# DAPPER

Dapper Stackoverflow ekibi tarafından geliştirilmiş bir Micro ORM aracıdır. Diğer ORM lere göre oldukça hızlıdır. Dapper’da Entity Frameworkte olduğu gibi bir **Code First** yapısı söz konusu **değildir**. Siz Dapper ile bir uygulama geliştirecekseniz hali hazırda var olan bir veritabanı kullanmanız gerekecektir. Dapper diğer Mikro ORM araçlarından oldukça hızlıdır. (**Entity Framework e göre oldukça Hızlıdır**) (Micro ORM kavramı olarak adlandırılan kavram Normal ORM lere nazaran daha az özelliklerle lite versiyonu gibi düşünülebilir. ORM lerden Entity Framework düşünüldüğünde Sıfırdan tablo oluşturma olan tabolara göre ilgili neslerinü üretmek gibi çeşitli özellikler söz konusu iken Dapper da bu özellikler söz konusu değildir. )

metin, ekran görüntüsü, yazı tipi, sayı, numara içeren bir resim

Açıklama otomatik olarak oluşturuldu

(**Entity framework Core** artık **150ms** ile çalışmaktadır bu tablo entity framework hariç güncel)

* Dapper’ı kullanabilmek için bir database oluşturalım. (Halihazırda olan bir database e ihtiyaç her zaman dapper için söz konusudur )

metin, ekran görüntüsü, yazılım, sayı, numara içeren bir resim

Açıklama otomatik olarak oluşturuldu

Burada bir **DapperDb** isminde bir database oluşturalım. Isteğe bağlı olarak burada kod ile oluşturulabileceği gibi new database ile de bir database oluşturabiliriz.

metin, ekran görüntüsü, yazı tipi, yazılım içeren bir resim

Açıklama otomatik olarak oluşturuldu

Bu şekilde hem database oluşturduk hem de dapperdb veri tabanını aktif hale getirerek yapılacak işlemlerin üzerinde koşulmasını sağladık. Go script dosyasının sıralı bir şekilde çalıştırılmasını sağlar.

metin, yazı tipi, ekran görüntüsü, çizgi içeren bir resim

Açıklama otomatik olarak oluşturuldu

Işleyiş boyunca bu tablo kullanılacaktır.

## Dapper Proje Örneği

Bu aşamadan sonra Dapper methodları incelenecektir. İnceleme süresince yeni bir proje oluşturulacaktır. İlgili süreç için Ado.Net ile karşılaştırma yapılacaktır. Bu işlemler için **Connected mimaride Ado.Net** ile bir işlem gerçekleştirilip beraberinde Dapper ile aynı iş süreci gerçekleştirilecek ve kodsal maliyet, esneklik gibi parametreler karşılaştırılacaktır. (Ado.Net e göre performans kıssası karşılaştırılacaktır.)

metin, yazılım, multimedya yazılımı, grafik yazılımı içeren bir resim

Açıklama otomatik olarak oluşturuldu

Nuget üzerinden ilgili dapper paketini ekleyelim

metin, ekran görüntüsü, yazılım, multimedya yazılımı içeren bir resim

Açıklama otomatik olarak oluşturuldu

metin, ekran görüntüsü, yazılım, multimedya yazılımı içeren bir resim

Açıklama otomatik olarak oluşturuldu

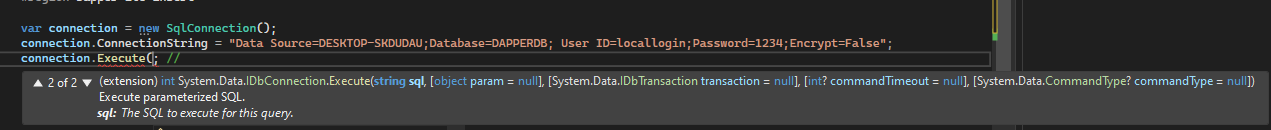
Ilgili süreçte **Ado.net** ile **Dapper**’I karşılaştırabilmek için connected mimari ile insert atalım. Burada tabi ado.net kullanacağımızdan ötürü sqlclient ile işlem gerçekleşecek. **(Ek olarak eklemek istediğim .Net frameworkde SqlClient kütüphanesi hali hazırda gelirken Core da gelmemekte biz bunu nugetten tedarik etmemiz gerekecektir)**

