Reportgenerator REP
Release 2.9.5a

Handbuch

1 Vorwort

REP wurde entwickelt und wird vertrieben von:

Nonne & Schneider Informationssysteme GmbH

Fridrich-List-Str. 31. 11

35398 Gießen

Tel: 0641 / 97477-0

Fax: 0641 / 97477-77

Voraussetzungen für den Einsatz von REP:

REP steht zur Verfügung auf den gängigen UNIX-Versionen und Windows.

Bei der Verwendung des REP benötigen Sie die entsprechenden Datenbank-Runtime-Systeme. Im Falle von Informix 4.x sind dies z.B. ESQL/C-RT, sowie das jeweils verwendete Informix-Backend SE oder ONLINE.

Die verfügbaren Plattformen entnehmen Sie bitte unserer Portierungsliste, die Sie direkt bei dem für Sie zuständigen Vertriebsmitarbeiter erhalten können.

Die REP-Runtime-Version besteht aus *repgo*.

Die REP-Development-Version umfaßt *repprep* und *librepgo.a.*

Den genauen Nutzungsumfang entnehmen Sie bitte der REP-Lizenzvereinbarung, die Sie über unseren Vertrieb erhalten können.

Im Laufe der Entwicklung des Produkts können Leistungsmerkmale hinzugefügt, bzw. verändert werden oder entfallen.

Handbuchversion: 1.1 Januar 2007 zu REP-Version 2.9.5a

© **Nonne & Schneider Informationssysteme GmbH, 1992-2007**

Jegliche Vervielfältigungen dieses Buches, Übersetzungen, Nachdrucke u.dgl., auszugsweise und auch gesamt, sind nur mit ausdrücklicher Genehmigung des Herstellers gestattet.

Die Nichterwähnung von Warenzeichen, Gebrauchsmustern etc. berechtigt nicht zu der Annahme, eine Ware, Name etc. sei frei.

2 Einleitung

1 Was ist ein Reportgenerator?

Die Notwendigkeit in Verwaltung und Technik immer größere Datenmengen zu verarbeiten, hat zur Entwicklung von Datenbanksystemen geführt.

Mit auf das jeweilige Datenbanksystem abgestimmten Sprachen und Werkzeugen können Daten nach unterschiedlichen Kriterien gespeichert, abgefragt, geändert und gelöscht werden.

Als einheitliche Abfragesprache ist von der X/OPEN-Gruppe SQL (Structured Query Language) definiert worden.

Die Bereitstellung der Daten in der von dem Benutzer geforderten Form, z.B. Drucken von Lohnabrechnungen, Artikellisten, etc., erfolgt über problemorientierte Spezialsprachen (Werkzeuge, Tools).

Eines der Werkzeuge, die hier eingesetzt werden, ist der Reportgenerator.

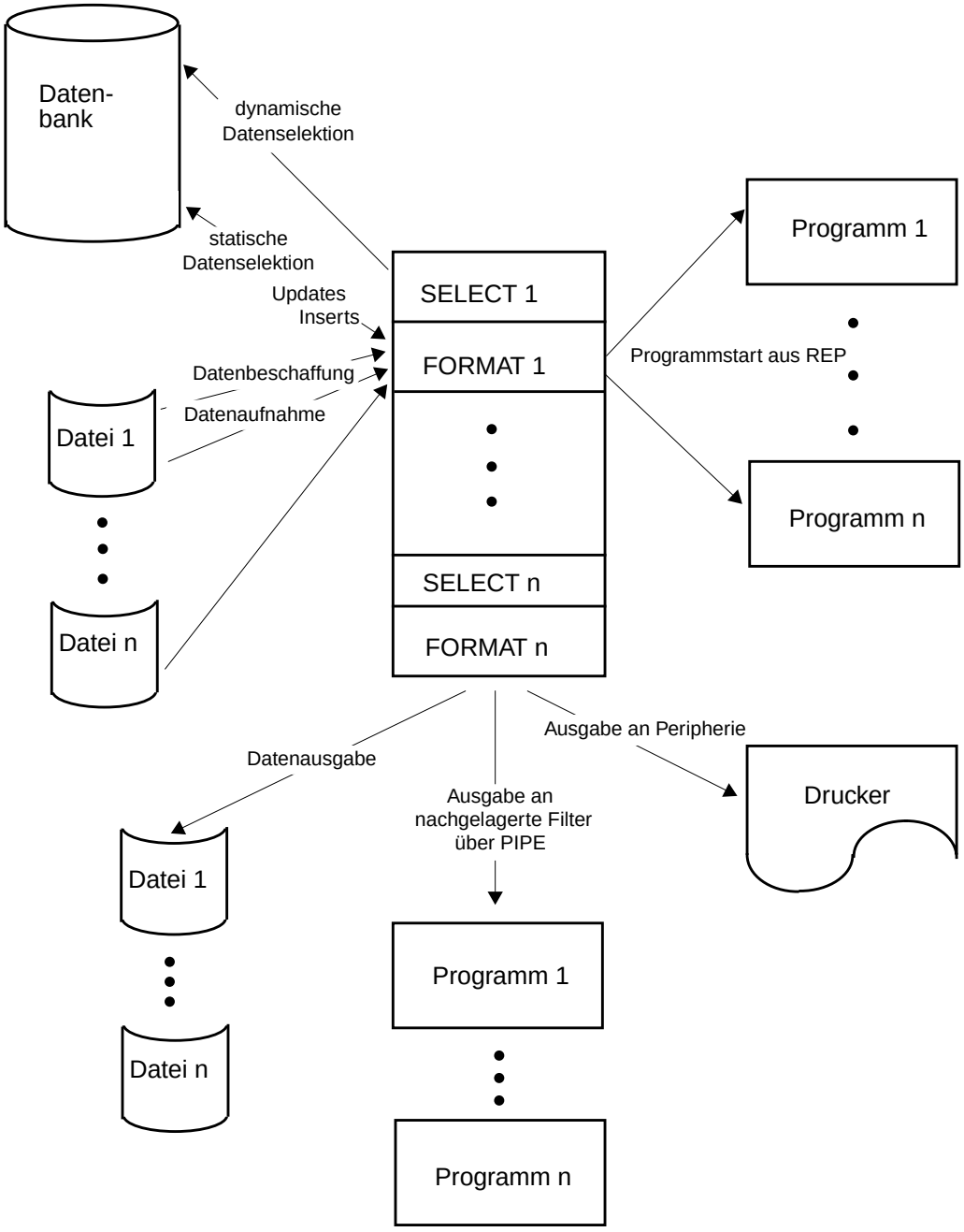
2 Der Reportgenerator REP

Der Report-Generator REP ist ein vollständig in der Programmiersprache C realisierter Listenprogramm-Generator für relationale Datenbanksysteme.

Derzeit wird er für eine Vielzahl von UNIX-Plattformen angeboten. REP wird heute überwiegend im Zugriff auf das Datenbanksystem Informix angewendet, doch existiert auch eine Portierung auf ORACLE, ADABAS D (ehemals DDB/4). Portierungen auf andere POSIX-kompatible Plattformen (Windows/NT, MPE, VMS, ...) und andere Datenbanksysteme sind auf Bedarf hin möglich.

REP bietet die Möglichkeit, Listen unterschiedlichster Art zu erzeugen. Die REP-Syntax hat große Ähnlichkeiten mit der des Informix-Reportgenerators ACE (weitgehende Aufwärtskompatibilität), enthält jedoch beträchtliche Erweiterungen, die seinen Einsatz auch dort erlauben, wo heute häufig noch auf der Embedded-SQL Ebene aufwendig und nicht binaerkompatibel entwikkelt werden muß.

Neben der listenorientierten Ausgabe bietet REP auch die selten in dieser Kombination anzutreffende Möglichkeit der Formulareinbettung und der formulargesteuerten Ausgabe.



Datenzugriff unter REP

3 Notation

Zur Beschreibung der Syntax werden folgende Darstellungen verwendet:

Raster	kennzeichnet die Syntaxdefinitionen.
GROSS	Schlüsselwort, das sowohl in Klein- als auch in Großbuchstaben verwendet werden kann.
klein	Parameter, für die Werte eingegeben werden müssen.
[ ]	In diesen Klammern eingeschlossene Angaben sind optional. Werden die Klammern jedoch fett gedruckt, dann sind sie Bestandteil einer Angabe.
{ }	In geschweiften Klammern eingeschlossene Angaben sind alternativ.
_____	Die Unterstreichung kennzeichnet einen Standardwert.
...	Wiederholungszeichen
...,	Wiederholungszeichen, wobei die Angaben durch Komma getrennt werden.

Die in den Beispielen verwendeten Umlaute ä, ö, ü sind im Code als ae,... , ß als ss darzustellen.

4 REP im Überblick

1 Report-Generator REP

REP ist ein Werkzeug, welches Sie in die Lage versetzt, Listen zu erstellen und Datenbestände in mannigfaltiger Art zu manipulieren. Eine Liste wird aus Daten generiert, die aus den Tabellen einer Datenbank entnommen werden.
In der Regel werden wohl Tabellen und Datenbank sowie die entsprechenden Daten existieren, wobei dies jedoch nicht zwingend ist. REP ermöglicht Ihnen zu jedem Zeitpunkt sowohl lesend als auch schreibend auf die Datenbestände der selektierten Datenbank zuzugreifen.

Der Report-Generator REP setzt sich aus 2 Komponenten zusammen:

- **repprep**
- **repgo**

1.1 Übersetzen mit repprep

Nachdem Sie ein Listen-Quellprogramm geschrieben haben, müssen Sie dieses mit dem REP-Compiler "repprep" übersetzen.

repprep [-w] [-o] [-m]"*datei* "[.rep]

datei	Name des Quellprogramms
-w	wird dieser Schalter angegeben, dann wird die Ausgabe von Warnungen unterdrückt.
-o	wird dieser Schalter angegeben, dann werden Zeilennummernangaben in der .rpp Datei unterdrückt. repgo kann den Report so etwas schneller abarbeiten. Er ist jedoch dann nicht in der Lage bei einem Fehler die Zeilennummer mit auszugeben.
-m	Dies ist ein Sicherheitsschalter. Er erlaubt die Ausführung von modifizierenden Datenbank Statements innerhalb eines REP Skripts.

Erfolgt keine Fehlermeldung durch **repprep**, so konnte Ihr Programm erfolgreich compiliert werden. Es liegt nun ein dem **repgo** verständlicher Zwischencode vor ("datei".rpp).

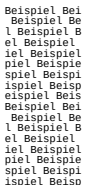
1.2 Starten mit repgo

Nach erfolgter Compilierung mittels “repprep” können Sie Ihr ablauffähiges REP-Programm mit “repgo” starten. Das Kommando zum Starten des Reports hat folgende Form:

repgo [**-q**] [**-h**] [**-F**] [**-s** <SQL-Anweisung>] [**-w** <*customization-flags*>] [**-d** <*db*>] [**-p**<*seitengröße*>] [**-a**] [*-v connectstring*] [**-S** *connectstring* cpi] [**-D** *printparam*] [**-f** *datei*] [-l] [**-U** *user*] [**-P** *password*] *datei* [*parameter*]

Hierbei haben die einzelnen Parameter folgende Bedeutung:

-q	Es werden keine Meldungen ausgegeben.
-h	Es erfolgt keine Ausgabe, wenn kein Satz gefunden wird.
-F	Unmittelbar vor einem Seitenwechsel erfolgt ein Formfeed.
-w <*customization-flags*>	Spezifiziert Customization Flags.
-d *db*	Die Datenbank[1] db wird verwendet.
-p <*seitengröße*>	Die angegebene Seitengröße[2] wird verwendet.
-a	Es werden keine Attributte ausgegeben
-s	Die angegebenen SQL-Anweisungen werden vor dem Start des Hauptselects ausgeführt.
-v <*connectstring*>	Baut Verbindung zu einem VisualREP Server auf. (Näheres hierzu im VisualREP Handbuch)
-S <*connectstring cpi*>	Baut Verbindung zu einem VisualREP Server im Simple Moduls auf. (Näheres hierzu im VisualREP Handbuch)
-D <*printparam*>	Definert Druckparameter für den Ausdruck mit VisualREP.(Näheres hierzu im VisualREP Handbuch)
-f <*datei*>	Legt eine Ausgabedatei für den Ausdruck mit VisualREP fest.
-l	Bei Angabe dieses Schalters werden alle Fehlermeldungen auch in das VisualREP Logfile ausgegeben.
-U <user>	Definiert einen Benutzernamen für die Datenbankverbindung
-P <pass>	Definiert ein Password für die Datenbankverbindung.
datei	Die zu übersetzende .rpp-Datei (Die Endung .rpp muß nicht angegeben werden).
parameter [[*param1*, . . .]]	Werte[3], die der Report als Parameter verwenden kann.



Der Report "*auftrag*" soll unter Verwendung der Datenbank *m_db* gestartet werden. Die Seitengröße soll *66* betragen. Wird kein Satz gefunden, soll keine Ausgabe erfolgen. Als Parameter werden die Kundennummern *1* und *99999* mitgegeben:

repgo *-d m_db -h -p66 auftrag 1 99999*

1.3 Customization Flags

Mit den nachfolgenden Schaltern, welche hinter dem Flag *-w* angegeben werden, können Sie festlegen, wie REP in verschiedenen Situationen reagieren soll. Die Reihenfolge der Schalter spielt bei der Angabe keine Rolle.

FORMAT-Ausdrücke

f	Bei bestimmten FORMAT-Ausdrücken erfolgt im Fehlerfall eine Warnung. Z.B. eine Anweisung enthält eine Division durch 0: LET x=y/0 Wird dieses Flag nicht angegeben, wird im Fehlerfall abgebrochen.

1. Kap. 2 Seite 17 Struktur eines REPORTS Ermitteln der Version
2. Kap. 2 Seite 32 Randgrößen
3. Kap. 1 Seite 22 Variablen-Gruppen PARAM

k Stellt eine dreiwertige Logik bei IF-Anweisungen ein. Ergibt bei der IF-Anweisung die Bedingung NULL, so wird weder der IF noch der ELSE-Zweig ausgeführt. Zusätzlich wird eine Warnung nach Fehlerkanal geschrieben.

EXEC SQL-Anweisungen

e Bei bestimmten EXEC-SQL Anweisungen erfolgt im Fehlerfall eine Warnung.

EXEC-SQL DECLARE

EXEC-SQL OPEN

EXEC-SQL FETCH

EXEC-SQL CLOSE

EXEC-SQL FREE

Wird dieses Flag nicht angegeben, wird im Fehlerfall abgebrochen.

h Der Cursor der SELECT-Anweisung im SELECT-Abschnitt und selbst definierte Cursor werden mit der Option “with hold” deklariert. Dadurch ist es erlaubt, EXEC SQL Anweisungen im FORMAT-Abschnitt mit einem “COMMIT WORK” zu bestätigen, ohne daß der Cursor geschlossen wird.
Achtung: Es ist unbedingt darauf zu achten, daß eigene Cursor vor Beendigung des Reports korrekt geschlossen und freigegeben werden.

Ausgabe-Anweisungen

a Bei folgenden Ausgabe-Anweisungen erfolgt im Fehlerfall eine Warnung:

MOVEYX

CALLYX

SAVEYX

SET ATTRIBUTES

PRINT (nur im PAGEMODE)

In den folgenden Fällen wird nur gewarnt:

- Rekursion bei der Ausführung des PAGE TRAILERS

- Überschreiben der virtuellen Seite

Wird dieses Flag nicht angegeben, wird im Fehlerfall abgebrochen.

Die folgenden Schalter nehmen Einfluß auf das Verhalten von REP beim Formatieren einer Seite und dem Lesen eines Datensatzes.

Seitenwechsel

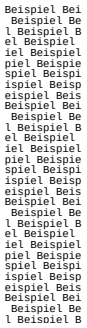
s Die Ausgabe des PAGE-HEADERS erfolgt ausschließlich bei ausgabeorientierten Anweisungen. Wird dieses Flag nicht angegeben, dann wird unmittelbar vor Abarbeitung einer Ortsangabe ein Seitenwechsel vollzogen.

Letzte Seite

l Existiert keine PAGE-TRAILER Ortsangabe und wurde die letzte Seite nicht vollständig beschrieben, so wird sie nicht mit Leerzeichen aufgefüllt. Wird dieses Flag nicht angegeben, wird die letzte Seite vervollständigt.

Satzsperren

i	Satzsperren werden ignoriert.
w	REP wartet auf die Freigabe des Satzes.



Tritt beim Positionieren auf der Seite ein Fehler auf, so soll nur gewarnt, ansonsten abgebrochen werden. Weiterhin sollen Satzsperren überlesen werden und die letzte Seite nicht mit Leerzeilen aufgefüllt werden.

repgo -w ail -d m_db move_rep

Tritt beim Positionieren auf der Seite ein Fehler auf, so soll keine Warnung ausgegeben werden. Das gleiche gilt für eine EXEC SQL Anweisung im Fehlerfall.

repgo -w ea -q -d m_db move_rep

Achtung:

Ist zusätzlich der Schalter -q gesetzt, werden die Warnungen nicht ausgegeben.

Es wird nur bei einem Abbruch eine Fehlermeldung geschrieben, ansonsten erscheint im Fehlerfall keine Meldung. Ist der Schalter -w nicht gesetzt, so wird im Fehlerfall grundsätzlich abgebrochen.

WARNUNGEN SIND AUCH FEHLER und sollten grundsätzlich nicht ignoriert werden.

1.4 Ermitteln der Version

Um Supportanfragen bearbeiten zu können, ist es wichtig, die Version des verwendeten REP zu kennen. Durch den Aufruf von **repprep** oder **repgo** mit dem Schalter **-V** kann die Version des jeweiligen Programms ermittelt werden.

2 Struktur eines REPORTS

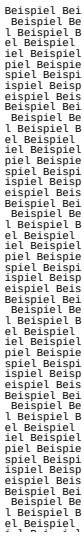
Ein Report setzt sich aus mehreren Abschnitten zusammen, wobei 3 Abschnitte obligatorisch und 5 optional sind.Die einzelnen Abschnitte sind in der Reihenfolge zu deklarieren, wie sie nachfolgend gegliedert sind.

DATABASE-Abschnitt	obligatorisch	Datenbankdeklaration
DEFINE-Abschnitt	optional	Variablendeklaration
INIT-Abschnitt	optional	Konstantendeklaration
RESOURCES-Abschnitt	optional	Resourcendeklaration
INPUT-Abschnitt	optional	Interaktive Datenanforderung
OUTPUT-Abschnitt	optional	Layout-Festlegungen (Standardausgabe)
SELECT-Abschnitt	obligatorisch	Datenselektion (statisch)
FORMAT-Abschnitt	obligatorisch	Datenselektion (dynamisch), Allgemeine Anweisungen, Layout-Festlegungen (alle Ausgabekanäle) und Datenmanipulation

Ein Report muß mindestens einen, kann jedoch mehrere SELECT-FORMAT-Abschnitte enthalten. So ist es Ihnen möglich, mehrmals statisch Daten zu selektieren und im FORMAT-Abschnitt aufzubereiten, wobei alle in den darüberliegenden Abschnitten getroffenen Deklarationen und Zuweisungen gültig bleiben.

Achtung:

Selektierte Felder können nicht in einem nachfolgenden SELECT-FORMAT-Abschnitt verwendet werden.



```
        { Erster SELECT-FORMAT-Abschnitt }
SELECT *
        FROM kunde
END
FORMAT
        EVERY ROW
END
        { Zweiter SELECT-FORMAT-Abschnitt }
SELECT *
        FROM auftrag
END
FORMAT
        EVERY ROW
END
```

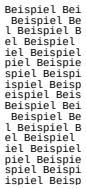
Die einzelnen Abschnitte eines Reports werden in den folgenden Kapiteln des Handbuches detailliert erläutert.

5 DATABASE-Abschnitt

In diesem Teil des Reports deklarieren Sie die Datenbank, mit deren Daten gearbeitet werden soll. Wird beim Aufruf von **repgo** der Schalter **-d** zusammen mit einem Datenbanknamen angegeben, dann wird der DATABASE-Abschnitt ignoriert.

```
DATABASE datenbank
END
```

datenbank die im Report verwendete Datenbank



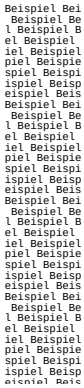
Für den Report soll Datenbank *m_db* verwendet werden.

```
DATABASE m_db
END
```

6 DEFINE-Abschnitt

Im DEFINE-Abschnitt werden alle Variablen deklariert, die in dem Report verwendet werden. Eine Variablendeklaration besteht aus einem Variablen-Namen, einem entsprechenden Datentyp und der Zuordnung zu einer Variablengruppe.

DEFINE
[**VARIABLE** *variable datentyp* [. . .]]
[**PARAM [***position***]** *variable datentyp* [. . .]]
[**CURSOR** *variable* [. . .]]
END



Um eine Liste zu erstellen, werden die Variablen *name* (30 Zeichen), *vorname* (30 Zeichen) und *konto_nr* (Zahl) benötigt. Zusätzlich wird die Auftragsnummer *auftr* (Zahl) beim Aufruf des Reports als 1. Parameter mitgegeben.

```
DEFINE
    VARIABLE name CHAR(30)
    VARIABLE vorname CHAR(30)
    VARIABLE konto_nr  INTEGER
    PARAM[1] auftr  INTEGER
END
```

Achtung:

Die Variablennamen in einem Report müssen in den ersten 18 Stellen eindeutig sein. Selektierte Spalten dürfen nicht den gleichen Namen wie eine deklarierte Variable haben.

1 Variablen-Gruppen

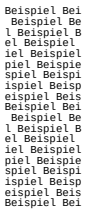
Mit einem der nachfolgenden Schlüsselwörter VARIABLE, PARAM oder CURSOR ordnen Sie eine Variable einer Gruppe zu, welche die Herkunft bzw. Verwendung der Variablen festlegt.

1.1 VARIABLE

Der Gruppe VARIABLE werden alle Variablen zugeordnet, die weder als Parameter übergeben, noch als Cursor verwendet werden.

VARIABLE *variable datentyp*

variable	Name der Variablen
datentyp	Typ der Variablen (siehe nachfolgende Seiten)



In dem Report wird ein Zähler, eine Variable zur Aufnahme eines Datums und eine Variable zur Aufnahme eines Geldbetrages benötigt.

VARIABLE *zähler INTEGER*
VARIABLE *buchungs_datum DATE*
VARIABLE *geld_betrag MONEY*

1.2 PARAM

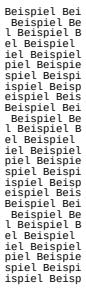
Mit PARAM spezifizieren Sie eine Variable, die aus der Kommandozeile beim Aufruf des Reports[1] ihren Wert erhält.

PARAM [*position***]** *variable datentyp*

position	Integer-Wert, der die Position eines Parameters bei dem Aufruf in der Kommandozeile anzeigt.

Achtung:

PARAM [0] ist mit dem Namen des Reports belegt. Die Größen der Variablen müssen in Größe und Datentyp (s.u.) so gewählt werden, daß die Übergabeparameter aufgenommen werden können.



Als Parameter sollen 2 Zahlen, 2 Zeichenketten und 1 Datum übergeben werden.

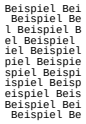
PARAM [0]	*report_name*	CHAR(15)
PARAM [1]	*p_auftr_anf*	INTEGER
PARAM [2]	*p_auftr_end*	INTEGER
PARAM [3]	*p_auftr_name1*	CHAR(30)
PARAM [4]	*p_auftr_name2*	CHAR(30)
PARAM [5]	*p_auftr_dat*	DATE

1.3 CURSOR

Wollen Sie im FORMAT-Abschnitt[2] ein SELECT mittels EXEC-SQL-Anweisungen[3] tätigen, so benötigen Sie eine Variable, die der Variablen-Gruppe CURSOR angehört.

CURSOR *variable*

1. siehe *Starten mit repgo* auf Seite 13
2. siehe *FORMAT-Abschnitt* auf Seite 49
3. *EXEC SQL DECLARE cursor CURSOR FOR “SELECT-Anweisung”* auf Seite 54



Mit der Variablen *mein_cursor* sollen im Formatteil Daten selektiert werden.

CURSOR *mein_cursor*

2 Datentypen

Bei der Deklaration von Variablen (VARIABLE), bzw.Parametern (PARAM), können Sie folgende Datentypen verwenden:

2.1 CHAR (größe)

dient zur Aufnahme von Zeichen. Mit *größe* können Sie durch Angabe eines Integer-Wertes die Anzahl der Zeichen festlegen.

2.2 DECIMAL ([n],[m])

Typ zur Aufnahme von Dezimal-Werten. Mit *(n,m)* können Sie Angaben zur Länge (n) und den Nachkommastellen (m) machen.

2.3 FILE

dient zur Aufnahme eines Dateizeigers, den Sie benötigen, um in Ihrem Report mit Dateien arbeiten zu können.

2.4 FLOAT

dient zur Aufnahme von Fließkomma-Zahlen, die nicht größer als 8 Byte sind.

2.5 INTEGER

kennzeichnet ganze Zahlen, die nicht größer als 4 Bytes sind.

2.6 MONEY ([n],[m])

Feldtyp zur Aufnahme eines Geldbetrages. Mit *n,m* können Sie Angaben zur Anzahl der Dezimalstellen (n; Default=16) und den Nachkommastellen (m; Default=2) machen.

2.7 SMALLINT

kennzeichnet ganze Zahlen, die nicht größer als 2 Byte sind.

2.8 SMALLFLOAT

dient zur Aufnahme von Fließkomma-Zahlen, die nicht größer als 4 Byte sind.

2.9 DATE

dient zur Aufnahme eines Datums. Die Ausgabe erfolgt immer formatiert als Character-String.

2.10 DATETIME k1 TO k2

dient zur Aufnahme von Zeitpunkten. *k1* und *k2* beschreiben die Genauigkeit des Datentyps. Folgende Schlüsselwörter sind dabei zulässig:

YEAR, MONTH, DAY, HOUR, MINUTE, SECOND, FRACTION

k2 muß dabei immer aus einem weiter hinten aufgeführten Schlüsselwort bestehen als *k1*.

2.11 INTERVAL k1 [(n)] TO k2

dient zur Aufnahme von Zeitspannen. *k1*, *n* und *k2* beschreiben die Genauigkeit des Datentyps. Dabei gelten die gleichen Regeln und Schlüsselwörter wie beim Typ DATETIME. Zusätzlich gilt folgende Regel:

Ist einer der Werte *k1* oder *k2* YEAR oder MONTH, so muß auch der andere Wert YEAR oder MONTH sein.

Das bedeutet, daß Intervalwerte in zwei Gruppen geteilt werden, welche an der Monat/Tag[1] Grenze getrennt werden. Weiterhin kann im Gegensatz zum Typ DATETIME mit *n* eine Stellenanzahl der ersten Komponente angegeben werden.

2.12 TEXT

dient zur Aufnahme von TEXT Spalten aus der Datenbank. Dieser Datentyp ist z.Z nicht vollständig implementiert.Variablen dieses Typs dürfen lediglich in EXEC SQL Anweisungen als Zielvariablen verwendet werden. Zuweisungen an diese Variablen funktionen nicht. Beim Zugriff auf die Variable, wird ihr Inhalt in den Datentyp CHAR umgewandelt.

1. Grund dafür ist, daß ein Monat nicht genau in Tagen ausgedrückt werden kann

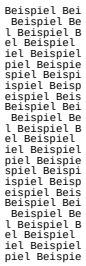
7 INIT-Abschnitt

In diesem Abschnitt haben Sie die Möglichkeit, Variablen einen Startwert zuzuordnen.

```
INIT
   [ [SET variable = konstante]  [. . .] ]
END
```

Achtung:

Der Datentyp der Variablen muß mit dem der Konstanten übereinstimmen. Eine Konstante, welche einer Variablen vom Typ CHAR zugeordnet wird, muß in Hochkommata “ “ geschrieben werden. Weiterhin dürfen im INIT-Abschnitt keine Funktionen verwendet werden.



Der Variablen *meine_datei (char(20))* soll der Dateiname *daten.txt* zugeordnet und die Variable *zähler* mit 0 initialisiert werden.

```
INIT
        SET meine_datei = “daten.txt”
        SET zähler = 0
END
```

8 Resource-Abschnitt

In diesem Abschnitt haben Sie die Möglichkeit, in Resourcedateien[1] gesetzte Resourcen für den aktuellen Report umzusetzen oder eigene zu definieren.

```
RESOURCES
   resourcename = string|zahl
   [. . .]
END
```

Beispiel Bei Beispiel Be l Beispiel B el Beispiel iel Beispiel piel Beispie spiel Beispi ispiel Beisp eispiel Beis Beispiel Bei Beispiel Be l Beispiel B el Beispiel iel Beispiel piel Beispie spiel Beispi ispiel Beisp eispiel Beis Beispiel Bei Beispiel Be l Beispiel B el Beispiel iel Beispiel piel Beispie

Für einen Report sollen die Attributte aus der Datei `neu.att` gelesen werden. Die Resourcedatei definiert die Resource **REPATT** jedoch mit `std.att`. Innerhalb des Resource-Abschnitts kann die Resource für diesen Report neu definiert werden, so daß die gewünschte Datei verwendet wird:

```
RESOURCES
        REPATT = “neu.att”
END
```

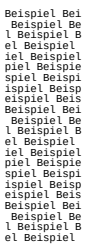
1. Siehe Abschnitt über Resourcedateien

9 INPUT-Abschnitt

Wollen Sie in Ihrem Report interaktiv Daten anfordern, so können Sie dies in diesem Abschnitt bewerkstelligen.

```
INPUT
   [PROMPT FOR variable
   USING "string" [. . .] ]
END
```

Die Ausgabe des Promptes erfolgt dabei auf die Standardausgabe und der von der Standardeingabe gelesene Wert wird in der Variablen abgelegt.



