

# **מבוא ל- ADO .NET**

# מבוא ל - ADO.NET

## תוכן עניינים

2.....	ActiveX Data Objects – ADO.NET
3.....	חיבור ל - DB באמצעות Visual studio
5.....	שמירת ה - connection string
7.....	מודל ה - Connected
10.....	מודל ה - Disconnected
14.....	DataBinding
15.....	מושגים נוספים ב - ADO.NET
21.....	Typed DataSet and TableAdapter
24.....	מודל השכבות לעבודה נכונה מול DB
25.....	מבוא ל - Entity Framework

## ADO.NET – ActiveX Data Objects

ADO.NET הינה ספריה של רכיבים שמטרתם לעבוד עם מידע. השימוש העיקרי של ADO.NET הוא עבודה מול Data base לשליפה ועדכון של מידע.

### ADO.NET מחולק לשתי גישות עבודה:

- **Connected** – גישה המאפשרת לעבוד מול DB במצבים בהם אנו יכולים להתחבר ל - DB לדוגמא: מערכת און ליין שכל עדכון מתעדכן מיד ב - DB.
- **Disconnected** – גישה המאפשר לשלוח נתונים מ - DB לעבוד איתם בתוכנית ובהמשך לסנכרן אותם מול ה - DB. גישה זו מתאימה יותר במקרים בהם אין חיבור רציף ל - DB. לדוגמא: מערכת הפצת מוצרים כך שלכל מפיץ יש מכשיר המכיל את סידור העבודה שלו לאותו היום ובזמן העבודה הוא צריך לעדכן מה הוא ביצע. בזמן העבודה הוא נמצא מחוץ למשרד ולכן לא תמיד יש לו חיבור ל - DB המרכזי.

### **Data base provider:**

לכל ספק DB (Microsoft, Oracle וכד') יש את שיטת התחברות שלו מול ה - DB (ברמת התקשורת). כל ספק DB מספק רכיבים אשר דורשים התקנה היכן שנמצאת תוכנת לקוח שמעוניינת להתחבר ל - DB. כדי שהמפתחים לא יצטרכו ללמוד מחלקות שונות לעבודה מול DB שונים, קבעו תקן ממנו יורשות כל המחלקות לעבודה מול DB, כאשר כל מחלקה מתאימה לעבודה מול סוג אחר של DB והמימוש הפנימי שלה הוא שונה (פולימורפיזם). לכל מחלקה כזו יש קידומת המרמזת על סוג ה - DB מולו היא יודעת לעבוד.

לדוגמא: **DBConnection** – מחלקת בסיס להתחברות ל - DB וממנה יורשים **SqlConnection** לעבודה מול **SQL Server** ו - **OracleConnection** לעבודה מול **Oracle** (וכך לכל סוגי ה - DB).

כל שמות המחלקות הכלליות (אבסטרקטיות) מתחילות ב - **DB**, כל שמות המחלקות לעבודה מול **SQLServer** מתחילות ב **SQL** וכך הלאה.

כל קבוצת מחלקות כזו שמתאימה לעבודה מול DB מסויים נמצאת בתוך namespace:

- **System.Data.SqlClient** – לעבודה מול **SqlServer**
- **System.Data.OracleClient** – לעבודה מול **Oracle**
- **System.Data.OleDb** – מתאים לכל סוגי ה - DB, משמש בעיקר לעבודה מול **Access**
- ויש כמובן עוד הרבה

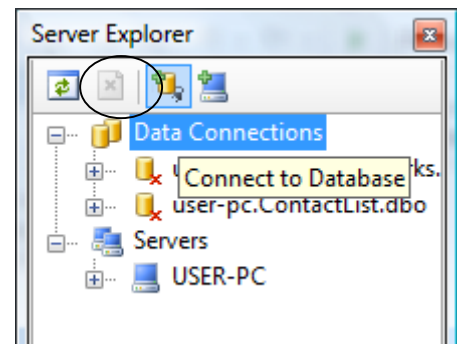
בהמשך החוברת נתייחס לעבודה מול **SqlServer** כך חשוב להבין ששיטת העבודה דומה מול כל DB שנעבוד מולו.

כל הדוגמאות בחוברת זו יתבצעו מול DB לדוגמא בשם **Northwind** שמגיע עם **SqlServer**.

## חיבור ל - DB באמצעות Visual studio

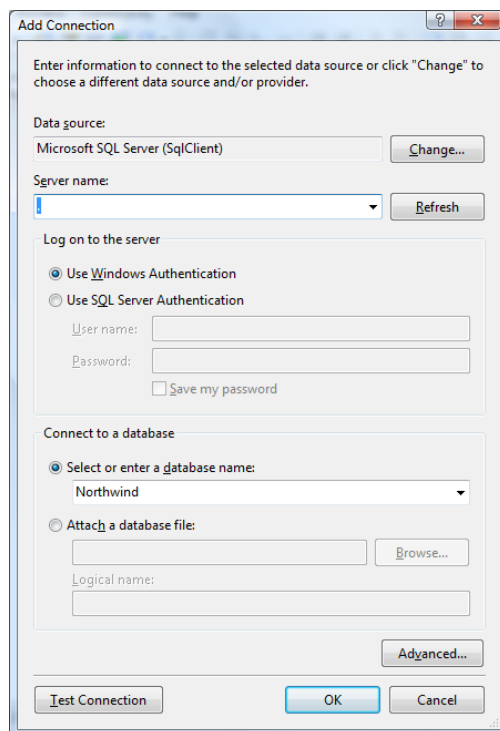
ניתן להתחבר ל - DB מתוך סביבת ה - Visual Studio. המטרה היא שבזמן פיתוח נוכל לראות את מצב ה - DB ולעדכן אותו. חיבור זה אינו קשור לתוכנית שלנו ואינו משפיע עליה. בנוסף ניתן בשיטה זו להעתיק אל התוכנית שלנו את ה - connection string שמיצג את פרטי החיבור ל - DB (מיקום, שם, פרטי הזדהות וכד').

יש לגשת לחלון ה - Server Explorer (במידה והוא לא מופיע אפשר להציג אותו מתפריט View) ולהוסיף חיבור חדש ל - DB באמצעות הלחצן connect to database:

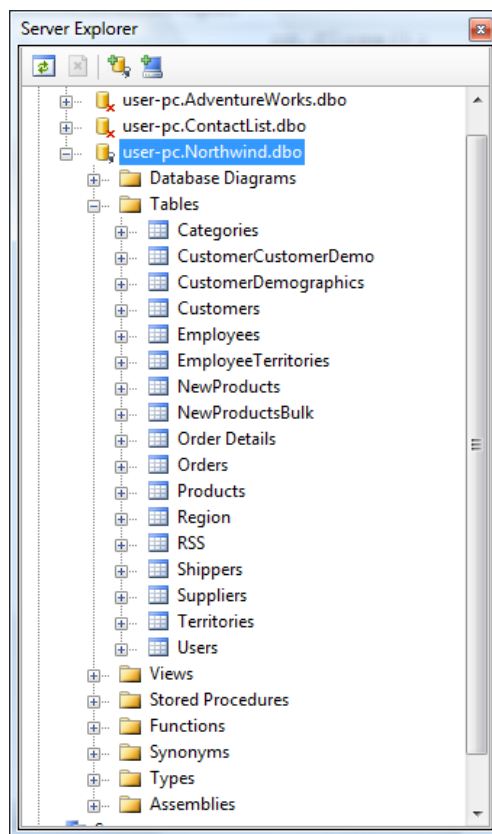


בחלון שיפתח יש לבחור:

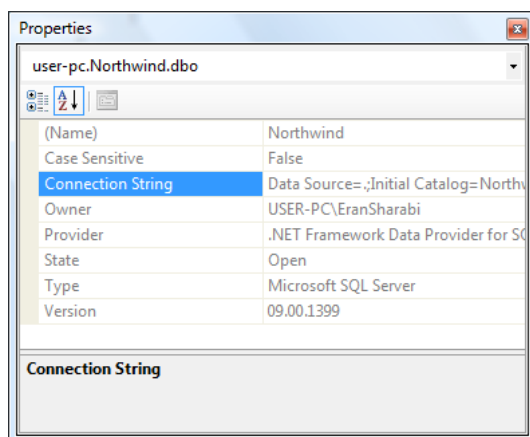
- סוג ה - DB (Data source)
- המיקום שלו Server name (כתובת IP / שם מחשב) או sql\express\שם המחשב. לעיתים אפשר לרשום כחלופה . נקודה או localhost כדי לציין שעובדים עם שרת מקומי
- שיטת ההזדהות (Windows / SqlServer)
- שם ה - DB (Northwind)
- בלחצן Advanced ישנם עוד מאפיינים שאפשר לקבוע ל - connection
- ניתן לבדוק את ה - connection באמצעות הלחצן Test Connection



לאחר לחיצה על OK יתווסף החיבור לחלון:



בחלון ה - properties ניתן לראות את מאפייני החיבור כולל ה - connection string (ניתן להעתיק אותו משם לתוכנית שלנו):



**תרגיל 1:**



הוסף חיבור ל - Northwind ונסה לבצע פעולות על ה DB דרך חלון ה - server explorer

## שמירת ה - connection string

את ה - connection string נהוג להחזיק במקום אחד בתוכנית, כדי שאם נצטרך לשנות אותו נשנה רק במקום אחד. כמו כן, מומלץ להחזיק את ה - connection string באופן חיצוני לתוכנית כדי שעדכון ב - connection string לא יצריך קימפול התוכנית מחדש. לכן נחזיק את ה - connection string בקובץ config חיצוני שמגיע יחד עם התוכנית ושמנו app.config

קובץ זה כתוב בפורמט xml ובתוכו ניתן להכניס את ה - connection string בצורה הבאה, כאשר השם הוא שם חופשי שנשתמש בו בהמשך:

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <startup>
    <supportedRuntime version="v4.0" sku=".NETFramework,Version=v4.6.1" />
  </startup>
  <connectionStrings>
    <add name="NorthwindConStr"
          connectionString="Data Source=.;Initial
Catalog=Northwind;Integrated Security=True"
          providerName="System.Data.SqlClient"/>
  </connectionStrings>
</configuration>
```

מתוך התוכנית ניתן לקרוא את ה - connection string מהקובץ לפי השם בצורה הבאה:

```
string conStr =
ConfigurationManager.ConnectionStrings["NorthwindConStr"].Connec
tionString;
```

חשוב לשים לב שלפקודה זו צריך Reference ל - System.Configuration.dll



בנוסף חשוב לשים לב שלאחר קומפילציה הקובץ יהפוך להיות כשם ה - exe בתוספת config בסוף. לדוגמא אם שם התוכנית הוא MyApp.exe שם ה - config יהיה MyApp.exe.config. את הקובץ הזה חובה לשים ביחד עם קובץ ה - exe כאשר רוצים להריץ את התוכנית.

## תרגיל 2:



- שמור את ה - connection string מהתרגיל הקודם בתוך קובץ config והצג אותו על מסך ה - console.
- העתק את קובץ ה - exe והרץ אותו ללא שימוש ב - Visual studio. בצע עדכון ל - connection string והרץ exe בלי הידור מחדש

## מודל ה - Connected

גישה המאפשרת לעבוד מול DB במצבים בהם אנו יכולים להתחבר ל - DB.

למודל זה שותפות 3 מחלקות:

- Connection – מיצג את החיבור ל - DB
- Command – מאפשר להריץ פקודות SQL דרך ה - Connection
- DataReader – מאפשר לקרוא תוצאה של פקודה (שאלתה) שהופעלה מול ה - DB

### מאפיינים ופונקציות עקריות של המחלקה Connection:

- connectionString – מחרוזת החיבור ל - DB, מגדירה לאובייקט פרטים לגבי החיבור, כגון: שם ה - DB, מיקום, פרטי הזדהות וכד'
- Open() – פתיחת ה - connection מול ה - DB
- Close() – סגירת ה - connection

### מאפיינים ופונקציות עקריות של המחלקה Command:

- Connection – האובייקט connection דרכו תופעל הפקודה
- CommandText – משפט ה SQL שנרצה להריץ מול ה DB
- ExecuteReader() – הפעלת פקודה לשליפת מידע מה - DB, מחזירה DataReader
- ExecuteNonQuery() – הפעלת פקודה שאינה שולפת מידע מה - DB (עדכון, מחיקה וכד'), מחזירה כמה רשומות הושפעו מהפעולה
- ExecuteScalar() – הפעלת פקודה שמחזירה סקאלר (ערך בודד)

### מאפיינים ופונקציות עקריות של המחלקה DataReader:

- Read() – קריאת רשומה מה - DB
- Indexer [] – גישה לרשומה שנקראה מה - DB באמצעות מספר עמודה או שם

### דוגמא לקריאה מ - DB:

בדוגמא זו אנו מציגים על מסך ה - Console אל השמות הפרטיים ושמות המשפחה של כל העובדים

