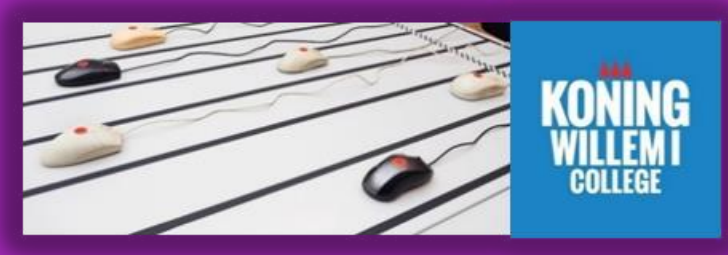
Gevorderd standalone

Realiseren

hoofdstuk

3

Toevoegen, bewerken en verwijderen van relationele gegevens



Algemene informatie

Onderwerp	Toevoegen, bewerken en verwijderen van relationele gegevens
Leerdoel(en)	1. De student kent het SQL SELECT commando en kan het toepassen om gegevens uit meerdere tabellen te combineren. 2. De student kan uitleggen wat een VIEW in SQL is 3. De student kan een view maken en aanpassen. 4. De student programmeert een Model 5. De student programmeert een Controller 6. De student programmeert een View De student koppelt deze 3 componenten aan elkaar om een werkende applicatie te krijgen.
Vereiste voorkennis	1. Student kent de stof uit reader 7 van thema 7.
Kwalificatiedossier	☐ B1-K1-W1: Plant werkzaamheden en bewaakt de voortgang ☐ B1-K1-W2: Ontwerpt software ☒ B1-K1-W3: Realiseert (onderdelen van) software ☐ B1-K1-W4: Test software ☐ B1-K1-W5: Doet verbetervoorstellen voor de software ☐ B1-K2-W1: Voert overleg ☐ B1-K2-W2: Presenteert het opgeleverde werk ☐ B1-K2-W3: Reflecteert op het werk

Inhoudsopgave

Introductie

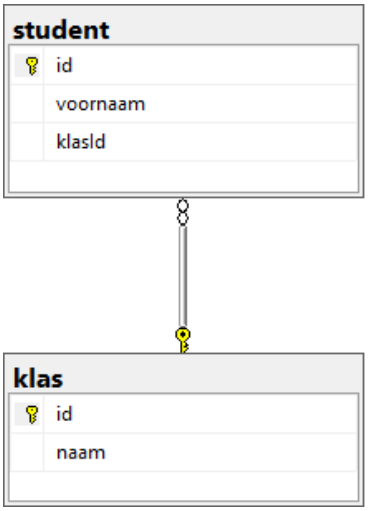
Binnen een applicatie maak je gebruik van gegevens die je wilt weergeven, toevoegen, aanpassen en/of verwijderen. In de praktijk wordt deze data opgeslagen in een database. De data die in de database zit in meerdere tabellen aangezien we werken met een relationele database. Dat betekent dat we gegevens uit meerdere tabellen moeten combineren om alle informatie van een entiteit te kunnen ophalen. Hoe je dat vanuit onze applicatie moet bouwen behandelen we in deze reader.

Inhoud

Verwijderen van relationele data

In het vorige hoofdstuk zijn we bezig geweest om gegevens uit meerdere tabellen op te halen met behulp van een of meerdere JOIN commando's. Nu gaan we kijken naar de andere bewerkingen in ons CRUD model. Daarbij beginnen we met de D van Delete. Waarom? Simpel, omdat deze het makkelijkste is.

Als we ons voorbeeld van de JOIN uit het vorige hoofdstuk erbij pakken hebben we een database met daarin klassen en leerlingen.



Onze RDBMS is verantwoordelijk voor de inhoud van de database. Deze moet consistent zijn en blijven. Zoals je kunt zien in de afbeelding hier langs is er een foreign key tussen student en klas. Ofwel het klasId in student is een deelverzameling van alle id's in de klas tabel.

