

By Smriti Sharan ([sfdcamlified](#))

## Salesforce Developer Interview Questions and Answers Set 1

*Hello, my Name is Smriti Sharan. I am [avid blogger](#) and [youtuber](#). Follow my Blog and youtube to learn various aspect of Salesforce.*

<https://t.me/sfdcamlified> feel free to join telegram group.

*I am sharing all the questions I have faced in interviews and detail answers for it to make learning easy.*

### 1.What is lifecycle hook? (very important)

Lifecycle hooks are used to handle the lifecycle of components. Here is list of all method

- constructor()
- connectedCallback()
- rendered()
- renderedCallback()
- disconnectedCallback()
- errorCallback()

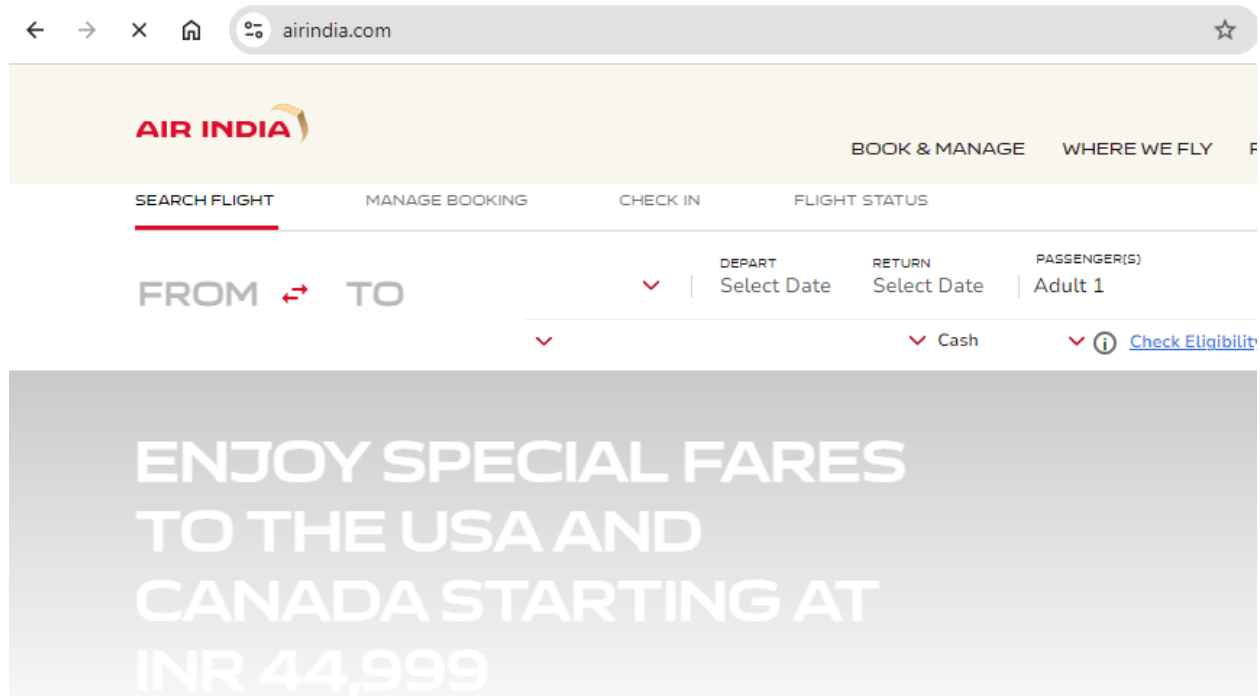
In Aura framework init(), render(), rerender(), afterRender(), unrender() methods were used to handle the lifecycle of components.

It's very important in terms of interview so let's understand it clearly. Let's say you want to book a Flight Ticket. Now we will understand booking of flight in LWC lifecycle context.

