অবশ্যই! নিচে **EF Core এর কিছু Advanced Level Interview Questions and Answers** বাংলায় ব্যাখ্যা করছি। এগুলো একজন **Lead বা Senior Software Engineer** ইন্টারভিউতে উপকারে আসবে।

**✅ EF Core Advanced Level Interview Questions (বাংলা ভার্সন)**

**১. Change Tracker কী এবং এটি কীভাবে কাজ করে?**

**উত্তর:**  
Change Tracker হলো EF Core এর একটি কম্পোনেন্ট, যা DbContext-এর মধ্যে থাকা সমস্ত entity-র অবস্থা (state) ট্র্যাক করে — যেমন: Added, Modified, Deleted, বা Unchanged।  
যখন SaveChanges() কল করা হয়, তখন Change Tracker দেখে কোন entity-র কি পরিবর্তন হয়েছে এবং সেই অনুযায়ী ডাটাবেজে SQL Command রান করে।

🔹 উদাহরণ:

var student = context.Students.Find(1);

student.Name = "Rahim";

context.SaveChanges();

এখানে Change Tracker বুঝে ফেলে যে Name পরিবর্তন হয়েছে, তাই শুধু সেই কলামের জন্য UPDATE রান হবে।

**২. Concurrency Handling EF Core-এ কিভাবে করেন?**

**উত্তর:**  
EF Core-এ Optimistic Concurrency ব্যবহার করা হয়। এজন্য আমরা একটি RowVersion বা Timestamp কলাম ব্যবহার করি।

🔹 উদাহরণ:

[Timestamp]

public byte[] RowVersion { get; set; }

যদি দুটি ইউজার একসাথে একই ডাটা আপডেট করতে চায়, তাহলে দ্বিতীয়জনের ক্ষেত্রে concurrency conflict হবে।

🔹 কাস্টম exception হ্যান্ডল করতে পারেন:

try {

context.SaveChanges();

}

catch (DbUpdateConcurrencyException ex) {

// হ্যান্ডল করুন

}

**৩. Global Query Filters কী এবং কোথায় ব্যবহার হয়?**

**উত্তর:**  
Global Query Filter হলো এমন এক ধরনের LINQ ফিল্টার, যেটা কোনো একটি Entity-র উপরে **সবসময়** প্রয়োগ হয়।

🔹 উদাহরণ: Soft Delete বা Multi-Tenant অ্যাপ্লিকেশনে।

modelBuilder.Entity<Product>().HasQueryFilter(p => !p.IsDeleted);

এতে করে প্রতিবার context.Products কোয়েরি করলেই IsDeleted == false অটোমেটিক্যালি যোগ হবে।

**৪. EF Core-এ Value Conversions কী?**

**উত্তর:**  
Value Converter ব্যবহার করে আপনি C# প্রপার্টিকে ডাটাবেজে ভিন্নভাবে সংরক্ষণ করতে পারেন।

🔹 উদাহরণ: enum কে string হিসেবে সেভ করা।

modelBuilder.Entity<User>()

.Property(u => u.Status)

.HasConversion<string>();

এতে করে Enum UserStatus.Active ডাটাবেজে “Active” হিসেবে সংরক্ষিত হবে।

**৫. Shadow Properties কী?**

**উত্তর:**  
Shadow Properties এমন প্রপার্টি যেগুলো আপনার C# ক্লাসে নেই, কিন্তু EF Core মডেলে আছে এবং ডাটাবেজে ম্যানেজ করে।

🔹 উদাহরণ:

modelBuilder.Entity<Product>()

.Property<DateTime>("CreatedDate");

এখানে CreatedDate ক্লাসে নেই, কিন্তু ডাটাবেজে থাকবে এবং ট্র্যাক হবে।

**৬. Split Queries বনাম Single Query**

**উত্তর:**

* **Single Query:** একবারেই সকল রিলেটেড ডাটা লোড করে, কিন্তু বড় join এবং cartesian explosion-এর ঝুঁকি থাকে।
* **Split Query:** মূল টেবিল এবং রিলেটেড টেবিল আলাদা করে আলাদা query করে, performance ভালো হতে পারে।

