside **Aura components**.
🟩 **Lightweight** → No extra framework like Aura, making it **efficient**.

---

3. What are the main files in an LWC component?

Answer:
Every LWC component consists of **three main files**:

1. **HTML File (.html)** → Defines the UI structure.
2. **JavaScript File (.js)** → Contains logic and event handling.
3. **Meta Configuration File (.js-meta.xml)** → Controls component visibility and usage.

Example:

```
myComponent.html
myComponent.js
myComponent.js-meta.xml
```

Additionally, we can have:

- **CSS File (.css)** → For styling.
- **SVG File (.svg)** → For custom icons.

---

4. What is the role of the .js-meta.xml file in LWC?

Answer:
The `.js-meta.xml` file defines **where and how** the component can be used in Salesforce.

Example XML file:

```
xml
CopyEdit
<?xml version="1.0" encoding="UTF-8"?>
<LightningComponentBundle xmlns="http://soap.sforce.com/2006/04/metadata"
    fqn="myComponent">
    <apiVersion>58.0</apiVersion>
    <isExposed>true</isExposed>
    <targets>
        <target>lightning__RecordPage</target>
        <target>lightning__AppPage</target>
    </targets>
</LightningComponentBundle>
```

Here,
✓ `isExposed` → Makes the component **available** for use.
✓ `targets` → Defines **where** the component can be used (**Record Page, App Page, etc.**).

---

5. What is Shadow DOM in LWC? How does it improve security?

Answer:
Shadow DOM is a web standard that **encapsulates** a component's structure and styles **within itself**, preventing external interference.

🟠 Benefits in LWC:
✓ **Better Security** → Other components **cannot modify** LWC's styles.

✓ **No CSS Conflicts** → Ensures styles **do not leak** outside the component.
✓ **Encapsulation** → Keeps LWC components **self-contained and reusable**.

Example:

```
<template>
    <p class="custom-style">This style is isolated!</p>
</template>
```

Even if a global CSS file has `.custom-style { color: red; }`, it **won't affect** this LWC component due to **Shadow DOM**.

---

6. What are lifecycle hooks in LWC? Name some commonly used ones.

Answer:
Lifecycle hooks are **methods** that execute at different stages of a component's lifecycle.

Common Lifecycle Hooks:
1. `constructor()` → Runs **when the component is created**.
2. `connectedCallback()` → Runs **when the component is inserted** into the DOM. 3. `renderedCallback()` → Runs **after rendering is done**.
4. `disconnectedCallback()` → Runs **when the component is removed** from the DOM.

Example:

```
connectedCallback() {
    console.log('Component is added to the DOM');
}
```

---

7. How do you pass data from a parent component to a child component in LWC?

Answer:
Use the `@api` decorator to pass data **from parent to child**.

🟩 Parent Component:

```
<c-child-component message="Hello from Parent"></c-child-component>
```

🟩 Child Component:

```
import { api } from 'lwc';
export default class ChildComponent extends LightningElement {
```

```
    @api message;
}
```

---

8. How do you pass data from a child component to a parent component in LWC?

Answer:
Use **Custom Events** to pass data **from child to parent**.

🟩 Child Component (Firing event):

```
javascript
CopyEdit
const event = new CustomEvent('mycustomevent', { detail: 'Hello Parent' });
this.dispatchEvent(event);
```

🟩 Parent Component (Listening to event):

```
html
CopyEdit
<c-child-component  onmycustomevent={handleEvent}></c-child-component>
```

🟩 Parent Component JS:

```
javascript
CopyEdit
handleEvent(event) {
    console.log(event.detail); // Output: "Hello Parent"
}
```

---

9. What are decorators in LWC? Explain @api, @track, and @wire.

Answer:
Decorators **enhance properties/functions** in LWC.

✓ `@api` → Makes a property **public** (used for parent-child communication).
✓ `@track` → (Deprecated) Previously used for **reactive properties**. Now, **JS objects are reactive by default**.
✓ `@wire` → Used for **calling Apex methods or Lightning Data Service**.

