Mocking a DBContext for Testing

The easiest means to mock a data context for testing, is to use an in-memory database.

An in-memory database is known by its string-name within a process. Each data context accessing the database instance must use its name, via options.

See this page for EF Logging implementation: EF: Logging

A few things are needed to mockup a dbcontext for testing:

- Entity class or classes
- Data Context class with DbSet for each entity class
- String name of the database
- Data Context builder options for an in-memory database
- Creating a Context Instance

1. Entity Class
   At least one entity class. This can be any simple entity class with an Id property, such as the following:

```
public class TestClass
{
    public int Id { get; set; }

    public string Val1 { get; set; }

    public int Val2 { get; set; }
}
```

2. Data Context
The data context class requires a DbSet for each test class. This can be as simple as the following:

```
public class TestDbContext : DbContext
{
    public TestDbContext([NotNullAttribute] DbContextOptions options) : base(options)
    // public cDBDContext_Base() : base()
```

```
    {
    }
    public TestDbContext() : base()
    {
    }

    protected override void OnModelCreating(ModelBuilder modelBuilder)
    {
        //base.OnModelCreating(modelBuilder);
    }

    public override int SaveChanges()
    {
        return base.SaveChanges();
    }

    public DbSet<TestClass> TestData { get; set; }
}
```

3. Database Name
This is the string-name of the database.
Any context using this name will reach the same in-memory database.

4. Data Context Builder Options
The following is a simple builder options for creating a data context to an in-memory database of a given name:

```
        var options = new DbContextOptionsBuilder<GenericRepositoryEF_Tests.TestDbContext>()
            .UseInMemoryDatabase(databaseName: stringdatabasename)
            .Options;
```

5. Creating a data context instance
The following is a method to create a data context instance and populate it with test data:

```
    string dbname = "testdatabase";

    // Create an options instance to give us a connection to the named in-memory database.
    var options = new DbContextOptionsBuilder<GenericRepositoryEF_Tests.TestDbContext>()
        .UseInMemoryDatabase(databaseName: dbname)
        .Options;
```

```
// Insert seed data into the database using one instance of the context...
using (var context = new GenericRepositoryEF_Tests.TestDbContext(options))
{
    // Populate the list...
    for(int x = 0; x < 1000; x++)
    {
        GenericRepositoryEF_Tests.TestClass tc = new TestClass();
        tc.Val1 = x.ToString();
        tc.Val2 = x;

        context.TestData.Add(tc);
    }

    context.SaveChanges();
}
```

6. Any subsequent context instance with the same database string name, will have access to the seeded data above.
Meaning, a second data context, created with the same options, can access the seed data, generated by the above code:

```
// Create an options instance to give us a connection to the named in-memory database.
var options1 = new DbContextOptionsBuilder<GenericRepositoryEF_Tests.TestDbContext>()
     .UseInMemoryDatabase(databaseName: "testdatabase")
     .Options;

// Use a clean instance of the context to access data...
using (var context = new GenericRepositoryEF_Tests.TestDbContext(options1))
{
    // Now, do a simple query for some portion of the list...
    var query = from b in context.TestData
                //where b.Val2 > 1000
                select b;

    var res = query.ToList();
}
```

Revision #3
Created 8 October 2025 02:36:09 by glwhite
Updated 19 February 2026 06:48:33 by glwhite