100+ React JS, .Net and SQ Server

Interview Questions and Answers

Sai Reddy 5/6/25 Interview Tips

100+ Full Stack Interview Q&A

React JS interview q&a (0–2 Years Experience):

1. **What is JSX?**
   JSX stands for JavaScript XML and allows writing HTML-like code inside JavaScript.
   It simplifies UI creation by blending markup and logic.
   Browsers don't understand JSX directly, so Babel compiles it to React.createElement().
   JSX makes components more readable and expressive.
   Example: `<h1>Hello, {name}</h1>`

2. **How do you create a functional component in React?**
   Functional components are JavaScript functions that return JSX.
   They are simple, reusable, and support React Hooks.
   They do not use this, unlike class components.
   Use them for stateless or presentational components.
   Example:
   ```
   function Greeting() { return <h1>Hello!</h1>; }
   ```

3. **What are props in React?**
   Props (short for properties) are inputs passed to components.
   They are read-only and help customize component behavior.
   Used for parent-child communication.
   You access props via function arguments.
   Example:
   ```
   <Greeting name="John" />
   ```

4. **What is the difference between state and props?**
   Props are external, immutable inputs passed from parent to child.
   State is internal, mutable data managed within a component.
   State changes trigger re-renders, while props are controlled by parent.
   Both are used for dynamic rendering.
   Example:
   ```
   const [count, setCount] = useState(0); // state
   ```

5. **How do you handle events in React?**
   React uses camelCase syntax for events (e.g., onClick).
   Event handlers are passed as functions.
   You can use inline or separate handler functions.
   React normalizes events using the SyntheticEvent wrapper.
   Example:
   ```
   <button onClick={handleClick}>Click Me</button>
   ```

6. **What is the purpose of useState hook?**
   useState manages local state in functional components.
   It returns an array: current value and a function to update it.
   State updates trigger re-renders.

Each call to useState is independent.
Example:
const [count, setCount] = useState(0);

7. **What is useEffect used for?**
useEffect handles side effects in functional components.
It runs after render and can be configured with dependencies.
Common uses: API calls, subscriptions, and DOM manipulation.
Cleanup functions can be returned.
Example:
useEffect(() => { fetchData(); }, []);

8. **How do you conditionally render elements in React?**
Use JavaScript conditional expressions inside JSX.
Options include ternary operators, &&, or if blocks.
Helps show/hide UI elements dynamically.
Avoid unnecessary JSX clutter with helpers.
Example:
{isLoggedIn ? <Logout /> : <Login />}

9. **What is the virtual DOM?**
Virtual DOM is an in-memory representation of the real DOM.
React uses it to optimize updates by diffing and batching changes.
Only changed elements are updated in the real DOM.
Improves performance and rendering speed.
Example: React compares two VDOMs before applying changes.

10. **How does React handle form submissions?**
React uses controlled components with state-bound inputs.
Handle the onSubmit event and prevent default behavior.
Use useState to manage input values.
Submit logic is triggered via handler functions.
Example:
<form onSubmit={handleSubmit}>...</form>

11. **What are controlled components?**
In controlled components, input value is controlled via state.
value and onChange props bind the input to React state.
Gives full control over the form logic.
Essential for form validation and dynamic behavior.
Example:
<input value={name} onChange={e => setName(e.target.value)} />

12. **How do you pass data from child to parent?**
Pass a function from parent to child via props.
The child calls the function and sends data as argument.
This enables upward communication.

Useful in forms and custom input handling.