By Smriti Sharan ([sfdcamlified](#))

## Constructor is called (Parent):

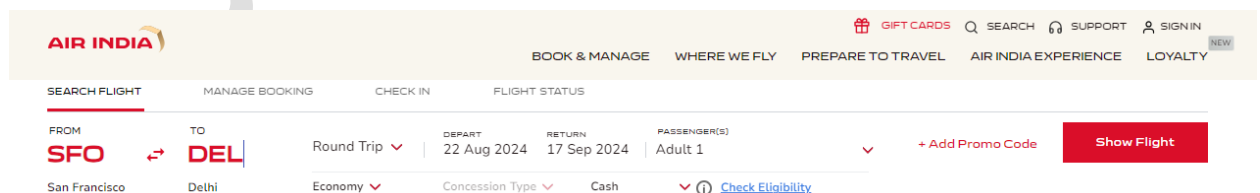
You decide to **book a flight and open the Air India website**. The airline website's main page starts loading and initializing resources. This is similar to the **constructor** method of the parent component being called.



The screenshot shows the Air India website homepage. The browser address bar displays 'airindia.com'. The website header includes the Air India logo and navigation links: 'BOOK & MANAGE', 'WHERE WE FLY', and 'F'. Below the header, there are four main sections: 'SEARCH FLIGHT', 'MANAGE BOOKING', 'CHECK IN', and 'FLIGHT STATUS'. The 'SEARCH FLIGHT' section is active and contains a search form with fields for 'FROM' (San Francisco), 'TO' (Delhi), 'DEPART' (22 Aug 2024), 'RETURN' (17 Sep 2024), and 'PASSENGER(S)' (Adult 1). There are also dropdown menus for 'Round Trip', 'Economy', and 'Cash'. A 'Show Flight' button is visible. A large banner below the search form reads 'ENJOY SPECIAL FARES TO THE USA AND CANADA STARTING AT INR 44,999'.

## Public property is set on the parent:

The website asks you to select your departure city. This is like setting a public property on the parent component. The website (parent component) needs this information to proceed with displaying available flights.



The screenshot shows the Air India website homepage with the flight search results. The search form is filled with 'FROM' (SFO - San Francisco), 'TO' (DEL - Delhi), 'DEPART' (22 Aug 2024), 'RETURN' (17 Sep 2024), and 'PASSENGER(S)' (Adult 1). The 'Round Trip' dropdown is selected. The 'Economy' dropdown is selected. The 'Cash' dropdown is selected. A 'Show Flight' button is visible. The website header includes the Air India logo and navigation links: 'BOOK & MANAGE', 'WHERE WE FLY', 'PREPARE TO TRAVEL', 'AIR INDIA EXPERIENCE', and 'LOYALTY'. There are also links for 'GIFT CARDS', 'SEARCH', 'SUPPORT', and 'SIGN IN'.

## Parent is inserted into the DOM:

## By Smriti Sharan ([sfdcamplified](#))

After selecting your departure city, the website shows available destinations and flights from that city. The parent component (main page) is now part of the webpage and you can interact with it. It's inserted into the DOM.

1 FLIGHTS

2 JOURNEY DETAILS

3 REVIEW & PAYMENT

THU, 22 AUG 24  
SFO

Departure

THU, 22 AUG 24  
DEL

TUE, 17 SEPT 24  
DEL

Return

TUE, 17 SEPT 24  
SFO

Modify Booking

<

SUNDAY, 18 AUG  
USD 673.20

MONDAY, 19 AUG  
USD 490.20

TUESDAY, 20 AUG  
USD 490.20

WEDNESDAY, 21 AUG  
USD 673.20

THURSDAY, 22 AUG  
USD 673.20

FRIDAY, 23 AUG  
USD 471.20

SATURDAY, 24 AUG  
USD 673.20

SUNDAY, 25 AUG  
USD 673.20

MONDAY, 26 AUG  
USD 471.20

>

Enjoy 10X Membership Rewards® points on booking flights.  
No enrolment | No minimum spend | No promo code | Earn unlimited\* points  
\*\*T&C apply

Upcoming:

Get up to ₹2000 off on flight bookings with **BOBCARD**.  
Offer valid every Friday until 27<sup>th</sup> September\*

Upcoming:

Get up to ₹2000 off on flight bookings with **ICICI Bank Credit & Debit cards**.  
Offer valid every Saturday & Sunday until 31<sup>st</sup> August\*

SELECT FLIGHT

14 Flights found for your trip

☐ Nonstop

Duration ▾

Filter

### Connected callback is called on the parent:

As you browse through available flights, the website loads additional data such as prices, flight times, and airline logos. The connected callback of the parent component is called.

