SQL/R

Report Generator for HP ELOQUENCE

The information contained in this document is subject to change without notice.

Marxmeier Softwareentwicklung (mse) makes no warranty of any kind with regard to this material, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. Marxmeier Softwareentwicklung shall not be liable for errors contained herein or for incidental or consequential damages in connection with the furnishing, performance or use of this material.

Published Editions:
A.01.00 - 1992

© 1992-1995 Marxmeier Softwareentwicklung, Wuppertal, Germany.

This document contains information which is protected by copyright. All rights are reserved. Reproduction, adaption or translation without prior written permission is prohibited, except under the copyright laws.

Restricted Rights Legend. Use, duplication or disclosure by the U.S. Government Department of Defense is subject to restrictions as set forth in subparagraph (c) (1) (ii) of the Rights in Technical Data and Computer Software clause in DFARS 252.227-7013. Rights for non-DOD Government Departments and Agencies are set as forth in the Commercial Computer Software Restricted Rights clause, FAR 52.227-19 (c) (1,2).

HP ELOQUENCE is a protected trademark of Hewlett-Packard GmbH.

HP-UX is a protected trademark of Hewlett-Packard Inc.

Preface

This manual is divided into the following chapters:

Chapter 1	Introduction and general information about **SQL/R**. Here you can find brief descriptions and instructions for installation.
Chapter 2	Not included in this version.
Chapter 3	A quick overview of **SQL/R** using examples to present the functions and use of **SQL/R**.
Chapter 4	Describes the editor program included with **SQL/R**.
Chapter 5	Explains in detail the various function used to generate reports.
Chapter 6	Defines the syntax and use of the **SQL/R** language.
Appendix A	Short reference guide of the **SQL/R** language.
Appendix B	Date and time formats.
Appendix C	Description of differences between **SQL/R** and SQL.
Appendix D	Environment Setup.
Appendix E	HP Eloquence format numbers.
Glossary	Explanation of terms.
Index	Key word index.

Typographical Conventions

Unless otherweise noted, this manual uses the following symbolic conventions:

`Computer Font` — Computer font indicates commands, keywords, options, literals, source codes, system outputs and path names.

[ ] — The symbol [ ] indicates a key on a computer keyboard or an area or "button" on screen that can be activated by your mouse. For example, [CTRL] indicates the Control key and [Continue] is an on screen button.

[CTRL]-[char] — The symbol [CTRL]-[char] indicates a control character. For example [CTRL]-[Y] means you have to simultaneous press the Control key and the `Y` key on the keyboard.

italics — Within syntax statements, a word in italics represents a formal parameter or argument that you have to replace with an actual value. In the following example, you must substitute *filename* by the name of the file to be printed:

`lp` *filename*

[ ] — Within syntax statements, brackets enclose optional elements. In the following example, brackets around `[-d`*dev*`]` indicate that the parameter and its delimiter are optional:

`lp [-d`*dev*`]` *filename*

{ } — Within syntax statements, braces indicate that you must choose one of the listed items. In the following example, the braces around {`-c`|`-x`|`-v`} indicate, that you must choose one othe the arguments:

`tar {-c|-x|-v}`

Additional Reading

The following additional documentation is referred to in this manual:

HP-UX (online) Documentation

References of the form `services`(4) refer to the given topic or item (here `services`) contained in the indicated section (here 4) of the HP-UX-reference manual. It is also possible to obtain this documentation on-line using the command `man`, whereby in the case of `services`(4) the user should enter the following statement:

```
man 4 services
```

Contents

1 Read This First **1**

1.1 Welcome 1

1.2 Requirements 2

1.3 Installation and Update 3

1.4 List of Files 4

1.5 Ordering 4

1.6 Software Support Contract 5

3 Introduction **6**

3.1 How to start **SQL/R** 8

3.1.1 How to Use the **SQL/R** Editor 8

3.1.2 Loading of a Sample Report File 9

3.1.3 Execution of the Examples 10

3.1.4 Termination of **SQL/R** 11

3.2 Specification of Instructions 12

3.3 Opening the Database 13

3.4 Selection of all Items from a Table 14

3.5 Selection of a Subset of Items from a Table 15

3.6 SELECT with WHERE 16

3.7 SELECT with AND 17

3.8 SELECT with OR 18

3.9 SELECT with IN 20

3.10 SELECT with BETWEEN 21

3.11 Sorting with ORDER BY 22

3.12 SELECT with DISTINCT 24
3.13 SELECT with String Constants 25
3.14 SELECT with arithmetic expressions 26
3.15 SELECT and Functions 27
3.16 SELECT with LIKE 29
3.17 SELECT with GROUP BY 32
3.18 GROUP BY with HAVING 32
3.19 The next step 34

4 Editor **35**

4.1 Keys for text processing 35
4.2 The Menu Structure 38
4.3 Main Menu Bar 39
4.3.1 SQL/R Start (f4) 39
4.3.2 Shell (f5) 40
4.3.3 Info (f6) 40
4.3.4 Exit Program (f8) 40
4.4 File Management 42
4.4.1 Read File 42
4.4.2 Import File 42
4.4.3 Save File 43
4.5 Text Block Management 45
4.5.1 Mark Block 45
4.5.2 Copy Block 45
4.5.3 Delete Block 45
4.5.4 Insert Block 46
4.5.5 Save Block 46

4.6 Search and Replace 47
4.6.1 Search 48
4.6.2 Replace 48
4.6.3 Global Replace 49

5 The Usage of SQL/R **51**
5.1 An Easy List of Customers 52
5.1.1 Opening the Database 52
5.1.2 Selecting Items from a Table 53
5.1.3 Formatting the Output without a Form File 55
5.1.4 Formatting the Output with a Form File 62
5.1.5 Using **SQL/R** and Parameters from the Shell 66
5.2 List of Customers Grouped by Sales Volume 69
5.3 Use of Multiple Tables 75
5.4 Summary 84

6 Reference **85**
6.1 Starting of **SQL/R** 86
6.2 Definition of Terms 88
6.3 Reserved Words 90
6.4 Data Types 91
6.5 Identifiers 93
6.6 Constants 93
6.6.1 Numeric Constants 93
6.6.2 Character String Constants 94
6.6.3 Date Constants 94
6.6.4 Time Constants 94

6.7 Arithmetic Expressions 95
6.7.1 Arithmetic Functions 96
6.7.2 Date Functions 97
6.8 String Expressions 98
6.8.1 String Functions 99
6.9 Condition Functions 99
6.10 Conditional Expressions 100
6.11 The CLOSE DATABASE Command 102
6.12 The CREATE VIEW Command 103
6.13 The DEFINE Command 106
6.14 The EXIT Command 107
6.15 The HELP Command 107
6.16 The FIELD Command 108
6.16.1 FIELD and Expression Pseudonyms 108
6.16.2 The VALUES ARE Rule 109
6.16.3 The DISPLAY AS Rule 110
6.17 The OPEN DATABASE command 112
6.17.1 Multiple Databases 112
6.17.2 The QIF File 112
6.18 The REPORT Command 114
6.18.1 The CALCULATE Rule and the BREAK ON Rule 114
6.18.2 Output Devices 116
6.18.3 Number of Lines per Page 116
6.18.4 Output Width 116
6.18.5 Output Format 117
6.18.6 The Use of Form Files 118
6.19 The RUN Command 121

6.20 The SELECT Command 123
6.20.1 The DISTINCT Rule 123
6.20.2 The FROM Rule 123
6.20.3 The WHERE Rule 124
6.20.4 The GROUP BY Rule 124
6.20.5 The HAVING Rule 125
6.20.6 The ORDER BY Rule 125
6.21 SET Commands 126
6.21.1 SET DATE 126
6.21.2 SET LENGTH 126
6.21.3 SET LOCALE 126
6.21.4 SET OUTPUT 127
6.21.5 SET PRINTER 127
6.21.6 SET WIDTH 128
6.22 SHOW Commands 129
6.22.1 SHOW DATE 129
6.22.2 SHOW FIELD 129
6.22.3 SHOW LENGTH 129
6.22.4 SHOW LOCALE 130
6.22.5 SHOW MACRO 130
6.22.6 SHOW OUTPUT 130
6.22.7 SHOW PRINTER 130
6.22.8 SHOW VIEW 131
6.22.9 SHOW WIDTH 132

A Quick Reference Guide **133**

B Date and Time Formats 136

C Differences between SQL/R and standard SQL 139

D Work Environment 140

E HP Eloquence Format Numbers 144

F Glossary 146

1 Read This First

1.1 Welcome

Welcome to **SQL REPORT** (**SQL/R**), the Report Generator for HP ELOQUENCE.

SQL/R is an extension of HP ELOQUENCE that allows you to create reports and formatted listings without being restricted to simple calculations.

The following list shows some main features of **SQL/R**.

- simultaneous access to different databases
- support of index items
- searching for and sorting on all kinds of items
- using a language related to SQL standards
- calculated items
- support of format definition files

1.2 Requirements

Prerequisites to successfully use **SQL/R** are:

- HP 9000 Series 800
- HP-UX Release 7.0 or later
- HP ELOQUENCE Version A.03.10 or later
- about 2 MB free disk space in the filesystem /usr
- DDS tape drive (1.3 GB)

SQL/R is available in two versions:

1. An evaluation copy that can be used for one month.

   If you decide to purchase a perpetual license, you will receive a password along with the license that changes the evaluation copy to a timely unlimited version.

2. A perpetual version that can only be used with the computer for which it was ordered.

Both versions come with the same material and do not differ in functionality.

The product is shipped on 60m DDS Cassette (1.3 GB) in tar format. Other media are available on request. Please contact your sales representative.

1.3 Installation and Update

This section describes how to install **SQL/R**. A list of all files together with a brief description can be found in the next section.

Prerequisites for installation:

- The password from your software license sheet to install a perpetual version of **SQL/R**
- HP-UX superuser (root) login

1. log on as `root` into your system.
2. insert the DDS Cassette containing **SQL/R** software.
3. change into the directory /tmp by typing this command:
   ```
   cd /tmp
   ```
4. execute the following tar command:
   ```
   tar -xv
   ```
5. change into the directory /tmp/sqlr by typing this command:
   ```
   cd sqlr
   ```
6. start the installation utility by typing this command.
   ```
   ./install
   ```

The installation utility displays further instructions.

1.4 List of Files

SQL/R software consists of the following files:

File/ Directory	Path	Description
sqlr	/usr/bin/	**SQL/R** main program
sqlred	/usr/bin/	**SQL/R** editor
sqlrexec	/usr/bin/	**SQL/R** execution modul
install	/usr/sqlr/	installation utility
installg	/usr/sqlr/	german installation utility
installe	/usr/sqlr/	english installation utility
sqlrbrand	/usr/sqlr/	subprogram for installation
C/sqlr.cat	/usr/lib/nls/	message catalog (default)
german/sqlr.cat	/usr/lib/nls/	message catalog (german)
db.g/	/usr/sqlr/	directory with (german) sample database
sample.g/	/usr/sqlr/	directory with (german) examples
db.e/	/usr/sqlr/	directory with (english) sample database
sample.e/	/usr/sqlr/	directory with (english) examples

1.5 Ordering

If you decide to purchase a perpetual license for **SQL/R**, you will receive a password along with the license sheet that allows you to change the evaluation copy into version. The price of the evaluation copy will be credited to the perpetual license.

To process your order, we need the serial number (also referred as SID - software ID) of your computer. To display your SID please type the following HP-UX command:

```
uname -i
```

1.6 Software Support Contract

We also offer a software support contract for **SQL/R**. Please contact your sales representative.

The support contract grants you access to our hotline, free-of-charge patches and bug fixes. You will be offered new releases under special update conditions.

3

Introduction

This chapter will give a brief overview of the **SQL/R** functionality and usage. It is recommended for novice and new users. The way **SQL/R** works in general is demonstrated by using these examples.

These examples are based on the table CUSTOMERS of the sample database. CUSTOMERS contains the following items:

Item	Description	Data type
CUSTNO	Customer number	STRING[6]
MATCHCODE	Search criteria	STRING[10]
NAME1	Customer name	STRING[32]
NAME2		STRING[32]
NAME3		STRING[32]
STREET	Street / Postbox	STRING[32]
ZIPCITY	ZIP Code and City	STRING[32]
PHONE	Phone number	STRING[18]
TURNOVER	Turnover (month, year, prev.year)	REAL(3)[1]
SALESAREA	Sales area	STRING[6]

All examples used in this chapter can be found in directory `/usr/sqlr/sample`. The corresponding filename is printed in the right margin of a page and starts with the term `tut` followed by a number.

The sample data base is stored in `/usr/sqlr/db` directory and is named `DB`.

To execute the examples `tut`*xx*, it is necessary to change to the directory `/usr/sqlr/sample` by using the following HP-UX command:

[1] Item $TURNOVER$ is an array with the following 3 elements:

- current month (month-to-date): turnover[0]
- current year (year-yo-date): turnover[1]
- previous year: turnover[2]

An array element is always accessed with the help of an index. Please note that the index count starts with 0, i.e. you retrieve the n-th element by specifying an index value of n-1.

```
cd /usr/sqlr/sample
```

3.1 How to start SQL/R

You can start **SQL/R** by typing the following

```
sqlr [filename]
```

at the HP-UX shell prompt. If you specify a filename (e.g. `tut02`) along with the above command, the file contents will be loaded immediately.

All messages of the editor utility and the labels of the function keys depend on the value of `LANG` environment variable. The text shown here assumes the variable to be set to `LANG=american` (⟶ Appendix D).

3.1.1 How to Use the SQL/R Editor

Entering the `sqlr` command calls the **SQL/R** editor. This chapter describes the functions of **SQL/R** and shows you how to try a few examples. For more detailed description on **SQL/R** editor, see chapter 4 “Editor”.

All input is entered at the current cursor position. If the text to be entered is longer than the screen display, the line will be shifted left as you enter more text. An inverse [!] exclamation mark appears as the last character in the right margin of the line, if the remainder of the line is outside the current display.

The following keys can be used to move the cursor on the screen and also modify the displayed text:

[→] Move cursor one character to the right. If used at the end of a line, the cursor moves to the beginning of the next line.

[←] Move cursor one character to the left. If used at the beginning of a line, the cursor moves to the end of the previous line.

[↑] Move the cursor up one line until it reaches the first line.

[↓] Move the cursor down one line until it reaches the last line.

[CTRL] [A] CTRL A. Move cursor to the first position of the current line.

CTRL E — CTRL E. Move cursor one position beyond the last character of the current line.

↩ — RETURN moves the cursor to the first position of the next line. If RETURN is used before the last character, the remainder of the sentence is moved down to the next line at the first position.

⟸ — BACKSPACE. Erase character before current cursor position. If the cursor is at the beginning of a line, this line will be attached to the previous one.

