Kapitel 6: SQL-Einbettung in Programmiersprachen

Kopplungsvarianten zwischen Programmiersprachen und Datenbanksprachen:

1) Erweiterung der Programmiersprache um Datenbankkonstrukte
   → Persistente Programmiersprachen, Datenbankprogrammiersprachen
   (z.B. Pascal/R, DBPL, Persistentes C++)
2) Erweiterung der Datenbanksprache um Programmierkonstrukte
   → "4th Generation Languages" (4GLs)
3) Einbettung der Datenbanksprache in die Programmiersprache (oder Web-Skriptsprache)
   → "Embedded SQL" (ESQL)

Beispiel für eine Datenbankprogrammiersprache (à la DBPL im Pascal- bzw. Modula-Stil):

```
TYPE    Produktrecordtyp        = RECORD
                                    PNR: INTEGER;
                                    ...
                                  END;
        Produkterelationentyp   = RELATION OF Produktrecordtyp;

VAR     Produkte: PERSISTENT Produkterelationentyp;
        Ergebnis: Produkterelationentyp;
        x: Produktrecordtyp;

Ergebnis :=  SELECT *
             FROM Produkte
             WHERE ...;

FOR EACH x IN Ergebnis
DO
   ...
END;
```

6.1 Eingebettetes SQL (Embedded SQL)

Kernproblem bei der SQL-Einbettung in konventionelle Programmiersprachen:

Abbildung von Tupelmengen auf die Datentypen der Wirtsprogrammiersprache

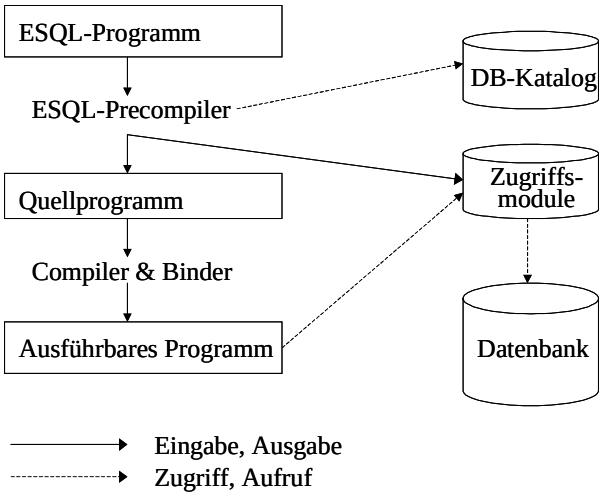
Realisierte Lösung:

Abbildung von Tupeln bzw. Attributen auf die Datentypen der Programmiersprache
→ Wirtsprogrammvariable (engl.: Host Variables)

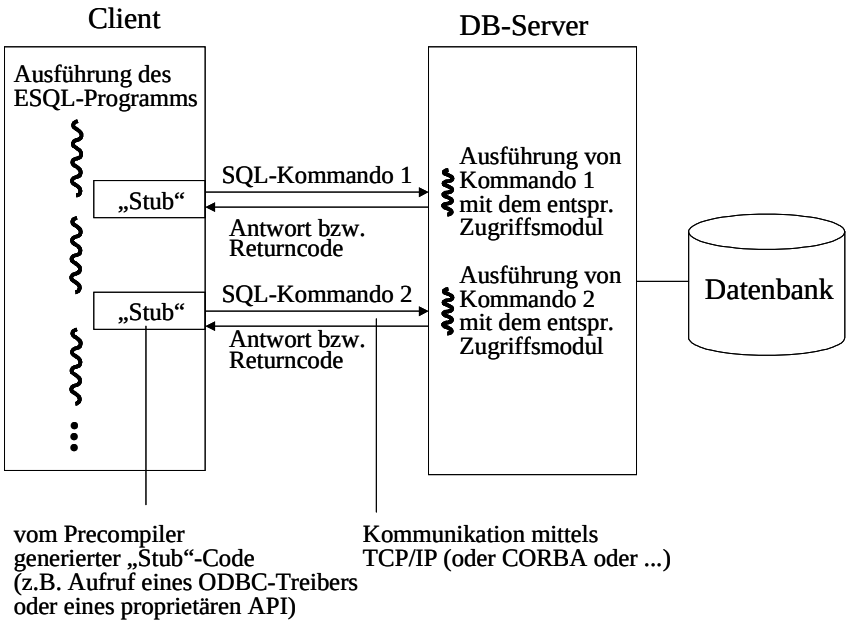
+

Iteratoren (Schleifen) zur Verarbeitung von Tupelmengen
→ Cursor-Konzept

Standardarchitektur:



Laufzeitarchitektur für ESQL-Programme in einem Client-Server-System:



Wirtsprogrammvariable (Host Variables)

Die Attribute eines Resultattupels einer SQL-Anfrage werden an speziell deklarierte Variable des Wirtsprogramms zugewiesen (INTO-Klausel).
Beispiel:

```
EXEC SQL    SELECT PNr, Menge, Status INTO :pnr, :menge, :status
            FROM Bestellungen WHERE BestNr = 555;
```

Analog können SQL-Anweisungen mittels solcher Variable mit Eingabeparametern versorgt werden.
Beispiele:

```
1) EXEC SQL    SELECT PNr, Menge, Status INTO :pnr, :menge, :status
               FROM Bestellungen WHERE BestNr = :bestnr;
2) EXEC SQL    INSERT INTO Bestellungen (BestNr, Monat, Tag, KNr, PNr, Menge)
               VALUES (:bestnr, :monat, :tag, :knr, :pnr, :menge);
```

Wirtsprogrammvariable dienen als "Übergabepuffer" zwischen DBS und Programm. Generell werden dabei die SQL-Datentypen auf die Datentypen der jeweiligen Wirtsprogrammiersprache abgebildet.

Indikatorvariable zum Erkennen von Nullwerten

Beispiel:

```
EXEC SQL BEGIN DECLARE SECTION;
   int      pnr;
   int      vorrat;
   short    vorrat_ind;
EXEC SQL END DECLARE SECTION;
...
EXEC SQL    SELECT Vorrat INTO :vorrat:vorrat_ind
            FROM Produkte WHERE PNr = :pnr;
if (vorrat_ind = = 0)
   { /* kein Nullwert */ ... }
else
   { /* Nullwert */ ... };
```

Allgemein können Indikatorvariable folgende Werte haben:

= 0	die entsprechende Wirtsprogrammvariable hat einen regulären Wert
= -1	die entsprechende Wirtsprogrammvariable hat einen Nullwert
> 0	die entsprechende Wirtsprogrammvariable enthält eine abgeschnittene Zeichenkette

Beispiel eines ESQL-Programms (lieferung.pc)

```
/*** Programm zur Erfassung von Lieferungen ***/

#include <stdio.h>
#include <string.h>

EXEC SQL INCLUDE SQLCA; /* Importieren der SQL Communication Area */
/*  Innerhalb der SQL-Anweisungen ist Gross-/Kleinschreibung irrelevant.
    Sie dient hier nur zur Hervorhebung der SQL-Anweisungen          */

/* Der Precompiler erzeugt an dieser Stelle die folgende Datenstruktur:
    struct sqlca {
        char    sqlcaid[8];
        long    sqlabc;
        long    sqlcode;
        struct  {
                unsigned short  sqlerrml;
                char            sqlerrmc[70];
                } sqlerrm;
        char    sqlerrp[8];
        long    sqlerrd[6];
        char    sqlwarn[8];
        char    sqlext[8];
        };
    struct sqlca sqlca;
*/

main ()
{
/* Deklaration der Wirtsprogrammvariablen */
EXEC SQL BEGIN DECLARE SECTION;
    /*  Die hier deklarierten Wirtsprogrammvariablen unterliegen bei ihrer
        Verwendung in SQL-Ausdrücken einer Typprüfung durch den Precompiler. */
    int bestnr;
    int pnr;
    int menge;
    VARCHAR status[10];

    /* Der Precompiler erzeugt aus einer solchen Deklaration die folgende Struktur:
        struct {
                unsigned short len;
                unsigned char arr[10];
                } status;
    */

    VARCHAR user[20];
    VARCHAR passwd[10];
EXEC SQL END DECLARE SECTION;

/* Globale Fehlerbehandlung */
EXEC SQL WHENEVER SQLERROR STOP;
```

```
/* Verbindungsaufbau mit dem DBS */
printf ("Benutzername?"); scanf ("%s", &(user.arr));
user.len = strlen(user.arr); /* Konvertierung von C-String in VARCHAR */
printf ("Kennwort?"); scanf ("%s", &(passwd.arr));
passwd.len = strlen(passwd.arr); /* Konvertierung von C-String in VARCHAR */
EXEC SQL CONNECT :user IDENTIFIED BY :passwd;
/* gleichzeitig Beginn einer Transaktion */

/* Dialogschleife */
printf ("Bitte geben Sie eine Bestellnummer ein. (0 = Programmende)\n");
scanf ("%d", &bestnr);

while (bestnr != 0)
   {
   EXEC SQL    SELECT PNr, Menge, Status
               INTO :pnr, :menge, :status
               FROM Bestellungen
               WHERE BestNr = :bestnr;
   if (sqlca.sqlcode = = 0) /* Testen des SQL-Returncodes */
      {
      status.arr[status.len] = '\0'; /* Konvertierung von VARCHAR in C-String */
      if (strcmp (status.arr, "neu") = = 0)
         {
         EXEC SQL    UPDATE Produkte
                     SET Vorrat = Vorrat - :menge
                     WHERE PNr = :pnr AND Vorrat >= :menge;
         if (sqlca.sqlcode = = 0) /* Testen des SQL-Returncodes */
            {
            strcpy (status.arr, "geliefert");
            status.len = strlen (status.arr);
            EXEC SQL    UPDATE Bestellungen
                        SET Status = :status
                        WHERE BestNr = :bestnr;
            }
         else
            printf ("\n *** Ungenuegender Lagervorrat ***\n");
         }
      else
         printf ("\n *** Lieferung bereits erfolgt ***\n");
      }
   else
      printf ("\n *** Bestellung nicht gefunden ***\n");
   EXEC SQL COMMIT WORK;  /* Ende Transaktion und Beginn neue Transaktion */
   printf ("Bitte geben Sie eine Bestellnummer ein. (0 = Programmende)\n");
   scanf ("%d", &bestnr);
   }; /*while*/

EXEC SQL DISCONNECT;    /* Verbindung mit dem DBS abbauen */
exit (0);
} /*main*/
```