THU, 22 AUG 24 10:30 SFO	15H 45Min Nonstop	FRI, 23 AUG 24 14:45 DEL	ECONOMY USD 471.20 ▾	PREMIUM ECONOMY USD 870.20 ▾	BUSINESS USD 2,368.20 ▾
AI 184					
THU, 22 AUG 24 22:00 SFO	15H 45Min Nonstop	SAT, 24 AUG 24 02:15 DEL	ECONOMY USD 471.20 ▾	PREMIUM ECONOMY NOT AVAILABLE	BUSINESS USD 2,368.20 ▾
AI 180					
THU, 22 AUG 24 20:30 SFO	23H 5Min 1 Stop - ROM - 2H 20Min	SAT, 24 AUG 24 08:05 DEL	ECONOMY	PREMIUM ECONOMY 8 seats left	BUSINESS 4 seats left

By Smriti Sharan ([sfdcamplified](#))

### Parent is rendered:

You see the list of available flights, complete with details and options to select. The parent component is fully rendered and visible to you.

### Child Component Lifecycle

#### Constructor is called on the child:

You select a flight, and a new page or section loads with passenger details and seating options.

#### Public property of child is set:

The page already knows which flight you selected and displays the correct details such as flight number and time.

#### Child is inserted into the DOM:

You're now viewing the passenger details and seating options, and you start entering your information.



#### Connected callback is called on the child:

As you enter your details, the page may update seat availability in realtime or validate your input.

#### Child is rendered:

## By Smriti Sharan ([sfdcemplified](#))

You see your entered details and selected seat, and the information is ready to be submitted. The child component is fully rendered with all the data you've entered.

### Rendered callback is called (Child & Parent):

You confirm your booking, and the page updates to show your booking summary. The rendered callback is called first on the child component

## 2. Why we use lifecycle hook?

### Constructor:

Purpose: To set initial state, perform simple initializations.

When Called: Before the component is inserted into the DOM.

### ConnectedCallback:

Purpose: To perform tasks that require the component to be in the DOM, such as data fetching or event listener registration.

When Called: When the component is inserted into the DOM.

### DisconnectedCallback:

Purpose: To perform cleanup tasks, such as removing event listeners or cancelling subscriptions.

When Called: When the component is removed from the DOM.

### RenderedCallback:

Purpose: To perform post-render tasks, such as interacting with the DOM or third-party libraries.

When Called: After every render of the component.

By Smriti Sharan ([sfdcamlified](#))

## Difference between render(), renderedCallback and errorCallback()?

### render()

Render is mainly used to **conditionally render a template**. It defines the business logic to decide which template (HTML file) to use.

### renderedCallback()

renderedCallback() is **unique to Lightning Web Component**. DOM is created after the connectedCallback and before the renderedCallback. renderedCallback() method is called after render() method.

This method is invoked when component is completely rendered, Basically when **all the elements on the component are inserted**. This method called is after every render of the component. This hook flows **from child to parent**.

**Note:** **As this method called after every render of the component, so we need to be careful**, if we want to perform some operation is specific conditions like performing one time operation, use a private boolean property like **hasRendered** to track whether renderedCallback() has been executed.

### errorCallback()

It Captures errors that may happen in all the child components lifecycle hooks. This method is **unique to LWC**.

It has two parameters error and stack. The error argument is a JavaScript native error object, and the stack argument is a string.

If there is error in child then parent handles the error but if there is error in parent then it is shown in the UI.

By Smriti Sharan ([sfdcamlified](#))

## connectedCallback() vs renderedCallback()?

connectedcallback()

- The connected callback is executed when the component is inserted into DOM. this is similar to init handler in aura.
- The connectedCallback() hook can fire more than once. For example, if you remove an element and then insert it into another position, such as when you reorder a list, the hook fires several times.
- The execution flow of connectedCallback is parent to child. So you cannot access child elements in the connectedCallback, because they are not inserted yet.
- Don't update a wire adapter configuration object property in renderedCallback()
- Don't update a public property or field in renderedCallback()

renderedcallback()

This gets called when the component is rendered.

## 3.What are aggregatefucntions ?

aggregate functions are used to perform calculations on a set of records and return a single result.

**COUNT():**

SELECT COUNT() FROM Account WHERE Industry = 'Technology'

**SUM():**

SELECT SUM(Field) FROM ObjectName WHERE condition

Returns the sum of a numeric field in the specified records.