```
SqlConnection cn = new SqlConnection();
cn.ConnectionString =
    "Data Source=.;Initial Catalog=Northwind;Integrated
    Security=True";
SqlCommand com = new SqlCommand();
com.Connection = cn;
com.CommandText = "SELECT FirstName, LastName FROM Employees";

SqlDataReader reader = null;
```



```

try
{
    cn.Open();

    reader = com.ExecuteReader();

    while (reader.Read())
    {
        string fName = (string)reader["FirstName"];
        string lName = (string)reader["LastName"];

        Console.WriteLine("{0}, {1}", fName, lName);
    }
}
catch (Exception ex)
{
    Console.WriteLine(ex.Message);
}
finally
{
    if (reader != null)
        reader.Close();

    cn.Close();
}

```

### דוגמא לעדכון מידע ב - DB:

בדוגמא זו נעדכן את שם העובד שה - ID שלו הוא 7 ל - Eran:

```

SqlConnection cn = new SqlConnection();
cn.ConnectionString =
    "Data Source=.;Initial Catalog=Northwind;Integrated
Security=True";
SqlCommand com = new SqlCommand();
com.Connection = cn;
com.CommandText = "UPDATE Employees SET FirstName='Eran' WHERE
EmployeeID = 7";

```

```

try
{
    cn.Open();

    int rowsAffected = com.ExecuteNonQuery();
}

```

```

        Console.WriteLine("rowsAffected = {0}", rowsAffected);
    }
    catch (Exception ex)
    {
        Console.WriteLine(ex.Message);
    }
    finally
    {
        cn.Close();
    }
}

```

### תרגיל 3:

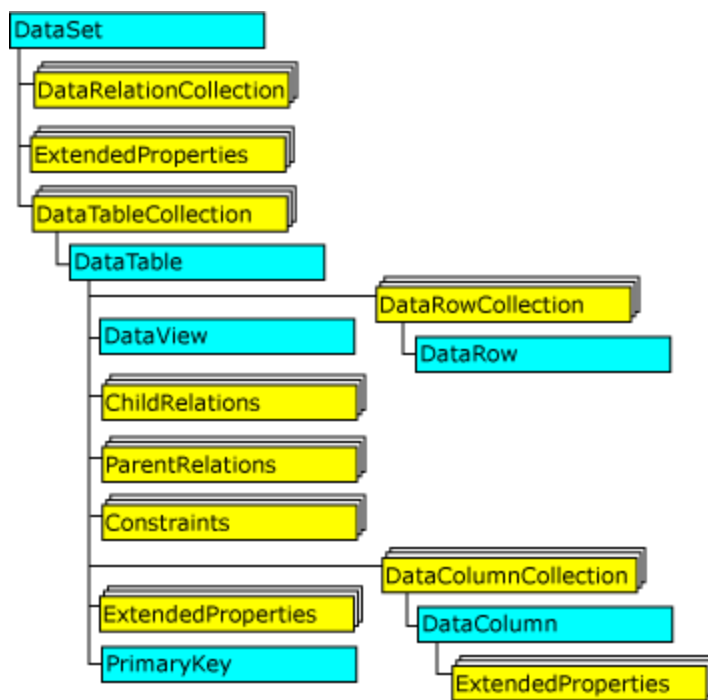


- בנה טופס שיוציג את רשימת ה - CategoryID מתוך הטבלה Categories בתוך ComboBox
- בבחירה של Category יש להציג את שמות כל המוצרים השייכים לקטגוריה מהטבלה Products

## מודל ה – Disconnected

גישה המאפשר לשלוח נתונים מ - DB לעבוד איתם בתוכנית ובהמשך לסנכרן אותם מול ה - DB. גישה זו מתאימה יותר במקרים בהם אין חיבור רציף ל - DB. בכדי לשמור את הנתונים בתוכנית יש להחזיק אותם בתוך מודל של אובייקטים שמייצגים DB בזיכרון התוכנית. בכדי לעבוד מול DB יש להשתמש באובייקט DataAdapter שידוע לסנכרן בין המודל נתונים לבין ה - DB.

### מודל האובייקטים:



DataSet – אוסף של טבלאות (Tables) וקשרים (Relations)

DataTable – טבלה, מכילה אוסף של עמודות (Columns) ורשומות (Rows)

DataColumn – מיצג עמודה בטבלה

DataRow – מיצג רשומה בטבלה

DataRelation – מיצג קשר בין טבלאות

נניח שיש אובייקט DataSet בשם ds המכיל המכיל 2 טבלאות: Categories, Products. להלן כמה דוגמאות לעבודה עם האובייקט:

גישה למספר הטבלאות שיש בתוך ה - DataSet:

ds.Tables.Count

גישה למספר הרשומות שיש בטבלה Products:

ds.Tables["Products"].Rows.Count

גישה לרשומה ראשונה בטבלה Categories ובתוכה לתא CategoryID:

ds.Tables["Categories"].Rows[0]["CategoryID"]

גישה לשם העמודה הראשונה בטבלה Categories:

ds.Tables["Categories"].Columns[0].ColumnName

### **:DataAdapter**

DataAdapter הוא למעשה מעטפת לאובייקטים שהכרנו במודל ה - connected המאפשר לסנכרן בין DataSet לבין ה - DB.

פונצקיות ומאפיינים עיקריים:

- Fill() – שליפת הנתונים מתוך ה - DB והעברתם ל - DataSet
- Update() – עדכון השינויים שנעשו ב - DataSet לתוך ה - DB
- SelectCommand – Command מכיל את פקודת ה - Select
- InsertCommand – Command המכיל את פקודת ה - Insert
- UpdateCommand – Command המכיל את פקודת ה - Update
- DeleteCommand – Command המכיל את פקודת ה - Delete

## :CommandBuilder

CommandBuilder מיצר את משפטי ה - Insert, Update, Delete עבור ה - DataAdapter לצורך עדכונים חזרה ל - DB.

דוגמא לשליפת נתונים מ - DB:

```
SqlDataAdapter ad = new SqlDataAdapter(  
    "SELECT EmployeeID,FirstName,LastName,Title FROM Employees",  
    "Data Source=.;Initial Catalog=Northwind;Integrated  
Security=True");  
  
DataSet ds = new DataSet();  
  
ad.Fill(ds, "Employees");
```

דוגמא להוספת רשומה לטבלה ב - DataSet:

```
DataRow row = ds.Tables["Employees"].NewRow();  
row["FirstName"] = "Eran";  
row["LastName"] = "Strong";  
row["Title"] = "EranStrongTitle";  
ds.Tables["Employees"].Rows.Add(row);
```

דוגמא למחיקת רשומה מטבלה ב - DataSet:

```
ds.Tables["Employees"].Rows[0].Delete();
```

דוגמא לעדכון השדה FirstName ברשומה מספר 7:

```
ds.Tables["Employees"].Rows[6]["FirstName"] = "EranSupper";
```

דוגמא לעדכון השינויים שנעשו חזרה לתוך ה - DB:

```
SqlCommandBuilder builder = new SqlCommandBuilder(ad);  
ad.Update(ds, "Employees");
```

דוגמא להצגת הנתונים שבתוך הטבלה על מסך ה - Console:

```
foreach (DataColumn col in ds.Tables["Employees"].Columns)
{
    Console.WriteLine("{0,-15}|", col.ColumnName);
}

Console.WriteLine("\b ");
Console.WriteLine("{0,-15}+{0,-15}+{0,-15}+{0,-15}", "-----
-----");

foreach (DataRow row in ds.Tables["Employees"].Rows)
{
    foreach (DataColumn col in ds.Tables["Employees"].Columns)
    {
        Console.WriteLine("{0,-15}|", row[col]);
    }

    Console.WriteLine("\b ");
    Console.WriteLine("{0,-15}+{0,-15}+{0,-15}+{0,-15}", "-----
-----");
}
```

#### תרגיל 4:



- בנה טופס עם אפשרות להוספה מחיקה ועדכון של רשומות מהטבלה Region
- ב - ListBox השמאלי תוצג רשימת ה - Regions
- כאשר המשתמש יבחר ערך ב - ListBox יש להציג לו את פרטי הרשומה בתיבות הטקסט בצד ימין
- להחצנים Insert, Update, Delete יעדכנו את השינוי ב - DataSet
- לחצן Save to DB ישמור את כל השינויים ל - DB

---

## DataBinding

---

תהליך בו מחברים בין מקור מידע כלשהוא לפקד, זאת כדי שלא נצטרך לכתוב הרבה קודים כדי להציג את הנתונים. אחד השימושים העקריים של DataBinding הוא בחיבור בין DataSet / DataTable לפקד שיציג אותו.

מספר מאפיינים הקיימים בפקדים שתומכים ב - DataBinding:

- DataSource – מקור המידע אותו נרצה להציג על הפקד בד"כ DataTable
- DisplayMember – הרשומה שתוצג מתוך מקור המידע
- ValueMember – הרשומה שתהיה Value מתוך המידע שמוצג בד"כ ה - Primary key
- SelectedValue – ה - Value של הרשומה שנבחרה

דוגמא להצגה של הטבלה Employees על ListBox כאשר העמודה שתוצג היא FirstName ועמודת ה Value היא EmployeeID:

```
lstEmployees.DataSource = ds.Tables["Employees"];  
lstEmployees.DisplayMember = "FirstName";  
lstEmployees.ValueMember = "EmployeeID";
```

דוגמא להצגה של ה - EmployeeID של הרשומה שנבחרה ע"י המשתמש:

```
MessageBox.Show(lstEmployees.SelectedValue.ToString());
```

### פקד ה - DataGridView:

פקד זה תומך ב - DataBinding הכולל הצגה ועריכה של נתונים בפורמט של טבלה. כל שינוי בפקד ישנה את הערך גם ב - DataSet / DataTable

דוגמא להצגת טבלה בתוך DataGridView:

```
dataGridView1.DataSource = ds.Tables["Employees"];
```

	EmployeeID	LastName	FirstName	Title	TitleOfCourtesy	BirthDate	HireDate
▶	1	Davolio	Nancy1	Sales Represent...	Ms.	08/12/1948	01/05/
	2	Fuller	Andrew	Vice President, S...	Dr.	19/02/1952	14/08/
	3	Leverling	Janet	Sales Represent...	Ms.	30/08/1963	01/04/
	4	Peacock	Margaret	Sales Represent...	Mrs.	19/09/1937	03/05/
	5	Buchanan	Steven	Sales Manager	Mr.	04/03/1955	17/10/
	6	Suyama	Michael	Sales Represent...	Mr.	02/07/1963	17/10/
	7	King	eran	Sales Represent...	Mr.	29/05/1960	02/01/
	8	Callahan	Laura	Inside Sales Coor...	Ms.	09/01/1958	05/03/
	9	Dodsworth	Anne	Sales Represent...	Ms.	27/01/1966	15/11/
	27	asdf	asdfas				
*							

## מושגים נוספים ב – ADO.NET

### עבודה מול Stored Procedures

בחלק גדול מה DB ישנם פעולות אשר נמצאות בתוך Stored Procedures. בפרק זה נדון איך להפעיל Stored Procedures ולקבל מהן תשובות לתוכנית.

בכדי להפעיל SP יש ליצור connection, command בשיטה הרגילה עם מספר הבדלים:

- ה - CommandText יהיה שם הפרוצדורה
- להגדיר את המאפיין CommandType של ה - Command לערך `CommandType.StoredProcedure`
- אם הפרוצדורה מקבלת פרמטרים יש להגדיר אותם ולהכניס לתוכם ערכים
- במידה והפרוצדורה מקבלת פרמטרים output יש להגדיר אותם ולקחת את ערכם לאחר הפעלת הפרוצדורה

ניתן גם להגדיר את ה - Commands של ה - DataAdapter כך שיפעילו SP.



דוגמא להפעלת SP ללא פרמטרים:

```
SqlConnection cn;  
SqlCommand com;  
cn = new SqlConnection(  
    "Data Source=.;Initial Catalog=Northwind;Integrated  
    Security=True");  
com = new SqlCommand("Ten Most Expensive Products", cn);  
com.CommandType = CommandType.StoredProcedure;  
  
cn.Open();  
SqlDataReader reader = com.ExecuteReader();  
  
while (reader.Read())  
{  
    listBox1.Items.Add(reader["TenMostExpensiveProducts"] +  
        "-" + reader["UnitPrice"]);  
}  
  
cn.Close();
```

דוגמא להפעלת SP שמקבלת פרמטרים:

```
SqlConnection cn;  
SqlCommand com;  
cn = new SqlConnection(  
    "Data Source=.;Initial Catalog=Northwind;Integrated  
Security=True");  
com = new SqlCommand("Sales by Year", cn);  
com.CommandType = CommandType.StoredProcedure;  
  
SqlParameter p1 = new SqlParameter("@Beginning_Date",  
    SqlDbType.DateTime);  
p1.Value = new DateTime(1996, 12, 1);  
com.Parameters.Add(p1);  
  
SqlParameter p2 = new SqlParameter("@Ending_Date",  
    SqlDbType.DateTime);  
p2.Value = new DateTime(1997, 1, 1);  
com.Parameters.Add(p2);  
  
SqlDataAdapter ad = new SqlDataAdapter();  
ad.SelectCommand = com;  
  
DataSet ds = new DataSet();  
ad.Fill(ds, "SalesByYear");  
  
dataGridView1.DataSource = ds.Tables["SalesByYear"];
```

דוגמא להפעלת SP שמחזירה ערך (פרמטר output), פרוצדורה זו לא קיימת במסד הנתונים Northwind והקוד שלה מצורף בהמשך:

```
SqlConnection cn;
SqlCommand com;
cn = new SqlConnection(
    "Data Source=.;Initial Catalog=Northwind;Integrated
    Security=True");
com = new SqlCommand("CountEmployees", cn);
com.CommandType = CommandType.StoredProcedure;

SqlParameter p1 = new SqlParameter("@Count", SqlDbType.Int);
p1.Direction = ParameterDirection.Output;
com.Parameters.Add(p1);

cn.Open();
com.ExecuteNonQuery();
cn.Close();

MessageBox.Show(com.Parameters["@Count"].Value.ToString());
```

קוד הפרוצדורה:

```
ALTER PROCEDURE dbo.CountEmployees
    (@count int output)

AS
    select @count = count(*) from employees
```

**תרגיל 5:**



- יש להפעיל את הפרוצדורה SalesByCategory המקבלת את שם הקטגוריה מתוך הטבלה Categories ושנת ההזמנה שיכול להיות בין 1996 – 1998
- יש לבנות ComboBox לכל אחד מהשדות הנ"ל ובלחיצה על Invoke יש להציג את התוצאה ב- DataGridView שנמצא למטה

Form1

Category name: Confections

Order year: 1997

Invoke

	ProductName	TotalPurchase
▶	Chocolate	1282.00
	Gumbär Gummib...	10443.00
	Maxilaku	3129.00
	NuNuCa Nuß-No...	1693.00
	Pavlova	8663.00
	Schoggi Schokol...	10974.00
	Scottish Longbre...	4158.00
	Sir Rodney's Mar...	7314.00
	Sir Rodney's Sco...	5273.00
	Tarte au sucre	21638.00
	Teatime Chocolat...	2987.00
	Valrhona culdes...	2173.00

## :Transactions

לעיתים נרצה לשלוט בטראנזקציה מתוך התוכנית שלנו בכדי לבצע מספר פעולות שיופעלו כמקשה אחת.

בכדי להפעיל טראנזקציה:

- יש להפעיל את הפונקציה BeginTransaction() של האובייקט connection ולשים את הערך החוזר בתוך ה - command הרצוי
- לאחר מכן להריץ את כל הפקודות דרך אותו command
- לבסוף יש לבצע commit או rollback לאובייקט Transaction

```
SqlConnection cn = new SqlConnection(
    "Data Source=.;Initial Catalog=Northwind;Integrated
    Security=True");
SqlCommand com = cn.CreateCommand();

SqlTransaction trans = null;
try
{
    cn.Open();
    trans = cn.BeginTransaction();
    com.Transaction = trans;

    com.CommandText = "INSERT INTO Employees (FirstName, LastName)
    "+
        "VALUES ('Eran', 'Strong')";
    com.ExecuteNonQuery();
    com.CommandText = "INSERT INTO
    Region (RegionID, RegionDescription) "+
        "VALUES (1, 'aaaa')";
    com.ExecuteNonQuery();

    trans.Commit();
}
catch (Exception ex)
{
    trans.Rollback();
    Console.WriteLine(ex.Message);
}
finally
{
    cn.Close();
}
```

## תרגיל 6:



הפעל 2 פקודות בטרנזקציה אחת כך שפקודה אחת תהיה לא חוקית (לדוגמא: מפתח ראשי כפול) ובדוק שאכן 2 הפקודות לא התבצעו.

## Typed DataSet and TableAdapter

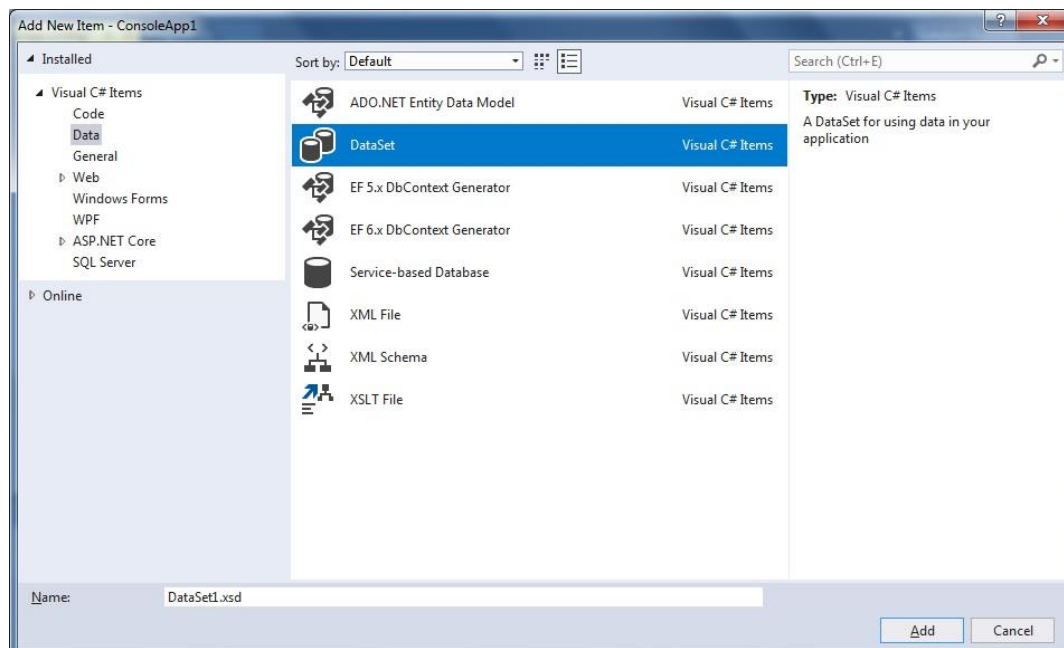
**TypedDataSet** – מחלקה היורשת מ DataSet מכילה את כל הטבלאות והשדות כמאפיינים עם הטיפוסים המתאימים.

**TableAdapter** – מחלקה המכילה DataAdapter שתפקידה לסנכרן בין טבלה ב TypedDataSet לבין ה DB.

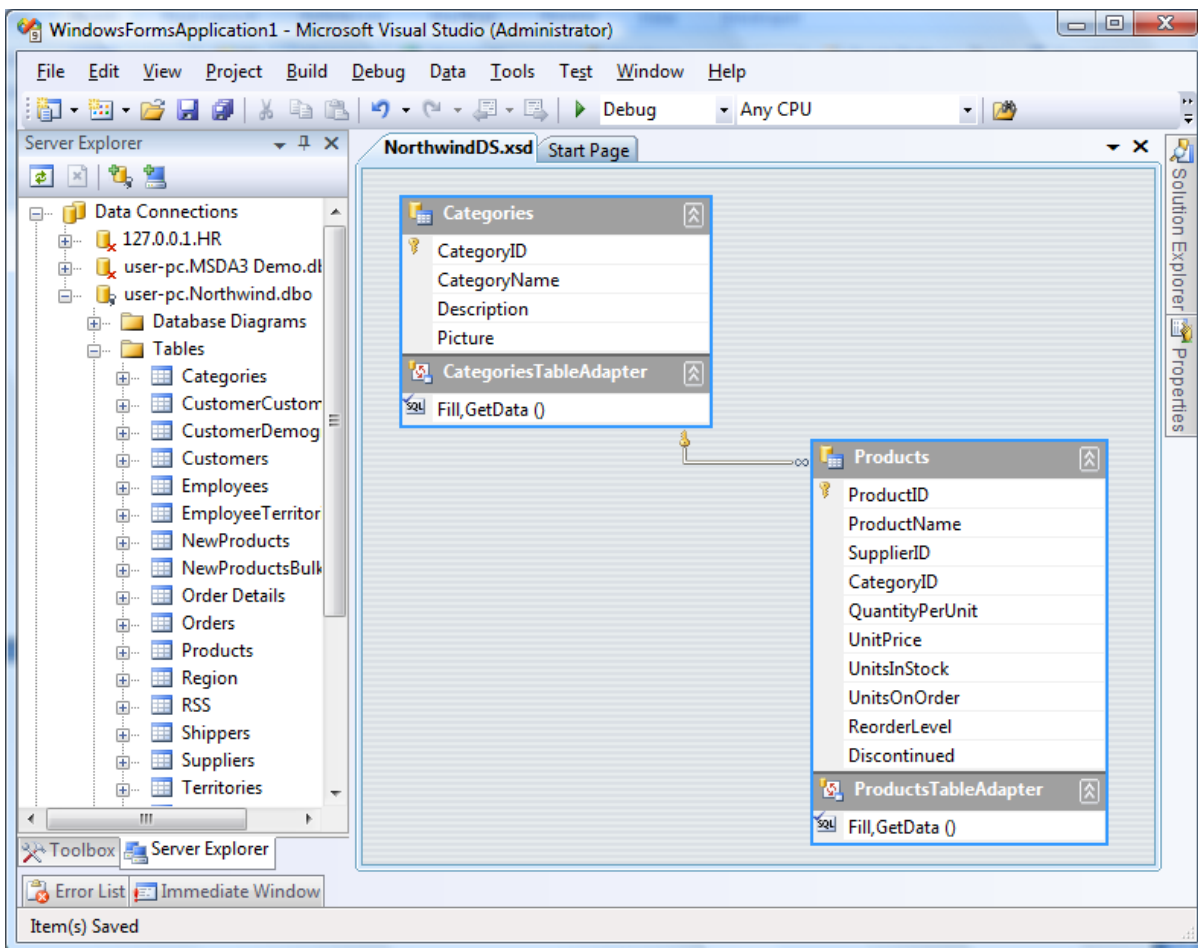
היתרון בעבודה עם 2 האובייקטים הנ"ל הוא הנושא של Type Safe כך שנוכל לפנות לטבלאות ולעמודות באמצעות Properties, ולא באמצעות string המציג רק את השם וכן הטיפוסים של ה - properties כבר מוגדרים כך שאם נטעה בטיפוס הנתונים נקבל שגיאת קומפילציה ולא שגיאת זמן ריצה.

יצירת Typed dataset ו TableAdapter:

יש לגשת לתפריט Add new item ולהוסיף DataSet:



בחלון שיפתח יש להוסיף את הטבלאות הרצויות מתוך ה - Server Explorer:



שימו לב שלכל טבלה יש למטה TableAdapter וכך ניתן לעבוד איתם:

שליפת נתונים (שימו לב לשמוש במאפיין Products שנוצר):

