

# Net.Data anwenden

Im ersten Teil hatten wir den Apache httpServer für statische Inhalte eingerichtet, im zweiten Teil hat CGI-Programms Net.Data eingerichtet und ein erstes Test-Makro erzeugt. Nun kommen wir zu den ab warum ich glaube, dass nahezu jeder iSeries-Anwender von dieser leistungsfähigen Makrosprache pr EDV-Crew selber – allein das kann schon ein sehr lohnendes, ernst zu nehmendes Einsatzgebiet sein!

## SQL Integration

Einer der Hauptbeweggründe, seine iSeries als Webserver zu betreiben, sind die wertvollen Daten, die diese Masc

Net.Data hat ein so genanntes „SQL-Language-Environment“. Das bedeutet vereinfacht ausgedrückt, dass in bes die Eingabe von SQL-Anweisungen möglich ist.

Bevor wir das aber tun können, prüfen Sie, ob die Datenbank Ihrer Maschine einen Namen hat. Tun Sie das mit \ „Verzeichnis der Relationalen Datenbanken“. Wenn Sie hier eine Zeile mit „Ferner Standort“ gleich „\*LOCAL“ seh

Relationale Datenbank	Ferner Standort	Text
A609A05B	*LOCAL	Entry added by system

Wenn die Anzeige leer ist oder Sie keinen „fernen“ Standort \*LOCAL finden, legen Sie ihn bitte wie folgt an.

**ADDRDBDIRE RDB(mySystem)RMTLOCNAME(\*LOCAL \*IP) TEXT('lokale Datenbank')**

Wie Ihre lokale Datenbank heißt, ist ziemlich egal. Der Systemname ist sicher keine schlechte Idee.

Fangen wir gleich mit einem Makro an, das uns die Testdaten für weitere Beispiele erzeugen soll.

Sie können nun eine SQL-Collection erstellen, da ich aber davon ausgehe, dass die wenigsten Leser ihre Daten in normalen Bibliotheken, werden wir hier auch eine Bibliothek mit Daten verwenden.

**CRTLIB LIB(TECHDATA)TEXT('Daten für TECHKNOW Instanz')**

Die Datei, die wir zum Testen verwenden, erstellen wir hier aber sehr wohl mit SQL; Sie können natürlich aber au extern beschriebene Datei verwenden. (Programmbeschriebene (alias „S/3x-Dateien“) können Sie nicht sinnvoll \ Environment.)

## Das SQL-Beispiel-Makro – der schwere Teil

Das folgende Makro habe ich als „SQL\_Beispiel1.ndm“ in „/www/TechKnow/Makros“ abgespeichert.

Keine Angst, wenn Sie nicht gleich alles verstehen, was hier abgeht! Dieses Makro erstellt einstweilen nur mal eir sehr viele Net.Data-Funktionen, die ich zwar alle erkläre, aber es ist doch ein bissl viel auf einmal. Viele Makros k ohnehin nur Daten anzeigen wollen, können Sie diesen Teil auch vorerst mal ganz überspringen.

Neu ist der %DEFINE-Block, ebenfalls mit %} zu schließen. Hier kann man globale Variablen definieren; Variable vorkommen

Net.data - SQL\_Beispiel1

```
%DEFINE{
DatenLib = "TECHDATA"
Vorname1 = "Albert"
Vorname2 = "Berta"
Vorname3 = "Claus"
Vorname4 = "Dieter"
Vorname5 = "Elsa"
Vorname6 = "Franz"
Vorname7 = "Gerhard"
Vorname8 = "Hannelore"
```

```

Vorname9 = "Ilse"

Nachname1 = "Antel"
Nachname2 = "Bohlen"
Nachname3 = "Cervantes"
Nachname4 = "Durst"
Nachname5 = "Einstein"
Nachname6 = "Fitz"
Nachname7 = "George"
Nachname8 = "Halbwert"
Nachname9 = "Immerzu"

%}

```

Zunächst definieren wir die Datenbibliothek, in der eine Datei „Kollegen“ erstellt werden soll. Wenn Sie statt TECI Bibliothek gewählt haben, ändern Sie den Namen hier, denn er kommt nur hier vor.

Die anderen Variablen enthalten Vor- und Nachnamen für unsere Testdaten; Ähnlichkeiten mit lebenden oder tot

Dann folgt die Funktion „CreateFileKollegen“. Diese Funktion gehört dem SQL-Language-Environment an, kenntlich durch die Schlüsselwörter **FUNCTION**.

Das leere Klammerpaar zeigt an, dass diese Funktion keine Parameter hat.

```

FUNCTION(DTW_SQL) CreateFileKollegen() {

CREATE TABLE $(DatenLib).KOLLEGEN
(
    PERSNR          DEC (3 , 0)          NOT NULL WITH DEFAULT,
    VNAME           CHAR (20 )          NOT NULL WITH DEFAULT,
    NNAME           CHAR (32 )          NOT NULL WITH DEFAULT,
    USRID400        CHAR (10 )          NOT NULL WITH DEFAULT,
    KLAPPE          CHAR (4 )           NOT NULL WITH DEFAULT,
    GEBDAT          DATE                 NOT NULL WITH DEFAULT)

%}

```

Zwischen den geschwungenen Klammern finden Sie das SQL-Statement. In dieses SQL-Statement können Sie durch komplexere Konstruktionen wie IFs und dergleichen sind aber nicht möglich.

Hier wird die globale Variable „DatenLib“ ausgegeben. Es wird einfach \$(Variablenname) geschrieben. Net.Data und Kleinschreibung, „DATENLIB“ und „DatenLib“ sind also zwei völlig unterschiedliche Variablen. Variablen müssen also niemals eine Fehlermeldung kriegen, wenn Sie eine Variable ausgeben (wollen), die es nicht gibt. Und noch: wenn Net.Data rechnet, erwartet es Strings als Parameter!

Dann folgt eine weitere SQL-Funktion. Diese Funktion erwartet Parameter, und zwar ausschließlich Eingabeparameter. Die in der Schnittstelle definierten Variablen sind lokal definiert und stehen keinen anderen Funktionen zur Verfügung (außer

```

%FUNCTION(DTW_SQL) InsertKollege(IN PERSNR,
                                VNAME,
                                NNAME,
                                USRID400,
                                KLAPPE,
                                GEBDAT) {

INSERT INTO $(DatenLib).KOLLEGEN
(
    PERSNR,
    VNAME,
    NNAME,
    USRID400,
    KLAPPE,
    GEBDAT)
VALUES ( 0$(PERSNR),
        '@DTW_rADDQUOTE(VNAME)',
        '@DTW_rADDQUOTE(NNAME)',
        '@DTW_rADDQUOTE(USRID400)',
        '@DTW_rADDQUOTE(KLAPPE)',
        DATE('1900-01-01') + 0$(Tage) DAYS))

%}

```

Danach folgt, wie gehabt, das SQL-Statement, in das wieder Variablen mit \$(Variablenname) ausgegeben werden

Das (echte) Datumsfeld GEBDAT mit dem Geburtsdatum unserer bemitleidenswerten Testpersonen wird durch Ac zum 1. Jänner 1900 gewonnen.

Weiters neu ist die Funktion @DTW\_rADDQUOTE. Alles, was mit „@DTW“ beginnt, ist eine in Net.Data eingebaut. @DTW\_ADDQUOTE hat die Aufgabe, einzelne Hochkommas durch doppelte zu ersetzen. Und das sollten Sie sich

Denken Sie an die Möglichkeit, dass der Nachname eines Kollegen ein Hochkomma enthält, etwa „O'Connor“, und herauskommt und die Verarbeitung mit einer Fehlermeldung abbricht. Bevor Sie sich nun denken, das geht Sie nicht an, lesen Sie bitte weiter.

