Complete .NET Core Interview Questions & Answers Guide

For 2+ Years Experienced Developers

Table of Contents

C# & OOP Fundamentals

1. What is Language and Framework?

Answer:
- **Language (C#)**: A programming language with syntax and rules for writing code
- **Framework (.NET)**: A platform providing libraries, runtime, and tools for building

applications
- C# is the language; .NET is the framework that executes C# code

2. What is the use of public static void Main() / how does a .NET application run?

Answer:
- Entry point of any .NET application
- `public`: Accessible from CLR
- `static`: No instance needed to invoke
- `void`: Returns nothing
- CLR looks for Main() method to start execution

3. What is Garbage Collector and how does GC work?

Answer:
- Automatic memory management system
- **Generations**: Gen 0 (short-lived), Gen 1 (medium), Gen 2 (long-lived)
- **Process**: Mark → Sweep → Compact
- Runs automatically when Gen 0 is full or memory pressure exists
- Can be triggered manually using `GC.Collect()`

4. What are Access Modifiers?

Answer:
- `public`: Accessible everywhere
- `private`: Only within the same class
- `protected`: Within class and derived classes
- `internal`: Within same assembly
- `protected internal`: Same assembly OR derived classes
- `private protected`: Same assembly AND derived classes

5. What is Polymorphism?

Answer:
- **Compile-time (Overloading)**: Same method name, different parameters
- **Runtime (Overriding)**: Base class virtual methods overridden in derived class

```
// Overloading
public int Add(int a, int b) { return a + b; }
public double Add(double a, double b) { return a + b; }

// Overriding
public virtual void Display() { }
public override void Display() { }
```

6. What are Interface & Abstract classes? Explain with code example.

Answer:

```
// Interface - contract, multiple inheritance
public interface IVehicle {
    void Start(); // no implementation
    int Speed { get; set; }
}

// Abstract class - partial implementation
public abstract class Vehicle {
    public string Brand { get; set; }
    public abstract void Start(); // must override
    public virtual void Stop() { } // can override
}

// Key Differences:
// - Interface: all members public, no fields, multiple inheritance
// - Abstract: can have fields, access modifiers, single inheritance
```

7. Difference between var and object

Answer:

- `var`: Compile-time type inference, strongly typed
- `object`: Base type of all types, requires boxing/unboxing

```
var num = 10; // int at compile time
object obj = 10; // boxed, requires casting
```

8. const vs readonly vs static?

Answer:
- `const` : Compile-time constant, implicitly static
- `readonly` : Runtime constant, instance or static
- `static` : Shared across all instances

```
public const int MaxValue = 100; // compile-time
public readonly int Id; // set in constructor
public static int Count; // shared value
```

9. Boxing vs Unboxing?

Answer:
- **Boxing**: Converting value type to reference type (object)
- **Unboxing**: Converting boxed object back to value type

```
int val = 10;
object obj = val; // Boxing
int newVal = (int)obj; // Unboxing
```

10. What are Key features of C#?

Answer:
- Object-oriented
- Type-safe
- Garbage collection
- LINQ support
- Async/await
- Properties and indexers
- Delegates and events
- Generics
- Nullable types

11. Interface vs Abstract class?

Answer:

Interface	Abstract Class
Multiple inheritance	Single inheritance
No implementation (before C# 8)	Can have implementation
All members public	Access modifiers allowed
No fields	Can have fields
No constructor	Can have constructor

12. SOLID Principles?

Answer:

- **S**: Single Responsibility - One class, one purpose
- **O**: Open/Closed - Open for extension, closed for modification
- **L**: Liskov Substitution - Derived classes substitutable for base
- **I**: Interface Segregation - Multiple specific interfaces
- **D**: Dependency Inversion - Depend on abstractions, not concretions

13. Sealed classes?

Answer:

- Cannot be inherited
- Prevents class extension
- All structs are implicitly sealed

```
public sealed class FinalClass { }
// public class Derived : FinalClass { } // Error
```

.NET Core & Framework

14. .NET Framework vs .NET Core vs .NET 5+?