**AVG():**

## By Smriti Sharan ([sfdcamlified](#))

SELECT AVG(Field) FROM ObjectName WHERE condition

Returns the average value of a numeric field in the specified records.

**MIN():**

SELECT MIN(Field) FROM ObjectName WHERE condition

**MAX():**

SELECT MAX(Field) FROM ObjectName WHERE condition

Returns the largest value of a field in the specified records.

### Example

// Query to get the total amount of each Account

List<AggregateResult> results = [

    SELECT AccountId, SUM(Amount) totalAmount

    FROM Opportunity

    GROUP BY AccountId

];

### 4.What is dynamic SOQL?

Dynamic SOQL refers to the creation of a SOQL string at runtime with Apex code. For example, you can create a search based on input from an end user, or update records with varying field names.

For Example :

```
string myTestString = 'TestName' ;
```

```
List<sObject> sl = Database.query(SELECT Id,Name FROM  
myCustomObject__c WHERE Name=: myTestString);
```



## By Smriti Sharan ([sfdcamlified](#))

Dynamic SOQL can be invoked by `Database.query(query_string);` where `query_string` is the query generated at runtime.

### What is FOR UPDATE clause in soql?

For Update clause will lock the records from getting updated from other transactions until the current transaction is completed.

syntax: `[select id, name from account for update]`

### 5.What is difference between where and having?

The WHERE clause filters records in a SOQL query that has no aggregate function.

The HAVING clause filters the results after data is aggregated by an aggregate function.

```
SELECT MAX(Status__c), Broker__r.Name FROM Property__c GROUP  
BY Broker__r.Name HAVING MAX(Status__c) = 'Closed'
```

### 6.Get the number of accounts owned by each sales rep?

```
SELECT  
  COUNT(Id) numAccounts,  
  OwnerId,  
  Owner.Name  
FROM Account  
GROUP BY OwnerId, Owner.Name
```

### 7.Explain difference between wire and imperative?

Wire Decorator is used to declaratively call an Apex method or a Lightning Data Service (LDS) function and bind the results to a property or function in your LWC.

By Smriti Sharan ([sfdcamplified](#))

Example: component that displays a list of contacts as soon as it loads.

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

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

    // The contacts property now automatically holds the data fetched from
    // the getContacts method.
}
```

## Imperative Calls

Imperative calls involve explicitly calling an Apex method within JavaScript logic, in response to an event or user action.

## 8.How many callouts you can make from Batch Apex?

Total number of callouts made in a single batch execution does not exceed the limit of 100 callouts per transaction.

We can set the number of records processed in each transaction by specifying the batch size when we call the Database.executeBatch method.

Example:

```
Database.executeBatch(batch, 50);
```

By Smriti Sharan ([sfdcamlified](#))

## 9.What is Governor Limits for Callouts in Queueable Apex?

Maximum number of HTTP callouts per transaction: 100

Maximum cumulative timeout for callouts per transaction: 120 seconds

## 10. Difference between Future and Queueable?

Future Method	Queueable Apex
<ol style="list-style-type: none"><li>1. Future will never use to work on SObjects or object types.</li><li>2. When using the future method we cannot monitor the jobs which are in process.</li><li>3. The future method cannot be called inside the future or batch class.</li><li>4. The future method will never be queued.</li></ol>	<ol style="list-style-type: none"><li>1. Queueable Jobs can contain the member variable as SObjects or custom Apex Types.</li><li>2. When using queueable jobs it will make the AsyncApexJob which we can monitor like Scheduled jobs.</li><li>3. Queueable Apex can be called from the future and batch class.</li><li>4. Using Queueable Apex will chain up to queueable jobs and in Developer Edition it is only 5 Jobs.</li></ol>

Use Future Methods for simple, asynchronous processing that doesn't require chaining or complex data types, and when making asynchronous callouts in a straightforward manner.

Use Queueable Apex when you need to chain jobs, handle complex data types, or require enhanced monitoring and tracking of asynchronous operations.

## 11.How to pass parameters in Queueable Apex?

In Queueable Apex, you can pass parameters to the Queueable job by using the constructor of the Queueable class. Constructor initializes the variable with the value passed when the class is instantiated.

By Smriti Sharan ([sfdcamlified](#))

