



Noël Vaes

Java Trainer & Consultant



JDBC

Roode Roosstraat 5
3500 Hasselt
België

+32 474 38 23 94
noel@noelvaes.eu
www.noelvaes.eu

Vrijwel alle namen van software- en hardwareproducten die in deze cursus worden genoemd, zijn tegelijkertijd ook handelsmerken en dienen dienovereenkomstig te worden behandeld.

Alle rechten voorbehouden. Niets uit deze uitgave mag worden verveelvoudigd, opgeslagen in een geautomatiseerd gegevensbestand of openbaar worden gemaakt in enige vorm of op enige wijze, hetzij elektronisch, mechanisch, door fotokopieën, opnamen of op enige andere manier, zonder voorafgaande schriftelijke toestemming van de auteur. De enige uitzondering die hierop bestaat, is dat eventuele programma's en door de gebruiker te typen voorbeelden mogen worden ingevoerd opgeslagen en uitgevoerd op een computersysteem, zolang deze voor privédoeleinden worden gebruikt, en niet bestemd zijn voor reproductie of publicatie.

Correspondentie inzake overnemen of reproductie kunt u richten aan:

Noël Vaes
Roode Roosstraat 5
3500 Hasselt
België

Tel: +32 474 38 23 94

noel@noelvaes.eu
www.noelvaes.eu

Ondanks alle aan de samenstelling van deze tekst bestede zorg, kan de auteur geen aansprakelijkheid aanvaarden voor eventuele schade die zou kunnen voortvloeien uit enige fout, die in deze uitgave zou kunnen voorkomen.

17/06/2017

Copyright© 2017 Noël Vaes



Inhoudsopgave

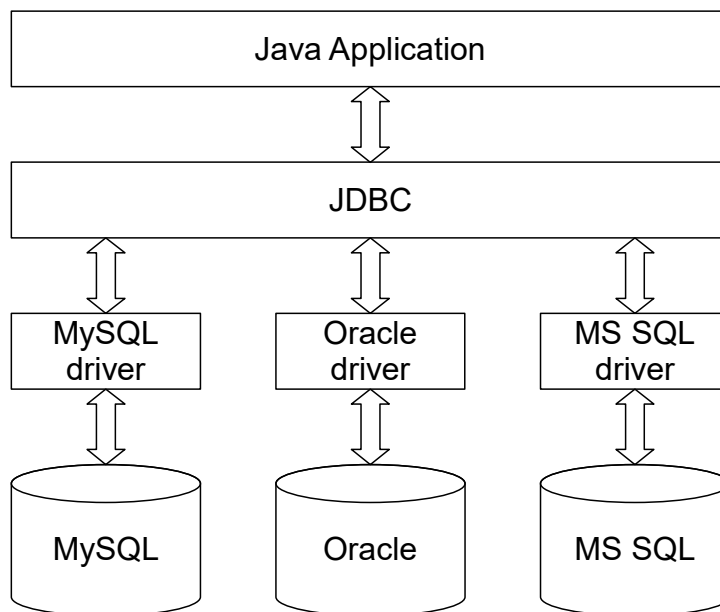
Hoofdstuk 1: JDBC.....	4
1.1 Inleiding.....	4
1.2 De databaseserver.....	4
1.3 Een verbinding maken met een database.....	5
1.3.1 De database-driver laden.....	5
1.3.2 Een connectie maken.....	6
1.4 SQL-commando's gebruiken.....	8
1.4.1 Een statement creëren.....	8
1.4.2 Gegevens uit een database opvragen	9
1.4.3 Gegevens wijzigen in een database.....	14
1.4.4 Gegevens toevoegen aan de database.....	14
1.4.5 Prepared statements.....	15
1.4.6 De execute() methode.....	16
1.5 Transacties.....	17
1.6 Batch updates.....	20
1.7 Stored procedures.....	21
1.8 Wijzigbare resultsets.....	21
1.8.1 Een wijzigbare resultset creëren.....	21
1.8.2 Gegevens in een rij wijzigen.....	22
1.8.3 Rijen toevoegen en verwijderen.....	23
1.9 Foutafhandeling.....	23
1.9.1 SQLException.....	23
1.9.2 SQLWarning.....	24
1.10 Metadata.....	24
1.10.1 ResultSetMetaData.....	24
1.10.2 DatabaseMetaData.....	24
1.11 Grote objecten.....	25
1.11.1 Het lezen van grote objecten.....	25
1.11.2 Het schrijven van grote objecten.....	25
1.12 Data Access Objects (DAO).....	26



