SQL/R

Report Generator
zu HP ELOQUENCE

Ergänzung zur Version A.01.41

Ausgabe:

A.01.00 - 1992
A.01.36 - Mai 1995, Addendum
A.01.41 - Mai 1996, Addendum

Übersicht

Die vorliegende Ergänzung zum **SQL/R**-Handbuch enthält eine vollständige Beschreibung der seit Erstellung des Handbuches neu hinzugekommenen Leistungsmerkmale.

Grundlage dieser Ergänzung ist die **SQL/R** Version A.01.41. Diese Version erfordert ein HP-UX Release ab 9.x.

Gliederung der Dokumentation

Typographische Konventionen

Solange nicht anders angegeben, werden in diesem Handbuch die folgenden symbolischen Konventionen verwendet:

`Computer Font`	Computer Font markiert Kommandos, Schlüsselworte, Optionen, Konstanten, Programmanweisungen, Ausgaben und Dateinamen.
[ ]	Das Symbol [ ] markiert eine Taste oder eine Schaltfläche auf dem Bildschirm („PushButton"), die mit der Maus angewählt wird. [CTRL] bezeichnet zum Beispiel die `Strg` („Control") Taste und [Weiter] eine Schaltfläche auf dem Bildschirm.
[CTRL]-[char]	[CTRL]-[char] markiert ein Steuerzeichen. Zum Beispiel [CTRL]-[Y] bedeutet, daß Sie *gleichzeitig* die Tasten `Strg` und `Y` auf der Tastatur drücken.
italics	Innerhalb von Syntaxbeschreibungen repräsentiert ein schräggestelltes Wort einen formalen Parameter oder ein Argument, das Sie durch den aktuellen Wert ersetzen müssen. Ersetzen Sie in dem folgenden Beispiel *filename* durch dem Namen der Datei, die sie drucken möchten: `lp` *filename*
[ ]	Innerhalb von Syntaxbeschreibungen werden optionale Elemente in eckigen Klammern eingeschlossen. In dem folgenden Beispiel bedeuten die eckigen Klammern um `[-d`*dev*`]`, daß der Parameter optional ist: `lp [-d`*dev*`]` *filename*
{ }	Innerhalb von Syntaxbeschreibungen werden verbindliche Elemente in geschweiften Klammern eingeschlossen. In dem folgenden Beispiel bedeuten die geschweiften Klammern um {`-c`|`-x`|`-v`}, daß einer der Parameter angegeben werden muß: `tar` {`-c`|`-x`|`-v`}

Zusätzliche Dokumentation

Auf die folgende Dokumentation wird im Handbuch verwiesen:

SQL/R Handbuch

Im **SQL/R** Handbuch finden Sie eine ausführliche Beschreibung der **SQL/R**-Anweisungen.

HP-UX (online) Dokumentation

Verweise im Handbuch in der Form `services`(4) beziehen sich auf den entsprechenden Eintrag (hier `services`) im angegebenen Abschnitt (hier Abschnitt 4) der HP-UX Referenz Dokumentation.

Mit Hilfe des `man` Kommandos ist diese Dokumentation auch online verfügbar. Für `services`(4) geben Sie bitte folgendes Kommando ein:

```
man 4 services
```

Inhaltsverzeichnis

1

Installation

Die Installationsprozedur für **SQL/R** hat sich, beginnend mit dem Release A.01.40 geändert. Die in der Dokumentation beschriebene Installationsanleitung ist nicht mehr gültig.

Dieses Release enthält außerdem noch folgende Änderungen:

- **SQL/R** kann ab der Version A.01.40 nur noch auf einem Betriebssystem HP-UX 9.x oder größer installiert werden. HP-UX 8.x wird nicht länger supportet.
- **SQL/R** wird nicht mehr wie bisher durch eine eigene Installationsroutine installiert sondern durch die üblichen HP-UX Installationsprogramme.
- **SQL/R** benutzt ab Version A.01.40 ein neues Lizenz-Schema. Anstatt die Programme mit der System-ID zu versehen, wird eine Lizenzdatei gelesen. Abhängig von der Betriebssystemversion finden Sie die Datei **`licence`** in den folgenden Verzeichnissen:

HP-UX 10.x	/etc/opt/sqlr
HP-UX 9.x	/opt/sqlr/etc

  Die Lizenzdatei ist eine reine Textdatei und enthält die Lizenzen aller **SQL/R** Produkte. Zur Prüfung der Lizenzdatei kann das Programm `/opt/sqlr/etc/chklic` verwendet werden.
- Der Server für **SQL/R Windows Client** muß nicht mehr gesondert installiert werden. Alle Dateien und Programme von **SQL/R** und **SQL/R Windows Client** werden automatisch installiert. Bitte beachten Sie jedoch, daß Sie einen Eintrag in der Lizenzdatei benötigen, um alle Programme ausführen zu können.
- **SQL/R A.01.40** hat gegenüber älteren Versionen eine gänderte Dateistruktur, um mit HP-UX 10.x kompatibel zu sein. Diese neue Dateistruktur wird auch unter HP-UX 9.x verwendet.

Weiterführende Informationen finden Sie in den Release-Notes, die im Verzeichnis `/opt/sqlr/newconfig/ReleaseNotes` installiert werden.

In den folgenden Abschnitten finden Sie gesonderte Anleitungen zur Installation von **SQL/R** unter HP-UX 9.x und HP-UX 10.x.

Hinweis:
Zur Installation werden mindestens 20 MB benötigt.
Bitte stellen Sie sicher, daß bei einem Update der Software **SQL/R** und der SQL/R Server für **SQL/R Windows Client** nicht mehr aktiv sind.
Sofern Ihre Lieferung keine gültige Lizenzukennung enthalten hat, fordern Sie vor Beginn der Installation eine Lizenzkennung mit beiliegendem Faxvordruck an.

1.1 Installation unter HP-UX 9.x

1. Melden Sie sich als Superuser (root) an
2. Legen Sie die DAT Kassette mit der SQL/R Software ein und laden Sie die Software vom Band in ein temporäres Verzeichnis

   ```
   cd /tmp
   tar xv /dev/rmt/0m SQLR.updt
   ```

   Hierbei ist */dev/rmt/0m* das DDS device file
3. Zur Installation von SQL/R starten Sie update (1m)

   ```
   /etc/update
   ```

   - Wählen Sie den Menüpunkt „Change Source or Destination ->" aus
   - Wählen Sie nun den Menüpunkt „From Tape Device to Local Sytem.."
   - Geben Sie im Feld „Source" den Pfad Ihrer Temporärdatei ein

     ```
     /tmp/SQLR.updt
     ```

   - Drücken Sie **F4** Done
   - Wählen Sie den Menüpunkt „Select/View Partitions and Filesets..." an
   - Aktivieren Sie den „SQL/R" Fileset und zusätzlich für eine deutschprachige Installation den Fileset „SQLR-G"
   - Wählen Sie **F4** „Start Loading"
   - Geben Sie ein „Y" auf die Frage „Start loading filesets now?" ein, um die Installation zu starten
4. Prüfen sie die Datei `/tmp/update.log` auf evtl. Fehlermeldungen
5. Wenn Sie **SQL/R** in den allgemeinen PATH für alle Benutzer eintragen möchten, editieren Sie die Datei `/etc/profile`. Hängen Sie die folgende Zeile dort an:

   ```
   PATH=$PATH:/opt/sqlr/bin
   ```

   Falls Sie **SQL/R** nur im PATH eines einzelnen Benutzers haben möchten, können Sie die o.g. Zeile in der Datei `.profile` im jeweiligen Home-Verzeichnis des entsprechenden Benutzers eintragen.
6. Tragen Sie Ihre Lizenzkennung in der Datei `licence` ein

1.2 Installation unter HP-UX 10.x

1. Melden Sie sich als Superuser (root) an
2. Legen Sie die DAT Kassette mit der SQL/R Software ein und laden Sie die Software vom Band in ein temporäres Verzeichnis:

   ```
   cd /tmp
   tar xv /dev/rmt/0m SQLR.sd
   ```

   Hierbei ist */dev/rmt/0m* das DDS device file.
3. Zum Start des Installationsprogrammes swinstall (1m) geben Sie bitte folgendes ein:

   ```
   /usr/sbin/swinstall -s /tmp/SQLR.sd
   ```

   Hierbei ist */tmp/SQLR.sd* der Pfad Ihrer Temporärdatei.

   - Sie befinden Sich nun im Menü „SD Install - Software Selection"
   - Selektieren Sie das Bundle „SQL/R-E" für eine englischsprachige oder das Bundle „SQLR-G" für eine deutschsprachige Installation
   - Danach wählen Sie aus dem Menü „Actions" den Menüpunkt „Mark for Install". Nun wird in der Spalte „Marked?" ein „Yes" ausgegeben
   - Wählen Sie nun aus dem Menü „Actions" den Menüpunkt „Install (analysis...)".
   - Wenn die Analyse ohne Fehlermeldung beendet wurde (`Status: Ready`), klicken Sie auf OK
   - Durch die Auswahl von **YES** im Menü „Confirmation" starten Sie die Installation.
4. Wenn die Installation abgeschlossen ist, wird Ihnen dies in einem Ausgabefenster angezeigt.

1.3 Löschen alter Versionen

Falls Sie bereits eine frühere Version von **SQL/R** in Verzeichnis `/usr/sqlr` installiert hatten, sollten Sie die alte Software im Verzeichnis **manuell** löschen.

Warnung: Wenn Sie dieser Anleitung folgen, werden alle Dateien und Verzeichnisse unter `/usr/sqlr` gelöscht!
Sofern Sie dort eigene Dateien (Scripts) oder Auswertungen gespeichert haben, müssen diese vorher gesichert oder in ein anderes Verzeichnis kopiert werden.

Zum Löschen der alten Version gehen Sie wie folgt vor:

1. Melden sie sich als Superuser (root) an.
2. Wechseln Sie das Verzeichnis /tmp:
   ```
   cd /usr
   ```
3. Löschen Sie das Verzeichnis `sqlr` mit allen Unterverzeichnissen
   ```
   rm -rf sqlr
   ```
4. Wechseln Sie das Verzeichnis /usr/bin:
   ```
   cd /usr/bin
   ```
5. Löschen Sie folgende Dateien:
   ```
   rm sqlr sqlred sqlrexec
   ```
6. Wechseln Sie in das Verzeichnis:
   ```
   cd /usr/bin
   ```
7. Löschen Sie die alten Message-Kataloge von SQL/R:
   ```
   rm /usr/lib/nls/C/sqlr.cat
   rm /usr/lib/nls/german/sqlr.cat
   ```

1.4 Weiterführende Hinweise

Bitte prüfen Sie bei einem Update auch die Angaben bezüglich Ihrer PATH-Variablen in den Dateien `/etc/profile` und/oder in den `.profile` Dateien der einzelnen Benutzer.