DEL CHAR — Delete character at the current cursor position and shift the remainder of the sentence one character to the left. If the cursor is at the end of a line, the next line will be attached to the current line's end.

CLR LINE — The line is erased from the current cursor position to the end of the line. If the cursor is at the first character position the entire line is deleted. If the cursor is at the end of the line, the next line will be appended to this line.

BREAK — The current activity or program will be aborted. The user will be prompted before the activity or program aborts.

CTRL L — CTRL L. The screen display is refreshed.

CTRL W — CTRL W. The display width is toggled between 80 and 132 characters per line. This feature is supported for terminal types 700/92, 700/94, 700/96 and 700/98.

3.1.2 Loading of a Sample Report File

To display the file function keys, press

To load a text file, press

If there is text currently in the editor work space that was changed, the message appears:

```
[...]  has been modified. Save text ? (y/n)
```

During the exercises with the examples it is not necessary to save the files. Therefore enter `n` for No when the above question appears.

Then the next text file can be retrieved. The following prompt appears:

```
Enter filename:
```

Enter the name of the file to be retrieved (e.g. `tut02`).

To return the the main menu, press

```
f8
MAIN
MENU
```

3.1.3 Execution of the Examples

SQL/R is not case sensitive, i.e. it does not differentiate between lower case letters and upper case letters. Therefore it is not necessary to enter the examples in the case printed in the manual. For your convenience and for better readability of the examples, however, you will find all words that are part of the **SQL/R** language in upper case letters. Item and table names consist of lower case letters.

Instructions can be split across lines and should be terminated with a semicolon (`;`). To enclose strings you can use single as well as double quotes; however, the string must begin and end with the same type of quote mark.

After entering one of the following examples, you can start execution by hitting function key:

```
f4
SQLR
Start
```

During execution of the instructions the following message appears:

```
working ...
```

The results are displayed and can be reviewed. If the number of lines exceeds one screen display, the last line will say:

```
-- press <return> to continue or q <return> to quit:
```

To view the next page of the results, press the RETURN key:

To terminate the display of results and return to the editor, press the following two keys in sequence:

q and ↩

When you are back in the editor, you can work on additional examples in the introduction.

3.1.4 Termination of SQL/R

To terminate your **SQL/R** session, press function key:

f8 EXIT

The following message appears directly above the function key menu:

```
[...]  has been modified. Save text ? (y/n)
```

When practicing with the examples, it is not necessary to save your changes, so enter n for no when prompted with this question.

3.2 Specification of Instructions

SQL/R contains keywords that are used in connection with item and table names to build up command statements.

Here are some of the keywords introduced in this chapter:

`SELECT`	selects the items to be retrieved. An asterisk indicates that all items in a table are to be retrieved.
`FROM`	specifies the 'source' of the items
`WHERE`	specifies the selection conditions
`ORDER BY`	defines the sort order
`GROUP BY`	groups data for further processing

The `SELECT` instruction is the most important command of **SQL/R**. It can be used to access all data of a data base.

The syntax of the select command is shown here:

`SELECT` *what* `FROM` *source* `WHERE` *condition*

what	a list of items or formulas
source	the name of a table containing the data
condition	data selection criteria

The following example displays the items *custno* and *name1* from the table *customers* if the item *matchcode* is equal to "KELLER". Note that you may use items within selection condition that aren't displayed as results (in our example: *matchcode*).

```
SELECT custno, name1
FROM customers
WHERE matchcode = "KELLER";
```

The search results consist of a header line followed by the relevant data lines:

```
KUNDNR  NAME1                        <-- header line
33007   KELLER, ERNST                <-- result (data) line
23062   Keller, Ihne & Tesch KG
11036   OSKAR KELLER
       <---------------------------- column (item)
```

3.3 Opening the Database

Before you can access a database, you must open it. This is done with the `OPEN DATABASE` command.

The sample database used here is called `db`. In addition to specifying the name, it is also necessary to also specify the path name of the directory where the database resides.

Starting from directory `/usr/sqlr/sample` the database can either be accessed by its relative path and name: `../db/db`, or with its absolute path and name: `/usr/sqlr/db`. Therefore your first command should be:

```
OPEN DATABASE "../db/db";
```

The name and path of the database is always specified within quotes.

3.4 Selection of all Items from a Table

Input:

```
OPEN DATABASE "../db/db";

SELECT * FROM customers;
```

tut02

Result:

```
CUSTNO MATCHCODE NAME1                        NAME2
21101    RAUT    TRAUTWEIN HERNE GMBH & CO
31003   1AFIOS   WAFIOS MASCHINENFABRIK
13002   29037    SIEMENS AG                   ABT. ZFELB 23
  .       .        .                            .
  .       .        .                            .
  .       .        .                            .
15046   ZUMTOB   ZUMTOBEL GMBH                LICHT
17054   ZÖLZER   HZV-SPORT, HORST ZÖLZER
```

In the example, the * (asterisk) specifies that all items in the table "customers" are selected. The FROM specifies which dataset (of the database) or table contains the data. The result of the above commands is a list of all data entries (records) of the table *customers* with all the items of each record. (The result example listed here is only a subset and does not contain all the columns and rows).

3.5 Selection of a Subset of Items from a Table

Input:

```
OPEN DATABASE "../db/db";

SELECT custno, name1, name2 FROM customers;
```

tut03

Result:

```
CUSTNO  NAME1                         NAME2
21101   TRAUTWEIN HERNE GMBH & CO
31003   WAFIOS MASCHINENFABRIK
13002   SIEMENS AG                    ABT. ZFELB 23
  .       .                             .
  .       .                             .
  .       .                             .
15046   ZUMTOBEL GMBH                 LICHT
17054   HZV-SPORT, HORST ZÖLZER
```

In this example, only the CUSTNO, NAME1, and NAME2 of each record are displayed. The subset of items is defined by listing the items, separated by commas in the `SELECT` command.

3.6 SELECT with WHERE

Input:

```
OPEN DATABASE "../db/db";

SELECT custno, matchcode, name1
FROM customers
WHERE matchcode = "KELLER";
```

tut04

Result:

```
CUSTNO  MATCHCODE  NAME1
33007   KELLER     KELLER, ERNST
23062   KELLER     Keller, Ihne & Tesch KG
11036   KELLER     OSKAR KELLER
```

Now we will use the `SELECT ... WHERE` command. This allows you to retrieve only those data records that satisfy a given condition (in the above example: matchcode equals “KELLER”). The condition may contain boolean operators such as `AND`, `OR`, `NOT`; relational operators such as =, <, <=, >, >=, <> and language specific operators such as (`LIKE`, `IN`, `BETWEEN`).

The following examples will the usage of these complex conditions.

String values <u>must</u> be enclosed in quotes. Numeric values for calculations, as well as date and time values <u>do not</u> use quotes !

3.7 SELECT with AND

Input:

```
OPEN DATABASE "../db/db";

SELECT custno, matchcode, name1, name2
FROM customers
WHERE custno > "11000" AND custno < "12000";
```
tut05

Result:

```
CUSTNO  MATCHCODE  NAME1                           NAME2
11001   GROZ-B     GROZ-BECKERT                    NADELFABRIKEN
11002   ESJOT      ESJOT SCHUHTECHNIK
  .       .          .
  .       .          .
  .       .          .
11044   WERKST     WZB WERKSTATT FÜR BEHINDERTE
11045   WESTLA     Westland Gummiwerke GmbH & Co
```

This example shows the selection of two combined conditions with the keyword AND. Only those data records are selected where customer number is greater than 11000 *and* is smaller than 12000.

3.8 SELECT with OR

Input:

```
OPEN DATABASE "../db/db";

SELECT custno, matchcode, zipcity
FROM customers
WHERE (matchcode = "KELLER" OR matchcode = "FICHTE") AND zipcity > "73";
```
tut06

Result:

```
CUSTNO  MATCHCODE   ZIPCITY
38004   FICHTE      8646 Nordhalben
29030   FICHTE      8641 Marktrodach
32006   FICHTE      8626 Michelau
33007   KELLER      7300 ESSLINGEN
```

In the next example, we use the boolean operator `OR` in addition to the operator `AND`. The records retrieved contain a matchcode value of either “KELLER” *or* “FICHTE” *and* a zip code value greater than “73”. The parentheses change the sequence of evaluating the conditions. It is very important to correctly use parentheses to obtain the desired results. Changing the location of the parentheses can change the results.

Now enter the following instructions:

```
OPEN DATABASE "../db/db";

SELECT custno, matchcode, zipcity
FROM customers
WHERE matchcode = "KELLER" OR matchcode = "FICHTE" AND zipcity > "73";
```
tut07

The results are identical to those retrieved using these commands:

```
OPEN DATABASE "../db/db";

SELECT custno, matchcode, zipcity
FROM customers
WHERE matchcode = "KELLER" OR (matchcode = "FICHTE" AND zipcity > "73");
```

Result:

```
CUSTNO  MATCHCODE   ZIPCITY
38004   FICHTE      8646 Nordhalben
29030   FICHTE      8641 Marktrodach
32006   FICHTE      8626 Michelau
33007   KELLER      7300 ESSLINGEN
23062   KELLER      7297 ALPIRSBACH
11036   KELLER      7293 PFALZGRAFENWEILER
```

This example lists all data records with either

matchcode = “KELLER”

or

matchcode = “FICHTE” *and* zipcity > “73”.

The condition *zipcity* > “73” is only relevant for those data records that have a matchcode value equal “FICHTE”.

3.9 SELECT with IN

Input:

```
OPEN DATABASE "../db/db";

SELECT custno, matchcode, zipcity
FROM customers
WHERE matchcode IN ("KELLER", "FICHTE", "KLÖCKN");
```

tut08

Result:

```
CUSTNO  MATCHCODE   ZIPCITY
33007   KELLER      7300 ESSLINGEN
23062   KELLER      7297 ALPIRSBACH
11036   KELLER      7293 PFALZGRAFENWEILER
38004   FICHTE      8646 Nordhalben
29030   FICHTE      8641 Marktrodach
32006   FICHTE      8626 Michelau
22032   KLÖCKN      7200 TUTTLINGEN
16037   KLÖCKN      7200 TUTTLINGEN
22020   KLÖCKN      7186 BLAUFELDEN
23065   KLÖCKN      7186 BLAUFELDEN
17046   KLÖCKN      7156 WÜSTENROT 1
11038   KLÖCKN      7151 AFFALTERBACH
22033   KLÖCKN      7141 Benningen
```

The keyword `IN` is used to search for data records with a list of possible values. The values are separated with a comma and the list is enclosed in parentheses. The `IN` operator can be used is most cases as a replacement for the `OR` operator.

Therefore the instuctions shown above can also be written as shown here:

```
OPEN DATABASE "../db/db";

SELECT custno, matchcode, zipcity
FROM customers
WHERE matchcode = "KELLER" OR matchcode = "FICHTE" OR matchcode = "KLÖCKN";
```

3.10 SELECT with BETWEEN

Input:

```
OPEN DATABASE "../db/db";

SELECT custno, matchcode, zipcity
FROM customers
WHERE zipcity BETWEEN "7000" AND "7100";
```

tut10

Result:

```
CUSTNO  MATCHCODE   ZIPCITY
21004   KOPEMA      7090 ELLWANGEN/JAGST
17007   KORALL      7080 AALEN
  .       .           .
  .       .           .
  .       .           .
26009   KÄSBOH      7000 STUTTGART 10
24009   KÖLLI       7000 STUTTGART 1
```

`BETWEEN` *val1* `AND` *val2* is used for searching within a given range of values. The two values *val1* and *val2* are part of the range.

3.11 Sorting with ORDER BY

Up to now we have only selected data records. The retrieved data has been displayed in the same sequence as found in the table (data set). Normally, you would format the results using the `ORDER BY` command. For example:

Input:

```
OPEN DATABASE "../db/db";

SELECT custno, matchcode, zipcity
FROM customers
WHERE zipcity BETWEEN "7200" AND "7300"
ORDER BY matchcode, zipcity;
```

tut11

Result:

```
CUSTNO  MATCHCODE   ZIPCITY
11036   KELLER      7293 PFALZGRAFENWEILER
23062   KELLER      7297 ALPIRSBACH
  .        .           .
  .        .           .
  .        .           .
22032   KLÖCKN      7200 TUTTLINGEN
16037   KLÖCKN      7200 TUTTLINGEN
```

This example uses the items *matchcode* and *zipcity* for sorting. Multiple level sorts are possible by specifying several items for the sort. The position of the item within the `ORDER BY` command determines the sequence for the sort. The results are sorted by the order in which the sort items are listed (i.e. the first item defines the primary sort, etc). In the example, *matchcode* is the primary value. For identical values of *matchcode*, the *zipcity* value is used as the secondary sort value.

The keywords `ASC` and `DESC` define whether the data should be sorted in ascending or descending order. Ascending is the default, i.e. if no additional keyword is used then `ASC` is assumed. The following example uses a descending order. Note that instead of an item name, it uses the column number to specify the sort criteria.

Input:

```
OPEN DATABASE "../db/db";

SELECT custno, matchcode, turnover[1]
FROM customers
WHERE zipcity BETWEEN "7200" AND "7300" AND turnover[1] > 0
ORDER BY 3 DESC;
```

tut11a

Result:

```
CUSTNO  MATCHCODE       TURNOVER[1]
26039   KIERCH             98602.02
17040   KEWEST             95550.39
  .       .                   .
  .       .                   .
  .       .                   .
20012   KLAFFEI             4667.70
```

The result was sorted by *turnover[1]* in descending order. The “3” specifies that the values of the third item of the `SELECT` command *turnover* are used for the sort order.

3.12 SELECT with DISTINCT

The DISTINCT condition is used in connection with the SELECT command to retrieve only those items with a unique value. If a value occurs more than once in the table, only the first occurence will be listed.

Input:

```
OPEN DATABASE "../db/db";

SELECT DISTINCT zipcity
FROM customers
WHERE zipcity > "7000" AND zipcity < "7100"
ORDER BY zipcity;
```

tut12

Result:

```
ZIPCITY
7000 STUTTGART 1
7000 STUTTGART 10
   .     .
   .     .
   .     .
7080 AALEN
7090 ELLWANGEN/JAGST
```

The above example shows how the use of the DISTINCT option suppressed all records with the same value for item *zipcity*.

3.13 SELECT with String Constants

Input:

```
OPEN DATABASE "../db/db";

SELECT "Customer:", custno, "Name:", name1
FROM customers
WHERE custno < "11010"
ORDER BY custno;
```

tut13

Result:

```
"Customer:" CUSTNO  "Name:" NAME1
Customer:   100     Name:   SCHAFFER
Customer:   11001   Name:   GROZ-BECKERT
   .         .        .        .
   .         .        .        .
   .         .        .        .
Customer:   11008   Name:   SOCIETE
Customer:   11009   Name:   G. NOLL
```

The use of strings in the list of items to be selected allows us to define fixed text partitions that appear in the output. Each text partition consists of the string and the item. The text partitions are displays in the order listed in the `SELECT` command.

