Belegarbeit

im Fach Datenbankprogrammierung, Sommersemester 2002

Aufgabe 1: DB-Portierung nach ASE 12.x

Das Ziel der ersten Übung war es, ein SQL-Skript zur Einrichtung eines DB-Accounts und einer Datenbank auf *athene12* zu erstellen, dieses ablaufen zu lassen und anschließend bestehende Tabellen von *athene* in die neu erstellte Datenbank zu übernehmen.

Mein Vorgehen hierzu:

Zuerst erzeugte ich das Skript zum erstellen der Datenbank. Die Dokumentation zu den verwendeten stored procedures findet sich in den Online Books von Sybase, SysAdmin Guide, Kapitel 21.

Das Skript:

```
use master
go
create database muehlber_db
 on daten = 3
 log on log = 2
go
sp_addlogin 'muehlber', '992024'
go
use muehlber_db
go
sp_changeowner muehlber
go
```

Führt man dises Skript in einem InteractiveSQL unter dem DB-Root-Account *SA* aus, so erstellt es eine Datenbank *muehlber_db* mit einer maximalen Größe von 3 MB und einer maximalen Logfilegröße von 2 MB auf den devices *daten* und *log*. Anschließend wird ein Datenbanklogin *muehlber* mit dem Passwort *„992024“* angelegt. Damit der Nutzer *muehlber* mit der neu erzeugten Datenbank arbeiten kann, ändern die letzten vier Zeilen des Skriptes den Besitzer der DB *muehlber_db* auf *„muehlber“*.

Mit dem zweiten Teil dieser Aufgabe, der Portierung der der bereits bestehenden Datenbang von *athene* auf *athene12* hatte ich anfänglich ein kleines Problem: Ich hatte die Sybase-Server vorher noch nie benutzt und meine Datenbank auf *athene* war dementsprechend leer. Als Datenquelle verwendete

ich aus diesem Grund die Datenbank eines Mitstudenten.
Als erstes mußten also die Tabellenstrukturen übernommen werden. Hierzu nahm ich mit dem Sybase Powerdesigner ein Reverse-Engineering der vorhandenen Datenbank vor, das Ergebnis war ein graphisches Datenmodell (siehe *daten/kneipe.pdm*). Anschließend änderte ich die Zieldatenbank über die Option *Change Database* auf *AS Enterprise 12.0* und exportierte das graphische Datenmodell in ein SQL-Skript:

```
/*==============================================================*/
/* Database name:  PHYSICALDATAMODEL_2                          */
/* DBMS name:      Sybase AS Enterprise 12.0                    */
/* Created on:     28.06.2002 15:31:36                          */
/*==============================================================*/

/*==============================================================*/
/* Table : Kneipe                                               */
/*==============================================================*/
create table dbo.Kneipe (
     Name                 varchar(30)                    not null,
     Inhaber              varchar(30)                    not null,
     PLZ                  int                            not null,
     Ort                  varchar(20)                    not null,
     Strasse              varchar(30)                    not null,
     constraint PK_KNEIPE primary key (Name)
)
go

/*==============================================================*/
/* Table : Student                                              */
/*==============================================================*/
create table dbo.Student (
     MatNr                int                            not null,
     Vorname              varchar(30)                    not null,
     Name                 varchar(30)                    not null,
     Studgang             varchar(5)                     not null,
     Alt                  int                            not null,
     constraint PK_STUDENT primary key (MatNr)
)
go
```

```
/*==============================================================*/
/* Table : Studi_kneipe                                         */
/*==============================================================*/
create table dbo.Studi_kneipe (
     Name                 varchar(30)                    not null,
     MatNr                int                            not null
)
go

/*==============================================================*/
/* Index: REFERENCE_2_FK                                        */
/*==============================================================*/
create index REFERENCE_2_FK on dbo.Studi_kneipe (
Name ASC
)
go

/*==============================================================*/
/* Index: REFERENCE_3_FK                                        */
/*==============================================================*/
create index REFERENCE_3_FK on dbo.Studi_kneipe (
MatNr ASC
)
go

alter table dbo.Studi_kneipe
   add constraint FK_STUDI_KN_REFERENCE_KNEIPE foreign key (Name)
      references dbo.Kneipe (Name)
go

alter table dbo.Studi_kneipe
   add constraint FK_STUDI_KN_REFERENCE_STUDENT foreign key (MatNr)
      references dbo.Student (MatNr)
go
```