```
NorthwindDS ds = new NorthwindDS();
ProductsTableAdapter ad = new ProductsTableAdapter();
ad.Fill(ds.Products);
```

הוספת רשומה (שימו לב לעבודה עם המאפיינים):

```
NorthwindDS.ProductsRow row = ds.Products.NewProductsRow();
row.CategoryID = 1;
row.ProductName = "aaaa";
ds.Products.Rows.Add(row);
```

## תרגיל 7:



- בנה TypedDataSet עם הטבלאות Categories, Products
- הצג את הטבלה Categories על DataGridView בלחיצה על Fill
- בלחיצה על Add יש להוסיף חדשה לטבלה ול - DB דרך הקוד

Form1

	CategoryID	CategoryName	Description	Picture
▶	1	Beverages	Soft drinks, coffe...	
	2	Condiments	Sweet and savor...	
	3	Confections	Desserts, candie...	
	4	Dairy Products	Cheeses	
	5	Grains/Cereals	Breads, crackers,...	
	6	Meat/Poultry	Prepared meats	
	7	Produce	Dried fruit and be...	
	8	Seafood1	Seaweed and fish	
	9	eran	eran	
	10	udi	12345	
	11	asdfasdf	asdfasdf	
	12	eran1	eran12345	
	14	bbbb	bbbb	
*				

Fill Add



## מודל השכבות לעבודה מול DB

במערכות גדולות מומלץ לחלק את המערכת לשכבות, כך שלכל שכבה יהיה תפקיד שונה. היתרון בחלוקה לשכבות הוא תחזוקה קלה כך שניתן יחסית בקלות להחליף כל שכבה בלי לגעת בשכבות האחרות כאשר נרצה לעשות שינויים בתוכנית.

לדוגמא: אם נרצה להחליף DB נצטרך רק להחליף את השכבה העובדת מול ה - DB ואם נרצה לבנות לאותה תוכנית ממשיק חלונאי וגם ממשיק אינטרנטי לא נצטרך לכתוב את הלוגיקה פעמיים.

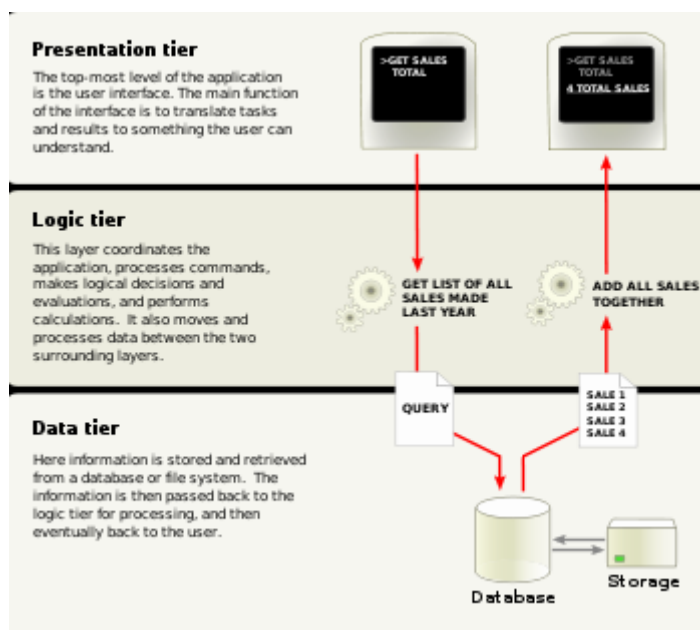
המודל הנפוץ ביותר הוא מודל 3 השכבות הנקרא גם Tier Architecture – 3.

שכבות במודל:

- DAL – Data Access Layer  
שכבה המטפלת בעבודה מול ה - DB
- BLL – Business Logic Layer  
שכבה המטפלת בלוגיקה של המערכת
- PL – Presentation Layer  
שכבה המטפלת בתצוגה למשתמש

חשוב מאד להקפיד ולכתוב את הקוד המתאים בשכבה המתאימה כדי שלא ניתקל בהמשך בבעיות בתחזוקת המערכת.

לדוגמא: אם נציג הודעת שגיאה בתוך BLL אנו עלולים להיקלע לבעיה כאשר נפעיל שוב את הקוד מטופס אחר ובו לא נרצה להציג הודעת שגיאה. הצגת הודעות חייבת להיות רק ב - PL.



[http://en.wikipedia.org/wiki/Multitier\\_architecture](http://en.wikipedia.org/wiki/Multitier_architecture)



---

## מבוא ל – Entity framework

---

מסדי נתונים ותכניות מחשב שונים מאד אחד מהשני במבנה ובייעוד. התפקיד של entity framework או בקיצור ef הוא לצמצם את הפערים ביניהם.

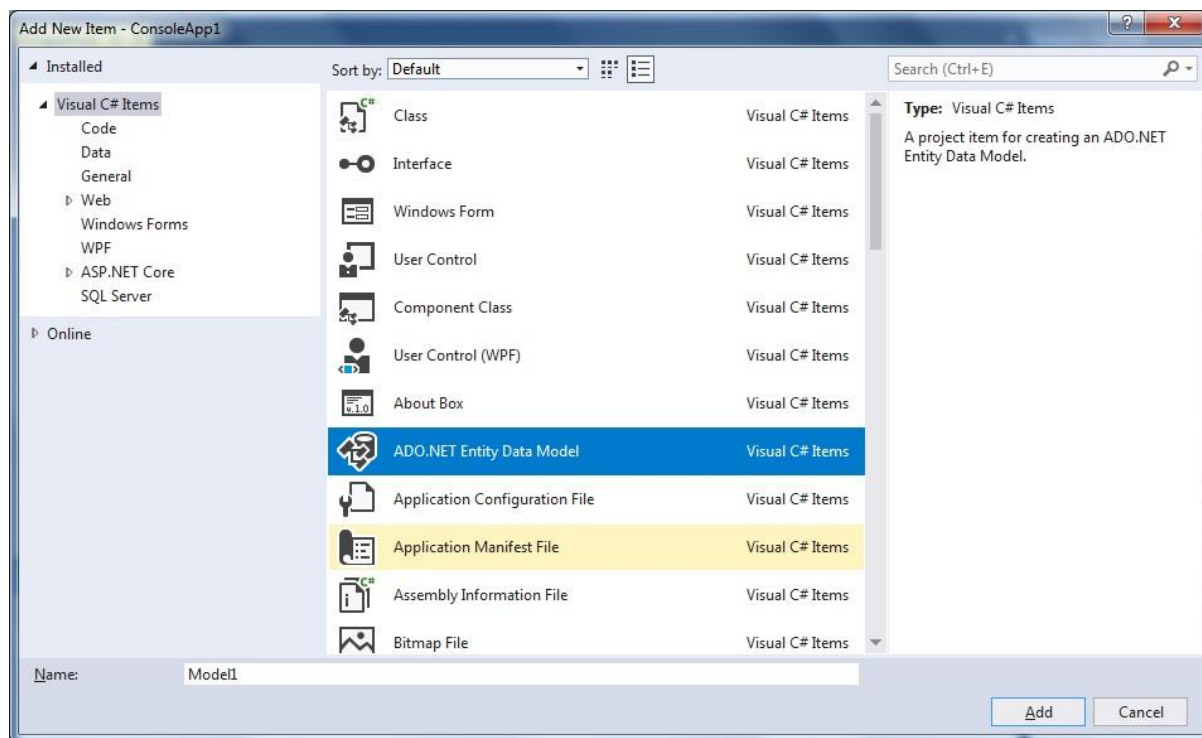
מאחורי הקלעים ef כוללת שלושה מרכיבים:

1. Ssdl (storage schema definition language) – כולל מפרט של המבנה במסד הנתונים איתו רוצים לעבוד בתכנית
2. CSDL (conceptual storage definition language) – כולל מפרט של המחלקות שייצגו טבלאות מהמסד נתונים.
3. MSL (mapping specification language) - מקשר בין המפרט במסד הנתונים לבין הייצוג של המחלקות בתכנית

בין היתרונות של ef אפשר לציין עבודה עם מושגים בתחום הישומים כגון מחלקות ומאפיינים וחופש מהתלות במפרט האחסנה של המידע. תמיכה בעבודה עם LINQ.

### Entity data model (edmx):

לעבודה עם ef יש לבחור בתפריט project <- add new item <- ADO.NET Entity Data Model



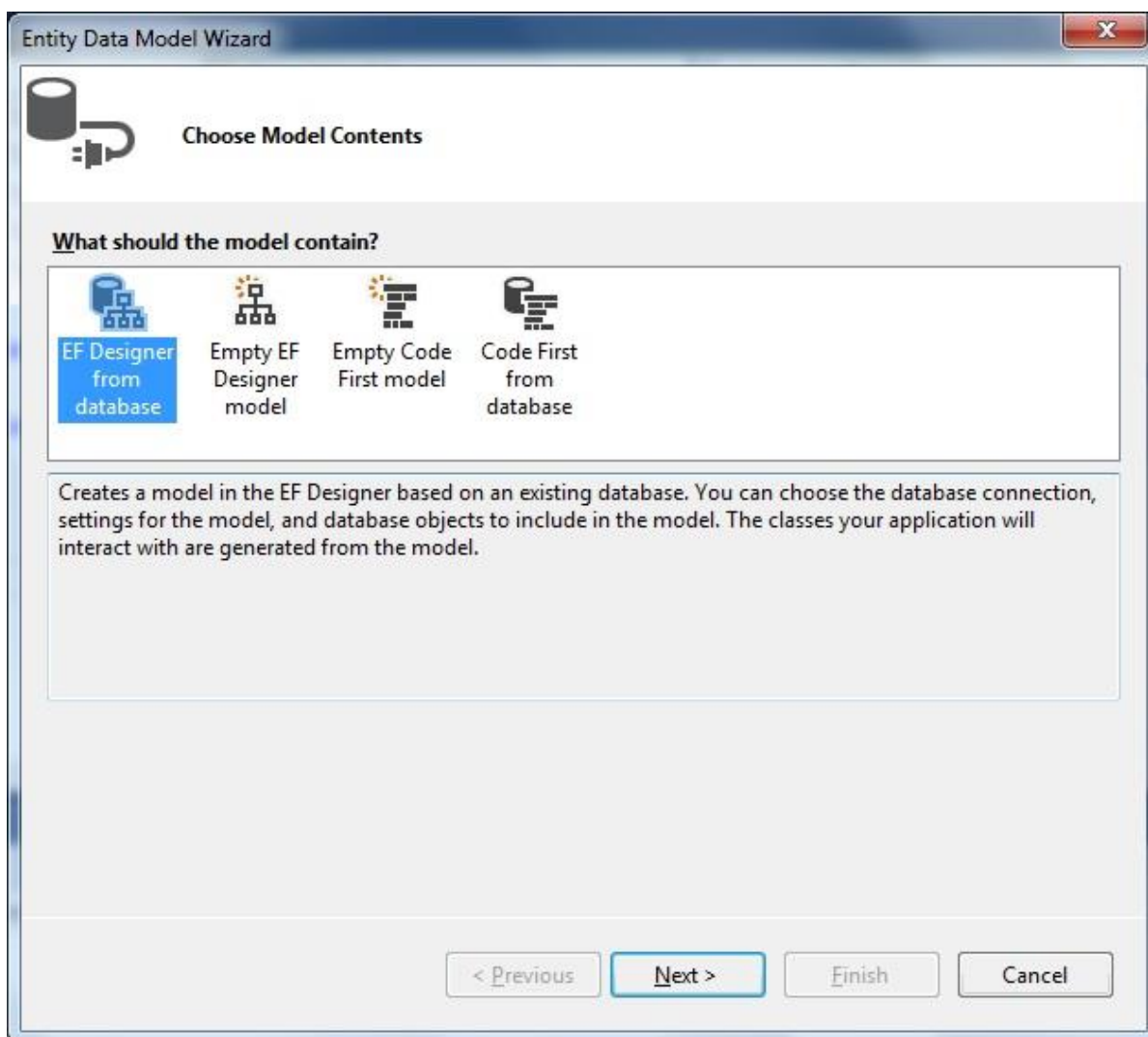
השלב הבא באשף (wizard) של עבודה עם מסדי נתונים דרך ef נפתח חלון נפתח חלון שבו בוחרים את אופן העבודה עם ef. ישנן 3 דרכים לעבודה עם ef:

1. Data base first - גישה בה יש כבר מסד נתונים קיים ואנו מהתכנית מנסים להתחבר למסד הקיים.
2. Code first - גישה בה כותבים קוד בתכנית שייצור את מסד הנתונים בעת ההרצה.
3. Model first - יצירת מסד נתונים על פי הדגם (model) שניצור בלי לכתוב קוד

### נתחיל בגישה הראשונה של data base first

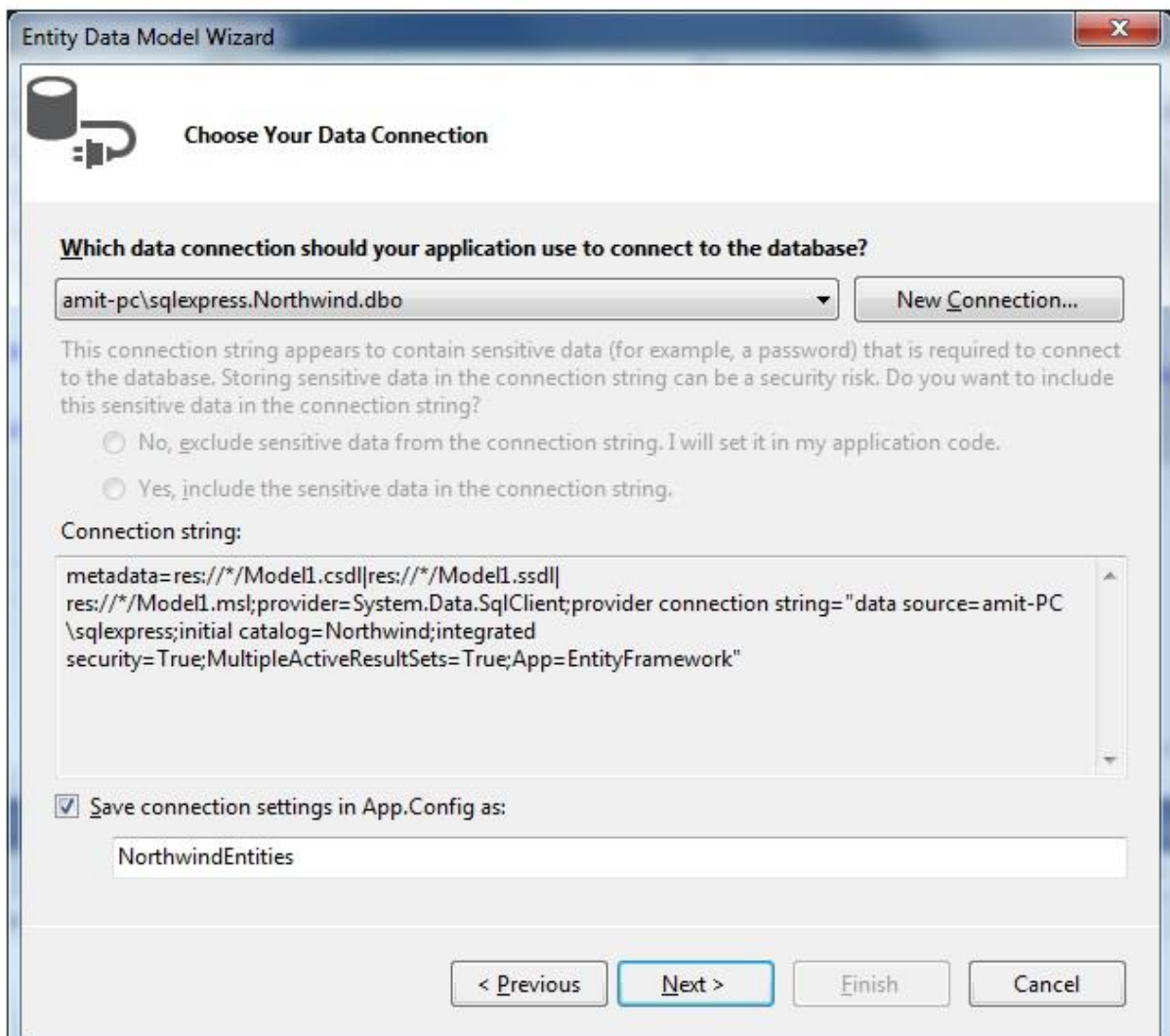
בשלב הבא באשף אחרי בחירת ADO.NET Entity Data Model, נבחר באפשרות של

EF Designer from database



ונלחץ על כפתור Next בתחתית החלון

יפתח החלון הבא:



**Entity Data Model Wizard**

**Choose Your Data Connection**

**Which data connection should your application use to connect to the database?**

amit-pc\squlexpress.Northwind.dbo New Connection...

This connection string appears to contain sensitive data (for example, a password) that is required to connect to the database. Storing sensitive data in the connection string can be a security risk. Do you want to include this sensitive data in the connection string?

☐ No, exclude sensitive data from the connection string. I will set it in my application code.

☐ Yes, include the sensitive data in the connection string.

**Connection string:**