metin, ekran görüntüsü, yazılım, yazı tipi içeren bir resim

Açıklama otomatik olarak oluşturuldu

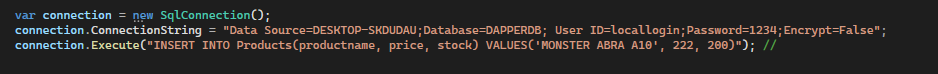
Connection nesnesi tanımlanması, Connection open olması, komut tanımlanması ve execture edilmesi gibi işlemler sırası ile gerçekleştirildi.

Aynı işlemi şimdi **Dapper** ile gerçekleştirelim.



Görüldüğü gibi Dapper çalışırken **SqlConnection** sınıfı için **SqlClient** kütüphanesine ihtiyaç duyar. Neticesinde sizler bir sorgu çalıştırabilmeniz için **excute** **komutunu** kullanmanız yetecektir (**ekleme, güncelleme, silme**).

**Parametreleri içerisinde Query, Çoklu işlem için parameter array veya object, Command Type, Transaction, Zaman aşımı gibi parametreler mevcuttur. Bu parametreler de ayarlanabilmektedir.**



metin, yazılım, yazı tipi, çizgi içeren bir resim

Açıklama otomatik olarak oluşturuldu

**Görüldüğü gibi aynı işlem gerçekleşmiş bulunmaktadır. Neticesinde burda dinamik sql yazabildiğimiz gibi parametreli insert işlemi de gerçekleştireilbilirdi.**

metin, yazı tipi, ekran görüntüsü içeren bir resim

Açıklama otomatik olarak oluşturuldu

Ilgili @ ibareindeki alanlar kafamıza göre isimlendirme ile belirleyebildiğimiz gibi anonymous type içerisinde de isimlerin eşleşmesine özen göstermemiz gerekecektir. Yani values kısmında @ li yazan ibarelere kolonlara ismen karşılık denk gelmesede aynı isim Anonymous olarak dışardan verilen değişkenler ile eşleşmesi gerekecektir.

Aynı şekilde bu yöntem ile çoklu insert işlemi de gerçekleştirebilmekteyiz.

metin, ekran görüntüsü içeren bir resim

Açıklama otomatik olarak oluşturuldu

(Anonymous tipten ziyade normal array, liste de atabilirsin ama model içerisindeki alan adları ile values içerisinbde @ile belirtilmiş alan adları uyumlu olmak zorunda)

metin, ekran görüntüsü, yazı tipi içeren bir resim

Açıklama otomatik olarak oluşturuldu

Parametre ile insert işlemi bu şekilde gerçekleşmektedir.

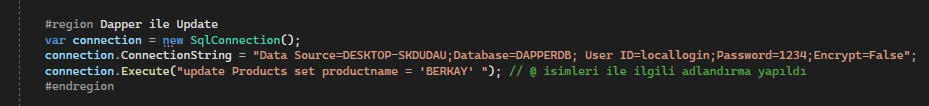
**!!!!! (Update, Delete, İnsert işlemleride Dapper da Execute methodu kullanılıyorken Ado.Net üzerinde ise ExecuteNonQuery kullanılmaktadır. Birbirlerinin karşılıkları gibi düşünülebilir ve iki method içerisinde de output olarak etkilenen satır sayıları dönmektedir. ) !!!!!**

Aynı şekilde update delete yaparken de execute methodu ile işlemler gerçekleştirebilirsiniz.

metin, ekran görüntüsü, yazı tipi içeren bir resim

Açıklama otomatik olarak oluşturuldu

Görüleceği gibi delete ve update işlemleride bu şekilde gerçekleştirilebilmektedir.