Dat betekent dus dat je geen student kan aanmaken van een klas die niet bestaat. Maar ook, dat je een klas die gekoppeld is aan een of meerdere studenten niet mag weghalen. Dan zou immer bij die studenten een verwijzing naar een niet (meer) bestaande klas staan. En heb je dus een database die niet meer klopt.

Stel dat de inhoud van de tabellen student en klas als volgt is:

id	voornaam	klasId
1	Joris	1
2	Thomas	1
3	Niels	2
4	Hendrik	2
5	Maartje	3
6	Koen	3
7	Pieter	NULL

id	naam
1	IO2A4
2	IO2B4
3	IO2C4
4	IO2D4

Dan zie je dat er geen student gekoppeld is aan klas met id 4 (IO2D4). Deze klas kunnen we dus zonder problemen verwijderen.

Echter er zijn telkens twee studenten gekoppeld aan de klassen met id 1,2 en 3. Zouden we een van deze klassen weggooien klopt de verwijzing vanuit de student tabel niet meer en hebben we een inconsistente database. Ons RDBMS zorgt er dus voor dat je deze klassen niet kan weggooien en geeft je een foutmelding zoals deze:

```
Msg 547, Level 16, State 0, Line 1
The DELETE statement conflicted with the REFERENCE constraint "FK_STUDENT_KLAS". The conflict
occurred in database "school", table "dbo.student", column 'klasId'.
```

Zonder dat we maatregelen nemen, zal uiteindelijk de gebruiker van onze applicatie deze foutmelding te zien krijgen. Dat willen we natuurlijk voorkomen!

Vandaar dat we (zoals we al geleerd hebben) zorgen voor foutafhandeling. In dit hoofdstuk gaan we echter iets verder. In plaats van een algemene fout/exception afvangen, zoals we in het vorige hoofdstuk gedaan hebben, gaan we nu een specifieke database fout afvangen.

```
try
{
    employeeController.Delete(employee);
}
catch (SqlException ex)
{
    if (ex.Number==547)
        MessageBox.Show("Kan klant niet verwijderen omdat deze orders in het systeem heeft.");
}
```

Je ziet dat de catch nu alleen een exception van het type SqlException afvangt. Vervolgens controleert hij het foutnummer. Een fout met nummer 547 geeft aan dat er iets met een CONSTRAINT fout gaat. Bij verwijderen kan dat feitelijk niet anders zijn dan een fout van een FOREIGN KEY. Immers, dat is een specifieke constraint. Dus kunnen we nu een heel gerichte foutmelding laten zien!

Het is ook mogelijk om meerdere fouten af te vangen zoals je hieronder ziet. Als de fout geen SqlException oplevert wil je de gebruiker wel een door jouw gecontroleerde foutmelding laten zien namelijk…

```
try
{
    employeeController.Delete(employee);
}
catch (SqlException ex)
{
    if (ex.Number==547)
        MessageBox.Show("Kan klant niet verwijderen omdat deze orders in het systeem heeft.");
}
catch (Exception ex)
{
    MessageBox.Show("Er is een onbekende fout opgetreden met melding: "+ ex.Message);
}
```

 Je kunt nu **oefening 3.1** maken.

