Kapitel 4: 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
   → "Embedded SQL" (ESQL)

Beispiel für eine Datenbankprogrammiersprache (à la DBPL):

```
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;
```

Kernproblem bei der SQL−Einbettung in konventionelle Programmiersprachen:

Abbildung von Tupelmengen auf die Datentypen der Wirtsprogrammiersprache

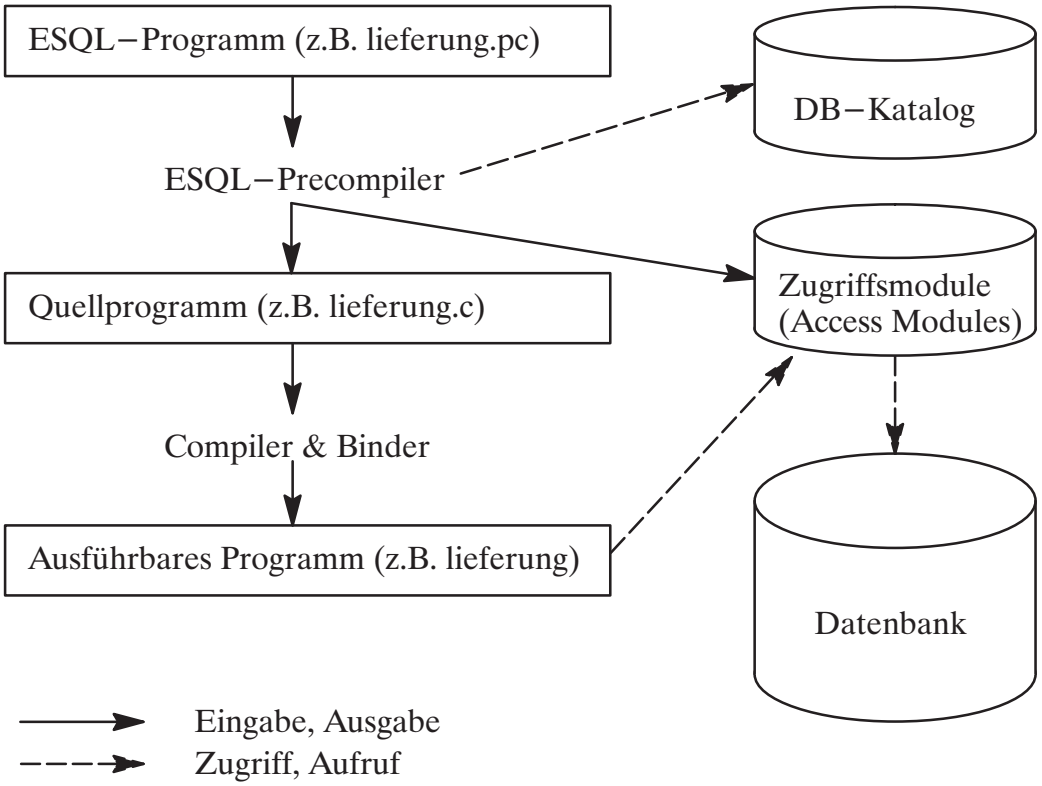
Realisierte Lösung:

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

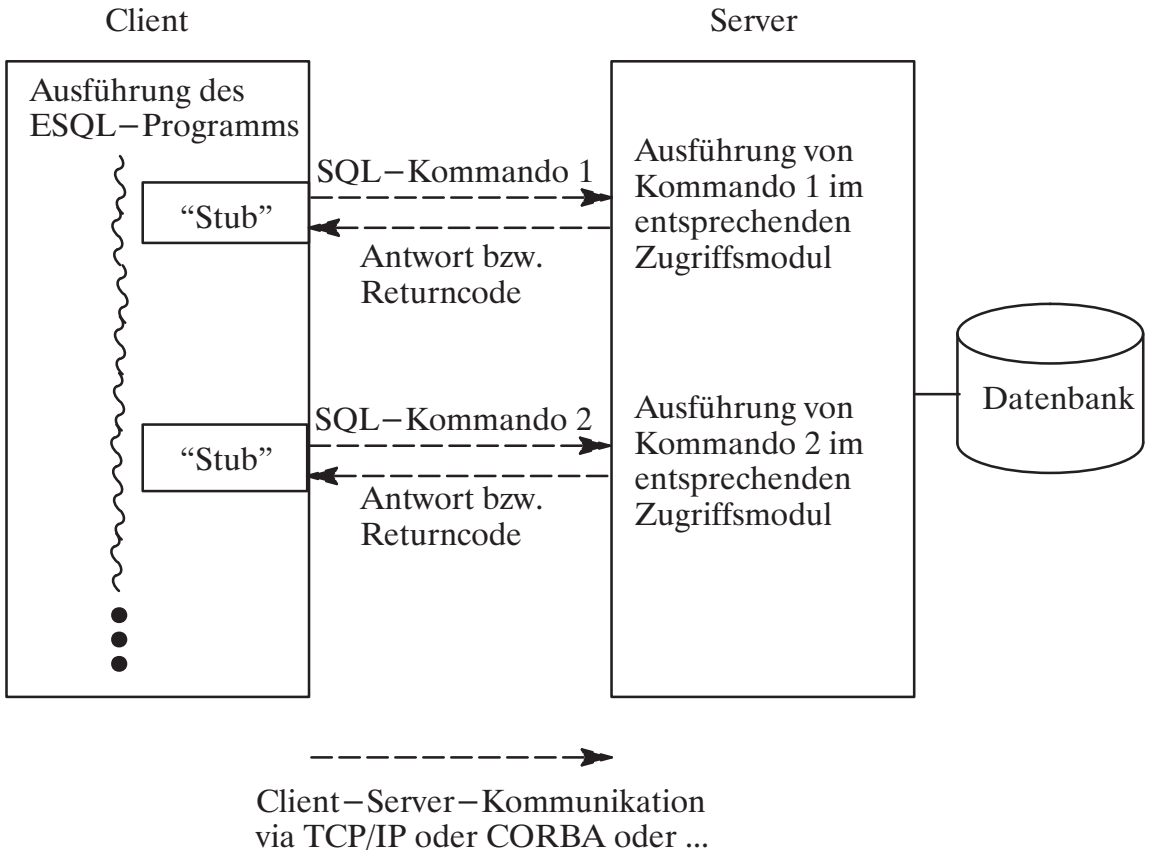
+

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 */
```

Dynamisches ESQL

Zweck:
Einbettung von SQL−Anweisungen, deren Struktur erst zur Laufzeit des Wirtsprogramms bekannt ist.

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

Komplexer Fall: Vorgehensweise bei dynamisch festgelegtem Resultatschema und/oder dynamisch festgelegten Eingabeparametern:

Beispiel (”dynamischer Anteil” ist unterstrichen):

```
SELECT *
FROM Bestellungen, Produkte WHERE Bestellungen.PNr = Produkte.PNR
AND ( ( Bez LIKE :bez1 OR Bez LIKE :bez2 ) AND Lagerort = :lort AND Vorrat < :vorr )
ORDER BY Monat DESC, Tag DESC
```

Vorgehensweise:

1) Deklaration des SQLDA−Datentyps für ”Descriptor−Areas” (Typbeschreibung des Resultats und/oder der Eingabeparameter)
2) Aufbau der SQL−Anweisung als Zeichenkette
3) Allokation einer ”Bind−Descriptor−Area” für die Beschreibung der Eingabeparameter
4) Allokation einer ”Select−Descriptor−Area” für die Beschreibung des Resultats
5) Dynamische Precompilation: PREPARE DynQuery FROM :sqltext
6) DECLARE Iterator CURSOR FOR DynQuery

7) Beschreibung der Eingabeparameter abfragen: DESCRIBE BIND VARIABLES FOR DynQuery INTO bind−desciptor−area−ptr
8) Puffer für die Eingabeparameter allozieren, in der Bind−Descriptor−Area eintragen und Puffer mit Werten füllen

9) Auswertung der Eingabeparamter und Vorbereitung der ”Cursor”−Schleife OPEN Iterator USING DESCRIPTOR bind−descriptor−area−ptr

10) Beschreibung der Resultatsattribute abfragen: DESCRIBE [SELECT LIST FOR] DynQuery INTO select−descriptor−area−ptr
11) Puffer für die Resultatsattribute allozieren und in der Select−Descriptor−Area eintragen
12) Gewünschte Resultatstypen in die Select−Descriptor−Area eintragen

13) FETCH Iterator USING DESCRIPTOR select−descriptor−area−ptr
14) Resultatsattribute in den Übergabepuffern entsprechend ihres Typs verarbeiten
15) Wiederholung der Schritte 13 und 14 für jedes Resultatstupel
16) CLOSE Iterator

17) bei Bedarf Wiederholung der Schritte 8 bis 16 mit neuen Eingabeparametern (ohne erneute Precompilation)

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*/
```