Example:

```
props.onDataSubmit(data);
```

13. What is prop drilling?

Prop drilling is passing data through multiple component layers.
Becomes cumbersome when many intermediate components are involved.
Can be avoided with context or state management libraries.
Leads to tightly coupled code.
Example: Parent → A → B → C → Child

14. **How can you avoid prop drilling?**

Use React Context API or state management tools like Redux.
Context lets you share data across components without prop chains.
Keep the context specific to avoid re-renders.
Simplifies code and enhances maintainability.
Example:

```
<MyContext.Provider value={data}>...</MyContext.Provider>
```

15. **What is the purpose of React.Fragment?**

It lets you return multiple elements without adding extra nodes.
Avoids unnecessary divs in the DOM.
Use shorthand syntax <>...</>.
Improves semantic structure and styling control.
Example:

```
<><Header /><Content /></>
```

16. **What are keys in React lists?**

Keys help React identify elements during re-renders.
They must be unique among siblings.
Improve performance and avoid bugs in list rendering.
Usually derived from data like IDs.
Example:

```
{items.map(i => <li key={i.id}>{i.name}</li>)}
```

17. How do you debug React apps?

Use browser dev tools and React Developer Tools extension.
Add console.log() in components to trace logic.
Check network tab for API calls.
React DevTools shows component tree and state.
Example:

```
Inspect state updates in React DevTools.
```

18. **What is the role of defaultProps?**

defaultProps provide fallback values for props.
Used when a parent component doesn't pass some props.
Ensures components don’t break due to undefined values.

Mostly used in class components or outside function.

Example:

```
Component.defaultProps = { name: 'Guest' }
```

19. **What is the difference between functional and class components?**

    Functional components are simpler, use hooks, and don't need this.
    Class components use lifecycle methods and manage state via this.state.
    Hooks made functional components more powerful.
    Functional components are preferred in modern React.

    Example:

    ```
    function Greet() { return <h1>Hi</h1>; }
    ```

20. **How do you handle errors in React?**

    Use error boundaries for class components.
    In functional components, use try-catch in async logic.
    React 18+ supports useEffect error capture with ErrorBoundary.
    Fallback UI can be rendered during errors.

    Example:

    ```
    componentDidCatch(error, info) { ... }
    ```

React JS Interview Questions (3–6 Years Experience)

1. **How do you manage global state in React?**

   Global state is shared across multiple components.
   You can use Context API, Redux, Zustand, or Recoil.
   Redux offers centralized, predictable state management.
   Context API is simpler for smaller apps.

   Example:

   ```
   <MyContext.Provider value={user}>...</MyContext.Provider>
   ```

2. **What is Context API and how does it work?**

   Context API allows passing data without prop drilling.
   You create context with createContext() and access via useContext().
   It provides global state access across the component tree.
   Avoids passing props at multiple levels.

   Example:

   ```
   const user = useContext(UserContext);
   ```

3. **How do you optimize React app performance?**

   Use memoization, code splitting, lazy loading, and virtualization.
   Avoid unnecessary re-renders with React.memo, useCallback, etc.
   Split code using dynamic import() statements.
   Monitor performance using dev tools.

   Example:

   ```
   const MemoComp = React.memo(MyComponent);
   ```

4. **What is memoization in React?**

   Memoization caches the result of expensive operations.
   React provides useMemo and React.memo for this purpose.

It avoids recalculations on every render.
Helpful for performance-intensive components.
Example:
```
const result = useMemo(() => computeValue(data), [data]);
```

5. **What is useCallback and when to use it?**
   useCallback memoizes function definitions.
   Useful when passing callbacks to child components.
   It prevents unnecessary re-renders.
   It only changes when dependencies change.
   Example:
   ```
   const handleClick = useCallback(() => doSomething(), []);
   ```

6. **What is useMemo and when to use it?**
   useMemo memoizes values returned from functions.
   Used for expensive calculations that shouldn't rerun on every render.
   Helps boost performance when dependencies don’t change.
   Only recalculates when dependencies change.
   Example:
   ```
   const sortedItems = useMemo(() => sort(items), [items]);
   ```

7. **Explain React Refs and useRef hook.**
   Refs give direct access to DOM elements or mutable variables.
   useRef returns a persistent object across renders.
   Commonly used for focusing inputs or storing previous values.
   Avoids unnecessary state-based re-renders.
   Example:
   ```
   const inputRef = useRef(); inputRef.current.focus();
   ```

