REXX-Skript zum Auslesen einer DB2-Datenbank

Assignment zum Modul:

z/OS-Mainframe-Technologien (ANS42)

26.11.2019

Vladimir Zhelezarov

.....

Studiengang: Digital Engineering und angewandte Informatik - Bachelor of Engineering

AKAD University

Inhaltsverzeichnis

Inhaltsverzeichnis **i**

Abbildungsverzeichnis **ii**

1 Einleitung **1**

2 Voraussetzungen und Test-Umgebung **2**

2.1 Zugang auf dem Mainframe 2

2.2 Vorhandene Datasets 2

2.3 Konventionen 3

3 REXX Skripten **3**

3.1 Hauptrahmen 3

3.1.1 DSNREXX Schnittstelle 3

3.1.2 EXECSQL 4

3.2 REXX-Skript Grundversion 5

3.3 Erweiterung: Eingabeparameter 6

3.4 Erweiterung: Ausgabe in Datei 8

3.5 Vollständiger REXX Skript 12

4 Zusammenfassung **14**

Literaturverzeichnis **iii**

Abbildungsverzeichnis

1 Aufruf mit *TSO EXEC 'PRAK150.REXX.EXEC(V1)'* 6
2 Ausgabe mit der *ASKUSER* Funktion 8
3 Resultierende Datei nach Aufruf mit 'Y' für Leerzeichenreduzierung 10
4 Resultierende Datei nach Aufruf mit 'N' für keine Leerzeichenreduzierung . . 11

1 Einleitung

Die Skript-Sprache REXX von IBM bietet leistungsfähige Möglichkeiten, auf dem Mainframe Aufgaben auszuführen und zu automatisieren sind, an. Weil die verschiedenen Anwendungen Daten oftmals in einer Datenbank speichern, ist eine typische Aufgabe, diese Daten aus der Datenbank zu lesen. Die Datenbank ist nicht als direkt lesbare Datei gedacht, also müssen die Daten zuerst von der Datenbanksoftware abgefragt werden. Eine von der Möglichkeiten dies zu erfüllen, besteht in der Benutzung von REXX Skripten, die eine Schnittstelle zur Datenbank benutzen. Die Entwicklung eines solchen REXX-Skriptes für diese Aufgabe ist das Ziel dieser Arbeit, wobei auch mögliche Erweiterungen beachtet werden.

Um eine DB2-Datenbank auszulesen, muss sie natürlich zuerst vorhanden sein. Diese und andere Voraussetzungen betrachten wir, bevor wir mit dem Skript-Aufbau anfangen können.

Es gibt ein von IBM empfohlenen, bestimmten Rahmen, im welchem ein REXX Skript bleiben muss, um mit der DB2 Datenbank Software zu kommunizieren. Wenn dieser schon vorhanden ist, kann der Skript auch weiter entwickelt werden.

Als typische Aufgaben für die Weiterentwicklung betrachten wir Eingabeparameter, Formatieren der Ausgabe und Ausgabe in Datei. Für das Letzte wird der Skript noch eine andere Schnittstelle benutzen - die Schnittstelle zu TSO.

Als Erfolgskontrolle für die Entwicklung bietet sich an schrittweise zu erweitern und jede neue Version gleich auf dem Mainframe zu testen. Der benötigte Zugang und eine Entwicklungsumgebung als Lernsystem ist freundlicherweise von Uni-Leipzig angeboten.

2 Voraussetzungen und Test-Umgebung

2.1 Zugang auf dem Mainframe

Die Entwicklung findet im Lernsystem des Mainframes von der Universität Leipzig statt, wo begrenzter freier Zugang für Studierende angeboten wird. Der Mainframe ist ein IBM z9 EC, Modell 2094-S08[1]. Das Betriebssystem ist z/OS 2.1.0[2], die DB2 ist Version 12. Die letzten zwei Werte bestimmen auch die relevante Dokumentation von IBM, die als Referenz für die Erstellung der Skripten nötig ist.