Hoofdstuk 1: JDBC

1.1 Inleiding

JDBC is de afkorting van *Java DataBase Connectivity*. Het is een technologie die communicatie met allerlei relationele databases mogelijk maakt. JDBC biedt aan een Java-applicatie immers een uniforme interface naar databases van verschillende producenten. Om een bestaande database toegankelijk te maken via JDBC moet een specifieke *driver* geïnstalleerd worden. Voor de meest gangbare databases zijn er momenteel JDBC-drivers beschikbaar.



JDBC is vervat in het pakket `java.sql` dat deel uitmaakt van de JSE. Meer geavanceerde mogelijkheden zijn ondergebracht in het pakket `javax.sql`.

Van JDBC zijn er verschillende versies die ondersteund worden in de volgende Java-versies:

JDK Versie	JDBC Versie
1.2	2.1
1.4	3.0
5	3.0
6	4.0
7	4.1
8	4.2

In de volgende paragrafen zullen we JDBC 4.2 behandelen.

1.2 De databaseserver

Databaseservers zijn er allerhande. Zo hebben we bekende producten als *Oracle* en



Microsoft SQL Server. Voor huis-, tuin- en keukengebruik op *Microsoft*-systemen wordt ook wel *Access* gebruikt.

In de *Open Source* wereld wordt onder andere gebruikgemaakt van *MySQL*. Deze databaseserver kan vrij gedownload worden van www.mysql.com. In het verdere verloop van deze cursus zullen we gebruikmaken van *MySQL* maar alle voorbeelden zijn ook toepasbaar op andere databaseservers.

Er kan gebruikgemaakt worden van volgende database die beschikbaar is op het internet:

- hostname: **noelvaes.eu**
- login: **student**
- wachtwoord: **student123**
- databank: **StudentDB**

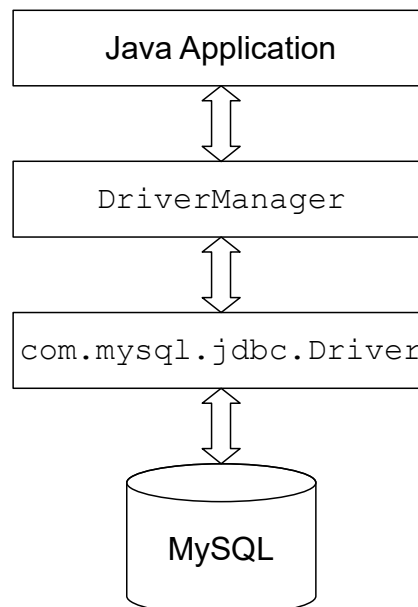
Om de tabellen te beheren of nieuwe aan te maken kan je eventueel gebruikmaken van ***MySQL Workbench***.

1.3 Een verbinding maken met een database

1.3.1 De database-*driver* laden

Vermits JDBC een universele interface is naar om het even welke database, verloopt de communicatie via een laag die volledig abstractie maakt van de onderliggende database. Hierbij speelt de klasse `java.sql.DriverManager` een cruciale rol. Het opzetten van de communicatie met de database verloopt via deze klasse, die alleen maar statische methoden heeft.

Specifieke database-*drivers*, zoals die voor *MySQL* moeten zich registreren bij de `DriverManager` klasse.



Deze registratie kan gebeuren met volgende methode:



```
DriverManager.registerDriver().
```

Van JDBC-*drivers* wordt echter verondersteld dat ze zichzelf registreren bij de *DriverManager* zodra ze in de JVM worden ingeladen. Deze registratie hoeft de gebruiker van de *driver* dus niet zelf te doen. Men moet er alleen voor zorgen dat de *driver* ingeladen wordt in de JVM, de rest gebeurt dan vanzelf.