Im Verzeichnis `/opt/sqlr/newconfig/Release.Notes` finden Sie in den Dateien A.01.40 und A.01.41 weiterführende Informationen über gelöste Probleme oder neue Funktionalität von **SQL/R**.

Das Verzeichnis `/opt/sqlr/newconfig/Release.Notes` enthält folgende Dateien:

README.g	Release Notes in Deutsch
README.e	Release Notes in Englisch
INSTALL	Installationsanleitung
A.01.35	Release Notes zu A.01.35
A.01.40	Release Notes zu A.01.40
A.01.41	Release Notes zu A.01.41
README.srv	Release Notes zu SQL/R Windows Client, Server Side

1.5 Neue Produktstruktur

Mit Freigabe von HP-UX 10.0 hat Hewlett Packard die Dateistruktur geändert. Diese lehnt sich nun an SVR4 und OSF an. Dieses Modell hat unter anderem den Vorteil, daß Betriebssystem-Bereiche und Applikationen getrennt sind.

SQL/R ab Version A.01.40 basiert auf den Vorgaben der HP-UX 10.0 Dateistruktur und verwendet die gleiche Dateistruktur auch auf HP-UX 9.x.

```
/opt/sqlr/                    Base directory
   |
   |-- etc/                   Utility programs
   |                          HP-UX 9.x: Application specific
   |                                     configuration files
   |
   |--- bin/                  Executable programs
   |
   |--- lib/                  Libraries
   |       |
   |       |-- nls/
   |             |
   |             |-- C/       Default message catalog
   |             |-- german/  German message catalog
   |
   |--- newconfig/
   |       |
   |       |-- ReleaseNotes/  Release Notes
   |       |
   |       |-- etc/           Example licence file
   |       |
   |       |-- startup/       HP-UX 10.0 startup/shutdown scripts
```

```
 |
 |
 |--- share/
 |      |
 |      |--- map/           Character set mappings
 |      |
 |      |--- sample/        Example scripts (from manual)
 |      |                   Linked to a language dependend directory
 |      |--- sample.e/      - English example scripts
 |      |--- sample.g/      - German example scripts
 |      |
 |      |--- db/            Example database
 |      |                   Linked to a language dependend directory
 |      |--- db.e/          - English example database
 |      |--- db.g/          - German example database
 |      |
 |      |--- example/       Example programs
```

Die folgenden Verzeichnisse sind nur bei HP-UX 10.x vorhanden:

```
/etc/opt/sqlr/              Application specific
                            configuration files

/var/opt/sqlr/              Application specific temporary
                            files

/etc/rc.config.d/           Startup configuration files

/sbin/
 |
 | --- init.d/              Startup and shutdown scripts
 |
 | --- rc1.d/               Startup and shutdown link files
 | --- rc2.d/               for script sequencing
```

2

Neue Funktionen

In diesem Kapitel werden die neuen Funktionen von **SQL/R** ausführlich beschrieben. Einige Funktionen werden als "Familie" gemeinsam beschrieben, da sie verwandte bzw. ähnliche Arbeitsweisen und Aufgaben haben.

Jeder Eintrag besteht aus den folgenden einzelnen Punkten:

- Kurzbeschreibung der Funktion
- Syntaxdefinition für die Verwendung der Funktion
- Ausführliche Beschreibung der Arbeitsweise
- Eventueller Rückgabewert
- Beispiel für die Verwendung

Die Funktionen werden vor allem bei der Modifikation bestehender bzw. der Definition neuer Felder und Ausdrücke verwendet. Daher beginnen auch alle mit dem Zeichen „ @ ", um mögliche Konflikte mit bestehenden Feldnamen zu vermeiden.

Die Funktionen lassen sich in die folgenden Kategorien einteilen:

- **Konvertierung von Datentypen**
  Mit Hilfe dieser Funktionen können Datentypkonvertierungen durchgeführt werden. So werden zum Beispiel Zeichen und Strings in numerische Werte oder Datumsformate umgewandelt und umgekehrt.

@CHAR	@NUM	@VALUE
@DATETOCHAR	@STRING	
@DATEVALUE	@TIMEVALUE	

- **Manipulation von Zeichenketten**

@LEFT	@POS	@SUBSTR
@LENGTH	@RIGHT	@TRIM
@LOWER	@RPT	@UPPER

- **Numerische Funktionen**

@ABS	@FRACT	@MOD
@DIV	@INT	@ROUND

- **Datums- und Zeitfunktionen**

@DATE	@MINUTE	@SECONDS
@DATETOCHAR	@MINUTES	@TIME
@DATEVALUE	@MONTH	@TIMEVALUE
@DAY	@MONTHBEG	@WEEKBEG
@DAYS	@NOW	@WEEKDAY
@DIFFTIME	@QUARTER	@WEEKS
@HOUR	@QUARTERBEG	@YEAR
@HOURS	@SECOND	@YEARBEG

Einige der hier aufgeführten Funktionen sind bereits Bestandteil von **SQL/R** (siehe Kapitel 6.7 ff.), jedoch ohne das führende Zeichen „@". Diese Funktionen werden aus Gründen der Kompatibilität auch weiterhin zusätzlich in ihrer alten Schreibweise unterstützt. Sie sollten aber, auch wegen der einheitlichen Syntax, in neuen Skripts nur noch die aktuelle Schreibweise verwenden.

2.1 Konvertierung von Datentypen

2.1.1 @CHAR

Aktion

Umwandlung eines numerischen Wertes in ein Zeichen.

Syntax

`@CHAR( Wert )`

Beschreibung

`@CHAR` liefert das Zeichen, das dem angegebenen Argument *Wert* entspricht.

Der Wert sollte zwischen 0 und 255 liegen, um ein sinnvolles Ergebnis zu erhalten. Bei abweichenden Werten wird dieser bitweise mit 256 verknüpft, so daß die Funktion immer ein entsprechendes Zeichen liefert.

Rückgabewert

Die Funktion liefert das zum Argument *Wert* gehörige Zeichen.

Beispiel

```
@CHAR( 49 ) und @CHAR( -207 ) liefern beide "1"
@CHAR( 65 ) und @CHAR(  321 ) liefern beide "A"
```

2.1.2 @DATETOCHAR

Aktion

Umwandlung eines Datumswertes in einen String.

Syntax

`@DATETOCHAR( Datum, Format )`

Beschreibung

`@DATETOCHAR` wandelt den angegebenen Datumswert *Datum* in einen String um und liefert diesen zurück. Die Ausgabe des erstellten Strings wird mit dem Argument *Format* festgelegt. Eine vollständige Liste der zulässigen Formate finden sie im Anhang B des **SQL/R** Handbuches.

Rückgabewert

Die Funktion liefert den erzeugten String zurück.

Beispiel

Die im Beispiel verwendete Konstante `@NOW` beinhaltet den Datumswert für das aktuelle Tagesdatum.

Mit entsprechenden Formatangaben ergeben sich die folgenden unterschiedlichen Ergebnisse.

```
@DATETOCHAR( @NOW, "%c") liefert:
      "Mo., 22. Mai 1995, 17:58:29"
@DATETOCHAR( @NOW, "Heute ist der %d.%m.%y") liefert:
      "Heute ist der 22.05.95"
```

2.1.3 @DATEVALUE

Aktion

Umwandlung eines Strings bzw. String-Ausdrucks in einen Datumswert.

Syntax

`@DATEVALUE( String )`

Beschreibung

`@DATEVALUE` wandelt den im Argument *String* enthaltenen String in das interne Datumsformat (Anzahl der Sekunden seit dem 01.01.1970) um. Das im String verwendete Datumsformat spielt dabei für die korrekte Berechnung des Wertes keine Rolle. Beachten Sie, daß Datumskonstanten (z.B. "@01.01.95") nicht explizit umgewandelt werden müssen, da dies von **SQL/R** intern vorgenommen wird.

Rückgabewert

Die Funktion liefert den zugehörigen Datumswert zurück. Der Rückgabewert ist NULL, wenn der String kein korrektes Datum enthält (z.B. "Heute ist der 22.07.93").

Beispiel

Die folgenden Aufrufe von @DATEVALUE liefern alle den Wert `743292000`:

```
@DATEVALUE( "07/22/93" )
@DATEVALUE( "22.07.93" )
@DATEVALUE( "930722"   )
```

Damit lassen sich z.B. Datumsabfragen lesbarer formulieren:

```
SELECT ... WHERE @DATEVALUE("01.01.95") <= buchdat <= @NOW;
```

Mit dieser Anweisung werden alle Datensätze gesucht, in denen das Buchungsdatum zwischen dem 01.01.95 und dem aktuellen Tagesdatum liegt.

Dieselbe Anweisung mit `BETWEEN ... AND ...` führt jedoch in diesem Fall zu einer Fehlermeldung, da die Ergebnisse von Funktionen nicht als Konstanten angesehen werden.

2.1.4 @NUM

Aktion

Ermittlung des ASCII-Wertes vom ersten Zeichen des Stringarguments.

Syntax

`@NUM( String )`

Beschreibung

`@NUM` liefert den numerischen Wert des ersten Zeichens im Argument *String*. Dies kann ein normaler Text oder auch ein String sein, der aus einem einzelnen Zeichen besteht.

Beachten sie, daß das Zeichen "\" innerhalb eines Strings ein Steuerzeichen ist. Um den gewünschten Wert zu erhalten muß der String in diesem Fall mit einem doppelten "\\" beginnen (z.B. "\\Text"). Wollen sie den Wert für das Anführungszeichen ("), so muß diesem ein "\" vorangestellt werden.

Rückgabewert

Die Funktion liefert den ermittelten Wert zurück.

Beispiel

```
@NUM( "A" )        liefert den Wert 65
@NUM( "\A" )       liefert den Wert 65
@NUM( "Beispiel" ) liefert den Wert 66

@NUM( "\" )         FALSCH
@NUM( "\\" )        RICHTIG
@NUM( """ )         FALSCH
@NUM( "\"" )        RICHTIG
@NUM( '"' )         EINFACHER
```

2.1.5 @STRING

Aktion

Umwandlung eines numerischen Wertes in einen String.