Fehlerbehandlung in ESQL

1) Explizites Testen von sqlca.sqlcode im Wirtsprogramm nach einer SQL-Anweisung:

= 0 → Anweisung korrekt ausgeführt, keine besonderen Vorkommnisse
< 0 → Fehler bei der Ausführung (siehe Fehlercodes im Manual)
> 0 → Anweisung ausgeführt, Auftreten eines Sonderfalls,
z.B. signalisiert der Wert 1403, daß keine (weiteren) Treffertupel existieren

Zusatzinformation in den restlichen Komponenten von sqlca, z.B.:

sqlca.sqlerrd[2]	Anzahl der Tupel, die von einer Insert-, Update- oder Delete-Anweisung betroffen waren
sqlca.sqlerrm.sqlerrmc	Fehlermeldung als Ascii-Text (max. 70 Zeichen)
sqlca.sqlerrm.sqlerrml	Länge der Fehlermeldung

2) Deklaration von "Exception-Handling"-Strategien:

```
EXEC SQL WHENEVER   ( SQLERROR | NOT FOUND | SQLWARNING )
                        ( STOP | GOTO label | CONTINUE )
```

wobei
SQLERROR einem SQLCODE < 0 entspricht,
NOT FOUND dem SQLCODE 1403 und
SQLWARNING einem SQLCODE > 0 (aber ungleich 1403).

Der ESQL-Precompiler erzeugt automatisch nach jeder SQL-Anweisung
einen entsprechenden Vergleich mit sqlca.sqlcode, und zwar jeweils aufgrund
der *textuell* letzten WHENEVER-Anweisung vor der SQL-Anweisung.
Extrem schwerwiegende Fehler führen immer zum sofortigen Abbruch.

Achtung: Die WHENEVER-Klausel hat potentielle Fallen.

1) Im folgenden Programm ist in f die WHENEVER-Spezifikation von g nicht wirksam.

```
void f (...) ...
   { ...   EXEC SQL UPDATE ...; ... };
void g (..) ...
   { ...   EXEC SQL WHENEVER SQLERROR GOTO handle_error; ...
           f (...); ...
   };
```

2) Das folgende Programm führt u.U. zu einer Endlosschleife.

```
...
EXEC SQL WHENEVER SQLERROR GOTO handle_error;
EXEC SQL CREATE TABLE Mytable ( ... );
...
handle_error:
   EXEC SQL DROP TABLE Mytable;
   EXEC SQL DISCONNECT;
   exit (1);
```

Korrektur:
Im Teil hinter der Marke "handle_error" als erstes spezifizieren:
EXEC SQL WHENEVER SQLERROR CONTINUE;

Verarbeitung von Tupelmengen mittels Cursor-Konzept

Zweck:
Verarbeitung von Tupelmengen in einer nichtmengenorientierten Wirtsprogrammiersprache.
Notwendig für alle Anfragen mit mehr als einem Resultattupel.

Beispiel: Ausgabe aller Bestellungen eines Kunden

```
#define TRUE 1
#define FALSE 0
...
printf ("Bitte geben Sie eine Kundennummer ein. (0 = Programmende)\n");
scanf ("%d", &knr);
EXEC SQL DECLARE Kundeniterator CURSOR FOR
    SELECT Monat, Tag, Bez, Menge
    FROM Bestellungen WHERE KNr = :knr
    ORDER BY Monat DESC, Tag DESC;
/* Cursor-Deklaration immer ohne INTO-Klausel */
...
EXEC SQL OPEN Kundeniterator;
/*  An dieser Stelle werden die Eingabeparameter der SQL-Anweisung (:knr) ausgewertet,
    und gedanklich wird hier die Resultattupelmenge ermittelt.                    */
found = TRUE;
while (found) /* solange es noch Resultattupel gibt */
    {
    EXEC SQL FETCH Kundeniterator INTO :monat, :tag, :bez, :menge;
    /* Hier werden die Wirtsprogrammvariable für die Resultattupel festgelegt. */
    bez.arr[bez.len] = '\0';
    if ((sqlca.sqlcode >= 0) && (sqlca.sqlcode != 1403))
        printf ("%d.%d.: %d %s", tag, monat, menge, bez);
    else
        found = FALSE;
    }; /*while*/
EXEC SQL CLOSE Kundeniterator;
/* Freigabe des Cursors und der damit verbundenen DBS-Ressourcen */
```

6.2 Dynamisches ESQL

Zweck:

Einbettung von SQL-Anweisungen, deren Struktur erst zur Laufzeit des Wirtsprogramms bekannt ist.

Bei Einbettung von SQL in die (auf Bytecodebene interpretierte) Programmiersprache Java mit JDBC als Datenbankzugriffsprotokoll (einer Erweiterung von ODBC) ist dynamisches SQL der Regelfall. Die Vorübersetzung von SQL-Anweisungen im Java-Programm ist im Standard SQLJ vorgesehen, bislang aber wenig verbreitet.

Einfacher Fall: Vorgehensweise bei statisch festgelegtem Resultatschema und statisch festgelegten Eingabeparametern:

Beispiel ("dynamischer Anteil" ist unterstrichen):

```
SELECT Monat, Tag, Bestellungen.PNr, Bez, Menge
FROM Bestellungen, Produkte WHERE Bestellungen.PNr = Produkte.PNR
AND ( Bez LIKE 'Druck%' OR Bez LIKE 'Papier%' OR Bez LIKE 'Platte%' )
ORDER BY Monat DESC, Tag DESC
```

Vorgehensweise:

1) Deklaration der Wirtsprogrammvariablen
2) Aufbau der SQL-Anweisung als Zeichenkette
3) Dynamische Precompilation: PREPARE DynQuery FROM :sqltext
4) DECLARE Iterator CURSOR FOR DynQuery
5) Auswertung der Eingabeparameter und Vorbereitung der "Cursor"-Schleife:
   OPEN Iterator
6) FETCH Iterator INTO Wirtsprogrammvariablen
7) Resultatsattribute in den Wirtsprogrammvariablen verarbeiten
8) Wiederholung der Schritte 6 und 7
   für jedes Resultatstupel
9) CLOSE Iterator

Sonderfall: Anfrageresult hat höchstens ein Tupel
(z.B. bei Anfragen über den Primärschlüssel oder bei Änderungsoperationen)

statt 3) bis 9) einfacher:

```
EXECUTE IMMEDIATE :sqltext
```

Komplexer Fall: Vorgehensweise bei dynamisch festgelegtem Resultatschema oder dynamisch festgelegten Eingabeparametern (mit dynamischer Allokation der Datenübergabebereiche und dynamischer Übersetzung für wiederholte Ausführung):

Abfrage der Typbeschreibung mit DESCRIBE queryname INTO ptr-to-sqldescriptorarea,
dynamische Allokation der Übergabebereiche im Programm und
Eintragen entsprechender Pointer in die sqldescriptorarea (SQLDA) vor der Queryausführung.
(siehe Handbücher; für Vorlesung nicht relevant)

Beispiel für den einfachen Fall:

```
/***  Programm zum Ausdrucken der Bestellungen fuer eine Liste von Produkten  ***/
/***  Eingabe: Liste von Produktbezeichnungen                                 ***/
/***  Ausgabe: Bestellungen dieser Produkte, absteigend sortiert nach Datum   ***/

#include <stdio.h>
#include <string.h>
#define TRUE 1
#define FALSE 0
#define MAXPRODUKTE 10
/* Die Bestellungen umfassen maximal 10 Produkte */
EXEC SQL INCLUDE SQLCA; /* Importieren der SQLCA */

main () {
EXEC SQL BEGIN DECLARE SECTION;
    int monat, tag, pnr, menge;
    VARCHAR bez[31];
    VARCHAR sqltext[512];
    VARCHAR user[20];
    VARCHAR passwd[10];
EXEC SQL END DECLARE SECTION;

EXEC SQL WHENEVER SQLERROR GOTO handle_error;

/* Typdeklarationen */
typedef char prod_bez_t[31];
typedef int bool;

/* Variablendeklarationen */
prod_bez_t    prod_bez[10];
int           i;
int           num_of_prod;
bool          found;

printf ("Benutzername?"); scanf ("%s", &(user.arr)); user.len = strlen(user.arr);
printf ("Kennwort?"); scanf ("%s", &(passwd.arr)); passwd.len = strlen(passwd.arr);
EXEC SQL CONNECT :user IDENTIFIED BY :passwd;

/* Eingabe */
i = 0;
printf ("%s%s", "Bitte geben Sie eine Produktbezeichnung oder \"ende\" ein.\n");
scanf ("%s", &(prod_bez[i]));
while ((i < MAXPRODUKTE) && (strcmp (prod_bez[i], "ende") != 0))
    {
    i++;
    printf ("%s%s", "Bitte geben Sie eine Produktbezeichnung oder \"ende\" ein.\n");
    scanf ("%s", &(prod_bez[i]));
    }; /*while*/
num_of_prod = i;
if (num_of_prod = = 0) exit(-1);
```

```
/* Aufbau der SQL-Anfrage */
strcpy (sqltext.arr, "SELECT Monat, Tag, Bestellungen.PNr, Bez, Menge ");
strcat (sqltext.arr, "FROM Bestellungen, Produkte ");
strcat (sqltext.arr, "WHERE Bestellungen.PNr = Produkte.PNr ");
strcat (sqltext.arr, "AND ( ");
strcat (sqltext.arr, "Bez LIKE \'");
strcat (sqltext.arr, prod_bez[0]);
strcat (sqltext.arr, "%\' ");
for (i=1; i < num_of_prod; i++) {
    strcat (sqltext.arr, "OR Bez LIKE \'");
    strcat (sqltext.arr, prod_bez[i]);
    strcat (sqltext.arr, "%\' ");
    }; /*for*/
strcat (sqltext.arr, ") ");
strcat (sqltext.arr, "ORDER BY Monat DESC, Tag DESC");
sqltext.len = strlen(sqltext.arr);

/*  Nur zu Testzwecken */
printf ("\nDie generierte SQL-Anfrage lautet:\n"); printf ("%s\n\n", sqltext.arr);
/*  zum Beispiel:
    SELECT Monat, Tag, Bestellungen.PNr, Bez, Menge
    FROM Bestellungen, Produkte WHERE Bestellungen.PNr = Produkte.PNR
    AND ( Bez LIKE 'Druck%' OR Bez LIKE 'Papier%' OR Bez LIKE 'Platte%' )
    ORDER BY Monat DESC, Tag DESC                                          */

EXEC SQL PREPARE DynQuery FROM :sqltext;
EXEC SQL DECLARE BestIterator CURSOR FOR DynQuery;
EXEC SQL OPEN BestIterator;
found = TRUE;
while (found)
    {
    EXEC SQL FETCH BestIterator INTO :monat, :tag, :pnr, :bez, :menge;
    if (sqlca.sqlcode = = 0) {
        bez.arr[bez.len] = '\0';
        /* Ausgabe */
        printf ("%d.%d.: %d Stueck %s (PNr %d)\n", tag, monat, menge, bez.arr, pnr);
        }
    else
        found = FALSE;
    }; /*while*/
EXEC SQL CLOSE BestIterator;
EXEC SQL COMMIT WORK RELEASE;
exit (0);

/* Fehlerbehandlung */
handle_error:
    printf ("\n*** Error %d in program. ***\n", sqlca.sqlcode);
    printf ("%.70s\n", sqlca.sqlerrm.sqlerrmc);
    exit (-1);
} /*main*/
```