Answer:
- **.NET Framework**: Windows-only, legacy
- **.NET Core**: Cross-platform, modern, lightweight
- **.NET 5+**: Unified platform, successor to Core and Framework

15. CLR & CTS?

Answer:
- **CLR (Common Language Runtime)**: Execution environment, manages memory, security, exceptions
- **CTS (Common Type System)**: Defines types that can be used across .NET languages

16. IIS vs Kestrel?

Answer:
- **IIS**: Windows-only web server, full-featured
- **Kestrel**: Cross-platform, lightweight, edge server
- Best practice: Kestrel behind IIS/Nginx as reverse proxy

17. What are NuGet packages?

Answer:
- .NET package manager
- Contains compiled code (DLLs) and dependencies
- Installed via Package Manager Console or UI

```
Install-Package Newtonsoft.Json
```

18. Middlewares in .NET Core?

Answer:
- Components in request pipeline

- Execute in order
- Can short-circuit pipeline

```
app.Use(async (context, next) => {
    // Before
    await next();
    // After
});
```

19. Role of appsettings.json?

Answer:
- Application configuration file
- Environment-specific settings
- Accessed via IConfiguration

```
{
  "ConnectionStrings": {
    "DefaultConnection": "Server=..."
  }
}
```

20. Request pipeline?

Answer:
- Series of middleware components
- Each can handle request/response
- Order matters

```
app.UseAuthentication();
app.UseAuthorization();
app.UseEndpoints();
```

21. Use() vs UseMiddleware() vs Run()?

Answer:
- `Use()`: Inline middleware, can call next
- `UseMiddleware<T>()`: Class-based middleware
- `Run()`: Terminal middleware, doesn't call next

22. Map vs Use vs Run - Middleware registration methods?

Answer:
- **Map**: Branches pipeline based on path
- **Use**: Adds middleware that can call next
- **Run**: Terminal middleware

```
app.Map("/api", apiApp => { });
app.Use(async (ctx, next) => await next());
app.Run(async ctx => await ctx.Response.WriteAsync("End"));
```

23. Middleware Pipeline - How does request/response pipeline work?

Answer:
- Request flows through middleware in order
- Each middleware can:
- Process request before calling next
- Call next middleware
- Process response after next returns
- Short-circuit by not calling next

ASP.NET MVC / Web API

24. What is MVC?

Answer:
- **Model**: Data and business logic
- **View**: Presentation layer (UI)

- **Controller**: Handles requests, coordinates M&V
- Separation of concerns pattern

25. ViewBag vs ViewData vs TempData?

Answer:
- **ViewBag**: Dynamic, no casting, wrapper over ViewData
- **ViewData**: Dictionary, requires casting
- **TempData**: Survives redirect, uses session

```
ViewBag.Message = "Hello";
ViewData["Message"] = "Hello";
TempData["Message"] = "Hello"; // Survives redirect
```

26. Action filters?

Answer:
- Execute before/after action methods
- Types: Authorization, Action, Result, Exception

```
public class LogActionFilter : ActionFilterAttribute {
    public override void OnActionExecuting(ActionExecutingContext context) {
}
```

27. Routing?

Answer:
- URL pattern matching
- Conventional: `"{controller}/{action}/{id?}"`
- Attribute: `[Route("api/[controller]")]`

```
[HttpGet("users/{id:int}")]
public IActionResult GetUser(int id) { }
```

28. Web API vs MVC?

Answer:

Web API	MVC
Returns data (JSON/XML)	Returns views (HTML)
RESTful services	Web applications
No view state	View state management
Stateless	Can maintain state

29. What is CORS?

Answer:

- Cross-Origin Resource Sharing
- Browser security feature
- Allows/restricts cross-domain requests

```
services.AddCors(options => {
    options.AddPolicy("AllowAll",
        builder => builder.AllowAnyOrigin()
                          .AllowAnyMethod()
                          .AllowAnyHeader());
});
```

30. SOAP vs REST?

Answer:

SOAP	REST
Protocol	Architecture style
XML only	Multiple formats
WS-Security	HTTPS
Heavyweight	Lightweight
WSDL contract	No contract

31. What is Swagger?

Answer:
- API documentation tool
- Interactive API testing
- Auto-generates from code

```
services.AddSwaggerGen(c => {
    c.SwaggerDoc("v1", new OpenApiInfo { Title = "API", Version = "v1" });
});
```

32. Global exception handling?

Answer:

```
// Middleware approach
public class GlobalExceptionMiddleware {
    public async Task InvokeAsync(HttpContext context, RequestDelegate next)
        try {
            await next(context);
        } catch (Exception ex) {
            await HandleExceptionAsync(context, ex);
        }
    }
}
```