🔹 উদাহরণ:

context.Blogs.Include(b => b.Posts).AsSplitQuery();

**৭. EF Core-এ Table-per-Hierarchy (TPH) vs Table-per-Type (TPT)**

**উত্তর:**

* **TPH:** সব ইনহেরিটেড ক্লাস এক টেবিলেই সেভ হয়।
* **TPT:** প্রতিটি সাব-ক্লাস আলাদা টেবিলে সেভ হয়।

🔹 উদাহরণ TPH:

modelBuilder.Entity<BaseClass>()

.HasDiscriminator<string>("Type")

.HasValue<ChildA>("A")

.HasValue<ChildB>("B");

TPH পারফর্মেন্স ভালো হলেও ডাটার রিডেবিলিটি কম, আর TPT রিলেশনাল ডিজাইনের জন্য ভালো, কিন্তু পারফর্মেন্স কম।

**৮. Compiled Query কী এবং কখন ব্যবহার করবেন?**

**উত্তর:**  
EF Core-এ LINQ কোয়েরিগুলো রান টাইমে SQL-এ রূপান্তরিত হয়। বারবার একই কোয়েরি চালালে পারফর্মেন্স কমে।  
**Compiled Query** ব্যবহার করলে কোয়েরি প্রি-কম্পাইল হয় ও ভালো পারফর্মেন্স দেয়।

🔹 উদাহরণ:

static readonly Func<AppDbContext, int, Product> \_getProduct =

EF.CompileQuery((AppDbContext ctx, int id) =>

ctx.Products.Single(p => p.Id == id));

**৯. EF Core-এ Interceptor কী এবং কীভাবে কাজ করে?**

**উত্তর:**  
Interceptor হলো এমন এক ফিচার, যা ডাটাবেজ অপারেশনের আগে-পরে আপনার কোড চালাতে দেয়।  
যেমন: লগিং, পারফরমেন্স মনিটরিং, সিকিউরিটি চেক।

🔹 উদাহরণ: CommandInterceptor ব্যবহার করে SQL লগিং:

public class MyInterceptor : DbCommandInterceptor

{

public override InterceptionResult<DbDataReader> ReaderExecuting(

DbCommand command, CommandEventData eventData, InterceptionResult<DbDataReader> result)

{

Console.WriteLine(command.CommandText);

return base.ReaderExecuting(command, eventData, result);

}

}

**১০. Batching in EF Core কী?**

**উত্তর:**  
EF Core একাধিক INSERT, UPDATE, DELETE অপারেশনকে একসাথে ব্যাচ করে রান করতে পারে, যাতে SQL সার্ভারে রাউন্ড ট্রিপ কম হয়।

🔹 এটা SaveChanges() এর মাধ্যমে অটোমেটিক হয়।

**উদাহরণ:** একসাথে ৫টি নতুন রেকর্ড যোগ করলে — EF Core একটি বড় SQL কমান্ড তৈরি করে।

**➕ আরও জানতে চান?**

* Soft Delete কাস্টম ইমপ্লিমেন্টেশন?
* Owned Entity?
* Cosmos DB অথবা অন্য NoSQL সাপোর্ট?
* Performance tuning best practices?

চমৎকার! এখন আমি EF Core-এর আরও **Advanced Real-Life Scenario Based Interview Questions & Answers** বাংলায় দিচ্ছি। এগুলো একজন Lead/Senior Software Engineer-এর ইন্টারভিউতে খুবই গুরুত্বপূর্ণ হতে পারে।

**✅ Real-Life Scenario Based EF Core Advanced Interview Questions (বাংলা ভার্সন)**

**🔶 প্রশ্ন ১: আপনার অ্যাপ্লিকেশনে Multi-Tenant ফিচার আছে। আপনি কীভাবে EF Core-এ প্রতিটি টেন্যান্টের জন্য আলাদা ডাটা ফিল্টার করবেন?**

**উত্তর:**  
এই ধরনের সিচুয়েশনে আমরা **Global Query Filter** ব্যবহার করতে পারি, যাতে করে প্রতিটি DbContext-এর সাথে টেন্যান্ট আইডি বাই ডিফল্ট ফিল্টার হয়ে যায়।