Der Zugang verläuft über dem Software-Emulator x3270[3]

2.2 Vorhandene Datasets

Die vorgestellten REXX-Skripten setzen folgende vorhandene Dateien aus dem Mainframe-Lernsystem[4] voraus: eine DB2 Datenbank „D121“, mit Tabelle namens „TAB150“, wo Einträge in der Form „Vorname-Nachname“ zu erwarten sind. Die Felder in der Tabelle sind 20-char breit. Benutzt wurde folgender SQL-Code[5]:

```
CREATE TABLESPACE TABSP150
  IN PRAK150
  USING STOGROUP SYSDEFLT
    PRIQTY 60
    SECQTY 60
    ERASE NO
  BUFFERPOOL BP0
  CLOSE NO;

CREATE TABLE TAB150
 (
    VNAME CHAR(20) NOT NULL,
    NNAME CHAR(20) NOT NULL
 )
    IN PRAK150.TABSP150;
```

Ob die Datenbank-Tabelle gefüllt mit Eingaben ist oder was für Werte diese haben, ist für die Skripten irrelevant. Allerdings wurde die Tabelle mit ein paar Namen vorgesehen, um die Ausgabe der Skripten besser zu demonstrieren:

---

[1] http://jedi.informatik.uni-leipzig.de/de/mainframe.html (eventuell veraltete Info)
[2] gemeldet von ISPF im Menü 7.3 (Variables)
[3] http://x3270.bgp.nu/
[4] Bogdan, M. (o.J.), Internetquelle
[5] Vgl. IBM Corporation (2019 c), S.1648 (CREATE TABLESPACE) und IBM Corporation (2019 c), S.1580 (CREATE TABLE)

```
INSERT INTO TAB150
  VALUES ('VLADIMIR', 'ZHELEZAROV');
INSERT INTO TAB150
  VALUES ('HANS', 'PETER');
INSERT INTO TAB150
  VALUES ('JULIA', 'SCHMIDT');
INSERT INTO TAB150
  VALUES('KARL', 'HEINZ');
```

Außerdem wird ein PDS Dataset benötigt, wo die Skripten ihre Ausgaben schreiben können. Als Einstellungswerte wurden für den Dataset wie folgt übernommen:

Record format	FB
Record length	80
Block size	320
1st extend kilobytes	18
Secondary kilobytes	1
Data set name type	PDS
Units	KB

2.3 Konventionen

- Bei manchen Befehlen besteht die Möglichkeit Abkürzungen zu benutzen. Um die Lesbarkeit zu erhöhen, benutzen die vorgestellten Skripten immer die volle Version.
- Oftmals gibt es viele Wege ein Programmziel zu erreichen. Diese Arbeit hält sich an den von IBM vorgeschlagene Vorgehensweisen.

3 REXX Skripten

3.1 Hauptrahmen

3.1.1 DSNREXX Schnittstelle

Der REXX-Skript kann die DSNREXX Schnittstelle[1] benutzen, um sich bei der DB2 Datenbank anzumelden, SQL-Befehle auszuführen und Daten abzulesen. Der empfohlene Rahmen dafür ist wie folgt[2]:

[1] IBM Corporation (2019 b), S.430
[2] IBM Corporation (2019 b), S.432

```
'SUBCOM DSNREXX'
IF RC<>0 THEN
  S_RC = RXSUBCOM('ADD','DSNREXX','DSNREXX')
ADDRESS DSNREXX
/* ... */
/* Arbeiten mit DSNREXX (CONNECT, EXECSQL und DISCONNECT) */
/* ... */
S_RC = RXSUBCOM('DELETE','DSNREXX','DSNREXX')
```

Der Code funktioniert wie folgt:

- Zeile 1 prüft mit *SUBCOM*[1] ob die *DSNREXX* Schnittstelle in der Umgebung verfügbar ist;
- Das Ergebnis wird in die spezielle Variable RC geladen und kann 0/vorhanden oder 1/nicht-vorhanden sein. Die Zeile 2 prüft diese Variable und Zeile 3 lädt die Schnittstelle, wenn sie nicht schon verfügbar ist;
- Nach dem Befehl *ADDRESS*[2] werden alle nicht-REXX Befehle an der mit dem Parameter angegebenen Schnittstelle weitergeleitet - im diesem Fall an *DSNREXX*;
- Die letzte Zeile entfernt die *DSNREXX* Schnittstelle nach der abgeschlossenen Arbeit.

Der Ablauf bei der Benutzung von *DSNREXX* ist:

- *CONNECT* meldet den REXX Task als Benutzer und verbindet ihn mit der Datenbank. Als Parameter wird der Datenbankname angegeben;
- *EXECSQL* führt SQL Code auf der Datenbank aus;
- *DISCONNECT* beendet die Verbindung.