Example:

```
javascript
CopyEdit
import { api, wire } from 'lwc';
export default class Example extends LightningElement {
```

```
    @api publicValue;

    @wire(getContacts) contacts;
}
```

---

10. How does LWC handle event handling?

Answer:
LWC **uses standard JavaScript event handling**.

- Use **addEventListener** for built-in events like `click`.
- Use **Custom Events** for **component communication**.

🟩 **Example of handling a button click:**

```
<lightning-button label="Click Me" onclick={handleClick}></lightning-button>
javascript
CopyEdit
handleClick() {
    console.log('Button clicked!');
}
```

Intermediate LWC Interview Questions

11. How do you call an Apex method in LWC using @wire?

Answer:
The `@wire` decorator **automatically calls an Apex method** and **reactively updates** the component when data changes.

🟩 **Apex Class:**

```
apex
CopyEdit
public with sharing class ContactController
    { @AuraEnabled(cacheable=true)
    public static List<Contact> getContacts() {
        return [SELECT Id, Name, Email FROM Contact LIMIT 10];
    }
}
```

🟩 **LWC JavaScript:**

```
import { LightningElement, wire } from 'lwc';
import getContacts from '@salesforce/apex/ContactController.getContacts';

export default class ContactList extends LightningElement
    { @wire(getContacts) contacts;
}
```

🟩 LWC HTML:

```
html
CopyEdit
<template if:true={contacts.data}>
    <template for:each={contacts.data} for:item="contact">
        <p key={contact.Id}>{contact.Name} - {contact.Email}</p>
    </template>
</template>
```

🟠 Key Points:

✓ Use `@AuraEnabled(cacheable=true)` for better performance.
✓ The `@wire` decorator ensures **reactivity** and **automatic updates**.

---

12. How do you call an Apex method imperatively in LWC?

Answer:
Imperative Apex calls **run only when triggered** (like a button click).

🟩 Apex Class:

```
@AuraEnabled
public static List<Contact> getContacts() {
    return [SELECT Id, Name, Email FROM Contact LIMIT 10];
}
```

🟩 LWC JavaScript:

```
import { LightningElement, track } from 'lwc';
import getContacts from '@salesforce/apex/ContactController.getContacts';

export default class ContactList extends LightningElement
    { @track contacts;

    handleLoad()
        { getContacts()
            .then(result =>
                { this.contacts = result;
            })
            .catch(error =>
                { console.error(error);
            });
```

```
    }
}
```

🟩 LWC HTML:

```
<lightning-button label="Load Contacts" onclick={handleLoad}></lightning-button>
<template if:true={contacts}>
    <template for:each={contacts} for:item="contact">
        <p key={contact.Id}>{contact.Name}</p>
    </template>
</template>
```

🟠 Key Points:

✓ Imperative calls are **triggered manually**.
✓ They **don't cache data** like `@wire`.

---

13. What is the difference between @wire and imperative Apex calls?

Feature	`@wire`	Imperative Call
Execution	**Automatically** on load	**Manually** triggered
Caching	**Supports** caching (`cacheable=true`)	**Does not** cache
Error Handling	Uses `error` property	Requires `try...catch`
Best Use Case	Displaying **auto-refreshing** data	**User-triggered actions**, like button clicks

---

14. What are Lightning Data Services (LDS) in LWC? How do they work?

Answer:
LDS allows LWC to **fetch, create, update, and delete records without writing Apex**. It is built on **User Interface API** and ensures **security (CRUD, FLS, Sharing Rules)** automatically.

🟩 Example using LDS (`lightning-record-form`)

```
<lightning-record-form
    record-id="003XXXXXXXXXXXX"
    object-api-name="Contact"
    layout-type="Full">
</lightning-record-form>
```

🔹 **Key Points:**
✓ **No Apex required!**
✓ Automatically **handles security (CRUD, FLS, Sharing Rules)**.
✓ **Optimized** → Uses Salesforce's **caching** for better performance.

---

15. How do you navigate between pages using NavigationMixin in LWC?

Answer:
Use **NavigationMixin** to navigate between **Record Pages, Objects, Custom Tabs, or External Links**.

