**::SQL::**

**1. Self Join vs. Regular Join**

**Question:** Explain the difference between a self join and a regular join. Can you provide an example scenario where a self join is necessary?

**Answer:** A self join is a regular join but the table is joined with itself. This is useful when you need to compare rows within the same table. For example, in an employee table, if you want to find pairs of employees who share the same manager, a self join can be used:

sql

Copy code

SELECT e1.name AS Employee, e2.name AS Manager

FROM employees e1

JOIN employees e2 ON e1.manager\_id = e2.id;

**2. CROSS JOIN Explanation**

**Question:** What is a CROSS JOIN and when would you use it?

**Answer:** A CROSS JOIN produces a Cartesian product of the two tables, meaning every row from the first table is combined with every row from the second table. It’s useful in scenarios where you need all combinations of two sets of data, such as generating test data or combinations of options.

sql

Copy code

SELECT a.option, b.value

FROM options a

CROSS JOIN values b;

**3. Handling NULLs**

**Question:** How do you handle NULL values in a SQL query, and why is it important?

**Answer:** NULL values can affect the results of calculations and comparisons. Use the IS NULL or IS NOT NULL conditions to filter them. Additionally, functions like COALESCE() can be used to replace NULLs with default values. For example:

sql

Copy code

SELECT name, COALESCE(phone, 'No Phone') AS phone

FROM contacts;

**4. Window Functions**

**Question:** What are window functions, and how do they differ from regular aggregate functions?

**Answer:** Window functions perform calculations across a set of table rows that are related to the current row. Unlike aggregate functions, they do not group rows into a single output row. An example is calculating a running total:

sql

Copy code

SELECT order\_id, amount,

SUM(amount) OVER (ORDER BY order\_id) AS running\_total

FROM orders;

**5. Indexing Strategy**

**Question:** When would you decide to use a composite index over single-column indexes, and what are the potential drawbacks?

**Answer:** A composite index is beneficial when queries filter or sort on multiple columns. It can improve performance significantly for those queries. However, it can slow down INSERT and UPDATE operations due to increased index maintenance. The choice should be based on query patterns and performance testing.

**6. CTE vs. Subquery**

**Question:** Compare Common Table Expressions (CTEs) and subqueries. When might you prefer one over the other?

**Answer:** CTEs improve readability and can be referenced multiple times within a query, while subqueries can be embedded directly within another query but can make it harder to read. Prefer CTEs for complex queries that require recursion or when you need to reference the same result set multiple times.

**7. Normalization vs. Denormalization**

**Question:** What is the trade-off between normalization and denormalization?

**Answer:** Normalization reduces redundancy and improves data integrity but can lead to complex queries and performance overhead due to multiple joins. Denormalization improves read performance and simplifies queries at the cost of increased redundancy and potential data anomalies. The choice depends on application requirements and workload characteristics.

**8. Using EXISTS vs. IN**

**Question:** What are the differences between using EXISTS and IN, and when would you choose one over the other?

**Answer:** EXISTS checks for the existence of rows returned by a subquery and stops processing once a match is found, making it more efficient with larger datasets. IN evaluates the result of a subquery into a list. Prefer EXISTS when the subquery might return a large number of rows, and IN when the subquery returns a small, fixed set of values.

**9. Performance Tuning**

**Question:** Describe a scenario where you had to troubleshoot and optimize a slow-running SQL query. What steps did you take?

**Answer:** (Provide a personal example here, outlining steps like analyzing the execution plan, identifying bottlenecks, adding indexes, rewriting joins, etc.)

**10. Transactions**

**Question:** What are ACID properties in transactions, and why are they important?

**Answer:** ACID stands for Atomicity, Consistency, Isolation, and Durability. These properties ensure reliable processing of database transactions. Atomicity guarantees that all operations in a transaction complete successfully, or none do. Consistency ensures that a transaction only brings the database from one valid state to another. Isolation ensures transactions do not interfere with each other, and durability guarantees that once a transaction is committed, it remains so, even in the event of a system failure.

### ::Simple Scenarios to write query.

### 1. ****Find Top N Salaries****

**Question:** Write a query to find the second highest salary from an employees table.

**Table Structure:**

sql

Copy code

CREATE TABLE employees (

id INT,

name VARCHAR(100),

salary DECIMAL(10, 2)

);

**Expected Query:**

sql

Copy code

SELECT MAX(salary) AS SecondHighestSalary

FROM employees

WHERE salary < (SELECT MAX(salary) FROM employees);

### 2. ****Count Employees by Department****

**Question:** Write a query to count the number of employees in each department from a departments table and an employees table.