3.1.2 EXECSQL

Alle SQL Befehle in REXX beginnen mit EXECSQL[3]. Vor der Ausführung der SQL-Anfragen an der Datenbank, kann mit „*SET CURRENT PACKAGESET*“[4] das Sperrverhalten entschieden werden, wenn es dazu kommt, dass auch andere Prozesse auf der Datenbank zugreifen. *DSNREXCS*[5] stellt das Verhalten auf „*cursor stability*“, was eine maximale Konkurrenz mit gleichzeitig Datenintegrität anbietet[6].

[1] IBM Corporation (2013 b), S.259
[2] IBM Corporation (2013 b), S.46
[3] IBM Corporation (2019 b), S.407
[4] IBM Corporation (2019 c), S.2062
[5] IBM Corporation (2019 b), S.433f.
[6] IBM Corporation (2019 a), S.258

Beim Ablesen von Daten aus einer DB2 Datenbank, wird der sogenannte „*cursor*“ benutzt. Das ist ein Objekt, das auf die zurzeit gelesene Zeile zeigt[1]. Die Zeilen werden dann mit *FETCH* nacheinander gelesen, dabei muss sich der Skript kümmern, die abgelesenen Daten in entsprechende Variablen zu speichern.

Anstatt direkt SQL Befehle in *EXECSQL* einzubetten, kann auch zuerst der Befehl einer Variable zugewiesen werden, die dann mit einem *PREPARE* Statement abgelesen wird[2].

3.2 REXX-Skript Grundversion

Mit den obigen Überlegungen sieht der REXX-Skript zum Auslesen der Datenbank in seiner einfachsten Form folgendermaßen aus[3]:

```
/* REXX */
"SUBCOM DSNREXX"
IF RC<>0 THEN
  S_RC=RXSUBCOM('ADD','DSNREXX','DSNREXX')
ADDRESS DSNREXX
"CONNECT" D121
"EXECSQL SET CURRENT PACKAGESET='DSNREXCS'"
SQLSTMT = "SELECT VNAME,NNAME",
  "FROM PRAK150.TAB150;"
"EXECSQL DECLARE C1 CURSOR FOR S1"
"EXECSQL PREPARE S1 FROM :SQLSTMT"
"EXECSQL OPEN C1"
DO WHILE SQLCODE=0
  "EXECSQL FETCH C1 INTO :X,:Y"
  IF SQLCODE<>0 THEN
    LEAVE
  SAY X||Y
END
"DISCONNECT";
S_RC=RXSUBCOM('DELETE','DSNREXX','DSNREXX')
```

[1] IBM Corporation (2019 b), S.747
[2] IBM Corporation (2019 c), S.1959-1960
[3] In Anlehnung an Bogdan, M. (o.J.), Internetquelle

Abbildung 1: Aufruf mit *TSO EXEC 'PRAK150.REXX.EXEC(V1)'*

3.3 Erweiterung: Eingabeparameter

Es fällt bei der Ausgabe auf, dass die Felder viel Abstand zwischen einander haben, dafür aber als Spalten aneinander ausgerichtet sind. Das liegt daran, dass sie eine Länge von 20-char haben und was nach den Einträgen übrig bleibt, wird mit Leerzeichen ausgefüllt. Wenn aber diese Art von Ausgabe nicht erwünscht ist, kommt die REXX Funktion *STRIP*[1] zur Hilfe - sie entfernt alle Leerzeichen vom Anfang und vom Ende einer Zeichenkette. Somit kann die Zeile für die Ausgabe so geändert werden:

```
SAY STRIP(X) STRIP(Y)
```

[1]IBM Corporation (2013 b), S.114

Noch ein Schritt weiter führt die Überlegung, dass dieses Verhalten (mit/ohne Leerzeichen) ausgewählt werden könnte. *PARSE ARG*[1] ist nützlich, um die Aufrufargumente zu dem Skript zu übernehmen. Von TSO aus werden diese einfach nach dem Name des Datasets als Parameter angegeben:

```
TSO EXEC 'PRAK150.REXX.EXEC(V2)' 'PARAM1,PARAM2'
```

Ob der Benutzer Parameter angegeben hat, prüft REXX mit der ARG()[2] Funktion, die die Zahl der Aufrufparameter zurück gibt.

Wenn keine Parameter zu finden sind, kann der Skript den Benutzer nach solchen fragen.

Es stellt sich die Frage, was zu tun ist, wenn der Benutzer andere als die zugelassene Parameterwerte eingibt - das muss auch geprüft werden und bei Bedarf muss die Anfrage wiederholt werden.