3.14 SELECT with arithmetic expressions

Input:

```
OPEN DATABASE "../db/db";

SELECT
   custno, turnover[0], turnover[1],
   (turnover[0]*100)/turnover[1] "percentage"
FROM customers
WHERE turnover[0] > 0 AND turnover[1] > 0
ORDER BY custno;
```

tut14

Result:

```
CUSTNO  TURNOVER[0]  TURNOVER[1]   percentage
11001       4058.98     18976.81        21.39
11002       7024.89     85839.26         8.18
  .            .            .              .
  .            .            .              .
  .            .            .              .
HOPPE       8401.20     67719.07        12.41
MONT        6196.23     65231.63         9.50
```

Arithmetic operators (+, -, *, /) can be used to calculate item values for retrieved data records as well as construct new items. However, all these calculations exist only in the report. All data in the database remains unchanged.

In the example, only those records where *turnover[1]* is greater than zero were selected. This was specified by using the `WHERE` condition. These records were selected to avoid an error caused by dividing a number by zero.

3.15 SELECT and Functions

Input:

```
OPEN DATABASE "../db/db";

SELECT COUNT(*) FROM customers;
```

tut15

Result:

```
COUNT(*)
    1177
```

There are 5 arithmetic functions available: `COUNT`, `SUM`, `AVG`, `MAX` and `MIN`. All functions have an item name as an argument which will be applied to this parameter. The `COUNT` function is the only arithmetic function that allows an asterisk (`*`) instead of an item name. The asterisk (`*`) instructs **SQL/R** to count all the records in the table (dataset).

Input:

```
OPEN DATABASE "../db/db";

SELECT AVG(turnover[0]), AVG(turnover[1]/12)
FROM customers
WHERE turnover[0] > 0;
```

tut16

Result:

```
AVG(TURNOVER[0]) AVG(TURNOVER[1]/12)
         4986.98             4265.98
```

This example shows how to calculate the average value for the items *turnover[0]* (month-to-date) and *turnover[1]*/12 (year-to-date).

Input:

```
OPEN DATABASE "../db/db";

SELECT SUM(turnover[0]), SUM(turnover[1]/12)
FROM customers
WHERE turnover[0] > 0;
```
tut17

Result:

```
 SUM(TURNOVER[0]) SUM(TURNOVER[1]/12)
       2937330.82         2512659.58
```

This example shows how to calculate the total for the items *turnover[0]* (month-to-date) and *turnover[1]/12* (year-to-date).

Input:

```
OPEN DATABASE "../db/db";

SELECT COUNT(DISTINCT matchcode) FROM customers;
```
tut18

Result:

```
COUNT(DISTINCT MATCHCODE)
                     1012
```

In this final example, we are using the `COUNT` and `DISTINCT` conditions to calculate the number of unique values for *matchcode*. Without the `DISTINCT` condition within the instruction, each value is counted and the result is identical to the total number of records in the table.

3.16 SELECT with LIKE

Input:

```
OPEN DATABASE "../db/db";

SELECT custno, matchcode, zipcity
FROM customers
WHERE matchcode LIKE "KELLER"
ORDER BY custno;
```
tut19

Result:

```
CUSTNO  MATCHCODE   ZIPCITY
11036   KELLER      7293 PFALZGRAFENWEILER
23062   KELLER      7297 ALPIRSBACH
33007   KELLER      7300 ESSLINGEN
```

The operator `LIKE` allows you to specify a character pattern to use for comparision with string items. The simplest pattern is a string without wildcards (see example above). Each question mark (`?`) represents a single character and an asterisk (`*`) can represent either no characters or a combination of characters.

Input:

```
OPEN DATABASE "../db/db";

SELECT custno, matchcode, zipcity
FROM customers
WHERE zipcity LIKE "7000*"
ORDER BY custno;
```
tut21

Result:

```
CUSTNO  MATCHCODE   ZIPCITY
17004   KUNSTO      7000 Stuttgart-Zuffenhausen
24009   KÖLLI       7000 STUTTGART 1
26009   KÄSBOH      7000 STUTTGART 10
29007   KUNSTS      7000 Stuttgart 1
30008   KUTZNE      7000 STUTTGART 80
32008   KUTSCH      7000 STUTTGART 80
35006   KUNSTS      7000 Stuttgart 1
```

This example showes how to extract all data records where the value for *zipcity* starts with “7000”.

Input:

```
OPEN DATABASE "../db/db";

SELECT custno, matchcode, zipcity
FROM customers
WHERE zipcity LIKE "7?00*"
ORDER BY zipcity;
```

tut22

Result:

```
CUSTNO  MATCHCODE   ZIPCITY
24009   KÖLLI       7000 STUTTGART 1
26009   KÄSBOH      7000 STUTTGART 10
  .        .           .
  .        .           .
  .        .           .
11005   HERBER      7900 ULM/DONAU
14011   HERAEU      7900 Ulm
```

This example retrieves all customer records where the value for *zipcity* is as follows:

- The 1st character is a “7”
- the second character is any single character
- the third and fourth characters are “0”

- followed by a combination of any characters (or no characters)

The `LIKE` condition can also be used to define a string of characters within or at the end of an item:

Input:

```
OPEN DATABASE "../db/db";

SELECT custno, matchcode, zipcity
FROM customers
WHERE zipcity LIKE "*80";
```

tut22a

Result:

```
CUSTNO  MATCHCODE   ZIPCITY
13018   GEYER       8500 Nürnberg 80
32026   HAPS        8000 MÜNCHEN 80
32008   KUTSCH      7000 STUTTGART 80
30008   KUTZNE      7000 STUTTGART 80
23073   WICKE       2000 Hamburg 80
```

The example above selects only those customer records with an item value for *zipcity* ending with "80".

3.17 SELECT with GROUP BY

Input:

```
OPEN DATABASE "../db/db";

SELECT salesarea, SUM(turnover[0])
FROM customers
WHERE salesarea BETWEEN "0" AND "9"
GROUP BY salesarea;
```
tut23

Result:

```
SALESAREA  SUM(TURNOVER[0])
0                   8864.09
1                  53252.06
2                 182403.50
3                  75383.51
4                 262745.05
5                 524570.07
6                 455429.99
7                 497460.46
8                 378855.36
9                   1065.71
```

The option GROUP BY consolidates data records with identical values for a specified item into a single result line. The values of all other items should be combined using the numeric functions, because each item in the consolidated result line can only hold one value.

The above SELECT ... WHERE statement retrieves all records of the table *customers* that have a *salesarea* value of between “0” and “9”. The GROUP BY option then consolidates the data records by *salesarea*. We use the SUM function for the item *turnover[0]* (month-to-date turnover) to calculate a group total for each *salesarea*. This way each value for *salesarea* shows up only once and the values of item *turnover[0]* are totaled.

3.18 GROUP BY with HAVING

The HAVING option can be compared with the keyword WHERE, because it is used in a similar way: specifically to extract only those consolidated result lines that fulfill a given

condition. The `HAVING` instruction is processed **after** execution of the `GROUP BY` rule and applies to the `GROUP BY` results.

Input:

```
OPEN DATABASE "../db/db";

SELECT salesarea, SUM(turnover[0])
FROM customers
WHERE salesarea BETWEEN "0" AND "9"
GROUP BY salesarea
HAVING SUM(turnover[0]) > 100000;
```

tut24

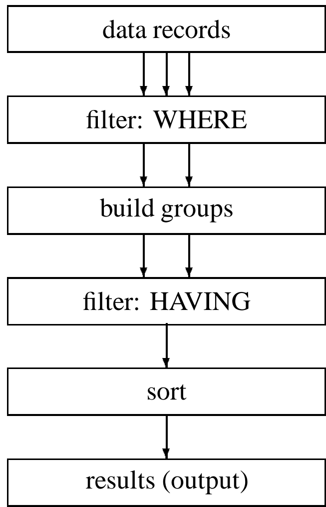
Result:

```
SALESAREA  SUM(TURNOVER[0])
2                 182403.50
4                 262745.05
5                 524570.07
6                 455429.99
7                 497460.46
8                 378855.36
```

The `HAVING` instruction suppresses all of those result lines (groups) that do not satisfy the condition (SUM(turnover[1]) > 100000). For this condition we can use the same operators and expressions as for conditions using the `WHERE` option.

You can compare the output of this example with the output of the previous example, which used the `WHERE` condition.

The HAVING option works like a additional filter on the results:



3.19 The next step

We are now at the end of our short introduction to **SQL/R**. You can use the sample database for further exercises, e.g. to explore the **SQL/R** options in more detail as described in the reference part of this manual.

As you cannot modify data in a database but only read data with **SQL/R**, you can apply the examples of this introduction easily and without risk to your own databases. It will help you gain more experience with your first **SQL/R** reports.

In chapter 5 you will find some step-by-step instructions on how to develop your own reports.

4

Editor

This chapter describes how to use the **SQL/R** editor. To start the **SQL/R** editor, type this command at the HP-UX shell prompt:

```
sqlr
```

If you specify a file name along with the command, this file will immediately be loaded into the editor.

After you enter the **SQL/R** command you are in the **SQL/R** editor environment. You can now select a function key to continue. Within the editor you can also enter instructions, execute them, and create QRF files (refer to RUN command) and form files (refer to REPORT command).

All input is inserted at the current cursor position. If the text to be entered is longer than the screen width, then the line is moved to the left. If the end of line is not visible from the current position, an inverse [!] is displayed in the right margin of the line.

4.1 Keys for text processing

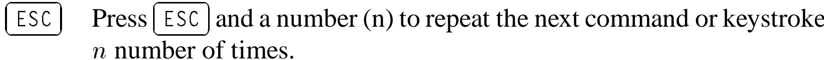
The following keys can be used for cursor movement and text processing, e.g. deletion of characters, words or lines:

[→] moves the cursor one position to the right. If the cursorr is at the end of the line, the cursor is moved to the first position of the next line.

[←] moves the cursor one position to the left. If the cursor is at the beginning of the line, the cursor is moved to the last position of the previous line.

[↑] moves the cursor up one line. If the cursor it is already at the top line, the cursor stays in the same position. If the previous line is shorter than the current cursor position, the cursor is moved to the end of that line.

Key	Description
↓	moves the cursor down one line. If the cursor it is already on the last line, the cursor stays in the same position. If the next line is shorter than the current cursor position, the cursor is moved to the end of that line.
↖	HOME: moves the cursor to the beginning of the first line.
SHFT ↖	SHIFT HOME: moves the cursor one position behind the last character of the last line.
CTRL A	CTRL-A: moves the cursor to the beginning of the current line.
CTRL E	CTRL-E: moves the cursor to the next position after the end of line.
↩	RETURN: inserts a new line and moves the cursor to the first position of the new line. If the cursor was at the end of line when the RETURN key was used, an empty line is inserted, otherwise the text will be split into two lines at the current cursor position.
⟸	BACKSPACE. The character one position to the left left of the current cursor position is deleted. If the cursor is at the beginning of a line, this line is appended to the previous line.
INS LINE	INS LINE: deletes the word left to the current cursor position. If the cursor is at the beginning of the line, the deletion will be done / continued in the previous line. The cursor position changes accordingly.
DEL LINE	DEL LINE: deletes the word from the current cursor position. If the end of line has been reached, then the following line is appended.
INS CHAR	inserts a newline at the current cursor position. The cursor does not move.
DEL CHAR	deletes character at the cursor position. If the cursor is at the end of the line, the next line will be concatenated to this line.
CLR LINE	the line is deleted from the cursor position to the end of the line. To delete the entire line, position the cursor at the beginning of the line and press CLR LINE. If the cursor is at the end of the line, the next line will be concatenated to this line.
BREAK	interrupts the process or program. A confirmation question is displayed before the program is aborted.



ESC — Press ESC and a number (n) to repeat the next command or keystroke n number of times.

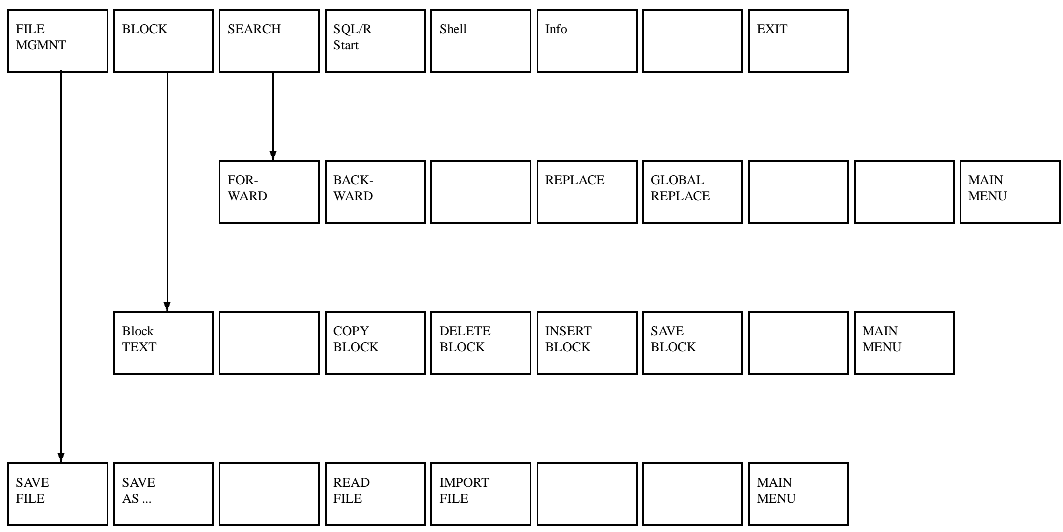
CTRL V — the following control character will be inserted into the text. Control characters are displayed in inverse mode. This is useful for sending some special control characters to your printer for printing form files.

CTRL G — a number (n) followed by CTRL-G moves the cursor to the line n. The column position remains the same unless the current line is shorter. If the current line is shorter, the cursor will be positioned at the end of the line n.

CTRL L — CTRL L refreshes the display.

CTRL W — Toggles the screen configuration between 80 characters and 132 characters per line. This is currently supported with the following terminal types: 700/92, 700/94, 700/96 and 700/98.

4.2 The Menu Structure



Main menu function keys f1 through f3 each display a submenu containing specific commands. The F1 function key displays file management commands, the F2 function key displays text block commands and the F3 function key displays text search/replace commands.

Pressing the f8 function key from any submenu returns you to the main menu.

Screen messages and function key labels are controlled by the `LANG` variable selected. The examples given assume the `LANG=american` configuration. (⟶ Appendix D).