**Table Structures:**

sql

Copy code

CREATE TABLE departments (

id INT,

name VARCHAR(100)

);

CREATE TABLE employees (

id INT,

name VARCHAR(100),

department\_id INT

);

**Expected Query:**

sql

Copy code

SELECT d.name AS DepartmentName, COUNT(e.id) AS EmployeeCount

FROM departments d

LEFT JOIN employees e ON d.id = e.department\_id

GROUP BY d.name;

### 3. ****Find Employees with No Projects****

**Question:** Write a query to find all employees who are not assigned to any projects from an employees table and a projects table.

**Table Structures:**

sql

Copy code

CREATE TABLE employees (

id INT,

name VARCHAR(100)

);

CREATE TABLE project\_assignments (

employee\_id INT,

project\_id INT

);

**Expected Query:**

sql

Copy code

SELECT e.name

FROM employees e

LEFT JOIN project\_assignments pa ON e.id = pa.employee\_id

WHERE pa.project\_id IS NULL;

### 4. ****Calculate Yearly Sales****

**Question:** Write a query to calculate the total sales for each year from a sales table.

**Table Structure:**

sql

Copy code

CREATE TABLE sales (

id INT,

sale\_date DATE,

amount DECIMAL(10, 2)

);

**Expected Query:**

sql

Copy code

SELECT YEAR(sale\_date) AS SaleYear, SUM(amount) AS TotalSales

FROM sales

GROUP BY YEAR(sale\_date)

ORDER BY SaleYear;

### 5. ****Identify Duplicate Entries****

**Question:** Write a query to find duplicate email addresses from a users table.

**Table Structure:**

sql

Copy code

CREATE TABLE users (

id INT,

email VARCHAR(100)

);

**Expected Query:**

sql

Copy code

SELECT email, COUNT(\*) AS Count

FROM users

GROUP BY email

HAVING COUNT(\*) > 1;

### ::Complex Scenarios to write query.

### 1. ****Find Gaps in Sequences****

**Question:** Write a query to find all missing numbers in a sequence from a numbers table, where the numbers are supposed to be consecutive.

**Table Structure:**

sql

Copy code

CREATE TABLE numbers (

num INT

);

**Expected Query:**

sql

Copy code

WITH seq AS (

SELECT MIN(num) AS start, MAX(num) AS end

FROM numbers

),

all\_nums AS (

SELECT start + n AS num

FROM seq, generate\_series(0, (SELECT end - start FROM seq)) AS n

)

SELECT num

FROM all\_nums

WHERE num NOT IN (SELECT num FROM numbers);

### 2. ****Running Total with Gaps****

**Question:** Write a query to calculate the running total of sales from a sales table, ensuring that if there are gaps in the dates, the running total continues without resetting.

**Table Structure:**

sql

Copy code

CREATE TABLE sales (

sale\_date DATE,

amount DECIMAL(10, 2)

);

**Expected Query:**

sql

Copy code

SELECT sale\_date,

SUM(amount) OVER (ORDER BY sale\_date ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW) AS running\_total

FROM sales

ORDER BY sale\_date;

### 3. ****Detecting Cycles in Relationships****

**Question:** Write a query to detect cycles in a parent-child relationship within a hierarchy table.

**Table Structure:**

sql

Copy code

CREATE TABLE hierarchy (

id INT,

parent\_id INT

);

**Expected Query:**

sql

Copy code

WITH RECURSIVE hierarchy\_path AS (

SELECT id, parent\_id, ARRAY[id] AS path

FROM hierarchy

WHERE parent\_id IS NOT NULL

UNION ALL

SELECT h.id, h.parent\_id, path || h.id

FROM hierarchy h

JOIN hierarchy\_path hp ON hp.parent\_id = h.id

WHERE NOT h.id = ANY(hp.path)

)

SELECT DISTINCT id

FROM hierarchy\_path

WHERE id = ANY(path);

### 4. ****Employee Hierarchy****

**Question:** Write a query to retrieve each employee along with their direct manager from an employees table.

**Table Structure:**

sql

Copy code

CREATE TABLE employees (

id INT,

name VARCHAR(100),

manager\_id INT

);

**Expected Query:**

sql

Copy code

SELECT e.name AS Employee, m.name AS Manager

FROM employees e

LEFT JOIN employees m ON e.manager\_id = m.id;

### 5. ****Date Difference in Days with Exclusions****

**Question:** Write a query to calculate the difference in days between two dates while excluding weekends from a date\_ranges table.