Um Code-Wiederholungen zu vermeiden und gleichzeitig die Lesbarkeit zu verbessern, können für solche Ziele REXX Funktionen benutzt werden. Das allgemeine Format einer Funktion mit Rückgabewert ist[3]:

```
FUNKTIONNAME:
PARSE ARG VARIABLE1, VARIABLE2 /* ... */
/* ... Arbeit ... */
RETURN RÜCKGABEWERT
```

Um sich die Auswahl zu merken, führt unser REXX Skript die Variable *DOSTRIP* ein, die die Werte 'Y'/'N' haben kann.

```
IF ARG()==0 THEN
  DOSTRIP=ASKUSER()
ELSE DO
  PARSE ARG DOSTRIP
  IF (DOSTRIP<>'Y' & DOSTRIP<>'N') THEN
    DOSTRIP=ASKUSER()
END
```

Die *ASKUSER* Funktion sieht dann so aus:

```
ASKUSER:
DO WHILE (ANSWER<>'Y' & ANSWER<>'N')
  SAY "STRIP SPACES (Y/N)?"
  PULL ANSWER .
END
RETURN ANSWER
```

Die *DOSTRIP* Variable kann dann bei der Ausgabe abgefragt werden und dementsprechend mit oder ohne Leerzeichen die Daten ausgeben.

[1] IBM Corporation (2013 b), S.65-66
[2] IBM Corporation (2013 b), S47-48
[3] IBM Corporation (2013 c), S.75-76

