ADO.NET (ActiveX Data Objects for .NET), .NET Framework tarafından sağlanan bir veri erişim teknolojisidir. ADO.NET, veritabanlarına bağlanmak, veri alışverişi yapmak ve verileri işlemek için kullanılır. ADO.NET, Microsoft'un .NET Framework'ünü destekleyen bir API setidir ve genellikle ASP.NET gibi web uygulamalarında ve Windows Forms gibi masaüstü uygulamalarında kullanılır.

ADO.NET'in Temel Bileşenleri

1-Connection (Bağlantı):

.Veritabanına bağlanmak için kullanılan sınıftır.

.SqlConnection, OleDbConnection, OracleConnection gibi farklı veritabanı sağlayıcıları için özelleştirilmiş bağlantı sınıfları bulunur.

.Bağlantı açma, kapatma ve durum yönetimi işlemleri yapılır.

SqlConnection Kullanımı:

SqlConnection sınıfı, SQL Server veritabanına bağlanmak için kullanılır. Bağlantı açma, kapatma ve durum yönetimi gibi işlemleri gerçekleştirir.

```
using System;

using System.Data.SqlClient;

public class Program

{

   public static void Main()

   {

      // Veritabanı bağlantı dizesi

      string connectionString = "Server=myServerAddress;Database=myDataBase;User
Id=myUsername;Password=myPassword;";

      // SqlConnection nesnesi oluşturma

      using (SqlConnection connection = new SqlConnection(connectionString))

      {

         try

         {
```

```
            // Bağlantıyı açma
            connection.Open();
            Console.WriteLine("Bağlantı açıldı.");
            // Veritabanı işlemleri burada yapılır
            // Bağlantıyı kapatma
            connection.Close();
            Console.WriteLine("Bağlantı kapatıldı.");
        }
        catch (Exception ex)
        {
            Console.WriteLine("Hata: " + ex.Message);
        }
    }
  }
}
```

Yukarıdaki örnekte:

.SqlConnection sınıfı, SQL Server veritabanına bağlanmak için kullanılmıştır.

.connectionString değişkeni, bağlantı bilgilerini içerir (sunucu adı, veritabanı adı, kullanıcı adı, parola).

.SqlConnection nesnesi using bloğu içinde tanımlanmış ve otomatik olarak Dispose() metodu çağrılarak temizlenmiştir. Bu, kaynakların güvenli bir şekilde serbest bırakılmasını sağlar.

.Open() metoduyla bağlantı açılmış ve Close() metoduyla bağlantı kapatılmıştır.

.Hata durumlarında try-catch bloğu kullanılarak hata yönetimi sağlanmıştır.

Bu örnek, SqlConnection sınıfının temel kullanımını ve bağlantı işlemlerini göstermektedir. Bağlantı dizesi ve diğer parametreler kendi ortamınıza göre ayarlanmalıdır (Server, Database, User Id, Password vb.).

2-Command (Komut):

.Veritabanında sorgu veya komutları yürütmek için kullanılır.

.SqlCommand, OleDbCommand, OracleCommand gibi farklı veritabanı sağlayıcıları için özelleştirilmiş komut sınıfları bulunur.

.SELECT, INSERT, UPDATE, DELETE gibi SQL ifadeleri veya saklı prosedürler komut nesnesiyle yürütülür.

SqlCommand Kullanımı:

```
using System;

using System.Data;

using System.Data.SqlClient;

public class Program

{

   public static void Main()

   {

      // Veritabanı bağlantı dizesi

      string connectionString = "Server=myServerAddress;Database=myDataBase;User
Id=myUsername;Password=myPassword;";

      // SQL sorgusu

      string sqlQuery = "SELECT * FROM Products";

      // SqlConnection ve SqlCommand nesneleri

      using (SqlConnection connection = new SqlConnection(connectionString))

      using (SqlCommand command = new SqlCommand(sqlQuery, connection))

      {

         try

         {

            // Bağlantıyı açma

            connection.Open();

            Console.WriteLine("Bağlantı açıldı.");

            // SqlCommand nesnesiyle sorguyu yürütme
```

```
            SqlDataReader reader = command.ExecuteReader();
            // Verileri okuma
            while (reader.Read())
            {
                Console.WriteLine($"ProductId: {reader["ProductId"]}, Name: {reader["Name"]}, Price:
{reader["Price"]}");
            }
            // DataReader'ı kapatma
            reader.Close();
        }
        catch (Exception ex)
        {
            Console.WriteLine("Hata: " + ex.Message);
        }
        finally
        {
            // Bağlantıyı kapatma
            connection.Close();
            Console.WriteLine("Bağlantı kapatıldı.");
        }
    }
  }
}
```

Yukarıdaki örnekte:

.SqlCommand nesnesi, SqlConnection nesnesi ile birlikte kullanılmıştır.