🔹 উদাহরণ:

public class ApplicationDbContext : DbContext

{

private readonly Guid \_tenantId;

public ApplicationDbContext(Guid tenantId)

{

\_tenantId = tenantId;

}

protected override void OnModelCreating(ModelBuilder modelBuilder)

{

modelBuilder.Entity<Customer>().HasQueryFilter(c => c.TenantId == \_tenantId);

}

}

এভাবে সব কোয়েরিতে টেন্যান্ট ফিল্টার অটো অ্যাপ্লাই হবে।

**🔶 প্রশ্ন ২: আপনি Soft Delete ইমপ্লিমেন্ট করতে চান যাতে ডাটা ডিলিট না হয়ে চিহ্নিত হয়। কীভাবে করবেন?**

**উত্তর:**

* প্রথমে মডেল ক্লাসে একটি IsDeleted ফ্ল্যাগ রাখবেন।
* এরপর Global Query Filter দিয়ে সেটি অটো ফিল্টার করবেন।
* আর override SaveChanges() করে “Delete” এর সময় IsDeleted = true সেট করবেন।

🔹 মডেল:

public class Product

{

public int Id { get; set; }

public string Name { get; set; }

public bool IsDeleted { get; set; }

}

🔹 Global Filter:

modelBuilder.Entity<Product>().HasQueryFilter(p => !p.IsDeleted);

🔹 Soft Delete Logic:

public override int SaveChanges()

{

foreach (var entry in ChangeTracker.Entries()

.Where(e => e.State == EntityState.Deleted))

{

entry.State = EntityState.Modified;

entry.CurrentValues["IsDeleted"] = true;

}

return base.SaveChanges();

}

**🔶 প্রশ্ন ৩: একটি ট্রানজেকশন দরকার যেখানে একাধিক SaveChanges() একসাথে কমিট করতে চান। কীভাবে করবেন?**

**উত্তর:**

EF Core-এ IDbContextTransaction ব্যবহার করে Transaction Control করা যায়।

🔹 উদাহরণ:

using var transaction = context.Database.BeginTransaction();

try

{

context.Orders.Add(new Order { ... });

context.SaveChanges();

context.Payments.Add(new Payment { ... });

context.SaveChanges();

transaction.Commit();

}

catch

{

transaction.Rollback();

throw;

}

এইভাবে আপনি একাধিক DbContext অপারেশন একসাথে কমিট বা রোলব্যাক করতে পারবেন।

**🔶 প্রশ্ন ৪: আপনি চাইছেন প্রতিটি Insert/Update/Delete অপারেশনের লগ সংরক্ষণ করতে। কীভাবে করবেন?**

**উত্তর:**

এই কাজের জন্য Interceptor অথবা SaveChanges() override করে Audit Logs তৈরি করা যায়।

🔹 উদাহরণ:

public override int SaveChanges()

{

var auditEntries = new List<AuditLog>();

foreach (var entry in ChangeTracker.Entries())

{

if (entry.State == EntityState.Modified || entry.State == EntityState.Added || entry.State == EntityState.Deleted)

{

var audit = new AuditLog

{

TableName = entry.Entity.GetType().Name,

Action = entry.State.ToString(),

ChangedAt = DateTime.Now

};

auditEntries.Add(audit);

}

}

AuditLogs.AddRange(auditEntries);

return base.SaveChanges();

}

**🔶 প্রশ্ন ৫: আপনি একটি বড় রিলেটেড ডাটা (Master-Detail) লোড করতে চান কিন্তু Memory/Performance সমস্যা হচ্ছে। কীভাবে করবেন?**

**উত্তর:**

এই ক্ষেত্রে AsSplitQuery() ব্যবহার করতে হবে।

🔹 উদাহরণ:

var orders = context.Orders

.Include(o => o.OrderItems)

.AsSplitQuery()

.ToList();

এতে করে Orders এবং OrderItems আলাদা SQL query দিয়ে আনবে, ফলে memory-efficient হয়।