33. JWT authentication?

Answer:
- JSON Web Tokens for stateless authentication
- Contains header, payload, signature

```
services.AddAuthentication(JwtBearerDefaults.AuthenticationScheme)
    .AddJwtBearer(options => {
        options.TokenValidationParameters = new TokenValidationParameters {
            ValidateIssuer = true,
            ValidateAudience = true,
            ValidateLifetime = true
        };
    });
```

Entity Framework

34. EF Core vs EF6?

Answer:
- **EF Core**: Cross-platform, lightweight, better performance
- **EF6**: Windows-only, more features, legacy

35. Code-first vs DB-first?

Answer:
- **Code-first**: Create models → generate database
- **DB-first**: Existing database → generate models

```
// Code-first
public class Product {
    public int Id { get; set; }
    public string Name { get; set; }
}
```

36. Lazy vs Eager vs Explicit loading?

Answer:

```
// Lazy (loads on access)
var blog = context.Blogs.First();
var posts = blog.Posts; // Loaded here

// Eager (loads immediately)
var blogs = context.Blogs.Include(b => b.Posts);

// Explicit (load when needed)
context.Entry(blog).Collection(b => b.Posts).Load();
```

37. What are migrations?

Answer:
- Version control for database schema
- Track and apply database changes

```
Add-Migration InitialCreate
Update-Database
```

38. Raw SQL in EF?

Answer:

```
// Query
var blogs = context.Blogs
    .FromSqlRaw("SELECT * FROM Blogs WHERE Rating > {0}", rating);

// Command
context.Database.ExecuteSqlRaw("UPDATE Blogs SET Rating = 5");
```

39. DbContext vs DbSet?

Answer:
- **DbContext**: Database session, unit of work
- **DbSet**: Table representation, queryable collection

```
public class AppDbContext : DbContext {
    public DbSet<Product> Products { get; set; }
}
```

40. What is Entity Framework & ORM?

Answer:
- **ORM**: Object-Relational Mapping
- Maps database tables to C# objects
- Eliminates manual SQL writing
- Provides LINQ queries

Async & Multithreading

41. Task vs Thread vs async/await?

Answer:
- **Thread**: OS-level, expensive, low-level
- **Task**: Higher abstraction, uses ThreadPool
- **async/await**: Syntactic sugar for async operations

```
// Thread
Thread thread = new Thread(() => { });

// Task
Task.Run(() => { });

// async/await
public async Task<string> GetDataAsync() {
    return await httpClient.GetStringAsync(url);
}
```

42. ThreadPool?

Answer:
- Manages pool of reusable threads
- Reduces overhead of thread creation
- Used by Task.Run internally

```
ThreadPool.QueueUserWorkItem(_ => {
    // Work here
});
```

43. lock keyword & race conditions?

Answer:
- Prevents concurrent access to shared resources
- Prevents race conditions

```
private readonly object _lock = new object();
lock (_lock) {
    // Critical section
}
```

44. ConcurrentDictionary vs Dictionary?

Answer:
- **Dictionary**: Not thread-safe
- **ConcurrentDictionary**: Thread-safe operations

```
ConcurrentDictionary<int, string> dict = new();
dict.TryAdd(1, "value");
```

45. Parallel.ForEach?

Answer:
- Parallel execution of iterations
- Uses multiple threads from ThreadPool

```
Parallel.ForEach(items, item => {
    ProcessItem(item);
});
```

46. CancellationToken?

Answer:

- Cooperative cancellation mechanism

```
public async Task DoWorkAsync(CancellationToken ct) {
    while (!ct.IsCancellationRequested) {
        await Task.Delay(1000, ct);
    }
}
```

47. What is a deadlock?

Answer:

- Two or more threads waiting for each other
- Common with nested locks or .Result on async

```
// Avoid: can cause deadlock
var result = GetDataAsync().Result;

// Use: proper async
var result = await GetDataAsync();
```

48. What is Asynchronous Programming?

Answer:

- Non-blocking execution
- Improves scalability and responsiveness
- Uses async/await keywords
- Returns Task/Task

49. What is Event-Driven vs Data-Driven mechanism?

Answer:
- **Event-Driven**: Actions triggered by events (button click)
- **Data-Driven**: Actions based on data changes (data binding)

