In **Entity Framework Core**, when you call `Add()` or `AddRange()`, EF will **track** the entity in memory until you call `SaveChanges()`.
If you want to **insert a record without retaining it in memory** (no change tracking), you can use `AsNoTracking` for queries, but for inserts you need to **detach** the entity after saving, or use a bulk insert library that bypasses tracking entirely.

Option 1 — Insert and Detach Immediately

```
using System;
using Microsoft.EntityFrameworkCore;

public class MyEntity
{
    public int Id { get; set; }
    public string Name { get; set; }
}

public class MyDbContext : DbContext
{
    public DbSet<MyEntity> MyEntities { get; set; }

    protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
    {
        optionsBuilder.UseSqlServer("Your_Connection_String");
    }
}

class Program
```

```
{
    static void Main()
    {
        using var context = new MyDbContext();

        var entity = new MyEntity { Name = "Test" };

        context.MyEntities.Add(entity);
        context.SaveChanges();

        // Detach to avoid keeping it in memory
        context.Entry(entity).State = EntityState.Detached;

        Console.WriteLine($"Inserted entity with ID: {entity.Id}");
    }
}
```

How it works:

- EF tracks the entity only until `SaveChanges()`.
- After saving, `EntityState.Detached` removes it from the change tracker.

Option 2 — Use `ExecuteSqlRaw` for Direct Insert (No Tracking at All)

```
using Microsoft.EntityFrameworkCore;

using var context = new MyDbContext();
```

```
string name = "Test";
context.Database.ExecuteSqlRaw(
    "INSERT INTO MyEntities (Name) VALUES ({0})", name
);
```

Pros:

- No tracking, no memory overhead.
- Direct SQL execution.

Cons:

- No automatic ID retrieval unless you query afterward.

Option 3 — Use EFCore.BulkExtensions (Best for Many Inserts)

```csharp
using EFCore.BulkExtensions;

using var context = new MyDbContext();

var entities = new List<MyEntity>
{
    new MyEntity { Name = "Bulk 1" },
    new MyEntity { Name = "Bulk 2" }
};

context.BulkInsert(entities); // No tracking, very fast
```

Pros:

- Extremely fast for large datasets.
- No tracking overhead.

Cons:

- Requires installing `EFCore.BulkExtensions` NuGet package.

✅ **Recommendation:**

- For **single inserts**, use **Option 1** (insert + detach).
- For **large batches**, use **Option 3** with bulk insert.

If you want, I can give you a **fully runnable EF Core example** that inserts **without ever tracking** the entity in memory.
Do you want me to prepare that?