Om de *driver* in te laden wordt er gebruikgemaakt van de methode `Class.forName()` die een klassenobject van een bepaalde klasse maakt. Het neveneffect van deze methode is uiteraard dat de klasse in de JVM geladen wordt, indien dat tenminste nog niet gebeurd was. Dit neveneffect wordt meestal gebruikt om JDBC-*drivers* te laden.

Bij het inladen van een klasse worden de `static` codeblokken uitgevoerd en de klasse van een JDBC-*driver* zal in zo'n codeblok zichzelf registreren bij de `DriverManager`.

Voor de **MySQL**-database is de *driver* een klasse met de naam `com.mysql.jdbc.Driver`. Deze klasse vinden we in het JAR-bestand **mysql-connector-java-x.y.z.jar** dat gedownload kan worden van de MySQL-site (onder de rubriek Connector/J).

We kunnen deze driver dan ook als volgt inladen:

```
Class.forName("com.mysql.jdbc.Driver");
```

De methode `Class.forName()` gooit een `ClassNotFoundException` indien de opgegeven klasse niet gevonden wordt in het **classpath**. We moeten het JAR-bestand met de *driver* daarom ook opnemen in ons *classpath*.

Elke database heeft zijn eigen *driver*-klasse die op deze wijze geladen moet worden. Voor de precieze naam van de *driver*-klasse moet men de bijgeleverde documentatie raadplegen.

Sinds JDK 6 en JDBC 4.0 is dit laden van de driver evenwel niet meer expliciet nodig. Men komt het wel nog vaak tegen in oude code die gemaakt is voor JDK 6. In sommige omstandigheden werkt dit automatisch laden evenwel niet en moeten we toch terugvallen op de klassieke techniek. Dit is onder andere het geval indien we een webapplicatie maken.

1.3.2 Een connectie maken

Zodra de database-*driver* geladen is, kunnen we een connectie maken met de database. Deze connectie wordt nadien gebruikt om te communiceren met de database.

Een connectie wordt gemaakt met volgende methode van de klasse `DriverManager`

```
DriverManager.getConnection(url, login, password)
```

De methode neemt als eerste parameter een string met de database URL. Deze URL heeft de volgende vorm:

```
jdbc:subprotocol:subname
```

Het *subprotocol* geeft aan welke driver de *DriverManager* moet gebruiken. Iedere geregistreerde driver heeft zijn eigen subprotocol. De *subname* is extra informatie die doorgegeven wordt aan de betreffende driver. Meestal bevat deze de naam van de *host* en de naam van de database.

De precieze URL hangt dus af van de database waarmee men communiceert. Voor de



database die we gecreëerd hebben is dat het volgende protocol:

```
jdbc:mysql://noelvaes.eu/StudentDB
```

De tweede parameter van de methode `getConnection()` is de gebruikersnaam en de derde parameter is het wachtwoord van de gebruiker.

De methode `getConnection()` geeft een object terug van de interface `Connection`. Dit object representeert de geopende connectie met de database.

We kunnen dus als volgt een connectie maken met de *MySQL* database:

```
Connection con = DriverManager.getConnection  
("jdbc:mysql://noelvaes.eu/StudentDB", "student", "student123");
```

De interface `Connection` is afgeleid van `AutoCloseable` zodat we hier gebruik kunnen maken van een *try with resources* (vanaf JDK 7). De connectie wordt dus automatisch afgesloten na het *try*-blok.

De code voor ons eerste programma ziet er als volgt uit.

```
package eu.noelvaes.jdbc;  
  
import java.sql.*;  
public class ConnectDB {  
    public static void main(String[] args) {  
        try (Connection con = DriverManager.getConnection(  
            "jdbc:mysql://noelvaes.eu/StudentDB", "student",  
            "student123")) {  
            System.out.println("Connection OK");  
        }  
        catch (Exception ex) {  
            System.out.println("Oops, something went wrong!");  
            ex.printStackTrace(System.err);  
        }  
    }  
}
```

Het laden van de driver laten we hier achterwege omdat dit sinds JDK 1.6 niet meer nodig is.