8. **How do you handle side effects in React?**
   Use useEffect to handle side effects like data fetching or DOM changes.
   It runs after the component renders.
   Can return cleanup logic for unmounting or updates.
   Dependencies control when it re-runs.
   Example:
   ```
   useEffect(() => fetchData(), []);
   ```

9. **How does React handle component lifecycle?**
   React functional components use useEffect to mimic lifecycle stages.
   Mounting: useEffect(() => {}, [])
   Updating: useEffect(() => {}, [deps])
   Unmounting: return a cleanup function.
   Example:
   ```
   useEffect(() => { return () => cleanup(); }, []);
   ```

10. **What is the difference between useEffect and componentDidMount?**
    componentDidMount is a class method, runs once after mount.

useEffect(() => {}, []) behaves similarly in functional components.
useEffect is more versatile and handles all lifecycle events.
You can manage side effects in one place.
Example:

```
useEffect(() => { loadData(); }, []);
```

11. **How do you implement code splitting in React?**
    Split large bundles using dynamic import() statements.
    React's React.lazy() enables loading components on demand.
    Improves initial load performance.
    Use Suspense to show fallback during loading.
    Example:

```
const LazyComp = React.lazy(() => import('./MyComp'));
```

12. **How do you lazy load components in React?**
    Use React.lazy() and wrap the component in <Suspense>.
    Only loads the component when it's needed.
    Improves app startup speed.
    Ideal for routes and heavy UI components.
    Example:

```
<Suspense fallback={<Spinner />}><LazyComp /></Suspense>
```

13. **What is React Router and how is it used?**
    React Router manages navigation in single-page apps.
    Use BrowserRouter, Routes, and Route components.
    Allows nested, dynamic, and parameterized routing.
    Use useNavigate and useParams for control.
    Example:

```
<Route path="/about" element={<About />} />
```

14. **How do you handle authentication in React?**
    Use token-based auth (e.g., JWT) stored in localStorage or cookies.
    Guard routes using PrivateRoute components.
    Manage auth state via context or Redux.
    Validate token on component mount.
    Example:

```
if (!isAuthenticated) return <Navigate to="/login" />
```

15. **What is HOC (Higher Order Component)?**
    HOC is a function that takes a component and returns a new one.
    Used to reuse logic across components.
    It enhances the original component with extra props or behavior.
    Often used for auth or logging.
    Example:

```
const WithLogger = Wrapped => props => { console.log(props); return <Wrapped {...props} />; };
```

16. **What is a custom hook and why would you use one?**
    Custom hooks encapsulate reusable logic using other hooks.
    They start with use prefix.
    Improve code modularity and reusability.
    Used for form handling, fetching, or complex logic.
    Example:
    `function useAuth() { return useContext(AuthContext); }`

17. **How do you handle pagination in React?**
    Divide data into chunks and show one page at a time.
    Use states for currentPage and pageSize.
    Slice the data array accordingly.
    Optionally fetch paginated data from API.
    Example:
    `const current = data.slice(startIndex, endIndex);`

18. **How do you structure a large-scale React app?**
    Organize by feature folders or domains (e.g., /users, /orders).
    Separate concerns: components, hooks, services, contexts.
    Use absolute imports and lazy loading.
    Maintain reusable UI libraries and consistent naming.
    Example:
    `src/components, src/hooks, src/pages, src/services`

19. **What are React performance tools?**
    Use React Developer Tools, Chrome Profiler, and Lighthouse.
    React DevTools shows re-renders and state changes.
    Profiler helps find performance bottlenecks.
    Lighthouse audits app speed and accessibility.
    Example:
    `Use React Profiler tab to inspect component re-renders.`

20. **What are common security issues in React?**
    Cross-Site Scripting (XSS), insecure localStorage, CSRF.
    Always sanitize user input and escape output.
    Avoid dangerouslySetInnerHTML.
    Use HTTPS, content security policies, and secure auth flows.
    Example:
    `<div>{DOMPurify.sanitize(userInput)}</div>`