4.3 Main Menu Bar

Main menu function keys f1 through f3 each display a submenu containing specific commands. The F1 function key displays file management commands, the F2 function key displays text block commands and the F3 function key displays text search/replace commands.

4.3.1 SQL/R Start (f4)

Function key f4 starts processing the currently loaded or created commands. During processing the following message is displayed:

```
Request is being processed ...
```

The process results are then displayed and you can review these. If the results are longer than screen length, the message will be displayed as the last line on the screen:

```
-- press <return> to continue or  q<return> to quit:
```

Press return to view the next screen.

To return to the text editor, press the following keys:

q and ↵

4.3.2 Shell (f5)

To access the HP-UX shell while in Editor, press the f5 function key.The following message will appear:

```
To return to the editor, type exit <return>
```

To return to the editor, type:

```
exit ↩
```

4.3.3 Info (f6)

To display the information bar, press the f6 function key: The information bar above the function keys will display:

- file name
- file access (read only, read/write)
- number of lines in file
- number of characters in file
- number of characters in the marked text block
- line number of cursor position

The information bar remains on the screen until any key is pressed. In addition to this callable information bar, all text changes involving more than one line of the text will result in a short message being displayed on the screen.

4.3.4 Exit Program (f8)

To exit **SQL/R**, press the f8 function key.

If you have modified an existing file, the following message appears:

```
[filename] was modified.  Save changes (y/n) ?
```

Press [y] to save the changes.

Press [n] to exit without saving the changes. The existing file remains unchanged.

If the file is a new file, the following message is displayed:

```
[memory] was modified.  Save changes (y/n) ?
```

If you press [y], the following message appears:

```
Please enter filename:
```

Enter a filename and press RETURN:

[↵]

to save the file and exit **SQL/R**.

4.4 File Management

The menu bar displays the commands for loading and saving text and files. When prompted to enter a filename, enter the filename and press the RETURN key begin processing that file. Pressing the `f8` `MAIN MENU` function key displays the main menu bar.

Press the `BREAK` key to abort the execution.

4.4.1 Read File

To load a text file, press

```
f4 Read File
```

If the text file was modified, the following message appears:

```
[...]  was modified.  Save changes (y/n)?
```

Within the brackets appears either the filename (existing file) or "memory" (new file). The name of the new file appears only after the file has been saved with this filename. Then the following message appears:

```
Read file:
```

Enter the file name. After the file is loaded, the following message appears:

`Read:` *`Infotext`*

The information block displays the filename, number of lines and characters read, and the line number of the cursor position.

4.4.2 Import File

In addition to loading and reading a file, it is also possible to import a file into the current text file. To import a file, position the cursor where the new file should be inserted and press:

`f5` `Import File`

The message appears:

```
Import File:
```

Enter the filename of the file to be imported. The import file is then read and inserted at the cursor position. In addition, a message is displayed with the number of lines inserted.

4.4.3 Save File

To save a file, press:

`f1` `Save File`

If this is a new file, the following message appears:

```
Please enter filename:
```

Enter a file name and press RETURN. The following message appears:

`Saved:` *`Infotext`*

The information block contains the filename, number of lines and characters, and the line number of the cursor position.

If the file is an existing file, the file is saved immediately. If the existing file was not modified, no save is necessary and thedie Sicherung ohne zusätzliche Eingaben durchgeführt. Wurde die following message appears:

```
Save not needed
```

To save an existing file using a different name press

`f2` `Save as`

The following message appears:

```
Save as:
```

Enter the new filename and press RETURN. The information bar will now display the new filename.

4.5 Text Block Management

The commands discussed so far modify individual characters and lines, but it is also possible to modify blocks of text with the commands of the function key set. Modifying blocks of text involves 2 steps: first marking the text block and second selecting the action to be taken.

To return to the main menubar, press the `f8` `MAIN MENU` function key.

4.5.1 Mark Block

Position the cursor at the beginning of the text to be blocked and press

`f1` `BLOCK TEXT`

The following message appears:

```
Mark set
```

Now position the cursor at the end of the text to be blocked. All text between the first cursor position and the last cursor position will be included in the block and therefore modified by the selected action.

4.5.2 Copy Block

The text block will be copied into the "block memory" area of memory. When this has completed, the following message appears:

n `characters copied into memory`

The "block memory" remains unchanged until either replaced by a new block of text or deleted.

4.5.3 Delete Block

The blocked text is copied into "block memory" and deleted from the text. When completed, the following message appears:

n `characters moved into memory`

4.5.4 Insert Block

The contents of “block memory” are inserted into the text at the cursor position. When completed, the following message appears:

```
n lines inserted
```

You may repeat this command to insert the same text block into the text in several locations.

To move a text block within a file press `f4` `DELETE BLOCK`, then position the cursor to the new location and press `f5` `INSERT BLOCK`.

4.5.5 Save Block

To save a text block from “block memory” as a separate file, press `f6` `SAVE BLOCK`. The following message appears:

```
write block to filename:
```

Enter a filename and press ↩.

The block is saved as a file and the following message appears:

```
block text saved:  infotext
```

The infotext displays the filename, the number of saved characters and lines, and the line number of the cursor position.

4.6 Search and Replace

The editor provides a search and replace feature to locate specific text and replace it with different text. The specified search pattern can consist of regular expressions or specific character strings.

A regular expression is a sequence of characters that defines a set of character strings.

- A normal character represents the same character in the text.

  `Smith` searches for "Smith" in the text

- A dot `.` is a placeholder for any single character.

  `de.` searches for "der", "des", "den", etc.

- A `^` means that the following expression occurs at the beginning of a line.

  `^SELECT` searches for "SELECT" at the beginning of a line.

- A `$` means that the following expression occurs at the end of a line.

  `SELECT$` searches for "SELECT" at the end of a line

- Characters enclosed in square brackets `[ ]` are searched regardless of the order in which they are listed. Sequential characters can be abbreviated with a `-` (hyphen).

  `[0-9]` searches for all numeric characters

  `[abc]` searches for character strings containing "a", "b" or "c"

- If one of the special search characters is followed by an (*) asterisk, then the characters represented by the search characters can be repeated several times.

  `A[0-9]*B` searches "A12...76B".

  Several numeric characters can occur between "A" and "B".

- To search any of the above wildcard characters as a literal, place a backslash (\) before the character.

  `20.\.00` searches for 201.00, 202.00, 20a.00 etc.

4.6.1 Search

You can search forwards as well as backwards. Forwards means from the cursor position to the end of the text and backwards means from the cursor position to the beginning of the text. To search forwards, press

f1 FORWARD

The following message appears:

```
Forward search:
```

Enter the search pattern and press ↩.

When the search pattern is found, the string is highlighted and the cursor is positioned at the beginning of the string. If the search string is not found, the following message appears:

```
pattern not found
```

To search the same pattern in another direction, press RETURN when prompted for the search pattern. The search proceeds as normal.

4.6.2 Replace

Press the key

f4 REPLACE

The following message appears:

```
Replace:
```

Enter the text that should be replaced and press RETURN.

The search text can contain regular expressions. When the text is for example `A[1-9]B` and should be replaced by `AB`, then all expressions such as `A0B`, `A1B`, `...` `A9B` will be replaced by `AB`.

To replace a constant string containing characters that are used to represent regular expressions, use a backslash (\) before the character. For example, to replace `20.00`, use the string `20\.00`.

The following message then appears, prompting for the replacement text:

```
Replace:  ... by:
```

Enter the desired replacement text and press RETURN. If the replacement text contains an ampersand `&`, the original text will be inserted in this position. To avoid this, designate the ampersand `&` as a literal by preceding it with a backslash `\`.

The search text will be searched forward of the cursor position, and when found, the cursor will stop at the first position of the found search text.

The following message appears:

```
replace (old text) by (new text) ? (!/y/n)
```

Pressing `y` for "yes" replaces the old text with the new text and displays the next occurence of the search text. Pressing `n` for "no" leaves the text unchanged and displays the next occurence of the search text. Pressing `!` for "all" replaces all occurences of the search text with the new text without prompting for each occurence.

To end the search / replace operation, press `BREAK` at any time.

If the search text is not found, the following message appears:

```
pattern not found
```

After a successful text replacement, the following message appears:

n `replacement(s) in` m `line(s)`

N represents the number of times the search text was found and replaced with the new text. M represents the number of modified lines resulting from the search / replace.

4.6.3 Global Replace

Press the key

f5 `GLOBAL REPLACE`

The entry of the search and replace texts functions as described previously, the search text is replaced without prompting the user to confirm the change. The cursor position remains unchanged during the operation. After the global replace ends, the following message appears:

n `replacement(s) in` m `line(s)`

N represents the number of times the search text was found and replaced with the new text. M represents the number of modified lines resulting from the search / replace.

5

The Usage of SQL/R

This chapter contains a detailed description of the **SQL/R** language and specific examples. All examples are based on the accompanying sample database and can be performed by you. The examples were designed to produce lists similar to those commonly used by business. This way, you can probably adapt these sample reports by simply modifying the item and table names.

Before beginning this chapter, you should be familiar with the **SQL/R** basics covered in chapter 3.

All examples are located in the `/usr/sqlr/sample` directory and are designated with the file name `man` and a number. The exact file name is shown in small print in the right margin of the page.

To use these examples, it will be necessary to change to the directory `/usr/sqlr/sample`. To do this, enter:

```
cd /usr/sqlr/sample
```

The sample database is located in the `/usr/sqlr/db` directory and is named `DB`.

The sample results shown in this chapter are printed in simplified form to show the results format. To view actual results, practice the examples on the accompanying sample database.

5.1 An Easy List of Customers

The goal of this section is to explain the steps necessary to produce a list. To demonstrate these steps, an example is presented in which a list is produced using the basic elements of **SQL/R** language.

The individual steps:

- Opening the database
- Selecting items (columns) from a table
- Formating output without a form file
- Formating output with a form file
- Using batch files and parameters

We want to produce a list of customers from the "CUSTOMERS" table, which is part of the "DB" database. The list will report the customer number, the customer matchcode, the complete customer name, and the month-to-date turnover. We will only select customers with actual turnover. The list will be sorted by customer number in ascending order.

5.1.1 Opening the Database

A database must be opened before records can be extracted. To open a database, enter the command:

```
OPEN DATABASE "name";
```

The name of the database is enclosed in quotes. In this example, we will give the command:

```
OPEN DATABASE "../db/db";
```

Please note that each **SQL/R** command ends in a semicolon.

You can also open multiple databases and use tables from each of these databases to produce a list.

5.1.2 Selecting Items from a Table

The selection of items (columns) from a table is done with the `SELECT` command. This command consists of several parts:

selection of n items	: `SELECT`	S1, S2, ... , Sn
selection of the table	: `FROM`	table name
conditions for the selection	: `WHERE`	conditions
sort order	: `ORDER BY`	O1, O2, ... , On

To produce the list in our example, enter the following commands as shown:

```
OPEN DATABASE "../db/db";

SELECT custno, matchcode, name1, name2, turnover[0]
FROM customers
WHERE turnover[0] > 0
ORDER BY custno;
```

man11

Result:

```
                                                            PAGE 1
CUSTNO  MATCHCODE  NAME1              NAME2              TURNOVER[0]
00001   KUGEL      Kugelfischer       Maschinenfabrik        1000.00
...     ...        ...                ...                        ...
```

While customer number and matchcode correspond to exactly one item, customer name and turnover are stored in a different way in table CUSTOMERS.

The customer name is a combination of two fields, $name1$ and $name2$.

The field $turnover$ is an array composed of 3 elements that each contains a different value as shown here:

turnover[0] = month-to-date
turnover[1] = year-to-date
turnover[2] = previous year

A particular element in an array will always be retrieved by use of a index number, which in this example is 0. Remember to number your elements beginning with zero. This means that the n-th element in an array is numbered n-1 in the index.

The commands described have extracted the desired records from the table. Now we will format these records to produce the final list.

5.1.3 Formatting the Output without a Form File

Because the customer name is composed of the data in two separate fields, two fields will also be produced in the list. To join the two fields in the output, use the & (ampersand) operator and an empty space enclosed in quotes to display the two name parts together with a blank between the names.

There are two ways to do this. One way is to include this expression in the SELECT command:

```
OPEN DATABASE "../db/db";

SELECT custno, matchcode, name1 & " " & name2, turnover[0]
FROM customers
WHERE turnover[0] > 0
ORDER BY custno;
```

man12

Result:

```
                                                             PAGE 1
CUSTNO  MATCHCODE  NAME1&" "&NAME2                       TURNOVER[0]
00001   KUGEL      Kugelfischer Maschinenfabrik              1000.00
...     ...        ...                                           ...
```

The second alternative is to use the FIELD command to define an alternate name. This alternate name is then inserted into the SELECT command in place of the two original item names.

```
OPEN DATABASE "../db/db";

FIELD name = name1 & " " & name2;

SELECT custno, matchcode, name, turnover[0]
FROM customers
WHERE turnover[0] > 0
ORDER BY custno;
```

man13

Result:

```
                                                            PAGE 1
CUSTNO  MATCHCODE  NAME                                TURNOVER[0]
00001   KUGEL      Kugelfischer Maschinenfabrik            1000.00
...     ...        ...                                         ...
```

The actual field name is used as a heading in the report. This field name may not be self-explanatory though. Therefore the capability exists to rename this report heading in the `SELECT` command line. In the previous report examples we have always used the field name as it appears in the `SELECT` statement. However, you can also refer to a field (item) by a number representing the field position in the `SELECT` command. This numerical alias can be used in the `ORDER BY`, `GROUP`, and arithmetic calculation commands (such as `SUM`).

In our example above, the first field name is *custno*. The numerical alias for this field would be 1, since it is the first field when counting from left to right. The same principle applies to the other fields as well.

NOTE: You must use a numerical alias for a field if you assigned an alternate name to one or more fields in the `SELECT` command.

The next example shows how you would use alternate headings and the alias naming feature:

```
OPEN DATABASE "../db/db";

FIELD name = name1 & " " & name2;

SELECT
  custno "Custno.", matchcode "Matchcode",
  name "Company", turnover[0] "Monthly Sales"
FROM customers
WHERE turnover[0] > 0
ORDER BY 1;
```

man14

Result:

```
                                                                  PAGE 1
Custno.  Matchcode  Company                                   Monthly Sales
...      ...        ...                                                 ...
23062    KELLER     Keller, Ihne & Tesch KG                     1000.00
...      ...        ...                                                 ...
```

To further enhance the report, we can use the `REPORT` command. Now we will add a report title and the current date to the report:

```
OPEN DATABASE "../db/db";

FIELD name = name1 & " " & name2;

REPORT
   SELECT
      custno "Custno.", matchcode "Matchcode",
      name "Company", turnover[0] "Monthly Sales"
   FROM customers
   WHERE turnover[0] > 0
   ORDER BY 1
TITLE AS "CUSTOMER SALES FOR CURRENT MONTH/SORTED BY CUSTOMER NUMBER/"
DATE AS TODAY;
```

man15

Result:

```
08/01/93       CUSTOMER SALES FOR CURRENT MONTH          PAGE 1
                    SORTED BY CUSTOMER NUMBER

Custno.  Matchcode  Company                             Monthly Sales
...      ...        ...                                          ...
23062    KELLER     Keller, Ihne & Tesch KG                  1000.00
...      ...        ...                                          ...
```

As you can see, we used `TITLE AS` clause to add a report title. If the title consists of more than one line, use the slash (/) to mark the separation between the individual lines of the title. Each line will then be automatically centered. Page numbers always appear in the right margin.

