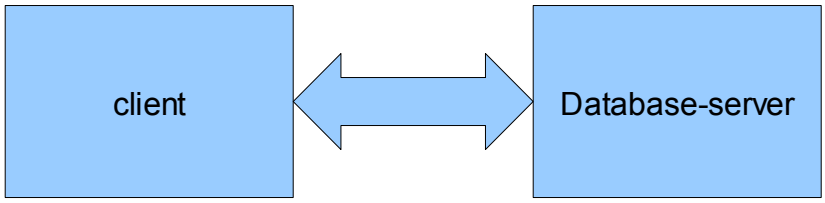
Practicum Orientatie op Webserver

Doel:

- ✔ Een goed beeld krijgen van een drie-tier applicatie.
- ✔ Het gebruik van drivers om database-onafhankelijk te werken.
- ✔ Het kunnen lezen van een java servlet.

Database in two-tier omgeving

In de volgende tekening staan twee verschillende tiers: een client en een database-server. Elke tier kan in principe op een andere computer draaien. Maar het is ook goed mogelijk ze alebeide op dezelfde computer te plaatsen. De client zal een connectie opzetten met een database-server en via een databaseprotocol SQL-opdrachten naar de database-server sturen en de gevonden gegevens daarvan weer terugkrijgen.

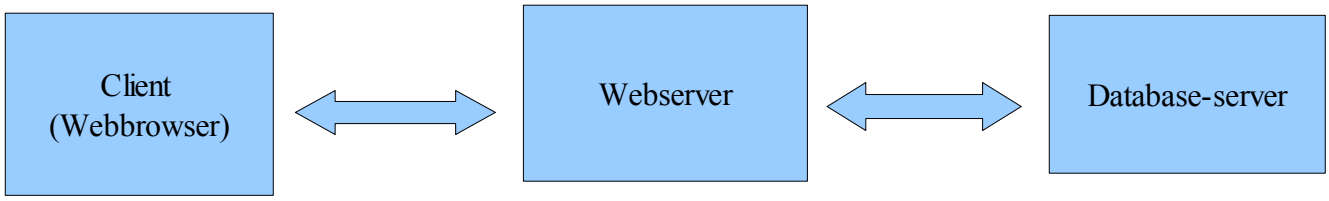


In dit geval met twee tiers is de client in direct contact met de database-server. De client zal drivers moeten laden om met de database te kunnen 'praten'. Voorbeelden van deze situatie zijn:

- ✔ het gebruik van een formulier gemaakt in Base van OpenOffice. In OpenOffice (en ook in Microsoft Office) is het mogelijk om formulieren te maken die invoer en uitvoer van gegevens naar een database mogelijk maken. Voordat het formulier wordt gerund moet er via een configuratie-file worden opgegeven welke soort database wordt gebruikt zodat de driver voor die database kan worden geladen en waar de database zijn diensten aanbiedt (bijvoorbeeld: pluto.sict.hanze.nl:1521).
- ✔ een zelfgeschreven java-programmaatje. Natuurlijk is het mogelijk programma's te schrijven die direct contact opnemen met een database. Als het programa geschreven is in java en de database is Oracle, dan zal in dat programma hetvolgende statement zijn opgenomen (of een utility class worden gebruikt waar dat statement in voorkomt): Class.forName ("oracle.jdbc.driver.OracleDriver"); waardoor de driver wordt geladen. Deze driver staat in de bibliotheek classes12.jar.
- ✔ een SQL client. In een practicum of als database-beheerder zullen we vaak gebruik maken van een tool waarmee direct SQL naar een database-server gestuurd kan worden. Bij Oracle is het standaard programma SQL-plus. Een iets gebruikersvriendelijker tool is SQLTools. In beide moet worden opgegeven waar de database zijn diensten aanbiedt en moeten bij de configuratie de benodigde bibliotheken goed worden geinstalleerd om met de database te kunnen communiceren.