**Table Structure:**

sql

Copy code

CREATE TABLE date\_ranges (

start\_date DATE,

end\_date DATE

);

**Expected Query:**

sql

Copy code

WITH RECURSIVE date\_series AS (

SELECT start\_date AS current\_date

FROM date\_ranges

UNION ALL

SELECT current\_date + INTERVAL '1 day'

FROM date\_series

WHERE current\_date + INTERVAL '1 day' <= (SELECT end\_date FROM date\_ranges)

)

SELECT COUNT(\*) AS working\_days

FROM date\_series

WHERE EXTRACT(DOW FROM current\_date) NOT IN (0, 6; -- 0 = Sunday, 6 = Saturday

**:: .Net ::**

1. **Question 1: What is the difference between == and Equals() in .NET?**

**Answer:**

* + == is used to compare reference equality for reference types and value equality for value types.
  + Equals() method is used to compare the values of objects.
  + For custom types, Equals() can be overridden to provide custom comparison logic.

csharp

Copy code

string a = "Hello";

string b = "Hello";

Console.WriteLine(a == b); // True (compares values)

Console.WriteLine(a.Equals(b)); // True (compares values)

1. **Question 2: What is the purpose of using statement in .NET?**

**Answer:**

* + The using statement is used to ensure that IDisposable objects are properly disposed of.
  + It is typically used when dealing with unmanaged resources like file handles or database connections.

csharp

Copy code

using (StreamReader reader = new StreamReader("file.txt"))

{

Console.WriteLine(reader.ReadToEnd());

} // The reader is disposed here.

1. **Question 3: What are Value types and Reference types in .NET?**

**Answer:**

* + Value types are stored in the stack and hold data directly (e.g., int, double).
  + Reference types are stored in the heap and hold a reference to the data (e.g., class, string).

csharp

Copy code

int valueType = 10; // Value type

string referenceType = "Hello"; // Reference type

1. **Question 4: What is the difference between const and readonly in C#?**

**Answer:**

* + const variables are assigned at compile-time and cannot be changed.
  + readonly variables can be assigned at runtime or in a constructor but cannot be changed afterward.

csharp

Copy code

public const int MyConst = 10; // Compile-time constant

public readonly int MyReadonly = 20; // Assigned in the constructor

1. **Question 5: What is the purpose of async and await in .NET?**

**Answer:**

* + async and await are used to handle asynchronous programming in C#.
  + The async keyword marks a method as asynchronous, and the await keyword is used to wait for the completion of an asynchronous operation without blocking the main thread.

csharp

Copy code

public async Task<int> FetchDataAsync()

{

await Task.Delay(1000); // Simulating async operation

return 42;

}

1. **Question 1: How would you implement Dependency Injection (DI) in .NET Core?**

**Answer:**

* + DI is a design pattern that allows for loose coupling between classes by injecting dependencies from outside.
  + In .NET Core, DI is built-in and can be configured using the Startup class and IServiceCollection.

csharp

Copy code

// Service Interface

public interface IMyService

{

void Serve();

}

// Service Implementation

public class MyService : IMyService

{

public void Serve() => Console.WriteLine("Service Called");

}

// Register service in Startup.cs

public void ConfigureServices(IServiceCollection services)

{

services.AddTransient<IMyService, MyService>();

}

// Using the service

public class HomeController : Controller

{

private readonly IMyService \_service;

public HomeController(IMyService service)

{

\_service = service;

}

public IActionResult Index()

{

\_service.Serve();

return View();

}

}

1. **Question 2: How do you handle multithreading in .NET using Task and Thread classes?**

**Answer:**

* + The Task class is part of the Task Parallel Library (TPL) and is used for managing asynchronous operations.
  + The Thread class provides lower-level control over thread creation and execution but is more complex to manage.

csharp

Copy code

// Using Task

Task.Run(() =>

{

Console.WriteLine("Task Running");

});

// Using Thread

Thread thread = new Thread(() =>

{

Console.WriteLine("Thread Running");

});

thread.Start();

1. **Question 3: How do you create and handle custom exceptions in .NET?**

**Answer:**

* + Custom exceptions in .NET can be created by extending the Exception class and providing custom behavior.
  + Proper handling of custom exceptions ensures that specific issues in your application can be easily identified.

csharp

Copy code

public class MyCustomException : Exception

{

public MyCustomException(string message) : base(message)

{

}

}

// Usage

try

{

throw new MyCustomException("This is a custom exception.");

}

catch (MyCustomException ex)