The use of the `DATE AS` command prints the current date in the left margin. You can either enter a specific date format with the `DATE AS` command or use the word `TODAY`. In this case, the date format configured with the `SET DATE` command will be used. Default is the American date format MM/DD/YY.

A specific date format can be configured using the `SET DATE` command as follows:

```
SET DATE = "%d.%m.%y";
```

This command produces the European date format DD.MM.YY. The allowable date formats are shown in appendix B.

The `REPORT` command also gives you the capability to calculate subtotals and totals by using the `CALCULATE` command. In the next example, we want a list of customers with a total showing the number of customers and the total sales for all customers.

```
OPEN DATABASE "../db/db";

FIELD name = name1 & " " & name2;

REPORT
   SELECT
      custno "Custno.", matchcode "Matchcode",
      name "Company", turnover[0] "Monthly Sales"
   FROM customers
   WHERE turnover[0] > 0
   ORDER BY 1
CALCULATE
   COUNT( 1 ) BREAK ON REPORT,
   SUM( 4 )   BREAK ON REPORT
TITLE AS "CUSTOMER SALES FOR CURRENT MONTH/SORTED BY CUSTOMER NUMBER/"
DATE AS "Date: %d.%m.%y";
```

man16

Result:

```
Date: 30.01.93    CUSTOMER SALES FOR CURRENT MONTH       PAGE  1
                        SORTED BY CUSTOMER NUMBER

Custno.  Matchcode  Company                          Monthly Sales
...      ...        ...                                        ...
23062    KELLER    Keller, Ihne & Tesch KG                 1000.00
...      ...        ...                                        ...
-------- ---------- ----------------------------- ------------
  100                                                          COUNT
                                                    10000.00 SUM
```

The calculations are determined by an arithmetic operator followed by a list of field numbers, enclosed in parentheses, to which the arithmetic is applied. Within a REPORT command, you can define several calculations, separated by commas. The results of these calculations will all be reported on one line in the output, and in the order in which they were entered in the command.

The BREAK ON command defines when a subtotal should be listed. To display a total for the entire report, use the BREAK ON REPORT command. In the previous example, the BREAK ON REPORT command was used to display a total for the number of customers and their total sales at the end of the report.

By default, subtotals and totals will be followed by the name of the arithmetic function used to calculate the number. In our previous example this was COUNT and SUM. However, you can also substitute a specific text label for this arithmetic label. This is done by entering such a label in quotes directly after the arithmetic function in the CALCULATE command.

The CALCULATE command is then entered as follows:

```
CALCULATE
   COUNT( 1 ) "Customers"
      BREAK ON REPORT
   SUM( 4 )   "Sales Total"
      BREAK ON REPORT
```

After you have the desired output online, you can print the report. The output will appear with the default length of 24 lines (normal screen length).

This value is probably not appropriate for your printer. Therefore you may want to reset it with the LENGTH command. This command applies to output sent to the printer as well

as to your display screen. Output can be sent to the printer by using the INTO PRINTER command, as shown in this example:

```
OPEN DATABASE "../db/db";

FIELD name = name1 & " " & name2;

REPORT
   SELECT
      custno "Custno.", matchcode "Matchcode",
      name "Company", turnover[0] "Monthly Sales"
   FROM customers
   WHERE turnover[0] > 0
   ORDER BY 1
CALCULATE
   COUNT( 1 ) BREAK ON REPORT,
   SUM( 4 )  BREAK ON REPORT
INTO PRINTER
TITLE AS "CUSTOMER SALES FOR CURRENT MONTH/SORTED BY CUSTOMER NUMBER/"
DATE AS "Date: %d.%m.%y"
LENGTH = 72;
```

man17

NOTE: The default printer is the system printer. To select a different printer, use the SET PRINTER command. For more information on this command, see 6.21.5 on page 127.

5.1.4 Formatting the Output with a Form File

Report formats can also be enhanced by the use of a form file. Form files are referenced by the `USING` command in the `REPORT` command section. It is not necessary to define report and field titles if a form file is used. These can be defined in the form file.

The report defined in the previous example is modified to use a form file in this way:

```
OPEN DATABASE "../db/db";

FIELD name = name1 & " " & name2;

REPORT
   SELECT custno, matchcode, name, turnover[0]
   FROM customers
   WHERE turnover[0] > 0
   ORDER BY 1
CALCULATE
   COUNT(1) BREAK ON REPORT,
   SUM(4) BREAK ON REPORT
INTO PRINTER
DATE AS "%d.%m.%y"
LENGTH = 72
USING "man18.frm";
```

man18

NOTE: Use of the `TITLE AS` command is not allowed when using a form file. Also the `USING` command <u>must</u> be the last command in the `REPORT` section.

A form file consists of a number of sections, separated by comment lines. These comment lines always begin with two percentage symbols `%%`.

The first section is the page heading that appears at the top of each new page. This page heading processes the report title, date and page number. You also have the option of including results of the `SELECT` command in this page heading section.

The following lines illustrate the page heading shown at the end of this section:

```
LIST OF CUSTOMER SALES FOR CURRENT MONTH             Page: $page
SORTED BY CUSTOMER NUMBER                            Date: $date
```

```
Custno   Matchcode Company                           Monthly Sales
------------------------------------------------------------------
%% end of page header
```

The page header section shown includes two special options, namely the `$page` and the `$date`. The `$page` option is important, because it consecutively numbers the pages of the output. The page number position is defined in the form file. The only requirement is that the `$page` option must appear in either the page header or footer sections.

Similarly, the `$date` option defines the date in the report. The `$date` option is different from the `$page` option in that the `$date` option can be used in any section of the form file. If no alternate date format is defined by using the `DATE AS` command, then the date format default defined by the `SET DATE` command is used.

The second section of the form file defines the format of the lines of output produced as a result of the `SELECT` command:

```
@custno  @matchcode  @name                              @turnover[0]
%% end of line format section
```

or

```
@1       @2          @3                                 @4
%% end of line format section
```

The "@" character sets the alignment of the field columns in the output. The value of the individual fields in a line are referenced in the form file by using either the field name or the field position in the `SELECT` command.

Each line of the output, retrieved using the `SELECT` command, will be in the defined format.

The field values retrieved with the `SELECT` command can also be used in the page heading section. The values shown in the page heading are always the actual field values at the time the page heading is produced.

The form file must include a break section for each calculation defined within a `REPORT` or `CALCULATE` command. This break section then defines the output format for the calculated values. The break sections should be defined in the form file in the same sequence as they appear in the `REPORT` command.

```
Customers:        @1
%% end of the break section for COUNT(1)
Sales total:                                             @4
%% end of the break section for SUM(4)
```

A page footer section can also be defined in the form file. This page footer will appear at the bottom of each page.

Finally, the complete form file has the following format:

```
CUSTOMER SALES FOR CURRENT MONTH                               Page: $page
SORTED BY CUSTOMER NUMBERS                                     Date: $date

Custno  Matchcode  Company                                     Monthly Sales
------------------------------------------------------------------------------
%% end of page heading
@1      @2         @3                                          @4
%% end of line format section
------------------------------------------------------------------------------
Customers:         @1
%% end of break section for COUNT(1)
Sales total:                                                   @4
%% end of break section for SUM(4)
```

man18.frm

This form file produces the following result:

```
CUSTOMER SALES FOR CURRENT MONTH                        Page: 1
SORTED BY CUSTOMER NUMBER                               Date: 01.08.92

Custno   Matchcode  Company                             Monthly Sales
----------------------------------------------------------------
...      ...        ...
23062    KELLER     Keller, Ihne & Tesch KG                   1000.00
...      ...        ...                                           ...
----------------------------------------------------------------
Customers:          100
Sales total:                                                 10000.00
```

NOTE: The length of a field display is determined by the field type and the settings defined with the `FIELD ... DISPLAY AS` command. The appearance of a field in the output is defined in the form file. Field values longer than the space available in the output will be right truncated.

5.1.5 Using SQL/R and Parameters from the Shell

The previous section explained how to interactively use **SQL/R**. It is also possible to store these commands in a file and execute this file later. You use the `sqlrexec` command to do this. The `sqlrexec` command is used as follows:

```
sqlrexec customers
```

Where "customers" is the name of the **SQL/R** script file. In addition, you can specify up to 8 parameters at runtime with this command. In the following example, all customers with sales are reported. To request a list of all customers with a minimum of 1,000 in sales, a value of 1,000 is needed for the `WHERE` command. It is possible to provide this value with the `sqlrexec` command for use with the stored **SQL/R** commands.

Please note: Since commas are used to separate the different parameter values, no commas should be used in the value itself.

```
sqlrexec customers 1,000.00          wrong
sqlrexec customers 1000,00           wrong

sqlrexec customers 1000.00           right
sqlrexec customers 1000              right
```

The name of the form file used to format the output should be specified as follows:

```
sqlrexec customers 1000 customers.frm
```

Where "customers.frm" is the name of the form file. To properly execute this command, a modification is required for the command file "customers". Parameters used in calculations should be referenced by a `$` character followed by a number representing the sequence in which the parameter appears in the `sqlrexec` command. These `$n` designations are then calculated as actual values during the operation.

```
OPEN DATABASE "../db/db";

FIELD name = name1 & " " & name2;

REPORT
   SELECT custno, matchcode, name, turnover[0]
   FROM customers
   WHERE turnover[0] > $1
   ORDER BY 1
CALCULATE
   COUNT(1) BREAK ON REPORT,
   SUM(4) BREAK ON REPORT
DATE AS "%d.%m.%y"
LENGTH = 72
USING "$2";
```
man19

Please note that as shown in this example the form file is passed as a character string to the `USING` command. Therefore the parameter has to be enclosed in quotation marks too.

The parameters from the `sqlrexec` command can also be carried to the form file in which `$n` marks are used to indicate which parameter should be used where.

```
LIST OF CUSTOMERS WITH SALES GREATER THAN $1                  Page: $page
IN CURRENT MONTH SORTED BY CUSTOMER NUMBERS                   Date: $date

Custno  Matchcode  Company                                    Monthly Sales
-------------------------------------------------------------------------------
%% end of page heading
@1      @2         @3                                         @4
%% end of line format section
-------------------------------------------------------------------------------
Customers:         @1
%% end of break section for COUNT(1)
Sales total:                                                  @4
%% end of break section for SUM(4)
```
man19.frm

The form file had been modified to show a different value for the sales minimum in the page header.

You can also create a short shell script to prompt the user for the parameter values as shown here:

```
#!/bin/sh
# man19.sh

echo "LIST OF CUSTOMERS WITH SALES\n"

echo "SALES MINIMUM GREATER THAN: \c"
read sales
if [ -z "$sales" ]
then
   sales="0"
fi

echo "FORM FILE : \c"
read form
if [ -z "$form" ]
then
   form=man19.frm
fi

sqlrexec -n $sales $form | lp -onb -l72
```

man19.sh

As shown, the user is prompted for the necessary input and these inputs are then used for the report generation. The resulting output (stdout) is sent to the system printer.

5.2 List of Customers Grouped by Sales Volume

The second example illustrates additional capabilities of the commands explained so far.

We want to develop a list of customers with the customer number, name and sales for the previous year. The customers should be grouped by sales and the output should be sorted by sales in descending order and in addition, the sales of each group should be subtotaled. This same report should contain a summary list with the number of customers per group and a comparison of current and previous year sales.

The following example shows the **SQL/R** commands used to prepare the report. Following this example is a step-by-step description of the commands.

The command file `man21`:

```
OPEN DATABASE "../db/db";

SET DATE = "%d.%m.%y";

FIELD prevsales = turnover[2] DISPLAY AS MONEY(12, 0);
FIELD ytdsales  = turnover[1] DISPLAY AS MONEY(12, 0);
FIELD groups     = IF (prevsales >= 800000, "A",
                     IF (prevsales >= 250000, "B", "C"));

REPORT
   SELECT groups, custno, name1, prevsales
   FROM customers
   WHERE prevsales <> 0
   ORDER BY 4 DESC
CALCULATE
   SUM(4) BREAK ON (1) PAGE
USING "man21a.frm";
```

```
REPORT
   SELECT
      group, COUNT(custno), SUM(prevsales), SUM(ytdsales)
   FROM customers
   GROUP BY 1
CALCULATE
   SUM(2,3,4) BREAK ON REPORT
USING "man21b.frm";

EXIT;
```

man21

The form file `man21a.frm`:

```
               CUSTOMER SALES REPORT       page: $page
                    - class @1 -           date: $date
------------------------------------------------------------
Customer   Company                     previous year sales

%% End of heading
@2         @3                                @4
%% End of results
------------------------------------------------------------
T O T A L                                    @4
------------------------------------------------------------
%% End of break on SUM(4)
```

man21a.frm

The form file `man21b.frm`:

```
             CUSTOMER SALES REPORT         page: $page
                   - summary -             date: $date

Class   Count   previous year sales      YTD sales
------------------------------------------------------------
%% End of heading
@1   @2              @3                     @4
%% End of results
------------------------------------------------------------
**   @2              @3                     @4
------------------------------------------------------------
%% End of break on SUM(2,3,4)
```

man21b.frm

Note that because each of the two `REPORT` commands creates a complete list, several form files can be used. Therefore, you can create a command file in which there are several `REPORT` commands, each producing a separate list. You can then execute this command file by using the `sqlrexec` command.

The previous commands shows the following enhanced **SQL/R** features:

- the `DISPLAY AS` rule in the `FIELD` command
- the use of conditional expressions `IF`($condition$, yes, no)
- use of sort order
- enhancement of the `CALCULATE` rule within the `REPORT` command
- use of the `GROUP BY` rule and its function in selecting columns in a table
- the `EXIT` command

In this example, because the selected sales are not individual items, but rather elements of an array, we use the `FIELD` command to define an alternate name for the element. This helps to make the following commands more readable. In addition, we want to display the value of the sales in Dollars ($) without decimals. To do this we use the `DISPLAY AS` rule.

```
FIELD prevsales = turnover[2] DISPLAY AS MONEY(12, 0);
```

The next task is to define the groups and to produce the desired values for these groups. You use the `FIELD` command to define a temporary label for the columns. We did that in a previous example and called a column $name$. The values for the entries of the column $group$ are calculated through a nested `IF` command.

