

◆ 1. Unit of Work (UoW)

الفكرة:

هي طبقة بتجمع كل العمليات على قاعدة البيانات داخل معاملة واحدة. (Transaction)
يعني لو عندك أكثر من Repository وكل واحد بيعمل في جدول، الـ Unit of Work بتخليهم كلهم جزء من عملية واحدة:

يا كله ينجح ويتحفظ ☒ ، يا كله يفشل ويتلغي. ☐

مثال عملي:

```
public class UnitOfWork : IUnitOfWork
{
    private readonly ApplicationDbContext _context;
    public ITaskRepository Tasks { get; private set; }
    public IUserRepository Users { get; private set; }

    public UnitOfWork(ApplicationDbContext context)
    {
        _context = context;
        Tasks = new TaskRepository(_context);
        Users = new UserRepository(_context);
    }

    public int Complete()
    {
        return _context.SaveChanges(); // ☒ كل التغييرات تتحفظ مرة واحدة
    }

    public void Dispose()
```

```

    {
        _context.Dispose();
    }
}

```

النتيجة:

بدل ما تستدعي `SaveChanges()` جوه كل `Repository` ، تستدعيها مرة واحدة من الـ `UnitOfWork`.

◆ 2. TCL (Transaction Control Language)

المعنى:

ده جزء من **SQL** ، خاص بالتحكم في المعاملات (Transactions).

الأوامر الأساسية:

الوظيفة	الأمر
بدء معاملة	BEGIN TRANSACTION
تنفيذ التغييرات	COMMIT
إلغاء كل التغييرات	ROLLBACK

مثال:

```
BEGIN TRANSACTION;
```

```
UPDATE Accounts SET Balance = Balance - 500 WHERE Id = 1;
```

```
UPDATE Accounts SET Balance = Balance + 500 WHERE Id = 2;
```

```
IF @@ERROR <> 0
```

```
ROLLBACK TRANSACTION;
```

```
ELSE
```

```
COMMIT TRANSACTION;
```

الهدف هو ضمان التكامل والاتساق في البيانات (Data Consistency).

◆ 3. Dependency Injection (DI)


الفكرة:

بدل ما تنشئ الكائن بنفسك (new Repository()) ، بتخلي الـ Framework يحقنه تلقائيًا.

مثال:

```
public class TaskitemsController : Controller
{
```

```
    private readonly IUnitOfWork _unitOfWork;
```

```
    public TaskitemsController(IUnitOfWork unitOfWork) // 
```

```
    {
```

```
        _unitOfWork = unitOfWork;
```

```
    }
```

```
    public IActionResult Index()
```

```
    {
```

```
        var tasks = _unitOfWork.Tasks.GetAll();
```

```
        return View(tasks);
```

```
    }
```

```
}
```

في: Program.cs

```
builder.Services.AddScoped<IUnitOfWork, UnitOfWork>();
```

```
builder.Services.AddScoped<ITaskRepository, TaskRepository>();
```

الهدف:

تقليل التبعية (Decoupling) وزيادة سهولة الاختبار والتبديل بين الـ implementations.

◆ 4. Testing Modularity

المعنى:

هو اختبار الأجزاء (Modules) أو الطبقات (Layers) بشكل مستقل.
وده بيتحقق بسهولة لما تستخدم **DI** و **UoW** لأنك بتقدر تعزل كل جزء وتختبره لوحده.

مثال اختبار بسيط للـ **Repository**:

[Fact]

```
public void AddTask_ShouldIncreaseCount()
```

```
{
```

```
    // Arrange
```

```
    var mockContext = new Mock<AppDbContext>();
```

```
    var repo = new TaskRepository(mockContext.Object);
```

```
    // Act
```

```
    repo.Add(new Taskitem { Title = "Test Task" });
```

```
    // Assert
```

```
    Assert.NotNull(repo);
```

```
}
```

الهدف:

تحافظ على Modular Design وتقدر تعمل Unit Tests لأي طبقة بسهولة.