Beispiel für den komplexen Fall:

```
/***  Programm zum Ausdrucken der Bestellungen fuer eine Liste von Produkten     ***/
/***  Eingabe: Liste von Produktbezeichnungen                                    ***/
/***  Ausgabe: Bestellungen dieser Produkte, absteigend sortiert nach Datum      ***/
/***  Erschwerende Annahme: Das Resultatschema ist erst zur Laufzeit bekannt !   ***/

/***  Vereinfachung:  keine Wirtsprogrammvariable als Eingabeparameter           ***/
/***                  => keine Bind-Descriptor-Area notwendig                    ***/

#include <stdio.h>
#include <string.h>
#define TRUE 1
#define FALSE 0
#define MAXPRODUKTE 10
/* Die Bestellungen umfassen maximal 10 Produkte */

/* Dimensionierung der SQLDA */
#define MAXATTR 20
#define MAXNAMELEN 30
/* Datentypen fuer Oracle-Attributwerte */
#define CHAR_TYPE 1
#define NUMBER_TYPE 2
/* Datentypen fuer Resultate in den Uebergabepuffern */
#define INT_TYPE 3
#define CCHAR_TYPE 5

EXEC SQL INCLUDE SQLCA; /* Importieren der SQLCA */
EXEC SQL INCLUDE SQLDA; /* Importieren der SQLDA-Typdefinition */

/* Der Precompiler erzeugt hier die folgende Typdefinition für eine Descriptor Area:
struct SQLDA {
    long    N;      /* Descriptor size in number of entries          */
    char  **V;      /* Ptr to Arr of addresses of main variables     */
    long   *L;      /* Ptr to Arr of lengths of buffers              */
    short  *T;      /* Ptr to Arr of types of buffers                */
    short **I;      /* Ptr to Arr of addresses of indicator vars     */
    long    F;      /* Number of variables found by DESCRIBE         */
    char  **S;      /* Ptr to Arr of variable name pointers          */
    short  *M;      /* Ptr to Arr of max lengths of var. names       */
    short  *C;      /* Ptr to Arr of current lengths of var. names   */
    char  **X;      /* Ptr to Arr of ind. var. name pointers         */
    short  *Y;      /* Ptr to Arr of max lengths of ind. var. names  */
    short  *Z;      /* Ptr to Arr of cur lengths of ind. var. names  */
  };
 typedef struct SQLDA SQLDA;                                           */

/* Importieren der Oracle-Hilfsfunktion sqlald und sqlnul */
extern SQLDA *sqlald ( int MaxAttr, int MaxNameLen, int MaxIndNameLen );
extern void sqlnul ( short *AttrTypeInfo, short *AttrType, short *NullStatus );
```

```
main () {
EXEC SQL BEGIN DECLARE SECTION;
    /* Achtung: Keine Wirtsprogrammvariablen fuer das Anfrageresultat */
   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;
SQLDA        *selda; /* Zeiger auf (zu allozierende) SQLDA−Struktur */
char         *s;
short        nullstatus;

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 * ");
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 *
   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                                          */

/* Allokation einer Select-Descriptor-Area (Datentyp SQLDA) */
selda = sqlald (MAXATTR, MAXNAMELEN, 0);

EXEC SQL PREPARE DynQuery FROM :sqltext;
EXEC SQL DECLARE BestIterator CURSOR FOR DynQuery;
EXEC SQL OPEN BestIterator;
EXEC SQL DESCRIBE SELECT LIST FOR DynQuery INTO selda;
selda->N = selda->F; /* Anzahl der Resultatsattribute */
/* Weitere Vorbereitungen fuer die SQLDA-Struktur */
for (i=0; i < selda->F; i++) {
   /* Ruecksetzen des "Nullwert-Flags" im Typcode */
   sqlnul (&(selda->T[i]), &(selda->T[i]), &nullstatus);
   /* Setzen der Pufferlaengen und Typcodes fuer die Resultatsattribute */
   switch (selda->T[i])
      {
      case CHAR_TYPE:
         selda->L[i]++; /* Extrabyte fuer String-Terminator */
         selda->T[i] = CCHAR_TYPE; /* gewuenschter Resultattyp */
         break;
      case NUMBER_TYPE:
         /***  Achtung:   Dies ist eine Loesung fuer den Oracle-Datentyp INTEGER.
                Allgemeinere NUMBER-Datentypen werden hier nicht behandelt.    ***/
         selda->L[i] = sizeof(int);
         selda->T[i] = INT_TYPE;
         /* Allgemeinere, aber umstaendlichere Alternative:
         selda->L[i] = 51; * lang genug fuer alle NUMBER-Typen *
         selda->T[i] = CCHAR_TYPE; * gewuenschter Resultattyp *
         */
         break;
      default:
         printf ("\n*** Unerwarteter Datentyp %d ***\n", selda->T[i]);
      }; /*switch*/
   /* Allokation der Uebergabepuffer fuer die Resultatsattribute */
   selda->V[i] = malloc(selda->L[i]);
   /* Allokation der Nullwertindikatoren */
   selda->I[i] = malloc(sizeof(short));
   }; /*for*/
```

```
found = TRUE;
while (found)
   {
   EXEC SQL FETCH BestIterator USING DESCRIPTOR selda;
   if (sqlca.sqlcode == 0)
      {
      for (i=0; i < selda->N; i++)
         {
         /* Attributname ausgeben */
         s = selda->S[i]; /* i-ter Attributname */
         s[selda->C[i]] = '\0'; /* Konvertierung in C-String */
         printf ("%s: ", s);
         /* Verzweigung nach Attributdatentyp */
         switch (selda->T[i])
            {
            case CHAR_TYPE: ;
            case CCHAR_TYPE:
               s = selda->V[i]; /* i-ter Attributwert */
               s[selda->L[i]] = '\0'; /* Terminator fuer C-String */
               printf ("%s ", s);
               break;
            case INT_TYPE:
               printf ("%d ", *((int*) selda->V[i]));
               break;
            default:
               printf ("\n*** Unerwarteter Datentyp %d ***\n", selda->T[i]);
            }; /*switch*/
         }; /*for*/
      printf ("\n");
      }
   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 komplexe 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:

ACCESSIBLE_TABLES

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

ACCESSIBLE_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	ACCESSIBLE_TABLES	OWNER	CHAR	30		1	
SYS	ACCESSIBLE_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
```