Syntax

`@STRING( Format, Zahl )`

Beschreibung

`@STRING` wandelt den angegeben Wert bzw. das Ergebnis des numerischen Ausdrucks *Zahl* in einen String um und liefert diesen zurück. Dazu verwendet die Funktion das mit dem Argument *Format* festgelegte Ausgabeformat. Darüberhinaus hängt das Ergebnis auch von der eingestellten Arbeitumgebung ab, insbesondere den Ländereinstellungen.

Das Argument *Format* ist ein String, der Formatierungsanweisungen enthält, welche die Auswertung des Arguments *Zahl* bestimmen.

Die Formatierungsanweisungen haben die folgende Syntax:
`% [Flags] [Breite] [.Genauigkeit] Typ`

Zeichen	Bedeutung
`%`	Stellt den Beginn der Formatanweisung dar. Fehlt es, wird der Rest der Anweisung als normaler Text interpretiert und ausgegeben.
`[Flags]`	Bestimmt Ausrichtung der Ausgabe, Vorzeichen, Dezimalpunkt, führende bzw. folgende Nullen.
`[Breite]`	Legt die Minimalzahl der ausgegebenen Zeichen fest, wobei notfalls mit Leerzeichen oder Nullen aufgefüllt wird.
`[.Genauigkeit]`	Legt die Maximalzahl von ausgegebenen Zeichen fest, wobei die genaue Bedeutung von gewählten Typ abhängt.
`[Typ]`	Ist das abschließende Zeichen der Formatanweisung. Fehlt diese Angabe, erzeugt die Funktion fehlerhafte Ergebnisse. Die einzigen zulässigen Typen sind "f" und "g".

[Flags]	Bedeutung
`Minuszeichen (-)`	Linksbündige Ausgabe, wobei notfalls Leerzeichen angefügt werden. Standard ist die rechtsbündige Ausgabe mit führenden Nullen oder Leerzeichen.
`Pluszeichen (+)`	Das Vorzeichen des Werts wird stets ausgegeben. Standard ist die Ausgabe des Vorzeichens nur bei negativen Werten.
`Leerzeichen ( )`	Positive Werte beginnen mit einem Leerzeichen. Negative Werte beginnen weiterhin mit einem Minuszeichen.
`Doppelkreuz (#)`	Es wird immer ein Dezimalpunkt ausgegeben. Zusätzlich werden bei Typ *g* abschließende Nullen nicht unterdrückt.

[Breite]	Bedeutung
`n`	Ausgabe von mindestens `n` Zeichen. Notfalls werden Leerzeichen vorangestellt (Standard) oder angefügt (Linksbündige Ausgabe mit `Flag -`).
`0n`	Ausgabe von mindestens `n` Zeichen. Wenn nötig, wird eine entsprechende Anzahl führender Nullen vorangestellt.

Typ	[.Genauigkeit]	Bedeutung
f		
	keine	Es werden 6 Nachkommastellen ausgegeben.
	.0 oder .	Keine Ausgabe des Dezimalpunktes und von Nachkommastellen. Wenn nötig, wird die Ausgabe entsprechend gerundet.
	.n	Ausgabe von mindestens n Stellen nach dem Komma. Die letzte Stelle wird, wenn nötig, gerundet.
g		
	keine	Es wer 6 signifikante Dezimalstellen ausgegeben (inklusive der Stellen vor dem Komma). Die Ausgabe erfolgt in Exponenten-Darstellung ([-]d.ddd**e**[+/-]dd).
	.0 oder .	Es erfogt eine Ausgabe in Exponenten-Darstellung ([-]d**e**[+/-]dd).
	.n	n legt die Anzahl signifikanter Dezimalstellen fest, und zwar inclusive der Stellen vor dem Komma. Ist n kleiner als die Anzahl der Vorkommastellen, wird wieder in Exponenten-Darstellung ausgegeben.

Im Gegensatz zu Typ **f** werden bei Typ **g** abschließende Nullen und der Dezimalpunkt nur ausgegeben, wenn dies notwendig ist.

Rückgabewert

Die Funktion liefert den erzeugten String zurück.

Beispiel

```
SET LOCALE "NUMERIC=C";
@STRING( "%06.3g", 1.2345) liefert "001.23"

SET LOCALE "NUMERIC=german";
@STRING( "%06.3f", 1.2345) liefert "01,234"
@STRING( "%06.3g", 1.2345) liefert "001,23"
@STRING( "%+6.3f", 1.2345) liefert "+1,234"
@STRING( "%+6.3g", 1.2345) liefert " +1,23"
@STRING( "%-6.3f", 1.2345) liefert "1,234 "
@STRING( "%-6.3g", 1.2345) liefert "1,23  "
```

2.1.6 @TIMEVALUE

Aktion

Umwandlung eines Strings bzw. String-Ausdrucks in einen Zeitwert.

Syntax

`@TIMEVALUE( String )`

Beschreibung

`@TIMEVALUE` wandelt den im Argument *String* enthaltenen String in den zugehörigen Zeitwert um, also in die Anzahl der Sekunden seit 0:00:00 Uhr. Das im String verwendete Format spielt dabei für die korrekte Berechnung des Wertes keine Rolle. Beachten Sie, daß Zeitkonstanten (z.B. "@12:30:15") nicht explizit umgewandelt werden müssen, da dies von **SQL/R** intern vorgenommen wird.

Rückgabewert

Die Funktion liefert den zugehörigen Zeitwert zurück. Der Rückgabewert ist NULL, wenn der String keine korrekte Zeitangabe enthält (z.B. "Es ist jetzt 15:10:20").

Beispiel

Die folgenden Aufrufe von @TIMEVALUE liefern beide den Wert `54302`:

```
@TIMEVALUE( "15:05:02" )
@TIMEVALUE( "150502"   )
```

Die Aufrufe von @TIMEVALUE ohne die Sekundenangabe im String liefern entsprechend den Wert `54300`:

```
@TIMEVALUE( "15:05" )
@TIMEVALUE( "1505" )
```

2.1.7 @VALUE

Aktion

Umwandlung eines Strings bzw. String-Ausdrucks in eine Zahl.

Syntax

`@VALUE( String )`

Beschreibung

`@VALUE` wandelt den im Argument *String* enthaltenen String in den zugehörigen numerischen Wert um, also eine Zahl. Der eventuell im String enthaltene Dezimalpunkt kann entweder ein "." sein oder ein Zeichen, das durch die aktuelle Arbeitsumgebung festgelegt ist.

Bitte beachten sie, daß die Konvertierung immer beim ersten unzulässigen Zeichen im String beendet wird.

Rückgabewert

Die Funktion liefert den ermittelten Wert zurück. Das Ergebnis ist immer vom Typ **double**. Dies gilt auch, wenn das Ergebnis ganzzahlig ist.

Beispiel

```
SET LOCALE "NUMERIC=C";
@VALUE( "12.34" ) liefert 12.34
@VALUE( "12,34" ) liefert 12.00
```

Im zweiten Fall wird das Komma (",") als ein unzulässiges Zeichen angesehen und die Konvertierung abgebrochen.

```
SET LOCALE "NUMERIC=german";
@VALUE( "12.34" ) liefert 12,34
@VALUE( "12,34" ) liefert 12,34
```

Beachten sie die Tatsache, daß der Rückgabewert vom Typ `double` ist:

```
FIELD xx = IF (..., @VALUE( "100" ), 10);
```

Diese Anweisung erzeugt eine Fehlermeldung wegen der unterschiedlichen Datentypen innerhalb der Abfrage. Richtig muß diese Anweisung wie folgt formuliert werden:

```
FIELD xx = IF (..., @VALUE( "100" ), 10.);
```

Durch den Punkt wird die Konstante "10" als Wert vom Typ `double` kenntlich gemacht.

2.2 Manipulation von Zeichenketten

2.2.1 @LEFT, @RIGHT, @SUBSTR

Aktion

Mit diesen Funktionen können Teile aus bestehenden Strings herauskopiert werden.

Syntax

```
@LEFT( String, Anzahl )
@RIGHT( String, Anzahl )
@SUBSTR( String, Start, Anzahl )
```

Beschreibung

Die Funktionen `@LEFT`, `@RIGHT` und `@SUBSTR` liefern jeweils die durch das Argument *Anzahl* festgelegte Anzahl von Zeichen aus dem im Argument *String* angegebenen String.

Ist die gewünschte Anzahl größer als die Länge des vorgegebenen Strings, wird der gesamte String geliefert. Es wird nicht mit Leerzeichen aufgefüllt.

Der Unterschied dieser drei Funktionen besteht im Startpunkt. Die Funktion `@LEFT` kopiert die Zeichen vom Anfang, die Funktion `@RIGHT` vom Ende des Strings. Die Funktion `@SUBSTR` beginnt den Kopiervorgang an der mit dem Argument *Start* festgelegten Position. Dabei hat das erste Zeichen des Strings die Position 0.

Rückgabewert

Die Funktion liefert den erstellten String zurück. Sie liefert `NULL`, wenn die Argumente *Anzahl* bzw. *Start* unzulässige Werte (z.B. kleiner 0) enthalten.

Beispiel

```
@LEFT( "1234567890",   5)    liefert "12345"
@LEFT( "123",          5)    liefert "123"

@RIGHT( "1234567890",  5)    liefert "67890"
@RIGHT( "123",         5)    liefert "123"

@SUBSTR( "1234567890", 4, 3) liefert "567"
@SUBSTR( "1234567890", 7, 5) liefert "890"
```

2.2.2 @LENGTH

Aktion

Bestimmt die Länge eines Strings.

Syntax

`@LENGTH( String )`

Beschreibung

Die Funktion `@LENGTH` liefert die Anzahl der Zeichen (Länge) der im Argument *String* angegebenen Zeichenkette zurück.

Rückgabewert

Die Funktion liefert die ermittelte Länge zurück.

Beispiel

```
@LENGTH( "1234567890" ) liefert den Wert 10
```

2.2.3 @LOWER, @UPPER

Aktion

Umwandlung von Groß- in Kleinbuchstaben bzw. von Klein- in Großbuchstaben.

Syntax

```
@LOWER( String )
@UPPER( String )
```

Beschreibung

Die Funktion `@LOWER` wandelt die in der Zeichenkette *String* enthaltenen Großbuchstaben in Kleinbuchstaben um. Die Funktion `@UPPER` wandelt umgekehrt Kleinbuchstaben in Großbuchstaben um. Alle anderen Zeichen bleiben unverändert.

Die Umwandlung ist abhängig von der gewählten Arbeitsumgebung (texttLANG bzw. LC_CTYPE). Diese ist ausschlaggebend bei der Umwandlung von Sonderzeichen (z.B. ä, ö, ü, Ä, Ö, Ü).