```
public class DemoQueueable implements Queueable {  
    private String message;  
  
    // Constructor to initialize the parameter  
    public DemoQueueable (String message) {  
        this.message = message;  
    }  
  
    // Execute method to define the job logic  
    public void execute(QueueableContext context) {  
        System.debug('Queueable job executed with message: ' + message);  
    }  
}
```

## 12.How to pass subject in future method?

We cannot directly pass sObject types to future methods. Future methods are limited to only accepting primitive data types, collections of primitive data types, and strings. However, we can work around this limitation by passing the sObject's ID or serializing the sObject to a JSON string and then deserializing it within the future method.

By Smriti Sharan ([sfdcamlified](#))

### Workaround1: Pass the Id of an Account to the future method:

```
public class FutureExample {  
    @future  
    public static void updateAccountName(Id accountId, String newName) {  
        Account acc = [SELECT Id, Name FROM Account WHERE Id =  
:accountId LIMIT 1];  
        if (acc != null) {  
            acc.Name = newName;  
            update acc;  
        }  
    }  
}
```

### Workaround 2: Serializing the sObject to JSON

Serialize the sObject to a JSON string, pass the JSON string to the future method, and then deserialize it back to an sObject.

```
public class FutureExample {  
    @future  
    public static void processAccount(String accountJson) {  
        Account acc = (Account)JSON.deserialize(accountJson,  
Account.class);  
    }  
}
```

## By Smriti Sharan ([sfdcamlified](#))

```
acc.Name = 'Updated Name';  
update acc;  
}  
}
```

### 13.How to schedule a queueable job?

Scheduling a Queueable job involves using the `System.enqueueJob` method to queue the job and the `System.schedule` method to schedule a job to run at a specific time.

```
public class MySchedulableJob implements Schedulable {  
    public void execute(SchedulableContext sc) {  
        String message = 'Hello scheduled Queueable job';  
        System.enqueueJob(new MyQueueableJob(message));  
    }  
}
```

### 14.How to schedule the Job using apex?

Use the `System.schedule` method to schedule the `Schedulable` class to run at a specific time.

```
String jobName = 'ScheduledQueueableJob';  
String cronExpression = '0 0 12 * * ?'; // Schedule to run at 12 PM every day  
System.schedule(jobName, cronExpression, new MySchedulableJob());
```

### 15.How to get information about scheduled jobs?

## By Smriti Sharan ([sfdcamlified](#))

We can query the CronTrigger object. The CronTrigger object contains information about scheduled Apex jobs, such as the cron expression, next fire time, and job status.

```
SELECT Id, CronExpression, CronJobDetail.Name, NextFireTime, State  
FROM CronTrigger
```

### 16.What is database.stateful?

Database.Stateful interface is used in Batch Apex to maintain state across multiple transactions within the same batch job. Normally, each execution of the execute method in a Batch Apex job runs in its own transaction. By implementing the Database.Stateful interface, we can retain the values of instance variables between these transactions.

Common use cases include aggregating results, keeping counters, or accumulating a list of records to process at the end of the batch job.

Example: Suppose we want to count the number of Opportunity records processed in a batch job. We can use Database.Stateful to maintain a counter across transactions.

### 17.Give a use case where we need to use Iterator rather than database.querylocator?

We need to process Account records where the criteria are complex. For example, we want to process accounts with:

- Accounts with a Rating of 'Hot'
- Custom field Score\_\_c greater than 80.
- Only if the account has a related Opportunity with an amount greater than \$100,000.

### Limitations of Database.QueryLocator

Subqueries and Aggregates: While SOQL supports subqueries, it does not support complex subqueries and aggregate functions in a way that would allow us to directly filter Accounts based on related Opportunities.

By Smriti Sharan ([sfdcamlified](#))

## Syntax

global class ProcessComplexAccounts implements  
Database.Batchable<SObject> {

```
    global Iterable<SObject> start(Database.BatchableContext BC) {  
        return new ComplexAccountIterable();  
    }
```

## 18.What is recursion and how to stop recursion?

Recursion can happen when a trigger updates a record, which then causes the same trigger to fire again and update the record again, creating a loop.

Example: an Account trigger updates related Contacts, and the update on Contacts, in turn, fires another trigger that updates the Account. This can cause a continuous loop of updates between the Account and Contacts.

To prevent recursion, we can use a static variable to keep track of whether the trigger has already run for a particular transaction.

1.Create a static variable to track whether the trigger has already executed.

