Gevorderd standalone

Realiseren

hoofdstuk

2

Selecteren en weergeven van gegevens uit meerder tabellen

Algemene informatie

Onderwerp	Selecteren en weergeven van 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

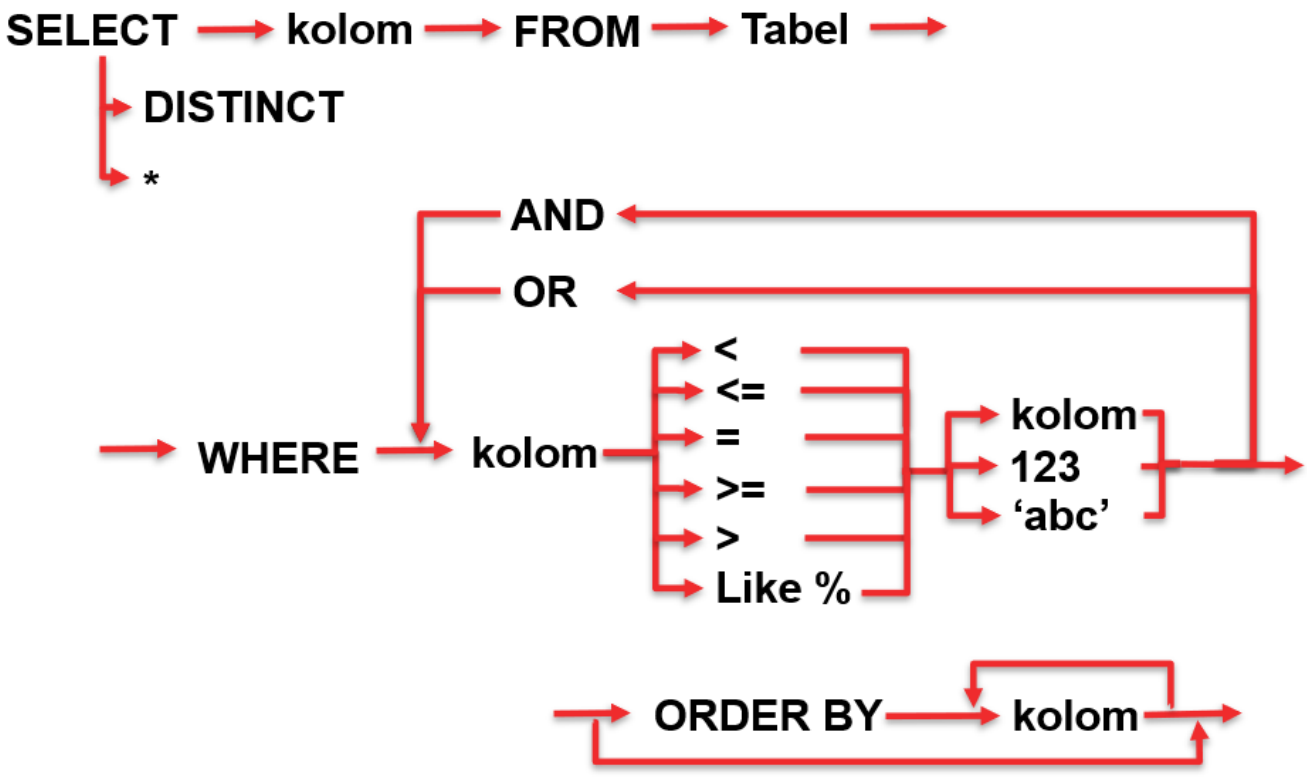
Binnen een applicatie maak je gebruik van gegevens die je wilt weergeven, toevoegen, aanpassen en/of verwijderen. In de praktijk wordt deze data opgeslagen in een database. De data die in de database zit in meerdere tabellen aangezien we werken met een relationele database. Dat betekent dat we gegevens 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

Selecteren van gegevens

In thema 7 hebben jullie gekeken naar het SQL statement SELECT. Dit (DML-statement) wordt gebruikt voor het raadplegen van gegevens uit een SQL-database. Elk statement heeft een syntax. Dit is de schrijfwijze van het statement. Hieronder staat de vereenvoudigde syntax van het select-statement weergegeven.

Select-statement



Gegevens selecteren doe je dus met het SELECT statement. Tot nu toe hebben achter FROM telkens maar 1 tabel opgenomen. In deze reader gaan we leren hoe we hier meerdere tabellen kunnen opnemen zodat we informatie uit meerdere tabellen kunnen samenvoegen.