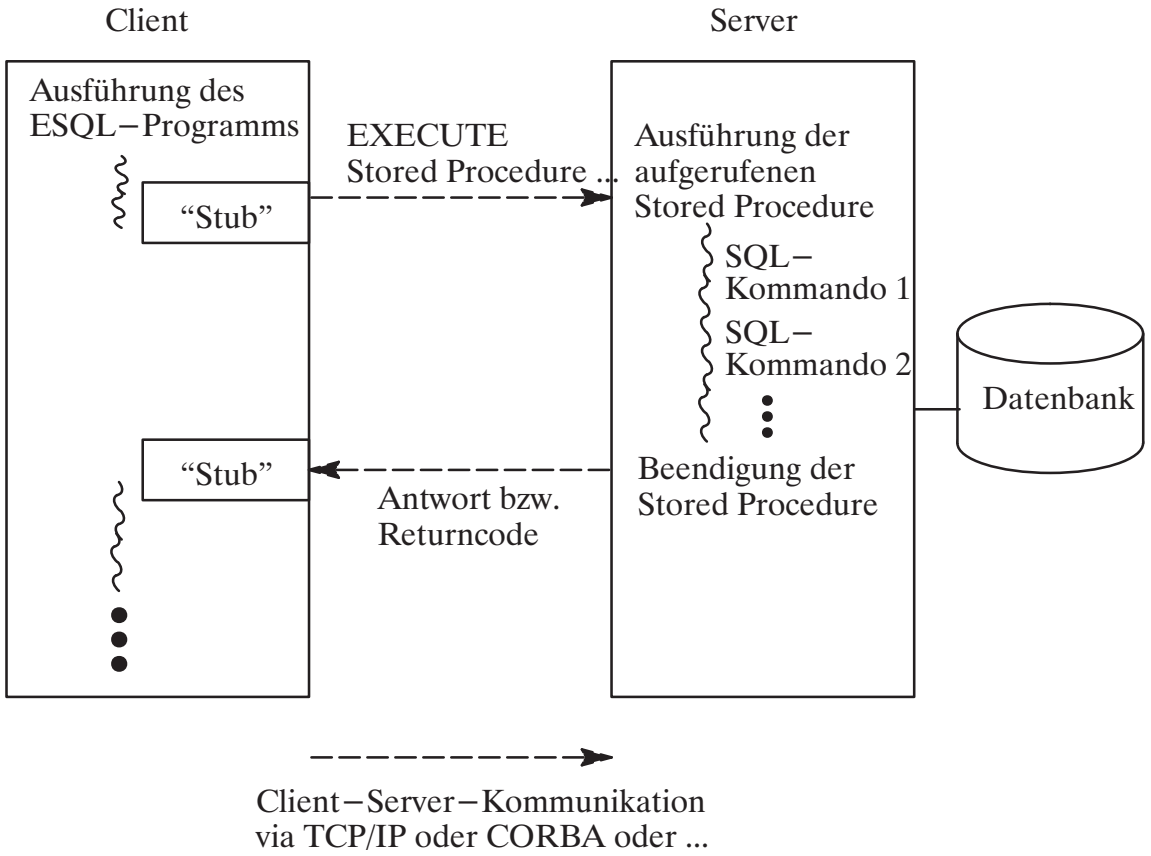
Beispiel:

```
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:



Entwicklung Internet−fähiger Anwendungen
mit dem Oracle Web Application Server Toolkit

Prinzip:
Programmierung von PL/SQL−Packages (Java oder Perl oder ...)
unter Verwendung von Bibliotheksfunktionen des Standard−Packages “htp”
zum Aufbau dynamischer HTML−Seiten

Beispiel:
ein kleines Package zur Verwaltung der Buchausleihe einer “Mini−Bibliothek”