Nachdem ich dieses automatisch erstellte Skript um die Zeilen

```
use muehlber_db
go
```

erweitert hatte (siehe *daten/kneipe.sql*) lies ich es mittels *isql -i kneipe.sql*[1] durchlaufen. Hierbei wurden die im Skript definierten Tabellen angelegt.

Anschließend exportierte ich mit dem Tool *bcp* die Daten aus der ASE-11 Datenbank:

```
bcp seidelh_db.dbo.Kneipe out Kneipe.asc -c -t';' -r '\n' \
   -Sathene -Useidelh
bcp seidelh_db.dbo.Student out Student.asc -c -t';' -r '\n' \
   -Sathene -Useidelh
bcp seidelh_db.dbo.Studi_kneipe out Studi_kneipe.asc -c -t';' \
   -r '\n' -Sathene -Useidelh
```

Auf diese Weise konnte ich, natürlich nur mit Herrn Seidels Hilfe - ich brauchte schließlich sein Passwort, die Inhalte der Tabellen auslesen und in die ASCII-Dateien *Kneipe.asc*, *Student.asc* und *Studi_kneipe.asc* schreiben - immer ein Datensatz pro Zeile, die einzelnen Werte durch Semikola getrennt. Hier die Inhalte der Dateien:

Kneipe.asc

```
Knallfrosch;Knall;39110;MD;Knallstr.
Schluckspecht;Schluck;39106;MD;Schluckstr.
```

Student.asc

```
4211;Marge;Simpson;WIF;19
4712;Jogi;Br;IF;21
7713;Lara;Croft;CV;19
```

[1] Um *isql* ohne weitere Parameter einsetzen zu können, müssen einige Umgebungsvariablen wie beispielsweise *DSQUERY* korrekt gesetzt sein.

Studi_kneipe.asc

```
Knallfrosch;4211
Knallfrosch;4712
Schluckspecht;7713
```

Um Daten aus den ASCII-Dateien in die neu angelegten Tabellen auf *athene12* zu kopieren, wird abermals das *bcp*-Tool benutzt:

```
bcp muehlber_db.dbo.Kneipe in Kneipe.asc -c -t';' -r '\n'
bcp muehlber_db.dbo.Student in Student.asc -c -t';' -r '\n'
bcp muehlber_db.dbo.Studi_kneipe in Studi_kneipe.asc -c -t';' -r '\n'
```

Damit ist Aufgabe 1 abgeschlossen.

Aufgabe 2: Embedded SQL/C

Gegenstand dieser Aufgabe war es, in Anlehnung an ein in der Vorlesung demonstriertes Beispiel eine eigene *ESQL/C*-Applikation zu schreiben, die nach Vorgabe eines Suchbegriffes Daten aus mindestens zwei miteinander verbundenen Tabellen ausgibt und eine einfache Fehler- und Ausnahmezustandsbehandlung demonstriert.