Beispiele für Anwendungen, die dynamisches ESQL erfordern:

- SQL*Plus und ähnliche universelle, interaktive DBS-Schnittstellen sind Programme, die dynamisches ESQL verwenden.

- Anwendungen, die sowohl Daten als auch Metadaten (Relationen des DB-Katalogs) verarbeiten, benötigen in vielen Fällen dynamisches ESQL.
  Solche Anwendungen sind mit dynamischem ESQL realisierbar, weil in praktisch allen relationalen DBS der DB-Katalog in Form von Relationen zugreifbar ist.

  Beispiel (Achtung: in SQL so nicht möglich !):
  SELECT * FROM * WHERE * LIKE '%Lafontaine%'

Wichtige Katalogrelationen in Oracle:

SYSCATALOG bzw. SYS.ALL_TABLES UNION SYS.ALL_VIEWS

OWNER	TABLE_NAME	TABLE_TYPE
DBS	BESTELLUNGEN	TABLE
DBS	KUNDEN	TABLE
DBS	PRODUKTE	TABLE
SYS	ALL_TABLES	VIEW
. . .		

SYS.ALL_TAB_COLUMNS

OWNER	TABLE_NAME	COLUMN_ NAME	DATA_ TYPE	DATA_ LENGTH	...	COLUMN_ ID	...
DBS	KUNDEN	KNR	NUMBER	22		1	
DBS	KUNDEN	NAME	CHAR	30		2	
DBS	KUNDEN	STADT	CHAR	30		3	
DBS	KUNDEN	SALDO	NUMBER	22		4	
DBS	KUNDEN	RABATT	NUMBER	22		5	
DBS	PRODUKTE	PNR	NUMBER	22		1	
. . .							
SYS	ALL_TABLES	OWNER	CHAR	30		1	
SYS	ALL_TABLES	TABLE_NAME	CHAR	30		2	
. . .							

Stored Procedures: Einbettung von SQL in eine 4GL
(Beispiel: PL/SQL von Oracle)

Die Prinzipien sind wie bei ESQL. Syntaktisch ergeben sich einige Vereinfachungen. Andererseits sind 4GLs häufig gegenüber Standardprogrammiersprachen wie (z.B. C) in ihren Datentypen und sonstigen Ausdrucksmitteln beschränkt.

Grobsyntax einer Prozedurdeklaration:

```
proc-decl =    CREATE PROCEDURE proc-name
               [ ”(" param-name type { ”," param-name type } ”)" ] AS proc-body
proc-body =    decl-part block
block =  BEGIN statement-list [ exception-part ] END
```

Beispiel:

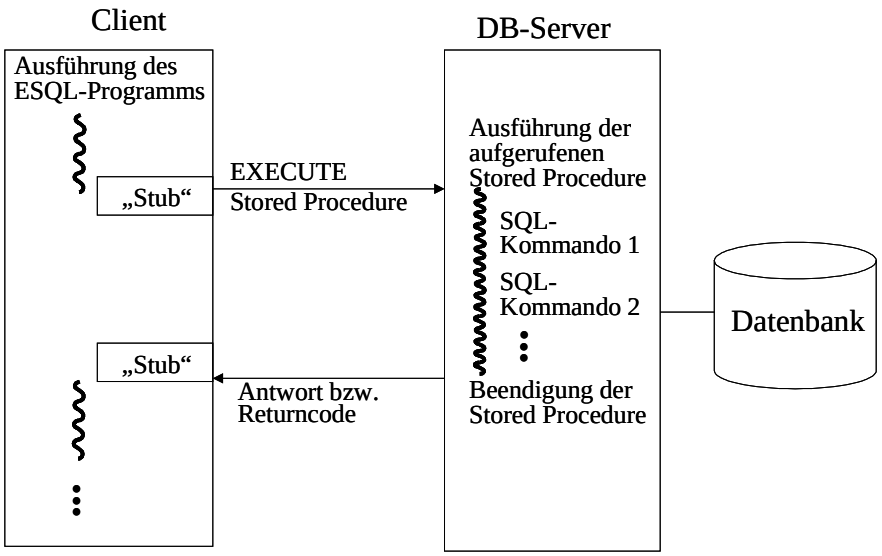
```
CREATE PROCEDURE Lager_Auffuellen AS
DECLARE ...;
BEGIN
   DECLARE CURSOR Knappe_Produkte IS
      SELECT PNr, Vorrat
      FROM Produkte
      WHERE Vorrat < 100;
   KP Knappe_Produkte%ROWTYPE;
   BEGIN
      OPEN Knappe_Produkte;
      LOOP
         FETCH Knappe_Produkte INTO KP;
         EXIT WHEN Knappe_Produkte%NOTFOUND;
         ...
         IF KP.Vorrat > 10
         THEN
            EXECUTE Nachbestellen (KP.PNr, 1000 - KP.Vorrat)
         ELSE
            EXECUTE Express_Bestellen (KP.PNr, 10);
            EXECUTE Nachbestellen (KP.PNr, 1000 - KP.Vorrat - 10);
         END IF;
      END LOOP;
   END;
END;
```

Aufruf der PL/SQL Stored Procedure direkt von SQL*PLus, SQL*Forms oder ESQL-Programmen möglich,

```
z.B.: EXECUTE Lager_Auffuellen
```

Mehrere Prozeduren und Funktionen können zu einem Modul - in Oracle-Terminologie einem *Package* - zusammengefaßt und unter einem global bekannten Namen den Anwendungsentwicklern zur Verfügung gestellt werden.

Laufzeitarchitektur für Stored Procedures:



6.3 Web-Anwendungen mit PHP

6.3.1 Exkurs Web-Technologien

Das Web (WWW = World Wide Web) ist eine – riesige – Kollektion von Hypertextdokumenten: ein Dokument besteht dabei aus (formatiertem und teilweise strukturiertem) Text, eingebetteten Bildern, Grafiken und anderen Objekten sowie Zeigern, sog. Hyperlinks, auf andere Dokumente (daher der Name Hypertext). Das Web umfaßt, soweit es durch die Crawler (Robots) der führenden Suchmaschinen (Altavista, Google, etc.) abgedeckt ist, ca. 1 Milliarde Dokumente. Hinter sog. Business-Portals (z.B. Patentsammlungen, Nachrichtenarchive, digitale Bibliotheken, technische Diskussionsforen, usw.) gibt es jedoch weitere Informationen, die für globale Suchmaschinen nicht (oder nur sehr eingeschränkt) zugänglich sind; dieses sog. „Deep Web“ wird auf 500 Milliarden "Dateneinheiten" geschätzt.

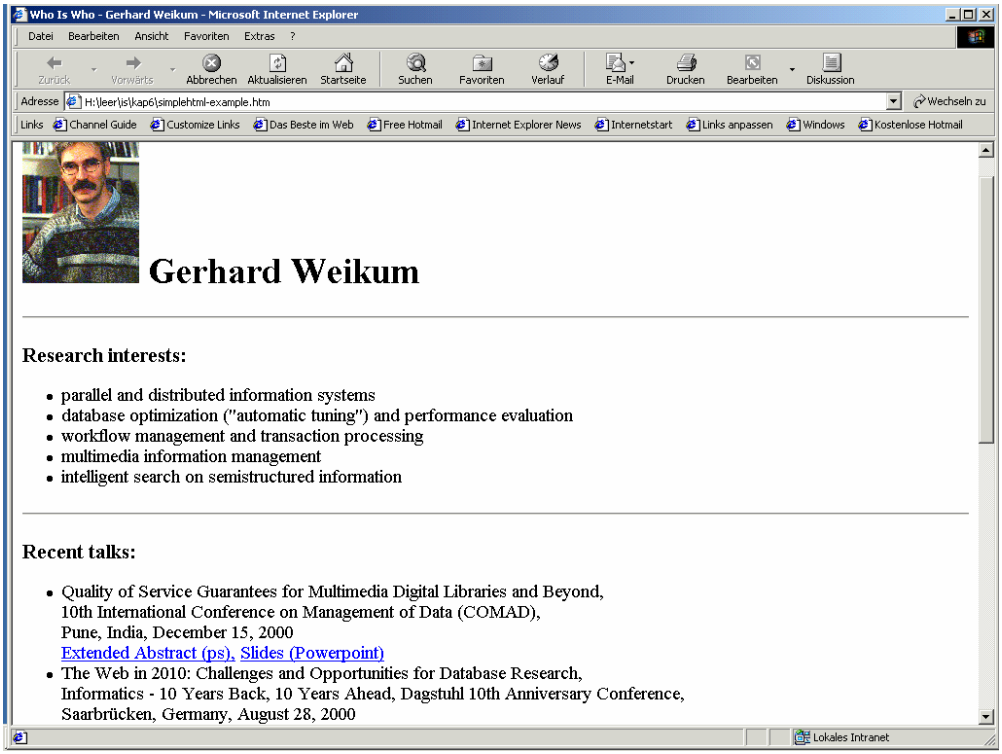
Web-Dokumente werden in der Regel in Form von HTML-Seiten repräsentiert und von einem auf praktisch jedem PC oder sonstigem Client verfügbaren Internet-Browser (Internet Explorer oder Netscape Communicator) visualisiert. **HTML (Hypertext Markup Language)** ist eine Dokumentbeschreibungssprache, mit der die Grobstruktur (z.B. Überschriften, Listen, u.ä.) und Präsentationsform (z.B. Fonts) von Hypertextdokumenten in Form von speziell markierten Annotationen – sogenannten „Tags“ spezifiziert wird. Es gibt eine festgelegte Menge möglicher Tags, und Tags werden in der Regel (aber nicht notwendigerweise) geschachtelt. HTML ist ein Standard des W3C (World Wide Web Consortium, siehe www.w3.org).