```
DROP TABLE bib_table;
CREATE TABLE bib_table
(
        bid       NUMBER  PRIMARY KEY,
        verfasser VARCHAR2(200),
        titel     VARCHAR2(200),
        verlag    VARCHAR2(200),
        jahr      NUMBER,
        datum     VARCHAR2(20),
        status    VARCHAR2(15),
        ausleiher VARCHAR2(200),
        CHECK (status IN ( 'bestellt', 'geliefert', 'vergriffen', 'verschwunden'))
);

set define off

DROP PACKAGE bib_user;
CREATE PACKAGE bib_user IS
   PROCEDURE home;
   PROCEDURE retrieve_form;
   PROCEDURE build_query (
                        verfasser IN VARCHAR2,
                        titel     IN VARCHAR2,
                        search_mode IN VARCHAR2);
   PROCEDURE retrieve  (v       IN VARCHAR2,
                        t       IN VARCHAR2);
   PROCEDURE lend_form
                        (id      IN INT,
                        titel    IN VARCHAR2);
   PROCEDURE update_ausleiher (
                        id      IN INT,
                        name    IN VARCHAR2,
                        modi    IN VARCHAR2,
                        titel   IN VARCHAR2);
   PROCEDURE back_form;
   PROCEDURE back (    name    IN VARCHAR2);
END;
/
show errors;
```

```
CREATE PACKAGE BODY bib_user IS

  PROCEDURE home IS
    BEGIN
      htp.htmlOpen;  htp.headOpen;  htp.htitle ('User–Homepage');  htp.headClose;
      htp.bodyOpen;  htp.line;  htp.nl;
      htp.print ('Recherche Programm für die Handbibliothek des Lehrstuhls Prof. Weikum'); htp.nl;
      htp.anchor (owa_util.get_owa_service_path||'bib_user.retrieve_form',  'Buch suchen / ausleihen'); htp.nl;
      htp.anchor (owa_util.get_owa_service_path||'bib_user.back_form', 'Ausleihe löschen'); htp.nl;
      htp.anchor ('/bib–doc/bib_help.html', 'Anleitung / Hilfe'); htp.line;
      htp.address ('Markus Sinnwell, Jan. 97'); htp.bodyClose;
      htp.htmlClose;
    END;

