Evet, doğru! Eğer bir backend uygulaması kullanıyorsanız (örneğin, Python Flask veya başka bir framework), bu SQL sorgusunu doğrudan çalıştırmanız gerekmez. Arka uç uygulamanız, gelen kullanıcı taleplerine (örneğin, bir REST API isteğine) göre bu sorguyu **dinamik olarak oluşturur** ve veri tabanına gönderir.

**Süreç Nasıl İşler?**

1. **Kullanıcı İsteği:** Kullanıcı bir güncelleme yapmak ister (örneğin, yorum ekler veya fotoğraf güncelleri).
2. **API Üzerinden Gönderim:** Kullanıcı, bir HTTP isteği (POST/PUT) ile arka uç'a gerekli bilgileri gönderir.
3. **Arka Uç Mantığı:**
   * Gelen JSON verisini işler.
   * Sadece gönderilen bölgelere göre bir **UPDATE** sorgusu oluşturulur.
   * Bu sorguyu veri tabanına gönderir (örneğin, SQLite).
4. **Veritabanında Güncelleme:** Arka uç, SQL sorgulaması çalıştırılır ve güncellenir.

dinamik bir sistem oluşturmak ve arayüzde yapılan işlemleri doğrudan veritabanına kaydetmek için bir **backend** kısmı oluşturmanız gerekiyor. Backend, kullanıcı arayüzü (frontend) ile veritabanı arasında köprü görevi görür. İşte dinamik bir sistem oluşturmak için yapmanız gerekenler:

**1. Backend Oluşturma**

Backend, veritabanıyla etkileşim kuracak API'ler sağlar ve bu API'ler frontend tarafından çağrılır.

**Teknoloji Seçimi**

Backend geliştirmek için popüler teknolojilerden birini seçebilirsiniz:

* **Node.js (Express)**: JavaScript kullanıyorsanız uygun bir seçenek.
* **Python (Flask/Django)**: Python tercih edenler için güçlü seçenekler.
* **PHP**: Basit uygulamalar için sık kullanılan bir seçenek.
* **Java (Spring Boot)** veya **C# (ASP.NET)**: Daha büyük projeler için ideal.

**1. Backend'i Önce Oluşturmak**

Backend (arka uç) API’lerini oluşturmak, projenizin temel mantığını ve veri akışını düzenlemek için en mantıklı adımdır. Bunun nedeni:

* **Tutarlılık:** Frontend, backend API’lerine bağlanacağı için önceden API’lerin doğru çalışmasını sağlamak önemlidir.
* **Esneklik:** Backend tamamlandıktan sonra veritabanı tasarımı buna uyacak şekilde düzenlenebilir.

 **Backend’i İlk Oluşturma:** API’ler ve iş mantığı hazır olduğunda frontend’inizi kolayca bu API’lere bağlayabilirsiniz.

 **Veritabanı İkinci:** Backend, veritabanı ihtiyaçlarını daha iyi belirleyeceği için sonraki adım olarak tasarlanmalıdır.

 **Frontend’i Entegre Etme:** Mock (sahte) veriler yerine gerçek API çağrılarıyla çalışarak frontend’i geliştirin.

**Projeyi Çalıştırmak**

1. **Veritabanı Ayarı:** Kodda UseInMemoryDatabase("TodoList") kısmı, test amaçlı olarak bellekte çalışan bir veritabanı kullanıyor. Daha sonra SQL Server veya başka bir veritabanıyla değiştirebilirsiniz.
   * Örneğin, SQL Server kullanmak için UseSqlServer metodunu ve gerekli bağlantı dizesini kullanabilirsiniz:

options.UseSqlServer(Configuration.GetConnectionString("DefaultConnection"));

1. **API Testi:** Backend’i çalıştırdıktan sonra Postman veya bir frontend isteğiyle API’lerinizi test edebilirsiniz:
   * Kullanıcı Kayıt: POST /api/users/register
   * Görev Ekleme: POST /api/tasks

**. Visual Studio’da Projeyi Çalıştırma**

1. **Çözümü (Solution) Açın:** .sln dosyasını açarak projeyi Visual Studio’ya yükleyin.
2. **Projeyi Çalıştırın:** F5 veya IIS Express seçeneğiyle projeyi başlatın.
3. **API’leri Test Edin:** Tarayıcıda http://localhost:{port}/api/tasks gibi URL’leri test edebilirsiniz.

Migrationsbölümler içinde bir göç (veritabanını değiştirir) oluşturulur.