{

Console.WriteLine(ex.Message);

}

1. **Question 4: How would you implement a thread-safe Singleton pattern in .NET?**

**Answer:**

* + A thread-safe Singleton ensures that only one instance of a class is created, even in multi-threaded environments.
  + This is typically done using the Lazy<T> type or double-checked locking.

csharp

Copy code

public class Singleton

{

private static Singleton \_instance;

private static readonly object \_lock = new object();

private Singleton() { }

public static Singleton Instance

{

get

{

if (\_instance == null)

{

lock (\_lock)

{

if (\_instance == null)

{

\_instance = new Singleton();

}

}

}

return \_instance;

}

}

}

1. **Question 5: How would you implement a custom IEnumerable in .NET?**

**Answer:**

* + To implement a custom IEnumerable, you need to define an enumerator for your collection by implementing the IEnumerable and IEnumerator interfaces.
  + This allows you to define custom iteration behavior.

csharp

Copy code

public class CustomCollection : IEnumerable<int>

{

private int[] \_data = { 1, 2, 3, 4, 5 };

public IEnumerator<int> GetEnumerator()

{

for (int i = 0; i < \_data.Length; i++)

{

yield return \_data[i];

}

}

IEnumerator IEnumerable.GetEnumerator()

{

return GetEnumerator();

}

}

// Usage

foreach (var item in new CustomCollection())

{

Console.WriteLine(item);

}

**:: .Angular ::**

### ****1. How can you optimize the performance of an Angular component when displaying a large list?****

**Answer**: Use Angular's trackBy function with ngFor to avoid unnecessary re-rendering of DOM elements. Additionally, implement virtual scrolling using cdk-virtual-scroll-viewport from Angular Material to render only the visible portion of the list.

### ****2. What happens if you forget to unsubscribe from an observable in Angular, and how can you avoid it?****

**Answer**: Forgetting to unsubscribe can lead to memory leaks, especially when dealing with long-lived observables. You can avoid this by using the async pipe in templates, or operators like takeUntil, or the Subscription’s unsubscribe() method in the ngOnDestroy lifecycle hook.

### ****3. How would you communicate between two sibling components in Angular?****

**Answer**: Use a shared service with a Subject or BehaviorSubject to emit data between sibling components. Both components can subscribe to the service’s observable to share data.

typescript

Copy code

// shared.service.ts

private dataSubject = new BehaviorSubject<string>("initial value");

data$ = this.dataSubject.asObservable();

sendData(data: string) {

this.dataSubject.next(data);

}

### ****4. What is the difference between a template-driven form and a reactive form?****

**Answer**: Template-driven forms are simpler and rely on Angular's directives in the template (ngModel), whereas reactive forms provide greater control over form state and logic by defining the form model directly in the component (FormControl, FormGroup). Reactive forms are better suited for complex validation and dynamic forms.

### ****5. How would you create a custom structural directive like**** \*ngIf****?****

**Answer**: A custom structural directive can be created by using the TemplateRef and ViewContainerRef to dynamically insert or remove template content. Example for a custom \*appShowIf directive:

typescript

Copy code

@Directive({

selector: '[appShowIf]'

})

export class ShowIfDirective {

constructor(private templateRef: TemplateRef<any>, private viewContainer: ViewContainerRef) {}

@Input() set appShowIf(condition: boolean) {

if (condition) {

this.viewContainer.createEmbeddedView(this.templateRef);

} else {

this.viewContainer.clear();

}

}

}

### ****6. How would you handle a scenario where a route requires data to be fetched before rendering the component?****

**Answer**: Use Angular route resolvers. A resolver is a service that implements the Resolve interface and is used to pre-fetch data before a route activates. The resolved data is then passed to the component via ActivatedRoute.

typescript

Copy code

@Injectable({

providedIn: 'root',

})

export class DataResolver implements Resolve<any> {

constructor(private dataService: DataService) {}

resolve(): Observable<any> {

return this.dataService.getData();

}

}

// Route config

{ path: 'data-route', component: DataComponent, resolve: { data: DataResolver } }

### ****7. What are Angular lifecycle hooks, and how would you use**** ngOnChanges ****to detect input changes?****

**Answer**: Angular lifecycle hooks are methods that get called at different phases of a component's lifecycle. ngOnChanges is triggered whenever an @Input() property changes. You can use this hook to react to changes in input properties.

typescript

Copy code

@Component({ /\* ... \*/ })

export class MyComponent implements OnChanges {

@Input() inputData: string;

ngOnChanges(changes: SimpleChanges) {

if (changes['inputData']) {

console.log('inputData changed:', changes['inputData'].currentValue);