The arrangement of the commands is illustrated by the following questions and relative answers:

Is last year's sales amount greater or equal to 800,000.00 ?

Yes the customer belongs in group A.

No Is last year's sales amount greater or equal to 250,000.00 ?

  Yes the customer belongs in group B

  No the customer belongs in group C

This decision tree could also be expanded to handle a larger number of groups.

The following FIELD command in connection with the IF command shows how the **SQL/R** language can be used to answer the questions shown above.

```
FIELD group = IF (prevsales >= 800000, "A",
                  IF (prevsales >= 250000, "B", "C"));
```

The IF command is used to choose between two selections, based on the result of a previously evaluated condition. These selections can be constants, variables, calculations, or, as shown in this example, specific conditions.

The first list will be sorted by previous year's sales amounts. Generally, lists are sorted in ascending order. To sort the list in descending order, use the word DESC in addition to the column label or item number.

```
ORDER BY 4 DESC
```

To calculate subtotals, in this case the totals of the previous year's sales amounts for groups A, B, and C, you use the CALCULATE rule. The list should also cover these three points:

- Calculate the previous year's total sales $prevsales$
- Display the subtotal whenever the value in the column $group$ changes, and reset the subtotal to zero
- Generate a page break after the display of each subtotal

To format the output this way, use the following commands:

```
CALCULATE
   SUM(prevsales) BREAK ON (group)
```

or

```
   SUM(4) BREAK ON (1)
```

In contrast to the first example, we defined totals to be calculated depending on columns. The command `BREAK ON REPORT` is used to define calculations (e.g. building totals) using all entries of a list. To retain subtotals, it is necessary to define exactly where each subtotal should be reported. To report these subtotals, include the column label after the `BREAK ON` command. When the `BREAK ON` command is followed by a column label, the subtotal is reported and the counter is reset to zero whenever the value of the column changes and the calculation is restarted.

```
BREAK ON (ref1, ref2, ...)
```

It is also possible to define a line or page advance using the `BREAK ON` command. The option `SKIP`

n

causes the output to move forward n lines. The `PAGE[n]` option causes the output to advance n pages, where the default is n = 1. The line and page advances are performed after each subtotal.

To print our list with only one group per page, we will use the `PAGE` option as follows:

```
CALCULATE
   SUM(prevsales) BREAK ON (group) PAGE
```

or

```
   SUM(4) BREAK ON (1) PAGE
```

In the second report, a total is calculated for all customers and all sales. Therefore we will calculate three subtotals using the function `SUM` within the `CALCULATE` command. Because we want a grand total, we will use the `BREAK ON REPORT` command as shown here:

```
CALCULATE
   SUM(2) BREAK ON REPORT,
   SUM(3) BREAK ON REPORT,
   SUM(4) BREAK ON REPORT
```

Because an arithmetic function can contain a list of arguments, we can simplify the command as shown here:

```
CALCULATE
   SUM(2,3,4) BREAK ON REPORT
```

Both ways of structuring the command produce the same totals. The fundamental difference is in the output: Because each BREAK ON command produces a separate break and total, the first method would produce three separate breaks and totals and the output would show each grand total on a separate line. Using the second method would produce only one break and would show the same grand totals but on one line.

The fundamental enhancement to our first example is to include the GROUP BY option within our SELECT command. Our objective is to produce a list showing only the groups and their subtotals.

To do this, we need to group all customers into group A, B, or C with the help of the GROUP BY option:

```
GROUP BY ref1, ref2, ...
```

All entries in the table which have an identical value for the specified column (*ref1, ref2, ...*) are grouped together. In our example this means that all customers are separated by sales into group A, B, or C.

Please note that the result is only one record. In order to clearly match the values of the other selected columns in this result line it is necessary that the columns either contain a constant value or are the result of a calculation. For our example this means that the yearly sales, which are not identical, must be totaled using the SUM function, and the customer names must be counted using the COUNT function.

Shown here is the full SELECT command:

```
SELECT
   group, COUNT(custno), SUM(prevsales), SUM(ytdsales)
FROM customers
GROUP BY 1
```

The last command given is the EXIT command. This command ends the **SQL/R** process.

After the EXIT command you may add comment lines, since all information listed after the EXIT command is ignored.

5.3 Use of Multiple Tables

In the previous examples we focused only on the data in an individual table. Normally you would use the data from several tables to produce a list.

Assume that we want to produce a list of all customer sales orders. For this list we will need the following information:

Table $orders$ contains the following order heading information:

orderno	order number
orderid	ID-number for identifying the line items
custno	customer number
ordertype	order type (here: sales)
orderstat	order status

Table $lineitems$ contains information relating to the ordered items:

orderid	ID-number for identifying the heading information
itemno	item or part number
qty	quantity
price	price per unit
ic	item count code
delivdate	scheduled delivery date

Table $parts$ contains the following information on the parts / items:

partno	part number
descripa	part description (first part)
descripb	part description (second part)

Table $customers$ contains the following customer related information:

custno	customer number
matchcode	customer id key
name1	customer name (first part)
name2	customer name (second part)

We will need to use data from all four tables to produce the list. It is not possible to use the `SELECT` command to retrieve data from multiple tables. Therefore we must find a way to link the four tables respective the data records in them. We can do this by using the `CREATE VIEW` command.

By using the `CREATE VIEW` command, we are able to create a new record type, and therefore a new table, also called a VIEW. This table contains the various record types and

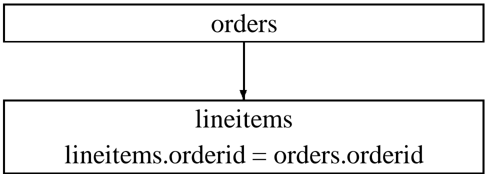
exists only logically, not physically. The various record types are arranged in a specific hierarchy (PATH) and are linked by common data items.

For the first step, we will link the order fields from the *orders* table to the related fields from the *lineitems* table. The common data field is the ID-number *orderid*. The new record type will be called *temp*.

So we build the the following CREATE VIEW command:

```
CREATE VIEW temp PATH orders
   TO lineitems WHERE orderid = orderid;
```

This link can be illustrated as follows:



We can now work with the new table *temp* as shown in the previous examples and define the format of our list. The field *ic* does not have a definite value, but only a code. Therefore we will use the FIELD command to define the item *itemcount* and assign a specific value. In addition we will use the FIELD command to specify the total value of one line item entry and to define the output format. Because the list will only contain sales orders, we define the *ordertype* as “SO”. In summary, we will use the following list of commands:

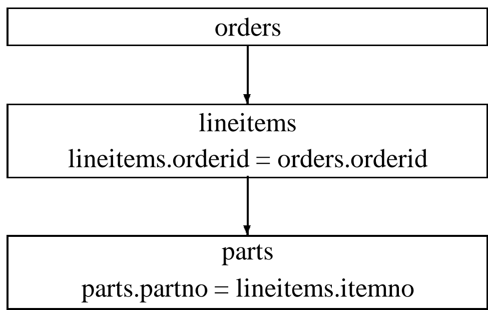
```
OPEN DATABASE "../db/db";

FIELD delivdate  DISPLAY AS DATE( "%d%m%y");

FIELD icnt     = IF (itemcount = "1", 10,
                    IF (itemcount = "2", 100,
                       IF (itemcount = "3", 1000, 1)))
                  DISPLAY AS INT(4);

FIELD amount = (qty * price / icnt)
                 DISPLAY AS MONEY(10, 2);

CREATE VIEW temp PATH orders
   TO lineitems WHERE orderid = orderid;

SELECT
   custno,
   delivdate, orderno, ordertype, orderstat,
   itemno, qty, price, icnt, amount
FROM temp
WHERE ordertype = "VK" AND itemno <> "";
```

man31

The condition *itemno* <> "" appears to be unnecessary, but it is important for the following reason: the `CREATE VIEW` command builds new records even if there are no line items for a given order. In this case, the fields of the line item record part would be empty. In order to limit the report to orders containing line items, it is necessary to include this condition check.

The first enhancement to the list consists of including the item information. To do this, we will expand our use of the CREATE VIEW command. We will broaden the record *temp* to join the table *lineitems* to the item description table *parts*. The common field is therefore the item number, which is called *itemno* in the table *lineitems* and called *partno* in the *parts* table. The new CREATE VIEW command then reads as follows:

```
CREATE VIEW temp PATH orders
   TO lineitems WHERE orderid = orderid
      TO parts WHERE partno = itemno;
```



The recordtype *temp* consists of three record types, which are linked in sequential order. The linking of different records can continue as necessary, as long as common data fields exist.

We can now access the parts information table and expand our command list as follows:

```
OPEN DATABASE "../db/db";

FIELD delivdate  DISPLAY AS DATE("%d%m%y");

FIELD icnt = IF (itemcount= "1", 10,
                    IF (itemcount= "2", 100,
                       IF (itemcount= "3", 1000, 1)))
                  DISPLAY AS INT(4);

FIELD amount = (qty * price / icnt)
                 DISPLAY AS MONEY(10, 2);

CREATE VIEW temp PATH orders
   TO lineitems WHERE orderid = orderid
      TO parts WHERE partno = itemno;

SELECT
   custno,
   delivdate, orderno, ordertype, orderstat,
   itemno, descripa, descripb,
   qty, price, icnt, amount
FROM temp
WHERE ordertype = "VK" AND itemno <> "";
```

man32

To further enhance our list, we will include additional customer information. To access this data, we need to define another link. The customer number *custno* is part of the order heading record *orders*. Therefore we need to join the order heading *orders* to the customer table *customers*. This link will supplement the existing link. We will create this link by using the term AND within the `CREATE VIEW` command. The term, AND, always indicates a new path which has no relationship to the previously defined path.

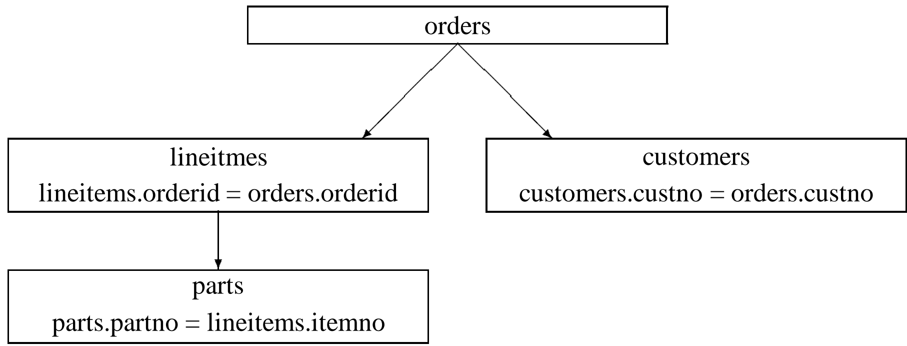
The modified `CREATE VIEW` command is arranged as follows:

```
CREATE VIEW temp PATH orders
   TO customers WHERE custno = custno
   AND lineitems WHERE orderid = orderid
      TO parts WHERE partno = itemno;
```

An alternative arrangement for this command is:

```
CREATE VIEW temp PATH orders
   TO (lineitems WHERE orderid = orderid
      TO parts WHERE partno = itemno)
   AND customers WHERE custno = custno;
```

The parentheses are required in the alternative arrangement in order to clearly define the connection. Without the parentheses the term `AND` would apply to the second `TO` statement, creating an incorrect link. This would result in an error message, because the $parts$ table has no $custno$ field. In certain cases, instead of an error you could produce an incorrect list.



After modifying the `CREATE VIEW` command we can expand the `SELECT` command to include the columns we want to display, as shown here:

```
OPEN DATABASE "../db/db";

FIELD delivdate  DISPLAY AS DATE("%d%m%y");

FIELD icnt = IF (itemcount = "1", 10,
                    IF (itemcount = "2", 100,
                       IF (itemcount = "3", 1000, 1)))
                  DISPLAY AS INT(4);

FIELD amount = (qty * price / icnt)
                 DISPLAY AS MONEY(10, 2);

CREATE VIEW temp PATH orders
   TO customers WHERE custno = custno
   AND lineitems WHERE orderid = orderid
      TO parts WHERE partno = itemno;

SELECT
   custno, matchcode, name1, name2,
   delivdate, orderno, ordertype, orderstat,
   itemno, descripa, descripb,
   qty, price, icnt, amount
FROM temp
WHERE ordertype = "VK" AND itemno <> "";
```

man33

Now that we have defined what we want to display, the remaining task is to define how this data will be displayed. This includes defining the format, the sort order, the usage of the `REPORT` command, and the calculation and display of subtotals. We will create a form file to define the output format. The use of a form file allows us to define and save complex report formats involving many fields. In addition, the command list is more readable.

The results will be sorted by customer number. Within each customer order, the individual line items will be sorted by scheduled delivery date where identical delivery dates occur, and the line items will be further sorted by order number. A subtotal will be displayed for each customer. Page headers will display the customer information for that page and there will be a page break after each customer. The final page of the list will contain a grand total for the report.

For the order status we will use the `VALUES ARE` rule of the `FIELD` command. The item *orderstat* is a coded data field, therefore the field content is a code with a specific meaning. The `VALUES ARE` rule allows you to convert this code into a more readable format in the list.

The complete list of commands for this example is as follows:

```
OPEN DATABASE "../db/db";

SET DATE = "%d.%m.%y";

FIELD delivdate  DISPLAY AS DATE("%d%m%y");
FIELD orderno    DISPLAY AS (10);
FIELD qty        DISPLAY AS DOUBLE(6, 0);
FIELD price      DISPLAY AS MONEY(8, 2);

FIELD status = orderstat
               VALUES ARE ( 0 = "OPEN",
                            5 = "IN PROCESSING",
                            6 = "RELEASED TO AB",
                            7 = "AB PRINTED",
                            8 = "RELEASED TO LS",
                            9 = "RELEASED TO RG",
                           10 = "INVOICE PRINTED",
                           12 = "ACCOUNTING NOTIF.",
                           13 = "TRANSACTION COMPL.")
               DISPLAY AS LEFT(18);

FIELD itemcount = IF (ic = "1", 10,
                    IF (ic = "2", 100,
                       IF (ic = "3", 1000, 1)))
                  DISPLAY AS INT(4);

FIELD amount = (qty * price / itemcount)
                DISPLAY AS MONEY(10, 2);
```

```
CREATE VIEW temp PATH orders
   TO customers WHERE custno = custno
   AND lineitems WHERE orderid = orderid
      TO parts WHERE partno = itemno;

REPORT
   SELECT
      custno, matchcode, name1, name2,
      delivdate, orderno, ordertype, status,
      itemno, descripa, descripb,
      qty, price, itemcount, amount
   FROM temp
   WHERE ordertype = "VK" AND itemno <>""
   ORDER BY 1, 5, 6
CALCULATE
   SUM(15) BREAK ON (1) PAGE,
   SUM(15) BREAK ON REPORT
USING "man34.frm";
```

man34

The form file used in this example produces the following format:

```
                SALES ORDERS BY CUSTOMERS                      Page: $page
        Sorted by delivery date and order number               Date: $date

Customer number: @1          Name: @3
Matchcode      : @2                @4
-----------------------------------------------------------------------------------
DELIV.  ORDER NUMBER        ITEM NUMBER            QTY       PRICE/  IC      AMOUNT
DATE    STATUS              DESCRIIPTION
-----------------------------------------------------------------------------------
%% End of heading
@5      @6                  @9                    @12      @13      /@14   @15
        @8                  @10
                            @11
%% End of detail line
Total for Customer  @1                                              *** @15
%% End of break section   SUM(15) BREAK ON 1 PAGE

T O T A L   A M O U N T:                                            *** @15
%% End of break section   SUM(15) BREAK ON REPORT
```

man34.frm

5.4 Summary

The goal of this chapter was to present the most important features of the **SQL/R** language through specific examples. For information on using syntax not covered in this chapter, for example some arithmetic functions, please see the reference section of this manual. The reference section includes a complete description of the syntax, including some simple examples that aren't always applicable to our example database.