Es sollen die Daten eines Kunden bearbeitet werden, dessen Name interaktiv erfragt wird.

```
INPUT
      PROMPT FOR kunde USING "Name des Kunden ?"
END
```

10 OUTPUT-Abschnitt

In diesem Abschnitt haben Sie die Möglichkeit, das Layout Ihrer Report-Ausgabe zu bestimmen.
Mit diesen Angaben legen Sie das Seitenlayout der Standardausgabe fest.
Eine 2.Möglichkeit, das Layout festzulegen, ist das SET-PAGE[1] Kommando, welches das Beschreiben aller Ausgabekanäle ermöglicht.

```
OUTPUT
   [REPORT TO { {"datei"}
                        {PRINTER}
                        {PIPE "programm" } } ]
                        {VISUALREP "Verbindung"}
                        {VISUALSIMPLE "Verbindung" Cpi}
   [LEFT MARGIN position]
   [RIGHT MARGIN position]
   [TOP MARGIN position]
   [BOTTOM MARGIN position]
   [PAGE LENGTH seitengröße]
   [FIRST PAGE HEADER LENGTH höhe]
   [PAGE HEADER LENGTH höhe]
   [PAGE TRAILER LENGTH höhe]
END
```

1 Allgemeine Angaben

REPORT TO "*datei*"

Die Report-Ausgabe erfolgt in die angegebene Datei "datei"

REPORT TO PRINTER

1. siehe *FORMAT-Anweisungen* auf Seite 53: SET PAGE

Die Ausgabe erfolgt direkt über “lpr” an den Drucker.

REPORT TO PIPE “*programm*”

Die Report-Ausgabe wird über eine Pipe in die Standardeingabe des angegebenen Programmes geschrieben.

REPORT TO VISUALREP “*Verbindung*”

Die Report-Ausgabe erfolgt über einen VisualREP Server. “Verbindung” definiert die Verbindung zu dem Server. Weitere Informationen hierzu sind im VisualREP Handbuch zu finden.

REPORT TO VISUALSIMPLE “*Verbindung*” *Cpi*

Die Report-Ausgabe erfolgt über einen VisualREP Server im Simple Modus. “Verbindung” definiert die Verbindung zum Server. Cpi definiert die Anzahl Zeichen pro Zoll. Weitere Informationen sind im VisualREP Handbuch zu finden.

2 Randgrößen

Die nachfolgenden Konstrukte bestimmen das Seitenlayout des Reports, wobei mit *position* der jeweilige Randabstand festgelegt wird.

LEFT MARGIN *position*

Mit LEFT MARGIN bestimmen Sie die Größe des linken Randes einer Seite. Der Standard-Wert beträgt 5 Spalten.

RIGHT MARGIN *position*

Mit RIGHT MARGIN bestimmen Sie, bis zu welcher Spalte gedruckt werden kann. Der Standard-Wert beträgt 132 Spalten.

Achtung:

Die tatsächliche Länge einer Zeile ergibt sich aus:

```
RIGHT MARGIN - LEFT MARGIN
```

TOP MARGIN *position*

Mit TOP MARGIN bestimmen Sie die Größe des oberen Randes einer Seite. Der Standard-Wert beträgt 3 Zeilen.

BOTTOM MARGIN *position*

Mit BOTTOM MARGIN bestimmen Sie die Größe des unteren Randes einer Seite. Der Standard-Wert beträgt 3 Zeilen.

PAGE LENGTH *seitengröße*

Mit PAGE LENGTH bestimmen Sie die gesamte Größe einer Seite. Der Standard-Wert beträgt 66 Zeilen.

Achtung:

Die tatsächliche Größe einer Seite (bedruckbarer Seitenabschnitt) ergibt sich aus:

```
PAGE LENGTH - (TOP MARGIN + BOTTOM MARGIN).
```

FIRST PAGE HEADER LENGTH *höhe*

Mit *höhe* legen Sie fest, aus wievielen Zeilen der FIRST PAGE HEADER auf der ersten Seite bestehen soll.

Achtung:

- Differieren FIRST PAGE HEADER LENGTH und PAGE HEADER LENGTH, so verändert sich die Höhe des Datenbereiches.
- Die Position des PAGE TRAILERS bleibt unverändert.
- Hat REP eine größere Höhe (Summe der ausgabeorientierten Anweisungen) als angegeben errechnet, so wird diese verwendet.

PAGE HEADER LENGTH *höhe*

Mit *höhe* legen Sie fest, aus wievielen Zeilen der PAGE HEADER bestehen soll.

Achtung:

- Differieren FIRST PAGE HEADER LENGTH und PAGE HEADER LENGTH, so verändert sich die Höhe des Datenbereiches.
- Die Position des PAGE TRAILERS bleibt unverändert. Hat REP eine größere Höhe (Summe der ausgabeorientierten Anweisungen) als angegeben errechnet, so wird diese verwendet.

PAGE TRAILER LENGTH *höhe*

Mit *höhe* legen Sie fest, aus wievielen Zeilen der PAGE TRAILER auf der ersten Seite bestehen soll.

Achtung:

- Folgende Bedingung gilt:

  ```
  [FIRST] PAGE HEADER + PAGE TRAILER + Datenblock <=
  PAGE LENGTH - (TOP MARGIN + BOTTOM MARGIN)
  ```

- Der Start des TRAILER-Druckes errechnet sich wie folgt:

  ```
  Start des PAGE TRAILERS =
  PAGE LENGTH - (TOP MARGIN + BOTTOM MARGIN) - PAGE TRAILER LENGTH + 1
  ```

- Hat REP eine größere Höhe (Summe der ausgabeorientierten Anweisungen) als die angegebene errechnet, so wird diese verwendet.

11 SELECT-Abschnitt

Der SELECT-Abschnitt ist obligatorischer Bestandteil eines Reports. Er setzt sich aus einer oder mehreren SELECT-Anweisungen zusammmen. Aufgrund der EXEC-SQL-Anweisung haben Sie bei REP auch die Möglichkeit, ein SELECT im FORMAT-Abschnitt einzubauen.

1 SELECT-Anweisung

[SELECT-Anweisung [;SELECT-Anweisung] . . .] END

Die SELECT-Anweisung ist die von der X/OPEN-Gruppe in der Datenbankabfragesprache SQL (Structured Query Language) definierte Anweisung, um Daten aus einer Datenbank zu lesen. Mit der SELECT-Anweisung werden die im Report auszugebenden Daten aus der Datenbank abgerufen. Die ausgewählten Daten können auch über mehrere SELECT-Anweisungen, die nacheinander durchlaufen werden, erhalten werden. Alle SELECT-Anweisungen, außer der letzten, enden mit einem Semikolon. Einige Einschränkungen sollten Sie bei dem Aufbau der SELECT-Anweisungen berücksichtigen:

- Alle SELECT-Anweisungen, außer der letzten, müssen als letzte Anweisung INTO TEMP *tabelle* enthalten.
- Eine ORDER BY-Klausel sollte nicht zusammen mit INTO TEMP verwendet werden.
- Für den Report stehen nur die vom letzten SELECT ausgewählten Daten zur Verfügung. Sie sollten beachten, daß das SELECT keinen Feldnamen enthält, der mehrmals in der verwendeten Datenbank existiert. Bei Doppelbenennungen muß das Feld umbenannt oder entsprechend qualifiziert werden.
- Verwenden Sie ORDER BY, so darf in dieser Klausel kein zusammengesetzter Name, z.B (*tabelle.spalte*), verwendet werden.
- Verwenden Sie eine Variable im SELECT, so muß dieser ein ‘:’ oder ‘$’ vorangestellt werden.

1.1 SELECT-Anweisung

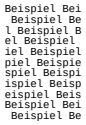
SELECT wählt Sätze aus der Datenbank aus, wobei mit FROM Sätze aus mehreren Tabellen verknüpft werden können. Das Resultat ist eine Tabelle, welche die in der Spaltenauswahl festgelegten Felder und die mit FROM und/oder WHERE ausgewählten Sätze enthält.

SELECT [{ALL | DISTINCT | UNIQUE}] *spaltenauswahl*
FROM *tabellenangabe, . . .*
[WHERE SELECT-Bedingung]
[GROUP BY { *spalte* | *spaltennummer* [, . . .] }]
[HAVING SELECT-Bedingung]
[ORDER BY { *spalte* | *spaltennummer* } [*ASC* | *DESC*] [, . . .]]
[INTO TEMP *tabelle* [**WITH NO LOG**]]
[UNION [ALL] SELECT-Anweisung]

Nachfolgend werden die Bestandteile der SELECT-Anweisung im Detail beschrieben.

SELECT [ALL] *spaltenauswahl*

Es werden alle ausgewählten Sätze geliefert, ohne die Duplikate auszusortieren. Dieser Wert ist voreingestellt.



Es sollen *name* und *vorname* von der Tabelle *person* selektiert werden.

SELECT *name, vorname*
FROM *person*

SELECT [**DISTINCT** | **UNIQUE**] *spaltenauswahl*

DISTINCT, bzw. UNIQUE, bewirken, daß bei dem SELECT Duplikate aussortiert werden.

Aus der Tabelle Person sollen alle Sätze selektiert werden, die einen eindeutigen Eintrag in der Spalte *name* und *vorname* haben.

SELECT DISTINCT *name, vorname*
FROM *person*

1.2 Spaltenauswahl

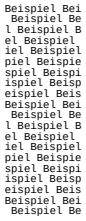
An dieser Stelle legen Sie fest, welche Spalten in der Ergebnistabelle des SELECT vorkommen.

spaltenauswahl::= {*****| *user***.** {*tabelle* |*****} | *tabelle***.** {*****| *spaltenname* } |
spaltenname | *ausdruck* [*alias-name*] [**,** . . .] }

*	Es werden alle Felder der Tabellen, die bei FROM aufgeführt sind, ausgelesen.
spaltenname	Name des zu selektierenden Feldes.
*tabelle.**	Es werden alle Felder der Tabelle tabelle ausgelesen.

ausdruck — An dieser Stelle kann ein Rechenausdruck stehen, der aus Feldern, Funktionen, Konstanten oder SELECT-Aggregaten aufgebaut sein kann. Er darf aber keine SELECT-Anweisung enthalten. Wird bei der Spaltenauswahl eine Mengenfunktion angewendet, gilt folgende Einschränkung: in der Spaltenauswahl dürfen nur Spaltennamen vorkommen, die in der GROUP BY-Klausel (s.u.) aufgeführt sind oder Argument einer Mengenfunktion sind.
Weiterhin ist zu bemerken, daß das Argument eines UNITS Ausdrucks im Gegensatz zur INFORMIX Schreibweise geklammert werden muß (also (anz) UNITS DAY statt anz UNITS DAY). Dies ist darin begründet, daß REP keine Operatatorpräferenzen für UNITS kennt.

alias-name — Name für die Ergebnisspalte, die mit *ausdruck* angegeben ist.



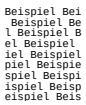
Aus der Tabelle Posten soll die Auftragsnummer selektiert werden. Weiterhin soll die Anzahl der Datensätze, sowie die Summe, welche aus dem Gesamtpreis resultiert, bestimmt werden.

SELECT *aufnr*, COUNT(*) *anzahl*, SUM*(gesamtpreis) summe*
FROM *posten*

1.3 Tabellenangabe

tabellenangabe ::= {*tabelle* [*referenz*] | **OUTER** *tabelle* [*referenz*] |
OUTER (*tabellenangabe*, . . .) }

Hier geben Sie die Tabellen an, aus welchen die Spalten selektiert werden. FROM kombiniert alle angegebenen Tabellen zu einer einzigen Tabelle, auf der das SELECT arbeitet. Mit r*eferenz* erhält eine Tabelle für die Dauer des SELECT's einen neuen Namen.



SELECT *A.name B.name*
FROM *kunde A, kunde B*

WHERE *SELECT-Bedingung*

In der WHERE-Klausel können Sie SELECT-Bedingungen festlegen, welche Maßstab für die zu selektierenden Daten sind.

Es sollen alle Datensätze selektiert werden, deren Artikelnummer kleiner 1001 ist.

SELECT *artikelnummer, artikel*
FROM *artikel*
WHERE *artikelnummer < 1001*

GROUP BY { *spalte* | *spaltennummer* [**,** . . .] }

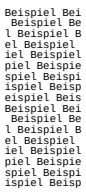
spalte — Alle Sätze, die in dem angegebenen Feld den gleichen Wert haben, werden zu einer Gruppe zusammengefaßt.

spaltennummer	Feldnummer in der Ergebnistabelle. Auf diese Weise kann auch ein Feld ohne Namen angesprochen werden.

Die von FROM... und WHERE... ausgewählten Sätze, die in den angegebenen Gruppenkennzeichen übereinstimmen, werden so überlagert, daß die Gruppe als ein Satz erscheint. Mit der HAVING -Klausel können eine oder mehrere SELECT-Bedingungen bzw. SELECT-Aggregate auf die Gruppen angewendet werden.

Achtung:

Die Spaltenauswahl darf keinen Feldnamen enthalten, der nicht zum angegebenen Gruppenkennzeichen gehört. In der Spaltenauswahl definierte Rechenausdrücke dürfen auch in das Gruppenkennzeichen aufgenommen werden.

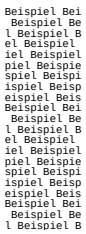


Für jeden Kundenauftrag soll der Gesamtpreis festgelegt werden

SELECT *kundennr*, SUM (*preis*)
FROM *posten*
GROUP BY *kundennr*

HAVING *SELECT-Bedingung*

wählt aus den Gruppen der Ergebnistabelle diejenigen aus, welche die SELECT-Bedingung erfüllen. Die Bedingung darf nur aus SELECT-Aggregaten, Konstanten oder untergeordneten SELECT-Anweisungen bestehen.



Es soll der Durchschnittspreis aller Auftragsgruppen berechnet werden, für die mehr als 2 Einträge existieren.

SELECT *auftrnr,* AVG*(ges_preis)*
FROM *auftr_tab*
GROUP BY *auftrnr*
HAVING COUNT(*) > 2

ORDER BY { *spalte* | *spaltennummer* } [**ASC** | **DESC**] [**,** . . .]

spalte	Feldname, nach welchem die Ergebnis-Tabelle sortiert wird.
spaltennummer	Feldnummer in der Ergebnistabelle. Auf diese Weise kann auch ein Feld ohne Namen angesprochen werden.
ASC	Die Sortierreihenfolge ist aufsteigend (Standard).
DESC	Die Sortierreihenfolge ist absteigend.

Die Sätze der Ergebnistabelle werden in der angegebenen Reihenfolge sortiert. Werden mehrere Felder (Spalten) angegeben, so wird zuerst nach dem ersten Feld sortiert; die danach angegebenen Felder werden nur benutzt, wenn in dem vorhergehenden Feld gleiche Werte aufgetreten sind.

Achtung:

- ORDER BY sollte nicht zusammen mit INTO TEMP benutzt werden (Erst ab INFORMIX 5.0 ist ein ORDER BY auch bei temporären Tabellen zulässig).
- In der ORDER BY-Klausel sind maximal 8 (INFORMIX 2.1), bzw. 16 Spalten (INFORMIX 4.x) zulässig.

Die Kundennamen aus der Tabelle *kunden* sollen sortiert werden.

```
SELECT kundenna FROM kunden
ORDER BY kundenna
INTO TEMP tabelle
```

INTO TEMP *tabelle* [**WITH NO LOG**]

Die Ergebnistabelle erhält den Namen *tabelle* und wird als temporäre Tabelle gespeichert. In nachfolgenden SELECT-Anweisungen kann die temporäre Tabelle wie eine normale Tabelle angesprochen werden.

Achtung:

INTO TEMP sollte nicht zusammen mit ORDER BY benutzt werden, da dies erst ab INFORMIX 5.0 zulässig ist.

UNION [**ALL**] *SELECT-Anweisung*

Mittels des UNION-Operators können zwei SELECT-Anweisungen miteinander kombiniert werden. Er bewirkt, daß alle Spalten zweier SELECT-Anweisungen selektiert werden. Die Angabe von ALL bewirkt, daß Duplikate in der Ergebnismenge nicht gelöscht werden. Wird ALL nicht angegeben, so werden die doppelten Sätze entfernt.

2 SELECT-Bedingung

SELECT-Bedingungen zeichnen sich dadurch aus, daß mittels einer oder mehrerer Bedingungen Ausdrücke verknüpft werden können, bzw. die Ergebnistabelle manipuliert werden kann.

2.1 Vergleichsoperatoren

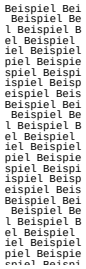
In einem SELECT können folgende Relationsoperatoren verwendet werden:

=	gleich
<	kleiner
>	größer
<=	kleiner gleich
>=	größer gleich
<> oder!=	ungleich
IS [NOT] NULL	ein [kein] NULL-Wert

2.2 Logische Operatoren

Folgende logische Operatoren (nähere Beschreibung im Kapitel FORMAT-Bedingung) werden in einer SELECT-Anweisung in der angegebenen Reihenfolge ausgewertet:

- NOT
- AND
- OR



Es soll die *posten_nr* von allen den Datensätzen selektiert werden, in welchen der Code kleiner als *10* und der Name nicht mit “A” beginnt.

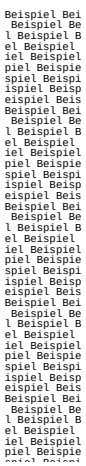
SELECT *posten_nr*
FROM *posten*
WHERE *code* < *10*
AND *name* **NOT** LIKE “A*”

2.3 Selektions-Operatoren

Mittels der Operatoren BETWEEN, LIKE und MATCHES können bestimmte Wertebereiche formuliert werden.

SELECT-Ausdruck [**NOT**] **BETWEEN** *SELECT-Ausdruck* **AND** *SELECT-Ausdruck*

Mit BETWEEN kann geprüft werden, ob der Wert eines SELECT-Ausdrucks in einem angegebenen Bereich liegt.



Es sollen alle Sätze der Tabelle *artikel* selektiert werden, bei welchen der Preis zwischen 100 und 200 liegt (Datum analog).

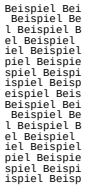
SELECT *artnr*
FROM *artikel*
WHERE *preis*
BETWEEN 100 **AND** 200

SELECT *artnr*
FROM *artikel*
WHERE *datum*
BETWEEN 6/11/91 **AND** 12/11/91

spaltenname [**NOT**] **LIKE** *konstante*

Vergleich mit einer Konstante, die ein Suchmuster enthält.

“_” steht für ein beliebiges Zeichen, “%” für eine beliebige Zeichenfolge an der Stelle im Suchmuster. Jedes andere Zeichen steht für sich selbst. Existiert ein solcher Feldinhalt oder eine solche Konstante, so ist die Bedingung erfüllt.



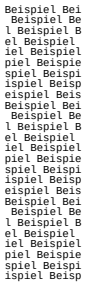
Es sollen alle Adressen von Personen selektiert werden, deren Name mit “Me” beginnt.

SELECT *adresse*
FROM *person*
WHERE *name* **LIKE** “Me%”

spaltenname [**NOT**] **MATCHES** *konstante*

MATCHES stellt eine Erweiterung von LIKE dar. Das Suchmuster kann folgende Platzhalter enthalten:

Platzhalter	Bedeutung
zeichen	genau das Zeichen
?	ein beliebiges Zeichen
*	0 oder beliebig viele Zeichen
[zeichen...]	eines der angegeben Zeichen
[zeichen-zeichen...]	ein Zeichen aus dem Bereich
[^zeichen...]	keines der aufgeführten Zeichen
[^zeichen-zeichen...]	kein Zeichen aus dem angegebenen Bereich
*	das Zeichen *
\?	das Zeichen ?



SELECT *adresse*
FROM *person*
WHERE *name* **MATCHES** *“Me*”*

SELECT *adresse*
FROM *person*
WHERE *name* **MATCHES** *“M[a-e]*”*

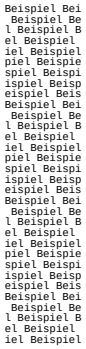
2.4 Subqueries

Bestandteil einer SELECT-Bedingung kann eine weiteres SELECT sein (Subquery), dessen Ergebnistabelle quantitativ und inhaltlich ausgewertet werden kann.

ALL

SELECT-Ausdruck [**NOT**] **ALL** *SELECT-Anweisung*

Ist der Vergleich für alle Werte, die die SELECT-Anweisung liefert erfüllt, oder liefert die SELECT-Anweisung keinen Wert, so ist die SELECT-Bedingung erfüllt.



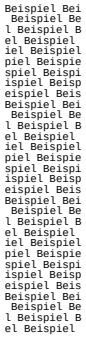
Es soll festgestellt werden, ob alle Aufträge, die mehr als 100 Posten beinhalten, von ein und demselben Kunden sind.

```
SELECT kunde.name
FROM kunde
WHERE kunde.nr = ALL(
                    SELECT ku_nr
                    FROM auftrag
                    WHERE posten > 100
                    )
```

ANY oder SOME

SELECT-Ausdruck [**NOT**] [**ANY**|**SOME**] *SELECT-Anweisung*

Ist der Vergleich für mindestens einen Wert, den die SELECT- Anweisung liefert erfüllt, so ist die SELECT-Bedingung erfüllt. ANY ist gleichbedeutend mit SOME.



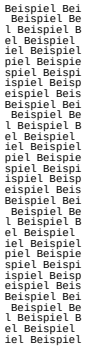
Es sollen alle Kunden aus der Tabelle *kunde* selektiert werden, für welche in der Tabelle *auftrag* ein Eintrag existiert.

```
SELECT kunde.name
FROM kunde
WHERE kunde.nr = ANY(
                    SELECT ku_nr
                    FROM auftrag
                    )
```

EXISTS

SELECT-Ausdruck [**NOT**] **EXISTS** *SELECT-Anweisung*

Liefert die SELECT-Anweisung als Ergebnis mindestens einen Satz, so ist die Bedingung erfüllt.



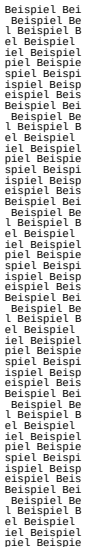
Es sollen alle Kunden aus der Tabelle *kunde* selektiert werden, für welche in der Tabelle *auftrag* ein Eintrag existiert.

```
SELECT kunde.name
FROM kunde
WHERE EXISTS
            (SELECT *
            FROM auftrag
            WHERE auftr.nr = kunde.nr)
```

IN

SELECT-Ausdruck [**NOT**] **IN** [*SELECT-Anweisung* | *SELECT-Ausdruck*]

Entspricht der SELECT-Ausdruck einem von der SELECT-Anweisung gelieferten Wert oder einer der angegebenen Konstanten, so ist die Bedingung erfüllt.



Es sollen alle Adressen in der Tabelle *person* gefunden werden, deren Name Meier oder Meister ist.

```
SELECT adresse
FROM person
WHERE name IN ("Meier","Meister")
```

Es sollen alle Kunden gefunden werden, für welche ein Auftrag erfaßt ist.

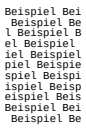
```
SELECT *
FROM kunde
WHERE k_name IN (
                    SELECT ku_name
                    FROM auftrag
                    )
```

3 SELECT-Ausdruck

Ein SELECT-Ausdruck kann eine Variable, ein Feldname, eine Konstante oder eine Funktion sein.

3.1 Variablen

Soll eine im DEFINE-Abschnitt definierte Variable in einer SELECT-Anweisung benutzt werden, so muß dem Variablennamen ein ':' vorangestellt werden. Die Variable muß mit einem Wert belegt sein (z.B. im INIT-Abschnitt oder als Parameter).



```
SELECT *
FROM person
WHERE name = :mein_name
```

3.2 Feldnamen

Ein angegebenes Feld muß zu einer bei FROM genannten Tabelle gehören. Kommt ein Feldname in mehr als einer Tabelle vor, so wird das Feld mit *tabelle.feld* angesprochen.

3.3 Konstanten

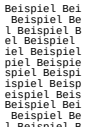
Eine Zeichenkette oder ein Datum müssen in Anführungszeichen eingeschlossen werden. Kommazahlen müssen den Dezimalpunkt enthalten, wenn Nachkommastellen existieren.

3.4 Funktionen

Die folgenden Funktionen[1] können in einem SELECT-Ausdruck verwendet werden. Hier nicht aufgeführte Funktionen oder Stored Procedures sollten vermieden werden, da der Report dann nicht mehr portabel ist. **repprep** erzeugt in diesem Fall eine Warnung. Die Lauffähigkeit des Reports durch Starten mit **repgo** ist dann von der jeweiligen Datenbankumgebung abhängig.

DAY (*SELECT-Ausdruck*)

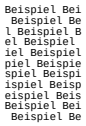
Ausgabe einer Zahl zwischen 1 und 31 für den Tag



SELECT **DAY***(auftragsdatum)*
FROM *auftrag*
WHERE *nr = 1001*

WEEKDAY (*SELECT-Ausdruck*)

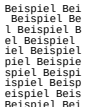
Ausgabe einer Zahl zwischen 0 für Sonntag und 6 für Samstag.



SELECT **WEEKDAY**(*auftragsdatum*)
FROM *auftrag*
WHERE *nr = 1001*

MONTH (*SELECT-Ausdruck*)

Ausgabe einer Zahl zwischen 1 und 12 für den Monat.

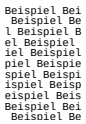


SELECT **MONTH** (*auftragsdatum*)
FROM *auftrag*
WHERE *nr = 1001*

YEAR (*SELECT-Ausdruck*)

Die Jahreszahl wird 4-stellig ausgegeben. Der Ausdruck muß eine Zahl ergeben und stellt die Zahl des Tages, gezählt ab dem 31.12.1899, dar.

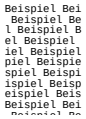
1. Der Umfang bezieht sich auf INFORMIX 4.0



```
SELECT YEAR (auftragsdatum)
FROM auftrag
WHERE nr = 1001
```

DATE (*SELECT-Ausdruck*)

Ausgabe des Datums.

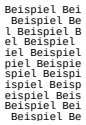


```
SELECT DATE (auftragsdatum)
FROM auftrag
WHERE nr = 1001
```

MDY (*SELECT-Ausdruck1* , *SELECT-Ausdruck2* , *SELECT-Ausdruck3*)

Dient zur Umwandlung in den Datentyp DATE. Hierbei stehen die Ausdrücke für folgende Inhalte:

SELECT-Ausdruck1 :	Monat als Zahl (1 .. 12)
SELECT-Ausdruck2 :	Tag als Zahl (1 .. 31)
SELECT-Ausdruck3 :	Jahreszahl (vierstellig)



```
SELECT artnr
FROM artikel
WHERE datum = MDY(11, 12, 1991)
```

LENGTH (*SELECT-Ausdruck*)

Gibt die Anzahl der Zeichen in SELECT-Ausdruck zurück. SELECT-Ausdruck kann eine in Hochkomma geschriebene Zeichenkette oder eine Variable vom Typ CHAR sein.

TODAY

Diese Funktion liefert das aktuelle Datum im Datentyp DATE.

CURRENT [*komponente1* TO *komponente2*]

Diese Funktion liefert das aktuelle Datum und die aktuelle Uhrzeit im Datentyp DATETIME. Optional kann ein Bereich durch 2 Komponenten vorgegeben werden. Die Werte für *komponente1* und *komponente2* sind im Abschnitt *DATETIME k1 TO k2* auf Seite 24 beschrieben.

4 SELECT-Aggregate

Folgende Mengenfunktionen stehen bei einem SELECT zur Verfügung :

AVG ([ALL|DISTINCT] *spalte*)

berechnet für jede Gruppe den Durchschnittswert. DISTINCT berechnet die Duplikate nicht mit.

SELECT **AVG***(gesamtpreis) durchschnitt*
FROM *posten*

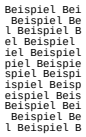
COUNT (*)

Ermittelt die Anzahl von Sätzen pro Gruppe.

SELECT *gesamtpreis*, **COUNT**(*) *anzahl*
FROM *posten*

COUNT (DISTINCT *feld*)

Ermittelt die Anzahl von Sätzen, die in der Gruppe im angegebenen Feld einen unterschiedlichen Wert haben. Duplikate und NULL-Werte werden nicht mitgezählt.



SELECT *aufnr*, **COUNT**(DISTINCT *artnr*) *anzahl_artikel*
FROM *posten*
GROUP BY *aufnr*

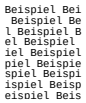
MAX ([ALL|DISTINCT] *spalte*)

liefert den größten Wert der Gruppe. Bei DISTINCT werden die Duplikate nicht mitgerechnet.

SELECT **MAX***(gesamtpreis) preis_max*
FROM *posten*

MIN ([ALL|DISTINCT] *spalte*)

liefert den kleinsten Wert der Gruppe. Bei DISTINCT werden die Duplikate nicht mitgerechnet.



SELECT **MIN***(gesamtpreis) preis_min*
FROM *posten*

SUM ([ALL|DISTINCT] *spalte*)

Bildet die Gruppensumme. Bei DISTINCT werden die Duplikate nicht mitgerechnet.

SELECT DISTINCT **SUM***(gesamtpreis) preis_summe*
FROM *posten*

5 NOCHECK Anweisungen

Mit Hilfe der Schlüsselworte **{NOCHECK}** und **{ENDNOCHECK}** kann **repprep** veranlasst werden eine Select Anweisung nicht zu parsen, sondern lediglich in die .rpp Datei zu kopieren. Damit ist es möglich, neue Features von SQL zu nutzen, auf die REP nicht vorbereitet ist. Auf der anderen Seite erfolgt keine Prüfung dieser Anweisungen durch **repprep** und ein Fehler wird erst zur Laufzeit von **repgo** erkannt.

Um das NOCHECK Feature zu verwenden, ist in der Zeile vor der Select Anweisung der Kommentar '**{NOCHECK}**' zu schreiben. Nach der Select Anweisung (also noch vor dem **END** des Select Abschnittes) ist der Block mit '**{ENDNOCHECK}**' abzuschließen. Bei mehreren Select Anweisungen ist jede (nicht zu prüfende) Anweisung in einen eigenen '**{NOCHECK} ... {ENDNOCHECK}**' Block einzufassen.