🟩 Navigate to a Record Page

```
import { LightningElement } from 'lwc';
import { NavigationMixin } from 'lightning/navigation';

export default class NavigateToRecord extends
NavigationMixin(LightningElement) {
    navigateToRecord()
        { this[NavigationMixin.Navigate]({
            type: 'standard__recordPage',
            attributes: {
                recordId: '003XXXXXXXXXXX',
                actionName: 'view'
            }
        });
    }
}
```

🟩 LWC HTML:

```
<lightning-button label="View Record" onclick={navigateToRecord}></lightning-
button>
```

🔹 **Key Points:**
✓ Supports **record pages, list views, objects, and external links**.
✓ Uses **NavigationMixin.Navigate()**.

---

16. How can you style an LWC component? What are different ways to apply CSS?

Answer:
LWC supports three types of styling:

1. **Component CSS (`.css` file) → Scoped to the component**.
2. **Global CSS (`SLDS`)** → Standard **Salesforce Lightning Design System**. 3.**Custom Styles with `:host` and `:host-context`** → Advanced styling.

🟩 Example: Scoped CSS (`.css` file)

```
p {
    color: blue;
    font-size: 16px;
}
```

🟩 Example: Global SLDS Class (`HTML`)

```
<p class="slds-text-color_success">Green text using SLDS</p>
```

🟠 Key Points:

✓ **Component styles** do not leak outside.
✓ Use `:host` for **custom component styles**.

---

17 How do you share data between two sibling components in LWC?

Answer:
Use **Lightning Message Service (LMS)** for sibling communication.

🟩 Message Channel (`messageChannel.message-meta.xml`)

```
<?xml version="1.0" encoding="UTF-8"?>
<LightningMessageChannel xmlns="http://soap.sforce.com/2006/04/metadata"
    fqn="MyMessageChannel">
</LightningMessageChannel>
```

🟩 Publisher Component (Firing the Event)

```
import { publish, MessageContext } from 'lightning/messageService';
import MY_MESSAGE_CHANNEL from '@salesforce/messageChannel/MyMessageChannel';

export default class Publisher extends LightningElement
    { @wire(MessageContext) messageContext;

    sendMessage() {
        const message = { data: 'Hello from Publisher' };
        publish(this.messageContext, MY_MESSAGE_CHANNEL, message);
    }
}
```

🟩 Subscriber Component (Receiving the Event)

```
import { subscribe, MessageContext } from 'lightning/messageService';
import MY_MESSAGE_CHANNEL from '@salesforce/messageChannel/MyMessageChannel';

export default class Subscriber extends LightningElement
    { @wire(MessageContext) messageContext;

    connectedCallback() {
        subscribe(this.messageContext, MY_MESSAGE_CHANNEL, (message) =>
            { console.log(message.data);
        });
    }
}
```

🟠 **Key Points:**
✓ LMS is **best for sibling components**.
✓ Works across **Lightning Apps, Aura, and LWC**.

I'll continue with **Advanced LWC questions** in the next response.

You said:

yes

ChatGPT said:

Advanced LWC Interview Questions

21. How does LWC work with Salesforce security (CRUD, FLS, Sharing Rules)?

Answer:
LWC **does not** automatically enforce security. You must handle:
1. **CRUD (Create, Read, Update, Delete)** – Check if a user has permission on the object. 2. **FLS (Field-Level Security)** – Ensure users can access specific fields.
3. **Sharing Rules** – Control record visibility based on org-wide defaults & sharing rules.

🟩 **Apex (Enforcing Security with Security.stripInaccessible)**

```
@AuraEnabled
public static List<Contact> getContacts() {
    List<Contact> contacts = [SELECT Id, Name, Email FROM Contact];
    return (List<Contact>) Security.stripInaccessible(AccessType.READABLE,
contacts);
}
```

🟩 **LWC (Calling Secure Apex)**

```
@wire(getContacts) contacts;
```

🟠 **Key Points:**
✓ Use `Security.stripInaccessible()` to prevent unauthorized field access.
✓ Use `with sharing` in Apex to enforce sharing rules.