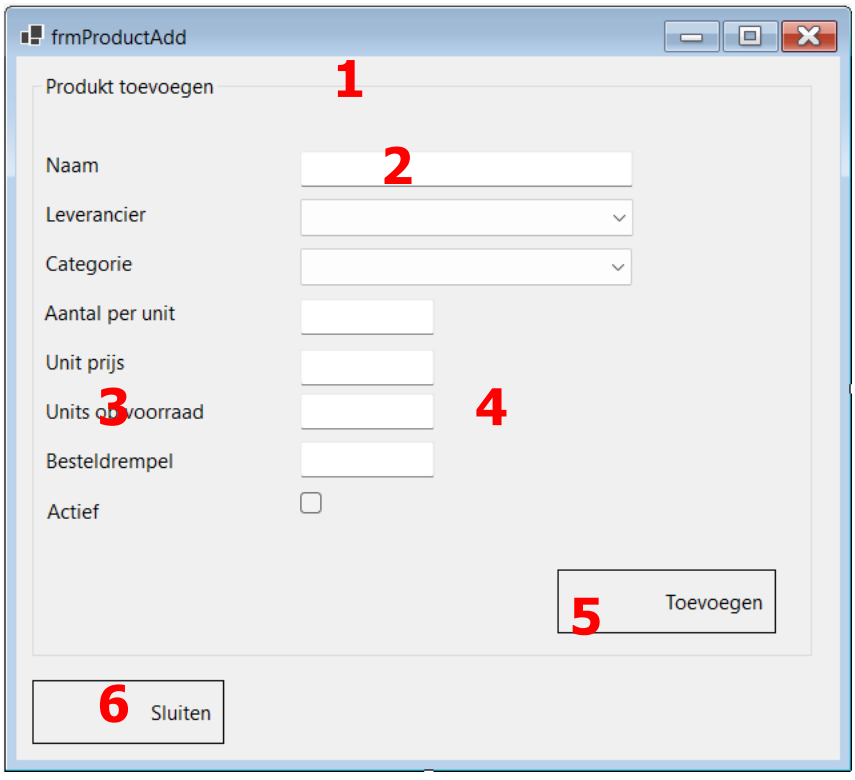
Toevoegen van relationele data

In het vorige hoofdstuk hebben we een ProductModel gemaakt met daarin verwijzingen naar twee andere modellen:

```
public class ProductModel
{
    0 references | 0 changes | 0 authors, 0 changes
    public int ProductID { get; set; }
    0 references | 0 changes | 0 authors, 0 changes
    public string ProductName { get; set; }
    0 references | 0 changes | 0 authors, 0 changes
    public SupplierModel Supplier { get; set; }
    0 references | 0 changes | 0 authors, 0 changes
    public CategoryModel Category { get; set; }
    0 references | 0 changes | 0 authors, 0 changes
    public int QuantityPerUnit { get; set; }
    0 references | 0 changes | 0 authors, 0 changes
    public decimal UnitPrice { get; set; }
    0 references | 0 changes | 0 authors, 0 changes
    public int UnitsInStock { get; set; }
    0 references | 0 changes | 0 authors, 0 changes
    public int UnitsOnOrder { get; set; }
    0 references | 0 changes | 0 authors, 0 changes
    public int RecorderLevel { get; set; }
    0 references | 0 changes | 0 authors, 0 changes
    public bool Discontinued { get; set; }
}
```

Als we een scherm gaan maken waarin we een nieuw product aanmaken (ofwel een ProductModel gaan vullen), zullen we dus een verwijzing naar deze classes moeten maken.

We gaan het scherm wederom maken volgens de huisstijl van deze module (de rode cijfers staan natuurlijk niet in je scherm maar worden gebruikt om hieronder instellingen toe te lichten):



Hieronder vind je per nummer de naam van het control wat gebruikt is en de instellingen die zijn aangepast in de standaard waarde van dit control. Je ziet dat we een Panel gebruiken om de invoervelden (met bijbehorende labels) te 'groeperen'.

Nummer	Opmerkingen
1	De naam van het formulier. Geef hier duidelijk aan dat het een toevoegen (Add) scherm betreft en wat je gaat toevoegen (in dit geval producten).
2	Control: Panel Text: <Naam vd entiteit> gevolgd door toevoegen Font: Segoe UI; 9 pt (die rond hij vaak wat af…)
3	Control: Label Name: Standaard naam mag je laten staan. Text: Naam van het attribuut wat je wil invoeren.
4	Controls die je gebruikt om gegevens in te laten voeren. Dit is natuurlijk afhankelijk van het type attribuut wat je hiermee vult. Dus kunnen het verschillende controls zijn: Control(s): Textbox (txt) of Combobox (cmb) of DateTimePicker (dtp) of NumericUpDown (num) Name: <drie letters (zie hierboven tussen haakjes)><attribuutnaam> bijvoorbeeld txtNaam, cmbLeverancier
5	Control: Button Name: btnAdd Flatsyle: Flast Tekst: Toevoegen Textalign: MiddleRight Width: 164 Height: 48
6	Control: Button Name: btnClose Flatsyle: Flast Tekst: Sluiten Textalign: MiddleRight Width: 144 Height: 48

Zoals je kunt zien staan er bij de twee attributen die een link naar een ander model zijn comboboxen. Dit control kun je gebruiken om de gebruiker van je applicatie te laten kiezen uit een lijst van waarden. Perfect voor onze applicatie. Bij leverancier willen we de gebruiker namelijk laten kiezen uit alle leveranciers in onze applicatie!