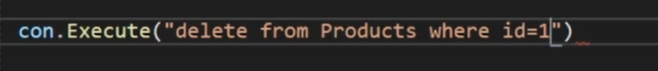
metin, ekran görüntüsü, ekran, görüntüleme, yazılım içeren bir resim

Açıklama otomatik olarak oluşturuldu

metin, ekran görüntüsü, yazı tipi içeren bir resim

Açıklama otomatik olarak oluşturuldu

Görüldüğü gibi update işleminde de bu şekilde çalışmaktadır. Ve toplu bir şekilde işlemler yapılabilmektedir execute konutunun parametresi gereği





Yukarıda gözüktüğü gibi silme işlemi de Execute fonkisyonu ile birlikte gerçekleştirilebilmektedir.

(**Bir adonet te Ekleme, Silme, Güncelleme için executenonquery, dapper da ise Execute kullanılmaktadır.**)

Dapper ile birlikte temel seviyede ekleme, silme ve güncelleme işlemleri gerçekleştirilmiş bulunmaktadır. Şimdi ise veritabanından değer okumaya çalışalım. Bu başlıkta veritabanından değer getirilecektir.

# Dapper ile Veri Getirme

**Veri getirme** işlemlerinden birisi **ExecuteScalar** methodudur. **Execute scalar dönen tablo içerisinden ilk satır, ilk sütun bilgisini getirir.**

metin, yazılım, bilgisayar simgesi, web sayfası içeren bir resim

Açıklama otomatik olarak oluşturuldu

Execute scalar ile bize dönecek veri 1002 olacaktır.

metin, yazılım, multimedya yazılımı, yazı tipi içeren bir resim

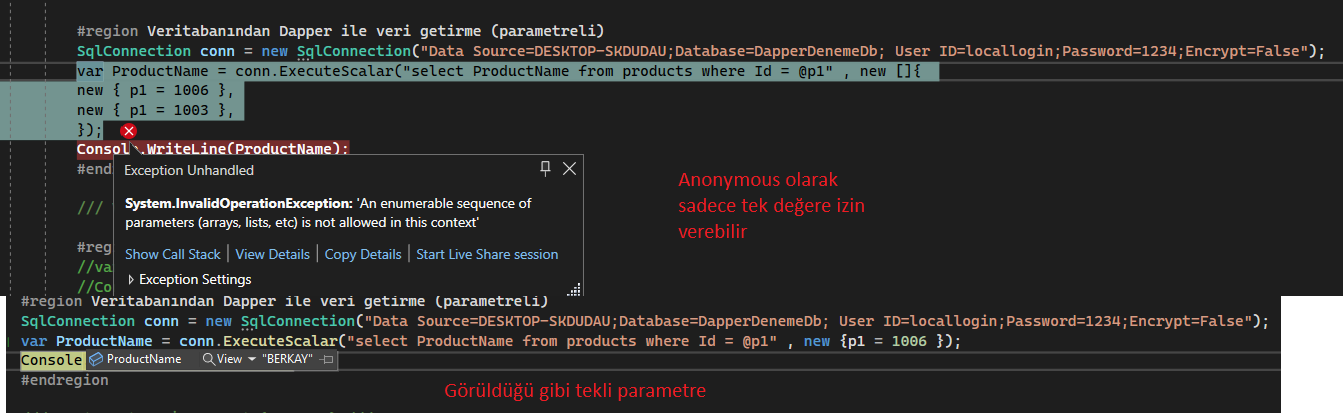
Açıklama otomatik olarak oluşturuldu

**Select cümlesinden gelen ilk satır, ilk sütun okunur.** Execute scalar dönüş tipi objedir. Bu sebeple her veri döner. Sadece ilk satır ve soldan ilk sütun verisini alır. Dönen verinin object tipinde olmadan direk cast edilmiş şekilde kullanmak isterseniz eğer generic tip ile kullanmanız gerekecektir. Ilgili örnek aşağıda verilmiştir.



Strinmg olarak gelen değerin dönüş tipini verdik istersen value içerisinde de convert edebilirdik ama generic bir mimaride yazılmış ilgli method.

Aynı şekilde execute scalar da da anonymous değişken kullanarak parametreler dışarıdan verilebilmektedir. Bu şekilde ilgili sorgu kriterleri verilerek işleyişe devam edilebilir.