22.What are the different types of events in LWC?

Answer:
LWC supports three types of events:

Event Type	Description	Example
Standard Events	Built-in browser events	`click`, `keyup`, `change`
Custom Events	Developer-defined events	`new CustomEvent('myEvent')`
Lightning Events	Cross-component events	LMS, `lightning/navigation`

🟩 **Example: Dispatching a Custom Event (Child to Parent)**

```
const event = new CustomEvent('myevent', { detail: 'Hello Parent' });
this.dispatchEvent(event);
```

🟠 **Key Points:**
✓ Use `CustomEvent` for parent-child communication.
✓ Use **Lightning Message Service (LMS)** for sibling components.

---

23. How do you use Platform Events in LWC?

Answer:
Platform Events allow real-time communication across systems.

🟩 **1. Create a Platform Event (`MyEvent__e`)**
🟩 **2. Subscribe in LWC using CometD API**

```
import { subscribe, onError } from 'lightning/empApi';

connectedCallback() {
    subscribe('/event/MyEvent__e', -1, (event) =>
        { console.log('Received event: ', event);
    });
}
```

🔹 **Key Points:**
✓ **Used for real-time updates** (e.g., order tracking, notifications).
✓ Uses **CometD API** for streaming.

---

24. How do you integrate an external system with LWC using REST API?

Answer:
LWC **cannot** call external APIs directly due to **CORS restrictions**. Instead, use **Apex Callouts**.

🟩 Apex Callout to External API

```
@AuraEnabled
public static String fetchData()
    { Http http = new Http();
    HttpRequest request = new HttpRequest();
    request.setEndpoint('https://api.example.com/data');
    request.setMethod('GET');
    HttpResponse response = http.send(request);
    return response.getBody();
}
```

🟩 LWC Calling Apex

```
import fetchData from '@salesforce/apex/ExternalAPIHandler.fetchData';

fetchData().then(result =>
    { console.log(result);
});
```

🔹 **Key Points:**
✓ **LWC → Apex → External API** to bypass CORS.
✓ **Enable Named Credentials** for authentication.

---

25 What is Salesforce Connect, and how can LWC interact with external data?

Answer:
Salesforce Connect allows **real-time access** to external data **without storing it**.
✓ Uses **External Objects** (`Account_External__x`).
✓ Fetches data via **OData, REST, or SOAP**.

🟩 LWC Querying External Object

```
import { getRecord } from 'lightning/uiRecordApi';
import ACCOUNT_EXTERNAL_OBJECT from '@salesforce/schema/Account_External__x';

@wire(getRecord, { recordId: '001XXXXXXXXXXX', fields:
[ACCOUNT_EXTERNAL_OBJECT.Name] })
account;
```

🟠 **Key Points:**
✓ **No data storage needed** in Salesforce.
✓ Ideal for **large datasets** from external systems.

---

26. How do you create a custom reusable LWC component?

Answer:
Create a **Generic Component** that accepts **dynamic properties**.

🟩 **Generic Card Component (Reusable)**
HTML:

```
<template>
    <lightning-card title={title}>
        <slot></slot> <!-- Allows dynamic content -->
    </lightning-card>
</template>
```

JS:

```
import { LightningElement, api } from 'lwc';
export default class ReusableCard extends LightningElement
    { @api title;
}
```

🟠 **Key Points:**
✓ Use **@api** to make properties **configurable**.
✓ Use **slots** for **dynamic content** inside the component.

---

27. What is the difference between Composition and Inheritance in LWC?

Answer:

Feature	Composition	Inheritance
Definition	**Combining components** inside each other	**Extending** from another class
Example	`<c-child></c-child>` inside Parent	`class Child extends Parent {}`
Reusability	**More reusable**	Less flexible
Preferred?	🟩 **Best Practice**	➕ Avoid in LWC

🟠 **Key Points:**
✓ **Use Composition** (`<c-child>`) instead of inheritance.
✓ **LWC does not support multiple inheritance**.