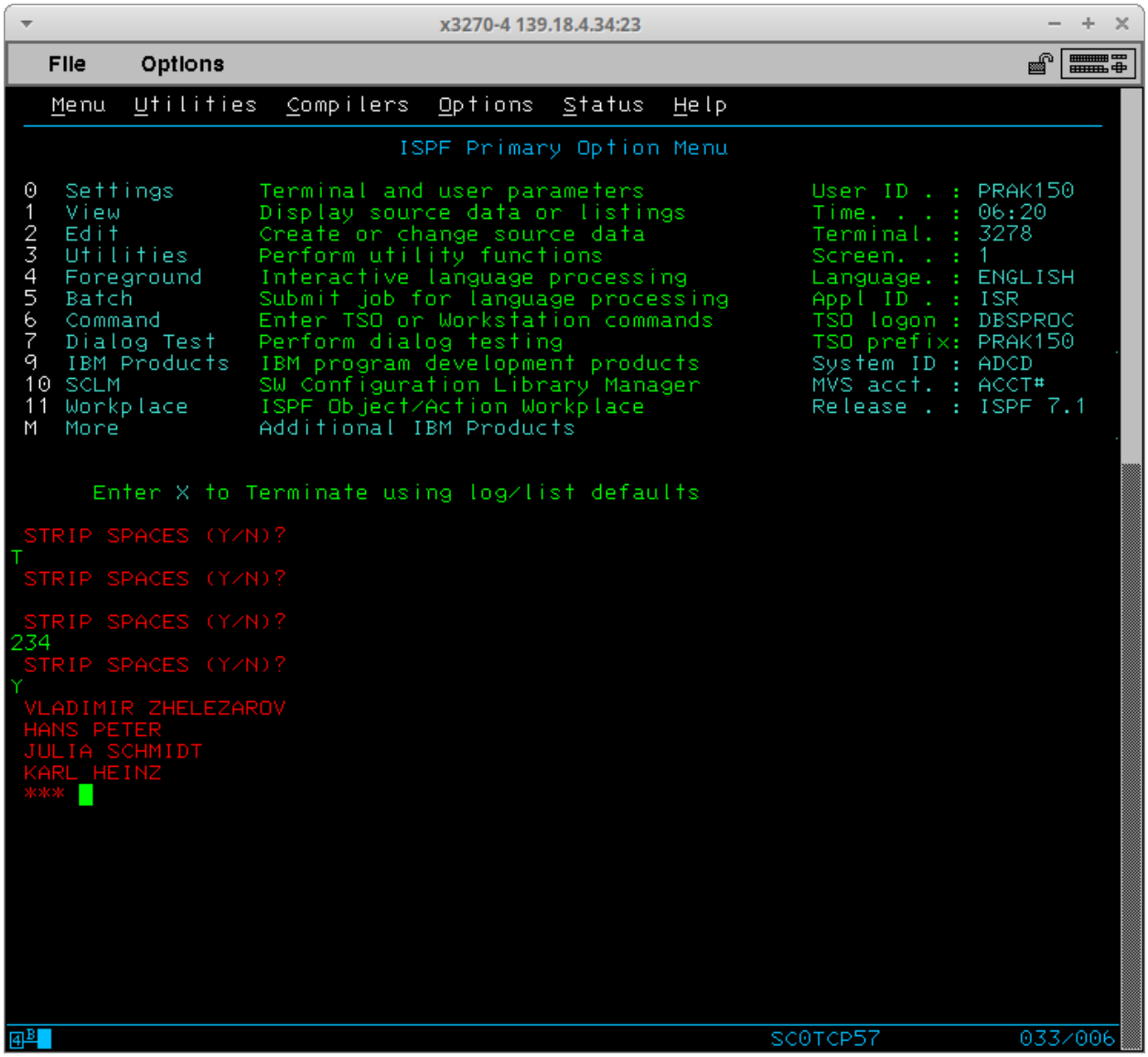

Abbildung 2: Ausgabe mit der *ASKUSER* Funktion

3.4 Erweiterung: Ausgabe in Datei

Bis jetzt hat der Skript immer die Daten auf dem Bildschirm angezeigt. Vorausgesetzt das ist unerwünscht und eine dauerhafte Speicherung in Datei ist bevorzugt, braucht der Skript noch eine Funktion, die die ausgelesenen Daten in einem Dataset speichert.

Zuständig für die Arbeit mit Dateien über TSO bei REXX ist der *EXECIO* Befehl. Er kann Informationen aus einem Dataset in Stack einlesen und umgekehrt - Daten aus dem Stack in Dataset speichern[1]. Für unsere Ziele braucht der Skript den *DISKW* Parameter um Daten in Dataset zu speichern. Voraussetzung dafür ist, dass das Dataset zuerst mittels *ALLOCATE* einer Zugriffs-variable zugeteilt wird. Über diese wird dann der Zugriff durchge-

[1]IBM Corporation (2013 c), S.152

führt. ALLOCATE hat den Parameter SHR, womit angegeben wird, dass das Dataset schon existiert und keine exklusive Benutzung erforderlich ist[1]. Wichtige Parameter bei *DISKW* sind, ob das Dataset nach der Arbeit wieder geschlossen werden muss (*FINIS*) und welche Variable die ausgelesene Daten speichern wird (*STEM*)[2]. Wenn der Prozess durchgeführt ist, kann die Zugriffs-variable mittels *FREE*[3] freigegeben werden. Weil alle diese drei Befehle - ALLOCATE, EXECIO und FREE - über TSO laufen, muss natürlich im Skript dies auch angegeben werden. Genau wie bei der *DSNREXX* Schnittstelle, erfolgt das über *ADDRESS*.

Alle Kommandos schreiben das Ergebnis ihrer Ausführung wie gewohnt in der RC-Variable mit 0 für Erfolg und 1 für Fehler. Diese fragt der Skript ab, um bei Probleme in der Ausführung nicht weiter zu laufen.

Somit besteht die *EXPORTDATA* Funktion in unserem REXX Skript aus folgendes:

```
EXPORTDATA:
PARSE ARG DATA
ADDRESS TSO
DSNNAME='PRAK150.REXX.OUT(OUT)'
"ALLOCATE FILE(OUTPUTDD) DATASET('"DSNNAME"') SHR"
IF RC<>0 THEN
  RETURN RC
"EXECIO * DISKW OUTPUTDD (STEM DATA. FINIS)"
IF RC<>0 THEN
  RETURN RC
"FREE FILE(OUTPUTDD)"
RETURN RC
```

Die nächste Entscheidung ist, ob diese Funktion für jede ausgelesene Reihe aus der Datenbank aufgerufen oder alles auf einmal übergeben wird. Um ein ständiges Wechsel zwischen den Schnittstellen *DSNREXX* und *TSO* zu vermeiden, kann der Skript die zweite Variante benutzen. Dafür speichert er zuerst alle ausgelesenen Reihen in einem Array, der nachher an *EXPORTDATA* übergeben wird.