# Dapper ile Veri Getirme (Çoklu Satır)

Bu başlık altında çok satırlı veri getirmek , bir tüm tabloyu getirmek işlenecektir. Öncesinde **Dynamic** keywordü ile çalışma yapalım. Bu başlık altında ilgili keyword çok kullanılacaktır.

Genellikle tip güvenli çalışmayı tercih ederiz. Dynamic keywwordu ile bir değişken belirtebilir ve değer ataması gerçekleştirebilirsiniz.

metin, yazı tipi, ekran görüntüsü, tasarım içeren bir resim

Açıklama otomatik olarak oluşturuldu

Var varken dynamic kullanmamızın temel esası ilk değer atamasından sonra farklı br tipde değer ataması daha gerçekleştirebilirsiniz. Yani şu ben ilk olarak verable1 e gidip integer atıyorsam sonrasında bir string te atayabilirim. Atadığınız değerden farklı bir tip atayabilirsiniz ama varda ilk değer ataması neyse o değer tipinde yeni değerler atayabilirsiniz. Var ile tanımlamış oldugunuz tip tip güvenlidir ve o tipte değer alması gerekmektedir. Ilk olarak int atarsan hep int gideceksin ama dynamic ile bu sorun ortadan kalkmıştır. Var ile int tanımına int olarak devam etmemiz gerekir.

metin, yazı tipi, ekran görüntüsü içeren bir resim

Açıklama otomatik olarak oluşturuldu

Ilerleyen aşamada query method bizlere dynamic bir veri dönecektir. Bu sebeple bu anlatım bu başlık altında gerçekleştirilmiştir.

* Şimdi ise birden fazla veriyi **(tabloyu yani çoklu satır verisinin)** nasıl çekebileceğimzi inceleyelim. bu işlem için **Query** denen method hazırlanacaktır.

metin, yazı tipi, ekran görüntüsü içeren bir resim

Açıklama otomatik olarak oluşturuldu

Görüldüğü gibi query method bizlere dynamic bir veri tipi olarak 3 satırı anonymous tipte döndürmüş bulundu.

* Neticesinde ilgili çıktı bu şekildedir. Bizlere Query Methodundan çıkan Sonuç **IEnumerable<Dynamic>** tir. Bu neticede bizler tip güvensiz çalıştığımız için Anonymous listesi döndü.
* Ilgili query çalışınca bir liste olarak anonymous tipte bir liste dizisi gelecektir.

metin, yazılım, multimedya yazılımı, grafik yazılımı içeren bir resim

Açıklama otomatik olarak oluşturuldu

* Görüldüğü gibi ilgili alan bizlere **IEnumerable<Dynamic>** olaraktip güvensiz dönmüştür. Dynamic dönüşlerde almış oldugunuz dataları dynamic içerisinde kolon adları ile birlikte çevirmeniz gerekmektedir. Yazılan değerler kolon isimleri ile aynı olması gerekir.

metin, ekran görüntüsü, yazı tipi içeren bir resim

Açıklama otomatik olarak oluşturuldumetin, ekran görüntüsü, sayı, numara, yazı tipi içeren bir resim

Açıklama otomatik olarak oluşturuldu

* Ilgili alanları tip güvenli olarak çevirebiliriz.
* Tip güvensiz çağırmada veritabanından dönen kolon isimleri kullanılarak dinamik verilere erişim sağlanabilir.
* Eğerki bizler tip güvenli çalışmak istiyorsak bir model hazırlanıp generic mimari içerisinde verilmesi gerekecektir.