Wat we dus moeten doen is zorgen dat onze leveranciers worden toegevoegd aan de combobox. Dat kunnen we op de volgende manier doen:

```
SupplierController supplierController = new SupplierController();

List<SupplierModel> suppliers = supplierController.ReadAll();

foreach (SupplierModel supplier in suppliers)
{
    cmbSupplier.Items.Add(supplier);
}
```

Ofwel, we halen via een SupplierController alle suppliers (leveranciers) op en voegen die toe aan de combobox.

We hebben dan echter 1 probleem. De combobox is een control wat gemaakt is om tekst(strings) te laten zien. We voegen nu geen tekst (ofwel een string) toe, maar we voegen een SupplierModel toe. Standaard weet een combobox niet wat hij daarmee moet doen. Wat hij dan doet is van dit model de methode ToString() aanroepen. Ieder object in C# bevat deze methode. Dat is zo ontworpen. Dat is dan ook de reden dat deze code zo werkt, al werkt hij nog niet zoals we zouden willen. Om dat voor elkaar te krijgen gaan we zelf bepalen wat de ToString methode van een SupplierModel teruggeeft.

Dat doen we door het aanpassen van onze SupplierModel class. Hieraan voegen we de volgende code toe:

```
public override string ToString()
{
    return $"{CompanyName} - {City}";
}
```

Hierdoor geven we nu aan dat als een control (in ons geval de combobox) een stringwaarde van ons object wil hebben hij de naam gevolgd door een – en de stad van ons object terugkrijgt.

Wat we teruggeven via deze methode is aan ons. Voor iedere klasse kan dit iets anders zijn, andere attributen of andere standaard tekst of tekens!

We hebben op deze manier dus een combobox waarvan de items objecten van de klasse SupplierModel zijn. Het gekozen object kunnen we dus toevoegen aan ons object van de klasse ProductModel, immers die heeft een attribuut Supplier wat deze klasse kan bevatten!

Als de gebruiker van de applicatie dus op de knop Toevoegen klikt kunnen we als volgt een nieuw product maken en gaan vullen:

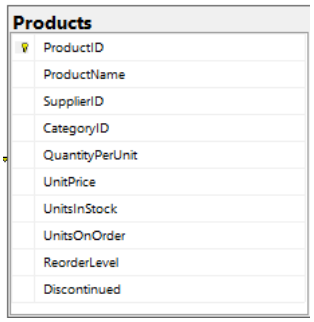
```
ProductModel product = new ProductModel();

product.ProductName = txtProductName.Text;
product.Supplier = (SupplierModel)cmbSupplier.SelectedItem;
...
```

Je ziet dat we het Supplier attribuut vullen met het SelectedItem van de combobox. Omdat dit SelectedItem het datatype object heeft moeten we deze even omzetten (typecasten) naar een SupplierModel.

We hebben nu dus een object waarin andere objecten als attribuut zijn toegevoegd. Dit object willen we via onze controller gaan opslaan in de database. In de database maken we de verbinding met een andere entiteit via een Foreign key.