Rückgabewert

Die Funktion liefert den umgewandelten String zurück.

Beispiel

```
@UPPER( "test" ) liefert "TEST"
@LOWER( "TEST" ) liefert "test"

SET LOCALE "CTYPE=C";
SELECT @UPPER( "äöüß" ), @LOWER( "ÄÖÜß" );

ergibt die Ausgabe:  "äöüß   ÄÖÜß"

SET LOCALE "CTYPE=german";
SELECT @UPPER( "äöüß" ), @LOWER( "ÄÖÜß" );

ergibt die Ausgabe:  "ÄÖÜß   äöüß"
```

2.2.4 @POS

Aktion

Suche einen String nach dem Vorkommen eines bestimmten Teilstrings ab.

Syntax

`@POS( String1, String2 )`

Beschreibung

Die Funktion `@POS` sucht die über das Argument *String1* angegebene Zeichenkette nach dem ersten Vorkommen des Teilstrings *String2* ab und ermittelt dessen Startposition in *String1*.

Rückgabewert

Die Funktion liefert die ermittelte Position zurück. Der Rückgabewert ist 0, wenn *String2* nicht in *String1* enthalten ist.

Beispiel

```
@POS( "1234512345", "34" ) liefert Position 3
@POS( "1234512345", "XX" ) liefert den Wert 0
```

2.2.5 @RPT

Aktion

Bildet einen neuen String durch mehrmaliges Aneinanderhängen eines vorgegebenen Strings.

Syntax

`@RPT( String, Anzahl )`

Beschreibung

Die Funktion `@RPT` kopiert die über das Argument *String* angegebene Zeichenkette in einen neuen String und wiederholt diesen Vorgang so oft, wie es mit dem Argument *Anzahl* festgelegt worden ist.

Rückgabewert

Die Funktion liefert den erstellten String zurück oder NULL, falls der Wert für das Argument *Anzahl* unzulässig ist.

Beispiel

```
@RPT( "ABC",  3 ) liefert den String "ABCABCABC"
@RPT( "-",   10 ) liefert den String "----------"
@RPT( "ABC",  0 ) liefert einen Leerstring ""
@RPT( "ABC", -1 ) liefert den Wert NULL
```

2.2.6 @TRIM

Aktion

Lösche führende und angehängte Leerzeichen in einem String.

Syntax

`@TRIM( String )`

Beschreibung

Die Funktion `@TRIM` löscht alle führenden und angehängten Leerzeichen (" ") in der angegebenen Zeichenkette *String*.

Rückgabewert

Die Funktion liefert den modifizierten String zurück.

Beispiel

```
@TRIM( "   A B C  " ) liefert "A B C"
@TRIM( "1234      " ) liefert "1234"
```

2.3 Numerische Funktionen

2.3.1 @ABS

Aktion

Berechnet den absoluten Wert einer Zahl.

Syntax

`@ABS( Zahl )`

Beschreibung

Die Funktion `@ABS` berechnet den absoluten Wert des Arguments *Zahl*. Beachten Sie, daß die Ausgabe dieses Wertes standardmäßig mit 2 Nachkommastellen erfolgt, so daß die Ausgabe auch gerundet sein kann, obwohl der Wert selbst nicht gerundet wird. Für eine korrekte Ausgabe sollten Sie in diesem Fall eine `FIELD ... DISPLAY AS` Anweisung verwenden.

Rückgabewert

Die Funktion liefert den Absolutwert zurück. Der Rückgabewert ist immer von Datentyp **double**.

Beispiel

```
FIELD v1 = -1.236;
FIELD v2 = @ABS( v1 );
FIELD v3 = v2 * 100.0;
FIELD v4 = @ABS( v1 ) DISPLAY AS DOUBLE(8,4);

SELECT v1, v2, v3, v4;

Die Anweisungen liefern das folgende Ergebnis:
          V1      V2       V3          V4
      -1,236    1,24    123,6      1,2360
```

2.3.2 @DIV

Aktion

Ermittelt den ganzzahligen Anteil der Division zweier Werte.

Syntax

`@DIV( Zahl, Divisor )`

Beschreibung

Die Funktion `@DIV` dividiert die mit den Argumenten *Zahl* und *Divisor* übergebenen Werte und liefert den ganzzahligen Anteil dieser Division zurück.

Rückgabewert

Die Funktion liefert den berechneten ganzzahligen Teil der Division zurück. Der Datentyp des Rückgabewert ist identisch mit dem Datentyp des Arguments *Zahl*.

Beispiel

```
@DIV(   20, 3 ) liefert den Wert 6    vom Typ int
@DIV( 20.0, 3 ) liefert den Wert 6.00 vom Typ double
```

2.3.3 @FRACT

Aktion

Berechnet die Differenz einer beliebigen Zahl zur nächstkleineren ganzen Zahl.

Syntax

`@FRACT( Zahl )`

Beschreibung

Die Funktion `@FRACT` ermittelt die Differenz zwischen dem übergebenen Wert *Zahl* und der nächstkleineren ganzen Zahl. Beachten Sie dabei, daß das Ergebnis dieser Funktion von HP Eloquence abweicht, wenn der übergebene Wert negativ ist.

Rückgabewert

Die Funktion liefert den berechneten Wert zurück. Dieser Wert ist immer positiv.

Beispiel

```
@FRACT(  1.48 ) liefert  1.48 -   1.00  = 0.48
@FRACT(  1.78 ) liefert  1.78 -   1.00  = 0.78
@FRACT( -1.48 ) liefert -1.48 - (-2.00) = 0.52
@FRACT( -1.78 ) liefert -1.78 - (-2.00) = 0.22
```

2.3.4 @INT

Aktion

Ermittelt die nächstkleinere ganze Zahl.

Syntax

`@INT( Zahl )`

Beschreibung

Die Funktion `@INT` rundet den mit dem Argument *Zahl* übergebenen Wert auf die größte ganze Zahl ab, die nicht größer als *Zahl* ist.

Rückgabewert

Die Funktion liefert den berechneten Wert zurück.

Beispiel

```
@INT(  1.48 ) liefert  1.00,   @INT( -1.48 ) liefert -2.00
@INT(  1.78 ) liefert  1.00,   @INT( -1.78 ) liefert -2.00
```

2.3.5 @MOD

Aktion

Ermittelt den Rest einer Division zweier beliebiger Werte.

Syntax

`@MOD( Zahl, Divisor )`

Beschreibung

Die Funktion `@MOD` dividiert die mit den Argumenten *Zahl* und *Divisor* übergebenen Werte und liefert den Divisionrest zurück. Sei Zahl = N * Divisor + X, wobei N eine ganze Zahl ist, dann ermittelt `@MOD` den Rest X.

Beachten Sie bitte, daß das Ergebnis dieser Funktion von HP Eloquence abweicht, wenn die übergebenen Werte negative Vorzeichen haben.

Rückgabewert

Die Funktion liefert den berechneten Divisionsrest zurück. Der Datentyp des Rückgabewert ist abhängig vom Datentyp des Arguments *Zahl* und dem berechneten Wert. Ist das Argument vom Typ `int` und der Rest ganzzahlig, so ist auch der Datentyp des Ergebnisses `int`. Ansonsten ist der Rückgabewert immer vom Typ `double`.

Beispiel

```
@MOD(  20 ,  3.5 ) liefert  2.5 vom Typ double
@MOD( -20 ,  3   ) liefert -2   vom Typ int
@MOD(  20., -3   ) liefert  2.0 vom Typ double
@MOD( -20., -3.5 ) liefert -2.5 vom Typ double
```

2.3.6 @ROUND

Aktion

Rundet eine beliebige Zahl auf eine vorgegebene Genauigkeit.

Syntax

`@ROUND( Zahl, Genauigkeit )`

Beschreibung

Die Funktion `@ROUND` rundet den im Argument *Zahl* übergebenen Wert auf die im Argument *Genauigkeit* angegebene Zehnerpotenz, also 10ˆ*Genauigkeit*. Dieser Wert legt damit fest, auf wieviel Vorkommastellen (*Genauigkeit* >= 0) bzw. Nachkommastellen (*Genauigkeit* < 0) gerundet werden soll.

Rückgabewert

Die Funktion liefert den gerundeten Wert zurück. Dieser ist immer vom Datentyp `double`.

Beispiel

```
@ROUND(  1.494,  0) ist  1.00,  @ROUND(  1.5,  0) ist   2.00
@ROUND( -1.494,  0) ist -1.00,  @ROUND( -1.5,  0) ist  -2.00
@ROUND(  1.494, -2) ist  1.49,  @ROUND(  1.5, -1) ist   1.50
@ROUND( -1.495, -2) ist -1.50,  @ROUND( -1.5, -1) ist  -1.50
@ROUND(  4.995,  1) ist  0.00,  @ROUND(  5.0,  1) ist  10.00
@ROUND( -4.995,  1) ist  0.00,  @ROUND( -5.0,  1) ist -10.00
```

2.4 Datums- und Zeitfunktionen

2.4.1 @DATE

Aktion

Berechnet den Datumswert aus vorgebenen Werten für Jahr, Monat und Tag.

Syntax

`@DATE( Jahr, Monat, Tag )`

Beschreibung

Die Funktion `@DATE` bildet aus den Argumenten *Jahr*, *Monat* und *Tag* ein vollständiges Datum und ermittelt den zugehörigen Datumswert, also die Anzahl der Sekunden seit dem 01.01.1970. Mit Hilfe der `DISPLAY AS DATE` Anweisung kann das so ermittelte Datum im gewünschten Format ausgegeben werden.

Rückgabewert

Die Funktion liefert den berechneten Datumswert als `long` Wert zurück.

Beispiel

```
FIELD f_date = @DATE( 95, 5, 20 )
                  DISPLAY AS DATE("%d.%m.%y");
SELECT f_date, @DATE( 95, 5, 20 ), @20.05.95;

Die beiden Anweisungen liefern das folgende Ergebnis:
     F_DATE     @DATE(95,5,20)     20.05.95
     20.05.95        800920800    800920800
```

2.4.2 @TIME

Aktion

Berechnet den Zeitwert aus vorgebenen Werten für Stunde, Minute und Sekunde.

Syntax

`@TIME( Stunde, Minute, Sekunde )`

Beschreibung

Die Funktion `@TIME` bildet aus den Argumenten *Stunde*, *Minute* und *Sekunde* ein vollständige Zeitangabe und ermittelt daraus den zugehörigen Zeitwert, also die Anzahl der Sekunden seit 00:00:00 Uhr. Mit Hilfe der `DISPLAY AS TIME` Anweisung kann der so ermittelte Zeitwert in gewünschten Format ausgegeben werden.