React JS Interview Questions (7–12 Years Experience)

1. **How do you implement SSR (Server Side Rendering) in React?**
    SSR renders HTML on the server before sending it to the client.
    It improves SEO and initial load performance.
    Frameworks like **Next.js** simplify SSR implementation.
    You use getServerSideProps or getStaticProps for data fetching.
    Example:

```
export async function getServerSideProps() { return { props: { user } }; }
```

2. **Compare Redux vs Context API.**
   Redux is powerful for large-scale apps with complex state logic.
   Context API suits smaller apps or light global state.
   Redux offers middleware, time-travel debugging, and DevTools.
   Context may re-render too many components if misused.
   Example:
   Use useSelector in Redux vs useContext in Context API.

3. **What is Redux middleware and how do you use it?**
   Middleware intercepts actions before reaching reducers.
   Used for async logic, logging, and error handling.
   Popular middleware includes redux-thunk and redux-saga.
   Apply with applyMiddleware() in Redux store.
   Example:
   ```
   const store = createStore(reducer, applyMiddleware(thunk));
   ```

4. **How do you implement authentication and authorization with roles?**
   Store JWT in secure HTTP-only cookies or localStorage.
   Use React Router to guard routes based on user roles.
   Backend should return user roles with token.
   Validate roles before rendering components.
   Example:
   ```
   {user.role === 'admin' ? <AdminPanel /> : <AccessDenied />}
   ```

5. **What are some common anti-patterns in React?**
   Mutating state directly, excessive prop drilling, large components,
   placing logic in JSX, or abusing useEffect.
   These can cause bugs and performance issues.
   Use clean architecture, hooks, and modular code.
   Example:
   Avoid: state.count++ → Correct: setCount(prev => prev + 1)

6. **How do you manage environment-based configuration in React apps?**
   Use .env, .env.production, and .env.development files.
   Variables must start with REACT_APP_.
   Use process.env.REACT_APP_API_URL in your code.
   Web server should handle environment separation in CI/CD.
   Example:
   ```
   const API = process.env.REACT_APP_API_URL;
   ```

7. **How do you test React components using Jest and React Testing Library?**
   Use **Jest** for test runner and **RTL** for UI behavior testing.
   Use render, fireEvent, and screen to simulate interaction.
   Focus on user-centric queries, not implementation.

Test logic, rendering, and async behavior.
Example:
```
render(<Button />); fireEvent.click(screen.getByText('Click'));
```

8. **What is reconciliation in React?**
Reconciliation is React's diffing algorithm.
It compares old and new virtual DOM trees.
Only the changed parts are updated in the real DOM.
Keys in lists help React optimize this process.
Example:
```
<ul>{items.map(i => <li key={i.id}>{i.name}</li>)}</ul>
```

9. **Explain the concept of portals in React.**
Portals allow rendering children into a different DOM subtree.
Useful for modals, tooltips, or dialogs outside parent hierarchy.
Use ReactDOM.createPortal(child, container).
Maintains React context and lifecycle.
Example:
```
createPortal(<Modal />, document.getElementById('modal-root'))
```

10. **How does React differ from other frameworks like Angular or Vue?**
React is a library focusing on UI with unidirectional data flow.
Angular is a full framework with DI and built-in services.
Vue combines features of both but is more template-driven.
React uses JSX, while others use templating languages.
Example:
```
React: const element = <div>Hello</div>;
```

11. **What is the purpose of Error Boundaries?**
Error boundaries catch JavaScript errors in child components.
They prevent app crashes by displaying fallback UI.
Use componentDidCatch and getDerivedStateFromError.
Only class components can be error boundaries.
Example:
```
<ErrorBoundary><MyComponent /></ErrorBoundary>
```

12. **How do you handle WebSockets in React?**
Use native WebSocket API or libraries like **socket.io-client**.
Initiate connection in useEffect, clean up on unmount.
Manage messages using state or context.
Useful for chat, live notifications, etc.
Example:
```
useEffect(() => { const socket = new WebSocket(url); return () => socket.close(); }, []);
```

13. **How would you implement a design system in React?**
Create reusable UI components and define global styles.
Use tools like **Storybook**, **Styled Components**, or **Tailwind**.