Als we ons voorbeeld van een product nemen. In de database ziet onze products tabel er als volgt uit:



Je ziet dat we voor de verwijzing naar een leverancier (supplier) in de tabel gebruik maken van de foreign key SupplierID. Als we dus een product gaan aanmaken hebben van het Supplier attribuut alleen het SupplierID nodig. Dit doen we als volgt:

```
public int Create(ProductModel product)
{
    int affectedRows = 0;

    using (SqlConnection con = new SqlConnection(connectionString))
    {
        //Sql statement (select, update en insert)
        string sqlQuery = "INSERT INTO products VALUES (@ProductNameValue, @SupplierIDValue, @Ca
        using (SqlCommand command = new SqlCommand(sqlQuery, con))
        {
            command.Parameters.AddWithValue("ProductNameValue", product.ProductName);
            command.Parameters.AddWithValue("SupplierIDValue", product.Supplier.SupplierID);
            command.Parameters.AddWithValue("CategoryIDValue", product.Category.CategorieID);
            command.Parameters.AddWithValue("QuantityPerUnitValue", product.QuantityPerUnit);
```

Je ziet dat we van ons product naar het Supplier attribuut gaan en daaruit het SupplierID halen. Dat geven we mee aan het INSERT statement wat we uiteindelijk naar de database sturen. Hetzelfde doen we om het CategorieID op te halen.

Je kunt nu **oefening 3.2** maken.

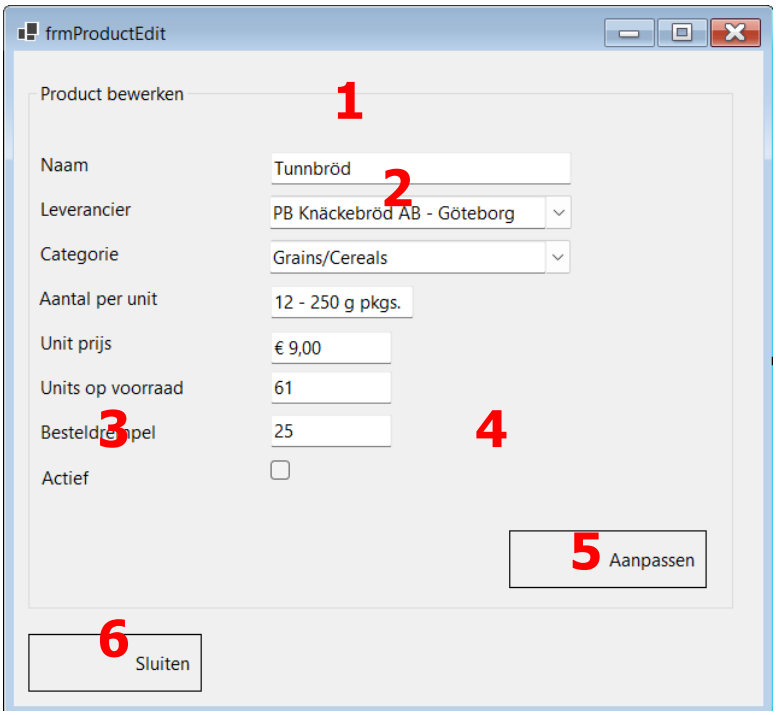
Aanpassen van relationele data

In de vorige paragraaf staat beschreven hoe je een Model met relaties naar andere modellen kunt toevoegen aan de database. In deze paragraaf gaan we kijken hoe we een bestaand object met relaties naar andere modellen kunt aanpassen. We doen dit wederom op basis van het Productmodel dat er als volgt uit ziet:

```
public class ProductModel
{
    0 references | 0 changes | 0 authors, 0 changes
    public int ProductID { get; set; }
    0 references | 0 changes | 0 authors, 0 changes
    public string ProductName { get; set; }
    0 references | 0 changes | 0 authors, 0 changes
    public SupplierModel Supplier { get; set; }
    0 references | 0 changes | 0 authors, 0 changes
    public CategoryModel Category { get; set; }
    0 references | 0 changes | 0 authors, 0 changes
    public int QuantityPerUnit { get; set; }
    0 references | 0 changes | 0 authors, 0 changes
    public decimal UnitPrice { get; set; }
    0 references | 0 changes | 0 authors, 0 changes
    public int UnitsInStock { get; set; }
    0 references | 0 changes | 0 authors, 0 changes
    public int UnitsOnOrder { get; set; }
    0 references | 0 changes | 0 authors, 0 changes
    public int RecorderLevel { get; set; }
    0 references | 0 changes | 0 authors, 0 changes
    public bool Discontinued { get; set; }
}
```

Als we een scherm gaan maken waarin we een bestaand product aanpassen (ofwel een ProductModel gaan voorzien van eventuele nieuwe waarden), zullen we dus een verwijzing naar deze classes moeten maken.