.sqlQuery değişkeni, yürütülecek SQL sorgusunu içerir (SELECT * FROM Products).

.SqlCommand nesnesi, sqlQuery ve SqlConnection nesnesi ile başlatılmıştır.

.ExecuteReader() metodu ile sorgu veritabanında yürütülmüş ve sonuçlar SqlDataReader nesnesi ile okunmuştur.

.SqlDataReader ile veri okuma işlemi yapılmış ve sonuçlar ekrana yazdırılmıştır.

.Close() ve Dispose() metotları kullanılarak kaynakların temizlenmesi sağlanmıştır (using bloğu ile otomatik olarak yapılır).

Bu örnek, SqlCommand sınıfının temel kullanımını ve SQL sorgularını nasıl yürütebileceğinizi göstermektedir. Bağlantı dizesi ve sorgu kendi ortamınıza göre ayarlanmalıdır (Server, Database, User Id, Password vb.).

3-DataReader (Veri Okuyucu):

.Veritabanından okunan verileri satır satır okumak için kullanılır.

.SqlDataReader, OleDbDataReader, OracleDataReader gibi farklı veritabanı sağlayıcıları için özelleştirilmiş veri okuyucu sınıfları bulunur.

.Veri okuma işlemleri hızlı ve performanslıdır, çünkü veriler doğrudan veritabanından belleğe yüklenmeden okunur.

SqlDataReader Kullanımı:

```
using System;

using System.Data;

using System.Data.SqlClient;

public class Program

{

   public static void Main()

   {

      // Veritabanı bağlantı dizesi

      string connectionString = "Server=myServerAddress;Database=myDataBase;User
Id=myUsername;Password=myPassword;";

      // SQL sorgusu

      string sqlQuery = "SELECT * FROM Products";
```

```
        // SqlConnection ve SqlCommand nesneleri
        using (SqlConnection connection = new SqlConnection(connectionString))
        using (SqlCommand command = new SqlCommand(sqlQuery, connection))
        {
            try
            {
                // Bağlantıyı açma
                connection.Open();
                Console.WriteLine("Bağlantı açıldı.");
                // SqlDataReader nesnesiyle sorguyu yürütme
                SqlDataReader reader = command.ExecuteReader();
                // Verileri okuma
                while (reader.Read())
                {
                    Console.WriteLine($"ProductId: {reader["ProductId"]}, Name: {reader["Name"]}, Price:
{reader["Price"]}");
                }
                // DataReader'ı kapatma
                reader.Close();
            }
            catch (Exception ex)
            {
                Console.WriteLine("Hata: " + ex.Message);
            }
            finally
            {
```

```
            // Bağlantıyı kapatma
            connection.Close();
            Console.WriteLine("Bağlantı kapatıldı.");
        }
    }
  }
}
```

Yukarıdaki örnekte:

.SqlDataReader nesnesi, SqlCommand nesnesi ile birlikte kullanılmıştır.

.SqlDataReader nesnesi, ExecuteReader() metodu ile sorgu sonuçlarını okumak için oluşturulmuştur.

.Read() metodu, sorgudan bir sonraki satırı okur. Her satır reader nesnesi tarafından okunur ve ProductId, Name, Price gibi sütunların değerleri ekrana yazdırılır.

.Close() metodu ile SqlDataReader nesnesi kapatılarak kaynaklar serbest bırakılır.

.finally bloğunda bağlantı kapatılır ve kaynakların temizlenmesi sağlanır.

Bu örnek, SqlDataReader sınıfının temel kullanımını ve veritabanından veri okuma işlemlerini göstermektedir. Bağlantı dizesi ve sorgu kendi ortamınıza göre ayarlanmalıdır (Server, Database, User Id, Password vb.).

4-DataAdapter (Veri Adaptörü):

.Veritabanından veri okumak, veri değişikliklerini veritabanına uygulamak veya veritabanına veri eklemek için kullanılır.

.SqlDataAdapter, OleDbDataAdapter, OracleDataAdapter gibi farklı veritabanı sağlayıcıları için özelleştirilmiş veri adaptörü sınıfları bulunur.

.DataSet ve DataTable gibi veri yapılarıyla birlikte kullanılarak veri işleme işlevselliği sağlar.