[1]IBM Corporation (2013 a), S.17
[2]IBM Corporation (2013 c), S.155-156
[3]IBM Corporation (2013 a), S.145

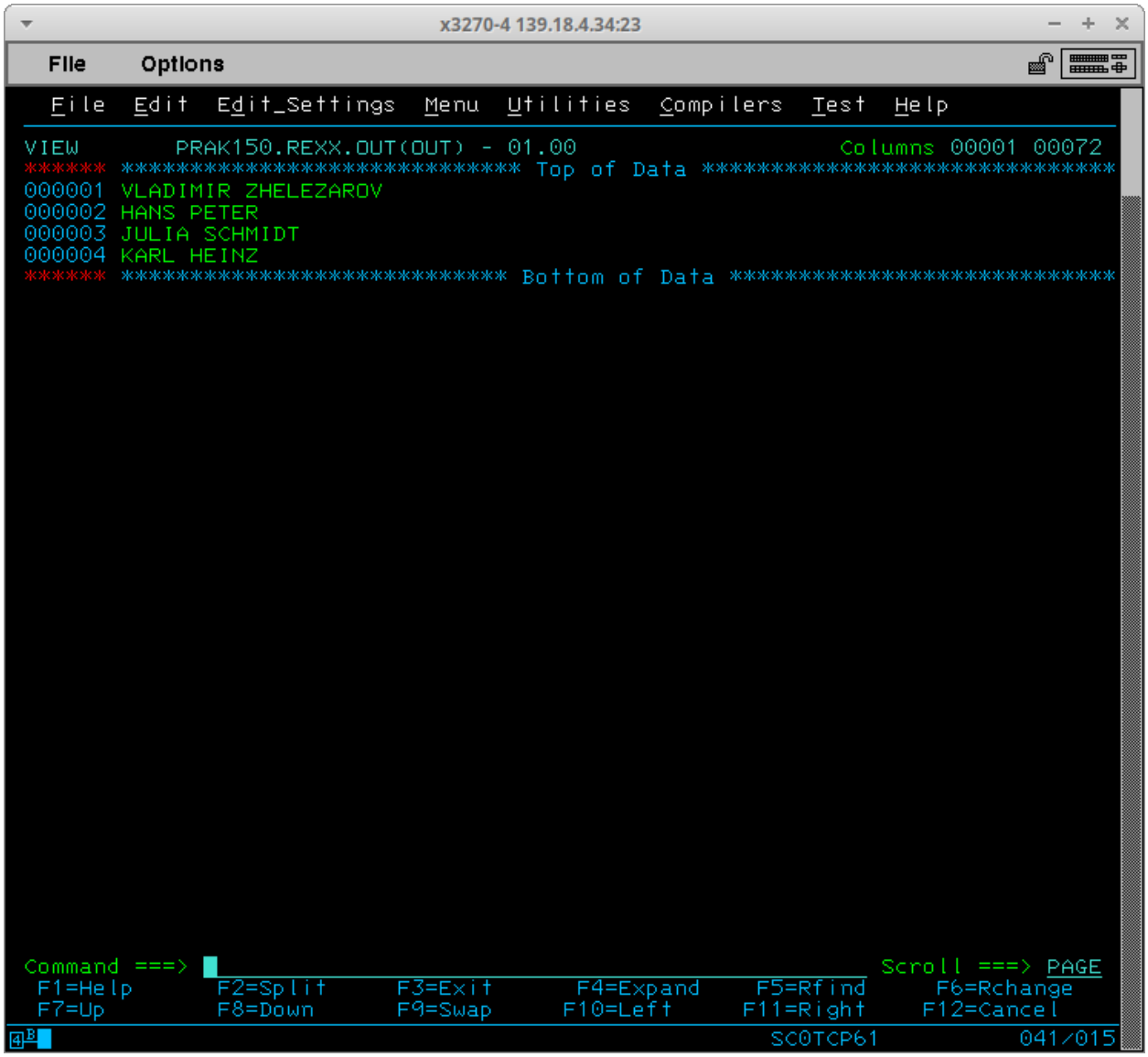

Abbildung 3: Resultierende Datei nach Aufruf mit 'Y' für Leerzeichenreduzierung

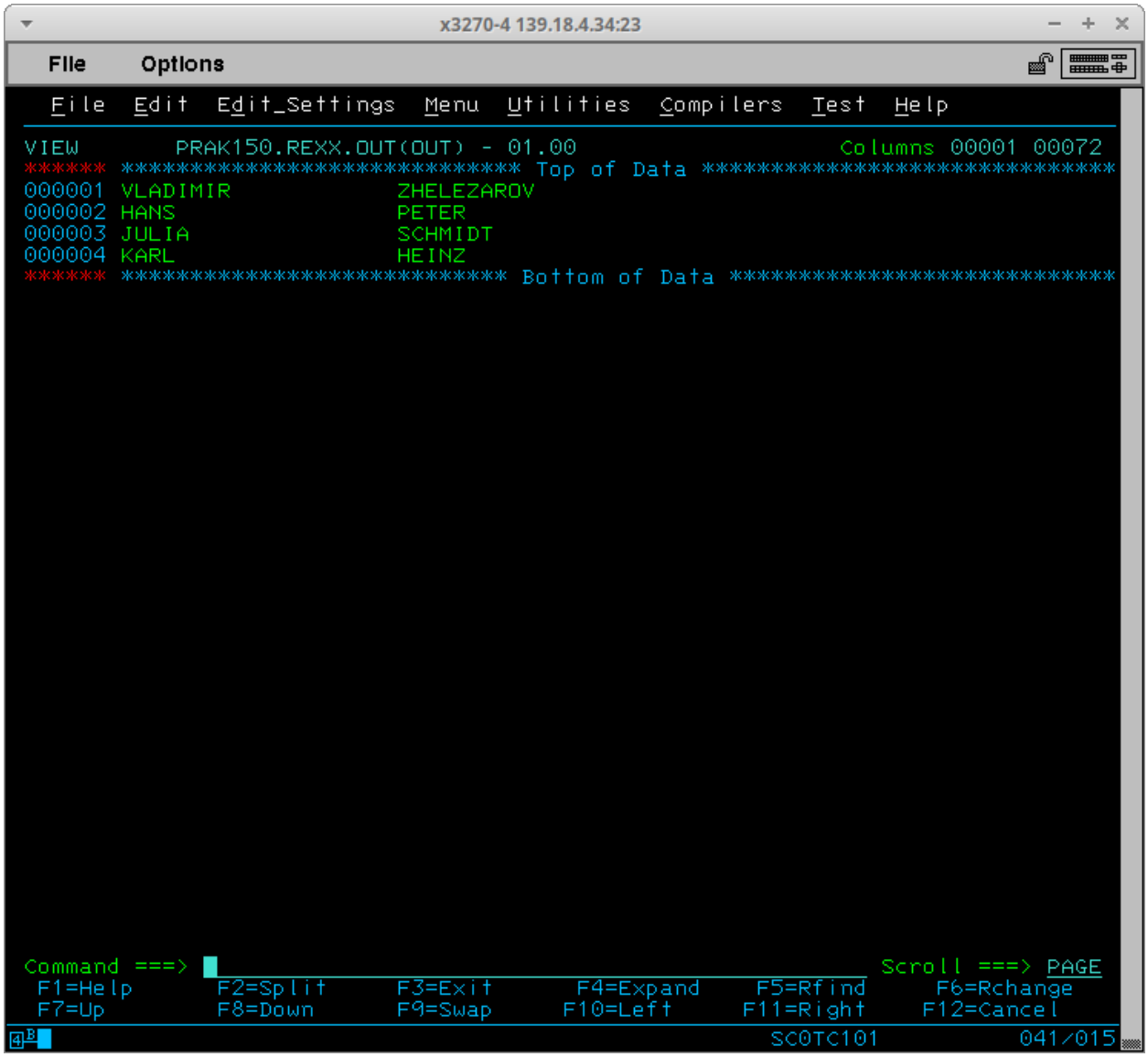

Abbildung 4: Resultierende Datei nach Aufruf mit 'N' für keine Leerzeichenreduzierung

3.5 Vollständiger REXX Skript

```
000100 /* REXX MAIN */
000200 IF ARG()==0 THEN
000300   DOSTRIP=ASKUSER()
000400 ELSE DO
000500   PARSE ARG DOSTRIP
000600   IF (DOSTRIP<>'Y' & DOSTRIP<>'N') THEN
000700     DOSTRIP=ASKUSER()
000800 END
000900 "SUBCOM DSNREXX"
001000 IF RC<>0 THEN
001100   S_RC=RXSUBCOM('ADD','DSNREXX','DSNREXX')
001200 ADDRESS DSNREXX
001300 "CONNECT" D121
001400 "EXECSQL SET CURRENT PACKAGESET='DSNREXCS'"
001500 SQLSTMT = "SELECT VNAME,NNAME FROM PRAK150.TAB150;"
001600 "EXECSQL DECLARE C1 CURSOR FOR S1"
001700 "EXECSQL PREPARE S1 FROM :SQLSTMT"
001800 "EXECSQL OPEN C1"
001900 I=1