**🔶 প্রশ্ন ৬: আপনার ক্লাসে একটি Complex Type আছে, যেটা DB-তে আলাদা table হবে না, বরং মূল table-এ embedded থাকবে। কীভাবে করবেন?**

**উত্তর:**

এটা **Owned Entity Types** দিয়ে করা যায়।

🔹 উদাহরণ:

public class Address

{

public string City { get; set; }

public string Country { get; set; }

}

public class Customer

{

public int Id { get; set; }

public Address Address { get; set; }

}

🔹 DbContext:

modelBuilder.Entity<Customer>().OwnsOne(c => c.Address);

এতে Customer table-এ City এবং Country কলাম হিসেবে থাকবে।

**🔶 প্রশ্ন ৭: কিভাবে আপনি নির্দিষ্ট DbContext অপারেশনের জন্য SQL Statement দেখতে পারেন?**

**উত্তর:**

Logger ব্যবহার করে অথবা Interceptor দিয়ে SQL Log বের করা যায়।

🔹 উদাহরণ (Logger দিয়ে):

services.AddDbContext<AppDbContext>(options =>

options.UseSqlServer(connStr)

.LogTo(Console.WriteLine, LogLevel.Information));

**✅ Bonus: EF Core Best Practices (বাংলায়)**

| **বিষয়** | **ভালো অভ্যাস** |
| --- | --- |
| **DbContext Lifetime** | Scoped রাখা উচিত — Singleton হলে মেমোরি সমস্যা |
| **Tracking vs NoTracking** | রিড-ওনলি জন্য AsNoTracking() ব্যবহার করুন |
| **Batching** | একাধিক SaveChanges() না করে একবারে SaveChanges() করুন |
| **Projection** | কেবল প্রয়োজনীয় ডাটা লোড করুন (Select) |
| **Indexes** | প্রয়োজনীয় কলামে [Index] দিয়ে পারফরমেন্স বাড়ান |

নিশ্চয়ই! নিচে আমি **Entity Framework Core Performance Optimization Techniques** বাংলায় বিস্তারিতভাবে উদাহরণসহ ব্যাখ্যা করছি।

**✅ Entity Framework Core Performance Optimization Techniques (বাংলা ব্যাখ্যা সহ)**

**🔶 ১. AsNoTracking() ব্যবহার করুন শুধুমাত্র Read-Only ডেটার জন্য**

**কেন:** Default ভাবে EF Core প্রতিটি entity track করে যাতে পরে পরিবর্তন হলে update করা যায়। কিন্তু Read-Only data এর জন্য এটা অপ্রয়োজনীয়।

**সমাধান:**

var products = context.Products.AsNoTracking().ToList();

➡️ এতে Change Tracker disable হয়, এবং performance বেড়ে যায়।

**🔶 ২. Select() দিয়ে শুধুমাত্র প্রয়োজনীয় কলাম আনুন**

**কেন:** পুরো entity আনলে extra memory & unnecessary joins হয়। Projection ব্যবহার করলে query ছোট হয়।

**সমাধান:**

var productNames = context.Products

.Select(p => new { p.Id, p.Name })

.ToList();

➡️ কেবল প্রয়োজনীয় field নিয়ে আসুন, entity না।

**🔶 ৩. Batch করে SaveChanges() ব্যবহার করুন**

**কেন:** প্রতি SaveChanges() call একটি SQL রাউন্ডট্রিপ করে। বারবার call করলে পারফর্মেন্স কমে।

**ভুল:**

foreach (var item in items)

{

context.Add(item);

context.SaveChanges(); // প্রতি লুপে ডাটাবেজ হিট

}

**সঠিক:**

foreach (var item in items)

{

context.Add(item);

}

context.SaveChanges(); // একবারেই batch করে

➡️ কম SQL call, ভালো পারফর্মেন্স।

**🔶 ৪. Include() কম ব্যবহার করুন, যতটা সম্ভব Lazy Load বা Projection করুন**

**কেন:** .Include() করলে বড় join তৈরি হয়, এবং performance bottleneck হতে পারে।

**ভুল:**

context.Orders.Include(o => o.OrderItems).ToList();