Hier der Quellcode der Applikation (siehe auch *data/esql/esqldemo.cp*):

```
#include <stdio.h>
#include <stdlib.h>
#include "sybsqlex.h"


/* Declare the SQLCA. */
EXEC SQL INCLUDE SQLCA;


/*
 * void error_handler()
 * Displays error codes and numbers from the SQLCA and exits with
 * an ERREXIT status.
 */
void error_handler()
{
 fprintf(stderr, "\n** SQLCODE=(%d)", sqlca.sqlcode);
 if (sqlca.sqlerrm.sqlerrml)
  {
   fprintf(stderr, "\n** SQL Server Error ");
   fprintf(stderr, "\n** %s", sqlca.sqlerrm.sqlerrmc);
  }
 fprintf(stderr, "\n\n");
 exit(ERREXIT);
} /* error_handler */


/*
 * void warning_handler()
 * Displays warning messages.
```

```
 */
void warning_handler()
{
 if (sqlca.sqlwarn[1] == 'W')
  {
   fprintf(stderr, "\n** Data truncated.\n");
  }
 if (sqlca.sqlwarn[3] == 'W')
  {
   fprintf(stderr, "\n** Insufficient host variables to store results.\n");
  }
 return;
} /* warning_handler */


int main(int argc,char **argv,char **envp)
/*************************************************************************/
/*                                                                       */
/*Hauptprogramm                                                          */
/*                                                                       */
/*************************************************************************/
{
 EXEC SQL BEGIN DECLARE SECTION;
  charusername[30];
  charpassword[30];
  chardatenbank[30];
  char StudName[30],
       StudVorname[30],
       KneipName[30],
       KneipPLZ[10],
       KneipOrt[50],
       KneipStrasse[50];
 EXEC SQL END DECLARE SECTION;

 char *passw,
      *user,
      *db,
       vname[30],
       lname[30],
       buf[30];
 int i;
```

```
user = USER;
db = DATABASE;
if (user == "" || db == "")
 {
  fprintf(stderr,"Bitte definieren Sie USER und DATABASE in der %s",
                 "Datei sybsqlex.h und kompilieren Sie das Programm neu. \n");
 } /* endif */

passw = PASSWORD;
if (passw == "")
 passw = getpass("Ihr Datenbankpasswort: ");

EXEC SQL WHENEVER SQLERROR CALL error_handler();
EXEC SQL WHENEVER SQLWARNING CALL warning_handler();

strcpy(username, user);
strcpy(datenbank, db);
strcpy(password, passw);

/* Login in die Datenbank */
EXEC SQL CONNECT :username IDENTIFIED BY :password;

/* Selektieren der DB */
EXEC SQL USE :datenbank;

EXEC SQL DECLARE Student_Cursor CURSOR FOR
 SELECT   Student.Name, Kneipe.Name, Vorname, Strasse, PLZ, Ort
 FROM     Student, Kneipe, Studi_kneipe
 WHERE    Student.MatNr = Studi_kneipe.MatNr and
          Kneipe.Name = Studi_kneipe.Name and
          (Student.Name LIKE :StudName or Vorname LIKE :StudName)
 ORDER BY Kneipe.Name;

EXEC SQL WHENEVER NOT FOUND GOTO not_found;

printf("\nName des gesuchten Studenten (WildCard ist \"%%\", \n");
printf("bitte nur Vorname ODER Nachname angeben): ");
scanf("%s", StudName);
printf("\nGesuchter Name: %s\n\n", StudName);
```

```
EXEC SQL OPEN Student_Cursor;

for (;;)
 {
  EXEC SQL WHENEVER NOT FOUND GOTO finish;
  EXEC SQL FETCH Student_Cursor INTO :StudName, :KneipName,
   :StudVorname, :KneipStrasse, :KneipPLZ, :KneipOrt;

  if (sqlca.sqlcode == 100)
   break;

  sprintf(lname, "");
  sprintf(vname, "");
  buf[0] = "\0";
  i=0;
  while (i <= 30)
   {
    if ( StudName[i] != 32 )
     {
      sprintf(buf, "%c", StudName[i]);
      strcat(lname, buf);
     }
    if ( StudVorname[i] != 32 )
     {
      sprintf(buf, "%c", StudVorname[i]);
      strcat(vname, buf);
     }
    i++;
   }

  printf("\nName des Studenten   : %s %s \n", vname, lname);
  printf("Adresse der \n");
  printf(" Lieblingsnkeipe     : %s \n", KneipName);
  printf("                       %s \n", KneipStrasse);
  printf("                       %s %s \n", KneipPLZ, KneipOrt);
 } /* endfor */

EXEC SQL CLOSE Student_Cursor;
EXEC SQL DISCONNECT DEFAULT;

return (STDEXIT);
```

```
 not_found:
  printf("Keine Studenten gefunden!\n");
  EXEC SQL CLOSE Student_Cursor;
  EXEC SQL DISCONNECT DEFAULT;
  return (STDEXIT);

 finish:
  printf("Bearbeitung abgeschlossen!\n");
  EXEC SQL CLOSE Student_Cursor;
  EXEC SQL DISCONNECT DEFAULT;

return (STDEXIT);
} /* ende main */
```