}

}

}

### ****8. How do you use an Angular**** HttpInterceptor ****to append a token to all outgoing HTTP requests?****

**Answer**: An HttpInterceptor can be implemented to modify HTTP requests globally. It can be used to append an authorization token to every request’s headers.

typescript

Copy code

@Injectable()

export class AuthInterceptor implements HttpInterceptor {

intercept(req: HttpRequest<any>, next: HttpHandler): Observable<HttpEvent<any>> {

const authToken = 'Bearer some-token';

const authReq = req.clone({

headers: req.headers.set('Authorization', authToken),

});

return next.handle(authReq);

}

}

### ****9. What is the purpose of**** RouterModule.forRoot() ****and**** RouterModule.forChild()****?****

**Answer**: RouterModule.forRoot() is used to configure the root-level routing for an Angular application, typically in the AppModule. RouterModule.forChild() is used to configure routing in feature modules. It enables modular routing and lazy loading.

### ****10. How can you conditionally apply multiple classes to an element in Angular?****

**Answer**: You can use [ngClass] to conditionally apply multiple classes based on the evaluation of expressions. It allows you to dynamically toggle classes on an element.

html

Copy code

<div [ngClass]="{'class-a': conditionA, 'class-b': conditionB}">

Conditional Classes

</div>

**1. Explain the Angular Dependency Injection (DI) mechanism and how you would create a custom injector.**

* **Answer**: Angular uses a hierarchical Dependency Injection (DI) system. It provides services at different levels: module, component, and service levels. The injector creates and maintains instances of services. A custom injector can be created by extending the Injector class or using ReflectiveInjector for more fine-grained control over service injection.

**2. How does Angular handle change detection, and how can you optimize it for better performance?**

* **Answer**: Angular uses the Zone.js library for change detection, automatically triggering it when asynchronous operations occur. Optimizing change detection can be done by using OnPush strategy in components to limit checks to only when input properties change. Additionally, using trackBy in ngFor can prevent unnecessary DOM updates.

**3. How would you implement lazy loading and preloading strategies in Angular for optimizing large applications?**

* **Answer**: Lazy loading in Angular is done using the loadChildren syntax in the route configuration. Preloading strategies, such as the built-in PreloadAllModules or custom preloading strategies, allow modules to load in the background after the main app has loaded.

**4. What are Angular templates and view encapsulation modes, and how do they affect component styles?**

* **Answer**: Angular provides three view encapsulation modes: Emulated, None, and ShadowDom. Emulated is the default, which scopes styles to the component using attribute selectors. None disables encapsulation, and ShadowDom uses native shadow DOM technology for true encapsulation, making the styles completely isolated.

**5. Describe the concept of state management in Angular. How would you implement global state management using NgRx?**

* **Answer**: State management involves managing the state of an application in a predictable and consistent way. NgRx (based on Redux) manages global state using actions, reducers, and selectors. Components dispatch actions to modify the state, and selectors are used to read the state.

**6. What is Angular's AOT (Ahead-of-Time) compilation, and how does it differ from JIT (Just-in-Time) compilation?**

* **Answer**: AOT compiles Angular templates and TypeScript code during the build time, resulting in smaller bundles and faster startup times. JIT compiles templates in the browser during runtime, which can result in larger bundles and slower performance. AOT is preferred for production environments due to its efficiency.

**7. How would you handle error handling and logging in an Angular application?**

* **Answer**: Error handling in Angular can be done using the HttpInterceptor for capturing HTTP errors and global error handling with the ErrorHandler service. Logging can be implemented through a centralized logging service that sends logs to external services like ElasticSearch or third-party tools such as Sentry.

**8. How does Angular handle forms (template-driven vs reactive) and how do you choose between them?**

* **Answer**: Angular offers two types of forms: template-driven and reactive. Template-driven forms are easier for simple forms but harder to test and maintain in large applications. Reactive forms provide more control over form state and validation, making them more suitable for complex forms.

**9. What is Angular Universal, and how would you implement server-side rendering (SSR) for an Angular application?**

* **Answer**: Angular Universal enables server-side rendering (SSR) for Angular applications, allowing faster initial page load and better SEO. SSR can be implemented by setting up an Angular Universal project using the @angular/platform-server package and configuring it with Node.js for serving the application on the server.