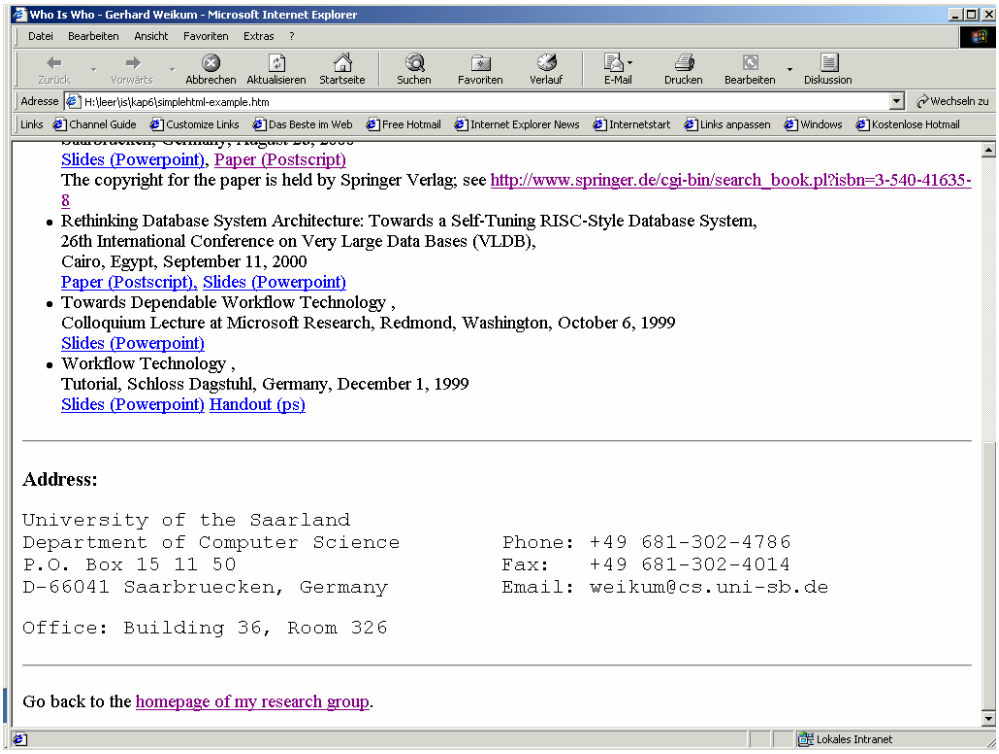
Bei dem neueren W3C-Standard **XML (Extensible Markup Language)** werden die Beschreibung der Dokumentstruktur und der Präsentation getrennt. XML bezieht sich nur auf die Dokumentstruktur, wobei anwendungsspezifisch beliebige Tags eingeführt werden können; die Verwendung der Tags kann wiederum gemäß einer DTD (Document Type Definition) oder eines XML-Schemas eingeschränkt werden. Die Präsentation eines XML-Dokuments im Browser wird über ein in der XSL (Extensible Stylesheet Language) verfaßtes sog. Stylesheet gesteuert.

Beispiel einer einfachen HTML-Datei:

```
<html>

<head>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
<meta name="GENERATOR" content="Microsoft FrontPage 4.0">
<title> Who Is Who - Gerhard Weikum </title>
</head>

<body>

<h1><img src="weikum-photo2-72dpi.gif" width="240" height="342">    Gerhard Weikum </h1>
<hr>

<h3>Research interests: </h3>
<ul>
    <li>parallel and distributed information systems </li>
    <li>database optimization (&quot;automatic tuning&quot;) and performance evaluation </li>
    <li>workflow management and transaction processing</li>
    <li>multimedia information management</li>
    <li>intelligent search on semistructured information</li>
</ul>
<hr>

<h3>Recent talks: </h3>
<ul>
<li>Quality of Service Guarantees for Multimedia Digital Libraries and Beyond,<br>
    10th International Conference on Management of Data (COMAD),<br>
    Pune, India, December 15, 2000<br>
    <a href="http://www-dbs.cs.uni-sb.de/~weikum/comad00.ps">Extended Abstract (ps),</a>
    <a href="http://www-dbs.cs.uni-sb.de/~weikum/comad00.ppt">Slides (Powerpoint)</a></li>
<li>The Web in 2010: Challenges and Opportunities for Database Research,<br>
    Informatics - 10 Years Back, 10 Years Ahead, Dagstuhl 10th Anniversary Conference, <br>
    Saarbrücken, Germany, August 28, 2000<br>
    <a href="http://www-dbs.cs.uni-sb.de/~weikum/dagstuhl00.ppt">Slides (Powerpoint)</a>,
    <a href="http://www-dbs.cs.uni-sb.de/~weikum/dagstuhl00.ps">Paper (Postscript)</a><br>
    The copyright for the paper is held by Springer Verlag; see
    <a href="http://www.springer.de/cgi-bin/search_book.pl?isbn=3-540-416358">
    http://www.springer.de/cgi-bin/search_book.pl?isbn=3-540-41635-8</a><br></li>
<li>Rethinking Database System Architecture:
    Towards a Self-Tuning RISC-Style Database System,<br>
    26th International Conference on Very Large Data Bases (VLDB),<br>
    Cairo, Egypt, September 11, 2000<br>
    <a href="http://www-dbs.cs.uni-sb.de/~weikum/vldb00.pdf">Paper (Postscript),</a>
    <a href="http://www-dbs.cs.uni-sb.de/~weikum/vldb00.ppt">Slides (Powerpoint)<br></a></li>
<li>Towards Dependable Workflow Technology ,<br>
    Colloquium Lecture at Microsoft Research, Redmond, Washington, October 6, 1999<br>
    <a href="http://www-dbs.cs.uni-sb.de/~weikum/msr99-talk.ppt">Slides (Powerpoint)</a><br>
    </li>
<li>Workflow Technology ,<br>
    Tutorial, Schloss Dagstuhl, Germany, December 1, 1999<br>
    <a href="http://www-dbs.cs.uni-sb.de/~weikum/dagstuhl-lehrer99.ppt">Slides(Powerpoint)</a>
    <a href="http://www-dbs.cs.uni-sb.de/~weikum/dagstuhl-lehrer99.ps">Handout (ps)</a> </li>
</ul>
<hr>

<h3>Address: </h3>
<pre><font size="4">University of the Saarland
Department of Computer Science        Phone: +49 681-302-4786
P.O. Box 15 11 50                     Fax:   +49 681-302-4014
D-66041 Saarbruecken, Germany         Email: weikum@cs.uni-sb.de</font>
</pre>
<pre><font size="4">Office: Building 36, Room 326</font></pre>
<hr>
<p>Go back to the <a href="http://www-dbs.cs.uni-sb.de/">homepage of my research group</a></p>
</body>
</html>
```

Diese Datei sieht in einem Web-Browser folgendermaßen aus:





Web-Seiten werden mittels einer URL (Uniform Resource Locator) identifiziert und mit dem **HTTP-Protokoll (Hypertext Transfer Protocol)** von den Server-Rechnern, auf denen sie jeweils gespeichert sind, zu dem anfordernden Client (dem Web-Browser auf dem Rechner des Benutzers) transportiert. Dazu wird die **GET-Methode** von HTTP verwendet, die als Eingabeparameter außer der URL auch noch weitere sog. Header-Informationen wie z.B. „If-Modified-Since <date>“ haben kann. In diesem konkreten Fall wird nur dann ein Dokument zurückgeliefert, wenn es seit <date> geändert wurde. Das Dokument wird dann typischerweise aus dem Client-Cache oder einem sog. Proxy-Cache geladen; durch den bedingten GET-Aufruf ist aber die Aktualität der Daten sichergestellt. Das Resultat eines GET-Aufrufs besteht in der Regel aus den Daten der angeforderten Web-Seite und/oder Statuscodes wie z.B. „404 Not Found“. Außer GET gibt es noch einige weitere http-Methoden: die POST-Methode spielt bei dynamischen Web-Seiten mit Formularmasken eine wichtige Rolle (siehe unten), die anderen Methoden dienen Testzwecken oder haben eine untergeordnete Bedeutung.