We maken onmiddellijk een connectie. Deze methode kan een *exception* genereren zodat we de nodige afhandeling daarvoor moeten voorzien.

Verder dienen we het pakket `java.sql` te importeren. Om het programma uit te voeren moeten we tevens de pakketten van **MySQL** in ons *classpath* opnemen.

```
java -cp mysql-connector-java-x.y.z.jar;. eu.noelvaes.jdbc.ConnectDB
```

Opdracht 1: Een connectie maken met de database

In deze opdracht maken we een programma dat de *JDBC-driver* laadt en een connectie maakt met de database.

- Maak een nieuw Java-project in je IDE.
- Voeg het JAR-bestand van de *MySQL-driver* aan je project toe. Indien je gebruikt maakt van *Maven* kan je volgende *dependency* toevoegen aan de POM:



```

...
<dependencies>
  <dependency>
    <groupId>mysql</groupId>
    <artifactId>mysql-connector-java</artifactId>
    <version>5.1.41</version>
  </dependency>
</dependencies>
...

```

- Open de documentatie van het pakket `java.sql` en zoek de beschrijving van de klasse `DriverManager` en de interface `Connection`.
- Maak een nieuw bestand met de naam **ConnectDB.Java** en geef bovenstaande code in.
- Compileer het programma en voer het uit.

1.4 SQL-commando's gebruiken

1.4.1 Een statement creëren

SQL (*Standard Query Language*) is de meest gebruikte taal om te communiceren met een database. JDBC biedt de mogelijkheid SQL-commando's naar de database te sturen. Hiervoor gebruiken we een object van de interface `Statement` dat een SQL-commando bevat. Zo'n object wordt gemaakt met de methode `createStatement()` van de interface `Connection`.

Concreet ziet dat er als volgt uit:

```

try (Connection con = DriverManager.getConnection(
    "jdbc:mysql://noelvaes.eu/StudentDB", "student", "student123");
    Statement stmt = con.createStatement()) {
    ...
}

```

Het statement dient na gebruik afgesloten te worden met de methode `close()`. Vermits de interface `AutoCloseable` geïmplementeerd wordt, kunnen we ook hier gebruikmaken van de *try with resources*.

Het *statement*-object kan op verschillende manieren SQL-commando's naar de database sturen. In de volgende tabel worden enkele methoden van de interface `Statement` opgesomd. Voor een volledige en precieze omschrijving verwijzen we naar de Java-documentatie.

Method	Omschrijving
<code>executeUpdate(String sql)</code>	Voert een CREATE, INSERT, UPDATE of DELETE commando uit.
<code>executeQuery(String sql)</code>	Voert een commando uit dat als resultaat een tabel heeft (<i>resultset</i>). Dit is typisch het geval bij een SELECT-commando.
<code>executeBatch()</code>	Voert een reeks van UPDATE-commando's uit. De commando's worden toegevoegd met de methode <code>addBatch()</code> .
<code>execute()</code>	Voert om het even welk SQL-commando uit.



De methoden `executeUpdate()` en `executeQuery()` nemen als parameter een tekenreeks die het SQL-commando bevat.

De methode `executeUpdate()` geeft een integer als resultaat terug die het aantal rijen bevat waarin een aanpassing is gebeurd.

De methode `executeBatch()` geeft een reeks van integers terug, waarbij elk element het resultaat is van de overeenkomstige opdracht in de *batch*.

De methode `executeQuery()` geeft als resultaat een `ResultSet` terug die de gegevens bevat van de zoekopdracht.

De methode `execute()` geeft een `boolean` terug die aangeeft of het resultaat al dan niet een `ResultSet` is.

In het verder verloop van de cursus komen deze verschillende methoden één voor één aan bod.

1.4.2 Gegevens uit een database opvragen

Gegevens worden uit een database opgevraagd met het SQL-commando `SELECT`:
Om alle gegevens op te vragen uit de tabel **Beers** gebruiken we het volgende commando:

```
SELECT * FROM Beers
```

Om het commando uit te voeren, gebruiken we de methode `executeQuery()`. Deze methode geeft een object van de interface `ResultSet` terug.

```
String sql = "SELECT * FROM Beers";
try (Connection con = DriverManager.getConnection(
    "jdbc:mysql://noelvaes.eu/StudentDB", "student", "student123");
    Statement stmt = con.createStatement();
    ResultSet rs = stmt.executeQuery(sql)) {
    ...
}
```

Ook hier maken we gebruik van de *try with resources* om de *resultset* na gebruik correct af te sluiten.