Denn wenn Sie sich das folgende SQL-Statement vorstellen, das zum Beispiel in einer Intranet-Applikation dazu genutzt für MitarbeiterINNEN änderbar zu machen, wird Ihnen auf den ersten Blick nichts auffallen.

```
%FUNCTION(DTW_SQL) HandyEintragen(IN PersNr, Handy) {  
  
    UPDATE PERSLIB.PERSSTAMM  
        SET HANDYNR = '$(Handy)'  
        WHERE PERSNR = $(PersNr)  
%}
```

In die Handynummer wird schon keiner ein Hochkomma eingeben.

Stimmt, außer er will Böses. Gesetzt den Fall, in der Datei PERSSTAMM gäbe es auch ein Feld GEHALT, dann könnte

```
ja', GEHALT=GEHALT*2, HANDYNR='ja
```

in ein Eingabefeld Handynummer eine dramatische Erhöhung der Personalkosten bewirken.

Denn lassen Sie sich das daraus entstehende SQL-Statement auf der Zunge zergehen:

```
UPDATE PERSLIB.PERSSTAMM SET HANDYNR = 'ja', GEHALT = GEHALT * 2, HANDYNR='ja' WHERE PERSNR = $
```

Das Statement ist gültig, wird wahrscheinlich ohne Probleme ausgeführt, und die Handynummer kann schon im Browser

Keine Panik, ich will Ihnen keine Angst vor Web-Anwendungen machen, ich will Ihnen bloß die konsequente Verwendung

Vielleicht ist Ihnen auch meine seltsame Art, numerische Variablen in das SQL-Statement einzubetten, aufgefallen

```
UPDATE BIBLIOTHEK.DATEI SET WERT = 0$(Variable)
```

Falls aus irgendeinem Grund in der Variable „nichts“ drin steht, geht das SQL-Statement trotzdem durch, weil das

Allerdings schützt das nicht vor den beschriebenen Möglichkeiten, mit bösen Eingaben im Browser SQL-Statements

Das betrifft übrigens nicht nur Net.Data-Anwendungen, alle Anwendungen, die eingebare Strings zu SQL-Befehlen

Problem! (Aber nicht alle haben ein derart einfaches Gegenmittel.)

Zurück zu unserem Makro.

Wir landen nun bei dem schon aus dem letzten Teil bekannten HTML-Abschnitt.

Hier folgt nun ein Ausflug in die schon etwas fortgeschrittenere Programmierung; für den Anfang werden Sie auch feiern können.

```
%HTML (CREATE) {@DTW_ASSIGN(SHOWSQL,"NO")
```

Der HTML-Abschnitt heißt CREATE, und gleich als erste Anweisung wird ein Befehl ausgeführt, den Sie sich ebenfalls sollten; zumindest wenn Sie die INI-Datei aus dem vorigen Teil verwenden.

Damit haben Sie die Möglichkeit, SQL-Anweisungen im Browser zu sehen. In Ihrer Net.Data-Laufbahn werden die und es ist dann gut, sehen zu können, was man da fabriziert hat.

Damit das nicht ungewollt durch Besucher Ihrer Seiten erfolgt, sollten Sie die Variable SHOWSQL gleich am Anfang der Net.Data-Funktion @DTW\_ASSIGN auch schon erklärt. Sie können Sie bei Bedarf – beim Debuggen – an anderer

Es folgt absolut statischer HTML-Code mit dem Anfang der an den Browser zu sendenden HTML-Seite:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>
<title>SQL Beispiel
</head>
<body>
<h2>Erstellen der Testdaten</h2>
```

Dann wird die Uhrzeit mit tausendstel Sekunden ausgegeben und dahinter ein Text.

```
@DTW_rTIME("X"): Datei KOLLEGEN in $(DatenLib) wird erstellt ...
```

Dann folgt der Aufruf unserer ersten SQL-Funktion! Damit wir das SQL-Statement, das ausgeführt wird, mal sehen können, rufen dann unsere Funktion auf und stellen dann SHOWSQL wieder auf NO.

```
@DTW_ASSIGN(SHOWSQL,"YES")
@CreateFileKollegen()
@DTW_ASSIGN(SHOWSQL,"NO")
```

Dann folgen wieder Dinge, die Sie schon kennen:

```
<hr>
@DTW_rTIME("X"): Datei KOLLEGEN in $(DatenLib) wird befüllt ...<br>
```

Und nun sehen Sie, wie man in Net.Data programmiert. Es macht gar nichts, wenn Sie beim ersten Durchlesen nicht alle Makros sind einfacher. Aber man muss ja auch die Leute, die Net.Data schon (etwas) kennen, bei der St

@DTW\_ASSIGN kennen Sie schon; wir definieren hier eine Zählvariable für die Personalnummer, und einen „Hilfs

```
@DTW_ASSIGN(myPersNr,"0")
@DTW_ASSIGN(vi,"1")
```

Obwohl wir damit rechnen werden, es handelt sich um Strings! Rechnen ist – wenig überraschend für ein Werkzeug unbedingt eine Stärke von Net.Data. Nein, ehrlicherweise muss man schon sagen, es ist eine Schwäche.

Nun folgt eine WHILE-Konstruktion, solange die Bedingung, die in Klammer angegeben wird, zutrifft, wird alles zu ausgeführt.

```
%WHILE (vi <= "9") {
```

Dann definieren wir einen „Index“ für die Nachnamen und öffnen eine zweite WHILE-Konstruktion:

```
@DTW_ASSIGN (ni, "1")  
%WHILE (ni <= "9") {
```

Hier wird gerechnet, und zwar wird myPersNr um 1 erhöht:

```
@DTW_ADD (myPersNr, "1", myPersNr)
```

Sicher, viel logischer wäre 1 ohne Anführungszeichen, aber es sind nun mal Strings, und das sollte man nicht ver auf numerisch konvertiert, danach ist alles wieder String. Natürlich dürfen Sie nur Strings verwenden, die gültige

```
@DTW_ASSIGN (myVorname, "$(Vorname$(vi)) ")
```

Hier wird das schon bekannte ASSIGN etwas anders als gewohnt verwendet. Es wird von innen nach außen aufgedas so aus:

```
@DTW_ASSIGN (myVorname, "$(Vorname1) ")
```

dann wird der entstandene Ausdruck wieder bewertet, und also die Variable „Vorname1“ eingesetzt, und dann kann werden:

```
@DTW_ASSIGN (myVorname, "Albert ")
```

Dasselbe folgt für den Nachnamen.