SQL JOIN

Als we gegevens uit meer dan 1 tabel willen selecteren gebruiken we daarvoor een JOIN. Een JOIN-clause is een onderdeel van een SQL-query, waardoor records van twee of meer tabellen uit een database gecombineerd kunnen worden. Hierdoor kunnen we het bovenste gedeelte van de syntax van een SELECT statement uitbreiden:

SELECT → **Kolom**

FROM → **Tabel1**

JOIN → **Tabel2**

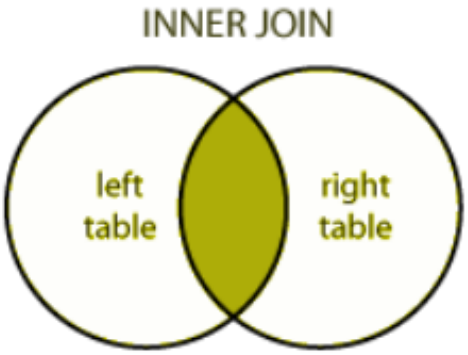
ON → **Tabel1.id = Tabel2.id**

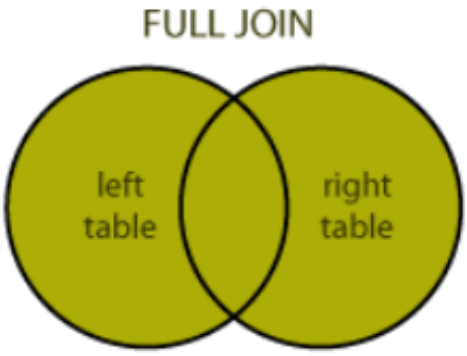
etc

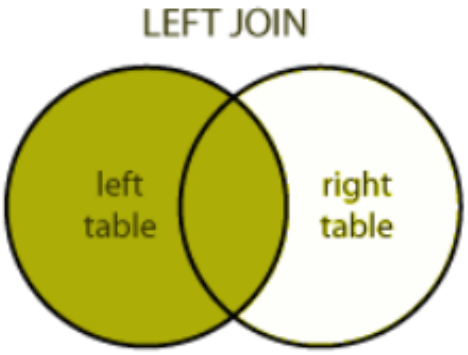
Hiermee voegen we alle rijen uit tabel1 en alle rijen uit tabel2 samen, wanneer het veld id in beide tabellen dezelfde waarde heeft.

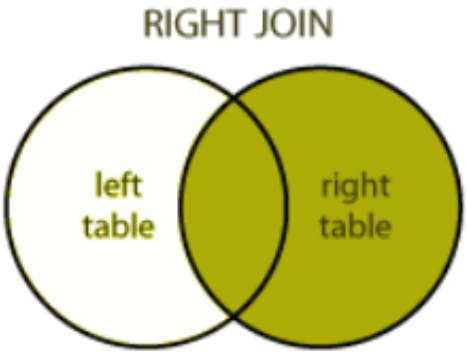
Soorten joins

Er zijn verschillende manieren waarop we informatie uit twee tabellen kunnen samenvoegen. Deze zijn hieronder grafisch weergegeven. We zullen de diverse opties ook nog even los bespreken en uitleggen.

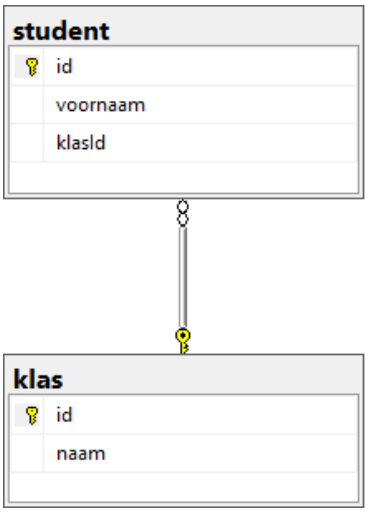








Bij de uitleg van de verschillende soorten JOINS maken we in de uitleg gebruikt van twee tabellen (student en klas). Het ERD van deze twee tabellen is als volgt:



Zoals je ziet sla je bij een student op in welke klas deze student zit (klasId). Er bestaat dus een een-op-veel relatie tussen student en klas, namelijk 1 student zit in 1 klas, maar in 1 klas kunnen meerdere studenten zitten.

De inhoud van de tabel student is in onze voorbeelden als volgt:

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

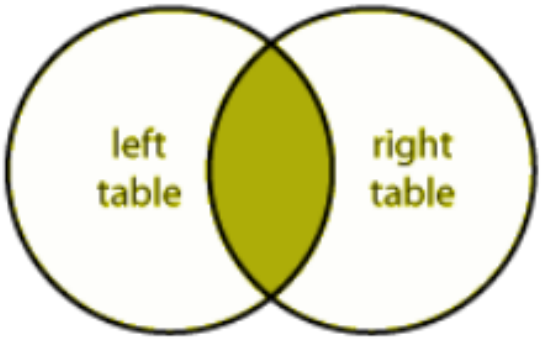
Je ziet dus dat Joris in de klas met KlasId 1 zit, Maartje zit in de klas met KlasId 3 en Pieter heeft geen klas aangezien zijn klasID NULL (oftewel geen waarde) bevat.

De inhoud van de tabel klas is in onze voorbeelden als volgt:

id	naam
1	IO2A4
2	IO2B4
3	IO2C4
4	IO2D4

Door de inhoud van de twee tabellen te combineren weten we nu dat Joris in klas IO2A4 zit (die heeft immers KlasId 1), Maartje zit in de klas IO2C4 (die heeft immers KlasId 3).