Rückgabewert

Die Funktion liefert den berechneten Zeitwert zurück.

Beispiel

```
FIELD f_time = @TIME( 15, 02, 02 ) DISPLAY AS TIME(8);
SELECT f_time, @TIME( 15, 05, 02 ), @15:05:02, @150502;

Die beiden Anweisungen liefern das folgende Ergebnis:
     F_TIME     @TIME(15,05,02)   15:05:02     150502
     15:02:02             54302      54302      54302
```

2.4.3 @DIFFTIME

Aktion

Berechnet die Differenz von zwei Datumswerten in Sekunden.

Syntax

`@DIFFTIME( Datum1, Datum2 )`

Beschreibung

Die Funktion `@DIFFTIME` berechnet die Anzahl von Sekunden zwischen den gegebenen Datumswerten *Datum1* und *Datum2*.

Rückgabewert

Die Funktion liefert die berechnete Differenz zurück. Gilt *Datum2* größer als *Datum1*, ist der Rückgabewert negativ.

Beispiel

```
Datum 1 = @NOW               = 802983720
Datum 2 = @DATE( 93, 7, 22) = 743292000

Der Aufruf von @DIFFTIME liefert mit diesen Argumenten
das folgende Ergebnis
     @DIFFTIME( @NOW, @DATE( 93, 7, 22) ) =  59691720
     @DIFFTIME( @DATE( 93, 7, 22), @NOW ) = -59691720
```

2.4.4 @DAY, @MONTH, @YEAR, @QUARTER, @WEEKDAY

Aktion

Ermittelt den gewünschten Wert aus einem gegebenen Datumswert.

Syntax

```
@DAY( Datum )
@MONTH( Datum )
@YEAR( Datum )
@QUARTER( Datum )
@WEEKDAY( Datum )
```

Beschreibung

Alle Funktionen haben als Argument *Datum* einen Datumswert, also die Anzahl der Sekunden seit 01.01.1970. Die Funktion `@DAY` berechnet die zu diesem Datumswert gehörende Tageszahl, `@MONTH` die entsprechende Monatszahl und `@YEAR` das Jahr. Die Funktion `@QUARTER` liefert das zugehörige Quartal, also einen Wert zwischen 1 und 4. Die Funktion `@WEEKDAY` ermittelt den Wochentag, wobei der Wert 0 den Sonntag repräsentiert, der Wert 1 den Montag, usw. Die Funktion liefert immer einen Wert zwischen 0 und 6.

Rückgabewert

Die Funktionen liefern den jeweils aus dem Datumswert ermittelten Wert.

Beispiel

```
Sei etwa @NOW = 24.08.1993

Dann liefern die Funktionen die folgenden Ergebnisse:

     @DAY(     @NOW ) = 24
     @MONTH(   @NOW ) =  8
     @YEAR(    @NOW ) = 93
     @QUARTER( @NOW ) =  3
     @WEEKDAY( @NOW ) =  2  = Dienstag
```

2.4.5 @HOUR, @MINUTE, @SECOND

Aktion

Ermittelt den gewünschten Wert aus einem gegebenen Datums- oder Zeitwert.

Syntax

```
@HOUR( Datum )
@MINUTE( Datum )
@SECOND( Datum )
```

Beschreibung

Alle Funktionen haben als Argument *Datum* einen Datums- oder Zeitwert, also die Anzahl der Sekunden seit 01.01.1970. Die Funktion `@HOUR` berechnet die zum Argument *Datum* gehörende Stundenzahl, `@MINUTE` die entsprechende Anzahl der Minuten und `@SECOND` die Sekunden.

Die Funktionen arbeiten auch, wenn das übergebene Argument ein Zeitwert ist. Es können jedoch in der aktuellen Version Differenzen auftreten, die ihren Grund in der Zeitzone haben, die in der Arbeitsumgebung festgelegt wurde.

Rückgabewert

Die Funktionen liefern den jeweils aus dem Argument ermittelten Wert.

Beispiel

```
Sei etwa @NOW = 24.08.1993, 20:26:31

Dann liefern die Funktionen die folgenden Ergebnisse:

     @HOUR(   @NOW )  = 20
     @MINUTE( @NOW )  = 26
     @SECOND( @NOW )  = 31
```

2.4.6 @WEEKBEG, @MONTHBEG, @QUARTERBEG, @YEARBEG

Aktion

Ermittelt den gewünschten Datumswert aus einem gegebenen Datumswert.

Syntax

```
@WEEKBEG( Datum )
@MONTHBEG( Datum )
@QUARTERBEG( Datum )
@YEARBEG( Datum )
```

Beschreibung

Alle Funktionen haben als Argument *Datum* einen Datumswert, also die Anzahl der Sekunden seit 01.01.1970. Die Funktion `@WEEKBEG` berechnet den ersten Tag der Woche, die sich aus dem gegebenen Wert *Datum* ergibt. `@MONTHBEG` ermittelt analog den ersten Tag des zugehörigen Monats, `@QUARTERBEG` den entsprechenden Datumswert für den ersten Tag des Quartals und `@YEARBEG` für den ersten Tag des Jahres.

Rückgabewert

Die Funktionen berechnen das jeweils gewünschte Datum und geben den dazugehörigen Datumswert vom Typ `long` zurück.

Beispiel

```
Sei etwa @NOW = 24.08.1993

SET DATE    = "%d.%m.%y";
FIELD w_beg = @WEEKBEG(    @NOW);
FIELD m_beg = @MONTHBEG(   @NOW);
FIELD q_beg = @QUARTERBEG( @NOW);
FIELD y_beg = @YEARBEG(    @NOW);

SELECT w_beg, m_beg, q_beg, y_beg;

Daraus ergeben sich die folgenden Ergebnisse:

     W_BEG     M_BEG     Q_BEG     Y_BEG
     23.08.93  01.08.93  01.07.93  01.01.93
```

2.4.7 @WEEKS, @DAYS, @HOURS, @MINUTES, @SECONDS

Aktion

Berechnen auf Grund einer gegebenen Zahl bzw. eines numerischen Ausdrucks die entsprechende Anzahl von Sekunden.

Syntax

```
@WEEKS( Zahl )
@DAYS( Zahl )
@HOURS( Zahl )
@MINUTES( Zahl )
@SECONDS( Zahl )
```

Beschreibung

Intern werden alle Datums- und Zeitangaben in Sekunden seit dem 01.01.1970 abgespeichert. Dies ist das standardmäßige Datums- bzw. Zeitformat von Unix. Dieses Format macht es nun genauso einfach, mit Datums- und Zeitwerten zu rechnen wie mit normalen Zahlen. Alle Funktionen berechnen eine Anzahl von Sekunden, genauer das Produkt aus der zur Funktion gehörenden Konstanten und dem Wert, der mit dem Argument *Zahl* übergeben wird.

Die jeweiligen Konstanten sind der folgenden Auflistung zu entnehmen:

Funktion	Konstante	
@SECONDS( )	1 Sekunde	
@MINUTES( )	60 Sekunden	(60 * 1)
@HOURS( )	3600 Sekunden	(60 * 60)
@DAYS( )	86400 Sekunden	(24 * 3600)
@WEEKS( )	604800 Sekunden	(7 * 86400)

Rückgabewert

Die Funktionen liefern die berechnete Anzahl von Sekunden zurück.

Beispiel

```
Sei etwa @NOW = 24.08.1993

SET DATE = "%d.%m.%y";
FIELD morgen          = @NOW + DAYS(1);
FIELD vor_zwei_wochen = @NOW - @WEEKS(2);

SELECT morgen, vor_zwei_wochen;

Daraus ergeben sich die folgenden Ergebnisse:

     MORGEN     VOR_ZWEI_WOCHEN
     25.08.93   10.08.93
```

3

Neue Bestandteile von SQL/R

3.1 Kommentare

In der aktuellen Version von **SQL/R** haben Sie die Möglichkeit, in ihre **SQL/R** Skripts beliebige Kommentare einzufügen.

Dabei sind zwei Formen von Kommentaren zu unterscheiden. Zum einen eine einzelne Kommentarzeile und zum anderen ein ganzer Abschnitt des Skripts als Kommentar.

Kommentarzeile

Beginnt eine Zeile mit dem Zeichen „#", so wird sie als Kommentarzeile angesehen und von **SQL/R** bei der Auswertung vollständig ignoriert.

Darüberhinaus bietet die Kommentarzeile die Möglichkeit, **SQL/R** Skripts direkt ausführbar zu machen, und zwar auf die folgende Art und Weise. Steht in der allerersten Zeile eines Skripts die folgende Anweisung, so wird `sqlrexec` automatisch gestartet:

`#! /usr/bin/sqlrexec -n`

Statt des konkreten Starts mit "/usr/bin/sqlrexec -n xxx.sql" reicht die Eingabe des Skriptnamens "xxx.sql" für die Ausführung.

Beispiel

```
# Kommentar
Anweisungen
...
# Kommentar
Anweisungen
...
```

Kommentarabschnitt

Alle Zeichen, die zwischen 2 geschweiften Klammern („{" und „}") stehen, werden als Kommentar angesehen. Damit können Sie ganze Abschnitte eines Skripts als Kommentar kennzeichnen.

Bei der Verwendung dieser Klammern sind zwei Einschränkungen zu beachten:

- Das letzte Zeichen einer Zeile innerhalb eines Kommentarabschnitts darf kein Semikolon („;") sein.
- Nach einem Semikolon, welches das Ende einer Anweisung kennzeichnet, darf der Kommentarabschnitt nicht in derselben Zeile beginnen. **SQL/R** interpretiert das Ende einer Anweisung gleichzeitig als Ende dieser Zeile und ignoriert den Rest, so daß in diesem Fall die geschweifte Klammer „{" verloren geht und der Beginn des Kommentars nicht erkannt wird.

Für eine übersichtliche Gestaltung empfiehlt es sich, die Klammern, die den Beginn und das Ende des Kommentars markieren, in eigene Zeilen zu schreiben und den zugehörigen Text einzurücken.

Beispiel

```
Anweisungen
...
{
    Beginn Kommentartext
    ...
    Ende Kommentartext
}
Anweisungen
...
```

3.2 Verwendung von Umgebungsvariablen

Es besteht die Möglichkeit, in **SQL/R** Skripts die Werte bzw. Inhalte von Umgebungsvariablen (Environment-Variablen) zu verwenden. Eine Umgebungsvariable ist gekennzeichnet durch ihren Namen und einem vorangestellten „$" Zeichen. Auf diese Art und Weise können allgemeingültige Angaben, wie zum Beispiel Datenbankpfade, einmalig festgelegt werden und müssen bei Änderungen nicht mehr in jedem Skript geändert werden.