Beim Kopieren des Anweisung in die .rpp Datei werden Leerzeichen und Zeilenumbrüche entfernt. Kommentare werden jedoch mit übernommen. Dieses Verhalten kann nützlich sein, um Optimizer Hints zu definieren, welche beim Informix in Form eines Kommentars angegeben werden.

12 FORMAT-Abschnitt

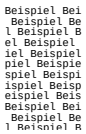
In diesem Teil des Reports können die durch ein oder mehrere SELECT-Anweisungen bestimmten Daten verarbeitet werden.

```
FORMAT
   EVERY ROW
   | [ortsangabe FORMAT-Anweisung [FORMAT-Anweisung . . .] . . .]
END
```

EVERY ROW — REP erzeugt einen Standard-Report. Ist ein Datensatz der Ergebnistabelle der SELECT-Anweisung nicht länger als die eingestellte Zeilenlänge, so wird ein Satz pro Zeile ausgegeben. Andernfalls wird jedes Feld in eine eigene Zeile geschrieben. EVERY ROW schließt alle anderen Ortsangaben aus.

ortsangabe — Der Report-Ersteller legt fest, an welcher Stelle und bei welchem Verarbeitungsstand REP eine oder mehrere FORMAT-Anweisungen ausführen soll.

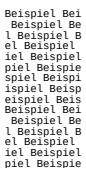
FORMAT-Anweisung — Sie beschreibt die Aktionen, die REP ausführen soll.



```
FORMAT
    EVERY ROW
END
```

1 Ortsangaben

Der Report-Ersteller legt fest, an welcher Stelle und bei welchem Verarbeitungsstand REP eine oder mehrere FORMAT-Anweisungen ausführen soll. Jede dieser Ortsangaben muß mindestens eine FORMAT-Anweisung enthalten. Wird EVERY ROW nicht benutzt, so muß der Report mindestens eine Ortsangabe enthalten.



```
. . .
FORMAT
    ON LAST ROW
        PRINT “Summe der Sätze”, COUNT
```

1.1 Gruppenwechsel

Ein Gruppenwechsel liegt vor, wenn sich der Inhalt des genannten Feldes oder der Inhalt eines übergeordneten Feldes ändert. Übergeordnete Felder sind solche, die in der ORDER BY-Klausel[1] an vorhergehender Stelle genannt sind.
Die Abarbeitung mehrerer BEFORE GROUP OF Blöcke, bzw. AFTER GROUP OF Blöcke, erfolgt in der Reihenfolge, wie es

das nachfolgende Schema zeigt. Es geht davon aus, daß im SELECT-Abschnitt die Spalten nach *feld1*, *feld2* und *feld3* sortiert werden.

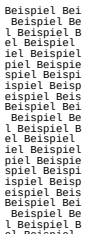
```
BEFORE GROUP OF feld1
    FORMAT-Anweisung . . .
        BEFORE GROUP OF feld2
            FORMAT-Anweisung . . .
                BEFORE GROUP OF feld3
                    FORMAT-Anweisung . . .
                            ON EVERY ROW
                                    FORMAT-Anweisung . . .
                AFTER GROUP OF feld3
                    FORMAT-Anweisung . . .
        AFTER GROUP OF feld2
            FORMAT-Anweisung . . .
AFTER GROUP OF feld1
    FORMAT-Anweisung . . .
```

1.2 Ortsangaben im einzelnen

Folgende Ortsangaben können in einem Report verwendet werden:

AFTER GROUP OF *feld*

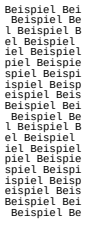
Nach einem Gruppenwechsel und am Reportende werden die FORMAT-Anweisungen ausgeführt. *feld* muß in der ORDER BY-Klausel des letzten SELECT genannt sein.



```
. . .
FORMAT
        . . .
        AFTER GROUP OF auftr_nr
                PRINT "Anzahl der Positionen: ",GROUP COUNT USING "####"
                . . .
```

BEFORE GROUP OF *feld*

Vor einem Gruppenwechsel und am Reportanfang werden die FORMAT-Anweisungen ausgeführt. *feld* muß in der ORDER BY-Klausel des letzten SELECT genannt sein.



```
. . .
FORMAT
        . . .
        BEFORE GROUP OF auftr_nr
                PRINT "Positionen für Auftrag: ",auftr_nr USING "####"
                . . .
```

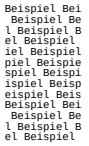
1. siehe *ORDER BY { spalte | spaltennummer } [ASC | DESC] [, . . .]* auf Seite 38

FIRST PAGE HEADER

Die FORMAT-Anweisungen werden am Anfang der ersten Seite ausgeführt. An dieser Stelle kann nicht die SKIP TO TOP OF PAGE-Anweisung benutzt werden. Verwenden Sie IF ... THEN ... ELSE, so muß die Zeilenanzahl bei einer PRINT- oder SKIP-Anweisung im THEN-Zweig mit der im ELSE-Zweig identisch sein.

Achtung:

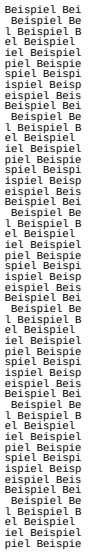
Verwenden Sie eine der oben beschriebenen Anweisungen, so ist es ratsam, die Höhen der einzelnen Ortsangaben anzugeben, da die von REP bestimmten Höhen in diesem Falle statistischen Charakter haben.



```
. . .
FIRST PAGE HEADER
        PRINT “Erste Seite”
. . .
```

Achtung:

Beinhalten FIRST PAGE HEADER und PAGE HEADER die gleiche Anzahl an Newlines (PRINT ohne ";" oder SKIP *anzahl* LINES), wird ein SKIP TO TOP OF PAGE auf der ersten Seite nur dann ausgeführt, wenn vor Erreichen des SKIP TO TOP OF PAGE-Befehles im Datenbereich (ON EVERY ROW ...) eine Ausgabe erfolgt ist.



```
FORMAT
        FIRST PAGE HEADER
                PRINT "++++ Beginn der Auswertung ++++"
        PAGE HEADER
                PRINT "++++ SEITE ", PAGENO { Anzahl der Newlines gleich (1) }
        ON EVERY ROW
                IF ku_nr > 100 THEN
                BEGIN
                        PRINT "*** Statistik für ",ku_name
                        . . .
                END
        SKIP TO TOP OF PAGE   { Ist auf der ersten Seite ku_nr größer als
                                100, so wird ein SKIP TO TOP PAGE
                                ausgeführt. }
END
```

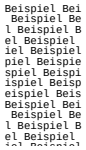
PAGE HEADER

Die FORMAT-Anweisungen werden am Anfang jeder Seite, außer der ersten, wenn FIRST PAGE HEADER angegeben ist, ausgeführt. Ist ein PAGE TRAILER-Code vorhanden, so geht grundsätzlich die Ausführung desselben der Ausführung des PAGE HEADERS voraus.

An dieser Stelle kann nicht die SKIP TO TOP OF PAGE-Anweisung benutzt werden.
Verwenden Sie IF ... THEN ... ELSE, so sollte die Zeilenanzahl bei einer PRINT- oder SKIP-Anweisung im THEN-Zweig mit der in dem ELSE-Zweig identisch sein. Weiterhin sind Schleifen (WHILE oder FOR), sowie ein PRINT FILE in dieser Ortsangabe nicht erlaubt.

Achtung:

Arbeiten Sie nicht im PAGEMODE, in welchem Sie die Höhen der einzelnen Ortsangaben angeben, sollten grundsätzlich die oben aufgeführten Anweisungen vermieden werden, da die von REP bestimmten Höhen in diesem Falle statistischen Charakter haben.



```
. . .
PAGE HEADER
        PRINT “Abrechnung des Kunden “,kund_na
        . . .
```

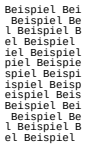
PAGE TRAILER

Die FORMAT-Anweisungen werden am Ende jeder Seite ausgeführt.
An dieser Stelle kann nicht die SKIP TO TOP OF PAGE-Anweisung benutzt werden.

Benutzen Sie IF ... THEN ... ELSE, so sollte die Zeilenanzahl bei einer PRINT- oder SKIP-Anweisung im THEN-Zweig mit der in dem ELSE-Zweig identisch sein. Weiterhin sind Schleifen (WHILE oder FOR), sowie ein PRINT FILE in dieser Ortsangabe nicht erlaubt.

Achtung:

Arbeiten Sie nicht im PAGEMODE, in welchem Sie die Höhen der einzelnen Ortsangaben angeben, sollten grundsätzlich die oben aufgeführten Anweisungen vermieden werden, da die von REP bestimmten Höhen in diesem Falle statistischen Charakter haben.



```
. . .
PAGE TRAILER
        PRINT COLUMN 38, “Seite -”, PAGENO USING “###”’ “-”
        . . .
```

INIT

Die Format-Anweisungen in dieser Ortsangabe werden am Anfang eines jeden FORMAT-Teiles abgearbeitet. Sie dient dazu, alle Format-Anweisungen, die initialen Charakter haben, zusammen zu fassen.

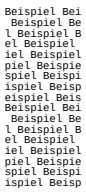
Alle FORMAT-Anweisungen, die eine Ausgabe produzieren oder eine nachfolgende Ausgabe implizieren, sind in der INIT-Ortsangabe nicht zulässig:

- PRINT
- SKIP
- PRINTFORM
- MOVEYX
- SAVEYX

Die SET PAGE-Anweisung, welche das Layout der Standardausgabe festlegt, darf nicht außerhalb der INIT-Ortsangabe verwendet werden.

ON EVERY ROW/RECORD

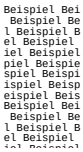
Die FORMAT-Anweisungen werden für jeden Datensatz durchlaufen.



```
. . .
ON EVERY ROW
        PRINT “Auftragsnr.      :”,auftr_nr
        PRINT “Kundenname       :”,kund_na
        . . .
```

ON LAST ROW/RECORD

Die FORMAT-Anweisungen werden durchlaufen, nachdem der letzte Datensatz ausgelesen wurde und alle Anweisungen von ON EVERY ROW Blöcken und AFTER GROUP OF Blöcken durchlaufen wurden. Bestimmte Anweisungen sind ausschließlich in dieser Ortsangabe zulässig.



```
. . .
ON LAST ROW
        PRINT “Anzahl der Aufträge :”,COUNT
        . . .
```

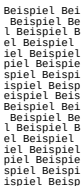
2 FORMAT-Anweisungen

Mit den FORMAT-Anweisungen legen Sie fest, welche Aktionen im Report ausgeführt werden sollen. Eine FORMAT-Anweisung kann für sich alleine stehen oder in einem Block zusammengefaßt werden.

2.1 Blockanweisung

BEGIN FORMAT-Anweisung FORMAT-Anweisung ... END

Mehrere FORMAT-Anweisungen werden zu einer syntaktischen Einheit zusammengefaßt. Diese Zusammenfassung wird bei WHILE, IF und FOR gebraucht, wenn mehr als eine FORMAT-Anweisung auszuführen ist.



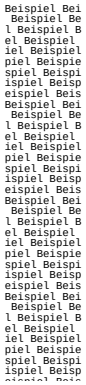
```
IF anzahl > 0 THEN
BEGIN
        PRINT “Posten :”, artikel
        PRINT “Bestand:”, anzahl
END
```

2.2 Format-Anweisungen im einzelnen

Nachfolgend sind die Format-Anweisungen in alphabetischer Reihenfolge erläutert.

CALLYX

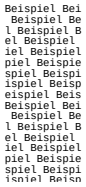
Diese Anweisung bewirkt, daß auf der virtuellen Seite auf die mit SAVEYX gespeicherte Position positioniert wird. CALLYX setzt voraus, daß mit SAVEYX vorher eine Position gespeichert wurde.



```
...
PAGE HEADER
        PRINT COLUMN 10 "SUMME :";
        SAVEXY
...
PAGE TRAILER
        CALLYX
        PRINT summe
        SKIP TO TOP OF PAGE
...
```

CLOSE (*datei-variable*)

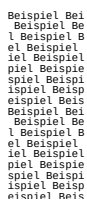
Die mit der *datei-variable* verbundene Betriebssystemdatei wird geschlossen. Wurde mit der SET PAGE-Anweisung ein Layout festgelegt, so wird dieses gelöscht.



```
.......
LET datei_zeiger = OPEN("meine_datei","r")
...
IF EOF(datei_zeiger) = 1 THEN
        CLOSE(datei_zeiger)
```

EXEC SQL DECLARE cursor **CURSOR FOR** "SELECT-Anweisung" [**INTO** *variablenliste*]

Mit dieser Anweisung kann ein Cursor deklariert und mit einer SELECT-Anweisung parametrisiert werden. Optional kann hinter dem Schlüsselwort INTO eine Variablenliste angegeben werden, welche die Variablen[1] bestimmt, in die das Ergebnis eines folgenden EXEC SQL FETCH Aufrufes abgelegt wird. Die Reihenfolge der angegebenen Variablen muß der Reihenfolge der Felder entsprechen, welche die SELECT-Anweisung liefert. Werden weniger Variablen angegeben, als Felder selektiert werden, dann werden nur die angegebenen Variablen mit Feldwerten gefüllt. Werden mehr Variablen angegeben, als Felder selektiert werden, dann behalten die zuviel angegebenen Variablen ihren ursprünglichen Wert. Alle Variablen müssen im DEFINE-Abschnitt deklariert sein und zur Gruppe VARIABLE gehören. Wird keine Variablenliste deklariert, dann werden wie im SELECT-Abschnitt die selektierten Daten als Felder angelegt, auf die mit dem Feldnamen zugegriffen werden kann. Diese Felder werden beim Freigeben des Cursors wieder entfernt. Alternativ zur SELECT-Anweisung kann auch eine EXECUTE PROCEDURE-Anweisung angegeben werden. Da innerhalb dieser Anweisung keine Feldwerte angegeben werden, ist in diesem Fall die Angabe einer Variablenliste zwingend. In beiden Fällen (SELECT und EXECUTE PROCEDURE) kann auf bestehende Variablenwerte lesend mittels :*variablenname* zugegriffen werden. *cursor* bezeichnet einen im DEFINE-Abschnitt vereinbarten Cursornamen.



```
EXEC SQL DECLARE mein_cursor CURSOR
        FOR "SELECT * FROM person WHERE ku_name = :name"
EXEC SQL DECLARE mein_cursor CURSOR
        FOR "SELECT name, vorname FROM adressen" into nname, vname
```

EXEC SQL OPEN *cursor*

1. EXEC SQL DECLARE unterstützt nicht den Oracle Spaltentyp CLOB

Öffnet den mit SQL DECLARE deklarierten Cursor *cursor*.

EXEC SQL OPEN *mein_cursor*

EXEC SQL FETCH *cursor*

Liest einen Satz des Cursors *cursor*. Dieser Cursor muß zuvor mit EXEC SQL DECLARE deklariert und mit EXEC SQL OPEN geöffnet worden sein.

WHILE SQLCODE = 0 DO
    EXEC SQL FETCH *mein_cursor*

EXEC SQL CLOSE *cursor*

Schließt den mit EXEC SQL DECLARE deklarierten und mit EXEC SQL OPEN geöffneten Cursor *cursor*.

EXEC SQL CLOSE *mein_cursor*

EXEC SQL FREE *cursor*

Gibt den mit EXEC SQL DECLARE deklarierten Cursor *cursor* frei. Alle durch das SELECT erzeugten Symbole werden freigegeben und können nochmals verwendet werden.

EXEC SQL FREE *mein_cursor*

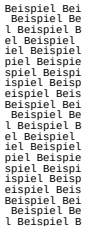
EXEC SQL “SQL-Anweisung”

Mit dieser Anweisung kann im Format-Teil ein SQL-Statement abgearbeitet werden. Variablen werden in der SQL-Anweisung durch Voranstellen eines ‘:’ markiert.

EXEC SQL “SELECT COUNT(*) INTO :*anzahl* FROM *pos*”

EXIT (FORMAT-Ausdruck)

Die Report-Ausgabe wird mit dem sich aus dem FORMAT-Ausdruck ergebenden Wert beendet. Dieser Wert ist auf Betriebssystemebene auswertbar (z.B. von einem Programm, das **repgo** aufgerufen hat).
Wird in einem Report ein EXIT durchlaufen, so wird an dieser Stelle die Bearbeitung abgebrochen.
Im PAGE MODE führt die Abarbeitung der EXIT-Anweisung zur Ausgabe der virtuellen Seite in der zu diesem Zeitpunkt beschriebenen Form.



```
        IF SQLCODE <> 0 THEN
                EXIT (SQLCODE)
```

Mit echo $? können Sie den SQLCODE in der Shell abfragen:

```
$echo $?
1oo
```

FITOBJECT “*Objektname*” Anzahl

Diese Anweisung dient zum Hinzufügen von Subobjekten zu einem Vaterobjekt, welches mit TESTOBJECT definiert wurde. “Objektname” enthält den Namen des Objekts. Anzahl definert, wieviele dieser Objekte gedruckt werden sollen. Weitere Hinweise finden sich im VisualREP Handbuch.

FITONPAGE

Mit dieser Anweisuung wird für die mit TESTOBJECT und FIXOBJECT definierten Objekte geprüft, ob diese noch auf die aktuellen Seite passen. Andernfalls erfolgt ein Seitenumbruch. Weitere Hinweise finden sich im VisualREP Handbuch.

FLUSH HEADER

Mit dieser Anweisung haben Sie die Möglichkeit die Ausgabe bzw. die Abarbeitung des PAGE-HEADERS zu veranlassen. An dieser Stelle sei auf einen späteren Abschnitt[1] verwiesen, in welchem die Problematik von Seiten- und Satzwechsel detailliert erläutert wird.

Achtung:

Erfolgt nach einem FLUSH HEADER keine Ausgabe mehr, so wird eine Leerseite gedruckt.

1. siehe *Seitenwechsel* auf Seite 88

```
...
PAGE HEADER
...
        IF flag = 1 THEN
                LET flag = 0
        ...
ON EVERY ROW
        ...
        LET flag = 1
        ...
        PRINT “KUNDE :”, kundna
{ Wurde durch das obige PRINT die Ausführung des PAGE TRAILERS
veranlaßt, so wird durch  FLUSH HEADER gewährleistet, daß bei der Ab-
arbeitung der IF-Anweisung bereits der PAGE HEADER abgearbeitet wurde. }
FLUSH HEADER                                    { flag ist jetzt wieder auf 0 gesetzt }
        IF flag = 0 THEN
                PRINT “----------”
        ELSE
                PRINT “++++++++++”
                ...
```

Ausgabe (flag=0) : KUNDE : Mueller

FOR *variable* = FORMAT-Ausdruck1 **TO** FORMAT-Ausdruck2
[**STEP** FORMAT-Ausdruck3] **DO** FORMAT-Anweisung

Mit dieser Anweisung kann eine Schleife gebildet werden. Dabei bekommt *variable* als Startwert den Wert von *FORMAT-Ausdruck1*. Ist STEP angegeben, dann wird *variable* nach jedem Schleifendurchlauf um den Wert von *FORMAT-Ausdruck3* erhöht. Ist STEP nicht angegeben dann wird *variable* nach jedem Schleifendurchlauf um 1 erhöht. In jedem Schleifendurchlauf wird die FORMAT-Anweisung ausgeführt. Durch die Blockanweisung können mehrere FORMAT-Anweisungen zusammengefaßt werden. Die Schleife wird nach einem Schleifendurchlauf abgebrochen, wenn der Wert von *variable FORMAT-Ausdruck2* entspricht.

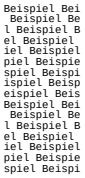
Achtung:

Werden in einer FOR-Schleife Ausgabeanweisungen verwendet und ist die Schleife in FIRST PAGE HEADER, PAGE HEADER oder PAGE TRAILER positioniert, so sollten die Höhen der einzelnen Ortsangaben angegeben werden, da REP in diesem Fall zur Compilierzeit die konkreten Höhen nicht bestimmen kann.

```
FOR i = 1 TO 10 STEP 2 DO
        PRINT i

FOR k = 10 TO 1 STEP -1 DO
        PRINT k
```

IF FORMAT-Bedingung **THEN** FORMAT-Anweisung1
ELSE FORMAT-Anweisung2

FORMAT-Anweisung1 wird ausgeführt, wenn die *FORMAT-Bedingung* wahr ist.
Ist die *FORMAT-Bedingung* nicht wahr und ELSE angegeben, so wird *FORMAT-Anweisung2* ausgeführt. Durch die Blockanweisung können mehrere FORMAT-Anweisungen zusammengefaßt werden.



```
IF SQLCODE <> 0 THEN
BEGIN
        PRINT "Fehler bei EXEC SQL-Anweisung"
        EXIT(SQLCODE)
END
```

LET *variable* = FORMAT-Ausdruck

Wertzuweisung an eine im DEFINE-Abschnitt definierte Variable.

KONVERSIONSTABELLE

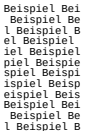
von / nach	CHAR	DATE	DECIMAL	FILE	FLOAT	SMALLFLOAT	INTEGER	SMALLINT	MONEY
CHAR	X	X	X	/	X	X	X	X	X
DATE	Datums-string	X	/	/	/	/	X	X	/
DECIMAL	num. Zeichen	/	X	/	X	X	X	X	X
FILE	/	/	/	X	/	/	/	/	/
FLOAT	num. Zeichen	X	X	/	X	X	X	X	X
SMALLFLOAT	num. Zeichen	X	X	/	X	X	X	X	X
INTEGER	num. Zeichen	X	X	/	X	X	X	X	X
SMALLINT	num. Zeichen	X	X	/	X	X	X	X	X
MONEY	num. Zeichen	/	X	/	X	X	X	X	X

X = Konversion wird versucht / = Konversion ist nicht möglich

Der Datentyp des zuzuweisenden FORMAT-Ausdrucks muß mit dem der Variablen im Sinne der obigen Konversionstabelle verträglich sein. Zeichenkette und Datum müssen in Hochkommata “ “ eingeschlossen werden.

Variablen vom Typ CHAR:

Es können mehrere Ausdrücke, durch Komma getrennt, zugewiesen werden. Zugewiesen wird die Zusammenfassung der Einzelausdrücke.

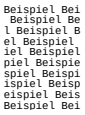


```
LET name    = "Karl Schiller "
LET adresse = “Karl Schiller ”, ort
LET tel     = vorwahl,”/”,durchwahl
```

Variablenausschnitte

variable [ausdruck]

definiert einen Ausschnitt der Variablen.
Der Ausschnitt besteht aus genau einem Zeichen an der angegebenen Position.



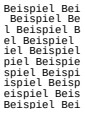
```
LET ja_nein  = antwort[1]
LET z4       = wort[4]
LET z12      = antwort[12]
```

variable [ausdruck1,ausdruck2]

definiert einen Ausschnitt aus der Variablen.
Der Ausschnitt beginnt an der Position ausdruck1 und endet an der Position ausdruck2.

Die Zuweisung an einen Feldausschnitt erfolgt bei

- einem Index: von Position bis Stringende
- zwei Indizes: von Position bis Endposition.



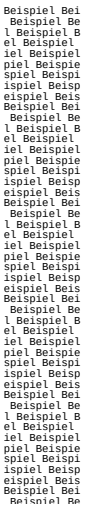
```
LET tag      = datum[1,2]
LET mon      = datum[4,6]
LET jahr     = datum[8,12]
```

Wertzuweisungen mit Zeitvariablen

Der Format-Ausdruck wird dem Datentyp bzw. dem Zeitausschnitt der Variablen angepaßt.
Hierbei gilt folgende Regel:

> Ist der Zeitausschnitt des Format-Ausdrucks kleiner als der der Variablen, so werden die links stehenden Komponenten mit dem aktuellen Datum, die rechts stehenden mit Nullen aufgefüllt.

Diese Regel gilt sowohl bei dem Datentyp DATETIME als auch bei dem Datentyp INTERVAL.



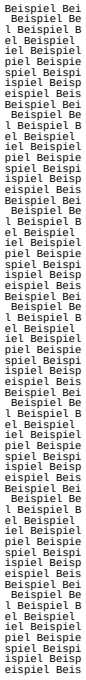
```
...
VARIABLE zp1     DATETIME YEAR TO SECOND
VARIABLE zp2     DATETIME HOUR TO SECOND
VARIABLE zp3     DATETIME DAY  TO HOUR
VARIABLE zs1     INTERVAL HOUR TO SECOND
...

LET zp1 = CURRENT                          { 1993-11-01 8:10:00}
LET zp2 = zp1                              { 8:10:00            }
LET zp1 = EXTEND ( zp2, YEAR TO SECOND )   { 1993-11-01 8:10:00}
LET zp3 = zp2                              { 01 8               }
LET zp1 = zp3                              { 1993-11-01 8:00:00 }
```

MOVEYX FORMAT-Ausdruck1 , FORMAT-Ausdruck2

Mit MOVEYX können Sie auf der aktuellen Seite[1] an eine beliebige Stelle positionieren.

FORMAT-Ausdruck1	Zeilenindex
FORMAT-Ausdruck2	Spaltenindex

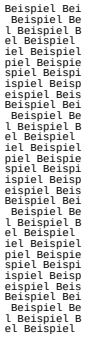
Die linke obere Ecke des beschreibbaren Bereiches hat als Zeilen- und Spaltenindex jeweils 1.



Es soll in die linke obere Ecke positioniert und der Wert der Variablen *zsumme* ausgegeben werden.

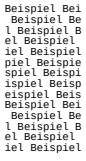
```
PAGE HEADER
    ...
    PRINT "ZWISCHENSUMME :";      { Die Ausgabe soll in Zeile 1,
                                    Spalte 8 erfolgen. }
PAGE TRAILER
...
    SAVEYX                        { Die aktuelle Position im
                                    PAGE TRAILER merken ... }
    MOVEYX 1,8                    { nach [1,8] positionieren ... }
    PRINT zsumme USING "###"      { Ausgabe tätigen ... }
    CALLYX                        { wieder in den PAGE TRAILER
                                    zurück. }
    PRINT "-",PAGENO USING "&&", "-"
```

Nach Abarbeitung der letzten PRINT-Anweisung würde ein Seitenumbruch stattfinden.



Es soll 1 Spalte mit 3 Einträgen angelegt werden, beginnend in Zeile 3.

```
...
MOVEYX 3, 1
PRINT wert1
MOVEYX 4, 1
PRINT wert2
MOVEYX 5, 1
PRINT wert3
...
```



Es soll in der jeweils aktuellen Zeile in die 4. Spalte positioniert werden.

```
...
MOVEYX LINENO, colno + 4
...
```

Positionieren Sie außerhalb der virtuellen Seite oder wird diese überschrieben, so wird je nach Warnstufe terminiert oder nach einer Warnung fortgefahren. Ein Seitenwechsel erfolgt nur dann, wenn mit einer Ausgabe-Anweisung (PRINT, SKIP, ...) das Ende der Seite erreicht oder SKIP TO TOP OF PAGE verwendet wird.

Achtung:

REP überprüft nicht, ob bereits an der aktuellen Positon eine Ausgabe stattgefunden hat, d.h. Daten werden grundsätzlich überschrieben. MOVEYX kann nicht über die aktuelle Seite hinaus positionieren.
MOVEYX darf in der INIT-Ortsangabe nicht verwendet werden.

1. siehe *Virtuelle Seite* auf Seite 86

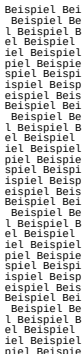
NEED FORMAT-Ausdruck **LINES**

Der *FORMAT-Ausdruck* muß eine ganze Zahl ergeben. NEED sorgt dann dafür, daß sich die betreffende Anzahl Zeilen auf einer Seite befinden, ohne durch einen Seitenwechsel getrennt zu sein. Ist auf der aktuellen Seite nicht mehr genügend Platz vorhanden, so wird ein Seitenwechsel[1] vollzogen, was die Ausführung des PAGE TRAILERS und unmittelbar danach die Ausführung des PAGE HEADERS zur Folge hat.

Achtung:

Hierbei ist folgender Effekt zu berücksichtigen:

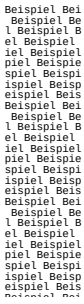
Bei der Ausführung des PAGE HEADERS hat ein Satzwechsel (Ende der ON EVERY ROW- Anweisungen) noch nicht stattgefunden, d.h. im PAGE HEADER aufgeführte Spalten beinhalten noch den Wert des vorherigen Satzes.



```
PAGE HEADER
    PRINT kundennummer
ON EVERY ROW
    PRINT auftragsname
    PRINT auftragsnummer
    PRINT auftragstext
    NEED 5 LINES
PAGE TRAILER
    PRINT "----------"
```

PRINT [FORMAT-Ausdruck, . . .]

gibt eine Zeile, beginnend auf der aktuellen Position, im Report auf dem im OUTPUT-Abschnitt angegebenen Ausgabemedium aus. Mehrere FORMAT-Ausdrücke werden direkt hintereinander ausgegeben. Das Ausgabeformat kann mit CLIPPED und USING beeinflußt werden.
PRINT darf in der INIT Ortsangabe nicht verwendet werden.



```
PRINT "Name :",name
PRINT "Name :",name[1,5]; { printed Zeichen 1 bis Zeichen 5 von Name }
PRINT " Vorname",vorname
PRINT 10 spaces,"Name :",name
PRINT COLUMN 10,"Name",name
PRINT "Preis:",preis USING "###.##"
PRINT "Seite -  ",PAGENO,"-"
PRINT "Name :",name WITHOUT TRAILING BLANKS
```

PRINT [FORMAT-Ausdruck, [FORMAT-Ausdruck, . . .]]**TO** *datei-variable*

Die Ausgabe erfolgt auf die mit OPEN im Schreibmodus (“w” oder “a”) geöffnete Datei. *datei-variable* muß im DEFINE-Abschnitt mit Typ FILE definiert und mit LET das Ergebnis von OPEN zugewiesen worden sein. Das Seitenlayout kann mit SET PAGE festgelegt werden.