Ilk olarak product sınıfımız ile karşılayacağımız sınıfı yazalım

ekran görüntüsü, multimedya yazılımı, yazılım, grafik yazılımı içeren bir resim

Açıklama otomatik olarak oluşturuldu

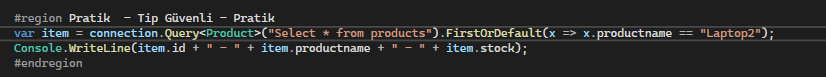
Modelimizi yazdıktan sonra **generic** bir şekilde **query** method içerisinde verelim

metin, yazı tipi, ekran görüntüsü içeren bir resim

Açıklama otomatik olarak oluşturuldu

metin, ekran görüntüsü, yazılım, multimedya yazılımı içeren bir resim

Açıklama otomatik olarak oluşturuldu



Yukarıda görüldüğü gibi Hem AnonymOus tip te hem de Tip güvenli bir şekilde çalışma yapmamız mümkün kılınmıştır.

* Beraberinde Query Methodlarına benzer olarak Linq yapısından tanıdık gelinen bazı Methodlar Vardır. **QueryFirst, QuerySingle, QueryFirstOrDefault, QuerySingleOrDefault** ve bunların **generic** yapılı helleri.
* **QueryFirst:** bizlere birden fazla dönen veri içerisinden ilk veriyi seçer, Null Gelirse bizlere hata atar.
* **QuerySingle :** sorgudan bizlere tek bir veri dönmesi gerekir. Null veya birden fazla veri dönerse hata atar
* **Default Versiyonları:** ikisininde defaultlu versiyonunda null Kabul edilebilir şekilde işlem yapmaya yarar.

# Dapper ile Tablo Joinleme Süreçleri:

Dapper ile **CRUD** işlemleri gerçekleştirmeyi öğrendik. Joinleme Süreçleri işlenecektir. Diğer projelerden bağımsız bir şekilde yeni bir proje kurarak işe başlıyorum

metin, yazılım, multimedya yazılımı, grafik yazılımı içeren bir resim

Açıklama otomatik olarak oluşturuldu

Ilgili kütüphane kurulumları gerçekleştikten sonra Senaryoyu kurgulayabilmek için Veritabanı tablolarını ve ilişkilerini oluşturuyorum. Çoğa çok bir yapı kurulacak ve senaryolar oluşturulacaktır.

metin, elektronik donanım, ekran görüntüsü, ekran, görüntüleme içeren bir resim

Açıklama otomatik olarak oluşturuldu

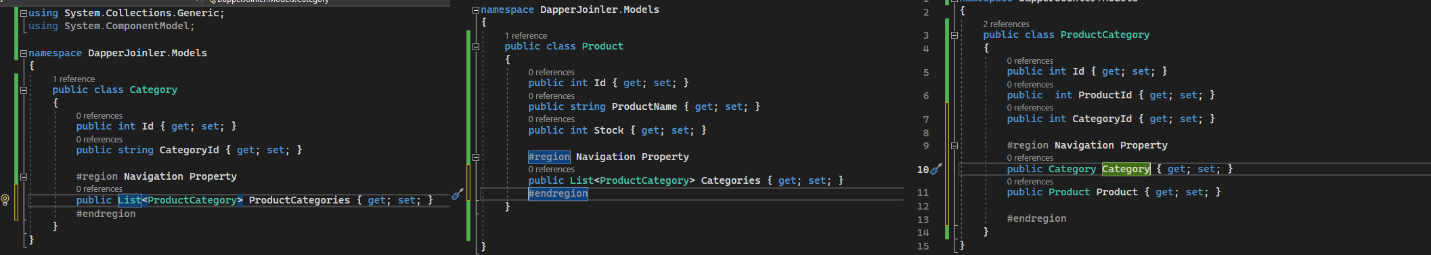
Kategori ve Product arasında çoka çok bir ilişki oluşturulabilecek bir şekilde ilgili senaryo kurgulandı.

metin, ekran görüntüsü, çizgi, yazı tipi içeren bir resim

Açıklama otomatik olarak oluşturuldu

* Şeması yukarıda gözüktüğü şekildeki gibidir. Içerisinde Data Girişleri yapıldıktan sonra sırası ile modellerimizi oluşturmaya başlıyoruz.
* Modellerden kastımız aslında C# taki entitylerdir.

Bu aşamadan sonra işlemlere başlayabiliriz. Bir Join Sorgusu yazalım. Joinde verilen category id ye göre ilgili productları getirelim. Category id si 3 olan Product ları getir. Entity Frameworkteki gibi Navigation Propertyleri geçelim.