Encapsulate themes, tokens, and responsive breakpoints.
Ensure accessibility and consistency.
Example:
```
<Button variant="primary">Submit</Button>
```

14. **How do you ensure accessibility (a11y) in React apps?**
Use semantic HTML tags (<nav>, <header>, etc.).
Add aria-* attributes and keyboard navigation.
Label form inputs properly and use accessible components.
Audit using tools like Lighthouse or aXe.
Example:
```
<button aria-label="Close modal">X</button>
```

15. **How do you integrate TypeScript in React apps?**
Use create-react-app --template typescript or add manually.
Define props and state with interfaces or types.
TS improves type safety, autocomplete, and maintainability.
Helps catch bugs during development.
Example:
```
interface Props { title: string; } const Header: React.FC<Props> = ({ title }) =>
<h1>{title}</h1>;
```

16. **How do you handle real-time data in React?**
Use WebSockets, Server-Sent Events (SSE), or polling.
Store updates in state or context.
Use useEffect for subscriptions and cleanups.
Libraries like **SWR** or **React Query** support real-time sync.
Example:
```
useEffect(() => { socket.on('message', setMessage); }, []);
```

17. **How do you manage complex form validation?**
Use libraries like **Formik** or **React Hook Form**.
Define schema with **Yup** or custom validators.
Support field-level and form-level validation.
Provide real-time feedback to users.
Example:
```
useForm({ resolver: yupResolver(schema) });
```

18. **What is Suspense and Concurrent Mode in React?**
Suspense handles lazy loading fallback UI.
Concurrent Mode (experimental) enables interruptible rendering.
Improves responsiveness for high-load UIs.
Requires features like startTransition, useDeferredValue.
Example:
```
<Suspense fallback={<Loader />}><LazyComp /></Suspense>
```

19. **How do you handle versioning and deployment of React apps?**
    Use Git for versioning, tag releases with semantic versioning.
    CI/CD tools (e.g., GitHub Actions, Vercel, Netlify) handle deployment.
    Configure environments for dev, staging, and production.
    Ensure cache busting with unique build hashes.
    Example:
    Deploy with: npm run build && firebase deploy

20. **What is the role of React DevTools in debugging?**
    React DevTools helps inspect component tree, props, and state.
    Track renders, hook values, and performance issues.
    Use Profiler tab to analyze rendering behavior.
    Great for tracking re-renders and context updates.
    Example:
    Highlight unnecessary re-renders in Profiler.

.NET Interview Questions (3–8 Years Experience)

1. **What are the differences between .NET Core and .NET Framework?**
    .NET Core is cross-platform and open-source.
    .NET Framework is Windows-only and more mature.
    .NET Core offers better performance and cloud support.
    It supports side-by-side deployments.
    Example:
    Use .NET Core to host APIs on Linux with Docker.

2. **How do you implement dependency injection in .NET Core?**
    Register services in Program.cs using AddScoped, AddTransient, or AddSingleton.
    Inject services via constructor.
    Built-in DI container is lightweight and extensible.
    Promotes loose coupling and testability.
    Example:

```
services.AddScoped<IUserService, UserService>();
```

3. **Explain the repository and unit of work patterns.**
    Repository handles data access logic for a specific entity.
    Unit of Work manages multiple repositories as a single transaction.
    They decouple business logic from data access.
    Improves testability and code maintenance.
    Example:
    _unitOfWork.UserRepository.Add(user); _unitOfWork.Complete();

4. **What is middleware in ASP.NET Core?**
    Middleware is software that handles requests/responses in the pipeline.
    They run in sequence and can short-circuit the request.
    Use app.UseMiddleware<>() or built-in like UseRouting().
    Common for logging, authentication, and error handling.
    Example:

```
app.UseMiddleware<RequestLoggingMiddleware>();
```