---

28. How does LWC handle state management?

Answer:
LWC uses **JavaScript properties** for state management.
✓ Use **@track** (older method, now automatic).
✓ Use **@api** for parent-child state sharing.
✓ Use **Lightning Message Service (LMS)** for app-wide state sharing.

🟩 **Example: Managing State in LWC**

```
import { LightningElement } from 'lwc';
export default class StateExample extends LightningElement
    { count = 0;

    increment() {
        this.count++;
    }
}
```

🟠 **Key Points:**
✓ LWC state is managed **reactively**.
✓ Use **LMS** for global state sharing.

29.What is the difference between `renderedCallback()` and `connectedCallback()`?

Answer:

Method	When It Runs	Use Case
`connectedCallback()`	Runs **once** when component is inserted into DOM	Fetching data, setting up listeners
`renderedCallback()`	Runs **after every render**	DOM manipulation, checking UI updates

🟩 Example:

```
connectedCallback() {
    console.log('Component inserted in DOM');
}

renderedCallback()
    { console.log('Component rendered');
}
```

🟠 Key Points:

✓ **Use `connectedCallback()` for initial data loading.**
✓ **Use `renderedCallback()` carefully** to avoid infinite loops.

---

30. How do you debug LWC components effectively?

Answer:
Use `console.log()` – Print values to the browser console.
Use Chrome DevTools – Inspect elements and view errors.
Enable Debug Mode in Salesforce – Helps in detailed error tracking.
Use `try...catch` blocks – Handle exceptions in Apex & JS.
Use `@wire` debug logs – Log wired method data.

🟩 Example:

```
try {
    console.log('Data:', this.data);
} catch (error)
    { console.error('Error:', error);
}
```

🟠 Key Points:

✓ Use **Chrome DevTools & console.log()** for debugging.
✓ Enable **Debug Mode** in **Setup → Debug Mode**.

3

What is Salesforce Integration?

Salesforce Integration means **connecting Salesforce with other applications, databases, or services** to exchange data and functionalities. For example:

- Connecting Salesforce with **WhatsApp** to send notifications.
- Integrating Salesforce with **Google Maps** to show locations.
- Syncing Salesforce with **ERP systems** like SAP.

Why is it important?

- Avoids **manual data entry** (reduces errors).
- Makes data **available in real-time**.
- Connects different business systems.

Types of Salesforce Integration

There are **4 main types** of integrations in Salesforce:

1. **Data Integration** – Syncs data between Salesforce and other systems (e.g., CRM, ERP).
   - Example: Integrating Salesforce with a database like **MySQL**.
2. **Process Integration** – Connects business processes across systems.
   - Example: When an order is placed in an **e-commerce site**, it is automatically created as an **Opportunity** in Salesforce.
3. **User Interface (UI) Integration** – Merges multiple applications into a single interface.
   - Example: Embedding **Google Maps** inside Salesforce.
4. **Security Integration** – Ensures authentication and access control.
   - Example: Using **OAuth** for single sign-on (SSO).

Ways to Integrate Salesforce

1. Using APIs (Application Programming Interface)

APIs allow Salesforce to communicate with other systems. There are two main types:

REST API – Lightweight & best for web/mobile apps.

- Example: Fetching **contact details** from Salesforce.
- Used for: **CRUD operations (Create, Read, Update, Delete)**.

SOAP API – More structured & used for complex integrations.

- Example: **Syncing customer records** between systems.

Other APIs:

- **Bulk API** – Used for handling large volumes of data.
- **Streaming API** – Used for real-time data updates.
- **Metadata API** – Used to deploy customizations (like objects & fields).

2. Using Middleware (Integration Tools)

Middleware acts as a bridge between Salesforce and other applications. Examples:

- **Mulesoft** (Salesforce's official integration platform).
- **Zapier** (No-code automation tool).
- **Boomi** (Cloud-based integration).

Middleware **reduces coding effort** and makes integrations easy.

3. Using Salesforce Connect (External Objects)

- If your data is stored in **another system (like SAP, SQL Server)** but you want to access it in Salesforce **without storing it**, use **Salesforce Connect**.
- It uses **OData (Open Data Protocol)** to fetch data in real time.