(Inner) JOIN



Met een (Inner) JOIN vraag je het resultaat op waarbij geld dat het gekoppelde veld in allebei de tabellen moet voorkomen. Zoals je in onderstaand voorbeeld kunt zien krijg je als resultaat nu alle studenten die een klas hebben. Je mist in het resultaat Pieter omdat hij geen klas heeft.

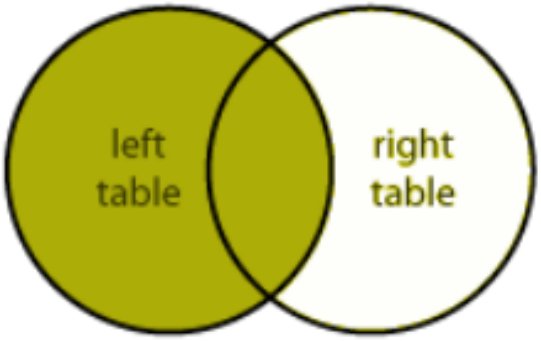
Voorbeeld statement:

```
SELECT student.voornaam, klas.naam AS klas
FROM student
JOIN klas
ON klas.id = student.klasId;
```

Resultaat:

voornaam	klas
Joris	IO2A4
Thomas	IO2A4
Niels	IO2B4
Hendrik	IO2B4
Maartje	IO2C4
Koen	IO2C4

LEFT JOIN



Met een LEFT JOIN vraag je het resultaat op waarbij geld dat we in ieder geval alle gegevens uit de linkse (ofwel eerste) tabel willen laten terugkomen in het resultaat. Deze gegevens vullen we aan met gegevens uit de rechtse tabel, mits deze bestaan. Anders komt er NULL te staan. Zoals je in onderstaand voorbeeld kunt zien krijg je als resultaat nu alle studenten. Bij Pieter staat geen klas (NULL) omdat hij geen klas heeft.

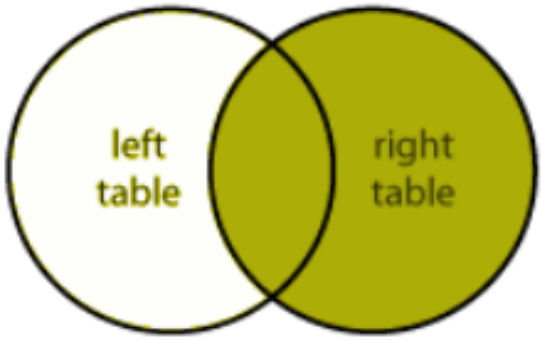
Voorbeeld statement:

```
SELECT student.voornaam, klas.naam AS klas

FROM student

LEFT JOIN klas

ON klas.id = student.klasId;
```

Resultaat:

voornaam	klas
Joris	IO2A4
Thomas	IO2A4
Niels	IO2B4
Hendrik	IO2B4
Maartje	IO2C4
Koen	IO2C4
Pieter	NULL

RIGHT JOIN



Een RIGHT JOIN is het tegenovergestelde van de LEFT JOIN. Met een RIGHT JOIN vraag je het resultaat op waarbij geld dat we in ieder geval alle gegevens uit de rechtse (ofwel tweede) tabel willen laten terugkomen in het resultaat. Deze gegevens vullen we aan met gegevens uit de linkse (ofwel eerste) tabel, mits deze bestaan. Anders komt er NULL te staan. Zoals je in onderstaand voorbeeld kunt zien krijg je als resultaat nu alle klassen. Bij IO2D4 staat geen student (NULL) omdat er geen student aan deze klas is gekoppeld. Ook missen we student Pieter omdat deze aan geen enkele klas is gekoppeld.

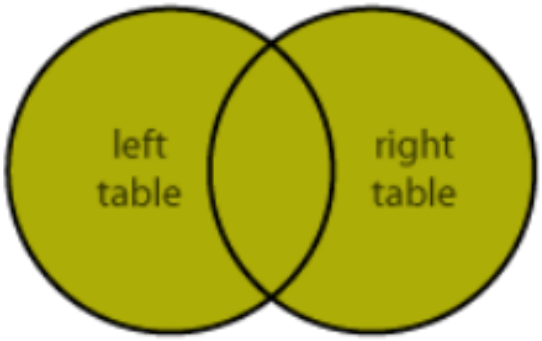
Voorbeeld statement:

```
SELECT student.voornaam, klas.naam AS klas

FROM student

RIGHT JOIN klas

ON klas.id = student.klasId;
```

Resultaat:

voornaam	klas
Joris	IO2A4
Thomas	IO2A4
Niels	IO2B4
Hendrik	IO2B4
Maartje	IO2C4
Koen	IO2C4
NULL	IO2D4

FULL JOIN



Met een FULL JOIN vraag je het resultaat op waarbij geld dat we in ieder geval alle gegevens uit de linkse (ofwel eerste) tabel EN de rechtse (ofwel tweede) willen laten terugkomen in het resultaat. Deze gegevens vullen we aan met gegevens uit de andere tabel, mits deze bestaan. Anders komt er NULL te staan. Zoals je in onderstaand voorbeeld kunt zien krijg je als resultaat nu alle studenten EN alle klassen. Bij Pieter staat geen klas (NULL) omdat hij geen klas heeft. Bij IO2D4 staat geen student (NULL) omdat er geen student aan deze klas is gekoppeld.