```
metadata=res://*/Model1.csdl|res://*/Model1.ssdl|
res://*/Model1.msl;provider=System.Data.SqlClient;provider connection string="data source=amit-PC
\squlexpress;initial catalog=Northwind;integrated
security=True;MultipleActiveResultSets=True;App=EntityFramework"
```

☒ Save connection settings in App.Config as:

NorthwindEntities

< Previous **Next >** Finish Cancel

בו אפשר ללחוץ על כפתור New Connection בחלק העליון כדי לבחור את מסד הנתונים איתו נרצה לעבוד ויפתח חלון שפגשנו בתחילת החוברת של עבודה מול מסד נתונים בלי ef

Connection Properties

Enter information to connect to the selected data source or click "Change" to choose a different data source and/or provider.

Data source:  
Microsoft SQL Server (SqlClient) Change...

Server name:  
amit-pc\squlexpress Refresh

Log on to the server  
Authentication: Windows Authentication

User name:   
Password:   
☐ Save my password

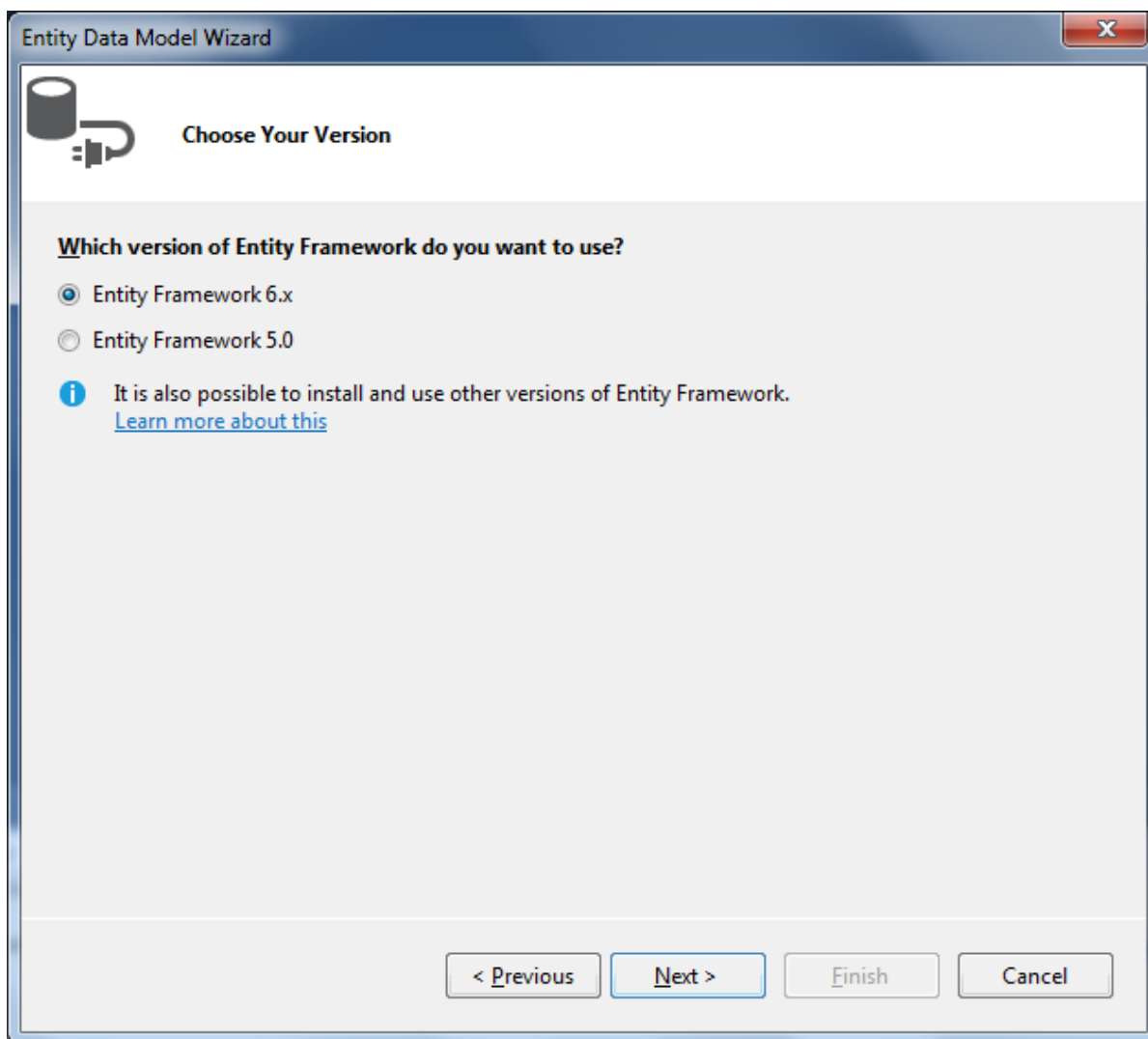
Connect to a database  
☒ Select or enter a database name:  
  
 dbStudents  
 master  
 model  
 msdb  
 Northwind  
 tempdb

Advanced...

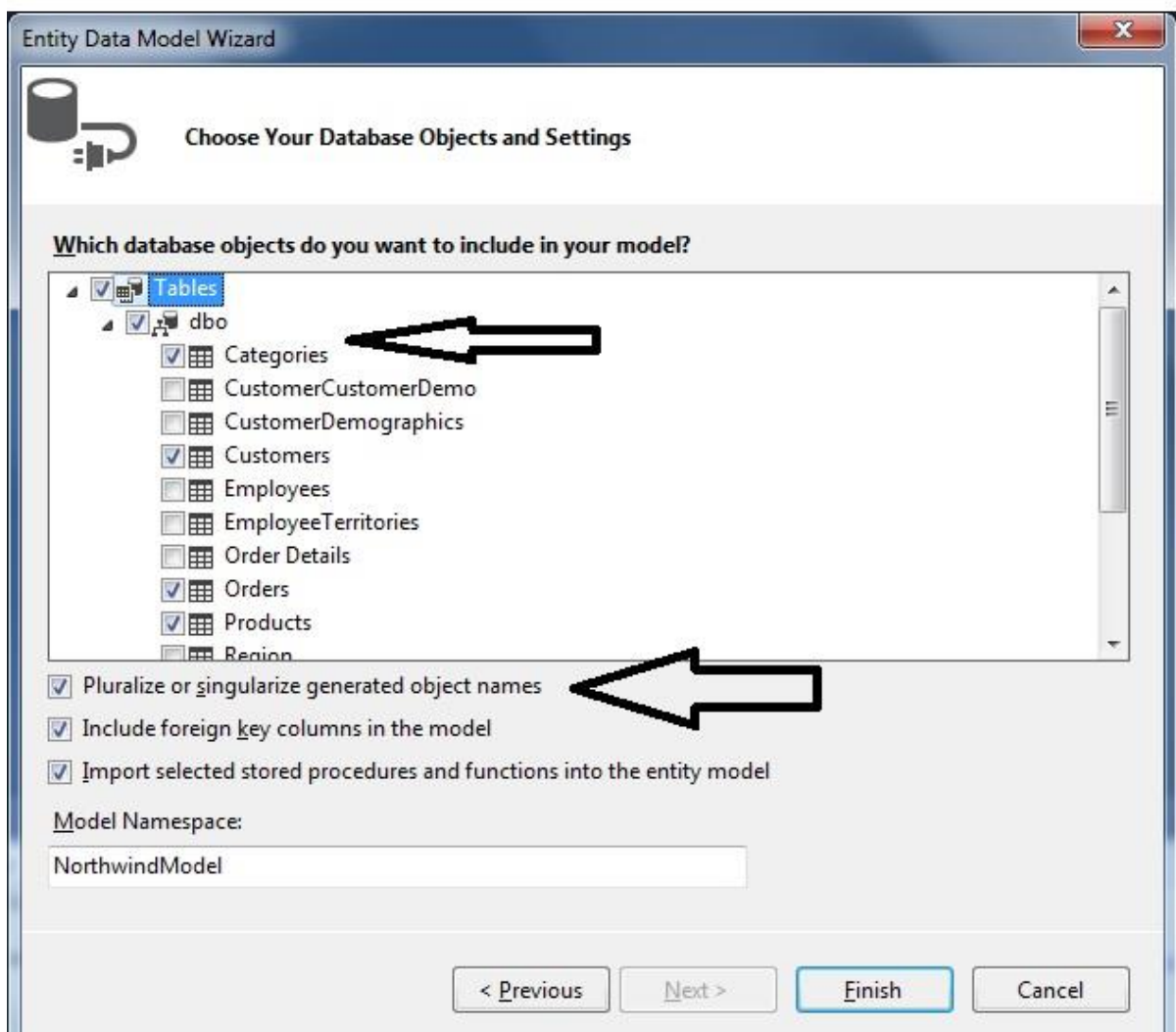
Test Connection OK Cancel

אחרי לחיצה על כפתור OK בתחתית החלון, נחזור לחלון הקודם בו אפשר ללחוץ על כפתור next בתחתית החלון כדי לעבור לשלב הבא.

בחלון הבא:



יש לבחור במיספר גרסה וללחוץ על כפתור next בתחתית החלון



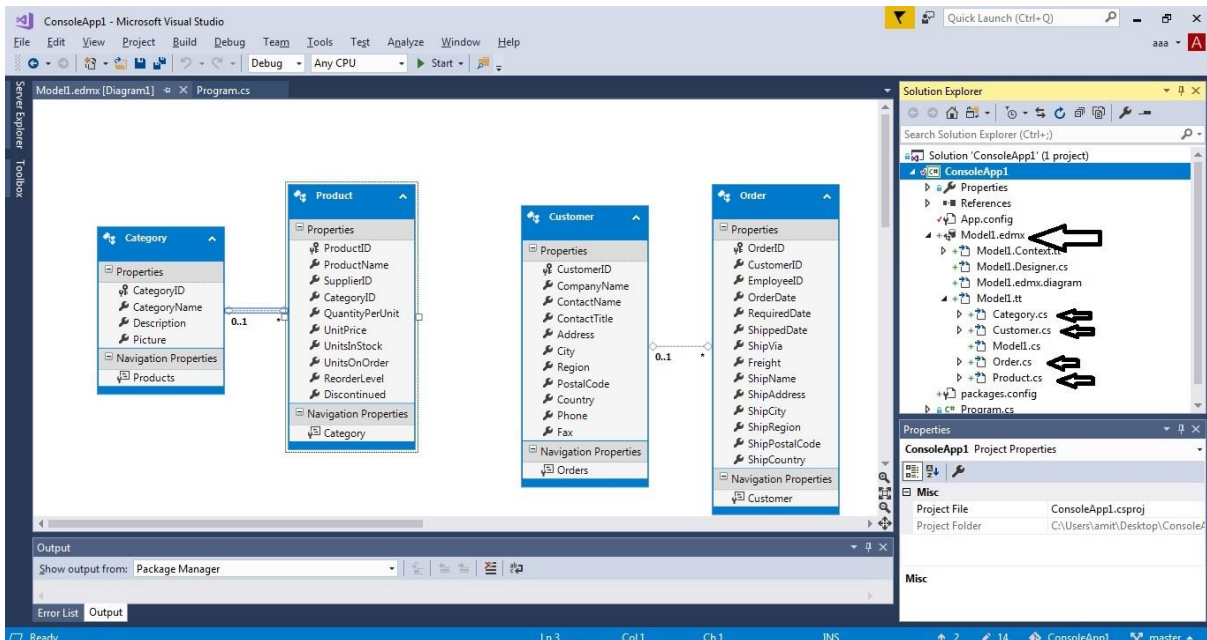
בחלון הבא מומלץ לסמן את תיבת הסימון (checkbox) של Pluralize or singularize generated object names שתאפשר לתת שמות משמעותיים יותר למחלקות שייווצרו על ידי ef.

בנוסף, יש לסמן את הטבלאות שמהן רוצים לשלוף מידע כגון טבלאות Categories, Customers, ו-Products ועוד.

לאחר בחירת הטבלאות יש ללחוץ על כפתור Finish בתחתית החלון.

כשנחזור לסביבת העבודה של visual studio, נוכל לראות בחלון של solution explorer שנוספו מספר קבצים ליישום.





בחלק המרכזי של סביבת העבודה אפשר לראות את התצוגה של קובץ edmx המציג את הטבלאות שמשכנו ממסד הנתונים ובמידה שהיו קשרים בין הטבלאות, נראה את הקשרים בין הטבלאות כפי שהוגדרו במסד הנתונים.

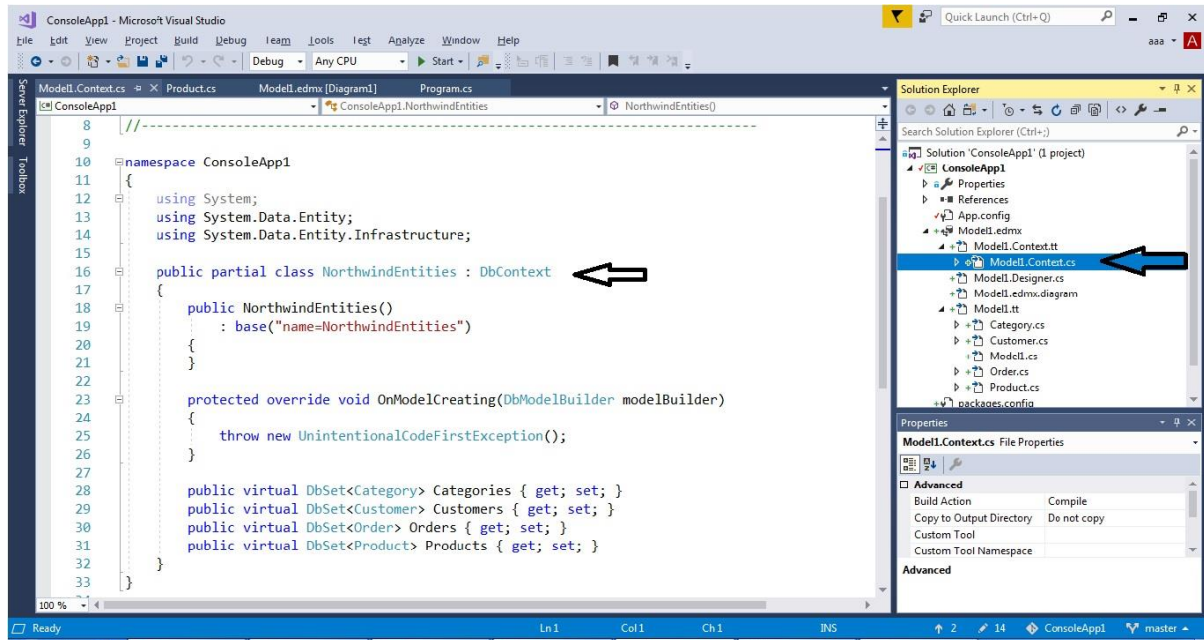
בחלון של solution explorer נוכל לראות שלכל טבלה שמשכנו יש יצוג בתכנית כמחלקה. לדוגמא לטבלת מוצרים (products) נוצרה מחלקה בשם product ולכל שדה בטבלה, נוצר מאפיין באותו שם במחלקה