SqlDataAdapter Kullanımı:

```
using System;
using System.Data;
using System.Data.SqlClient;
```

```
public class Program
{
    public static void Main()
    {
        // Veritabanı bağlantı dizesi
        string connectionString = "Server=myServerAddress;Database=myDataBase;User Id=myUsername;Password=myPassword;";
        // SQL sorgusu
        string sqlQuery = "SELECT * FROM Products";
        // SqlConnection ve SqlDataAdapter nesneleri
        using (SqlConnection connection = new SqlConnection(connectionString))
        using (SqlDataAdapter adapter = new SqlDataAdapter(sqlQuery, connection))
        {
            try
            {
                // DataSet oluşturma
                DataSet dataSet = new DataSet();
                // Verileri adapter ile doldurma
                adapter.Fill(dataSet, "Products");
                // DataTable seçme
                DataTable productsTable = dataSet.Tables["Products"];
                // Verileri ekrana yazdırma
                foreach (DataRow row in productsTable.Rows)
                {
                    Console.WriteLine($"ProductId: {row["ProductId"]}, Name: {row["Name"]}, Price: {row["Price"]}");
```

```
            }
        }
        catch (Exception ex)
        {
            Console.WriteLine("Hata: " + ex.Message);
        }
    }
  }
}
```

Yukarıdaki örnekte:

.SqlDataAdapter nesnesi, SqlConnection nesnesi ile birlikte kullanılmıştır.

.DataSet nesnesi oluşturularak SqlDataAdapter ile veritabanından veri okunmuştur.

.Fill() metodu, SqlDataAdapter ile DataSet ve DataTable yapısına verilerin doldurulmasını sağlar.

.DataSet.Tables["Products"] ile Products adlı DataTable seçilmiştir.

.foreach döngüsü ile DataTable.Rows koleksiyonundaki her bir satır DataRow nesnesi olarak alınmış ve ekrana yazdırılmıştır.

Bu örnek, SqlDataAdapter sınıfının temel kullanımını ve veri okuma işlemlerini göstermektedir. Bağlantı dizesi ve sorgu kendi ortamınıza göre ayarlanmalıdır (Server, Database, User Id, Password vb.).

5-DataSet ve DataTable:

.Veri saklama ve işleme için kullanılan veri yapılarıdır.

.DataSet, birden çok DataTable içerebilir ve veritabanından gelen verileri geçici olarak depolar.

.DataTable, veritabanından gelen tek bir tabloyu temsil eder.

DataSet ve DataTable Kullanımı:

```
using System;
using System.Data;
using System.Data.SqlClient;
```

```
public class Program
{
    public static void Main()
    {
        // Veritabanı bağlantı dizesi
        string connectionString = "Server=myServerAddress;Database=myDataBase;User Id=myUsername;Password=myPassword;";
        // SQL sorgusu
        string sqlQuery = "SELECT * FROM Products";
        // SqlConnection ve SqlDataAdapter nesneleri
        using (SqlConnection connection = new SqlConnection(connectionString))
        using (SqlDataAdapter adapter = new SqlDataAdapter(sqlQuery, connection))
        {
            try
            {
                // DataSet oluşturma
                DataSet dataSet = new DataSet();
                // Verileri adapter ile doldurma
                adapter.Fill(dataSet, "Products");
                // DataTable seçme
                DataTable productsTable = dataSet.Tables["Products"];
                // Verileri ekrana yazdırma
                foreach (DataRow row in productsTable.Rows)
                {
                    Console.WriteLine($"ProductId: {row["ProductId"]}, Name: {row["Name"]}, Price: {row["Price"]}");
```

```
            }
        }
        catch (Exception ex)
        {
            Console.WriteLine("Hata: " + ex.Message);
        }
    }
  }
}
```

Yukarıdaki örnekte:

.DataSet ve DataTable kullanılarak veri saklama ve işleme işlevselliği sağlanmıştır.

.DataSet nesnesi oluşturularak SqlDataAdapter ile veritabanından veri okunmuştur.

.Fill() metodu, SqlDataAdapter ile DataSet ve DataTable yapısına verilerin doldurulmasını sağlar.

.DataSet.Tables["Products"] ile Products adlı DataTable seçilmiştir.

.foreach döngüsü ile DataTable.Rows koleksiyonundaki her bir satır DataRow nesnesi olarak alınmış ve ekrana yazdırılmıştır.