Database in three-tier omgeving

In de volgende tekening staan drie verschillende tiers: een client (meestal een webbrowser) die via het HTTP protocol communiceert met de webbrowser. Elke tier kan in principe weer op een andere computer draaien.
De webbrowser stuurt over dat HTTP protocol opdrachten zoals GET, HEAD en PUT. Voor een korte beschrijving van het HTTP-protocol zie http://nl.wikipedia.org/wiki/HTTP. De webbrowser stuurt naar de browser files terug bijvoorbeeld html-teksten , plaatjes en geluiden. De webserver kan op verschillende manieren aan deze files komen. Het kan zijn dat de gegevens in het filesysteem op de eigen computer staan. Het kan ook zijn dat de gegevens opgehaald worden van een database. In het laatste geval zal de webserver een connectie opzetten met een database-server en via een databaseprotocol SQL-opdrachten naar de databaseserver sturen en de gevonden gegevens daarvan weer terugkrijgen.



In deze situatie is de web-server de client van de database-server. Op de webserver staan programma's die contact zoeken met de database-server. Deze programma's zullen weer gebruik maken van drivers om met de database te communiceren. Als de programma's in java zijn geschreven (java servlets), dan zal op de webserver dus ook de bibliotheek classes12.jar weer geinstalleerd moeten worden.
Verder zal er voor gezorgd moeten worden dat een opdracht van een client vertaald wordt naar de aanroep van een programma. In het geval van een J2EE webserver is de belangrijkste file om dit mogelijk te maken is de file 'web.xml'. Een webapplicatie bestaat meestal uit meerdere html-pagina's en uit meerdere servlets. Bij zo'n webapplicatie hoort één file web.xml. In die file staat de vertaalslag van een url naar de java-class (de servlet is een java-class) die bij die url hoort. Als via de webbrowser de url wordt opgevraagd, dan zal de java-class worden uitgevoerd, bijvoorbeeld om alle docenten die in de docenten-tabel op de database-server staan te produceren. De servlet heeft ook als opdracht om de benodigde html-tags toe te voegen en de gemaakte file terug te sturen naar de browser.

Html-forms

Volg de tutorial van www.w3schools.com over Html, onderdeel Html-forms. Het gaat er vooral om dat begrepen is dat via onderstaand formulier een HTTP-command POST naar de “pagina” (lees servlet) guestbook wordt gestuurd, samen met parameters name en address. De waarden van de parameters worden meegestuurd. Deze waarden wordn door de gebruiker in het formulier ingevuld voordat hij op de submit-knop drukt.

<html>

```
<body>

<form method="post" action="guestbook">
<input type="text" name="name" size="20"/>
<input type="text" name="address" size="20"/>
<input type="submit" value="Submit"/>
</form>
<p>
If you click the "Submit" button, you will send your input to a new page called “guestbook”.
</p>

</body>
</html>
```

Servlet code

De volgende servlet geeft de mogelijkheid om naam en adres van een persoon aan de tabel Guestbook toe te voegen en laat vervolgens alle huidige personen uit Guestbook zien. Hiervoor wordt gebruik gemaakt van SQL. Het invoegen gebeurt met:

insert into guestbook values (name,address);
En het ophalen van alle personen in Guestbook gebeurt met:
select * from guestbook;

Een servlet heeft twee belangrijke methodes:

doGet(HttpServletRequest request, HttpServletResponse response) en

doPost(HttpServletRequest request, HttpServletResponse response).

In deze methodes beschrijft de programmeur wat er moet gebeuren als er een HTTP GET of een HTTP POST commando binnenkomt. Als de servlet met hetzelfde effect zowel via een GET als via een POST aangeroepen kan worden, kan volstaan worden met het vullen van één van beide methoden. De andere methode (in dit voorbeeld doPost) roept dan gewoon de eerste aan.