**সঠিক:**

var orders = context.Orders

.Select(o => new

{

o.Id,

o.Total,

Items = o.OrderItems.Select(i => new { i.Name, i.Quantity }).ToList()

}).ToList();

➡️ Projection + Select + কম কলাম = কম ডেটা = ভালো পারফর্মেন্স

**🔶 ৫. Query Caching বা Compiled Query ব্যবহার করুন**

**কেন:** প্রতিবার EF Core LINQ query translate করে SQL বানায়। Compiled query এ সেটি একবার করে রেখে দেয়।

**উদাহরণ:**

static readonly Func<AppDbContext, int, Product> getProductById =

EF.CompileQuery((AppDbContext ctx, int id) =>

ctx.Products.FirstOrDefault(p => p.Id == id));

var product = getProductById(context, 5);

➡️ Reusable query, fast execution।

**🔶 ৬. Proper Indexes দিন আপনার frequently searched/filtered ফিল্ডে**

**কেন:** ডাটাবেজে index না থাকলে search query গুলি স্লো হয়।

**উদাহরণ:**

[Index(nameof(Email))]

public class User

{

public int Id { get; set; }

public string Email { get; set; }

}

➡️ Faster WHERE Email = 'abc@x.com' searches।

**🔶 ৭. Avoid N+1 Query Problem (Lazy Load সমস্যা)**

**কেন:** যদি আপনি একটি list আনেন এবং প্রতিটি item এর জন্য আলাদা query রান হয়, তাহলে N+1 সমস্যা হয়।

**ভুল:**

var orders = context.Orders.ToList();

foreach (var order in orders)

{

var items = context.OrderItems.Where(i => i.OrderId == order.Id).ToList(); // প্রতি অর্ডারে query

}

**সঠিক:**

var orders = context.Orders

.Include(o => o.OrderItems)

.ToList();

অথবা Projection দিয়ে আরও ভালো করা যায়।

**🔶 ৮. ChangeTracker.AutoDetectChangesEnabled = false যখন bulk insert**

**কেন:** প্রতিটি Add()-এর সময় EF Core change detect করে। Bulk insert এ এটা disable করলে performance বাড়ে।

**উদাহরণ:**

context.ChangeTracker.AutoDetectChangesEnabled = false;

foreach (var item in bigList)

{

context.Add(item);

}

context.SaveChanges();

context.ChangeTracker.AutoDetectChangesEnabled = true;

➡️ Bulk insert ৫-১০ গুণ দ্রুত হতে পারে।

**🔶 ৯. Query Execution Time লগ করে মনিটর করুন**

**সমাধান:**

services.AddDbContext<AppDbContext>(options =>

options.UseSqlServer(connStr)

.LogTo(Console.WriteLine, LogLevel.Information));

➡️ দেখুন কোন query বেশি টাইম নিচ্ছে।

**🔶 ১০. AsSplitQuery() ব্যবহার করুন বড় Include গুলির জন্য**

**কেন:** Big Join হলে memory/cartesian explosion হতে পারে।

**সমাধান:**

var blogs = context.Blogs

.Include(b => b.Posts)

.AsSplitQuery()

.ToList();

➡️ একাধিক ছোট query হয়ে performance improve হয়।

**✅ সারাংশে EF Core Performance Tips**

| **Optimization** | **উপকারিতা** |
| --- | --- |
| AsNoTracking() | Tracking ছাড়াই দ্রুত read |
| Projection (Select) | কম ডেটা ট্রান্সফার |
| Compiled Query | Reused LINQ → Faster |
| Indexing | দ্রুত search |
| Bulk Save | কম SQL call |
| Query Log | Bottleneck ধরতে সাহায্য |
| Disable AutoDetectChanges | Bulk Insert-এ fast |

অবশ্যই! নিচে **EF Core-এ CQRS Implementation এর একটি সহজ উদাহরণ বাংলায়** দিচ্ছি। CQRS (Command Query Responsibility Segregation) হলো এমন একটি ডিজাইন প্যাটার্ন যেখানে **Query** এবং **Command** আলাদা করা হয়।