```
    public static Boolean isTriggerExecuted = false;
```

2. Before executing the trigger logic, check the static variable. If the trigger has already executed, exit the trigger.

```
        if (TriggerHelper.isTriggerExecuted)
```

3.Set the static variable to indicate that the trigger has executed.

## 19.Explain Data Load in Serialization Mode vs. Parallel Mode ?

Let's take an example:

Accounts: We have Account A and Account B that need to be updated.



## By Smriti Sharan ([sfdcamlified](#))

Contacts: Contacts C1, C2 for Account A and Contacts D1, D2 for Account B.

There are two data load files—one for Accounts and one for Contacts.

### Parallel Mode

In parallel mode, Salesforce processes **multiple batches of records simultaneously**. This can lead to faster data loading but may cause **issues with record locking**.

Example, while Account A is being updated, the system might also try to insert Contact C1 and C2. If Account A is locked due to the update, the insert operation for Contacts will fail.

### Serialization Mode

In serialization mode, Salesforce processes **one batch of records at a time**, in a specific order. This mode ensures that updates and inserts are handled sequentially, **avoiding record locking issues**.

Best way to achieve this scenario:

1. Update Accounts First
2. Insert Contacts After Accounts. Since Accounts are already updated and not locked, inserting Contacts proceeds without issues.

## 20. How to Unlocking Records in Salesforce Approval Processes with Apex?

Once a record enters the approval process, **it gets locked to prevent further changes until the process is complete**. However, there can be scenarios where we need to unlock these records before the approval process is finished. This can be achieved using Apex.

**Using `Approval.unlock()`** to programmatically unlock records that are locked as part of an approval process.

## By Smriti Sharan ([sfdcamplified](#))

Syntax

```
Approval.UnlockResult result = Approval.unlock(recordId);
```

### 21.What are key considerations when converting from lookup to master?

- Make sure that all child records have associated parent records before changing the relationship type.
- Understand that child records will now inherit the sharing settings of the parent record. This means changes in parent record ownership and sharing rules will impact the child records.
- Deleting a parent record will automatically delete all related child records.
- We can create roll-up summary fields on the parent object to summarize data from the child records in a master-detail relationship.

### 22.What are ways to Parse Json in Apex?

**JSON.deserialize():** Use when you have a well-defined and consistent JSON structure and can create a corresponding Apex class.

**JSON.deserializeUntyped():** Use when dealing with dynamic or unknown JSON structures that require more flexibility.

**JSONParser:** Use for complex or deeply nested JSON structures where we need granular control over the parsing process.

### 23.How to maintain code reusability in LWC?

#### 1. Creating Reusable Components

## By Smriti Sharan ([sfdcamlified](#))

In LWC, we can create small, self-contained components that can be used in multiple places within the application. For example, we can create a button component that has customizable properties like label, style, and action. Once created, this button can be used in various other components without having to rewrite its logic or styling.

### 2. Using Utility Libraries

Utility libraries are collections of helper functions and constants that can be imported and used across different components. For example, we might have utility functions for formatting dates, numbers, or currency values. By placing these utilities in a shared library, we can avoid duplicating this logic in multiple components.

### 4. Inheritance

We might have a base component that includes methods for logging or common error handling. Other components can extend this base component to inherit these functionalities

## 24.How to ensure FLS in LWC? (very important)

### 1. Using Lightning Data Service (LDS)

When we use LDS-based components like lightning-record-form, lightning-record-view-form, and lightning-record-edit-form, FLS is enforced automatically.

### 2. Using @wire Adapters

When we use @wire with methods such as getRecord or getRecordUi, it automatically respects FLS.

### 3. Using Apex Controllers

If we need to fetch data using custom Apex controllers. In these cases, we must explicitly check for FLS in your Apex code.

Salesforce provides methods to check FLS, like Schema.sObjectType methods, which can be used to make sure that your Apex code only accesses fields the user has permission to see.

**By Smriti Sharan ([sfdcamlified](#))**