1. siehe *Seitenwechsel* auf Seite 88

PRINT "Name :",*name* **TO** *meine_datei*
PRINT "Vorname",*vorname* **TO** *meine_datei*

PRINT [FORMAT-Ausdruck, [FORMAT-Ausdruck, . . .]] **TO STDERR**

Die Ausgabe erfolgt auf den Standardfehlerkanal. Dies ist für eigene Fehlerausgaben in Reports sinnvoll. Die Ausgabe erfolgt dabei unformatiert, d.h. es werden keine Seitenlängen und Ränder berücksichtigt. Auch die Angabe von COLUMN hat keine Wirkung.

PRINT [FORMAT-Ausdruck, . . .];

Der Zeilenvorschub nach Abschluß der PRINT-Anweisung wird unterdrückt.

PRINT "Name :",*name***;**
PRINT COLUMN 15, "Vorname :", *vorname*

PRINT FILE "*datei* " [TO *dateivariable*]

Die angegebene Datei *"datei"* wird auf das im OUTPUT-Abschnitt genannte Ausgabemedium oder auf die mit *dateivariable* verbundene Datei ausgegeben.

Achtung:

Das PRINT FILE-Kommando ist im FIRST PAGE HEADER, PAGE HEADER und im PAGE TRAILER mit besonderer Vorsicht zu verwenden, vor allem dann, wenn die Höhen der einzelnen Ortsangaben nicht angegeben werden.

PRINT FILE *"meine_statistik.txt"*

PRINTOBJECT "*Objektname* " ("Feldname1" = Ausdruck1, ..
"FeldnameN" = AusdruckN)

Füllt die Felder eines Objekts mit Werten. "Objektname" ist der Name des Objektes. Die Felder werden über "Feldname1" bis "FeldnameN" angegeben. Die Werte der Felder werden aus Ausdruck1 bis AusdruckN gebildet.

Weitere Hinweise zu diesem Befehl finden sich im VisualREP Handbuch.

PRINT PIPE "*programm*"

Die Standardausgabe des Programmes *"programm"* wird auf das im OUTPUT-Abschnitt gewählte Ausgabemedium ausgegeben.

PRINTTEXT variable

Mittels dieses Statements kann eine Variable vom Typ Text ausgegeben werden. Solche Variablen entstehen, wenn Daten aus einer Spalte vom Typ TEXT (CLOB unter Oracle) selektiert wird. Beim Zugriff auf die Variable wird sie in eine Zeichenkette umgewandelt Im Gegensatz zum normalen Printbefehl werden von PRINTTEXT Zeilenumbrüche erkannt und als solche ausgegeben. Auch ein Seitenumbruch zwischen den einzelnen Zeilen ist möglich.

SAVEYX

Die Anweisung SAVEYX speichert die aktuelle Position der virtuellen Seite. Mit CALLYX (s.u.) bewegen Sie sich auf die mit SAVEYX gespeicherte Position. Wird SAVEYX mehrmals nacheinander aufgerufen, so wird die gespeicherte Position mit der jeweils neuen überschrieben, d.h. die in SAVEYX gespeicherte Position bleibt bis zur Speicherung einer neuen Position durch SAVEYX erhalten.

Achtung:

SAVEYX darf in der INIT-Ortsangabe nicht verwendet werden.

SET ATTRIBUTES FORMAT-Ausdruck

Mittels dieser Anweisung können die aktuellen Attributte für die nächste Ausgabe festgelegt werden. *FORMAT-Ausdruck* muß dabei einen Bitvektor repräsentieren, welcher die Attribute festlegt. Diese Attribute sind bis zur nächsten SET ATTRIBUTES Anweisung gültig. Alle zu diesem Zeitpunkt gesetzen Attribute werden überschrieben.

```
DEFINE
        VARIABLE ul INTEGER             { ul entspricht Unterstreichen }
        VARIABLE bold INTEGER           { bold entspricht Fettdruck }
…
INIT
        LET bold = 1
        LET ul = 2
PAGE HEADER
        SET ATTRIBUTES ul
        PRINT “Liste aller “;
        SET ATTRIBUTES bold
        PRINT “Kunden vom “;
        SET ATTRIBUTES ul|bold
        PRINT “20.10.96”
                ...
```

Der Text “Liste aller” wird unterstrichen ausgegeben. Der Text “Kunden vom” wird in Fettschrift ausgegeben und das Datum in Fettschrift und unterstrichen.

SET PAGE TM, **BM**, **LM**, **RM**, **PL**, **FP**, **FPHL**, **PHL**, **PTL** [**FOR** *dateizeiger*]

Hiermit kann das Layout sowohl für die Standardausgabe als auch für eine Datei festgelegt werden. Soll das Ausgabeformat für eine Datei konfiguriert werden, so muß der entsprechende Dateizeiger angegeben werden.

Parameter:

TM:	Oberer Report-Rand
BM:	Untere Report-Rand
LM:	Linker Report-Rand
RM:	Rechter Report-Rand
PL:	Seitengröße
FP:	Anfangs-Seitennummer

** Die nachfolgenden Parameter sind nur bei der Standardausgabe zulässig **

FPHL:	Höhe des FIRST PAGE HEADERS
PHL:	Höhe des PAGE HEADERS
PTL:	Höhe des PAGE TRAILERS
dateizeiger	ist ein Zeiger auf die Datei, welche mit dem OPEN-Befehl im Schreibmodus (“w”) geöffnet sein muß.

Alle Parameter können als FORMAT-Ausdrücke angegeben werden.

Eine Plausibilitätsprüfung findet zur Laufzeit statt.

Achtung:

- Alle Ausgabekanäle mit Ausnahme der Standardausgabe können genau einmal initialisiert werden. Ein Ausgabekanal, der als Ziel eine Datei hat, wird mit OPEN erzeugt und mit CLOSE gelöscht.
- Ist eine Layout-Festlegung nach dem ersten PRINT TO nicht erfolgt, so gilt das DEFAULT-Layout (OUTPUT-Abschnitt).
- Bei der Standardausgabe kann eine im OUTPUT-Abschnitt implementierte Layout-Beschreibung mittels SET PAGE überschrieben werden.
- Die SET PAGE-Anweisung für die Standardausgabe ist nur innerhalb der INIT-Ortsangabe zulässig.

```
...
        DEFINE
                param[1]    links       INTEGER
                param[2]    rechts      INTEGER
                variable    datei_zeiger FILE
        ...
        INIT
            ...
            datei_zeiger = OPEN(“zweite_ausgabe”,”w”)

            { Layout-Festlegung für die Datei zweite_ausgabe }
            SET PAGE 5, 5, 10, 90, 66, 1 FOR datei_zeiger

            { Layout-Festlegung für die MAIN-PAGE
            “links” und “rechts” sind Parameter und legen die Zeilenbreite fest. }
            SET PAGE 5, 5, links, rechts, 66, 1, 3, 3, 5
            ...
        ON EVERY ROW
        ...
            PRINT “Name :”,name TO datei_zeiger
            PRINT “Vorname”,vorname TO datei_zeiger
```

SETNULL *variable*

Setzt die Variable *variable* auf den Wert NULL. Die Variable muß im Define-Abschnitt deklariert worden sein und der Gruppe VARIABLE zugeordnet sein. Sie darf nicht vom Datentyp FILE sein.

SET OBJECT *“Objektname”*

Legt ein Objekt für die Ausgabe im VisualSimple Modus fest. “Objektname” ist der Name des Objektes. Weitere Informationen finden sich im VisualREP Handbuch.

SKIP *anzahl* **LINES**

Erzeugt *anzahl* Leerzeilen.
Werden mehr Leerzeilen geschrieben, als auf der aktuellen Seite Platz ist, wird ein Seitenwechsel vollzogen (Ausführung des PAGE HEADERS, PAGE TRAILERS).

SKIP *10* **LINES**

SKIP TO TOP OF PAGE

Erzeugt einen Vorschub auf den Anfang der nächsten Seite, wobei PAGE HEADER und PAGE TRAILER ausgeführt werden. Die Anweisung darf nicht bei den Ortsangaben FIRST PAGE HEADER, PAGE-HEADER und PAGE-TRAILER angegeben werden.

Achtung:

Bitte beachten Sie, daß bei einem SKIP TO TOP OF PAGE das Flag (-w l) keine Wirkung hat.

SYSTEM (FORMAT-Ausdruck)

Ausführen eines UNIX-Kommandos mit dem Interpreter. Der FORMAT-Ausdruck kann eine in Hochkommata “ “ eingeschlossene Zeichenkette oder eine Variable vom Typ CHAR sein. Hierbei ist zu beachten, daß die Ausgabe eines durch SYSTEM gestarteten Programms nach Standard-Output geht. Daher ist die Ausgabe eines Kommandos nicht als Reportausgabe zu verstehen. Im PAGEMODE erfolgt demnach die Ausgabe nicht auf die virtuelle Seite.

Siehe auch: *SPACES FORMAT-Ausdruck* auf Seite 81

LET *cmd* = “echo *’10.000 Seiten bearbeitet . . . ‘* ”
SYSTEM(*cmd*)

TESTOBJECT “Objectname**”**

Legt ein Objekt fest, für welches geprüft werden soll, ob es noch auf die Seite passt. “Objectname” enthält den Namen des Objekts. Weitere Informationen sind im VisualREP Handbuch zu finden.

WHILE FORMAT-Bedingung **DO** FORMAT-Anweisung

Die FORMAT-Anweisung wird, solange die FORMAT-Bedingung wahr ist, ausgeführt. Durch die Blockanweisung können mehrere FORMAT-Anweisungen zusammengefaßt werden.

Achtung:

Werden in einer WHILE -Schleife Ausgabeanweisungen verwendet und ist die Schleife in FIRST PAGE HEADER, PAGE HEADER oder PAGE TRAILER positioniert, so sollten die Höhen der einzelnen Ortsangaben angegeben werden, da REP in diesem Fall zur Compilierzeit die konkreten Höhen nicht bestimmen kann.


```
LET X = 0
WHILE X <> 10 DO
        BEGIN
                LET X = X + 1
                PRINT X
        END
```

3 FORMAT-Bedingung

Bedingungen finden bei der WHILE- und der IF- Anweisung ihre Anwendung.
Eine Bedingung kann sich aus Operatoren mit einem Argument oder aus Operatoren mit zwei Argumenten zusammensetzen, wobei die Argumente FORMAT-Ausdrücke sind. Boolsche Operatoren (OR, AND und NOT) kombinieren FORMAT-Bedingungen.

3.1 Operatoren

Vergleichsoperatoren mit einem Argument

Die allgemeine Form eines Vergleiches mit einem Argument lautet:

FORMAT-Ausdruck vergleichsoperator

vergleichsoperator

IS NULL	wahr, wenn FORMAT-Ausdruck den Wert NULL hat.
IS NOT NULL	wahr, wenn FORMAT-Ausdruck nicht den Wert NULL hat.
MATCHES "String"	wahr, wenn in FORMAT-Ausdruck der String enthalten ist.

Vergleichsoperatoren mit 2 Argumenten

Operatoren mit 2 Argumenten haben folgende Form:

FORMAT-Ausdruck1 vergleichsoperator **FORMAT-Ausdruck2**

Mögliche Operatoren, welche zum Vergleich zweier FORMAT-Ausdrücke verwendet werden können, sind nachfolgend beschrieben.

vergleichsoperator

=	wahr, wenn FORMAT-Ausdruck1 gleich FORMAT-Ausdruck2
!=	wahr, wenn FORMAT-Ausdruck1 ungleich FORMAT-Ausdruck2
<	wahr, wenn FORMAT-Ausdruck1 kleiner FORMAT-Ausdruck2
>	wahr, wenn FORMAT-Ausdruck1 größer FORMAT-Ausdruck2
<=	wahr, wenn FORMAT-Ausdruck1 kleiner oder gleich FORMAT- Ausdruck2

>=	wahr, wenn FORMAT-Ausdruck1 größer oder gleich FORMAT-Ausdruck2

Bool'sche Operatoren

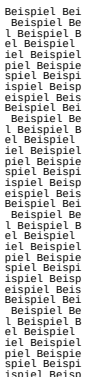
AND und OR benötigen als Argumente zwei FORMAT-Bedingungen, NOT erwartet eine FORMAT-Bedingung.

AND	wahr, wenn FORMAT-Bedingung1 und FORMAT-Bedingung2 wahr ist.
OR	wahr, wenn FORMAT-Bedingung1 oder FORMAT-Bedingung2 wahr ist.
NOT	wahr, wenn FORMAT-Bedingung nicht wahr ist.

Achtung:

Ist in einer Bedingung ein FORMAT-Ausdruck enthalten, welcher einen NULL-Wert enthält, treffen folgende Ereignisse zu:

IF-Bedingung:	Es wird der ELSE Zweig ausgeführt.
WHILE-Bedingung:	Beinhaltet der Format-Ausdruck in der Schleifen-Bedingung zu einem Zeitpunkt einen NULL-Wert, so wird die Schleife abgebrochen.



Format-Bedingungen:

zahl **>** 5

NOT(*zahl***<=**5)

z **<** 5 **OR** z **>** 6 **AND NOT** y **=** 5 **AND** Y **=** 3 entspricht
z **<** 5 **OR** (z **>** 6 **AND NOT**(y **=** 5 **AND** Y **=** 3))

NOT(*zahl* **IS NULL OR** *zahl* **=** 5) entspricht
zahl **IS NOT NULL AND** *zahl* **=** 5

Wirkung und Priorität aller Operatoren sind in dem Abschnitt FORMAT-Ausdruck nachzulesen.

3.2 Bedingungen mit Zeitdatentypen

Bedingungen mit Zeitpunkten

Sie können Zeitpunkt (DATETIME) und Datum (DATE) miteinander vergleichen, wobei ein Zeitpunkt bzw. Datum größer als ein anderer Zeitpunkt oder ein anderes Datum ist, wenn es jünger ist.

```
...
DEFINE VARIABLE d1    DATE
DEFINE VARIABLE dt1   DATETIME YEAR TO DAY
...
        LET d1 = "01-01-1991"
        LET dt1 = DATETIME ( 1993-01-10 ) YEAR TO DAY
        IF ( d1 > dt1 ) THEN
                PRINT "d1 liegt vor dt1"
        ELSE
                {
                Dieser Teil wird ausgefuehrt.
                }
                PRINT "dt1 ist juenger als d1"
```

Bedingungen mit Zeitspannen

Sie haben die Möglichkeit, Zeitspannen miteinander zu vergleichen. Voraussetzung hierfür ist, daß sie den gleichen Typ haben.

```
...
DEFINE VARIABLE zs1   INTERVAL YEAR TO MONTH
DEFINE VARIABLE zs2   INTERVAL YEAR TO MONTH
...
        LET zs1 = "10-02"
        LET zs2 = INTERVAL ( 12-01 ) YEAR TO MONTH
        IF ( zs1 > zs2 ) THEN
                PRINT "zs1 ist groesser als zs2"
        ELSE
                {
                Dieser Part wird ausgefuehrt.
                }
                PRINT "zs2 ist groesser oder gleich als zs1"
```

4 FORMAT-Ausdruck

FORMAT-Ausdrücke können einfache Zahlen, alphabetische Konstanten oder komplexe Folgen von Funktionen, Strings, Operatoren und Schlüsselwörtern sein. Wird ein Ausdruck berechnet, so werden alle Elemente, die durch Operatoren verbunden sind, ausgewertet. Welche Elemente zusammengefaßt werden, hängt von der Priorität der Operatoren ab, wobei diese durch die Klammersetzung verändert werden kann.
Priorität und Funktion der einzelnen Operatoren zeigt folgende Tabelle (1 ist die höchste Priorität).

Operator	Funktion	Priorität
-	Vorzeichen	1
**	Potenz	2

Operator	Funktion	Priorität
*	Multiplikation	3
/	Division	3
%	Modulo	3
&	Bit-Verundung	3
|	Bit-Veroderung	3
^	Bit-Xor	3
>>	Bit-Rechtsshift	3
<<	Bit-Linksshift	3
+	Addition	4
-	Subtraktion	4

FORMAT-Bedingung

IS[NOT]NULL	ein/kein Null-Wert	5
MATCHES	Stringvergleich	5
=	gleich	5
!= oder <>	ungleich	5
>	größer	5
<	kleiner	5
>=	größer gleich	5
<=	kleiner gleich	5
NOT	Negation	6
AND	Und-Verknüpfung	7
OR	Oder-Verknüpfung	8

4.1 Format-Ausdrücke mit Zeitdatentypen

Die nachfolgende Tabelle gibt Auskunft darüber, welche Format-Ausdrücke mit Datum, Zeitpunkt und Zeitspanne gebildet werden können.

Format-Ausdruck1	Operator	Format-Ausdruck2	Resultat
DATE	-	DATETIME	INTERVAL
DATETIME	-	DATE	INTERVAL
DATE	+/-	INTERVAL	DATETIME
DATETIME	-	DATETIME	INTERVAL
DATETIME	+/-	INTERVAL	DATETIME

Format-Ausdruck1	Operator	Format-Ausdruck2	Resultat
INTERVAL	+	DATETIME	DATETIME
INTERVAL	+/-	INTERVAL	INTERVAL
DATETIME	-	CURRENT	INTERVAL
CURRENT	-	DATETIME	INTERVAL
INTERVAL	+	CURRENT	DATETIME
CURRENT	+/-	INTERVAL	DATETIME
DATETIME	+/-	UNITS	DATETIME
INTERVAL	+/-	UNITS	INTERVAL

Weiterhin ist beim Rechnen mit Zeitdatentypen folgendes zu beachten:

- Bei der Addition und Subtraktion zweier Intervalwerte dürfen das Ergebnis und die Operanden nicht größer sein als:

  - der maximal darstellbare Wert vom Typ INTERVAL SECONDS(9) TO SECONDS, wenn die Intervalwerte im Bereich von Tagen bis Sekunden liegen.

  - der maximal darstellbare Wert vom Typ INTERVAL MONTH(9) TO MONTH, wenn die Intervalwerte im Bereich von Jahren bis Monaten liegen.

- Bei der Addition von 2 Intervalwerten ergibt die die Genauigkeit des Ergenisses aus der maximalen Genauigkeit der beiden Operanten.

- Intervalwerte im Bereich von Jahren und Monaten dürfen nicht zu Intervalwerten im Bereich von Tagen und Sekunden addiert werden.

- Bei der Subtraktion von Datentimes richtet sich die Genauigkeit des Ergebnises nach dem ersten Operanten (Minuend). Der zweite Wert (Subtrahend) wird vorher auf die Gnauigkeit des Minuenden durch Extend angepasst (vorne mit der aktuellen Zeit ergänzt, hinten mit 0 aufgefüllt und überflüssige Komponenten abschneiden).Umfasst der Qualifier des ersten Arguments eine der Komponenten DAY-F5 und ist die erste Komponete größer als DAY, dann ist die erste Komponente des Ergebnisses DAY. Ist die erste Komponente des Subtrahenden kleiner/gleich DAY, dann wird diese auch als erste Komponente des Ergebnisses verwendet. Als letzte Komponete des Ergebnisses wird die letzte Komponente des Subtrahenden verwendet.·Umfasst der Qualifier des Minuenden nur die Komponenten YEAR und/oder MONTH, dann ist das Ergebnis vom Typ YEAR(9) to MONTH.

```
...
DEFINE VARIABLE    zs1   INTERVAL  HOUR TO SECOND
DEFINE VARIABLE    zs2   INTERVAL  HOUR TO SECOND
DEFINE VARIABLE    zp1   DATETIME  HOUR TO SECOND
DEFINE VARIABLE    zp2   DATETIME  HOUR TO SECOND
DEFINE VARIABLE    d     DATE
...

    PRINT zs1 + zs2                          { Ergebnis : INTERVAL }
    PRINT zp1 + zs1                          { Ergebnis : DATETIME }
    PRINT zp1 - zs1                          { Ergebnis : DATETIME }
    PRINT zp1 + ( 3 ) UNITS DAY              { Ergebnis : DATETIME }
    PRINT CURRENT + zs1                      { Ergebnis : DATETIME }
    PRINT zp1 - d                            { Ergebnis : INTERVAL }
    PRINT zp1 - zp2                          { Ergebnis : INTERVAL }
    PRINT TODAY - INTERVAL ( 1:00:00 )
                HOUR TO SECOND               { Ergebnis : INTERVAL }
```

4.1.1 Zeitpunkte und Zeitspannen als Konstante

Eine Zeichenkette oder ein Datum müssen in Anführungszeichen eingeschlossen werden. Kommazahlen müssen den Dezimalpunkt enthalten, wenn Nachkommastellen existieren. Hinzu kommen die speziellen Angaben von Konstanten, welche den Datentyp DATETIME oder INTERVAL implizieren.

Eine DATETIME-Konstante kann folgende Form haben:

DATETIME Angabe als String

"komponentenwerte"

In diesem Fall werden die Werte zur Zeitpunktdarstellung in einem String angegeben. Das Format entspricht dem der Literal-Form.

DATETIME Angabe in Literal-Form

Die Komponentenwerte werden entsprechend der Anfangs- und Endkomponente konvertiert. Die möglichen Kombinationen der Komponenten entsprechen denen der Variablendeklaration einer Variablen vom Typ DATETIME.

DATETIME (*komponentenwerte*) *komponente1* **TO** *komponente2*

komponentenwerte Dieser Wert muß folgende Struktur aufweisen:

YY[YY]-MM-TT HH:NN:SS.FFF

- *YY[YY]* Angabe der Jahreszahl. Ist diese kleiner als 100, so wird 1900 hinzuaddiert.
- *MM*Angabe des Monats (01 - 12)
- *TT*Angabe des Tages (01 - 31)
- *HH*Angabe der Stunden (00 - 24)
- *NN*Angabe der Minuten (00 - 59)
- *SS* Angabe der Sekunden (00 - 59)
- *FFF*Angabe der Sek.-Bruchteile

Bei der Angabe der Komponentenwerte muß darauf geachtet werden, daß die Darstellung dem durch die Anfangs- und Endkomponente festgelegten Intervall entspricht. Die Trennzeichen zwischen den Komponentenwerten müssen exakt eingehalten werden.

Folgende Trennzeichen sind zu verwenden:

- Bindestrich zwischen Jahr, Monat und Tag
- Leerzeichen zwischen Tag und Stunde
- Doppelpunkt zwischen Stunde, Minute und Sekunde
- Punkt zwischen Sekunde und Sekundenbruchteile

```
...
DEFINE
        VARIABLE x          DATETIME YEAR TO DAY
      ...
END
FORMAT
        ....
        LET x = "1993-02-12"
        PRINT DATETIME (1993-02-12) YEAR TO DAY
        ...
END
```

Eine INTERVAL-Konstante kann folgende Form haben:

INTERVAL Angabe als String

"komponentenwerte"

In diesem Fall werden die Werte zur Zeitspannendarstellung in einem String angegeben. Das Format entspricht dem der Angabe in Literal-Form.

INTERVAL Angabe in Literal-Form

Die Komponentenwerte werden entsprechend der Anfangs- und Endkomponente konvertiert. Die möglichen Kombinationen der Komponenten entsprechen denen der Variablendeklaration einer Variablen vom Typ INTERVAL.

INTERVAL (*komponentenwerte*) *komponente1* **TO** *komponente2*

komponentenwerte Dieser Wert muß folgende Struktur aufweisen

- Jahre-Monate
- Tage Stunden:Minuten
- Jahre
- Monate
- Tage
- Stunden
- Minuten
- Sekunden
- Sekundenbruchteile

Bei der Angabe der Komponentenwerte muß darauf geachtet werden, daß die Darstellung dem durch die Anfangs- und Endkomponente festgelegten Intervall entspricht.
Die Trennzeichen zwischen den Komponentenwerten müssen exakt eingehalten werden.

Folgende Trennzeichen sind zu verwenden:

- Bindestrich zwischen Jahr und Monat
- Leerzeichen zwischen Tag und Stunde
- Doppelpunkt zwischen Stunde, Minute und Sekunde

- Punkt zwischen Sekunde und Sekundenbruchteil

```
...
DEFINE
        VARIABLE x        INTERVAL YEAR(2) TO MONTH
      ...
END
FORMAT
        ....
        LET x = "5-2"
        PRINT INTERVAL (5-2) YEAR(2) TO MONTH
        ...
END
```

4.2 Bitoperationen

REP verfügt über verschiedene Operatoren, welche ein Arbeiten auf Bit-Ebene ermöglichen. Dieses ist vor allem dann nützlich, wenn in einer Zahl verschiedene Informationen in den einzelnen Bits verschlüsselt sind.

4.2.1 Interne Darstellung von Zahlen

Zur internen Darstellung einer Variablen von einem bestimmten Datentyp werden verschiedene Anzahlen von Bits verwendet. Die interne Darstellung von ganzzahligen Werten wird in REP in Form eines Integers gehandhabt.

Folgende Operatoren können Sie in einem Report verwenden:

Operator	Funktion
|	ODER-Verknüpfung
^	EXCLUSIVE-ODER-Verknüpfung von 2 ganzzahligen Werten
&	UND-Verknüpfung von 2 ganzzahligen Werten
>>	Verschiebung der Bits nach rechts
<<	Verschiebung der Bits nach links

Alle Operatoren verknüpfen ihre Argumente bitweise. Dazu ein Beipiel:

255 & 64 wird wie folgt berechnet: `11111111`

`&  01000000`

`=  01000000`

Jedes Bit eines Operators wird mit dem entsprechenden Bit des anderen Operators anhand der gegebenen Operation verknüpft. Entsprechend wird das jeweilige Ergebnisbit gesetzt.

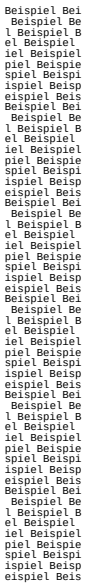
4.2.2 ODER-Verknüpfung

```
FORMAT-Ausdruck1 | FORMAT-Ausdruck2
```

FORMAT-Ausdruck1	ganze Zahl
FORMAT-Ausdruck2	ganze Zahl

Der ODER-Verknüpfung liegt folgende Wahrheitstabelle zugrunde:

Bit1	Bit2	
0	0	0
0	1	1
1	0	1
1	1	1



Es soll in der Variablen *aktion* das Druckbit gesetzt werden.

```
DEFINE
        VARIABLE drucken        SMALLINT
        VARIABLE aktion         SMALLINT
...
INIT
        LET aktion = 0          { Interne Darstellung 00000000 00000000 }
        LET drucken = 2         { Interne Darstellung 00000000 00000010 }
...
FORMAT
...
LET aktion = aktion | drucken
...
END
```

4.2.3 EXCLUSIVE-ODER-Verknüpfung

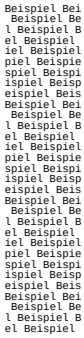
FORMAT-Ausdruck1 ^ FORMAT-Ausdruck2

FORMAT-Ausdruck1 ganze Zahl

FORMAT-Ausdruck2 ganze Zahl

Der EXCLUSIVE-ODER-Verknüpfung liegt folgende Wahrheitstabelle zugrunde:

Bit1	Bit2	
0	0	0
0	1	1
1	0	1
1	1	0



Es sollen alle die Bits gefunden werden, in welchen sich die Bitleisten *aktion1* und *aktion2* unterscheiden.

```
...
LET aktion_diff = aktion1 ^ aktion2
...
END
```

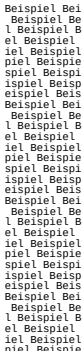
4.2.4 UND-Verknüpfung von Bits

FORMAT-Ausdruck1 & FORMAT-Ausdruck2

FORMAT-Ausdruck1	ganze Zahl
FORMAT-Ausdruck2	ganze Zahl

Der UND-Verknüpfung liegt folgende Wahrheitstabelle zugrunde:

Bit1	Bit2	
0	0	0
0	1	0
1	0	0
1	1	1



Es soll geprüft werden, ob bei der Variablen *aktion* das 2 Bit gesetzt ist.

```
...
FORMAT
...
IF aktion & drucken = drucken THEN
        PRINT “Druckbit ist gesetzt.”
...
END
```

4.2.5 Verschieben von Bits nach rechts

FORMAT-Ausdruck1 >> FORMAT-Ausdruck2

FORMAT-Ausdruck1	ganze Zahl, deren Bits um FORMAT-Ausdruck2 Stellen nach rechts verschoben werden.
FORMAT-Ausdruck2	ganze Zahl

Alle Bits der Variablen *aktion* sollen angezeigt werden.

```
...

{
In diesem Beispiel werden nacheinander von rechts nach links alle Bits an
die erste Position geshiftet und anschließend mit 1 verundet werden, d.h. das
Ergebnis ist entweder 1 oder 0.
}

FOR i = 0 TO 15 DO
    BEGIN
        PRINT aktion >> (15 - i) & 1;
    END
PRINT ""
...
```

4.2.6 Verschieben von Bits nach links

FORMAT-Ausdruck1 << FORMAT-Ausdruck2

FORMAT-Ausdruck1	ganze Zahl, deren Bits um FORMAT-Ausdruck2 Stellen nach links verschoben werden.
FORMAT-Ausdruck2	ganze Zahl

Bei dem Wert *val* sollen die Bits um 2 nach links geschoben werden.

```
...
LET val = val << 2
...
```

4.3 Funktionen

Nachfolgend werden in alphabetischer Reihenfolge alle Funktionen beschrieben, die in FORMAT-Ausdrücken verwenden werden können.

ASCII (Format-Ausdruck)

Ermittelt für einen ASCII-Code das entsprechenden ASCII-Zeichen.

Ausgabe des ASCII-Wertes 49 als Geldbetrag (1.00)

```
DEFINE
        VARIABLE asc_wert MONEY
END

LET asc_wert = ASCII(49)
PRINT asc_wert
```

ATAN(FORMAT-Ausdruck)

Berechnet den Arcustangens von *FORMAT-Ausdruck* im Bogenmaß.