De servlet kan uit het object request alle informatie halen die door de browser naar de webserver is gestuurd. Het eerste dat er in de doGet() methode gebeurt is het ophalen van de parameters die de gebruiker met de POST heeft meegestuurd (name en address).

Daarna begint de servlet met het terugsturen van gegevens naar de browser. Het response-object heeft een printer en naar deze printer worden gegevens weggeschreven (te beginnen met <HTML> <HEAD><TITLE>Guestbook</TITLE></HEAD>).

Hierna wordt begonnen met het wegschrijven van de gelezen name en address naar de database. De driver wordt geladen. De plaats van de database, de username en het password worden uit de file usr/inloggen.txt gelezen en er wordt een connectie opgezet met de database. In de file usr/inloggen.txt staan een drietal regels. Met readLine worden deze regels gelezen. De inhoud van de regels is bijvoorbeeld:

jdbc:oracle:thin:@pluto.sict.hanze.nl:1521:ondw

DT_50

DT_50

Hierna worden ge vereiste SQL-statements gevormd en uitgevoerd. Het resultaat van het select-statement wordt regel voor regel gelezen en naar de printer van het response-object gestuurd en tenslotte wordt </BODY></HTML> naar de printer gestuurd. Op deze manier heeft de servlet een volledige html-file aangemaakt.

```
1.  import java.sql.*;
2.  import java.io.*;
3.  import javax.servlet.ServletException;
4.  import javax.servlet.ServletOutputStream;
5.  import javax.servlet.http.HttpServletRequest;
6.  import javax.servlet.http.HttpServletResponse;
7.
8.  import java.util.*;
9.
10. //import javax.servlet.*;
11. //import javax.servlet.http.*;
12.
13. /**
14.  * Servlet implementation class for Servlet: guestbook
15.  *
16.  */
17.
18. public class guestbook extends javax.servlet.http.HttpServlet implements javax.servlet.Servlet
    {
19.    /* (non-Java-doc)
20.     * @see javax.servlet.http.HttpServlet#HttpServlet()
21.     */
22.          public guestbook() {
23.                  super();
24.          }
25.
26.
27.   /* (non-Java-doc)
28.    * @see javax.servlet.http.HttpServlet#doGet(HttpServletRequest request,
    HttpServletResponse response)
29.    */
30.          protected void doGet(HttpServletRequest request, HttpServletResponse response)
31.          throws ServletException, IOException {
32.                  //TODO Auto-generated method stub
33.
34.                  String name    = request.getParameter("name");
35.                  String address = request.getParameter("address");
```

```
36.
37.                     response.setContentType("text/html");
38.
39.                     ServletOutputStream out = response.getOutputStream();
40.
41.                     out.println("<HTML><HEAD><TITLE>Guestbook</TITLE></HEAD>");
42.                     out.println("<BODY><H2>guestbook</H2>");
43.                     out.println("<P>Java Servlet using JDBC. The servlet will include" +
44.                                     "name and address in guestbook and return all current persons in
    guestbook</P>");
45.
46.                     //Load the ORACLE JDBC driver. Put file classes12.zip in directory lib
47.                     try {
48.                        Class.forName ("oracle.jdbc.driver.OracleDriver");
49.                     } catch (Exception e) {
50.                        out.println("write to log: Cannot find Oracle driver (file: classes12.zip
    missing?)");
51.                     }
52.
53.                     //Read connectionstring from file
54.                     String database="";
55.                     String account="";
56.                     String password="";
57.                     try {
58.                        BufferedReader in = new BufferedReader( new
    InputStreamReader(getServletContext().getResourceAsStream("usr/inloggen.txt")));
59.                        database=in.readLine();
60.                        account=in.readLine();
61.                        password=in.readLine();
62.                     } catch (Exception e) {
63.                        out.println("guestbook: cannot read inloggen.txt");
64.                     }
65.
66.        // Connect to the database
67.                     try {
68.                        Connection conn = DriverManager.getConnection(database, account,
    password);
69.                        Statement stmt = conn.createStatement ();
70.                        ResultSet rset=null;
71.           //insert form values into guestbook
72.           try {
73.              rset = stmt.executeQuery("insert into guestbook values ('"+
74.                                          name+"', '"+address+"')");
75.              } catch (SQLException e) {
76.                    out.println("writetolog: Error in insert "+e+"<BR>");
77.                    }
78.
79.
80.              //Show current contents of guestbook
```

```
81.             rset = stmt.executeQuery ("select * from guestbook");
82.
83.             out.println("<TABLE BORDER CELLSPACING=3 CELLPADDING=3>");
84.
85.             while (rset.next ()) {
86.                 out.println("<TR>");
87.                 for (int i=1;i<=2;i++) {
88.                     out.println("<TD>"+rset.getString (i));
89.                 }
90.             }
91.         } catch (SQLException e) {
92.                 out.println("writetolog: SQLException "+e+"<BR>");
93.         }
94.         out.println("</TABLE>");
95.         out.println("</BODY></HTML>");
96.   }
97.
98.
99.         /* (non-Java-doc)
100.         * @see javax.servlet.http.HttpServlet#doPost(HttpServletRequest request,
    HttpServletResponse response)
101.         */
102.        protected void doPost(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
103.                 // TODO Auto-generated method stub
104.                 doGet(request, response);
105.         }
106.}
```