```

11 {
12     using System;
13     using System.Collections.Generic;
14
15     public partial class Product
16     {
17         public int ProductID { get; set; }
18         public string ProductName { get; set; }
19         public Nullable<int> SupplierID { get; set; }
20         public Nullable<int> CategoryID { get; set; }
21         public string QuantityPerUnit { get; set; }
22         public Nullable<decimal> UnitPrice { get; set; }
23         public Nullable<short> UnitsInStock { get; set; }
24         public Nullable<short> UnitsOnOrder { get; set; }
25         public Nullable<short> ReorderLevel { get; set; }
26         public bool Discontinued { get; set; }
27
28         public virtual Category Category { get; set; }
29     }
30 }
31

```

המחלקה הראשית של ef שנוספה לתכנית הינה מחלקה שירשת ממחלקה בשם DbContext זו המחלקה המרכזית של ef המשמשת לשלוף מידע אל היישום, לעקוב אחר שינויים במידע בהרצת התכנית וניתן לעדכן איתה חזרה אל מסד הנתונים כל עדכון, הוספה או מחיקת מידע שביצענו בעת הרצת היישום. שם המחלקה ניגזר משם מסד הנתונים ממנו שלפנו מידע בתוספת המילה entities. לדוגמא בשליפת מידע ממסד הנתונים של northwind, המחלקה תקרא NorthwindEntities.



### שליפת מידע ממסד הנתונים לתכנית

כדי להתחיל לעבוד מול מסד הנתונים בין אם לשליפת מידע או לעדכונים במסד הנתונים, תחילה ניצור מופע של העצם היוצר ממחלקת DbContext בדרך כלל שמו יהיה כשם מסד הנתונים בתוספת המילה Entities לדוגמא אם שלפנו מידע ממסד הנתונים של Northwind אז שם המחלקה יהיה NorthwindEntities ואם לדוגמא שלפנו מידע ממסד נתונים בשם dbStudents אז שם המחלקה יהיה dbStudentsEntities.

```
NorthwindEntities db = new NorthwindEntities();
```

כיון שהמחלקה יורשת ממשק IDisposable, מומלץ לעטוף את האתחול בבליק של using.