**EF Core + CQRS Implementation Example (বাংলায়)**

**১. CQRS কী?**

* **Command:** ডাটা পরিবর্তন (Insert, Update, Delete) করে।
* **Query:** শুধুমাত্র ডাটা রিড করে। কোনো পরিবর্তন করে না।

**২. প্রজেক্টের Structure (সিম্পল):**

/Commands

- CreateProductCommand.cs

- UpdateProductCommand.cs

/Queries

- GetProductByIdQuery.cs

- GetAllProductsQuery.cs

/Handlers

- CommandHandlers.cs

- QueryHandlers.cs

/Models

- Product.cs

/DbContext

- AppDbContext.cs

**৩. Model (Product.cs)**

public class Product

{

public int Id { get; set; }

public string Name { get; set; }

public decimal Price { get; set; }

}

**৪. Command & Command Handler উদাহরণ**

**CreateProductCommand.cs**

public class CreateProductCommand

{

public string Name { get; set; }

public decimal Price { get; set; }

}

**CommandHandler.cs**

public class CommandHandler

{

private readonly AppDbContext \_context;

public CommandHandler(AppDbContext context)

{

\_context = context;

}

public async Task<int> Handle(CreateProductCommand command)

{

var product = new Product

{

Name = command.Name,

Price = command.Price

};

\_context.Products.Add(product);

await \_context.SaveChangesAsync();

return product.Id;

}

}

**৫. Query & Query Handler উদাহরণ**

**GetProductByIdQuery.cs**

public class GetProductByIdQuery

{

public int Id { get; set; }

}

**QueryHandler.cs**

public class QueryHandler

{

private readonly AppDbContext \_context;

public QueryHandler(AppDbContext context)

{

\_context = context;

}

public async Task<Product> Handle(GetProductByIdQuery query)

{

return await \_context.Products.FindAsync(query.Id);

}

}

**৬. Controller বা Service থেকে কল করা**

// Command Example

var command = new CreateProductCommand { Name = "New Product", Price = 200 };

var productId = await commandHandler.Handle(command);

// Query Example

var query = new GetProductByIdQuery { Id = productId };

var product = await queryHandler.Handle(query);

**৭. গুরুত্বপূর্ণ পয়েন্ট**

* Command আর Query আলাদা ক্লাস ও হ্যান্ডলার হবে।
* Command হ্যান্ডলার ডাটাবেজে পরিবর্তন করে (SaveChangesAsync)।
* Query হ্যান্ডলার শুধুমাত্র রিড করে, ডাটাবেজে কিছু পরিবর্তন করে না।
* আপনি MediatR লাইব্রেরি ব্যবহার করে CQRS আরো সহজে ইমপ্লিমেন্ট করতে পারেন।

অবশ্যই! নিচে **EF Core Integration Testing** কিভাবে করবেন **In-Memory Database** ব্যবহার করে, সেটা বাংলায় সহজ উদাহরণসহ বুঝিয়ে দিলাম।

**EF Core Integration Testing with In-Memory Database (বাংলায়)**

**১. কেন Integration Testing?**

* Integration Test হচ্ছে পুরো ডাটাবেজ এবং EF Core ডাটাকন্টেক্সট সহ কোডের টেস্ট।
* Unit Test এর থেকে একটু বড়, প্রকৃত ডাটাবেজের সাথে কাজের মতো করে টেস্ট।
* **In-Memory Database** দিয়ে আসল ডাটাবেজ ছাড়াই দ্রুত টেস্ট করা যায়।

**২. প্রয়োজনীয় NuGet Package**

প্রথমে প্রজেক্টে এই প্যাকেজটি ইন্সটল করুন:

Microsoft.EntityFrameworkCore.InMemory

**৩. Sample Model ও DbContext**

public class Product

{

public int Id { get; set; }

public string Name { get; set; }

}

public class AppDbContext : DbContext

{

public DbSet<Product> Products { get; set; }

public AppDbContext(DbContextOptions<AppDbContext> options) : base(options) { }

}

**৪. Integration Test Setup (xUnit দিয়ে উদাহরণ)**

public class ProductRepositoryTests

{

private DbContextOptions<AppDbContext> \_dbContextOptions;

public ProductRepositoryTests()