```
@DTW_ASSIGN (myNachname, "$(Nachname$(ni)) ")
```

Dann kommt eine längere Konstruktion, die im Makro in einer Zeile stehen muss, die verwendet wird, um eine fikt Stellen des Nachnamens und den ersten 3 Stellen des Vornamens zu erzeugen:

```
@DTW_ASSIGN (myUSRID400, "@DTW_rSUBSTR (myNachname, "1", „3")@DTW_rSUBSTR (myVorname, "1", "3") ")
```

Funktioniert im Prinzip wie oben. Wenn es noch länger, sprich breiter, weil ja alles in eine Zeile passen muss, wenn nicht gefällt, lässt sich ganz genau dasselbe beispielsweise auch so kodieren:

```
@DTW_SUBSTR (myNachname, „1“, „3“, HilfsFeld1)  
@DTW_SUBSTR (myVorname, „1“, „3“, HilfsFeld2)
```

```
@DTW_CONCAT(HilfsFeld1, HilfsFeld2, myUSRID400)
```

@DTW\_SUBSTR holt ab inklusive der 1. Stelle 3 Zeichen aus dem Feld „myNachname“ und speichert die in dem r

Spätestens jetzt sollte ich Ihnen den Unterschied zwischen @DTW\_SUBSTR und @DTW\_rSUBSTR erklären.

@DTW\_SUBSTR speichert das Ergebnis im letzten Parameter.

@DTW\_rSUBSTR gibt das Ergebnis dort aus, wo die Funktion aufgerufen wurde.

Und DTW\_CONCAT hängt Parameter 1 und Parameter 2 zusammen und speichert das Ergebnis in „myUSRID400“

Das folgende Statement ist zweifellos selbsterklärend, eine Funktion DTW\_LOWERCASE für die Umwandlung in Kl

```
@DTW_UPPERCASE(myUSRID400,myUSRID400)
```

Wenn Sie die Gruppen-PTFs alle brav installiert haben (und das sollten Sie unbedingt tun, kumulierte PTFs sind zu DTW\_EVAL, die einen Ausdruck bewertet und das Ergebnis in der im zweiten Parameter anzugebenden Variable s

```
@DTW_EVAL("$ (vi) *10+$ (ni) ",myDays)
```

Wenn Sie nun in der unglücklichen Lage sind, sich die Zeit während des Herunterladens des Gruppen-PTFs zu ver Konstruktion, die das Gleiche tut:

```
@DTW_MULTIPLY(vi,"10",HilfsFeld1)
      @DTW_ADD(HilfsFeld1, ni, myDays)
```

Und natürlich führt auch dieser Weg nach Rom:

```
@DTW_ADD(@DTW_rMULTIPLY(vi,"10"), ni, myDays)
```

Aber zumindest für meinen Geschmack leidet die Lesbarkeit bei der letzten Variante doch sehr.

Dann rufen wir die SQL-Insert-Funktion auf und übergeben ihr nun unsere Parameter:

```
@InsertKollege(myPersNr,myVorname,myNachname,myUSRID400,"$(vi)$ (ni) ", "DATE('1900-01-01') + $(my
```

Das Dateifeld „Klappe“ erzeugen wir durch Zusammenhängen der beiden Zählvariablen on the fly, das erfolgt obe

Dann folgt noch das Hochzählen der beiden „Indizes“ und die Ausgabe einer Ende-Meldung mit Uhrzeit. Dann enc Makro endet auch.

```
@DTW_ADD(ni,"1",ni)
%}
```

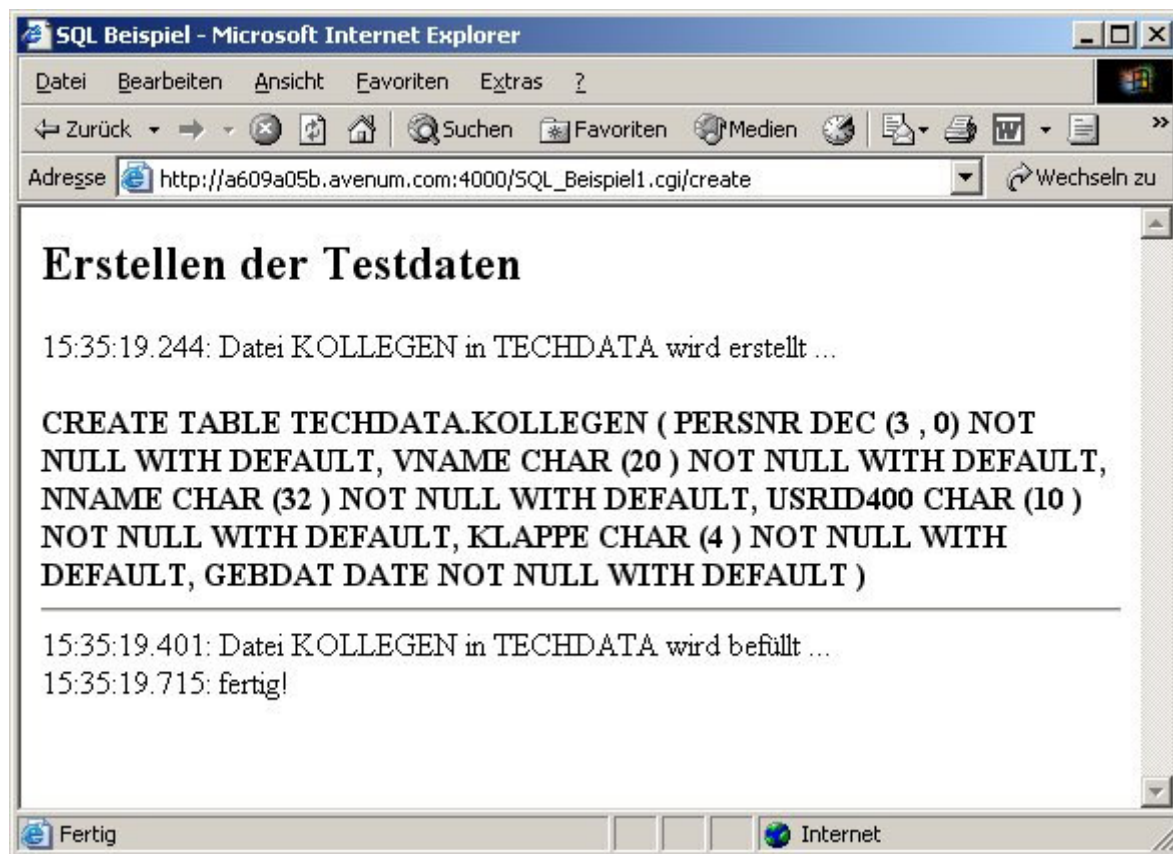
```
@DTW_ADD(vi,"1",vi)
%}
```