4. Using Platform Events (Event-Driven Integration)

- When **something happens** in one system, it triggers an event in another system.
- Example: If a **new lead** is created in Salesforce, send an SMS via **Twilio**.

Salesforce Authentication Methods (Security in Integration)

To protect data, Salesforce uses authentication methods:

1. **OAuth 2.0** – Secure & used for web apps.
2. **JWT (JSON Web Token)** – Used for **server-to-server** authentication.
3. **SAML (Security Assertion Markup Language)** – Used for **Single Sign-On (SSO)**.

🟩 Real-World Example: Salesforce Integration with WhatsApp

1. **Step 1:** Use **Twilio API** to connect Salesforce and WhatsApp.
2. **Step 2:** Configure an **Apex Trigger** to send messages when a new lead is created.
3. **Step 3:** Use **OAuth** for secure communication.

End Result: Whenever a lead is created, an **automated WhatsApp message** is sent using Twilio.

Summary

Salesforce Integration helps in **connecting Salesforce with other apps** to automate processes and sync data.

- **REST API & SOAP API** → Used for API-based integration.
- **Middleware (Mulesoft, Zapier, Boomi)** → Simplifies integration.
- **Salesforce Connect** → Access external data **without storing it**.
- **Platform Events** → Enable **real-time data sync**.
- **Authentication (OAuth, JWT, SAML)** → Secure connections.

1. REST API Integration in Salesforce (Step-by-Step Guide)

We'll create a simple **REST API integration** where we **fetch Account details** from Salesforce using **Postman**.

🔹 Step 1: Enable API Access in Salesforce

Before using APIs, make sure your Salesforce org has API access:

1. **Go to Setup → Profiles**
2. Open the **System Administrator Profile**
3. Scroll down to **Administrative Permissions**
4. Make sure **API Enabled** is **checked**

🔹 Step 2: Generate API Credentials (OAuth 2.0)

1. **Go to Setup → App Manager**
2. Click **New Connected App**
3. Fill in details:
    - Name: `My REST API App`
    - API Name: `My_REST_API_App`
    - Contact Email: Your email
4. Enable **OAuth Settings** ✅
5. In **Callback URL**, enter:

    ```
    https://login.salesforce.com/services/oauth2/callback
    ```

6. Select **OAuth Scopes** → `Full Access (full)`
7. Click **Save & Continue**
8. Copy **Consumer Key** and **Consumer Secret**

🔹 Step 3: Get Access Token (Postman)

1. Open **Postman**
2. Use the following URL (Replace `your_client_id` and `your_client_secret`):

    ```
    https://login.salesforce.com/services/oauth2/token
    ```

3. Select **POST** method
4. Go to **Body** → **x-www-form-urlencoded**
5. Enter:
    - `grant_type`: `password`
    - `client_id`: `your_client_id`
    - `client_secret`: `your_client_secret`
    - `username`: Your Salesforce username
    - `password`: Your Salesforce password + security token
6. Click **Send**
7. Copy the **Access Token** from the response

🔹 Step 4: Call the REST API (Fetch Accounts)

1. In **Postman**, create a new request
2. Set **GET** method
3. Use this URL to fetch Accounts:

```
https://yourInstance.salesforce.com/services/data/v58.0/sobjects/Account
```

(Replace `yourInstance` with your Salesforce instance URL)

4. Go to **Headers** → Add:
    - `Authorization`: `Bearer your_access_token`

```
Content-Type application/json
```

5. Click **Send**
6. You will get the list of **Accounts in JSON format**

2. REST API (Advanced) – Insert, Update, Delete & Apex Callouts

Now that we have fetched records using **GET**, let's perform **CRUD (Create, Read, Update, Delete) operations** using REST API.

🔹 Step 1: Create a New Record (POST)

Scenario

We will create a **new Account record** in Salesforce using Postman.