  PROCEDURE retrieve_form IS
    BEGIN
      htp.htmlOpen; htp.headOpen;  htp.htitle ('Suchen eines Eintrags');  htp.headClose;
      htp.bodyOpen; htp.line;
      htp.formOpen (owa_util.get_owa_service_path||'bib_user.build_query', 'POST');
      htp.formSelectOpen ('search_mode', 'Suchmodus: ');
        htp.formSelectOption ('substring');
        htp.formSelectOption ('exact match');
      htp.formSelectClose; htp.nl;
      htp.tableOpen; htp.tableRowOpen;
        htp.tableData ('Verfasser');  htp.tableData (htf.formtext ('verfasser'));
      htp.tableRowClose;
      htp.tableRowOpen;
        htp.tableData ('Titel'); htp.tableData (htf.formtext ('titel'));
      htp.tableRowClose; htp.tableClose; htp.line;
     htp.formSubmit (NULL, 'Anfrage starten');  htp.formReset ('Formular löschen');  htp.formClose; htp.line;
      htp.anchor (owa_util.get_owa_service_path||'bib_user.home','Zurück zur Home–Page');
      htp.bodyClose;  htp.htmlClose;
    END;

  PROCEDURE build_query (verfasser IN VARCHAR2, titel IN VARCHAR2, search_mode IN VARCHAR2)
    IS
      v  VARCHAR2(200);
      t  VARCHAR2(200);
    BEGIN
      IF verfasser IS NULL THEN
        v := '%';
      ELSIF search_mode = 'exact match' THEN  v := lower(verfasser);
      ELSE  v := '%'||lower(verfasser)||'%';
      END IF;
      IF titel IS NULL THEN
        t := '%';
      ELSIF search_mode = 'exact match' THEN
        t := lower(titel);
      ELSE
        t := '%'||lower(titel)||'%';
      END IF;
      retrieve (v, t);
    END;
```

```
PROCEDURE retrieve  (v  IN VARCHAR2,
                     t  IN VARCHAR2) IS
   CURSOR tupel IS   SELECT  bid, titel, verfasser, status, ausleiher
                     FROM  bib_table
                     WHERE lower(verfasser) LIKE v
                     AND   lower(titel) LIKE t
                     ORDER BY titel;
   id_out       INT;
   titel_out    VARCHAR2(200);
   verfasser_out VARCHAR2(200);
   status_out   VARCHAR2(15);
   ausleiher_out VARCHAR2(200);
   BEGIN
      htp.htmlOpen;  htp.headOpen;  htp.htitle ('Ergebnis der Anfrage');  htp.headClose;
      htp.bodyOpen; htp.line;  htp.tableOpen ('border');
      htp.tableRowOpen;
         htp.tableHeader ('Titel');  htp.tableHeader ('Verfasser');  htp.tableHeader ('Ausleiher');
      htp.tableRowClose;

      OPEN tupel;
      LOOP
         FETCH tupel INTO id_out, titel_out, verfasser_out, status_out, ausleiher_out;
         IF tupel%FOUND THEN
            htp.tableRowOpen; htp.tableData (titel_out); htp.tableData (verfasser_out);
            IF status_out = 'geliefert' THEN
               IF ausleiher_out IS NULL THEN
                  htp.tableData(htf.anchor(owa_util.get_owa_service_path || 'bib_user.lend_form?id='||id_out||
                                 '&titel='||translate(titel_out,' ', '+'), htf.em ('ausleihen')));
               ELSE  htp.tableData (ausleiher_out);
               END IF;
            ELSE htp.tableData(status_out);
            END IF;
            htp.tableRowClose;
         ELSE EXIT;
         END IF;
      END LOOP;
      htp.tableClose; htp.line;
      htp.anchor (owa_util.get_owa_service_path||'bib_user.home','Zurück zur Home–Page');
      htp.bodyClose; htp.htmlClose;
   END;


PROCEDURE lend_form (id IN INT, titel IN VARCHAR2) IS
     name VARCHAR2(200);
   BEGIN
      htp.htmlOpen;  htp.headOpen;  htp.htitle ('Ausleihe registrieren');  htp.headClose;
      htp.bodyOpen; htp.line;  htp.print ('Das Buch '||htf.em(titel)||' ausleihen.');
      htp.formOpen (owa_util.get_owa_service_path||'bib_user.update_ausleiher', 'POST');
      htp.print ('Name des Ausleihers: ');  htp.formtext ('name');
      htp.formhidden ('id', id);  htp.formhidden ('modi', 'lend');  htp.formhidden ('titel', titel);  htp.line;
    htp.formSubmit (NULL, 'Ausleihe registrieren');  htp.formReset ('Formular löschen');  htp.formClose; htp.line;
      htp.anchor (owa_util.get_owa_service_path||'bib_user.home','Zurück zur Home–Page');
      htp.bodyClose;  htp.htmlClose;
   END;
```