We gaan het scherm wederom maken volgens de huisstijl van deze module (de rode cijfers staan natuurlijk niet in je scherm maar worden gebruikt om hieronder instellingen toe te lichten):



Hieronder vind je per nummer de naam van het control wat gebruikt is en de instellingen die zijn aangepast in de standaard waarde van dit control. Je ziet dat we een Panel gebruiken om de invoervelden (met bijbehorende labels) te 'groeperen'.

Nummer	Opmerkingen
1	De naam van het formulier. Geef hier duidelijk aan dat het een bewerken (Edit) scherm betreft en wat je gaat bewerken (in dit geval producten).
2	Control: Panel Text: <Naam vd entiteit> gevolgd door toevoegen Font: Segoe UI; 9 pt (die rond hij vaak wat af…)
3	Control: Label Name: Standaard naam mag je laten staan. Text: Naam van het attribuut wat je wil aanpassen.
4	Controls die je gebruikt om gegevens aan te laten passen. Dit is natuurlijk afhankelijk van het type attribuut wat je hiermee vult. Dus kunnen het verschillende controls zijn: Control(s): Textbox (txt) of Combobox (cmb) of DateTimePicker (dtp) of NumericUpDown (num) Name: <drie letters (zie hierboven tussen haakjes)><attribuutnaam> bijvoorbeeld txtNaam, cmbLeverancier
5	Control: Button Name: btnEdit Flatsyle: Flast Tekst: Aanpassen Textalign: MiddleRight Width: 164 Height: 48
6	Control: Button Name: btnClose Flatsyle: Flast Tekst: Sluiten Textalign: MiddleRight Width: 144 Height: 48

Zoals je kunt zien staan er bij de twee attributen die een link naar een ander model zijn comboboxen. Dit control kun je gebruiken om de gebruiker van je applicatie te laten kiezen uit een lijst van waarden. Perfect voor onze applicatie. Bij leverancier willen we de gebruiker namelijk laten kiezen uit alle leveranciers in onze applicatie!

Natuurlijk moet het scherm de gegevens weten van de entiteit die/het item dat moet worden aangepast. De gegevens moeten immers bij het tonen van het scherm reeds ingevuld zijn. Dit doen we door bij het aanmaken van het scherm deze entiteit reeds mee te nemen. Dit doen we door het aanpassen van de Constructor van het scherm. Standaard ziet deze functie er in de code als volgt uit:

```
public frmProductEdit()
{
    InitializeComponent();
}
```

We passen dit aan naar de volgende code:

```
private ProductModel productItem;

0 references | 0 changes | 0 authors, 0 changes
public frmProductEdit(ProductModel product)
{
    InitializeComponent();
    productItem = product;
}
```

Je ziet dat we een private variabele productItem gemaakt hebben van het type ProductModel. Daarin bewaren we de entiteit die we moeten aanpassen. Deze employee geven we als parameter mee aan de constructor. In de code van de constructor zetten we vervolgens de meegegeven entiteit in de private variabele.

Vervolgens gaan we de labels die we hebben aangemaakt gaan vullen met de gegevens uit de entiteit. Dit KAN natuurlijk ook in de contructor, maar wij kiezen ervoor om dit in de het OnLoad event van het formulier te doen. Daar plaatsen we code om alle controls te vullen. Zoiets als dit:

```
private void frmProductEdit_Load(object sender, EventArgs e)
{
    txtProductName.Text = productItem.ProductName;
```

Dit werkt prima voor de 'standaard' items zoals strings, integer en decimals. Echter als we een relatie moeten vullen (via een combobox) gaat het helaas niet op deze manier. Om dat te doen moeten we (net als in de vorige paragraaf) onze combobox voorzien van waardes. Nu moeten we echter nog iets meer doen, want we gaan het formulier voorzien van de huidige waardes. Dus onze combobox moet ingesteld worden op de juiste waarde, het juiste item.