Design Patterns

50. What is a Design Pattern? Which ones have you used?

Answer:
Common patterns:
- **Singleton**: Single instance
- **Repository**: Data access abstraction
- **Factory**: Object creation
- **Dependency Injection**: Loose coupling
- **Observer**: Event notification

51. What is Dependency Injection? Provide sample code.

Answer:
- Inversion of Control technique
- Dependencies provided externally

```
public interface IEmailService {
    void Send(string message);
}

public class NotificationService {
    private readonly IEmailService _emailService;

    public NotificationService(IEmailService emailService) {
        _emailService = emailService;
    }
}

// Registration
services.AddScoped<IEmailService, EmailService>();
```

52. Repository & Unit of Work?

Answer:

```
// Repository
public interface IRepository<T> {
    Task<T> GetByIdAsync(int id);
    Task<IEnumerable<T>> GetAllAsync();
    Task AddAsync(T entity);
}

// Unit of Work
public interface IUnitOfWork {
    IRepository<Product> Products { get; }
    Task<int> CommitAsync();
}
```

53. Singleton vs Factory vs DI?

Answer:

- **Singleton**: Ensures single instance

- **Factory**: Creates objects without specifying exact class
- **DI**: Provides dependencies from external source

```
// Singleton
public sealed class Singleton {
    private static readonly Singleton instance = new();
    public static Singleton Instance => instance;
}

// Factory
public interface IVehicleFactory {
    IVehicle CreateVehicle(string type);
}
```

54. Mediator pattern?

Answer:
- Reduces coupling between components
- Central communication hub
- Used in CQRS with MediatR

```
public interface IMediator {
    Task Send(IRequest request);
}
```

55. Observer pattern?

Answer:
- Notify multiple objects about state changes
- Publisher-Subscriber pattern

```
public interface IObserver {
    void Update(string message);
}

public class Subject {
    private List<IObserver> observers = new();
    public void Attach(IObserver observer) => observers.Add(observer);
    public void Notify() => observers.ForEach(o => o.Update("Changed"));
}
```

Microservices

56. Monolith vs Microservices?

Answer:

Monolith	Microservices
Single codebase	Multiple services
Simple deployment	Complex orchestration
Shared database	Service-owned data
Vertical scaling	Horizontal scaling
Single tech stack	Polyglot

57. What is gRPC?

Answer:

- High-performance RPC framework
- Uses Protocol Buffers
- Bi-directional streaming
- Better than REST for service-to-service

```
public class GreeterService : Greeter.GreeterBase {
    public override Task<HelloReply> SayHello(HelloRequest request, ServerCal
        return Task.FromResult(new HelloReply { Message = "Hello " + request.
    }
}
```

58. Docker & deploying .NET apps?

Answer:
- Containerization for consistent deployment
- Dockerfile example:

```
FROM mcr.microsoft.com/dotnet/aspnet:8.0
WORKDIR /app
COPY . .
ENTRYPOINT ["dotnet", "MyApp.dll"]
```

59. CI/CD pipeline basics?

Answer:
- **CI**: Continuous Integration - automated build/test
- **CD**: Continuous Deployment - automated release
- Tools: Azure DevOps, Jenkins, GitHub Actions

60. Azure Functions?

Answer:
- Serverless compute service
- Event-driven execution
- Auto-scaling
- Pay per execution

```
[FunctionName("HttpTrigger")]
public static async Task<IActionResult> Run(
    [HttpTrigger(AuthorizationLevel.Function, "get")] HttpRequest req) {
    return new OkObjectResult("Hello");
}
```

61. OAuth2 / JWT in microservices?

Answer:
- Centralized authentication server
- JWT tokens for service communication
- API Gateway validates tokens
- Services trust validated requests

Performance & Security

62. How to implement CORS in .NET Core?

Answer:

```
// Startup.cs
services.AddCors(options => {
    options.AddPolicy("MyCorsPolicy",
        builder => builder
            .WithOrigins("https://example.com")
            .AllowAnyMethod()
            .AllowAnyHeader());
});

app.UseCors("MyCorsPolicy");
```