ATTRIBUTES

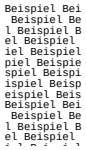
Diese Funktion liefert die aktuellen Attribute. Zusammen mit einer SET ATTRIBUTES Anweisung ist es damit möglich, bestimmte Attribute ein- oder auszublenden, wie das folgende Beispiel zeigt:

```
DEFINE
        VARIABLE bold                integer
        VARIABLE underline           integer
        VARIABLE italic              integer
END

SELECT v1 from ...

FORMAT
INIT
        LET bold          = 2
        LET underline     = 4
        LET italic        = 8
ON EVERY ROW
        IF (v1 = 1) THEN
                SET ATTRIBUTES bold
        ELSE
                SET ATTRIBUTES italic
        PRINT “TEXT1”

        SET ATTRIBUTES ATTRIBUTES | underline
        { setzt underline - bold oder italic bleiben erhalten }
        PRINT “TEXT2”

        SET ATTRIBUTE ATTRIBUTES & (255 - underline)
        { schaltet nur underline aus, bold oder italic bleiben erhalten}
      PRINT “TEXT3”
END
```

Hat *v1* den Wert 1, dann wird “Text1” und “Text3” mit dem Attribute *bold*, “Text2” mit dem Attribute *bold/underline* ausgegeben. Hat *v1* nicht den Wert 1, dann wird “Text1” und “Text3” mit dem Attribute *italic* , “Text2” mit dem Attribute *italic/underline* ausgegeben.

CHR (Format-Ausdruck)

Ermittelt für ein ASCII-Zeichen den entsprechenden ASCII-Code.



Ausgabe des ASCII-Codes des ASCII-Zeichens ‘0’ (= 48)

```
LET init_wert = CHR('0')
PRINT init_wert
```

CLIPPED

Bei CHAR-Werten werden nachfolgende Leerzeichen abgeschnitten.

Achtung:

Mit CLIPPED kann die eigentliche Größe einer Variablen vom Datentyp CHAR nicht verändert werden.

COLNO

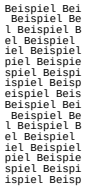
Ist die momentane Spaltennummer als numerische Konstante.

COLUMN FORMAT-Ausdruck

Positioniert in die durch FORMAT-Ausdruck spezifizierte Spalte des Reports (nur sinnvoll in Verbindung mit PRINT).

Achtung:

Achten Sie bei der Positionierung immer auf die Zeilenlänge. (RIGHT MARGIN - LEFT MARGIN): Darüber hinaus darf nicht positioniert werden. Je nach Customisation Flag[1] wird eine Warnung ausgegeben, bzw. abgebrochen.



```
DEFINE
    VARIABLE culpos INTEGER
    ...
    PRINT COLUMN 20, "Name : ", name
    PRINT COLUMN colpos + 1
```

COS (FORMAT-Ausdruck**)**

Berechnet den Cosinus von *FORMAT-Ausdruck* im Bogenmaß.

1. siehe *Customization Flags* auf Seite 14

CURRENT [komponente1 **TO** komponente2]

Ergibt das aktuelle Datum und die aktuelle Uhrzeit im Datentyp DATETIME.
Die möglichen Kombinationen der Komponenten entsprechen denen der Variablendeklaration einer Variablen vom Typ DATETIME.

DATE (FORMAT-Ausdruck)

Ergibt das Datum als Zeichenkette. Die Variable zur Aufnahme des Datums muß vom Datentyp CHAR sein.

DAY (FORMAT-Ausdruck)

Zahl des Tages als Zahl zwischen 1 und 31. Der FORMAT-Ausdruck muß vom Datentyp DATE sein.

EOF (*datei-variable*)

Prüft, ob das Ende der mit *datei-variable* verbundenen Betriebssystemdatei erreicht ist.
Rückgabewert :

1	Dateiende erreicht
0	Dateiende nicht erreicht

```
IF EOF(fp) = 1 THEN
        PRINT “Dateiende erreicht !”
```

ESYSTEM (FORMAT-Ausdruck**)**

Diese Funktion entspricht der Format-Anweisung SYSTEM und liefert den Exit-Code des gestarteten Kommandos.

EXP(FORMAT-Ausdruck**)**

Berechnet den Exponentialwert von *FORMAT-Ausdruck*.

EXTEND (zeitpunkt, [komponente1 **TO** komponente2])

Dient dazu, Komponenten eines Zeitpunktes zu ergänzen bzw. zu streichen.
Diese Funktion wird bei Berechnungen mit Zeitpunkten und Zeitspannen benötigt, um einzelne Terme in einem Format-Ausdruck anzupassen. Die möglichen Kombinationen der Komponenten entsprechen denen der Variablendeklaration einer Variablen vom Typ DATETIME.
zeitpunkt kann als DATETIME-Konstante oder als Variable angegeben werden.

GETENV (*Umgebungsvariable)*

Liefert den Wert der angegebenen *Umgebungsvariable* als String

GETRES (*Resourcename)*

Liefert den Wert der angegebenen Resource *Resourcename* als String.

INT(FORMAT-Ausdruck**)**

Berechnet den ganzzahligen Anteil von *FORMAT-Ausdruck*.

LINENO

Ist die momentane Zeilennummer als numerische Konstante. LINENO liefert den Wert 1 zurück, wenn die aktuelle Position die erste Zeile im beschreibbaren Bereich[1] ist.

Achtung:

TOP MARGIN und BOTTOM MARGIN sind mit LINENO nicht indiziert.

LOG10(FORMAT-Ausdruck**)**

Berechnet den Logarithmus von *FORMAT-Ausdruck* zur Basis 10.

LOG2(FORMAT-Ausdruck**)**

Berechnet den Logarithmus von *FORMAT-Ausdruck* zur Basis 2.

MDY(FORMAT-Ausdruck1, FORMAT-Ausdruck2, FORMAT-Ausdruck3)

Umwandlung eines Datums in den Typ DATE. Hierbei stehen die Ausdrücke für folgende Inhalte:

FORMAT-Ausdruck1 :	Monat als Zahl (1 .. 12)
FORMAT-Ausdruck2 :	Tag als Zahl (1 .. 31)
FORMAT-Ausdruck3 :	Jahreszahl (vierstellig)

MONTH (FORMAT-Ausdruck)

Zahl des Monats als Zahl zwischen 1 und 12. Der FORMAT-Ausdruck muß vom Datentyp DATE sein.

OPEN (*datei* , *modus*)

Die Betriebssystemdatei *datei* wird im angegebenen *modus* geöffnet. *datei* kann eine Zeichenkette in Hochkommata “ “ oder eine Variable vom Typ CHAR, die den Dateinamen enthält, sein.

modus kann einen der folgenden Zeichenketten enthalten:

“r”:	Lesemodus

1. siehe *Randgrößen* auf Seite 32

“w”: Schreibmodus

“a”: am Ende der Datei anfügen

Das Ergebnis von OPEN ist ein Dateizeiger. Er muß einer im DEFINE-Abschnitt definierten Variable vom Typ FILE zugewiesen werden. Das Ausgabeformat in eine Datei kann mit dem SET PAGE- Kommando festgelegt werden. Nach der ersten Ausgabe auf die mit OPEN geöffnete Datei kann mit der SET PAGE-Anweisung kein Layout für diesen Ausgabekanal mehr festgelegt werden.

```
LET fp = OPEN (“test.dat”, “w”)
SET PAGE 10,10,10,80,60,1 FOR fp
```

PAGENO

Ist die momentane Seitennummer als numerische Konstante.

READ (*datei-variable, modus*)

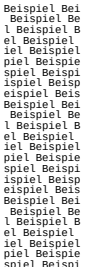
Aus der mit *datei-variable* verbundenen Betriebssystemdatei, die im Lesemodus eröffnet sein muß, wird in Abhängigkeit von *modus* gelesen.

modus kann dabei eine der folgenden Zeichenketten enthalten

"c": ein Zeichen wird gelesen

"w": ein Wort wird gelesen. Default-Trenner ist Blank.Definieren Sie im DEFINE-Abschnitt eine Variable vom Datentyp CHAR mit dem Namen **delimiter**, so können Sie den Trenner frei festlegen.

"l": eine ganze Zeile wird gelesen (bis zum Linefeed)



Als Worttrenner soll das Zeichen | festgelegt werden.

```
DEFINE
VARIABLE delimiter CHAR(1)
...
LET delimiter = “|”
...
LET satz = READ (fp, “w”)
```

SIN (FORMAT-Ausdruck)

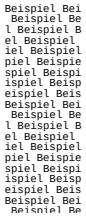
Berechnet den Sinus von *FORMAT-Ausdruck* im Bogenmaß.

SPACES FORMAT-Ausdruck

Es werden *FORMAT-Ausdruck* Leerzeichen ausgegeben (nur sinnvoll in Verbindung mit PRINT).

SQLCODE

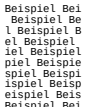
Der SQL-Code wird nach jeder Ausführung einer EXEC SQL-Anweisung gesetzt und kann in einem Report zu jedem Zeitpunkt im FORMAT-Abschnitt abgefragt werden. Ist der SQLCODE 0, so wurde die EXEC-SQL Anweisung fehlerfrei abgearbeitet. Ist er ungleich 0, so ist ein Fehler aufgetreten.



```
WHILE SQLCODE = 0 DO
BEGIN
        EXEC SQL FETCH mein_cursor
                IF SQLCODE = 100 THEN
                        PRINT “Keine Datensätze mehr vorhanden”
END
```

SQLROWS

Liefert die Anzahl der betroffenen Datensätze bei einer einfachen SQL-Anweisung (Also in der Form EXEC SQL “SQL-Anweisung”). Bei der SQL-Anweisung muß es sich um eine INSERT-, UPDATE- oder DELETE-Anweisung handeln.



```
EXEC SQL “UPDATE auftrag SET modified = 1 WHERE aufnr > 100”
PRINT SQLROWS, “ Sätze wurden modifiziert”
```

TIME

Ergibt die Uhrzeit als Zeichenkette im Format hh:mm:ss.
Die Variable zur Aufnahme der Zeit muß vom Datentyp CHAR sein.

TODAY

Datum des aktuellen Tages.

(Format-Ausdruck) **UNITS** *komponente*

Ergibt eine Zeitspanne mit der Komponente *komponente*, welcher der Wert *Format-Ausdruck* zugeordnet wird. Der Formatausdruck muß geklammert werden, um festlegen zu können, daß er zum UNITS Ausdruck gehört. Ansonsten könnte der Wert des folgenden Ausdrucks nicht korrekt ermittelt werden:

LET zs = zs + pro_zeit + tag UNITS DAY

Bedeutet dieser Ausdruck zs + pro_zeit + (tag) UNITS DAY oder zs + (pro_zeit + tag) UNITS DAY ? Erst die Klammerung macht den Ausdruck eindeutig.

Als Komponente sind die Werte YEAR, MONTH, DAY, HOUR, MINUTE, SECOND und FRACTION zulässig.

```
LET zs =  zs + ( pro_zeit * tages_faktor ) UNITS  DAY
```

FORMAT-Ausdruck1 **USING** FORMAT-Ausdruck2

Der Inhalt von *FORMAT-Ausdruck1* wird entsprechend dem *FORMAT-Ausdruck2* formatiert.
FORMAT-Ausdruck2 muß ein String, bzw. eine CHAR-Variable sein .

Formatstring	Aussehen des Datum an dieser Stelle:
dd	2-stellige Tageszahl
ddd	Tageskürzel (3 Buchstaben)
mm	2-stellige Monatszahl
mmm	Monatskürzel (3 Buchstaben)
yy	2-stellige Jahreszahl
yyyy	4-stellige Jahreszahl
zeichen	das unveränderte Zeichen (nur bei Datum)
$	$
(	(bei Zahlen kleiner 0
-	- bei Zahlen kleiner 0
+	+ bei Zahlen größer 0
)	) bei Zahlen kleiner 0
.	Dezimalpunkt; Ausrichtung der Zahl
,	die Vorkommastelle muß größer 0 sein
#	Ersetzung durch ein Leerzeichen
&	Ersetzung durch 0
*	*
<	linksbündige Ausgabe
%b	Monatsname abgekürzt (bei Datetime und Interval)
%B	Monatsname ausgeschrieben (bei Datetime und Interval)
%d	Tag des aktuellen Monats (01 - 31, bei Datetime und Interval)
%Fn	Wert der Sekundenbruchteile an n ter Stelle (bei Datetimeund Interval)
%H	Anzahl Stunden (24 Stunden, bei Datetime und Interval)
%I	Anzahl Stunden (12 Stunden, Datetime und Interval)
%M	Anzahl Minuten (Datetime und Interval)
%m	Anzahl Monate (Datetime und Interval)
%p	A.M. oder P.M (Datetime und Interval)
%S	Anzahl Sekunden (Datetime und Interval)
&y	2 stellige Jahreszahl (Datetime und Interval)
%Y	4 stellige Jahreszahl (Datetime und Interval)
%%	% Zeichen (Datetime und Interval)

Ist eine Zahl für ein angegebenes Format zu groß, so wird die Ausgabe mit * gefüllt, um den Überlauf zu kennzeichnen. Die Formatzeichen $, (, - oder + können mehrmals hintereinander angegeben werden. Das hat zur Folge, daß das Zeichen mit der Größe der Zahl mitbewegt wird.

Achtung:

Mit der Umgebungsvariablen REPDOT können Sie festlegen, ob bei Dezimalzahlen ein Punkt oder ein Komma verwendet werden soll.

WEEKDAY (FORMAT-Ausdruck)

Zahl zwischen 0 für Sonntag und 6 für Samstag. Der FORMAT-Ausdruck muß vom Datentyp DATE sein.

YEAR (FORMAT-Ausdruck)

Vierstellige Ausgabe der Jahreszahl.
Der FORMAT-Ausdruck muß vom Datentyp DATE sein.

4.4 Aggregate

Mittels der Aggregate können bestimmte arithmetische Operationen durchgeführt werden.

[**GROUP**]
{**COUNT** | { {**TOTAL** | {**AVERAGE** | **AVG**} | **MIN** | **MAX**} **OF** *spaltenname*}}

spaltenname	Name der Spalte.
GROUP	Das optionale Schlüsselwort GROUP legt fest, daß sich das nachfolgende Aggregat auf eine spezifizierte Gruppe bezieht. Daher muß der Ausdruck in der ORDER BY-Klausel in der SELECT-Anweisung aufgeführt sein. **Achtung:** GROUP darf nur in einem AFTER GROUP OF -Kontrollblock verwendet werden. Wird das Schlüsselwort GROUP nicht verwendet, so sind die Ausgaben nur in "ON LAST ..." gültig.
COUNT	Es wird die Anzahl der ausgelesenen Datensätze bestimmt.
TOTAL	Es wird die Summe der Werte der Spalte bestimmt.
AVG	Es wird der Durchschnitt der Spalte berechnet.
MIN	Es wird das Minimum der Spalte bestimmt.
MAX	Es wird das Maximum der Spalte bestimmt.

Achtung:

Wird ein Aggregat als Term in einer Berechnung verwendet, so muß das Aggregat geklammert werden.
Enthält eine Spalte in einem Satz einen NULL-Wert, so wird dieser ignoriert.

LET *summe* = **(TOTAL OF** *preis***) +** *fix_betrag*

13 Die Ausgabe von REP

1 PAGEMODE

REP macht die Ausgabe von Daten auf die Standardausgabe vom Ausgabemodus abhängig. Im Normalfall werden die Daten direkt auf die Standardausgabe geschrieben.
Ist jedoch der PAGEMODE aktiv, so wird die Ausgabe in einer virtuellen Seite zwischengepuffert. Weiterhin produziert der Report nur dann eine Ausgabe, wenn bei der Abarbeitung der Anweisungen eine ausgabeorientierte Anweisung durchlaufen wurde.Welches Ausgabeverfahren verwendet wird, entscheidet REP zur Compilezeit, wobei folgendes Kriterium entscheidend ist:

Beinhaltet der Report eine der Anweisungen
- SAVEYX oder
- CALLYX oder
- MOVEYX oder
- SET ATTRIBUTES

so wird im PAGEMODE gearbeitet, ansonsten nicht.

Die virtuelle Seite ist als eine im Hauptspeicher befindliche Abbildung des beschreibbaren Bereiches der tatsächlichen Seite zu verstehen. Dieser Bereich wird durch folgende Größen festgelegt:

RIGHT MARGIN - LEFT MARGIN

Diese Differenz bestimmt die Länge einer Zeile. Geht eine Ausgabe über die Zeilenlänge hinaus, versucht REP die Zeile zu splitten, d.h. es erfolgt eine mehrzeilige Ausgabe. Die über eine Zeile hinausgehende Ausgabe kann von nachfolgenden Ausgaben überschrieben werden (nur im PAGEMODE).

PAGE LENGTH - (TOP MARGIN + BOTTOM MARGIN)

Dieser Ausdruck ergibt die Seitenlänge (bedruckbarer Seitenabschnitt).
Ein Überschreiben der Seitenlänge impliziert einen Seitenwechsel.

Achtung:

Seitenwechsel[1] bedeutet, daß zu diesem Zeitpunkt der PAGE TRAILER gestartet wird. Wird jetzt im PAGE TRAILER die Seite überschrieben, so erfolgt, je nach Customisation Flag, eine Warnung oder ein Abbruch.

[FIRST] PAGE HEADER LENGTH, PAGE TRAILER LENGTH

Mit diesen Angaben legen Sie die Höhe des FIRST-PAGE-HEADERS, PAGE-HEADERS und des PAGE-TRAILERS fest (nur im PAGE MODE).

Beinhaltet eine der Ortsangabe eine Anweisung, die ein Newline zur Folge hat (print, skip,...), so wird diese zuerst ausgegeben. Die Differenz zwischen Höhenangabe und der aktuellen Zeile nach Abarbeitung der Anweisungen werden als Newlines ausgegeben. Geben Sie keine Höhen an, so errechnen sich die Höhen der einzelnen Ortsangaben aus der Anzahl zu erwartender Newlines in der jeweiligen Ortsangabe.

Achtung:

1. *Beispiel 3* auf Seite 125

Verwenden Sie in Ihrem Report MOVEYX, SAVEYX oder CALLYX, ist es empfehlenswert, die Höhen anzugeben, um eine korrekte Seitenaufteilung zu garantieren.

2 Virtuelle Seite

Sie ist wiederum unterteilt in [FIRST] PAGE HEADER, Datenblock und den PAGE TRAILER.

Achtung:

[FIRST] PAGE HEADER + PAGE TRAILER + Datenblock <=
PAGE LENGTH - (TOP MARGIN + BOTTOM MARGIN)

Der Start des TRAILER-Druckes errechnet sich wie folgt:
Start des PAGE TRAILERS =
PAGE LENGTH - (TOP MARGIN + BOTTOM MARGIN) - PAGE TRAILER LENGTH + 1

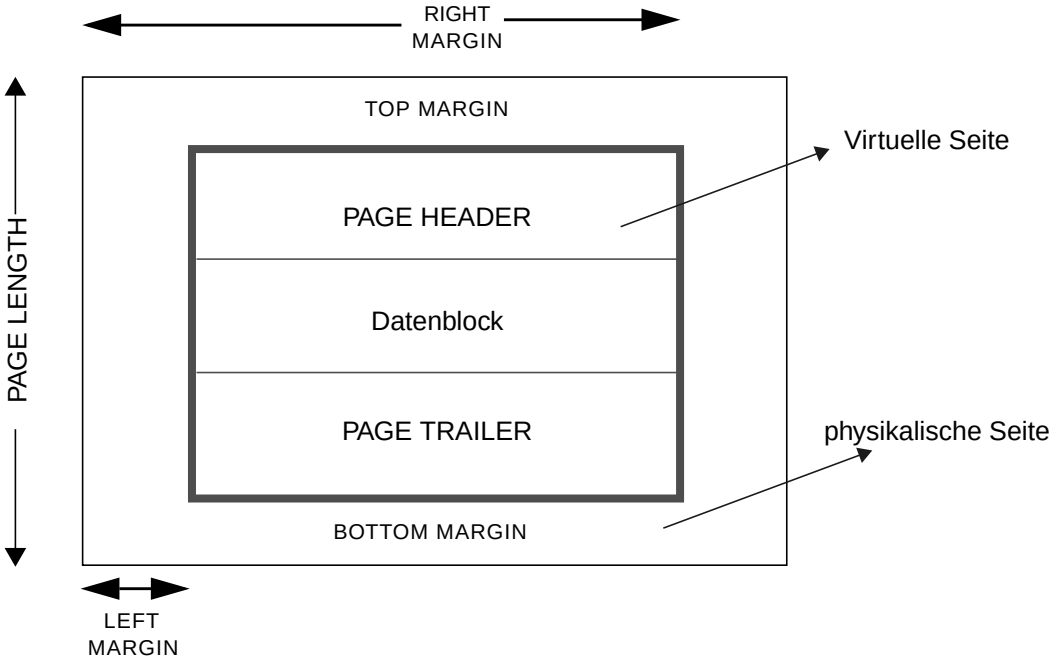


Abb.: Virtuelle Seite, bzw. beschreibbarer Bereich

```
{ ********** OUTPUT-ABSCHNITT ********** }
OUTPUT
        LEFT MARGIN             5
        RIGHT MARGIN            132
        TOP MARGIN              3
        BOTTOM MARGIN           3
        PAGE LENGTH             66
        PAGE HEADER LENGTH      5
        PAGE TRAILER LENGTH     10
END
```

Mit diesen Angaben wird folgendes Format für eine Seite festgelegt:

PAGE HEADER	beginnt in Zeile 1 und endet in Zeile 5	(5 Zeilen)
Datenblock	beginnt in Zeile 6 und endet in Zeile 50	(45 Zeilen)
PAGE TRAILER	beginnt in Zeile 51 und endet in Zeile 60	(10 Zeilen)

TOP MARGIN		3 Zeilen
BUTTOM MARGIN		3 Zeilen
PAGE LENGTH	=	66 Zeilen

3 Attribute

Im PAGEMODE ist es möglich, Zeichen mit Attributen zu versehen, um Schriftstile wie fett, kursiv oder unterstrichen auf dem Drucker auszugeben. Dazu ist in REP der Wert der aktuellen Attribute definiert, welcher aus einem (8 Bit langen) Bitvektor besteht, in dem jedes Attribut durch ein Bit definiert ist. Es stehen also 8 verschiedene Attribute zur Verfügung. Auf diese Weise können Schriftstile miteinander kombiniert werden, so daß z.B. auch die Ausgabe von kursivfetten Zeichen möglich ist. Werden Zeichen im PAGEMODE geschrieben, werden diese immer mit den aktuellen Attributen in der virtuellen Seite abgelegt. Bei der Ausgabe der Seite wird bei jeder Position verglichen, ob sich die Attribute von denen der vorherigen Position unterscheiden. Ist dies der Fall, werden durch Ausgabe einer Steuersequenz (Index 0) alle bisherigen Schriftstile abgeschaltet. Anschließend werden durch Ausgabe der Sequenzen der Attribute der aktuellen Position die Schriftstile dieses Zeichens eingeschaltet. Die auszugebenden Sequenzen werden entweder aus einer (selbst erstellten) Datei gelesen oder es werden die in folgender Tabelle aufgeführten Standardsequenzen verwendet:

Index	Bit Wert	Sequenz
0	0	^A
1	1	^B
2	2	^C
3	4	^D
4	8	^E

Diese Sequenzen können bei der Ausgabe durch einen Filter in die entsprechenden Steuerzeichen für Fettschrift, Großschrift, Grafikmode, u.s.w. für den jeweiligen Drucker umgesetzt werden. Dabei ist zu beachten, daß die Sequenz mit dem Index 0 (Defaultwert ^A) alle anderen Einstellungen zurücksetzen muß.

4 Definieren von eigenen Attributesequenzen

In einer Datei, deren vollständiger Name in der Umgebungsvariablen oder Resource REPATT definiert ist, können Attributte definiert werden. In jeder Zeile kann genau eine Definition festgelegt werden. Der Aufbau hat dabei die folgende Form:

<Index> <Sequenz>

<Index> bezeichnet das Bit innerhalb des Bitvektors. Begonnen wird dabei rechts mit Bit 1. Der erste Index wäre also 1. Trotzdem kann der Index 0 angegeben werden. Ihm wird die Initialisierungsequenz zugeordnet, welche alle anderen Attribute zurücksetzt. Diese Sequenz wird in folgenden Fällen ausgegeben:

- Am Anfang jeder Ausgabezeile

  Damit wird verhindert, daß der Rand mit dem aktuellen Attribut ausgegeben wird und so z.B. aus unterstrichenen Leerzeichen besteht. Nach der Ausgabe des Randes werden die aktuellen Attribute der Zeile wieder ausgegeben, um den vorherigen Zustand wieder herzustellen.

- Bei jedem Wechsel von Attributen

  Dadurch wird erreicht, daß keine bestehenden Attribute bei einem Wechsel erhalten bleiben

- Vor Ausgabe des unteren Randes

  Dies bewirkt, daß der obere Rand und der nachfolgende untere Rand der eventuell folgenden Seite mit dem Standardattribut ausgegeben werden.

Wird ein Index mehrfach in der Datei definiert, dann ist die letzte Definition gültig.

<Sequenz> bezeichnet die zu schreibende Sequenz, wenn ein Attribut mit diesem Index ausgegeben werden soll.

Beginnt eine Zeile in der ersten Spalte mit #, dann wird diese als Kommentarzeile überlesen.

5 Seitenwechsel

5.1 Das Problem des Seitenwechsels

Die Ausgaben eines Reports werden in den verschiedenen Ortsangaben implementiert, wobei zwischen 3 Positionen bezüglich einer Reportseite unterschieden werden kann:

Seitenkopf	In diesen Bereich werden die Ausgaben des FIRST PAGE HEADERS und des PAGE HEADERS getätigt.
Datenblock	Hier erscheinen die Ausgaben der Ortsangaben - ON EVERY ROW - ON LAST ROW - BEFORE GROUP OF <spalte> - AFTER GROUP OF <spalte>
Seitenfuß	Alle Ausgaben des PAGE TRAILERS werden in diesen Bereich geschrieben.

Um einen konsistenten Inhalt einer gedruckten Seite zu garantieren, ist es notwendig, daß die in der jeweiligen Ortsangabe verwendeten Daten in einem gemeinsamen Kontext stehen, d.h. der Inhalt von Seitenkopf und Seitenfuß muß zu dem Datenblock passen.

Ein weiteres Problem stellt der Zeitpunkt der Abarbeitung einer Anweisung in einer Ortsangabe dar. Dies betrifft sowohl die Ausgabe als auch die Semantik des Programmes.

REP macht ausschließlich einen impliziten Seitenwechsel (s.u.), um die oben beschriebene Konsistenz zu gewährleisten.

Satzwechsel

Unmittelbar nach dem Verlassen der ON EVERY ROW - Ortsangabe vollzieht sich der Satzwechsel.

Seitenwechsel

Ein Seitenwechsel beinhaltet sowohl die Ausgabe des PAGE TRAILERS als auch des PAGE HEADERS, wobei die Ausführung nicht zwingend nacheinander erfolgen muß. Bezogen auf die Anweisungen und Ortsangaben kann daher zwischen einem impliziten und expliziten Seitenwechsel unterschieden werden.

5.2 Impliziter Seitenwechsel

Eine impliziter Seitenwechsel liegt dann vor, wenn eine Anweisung die Ausführung des PAGE TRAILER und PAGE HEADER nicht vorsieht, sondern nur bei Bedarf die jeweilige Ortsangabe abarbeitet (PRINT). In diesen Fällen wird PAGE TRAILER und PAGE HEADER in der Regel zu nicht bestimmbaren zeitlich (örtlich) versetzten Zeitpunkten abgearbeitet.

Das Ereignis, welches zur Ausführung eines impliziten Seitenwechsels führt, soll den jeweiligen Ortsangaben und Anweisungen zugeordnet werden.

Ereignisse die die Ausführung des PAGE TRAILERS auslösen

PRINT	Ist die aktuelle Position die erste Zeile des PAGE TRAILERS, so wird der PAGE TRAILER ausgeführt.
SKIP	Ist nach der Ausgabe einer Leerzeile die aktuelle Position die erste Zeile des PAGE TRAILER, so wird der PAGE TRAILER ausgeführt.
SKIP TO TOP OF PAGE	Die Seite wird mit Leerzeilen aufgefüllt und der PAGE TRAILER ausgeführt.
NEED	Ist auf der aktuellen Seite nicht genügend Platz, so wird die Seite mit Leerzeilen aufgefüllt und der PAGE TRAILER ausgeführt.
PRINT FILE	Ist die aktuelle Position die erste Zeile des PAGE TRAILERS, so wird der PAGE TRAILER ausgeführt.
PRINTFORM	Ist zu wenig Platz auf der aktuellen Seite, wird der PAGE TRAILER ausgeführt. Ist nach der Ausgabe der Maske die aktuelle Zeile die erste Zeile des PAGE TRAILERS, wird der PAGE TRAILER ausgeführt.

Ereignisse die die Ausführung des PAGE HEADERS auslösen

Die Ausführung des PAGE HEADERS ist in REP an Anweisungen und Ortsangaben gebunden. Voraussetzung für die Ausführung des PAGE HEADERS ist, daß die letzten Ausgaben von dem PAGE TRAILER erfolgten.

FIRST PAGE HEADER	Existiert kein FIRST PAGE HEADER, wird der PAGE HEADER ausgeführt.
ON EVERY ROW	Bei Eintritt in die Ortsangabe vor Abarbeitung der ersten Anweisung wird der PAGE HEADER ausgeführt.
ON LAST ROW	Bei Eintritt in die Ortsangabe vor Abarbeitung der ersten Anweisung wird der PAGE HEADER ausgeführt.
BEFORE GROUP OF	Bei Eintritt in die Ortsangabe vor Abarbeitung der ersten Anweisung wird der PAGE HEADER ausgeführt.
AFTER GROUP OF	Bei Eintritt in die Ortsangabe vor Abarbeitung der ersten Anweisung wird der PAGE HEADER ausgeführt.
PRINT	Am Anfang vor der Abarbeitung einer PRINT-Anweisung wird der PAGE HEADER ausgeführt.

PRINT FILE	Nach dem Lesen eines Zeichens wird der PAGE HEADER ausgeführt.
PRINTFORM	Unmittelbar vor Abarbeitung der PRINTFORM-Anweisung wird der PAGE HEADER ausgeführt. Existiert nicht genügend Platz für die Maske, so wird die aktuelle Seite mit Leerzeilen aufgefüllt und vor der Ausgabe der Maske der PAGE TRAILER. Da PRINTFORM auf PRINT abgebildet ist, folgt bei der Ausgabe der Maske der PAGE HEADER.
SKIP	Geht die Anzahl der Newlines über eine Seite hinaus, so wird der PAGE TRAILER ausgeführt. Folgt nun eine weitere Leerzeile, wird der PAGE HEADER ausgeführt, ansonsten muß eines der obigen Ereignisse eintreffen.