```
@DTW_rTIME("X"): fertig!
</body>
</html>
```

%}

Wenn Sie dieses Makro „SQL\_Beispiel1.ndm“ nun mit [http://iSeries:4000/SQL\\_Beispiel1.cgi/create](http://iSeries:4000/SQL_Beispiel1.cgi/create) aufrufen, wer erste Aufruf dauert gern ein bisserl, und das Erstellen von Dateien war noch nie ein bedeutungsloser Klacks für ei

Die Anzeige wird ca. so aussehen:



**Abbildung 1 - Ausgabe des Makros SQL\_Beispiel1**

Und ein RUNQRY \*N TECHDATA/KOLLEGEN zur Kontrolle zeigt uns:

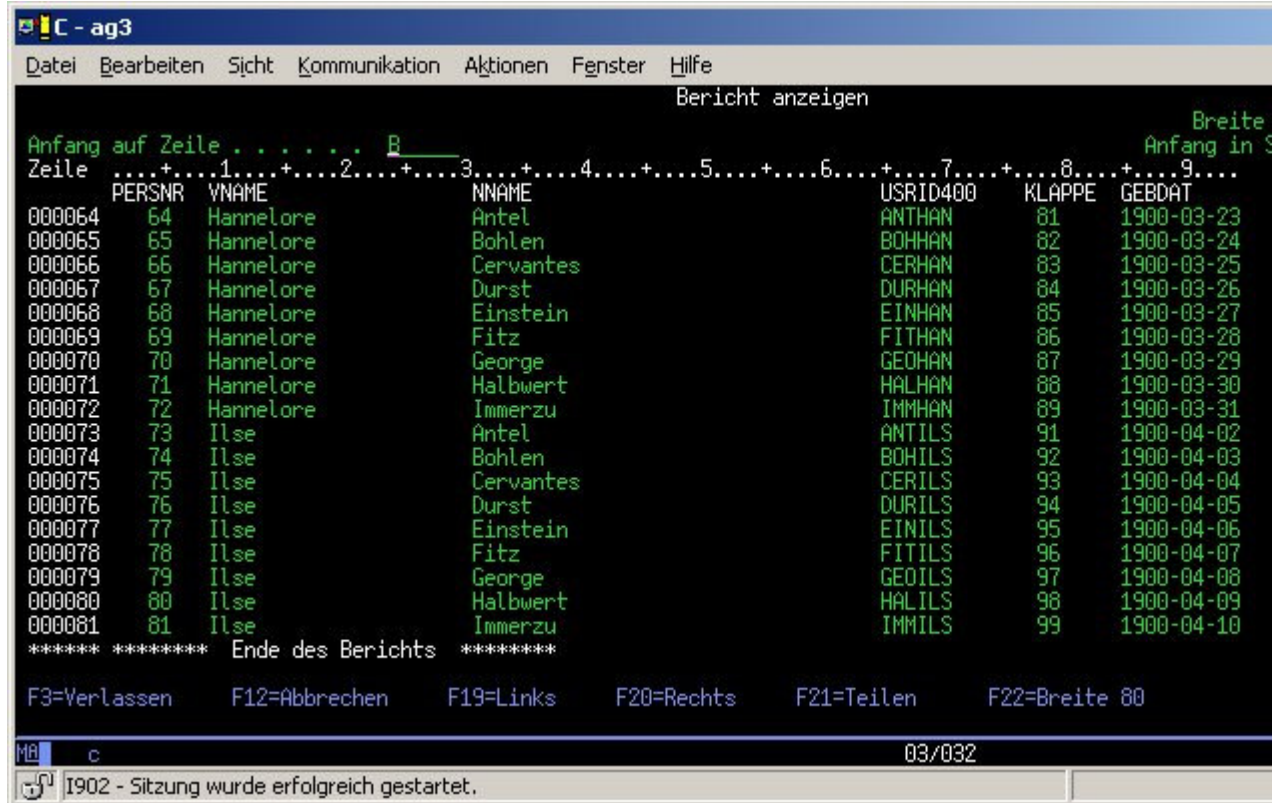
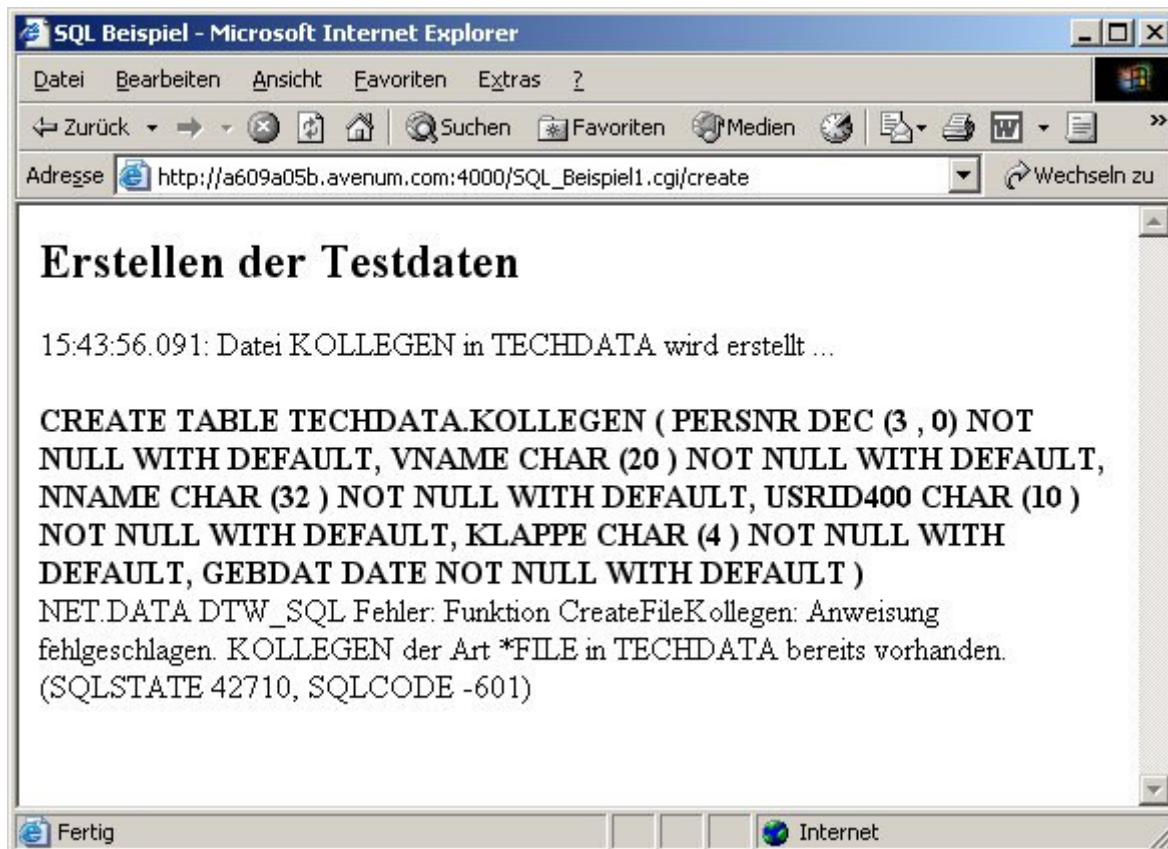


Abbildung 2 - Anzeige der Datei TECHDATA/KOLLEGEN

### Fehlerbehandlung in SQL-Funktionen

Wenn Sie nun einfach auf „Aktualisieren“ klicken, dann scheitert die Ausführung natürlich, weil es die mit CREATE



### Abbildung 3 – erneute Ausführung des Makros

Es gibt in Net.Data etwas Ähnliches wie MONMSG in CL, den %MESSAGE-Block. Der wird am Ende der Funktion k auch nach dem %DEFINE-Block kodieren, dann gilt er global für alle Funktionen im Makro.

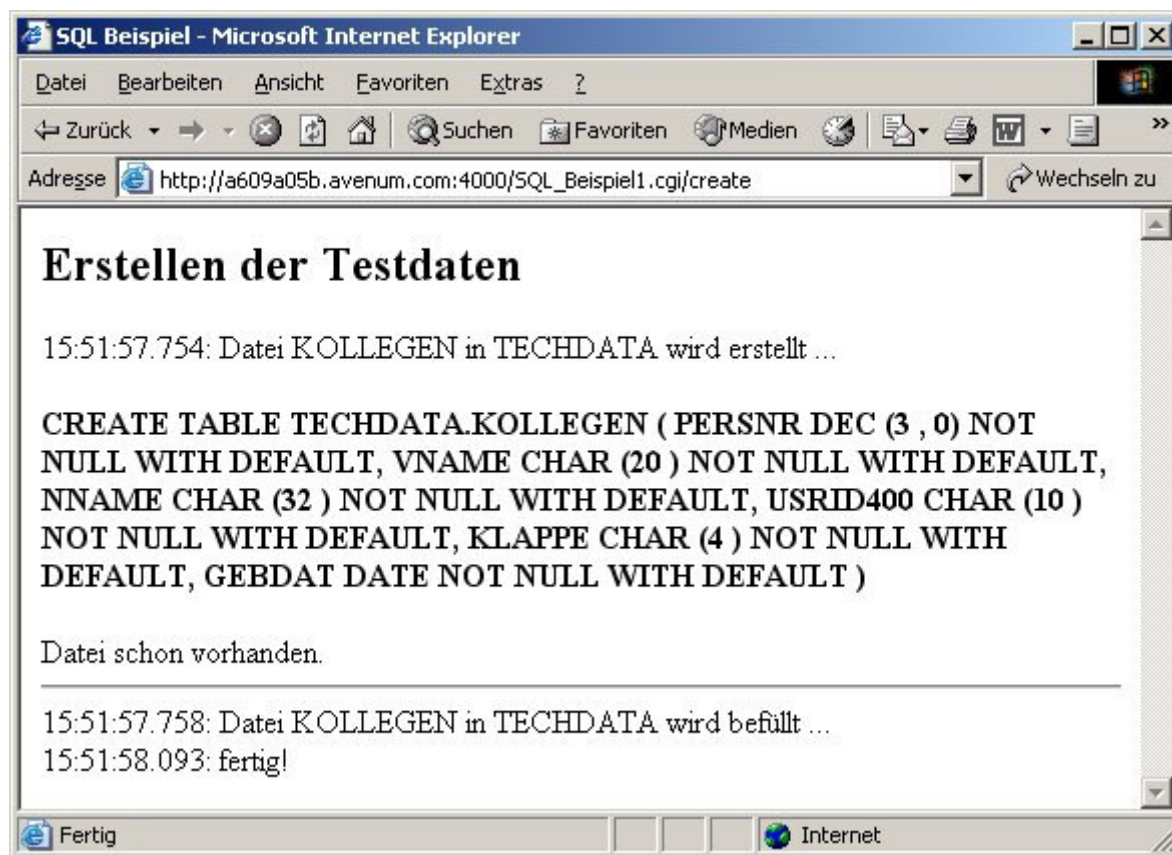
Ändern wir unsere CreateFileKollegen-Funktion auf:

```
%FUNCTION(DTW_SQL) CreateFileKollegen() {  
  
    CREATE TABLE $(DatenLib).KOLLEGEN  
        (  
            PERSNR          DEC (3 , 0)          NOT NULL WITH DEFAULT,  
            VNAME           CHAR (20 )          NOT NULL WITH DEFAULT,  
            NNAME           CHAR (32 )          NOT NULL WITH DEFAULT,  
            USRID400        CHAR (10 )          NOT NULL WITH DEFAULT,  
            KLAPPE          CHAR (4 )          NOT NULL WITH DEFAULT,  
            GEBDAT          DATE                NOT NULL WITH DEFAULT)  
  
    %MESSAGE {  
        -601 : "<br>Datei schon vorhanden.<br>" : continue  
    %}  
  
    %}
```

Wie Sie sehen, wird der SQLCODE -601 überwacht, der Text ausgegeben (dort könnten sich auch Funktionsaufrufe fortgesetzt).

Wenn Sie mehrere Fehlercodes überwachen wollen, geben Sie mehrere Zeilen an.

Wenn Sie nochmals aktualisieren, sieht die Anzeige so aus:



Damit wir den Test vervollständigen (und die nun natürlich doppelt in die Datei geschriebenen Testsätze wieder k einmal (DLTF TECHDATA/KOLLEGEN) und erstellen sie erneut, indem Sie die Anzeige im Browser erneut aktualisi

Wenn Sie bisher durchgehalten haben, haben Sie sich nun ein einfaches Makro verdient. Genauer gesagt, wir erw Funktion und um einen neuen Makro-Abschnitt.

## Das SQL-Beispiel-Makro – der leichte Teil

Im vorigen Abschnitt haben wir eine Datei „KOLLEGEN“ erstellt, die wir uns nun anzeigen lassen wollen; sicher da Deshalb gehen wir es zur Abwechslung mal ganz einfach an und wollen eine Telefonliste im Browser anzeigen.

Wir ergänzen unser Makro um die folgende Funktion, wir fügen das nach den anderen Funktionen ein, vor dem H

```
%FUNCTION(DTW_SQL)  Telefonliste() {

    SELECT VNAME, NNAME, KLAPPE
           FROM $(DatenLib).KOLLEGEN
           ORDER BY NNAME, VNAME

    %REPORT {
        <table border="1">
        <tr>      <th>Name</th>
                <th>Klappe</th>
        </tr>

    %ROW {
        <tr>      <td>$(V_NNAME), $(V_VNAME)</td>
                <td>$(V_KLAPPE)</td>
        </tr>
    %}
    </table>
    %}

%MESSAGE {
    100 : "Die Datei $(DatenLib)/KOLLEGEN ist leer." : continue
%}

%}
```

Die SQL-(Language-Environment)-Funktion Telefonliste führt ein SELECT-Statement aus. Daraus resultiert ein so abgearbeitet wird. %REPORT wird einmalig für jedes Result-Set ausgeführt, %ROW für jede einzelne Zeile. Im % (<table>) und in einer Zeile („table row“ = <tr>) und zwei Spaltenüberschriften („table header“ = <th>) ausgeg zeilenweise die Daten („table data“ = <td>) ausgegeben.

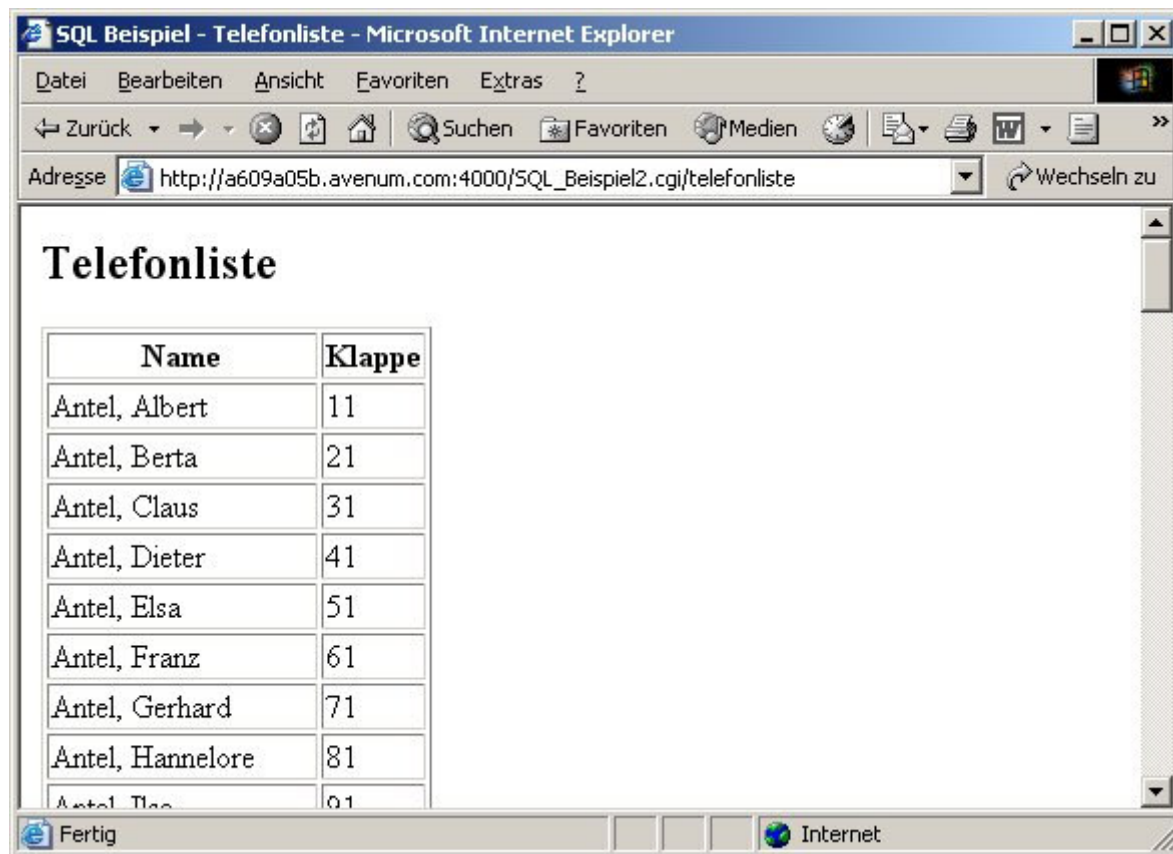
Auf die Variablen der aktuellen Result-Set-Zeile greift man mit einem vorangestellten V\_ zu. Aus dem Dateifeld N Dann werden mit %} der %ROW-Block und die Tabelle am Ende des REPORT-Blocks beendet. Dann folgt das Mo (SQLCODE 100) sowie das Ende der Funktion.

Einfach, oder?

Und der HTML-Abschnitt, der diese Funktion aufruft, ist noch einfacher:

```
%HTML (TELEFONLISTE) {@DTW_ASSIGN(SHOWSQL,"NO")}
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>
<title>SQL Beispiel - Telefonliste</title>
</head>
<body>
<h2>Telefonliste</h2>
@Telefonliste()
</body>
</html>
%}
```

Dieses Makro habe ich als SQL\_Beispiel2.ndm abgespeichert, und rufe es mit dem HTML-Abschnitt TELEFONLISTE



Name	Klappe
Antel, Albert	11
Antel, Berta	21
Antel, Claus	31
Antel, Dieter	41
Antel, Elsa	51
Antel, Franz	61
Antel, Gerhard	71
Antel, Hannelore	81
Antel, Ilse	91

Nicht schlecht, und bei ca. 100 Sätzen kein Problem. Was aber bei tausenden Kollegen?

Wir basteln uns ein Eingabefeld dazu und lassen die Telefonliste nur jene Kollegen anzeigen, auf deren Namen da

Die beiden neuen Einbauten werden also gleich wieder erweitert. Die neue Version habe ich als SQL\_Beispiel3.ndm sich wenig:

Net.data - SQL\_Beispiel3

```
%FUNCTION(DTW_SQL) Telefonliste(IN WHERE) {  
  
    SELECT VNAME, NNAME, KLAPPE  
           FROM $(DatenLib).KOLLEGEN  
           $(WHERE)  
           ORDER BY NNAME, VNAME  
  
    %REPORT {  
        <table border="1">  
        <tr>    <th>Name</th>  
              <th>Klappe</th>  
        </tr>  
  
        %ROW {  
            <tr>    <td>$(V_NNAME), $(V_VNAME)</td>  
                  <td>$(V_KLAPPE)</td>  
            </tr>  
        %}  
    </table>  
    %}  
  
    %MESSAGE {  
        100 : "Kein (passender) Datensatz gefunden." : continue  
    %}
```

```
%}
```

Ein neuer Parameter kam dazu, „WHERE“. Den geben wir im SQL-Statement aus und erweitern es so. Und die Fe Sätze gefunden“ muss nicht gleich bedeuten, dass die Datei leer ist.

Etwas mehr hat sich im HTML-Abschnitt getan:

```
%HTML (TELEFONLISTE) {@DTW_ASSIGN(SHOWSQL,"NO")
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>
<title>SQL Beispiel - Telefonliste</title>
</head>
<body>
<h2>Telefonliste</h2>
<form action="TELEFONLISTE">
Suchen: <input type="text" name="suchfeld" value="@DTW_rHTMLENCODE(suchfeld)">
<input type="submit" name="submitSuchen" value="suchen">
</form>
%IF (" " != submitSuchen)
    %IF (" " != suchfeld)
        @DTW_LOWERCASE(suchfeld, suchfeld)
        @DTW_ASSIGN(myWHERE, "WHERE LCASE(VNAME CONCAT ' ' CONCAT NNAME) LIKE '%@DTW
@Telefonliste(myWHERE)
%ENDIF
</body>
</html>
%}
```

Zunächst wurde ein „Formular“ eingebaut. Als „Action“ wurde der eigene HTML-Abschnitt, also TELEFONLISTE an wird aufgerufen, wenn der Benutzer auf einen Sende-Knopf klickt.

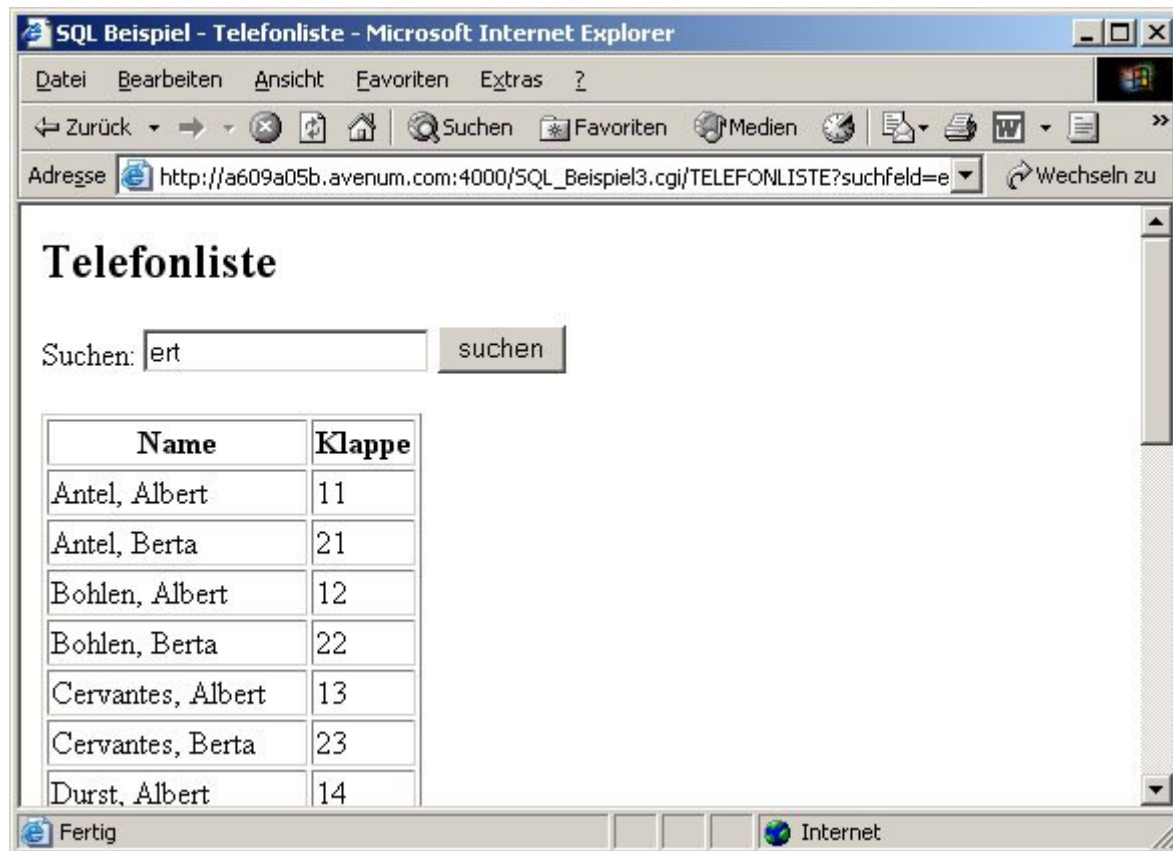
Dann haben wir ein Text-Feld namens „suchfeld“ definiert. Daneben soll ein mit „suchen“ beschrifteter Sende-Kn unser Formular auch wieder.

Wenn bereits etwas in „suchfeld“ eingegeben worden ist, dann wird es in das Feld ausgegeben. Auch hier wieder @DTW\_rHTMLENCODE(suchfeld) statt dem simplen \$(suchfeld) sind Sie auch vor „bösen“ Eingaben wie „</html> Kleiner-Zeichen wird ein harmloses & lt, das dasselbe am Schirm ausgibt, aber vom Browser nicht als HTML-Code

Danach wird abgefragt, ob der Knopf gedrückt wurde – dann ist die Variable „submitSuchen“ – so heißt unser Kn „suchen“). Sie können mehrere Knöpfe mit verschiedenen Aufgaben definieren, taufen Sie sie einfach unterschiedl ab. Es ist nur die Variable des Knopfes gefüllt, auf den geklickt wurde.

Wenn nun etwas ins Suchfeld eingegeben wurde, dann wird eine Hilfsvariable „myWHERE“ mit dem nötigen WHEI zu finden, in denen der Suchbegriff gefunden wird.

Aus dem gemischt geschriebenen „Albert“ und „Einstein“ wird im SQL „albert einstein“ gebildet, das dann mit der übersetzten Suchbegriff verglichen wird.



Nun fehlt uns noch eine wichtige „Kleinigkeit“, das Blättern. Dazu werden wir die Anzeige auf 5 Sätze pro Seite einrichten und die Anzahl der Sätze pro Seite in den SQL-Funktionen definieren.

Als zweite Komponente benötigen wir noch die erste anzuzeigende Zeile.

Wir erweitern wieder unsere beiden Funktionen; die neue Version habe ich als SQL\_Beispiel4.ndm abgespeichert.

Net.data - SQL\_Beispiel4

Ganz oben habe ich die Anzahl der auf einer Seite anzuzeigenden Sätze als globale Variable definiert – wenn Sie einfach den Wert.