Beachten Sie, daß Umgebungsvariablen immer als Zeichenketten bzw. Strings angesehen werden, auch wenn sie einen numerischen Wert beinhalten. Es ist daher gegebenenfalls notwendig, die gewünschte Variable in den benötigten Datentyp zu konvertieren.

Beispiel

```
OPEN DATABASE "$HOME/db";
```

Mit dieser Anweisung wird die Datenbank "db" geöffnet. Diese Datenbank befindet sich in dem Verzeichnis, das durch die Environment-Variable `HOME` definiert ist, zum Beispiel `HOME = /usr/sqlr`.

Die folgende Anweisung ist ein Beispiel für eine notwendige Datentypkonvertierung bei der Verwendung der Umgebungsvariablen.

Beispiel

```
SELECT ... FROM ... WHERE firma = @VALUE( $FIRMA );
```

3.3 Die Konstante NULL

Die Konstante `NULL` ermöglicht es, mit Hilfe eines Vergleichs Felder auf NULL-Werte zu untersuchen. Mit NULL-Wert ist dabei nicht der numerische Wert 0 oder bei Zeichenketten ein Leerstring gemeint sondern die Tatsache, daß ein Feld nicht gefüllt bzw. definiert ist. In diesem Fall enthält das Feld immer den Wert NULL.

Beispiel

```
FIELD wert = IF (feldwert = NULL, 0.0, feldwert);
...
SELECT ..., MAX( wert ), ...
```

Mit der `FIELD` Anweisung wird allen NULL-Werten der numerische Wert 0.00 zugeordnet. Dies ist notwendig, da alle arithmetischen Operationen als Ergebnis einen NULL-Wert erhalten, wenn einer der verwendeten Werte ein NULL-Wert ist. Dies würde letztendlich zu fehlerhaften Auswertungen führen.

Der Vergleich eines Feldes mit dem Wert NULL funktioniert nicht in der `WHERE` Regel einer `SELECT` Anweisung. Dies liegt in der internen Auswertung einer `WHERE` Bedingung begründet. In diesem Fall müssen Sie die `IS [NOT] NULL` Regel verwenden.

Beispiel

```
SELECT MAX( feldwert) FROM ... WHERE feldwert <> NULL;
```

Diese Anweisung funktioniert nicht wegen des Vergleichs in der `WHERE` Bedingung. Richtig muß die Anweisung wie folgt lauten.

```
SELECT MAX( feldwert) FROM ... WHERE feldwert IS NOT NULL;
```

3.4 Die Konstante @NOW

Die Konstante `@NOW` repräsentiert den Datumswert des aktuellen Tagesdatums. Der Wert ist vom Typ `long` und beinhaltet die Anzahl der Sekunden seit dem 01.01.1970. Mit Hilfe dieser Konstanten können Auswertungen bis zum aktuellen Tag gemacht werden, ohne das entsprechende **SQL/R** Skript jedesmal ändern zu müssen.

Beispiel

```
SELECT ..., date, ... FROM ...
WHERE @YEARBEG( @NOW ) <= date AND date <= @NOW;
```

Mit dieser Anweisung werden alle Sätze ausgewählt, in denen das Feld "date" zwischen dem 01.01. des aktuellen Jahres und dem aktuellen Tagesdatum liegt.

3.5 Neue SET-Kommandos

3.5.1 SET ECHO

Syntax

`SET ECHO = [ON | OFF]`

Beschreibung

Die Anweisung `SET ECHO` entspricht der Option „-e" beim Aufruf von `sqlrexec`. Wenn `ECHO = ON` gesetzt ist, wird jede **SQL/R** Anweisung an `stderr` geschickt bzw. auf dem Bildschirm ausgegeben, bevor die Anweisung ausgeführt wird.

3.5.2 SET COLSEP

Syntax

`SET COLSEP = "Character"`

Beschreibung

Mit der `SET COLSEP`-Anweisung wird das Zeichen definiert, das von **SQL/R** als Spaltentrenner der einzelnen Spalten bzw. Felder einer Ergebniszeile verwendet werden soll. Standardmaßig ist das Leerzeichen (" ") dieser Spaltentrenner.

Wird für die Ausgabe der Ergebnisse eine ASCII-Datei festgelegt (mit der **SQL/R** Anweisung `SET OUTPUT`) und zusätzlich ein Spaltentrenner definiert, dann wird dieses Trennzeichen innerhalb der ASCII-Datei als Feldtrenner verwendet.

Wurde als Ausgabedatei eine DIF-Datei festgelegt, dann hat der definierte Spalten- bzw. Feldtrenner keine Auswirkung.

Die Definition von Spalten- bzw. Feldtrennern ist oft notwendig, wenn die Ergebnisse von **SQL/R** Anweisungen in anderen Applikationen importiert werden sollen und diese Applikationen entsprechende Bedingungen an Importdateien stellen.

3.5.3 SET ROWSEP

Syntax

```
SET ROWSEP = "Character"
```

Beschreibung

Die `SET ROWSEP`-Anweisung definiert das Trennzeichen, das von **SQL/R** zwischen den einzelnen Ergebniszeilen verwendet werden soll. Standardmäßig ist dieses Zeichen ein Bindestrich („`-`").

Die Definition eines Zeilentrenners hat keine Auswirkung, wenn für die Ausgabe der Ergebnisse eine ASCII- oder DIF-Datei festgelegt wurde.

3.5.4 SET NULL

Syntax

```
SET NULL =  "String" | "Character"
```

Beschreibung

Mit der `SET NULL`-Anweisung wird festgelegt, welche Ausgabe erfolgen soll, wenn ein Feld einen NULL-Wert enthält. Ein NULL-Wert bedeutet, daß das Feld keinen Wert besitzt. Dies ist nicht zu verwechseln mit dem Wert 0 bei numerischen Feldern oder einem Leerstring bei String-Feldern, da dies korrekte Werte sind.

Ein Feld kann zum Beispiel einen NULL-Wert, also nichts, enthalten, wenn der zugehörige Datensatz innerhalb eines mit `CREATE VIEW` erstellten logischen Satzes nicht gefunden wurde.

Wird für die Ausgabe eines NULL-Wertes ein einzelnes Zeichen definiert, dann wird dieses Zeichen so oft ausgegeben, wie das Feld breit ist. Standardmäßig wird bei NULL-Werten der Text "NULL" ausgegeben.

Die Anweisung `SET NULL` hat keine Auswirkung, wenn die Ausgabe in eine ASCII- oder DIF-Datei erfolgt. In diesem Fall wird immer der numerische Wert 0 oder ein Leerstring "" ausgegeben, abhängig vom Datentyp des jeweiligen Feldes.

Beispiel

```
CREATE VIEW tmp PATH kunden
   TO auftrag WHERE kundnr = kundnr
```

Bei allen Kunden, für die keine Aufträge existieren, gibt es keinen zugehörigen Satz `auftrag`. Die entsprechenden Felder in `tmp`, wie zum Beispiel `auftrag.auftrid`, enthalten daher NULL-Werte.

3.5.5 SET OVERFLOW

Syntax

`SET OVERFLOW = "String" | "Character"`

Beschreibung

Mit der `SET OVERFLOW`-Anweisung wird die Ausgabe festgelegt, die erfolgen soll, wenn ein numerischer Wert auf Grund des gewählten Ausgabeformats nicht darstellbar ist. Diese Definition betrifft nur die Ausgabe und hat keine Auswirkung auf den eigentlichen Wert. Wird mit `SET OVERFLOW` ein einzelnes Zeichen definiert, dann wird dieses Zeichen in der gewählten Breite der Ausgabe wiederholt. Wird ein Text festgelegt, so wird dieser abgeschnitten, falls er länger ist als die vorgegebene Ausgabebreite.

Standardmäßig erfolgt in solchen Fällen eine Ausgabe von Sternchen („*") in der gewählten Breite des Feldes.

In früheren Versionen wurde die Ausgabe abgeschnitten, so daß fehlerhafte bzw. unvollständige Ausgaben möglich waren.

Die Anweisung `SET OVERFLOW` hat keine Auswirkung, wenn die Ausgabe in eine ASCII- oder DIF-Datei erfolgt. In diesem Fall wird immer der Wert 0 ausgegeben.

Beispiel

```
SET OVERFLOW = "#";
FIELD val    = 1234567890;
FIELD val1   = val DISPLAY AS LONG(10);
```

```
FIELD val2   = val DISPLAY AS LONG( 6);

SELECT val, val1, val2;
```

Diese Anweisungen führen zu der folgenden Ausgabe:

```
         VAL        VAL1        VAL2
  1234567890  1234567890      ######
```

Mit der folgenden Anweisung ergibt sich entsprechend:

```
SET OVERFLOW = "ZU KURZ GEWÄHLT";

       VAL        VAL1        VAL2
1234567890  1234567890      ZU KUR
```

4

Erweiterungen von SQL/R Anweisungen

4.1 Die REPORT SELECT - Anweisung

Syntax

```
REPORT
      SELECT ...
USING LINEAR LIST;
```

Beschreibung

Die **SQL/R** Anweisung `REPORT SELECT` wurde um die Option `USING LINEAR LIST` erweitert. Bei Verwendung dieser Option wird auf jeder Seite genau ein Ergebnissatz ausgegeben. Dabei werden alle Spaltennamen mit ihrem zugehörigen Ergebniswert untereinander geschrieben.

Die Verwendung dieser Erweiterung empfiehlt sich insbesondere für die Diagnose der Datenbank und während der Entwicklungs- und Testphase von **SQL/R** Skripts.