63. How to implement Authentication & Authorization?

Answer:

```
// Authentication
services.AddAuthentication(JwtBearerDefaults.AuthenticationScheme)
    .AddJwtBearer(options => { });

// Authorization
services.AddAuthorization(options => {
    options.AddPolicy("AdminOnly", policy =>
        policy.RequireClaim("role", "admin"));
});

[Authorize(Policy = "AdminOnly")]
public IActionResult AdminAction() { }
```

64. How is Session handled in .NET Core?

Answer:

```
// Configure
services.AddSession(options => {
    options.IdleTimeout = TimeSpan.FromMinutes(30);
});

app.UseSession();

// Use
HttpContext.Session.SetString("key", "value");
var value = HttpContext.Session.GetString("key");
```

65. App performance tuning in .NET?

Answer:

- Use async/await properly
- Implement caching (Memory, Redis)
- Database query optimization
- Connection pooling
- Minimize allocations

- Use StringBuilder for string concatenation
- Response compression

66. Memory leaks & GC optimization?

Answer:
- Dispose IDisposable objects
- Avoid circular references
- Use weak references when appropriate
- Profile with dotMemory/PerfView
- Configure GC mode (Server vs Workstation)

67. Caching strategies?

Answer:

```
// In-memory
services.AddMemoryCache();

// Distributed (Redis)
services.AddStackExchangeRedisCache(options => {
    options.Configuration = "localhost";
});

// Usage
public class Service {
    private readonly IMemoryCache _cache;

    public async Task<string> GetDataAsync(string key) {
        return await _cache.GetOrCreateAsync(key, async entry => {
            entry.SlidingExpiration = TimeSpan.FromMinutes(5);
            return await FetchDataAsync();
        });
    }
}
```

68. Preventing CSRF & XSS?

Answer:

- **CSRF**: Use anti-forgery tokens
- **XSS**: HTML encode output, validate input

```
// CSRF
services.AddAntiforgery();
[ValidateAntiForgeryToken]
public IActionResult Submit() { }

// XSS
@Html.Encode(userInput)
```

69. Encryption in C#?

Answer:

```
using System.Security.Cryptography;

// AES encryption
public static string Encrypt(string plainText, string key) {
    using var aes = Aes.Create();
    aes.Key = Encoding.UTF8.GetBytes(key);
    aes.GenerateIV();

    var encryptor = aes.CreateEncryptor();
    var encrypted = encryptor.TransformFinalBlock(
        Encoding.UTF8.GetBytes(plainText), 0, plainText.Length);

    return Convert.ToBase64String(encrypted);
}
```

70. How do you secure connection strings, keys, passwords in .NET Core?

Answer:
- Use User Secrets in development
- Azure Key Vault in production
- Environment variables
- Never commit to source control

```
// User Secrets
dotnet user-secrets set "ApiKey" "secret-value"

// Azure Key Vault
services.AddAzureKeyVault();
```

71. What performance improvements have you made in your projects?

Answer:
Common improvements:
- Implemented caching layers
- Optimized database queries (indexes, stored procedures)
- Used async/await throughout
- Implemented pagination
- Lazy loading for heavy operations
- CDN for static assets
- Response compression

Testing

72. What is unit testing?

Answer:
- Testing individual units in isolation
- Use mocking for dependencies
- Should be fast and repeatable

```
[Test]
public void Add_TwoNumbers_ReturnsSum() {
    var calculator = new Calculator();
    var result = calculator.Add(2, 3);
    Assert.AreEqual(5, result);
}
```

73. Moq framework?

Answer:
- Mocking framework for unit tests

```
var mockService = new Mock<IEmailService>();
mockService.Setup(x => x.Send(It.IsAny<string>()))
           .Returns(true);

var service = new NotificationService(mockService.Object);
```

74. TDD basics?

Answer:
- Test-Driven Development
- Red → Green → Refactor cycle
1. Write failing test
2. Write minimal code to pass
3. Refactor

75. Debugging tools in VS?

Answer:
- Breakpoints and conditional breakpoints
- Watch window
- Immediate window
- Call stack
- Diagnostic tools
- IntelliTrace

76. Exception handling in tests?

Answer:

```
[Test]
public void Divide_ByZero_ThrowsException() {
    Assert.Throws<DivideByZeroException>(() => {
        calculator.Divide(10, 0);
    });
}
```