```
using (NorthwindEntities db=new NorthwindEntities())
{
}
}
```

אם נרצה לדוגמא להציג את שמות המוצרים נוכל לכתוב את הלולאה הבאה

```
foreach (Product p in db.Products)
{
    Console.WriteLine(p.ProductName);
}
```

נוכל לעבוד עם התחביר של Linq כדי לשלוף את המידע

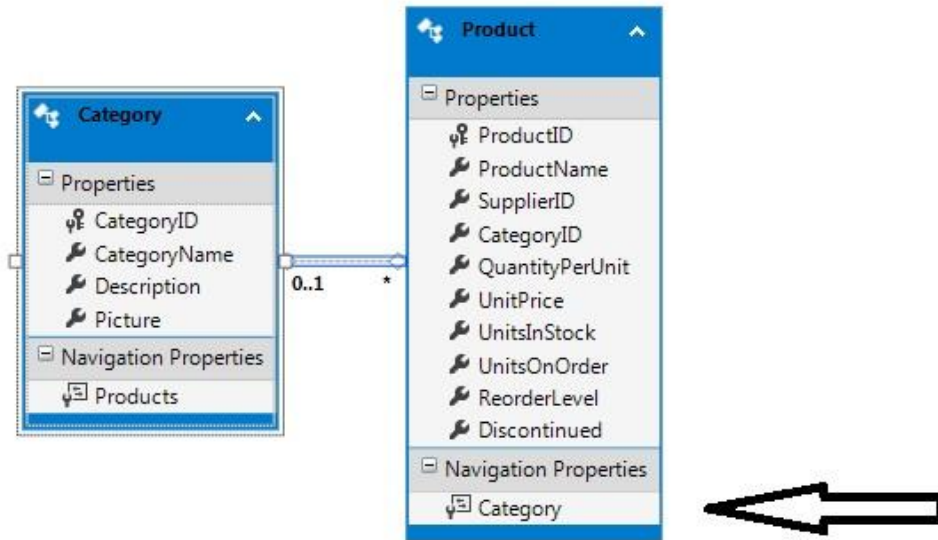
```
var q = from p in db.Products
        select p;
foreach (Product p in q)
{
    Console.WriteLine(p.ProductName);
}
```

### שליפת מידע ממספר טבלאות עם הפקודה Include

כשנרצה לשלוף מידע ממספר טבלאות שמקושרות ביניהן, לדוגמא מידע מטבלת מוצרים יחד עם מידע מטבלת קטגוריות כדי להציג את שם המוצר ואת שם הקטגוריה אליה המוצר משתייך, מומלץ בשליפת המידע מטבלת מוצרים להגדיר גם את שליפת המידע מטבלת קטגוריות באמצעות פקודת **include** אחרת במקום לשלוף את המידע פעם אחת ממסד הנתונים, יתבצעו מספר פניות אל מסד הנתונים גם לשליפת המידע מטבלת מוצרים וגם לשליפת מידע מטבלת קטגוריות.

כאשר יש קשר בין טבלאות ששלפנו ממסד הנתונים, נוכל להבחין בקובץ edmx שבטבלה א המקושרת אל טבלה ב, בחלק של Navigation Properties יש שדה נוסף על השדות המקוריים בטבלה עם שם הטבלה אליה מקושרים.

לדוגמא בטבלה Products המקושרת לטבלה Categories, יש שדה בשם Category בחלק של Navigation Properties.



כדי לשלוף את שם המוצר מטבלת מוצרים יחד עם שם הקטגוריה אליה משתייך המוצר מטבלת קטגוריות, נוכל לרשום את השאילתא הבאה:

```
using (NorthwindEntities db=new NorthwindEntities())
{
    var prods = db.Products.Include("Category").ToList();

    foreach (Product p in prods)
    {
        Console.WriteLine(p.ProductName+" "+p.Category.CategoryName);
    }
}
```

## הוספת רשומה למסד הנתונים

להוספת רשומה אחת או יותר למסד הנתונים יש ליצור מופע מהמחלקה המייצגת את הטבלה. לדוגמא אם נרצה להוסיף לטבלת קטגוריות רשומה חדשה

```
using (NorthwindEntities db = new NorthwindEntities())
{
    Category c = new Category { CategoryID = 9, CategoryName = "sweets",
                                Description = "chocolates and candies" };
    db.Categories.Add(c);
    db.SaveChanges();
}
```

בשורות אלו יצרנו מופע של עצם ממחלקת קטגוריה ואתחלנו חלק מהשדות בערכים. בשלב הבא הוספנו את העצם אל אוסף העצמים המייצגים את הרשומות שנישלפו ממסד הנתונים. בשורה האחרונה ביצענו עדכון חזרה למסד הנתונים.

## עדכון רשומה במסד הנתונים

לעדכון רשומה נגדיר משתנה מסוג המחלקה שיצביע על עצם שנמשוך מהרשימה של כל הקטגוריות. נעדכן את ערכי המאפיינים בעצם ובשורה האחרונה שוב נבצע עדכון חזרה למסד הנתונים

```
using (NorthwindEntities db = new NorthwindEntities())
{
    Category c = db.Categories.First(x => x.CategoryName == "sweets");
    c.Description += " from all over the world";
    db.SaveChanges();
}
```

## מחיקת רשומה במסד הנתונים

למחיקת רשומה שוב נגדיר משתנה מסוג קטגוריה המצביע על אחד המופעים. לאחר מכן נמחק את העצם מאוסף הקטגוריות ובשלב האחרון נעדכן את השינויים במסד הנתונים.

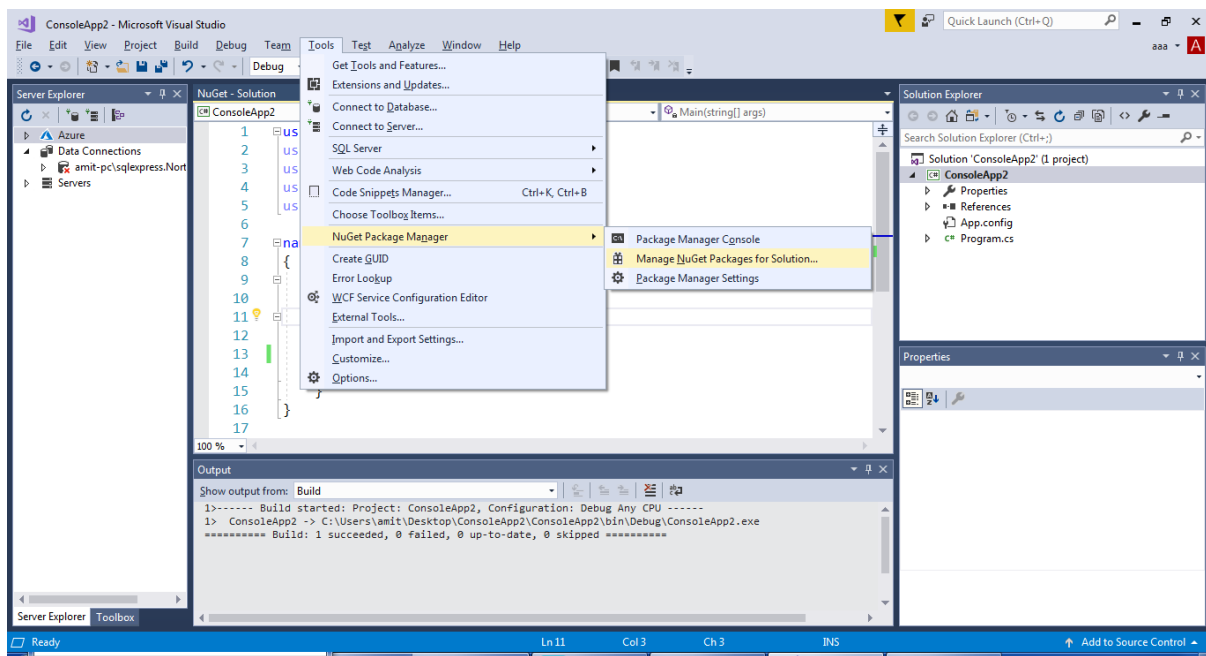
```
using (NorthwindEntities db = new NorthwindEntities())
{
    Category c = db.Categories.First(x => x.CategoryName == "sweets");
    db.Categories.Remove(c);
    db.SaveChanges();
}
```

## נעבור לגישה השניה של code first

לפעמים במערכות קטנות יהיה יעיל יותר לעבוד בגישה של code first שפירושה שהתכנית תייצר את מסד הנתונים בהתאם להוראות שנגדיר בתכנית בלי שיווצר לנו מעצב (designer) בקובץ edmx שמכיל פקודות ו־xml מורכבות עם הרבה יותר מהנדרש עבורנו. כך המפתח יכול להתרכז בתכנית שתייצר מסד נתונים בהתאם במקום להתאים את התכנית למסד נתונים קיים. עדכונים למסד נתונים יאבדו בגלל שהתכנית מגדירה את מבנה מסד הנתונים.

כדי להוסיף את כל הרכיבים הנדרשים לעבודה, נבחר בתפריט

Tools-> Nuget Package Manager-> Manager Nuget Packages for Solution



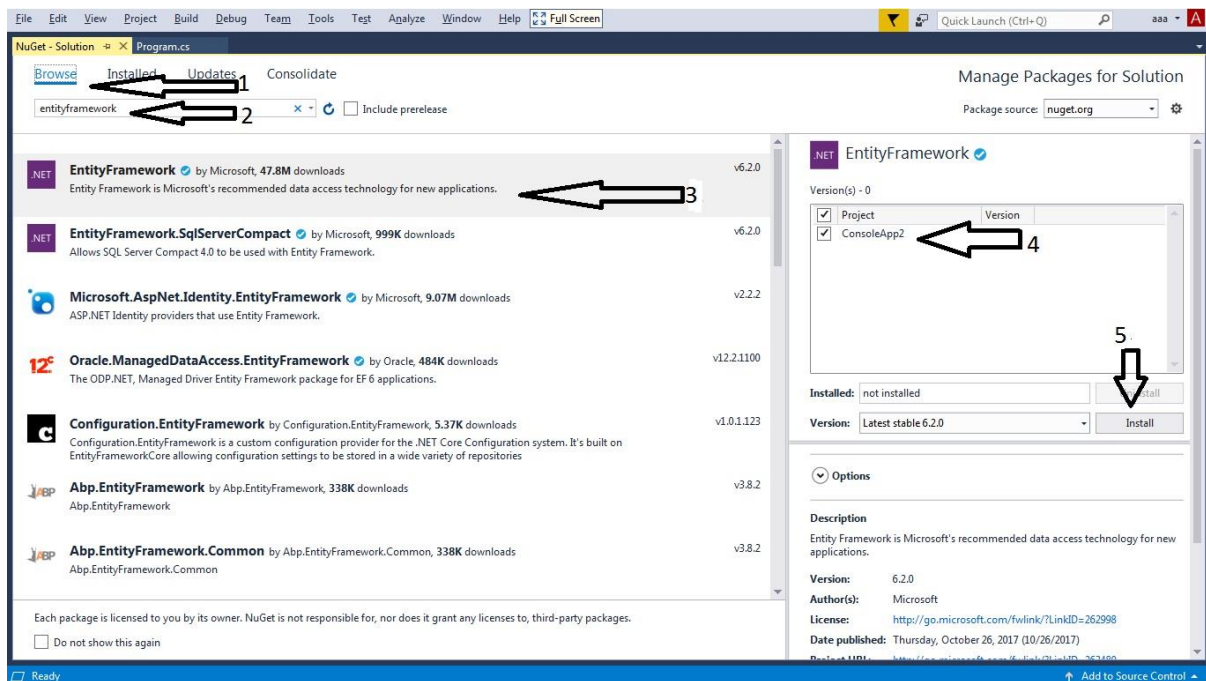
1 בחלון שיפתח נסמן בצד שמאל למעלה את האפשרות של Browse

2 בשדה החיפוש נרשום entityframework

3 נסמן את הגרסה החדשה ביותר של Microsoft

4 נסמן את הפרוייקט אליו נרצה לצרף את הרכיבים

5 נלחץ על כפתור install



בשלב הבא יפתחו חלונות לאישור הפעולה בהם נלחץ על OK ועל Accept |

### יצירת מסד הנתונים

השלב הראשון אחרי הקישור לרכיבים של entityframework יהיה להגדיר מחלקות שבעקבותיהן יוצרו טבלאות במסד הנתונים. לשם ההדגמה ניצור שתי טבלאות – טבלת מוצרים וטבלת קטגוריות של המוצרים.

```
public class Product
{
    public int ProductId { get; set; }
    public string ProductName { get; set; }
    public double Price { get; set; }

    public Category category { get; set; }
}
public class Category
{
    public int CategoryId { get; set; }
    public string CategoryName { get; set; }
    public string Description { get; set; }

    public ICollection<Product> products { get; set; }
}
```

בכל טבלה הגדרנו מאפיינים שייוצגו על ידי שדות בטבלאות ובנוסף כדי ליצור קשר בין הטבלאות, הוספנו בטבלת מוצרים מאפיין מסוג מחלקת קטגוריה ובמחלקת קטגוריה הוספנו מאפיין מסוג של אוסף של מוצרים. כך מגדירים יחס של אחד לרבים בין טבלת מוצרים לטבלת קטגוריות.

בשלב הבא נגדיר את המחלקה הראשית של entityframework שאחראית על העבודה מול מסד הנתונים ויורשת מהמחלקה DbContext.