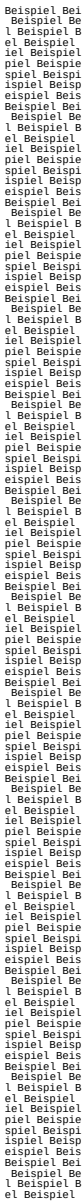
5.3 Expliziter Seitenwechsel

In diesem Fall werden PAGE TRAILER und PAGE HEADER unmittelbar nacheinander ausgeführt.
Um einen expliziten Seitenwechsel nachzubilden, stellt REP die FORMAT-Anweisung FLUSH HEADER zur Verfügung, welche die Abarbeitung des PAGE HEADERS veranlaßt, wenn der PAGE TRAILER ausgeführt wurde.

5.4 Beispiele

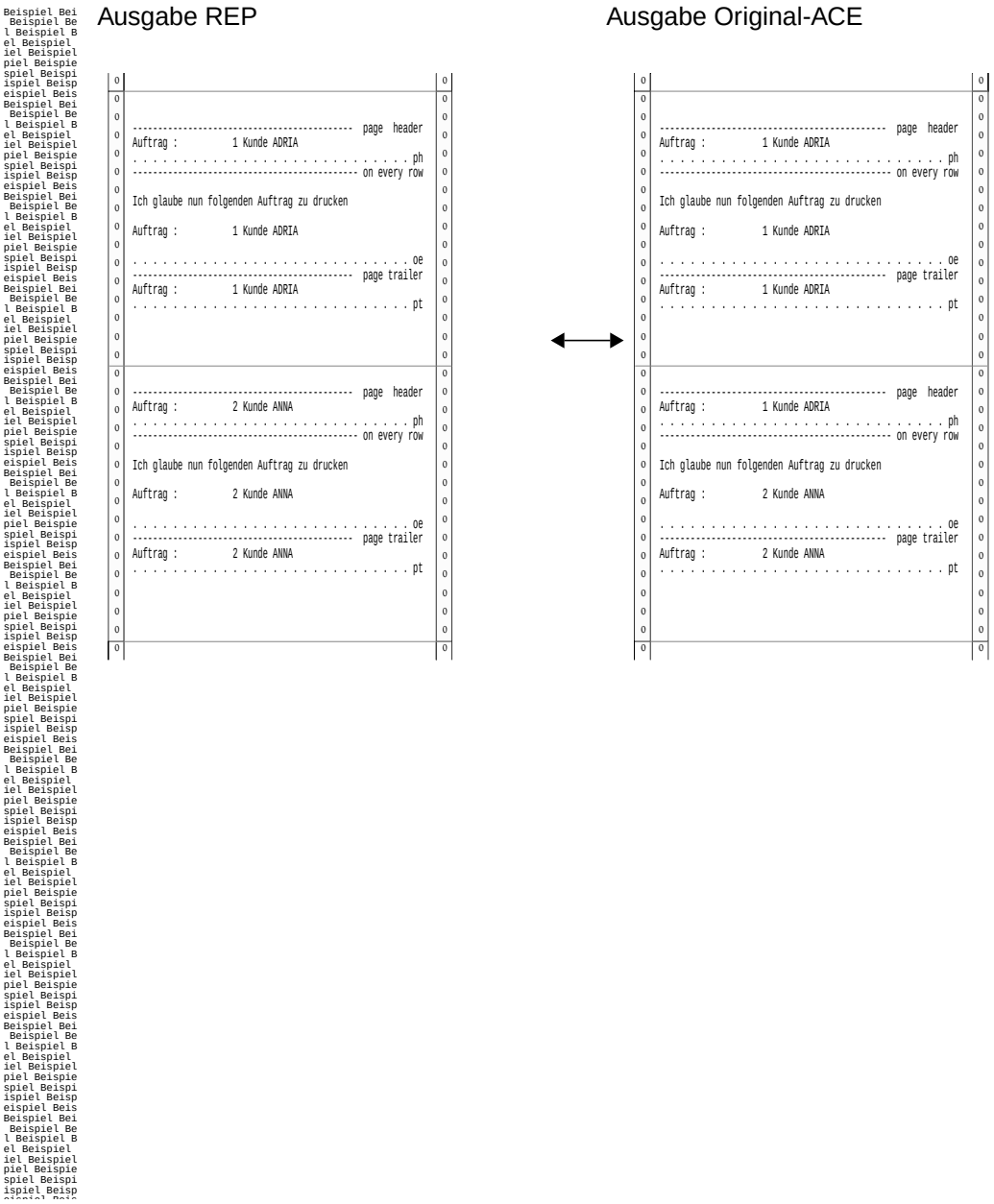
Satzwechsel und impliziter Seitenwechsel (Ausgabe)

Bei einem impliziten Seitenwechsel ist es möglich, daß zwischen PAGE TRAILER und PAGE HEADER ein gewollter Satzwechsel stattfinden kann.



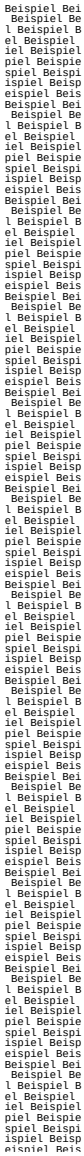
```
DATABASE
    test
END
DEFINE
    VARIABLE flag INTEGER
END
OUTPUT
    TOP MARGIN 5
    BOTTOM MARGIN 0
    PAGE LENGTH 18
END
SELECT
    * FROM auftr WHERE aufnr <= 2
END
FORMAT
    PAGE HEADER
        PRINT "----------------------------------------  page  header"
        PRINT "Auftrag : ",aufnr, " Kunde ",kucode
        PRINT ". . . . . . . . . . . . . . . . . . . . . . . . . . . . ph"
    ON EVERY ROW
        PRINT "---------------------------------------------- on every row"
        PRINT
        PRINT "Ich glaube nun folgenden Auftrag zu drucken"
        PRINT
        PRINT "Auftrag : ",aufnr, " Kunde ",kucode
        PRINT
        PRINT ". . . . . . . . . . . . . . . . . . . . . . . . . . . . oe"
    PAGE TRAILER
        PRINT "----------------------------------------  page trailer"
        PRINT "Auftrag : ",aufnr, " Kunde ",kucode
        PRINT ". . . . . . . . . . . . . . . . . . . . . . . . . . . . pt"
END
```

Da REP in seinem PAGE TRAILER mit dem im Datenblock (ON EVERY ROW) verwendeten Datensatz arbeitet und im PAGE HEADER bereits den Datensatz aktualisiert hat, ist eine Konsistenz zwischen Datenblock und Seitenfuß bzw. Seitenkopf und Datenblock gewährleistet.



Satzwechsel und impliziter Seitenwechsel (Anweisungen)

Ein impliziter Seitenwechsel kann durchaus dem Verständnis der Ablauflogik eines Reports abträglich sein. Am Beispiel einer LET-Anweisung sei dieses verdeutlicht.



```
DATABASE
    test
END
DEFINE
VARIABLE   flagph      CHAR(20)
VARIABLE   flagoer     CHAR(20)
END
OUTPUT
    TOP MARGIN 5
    BOTTOM MARGIN 0
    PAGE LENGTH 18
END
SELECT
    * FROM auftr WHERE aufnr < 2
END
FORMAT
    PAGE HEADER
        PRINT "----------------------------------------  page  header"
        PRINT "Auftrag : ",aufnr, " flagph  ", flagph
        PRINT ". . . . . . . . . . . . . . . . . . . . . . . . . . . ph"
        LET flagph = "set in Page header"
    ON EVERY ROW
        PRINT "---------------------------------------- on every row"
    SKIP TO TOP OF PAGE
    FLUSH HEADER
        LET flagph = "set in on every row"
        LET flagoer = flagph
        PRINT "Auftrag : ",aufnr, " flagph  ", flagph
        PRINT "Auftrag : ",aufnr, " flagoer ", flagoer
        PRINT
        PRINT ". . . . . . . . . . . . . . . . . . . . . . . . . . . oe"
    PAGE TRAILER
        PRINT "----------------------------------------  page trailer"
        PRINT "Auftrag : ",aufnr, " flagph  ", flagph
        PRINT ". . . . . . . . . . . . . . . . . . . . . . . . . . . pt"
END
```

```
Beispiel Bei
 Beispiel Be
l Beispiel B
el Beispiel
iel Beispiel
piel Beispie
spiel Beispi
ispiel Beisp
eispiel Beis
Beispiel Bei
 Beispiel Be
l Beispiel B
el Beispiel
iel Beispiel
piel Beispie
spiel Beispi
ispiel Beisp
eispiel Beis
Beispiel Bei
 Beispiel Be
l Beispiel B
el Beispiel
iel Beispiel
piel Beispie
spiel Beispi
ispiel Beisp
eispiel Beis
Beispiel Bei
 Beispiel Be
l Beispiel B
el Beispiel
iel Beispiel
piel Beispie
spiel Beispi
ispiel Beisp
eispiel Beis
Beispiel Bei
 Beispiel Be
l Beispiel B
el Beispiel
iel Beispiel
piel Beispie
spiel Beispi
ispiel Beisp
eispiel Beis
Beispiel Bei
 Beispiel Be
l Beispiel B
el Beispiel
iel Beispiel
piel Beispie
spiel Beispi
ispiel Beisp
eispiel Beis
Beispiel Bei
 Beispiel Be
l Beispiel B
el Beispiel
iel Beispiel
piel Beispie
spiel Beispi
ispiel Beisp
eispiel Beis
Beispiel Bei
 Beispiel Be
l Beispiel B
el Beispiel
iel Beispiel
piel spiel B
el Beispiel
iel Beispiel
piel Beispie
spiel Beispi
ispiel Beisp
eispiel Beis
Beispiel Bei
 Beispiel Be
l Beispiel B
el Beispiel
iel Beispiel
```

Ausgabe REP ohne FLUSH HEADER

```
-------------------------------------------  page  header
Auftrag :          1 flagph
. . . . . . . . . . . . . . . . . . . . . . . . . . . . ph
-------------------------------------------- on every row




-------------------------------------------  page trailer
Auftrag :          1 flagph  set in Page header
. . . . . . . . . . . . . . . . . . . . . . . . . . . . pt




-------------------------------------------  page  header
Auftrag :          1 flagph  set in on every row
. . . . . . . . . . . . . . . . . . . . . . . . . . . . ph
Auftrag :          1 flagph  set in Page header
Auftrag :          1 flagoer set in on every row

. . . . . . . . . . . . . . . . . . . . . . . . . . . . oe


-------------------------------------------  page trailer
Auftrag :          1 flagph  set in Page header
. . . . . . . . . . . . . . . . . . . . . . . . . . . . pt
```

Ausgabe REP mit FLUSH HEADER

```
-------------------------------------------  page  header
Auftrag :          1 flagph
. . . . . . . . . . . . . . . . . . . . . . . . . . . . ph
-------------------------------------------- on every row




-------------------------------------------  page trailer
Auftrag :          1 flagph  set in Page header
. . . . . . . . . . . . . . . . . . . . . . . . . . . . pt




-------------------------------------------  page  header
Auftrag :          1 flagph  set in Page header
. . . . . . . . . . . . . . . . . . . . . . . . . . . . ph
Auftrag :          1 flagph  set in on every row
Auftrag :          1 flagoer set in on every row

. . . . . . . . . . . . . . . . . . . . . . . . . . . . oe


-------------------------------------------  page trailer
Auftrag :          1 flagph  set in on every row
. . . . . . . . . . . . . . . . . . . . . . . . . . . . pt
```

14 Resourcen

Über Resourcen ist das Verhalten des Programms “repgo” einstellbar.

1 Auffinden der Resourcedatei

Alle Resourcen von REP werden aus dem Resourcefile `rep.rc` gelesen. Das Resourcefile wird nach folgendem Schema gesucht:

1. Wenn die Umgebungsvariable **FXHOME**[1] gesetzt ist, und dort die Datei `rep.rc` existiert, dann wird diese verwendet.
2. Wenn die Umgebungsvariable **FXDIR** gesetzt ist, und dort die Datei `rep.rc` vorhanden ist, dann wird diese Datei verwendet.
3. Sind beide Variablen nicht gesetzt, dann wird die Datei `rep.rc` im aktuellen Verzeichnis verwendet.

Zu beachten ist dabei, daß sobald eine Resourcedatei gefunden wurde, diese gelesen wird und die Suche abgebrochen wird. Sollen trotzdem mehrere Resourcedateien gelesen werden, dann kann der include Mechanismus verwendet werden. So ist z.B. folgende Vorgehensweise sinnvoll:

Im Verzeichnis $FXHOME wird eine Resourcedatei `rep.rc` angelegt, welche als erstes `$FXDIR/rep.rc` einschließt, um alle Resourcen auf Standardwerte zu setzen. Danach werden diejenigen Resourcen, welche umdefiniert werden sollen, in der Resourcedatei `$FXHOME/rep.rc` mit neuen Werten überschrieben.

Zusätzlich zum Lesen der Datei `rep.rc` besteht die Möglichkeit, eine weitere Resourcedatei einzulesen. Dies geschieht, wenn die Variable **RPRCFILE** auf eine gültige Resourcedatei gesetzt wird.

2 Aufbau der Resourcedatei

Eine Resourcedatei besteht aus mehreren Textzeilen. Eine Zeile kann dabei aus folgendem bestehen:

- eine Zuordnung von Resourcename zu Resourcewert mit optionalem Kommentar am Ende:

  <Resourcename>:<Wert> !Kommentar

- eine Kommentarzeile oder Leerzeile:

  ! Kommentar

- einer Anweisung zum Einschließen einer anderen Resourcedatei:

  #include <Datei>

1. Fall 1 und 2 gehen davon aus, daß REP zusammen mit FIX benutzt wird. Ist dies nicht der Fall, fehlen die entsprechenden Umgebungsvariablen und Fall 3 trifft zu.

2.1 Aufbau der Resourcezuweisung

Die Resourcezuweisung besteht aus einem Resourcenamen und einem Resourcewert. Der Resourcename besteht aus einem Identifier (Zeichenfolge von Buchstaben, Zahlen und Unterstrich, welche nicht mit einer Zahl beginnt) und einem optionalen Programmnamen am Anfang, welchem ein Punkt folgt. Dieser optionale Programmname wird von REP jedoch nicht unterstützt.

Beispiele:

`Resource1`	- gültiger Resourcename
`5te_Resource`	- ungültiger Resourcename
`Prog1.Res1`	- Resource `Res1` für Programm `Prog1`

Der Resourcewert besteht aus beliebigen Zeichen, wobei einige jedoch eine Sonderbedeutung haben. Weiterhin werden führende und folgende Leerzeichen entfernt. Folgende Zeichen haben eine Sonderbedeutung:

$	Zugriff auf andere Resourcewerte
\	Einleiten von Ersatzdarstellung

Zugriff auf andere Resourcewerte

Mittels des Konstrukts ‘$(<Resourcewert>)’ kann auf andere Resourcewerte zugegriffen werden. Dieser Resourcewert muß jedoch in der Resourcedatei vor dem Zugriff definiert sein. Beispiel:

Hauptverzeichnis:	/home/user
Unterverzeichnis:	$(Hauptverzeichnis)/subdir

Die Resource ‘`Unterverzeichnis`’ erhält den Wert ‘`/home/user/subdir`’.

Ersatzdarstellung

Mittels der Ersatzdarstellung können Zeichen im Resourcewert eingefügt werden, welche sonst nicht zulässig sind:

- \ooo definiert ein Zeichen durch seinen Oktalwert
- \xhh definiert ein Zeichen durch seinen Hexadezimalwert

So läßt sich z.B. mit dem Resourcewert ‘10 \x24’ ein ‘$’ in den Resourcewert einfügen, ohne daß dieses als Zugriff auf einen anderen Resourcewert interpretiert wird.

Maximalwerte

Für die Resourcedatei sind folgende Maximallängen definiert:

- Länge einer Zeile: 512
- Länge eines Resourcenamens: 255
- Länge eines Resourcewertes nach der Ersetzung anderer Resourcewerte einschließlich aller Zeichen zur Ersatzdarstellung: 255
- Länge einer Pfadangabe: 255

Einschließen von anderen Resourcedateien

Mittels der Anweisung ‘include <Datei>’ kann eine weitere Resourcedatei eingeschlossen werden. Der Wert ‘<Datei>’ muß einen gültigen Dateinamen enthalten. Er darf aber auch Zugriffe auf Resourcewerte enthalten.

Beispiel:

```
HOME:        /user/home
include $(HOME)/my.rc
```

liest die Datei '`/home/user/my.rc`' und setzt die dort angegebenen Resourcen.

3 Zugriff auf Resourcen

"repgo" greift auf bestimmte Resourcen zu. Der Zugriff auf eigene Resourcen kann mittels der REP Funktion GETRES analog zur Funktion GETENV erfolgen. Beim Zugriff wird folgende Suchreihenfolge verwendet:

1. existiert eine Umgebungsvariable mit dem Namen der Resource, wird deren Wert zurückgeliefert.
2. ist in der Resourcedatei eine Resource mit '<Resourcename>' definiert, dann wird deren Wert zurückgeliefert.
3. ansonsten wird ein Leerstring zurückgeliefert.

Mit diesem Verfahren ist es möglich, Resourcewerte durch Umgebungsvariablen zu überschreiben.

4 Resourcen für REP

Folgende Resourcen werden von "repgo" gelesen:

Resourcename	Bedeutung
REPATT	Name einer Datei, welche Attribute definiert.
REPMSG	Name und Pfad der REP Messagedatei
REPDOT	kann auf '.' oder ',' gesetzt werden und bestimmt damit das Aussehen des Dezimalpunktes
REPFMTDATE	Date Format zur Umwandlung eines Date Wertes in einen String (Default: dd.mm.yyyy)
REPDEFMTDATE	Date Format zur Umwandlung eines Strings in einen Date Wert (Default: dd.mm.yyyy)
ILOGINInfxServer	Alle mit ILOGIN beginnenden Resourcen belegen eine Componente
ILOGINDbDate	in der InetLogin Struktur, welche unter Windows zum Verbindungs-
ILOGINDbTime	aufbau mit der Datenbank genutzt wird. Der Name der Komonente
ILOGINDbLang	ergibt sich, indem der Präfix ILOGIN weggelassen wird.
ILOGINHost	Näheres zu dieser Struktur ist in der Informixdokumentation
ILOGINService	beschrieben.
ILOGINProtocol	
ILOGINClient_Loc	
ILOGINDB_Loc	
INFORMIXDIR	besetzt die Komponente InformixDir der InetLogin Struktur.

Folgende Resourcen werden von repgo definiert und können in einem REP Skript gelesen werden:

Resourcename	Bedeutung
rep_database	verwendete Datenbank (nur Informix)
rep_pagelength	Länge der Seite in Zeilen
rep_leftmargin	linker Rand
rep_rightmargin	rechter Rand
rep_topmargin	oberer Rand
rep_bottommargin	unterer Rand
rep_pageheaderlength	Länge des Page Headers
rep_fpagehaderlength	Länge des ersten Page Headers
rep_pagetrailerlength	Länge des Page Trailers
rep_rolname	Name der Roldatei (Parmeter -v oder -S)
rep_devmode	Parameter zur Druckersteuerung (Parameter -D)
rep_outfile	Name der Ausgabedatei (Parameter -f)
rep_user	Name des Benutzer (Parameter -U)

15 Anhang

1 Beschränkungen und Längen

Die folgende Tabelle gibt eine Übersicht über in REP vorhandene Beschränkungen und Längen.

Bezeichnung	Wert	Beschreibung
MAXSTRSIZE	32767	Maximale Länge einer Zeichenkette
MAXSTMTSIZE	8192	Maximale Länge eines SQL Statements
MAXTEMPTABLE	20	Maximale Anzahl verschiedener Temp Tables
MAX_KANAL	32	Maximale Anzahl offener Dateien
MAXVAR	10000	Maximale Anzahl Variablen
MAX_SEL	100	Maximale Anzahl Select Anweisungen
MAX_SYMLEN	18	Maximale Länge von Bezeichnern

2 Beispiele

Alle Beispiele basieren auf der Datenbank repdemo. Diese hat folgenden Aufbau:

repdemo

- *auftrag* Tabelle Auftrag
  Felder:
  - *kundna* Kundenname
  - *kundnr* Kundennummer
  - *auftrtxt* Beschreibung des Auftrages
  - *auftrnr* Auftragsnummer

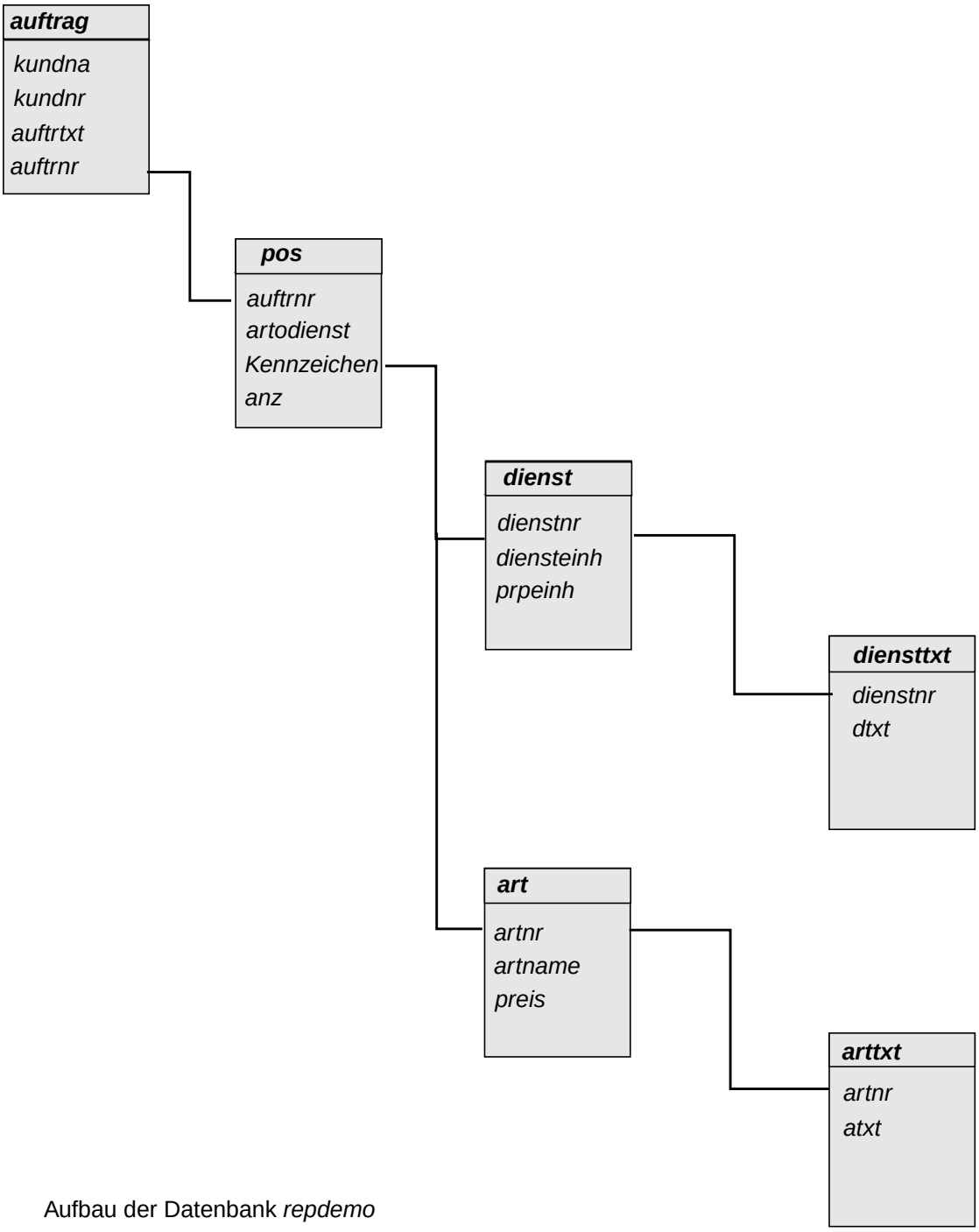
- *pos* Tabelle Positionen
  Felder:
  - *auftrnr* Auftragsnummer
  - *artodienst* A -> Artikel, D -> Dienstleistung
  - *Kennzeichen* Ist die Position ein Artikel (A), so repräsentiert das Kennzeichen die entsprechende Artikelnummer. Handelt es sich um eine Dienstleistung (D), dann steht das Kennzeichen für die Dienstleistungsnummer.
  - *anz* Anzahl der erworbenen A/D's

- *dienst* Tabelle Dienstleistungen
  Felder:
  - *dienstnr* Dienstleistungsnummer
  - *diensteinh* Dienstleistungseinheit (qm, std, ...)
  - *prpeinh* Preis pro Einheit

- *art* Tabelle Artikel
  Felder:
  - *artnr* Artikelnummer
  - *artname* Artikelname
  - *preis* Artikelpreis

- *diensttxt* Tabelle Dienstleistungserläuterungen
  Felder:
  - *dienstnr* Dienstleistungsnummer
  - *dtxt* Text

- *arttxt* Tabelle Artikelerläuterungen
  Felder:
  - *artnr* Artikelnummer
  - *atxt* Text



Aufbau der Datenbank *repdemo*

2.1 Beipiel 1

```
{           REPort-Listing      : beispiel1.rep
            Autor               : NSI
            Datum               : Tue, 14 Apr 1992

            Es werden alle Aufträge gedruckt. Anschließend soll eine Liste aller
            Artikel und letztendlich eine Liste aller Dienstleistungen gedruckt werden.
 }

{ ******************************** DATABASE-ABSCHNITT ******************************** }

    DATABASE repdemo                                    {_Datenbank-Name_}
    END


{ ********************************* DEFINE-ABSCHNITT ********************************* }

    DEFINE                                              {_Variablen-Deklarationen_}
        VARIABLE   head_txt {_Namen_}      CHAR ( 40 {_Größe_} )
        VARIABLE   trail_txt {_Namen_}     CHAR ( 40 {_Größe_} )
        VARIABLE   datei_jn {_Namen_}      CHAR ( 30 {_Größe_} )
        VARIABLE   datei_name {_Namen_}    CHAR ( 28 {_Größe_} )
        VARIABLE   txt_jn {_Namen_}        CHAR (1 {_Größe_} )

        VARIABLE   auftr_anz {_Variable_}  INTEGER
        VARIABLE   art_anz {_Variable_}    INTEGER
        VARIABLE   pr_head {_Variable_}    INTEGER

        VARIABLE   datei_zeiger            FILE
        VARIABLE   dat_out {_Variable_}    INTEGER
        VARIABLE   true {_Variable_}       INTEGER
    END
```

```
{ ********************************** INIT-ABSCHNITT ***********************************
}

    INIT                                                    {_Variablen-Deklarationen_}
        { Die Variable head_txt wird mit dem String “Liste aller” initialisiert. }
        SET head_txt      = “Liste aller”
        SET pr_head       = 1
        SET true          = 1
    END


{ ********************************** INPUT-ABSCHNITT**********************************
}

    INPUT
        PROMPT FOR datei_jn
        USING “ Aufträge in eine Datei (J dateiname oder N)?”
    END


{ ********************************** OUTPUT-ABSCHNITT ********************************** }

    OUTPUT
        LEFT MARGIN          5          {_Rand-Größe-links_}
        RIGHT MARGIN        80          {_Rand-Größe-rechts_}
        TOP MARGIN           3          {_Rand-Größe-oben_}
        BOTTOM MARGIN        3          {_Rand-Größe-unten_}
        PAGE LENGTH         25          {_Seiten-Größe_}
    END
```

{ **1.SELECT-FORMAT Abschnitt**

In diesem Abschnitt werden zum einen die Sätze aus der Tabelle auftrag formatiert ausgegeben, zum anderen wird die im INPUT Abschnitt erfragte

Datei geöffnet. Sie ist in jedem der Format-Abschnitte beschreibbar.

Im allgemeinen gilt, daß jede im DEFINE-Abschnitt deklarierte Variable in jedem SELECT-FORMAT-Abschnitt verwendet werden kann, wobei sich die Gültigkeit eines selektierten Feldes auf einen SELECT-FORMAT-Abschnitt beschränkt.

}

{ ********** **SELECT-ABSCHNITT** ********** }

SELECT * {_Spalten-Auswahl_}

FROM *auftrag* {_Tabellen_}

END

{ ********** **FORMAT-ABSCHNITT** ********** }

FORMAT {_Format-Anweisungen_}

FIRST PAGE HEADER

IF (*datei_jn*[1] = ‘j’ OR *datei_jn*[1] = ‘J’) THEN

BEGIN

{

Falls die Antwort “j” ist, wird der Name der Datei in ‘datei_jn’ beginnend an der 3.Position erwartet. Nachstehende Blanks werden gelöscht (CLIPPED)

}

LET *dat_out* = *true*

LET *datei_name* = *datei_jn*[3,30] CLIPPED

{

Wenn kein Dateiname eingegeben wurde, dann beendet das Programm mit Errorcode 2.

Dieser Code kann von dem aufrufenden Programm ausgewertet werden.