77. Logging in .NET Core?

Answer:

```
services.AddLogging(config => {
    config.AddConsole();
    config.AddDebug();
});

public class Service {
    private readonly ILogger<Service> _logger;

    public Service(ILogger<Service> logger) {
        _logger = logger;
    }

    public void DoWork() {
        _logger.LogInformation("Starting work");
    }
}
```

SQL & Database

78. What is REST and why do we use it?

Answer:
- REpresentational State Transfer
- Architectural style for APIs
- Stateless, cacheable, uniform interface
- Uses HTTP methods (GET, POST, PUT, DELETE)
- JSON/XML data format

79. How to update partial data using REST?

Answer:
- Use PATCH method
- JSON Patch format

```
[HttpPatch("{id}")]
public IActionResult PatchUser(int id, [FromBody] JsonPatchDocument<User> pat
    var user = GetUser(id);
    patch.ApplyTo(user);
    return Ok(user);
}
```

80. What is Normalization?

Answer:
- Organizing data to reduce redundancy
- **1NF**: Atomic values, unique rows
- **2NF**: 1NF + no partial dependencies
- **3NF**: 2NF + no transitive dependencies

81. What are 2NF and 3NF?

Answer:
- **2NF**: All non-key attributes fully dependent on primary key
- **3NF**: No non-key attribute depends on another non-key attribute

82. Difference between CTE and Temp Table + pros/cons

Answer:

CTE	Temp Table
Scope: Single query	Scope: Session
No indexes	Can have indexes
Not stored	Stored in tempdb
Better for recursion	Better for multiple uses

```
-- CTE
WITH CTE AS (SELECT * FROM Users WHERE Age > 18)
SELECT * FROM CTE;

-- Temp Table
CREATE TABLE #TempUsers (Id INT, Name VARCHAR(50));
```

83. Where to write business logic — Stored Procedure or C#? Why?

Answer:

C# preferred:

- Version control
- Unit testing
- Debugging tools
- Reusability
- Platform independence

Stored Procedures for:

- Complex data operations
- Performance-critical queries
- Reducing network traffic

84. What is LINQ and its benefits?

Answer:

- Language Integrated Query
- Type-safe queries

- IntelliSense support
- Deferred execution

```
var adults = users.Where(u => u.Age > 18)
                  .OrderBy(u => u.Name)
                  .Select(u => u.Name);
```

85. Difference between IEnumerable and IQueryable

Answer:

IEnumerable	IQueryable
In-memory queries	Database queries
LINQ to Objects	LINQ to SQL/EF
Client-side filtering	Server-side filtering
Immediate execution	Deferred execution

86. What is ACID?

Answer:

- **Atomicity**: All or nothing
- **Consistency**: Valid state transitions
- **Isolation**: Concurrent transactions don't interfere
- **Durability**: Committed data persists

87. Indexes & performance?

Answer:

- Speed up data retrieval
- Types: Clustered, Non-clustered, Composite
- Trade-off: Faster reads, slower writes

```
CREATE INDEX idx_users_email ON Users(Email);
```

88. Types of JOINs?

Answer:
- **INNER**: Matching records only
- **LEFT**: All from left + matches
- **RIGHT**: All from right + matches
- **FULL OUTER**: All from both
- **CROSS**: Cartesian product

89. Normalization vs Denormalization?

Answer:
- **Normalization**: Reduces redundancy, ensures consistency
- **Denormalization**: Improves read performance, adds redundancy

90. Preventing SQL Injection?

Answer:
- Use parameterized queries
- Stored procedures
- Input validation
- Never concatenate SQL strings

```
cmd.CommandText = "SELECT * FROM Users WHERE Id = @id";
cmd.Parameters.AddWithValue("@id", userId);
```

91. Query optimization?

Answer:
- Use appropriate indexes
- Avoid SELECT *
- Use EXISTS instead of IN
- Optimize JOINs
- Query execution plan analysis
- Update statistics

92. SQL vs NoSQL?

Answer:

SQL	NoSQL
Structured data	Flexible schema
ACID compliance	BASE (Eventually consistent)
Vertical scaling	Horizontal scaling
Complex queries	Simple queries
Relational	Document/Key-Value/Graph

93. Clustered vs Non-Clustered Indexes?

Answer:

- **Clustered**:
- One per table
- Determines physical order
- Leaf nodes contain data
- **Non-Clustered**:
- Multiple per table
- Separate structure
- Leaf nodes contain pointers