Beispiel

```
OPEN DATABASE "/usr/sqlr/db/db";

REPORT
   SELECT * FROM KUNDEN
USING LINEAR LIST;
```

Aus dieser Anweisung resultiert die folgende Ausgabeseite. Für jede Ergebniszeile wird eine solche Seite erzeugt.

```
KUNDNR   = "11006"
SUCHNAM  = "ABB"
NAME1    = "ABB-Tecnoservice"
NAME2    = ""
NAME3    = "Postfach 605"
STRPF    = ""
PLZORT   = "USA-LYONS, ILLINOIS 60534"
```

```
TEL      = "KYLLBURG 065632019"
UMSATZ1  = 1457.67
VKGEBIET = "U"
```

4.2 Die FIELD - Anweisung

Syntax

```
FIELD ... DISPLAY AS ...
      [ GROUP OPTION = {ON | OFF} ]
      [ CURRENCY OPTION = {ON | OFF} ]
      [ NULL = {"string" | "char"} ]
      [ OVERFLOW = {"string" | "char"} ]
      [ SUPPRESS REPEATING VALUES [UNTIL GROUP BREAK] ]
```

Beschreibung

Erweiterung	Bedeutung
GROUP OPTION	Diese Option ist nur für numerische Werte gültig. Wenn GROUP OPTION = ON gesetzt ist, wird der Wert gruppiert ausgegeben, wobei das zugehörige Trennzeichen und seine Positionen von der eingestellten Arbeitsumgebung abhängen (z.B. "12.000,00" oder "12,000.00"). Entscheidend für Zahlen vom Typ MONEY ist die Einstellung von LC_MONETARY, für alle anderen Datentypen gilt die Einstellung von LC_NUMERIC. Standardmäßig gilt GROUP OPTION = ON für Werte vom Typ MONEY, während für alle anderen Typen immer GROUP OPTION = OFF gesetzt ist.
CURRENCY OPTION	Diese Option ist nur für Felder vom Typ MONEY gültig. Wenn CURRENCY OPTION = ON gesetzt ist, wird der Wert mit dem zugehörigen Währungssymbol ausgegeben, wobei das Symbol und die Position von der eingestellten Arbeitsumgebung abhängen (z.B. "1.000,00 DM" oder "$ 1,000.00"). Sowohl das Symbol als auch seine Formatierung werden durch LC_MONETARY definiert. Standardmäßig gilt CURRENCY OPTION = OFF.

Erweiterung	Bedeutung
`NULL`	Mit dieser Option wird für das Feld die Ausgabe von NULL-Werten festgelegt, und zwar unabhängig von einer eventuell zuvor definierten globalen Einstellung (Siehe auch 3.5.4).
`OVERFLOW`	Mit dieser Option wird für das Feld die Ausgabe festgelegt, die erfolgen soll, wenn die vorgesehene Ausgabebreite für die Darstellung des Feldes nicht ausreicht. Eine eventuell zuvor festgelegte globale Einstellung (Siehe auch 3.5.5) verliert für dieses einzelne Feld jede Bedeutung.
`SUPPRESS ...`	Diese Option macht es möglich, die Ausgabe von sich wiederholenden Werten in einer Spalte zu unterdrücken. Anstatt des eigentlichen Feldinhaltes werden Leerzeichen ausgegeben. Die normale Ausgabe wird erst wieder fortgesetzt, wenn sich entweder der Feldinhalt ändert oder wenn die `UNTIL GROUP BREAK` Regel festgelegt wurde und ein Gruppenwechsel notwendig wird. Ein solcher Gruppenwechsel wird duch die Anwendung der `CALCULATE ... BREAK` Regel erzwungen (siehe auch Kapitel 6.18 ff.).

Beispiel

Das folgende Beipiel beinhaltet Erweiterungen einzelner `FIELD` Anweisungen des **SQL/R** Skripts `man34` im Kapitel 5.3, Seite 33 ff.

Diese Modifikationen haben die folgenden Auswirkungen:

- Die mehrmalige Ausgabe von Auftragsnummer `auftrnr` und Status `status` wird unterdrückt.
- Der Gesamtbetrag `betrag` wird immer mit 1000er-Trennzeichen und Währungssymbol ausgegeben. Im Falle eines NULL-Wertes erfolgt die Ausgabe des Textes "N/A". Ist die definierte Ausgabebreite zu klein, wird das Zeichen "#" ausgegeben.
- In den numerischen Feldern `auftrmng` und `auftpr` wird bei NULL-Werten ebenfalls der Text "N/A" ausgegeben.

```
FIELD auftrnr  DISPLAY AS (10)
               SUPPRESS REPEATING VALUES
               UNTIL GROUP BREAK;

FIELD status = auftrstatus VALUES ARE ...
                           DISPLAY AS LEFT(18)
                           SUPPRESS REPEATING VALUES
                           UNTIL GROUP BREAK;

FIELD auftrmng DISPLAY AS DOUBLE( 6, 0)
               NULL = "N/A";

FIELD auftrpr  DISPLAY AS DOUBLE( 8, 2)
               NULL = "N/A";

FIELD betrag = (auftrmng * auftrpr / peinheit)
                    DISPLAY AS DOUBLE( 10, 2)
                    GROUP OPTION = ON
                    CURRENCY OPTION = ON
                    NULL = "N/A"
                    OVERFLOW = "#";
```

4.3 Die WHERE - Bedingung

Syntax

`SELECT ... WHERE Ausdruck` **`IS [NOT] NULL`**

Beschreibung

Das Resultat einer `CREATE VIEW` Anweisung ist immer ein logischer, also physikalisch nicht existierender, Datensatz, der aus mehreren "echten" Datensätzen besteht, die über entsprechende Felder und zugehörige Bedingungen miteinander verknüpft sind.

Auf Grund der festgelegten Bedingungen ist es möglich, daß einzelne Datensätze innerhalb des Views nicht definiert und damit leer sind.

Die Erweiterung der `WHERE` Bedingung um die `IS NULL` bzw. die `IS NOT NULL` Regel macht es möglich, solche nicht definierten Datensätze zu verwenden oder auszuschließen.

Beachten Sie, daß die Verwendung der `IS NULL` Regel zu einer geringfügig längeren Verarbeitungszeit führt.

Beispiel

```
CREATE VIEW tmp PATH kunden
    TO auftrag WHERE auftrag.kundnr = kunden.kundnr

SELECT kundnr
FROM tmp
WHERE auftrag.kundnr IS NULL;

Diese SELECT Anweisung erzeugt eine Liste aller Kunden
ohne Aufträge.

...
WHERE auftrag.kundnr IS NOT NULL;

In Gegensatz zur ersten Liste werden hier alle Kunden
mit Aufträgen ausgegeben.
```

4.4 Die CREATE VIEW - Anweisung

Syntax

CREATE VIEW $view\_name$ PATH $quelle$
TO $ziel\_1$ WHERE $ziel\_1.schlüsselfeld$ = **<ausdruck_1>**
AND $ziel\_2$ WHERE $ziel\_2.schlüsselfeld$ = **<ausdruck_2>**

Beschreibung

Die CREATE VIEW Anweisung wurde dahingehend erweitert, daß in der zugehörigen WHERE Bedingung auch der Vergleich mit einem Ausdruck bzw. einer Formel möglich ist. Bis zur **SQL/R** Version A.01.31 war es lediglich möglich, ein einzelnes Datenfeld für die Definition der gewünschten Verknüpfung zu verwenden.

Beachten Sie, daß das Feld aus der Tabelle, mit der eine Verknüpfung hergestellt werden soll, ein Such- oder Indexfeld sein muß.

Beispiel

Im folgendem Beispiel wird zu jedem Datensatz aus `artikel` ein passender Datensatz aus `text` gesucht, dessen Schlüsselfeld `textnr` mit den ersten vier Zeichen der Artikelnummer `artikel.artnr` übereinstimmen muß.

```
CREATE VIEW tmp PATH artikel
    TO text WHERE text.textnr = @LEFT( artikel.artnr, 4);

Alternativ kann auch statt des konkreten Ausdrucks in der
CREATE VIEW Anweisung eine FIELD Anweisung verwendet werden.

FIELD f_textnr = @LEFT( artikel.artnr, 4);

CREATE VIEW tmp PATH artikel
    TO text WHERE text.textnr = f_textnr;
```

A

Aktualisierte Kurzreferenz

A.1 SQL/R Anweisungen

Die **fett** gekennzeichneten Einträge sind neu hinzugekommen.

`CREATE VIEW` view_name `PATH` *table_spec* *path_group*
[`DESCRIBE AS` "description"] ;

table_spec = [table_alias_name =] table_name

path_group = `TO` *path_element* [`AND` *path_element* [`AND` ...]] [`TO` ...]

path_element = { (*path_element* *path_group*) | *table_spec* `WHERE` [table_name.] field = **expression** }

`DEFINE` ["]macro_name["] `AS` "macro definition"
[`DESCRIBE AS` "description"] ;

`EXIT ;`

`FIELD` { alias = expression | field_name }
[`VALUES ARE` ([{"string" | num} =] "string" [, ...])]
[`DISPLAY AS` [[`LEFT` | `CENTER` | `RIGHT`] *format*] [*fmt_option* ...]]
[`DESCRIBE AS` "description"] ;

format = {
(Breite)
(Breite, Nachkommastellen)
INT (Breite)
LONG (Breite)
FLOAT (Breite, Nachkommastellen)
DOUBLE (Breite, Nachkommastellen)
FIXED (Breite, Nachkommastellen)
MONEY (Breite [, Nachkommastellen])
DATE [("date_format", Breite)]
TIME [(Breite)]
}

fmt_option = [
GROUP OPTION = {ON | OFF}
CURRENCY OPTION = {ON | OFF}
NULL = {"char" | "text"}
OVERFLOW = {"char" | "text"}
SUPPRESS REPEATING VALUES [**UNTIL GROUP BREAK**]
]

HELP [{ identifier | "string" }] ;

REPORT SELECT select_stmt
[CALCULATE *field_calc* [, . . .]]
[INTO { TERMINAL | PRINTER | [ASCII | EXPORT | DIF] FILE ["]filename["] | NULL }]
[*report_fmt* | USING report_form | **USING LINEAR LIST**] ;

$$
field\_calc = \left[\left\{ \begin{array}{l} \texttt{SUM} \\ \texttt{AVG} \\ \texttt{MIN} \\ \texttt{MAX} \\ \texttt{COUNT} \end{array} \right\} \text{(field\_ref [, . . .]) ["row label"]} \right]
$$

$$
\texttt{BREAK ON} \left\{ \begin{array}{l} \text{(field\_ref [, . . .])} \\ \texttt{REPORT} \end{array} \right\} \left[\begin{array}{l} \texttt{SKIP [n]} \\ \texttt{PAGE [n]} \end{array} \right]
$$

$report\_fmt$ = [TITLE AS "report title"]

[DATE AS { TODAY | "date string" }]

[LENGTH = num]

[WIDTH = num]

[RUN] file_name [("arg" [, "arg"] . . .)] ;

SELECT [ALL | DISTINCT]

{* | expression ["alternate_heading"] [, . . .] }

[FROM view_name]

[WHERE cond_expression]

[GROUP BY field_ref [, . . .] [HAVING cond_expression]]

[ORDER BY field_ref [ASC | DESC] [, field_ref [ASC | DESC]] . . .] ;