# Migration oluştur

(dotnet ef migrations add InitialCreate)

Mevcut migration ları temizler

(dotnet ef migrations remove)

(dotnet ef database update)

Bu işlemler sonunda:

1. Proje dizininde TodoList.dbbir SQLite dosyası oluşturulacak.
2. Tasksve Userstabloları bu veri tabanına eklenecek.

**Neden Yeni Migration Ekliyoruz?**

* Veritabanında zaten uygulanmış bir migration'ı **kaldırmak veya değiştirmek**, canlı projelerde tutarsızlığa neden olabilir.
* EF Core, **yeni migration** oluşturarak sadece değişiklikleri ekler ve mevcut tablo yapısını bozmadan günceller.

**Migration Listesini Kontrol Edin**:

dotnet ef migrations list

Bu **boş bir dizi** ([]) döndüğünü gösteriyor, yani şu an için veritabanında **hiç kayıtlı bir görev** (task) yok.

**POST** isteği göndererek veritabanına bir görev ekleyelim.

[FromBody] atributu, JSON verisinin HTTP isteği gövdesinden alınmasını sağlar.

 **POST** kullanarak **veri eklemelisiniz**.

 Daha sonra **GET** kullanarak eklenen veriyi **kontrol edebilirsiniz**.

 **POST** metodu veritabanına **yeni bir kayıt eklemek** için kullanılır.

 Eğer veritabanınız boşsa ve henüz veri eklenmemişse **GET** metodu çalıştırıldığında **boş bir liste ([])** döner.

 **Kullanıcı Yönetimi**:

* **POST /api/users/register**: Yeni kullanıcı kaydı.
* **POST /api/users/login**: Kullanıcı girişi.

 **Görev Yönetimi**:

* **GET /api/tasks**: Tüm görevleri getirir.
* **POST /api/tasks**: Yeni görev ekler.
* **PATCH /api/tasks/{id}**: Görevi günceller.
* **DELETE /api/tasks/{id}**: Görevi siler.

**API Çalıştırmadan Frontend Çalışır mı?**

* Frontend sayfanız yalnızca görsel olarak çalışabilir, ancak API çalışmıyorsa veri gönderemez veya alamaz.
* Örneğin:
  + Giriş yap butonuna tıkladığınızda, API çalışmıyorsa "Sunucuya bağlanırken bir hata oluştu" mesajını alırsınız.

**API Bilgilerini Nereye Yazmanız Gerekir?**

**a. API'de (Backend):**

* API'nin ne yapacağını ve nasıl çalışacağını tanımlarsınız:
  + Hangi uç noktanın hangi işlemi gerçekleştireceğini belirler (ör. POST /api/tasks görev ekler).
  + Veritabanı ile bağlantıyı sağlar.

**b. Frontend'de:**

* API'ye bağlanacak istekleri yazarsınız:
  + Örneğin, bir fetch veya axios çağrısı ile API'ye istek gönderirsiniz.
  + Kullanıcıdan alınan bilgiyi API'ye gönderir ve API'den dönen sonucu gösterirsiniz.

**CORS (Cross-Origin Resource Sharing)**, bir web sayfasının (frontend) başka bir alan adı, protokol veya port üzerinde çalışan bir sunucuya (backend) HTTP isteği yapmasını kontrol eden bir güvenlik mekanizmasıdır. **CORS hatası**, tarayıcının, bir CORS politikasının ihlal edildiğini tespit ettiğinde ortaya çıkar.

appsettings.json dosyası, bir .NET Core veya ASP.NET Core uygulamasında yapılandırma ayarlarını depolamak için kullanılan bir JSON dosyasıdır. Bu dosya, uygulamanızın çalışırken ihtiyaç duyduğu veritabanı bağlantı bilgileri, API anahtarları, uygulama ayarları ve diğer yapılandırma değerlerini saklamak için kullanılır.

**appsettings.json Dosyasının Avantajları**

1. **Ayrılmış Konfigürasyon**: Uygulamanın ayarlarını koddan bağımsız bir şekilde tanımlamanıza olanak tanır.
2. **Çevreye Özel Yapılandırma**: Üretim, test ve geliştirme ortamları için farklı ayar dosyaları oluşturabilirsiniz.
3. **Kolay Yönetim ve Güncelleme**: Ayarları değiştirmek için kodu yeniden derlemeye gerek yoktu

**Users Tablosu**:

* Kullanıcı bilgilerini saklıyor

**TodoItems Tablosu**:

* Kullanıcının görevlerini saklıyor