Voorbeeld statement:

```
SELECT student.voornaam, klas.naam AS klas
FROM student
FULL JOIN klas
ON klas.id = student.klasId;
```

Resultaat:

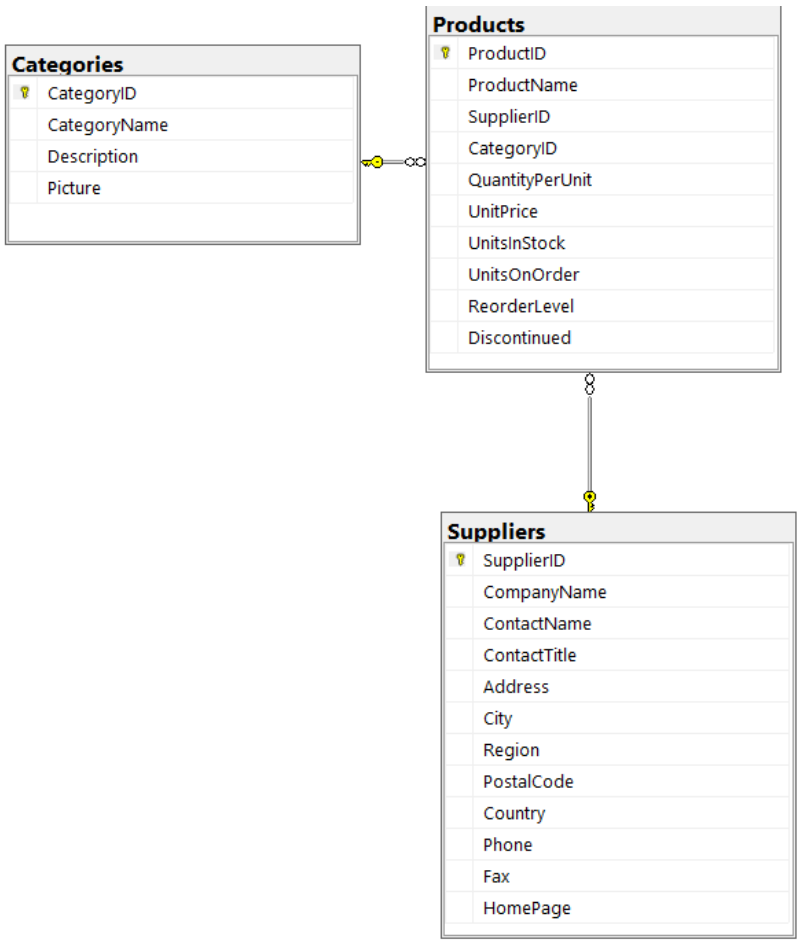
voornaam	klas
Joris	IO2A4
Thomas	IO2A4
Niels	IO2B4
Hendrik	IO2B4
Maartje	IO2C4
Koen	IO2C4
Pieter	NULL
NULL	IO2D4

Je kunt nu **oefening 2.1** en **oefening 2.2** maken.

Model bouwen

In de laag Model plaatsen we alle modellen (in de vorm van Classes) van onze entiteiten die we hebben kunnen vinden via de bekende modeleertechnieken. Vanuit je modelleer/data analyse proces heb je een overzicht van je modellen, alsmede een Entiteit Relatie Diagram (wat als basis dient voor je database).

Als we nu naar (een gedeelte van) het ERD van de NorthWind database kijken dan zien we daarin dat er een link is tussen de tabel Producs en de tabellen Categories en Suppliers. Overigens is er in die database ook nog een link tussen de tabel Products en Order Details, maar die laten we nu even achterwege.



Wat we hieruit kunnen afleiding is dat een Product hoort tot een Categorie (en een categorie meerdere Producten kan hebben) en dat een product hoort bij een Supplier (en een Supplier meerdere Producten kan hebben). Kortom we hebben hier te maken met twee relaties van het type een op veel.

Hoe gaan we van deze ERD nu modellen maken in onze applicatie? Om te beginnen kunnen we een model maken van een Categorie en een model maken van een Supplier. Hiervoor kunnen we kijken naar de velden in bovenstaande ERD om te weten welke attributen we nodig hebben. Feitelijk maken we deze modellen op dezelfde manier als dat we de modellen in hoofdstuk 7 van thema 7 gemaakt hebben.