```
PROCEDURE update_ausleiher (id IN INT, name IN VARCHAR2, modi IN VARCHAR2,
                            titel IN VARCHAR2) IS
  BEGIN
    htp.htmlOpen; htp.headOpen;
    IF modi = 'lend' THEN
      htp.htitle ('Bestätigung der Ausleihe');  htp.headClose; htp.bodyOpen;  htp.line;
      UPDATE bib_table SET ausleiher = name WHERE bid = id;
      htp.print ('Das Buch '||htf.em(titel)||' ist jetzt von '||htf.em(name)||' ausgeliehen');
    ELSE
      htp.htitle ('Bestätigung der Rückgabe'); htp.headClose; htp.bodyOpen;  htp.line;
      UPDATE bib_table SET ausleiher = NULL WHERE bid = id;
      htp.print ('Das Buch '||htf.em(titel)||' wurde von '||htf.em(name)||' zurückgegeben');
    END IF;
    htp.line; htp.anchor (owa_util.get_owa_service_path||'bib_user.home','Zurück zur Home–Page');
    htp.bodyClose;  htp.htmlClose;
  END;


PROCEDURE back_form IS
  BEGIN
    htp.htmlOpen; htp.headOpen; htp.htitle ('Rückgabe eines Buches'); htp.headClose;
    htp.bodyOpen;  htp.line;
    htp.formOpen (owa_util.get_owa_service_path||'bib_user.back', 'POST');
    htp.print ('Name des Ausleihers:');  htp.formtext ('name');  htp.line;
    htp.formSubmit (NULL, 'Ausgeliehene Bücher anzeigen');  htp.formReset ('Formular löschen');
    htp.formClose;  htp.line;
    htp.anchor (owa_util.get_owa_service_path||'bib_user.home','Zurück zur Home–Page');
    htp.bodyClose;  htp.htmlClose;
  END;
```

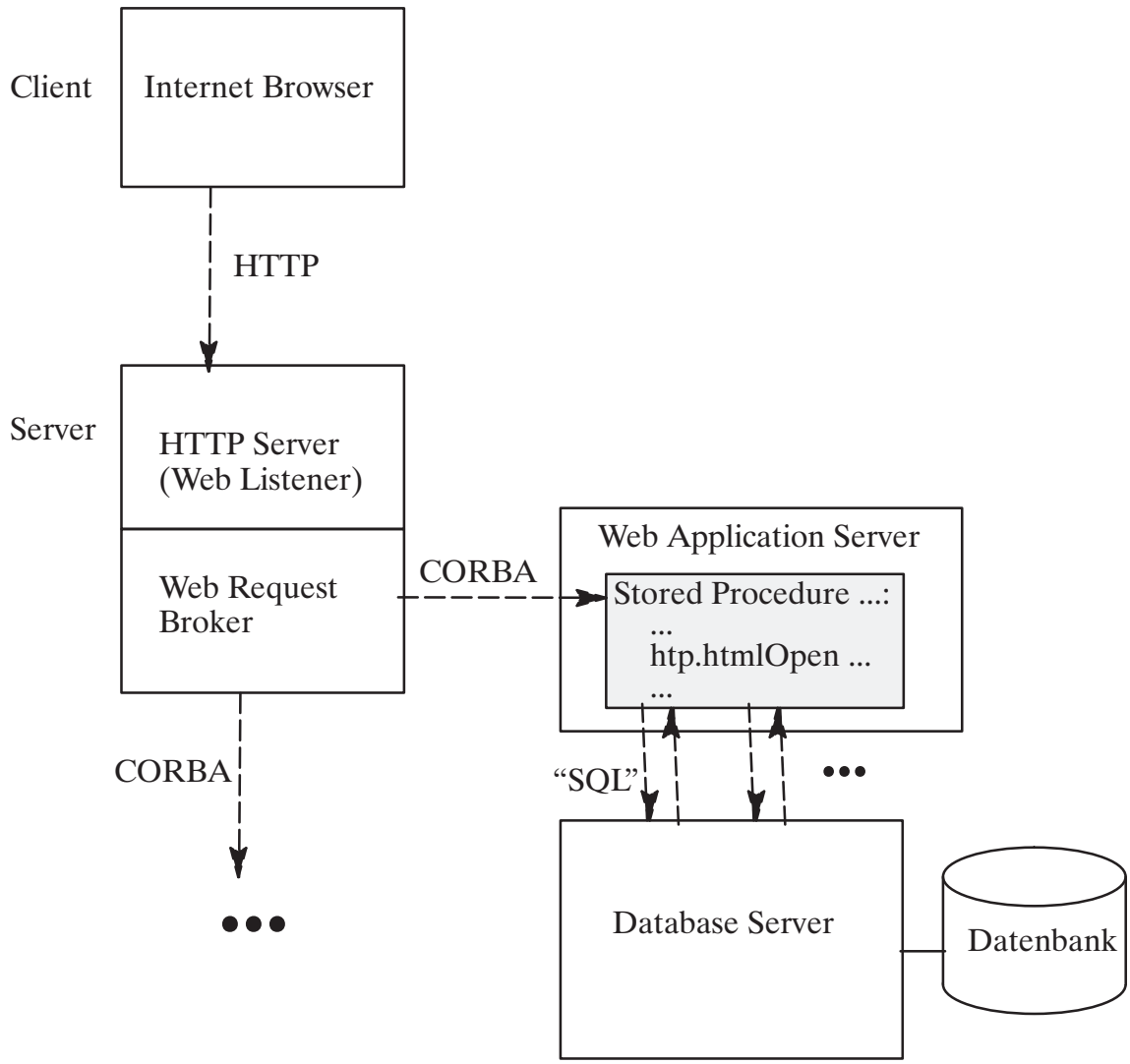
```
  PROCEDURE back (name IN VARCHAR2) IS
    CURSOR tupel IS     SELECT  bid, titel, verfasser, ausleiher
                        FROM  bib_table
                        WHERE lower(ausleiher) LIKE '%'||lower(name)||'%';
      id_out       INT;
      titel_out    VARCHAR2(200);
      verfasser_out VARCHAR2(200);
      ausleiher_out VARCHAR2(200);
    BEGIN
      htp.htmlOpen;  htp.headOpen;  htp.htitle ('Entliehene Bücher:');  htp.headClose;
      htp.bodyOpen;  htp.line; htp.print ('Von "'||htf.em(name)||'" ausgeliehene Bücher:');
      htp.tableOpen ('border');  htp.tableRowOpen;
        htp.tableHeader ('Titel');  htp.tableHeader ('Verfasser');  htp.tableHeader ('Rückgabe');
      htp.tableRowClose;