Dieses Programm stellt an die Datenbank folgende Anfrage:

```
SELECT Student.Name, Kneipe.Name, Vorname, Strasse, PLZ, Ort
 FROM   Student, Kneipe, Studi_kneipe
 WHERE  Student.MatNr = Studi_kneipe.MatNr and
        Kneipe.Name = Studi_kneipe.Name and
        (Student.Name LIKE :StudName or Vorname LIKE :StudName)
 ORDER BY Kneipe.Name;
```

Dabei ist StudName ein vorher einzugebender Suchbegriff. Anschließend werden dann die betreffenden Studenten mit den zugehörigen Lieblingskneipen in nach dem Kneipennamen geordneter Reihenfolge ausgegeben.
Die Daten, die das Programm fuer den Verbindungsaufbau benötigt, bekommt es aus der Datei *sybsqlex.h*. Ferner ist die Umgebungsvariable *DSQUERY* wichtig - hier sollte der Name des Datenbankservers drinnen stehen.

Aufgabe 3: Regeln, Standardwerte, benutzerdefinierte Datentypen

Teil 1: Eine Tabelle der eigenen Datenbank soll um eine Spalte mit einem benutzerdefinieren Datentyp erweiter werden.

Als erstes sollen eine Regel und ein Standartwert erzeugt werden:

```
use muehlber_db
go
create rule freigetraenke as @getraenk > 2
go
create default nix as 0
go
```

Die Regel *freigetraenke* ist nur erfüllt, wenn eine übergebene Variable *getraenk* größer als zwei ist. Dem Defaultwert *nix* wird *0* zugewiesen.
Anschließend wird ein neuer Datentyp *getraenke* angelegt. Er ist vom Typ *Int* und hat den Defaultwert *NULL*.

```
sp_addtype getraenke, 'int', null
go
```

Um der Tabelle *Student* eine neue Splate *drinks* unter Verwendung der neu angelegten Datentypen, Regeln und Defaults zu verpassen, müssen folgende Anweisungen ausgeführt werden:

```
alter table Student add drinks getraenke
go
sp_bindrule 'freigetraenke', 'Student.drinks'
go
sp_bindefault 'nix', 'Student.drinks'
go
```

Als Ergebnis hat die Tabelle eine Spalte *drinks*, in die nur Werte die entweder gleich *NULL* (*NULL* != 0 !) oder größer als zwei sind, eingetragen werden dürfen - der Student hat also zwei Freigetraenke - trinkt er aber mehr als zwei muss er die ersten beiden doch mitbezahlen, zumindest landen sie mit in der Datenbank.
Zum ausprobieren:

```
insert into Student (MatNr, Vorname, Name, Studgang, Alt, drinks)
 values (1538, 'Lisa', 'Simpson', 'IF', 21, 2)
insert into Student (MatNr, Vorname, Name, Studgang, Alt, drinks)
 values (1539, 'Bart', 'Simpson', 'ET', 23, 15)
insert into Student (MatNr, Vorname, Name, Studgang, Alt, drinks)
 values (1540, 'Nochein', 'Simpson', 'ET', 26, NULL)
go
```

Hierbei ginge das erste *insert* schief weil 2 !¿ 2.
Übrigens kann man mit *sp_unbindefault* und *sp_unbindrule* die bindings aufheben, mit *drop rule* bzw. *drop default* die Regeln und Defaults, sowie mit *sp_droptype* den Datentyp löschen.
Teil 2: Indizieren einer geeigneten Spalte mindestens einer Tabelle.