Bovenstaand servlet-programma is niet in een schitterende stijl geschreven. In een betere programmeerstijl zou in elk geval het opzetten van de connectie met de database in een aparte class worden geplaatst. In de gebruikte stijl moet elke servlet zijn eigen code hebben waarin de connectie wordt opgezet, wat ongewenst is.

Waar plaatsen we bedrijfsregels

In het practicum wordt gevraagd om een aantal bedrijfsregels te implementeren. De gevraagde regels staan in de case Shoestring. In de bedrijfsregels wordt tot uitdrukking gebracht dat de informatie van een bedrijf aan een aantal beperkingsregels moet voldoen. Een regel kan bijvoorbeeld zijn: 'Een factuur moet minimaal één factuurregel hebben, wil er van een factuur sprake zijn'.

Uit het voorbeeld van de servlet zal duidelijk zijn dat een servlet heel goed de code zou kunnen bevatten die binnenkomende parameters gaat valideren. In principe zouden alle bedrijfsregels op de middelste tier (de web-server) kunnen worden gevalideerd.
In het practicum wordt echter gevraagd om alle regels in de database te implementeren. Hiervoor zullen

we de taal PL/SQL leren. Dit is de traditionele aanpak. In een two-tier omgeving is de database-server ook de logische plaats om de bedrijfsregels te implementeren. In een three-tier omgeving is de keuze veel minder voor de hand liggend.

Voordeel web-tier
    database-onafhankelijkheid
Voordeel database-tier
    Betere responsietijden mogelijk