**10. What are dynamic components in Angular, and how would you load them at runtime?**

* **Answer**: Dynamic components are components that are loaded at runtime instead of being statically declared in a template. To load them dynamically, Angular provides ComponentFactoryResolver which allows you to create and insert a component into the view programmatically.

**11. Explain Angular’s RouterModule and how to implement advanced routing techniques such as guards, resolvers, and custom route animations.**

* **Answer**: Angular’s RouterModule provides routing functionality. Guards (CanActivate, CanDeactivate) control access to routes, resolvers pre-fetch data before navigation, and route animations can be implemented using Angular's @angular/animations package to animate route transitions.

**12. How do you manage memory leaks in Angular applications, particularly in components and services?**

* **Answer**: Memory leaks can be managed by properly unsubscribing from observables (e.g., using takeUntil or ngOnDestroy), avoiding direct DOM manipulation, and cleaning up resources in the ngOnDestroy lifecycle hook. Profiling tools like Chrome DevTools can help identify memory leaks.

**13. How does Angular handle large data sets, and what are some techniques for optimizing performance when dealing with them?**

* **Answer**: Techniques for optimizing large datasets in Angular include using pagination, virtual scrolling with cdk-virtual-scroll-viewport, memoizing heavy computations with ngrx selectors or async pipe, and debouncing or throttling user inputs for performance improvements.

**14. What is the role of zone.js in Angular, and how does it help in change detection?**

* **Answer**: Zone.js is a library that patches asynchronous operations and notifies Angular when to run change detection. It keeps track of tasks like promises, timeouts, and event listeners, and ensures the UI updates when data changes asynchronously.

**15. How do you implement custom pipes, and when would you use pure vs impure pipes?**

* **Answer**: Custom pipes transform data in templates. Pure pipes run only when their input changes, while impure pipes run on every change detection cycle, even if their inputs haven’t changed. Pure pipes are more performant and should be used unless dealing with collections or asynchronous operations.

**16. What is the purpose of ngZone.run() and ngZone.runOutsideAngular() in Angular applications?**

* **Answer**: ngZone.run() is used to execute code inside Angular's zone, triggering change detection after the code runs. ngZone.runOutsideAngular() is used to run code outside of Angular's zone, preventing unnecessary change detection for operations that don’t affect the view, such as performance-heavy background tasks.

**17. How do you implement authentication and authorization in Angular using route guards and JWT?**

* **Answer**: Route guards like CanActivate can be used to restrict access to certain routes. JWT (JSON Web Tokens) can be used for authentication by storing the token in localStorage or sessionStorage and adding it to HTTP requests via HttpInterceptor to secure API calls.

**18. What is the role of the async pipe in Angular, and how does it work with observables and promises?**

* **Answer**: The async pipe automatically subscribes to observables and promises and marks the component for change detection when new values are emitted. It also handles unsubscribing when the component is destroyed, preventing memory leaks.

**19. How would you create and implement a custom Angular decorator for a component or directive?**

* **Answer**: Custom decorators can be created by writing a function that modifies a class or method. For example, you can create a logging decorator that logs component lifecycle events. The decorator is applied to a class by using the @ syntax.

**20. How do you perform unit testing and integration testing in Angular, and what tools do you use?**

* **Answer**: Unit tests are performed using Jasmine and Karma to test components, services, and pipes in isolation. TestBed is used to configure and initialize the environment. For integration testing, tools like Protractor or Cypress are used to simulate end-to-end user interactions.

**:: .React JS ::**

**::General**

### 1. ****What is the virtual DOM, and how does React use it to improve performance?****

**Answer**: The virtual DOM is an in-memory representation of the actual DOM. React updates the virtual DOM whenever a component’s state or props change and compares it with the previous version using a diffing algorithm. Only the changes are then updated in the real DOM, minimizing expensive DOM operations and improving performance.

### 2. ****What are the differences between functional components and class components in React?****

**Answer**: Functional components are simpler and primarily used for presenting UI. They became more powerful with hooks, allowing state management and side effects. Class components use lifecycle methods and state, but since the introduction of hooks in React 16.8, functional components can achieve the same functionality, making class components less common.

### 3. ****Explain how the**** useState ****hook works in React.****

**Answer**: useState is a React hook that allows functional components to have a state. It returns an array with two values: the current state and a function to update that state. The useState hook is initialized with a default value, and the state can be updated using the setter function.

javascript

Copy code

const [count, setCount] = useState(0);

### 4. ****What is JSX, and how is it different from HTML?****

**Answer**: JSX is a syntax extension for JavaScript that allows you to write HTML-like code inside JavaScript. Unlike HTML, JSX allows JavaScript expressions within curly braces, and some HTML attributes, like class and for, are written as className and htmlFor to align with JavaScript naming conventions.