Das Indizieren von Spalten einer Tabelle bewirkt, daß bei späterem Durchsuchen der Tabelle eine optimale Performance erzielt wird.
Angenommen, die Tabelle *Student* wir unter Anderem auch zur Abrechnung oder Auswertung des Getränkekonsums der Studenten genutzt. Dann wäre es durchaus sinnvoll, hier beispielsweise die Spalten *Vorname*, *Name* und *drinks* zu indizieren. Dies geschieht folgendermaßen:

```
create index index1 on Student (Vorname, Name, drinks)
go
```

Aufgabe 4: Programmierung von Batches in TSQL

Teil 1: Erprobung der Arbeit mit dem *Set*-Befehl

set nocount on	die Ausgabe der bearbeiteten Zeilen („n rows affected'") wird unterdrückt
set noexec on	es werden keine Befehle mehr ausgeführt, mit Ausnahme des *Set*-Befehls
set statistics io on	die Anzahl der logischen und physischen Schreib-Lesevorgänge wird ausgegeben
set statistics time on	Anzeige der Ausführungszeit
set rowcount 3	bei der nächsten Anfrage werden nur drei Zeilen gelesen bzw. geschrieben

Teil 2: Auflistung aller Kursleiter, die mehr als einen Kurs leiten

```
use kurs_db12
go
if (
 select count(count(PersNr)) from Fuehrt_durch
  group by PersNr
  having count(*) > 1) > 4
   select count(count(PersNr)) from Fuehrt_durch
   group by PersNr
   having count(*) > 1
else select 'Niemand hat mehr als einen Kurs.'
go
```

Teil 3: Tabellen kopieren, Programmierung von Schleifen

Es soll eine Kopie der Tabelle *titles* aus der Datenbank *pubs2* in der eigenen Datenbank erstellt werden. Die neue Tabelle soll den Namen *titlesN* tragen.

```
use master
go
sp_dboption muehlber_db, 'select into', true
go
use muehlber_db
go
checkpoint
go
```

```
select *
 into titlesN
 from pubs2.dbo.titles
go
```

Den Durchschnittspreis der Bücher kann man mit der Funktion *avg()* sehr einfach ermitteln:

```
select avg(price) from titlesN
go
```

Unter der Bedingung, daß der Durchschnittspreis über einem bestimmten Wert liegt, sollen nun die Preise aller Bücher, die ebenfalls über diesem Wert liegen, um 5% reduziert werden. Dies soll solange wiederholt werden, bis der Durchschnittspreis den vorgegebenen Wert erreicht oder unterschreitet:

```
while (select avg(price) from titlesN) > 10
begin
 print "Preise werden aktualisiert"
 update titlesN
  set price = price * 0.95
  where price > 10
end
```

Teil 4: Die Variable *@@rowcount*

```
select * from Kneipe
```

führt zu einer Anzeige der Daten sätze in *Kneipe*, es werden zwei ausgegeben.

```
select @@rowcount
```

zeigt die Anzahl der beim letzten *SELECT* ausgegebenen Zeilen an, also auch zwei.
Ein nochmaliges

```
select @@rowcount
```

führt zur Ausgabe 1 weil das letzte *SELECT* nur eine Ausgabezeile erzeugt hat.
Teil 5: Batcherzeugung zur Datenausgabe