Het deployen van de webapplicatie

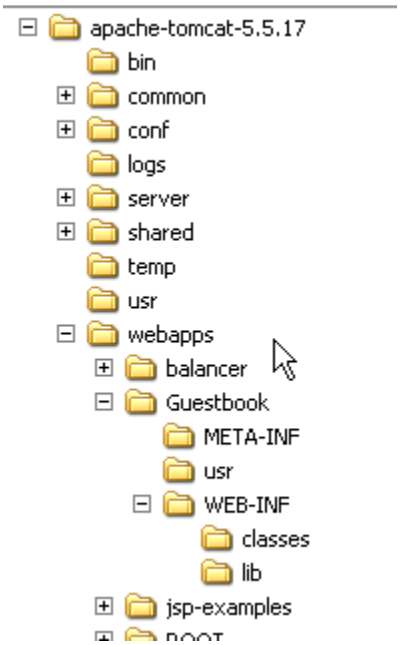
Een webapplicatie bestaat een (groot) aantal files. Er zijn statische html-pagina's, plaatjes, geluiden en dynamische html-pagina's (bijvoorbeeld bovenstaande java-servlet). Daarnaast is er in ons voorbeeld een file waarin de database-inloggegevens staan. Ook is de eerder genoemde file web.xml noodzakelijk. De files moeten allemaal in voorgedefinieerde directories staan. Gelukkig zijn er tools die het organiseren van webapplicaties ondersteunen.
De inhoud van de xml-file web.xml in het voorbeeld staat hieronder. Als de gebruiker een url /guestbook opvraagt zal in feite de servlet met de naam guestbook worden uitgevoerd (regel 11 en 12). De servlet met naam guestbook heeft ook de classnaam guestbook (regel 7 en 8).

```
1.  <?xml version="1.0" encoding="UTF-8"?>
2.  <web-app id="WebApp_ID" version="2.4" xmlns="http://java.sun.com/xml/ns/j2ee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee http://java.sun.com/xml/ns/j2ee/web-
    app_2_4.xsd">
3.      <display-name>Guestbook</display-name>
4.      <servlet>
5.          <description></description>
6.          <display-name>guestbook</display-name>
7.          <servlet-name>guestbook</servlet-name>
8.          <servlet-class>guestbook</servlet-class>
9.      </servlet>
10.     <servlet-mapping>
11.         <servlet-name>guestbook</servlet-name>
12.         <url-pattern>/guestbook</url-pattern>
13.     </servlet-mapping>
14.     <welcome-file-list>
15.         <welcome-file>index.html</welcome-file>
16.     </welcome-file-list>
17. </web-app>
```

Een mogelijkheid om een webapplicatie te deployen bestaat uit het aanmaken van een war-file (een soort zip-file) waarin alle onderdelen van de applicatie op de goede manier staan opgeslagen. Van de voorbeeld servlet en het html-formulier om de servlet aan te roepen is een war-file gemaakt, die ook op deze site is te downloaden. De file is getest op een webserver Tomcat5.5. Plaats de war-file in de directory apache-tomcat-5.5.17\webapps en start de tomcat server. De server zal de war-file zelf gaan

uitpakken. Er wordt een subdirectory Guestbook in apache-tomcat-5.5.17\webapps aangemaakt, met daarbinnen alle benodigde files en subdirectories. Door in de webbrowser het adres http://localhost:8080/Guestbook aan te vragen zal het html-formulier index.html naar de browser worden gestuurd. Door een naam en adres in te vullen kan een nieuwe gast aan de guestbook worden toegevoegd. Natuurlijk moet wel in Oracle een tabel Guestbook met de velden name en address aanwezig zijn en moet de inhoud van de file usr/inloggen.txt worden aangepast zodat de juiste database en gebruiker wordt gevonden.

De folderstructuur voor de applicatie Guestbook is in de volgende figuur weergegeven. De file web.xml moet in de directory WEB-INF staan.



Jdbc

In de servletcode is gebruik gemaakt van Jdbc API (programmeer interface). De API werkt database-onafhankelijk (te gebruiken voor alle types databases waarvoor een jdbc-driver aanwezig is). De JDBC API maakt onderdeel uit van zowel J2SE (Java2 platform Standard edition) als van J2EE (enterprise edition). De API maakt het mogelijk om een drietal zaken te doen:

- ✔ Een connectie opzetten met een database
- ✔ SQL opdrachten verzenden
- ✔ De resultaten van de opdrachten verder verwerken

Van alle drie is een voorbeeld in de servlet-code te vinden.
Op http://java.sun.com/developer/onlineTraining/Database/JDBCShortCourse/jdbc/jdbc.html is een inleidende tekst te vinden over Jdbc. Behalve de zaken die in de servlet al te zien waren is er hier ook aandacht voor jdbc en:

- ✔ de jdbc:odbc bridge om gebruik te maken van odbc-drivers in plaats van native jdbc drivers
- ✔ prepared statements

- ✔ het updaten van een rij in de database
- ✔ het aanroepen van een stored procedure met CallableStatement
- ✔ transacties

Vooral studenten die de I-richting hebben gekozen wordt aangeraden om deze leerstof ook door te nemen.