To produce a list, first retrieve the database entries. When the entries have been correctly selected, the calculations and links can be tested. Next perform the sort (`ORDER BY`) and grouping (`GROUP BY`) on these entries. The sort and grouping functions are performed on the retrieved entries before the output is produced. Depending on the complexity, processing these functions can take a long time.

Formatting the report should always be the last step in developing a list and should not be started before all data has been correctly generated.

In summary, the basic procedure for producing a report is as follows:

- Start **SQL/R** by entering the $sqlr$ command from the HP-UX shell prompt.
- Define the necessary links of the tables using the `CREATE VIEW` command.
- Define the necessary virtual fields, including the appropriate calculations and definitions using the `FIELD` command.
- Select the necessary data fields using the `SELECT` command and the `WHERE` condition.
- Test and correct the command list until the results are correct.
- Use the `SELECT` command within the `REPORT` command. Define subtotals and totals using the `CALCULATE` rule.
- Define output formats using the `FIELD` command and in some cases the `DISPLAY AS` rule.
- Enhance the `REPORT` command by using a form file (`USING` $filename$).
- Create the form file.
- Test the `REPORT` command by using the form file and checking for error-free output.
- Add the `ORDER BY` and `GROUP BY` options to the `SELECT` command.
- Finally define of the output device, the page width and length.

6

Reference

This chapter describes the use of the **SQL/R** module and includes a definition of the elements of the **SQL/R** language:

- Reserved words
- Identifiers
- Constants
- Arithmetic expressions
- Character strings
- Conditions
- Commands

This chapter is designed as a reference work. It is not a tutorial of the **SQL/R** language.

For an introduction to **SQL/R**, see chapter 3. For explanation of how to create a report, see chapter 5.

6.1 Starting of SQL/R

The product **SQL/R** consists of two modules:

- the user interface `sqlr` (and `sqlred`)
- the execution module `sqlrexec`

The creation of a database query is initiated through the `sqlr` user interface. The `sqlr` user interface is a shell script which can be customized. It calls the `sqlred` binary program. Pressing the function key labelled “Start **SQL/R**” starts the `sqlrexec` execution module with the actual text.

The `sqlr` command syntax is shown here:

```
Usage: sqlr [-d database] [-p password] [file]
```

You can use the `-d` and `-p` options to reference a database and a database password, respectively. This database is then opened each time `sqlrexec` is initiated. In this case the `OPEN DATABASE` command must not be used in a query.

for example:

```
sqlr customer
```

The `sqlrexec` command syntax is shown here:

```
Usage: sqlrexec [-e][-n][-d dbnm][-p pswd] [batchfile [arg ...]]
options:
 -help   = show usage (this list)
 -e      = echo batch processing
 -n      = suppress program banner
 -d dbnm = specify database name and path
 -p pswd = specify database password

If batchfile is not present, input will be requested from stdin.
Optional arguments will be passed to batchfile as $1 ... $8.
```

You use the `-d` and `-p` options to open a database and enter the password. In this case the `OPEN DATABASE` command is not available.

The `-e` option displays each line that is processed as it is entered.

The `-n` option suppresses the program banner for the report.

The first argument is the batch file name. If a batch file is specified, the report is produced automatically. All other arguments are treated as optional arguments $1 through $8, usable in the batch file e.g. to specify ranges for data selection. These optional arguments are overwritten when the `RUN` command is used.

for example:

```
sqlrexec -n customers 1000 2000
```

6.2 Definition of Terms

Field (or Item)

A field is the smallest logical unit of a database. Its contents are not limited to a word or a numeric value, but can consist of several words,

for example Street: Martin Luther King Boulevard

Array

An array is a group of fields of the same type (also called elements) that can be referenced with the same name and an index:

For example, when there are 12 values for monthly budget, the month of May: budget[4], the month of January: budget[0]. The index of the first element is zero.

Record (or Entry)

A record is a collection of fields, and includes the access methods and dependencies. Each field in a record has a unique name. Records are stored in tables.

For example: A customer record consists of: number, name, address, etc.
An obvious way to access a customer record is by using
the customer number

Table (or Dataset)

A table is a collection of records, arranged in columnar form.

Field Reference

A field reference is the name of a field and, optionally, fully referenced by adding a table name. The complete reference is important when the same field exists in more than one table:

for example custno, orders.custno or customers.custno

View

A view is a virtual table. In the simplest case, a view is a single table. The `CREATE VIEW` command allows you to create a view consisting of several tables. The view then appears as a single table that contains all the data fields of the individual tables.

Ocurrence

In cases where a single table is referenced several times in one view (for example, access to an article using its parts list header and positions), it is necessary to define an alternate name for each occurrence of a data item. This is to differentiate between

fields with the same name. This alternate name is different from an alias, because the alternate name is not merely an additional name for the same data, but rather an access to different field contents as well.

Path

A path is the (logical) link between data tables. The type of link must be predefined (in the database schema) before you use the `CREATE VIEW` command to link several tables.

Alias

An alias is a pseudonym (alternate name) for a database field and is defined using the `FIELD` command (see page 108).

6.3 Reserved Words

Reserved words are **SQL/R** predefined words with a special meaning. These words are not case sensitive.

Reserved Words			
ALL	DOUBLE	MIN	SKIP
AND	EXIT	MONEY	STRLEN
ARE	FIELD	MONTH	SUBSTR
ASC	FILE	NOT	SYSDATE
ASCII	FIXED	NULL	SUM
AVG	FLOAT	OPEN	TERMINAL
BETWEEN	FROM	OCCURRENCE	TIME
BREAK	GROUP	OF	TITLE
BY	HAVING	ON	TO
CALCULATE	HELP	OR	TODAY
CENTER	IF	ORDER	TRANSLATE
CLOSE	IN	OUTPUT	TRIM
COUNT	INT	PAGE	UPPER
CREATE	INTO	PATH	USING
DATABASE	LEFT	PRINTER	VALUES
DATE	LENGTH	REPORT	VIEW
DAY	LIKE	RIGHT	WHERE
DEFINE	LOCALE	RUN	WIDTH
DESC	LONG	SELECT	XOR
DESCRIBE	LOWER	SET	YEAR
DIF	MACRO	SHORT	
DISTINCT	MAX	SHOW	

6.4 Data Types

The HP Eloquence database supports the following data types:

Data Type		Description
String	Xn	a character string consisting of any chars
Integer	I	$-32768 \ldots 32767$
DInteger	D	$-2^{31} \ldots 2^{31}-1$
Short	S	floating point number, 6 digits
Long	L	floating point number, 12 digits

SQL/R uses its own data types, which include the HP ELOQUENCE data types.

SQL/R supports the following data types:

Data Type	Value Range	HP Eloquence
char	a character string of any chars except binary zero	String
short	$-32768 \ldots 32767$	Integer
int	$-2^{31} \ldots 2^{31}-1$	DInteger
long	$-2^{31} \ldots 2^{31}-1$	DInteger
float	floating point number, 7 digits	Short
double	floating point number, 15 digits	Long
date	short, int, long, float, double	
time	short, int, long	
fixed	short, int, long	
money	float, double	

The data types `DATE`, `TIME`, `FIXED`, and `MONEY` are *logical* data types. They describe, how the corresponding field contents are *interpreted* and presented. They don't describe the internal storage format nor the value range (see also `FIELD` command, page 108).

`DATE` The field contents are presented as a date. The internal format and output format are defined by using the `FIELD` command.

`TIME` The field contents are presented as time (HH:MM). The corresponding field type must be short, long, or integer and be defined as follows: field = 100 * hours + minutes.

`FIXED` The field contents are presented as a fixed point number. The corresponding field type must be short, long, or integer. The field contents are *divided* by

10^n and then output. The n represents the desired number of positions after the decimal point.

`MONEY` The field contents are presented as a monetary amount. The output format depends on the configured user environment (language).

Note: Character strings are internally ended with a null character. Therefore, it is impossible to correctly display a character string that contains such a binary zero.

Note: The floating and double data types support the representation of exponents (e.g. 1E10). This is not possible with **SQL/R**.

6.5 Identifiers

An identifier consists of a maximum of 31 characters. These characters can be alphabetic, numeric, or an underline (_). The identifier must always start with an alphabetic character.

Identifiers are not case sensitive. Therefore the identifiers “Name”, “NAME”, “name” are treated the same.

Identifiers can be used for all expressions such as table and fields names. The only limitation is that no **SQL/R** reserved words are permitted.

Identifiers consisting of a reserved word must be preceded by an underline (_). For example “_time” is an identifier and not a reserved word.

6.6 Constants

Constants are values that are constant, regardless of database values. Constants can consist of various data types, e.g. numbers, character strings, dates, times.

6.6.1 Numeric Constants

Numeric constants have the following format:

[+/−]nnn[.nnn]

Constants containing a decimal point are treated as double data types. All other numeric constants are treated as integer values.

The period ’.’ is used to represent the decimal point, regardless of the `LOCALE` value selected.

1234	integer
−123	integer
12.34	double
1.2345	double
−1234.567890	double
123456	integer

6.6.2 Character String Constants

A character string constant is a character string that begins and ends with quotation marks. Either single or double quotation marks can be used. Note that the same quotation mark must be used at both the beginning and the end of the string.

A character string constant can contain a maximum of 511 characters.

To use a quotation mark as a literal within the string, precede the quotation mark with a backslash (\).

```
'This is a character string'
"This is a \"new\" string"
```

6.6.3 Date Constants

Date constants can consist of either the European or the American format. Date constants are reformatted into an internal format. This is necessary for performing operations such as comparisons involving database fields that have been defined as of type date with the `FIELD... DISPLAY AS DATE` command.

@MM/DD/YY	American format
@DD.MM.YY	European format

A zero (0) can be used to represent a null date.

6.6.4 Time Constants

Time constants are reformatted into an internal format. This is necessary for performing operations such as comparisons and caluculations. Time constants are represented using the following formats:

@HH:MM
@HHMM

6.7 Arithmetic Expressions

Arithmetic expressions are used to perform calculations involving database variables and constants.

Arithmetic Expression = Operand [Operator Operand] . . .

$$\text{Operand} = \left\{ \begin{array}{l} \text{Constant} \\ \text{Field Reference} \\ \text{Alias} \\ Function \\ \text{(Arithmetic Expression)} \end{array} \right\}$$

Operator = { + | – | * | / }

$$Function = \left\{ \begin{array}{l} \left\{ \begin{array}{l} \texttt{AVG} \\ \texttt{MAX} \\ \texttt{MIN} \\ \texttt{SUM} \end{array} \right\} \texttt{(} \text{ [} \texttt{ALL} \text{ | } \texttt{DISTINCT} \text{] Arith. Expression } \texttt{)} \\ \texttt{COUNT(} \text{ [ALL | DISTINCT] \{ * | Field Name \} } \texttt{)} \\ \texttt{STRLEN(} \text{ String Expression } \texttt{)} \\ \left\{ \begin{array}{l} \texttt{DAY} \\ \texttt{MONTH} \\ \texttt{YEAR} \end{array} \right\} \texttt{(} \text{ Date Field } \texttt{)} \\ \texttt{IF(} \text{ Condition, Arith. Expression, Arith. Expression } \texttt{)} \end{array} \right\}$$

A field reference is the name of a database item. When the field is an array, the field reference refers to a single element of the array. If the individual element is not specified, then the first element of the array is used.

A field name can be used in several tables, views, and databases. If several tables are used in a command, you must distinguish the field names by preceding the field name with the table name (e.g. item.number and customer.number).

Order of precedence for operators:

()	parentheses
+, −	positive/negative marks
*, /	multiplication and division
+, −	addition and subtraction

Operators with equal priority are calculated from left to right.

6.7.1 Arithmetic Functions

`SUM(` [`DISTINCT`] arith. expression `)`

calculates the total of the arithmetic expression for all selected entries. This function can only be used with numeric expressions (including time values).

`AVG(` [`DISTINCT`] arith. expression `)`

calculates the average value of the arithmetic expression for all selected entries. This function can only be used with numeric expressions (including time values).

`MIN(` arith. expression `)`

calculates the minimum value of the arithmetic expression for all selected entries. This function can only be used with numeric expressions (including date and time values).

`MAX(` arith. expression `)`

calculates the maximum value of the arithmetic expression for all selected entries. This function can only be used with numeric expressions (including date and time values).

COUNT([DISTINCT | ALL] {* | field name})

counts the number of entries.

STRLEN(string expression)

calculates the length of the specified character string.

These arithmetic functions are used to develop the results of a SELECT command calculation. The sets of entries needed for the calculations are produced by using either the GROUP BY statement within a SELECT command or by using the CALCULATE statement of a REPORT command.

When the DISTINCT option is used, only unique entries are used to calculate the results.

The DISTINCT option can only be used once within a SELECT command.

Examples of arithmetic expressions and functions are shown here:

```
SELECT SUM( items.quantity * price );
SELECT AVG( DISTINCT customers.turnover[1] );
SELECT COUNT(*) FROM customers;
```

6.7.2 Date Functions

MONTH(date field)	returns the month (1-12)
YEAR(date field)	returns the year (4 digits)
DAY(date field)	returns the day (1-31)

The date functions produce integer values that can be used for calculations. The parameter is always a date, the result of the function is the day, month or year extracted from this date.

An example of a date function is show here:

```
SELECT YEAR(orders.date), items.sales
   FROM orders;
```

6.8 String Expressions

String expressions are specified the same way as string constants and fields.

string expression = operand [& operand] . . .

operand = { string constant | field reference | alias | function }

function = { { UPPER | LOWER | TRIM } (string expression) | SUBSTR(string expression , start , length) | IF(condition , string expression , string expression) }

The ampersand (&) operator is used to connect several character strings.