İlgili Nesnelerimizi oluşturduktan sonra sürece dair olarak **Connection** nesnesi içerisinden **Query** methodunu çağırarak ilgili süreci gerçekleştirebiliriz.

metin, multimedya yazılımı, yazılım, ekran görüntüsü içeren bir resim

Açıklama otomatik olarak oluşturuldu

Query methodunun overloadlarında gezersek return, first ve second olan bir yapıya denk geleceğiz. Neticesinde iki nesne joinlenerek 3. Nesne (return edilecek tip) belirlenir ve çıktı bu olur. First ilk sokulacak, second 2. Sokulacak ve sonuncu konan ise geriye dönen nesne olarak canlandırılacaktır.

metin, ekran görüntüsü, yazı tipi, yazılım içeren bir resim

Açıklama otomatik olarak oluşturuldu

metin, yazı tipi, yazılım, çizgi içeren bir resim

Açıklama otomatik olarak oluşturuldu

Yukarıdaki şemada mapleme mantığı gözükmektedir. Func delegte içerisinde verilen objeler sırası ile soldan tablolar halinde bölünerek karşılaştıkları nesneler dolmakta ve func delegate ile dolan nesneler dönmektedir.

Konunun daha iyi pekişebilmesi için bir örnek daha çözümlenecektir.

# Dapper Transaction :

Transaction işlemlerini bu başlık altında gerçekleştirilecektir. Transaction işlemi bilindiği gibi birden fazla ve birbirleri ile ilgili ilişkilerin aynı anda gerçekleştirilmesi ve kaydedilmesi, parça parça işlemlerin gerçekleşmemesi esasına dayalıdır. Örnek vermek gerekirse banka hesapları işlemlerinden bahsedelim. Iki kullanıcımız sistemde var ve birbirlerin para gönderileceği düşünülsün. A hesabından para çıkışı gerçekleşirken ilgili süreçte B hesabı içerisinde para akmaktadır. A hesabından Paranın çıkışında ilgili işlem kaydedilirse eğer B hesabına para geçmeden arada ilgili işlem hata verebilir. Ama a hasabından para çıkışı gerçekleştiğinden ötürü bu işlemde para uçmuş olacaktır. Bu sebeple aynı anda hem A hesabını hem de b hesabını güncelleyebilmemiz ve alınan hata sonucunda A ve B hesaplarının eski hallerine getirilebilmesi için bizler Transactionları kullanacağız.

metin, yazı tipi, ekran görüntüsü, çizgi içeren bir resim

Açıklama otomatik olarak oluşturuldu

metin, yazı tipi, çizgi, sayı, numara içeren bir resim

Açıklama otomatik olarak oluşturuldu

metin, ekran görüntüsü, yazı tipi içeren bir resim

Açıklama otomatik olarak oluşturuldu

metin, yazı tipi, ekran görüntüsü içeren bir resim

Açıklama otomatik olarak oluşturuldu

Transaction commit edilmeden ilgili işleyişe devam edilemez. Bir diğer dikkat edilmesi gereken önemli husus ise Connection açık olmadan Transaction başlatılamaz.

Transaction nesnesi içerisinde SavePointler, Commit, Rollback gibi yapılarda mevcuttur.

# Dapper Stored Procedurler İle Çalışmak :

Tek fark normal şartlar altında Ado.Net te nasıl Prosedürler ile çalışıyorsak Burada da aynı mantıkla çalışılması gerekmektedir. İlgili Querylerin içerisinde Commantype belirterek sorun çözümlenebilir.

metin, yazı tipi, ekran görüntüsü içeren bir resim

Açıklama otomatik olarak oluşturuldu

Örnek olarak bu sorgu verilebilir. Dikkat edilmesi gereken bir hususta burda anonymous type içerisindeki değişken adları Procedure içerisindeki ile aynı olması gerekmektedir .