{

\_dbContextOptions = new DbContextOptionsBuilder<AppDbContext>()

.UseInMemoryDatabase(databaseName: "TestDb")

.Options;

}

[Fact]

public async Task AddProduct\_ShouldAddProductToDatabase()

{

// Arrange

using var context = new AppDbContext(\_dbContextOptions);

var product = new Product { Name = "Test Product" };

// Act

context.Products.Add(product);

await context.SaveChangesAsync();

// Assert

var savedProduct = await context.Products.FirstOrDefaultAsync(p => p.Name == "Test Product");

Assert.NotNull(savedProduct);

Assert.Equal("Test Product", savedProduct.Name);

}

}

**৫. গুরুত্বপূর্ণ টিপস**

* **Database Name** ইউনিক রাখুন, অন্যথায় আগের টেস্টের ডাটা রয়ে যেতে পারে।
* প্রতিটি টেস্টে আলাদা DbContext ব্যবহার করুন।
* UseInMemoryDatabase শুধুমাত্র Integration Testing এর জন্য, Production এ ব্যবহার করবেন না।
* Real SQL behavior (যেমন FK constraints) ইমিট করে না, তাই কিছু রিলেশনাল কনস্ট্রেইন্ট টেস্টে পার্থক্য হতে পারে।

**৬. একাধিক টেস্টে Fresh ডাটাবেস**

প্রতিটি টেস্টে আলাদা ইন-মেমোরি ডাটাবেস দিতে পারেন:

var options = new DbContextOptionsBuilder<AppDbContext>()

.UseInMemoryDatabase(Guid.NewGuid().ToString()) // ইউনিক নাম

.Options;

**৭. Controller বা Service Integration Testing এর জন্য ব্যবহার**

[Fact]

public async Task GetProductById\_ShouldReturnProduct()

{

var options = new DbContextOptionsBuilder<AppDbContext>()

.UseInMemoryDatabase(Guid.NewGuid().ToString())

.Options;

// Arrange: Seed data

using (var context = new AppDbContext(options))

{

context.Products.Add(new Product { Id = 1, Name = "Product1" });

await context.SaveChangesAsync();

}

// Act: Test query

using (var context = new AppDbContext(options))

{

var product = await context.Products.FindAsync(1);

Assert.NotNull(product);

Assert.Equal("Product1", product.Name);

}

}

**সংক্ষেপে:**

| **বিষয়** | **ব্যাখ্যা** |
| --- | --- |
| Integration Test | EF Core ও ডাটাবেজ একত্রে টেস্ট করা |
| In-Memory DB | আসল ডাটাবেজ ছাড়া দ্রুত টেস্ট |
| DbContextOptions | In-Memory DB কনফিগারেশন |
| Fresh DB | প্রতিটি টেস্টে আলাদা ডাটাবেজ নাম দেওয়া উচিত |

অবশ্যই! নিচে **EF Core Performance Benchmarking Metrics** সম্পর্কে বাংলায় বিস্তারিত ব্যাখ্যা দিলাম, যাতে আপনি EF Core-র পারফরমেন্স কীভাবে পরিমাপ করবেন এবং উন্নত করবেন বুঝতে পারেন।

**EF Core Performance Benchmarking Metrics (বাংলায়)**

**১. Benchmarking মানে কি?**

Benchmarking হল কোনো সিস্টেম বা কোডের পারফরমেন্স পরিমাপ করা। EF Core-তে এটা মূলত ডাটাবেজ অপারেশনগুলো কত দ্রুত হচ্ছে, কত মেমোরি ব্যবহার হচ্ছে, ইত্যাদি বোঝার জন্য।

**২. EF Core Performance Metrics গুলো কি কি?**

**১) Execution Time (এক্সিকিউশন টাইম)**

* প্রতিটি Query বা Command ডাটাবেজে কত সময় নিচ্ছে।
* খুব বেশি সময় নিলে সেটাকে অপটিমাইজ করা দরকার।
* মাপার জন্য Stopwatch ব্যবহার করা যায়।

var stopwatch = Stopwatch.StartNew();

var products = await context.Products.ToListAsync();

stopwatch.Stop();