}

IF *datei_name* = “” THEN

EXIT(2)

Beispiel Bei
Beispiel Be
l Beispiel B
el Beispiel
iel Beispiel
piel Beispie
spiel Beispi
ispiel Beisp
eispiel Beis
Beispiel Bei

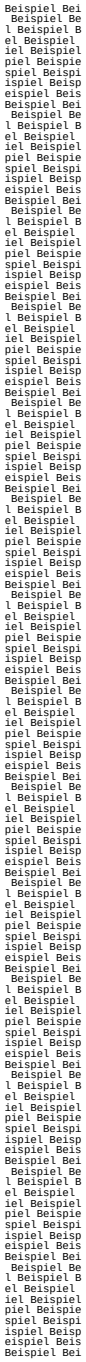
{
Öffnen der Datei “datei_name”. Das ‘w’ bewirkt, daß sie im Schreibmodus geöffnet wird. Von nun an kann die Datei über den Dateizeiger “datei_zeiger” angesprochen werden.
}
LET *datei_zeiger* = **OPEN**(*datei_name*,”w”)

{
Das Layout der Datei kann mit dem SET PAGE-Kommando festgelegt werden.
Es wird folgendes Layout festgelegt:

Oberer Rand:	: 5 Zeilen
Unterer Rand:	: 5 Zeilen
Linker Rand:	: 10 Zeichen
Rechter Rand:	: 70 Zeichen
Seitenlänge:	: 60 Zeilen

}
SET PAGE 5, 5, 10, 70, 60, 1 FOR *datei_zeiger*
END

PRINT *head_txt*
CLIPPED ,” Aufträge”
SKIP 1 LINE
PRINT
COLUMN 1 , “Kundenname”,
COLUMN 20 , “Kundennummer”,
COLUMN 35 , “Auftragsnummer”,
COLUMN 50 , “Eingang”,
COLUMN 64 , “Ende”
SKIP 2 LINE



PAGE HEADER

{

Am Anfang einer jeden Seite soll der “head_txt” gefolgt von einer jeweiligen Spezifikation stehen (Aufträge).

}

PRINT *head_txt*

CLIPPED ,” Aufträge”

SKIP 1 LINE

PRINT

COLUMN 1 , “Kundenname”,

COLUMN 20 , “Kundennummer”,

COLUMN 35 , “Auftragsnummer”,

COLUMN 50 , “Eingang”,

COLUMN 64 , “Ende”

SKIP 2 LINE

ON EVERY ROW

{

Für jeden gelesenen Satz werden die nachfolgenden Anweisungen ausgeführt.

}

PRINT

COLUMN 1 , *kundna* CLIPPED ,

COLUMN 20 , *kundnr* USING “&&&&&&”,

COLUMN 35 , *auftrnr* USING “####”,

COLUMN 50 , *auftreing* USING “dd.mm.yyyy”;

{

Wenn eine Datei geöffnet wurde, dann gebe die zu einem Auftrag gehörenden Daten aus.

}

IF dat_out = true THEN

PRINT

COLUMN 1 , *kundna* CLIPPED ,

COLUMN 20 , *kundnr* USING “&&&&&&”,

COLUMN 35 , *auftrnr* USING “####”,

COLUMN 50 , *auftreing* USING “dd.mm.yyyy” TO *datei_zeiger*

```
                {
                Wenn in das Feld auftrend noch kein Wert eingetragen ist, dann
                gebe einen Strich aus, ansonsten gebe den entsprechenden Wert
                aus.
                }
                IF (auftrend IS NULL) THEN
                BEGIN
                        PRINT
                                COLUMN 64 ,"----------"
                        IF dat_out = true THEN
                        PRINT
                                COLUMN 64 ,"----------" TO datei_zeiger
                END ELSE
                BEGIN
                        PRINT
                                COLUMN 64 , auftrend USING "dd.mm.yyyy"
                        IF dat_out = true THEN
                        PRINT
                        COLUMN 64 , auftrend USING "dd.mm.yyyy" TO
datei_zeiger
                END

        PAGE TRAILER
                PRINT "----------", PAGENO USING "&&&#", "----------"
        ON LAST ROW
                {
                Wurde eine Datei geöffnet, dann schließe die Datei
                }
                IF dat_out = true THEN
                        CLOSE(datei_zeiger)
                {
                Die Anzahl der selektierten Aufträge werden nach auftr_anz
                geschrieben
                }
                LET auftr_anz = COUNT
    END
```

{ **2.SELECT-FORMAT Abschnitt**

In diesem Abschnitt werden alle Artikel aus der Tabelle *art* formatiert und ausgegeben.

}

{ ********** **SELECT-ABSCHNITT** ********** }

```
SELECT *                {_Spalten-Auswahl_}
    FROM art            {_Tabellen_}
END
```

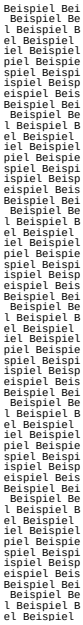
{ ********** **FORMAT-ABSCHNITT** ********** }

```
FORMAT {_Format-Anweisungen_}
    PAGE HEADER
        PRINT head_txt CLIPPED," Artikel" {_Anweisungen_}
        SKIP 1 LINE
            PRINT
                COLUMN 1 ,"Artikelnr",
                COLUMN 20 ,"Artikelname",
                COLUMN 40 ,"Preis"
            SKIP 2 LINES
    ON EVERY ROW
        PRINT
            COLUMN 1 ,artnr USING "&&&&&",
            COLUMN 20 ,artname CLIPPED,
            COLUMN 40 ,preis_stk USING "#&&.&&"
    PAGE TRAILER
        PRINT "----------", PAGENO USING "&#", "----------"
    ON LAST ROW
        LET art_anz = COUNT
END
```

{ **3.SELECT-FORMAT Abschnitt**

Letztendlich werden die Dienstleistungen aus der Tabelle dienst ausgelesen und formatiert ausgegeben. Nachdem der letzte Satz bearbeitet wurde, soll eine Statistik zusammengestellt werden, welche Resultate der vorherigen SELECT-FORMAT-Teile zusammenfaßt.
}

```
{ ********************************* 3. SELECT-ABSCHNITT ********************************* }
    SELECT *                {_Spalten-Auswahl_}
        FROM dienst         {_Tabellen_}
    END
{ ********************************* FORMAT-ABSCHNITT ********************************* }
    FORMAT                  {_Format-Anweisungen_}
        PAGE HEADER
            IF pr_head = 1 THEN
            BEGIN
                PRINT head_txt CLIPPED," Dienstleistungen" {_Anweisungen_}
                SKIP 1 LINE
                PRINT
                    COLUMN 1 ,"Dienstnr",
                    COLUMN 20 ,"Dienstbez.",
                    COLUMN 40 ,"Preis"
                SKIP 2 LINES
            END
            ELSE
            BEGIN
                PRINT "Statistik"
                SKIP 4 LINES
            END
        ON EVERY ROW
            PRINT
                COLUMN 1 ,dienstnr USING "&&&&&",
                COLUMN 20 ,diensteinh CLIPPED,
                COLUMN 40 ,preis_einh USING "#&&.&&"
        PAGE TRAILER
            PRINT "----------", PAGENO USING "&#", "----------"
        ON LAST ROW
```



```
{
Auf der letzten Seite soll im PAGE HEADER “Statistik” stehen. Anschließend
werden Anzahl der Aufträge, Artikel und Dienstleistungen aufgeführt
}
LET pr_head = 0
SKIP TO TOP OF PAGE
        PRINT “Es sind momentan “;
        PRINT auftr_anz USING “&&&&&”;
        PRINT “ Aufträge registriert. “
        PRINT “Die Produktpallette erstreckt sich über “, art_anz
                USING “&&&&&”, “ Artikel.”
        PRINT
        “Die Anzahl der angebotenen Dienstleistungen summieren sich auf “,
                COUNT USING “&&&&&”,”.”
END
```

Die Ausgabe des Programms finden Sie auf den nächsten Seiten.

Ausgabe Beispiel 1

```
Liste aller Aufträge

Kundenname          Kundennummer   Auftragsnummer Eingang       Ende

Schreiner Peter     100001         1111           13.11.1991    ----------
Schmidt Maria       100002         1112           13.03.1992    04.04.1992
Fink Arnold         100003         1113           13.04.1992    ----------




Seite -01-
```

```
Liste aller Artikel

Artikelnr          Artikelname         Preis

00100              Kabel (3 adrig)      19.00

00101              Kabel (5 adrig)      32.00
00103              Sicherung (16 Amp)   08.52
00104              Sicherung (11 Amp)   09.35
00105              Kippschalter         06.32
00106              Dimmer               49.12
00107              Verteilerdose (5)    00.32
00102              Sich.Kasten         223.12


Seite -02-
```

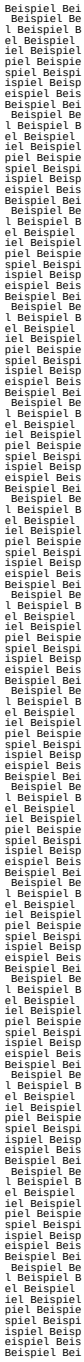
```
Liste aller Dienstleistungen

Dienstnr            Dienstbez.          Preis

00200               std                  63.50
00210               std                  36.08
00220               std                  22.12

Seite -03-
```

```
Statistik

Es sind momentan 00003 Aufträge registriert.
Die Produktpallette erstreckt sich über 00008 Artikel.
Die Anzahl der angebotenen Dienstleistungen summieren  sich auf 00003.



Seite -04-
```

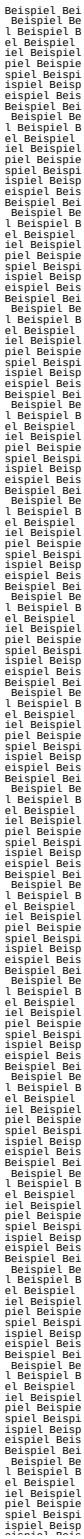
2.2 Beispiel 2

```
{       REPort-Listing      : beispiel2.rep
        Autor               : NSI
        Datum               : Mon, 27 Apr 1992

    Dieser Report liest alle Aufträge und die dazugehörigen Positionen.
    Je nach Positionsart (Artikel oder Dienstleistung) werden im Formatteil die
    entsprechenden Daten selektiert.
 }

{ ****************************** DATABASE-ABSCHNITT ******************************** }

    DATABASE
        repdemo
    END


{ ****************************** DEFINE-ABSCHNITT ********************************** }

    DEFINE
        CURSOR    cs_dienst              { Cursor dienst-Tabelle }
        CURSOR    cs_art                 { Cursor art-Tabelle }
        VARIABLE  poszaehler INTEGER
    END


{ ****************************** OUTPUT-ABSCHNITT ********************************** }

    OUTPUT
        PAGE LENGTH 40
        RIGHT MARGIN 80
        LEFT MARGIN 0
        TOP MARGIN 5
        BOTTOM MARGIN 5
    END
```

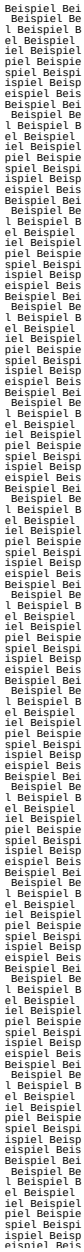
```
{ ********** SELECT-ABSCHNITT ********** }
    {
    Haupt-Select: An dieser Stelle sollen alle erfaßten Aufträge
    und die dazugehörigen Daten selektiert werden.
    }

    SELECT auftrag.auftrnr aufnr,pos.auftrnr,kundna,kundnr, auftrtxt,
           auftreing, auftrend, artodienst, kenn, anz
        FROM auftrag, pos
        WHERE auftrag.auftrnr = pos.auftrnr
        ORDER BY aufnr
    END


{ ********** FORMAT-ABSCHNITT ********** }

    FORMAT
        FIRST PAGE HEADER
            {
            Nun werden alle Cursor deklariert, die in dem Report verwendet
            werden. Für die Tabellen art und dienst wird jeweils ein Cursor
            deklariert.
            }

            { Ist eine Position eine Dienstleistung, so werden entspr. Daten
                best. }

            EXEC SQL DECLARE cs_dienst CURSOR
            FOR "SELECT diensteinh, preis_einh FROM dienst WHERE
dienstnr=:kenn"

            { Ist eine Position ein Artikel, so werden entspr. Daten best. }

            EXEC SQL DECLARE cs_art CURSOR
            FOR "SELECT artname, preis_stk FROM art WHERE artnr=:kenn"
```



```
BEFORE GROUP OF aufnr
    SKIP TO TOP OF PAGE
    PRINT '----------------------------------------';
    PRINT '-------------------------------------'
    PRINT '                          A U F T R A G'
    PRINT '----------------------------------------';
    PRINT '-------------------------------------'
    PRINT 'Auftr-Nr : ',auftrnr
    PRINT 'Kunden-Name: ',kundna,
        COLUMN 30, 'Kunden-Nr :',kundnr USING "##########"
    PRINT
    PRINT "Erbrachte Leistung : ",auftrtxt
    PRINT
    PRINT '----------------------------------------';
    PRINT '-------------------------------------'
    PRINT 'Nr',
        COLUMN 5,'PosArt',
        COLUMN 20,'Nummer';
    PRINT
        COLUMN 27,'Einheit',
        COLUMN 50,'Einzelpreis',
        COLUMN 65,"Anzahl",
        COLUMN 70,"   Preis"
    PRINT '----------------------------------------';
    PRINT '-------------------------------------'
    PRINT
    LET  poszaehler = 0
ON EVERY ROW
    LET  poszaehler = poszaehler + 1
    PRINT poszaehler USING "##";
```



```
{
Handelt es sich bei dem Posten um eine Dienstleistung,
        dann
                Öffne den Cursor auf die Tabelle dienst
                Lese entsprechende Daten aus
                Schliesse den Cursor
}
IF artodienst = 'D' THEN
BEGIN
        PRINT
                COLUMN 5,'Dienstleistung';
                EXEC SQL OPEN  cs_dienst    { Daten über Dienstleis
                EXEC SQL FETCH cs_dienst    { Da 1:1 Bez., nur ein S
        {
        Tritt bei dem Lesen eines Satzes ein Fehler auf, so
        soll abgebrochen werden.
        }
        IF SQLCODE <> 0 THEN
        BEGIN
                PRINT
                PRINT "Fehler beim Lesen aus der Tabelle dienst"
                PRINT "SqlCode :",SQLCODE USING "###"
                EXIT(SQLCODE)
        END
        PRINT
                COLUMN 20,kenn USING "&&&&&";
        PRINT
                COLUMN 27,diensteinh;
        PRINT
                COLUMN 50,preis_einh USING "######.##";
        PRINT
                COLUMN 65, anz USING "####",
                COLUMN 70, preis_einh * anz USING "######.##"
EXEC SQL CLOSE cs_dienst
END
```



```
        {
        Handelt es sich bei dem Posten um einen Artikel,
                dann
                        Öffne den Cursor auf die Tabelle art
                        Lese entsprechende Daten aus
                        Schliesse den Cursor
        }
        IF artodienst  = 'A' THEN
        BEGIN
                PRINT
                        COLUMN 5,'Artikel' ;
                EXEC SQL OPEN  cs_art        { Daten über Artikel }
                EXEC SQL FETCH cs_art        { Da 1:1 Bez., nur ein Satz}
                IF SQLCODE <> 0 THEN
                BEGIN
                        PRINT "Fehler beim Lesen aus der Tabelle art"
                EXIT(SQLCODE)
                END
                PRINT
                        COLUMN 20, kenn USING "&&&&&";
                PRINT
                        COLUMN 27, artname;
                PRINT
                        COLUMN 50, preis_stk USING "######.##";
                PRINT
                        COLUMN 65, anz USING "####",
                        COLUMN 70, preis_stk * anz USING "######.##"
                EXEC SQL CLOSE cs_art
        END
        PRINT
    ON LAST ROW
        PRINT'Stand der Aufträge vom ', TODAY USING "dd.mm.yyyy"
END
```

Die Ausgabe des Programms finden Sie auf den nächsten Seiten.

2.3 Ausgabe Beispiel 2

```
--------------------------------------------------------------------------------
                                A U F T R A G
--------------------------------------------------------------------------------
Auftr-Nr :       1111
Kunden-Name: Schreiner Peter              Kunden-Nr :   100001

Erbrachte Leistung : Rohbau-Installationen

--------------------------------------------------------------------------------
Nr  PosArt         Nummer Einheit               Einzelpreis    Anzahl   Preis
--------------------------------------------------------------------------------

 1  Artikel        00100  Kabel (3 adrig)           19.00       123   2337.00
 2  Artikel        00101  Kabel (5 adrig)           32.00        52   1664.00
 3  Artikel        00103  Sicherung (16 Amp)         8.52        48    408.96
 4  Artikel        00104  Sicherung (11 Amp)         9.35        32    299.20
 5  Artikel        00107  Verteilerdose (5)           .32       158     50.56
 6  Dienstleistung 00100  std                       63.50       222  14097.00
 7  Dienstleistung 00101  std                       36.08       480  17318.40
 8  Dienstleistung 00102  std                       22.12       300   6636.00
```

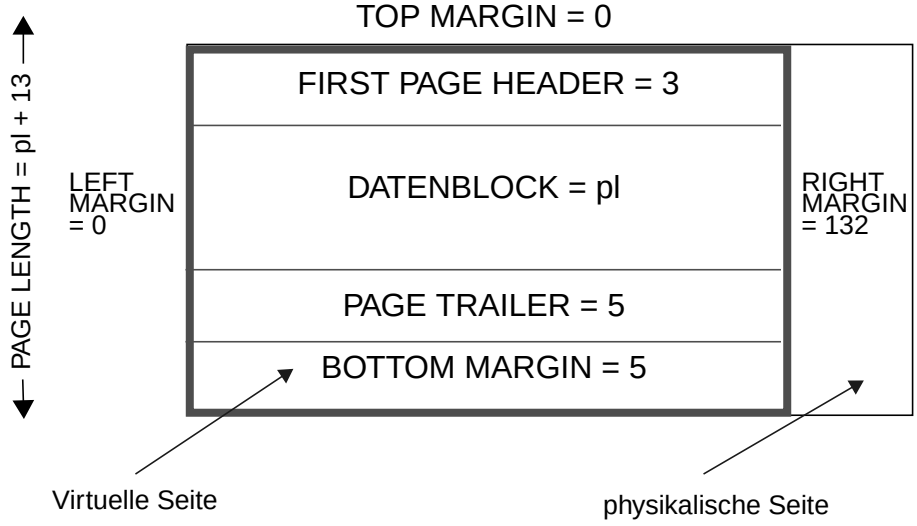
```
--------------------------------------------------------------------------------
                              A U F T R A G
--------------------------------------------------------------------------------
Auftr-Nr :        1112
Kunden-Name: Schmidt Maria                Kunden-Nr :   100002

Erbrachte Leistung : Verlegungen el. Leitungen u. verschiedene Installationen

--------------------------------------------------------------------------------
Nr  PosArt         Nummer Einheit               Einzelpreis    Anzahl   Preis
--------------------------------------------------------------------------------
 1  Artikel        00100  Kabel (3 adrig)            19.00        12    228.00
 2  Artikel        00101  Kabel (5 adrig)            32.00         3     96.00
 3  Dienstleistung 00102  std                        22.12        98   2167.76
```

```
--------------------------------------------------------------------------------
                                A U F T R A G
--------------------------------------------------------------------------------
Auftr-Nr :       1113
Kunden-Name: Fink Arnold                  Kunden-Nr :   100003

Erbrachte Leistung : Einbau eines neuen Sicherungskasten u.  Erneuerung
                     verbrauchter Teile
--------------------------------------------------------------------------------

Nr  PosArt         Nummer Einheit               Einzelpreis    Anzahl   Preis
--------------------------------------------------------------------------------
 1  Artikel        00103  Sicherung (16 Amp)          8.52        12    102.24
 2  Artikel        00104  Sicherung (11 Amp)          9.35         7     65.45
 3  Artikel        00102  Sich.Kasten               223.12         1    223.12
 4  Dienstleistung 00102  std                        22.12        98   2167.76

Stand der Aufträge vom 04.05.1992
```

2.4 Beispiel 3

```
{       REPort-Listing          : beispiel3.rep
        Autor                   : NSI
        Datum                   : Mon, 13 Jul 1992
    Dieser Report liest alle Aufträge und die dazugehörigen Positionen.
    Anschließend wird eine Tabelle erzeugt, die für jeden Auftrag (eine Spalte)
    die dazugehörigen Positionen auflistet.
}
{ ******************************** DATABASE-ABSCHNITT ******************************** }
    DATABASE
        repdemo                 { Deklaration der verwendeten Datenbank }
    END
{ ******************************** DEFINE-ABSCHNITT ******************************** }
    DEFINE                              { Variablendeklaration }
                                        { die aktuelle Spalte für einen Auftrag ... }
        VARIABLE col_pos_auf   INTEGER  {_Variable_}
                                        { ... die aktuelle Spalte für eine Position ...
        VARIABLE col_pos_posit INTEGER  {_Variable_}
                                        { ... die aktuelle Zeile }
        VARIABLE lin_pos       INTEGER  {_Variable_}
        VARIABLE cnt_auf       INTEGER
        VARIABLE pl            INTEGER  {_Variable_}
        VARIABLE max_pos       INTEGER  {_Variable_}
        VARIABLE true          INTEGER  {_Variable_}
        VARIABLE false         INTEGER  {_Variable_}
        VARIABLE f             INTEGER  {_Variable_}
        VARIABLE i             INTEGER  {_Variable_}
    END
{ ******************************** SELECT-ABSCHNITT ******************************** }
    {Haupt-Select: An dieser Stelle sollen alle erfaßten Aufträge und
     die dazugehörigen Positionen selektiert werden.}
    SELECT auftrag.auftrnr aufnr, pos.auftrnr,artodienst, kenn { Spaltenauswahl }
        FROM auftrag, pos                       { Tabellebangaben }
        WHERE auftrag.auftrnr = pos.auftrnr     { Bedingungen f. die zu
                                                  selektierenden Daten }
        ORDER BY aufnr                          { sortiert nach Auftragsnummer }
    END
```

```
{ ********************************* FORMAT-ABSCHNITT ********************************* }
FORMAT
        {
          In dem INIT-Abschnitt werden die einzelnen Variablen
          und der Ausgabekanal initialisiert.
        }
        INIT
                LET true = 1
                LET false = 0
                LET f = true
                LET col_pos_auf = 25
                LET lin_pos = 3
                LET cnt_auf = 1
                        { Die Seitenlaenge ist die Maximallaenge einer Spalte }
                EXEC SQL "SELECT COUNT(*) n FROM pos GROUP BY\
                auftrnr INTO TEMP tmp"
                EXEC SQL "SELECT MAX(n) INTO :max_pos FROM tmp"
                LET pl = max_pos + 5 + 3 + 5            { bm fph pt }
                SET PAGE 0,5,0,132,pl,1,3,3,5  { Parameter für virtuelle Seite }
                {
                  REP baut eine virtuelle Seite mit folgenden Größen auf:
```



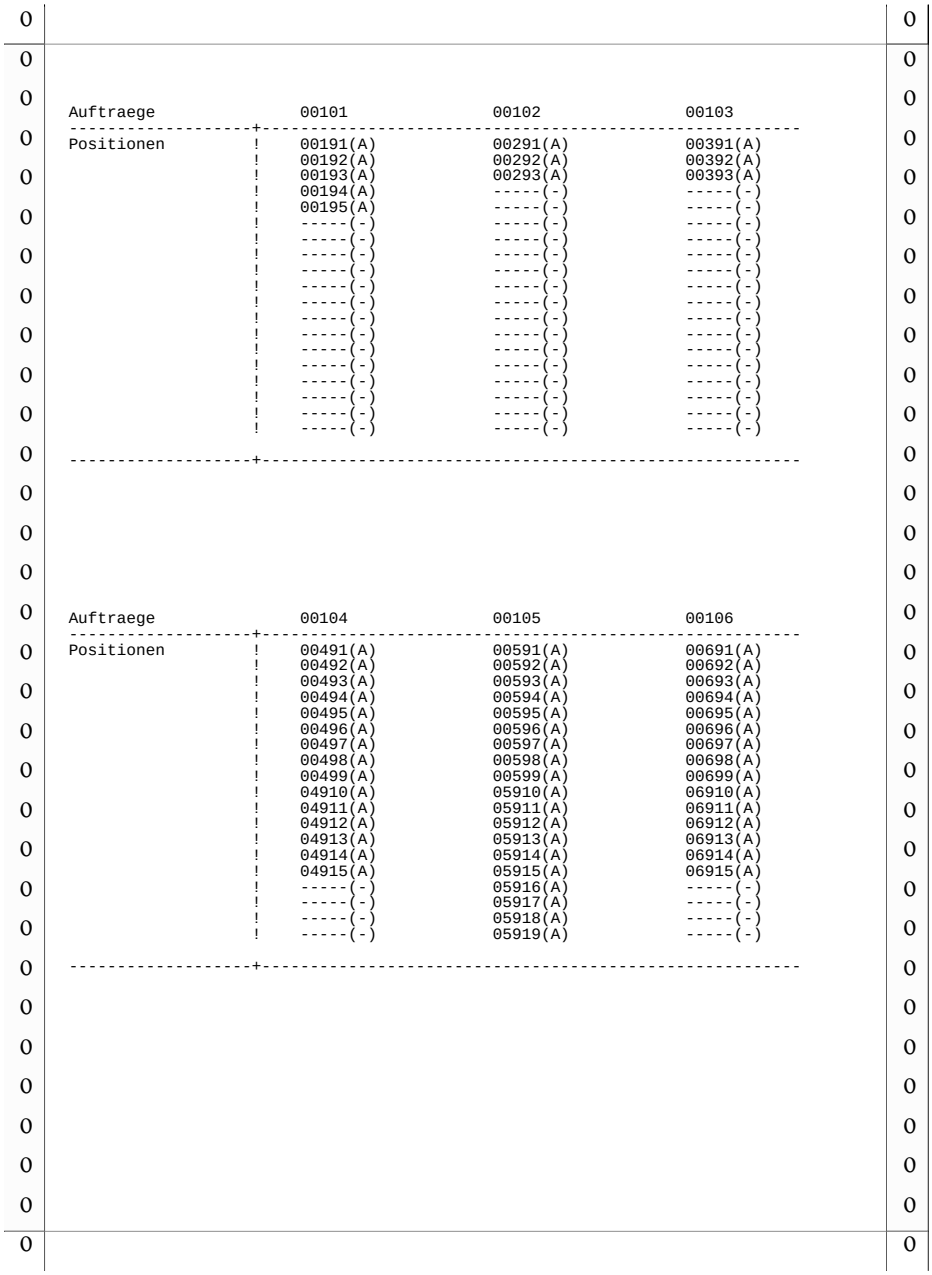

```
                  Der Spaltenindex ist in dem Intervall [ i | 1 <= i <= 132 ],
                  der Zeilenindex [ j | 1 <= j <= pl +13 ] zulässig.
                }
```

```
FIRST PAGE HEADER          { die FORMAT-Anweisungen werden am
                             Anfang der ersten Seite ausgeführt }
    PRINT "Auftraege"
    PRINT “-------------------+”;
    PRINT “------------------------------------------------------------”
    PRINT "Positionen"
BEFORE GROUP OF aufnr      { die Format-Anweisungen werden
                             am Reportanfang und vor jedem
                             Gruppenwechsel ausgeführt }
    { In diesem Teil des Reports werden die Spalten-Überschriften
    erzeugt und die Variablen zur Positionierung für die spaltenweise
    Ausgabe der jeweils zu einem Auftrag gehörenden Positionen (hier
    auf der virtuellen Seite) initialisiert.}
    IF f = false THEN
        FOR i = lin_pos TO (max_pos + 2) STEP 1 DO
            BEGIN
                MOVEYX i,20
                PRINT “!”
                MOVEYX i,col_pos_posit
                PRINT “-----(-)”
            END
        IF ( cnt_auf = 3 {_Bedingung_}) THEN
            BEGIN
                LET cnt_auf = 1
                LET col_pos_auf = 25
                LET col_pos_posit = col_pos_auf
                SKIP TO TOP OF PAGE {_Anweisung_}
                MOVEYX 1,1
                PRINT “Auftraege”
                PRINT “-------------------+”;
                PRINT “------------------------------------------------------
-”

                PRINT “Positionen”
            END
        ELSE
            IF (f = false) THEN
                LET cnt_auf = cnt_auf + 1
            ELSE
                LET f = false
```

```
                MOVEYX 1 col_pos_auf          { Positionieren Zeile/ Auftragsspalte }
                PRINT aufnr USING "&&&&&";          { Auftragsnummer }
                LET col_pos_posit = col_pos_auf     { Pos.spalte neu initialisie-
ren }
                LET lin_pos = 3                     { wieder auf 3. Zeile positionieren }
                {
                  Mit der Funktion COLNO kann die
                  aktuelle Spaltenposition abgefragt werden.
                }
                LET col_pos_auf = COLNO + 15        { Auftragsspalte aktualisie-
ren }
        ON EVERY ROW
        { Die FORMAT-Anweisungen werden immer dann durchlaufen, wenn ein
          neuer Datensatz gelesen wurde.
          An dieser Stelle erfolgt die Ausgabe der jeweiligen Positionen.
        Ein Gruppenwechsel bewirkt, daß die Zeilenvariable neu initialisiert wird.
        }
                MOVEYX lin_pos 20             { Positionieren Zeile/ Auftragsspalte }
                PRINT "|"
                MOVEYX lin_pos col_pos_posit
                { Positionieren Zeile/Pos.spalte }
                PRINT kenn USING "&&&&&", "(", artodienst, ")"
                LET lin_pos = lin_pos + 1           { nächste Zeile }

        { Die letzten Spalten werden mit dem Empty-String gefuellt. }
        ON LAST ROW
                FOR i = lin_pos TO (max_pos + 2) DO
                        BEGIN
                                MOVEYX i,20
                                PRINT "|"
                                MOVEYX i,col_pos_posit
                                PRINT "-----(-)"
                        END
        PAGE TRAILER
                PRINT "-------------------+";
                PRINT "-----------------------------------------------------------"
                { PRINT "Stand vom ",TODAY }
END
```

Die Ausgabe des Programms finden Sie auf der nächsten Seite.