SET LOCALE "$category$=language[@modifier]" ;

$$
category = \left\{ \begin{array}{l} \texttt{LC\_ALL} \\ \texttt{LC\_COLLATE} \\ \texttt{LC\_CTYPE} \\ \texttt{LC\_MONETARY} \\ \texttt{LC\_NUMERIC} \\ \texttt{LC\_TIME} \end{array} \right\}
$$

```
SET OUTPUT = { TERMINAL
             { PRINTER
             { [ ASCII  ]
             { [ EXPORT ] FILE ["]filename["]   } ;
             { [ DIF    ]
             { NULL
```

```
SET { DATE     = "date_fmt"
    { ECHO     = [ON | OFF]
    { COLSEP   = "char"
    { LENGTH   = lines
    { NULL     = ["char" | "string"]
    { OVERFOW  = ["char" | "string"]   } ;
    { PRINTER  = "device"
    { PROMPT   = "prompt string"
    { ROWSEP   = "char"
    { WIDTH    = columns
```

```
SHOW { DATE
     { FIELD { * | field_name }
     { LENGTH
     { LOCALE
     { MACRO { * | "macro_name" }   } ;
     { OUTPUT
     { PRINTER
     { VIEW { * | view_name }
     { WIDTH
```

A.2 Konstanten

Konstante	Bedeutung
`NULL`	repräsentiert leere, nicht definierte Felder
`@date`	repräsentiert den zu *date* gehörigen Datumswert
`@time`	repräsentiert den zu *time* gehörigen Zeitwert
`@NOW`	repräsentiert den Datumswert des aktuellen Datums

A.3 Funktionen

Alle Argumente der aufgeführten Funktionen können Konstanten, einzelne Feldnamen oder Formeln bzw. Ausdrücke sein.

Funktionsname	(**Argumente**)	**Funktionsname**	(**Argumente**)
`@ABS`	(number)	`@NUM`	(string)
`@CHAR`	(number)	`@POS`	(string1, string2)
`@DATE`	(year, month, day)	`@QUARTER`	(date)
`@DATETOCHAR`	(date, format)	`@QUARTERBEG`	(date)
`@DATEVALUE`	(date)	`@RIGHT`	(string, length)
`@DAY`	(date)	`@ROUND`	(number, precision)
`@DAYS`	(number)	`@RPT`	(string, count)
`@DIFFTIME`	(date1, date2)	`@SECOND`	(date)
`@DIV`	(number, divisor)	`@SECONDS`	(number)
`@FRACT`	(number)	`@STRING`	(format, number)
`@HOUR`	(date)	`@SUBSTR`	(string, start, length)
`@HOURS`	(number)	`@TIME`	(hour, minute, second)
`@INT`	(number)	`@TIMEVALUE`	(string)
`@LEFT`	(string, length)	`@TRIM`	(string)
`@LENGTH`	(string)	`@UPPER`	(string)
`@LOWER`	(string)	`@VALUE`	(string)
`@MINUTE`	(date)	`@WEEKBEG`	(date)
`@MINUTES`	(number)	`@WEEKDAY`	(date)
`@MOD`	(number, divisor)	`@WEEKS`	(number)
`@MONTH`	(date)	`@YEAR`	(date)
`@MONTHBEG`	(date)	`@YEARBEG`	(date)

B

Anpassung des Zeichensatzes

SQL/R arbeitet intern (wie auch HP Eloquence) im **Roman8** Zeichensatz. Innerhalb des **SQL/R** Editors und mit Hilfe des Programms `tmap` können jedoch Daten in einen Terminal-abhängigen Zeichensatz umgesetzt werden.

B.1 Wie funktioniert das ?

Mit Hilfe der **`TERM`** Umgebungsvariable wird im Verzeichnis **`/usr/sqlr/map`** eine Datei gesucht, die dem Typ des Terminals mit der Endung **`.map`** entspricht. Wird diese Datei gefunden, wird der Zeichensatz entsprechend der Definition in dieser Datei umgestellt.

Falls die Umgebungsvariable **`SQLR_MAP`** gesetzt ist, werden die Dateien in dem dort angegebenen Verzeichnis gesucht.

zum Beispiel:

Falls die Umgebungsvariable `TERM` den Wert **`70060`** hat, wird die Datei `/usr/sqlr/map/`**`70060`**`.map` zur Anpassung des Zeichensatzes verwendet.

B.2 SQL/R Editor

Der **SQL/R** Editor verwendet automatisch den oben beschriebenen Mechanismus, um alle Ein- und Ausgaben umzusetzen. Falls `sqlrexec` aus dem Editor gestartet wird, werden auch diese Ausgaben umgesetzt.

HINWEIS

> Falls Dateien mit einem anderen Editor als dem **SQL/R** Editor erstellt werden, muß die Datei in den **Roman8** Zeichensatz umgesetzt werden, sofern Umlaute oder Sonderzeichen enthalten sind (z.B. mit `iconv`, welches mit HP-UX ausgeliefert wird). Siehe auch `iconv`(1).

B.3 tmap

tmap ist ein Hilfsprogramm, welches mit **SQL/R** zur Verfügung gestellt wird, um den Zeichensatz bei Terminalausgaben anzupassen.

tmap verwendet den oben beschriebenen Mechanismus, um die Ausgaben terminalabhängig umzusetzen.

Aufruf:

```
/usr/sqlr/tmap [Datei]
```

Die Eingabe erfolgt entweder über `stdin` (Standard Eingabe) oder aus der angegebenen Datei.

z.B.:

```
/usr/sqlr/tmap README
```

Gibt die Datei README im Zeichensatz des Terminals aus.

```
/usr/bin/sqlrexec -tn xxx.sql | /usr/sqlr/tmap
```

Wandelt die Ausgabe von sqlrexec in den Zeichensatz des aktuellen Terminals um. (In diesem Fall muß das **-t** Argument angegeben werden, damit die Pause beim Seitenwechsel erhalten bleibt.)

B.4 iconv

Mit Hilfe des HP-UX Programms iconv kann der Zeichensatz von Dateien umgesetzt werden.

z.B.

```
iconv -f roman8 -t iso8859_1 README
```

Wandelt die Datei README bei der Ausgabe vom Roman8 Zeichensatz in den ISO 8859-1 Zeichensatz um. Die Ausgabe erfolgt auf stdout.

Weitere Informationen finden Sie unter `iconv`(1) mit

```
man 1 iconv
```

B.5 sqlrexec

Die Ausgaben von `sqlrexec` erfolgen im **Roman8** Zeichensatz. Falls die Ausgabe auf einem Gerät mit abweichendem Zeichensatz erfolgen soll, muß sie entsprechend umgesetzt werden (z.B. mit `iconv` oder `tmap`).

z.B.

```
sqlrexec -tn xxx.sql | /usr/sqlr/tmap
```

Wandelt die Ausgabe von sqlrexec in den Zeichensatz des aktuellen Terminals um. (In diesem Fall muß das `textbf-t` argument angegben werden, damit die Pause beim Seitenwechsel erhalten bleibt.)

```
sqlrexec -n xxx.sql | tr -d "\014" |
 iconv -f roman8 -t iso8859_1 | more
```

Wandelt die Ausgabe von sqlrexec vom **Roman8** Zeichensatz in den **ISO 8859-1** Zeichensatz um. Die Ausgbe erfolgt über `more`. (Der Seitenwechsel wird hier mit `tr` entfernt).

B.6 Unterstützung abweichender Terminal-Namen

Um abweichende Terminal-Namen mit **ISO 8859-1** Zeichensatz zu unterstützen, linken Sie die Datei **`iso8859_1.map`** auf den Terminal-Namen.

z.B.

```
cd /usr/sqlr/map
ln iso8859_1.map hp70060.map
```

C

Verwendung des Terminaldruckers (lprint)

C.1 verwendung von lprint

Dies ist die Beschreibung von **lprint** in der Version vom 23.06.93. Es ermöglicht die Ausgabe auf den Terminaldrucker. Es arbeitet ähnlich wie die PRINTER IS 10 Anweisung in HP Eloquence.

Aufruf:

```
lprint [-r] [file]
```

Es kann entweder ein Dateiname angegeben werden oder die Daten können an stdin übergeben werden. Die **-r** Option setzt den **raw** Druckmodus. Normalerweise wird ein Newline Zeichen (`\n`) in eine `\r\n` Sequenz umgesetzt. Dies kann durch Angabe der **-r** Option verhindert werden.

Zum Beispiel:

```
ll | lprint
lprint -r sample.DATA
```

C.2 Bemerkungen zur Implementation (und Anpassung) von lprint

1. Der Typ des Terminals, welches `lprint` verwendet, wird mit Hilfe der Variable `TERM` ermittelt. Ein entsprechender terminfo Eintrag muß existieren:

   - Falls die Variable `TERMINFO` nicht gesetzt ist:
     Es wird zunächst im Verzeichnis `/usr/eloquence/terminfo` und dann unter `/usr/lib/terminfo` gesucht.

- Falls die Variable `TERMINFO` gesetzt ist (z.B. /usr/uap/terminfo):

  Es werden zunächst die Verzeichnisse aus der Variable `TERMINFO` bearbeitet (z.B. /usr/uap/terminfo). Falls dort nichts gefunden wird, werden die Verzeichnisse `/usr/eloquence/terminfo` und dann `/usr/lib/terminfo` geprüft.

2. Falls der LONG Terminal Name mit „`hp` " beginnt, wird das HP Drucker Protokoll vorausgesetzt (Rückmeldung S/U/F), sonst keines.

   z.B.

```
70092|70092a|70092A|hp 7009x/239x series,
                    ^^^
                    Dies ist ein HP Terminal

70060|70060 Terminal(vt320; 7 bit),
      ^^^
      Dies ist kein HP Terminal

c1003|c1003a|1003|1003a|700-41,
                        ^^^
                        Dies ist kein HP Terminal
```

3. Die Ansteuerung eines lokalen Druckers benötigt die folgende(n) Steuersequenz(en) (aus terminfo):

   Entweder

```
prtr_non    (aktivieren des Druckers für n Zeichen)
```

   Oder

```
prtr_on     (aktivieren des Druckers)
prtr_off    (abschalten des Druckers)
```

   In der terminfo Beschreibung sind dies die folgenden Schlüsselwörter:

Name	Schlüsselwort	700/92	vt320	c1003
prtr_non	mc5p	mc5p=\E&p%p1%dW		
prtr_on	mc5	mc5=\E&p13C	mc5=\E[5i	mc5=^R
prtr_off	mc4	mc4=\E&p11C	mc4=\E[4i	mc4=^T

Index