Steps

1. **Open Postman**
2. Select **POST** method
3. Use this URL:

```
https://yourInstance.salesforce.com/services/data/v58.0/sobjects/Account
```

4. Go to **Headers** and add:
   - `Authorization: Bearer your_access_token`
   - `Content-Type: application/json`
5. Go to **Body → raw** and enter:

```
{
   "Name": "TechCorp Solutions",
   "Phone": "9876543210",
   "Industry": "Technology"
}
```

6. Click **Send**

7. Response will show **"id": "001XXXXXXXXXXXXXXX"** , meaning the record is created.

🔹 Step 2: Update a Record (PATCH)

Scenario

Now, let's update the **Phone number** of the Account we just created.

Steps

1. Select **PATCH** method
2. Use this URL (Replace `001XXXXXXXXXXXXXXX` with the record ID from the previous step):

   ```
   https://yourInstance.salesforce.com/services/data/v58.0/sobjects/Account/001XXXXXXXXXXXXXXX
   ```

3. Go to **Body → raw** and enter:

   ```
   {
      "Phone": "1234567890"
   }
   ```

4. Click **Send**
5. Response will show **"204 No Content"**, meaning the update is successful.

🔹 Step 3: Delete a Record (DELETE)

Scenario

Now, let's delete the Account we created.

Steps

1. Select **DELETE** method
2. Use this URL:

   ```
   https://yourInstance.salesforce.com/services/data/v58.0/sobjects/Account/001XXXXXXXXXXXXXXX
   ```

3. Click **Send**

4. Response will be **"204 No Content"**, meaning the record is deleted.

🔷 Step 4: Call an External API from Salesforce (Apex Callout)

Scenario

Salesforce will **call an external API** to fetch weather details from OpenWeather API.

Steps

1. Create a Remote Site Setting

Before making an API call, Salesforce needs to allow external sites.

1. **Go to Setup → Remote Site Settings**
2. Click **New Remote Site**
3. Enter:
   - Remote Site Name: `WeatherAPI`
   - Remote Site URL: `https://api.openweathermap.org`
4. Click **Save**

2. Write Apex Class for API Callout

1. **Go to Developer Console → File → New → Apex Class**
2. Create a new class **WeatherService** and paste this:

```
public class WeatherService {
    public static String getWeather(String city)
        { String url =
'https://api.openweathermap.org/data/2.5/weather?q=' + city +
'&appid=your_api_key';

        Http http = new Http();
        HttpRequest request = new HttpRequest();
        request.setEndpoint(url);
        request.setMethod('GET');

        HttpResponse response = http.send(request);
        if(response.getStatusCode() == 200) {
            return response.getBody();
        } else {
            return 'Error: ' + response.getStatusCode();
        }
    }
}
```

3. Replace `your_api_key` with your OpenWeather API Key.
4. Click **Save**

3. Test in Anonymous Apex

1. Open **Developer Console**
2. Click **Debug → Open Execute Anonymous Window**
3. Run:

```
apex

System.debug(WeatherService.getWeather('Indore'));
```

4. The debug logs will show weather data for Indore!

2. SOAP API Integration

SOAP API is **similar to REST API** but uses **XML format** instead of JSON.
It is mainly used when an **external system only supports SOAP**.

Steps to Integrate SOAP API

1. **Enable API in Salesforce** (Setup → Profiles → System Administrator → API Enabled).

2. **Get WSDL (Web Service Definition Language)** file from Salesforce.
3. **Use it in an external system** (e.g., Java, .NET) to communicate with Salesforce.

Basic SOAP API Example

- **Fetching an Account from Salesforce (SOAP Request in XML format)**

```
xml

<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
                  xmlns="urn:partner.soap.sforce.com">
   <soapenv:Header>
```

```
      <SessionHeader>
         <sessionId>your_access_token</sessionId>
      </SessionHeader>
   </soapenv:Header>
   <soapenv:Body>
      <query>
         <queryString>SELECT Name FROM Account WHERE Id =
'001XXXXXXXXXXXXXX'</queryString>
      </query>
   </soapenv:Body>
</soapenv:Envelope>
```

- This query will return **Account details** in XML format.

Do you want me to show how to use **SOAP API in Apex**?