### 5. ****How does React handle state and props? What is the difference between them?****

**Answer**: State is local to a component and can be changed internally by the component. Props, on the other hand, are passed from a parent component to child components and are immutable in the child component. State is used to handle dynamic data within a component, whereas props are used to pass data between components.

### 6. ****What are React lifecycle methods, and which ones are commonly used in class components?****

**Answer**: Lifecycle methods are hooks that allow class components to respond to events at different phases of their life. Commonly used lifecycle methods include componentDidMount, componentDidUpdate, and componentWillUnmount. These methods are used to execute code during the mounting, updating, and unmounting phases of a component’s lifecycle.

### 7. ****What is the purpose of**** useEffect ****in React?****

**Answer**: useEffect is a hook that lets you perform side effects in functional components, such as data fetching, subscribing to services, or manually updating the DOM. It runs after the render and can be controlled by providing dependencies to only run when certain variables change.

javascript

Copy code

useEffect(() => {

// Side effect code

}, [dependency]);

### 8. ****How does React handle events, and how is it different from handling events in plain HTML?****

**Answer**: React handles events using synthetic event handlers, which are wrappers around the browser's native events. The event handling in React is consistent across different browsers, and instead of using traditional event attributes (e.g., onclick), React uses camelCase syntax (e.g., onClick), with functions passed as event handlers.

### 9. ****What is the purpose of**** React.StrictMode****, and how does it help during development?****

**Answer**: React.StrictMode is a tool used during development to help identify potential issues in a React application, such as unsafe lifecycle methods, legacy API usage, and side effects. It does not render anything to the DOM but highlights these issues to ensure that your app adheres to best practices.

### 10. ****What are React keys, and why are they important in lists?****

**Answer**: Keys are unique identifiers that help React identify which items in a list have changed, been added, or removed. They help in efficiently updating and reordering list elements without re-rendering the entire list. Keys should be unique and stable among sibling elements.

**::Complex**

**1. Explain the concept of Reconciliation in React. How does React handle DOM updates efficiently?**

* **Answer**: Reconciliation is the process through which React updates the DOM. React creates a virtual DOM and compares it with the previous virtual DOM (diffing algorithm) to update only the changed parts in the actual DOM, making updates more efficient.

**2. What are the differences between controlled and uncontrolled components in React? When would you use each?**

* **Answer**: Controlled components rely on React state for form control (e.g., value stored in state), whereas uncontrolled components use the DOM to manage state (e.g., refs). Controlled components provide more control and are preferred in most cases, especially for form validation and handling.

**3. What are React hooks? How do they replace lifecycle methods in function components?**

* **Answer**: Hooks like useState, useEffect, useMemo, and useCallback allow you to use state and side effects in functional components. For example, useEffect replaces lifecycle methods like componentDidMount and componentDidUpdate.

**4. How would you optimize the performance of a large React application?**

* **Answer**: Performance optimization techniques include memoization (React.memo, useMemo, useCallback), lazy loading with React.lazy and Suspense, code-splitting with dynamic imports, avoiding unnecessary re-renders by using shouldComponentUpdate or PureComponent, and using a state management tool like Redux selectively.

**5. What is the significance of the key prop in React, and why should it be unique?**

* **Answer**: The key prop is essential for list rendering, enabling React to identify which items have changed, added, or removed, optimizing the update process. Keys must be unique among siblings to avoid unnecessary re-renders or state mismatches.

**6. Describe the concept of HOCs (Higher-Order Components) in React and provide a use case.**

* **Answer**: HOCs are functions that take a component and return a new component with additional props or behavior. A common use case is wrapping a component with functionality like authentication checks or logging.

**7. What are the differences between useEffect and useLayoutEffect? When would you use one over the other?**

* **Answer**: useEffect runs asynchronously after the render, while useLayoutEffect runs synchronously after the DOM mutations but before painting, blocking visual updates. Use useLayoutEffect when you need to make DOM updates before the screen refreshes, such as reading layout or style information.

**8. Explain the React Context API and when you would use it instead of Redux.**

* **Answer**: The Context API is used for prop drilling avoidance, sharing state between components without passing props explicitly. It is suited for simple, local state management across a few components, whereas Redux is better for large-scale state management with more complex logic.