Das HTTP-Protokoll erfordert auf Client-Seite einen Internet-Browser oder ein sonstiges Programm, das die HTTP-Aufrufe startet und deren Resultate entgegennimmt, sowie auf Server-Seite einen Web-Server (auch HTTP-Server genannt; z.B. Apache, Internet Information Server, usw.). HTTP basiert seinerseits auf dem **TCP/IP-Protokoll** des Internets sowie weiteren Internet-Dienste, insbesondere dem DNS (Domain Name Service). Ein DNS-Server wird aufgerufen, um die angesprochene URL in eine für TCP/IP geeignete Adresse umzusetzen. Eine URL besteht generell aus den vier Komponenten Zugriffsprotokoll (in der Regel http, oder auch ftp, usw.), Host-Name oder Domain-Name (z.B. www-dbs.cs.uni-sb.de), einem Port (in der Regel 80, so daß diese Angabe weggelassen werden kann) und einem Pfad im Filesystem des angesprochenen Hosts. Mittels DNS wird der Host- oder Domain-Name in eine IP-Nummer, der weltweit eindeutigen Nummer eines Rechners am Internet übersetzt (z.B. 134.96.246.99). Zwischen Client und Server wird dann mit dem **TCP-Protokoll** (Transmission Control Protocol) eine Sitzung eingerichtet; die beiden Kommunikationspartner werden jeweils über ihre IP-Nummer und eine lokale Port-Nummer identifiziert (aus Sicht der Programmierung ist dies die Verbindung sog. Sockets). Eine solche Sitzung garantiert die zuverlässige und ordnungserhaltende Übermittlung aller Nachrichten zwischen Sender und Empfänger (mit Hilfe von automatisch erzeugten Bestätigungsnachrichten, sog. Acknowledgements oder kurz Acks, und automatisch erzeugten Nachrichtensequenznummern). TCP selbst basiert auf dem **IP-Protokoll** (Internet Protocol), welches einzelne Datenpakete vom jeweiligen Sender zum Empfänger transferiert, dabei aber auch Pakete „verlieren“ kann. Das IP-Protokoll beinhaltet das automatische Routing von Paketen – über geeignet gewählte „Relais“-Stationen im Internet – sowie die Fehlererkennung und –korrektur bzgl. korrumpierter Bits in einem Paket.

In interessanten Anwendungen werden die Inhalte von Web-Seiten vom angesprochenen Web-Server häufig dynamisch bestimmt; beispielsweise werden Katalogangaben von E-Commerce-Händlern durch Datenbankzugriffe auf Seiten des Web-Servers gefüllt (nach Erhalt des GET-Auftrags und vor dem Senden der Antwort). Zu diesem Zweck unterstützen praktisch alle Web-Server das CGI-Protokoll (Common Gateway Interface) sowie weitere nicht standardisierte, aber weitverbreitete Protokolle zum Aufruf externer Programme, sog. Servlets. Diese Servlets sind häufig kurze Programme in einer Skriptsprache wie Perl, Active Server Pages oder PHP, können aber auch Java-Programme sein, wenn der Web-Server eine Java Virtual Machine hat (also die Laufzeitumgebung für Java Bytecode), und natürlich können Skripts selbst wiederum Programme in C, C++, Java, VisualBasic, etc. aufrufen. Alle diese Varianten von Servlets können auch mit Datenbanksystemen in SQL kommunizieren.

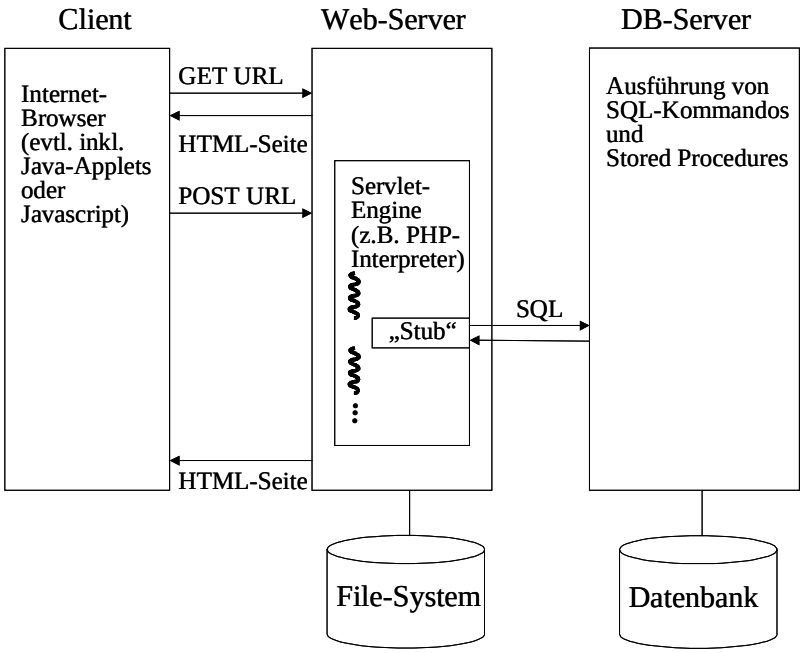
Servlets haben häufig Eingabeparameter (wie z.B. Kundennummern, Produkte, für die sich der Kunde interessiert, usw.); diese werden z.B. auf der Client-Seite durch Ausfüllen einer Formularmaske in der vom Browser dargestellten Web-Seite eingegeben. Wenn das Servlet mit der GET-Methode aufgerufen wird, werden solche Parameter geeignet codiert an die URL der Web-Seite angehängt, z.B. in der Form

http://www.my-eshop.de/ show_products.php?kundennummer=123456&Produkte=Buecher .

Ein besser strukturierte Form der Parameterübergabe vom Client an das Servlet erfolgt mit der http-Methode **POST** (Beispiele siehe unten). Diese übergibt die in eine Formularmaske eingegeben Werte an das Servlet in Form von expliziten (Environment-) Variablen, auf die das Servlet Zugriff hat.

Eine der elegantesten und am leichtesten handzuhabenden Skriptsprachen für Servlets ist PHP (PHP Hypertext Preprocessor), die für die beiden den Markt dominierenden Web-Server Apache und Internet Information Server (IIS) verfügbar ist.

Laufzeitarchitektur für Web-fähige Servlets



6.3.2 Programmierung dynamischer Web-Seiten mit PHP

Web-fähige Datenbankanwendungen basieren in der Regel auf Servlets, relativ kleinen Programmen bzw. Skripts, die unter der Kontrolle eines Web-Servers laufen (z.B. Apache oder IIS). Servlets werden durch eine HTTP-Nachricht (GET oder POST) vom Web-Server aktiviert und generieren - typischerweise mit Hilfe von Datenbankzugriffen - dynamische HTML-Seiten, die der Web-Server an den Internet-Browser des Clients sendet.

PHP (PHP Hypertext Preprocessor) ist eine - vermutlich die populärste - Skriptsprache zur Erstellung solcher Servlets. PHP wird direkt in eine HTML-Datei eingebettet, die die Datei-Extension .php haben muß. Wenn auf diese Datei via HTTP GET zugegriffen wird, interpretiert der Web-Server die Datei zunächst als HTML-Seite; er erkennt eingebetteten PHP-Code durch das Tag <?php ... ?> und schaltet die eigentliche PHP-Engine (deren Kern die sog. Zend-Engine ist) zur Interpretation des PHP-Codes ein. Die - mittels echo oder printf erzeugte - Ausgabe des PHP-Codes wird einfach an die entsprechenden Stellen der HTML-Seite hineinkopiert, und die so erzeugte dynamische HTML-Seite wird an den Internet-Browser geschickt. Auf diese Weise kann der Benutzer auf der Browser-Seite niemals den Quellcode der PHP-Seite sehen.

PHP selbst ist syntaktisch an C und die Skriptsprache Perl (die sich wiederum an Unix-Shells orientiert) angelehnt sowie in Teilen, die objektorientierten Charakter haben an C++. Man kann in PHP extrem flexibel und kompakt programmieren und hat ein umfangreiches Repertoire an "Features", was einerseits die Produktivität bei der Anwendungsentwicklung erhöhen kann, andererseits Ungeübten Schwierigkeiten beim Lesen von Skripts und vor allem beim Debugging bereiten kann. Beispielsweise sind Variablen praktisch typfrei. Zur Leistungsfähigkeit von PHP gehört eine extreme große Bibliothek an Funktionen (inkl. Datenbankzugriffsfunktionen) sowie ein komfortabler Zugriff auf die Umgebungsdaten der HTTP-Methoden (z.B. die IP-Adresse und der Browser-Typ des Clients, die evtl. mittels GET oder POST übergebenen Parameter, Cookies, etc.).

Ein erstes, sehr einfaches, Beispiel illustriert die Funktionsweise von PHP und den Zugriff auf Umgebungsdaten.

Beispiel: willkommen0.php

```
<html>
<head>
   <title> Einfaches PHP-Beispiel </title>
</head>
<body>
<h1> Willkommen bei der Bank Primitivo <br></h1>

Ihre IP-Nummer ist
<?php //eingebetteter PHP-Code
# alle Fehler- und Warnungsmeldungen aktivieren
error_reporting(E_ERROR | E_WARNING | E_PARSE | E_NOTICE);
echo $REMOTE_ADDR;
?>, <br>
und Sie benutzen den Browser <?php echo $HTTP_USER_AGENT; ?>.<br>
<?php printf("Heute ist "); /* C-Syntax ist ebenfalls erlaubt */
      printf("%s", date(DdFY));?><br>
<?php printf("Die aktuelle Uhrzeit ist ");
      printf("%s:%s", date(H), date(i)); ?><br>

</body>
</html>
```

Beim Aufruf von willkommen0.php via HTTP vom Internet-Browser wird die folgende HTML-Datei erzeugt:

```
<html>
<head>
   <title> Einfaches PHP-Beispiel </title>
</head>

<body>
   <h1> Willkommen
        bei der Bank Primitivo <br>
   </h1>

Ihre IP-Nummer ist
134.96.246.96, <br>
und Sie benutzen den Browser Mozilla/4.0 (compatible; MSIE 5.5; Windows NT
5.0).<br>
Heute ist Mon23April2001<br>
Die aktuelle Uhrzeit ist 08:38<br>

</body>
</html>
```

Diese sieht am Bildschirm wie folgt aus:

6.4 Datenbankzugriff in PHP

Der Zugriff auf Datenbanken von Servlets ist die wichigste Technik zur Erstellung dynamischer Web-Seiten. PHP unterstützt dies in Form von Funktionen, die sich an Embedded SQL anlehnen, aber im Detail Unterschiede aufweisen und zudem immer erst zur Laufzeit ausgewertet werden, also eher mit dynamischem ESQL zu vergleichen sind. Am besten wird der Zugriff auf das unter der GNU-Lizenz kostenfrei verfügbare System MySQL unterstützt, allerdings wird dort nur eine stark eingeschränkte Teilmenge von SQL unterstützt. Der - robusteste und effizienteste - Zugriffe auf Oracle8i-Datenbanken erfolgt mit den Funktionen des OCI (Oracle Call Interface). Die wichtigsten Funktionen dieser Schnittstelle sind im folgenden PHP-Beispiel illustriert; die entsprechenden Funktionsnamen beginnen alle mit "oci". Der besser strukturierte Umgang mit Wirtsprogrammvariablen verwendet die Funktion ocidefinebyname (für Resultatsparameter von SQL-Anweisungen) bzw. ocibindbyname (für Eingabeparameter von SQL-Anweisungen). Für Anfänger einfacher zu programmieren ist jedoch vermutlich die Verwendung von PHP-Stringmethoden (z.B. einfache Konkatenation mittels ".") für Eingabeparameter und ociresult für Anfrageresultaten; nur bei komplexeren Anwendungen wird dieses Vorgehen unübersichtlich und ineffizient. In beiden Fällen wird auf die Resultatstupel einer Anfrage mit ocifetch - ähnlich dem Cursor-Konzept von ESQL - zugegriffen.
Das folgende Beispiel illustriert beide Techniken. Dem Beispiel liegt das folgende Datenbankschema zugrunde:

```
-- Kunden-Tabelle:
create table customer
   (name varchar(20) primary key, ipnumber varchar(20), city varchar(20));
-- Konto-Tabelle:
create table account (name varchar(20) primary key, balance integer);
-- Kunden-Tabelle:
create table history (name varchar(20), bookingdate date, amount integer);
```

Die angestrebte Applikationslogik dieses für Web-Anwendungen typischen, hier in mehreren Verfeinerungsstufen entwickelten, Beispiels hat folgenden Aufbau:

look up customer in database
check if customer has filled form
if (form) parameters available
then
  insert customer info into database
  send personalized greeting
else send blank form
create output with links to „transfer“ and „audit“ servlets

Beispiel: willkommen1.php

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML3.2//EN">
<head><title> PHP-Beispiel mit einfachem SQL </title></head>
<?php
error_reporting(E_ERROR | E_PARSE); /* nur wirkliche Fehlermeldungen ausgeben */
$db = "(DESCRIPTION =
       (ADDRESS_LIST =(ADDRESS = (PROTOCOL = TCP)(HOST = tokyo)(PORT = 1521)))
       (CONNECT_DATA =(SERVICE_NAME = tokyo.world)))";
$connection = ocilogon ("lehre40", "leere4zig", $db);
# Achtung: eigentlich sollten Kennwörter niemals in dieser Datei stehen,
#          auch wenn der Endbenutzer diesen Sourcecode nicht im Browser sieht
# Test auf Fehlersituationen
$ocistatus = ocierror($stmt);
echo $ocistatus["code"] . " " . $ocistatus["message"] . "\n";
?>
<body>
   <h1> Willkommen,
        <?php
          $sqlstring = "SELECT * FROM CUSTOMER WHERE IPNUMBER = '"
                       . $REMOTE_ADDR . "'";
          $stmt = ociparse ($connection, $sqlstring);
          $ocistatus = ocierror($stmt);
          echo $ocistatus["code"] . " " . $ocistatus["message"] . "\n";
          ociexecute ($stmt);
          $ocistatus = ocierror($stmt);
          echo $ocistatus["code"] . " " . $ocistatus["message"] . "\n";
          ocifetch ($stmt);
          $customername = ociresult ($stmt, "NAME");
          $customeripnumber = ociresult ($stmt, "IPNUMBER");
          $customercity = ociresult ($stmt, "CITY");
          /* alternative Variante zur Übergabe der Resultatstupel:
             ocidefinebyname ($stmt, "NAME", $customername);
             ocidefinebyname ($stmt, "IPNUMBER", $customeripnumber);
             ocidefinebyname ($stmt, "CITY", $customercity);
             ociexecute ($stmt);
             $ocistatus = ocierror($stmt);
             echo $ocistatus["code"] . " " . $ocistatus["message"] . "\n";
             ocifetch ($stmt);
          */
          echo $customername;
         ?>
        , bei der Bank Primitivo <br>
   </h1>
Ihre IP-Nummer ist <?php echo $REMOTE_ADDR ?>,
und Sie benutzen den Browser <?php echo $HTTP_USER_AGENT ?>.<br>
<?php printf("Heute ist ");
      printf("%s", date(DdFY)); ?><br>
<?php printf("Die aktuelle Uhrzeit ist ");
      printf("%s:%s", date(H), date(i)); ?><br>
<?php echo "Wie ist das Wetter in " . $customercity . "? <br>" ?>
<?php ocifreestatement ($stmt); ocilogoff ($connection);
# solche cleanup-Anweisungen generiert PHP eigentlich automatisch
?>
</body></html>
```

Umgang mit HTML-Formularen und Übergabe von Parametern zwischen HTML-Seiten

Auf der Client-Seite werden häufig Eingaben über HTML-Formulare (Tag <form>) gemacht und mit der HTTP-Methode POST an das auszuführende Servlet übermittelt. PHP übermittelt stellt dem aufgerufenen Skript diese Parameterwerte in Form von globalen Variablen zur Verfügung. Generell ist bei der Programmierung solcher formularbasierten Anwendungen zu beachten, daß auf die Web-Seite erst mit GET zugegriffen wird, dann auf der Browser-Seite die Formularfelder mit Eingabeparametern gefüllt werden und anschließend die Seite ein zweites Mal mit POST aufgerufen wird. Das PHP-Skript muß diese beiden Aufrufe unterscheiden können, was es typischerweise durch Abfragen der den Formularparametern (inkl. Buttons) entsprechenden globalen Variablen tut. Dies ist im folgenden Beispiel illustriert, bei dem bekannte Kunden unmittelbar begrüßt werden und bislang unbekannte Kunden (bzw. solche, deren Client-IP bislang nicht bekannt war) nach Ausfüllen des Formulars in der Datenbank registriert werden.

Beispiel: willkommen2.php

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML3.2//EN">
<head>
   <title> Erweitertes PHP-Beispiel mit einfachem SQL und einem Eingabeformular
   </title>
</head>
<?php
error_reporting(E_ERROR | E_PARSE);
$db = "(DESCRIPTION =
       (ADDRESS_LIST =(ADDRESS = (PROTOCOL = TCP)(HOST = tokyo)(PORT = 1521)))
       (CONNECT_DATA =(SERVICE_NAME = tokyo.world)))";
$connection = ocilogon ("lehre40", "leere4zig", $db);
$ocistatus = ocierror($stmt);
echo $ocistatus["code"] . " " . $ocistatus["message"] . "\n";
?>
<body>
   <h1> Willkommen
        <?php
          $sqlstring = "SELECT * FROM CUSTOMER WHERE IPNUMBER = '"
                       . $REMOTE_ADDR . "'";
          $stmt = ociparse ($connection, $sqlstring);
          $ocistatus = ocierror($stmt);
          echo $ocistatus["code"] . " " . $ocistatus["message"] . "\n";
          ociexecute ($stmt);
          $ocistatus = ocierror($stmt);
          echo $ocistatus["code"] . " " . $ocistatus["message"] . "\n";
          ocifetch ($stmt);
          $customername = ociresult ($stmt, "NAME");
          $customeripnumber = ociresult ($stmt, "IPNUMBER");
          $customercity = ociresult ($stmt, "CITY");
          /* alternative Variante zur Übergabe der Resultatstupel:
             ocidefinebyname ($stmt, "NAME", $customername);
             ocidefinebyname ($stmt, "IPNUMBER", $customeripnumber);
             ocidefinebyname ($stmt, "CITY", $customercity);
             ociexecute ($stmt);
             $ocistatus = ocierror($stmt);
             echo $ocistatus["code"] . " " . $ocistatus["message"] . "\n";
             ocifetch ($stmt);
          */
          if ($customername != "")
             {echo ", " . $customername . ", ";};
         ?>
        bei der Bank Primitivo <br>
   </h1>
Ihre IP-Nummer ist <?php echo $REMOTE_ADDR ?>,
und Sie benutzen den Browser <?php echo $HTTP_USER_AGENT ?>.<br>
<?php printf("Heute ist "); printf("%s", date(DdFY)); ?>
<br>
<?php printf("Die aktuelle Uhrzeit ist ");printf("%s:%s",date(H),date(i)); ?><br>
```

```
<?php if ($customername != "")
         {echo "Wie ist das Wetter in " . $customercity . "?<br>";}
      else {

           echo "<form method=post action=willkommen2.php>";
           echo "<br> Bitte geben Sie Ihren Namen und Ihren Wohnort ein. <br>";
           echo "Ihr Name: <input type=text name=inputname><br>";
           echo "Ihr Wohnort: <input type=text name=inputcity
                              value=Saarbruecken><br>";
           echo "<input type=submit name=inputenter
                              value=\"Hier hast Du's\"><br>";
           echo "</form>";
           if (($inputenter) and ($inputname != ""))
             {
              $sqlinsertstring = "INSERT INTO CUSTOMER VALUES ('"
                                 . $inputname . "', '" . $REMOTE_ADDR . "', '"
                                 . $inputcity . "')";
              $stmt = ociparse ($connection, $sqlinsertstring);
              ociexecute ($stmt);
              $ocistatus = ocierror($stmt);
              echo $ocistatus["code"] . " " . $ocistatus["message"] . "\n";
              /* alternative Variante zur Übergabe der SQL-Argumente:
                 $sqlinsertstring = "INSERT INTO CUSTOMER VALUES
                                     (:inputname, :inputipnumber, :inputcity)";
                 $stmt = ociparse ($connection, $sqlinsertstring);
                 ocibindbyname ($stmt, ":inputname", &$inputname, 25);
                 ocibindbyname ($stmt, ":inputipnumber", &$inputipnumber, 25);
                 ocibindbyname ($stmt, ":inputcity", &$inputcity, 25);
                 $inputipnumber = $REMOTE_ADDR;
                 ociexecute ($stmt);
                 $ocistatus = ocierror($stmt);
                 echo $ocistatus["code"] . " " . $ocistatus["message"] . "\n";
              */
              $sqlinsertstring = "INSERT INTO ACCOUNT VALUES ('"
                                 . $inputname . "', 0)";
              $stmt = ociparse ($connection, $sqlinsertstring);
              ociexecute ($stmt);
              $ocistatus = ocierror($stmt);
              echo $ocistatus["code"] . " " . $ocistatus["message"] . "\n";
             };
           };
?>
<?php ocifreestatement ($stmt); ocilogoff ($connection); ?>
</body> </html>
```