Console.WriteLine($"Execution Time: {stopwatch.ElapsedMilliseconds} ms");

**২) SQL Query Count (কতবার SQL চলে)**

* Application এ একটি কাজের জন্য কতগুলো SQL Query এক্সিকিউট হচ্ছে।
* খুব বেশি Query মানে N+1 সমস্যা বা inefficient code।

**৩) Query Complexity (কোডের কমপ্লেক্সিটি)**

* EF Core কতটা জটিল Query জেনারেট করছে (Join, Subquery, Include ইত্যাদি)।
* বড় Join বা অপ্রয়োজনীয় Include কমাতে হবে।

**৪) Memory Usage (মেমোরি ব্যবহার)**

* Query execution সময় কত মেমোরি লাগে।
* বেশি মেমোরি হলে অ্যাপ স্লো হতে পারে।

**৫) Network Latency (নেটওয়ার্ক লেটেন্সি)**

* ক্লায়েন্ট এবং ডাটাবেজ সার্ভারের মধ্যে ডেটা আদান-প্রদান কত দ্রুত হচ্ছে।
* বড় Query বা বেশি Query মানে বেশি Latency।

**৩. EF Core এ Performance Metrics মাপার টুলস**

**(ক) Logging and Diagnostics**

EF Core নিজে থেকে SQL Log করতে পারে:

optionsBuilder.LogTo(Console.WriteLine, LogLevel.Information);

➡️ এই লগ থেকে Query গুলোর Execution Time ও SQL দেখতে পারবেন।

**(খ) EF Core Interceptors**

* CommandInterceptor দিয়ে Query Execution এর আগে ও পরে সময় মাপা যায়।
* Profiler বা Diagnostic tools ব্যবহার করা যায়।

**(গ) BenchmarkDotNet**

* .NET এর BenchmarkDotNet লাইব্রেরি দিয়ে কোডের Performance মাপা যায়।
* EF Core এর জন্য বড় টেস্ট করতে পারে।

**৪. Example: EF Core Query Execution Time Logging**

public class ApplicationDbContext : DbContext

{

protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)

{

optionsBuilder

.UseSqlServer("YourConnectionString")

.LogTo(Console.WriteLine, LogLevel.Information);

}

}

➡️ এইভাবে EF Core Console এ প্রতিটি Query ও Execution Time দেখাবে।

**৫. EF Core Performance Optimization এর জন্য মেট্রিকস ব্যবহার কিভাবে করবেন?**

| **Metric** | **কিভাবে সাহায্য করবে** | **করণীয়** |
| --- | --- | --- |
| Execution Time | কোন Query বেশি সময় নিচ্ছে তা জানতে পারবেন | Query Optimize করুন, Projection ব্যবহার করুন |
| SQL Query Count | বেশি Query হলে N+1 সমস্যা ধরতে পারবেন | Include বা Lazy Load ঠিক করুন |
| Memory Usage | বেশি মেমোরি হলে অ্যাপ ধীর হবে | AsNoTracking, Projection ব্যবহার করুন |
| Network Latency | বেশি Latency মানে ডেটা ট্রান্সফার কমান | Query কমান, Batch Process করুন |

**৬. টুলস ও প্র্যাকটিস**

* **SQL Server Profiler / Extended Events** (SQL Server হলে)
* **EF Core Logging**
* **BenchmarkDotNet**
* **MiniProfiler** (Web apps এর জন্য)
* **DotTrace / JetBrains Rider Profiler** (IDE Plugins)

**৭. সংক্ষেপে**

EF Core Performance Benchmarking Metrics বুঝতে ও পরিমাপ করতে হলে আপনার এই বিষয়গুলো জানতে হবে:

* কত সময় নিচ্ছে (Execution Time)
* কতগুলো SQL Query হচ্ছে (Query Count)
* Query কতটা কমপ্লেক্স (Complexity)
* Memory কত ব্যবহার করছে
* নেটওয়ার্ক ডেটা ট্রান্সফার কত বেশি হচ্ছে

এরপর টুলস দিয়ে লগ নিন, মেট্রিকস সংগ্রহ করুন এবং Optimize করুন।