```
if (!Schema.sObjectType.Contact.fields.Name.isAccessible()) {}
```

#### **4. Using lightning/ui\*Api Adapters**

createRecord, updateRecord for CRUD operations will automatically follow FLS

#### **25.A custom Opportunity detail component that shows different sets of fields based on the Opportunity Stage. How can we achieve it using LWC?**

Use **getRecord** to fetch the Opportunity record and implement the custom logic and template to display the fields accordingly.

#### **26.A component that displays the details of a Case record to support agents without allowing edits. How to achieve this using LWC?**

Use **lightning-record-view-form** to display the Case details in a read-only format.

#### **27.Display data from two different objects Account and Contact, ensuring that FLS is respected?**

Use lightning-record-form, lightning-record-view-form, or lightning-record-edit-form Components. These components handle FLS automatically.

#### **28.How to refresh components in LWC?**

**Using @wire with Dynamic Parameters:** Automatically refreshes data **when parameters change**. Useful for components like dashboards where filters or criteria may change.

## By Smriti Sharan ([sfdcamlified](#))

**Using refreshApex:** Explicitly refreshes data when certain actions occur, such as form submissions. Ideal for cases where you want to refresh data after a backend operation.

**Manual Data Refresh with Imperative Apex Calls:** Provides full control over when and how data is refreshed, typically triggered by user actions like clicking a button.

**Using Pub/Sub Model:** It makes sure related components stay in sync without direct dependency.

### 29. Difference between notifyRecordUpdateAvailable vs RefreshApex vs RefreshView API??

**1. notifyRecordUpdateAvailable(recordIds):** Used specifically to notify the Lightning Data Service (LDS) that certain records have been updated.

Ideal for refreshing wired methods using lightning/uiRecordApi like getRecord.

Example: LWC component displays account details using getRecord. When application updates the account, we want component to automatically refresh and show the updated details.

#### Syntax

```
import { getRecord, notifyRecordUpdateAvailable } from  
'lightning/uiRecordApi';
```

**2. refreshApex:** Used to refresh data that was fetched using the wire service in Apex.

LWC component displays a list of accounts fetched using an Apex method. After updating the account list through some interaction, we want to refresh the list.

#### Syntax

```
import { refreshApex } from '@salesforce/apex';
```

**3. RefreshView API:** Used to refresh the entire view or Lightning page.

## By Smriti Sharan ([sfdcamlified](#))

Suitable for scenarios where you need a full-page refresh, typically used in Aura components and not specific to LWC.

Syntax

```
import { refreshView } from 'lightning/uiListApi';
```

**30. After an Order is placed and its status is updated to "Confirmed", an order confirmation email needs to be sent to the customer. How to achieve this?**

- Create a Queueable class that handles the asynchronous processing, such as sending an order confirmation email.
- Set up a trigger on the Order object to detect when the order status is updated to "Confirmed" and enqueue the Queueable job.

**31. How we solve locking of records in batch?**

- Reduce Batch Size: Smaller batches minimize simultaneous record locks.

```
Database.executeBatch(new MyBatchClass(), 50); // Smaller batch size
```

- Use SOQL FOR UPDATE: Lock records during processing to prevent conflicts.

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

- Use scheduled Apex to run batch jobs at different times.  
String cronExp = '0 0 1 \* \* ?'; // Every day at 1 AM  
System.schedule('Daily Batch Job', cronExp, new  
ScheduledBatchJob());

**By Smriti Sharan ([sfdcamplified](#))**

### **32.Does with sharing respect FLS? (very important)**

No, using with sharing in Apex does not respect Field-Level Security (FLS) or Object-Level Security (OLS). The with sharing keyword only ensures that **sharing rules and role hierarchy** are respected during the execution of the code.

To enforce FLS and OLS, we must explicitly check these permissions in code. This can be done using methods provided by the **Schema.DescribeSObjectResult** and **Schema.DescribeFieldResult** classes, or by using tools such as **Security.stripInaccessible** for stripping fields that the current user does not have access to.

#### **To check FLS before querying a field:**

```
if (Schema.sObjectType.Contact.fields.Email.isAccessible()) {  
    Contact c = [SELECT Email FROM Contact WHERE Id= :Id];  
}
```

#### **To check FLS before updating a field:**