Folgender Batch gibt das aktuelle Datum, den aktuellen Datenbankanwendernamen und die gerade verwendete Datenbank in einer Zeile aus:

```
select getDate(), suser_name(), db_name()
```

Aufgabe 5: Programmierung von TSQL-Triggern

Teil 1: Kopieren von Tabellen in die eigene Datenbank

Anfänglich sind wiederum einige Tabellen aus der *kurs_db12* in die eigene Datenbank zu kopieren:

```
use muehlber_db
go
select *
 into Kurs
 from kurs_db12.dbo.Kurs
select *
 into Angebot
 from kurs_db12.dbo.Angebot
select *
 into Teilnehmer
 from kurs_db12.dbo.Teilnehmer
select *
 into Kurslit
 from kurs_db12.dbo.Kurslit
select *
 into Nimmt_teil
 from from kurs_db12.dbo.Nimmt_teil
go
```

Teil 2: es ist ein Lösch-Trigger *del_kurs* zu schreiben, der das Löschen einer Zeile aus *Kurs* cascadierend auf die Tabellen *Angebot* und *Nimmt_teil* überträgt:

```
create trigger del_kurs
on Kurs for delete
as
 if @@rowcount=0
  return
 delete Angebot from Angebot, deleted
  where Angebot.KursNr = deleted.KursNr
 delete Nimmt_teil from Nimmt_teil, deleted
  where Nimmt_teil.KursNr = deleted.KursNr
 return
```

Teil 3: Nun soll ein Insert-Trigger *ins_nt* geschrieben werden, der beim Einfügen einer Zeile in *Nimmt_teil* für die verwendete Kursnummer bereits eine Zeile existiert und den Inhalt des Attributes *Bedarf* um 1 erhöht:

```
create trigger ins_nt
on Nimmt_teil for insert
as
 declare @zeilen int
 select @zeilen = @@rowcount
 if @zeilen = 0
  return

 if not exists (select * from Kurslit, inserted
  where Kurslit.KursNr = inserted.KursNr)
   begin
    print 'rollback'
    rollback transaction
    return
   end

 print'update'
 update Kurslit
  set Kurslit.Bedarf = Kurslit.Bedarf+1
  from Kurslit, inserted
  where Kurslit.KursNr = inserted.KursNr
 return
```

Teil 4: Abschließend sollen alle Trigger getestet werden und die Anzeige von Triggern und Abhängigkeiten über die gespeicherten Prozeduren *sp_help*, *sp_helptext* und *sp_depends* durchgeführt werden.
Das Testen der Trigger:

Die Verwendung der Prozeduren:

sp_help del_kurs	es werden Erstellungsdatum, Besitzer, Typ (hier Trigger) und Name angezeigt
sp_helptext del_kurs	der Quellcode des Triggers wird ausgegeben
sp_depends del_kurs	hierbei wird die Tabelle angezeigt, auf die sich der Trigger bezieht

Aufgabe 6: Transaktionen und Recovery in TSQL

Teil 1: Erzeugen von Tabellen
Erzeugt werden sollen zwei Tabellen *Spar1* und *Spar2*. Diese Repräsentieren Sparbücher und haben jeweils die Spalten *Name* und *Betrag*. *Name* soll der Primärschlüssel sein. Beide Tabellen sollen die gleichen Personen enthalten, lediglich die Kontenbeträge sollen sich unterscheiden:

```
use muehlber_db
go
create table Spar1
 (Name varchar(20) primary key,
 Betrag numeric)
go
create table Spar2
 (Name varchar(20) primary key,
 Betrag numeric)
go
insert into Spar1 (Name, Betrag) values ('Martin', 32679)
insert into Spar1 (Name, Betrag) values ('Hannes', 74812)
insert into Spar1 (Name, Betrag) values ('Alex', 23427)
insert into Spar2 (Name, Betrag) values ('Martin', 45679)
insert into Spar2 (Name, Betrag) values ('Hannes', 21812)
insert into Spar2 (Name, Betrag) values ('Alex', 11427)
go
```