Een `ResultSet` is een soort tabel waarvan we de waarden kunnen opvragen. Om een bepaalde waarde uit deze tabel te lezen, moeten we eerst de juiste rij selecteren. Aanvankelijk staat de *cursor* op de positie voor de eerste rij in de tabel. Met volgende methoden kan de positie van de cursor in de tabel gewijzigd worden:

Method	Omschrijving
<code>next()</code>	Zet de cursor één positie verder.
<code>first()</code>	Zet de cursor op de eerste rij.
<code>last()</code>	Zet de cursor op de laatste rij.
<code>previous()</code>	Zet de cursor één positie terug.
<code>afterLast()</code>	Zet de cursor net na de laatste rij.
<code>beforeFirst()</code>	Zet de cursor net voor de eerste rij.
<code>absolute()</code>	Zet de cursor op een bepaalde rij.



<code>relative()</code>	Zet de cursor een aantal rijen vooruit of achteruit ten opzichte van de huidige positie.
-------------------------	--

Deze methoden geven `true` terug als ze op een geldige rij belanden, anders geven ze `false` terug. Op die manier kunnen deze methoden gemakkelijk in een `while()` lus verwerkt worden. Zo kan men alle rijen in een *resultset* op volgende manier afgaan:

```
while(rs.next()){
    // Do something with the current row
}
```

De *resultset* die we tot nu toe bekomen hebben, kan alleen gebruikmaken van de methode `next()`. Dit is de standaard-*resultset* voor JDBC 1.x-drivers. Om ook gebruik te kunnen maken van de andere positioneringsmethoden, moeten we een *scrollable resultset* maken. Deze wordt alleen ondersteund door JDBC-drivers van versie 2.x en hoger.

Bij de creatie van het *statement* wordt aangegeven dat we zo een *scrollable resultset* willen gebruiken. De methode `createStatement()` heeft een variant die twee parameters heeft. Met deze parameters wordt aangegeven welk soort *resultset* er gemaakt wordt.

De eerste parameter geeft het type van *resultset* aan en de tweede parameter geeft aan of de *resultset* *read-only* of *updatable* is.

Voor deze parameters gebuikt men constanten van de klasse `ResultSet`:

Constante	Omschrijving
<code>TYPE_FORWARD_ONLY</code>	De <i>resultset</i> kan alleen voorwaarts doorbladerd worden met de methode <code>next()</code> . Dit is de standaardwaarde voor JDBC 1.x-drivers.
<code>TYPE_SCROLL_INSENSITIVE</code>	De <i>resultset</i> kan in alle richtingen doorbladerd worden maar de data worden niet telkens met de database gesynchroniseerd.
<code>TYPE_SCROLL_SENSITIVE</code>	De <i>resultset</i> kan in alle richtingen doorbladerd worden en de data worden telkens met de database gesynchroniseerd.
<code>CONCUR_READ_ONLY</code>	De <i>resultset</i> kan enkel gelezen worden. Dit is de standaardwaarde voor JDBC 1.x-drivers.
<code>CONCUR_UPDATABLE</code>	De <i>resultset</i> kan ook gewijzigd worden.

Indien we een *statement* maken zonder parameters is deze standaard van het type `TYPE_FORWARD_ONLY` en `CONCUR_READ_ONLY`.

We maken een *scrollable resultset* op volgende wijze:

```
String cmd = "SELECT * FROM Beers";
try (Connection con = DriverManager.getConnection(
    "jdbc:mysql://noelvaes.eu/StudentDB", "student", "student123");
    Statement stmt= con.createStatement(
        (ResultSet.TYPE_SCROLL_INSENSITIVE, ResultSet.CONCUR_READ_ONLY);
    ResultSet rs = stmt.executeQuery(cmd)) {
    ...
}
```