```
%DEFINE {
SaetzeAufEinerSeite=      "5"
%}
```

Dann muss der SQL-Funktion die erste anzuzeigende Zeile und die Anzahl Zeilen für das Result-Set übergeben werden. Die reservierten Namen START\_ROW\_NUM und RPT\_MAX\_ROWS werden verwendet. Definieren Sie diese Variablen in den SQL-Funktionen aus und bewirkt zumindest seltsame, wenn nicht falsche Ergebnisse.

Es wird der uralte Trick verwendet, einen Satz mehr zu lesen, als eigentlich nötig. Wir übergeben als „6“ als RPT\_MAX\_ROWS, die die jeweils aktuelle Zeilennummer des Result-Sets enthält, auf diesen Maximalwert ab. Wenn der Wert kleiner ist, werden keine weiteren Sätze gegeben.

Wenn die Anzeige nicht mit „1“ begonnen hat, müssen Vorgänger-Seiten abrufbar sein, dann wird der „Zurück“-Knopf aktiviert. Und wenn unsere Hilfsvariable „Weitere“ mit „1“ gefüllt ist, dann gibt es weitere Seiten und der „Weitere“-Knopf ist aktiviert.

```
%FUNCTION(DTW_SQL) Telefonliste(IN START_ROW_NUM, RPT_MAX_ROWS, WHERE) {
SELECT VNAME, NNAME, KLAPPE
FROM $(DatenLib).KOLLEGEN
$(WHERE)
ORDER BY NNAME, VNAME
}
```

```

%REPORT {
    <table border="1">
        <tr>
            <th>Name</th>
            <th>Klappe</th>
        </tr>

%ROW {
    %IF (ROW_NUM == RPT_MAX_ROWS)
        @DTW_ASSIGN(Weitere, "1")
    %ELSE
        <tr>
            <td>$(V_NNAME), $(V_VNAME)</td>
            <td>$(V_KLAPPE)</td>
        </tr>
    %ENDIF
%}
</table>
<table border="1"><tr><td>
%IF ("1" != START_ROW_NUM)
    <input type="submit" name="submitPAGEUP" value="zurück">
%ENDIF
</td><td>
%IF ("1" == Weitere)
    <input type="submit" name="submitPAGEDN" value="weiter">
%ENDIF
</td></tr></table>
%}

%MESSAGE {
    100 : "Kein (passender) Datensatz gefunden." : continue
%}

%}

```

Und dann müssen wir natürlich auch noch den HTML-Abschnitt ändern, um beim Blättern ein bisschen zu rechnen

```

%HTML (TELEFONLISTE) {@DTW_ASSIGN(SHOWSQL,"NO")}
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>
<title>SQL Beispiel - Telefonliste</title>
</head>
<body>
<h2>Telefonliste</h2>
<form action="TELEFONLISTE">
Suchen: <input type="text" name="suchfeld" value="@DTW_rHTMLENCODE(suchfeld)">
<input type="submit" name="submitSuchen" value="suchen">
%IF (" " != submitSuchen || " " != submitPAGEDN || " " != submitPAGEUP)
    %IF (" " != suchfeld)
        @DTW_LOWERCASE(suchfeld, suchfeld)
        @DTW_ASSIGN(myWHERE, "WHERE LCASE(VNAME CONCAT ' ' CONCAT NNAME) LIKE '%"@DTW
    %ENDIF
    %IF (" " != submitPAGEDN)
        @DTW_ADD(myStart, SaetzeAufEinerSeite, myStart)
    %ENDIF
    %IF (" " != submitPAGEUP)
        @DTW_SUBTRACT(myStart, SaetzeAufEinerSeite, myStart)
    %ENDIF
    %IF ("1" > myStart)
        @DTW_ASSIGN(myStart, "1")
    %ENDIF
    <input type="hidden" name="myStart" value="$(myStart)">
    @DTW_ADD(SaetzeAufEinerSeite, "1", myAnzahlSaetze)
    @Telefonliste(myStart, myAnzahlSaetze, myWHERE)
%ENDIF
</form>
</body>
</html>
%}

```

Zunächst ist das </form> ans Ende der Seite gewandert, um die beiden Blätter-Knöpfe im Formular zu haben.

Ansonsten sind gerade mal 3 Abfragen dazu gekommen:

Wenn nach vorne geblättert werden soll, also die Variable des Knopfes submitPAGEDN gefüllt ist, weil drauf geklickt Anzeige dazugerechnet; vice versa beim Zurückblättern 5 abgezogen.

Dann wird noch geprüft, ob dadurch oder weil „myStart“ noch nie verwendet wurde, „1“ größer als „myStart“ ist. Damit die erste angezeigte Zeile beim nächsten Aufruf zur Verfügung steht, wird sie als versteckte („hidden“) Ein wieder an den Server geschickt.

Und um den erwähnten uralten Trick nutzen zu können, muss die Anzahl Sätze auf einer Seite um 1 erhöht werden. Damit werden Sie schon sehr viele Aufgaben lösen können.

Aber hin und wieder kommt man nicht um einen Programmaufruf herum. Der ist mit Net.Data allerdings wirklich

## Programmaufrufe aus Net.Data heraus

Rufen wir doch einfach mal ein CL-Programm auf, das uns zwei Systemwerte liefern soll.