Beim ersten Aufruf von einer zuvor unbekannten IP-Nummer erzeugt das Skript folgende Ausgabe:

Nach Eingabe von "Gerhard Weikum" und "Dudweiler" in den beiden Eingabefeldern des Formulars und Clicken des "Hier hast Du's"-Knopfes wird die folgende Ausgabe erzeugt (die - weil dann die IP-Nummer in der Datenbank registriert ist - auch bei allen späteren Aufrufen dieser Seite von derselben IP-Adresse unmittelbar angezeigt wird):

Ein weitere Möglichkeit, Parameterwerte zwischen Web-Seiten(-Aufrufen) zu übermitteln, besteht darin, Namens-Wert-Paare an eine URL anzuhängen; diese stehen der aufgerufenen Web-Seite als globale Variable zur Verfügung. Wenn in Parameternamen oder -werten Blanks vorkommen, müssen diese geeignet codiert werden. Das folgende Beispiel illustriert diese Technik. Von der Willkommens-Seite aus kann der Kunde jetzt zwei weitere Web-Seiten zum Ein- und Auszahlen von Geld und zum Erstellen eines Kontoauszugs anspringen. Beim Ein-/Auszahlen werden die beiden Möglichkeiten durch Abfragen des im Formular gewählten Buttons unterschieden. In beiden Fällen werden außerdem die SQL-Änderungen auf den beiden Tabellen Account und History zu einer atomaren Transaktion zusammengefaßt. Dies erfordert ein explizites OCI_DEFAULT-Flag in den entsprechenden ociexecute-Aufrufen (die Voreinstellung von PHP selbst ist der sog. "Autocommit"-Modus, bei dem jede einzelne SQL-Anweisung implizit und sofort ein "commit work" nach sich zieht). Das Erstellen eines Kontoauszugs illustriert die Darstellung der in einer Cursor-Schleife ermittelten Anfrageresultate in einer HTML-Tabelle.

Beispiel: willkommen3.php

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML3.2//EN">
<head><title> Kundenbegrüßung </title></head>
<?php
error_reporting(E_ERROR | E_PARSE);
$db = "(DESCRIPTION =
       (ADDRESS_LIST =(ADDRESS = (PROTOCOL = TCP)(HOST = tokyo)(PORT = 1521)))
       (CONNECT_DATA =(SERVICE_NAME = tokyo.world)))";
$connection = ocilogon ("lehre40", "leere4zig", $db);
$ocistatus = ocierror($stmt);
echo $ocistatus["code"] . " " . $ocistatus["message"] . "\n";
?>
<body>
   <h1> Willkommen
        <?php
          $sqlstring = "SELECT * FROM CUSTOMER WHERE IPNUMBER = '"
                       . $REMOTE_ADDR . "'";
          $stmt = ociparse ($connection, $sqlstring);
          $ocistatus = ocierror($stmt);
          echo $ocistatus["code"] . " " . $ocistatus["message"] . "\n";
          ociexecute ($stmt);
          $ocistatus = ocierror($stmt);
          echo $ocistatus["code"] . " " . $ocistatus["message"] . "\n";
          ocifetch ($stmt);
          $customername = ociresult ($stmt, "NAME");
          $customeripnumber = ociresult ($stmt, "IPNUMBER");
          $customercity = ociresult ($stmt, "CITY");
          if ($customername != "")
             {echo ", " . $customername . ", ";};
         ?>
        bei der Bank Primitivo <br>
   </h1>
Ihre IP-Nummer ist <?php echo $REMOTE_ADDR ?>,
und Sie benutzen den Browser <?php echo $HTTP_USER_AGENT ?>.<br>
<?php printf("Heute ist "); printf("%s", date(DdFY)); ?> <br>
<?php printf("Die aktuelle Uhrzeit ist ");printf("%s:%s",date(H),date(i)); ?><br>

<?php if ($customername != "")
         {echo "Wie ist das Wetter in " . $customercity . "?<br>";
          echo "<br><br>";
          echo "Was können wir heute für Sie tun?<br>";
          $nexturl = "einaus.php?customername=" . urlencode($customername);
          echo "<a href=" . $nexturl . "> Ein- oder Auszahlung</a><br>";
          $nexturl = "auszug.php?customername=" . urlencode($customername);
          echo "<a href=" . $nexturl . "> Kontoauszug</a><br>";
         }
```

```
      else {
           echo "<form method=post action=willkommen3.php>";
           echo "<br> Bitte geben Sie Ihren Namen und Ihren Wohnort ein. <br>";
           echo "Ihr Name: <input type=text name=inputname><br>";
           echo "Ihr Wohnort: <input type=text name=inputcity
                              value=Saarbruecken><br>";
           echo "<input type=submit name=inputenter
                  value=\"Hier hast Du's\"><br>";
           echo "</form>";
           if (($inputenter) and ($inputname != ""))
             {
              $sqlinsertstring = "INSERT INTO CUSTOMER VALUES ('"
                                 . $inputname . "', '" . $REMOTE_ADDR . "', '"
                                 . $inputcity . "')";
              $stmt = ociparse ($connection, $sqlinsertstring);
              ociexecute ($stmt, OCI_DEFAULT);
              $ocistatus = ocierror($stmt);
              echo $ocistatus["code"] . " " . $ocistatus["message"] . "\n";
              $insert1ok = 0;
              if (($ocistatus == "") and (ocirowcount($stmt) == 1))
                 {$insert1ok = 1;};
              $sqlinsertstring = "INSERT INTO ACCOUNT VALUES ('"
                                 . $inputname . "', 0)";
              $stmt = ociparse ($connection, $sqlinsertstring);
              ociexecute ($stmt, OCI_DEFAULT);
              $ocistatus = ocierror($stmt);
              echo $ocistatus["code"] . " " . $ocistatus["message"] . "\n";
              $insert2ok = 0;
              if (($ocistatus == "") and (ocirowcount($stmt) == 1))
                 {$insert2ok = 1;};
              if ($insert1ok and $insert2ok)
                 {ocicommit ($connection);}
              else
                 {ocirollback ($connection);};
             };
           };
?>
<?php ocifreestatement ($stmt); ocilogoff ($connection); ?>
</body> </html>
```

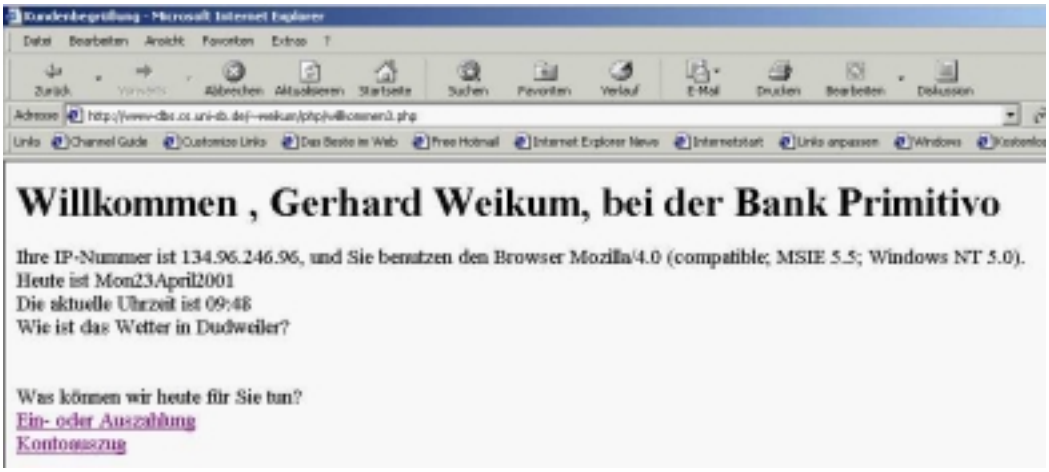
einaus.php

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML3.2//EN">
<head><title> Ein-/Auszahlung </title></head>
<?php
error_reporting(E_ERROR | E_PARSE);
$db = "(DESCRIPTION =
       (ADDRESS_LIST =(ADDRESS = (PROTOCOL = TCP)(HOST = tokyo)(PORT = 1521)))
       (CONNECT_DATA =(SERVICE_NAME = tokyo.world)))";
$connection = ocilogon ("lehre40", "leere4zig", $db);
$ocistatus = ocierror($stmt);
echo $ocistatus["code"] . " " . $ocistatus["message"] . "\n";    ?>
<body>
   <h1> Lieber
        <?php $customername = urldecode($customername);
              echo " " . $customername . ", "; ?>
        Hier können Sie Geld auf Ihr Konto einzahlen oder
        von Ihrem Konto auszahlen. <br><br>
   </h1>
<?php
$nexturl = $nexturl = "einaus.php?customername=" . urlencode($customername);
echo "<form method=post action=" . $nexturl . ">";
echo "<br> Bitte geben Sie den Betrag ein und drücken Sie auf
           Einzahlung oder Auszahlung. <br>";
echo "Betrag: <input type=text name=inputamount value=0><br>";
echo "<input type=submit name=inputdeposit value=Einzahlung><br>";
echo "<input type=submit name=inputwithdraw value=Auszahlung><br>";
echo "</form>";
echo $inputamount . $inputdeposit . $inputwithdraw;
$inputamount = (int) $inputamount;
echo $inputamount;
if ($inputwithdraw != "") {$inputamount = 0 - $inputamount;};
#echo $inputamount; /* zu Testzwecken */
if ((($inputwithdraw != "") or ($inputdeposit != "")) and ($inputamount != 0)) {
   $sqlstring = "INSERT INTO HISTORY VALUES ('"
                      . $customername . "', SYSDATE, " . $inputamount . ")";
   $stmt = ociparse ($connection, $sqlstring);
   $insertok = 0;
   ociexecute ($stmt, OCI_DEFAULT);
   $ocistatus = ocierror($stmt);
   echo $ocistatus["code"] . " " . $ocistatus["message"] . "\n";
   if (($ocistatus == "") and (ocirowcount($stmt) == 1)) {$insertok = 1;};
   $sqlstring = "UPDATE ACCOUNT SET BALANCE = BALANCE + ("
                      . $inputamount . ") WHERE NAME = '" . $customername . "'";
   $stmt = ociparse ($connection, $sqlstring);
   $updateok = 0;
   ociexecute ($stmt, OCI_DEFAULT);
   $ocistatus = ocierror($stmt);
   echo $ocistatus["code"] . " " . $ocistatus["message"] . "\n";
   if (($ocistatus == "") and (ocirowcount($stmt) == 1)) {$updateok = 1;};
   if ($insertok and $updateok) {
       ocicommit ($connection);
       echo "<br><br>Ihr Auftrag wurde ausgeführt.<br>";
       }
   else {
       ocirollback ($connection);
       echo "<br><br>Ihr Auftrag konnte leider nicht ausgeführt werden.<br>";
       };
   };
?>
<br><br>
<?php
$nexturl = "willkommen3.php?customername=" . urlencode($customername);
echo "<a href=" . $nexturl . "> Zurück zur Hauptseite</a>";
?>
<?php ocifreestatement ($stmt); ocilogoff ($connection); ?>
</body></html>
```