```
if (Schema.sObjectType.Contact.fields.Email.isUpdateable()) {  
    // Update contact  
}
```

### **34.You want to query contacts and ensures only readable fields are returned. How will you do that in apex?**

```
List<Contact> contacts = [SELECT Id, Email, Phone FROM Contact];  
SObjectAccessDecision decision =  
Security.stripInaccessible(AccessType.READABLE, contacts);  
List<Contact> secureContacts = decision.getRecords();
```

### **35.How to authenticate in Salesforce?**

**By Smriti Sharan ([sfdcamlified](#))**

## **1. Username and Password Authentication**

This is the simplest form of authentication, where you use your Salesforce username, password, and security token.

## **2. OAuth 2.0**

OAuth 2.0 is a secure and flexible authentication mechanism. It allows for various flows depending on the use case, such as Web Server Flow, User-Agent Flow, and JWT Bearer Flow.

Common OAuth Flows:

- Web Server Flow: Used for server-to-server integrations. Requires an authorization code.
- User-Agent Flow: Used for client-side applications, like mobile apps.
- JWT Bearer Flow: Used for server-to-server integrations without user interaction.

## **3. JWT Bearer Flow**

JWT Bearer Flow is used for server-to-server integrations where no user interaction is required.

## **4. SAML (Security Assertion Markup Language)**

SAML is used for Single Sign-On (SSO) and provides a secure way to authenticate users by using an Identity Provider (IdP).

**36. Difference between master detail relationship vs lookup relationship?**



## By Smriti Sharan ([sfdcamlified](#))

Feature	Master-Detail Relationship	Lookup Relationship
Ownership and Sharing	Inherits from master	Independent
Deletion Behavior	Cascade delete (detail deleted with master)	Independent (child remains unless specified)
Field Requirement	Always required	Optional or required
Roll-Up Summary Fields	Supported	Not supported
Reparenting	Can be configured to allow/disallow	Allowed
Access and Security	Inherited from master	Independent
Typical Use Cases	Invoices and Line Items, Account and Opportunities	Contacts and Accounts, Case and Contact

### 37.Which tool you have used for deployment from sandbox to production?

- **Change Sets** are ideal for straightforward, native Salesforce deployments with minimal setup.
- **Salesforce CLI (SFDX)** are suitable for more complex deployments requiring automation and scripting.
- Third-Party Tools like **Gearset, Copado, and AutoRABIT** offer advanced features and user-friendly interfaces for comprehensive deployment management.

### 38.What is Lazy loading in LWC?

Lazy loading can be implemented to load data or components only when they are required by the user, such as when they scroll to a certain part of the page or interact with a specific element.

**By Smriti Sharan ([sfdcamlified](#))**

### **39.What are Best practices of trigger? (very important)**

1. One Trigger per Object: This ensures all logic for a particular object is centralized, making it easier to manage and control the order of execution.
2. Logic-less Triggers: Keep the trigger logic minimal. Use the trigger only to delegate processing to handler classes or methods. This makes the trigger more readable and easier to maintain.
3. Bulkify Your Code :Ensure code can handle multiple records at once. Salesforce processes triggers in batches of up to 200 records, so code should efficiently handle collections of records rather than processing one record at a time.
4. Avoid SOQL and DML in Loops: Never put SOQL queries or DML operations inside loops. This practice helps avoid hitting Salesforce governor limits. Instead, perform these operations outside the loop by using collections.
5. Use Context Variables: Utilize trigger context variables like `Trigger.new`, `Trigger.old`, `Trigger.isInsert`, `Trigger.isUpdate` to understand the context in which your trigger is running and to manage records appropriately.
6. Handle Recursion: Use static variables to track whether a trigger has already run in the current context, ensuring it doesn't run multiple times and create infinite loops.
7. Write Thorough Test Classes:

**By Smriti Sharan ([sfdcamlified](#))**

8. Use Collections: Use sets, lists, and maps to manage records efficiently within your trigger logic.

9. Optimize your SOQL queries to retrieve only the necessary fields and records. This practice helps improve performance and stay within governor limits.

***My Name is Smriti Sharan. I am [avid blogger](#) and [youtuber](#). Follow my Blog and youtube to learn various aspect of Salesforce.***

<https://t.me/sfdcamlified> feel free to join telegram group.