```
PGM          PARM(&QMODEL &QPRCFEAT)
DCL          VAR(&QMODEL) TYPE(*CHAR) LEN(4)
DCL          VAR(&QPRCFEAT) TYPE(*CHAR) LEN(4)

RTVSYSVAL   SYSVAL(QMODEL) RTNVAR(&QMODEL)
RTVSYSVAL   SYSVAL(QPRCFEAT) RTNVAR(&QPRCFEAT)

ENDPGM
```

## CL-Programm „RTVSYSVALS“

### Net.data - RTVSYSVALS

Zum Aufruf von CL- oder RPG-Programmen dient das Language-Environment „DIRECTCALL“.

```
%FUNCTION(DTW_DIRECTCALL) CallRTVSYSVALS(OUT CHAR(4) MODEL,
                                           CHAR(4) PRCFEAT)

{ %EXEC { /QSYS.LIB/TECHKNOW.LIB/RTVSYSVALS.PGM %} %}
```

Und ein RPG-Programm, das Zahlen addiert, werden wir auch aufrufen:

```
H
D p_Zahl1      S              15P 2
D p_Zahl2      S              15P 2
D p_Summe      S              15P 2
C *ENTRY      PLIST
C             PARM              p_Zahl1
C             PARM              p_Zahl2
C             PARM              p_Summe
C             EVAL      p_Summe = p_Zahl1 + p_Zahl2
C             RETURN
```

## RPGLE-Programm „ADDVALUES“

### Net.data - ADDVALUES

```
%FUNCTION(DTW_DIRECTCALL) CallADDVALUES(IN      DECIMAL(15,2) ZAHL1,
                                           DECIMAL(15,2) ZAHL2,
                                           OUT      DECIMAL(15,2) SUMME)

{ %EXEC { /QSYS.LIB/TECHKNOW.LIB/ADDVALUES.PGM %} %}
```

Und weil wir gerade dabei sind, Sie können natürlich auch selbst „Makro-Funktionen“ in einem Makro definieren.

Die neue Funktion „Rechne“, die nur die beiden Eingabe-Parameter und das Ergebnis des RPG-Programm-Aufrufs

```
%MACRO_FUNCTION Rechne(          IN          Zahl1, Zahl2) {
@CallADDVALUES(Zahl1,Zahl2,Summe)
$(Zahl1)+$(Zahl2)=$(Summe)
%}
```

Diese Funktionen werden nun von diesem HTML-Abschnitt aufgerufen:

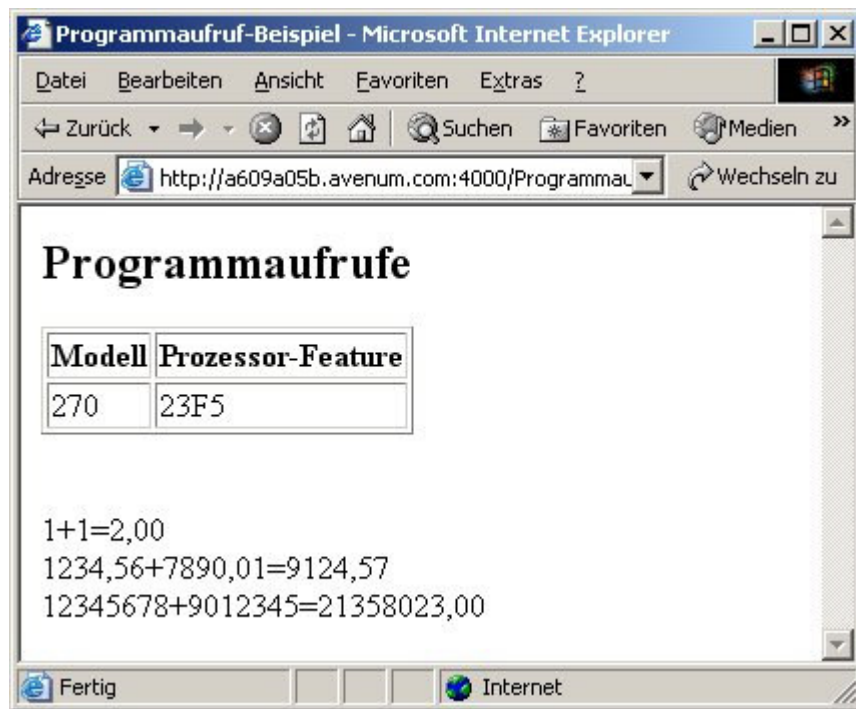
```
%HTML (START) {@DTW_ASSIGN(SHOWSQL,"NO")
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>
<title>Programmaufruf-Beispiel</title>
</head>
<body>
<h2>Programmaufrufe</h2>

@CallRTVSYSVALS(QMODEL,QPRCFEAT)
<table border="1">
<tr><th>Modell</th><th>Prozessor-Feature</th></tr>
<tr><td>$(QMODEL)</td><td>$(QPRCFEAT)</td></tr>
</table>

<br><br>
@Rechne("1","1")<br>
@Rechne("1234,56","7890,01")<br>
@Rechne("12345678","9012345")<br>
</body>
</html>
%}
```

Net.data - Programmaufruf  
Das Makro habe ich als Programmaufruf1.ndm abgespeichert.

Ausgeführt sieht das dann so aus:



Sie können nun SQL-Statements aufrufen, um Daten zu manipulieren, Dateien auflisten und Programme aufrufen

Das alles aus dem Browser heraus; obwohl Sie das Arbeiten hier wohl höchst interaktiv empfinden, für Ihre iSeries

Ich hoffe, ich konnte Ihnen Net.Data näher bringen und bedanke mich für Ihr Interesse!

## Links

Abschließend möchte ich Ihnen noch einige Links liefern, unter denen Sie weitere Informationen zu Net.Data finden Sprache.

### Die Net.Data homepage

<http://www-1.ibm.com/servers/eserver/series/software/netdata/>

Laden Sie sich unter „Library“ die Handbücher am besten als PDF herunter; die „Net.Data Reference“ sollten Sie einen kleinen Teil der verfügbaren Funktionen abhandeln.

Unter „Education“ finden Sie auch etliche Beispiel-Makros.

### Das Net.Data-Forum

<http://www-1.ibm.com/servers/eserver/series/software/netdata/>

Rat und Hilfe finden Sie in diesem Forum. Wenn Sie nicht weiterkommen, suchen Sie mal in diesem Forum; hier können Sie Ihre Frage dann noch immer nicht beantworten können, fragen Sie einfach selbst im Forum. Meist erhalten Sie kompetente Antworten.

### Deutschsprachige Net.Data mailing list

[http://de.groups.yahoo.com/group/commonA\\_NetData/](http://de.groups.yahoo.com/group/commonA_NetData/)

Common Österreich hat eine (noch recht übersichtliche) mailing list zum Thema „Net.Data“ ins Leben gerufen, in die Sie auch hier können Sie Hilfe erwarten, und das in Deutsch; teilnehmen kann jedermann! (Wir bedienen uns hier eine e-Mail-Liste (Umstellung auf unsere iSeries (und natürlich Net.Data!) scheitert bislang nur an Zeitmangel.)

Den Autor Anton Gombkötö erreichen Sie unter  
Avenum Technologie GmbH - Brigittenauer Lände 50-54/6, A-1200 Wien  
Tel. (+43) 1/92101-148, e-Mail: [ag@avenum.com](mailto:ag@avenum.com)