2.5 Ausgabe Beispiel3



Auftraege	00101	00102	00103
Positionen	00191(A)	00291(A)	00391(A)
	00192(A)	00292(A)	00392(A)
	00193(A)	00293(A)	00393(A)
	00194(A)	-----(-)	-----(-)
	00195(A)	-----(-)	-----(-)
	-----(-)	-----(-)	-----(-)
	-----(-)	-----(-)	-----(-)
	-----(-)	-----(-)	-----(-)
	-----(-)	-----(-)	-----(-)
	-----(-)	-----(-)	-----(-)
	-----(-)	-----(-)	-----(-)
	-----(-)	-----(-)	-----(-)
	-----(-)	-----(-)	-----(-)
	-----(-)	-----(-)	-----(-)
	-----(-)	-----(-)	-----(-)
	-----(-)	-----(-)	-----(-)

Auftraege	00104	00105	00106
Positionen	00491(A)	00591(A)	00691(A)
	00492(A)	00592(A)	00692(A)
	00493(A)	00593(A)	00693(A)
	00494(A)	00594(A)	00694(A)
	00495(A)	00595(A)	00695(A)
	00496(A)	00596(A)	00696(A)
	00497(A)	00597(A)	00697(A)
	00498(A)	00598(A)	00698(A)
	00499(A)	00599(A)	00699(A)
	04910(A)	05910(A)	06910(A)
	04911(A)	05911(A)	06911(A)
	04912(A)	05912(A)	06912(A)
	04913(A)	05913(A)	06913(A)
	04914(A)	05914(A)	06914(A)
	04915(A)	05915(A)	06915(A)
	-----(-)	05916(A)	-----(-)
	-----(-)	05917(A)	-----(-)
	-----(-)	05918(A)	-----(-)
	-----(-)	05919(A)	-----(-)

Auftraege	00107	00108	00109
Positionen	00791(A)	00891(A)	00991(A)
	00792(A)	00892(A)	00992(A)
	00793(A)	00893(A)	00993(A)
	00794(A)	-----(-)	00994(A)
	00795(A)	-----(-)	00995(A)
	00796(A)	-----(-)	00996(A)
	00797(A)	-----(-)	00997(A)
	00798(A)	-----(-)	00998(A)
	00799(A)	-----(-)	00999(A)
	07910(A)	-----(-)	09910(A)
	07911(A)	-----(-)	09911(A)
	-----(-)	-----(-)	09912(A)
	-----(-)	-----(-)	09913(A)
	-----(-)	-----(-)	09914(A)
	-----(-)	-----(-)	09915(A)
	-----(-)	-----(-)	-----(-)
	-----(-)	-----(-)	-----(-)
	-----(-)	-----(-)	-----(-)
	-----(-)	-----(-)	-----(-)

Auftraege	00110	00111	00112
Positionen	01091(A)	01191(A)	01291(A)
	01092(A)	01192(A)	01292(A)
	01093(A)	01193(A)	01293(A)
	01094(A)	01194(A)	01294(A)
	01095(A)	01195(A)	01295(A)
	01096(A)	01196(A)	01296(A)
	01097(A)	01197(A)	01297(A)
	-----(-)	-----(-)	01298(A)
	-----(-)	-----(-)	01299(A)
	-----(-)	-----(-)	12910(A)
	-----(-)	-----(-)	12911(A)
	-----(-)	-----(-)	12912(A)
	-----(-)	-----(-)	12913(A)
	-----(-)	-----(-)	12914(A)
	-----(-)	-----(-)	12915(A)
	-----(-)	-----(-)	12916(A)
	-----(-)	-----(-)	12917(A)
	-----(-)	-----(-)	12918(A)
	-----(-)	-----(-)	12919(A)

2.6. Fehlermeldungen

Im folgenden Text werden die Fehlermeldungen von “repprep” und “repgo” angegeben. Zu jeder Fehlermeldung wird Fehlernummer, Programm und Fehlertext ausgegeben. Wenn die Fehlermeldung nur in “repprep” auftritt, wird für “Programm” “repprep” angegeben, wenn sie nur in “repgo” auftritt, wird für “Programm” “repgo” angegeben. Wenn Fehlermeldung bei beiden Programmen auftritt, wird für “Programm” nichts angegeben.

Nach der Angabe einer Fehlermeldung kann eine Erklärung mit Angabe einer möglichen Maßnahme folgen, markiert durch ▼, oder ein bzw. mehrere Verweise auf Stellen im Manual, markiert durch ☞.

FehlerNr		Fehlermeldung
11001	repprep	Die Aufrufparameter für repprep sind nicht korrekt.
☞		Seite 13 Übersetzen mit repprep
11003		Die Datei <*dateiname*> kann nicht im Schreibmodus geöffnet werden.
☞	repprep	Seite 13 Übersetzen mit repprep
☞	repgo	Seite 13 Starten mit repgo
11004	repgo	Eine Pipe nach <*programm*> kann nicht geöffnet werden.
☞		Seite 31 OUTPUT-Abschnitt : REPORT TO PIPE “*programm*”
11005	repprep	Die Zwischencodedatei <*dateiname*> *w*ürde die Source-Datei überschreiben.
▼		Der Name der Source-Datei ist länger als 14 Zeichen
11007		Die Datei <*dateiname*> kann nicht im Lesemodus geöffnet werden.
▼	repprep	Die Source-Datei kann nicht gelesen werden
▼	repgo	Die Zwischencode-Datei kann nicht gelesen werden
11008	repgo	Die repgo-Version <*versionsnummer*> kann die Zwischencodedatei <*dateiname*> nicht interpretieren.
▼		Übersetzen Sie Ihren Report nochmals mit dem repprep.
11009	repgo	Die repgo-Version <*versionsnummer*> *s*timmt nicht mit der Version in <*zwischencodedateiname versionsnummer*> überein.
▼		Übersetzen Sie Ihren Report nochmals mit dem rep
11010	repprep	Die Zeichenkette <*zeichenkette*> ist zu lang (maschinenabhängig).
11015	repprep	Eine String-Konstante ist nicht mit einem Hochkomma abgeschlossen.
11020	repprep	Die Stringkonstante <*stringkonstante*> ist zu lang (maschinenabhängig).
11025	repprep	Ein Kommentar ist nicht mit einer schließenden Klammer } abgeschlossen.
11055		Die Variable <*variablenname*> ist nicht definiert.
☞		Seite 21 DEFINE-Abschnitt
11056		Die Variable <*variablenname*> ist bereits definiert.
☞		Seite 21 VARIABLE
11065	repprep	Der Cursor <*cursorname*> ist nicht definiert.
☞		Seite 21 DEFINE-Abschnitt
11066	repprep	Der Cursor <*cursorname*> *i*st bereits definiert.
☞		Seite 22 CURSOR
11080	repprep	Der Datentyp des Parameters ist nicht zulässig.
☞		Seite 23 Datentypen

11085 repprep Der Qualifier ist nicht zulässig.
▼ Der angegebene Qualifier zu einem Datetime oder Interval ist unzulässig. Möglicherweise wurde (bei Interval) die Tag/Monat Grenze überschritten oder der Startwert ist kleiner als der Endwert (z.B. DAY TO YEAR statt YEAR TO DAY)
☞ Seite 23 Datentypen

11080 repprep Die Funktion <*name*> ist nicht portabel.
▼ Dies ist nur eine Warnung, welche besagt, daß eine Funktion verwendet wurde, welche nicht bei allen Datenbanken vorhanden ist.

11100 repprep Der DATABASE-Abschnitt ist syntaktisch nicht korrekt.
☞ Seite 19 DATABASE-Abschnitt

11110 repprep Der DEFINE-Abschnitt ist syntaktisch nicht korrekt.
☞ Seite 21 DEFINE-Abschnitt

11150 repprep Der INIT-Abschnitt ist syntaktisch nicht korrekt.
☞ Seite 25 INIT-Abschnitt

11155 repprep Der RESOURCE-Abschnitt ist syntaktisch nicht korrekt
☞ Seite 27 Resource-Abschnitt

11156 repgo Ungueltiger Datetime-Qualifier(%d) - Default wird benutzt.
▼ Der für die Resource *OraDateQualifier* angegebene Wert ist ungültig.

11160 repprep Der INPUT-Abschnitt ist syntaktisch nicht korrekt.
☞ Seite 29 INPUT-Abschnitt

11200 repprep Der OUTPUT-Abschnitt ist syntaktisch nicht korrekt.
☞ Seite 31 OUTPUT-Abschnitt

11210 repprep Die Zeilengröße <*zeilengröße*> *i*n dem OUTPUT-ABSCHNITT ist nicht sinnvoll.
☞ Seite 32 Randgrößen

11213 repgo Die Zeilengröße <*spalten/zeile*> *i*n der SET PAGE-Anweisung ist nicht sinnvoll.
☞ Seite 63 Format-Anweisungen im einzelnen SET PAGE

11220 repprep Die Seitengröße <*seitengröße*> in dem OUTPUT-Abschnitt ist nicht sinnvoll.
☞ Seite 32 Randgrößen

11223 repgo Die Seitengröße <*seitengröße*> in der SET PAGE-Anweisung ist nicht sinnvoll.
☞ Seite 63 Format-Anweisungen im einzelnen SET PAGE

11230 repprep Der Datenbereich <*größe*> *i*n dem OUTPUT-Abschnitt ist nicht sinnvoll.
☞ Seite 31 OUTPUT-Abschnitt

11233 repgo Der Datenbereich <*größe*> *i*n der SET PAGE-Anweisung ist nicht sinnvoll.
☞ Seite 63 Format-Anweisungen im einzelnen SET PAGE

11260 repgo Die Layout-Beschreibung ist nicht zu verwenden.
☞ Seite 63 Format-Anweisungen im einzelnen SET PAGE

11400 repprep Der SELECT-Abschnitt ist syntaktisch nicht korrekt.
☞ Seite 35 SELECT-Abschnitt

11410 repprep Das SUB-SELECT ist syntaktisch nicht korrekt.
☞ Seite 41 Subqueries

11420 repprep Die Spaltenauswahl ist syntaktisch nicht korrekt.

☞ Seite 36 Spaltenauswahl

11430 repprep Die Tabellenauswahl ist syntaktisch nicht korrekt.
☞ Seite 37 Tabellenangabe

11440 repprep Die Spaltenauswahl in der GROUP BY-Klausel ist nicht korrekt.
☞ Seite 36 SELECT-Anweisung GROUP BY

11450 repprep Die Spaltenauswahl in der ORDER BY Klausel ist nicht korrekt.
☞ Seite 36 SELECT-Anweisung ORDER BY

11460 repprep Die INTO TEMP-Klausel ist nicht korrekt.
☞ Seite 36 SELECT-Anweisung INTO TEMP

11470 repprep An dieser Stelle ist ausschließlich "TODAY" zulässig.
☞ Seite 44 Funktionen

11480 repprep Der SELECT-Ausdruck ist nicht zulässig.
☞ Seite 43 SELECT-Ausdruck

11490 repprep Der SELECT-Ausdruck darf in der ORDER BY-Klausel nicht verwendet werden.
☞ Seite 43 SELECT-Ausdruck
☞ Seite 35 SELECT-Anweisung

11495 repprep Die SELECT-Bedingung ist syntaktisch nicht korrekt.
☞ Seite 39 SELECT-Bedingung

11499 repprep An dieser Stelle sollte ein “*” stehen.
☞ Seite 37 Tabellenangabe

11500 repgo Das ‘matches’ Suchmuster ‘[..]’ kann nicht umgesetzt werden
▼ Für dieses Suchmuster gibt es bei der Suche mit **like** keine Entsprechung.

11560 repprep ENDNOCHECK fehlt
☞ Seite 47 NOCHECK Anweisungen

11570 repprep NOCHECK Teil fehlerhaft
☞ Seite 47 NOCHECK Anweisungen

11600 repprep Der FORMAT-Abschnitt ist syntaktisch nicht korrekt.
☞ Seite 49 FORMAT-Abschnitt

11620 repprep Die Anweisung ist syntaktisch nicht korrekt.
☞ Seite 53 FORMAT-Anweisungen

11650 repprep Der Ausdruck ist in diesem Kontext nicht korrekt.
☞ Seite 68 FORMAT-Ausdruck

11700 repprep Die Anweisung ist nicht/nur in der Ortsangabe INIT zulässig.

11705 repprep Die Anweisung <*Anweisung*> ist im Visual Modus unzulaessig
▼ Diese Anweisung darf nicht zusammen mit Visual REP verwendet werden.

11706 repprep Die Anweisung <*Anweisung*> ist nur im Visual Modus zulaessig
▼ Diese Anweisung darf nur zusammen mit Visual REP verwendet werden.

11710 repprep SKIP TO TOP OF PAGE ist in dieser Ortsangabe nicht zulässig.
☞ Seite 53 FORMAT-Anweisungen

11750 repprep Das Aggregat darf nicht außerhalb der Ortsangabe AFTER GROUP verwendet werden.
☞ Seite 84 Aggregate
☞ Seite 49 FORMAT-Abschnitt AFTER GROUP

11760 repprep Das Aggregat darf nicht außerhalb der Ortsangabe ON LAST ROW verwendet werden.
☞ Seite 84 Aggregate
☞ Seite 53 Ortsangaben im einzelnen ON LAST ROW

11920 repprep In diesem Kontext sollte das Schlüsselwort <*schlüsselwort*> stehen.
☞ Seite 49 FORMAT-Abschnitt

19991 repprep Die Option <*optionsangabe*> ist in der Form nicht korrekt oder unbekannt.
☞ Seite 13 Starten mit repgo

20000 repgo Ein NULL-Wert als Argument ist unzulässig.
☞ Seite 29 INPUT-Abschnitt FOR
☞ Seite 80 Funktionen OPEN
☞ Seite 81 Funktionen READ
☞ Seite 82 Funktionen USING
☞ Seite 57 Format-Anweisungen im einzelnen FOR ... TO .. STEP
☞ Seite 54 Format-Anweisungen im einzelnen EXEC SQL

20003 repgo Die Variable <*variablenname*> ist in diesem Kontext unzulässig.
▼ Die Variable hat für eine Anweisung nicht den korrekten Datentyp (z.B. FILE)

20005 repgo Das Argument ist unzulässig.
☞ Seite 53 FORMAT-Anweisungen

20006 repgo Es wurde ein Argument erwartet.
☞ Kap. 2 Seite 53

20007 repgo Es ist nur ein ganzzahliger Wert zulässig (*variablenname*).
☞ Seite 66 Operatoren Bool'sche Operatoren
☞ Seite 76 Funktionen ASCII
☞ Seite 55 Format-Anweisungen im einzelnen EXIT
☞ Seite 58 Format-Anweisungen im einzelnen LET
☞ Seite 63 Format-Anweisungen im einzelnen SET PAGE
☞ Seite 65 Format-Anweisungen im einzelnen SKIP
☞ Seite 81 Funktionen SPACES
☞ Seite 78 Funktionen COLUMN
☞ Seite 80 Funktionen MDY
☞ Seite 59 Format-Anweisungen im einzelnen MOVEYX

20008 repgo Es ist nur eine Variable vom Datentyp DATE zulässig.
☞ Seite 44 Funktionen
☞ Seite 23 Datentypen

20009 repgo Es wurde eine Datei-Variable erwartet.
▼ Es wurde ein Dateizeiger erwartet

20011 repgo Es wurde ein Dateiname erwartet.
☞ Seite 80 Funktionen OPEN

20013 repgo Die Datei <*dateiname*> kann nicht im Modus r/w/a geöffnet werden.
☞ Seite 80 Funktionen OPEN
☞ Seite 80 Funktionen PRINT

20014 repgo Die Datei wurde im falschen Modus r/w/a geöffnet.
☞ Seite 80 Funktionen OPEN

☞ Seite 63 Format-Anweisungen im einzelnen SET PAGE

20015 repgo Die Datei <*dateiname*> ist nicht geöffnet.
☞ Seite 54 Format-Anweisungen im einzelnen CLOSE
☞ Seite 54 Format-Anweisungen im einzelnen EOF
☞ Seite 81 Funktionen READ
☞ Seite 80 Funktionen OPEN

20017 repgo Die Variable <*variablenname*> bzw. der Parameter <*parametername*> muß vom Typ CHAR sein.
▼ Der Datentyp der Variablen stimmt nicht mit dem von der Anweisung geforderten Datentyp überein.
☞ Kap. 2 Seite 53

20018 repgo Der <*ausdruck*> für das Aggregat <*aggregat*> muß eine Spalte (*spaltenname*) sein.
☞ Seite 84 Aggregate
☞ Seite 36 Spaltenauswahl

20019 repgo <*variablenname*> ist keine Variable.
☞ Seite 58 Format-Anweisungen im einzelnen LET
☞ Seite 21 DEFINE-Abschnitt

20021 repgo <*dateiname*> ist keine Datei.
☞ Seite 80 Funktionen OPEN

20023 repgo <*cursorname*> ist kein Cursor.
☞ Seite 54 Format-Anweisungen im einzelnen EXEC SQL

20024 repgo Es wurde ein Cursor erwartet.
☞ Seite 54 Format-Anweisungen im einzelnen CLOSE, DECLARE

20025 repgo Es kann kein Cursor deklariert werden.
☞ Seite 54 Format-Anweisungen im einzelnen DECLARE

20026 repgo Der Datentyp ist in diesem Kontext nicht zulässig (*spaltenname*).
▼ Der SQL-Datentyp der Spalte ist unbekannt.
☞ Seite 54 Format-Anweisungen im einzelnen EXEC SQL

20027 repgo Es wurde eine Variable von einem numerischen Datentyp, bzw. DATE, erwartet.

20029 repgo Der String konnte nicht eingelesen werden.

20040 repgo Die Variable bzw. der Wert muss ein INTEGER oder SMALLINT sein.
☞ Seite 73 Bitoperationen
☞ Seite 68 FORMAT-Ausdruck : Modulo Operator

20110 repgo Die Positionsindizes <*zeile, spalte*> müssen größer als 0 sein.
☞ Seite 59 Format-Anweisungen im einzelnen MOVEYX

20120 repgo Die angegebene Position <*zeile, spalte*> liegt außerhalb der virtuellen Seite.
☞ Seite 86 Virtuelle Seite
☞ Seite 59 Format-Anweisungen im einzelnen MOVEYX

20150 repgo Es wurde mit SAVEYX keine Position gespeichert.
☞ Seite 59 Format-Anweisungen im einzelnen MOVEYX
Seite 86 Virtuelle Seite

20160 repgo Das Zeichen <*c*> würde außerhalb der Seite geschrieben.
☞ Seite 59 Format-Anweisungen im einzelnen MOVEYX

☞ Seite 62 Format-Anweisungen im einzelnen PRINT
☞ Seite 86 Virtuelle Seite

20161 repgo Die Zeichenkette <*ccc*> würde außerhalb der Seite geschrieben.
☞ Seite 59 Format-Anweisungen im einzelnen MOVEYX
☞ Seite 61 Format-Anweisungen im einzelnen PRINT
☞ Seite 86 Virtuelle Seite

20200 repgo Unbekannter sqltype Datentyp.
☞ Seite 62 Format-Anweisungen im einzelnen PRINT
☞ Seite 49 FORMAT-Abschnitt EVERY ROW

20210 repgo Die Datei ist nicht im Schreibmodus geöffnet.
☞ Seite 80 Funktionen OPEN

20230 repgo In der Ortsangabe ist eine Rekursion aufgetreten
☞ Seite 49 Ortsangaben

20320 repgo Es wurden bereits *n* Dateien geöffnet.

20330 repgo <*modus*> ist ein unbekannter Modus.
Zulässig sind die Modi:
l, w, c bei READ und r, w, a bei OPEN
☞ Seite 81 Funktionen READ
☞ Seite 80 Funktionen OPEN

20340 repgo Der Modus ist unzulässig.
Zulässig sind die Modi:
l, w, c bei READ und r, w, a bei OPEN
☞ Seite 81 Funktionen READ
☞ Seite 80 Funktionen OPEN

20350 repgo Die Bedingung muß ein numerisches Ergebnis liefern.
☞ Seite 66 FORMAT-Bedingung

20360 repgo Die Laufvariable muß von einem ganzzahligen numerischen Datentyp sein.
☞ Seite 57 Format-Anweisungen im einzelnen FOR ...
☞ Seite 23 Datentypen

20370 repgo Der Startwert muß von einem ganzzahligen numerischen Datentyp sein.
☞ Seite 57 Format-Anweisungen im einzelnen FOR ...
☞ Seite 23 Datentypen

20380 repgo Die Schrittweite muß von einem ganzzahligen numerischen Datentyp sein.
☞ Seite 57 Format-Anweisungen im einzelnen FOR ... STEP
☞ Seite 23 Datentypen

20390 repgo Der Stopwert muß von einem ganzzahligen numerischen Datentyp sein.
☞ Seite 57 Format-Anweisungen im einzelnen FOR ... TO
☞ Seite 23 Datentypen

20400 repgo Der Ausgabekanal ist bereits mit einem Layout belegt.
☞ Seite 63 Format-Anweisungen im einzelnen SET PAGE ... *dateizeiger*

20500 repgo Die Variable zur Aufnahme einer Stringverkettung muß vom Datentyp CHAR sein.
☞ Seite 23 Datentypen

20510 repgo Die Zuweisung ist nicht durchführbar (*variablenname*).
☞ Seite 58 Format-Anweisungen im einzelnen LET

20600 repgo Eine Konversion von <*datentyp*> nach <*datentyp*> ist nicht möglich.

20700 repgo Es wurde eine deklarierte Variable erwartet.
☞ Seite 58 Format-Anweisungen im einzelnen LET

20710 repgo Die Variable <*variablenname*> ist zu diesem Zeitpunkt nicht deklariert.

20800 repgo <*variablenname*> und <*variablenname*> können nicht miteinander verglichen werden.

21000 repgo Bei der Addition ist ein Fehler aufgetreten.
☞ Seite 66 FORMAT-Bedingung

21010 repgo Bei der Subtraktion ist ein Fehler aufgetreten.
☞ Seite 68 FORMAT-Ausdruck

21020 repgo Bei der Multiplikation ist ein Fehler aufgetreten.
☞ Seite 68 FORMAT-Ausdruck

21030 repgo Bei der Division ist ein Fehler aufgetreten.
☞ Kap. 4 Seite 68
☞ Seite 58 Format-Anweisungen im einzelnen LET

21056 repgo Bei der Operation Links-Shift ist ein Fehler aufgetreten
☞ Seite 73 Bitoperationen

21060 repgo Bei der Operation Rechts-Shift ist ein Fehler aufgetreten
☞ Seite 73 Bitoperationen

21100 repgo Bei der Konversion von %6.2f nach DECIMAL ist ein Fehler aufgetreten
▼ Ein Wert zur Bildung eines Aggregates konnte nicht nach Decimal umgewandelt werden.
☞ Seite 84 Aggregate

21110 repgo Der REP enthält kein Aggregat <*aggregat*>.
☞ Seite 84 Aggregate

21120 repgo Für die Spalte <*spalte*> existiert kein Aggregat <*aggregat*>.
☞ Seite 84 Aggregate

21130 repgo Der Datentyp <*datentyp*> ist für Aggregat <*aggregat*> unzulässig.

21500 repgo In der ORDER BY-Klausel sind zu viele Spalten (*spaltenanzahl*) angegeben (INFORMIX - versionsspez.).
☞ Seite 36 SELECT-Anweisung ORDER BY

21530 repgo Die Spalte mit der Nummer <*spaltennummer*> wurde nicht selektiert.
☞ Seite 36 SELECT-Anweisung ORDER BY

21532 repgo Die Spalte mit dem Namen <*spaltenname*> wurde nicht selektiert.
☞ Seite 36 SELECT-Anweisung ORDER BY

22000 repgo Der String <*sss*> ist zu lang (INFORMIX - versionsspez.).
☞ Seite 55 Format-Anweisungen im einzelnen EXEC SQL “SQL-Anweisung”

22005 repgo Fehler beim Erzeugen der sqlda struct
▼ Beim Anlegen einer internen Speicherstruktur zum Zugriff auf die Datenbank ist ein Fehler aufgetreten.

22010 repgo Es sind zuviele Outputparameter angegeben (INFORMIX - versionsspez.).
☞ Seite 55 Format-Anweisungen im einzelnen EXEC SQL “SQL-Anweisung”

22016 repgo Fehler in den Inputparametern
☞ Seite 54 Format-Anweisungen im einzelnen EXEC SQL

22020 repgo Es sind zuviele Inputparameter angegeben (INFORMIX - versionsspez.)**.**
☞ Seite 55 Format-Anweisungen im einzelnen EXEC SQL “SQL-Anweisung”

22021 repgo Die Anweisung kann nicht ausgefuehrt werden.
☞ Seite 54 Format-Anweisungen im einzelnen EXEC SQL

22022 repgo Der Cursor ist bereits deklariert.
☞ Seite 54 Format-Anweisungen im einzelnen EXEC SQL

22023 repgo Der Cursor ist nicht deklariert.
☞ Seite 54 Format-Anweisungen im einzelnen EXEC SQL

22025 repgo Bei der ESQL-Anweisung <*anweisung*> *i*st ein Fehler aufgetreten (SQLCODE).

22026 repgo Bei der ESQL-Anweisung <*anweisung*> bei dem Statement <*argument*> ist ein Fehler aufgetreten (SQLCODE).

22027 repgo Beim Anlegen der temporaeren Tabelle \”%s\” ist ein Ueberlauf aufgetreten .
▼ Die maximale Anzahl temporärer Tabellennamen (z.Z. 20 unterschiedliche Namen) wurde überschritten (nur Oracle).

22028 repgo Fehler beim Zugriff auf die CLI Schnittstelle
▼ Die CLI Schnittstelle kann nicht initialisiert werden (nur DB2)

22030 Die ESQL-Anweisung <*anweisung*> darf nicht ausgefuehrt werden. (<*code*>)
☞ Seite 13 Übersetzen mit repprep - Schalter -m

22031 repgo Bei der ESQL-Anweisunng <*anweisung*> ist ein Fehler aufgetreten (SQLCODE=<*code*>) (ISAMCODE=<*code*>).

22032 repgo Bei der ESQL-Anweisung <*anweisung*> ist ein Fehler aufgetreten (SQLCODE=<*code*>) (SQLTXT=<*txt*>) (ISAMCODE=<*code*>)

22033 repgo Bei der ESQL-Anweisung <anweisung> bei dem Statement <stmt> ist ein Fehler aufgetreten (SQLCODE=<*code*>) (ISAMCODE=<*code*>).

22034 repgo Bei der ESQL-Anweisung <*anweisung*> bei dem Statement <*stmt*> ist ein Fehler aufgetreten (SQLCODE=<*code*>) (SQLTXT=<txt>) (ISAMCODE=<*code*>).

22035 repgo Bei der ESQL-Anweisung <*anweisung*> ist ein Fehler aufgetreten (SQLCODE=<*code*>) (SQLTXT=<*txt*>) (TOKEN=<*txt*>).

22036 repgo Bei der ESQL-Anweisung <*anweisung*> bei dem Statement <*stmt*> ist ein Fehler aufgetreten (SQLCODE=<*code*>) (SQLTXT=<*txt*>) (TOKEN=<*txt*>).
▼ Diese Meldungen (22031-22036) werden bei einem SQL Fehler ausgegeben. Je nach Datenbankversion und Situation stehen unterschiedliche Angaben zur Verfügung, welche jeweils durch eine andere Meldung ausgegeben werden.

22050 repgo Der Cursor ist bereits geöffnet.
☞ Seite 54 Format-Anweisungen im einzelnen EXEC SQL ...

22051 repgo Der Cursor ist nicht geöffnet
☞ Seite 54 Format-Anweisungen im einzelnen EXEC SQL ...

22052 repgo Fehler <*code*> in Cursorschnittstelle.
☞ Seite 54 Format-Anweisungen im einzelnen EXEC SQL ...

22053 repgo Der Cursor wurde nicht geschlossen.
☞ Seite 54 Format-Anweisungen im einzelnen EXEC SQL ...

22100 repgo Bei der Konversion in ein Datum ist ein Fehler aufgetreten.
☞ Seite 80 Funktionen MDY

22200 repgo Es wurden zuviele Argumente angegeben
(INFORMIX - versionsspez.).

22300 repgo Die Funktion <*funktionsname*> existiert nicht.

22310 repgo <*variablenname*> kann <*variablenname*> nicht zugewiesen werden.
☞ Seite 58 Format-Anweisungen im einzelnen LET

22320 repgo <*datentyp*> ist ein unbekannter C-Datentyp

22325 repgo <*datentyp*> ist ein unbekannter SQL-Datentyp

22330 repgo <*n*> Parameter wurden deklariert. Es sind jedoch <*m*> *a*ngegeben.
☞ Seite 13 Starten mit repgo
☞ Seite 22 PARAM

22340 repgo Der Input-Abschnitt konnte nicht korrekt abgearbeitet werden.
☞ Seite 29 INPUT-Abschnitt

22400 repgo Die Variable <*variablenname*> kann nicht in repgo aufgenommen werden.
☞ Seite 55 Format-Anweisungen im einzelnen EXEC SQL “SQL-Anweisung”

23000 repgo Position und Ortsangabe sind nicht konsistent (*aktuelle Seite, ~ Zeile, ~Spalte*).

23100 repgo Das Ergebnis der IF-Bedingung ist NULL.
☞ Seite 57 Format-Anweisungen im einzelnen IF .. THEN .. ELSE
☞ Seite 14 Customization Flags

23200 repgo Die Funktion ist unter WINDOWS nicht vorhanden.

24000 repgo Fehler beim Lesen der Attributsequenzen aus <datei>
☞ Seite 87 Attribute

25000 repgo VISUALREPHOST nicht gesetzt oder Host unbekannt

25001 repgo Fehler beim Verbindungsaufbau

25002 repgo Server antwortet nicht korrekt

25003 repgo Versionskonflikt zwischen REP und Visual Rep Server

25004 repgo unerlaubte Serveraktion

25005 repgo Datei konnte vom Visual Rep Server nicht geöffnet werden.

25006 repgo TESTOBJECT darf nicht zweimal hintereinander aufgerufen werden. (FITONPAGE fehlt)

25007 repgo Aufruf von TESTOBJECT fehlt.

25008 repgo Drucker konnte vom Visual REP Server nicht gefunden werden.

30000 repgo Es wurde ein Stack-Overflow festgestellt.

30010 repgo Es wurde ein Stack-Underflow festgestellt.

30030	repgo	Bei der Codegenerierung ist ein Fehler aufgetreten.
30100		Es ist ein Speicherfehler aufgetreten.

3 Stichwortverzeichnis

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

Y

Z