Dit levert de volgende twee classes in je applicatie op:

```
public class SupplierModel
{
    0 references | 0 changes | 0 authors, 0 changes
    public int SupplierID { get; set; }
    0 references | 0 changes | 0 authors, 0 changes
    public string CompanyName { get; set; }
    0 references | 0 changes | 0 authors, 0 changes
    public string ContactName { get; set; }
    0 references | 0 changes | 0 authors, 0 changes
    public string ContactTitle { get; set; }
    0 references | 0 changes | 0 authors, 0 changes
    public string Address { get; set; }
    0 references | 0 changes | 0 authors, 0 changes
    public string City { get; set; }
    0 references | 0 changes | 0 authors, 0 changes
    public string Region { get; set; }
    0 references | 0 changes | 0 authors, 0 changes
    public string PostalCode { get; set; }
    0 references | 0 changes | 0 authors, 0 changes
    public string Country { get; set; }
    0 references | 0 changes | 0 authors, 0 changes
    public string Phone { get; set; }
    0 references | 0 changes | 0 authors, 0 changes
    public string Fax { get; set; }
    0 references | 0 changes | 0 authors, 0 changes
    public string HomePage { get; set; }
}
```

```
public class CategoryModel
{
    0 references | 0 changes | 0 authors, 0 changes
    public int CategorieID { get; set; }
    0 references | 0 changes | 0 authors, 0 changes
    public string CategoryName { get; set; }
    0 references | 0 changes | 0 authors, 0 changes
    public string Description { get; set; }
    0 references | 0 changes | 0 authors, 0 changes
    public string Picture { get; set; }

}
```

Als we vervolgens naar het model gaan kijken voor een Product, dan zien we dat daarin twee verwijzingen staan. Een keer naar een Category en een keer naar een Supplier. In de database wordt dit opgelost met een zogenaamde Foreign key, in het model lossen we dit echter anders op. In een model kunnen we namelijk een attribuut toevoegen van b.v. het type CategoryModel. Op die manier verwijzen we direct naar het juiste model met daarin de informatie. We maken dus geen attribuut van het type int, string, bool, DateTime, maar we maken een attribuut van een door onszelf gemaakte klasse.

In plaats van alleen een verwijzing nemen we in het model dus een volledig ander model op als attribuut. Een attribuut waarin we een hele klasse stoppen. Dat geeft mogelijkheden natuurlijk. Hoe ziet het model van een Product er dan uit? Dat zou je als volgt kunnen opbouwen:

```
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; }
}
```

Zoals je ziet zijn de veldnamen nu ook niet één op één overgenomen uit de ERD. In plaats van SupplierID (het ID van de Supplier) noemen we het attribuut Supplier. Er zit immers een volledig supplierModel in, dus dekt de naam Supplier veel beter de lading/inhoud dan de naam SupplierID. Hetzelfde zie je gebeuren bij het attribuut Category.

Controller bouwen

Een controller is, zoals je inmiddels al weet, de verbinding van je code naar je datalaag. Inmiddels heb je in je controler vier methodes (ReadAll, Delete, Insert en Update) gemaakt. Dan zijn in principe de basis methodes in een controller.

Tot nog toe hebben we geleerd om in de controller queries te gebruiken die gegeven halen uit één tabel. Nu gaan we kijken hoe we in de controller gegevens uit meerdere tabellen kunnen halen. Dit is grotendeels hetzelfde als uit één tabel, dus het meeste zal je bekend voorkomen.

Om een database via een select query uit te lezen, dien je de volgende zaken (in volgorde) te programmeren:

1. Using System.Data.SqlClient
2. **ConnectionString** definiëren.
3. **SqlConnection** definiëren en starten (via **.Open()**).
4. **SqlCommand** definiëren (query komt hier), inclusief opgave **SqlParameters**
5. Database uitlezen via **.ExecuteReader()** en het **SqlDataReader** object vullen.
6. Door deze **SqlDataReader** itereren (loopen) via de methode **.Read()** om de rows één voor één uit te lezen en deze in een **List<>** met objecten te plaatsen.

Voor stap 1 t/m 3 verwijs ik je naar reader 7 van thema 7. Daar staan deze stappen uitgebreid beschreven en voor deze reader mag je volledig dezelfde stappen volgen.

SQLCommand

Met behulp van de classe SqlCommand sturen we de query naar de database. Deze query is nu natuurlijk iets uitgebreider dan de queries die we tot nu toe gebruikt hebben. In ons voorbeeld moeten we nu namelijk zowel gegevens van een product, alsook gegevens van een Supplier en gegevens van een Category ophalen. Hiervoor hebben we dus gegevens uit drie (!) tabellen nodig.

Onze query gaat er dus op de volgende manier uitzien:

```
SELECT *
FROM Products
LEFT JOIN Categories ON Products.CategoryID = Categories.CategoryID
LEFT JOIN Suppliers ON Products.SupplierID = Suppliers.SupplierID
```

We maken hier bewust de keuze voor een LEFT JOIN, waardoor we altijd ALLE producten ophalen, ook al heeft een product geen categorie of supplier.

Zoals je ziet is deze query een langere query. We plaatsen deze query daarom eerst in een variabele en die variabele geven we vervolgens mee als we een SqlCommand object aanmaken.