```
public class ShopContext : DbContext
{
    public ShopContext() : base()
    {
    }

    public DbSet<Product> Products { get; set; }
    public DbSet<Category> Categories { get; set; }
}
```

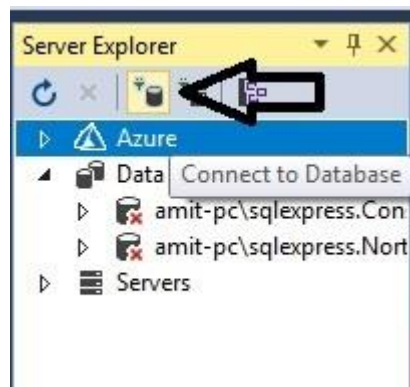
במחלקה נגדיר שני מאפיינים מסוג של אוסף של פריטים מסוג המחלקות שעבורן נרצה ליצור טבלאות. עכשיו אנחנו מוכנים להוספת נתונים למסד הנתונים:

```
static void Main(string[] args)
{
    using (var ctx = new ShopContext())
    {
        var prod = new Product() { ProductName = "bread" };
        ctx.Products.Add(prod);
        ctx.SaveChanges();
    }
}
```

בהרצת התכנית ייוצר מסד הנתונים עם הטבלאות במידה שאיננו קיים מסד הנתונים, ותתווסף הרשומה החדשה לטבלת מוצרים.

### מיקום מסד הנתונים:

מסד הנתונים נוצר תחת המסלול של sqlexpress אם נרצה לקשר אל מסד הנתונים דרך חלון server explorer בסביבה של visual studio, ניתן בחלון של server explorer ללחוץ על כפתור Connect to db



בחלון שיפתח יש לרשום בשם השרת \sqlexpress. ושם מסד הנתונים יהיה כשם התוכנית ושם המחלקה היורשת DbContext לדוגמא ConsoleApp2.ShopContext



**Add Connection**

Enter information to connect to the selected data source or click "Change" to choose a different data source and/or provider.

**Data source:**  
Microsoft SQL Server (SqlClient) Change...

**Server name:**  
.sqlexpress Refresh

**Log on to the server**  
Authentication: Windows Authentication  
User name:   
Password:   
☐ Save my password

**Connect to a database**  
☒ Select or enter a database name:  
ConsoleApp2.ShopContext  
dbStudents  
master  
model  
msdb  
Northwind  
tempdb

Advanced...

Test Connection OK Cancel

מסד הנתונים ימצא במחשב בתיקה של sqlexpress

C:\Program Files\Microsoft SQL Server\MSSQLSQLEXPRESS\MSSQL\DATA

במידה וניתקלים בקשיים ביצירת מסד הנתונים, ניתן גם להגדיר את יצירת מסד הנתונים על שרת sql server באמצעות הוספה של מחרוזת חיבור בקובץ App.config

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <connectionStrings>
    <add name="abc"
      connectionString="Data Source=.;Initial Catalog=ShopDb;Integrated Security=true"
      providerName="System.Data.SqlClient"/>
  </connectionStrings>
</configuration>
```

בנוסף בהגדרת המחלקה היורשת את DbContext יש לקשר אל מחרזת החיבור בדרך הבאה

```
public class ShopContext : DbContext
{
    public ShopContext() : base("name=abc")
    {
    }
}
```

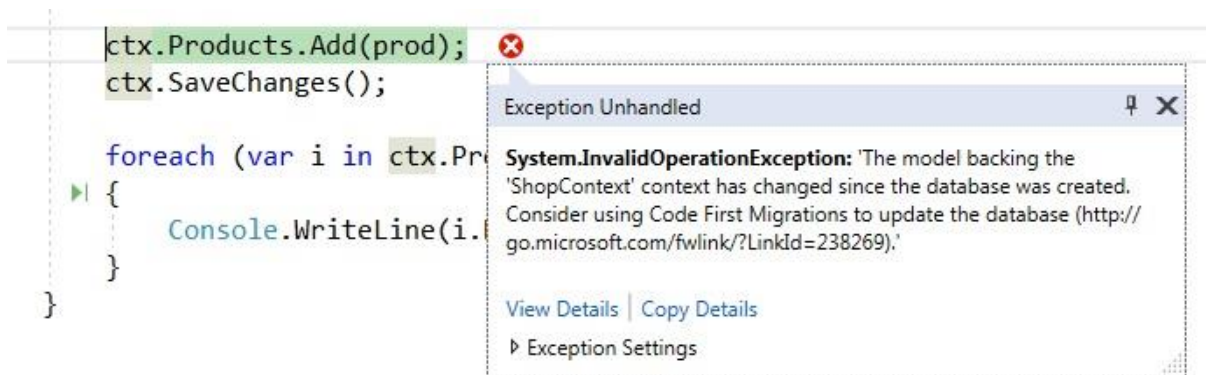
### עדכון במבנה מסד הנתונים:

במידה ונרצה לעדכן את מסד הנתונים כגון הוספת מאפיין נוסף לאחת המחלקות לדוגמא נוסיף למחלקת מוצר גם את המאפיין של כמות במלאי

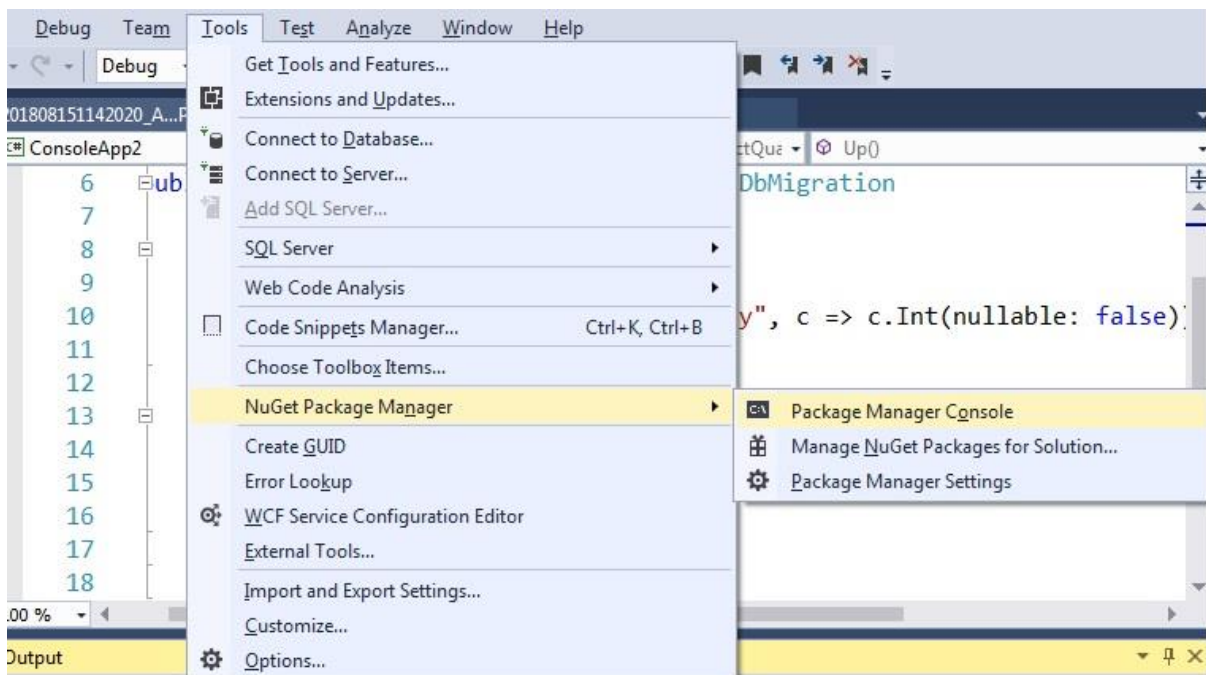
```
public class Product
{
    public int ProductId { get; set; }
    public string ProductName { get; set; }
    public double Price { get; set; }
    public int Quantity { get; set; }

    public Category category { get; set; }
}
```

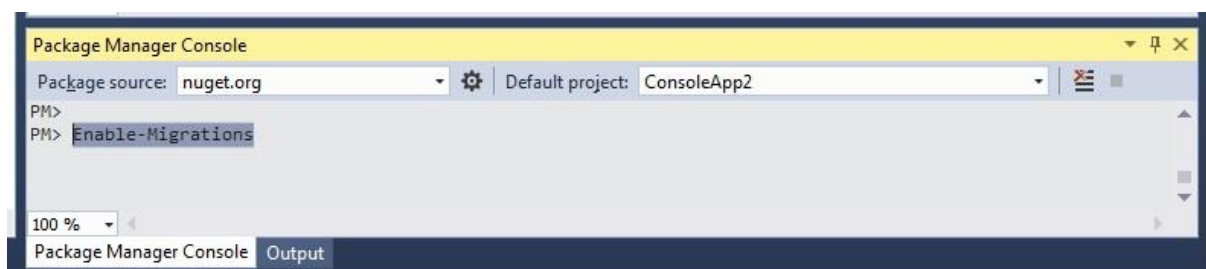
אם ננסה להריץ שוב את התכנית אחרי עדכון המחלקה נקבל את הודעת שגיאה הבאה:



כדי לאפשר עדכון למסד הנתונים נחזור לחלון Package Manager Console

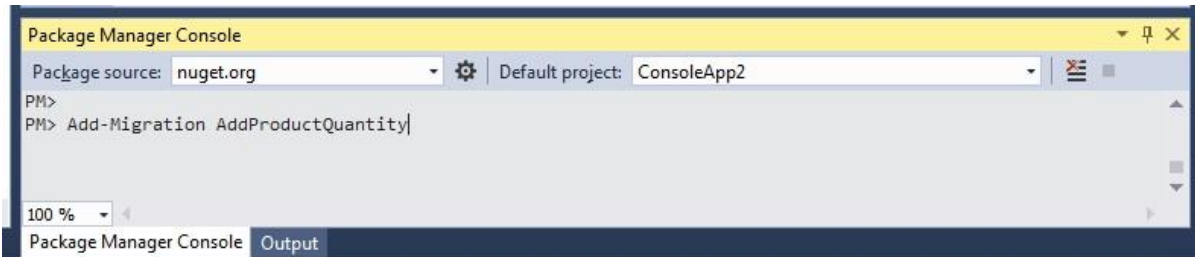


בחלון שיפתח נרשום Enable-Migrations



הפקודה תיצור לנו תיקיה עם שני קבצים שניתן לראותם בחלון solution explorer.

כדי להוסיף מאפיין בשם Quantity למחלקה Product נרשום את הפקודה הבאה בחלון

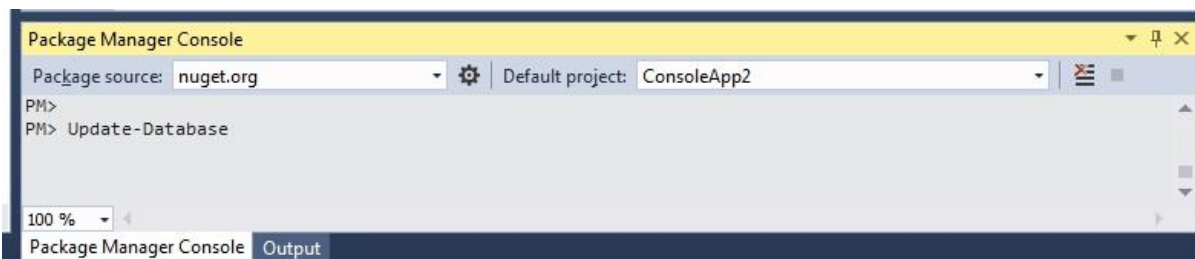


הפקודה תיצור לנו קובץ חדש בתיקיה עם המחלקה הבאה:

```
public partial class AddProductQuantity : DbMigration
{
    public override void Up()
    {
        AddColumn("dbo.Products", "Quantity", c => c.Int(nullable: false));
    }

    public override void Down()
    {
        DropColumn("dbo.Products", "Quantity");
    }
}
```

השלב הבא יהיה לעדכן את מסד הנתונים עם הפקודה הבאה:



אם נבדוק דרך חלון server explorer את מסד הנתונים נגלה שבטבלה Products נוסף שדה של Quantity.