002000 DO WHILE SQLCODE=0
002100   "EXECSQL FETCH C1 INTO :X,:Y"
002200   IF SQLCODE<>0 THEN
002300     LEAVE
002400   IF DOSTRIP=='Y' THEN
002500     DATA.I=STRIP(X) STRIP(Y)
002600   ELSE
002700     DATA.I=X||Y
002800   I=I+1
002900 END
003000 "DISCONNECT";
003100 S_RC=RXSUBCOM('DELETE','DSNREXX','DSNREXX')
003200 RES=EXPORTDATA(DATA)
003300 IF RES==0 THEN
003400   SAY 'DONE'
003500 EXIT
003600 /* END MAIN */
003700 /* ASKUSER */
003800 ASKUSER:
003900 DO WHILE (ANSWER<>'Y' & ANSWER<>'N')
004000   SAY "STRIP SPACES (Y/N)?"
004100   PULL ANSWER .
004200 END
004300 RETURN ANSWER
```

```
004400 /* EXPORTDATA */
004500 EXPORTDATA:
004600 PARSE ARG DATA
004700 ADDRESS TSO
004800 DSNNAME='PRAK150.REXX.OUT(OUT)'
004900 "ALLOCATE FILE(OUTPUTDD) DATASET('"DSNNAME"') SHR"
005000 IF RC<>0 THEN
005100   RETURN RC
005200 "EXECIO * DISKW OUTPUTDD (STEM DATA. FINIS)"
005300 IF RC<>0 THEN
005400   RETURN RC
005500 "FREE FILE(OUTPUTDD)"
005600 RETURN RC
```