Teil 2: Anschließend soll eine Prozedur zum Umbuchen eines Betrages *Spar1* nach *Spar2* geschrieben werden, wobei der umzubuchende Betrag zusammen mit dem Namen als Parameter bereitzustellen sind:

```
create proc umbuchung @Name varchar(20), @Betrag numeric
as
 begin tran muehlber
 if exists (select Spar1.Name from Spar1, Spar2
  where Spar1.Name like @Name and Spar2.Name like @Name)
 begin
  update Spar1
   set Betrag = Betrag - @Betrag
   where Name like @Name
  update Spar2
   set Betrag = Betrag + @Betrag
```

```
   where Name like @Name
 end
 else rollback tran
commit trans
return
go
```

Die Prozedur kann nun aufgerufen werden:

```
umbuchung Martin, 5000
go
```

Teil 3: Die Prozedur soll dahingehend erweitert werden, als daß eine Prüfung stattfinden soll, die absichert, daß der verbleibende Betrag auf *Spar1* den Wert 500 nicht unterschreitet:

```
create proc umbuchung2 @Name varchar(20), @Betrag numeric
as
 begin tran muehlber
  if exists (select Spar1.Name from Spar1, Spar2
   where Spar1.Name like @Name and Spar2.Name like @Name)
  begin
   if (select Betrag from Spar1 where Name like @Name) > 500 + @Betrag
    begin
     update Spar1
      set Betrag = Betrag - @Betrag
      where Name like @Name
     update Spar2
      set Betrag = Betrag + @Betrag
      where Name like @Name
    end
   else rollback tran
 end
 commit trans
return
```

Teil 4: Abschließend soll noch ein Batch geschrieben werden, dass die Kontostände vor und nach der Transaktion ausgeben kann:

```
select  * from Spar1, Spar2 where Spar1.Name like Spar2.Name
```

```
go
umbuchung Martin, 5000
go
select  * from Spar1, Spar2 where Spar1.Name like Spar2.Name
```

Aufgabe 7: Internet Datenbank Zugriff über PowerDynamo

Teil 1: Es ist ein WebSite in der eigenen Datenbank zu erzeugen. Dies geschieht mittels des PowerDynamo - zuerst muß ein Webknoten eingerichtet werden.

Teil 2: Es soll ein Template für den Zugriff auf zwei oder mehr miteinander verbundene Tabellen der eigenen Datenbank realisiert werden. Danach soll der Template-Code um *Text Substitution* und Formatierung der Ausgabe erweitert werden und ein Dynamo Script erzeugt und eingebunden werden. Hierzu kann man mittels eines Wizards ein Template erstellen. Allerdings muss es anschleißend noch manuell verändert werden, wofür leider die Verwendung eines vollkommen unbenutzbaren Editors von Nöten ist. In den Templates können zum einen SQL-Skripte (geklammert durch *¡!–SQL ... SQ–¿*) also auch DynamoScript (geklammert durch *¡!–SCRIPT ... SCRIPT–¿*), eine sehr JavaScript ähnliche Sprache verwendet werden. Beispiele:

```
<!--SQL
select * from Student
SQL-->
<!--SCRIPT
 document.Write("DynamoScript");
SCRIPT-->
```

Um die Ergebnisse von SQL-Anfragen darzustellen, kann man die Tabellenstruktur in HTML nachbilden und anschließend mittels *¡!–data–¿* die Daten automatisch einfügen lassen.
Ferner können sehr einfach Daten aus dem Formular eines Templates an ein andere Template übergeben und dort ausgewertet werden.

Teil 3: Herstellung der Voraussetzungen (Connection Profile, ...)
Hierzu muss der Web-Server neu konfiguriert werden, allerdings mußte ich das nicht selber machen und erinnere mich daher nicht mehr an die Details.