Bu örnek, DataSet ve DataTable yapılarının temel kullanımını ve veri işleme işlevselliğini göstermektedir. Bağlantı dizesi ve sorgu kendi ortamınıza göre ayarlanmalıdır (Server, Database, User Id, Password vb.).

6-Transaksiyon Yönetimi:

.Birden fazla işlemi bir arada yapmak ve bu işlemleri kontrol altında tutmak için transaksiyonlar kullanılır.

.Veritabanında işlem başlatma, işlemi tamamlama veya iptal etme işlemleri gerçekleştirilir.

```
using System;
using System.Data;
using System.Data.SqlClient;
public class Program
{
```

```
    public static void Main()
    {
        // Veritabanı bağlantı dizesi
        string connectionString = "Server=myServerAddress;Database=myDataBase;User
Id=myUsername;Password=myPassword;";
        // SqlConnection nesnesi ve SqlTransaction nesnesi oluşturma
        using (SqlConnection connection = new SqlConnection(connectionString))
        {
            // Bağlantıyı açma
            connection.Open();
            Console.WriteLine("Bağlantı açıldı.");
            // SqlTransaction nesnesi oluşturma
            SqlTransaction transaction = null;

            try
            {
                // Transaksiyon başlatma
                transaction = connection.BeginTransaction();
                // SqlCommand nesnesi ve transaksiyonu belirtme
                SqlCommand command = connection.CreateCommand();
                command.Transaction = transaction;
                // İlk SQL komutu
                command.CommandText = "INSERT INTO Products (Name, Price) VALUES ('Keyboard', 29.99)";
                command.ExecuteNonQuery();
                Console.WriteLine("Keyboard ürünü eklendi.");
                // İkinci SQL komutu
```

```
    command.CommandText = "UPDATE Products SET Price = 39.99 WHERE Name = 'Mouse'";
    command.ExecuteNonQuery();
    Console.WriteLine("Mouse ürününün fiyatı güncellendi.");
    // Transaksiyonu tamamlama
    transaction.Commit();
    Console.WriteLine("Transaksiyon başarıyla tamamlandı.");
}
catch (Exception ex)
{
    Console.WriteLine("Hata: " + ex.Message);

    try
    {
        // Hata durumunda transaksiyonu geri al
        transaction?.Rollback();
        Console.WriteLine("Transaksiyon geri alındı.");
    }
    catch (Exception rollbackEx)
    {
        Console.WriteLine("Rollback hatası: " + rollbackEx.Message);
    }
}
finally
{
    // Bağlantıyı kapatma
```

```
            connection.Close();
            Console.WriteLine("Bağlantı kapatıldı.");
        }
    }
  }
}
```

Yukarıdaki örnekte:

.SqlConnection ve SqlTransaction nesneleri kullanılarak transaksiyon yönetimi gerçekleştirilmiştir.

.connection.BeginTransaction() ile bir transaksiyon başlatılmıştır.

.command.Transaction = transaction; ile SqlCommand nesnesine transaksiyon atanmıştır.

.İlk olarak INSERT komutu ile Products tablosuna bir ürün eklenmiştir.

.Ardından UPDATE komutu ile Mouse adlı ürünün fiyatı güncellenmiştir.

.transaction.Commit() ile transaksiyon başarıyla tamamlanmıştır.

.transaction.Rollback() ile hata durumunda transaksiyon geri alınmıştır.

.connection.Close() ile bağlantı kapatılmıştır.

Bu örnekte, transaksiyonların nasıl başlatılacağı, işlemlerin nasıl yürütüleceği ve başarı durumunda veya hata durumunda nasıl işlem yapılacağı gösterilmektedir. Bağlantı dizesi ve SQL komutları kendi ortamınıza göre ayarlanmalıdır (Server, Database, User Id, Password vb.).

ADO.NET Avantajları

.Performans: Veri okuma işlemleri hızlıdır ve bellek kullanımı düşüktür.

.Veritabanı Bağımsızlığı: Farklı veritabanı sağlayıcıları için destek sağlar.

.Esneklik: SQL sorguları veya saklı prosedürler kullanarak veritabanı işlemleri gerçekleştirilebilir.

.Geniş Kullanım Alanı: Web uygulamaları, masaüstü uygulamaları ve servislerde kullanılabilir.

ADO.NET, .NET Framework ile birlikte gelen güçlü bir veri erişim teknolojisidir ve .NET geliştiricilerinin geniş bir veritabanı sağlayıcı desteğiyle veritabanı işlemlerini kolayca gerçekleştirmesini sağlar.