Nog eens gelaagdheid

In de voorbeeld servlet hierboven gebeuren te veel zaken. Het is beter de verschillende taken van de servlet te verdelen over verschillende classes. Aan de ene kant gebeuren in de servlet zaken die direct met de database te maken hebben (het opzetten van de connectie, het maken en uitvoeren van de leesopdracht naar de database tabel) en aan de andere kant gebeuren er zaken die meer te maken hebben met de opmaak (het toevoegen van de html-tags). Een eerste verbetering van de aanpak zou dan kunnen zijn om het geheel op te delen in twee lagen: een aantal classes te definieren die het contact met de database verzorgen en aan de andere kant een aantal classes te definieren die behulpzaam zijn met de opmaak.

Voor de laag die het contact met de database verzorgt zou als extra taak kunnen krijgen om ook de bedrijfsregels te implementeren, die nu in PL/SQL in de database laag liggen opgeslagen. Als voorbeeld zou men voor de volgende classes kunnen kiezen:

- een class DbManager: taak opzetten en beheren van connecties naar database. Heeft o.a. Een methode getConnection().
- per tabel een aantal classes: een class om de data die in een rij van de tabel staat op te slaan (class guestbookBean), een class met allerlei methodes om de objecten van de vorige class op te halen en op te slaan (class guestbookManager) en een class die aangeeft wat gebeuren moet bij het inserten-deleten-updaten van een tabel (een class die een interface guestbookListener moet implementeren).

Een methode van guestbookManager is bijvoorbeeld

1. public GuestbookBean loadByPrimaryKey(Integer guestbookId) throws SQLException { ... },

waardoor uit de database de GuestbookBean van de opgegeven waarde van de primaire sleutel wordt opgehaald.

En de code voor de interface guestbookListener ziet er bijvoorbeeld als volgt uit:

```
1. import java.sql.SQLException;
2.
3. /**
4.  * Listener that is notified of GUESTBOOK table changes.
5.  */
6.
7. public interface GuestbookListener
```

```
8.  // extends+
9.
10. // extends-
11. {
12.     /**
13.      * Invoked just before inserting a GuestbookBean record into the database.
14.      *
15.      * @param pObject the GuestbookBean that is about to be inserted
16.      */
17.     public void beforeInsert(GuestbookBean pObject) throws SQLException;
18.
19.
20.     /**
21.      * Invoked just after a GuestbookBean record is inserted in the database.
22.      *
23.      * @param pObject the GuestbookBean that was just inserted
24.      */
25.     public void afterInsert(GuestbookBean pObject) throws SQLException;
26.
27.
28.     /**
29.      * Invoked just before updating a GuestbookBean record in the database.
30.      *
31.      * @param pObject the GuestbookBean that is about to be updated
32.      */
33.     public void beforeUpdate(GuestbookBean pObject) throws SQLException;
34.
35.
36.     /**
37.      * Invoked just after updating a GuestbookBean record in the database.
38.      *
39.      * @param pObject the GuestbookBean that was just updated
40.      */
41.     public void afterUpdate(GuestbookBean pObject) throws SQLException;
42.
43.
44. }
45.
```

De bedoeling van deze code zal voor zich zelf spreken. De implementatie van deze interface zal dan de juiste code bij alle functies van de listener moeten hebben om de bedrijfsregels te implementeren.

Een uitgebreide applicatie zal dan bestaan uit een aantal servlets (en jsp-pagina's) die gebruik maken van de classes uit deze data-laag.