auszug.php

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML3.2//EN">
<head><title> Kontoauszug </title></head>
<?php
error_reporting(E_ERROR | E_PARSE);
$db = "(DESCRIPTION =
       (ADDRESS_LIST =(ADDRESS = (PROTOCOL = TCP)(HOST = tokyo)(PORT = 1521)))
       (CONNECT_DATA =(SERVICE_NAME = tokyo.world)))";
$connection = ocilogon ("lehre40", "leere4zig", $db);
$ocistatus = ocierror($stmt);
echo $ocistatus["code"] . " " . $ocistatus["message"] . "\n";
?>
<body>
   <h1> Lieber
        <?php $customername = urldecode($customername);
              echo " " . $customername . ", "; ?>
        Ihre bisherigen Buchungen sind: <br><br>
   </h1>
<?php
$sqlstring = "SELECT NAME, TO_CHAR(BOOKINGDATE, 'DY DD MONTH YYYY, HH24:MI:SS')
              AS PRINTABLEDATE, AMOUNT
              FROM HISTORY WHERE NAME = '"
             . $customername . "'" . "ORDER BY BOOKINGDATE DESC";
$stmt = ociparse ($connection, $sqlstring);
$ocistatus = ocierror($stmt);
echo $ocistatus["code"] . " " . $ocistatus["message"] . "\n";
ociexecute ($stmt);
$ocistatus = ocierror($stmt);
echo $ocistatus["code"] . " " . $ocistatus["message"] . "\n";
echo "<table border cols=2>";
echo "<tr> <th>Datum     </th> <th>Betrag     /></tr>";
while (ocifetch ($stmt)) {
   $bookingdate = ociresult ($stmt, "PRINTABLEDATE");
   $amount = ociresult ($stmt, "AMOUNT");
   echo "<tr>";
   echo "<td>" . $bookingdate . "     </td>";
   echo "<td>" . $amount . "     </td>";
   echo "</tr>";
};
echo "</table><br><br>";
?>
<br><br>
<?php
$nexturl = "willkommen3.php?customername=" . urlencode($customername);
echo "<a href=" . $nexturl . "> Zurück zur Hauptseite</a>";
?>
<?php ocifreestatement ($stmt); ocilogoff ($connection); ?>
</body></html>
```

Hier sind typische Resultate dieser 3 PHP-Seiten:



Willkommen , Gerhard Weikum, bei der Bank Primitivo

Ihre IP-Nummer ist 134.96.246.96, und Sie benutzen den Browser Mozilla/4.0 (compatible; MSIE 5.5; Windows NT 5.0).
Heute ist Mon23April2001
Die aktuelle Uhrzeit ist 09:48
Wie ist das Wetter in Dudweiler?

Was können wir heute für Sie tun?
Ein- oder Auszahlung
Kontoauszug

Lieber Gerhard Weikum, Hier können Sie Geld auf Ihr Konto einzahlen oder von Ihrem Konto auszahlen.

Bitte geben Sie den Betrag ein und drücken Sie auf Einzahlung oder Auszahlung.
Betrag: 50
Einzahlung
Auszahlung

Ihr Auftrag wurde ausgeführt.

Zurück zur Hauptseite

Lieber Gerhard Weikum, Ihre bisherigen Buchungen sind:

Datum	Betrag
MO 23 APRIL 2001, 09:55:13	-50
MO 23 APRIL 2001, 09:55:03	300
MO 23 APRIL 2001, 09:54:57	-50
MO 23 APRIL 2001, 09:54:45	200
MO 23 APRIL 2001, 09:54:39	100

Zurück zur Hauptseite

6.5 Cookies und Sitzungen in Web-Anwendungen

Eine der Hauptschwierigkeiten bei der Entwicklung von dynamischen Web-Anwendungen besteht darin, interaktive Sitzungen auf dem eigentlich zustandslosen HTTP-Protokoll zu implementieren. An sich würde nämlich jeder HTTP-Aufruf keinerlei Information über die vorherigen Aufrufe desselben Clients vorfinden. Zu diesem Zweck gibt es eine Reihe von Optionen:

1) Speichern von Sitzungsinformationen in HTML-Formularen (ggf. in zusätzlichen - für den Benutzer unsichtbaren - Feldern).
2) Anhängen von Sitzungsinformationen an URLs. Bei dieser wie auch bei der ersten Technik werden also alle diese Informationen immer wieder zwischen Client und Web-Server hin- und hergeschickt.
3) Verwendung sog. Cookies, bei denen der Web-Server im Header der Antwort auf einen GET-Aufruf spezifische Information über den Client (häufig nur die Identifikation des Clients) zurückliefert. Wenn der Browser des Clients so eingestellt ist, daß er Cookies akzeptiert, wird diese Information bei jedem nachfolgenden HTTP-Aufruf an denselben Web-Server automatisch mitgeliefert und ist vom Web-Server abfragbar.
4) Verwendung von PHP-Sitzungsvariablen, die in Files auf der Web-Server-Seite gespeichert und vom aufgerufenen PHP-Skript registriert, abgefragt und geändert werden. Dabei wird die zugehörige Session-Id mittels Cookies identifiziert.
5) Abspeichern von Sitzungsinformationen in einer Datenbank. Dabei braucht man auch eine Sitzungs- oder Kunden-Identifikation, um die zugehörigen Daten zu ermitteln; diese Identifikation liefern z.B. Cookies. Die Optionen 4 und 5 haben den Vorteil, daß außer einer - ggf. leicht verschlüsselbaren - Identifikation keinerlei Sitzungsdaten zum Client geschickt werden müssen, so daß der Client auch keine Möglichkeit hat, solche Daten zu manipulieren.

Beispiel: sitzung.php

```
<?php
session_start ();  /* muss vor dem HTML-Header stehen */
error_reporting(E_ERROR | E_PARSE);
if (!$willkommencounter) {
   $erstemal = 1;
   $cookie_name = "willkommencounter"; $cookie_value = "schon wieder hier";
   $cookie_expire = ""; $cookie_domain = "www-dbs.cs.uni-sb.de";
   setcookie ($cookie_name, $cookie_value, $cookie_expire, "/", $cookie_domain, 0);
   $visitcounter = 1;
   }
else $erstemal = 0;
?>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML3.2//EN">
<head><title> PHP-Beispiel für Cookies und Session-Variable </title></head>
<body>
   <h1> Willkommen bei der Bank Primitivo <br></h1>
Ihre IP-Nummer ist <?php echo $REMOTE_ADDR ?>,
und Sie benutzen den Browser <?php echo $HTTP_USER_AGENT ?>.<br>
<?php printf("Heute ist "); printf("%s", date(DdFY)); ?><br>
<?php printf("Die aktuelle Uhrzeit ist ");printf("%s:%s",date(H),date(i)); ?><br>
<?php if (!($erstemal)) {echo "Sie waren doch eben schon mal hier!<br>";}; ?>
<?php if (!$erstemal) {
         $visitcounter++;
         session_register("visitcounter");
         if ($visitcounter > 1)
            {echo "Sie sind heute schon das " . $visitcounter . ". Mal hier!";};
         };
?>
</body></html>
```

Ergänzende Literatur zu Kapitel 6:

J. Melton, A. Simon, Understanding the New SQL: A Complete Guide, Morgan Kaufmann, 1993

Oracle8i Pro*C/C++ Precompiler Programmer's Guide, Release 8.1.5, innerhalb der UdS zugreifbar über http://niniveh.cs.uni-sb.de/docs/Oracle8i/server.815/a68022/toc.htm

J.W. Schmidt, F. Matthes, The DBPL Project: Advances in Modular Database Programming, Information Systems Vol. 19 No. 2, 1994

G. Saake, K.-U. Sattler, Datenbanken und Java, dpunkt-Verlag, 2000

S. Spainhour, R. Eckstein, Webmaster in a Nutshell, O'Reilly, 1999

J.F. Kurose, K.W. Ross, Computer Networking: a Top-down Approach Featuring the Internet, Addison-Wesley, 2001

Diverse Online-Tutorials über Web-Technologien, http://www.w3schools.com

PHP Online-Dokumentation: http://www.php.net/docs.php (oder http://www.php-center.de/manual/index.htm)

J. Krause, PHP4: Grundlagen und Profiwissen, Hanser-Verlag, 2000

J. Castagnetto, H. Rawat, S. Schumann, C. Scollo, D. Veliath, Professional PHP Programming, Wrox Press Ltd., 2000

Rasmus Lerdorf, Introduction to PHP, Online-Tutorial über PHP, http://conf.php.net/ac2

Thies C. Arntzen, Making Efficient Use of Oracle8, Online-Tutorial über die PHP-Oracle8i-Kopplung, http://conf.php.net/oci