5. **How do you secure APIs using JWT?**
   Client sends token in Authorization header.
   Validate token using middleware (AddAuthentication, AddJwtBearer).
   Tokens contain claims for user identity and roles.
   Secret key is used for signing the token.
   Example:

```
options.TokenValidationParameters = new TokenValidationParameters { ValidateIssuerSigningKey = true };
```

6. **How do you handle exceptions globally in Web API?**
   Use UseExceptionHandler() or custom middleware.
   Log errors and return standardized error responses.
   Avoid exposing sensitive error details.
   Optionally use filters like ExceptionFilter.
   Example:

```
app.UseExceptionHandler("/error");
```

7. **What is Entity Framework Core and how is it different from EF6?**
   EF Core is lightweight, cross-platform, and open-source.
   EF6 is older and tied to .NET Framework.
   EF Core supports better performance, LINQ enhancements, and no tracking queries.
   EF Core lacks some features like lazy loading by default.
   Example:

```
var users = await context.Users.AsNoTracking().ToListAsync();
```

8. **What is model binding in Web API?**
   Model binding maps HTTP request data to action parameters.
   It supports query strings, route values, headers, and body.
   Uses attributes like [FromBody], [FromQuery], etc.
   Simplifies input parsing and validation.
   Example:

```
public IActionResult GetUser([FromQuery]int id)
```

9. **How do you implement caching in .NET Core?**
   Use IMemoryCache for in-memory caching.
   Register with DI and set cache expiration.
   Use IDistributedCache for distributed scenarios (e.g., Redis).
   Improves performance by reducing database hits.
   Example:

```
cache.Set("key", data, TimeSpan.FromMinutes(5));
```

10. **What are filters in ASP.NET Core?**
    Filters run before/after controller actions.
    Types: Authorization, Action, Exception, Result filters.
    Used for logging, validation, security, etc.

Can be global, controller, or method-level.

Example:

```
[ServiceFilter(typeof(LogActionFilter))]
```

11. **How do you configure appsettings.json?**

    Define settings in appsettings.json.
    Read using IConfiguration in Program.cs or services.
    Use section binding to map to strongly typed classes.
    Supports environment-specific files like appsettings.Production.json.

    Example:

```
Configuration.GetSection("AppSettings").Get<AppSettings>();
```

12. **What is async/await and how does it improve performance?**

    async/await enables non-blocking code execution.
    Frees up threads while awaiting I/O operations.
    Improves scalability for web APIs.
    Avoids thread starvation under high load.

    Example:

```
public async Task<IActionResult> GetUser() => Ok(await _repo.GetUserAsync());
```

13. **How do you use AutoMapper in .NET projects?**

    AutoMapper maps one object type to another.
    Configure profiles with CreateMap<Source, Destination>().
    Register in DI and inject IMapper.
    Reduces boilerplate code for DTO mapping.

    Example:

```
var dto = _mapper.Map<UserDto>(user);
```

14. **How do you log in ASP.NET Core applications?**

    Use built-in ILogger<T> interface.
    Supports providers like Console, File, Azure, Seq, etc.
    Configure logging levels in appsettings.json.
    Helps in tracking application flow and errors.

    Example:

```
_logger.LogInformation("User created successfully");
```

15. **What is Kestrel server?**

    Kestrel is the default cross-platform web server in ASP.NET Core.
    Lightweight, high-performance, built on libuv (now sockets).
    Used behind Nginx/IIS for production.
    Handles HTTP requests directly.

    Example:

    Run standalone with dotnet run using Kestrel.

16. **How do you perform unit testing in .NET Core?**

    Use **xUnit**, **NUnit**, or **MSTest**.
    Mock dependencies using Moq or similar libraries.