94. Materialized vs Non-Materialized Views?

Answer:

- **Materialized**:
- Stored physically
- Faster reads
- Requires refresh
- Uses storage
- **Non-Materialized (Standard)**:
- Virtual table
- Always current
- Slower performance
- No storage

95. Cascade Delete?

Answer:
- Automatically deletes related records
- Risks: Unintended data loss

```
ALTER TABLE Orders
ADD CONSTRAINT FK_Order_Customer
FOREIGN KEY (CustomerId) REFERENCES Customers(Id)
ON DELETE CASCADE;
```

Advanced Topics

96. Interoperability - How does .NET code work with unmanaged code?

Answer:
- **P/Invoke**: Call native DLLs
- **COM Interop**: Work with COM components
- **C++/CLI**: Managed wrapper

```
[DllImport("user32.dll")]
static extern int MessageBox(IntPtr hWnd, string text, string caption, uint t
```

97. Static members in Abstract classes?

Answer:
- Yes, abstract classes can have static members
- Static members belong to type, not instance
- Cannot be overridden

```
public abstract class Base {
    public static int Count { get; set; }
    public abstract void Method();
}
```

Network Programming Topics (from the LinkedIn post)

98. Load balancing algorithms

Answer:
Common algorithms:
- **Round Robin**: Sequential distribution
- **Least Connections**: Routes to server with fewest connections
- **IP Hash**: Based on client IP
- **Weighted**: Based on server capacity

99. Handling multiple concurrent clients

Answer:
- Use async/await pattern
- ThreadPool for CPU-bound work
- Connection pooling
- Rate limiting

```
public async Task HandleClientAsync(TcpClient client) {
    using var stream = client.GetStream();
    var buffer = new byte[1024];
    while (client.Connected) {
        var bytesRead = await stream.ReadAsync(buffer, 0, buffer.Length);
        // Process data
    }
}
```

100. HTTP Protocol handling

Answer:
- Request/Response model
- Methods: GET, POST, PUT, DELETE, PATCH
- Status codes: 2xx (Success), 4xx (Client error), 5xx (Server error)
- Headers for metadata

101. Building REST APIs

Answer:
- Resource-based URLs
- HTTP methods for operations
- Stateless communication
- JSON/XML responses

```
[ApiController]
[Route("api/[controller]")]
public class UsersController : ControllerBase {
    [HttpGet("{id}")]
    public async Task<IActionResult> Get(int id) {
        return Ok(await GetUserAsync(id));
    }
}
```

102. TCP and UDP protocols

Answer:
- **TCP**: Reliable, ordered, connection-oriented
- **UDP**: Fast, connectionless, no guarantee

```
// TCP
TcpListener listener = new TcpListener(IPAddress.Any, 8080);

// UDP
UdpClient udpClient = new UdpClient(8080);
```

103. DNS Protocol

Answer:
- Resolves domain names to IP addresses
- Uses UDP port 53 (TCP for large responses)

```
var hostEntry = await Dns.GetHostEntryAsync("example.com");
var ipAddress = hostEntry.AddressList[0];
```

104. ICMP Protocol (ping, traceroute)

Answer:
- Internet Control Message Protocol
- Used for diagnostics

```
using var ping = new Ping();
var reply = await ping.SendPingAsync("example.com");
Console.WriteLine($"RTT: {reply.RoundtripTime}ms");
```

105. NTP Protocol

Answer:
- Network Time Protocol
- Synchronizes clocks across network
- Uses UDP port 123

Summary

This comprehensive guide covers 105+ unique interview questions for .NET developers with 2+ years of experience. Focus areas include:

1. **Fundamentals**: Strong understanding of C#, OOP, and .NET architecture
2. **Modern Development**: .NET Core/5+, async programming, dependency injection
3. **Web Development**: MVC, Web API, REST principles

4. **Data Access**: Entity Framework, SQL optimization
5. **Architecture**: Microservices, design patterns, clean architecture
6. **DevOps**: Docker, CI/CD, cloud services
7. **Performance**: Caching, optimization, security best practices
8. **Testing**: Unit testing, TDD, mocking frameworks

Interview Preparation Tips:

- Practice coding implementations
- Understand concepts, don't memorize
- Be ready to explain your project experiences
- Know the trade-offs and when to use what
- Stay updated with latest .NET versions

Good luck with your interviews!