      OPEN tupel;
      LOOP
        FETCH tupel INTO id_out, titel_out, verfasser_out, ausleiher_out;
        IF tupel%FOUND THEN
          htp.tableRowOpen;
          htp.tableData (titel_out);
          htp.tableData (verfasser_out);
          IF ausleiher_out IS NOT NULL THEN
            htp.tableData (
              htf.anchor (owa_util.get_owa_service_path|| 'bib_user.update_ausleiher'||
                    '?id='||id_out|| '&name='||translate(ausleiher_out,' ','+')|| '&modi=return'||
                    '&titel='||translate(titel_out,' ', '+'),  htf.em (ausleiher_out)));
          END IF;
          htp.tableRowClose;
        ELSE
          EXIT;
        END IF;
      END LOOP;

      htp.tableClose; htp.line;
      htp.anchor (owa_util.get_owa_service_path||'bib_user.home','Zurück zur Home–Page');
      htp.bodyClose;  htp.htmlClose;
    END;

  BEGIN
    NULL;
  END;
/
show errors
```

Laufzeitarchitektur für Internet−fähige Stored Procedures bzw. Packages



Ergänzende Literatur zu Kapitel 4:

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

Programmer's Guide to the Oracle Precompilers, Oracle Press

Pro*C Supplement to the Oracle Precompilers Guide, Oracle Press

Oracle PL/SQL Programming, 2nd Edition, O'Reilly, 1997

Oracle Application Server Web Toolkit Reference, Osborne, 1998

D. Harvey, S. Beitler, The Developer's Guide to Oracle Web Application Server 3, Addison−Wesley, 1998

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

M. Atkinson et al., An Orthogonally Persistent Java, ACM SIGMOD Record Vol.25 No.4, 1996