4 Zusammenfassung

Es wurde ein REXX-Skript schrittweise aufgebaut, der einfache, voreingestellte Daten aus einer DB2 Datenbank auslesen und in einem Dataset speichern kann.

Voraussetzung dafür ist ein Zugang zu einem Mainframe mit der benötigten Software und eine vorhandene Datenbank mit einem bestimmten Name.

Der von IBM empfohlene Rahmen für die Arbeit mit DB2 basiert auf die Benutzung von der *DSNREXX* Schnittstelle, wodurch die Daten in REXX Variablen abgelesen werden. Als einfaches Beispiel für die Formatierung dieser Daten hat der Skript mit *STRIP* die Leerzeichen am Ende entfernt. Um das Ganze etwas mehr interaktiv zu gestalten, wurden dem Skript auch Eingabeparameter zugefügt, welche die Formatierung bestimmen. Durch die Schnittstelle zu TSO speichert der Skript seine Ausgabe direkt in einem Dataset.

Um diese Arbeit und die Größe des Skriptes in Grenzen zu halten, ist der Skript stark vereinfacht. Er erwartet eine vorhandene Datenbank und für die Speicherung ein vorhandenes Dataset. Wenn diese fehlen, wird die Ausführung einfach abgebrochen, ohne ein Ausweg, wie zum Beispiel die Ausgabedatei zu erstellen, zu suchen. Fehlermeldungen und allgemeine Fehlerbehebung sind überhaupt nicht im Skript vorhanden.

Der Name von der Datenbank, die Tabelle und ihren Felder sind vorgegeben und unterliegen keiner Anpassung vom Benutzer. Die Kommunikation mit dem Benutzer ist stark vereinfacht und besteht aus der Abfrage des Eingabeparameters. Der Code für die erlaubte Zeichen für die Benutzereingabe wiederholt sich, und könnte als Verbesserung in eigene Funktion umgelagert werden.

Mit diesen Beschränkungen ist der Skript voll funktionsfähig und erfüllt seine Aufgaben bei allen durchgeführten Test in der vorhandenen Umgebung mit den vorgegebenen Dateien. Als weitere Entwicklung würden sich mehr Checks anbieten, sowie mehr Kommunikation mit dem Benutzer durch mehr Eingabeparameter und mehr Kontrolle über der Ausführung.

Literaturverzeichnis

IBM Corporation (2019 a)
Db2 12 for z/OS: Introduction to Db2 for z/OS, o.O.

IBM Corporation (2019 b)
Db2 12 for z/OS: Application Programming and SQL Guide, o.O.

IBM Corporation (2019 c)
Db2 12 for z/OS: SQL Reference, o.O.

IBM Corporation (2013 a)
z/OS: TSO/E Command Reference. Version 2 Release 1, o.O.

IBM Corporation (2013 b)
z/OS: TSO/E REXX Reference. Version 2 Release 1, o.O.

IBM Corporation (2013 c)
z/OS: TSO/E REXX User's Guide. Version 2 Release 1, o.O.

Bogdan, M. (o.J.)
Universität Leipzig. Abteilung Technische Informatik. z/OS Training Tool Leipzig
https://mainframe.informatik.uni-leipzig.de/zottl/Lernsystem/index.php (Zugriff am 26.11.2019)