Write tests for services, controllers, and validation logic.
Use assertions to verify expected behavior.
Example:

```
Assert.Equal(expected, actual);
```

17. **What is the role of Startup.cs?**
    Startup.cs configures services and request pipeline.
    Contains ConfigureServices and Configure methods.
    Used to register dependencies and middleware.
    It's the application's entry point logic.
    Example:

```
services.AddControllers(); app.UseRouting();
```

18. **How do you perform database migrations using EF Core?**
    Use Add-Migration to scaffold migration scripts.
    Use Update-Database to apply them.
    Manages schema changes over time.
    Store migrations in version control.
    Example:

```
Add-Migration AddUserTable
```

19. **What is the role of IConfiguration and IOptions?**
    IConfiguration reads config values from various sources.
    IOptions<T> binds config sections to classes.
    Supports validation and reload-on-change.
    Used for accessing app settings in services.
    Example:

```
services.Configure<AppSettings>(Configuration.GetSection("AppSettings"));
```

20. **How do you implement role-based access control in .NET Core?**
    Assign roles to users during authentication.
    Use [Authorize(Roles = "Admin")] on controllers/actions.
    Roles stored in JWT claims or database.
    Use User.IsInRole("RoleName") for custom logic.
    Example:

```
[Authorize(Roles = "Manager")]
```

SQL Server Interview Questions (3–8 Years Experience)

1. **What are the types of indexes in SQL Server?**
   SQL Server supports Clustered, Non-Clustered, Unique, Filtered, XML, Spatial, and Full-Text indexes.
   Indexes improve query performance by speeding up data retrieval.
   Clustered indexes sort the actual data rows, while non-clustered maintain a separate structure.
   Filtered indexes apply to subsets of data.
   Example:

```
CREATE NONCLUSTERED INDEX IX_Users_Email ON Users(Email);
```

2. **What is the difference between clustered and non-clustered index?**
   Clustered index determines the physical order of rows.
   Only one clustered index allowed per table.
   Non-clustered index is a separate structure with pointers.
   Multiple non-clustered indexes are allowed.
   Example:

```
CREATE CLUSTERED INDEX IX_Employee_Id ON Employee(Id);
```

3. **How do you optimize slow running queries?**
   Use EXPLAIN or execution plans to identify bottlenecks.
   Create appropriate indexes on filters and joins.
   Avoid SELECT *, use only needed columns.
   Rewrite queries using CTEs or temp tables if needed.
   Example:

```
SELECT FirstName FROM Users WHERE LastName = 'Smith';
```

4. **What is a common table expression (CTE)?**
   CTE is a temporary result set used within a SELECT, INSERT, UPDATE, or DELETE.
   Defined using WITH keyword.
   Improves readability and supports recursion.
   Ideal for breaking complex queries.
   Example:

```
WITH CTE AS (SELECT * FROM Users WHERE Age > 30) SELECT * FROM CTE;
```

5. **How do you handle deadlocks in SQL Server?**
   Keep transactions short and access objects in a consistent order.
   Use TRY...CATCH for error handling.
   Use SET DEADLOCK_PRIORITY to control victim selection.
   Monitor using Extended Events or SQL Profiler.
   Example:

```
SET DEADLOCK_PRIORITY LOW;
```

6. **What are the ACID properties?**
   Atomicity: all or nothing execution.
   Consistency: maintains valid data state.
   Isolation: concurrent transactions don't affect each other.
   Durability: committed changes survive failures.
   Example:
   A successful bank transfer ensures both debit and credit happen.

7. **How do you implement transactions in SQL Server?**
   Use BEGIN TRANSACTION, COMMIT, and ROLLBACK.
   Ensure all operations succeed before committing.
   Rollback on error to maintain integrity.
   Can be nested within stored procedures.
   Example:

```
BEGIN TRAN
```

```
UPDATE Accounts SET Balance = Balance - 100 WHERE Id = 1;
COMMIT;
```

8. **What is the difference between DELETE and TRUNCATE?**
   DELETE logs each row and allows WHERE clause.
   TRUNCATE is faster and removes all rows without logging.
   TRUNCATE resets identity; DELETE does not.
   DELETE can be rolled back; TRUNCATE is less flexible.
   Example:

```
TRUNCATE TABLE Users;
```