**9. How do you handle error boundaries in React? What’s the difference between componentDidCatch and getDerivedStateFromError?**

* **Answer**: Error boundaries are React components that catch JavaScript errors anywhere in their child component tree. componentDidCatch logs errors, and getDerivedStateFromError is used to update the UI to a fallback. Functional components use ErrorBoundary components for error handling.

**10. What is code splitting in React and how does React.lazy enable it?**

* **Answer**: Code splitting is the practice of breaking the bundle into smaller chunks that can be loaded on demand. React.lazy allows components to be loaded lazily by dynamically importing them, reducing initial load times.

**11. What are the benefits and use cases of React.memo?**

* **Answer**: React.memo is a higher-order component that memoizes functional components, preventing unnecessary re-renders if props haven’t changed. It is useful in performance optimization, especially for components that re-render frequently with the same props.

**12. Explain the difference between useCallback and useMemo hooks.**

* **Answer**: useCallback memoizes a function reference to prevent re-creation between renders, while useMemo memoizes the return value of a computation. useCallback is used to optimize callback dependencies, and useMemo is used for optimizing expensive computations.

**13. How do you implement SSR (Server-Side Rendering) with React? What are its benefits?**

* **Answer**: SSR can be implemented using frameworks like Next.js or libraries like ReactDOMServer. The benefits include improved performance (faster initial load), better SEO, and enhanced user experience on slower networks by rendering the initial HTML on the server.

**14. Explain the role of shouldComponentUpdate and PureComponent in React.**

* **Answer**: shouldComponentUpdate is a lifecycle method that controls whether a component should re-render. PureComponent is a base class that implements shouldComponentUpdate with a shallow prop and state comparison. It prevents unnecessary re-renders if the props and state haven’t changed.

**15. What are render props, and how do they differ from HOCs?**

* **Answer**: Render props is a pattern where a component’s child is a function that returns JSX. Unlike HOCs, render props allow more flexible reuse by passing JSX or logic directly, whereas HOCs enhance a component by wrapping it with additional functionality.

**16. Explain the difference between useState and useReducer and when you would use each.**

* **Answer**: useState is used for simple state management, while useReducer is used for more complex state logic or when state depends on previous state, similar to Redux. useReducer is preferred when the state management logic is more intricate or involves multiple actions.

**17. What is React’s StrictMode, and how does it help in development?**

* **Answer**: StrictMode is a tool for highlighting potential problems in React applications, such as side effects that are not properly handled, deprecated lifecycle methods, and detecting unexpected side effects by double-invoking certain lifecycle methods in development mode.

**18. Explain the difference between synchronous and asynchronous state updates in React.**

* **Answer**: React batches state updates and processes them asynchronously in the next render cycle. Synchronous updates occur when using useState in an event handler, but asynchronous behavior appears when the update is in an asynchronous function or API call.

**19. How does React handle component updates in concurrent mode?**

* **Answer**: In concurrent mode, React pauses or interrupts rendering to prioritize more urgent updates, like user interactions. This improves user experience by ensuring the UI remains responsive even during heavy computation or rendering processes.

**20. How would you manage side effects in a React functional component using hooks?**

* **Answer**: Side effects are managed using the useEffect hook. You can perform operations like data fetching, subscriptions, or manually interacting with the DOM. To handle cleanup, return a cleanup function from useEffect to prevent memory leaks.

**21. What is Suspense in React, and how does it work with React.lazy?**

* **Answer**: Suspense is a component that allows you to show a fallback UI while a component is being lazily loaded. It works with React.lazy by wrapping the lazy-loaded component in Suspense, displaying a loading spinner or message until the component is ready.

**22. What are the limitations of using Context API for state management compared to Redux?**

* **Answer**: The Context API is simple and suitable for light state management, but it can lead to performance issues (like unnecessary re-renders) when dealing with complex or deeply nested state. Redux, with its middleware (like thunk or saga), is more scalable and better suited for large applications.

**23. How does React handle event delegation, and what are the benefits?**

* **Answer**: React uses a single event listener at the root of the DOM tree to handle all events through event delegation, capturing events higher up in the DOM tree and dispatching them to the appropriate component. This improves performance and simplifies event handling.

**24. What is a React fragment, and how does it differ from a standard HTML element?**

* **Answer**: A React fragment (<></> or <React.Fragment>) is used to group a list of children without adding extra nodes to the DOM. Unlike an HTML element like <div>, fragments do not render any additional DOM elements.

**25. Explain how to use PropTypes in React and why it is important.**

* **Answer**: PropTypes is a type-checking library built into React that allows you to validate the types of props passed to a component. This ensures components receive the correct data types, helping catch bugs early during development