Let er hierbij op dat ondanks dat deze string meerdere regels bevat hij feitelijk 1 lange tekst is, alsof je alle drie de regels achter elkaar plaatst. Daarom staat er als laatste teken in regel 1 en regel 2 een spatie. Als je die niet opneemt komt het eerste teken van de volgende regel direct achter het laatste teken en krijg je niet de query die je wilt.

```
string sqlQuery = "SELECT * FROM Products "+
"LEFT JOIN Categories ON Products.CategoryID = Categories.CategoryID "+
"LEFT JOIN Suppliers ON Products.SupplierID = Suppliers.SupplierID"

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

Net als met het connection object is het netjes om aan te geven tot wanneer je het object gebruikt, dus is het goed om ook bij het command een using commando te gebruiken.

```
using (SqlConnection con = new SqlConnection(connectionString))
{
      //Sql statement (select, update en insert)
      string sqlQuery = "SELECT * FROM Products "+
      "LEFT JOIN Categories ON Products.CategoryID = Categories.CategoryID "+
      "LEFT JOIN Suppliers ON Products.SupplierID = Suppliers.SupplierID";
      using (SqlCommand command = new SqlCommand(sqlQuery, con))
      {
      }
}
```

Let op dat je het using commando van je SqlCommand object wederom binnen de using van de SqlConnection object plaatst. Het SqlCommand heeft deze SqlConnection namelijk nodig. Binnen de tweede accolades gaan we nu de code plaatsen waarmee we de gegevens uit de database gaan halen.

ExecuteReader

We hebben inmiddels een object wat de verbinding naar de database verzorgd en een object wat een query naar een database kan sturen. Het wordt tijd dat we dat ook daadwerkelijk gaan doen. Om te beginnen gaan we zorgen dat het object wat de verbinding verzorgd ook daadwerkelijk die verbinding naar de database opent. Dat doen we met de methode Open(). Dus bijvoorbeeld zo:

```
con.Open();
```

Vervolgens willen we een SELECT statement uitvoeren. Die doen we door het aanroepen van de methode ExecuteReader() van het SqlCommand object. De methode geeft als resultaat gegevens van het datatype **SqlDataReader** terug. Dus maken voor een variabele van dit datatype aan en zetten we daar het resultaat in. Dus bijvoorbeeld zo:

```
SqlDataReader reader = command.ExecuteReader();
```

Wegschrijven naar een list

We hebben nu in ons object/variabele met de naam *reader* het resultaat van de SQL-query zitten. Het enige dat we nu nog moeten doen, is de resulaten uitlezen. Dit kunnen we doen via een while loop. Zo kunnen we de resultaten één voor één ophalen. Dit betekent dat we 0, 1 of meer rijen op gaan halen. Een datatype in C# dat we hiervoor kunnen gebruiken is de List. Een List is een lijst van objecten van een bepaald type. Dat kan dus een lijst van strings zijn (als volgt: List<string>) maar bijvoorbeeld ook een lijst van ProductModel objecten (als volgt: List< ProductModel>).

Als we onze query uitvoeren in de SQL Server Management Studio levert dat het volgende resultaat op:

```
1  SELECT *
2  FROM Products
3  LEFT JOIN Categories ON Products.CategoryID = Categories.CategoryID
4  LEFT JOIN Suppliers ON Products.SupplierID = Suppliers.SupplierID
5
6
```

146 %

Results | Messages

	ProductID	ProductName	SupplierID	CategoryID	QuantityPerUnit	UnitPrice	UnitsInStock	UnitsOnOrder	ReorderLevel	Discontinued	CategoryID	CategoryName	Description	Pictu
1	1	Chai	1	1	10 boxes x 20 bags	18,00	39	0	10	0	1	Beverages	Soft drinks, coffees, teas, beers, and ales	0x15
2	2	Chang	1	1	24 - 12 oz bottles	19,00	17	40	25	0	1	Beverages	Soft drinks, coffees, teas, beers, and ales	0x15
3	3	Aniseed Syrup	1	2	12 - 550 ml bottles	10,00	13	70	25	0	2	Condiments	Sweet and savory sauces, relishes, spreads, and ...	0x15
4	4	Chef Anton's Cajun Seasoning	2	2	48 - 6 oz jars	22,00	53	0	0	0	2	Condiments	Sweet and savory sauces, relishes, spreads, and ...	0x15
5	5	Chef Anton's Gumbo Mix	2	2	36 boxes	21,35	0	0	0	1	2	Condiments	Sweet and savory sauces, relishes, spreads, and ...	0x15
6	6	Grandma's Boysenberry Spread	3	2	12 - 8 oz jars	25,00	120	0	25	0	2	Condiments	Sweet and savory sauces, relishes, spreads, and ...	0x15
7	7	Uncle Bob's Organic Dried Pears	3	7	12 - 1 lb pkgs.	30,00	15	0	10	0	7	Produce	Dried fruit and bean curd	0x15
8	8	Northwoods Cranberry Sauce	3	2	12 - 12 oz jars	40,00	6	0	0	0	2	Condiments	Sweet and savory sauces, relishes, spreads, and ...	0x15
9	9	Mishi Kobe Niku	4	6	18 - 500 g pkgs.	97,00	29	0	0	1	6	Meat/Poultry	Prepared meats	0x15
10	10	Ikura	4	8	12 - 200 ml jars	31,00	31	0	0	0	8	Seafood	Seaweed and fish	0x15
11	11	Queso Cabrales	5	4	1 kg pkg.	21,00	22	30	30	0	4	Dairy Products	Cheeses	0x15
12	12	Queso Manchego La Pastora	5	4	10 - 500 g pkgs.	38,00	86	0	0	0	4	Dairy Products	Cheeses	0x15
13	13	Konbu	6	8	2 kg box	6,00	24	0	5	0	8	Seafood	Seaweed and fish	0x15
14	14	Tofu	6	7	40 - 100 g pkgs.	23,25	35	0	0	0	7	Produce	Dried fruit and bean curd	0x15
15	15	Genen Shouyu	6	2	24 - 250 ml bottles	15,50	39	0	5	0	2	Condiments	Sweet and savory sauces, relishes, spreads, and ...	0x15
16	16	Pavlova	7	3	32 - 500 g boxes	17,45	29	0	10	0	3	Confections	Desserts, candies, and sweet breads	0x15
17	17	Alice Mutton	7	6	20 - 1 kg tins	39,00	0	0	0	1	6	Meat/Poultry	Prepared meats	0x15
18	18	Carnarvon Tigers	7	8	16 kg pkg.	62,50	42	0	0	0	8	Seafood	Seaweed and fish	0x15
19	19	Teatime Chocolate Biscuits	8	3	10 boxes x 12 pie...	9,20	25	0	5	0	3	Confections	Desserts, candies, and sweet breads	0x15
20	20	Sir Rodney's Marmalade	8	3	30 gift boxes	81,00	40	0	0	0	3	Confections	Desserts, candies, and sweet breads	0x15
21	21	Sir Rodney's Scones	8	3	24 pkgs. x 4 pieces	10,00	3	40	5	0	3	Confections	Desserts, candies, and sweet breads	0x15
22	22	Gustaf's Knäckebröd	9	5	24 - 500 g pkgs.	21,00	104	0	25	0	5	Grains/Cereals	Breads, crackers, pasta, and cereal	0x15
23	23	Tunnbröd	9	5	12 - 250 g pkgs.	9,00	61	0	25	0	5	Grains/Cereals	Breads, crackers, pasta, and cereal	0x15
24	24	Guaraná Fantástica	10	1	12 - 355 ml cans	4,50	20	0	0	1	1	Beverages	Soft drinks, coffees, teas, beers, and ales	0x15
25	25	NuNuCa Nuß-Nougat-Creme	11	3	20 - 450 g glasses	14,00	76	0	30	0	3	Confections	Desserts, candies, and sweet breads	0x15
26	26	Gumbär Gummibärchen	11	3	100 - 250 g bags	31,23	15	0	0	0	3	Confections	Desserts, candies, and sweet breads	0x15

Query executed successfully. | DESKTOP-SLOJK6C\SQLEXPRESS ... | DESKTOP-SLOJK6C\Martij... | Northwind | 00:00:01 | 77 rows

Je ziet dat er het resultaat bestaat uit 77 rijen (ook wel records genoemd, dit zie je rechts onderin in de gele balk staan) die elk bestaan uit een groot aantal kolommen. Je krijgt namelijk ALLE kolommen van de products tabel terug, maar ook ALLE kolommen van de Categories tabel en ALLE kolommen van de Suppliers tabel!

Om deze rijen uit te lezen starten we met het lezen van de eerste rij. Dat doen we met de methode Read(). Deze methode heeft als resultaat een boolean die aangeeft of er een rij is opgehaald. Dus als het resultaat true is, is er een rij opgehaald en kunnen we de kolommen van die rij uitlezen. Hier kunnen we dus een while loop voor maken:

```
while (reader.Read())

{

        // Code om de gegevens per rij/record uit te lezen…

}
```

Vervolgens kunnen we de kolommen aan door de variabele reader aan te spreken als een List, waarbij we als optie de naam van de kolom opgeven. Zo levert reader[“ProductID”] bij de eerste rij de waarde 1 op. En levert reader[“ProductName”] bij de derde rij de waarde Aniseed Syrup op.

Per rij willen we de waardes natuurlijk opslaan in het geheugen. Dat doen we in een variabele van het type ProductModel (die we hebben aangemaakt als Model een paar bladzijdes terug. Dit model ziet er als volgt uit:

```
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; }
}
```

We gaan nu per rij een nieuwe variabele van dit type aanmaken. Dat doen we als volgt:

```
ProductModel product = new ProductModel();
```

Vervolgens kunnen we de attributen van dit object vullen. Bijvoorbeeld zo:

```
product.ProductID = (int) reader["ProductID"];
```

Je ziet dat we het resultaat meteen ‘casten’ naar het juiste datatype door er (int) voor te zetten. In plaats van *int* kan hier ook b.v. *DateTime*, *string*, *decimal* of *bool* staan.

Vervolgens kunnen we natuurlijk de andere attributen gaan vullen.