Wat we dus moeten doen is zorgen dat onze leveranciers worden toegevoegd aan de combobox. En terwijl we de combobox vullen, moeten we kijken of we op dat moment de huidige leverancier toevoegen. Als dat zo is, moeten we de combobox vertellen dat dat item het actieve item van de combobox moet worden. In code ziet dat er als volgt uit:

```
SupplierController supplierController = new SupplierController();
List<SupplierModel> supplierList = supplierController.ReadAll();

// Assignen van de Supplier objecten aan de combobox
foreach (SupplierModel supplierItem in supplierList)
{
    cmbSupplier.Items.Add(supplierItem);

    // Bepaal het geselecteerde item
    if (supplierItem.ToString() == productItem.Supplier.ToString())
    {
        cmbSupplier.SelectedIndex = cmbSupplier.Items.Count - 1;
    }
}
```

Als de gebruiker van de applicatie dus op de knop Aanpassen klikt kunnen we als volgt het bestaande product gaan vullen met de nieuwe data:

```
private void btnEdit_Click(object sender, EventArgs e)
{
    productItem.ProductName = txtProductName.Text;
    productItem.Supplier = (SupplierModel)cmbSupplier.SelectedItem;
```

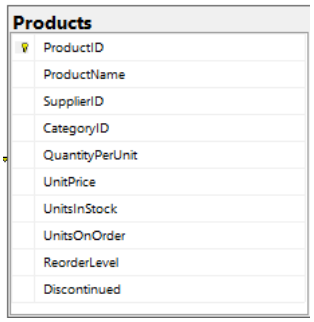
Je ziet dat we het Supplier attribuut vullen met het SelectedItem van de combobox. Omdat dit SelectedItem het datatype object heeft moeten we deze even omzetten (typecasten) naar een SupplierModel.

Vervolgens moeten we de gewijzigde gegevens opslaan in de database. Communicatie met de database verloopt via de Controller, dus dat betekent dus dat we in onze controller een methode moeten maken om een element uit de database aan te passen. Als we deze methode gemaakt hebben kunnen we die als volgt aanroepen:

```
employeeController.Update(employeeItem);
```

We hebben nu dus een object waarin andere objecten als attribuut zijn toegevoegd. Dit object willen we via onze controller gaan opslaan in de database. In de database maken we de verbinding met een andere entiteit via een Foreign key.

Als we ons voorbeeld van een product nemen. In de database ziet onze products tabel er als volgt uit:



Je ziet dat we voor de verwijzing naar een leverancier (supplier) in de tabel gebruik maken van de foreign key SupplierID. Als we dus een product gaan aanmaken hebben van het Supplier attribuut alleen het SupplierID nodig. Dit doen we als volgt:

```
public int Update(ProductModel product)
{
    int affectedRows = 0;
    using (SqlConnection con = new SqlConnection(connectionString))
    {
        //Sql statement (select, update en insert)
        string sqlQuery = "UPDATE products SET productname = @ProductNameValue SupplierID = @SupplierIDValue, CategoryI
        using (SqlCommand command = new SqlCommand(sqlQuery, con))
        {
            command.Parameters.AddWithValue("ProductIDValue", product.ProductID);
            command.Parameters.AddWithValue("ProductNameValue", product.ProductName);
            command.Parameters.AddWithValue("SupplierIDValue", product.Supplier.SupplierID);
            command.Parameters.AddWithValue("CategoryIDValue", product.Category.CategoryID);
```

Je ziet dat we van ons product naar het Supplier attribuut gaan en daaruit het SupplierID halen. Dat geven we mee aan het UPDATE statement wat we uiteindelijk naar de database sturen. Hetzelfde doen we om het CategorieID op te halen.

Je kunt nu **oefening 3.3** maken.