9. **How do you perform backup and restore in SQL Server?**
   Use BACKUP DATABASE to create a backup file.
   Use RESTORE DATABASE to restore from backup.
   Can be done via SQL or SSMS GUI.
   Supports full, differential, and transaction log backups.
   Example:

```
BACKUP DATABASE MyDB TO DISK = 'C:\MyDB.bak';
```

10. **What is a stored procedure?**
    A stored procedure is a precompiled batch of SQL statements.
    It improves performance and reusability.
    Can accept input/output parameters.
    Helps in abstracting complex logic.
    Example:

```
CREATE PROCEDURE GetUsers AS SELECT * FROM Users;
```

11. **How do you write a trigger and when to use it?**
    Trigger is a special procedure that runs on table events.
    Used for auditing, enforcing rules, or automation.
    Fires on INSERT, UPDATE, or DELETE.
    Can be AFTER or INSTEAD OF.
    Example:

```
CREATE TRIGGER trgAfterInsert ON Users AFTER INSERT AS BEGIN PRINT 'Inserted'; END;
```

12. **What are isolation levels in SQL Server?**
    Controls how data is visible in concurrent transactions.
    Levels: Read Uncommitted, Read Committed, Repeatable Read, Snapshot, Serializable.
    Higher isolation = fewer concurrency issues but more locking.
    Set using SET TRANSACTION ISOLATION LEVEL.
    Example:

```
SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;
```

13. **How do you monitor performance in SQL Server?**
    Use tools like SQL Profiler, Extended Events, and DMVs.
    Monitor CPU, IO, memory, and query stats.
    Look for long-running queries and missing indexes.
    Use Activity Monitor in SSMS.
    Example:

```
SELECT * FROM sys.dm_exec_requests;
```

14. What is the use of SQL Profiler?

Captures and logs SQL Server events in real-time.
Used for debugging, performance tuning, and auditing.
Tracks queries, logins, and errors.
Can filter specific databases or users.
Example:
Track slow stored procedure calls for optimization.

15. What are execution plans and how do you analyze them?

Execution plans show how SQL Server executes queries.
Helps identify table scans, joins, and index usage.
Use Estimated or Actual plan in SSMS.
Look for expensive steps and missing index suggestions.
Example:
Click "Display Actual Execution Plan" in SSMS after running query.

16. **How do you prevent SQL injection?**

Use parameterized queries or stored procedures.
Avoid dynamic SQL built with string concatenation.
Validate and sanitize user input.
Use ORM tools that support safe queries.
Example:

```
SELECT * FROM Users WHERE Email = @Email;
```

17. What is normalization? Explain different normal forms.

Normalization organizes data to reduce redundancy.
1NF: Atomic columns.
2NF: No partial dependency.
3NF: No transitive dependency.
BCNF: Stronger form of 3NF.
Example:
Separate Customer and Orders into different tables with FK.

18. **What is a temp table vs table variable?**

Temp tables use # and exist in tempdb.
Table variables use DECLARE and are memory-based.
Temp tables support indexes and statistics.
Table variables have limited scope and performance.
Example:

```
DECLARE @Temp TABLE (Id INT); -- table variable
```

19. **What is the difference between INNER JOIN and OUTER JOIN?**

INNER JOIN returns only matching rows.
LEFT/RIGHT OUTER JOIN returns all rows from one table and matching from the other.
FULL OUTER JOIN returns all rows from both tables.

Used for combining data from multiple tables.

Example:

```
SELECT * FROM A LEFT JOIN B ON A.Id = B.Id;
```

20. **How do you handle pagination in SQL Server?**

Use OFFSET-FETCH with ORDER BY.

Efficient for large datasets.

Specify starting row and number of rows.

Supports dynamic pagination in web apps.

Example:

```
SELECT * FROM Users ORDER BY Id OFFSET 10 ROWS FETCH NEXT 10 ROWS ONLY;
```