# Dapper.Contrib

Dapper.Contrib = Dapper in genişletilmiş ve geliştirilmiş versiyonudur. Entity Frameworkteki gibi SQL kodu yazmadan query yazabilmenize olanak tanır. İlgili süreç için **Dapper.Contrib** kütüphanesi ve **SqlConnection** kütüphanesinin yüklenmesi gerekmektedir.

metin, yazılım, multimedya yazılımı, ekran görüntüsü içeren bir resim

Açıklama otomatik olarak oluşturuldu

metin, ekran görüntüsü, yazılım, multimedya yazılımı içeren bir resim

Açıklama otomatik olarak oluşturuldu

Dapper Contrib içerisinde dapper da mevcuttur. Dapper Contrib kodsal maliyetten olabildiğince bizleri kurtarmaktadır.

metin, ekran görüntüsü, yazılım, multimedya yazılımı içeren bir resim

Açıklama otomatik olarak oluşturuldu

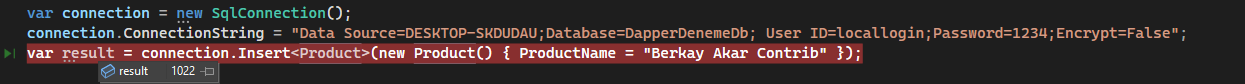
**Get** methodu içerisine verilen sınıfın veritabanı üzerinde ilgili id ye sahip nesneyi arar. Bulabilirse sınıfı çevirir, bulamazsa null çevirir hata vermez. (Veritabanından Producst yazmasına ragmen sınıfın adını product yazdım buldu).

Bir sonraki method ise **GetAll** Methodudur. Adından da anlaşılacağı üzere **GetAll** bizlere tüm nesneleri liste şeklinde getirebilmeyi sağlar.

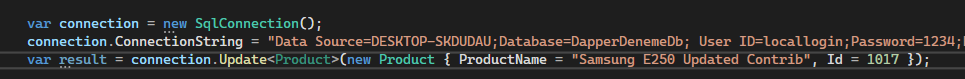
metin, ekran görüntüsü içeren bir resim

Açıklama otomatik olarak oluşturuldu

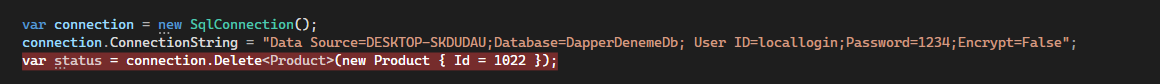
Bir sonraki incelenecek method ise insert methodudur. Aşağıda örneği verilmiştir. Output olarak eklenen değerin id değerini verir.



Bir sonraki aşamada ise Update methodunu inceleyelim.



Output olarak güncelleme işleminin başarılı olup olmadığının bilgisini vermektedir. Id değeri vermezsen update işlemi yapmaz. Hata vermez ama false döner.



Silme işlemi için kullanılmaktadır. Diğer parametreleri baz almaz sadece id ile iş yapar.

metin, yazı tipi, ekran görüntüsü, grafik içeren bir resim

Açıklama otomatik olarak oluşturuldu

Ilgili tablodaki tüm ürünleri siler.

* **Get<model>(dynamic id)** : id değeri verilen nesneyi çevirir. Transaction alabilir ve içerisine sadece id değeri alabilir parametrede birde transaction.
* **GetAll() :** görüldüğü üzere ilgili nesneye ait tüm kayıtların getirilmesini sağlar.
* **Insert<Class>(new Class (){ prop1 = … , prop2 = ..}) :** İnsertişlemine yarar. Bir diğer dikkat edilmesi gereken kısım ise ilişkiler için belirtilmiş olan navigation propertyler hata atmaya sebebiyet verir. Sonuç olarak ise eklenen değerin bizlere id değerini output olarak verir.
* **Update<Product>(new Product {id = 1 , ….}):** Update işlemi sonucunu output olarak verir. Id değeri zorunlu.
* **Delete<Product>(new Product{id = 1 , ….}) :** Silmeişlemini gerçekleştirir. Silme işlemi başarılı ise geriye bool olarak true döner. Id parametresini baz alır.
* **DeleteAll<Entity>() :** İlgili tablo için tüm kayıtları siler.