```
product.ProductName = (string) reader["ProductName"];
```

Het derde attribuut wat we echter tegenkomen is een attribuut van het type SupplierModel. Dat moeten we op een andere manier gaan vullen. Hier komt namelijk een volledig SupplierModel in te staan!

Dus als eerste gaan nu een nieuw object van deze klasse aanmaken!

```
SupplierModel supplier = new SupplierModel();
```

Vervolgens gaan we deze variabele vullen. De informatie voor de inhoud kunnen we uit de rij halen, immer onze query heeft ook de waardes uit de supplier tabel opgehaald! Dus gaan we als volgt verder:

```
supplier.SupplierID = (int) reader["SupplierID"];

supplier.CompanyName = (int) reader["CompanyName"];

…

supplier.HomePage= (int) reader["HomePage"];
```

Als we alle velden van het SupplierModel object hebben gevuld, kunnen we het SupplierModel attribuut van ons product vullen:

```
product.Supplier = supplier;
```

Op eenzelfde manier maken en vullen we een object van het type CategoryModel en geven het Category attribuut van ons product de waarde van dit object.

Nu is het product object gevuld met de gegevens uit de rij/record die we aan het inlezen zijn. Dit product voegen we toe aan een lijst. Deze lijst wordt uiteindelijk het resultaat van de ReadAll methode! Zorg dus dat je in de start van je ReadAll methode een variabele van het type List<modelnaam> aanmaakt. In ons voorbeeld zouden we de volgende lijst aan moeten maken:

```
List<ProductModel> productlist = new List<ProductModel>();
```

Aan deze lijst voegen we het zojuist gemaakte en gevulde object employee toe.

```
productlist.Add(product);
```

Nadat we alle items uit de database hebben gelezen (op dat moment zal reader.Read() als resultaat False opleveren) hoeven we alleen nog maar het resultaat van de methode gelijk te maken aan de inhoud van onze lijst met objecten.

```
return productlist;
```

Zoals je ziet verschilt deze implementatie van de ReadAll niet heel veel van de implementatie die je al kent. Het verschil is dat je nu bij het uitlezen niet één maar meerdere objecten maakt en vult met de gegevens!

View bouwen

We hebben inmiddels onze modellen gemaakt, onze controller methode geschreven dus we kunnen in de applicatie een lijst producten ophalen. Nu moeten we deze lijst natuurlijk nog laten zien aan de gebruiker!

Ook daarvoor verwijs ik hier grotendeels naar reader 7 van thema 7. Deze view gaat op dezelfde manier opgebouwd worden, dezelfde controls, dezelfde layout.

Het verschil met de view zoals we die in thema 7 gebouwd hebben is de hoeveelheid data. Zoals je gezien hebt in SQL Server Management Studio bij het uitvoeren van de query, halen we enorm veel informatie op. Dat geeft weinig overzicht voor een gebruiker, dus onze taak bij deze view is filteren! Welke informatie van (in ons voorbeeld een product) is interessant voor de gebruiker om te zien in het overzicht? Moeten we bijvoorbeeld alle informatie van de supplier tonen? Waarschijnlijk niet, waarschijnlijk is in dit voorbeeld de naam van de supplier al voldoende voor de gebruiker. Moeten we alle informatie van de categorie tonen? Waarschijnlijk ook niet en is ook hier de naam voldoende.

We kunnen dus een aantal kolommen in de listview aanmaken. In ons voobeeld zouden dat de volgende kolommen kunnen zijn:

```
lvProducts.Columns.Add("Productnaam");
lvProducts.Columns.Add("Leverancier");
lvProducts.Columns.Add("Categorie");
lvProducts.Columns.Add("Aantal per unit");
lvProducts.Columns.Add("Unit prijs");
lvProducts.Columns.Add("Units op voorraad");
lvProducts.Columns.Add("Unit in bestelling");
```

Zoals beschreven in reader 7, thema 7 plaatsen we deze code in het Load event van onze view (Form).

Onze view bouwen we verder op dezelfde manier, ook de methode waarmee we de Listview van data voorzien. In deze methode (private void FillListView()) geven we voor iedere kolom de waarde op. Dat gebeurt, zoals te lezen is in reader 7, thema 7 middels een foreach loop. Deze kunnen we in ons voorbeeld als volgt vorm geven:

```
foreach (ProductModel product in productList)
{
      ListViewItem lvItem = new ListViewItem(product.ProductName);
      lvItem.SubItems.Add(product.Supplier.CompanyName);
      lvItem.SubItems.Add(product.Category.CategoryName);
      …
      lvItem.SubItems.Add(product.UnitsOnOrder.ToString());

      lvItem.Tag = product;
      lvProducts.Items.Add(lvItem);
}
```

Zoals je hierboven ziet halen we de CompanyName van de Supplier uit het attribuut Supplier van ons product. Dat attribuut is immer een object van het type SupplierModel en heeft dus het attribuut CompanyName. We kunnen dit attribuut benaderen op dezelfde manier als dat we los object van het type SupplierModel zouden benaderen.

```
lvItem.SubItems.Add(product.Supplier.CompanyName);
```

Je kunt nu **oefening 2.3** maken.