---

3. Salesforce to External System Integration (Callouts)

This is when **Salesforce sends data** to another system.

Steps to Call an External API from Salesforce

1. **Enable Remote Site Setting** (Setup → Remote Site Settings → Add API URL).
2. **Write Apex Class to call API** using `HttpRequest`.

Basic Code Example (Apex Callout)

Calling an external weather API from Salesforce

```
apex

public class WeatherService {
    public static String getWeather(String city) {
        String url = 'https://api.openweathermap.org/data/2.5/weather?q=' +
city + '&appid=your_api_key';

        Http http = new Http();
        HttpRequest request = new HttpRequest();
        request.setEndpoint(url);
        request.setMethod('GET');

        HttpResponse response = http.send(request);
        return response.getBody();
    }
}
```

- This will fetch **weather details** for the given city.

🔵 4. External System to Salesforce Integration (Webhooks & API Calls)

This is when an **external system sends data** to Salesforce.

🔹 Steps to Receive Data in Salesforce

1. **Create a REST API in Salesforce** (Apex REST Class).
2. **External system sends data** to that API using `POST`.

🔸 Basic Code Example (Apex REST API)

🔷 External system sending Account data to Salesforce

```
apex

@RestResource(urlMapping='/createAccount/')
global class AccountAPI {
    @HttpPost
    global static String createAccount(String name)
        { Account acc = new Account(Name = name);
        insert acc;
        return 'Account Created with ID: ' + acc.Id;
    }
}
```

- The external system can **send a POST request** with the Account name, and it will create a new Account in Salesforce.

🟤 5. Salesforce Connect (Access External Data Without Storing)

Salesforce Connect allows **reading external data in real-time** without storing it in Salesforce.

🔹 Steps to Use Salesforce Connect

1. **Create an External Data Source** (Setup → External Data Sources).
2. **Define External Objects** (like normal objects, but data is stored externally).
3. **Use OData (Open Data Protocol) to fetch data**.

🔵 6. Middleware Integration (Mulesoft, Zapier, Boomi, etc.)

Middlewares help **connect multiple systems** without direct API coding.

Example Middleware Use Cases

- **Mulesoft**: Connects Salesforce with SAP, databases, etc.
- **Zapier**: Connects Salesforce with Gmail, Slack, Google Sheets.
- **Boomi**: Automates data flow between different systems.

Example (Zapier)

- **Trigger**: When a new lead is added in Salesforce.
- **Action**: Send an email via Gmail.
- **No coding required!**

🟢 7. Platform Events (Real-time Event-Based Integration)

Platform Events allow **real-time data transfer** between Salesforce and external systems.

Steps to Use Platform Events

1. **Create a Platform Event** (Setup → Platform Events → New).
2. **Publish the Event** (Apex Code).
3. **External System subscribes to the event**.

Basic Code Example (Publishing an Event)

```
PlatformEvent__e event = new PlatformEvent__e(Message__c = 'New Order
Received');
EventBus.publish(event);
```

- **External systems listen** for this event and take action.

🟡 8. Authentication & Security (OAuth, JWT, SAML)

This controls **how users log in and access Salesforce securely**.

🔹 OAuth 2.0 (Common for API Authentication)

1. External system requests an **access token** using **client ID & secret**.
2. Salesforce returns a **token**.
3. The token is used for API requests.

Basic OAuth Token Request (Postman)

```
diff

POST https://login.salesforce.com/services/oauth2/token
Body:
- grant_type = password
- client_id = your_client_id
- client_secret = your_client_secret
- username = your_salesforce_username
- password = your_password+security_token
```

- Response: `{ "access_token": "abc123xyz" }`
- Use this token for API requests.

Summary

Integration Type	Purpose
REST API	Fetch/Send data (JSON)
SOAP API	Fetch/Send data (XML)
Salesforce Callout	Salesforce calls external API
External Webhook	External system calls Salesforce
Salesforce Connect	Access external data without storing
Middleware	No-code automation between systems
Platform Events	Real-time event-based integration
OAuth & Security	Secure authentication