6.8.1 String Functions

`UPPER(` string expression `)`

shifts all lower case characters to upper case. Characters with an umlaut (") are handled as specified by the configured environment (see appendix D).

`LOWER(` string expression `)`

shifts all upper case characters to lower case. Characters with an umlaut (") are handled as specified by the configured environment (see appendix D).

`TRIM(` string expression `)`

right truncates all blanks (space characters) back to the last nonblank character.

`SUBSTR(` string expression `,` position `,` length `)`

uses the specified string to produce a substring. The substring starts at the position specified and continues for the length specified. Note that the first position of a character string is 0.

Examples of string expressions and functions are shown here:

```
"Mr. " & customers.name
UPPER("New Text")
TRIM(SUBSTR( customers.name, 0, 20) & customers.firstname)
```

6.9 Condition Functions

The `IF` command is used to select one of two possible expressions based on the result of a logical expression. Several `IF` commands can be nested or combined with a `SELECT` command.

`IF(` condition `,` expression `,` expression `)`

When the condition is TRUE, then the first expression is activated. If the condition is FALSE, then the second expression is evaluated. Both expressions must produce a result of the same data type.

```
    SELECT IF (quantity < 100, rebate[0],
               IF (quantity < 500, rebate[1], rebate[2]))
             "actual rebate"
       FROM orders
```

6.10 Conditional Expressions

Conditional (logical) expressions result in a value of TRUE or FALSE. These expressions are part of `IF` and `WHERE` commands or `HAVING` rules within the `SELECT` command.

Only those entries for which the conditional expression is true within the `SELECT` command are further processed. Conditional (logical) expressions are defined as shown here:

$$\begin{array}{l} \textit{Conditional} \\ \textit{Expression} \end{array} = \textit{logical operand} \left[\left\{ \begin{array}{l} \texttt{AND} \\ \texttt{OR} \\ \texttt{XOR} \end{array} \right\} \textit{logical operand} \right] \ldots$$

$$\textit{logical operand} = \left\{ \begin{array}{l} \text{expression [} \texttt{NOT} \text{] } \textit{logical operator} \text{ expression} \\ \text{expression} \quad \text{[} \texttt{NOT} \text{] } \texttt{BETWEEN} \text{ constant} \\ \qquad\qquad\qquad \texttt{AND} \text{ constant} \\ \text{expression [} \texttt{NOT} \text{] } \texttt{IN(} \text{ constant [, \ldots] } \texttt{)} \\ \text{string expression [} \texttt{NOT} \text{] } \texttt{LIKE} \text{ "pattern"} \\ \texttt{(} \textit{ conditional expression } \texttt{)} \end{array} \right\}$$

$$\begin{array}{l} \textit{logical} \\ \textit{operator} \end{array} = \left\{ \begin{array}{ll} = & \text{(equals)} \\ < & \text{(less than)} \\ > & \text{(greater than)} \\ <= & \text{(less than or equal to)} \\ >= & \text{(greater than or equal to)} \\ <> & \text{(not equal to)} \end{array} \right\}$$

Boolean operators

`NOT`	(TRUE, when the operand is FALSE)
`AND`	(TRUE, when both operands are TRUE)
`OR`	(TRUE, when one or both operands are TRUE)
`XOR`	(TRUE, when only one of the operands is TRUE)

Results of linking with Boolean operators:

operand 1	operand 2	AND	OR	XOR
TRUE	TRUE	TRUE	TRUE	FALSE
TRUE	FALSE	FALSE	TRUE	TRUE
FALSE	TRUE	FALSE	TRUE	TRUE
FALSE	FALSE	FALSE	FALSE	FALSE

The order of precedence for Boolean operators is shown here:

NOT
AND
OR, XOR

Operators of equal value are evaluated from left to right.

A conditional expression consists of a set of comparisons which produce either a TRUE or FALSE result and which are linked by Boolean operators. Each comparison may itself consist of a conditional (logical) expression.

In addition to the general comparisons (=, <, <=, >, >=, <>), there are three special comparisons, `BETWEEN ... AND`, `IN` and `LIKE`.

`BETWEEN ... AND` determines, whether or not the value produced by an expression falls within a specified range.

The operator `IN` produces a value of TRUE when the value produced by an expression appears in a previously defined list of specific values.

The `LIKE` operator allows the use of wildcards in string expressions. Each question mark (?) in the character string represents one character, an asterisk (`*`) is used to represent any number of characters. The specified expression must represent a character string.

```
SELECT customer, item FROM orders
       WHERE item BETWEEN 1000 AND 9000
             OR customer LIKE "M*er";
SELECT customer, item FROM orders
       WHERE item IN (1000,2000,3000);
```

6.11 The CLOSE DATABASE Command

```
CLOSE DATABASE ;
```

The `CLOSE DATABASE` command closes all open databases. This is necessary in order to be able to specify a new `OPEN DATABASE` command.

When **SQL/R** is exited, all open databases are automatically closed.

6.12 The CREATE VIEW Command

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

occur_spec = [OCCURRENCE occur_name OF | occur_name =] record_name

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

path_element = { (*path_element path_group*) | *occur_spec* WHERE field_name = [occur_name.] field_name }

The CREATE VIEW command is used to define a certain view in the database. This view will build a logical record made up from the fields of various tables. It exists only logically, and not physically, in the database.

Each view has a *view_name* and consists of records from various tables linked by a hierarchy (called PATH) with common data items (fields).

The view created using the CREATE VIEW command is treated as if it were a real table where each record contains all the data items (fields) defined in the records you included in the hierarchy.

The PATH rule specifies the access order of the records contained in the view. In addition, the PATH defines the hierarchy within the view. The PATH follows a line from the first record to the last record, where records are linked through a common data field. The first record *record_name* can be assigned a new name within the view by using either the OCCURRENCE *occur_name* OF or *occur_name* = option.

Several records on the same level of the hierarchy can be linked by using the AND rule. In some cases, it is necessary to use parentheses to preserve the hierarchy in the view. The use of parentheses is shown in the examples that follow.

The WHERE rule is used to specify which fields are the common data items to form the link.

For example:

In this first example, the tables *orderhead* and *orderpos* are linked by using the common item *order_no*.

```
CREATE VIEW orders
```

```
PATH orderhead
TO orderpos WHERE order_no = order_no;
```

The next example shows a continuation of the hierarchy definition. In this example, the view is a combination of the records *orderhead*, *orderpos* and *items*.

```
CREATE VIEW orders_and_items
PATH orderhead
TO orderpos WHERE order_no = order_no
    TO items WHERE item_no = item_no;
```

The following examples illustrate the branching of a path. The view orders_and_customers consists of linkages of equal priority, namely *orderhead* with *orderpos* and *orderhead* with *customers*.

```
CREATE VIEW orders_and_customers
PATH orderhead
TO orderpos WHERE order_no = order_no
AND customers WHERE cust_no = cust_no;
```

The next example demonstrates how to use parentheses to achieve specific results.

```
CREATE VIEW orders_items_text
PATH orderhead
TO (orderpos WHERE order_no = order_no
   TO items WHERE item_no = item_no)
AND text WHERE text_no = text_no;
```

In the previous example, orderhead was linked with orderpos, then orderpos was linked with items, and finally orderhead was linked with text.

```
CREATE VIEW orders_items_text
PATH orderhead
TO orderpos WHERE order_no = order_no
TO items WHERE item_no = item_no
AND text WHERE text_no = text_no;
```

In this example, orderhead is linked with orderpos, then orderpos is linked with items and finally orderpos is linked with text.

In cases where there are multiple references to the same table within one line, it is necessary to assign an individual name to each occurence of the data record.

```
CREATE VIEW items_items
PATH items
TO OCCURRENCE material OF items
   WHERE item_no = material_no;
```

In this example, the data field $material_no$ of the table items is used for a second access to the table items. For this second access, the data record is referenced by the temporary name $material$.

6.13 The DEFINE Command

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

The `DEFINE` command enables you to use a short notation ("*macro_name*") to represent specific text. These short notations are called macros. During processing the macro is automatically replaced by its qstringdefinition text.

The *macro_name* can consist of any words except reserved words, or existing table names, field names, view names, etc.

Macro names which are enclosed in quotation marks are not expanded.

Macros can be nested so that one macro can reference other macros. The maximum number of nesting levels is 8 levels.

The maximum length of a macro definition (the text represented by a macro) is 511 characters. To use quotation marks within a macro definition, precede each quotation mark with a backslash (\).

Example:

```
DEFINE cust_fields AS
   "customers.no, customers.name, customers.city";
DEFINE cust_list AS
  "SELECT cust_fields FROM customers ORDER BY customers.no"
```

Note that the first macro is nested within the second macro definition.

6.14 The EXIT Command

`EXIT;`

The `EXIT` command ends the **SQL/R** process.

In a batch file, all lines after the `EXIT` command are ignored. This feature can be used for comment lines.

6.15 The HELP Command

`HELP` [$\{identifier \mid string\}$]`;`

The `HELP` command can be used alone or with an $identifier$ or $string$. When the command is used alone, a short description of the **SQL/R** syntax is displayed. When the `HELP` command is followed by an identifier or string, the command shows if the identifier or string is a field, record, view, macro (strings only) or alias.

When the type represented by the identifier or string is known, you can use the `SHOW` command to get complete information about it.

6.16 The FIELD Command

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

           ⎧ (field length)                                        ⎫
           ⎪ INT( field length )                                   ⎪
           ⎪ LONG( field length )                                  ⎪
           ⎪ FLOAT( field length, decimals )                       ⎪
           ⎪ DOUBLE( field length, decimals )                      ⎪
format =   ⎨ FIXED( field length, decimals )                       ⎬
           ⎪ MONEY( field length [, decimals ] )                   ⎪
           ⎪ DATE   [ ( "date_format" [, field length ] ) ]        ⎪
           ⎪        [ FROM { SYSDATE | YYYY } ]                    ⎪
           ⎩ TIME [ ( field length ) ]                             ⎭
```

The FIELD command can be used in the following ways:

- to define an alternate name or pseudonym for a field or expression
- to establish a value for a coded data field
- to specify the output format of a data field

You can also use the DESCRIBE AS rule to describe fields. This description is displayed by using the SHOW FIELD command.

6.16.1 FIELD and Expression Pseudonyms

Pseudonyms are defined using the "$alias = expression$" parameter of the FIELD command. The $alias$ is a name that is used to represent an $expression$ in a SELECT command.

In the simplest case, $expression$ is the name of a data field. You can also use several pseudonyms to represent a single data field. Pseudonyms are often used to define more descriptive names for data fields (i.e. elements of an array) or to define different output formats for an item.

If you use the `FIELD` command to define a pseudonym for a data field, you can use the `VALUES ARE` rule. However, you can not use the `VALUES ARE` rule when defining a pseudonym for an expression.

Examples:

```
FIELD part_number = items.item_no;
FIELD salesJanuary = customers.turnover[0];
FIELD salesFebruary = customers.turnover[1];

FIELD salesMay = customers.turnover[4]
 DESCRIBE AS "May Sales";
```

6.16.2 The VALUES ARE Rule

The `VALUES ARE` rule allows you to translate data values in a specified field to other values.

A coded value is either a character string or a number (`SHORT`, `INT` or `LONG`).

A "translated" value can be defined for each coded value.

The following conditions apply:

- A pseudonym (alias) must be specified for the data field. Accessing this alias will return the translated value. A reference to the (original) field name will return the (untranslated) coded value.
- The `DISPLAY AS` rule defines the maximum width of the result.
- Coded values with no specified replacement text are converted to an empty field.

For example:

```
FIELD color = colornum
 VALUES ARE (0 = "NONE", 1 = "RED", 2 = "YELLOW", 3 = "BLUE")
 DISPLAY AS LEFT(7);
```

In this example, the coded values are used for the alias $color$. The $colornum$ data field remains unchanged.

```
FIELD street = customers.street
 VALUES ARE ("st"= "street", "rd" = "road", "dr" = "drive")
 DISPLAY AS LEFT(10);

FIELD city = customers.city
VALUES ARE (1 = "New York", "Chicago", "Denver",
                "San Francisco", "Seattle",
            20 = "Atlanta")
DISPLAY AS LEFT(20);
```

The *city* field contains a coded value between 1 and 20, where value=1 represents "New York" and value=20 represents "Atlanta". The values 6 through 19 have not been defined.

If the coded value is numeric, you can define a sequence of values and a starting value. If no starting value is defined, then the first numeric value is 0. The remaining numeric values follow in ascending order from left to right. Alternately, you can define a specific value in the list, in which case the next value to the right is incremented by 1.

6.16.3 The DISPLAY AS Rule

The `DISPLAY AS` rule defines the output format of the data fields or expressions. The output of a value can be defined within the output width as `LEFT` justified, `CENTER` justified, or `RIGHT` justified. If the actual width of a value is wider than the defined output format, then the output will be truncated without an error message. The `DISPLAY AS` rule is important for correctly displaying `DATE`, `FIXED`, and `MONEY` values.

The following table illustrates the **SQL/R** supported data types and their default width, default number of decimal places and the default justification:

data type	output width	decimal places	justification
char	string length	–	left
short	6	0	right
int	11	0	right
long	11	0	right
float	11	2	right
double	16	2	right

The formats `FIXED`, `MONEY`, `DATE` and `TIME` are not available in the HP ELOQUENCE database. Therefore it is necessary to define these formats using the `FIELD ... DISPLAY`

AS command[1].

The FIXED data types are stored as INT or LONG values. Therefore it is necessary to use the DISPLAY AS FIXED(...) command to define the number of decimal places in the output.

The MONEY data types are formatted according to the work environment, which is determined by the selected language (see Appendix D).

The DISPLAY AS DATE rule is used to define both the input and the output date formats. The format string *date_format* contains a user specified date and time format (see Appendix B). The date format string also can contain other user specified text to be output as a date (e.g. "Today is %d.%m.%y"). If the date format is not defined, the date format defaults to the value set using the SET DATE command. In addition, the output width can be defined.

You can also use the FROM option to define the format in which the date is stored in database.

Syntax	Description	Data type
default	YYMMDD	LONG
FROM SYSDATE	number of seconds since Jan 1,1970	LONG
FROM YYYY	number of days since Jan 1, YYYY	SHORT, INT, LONG

Fields containing a time value in the form HHMM can be displayed using the DISPLAY AS TIME command.

[1] It is also possible to define these formats using the format numbers contained in the schema file or specified with DBMODS (see Appendix E).

6.17 The OPEN DATABASE command

`OPEN DATABASE` "database" [`AS` "password"] [, . . .] `;`

A database must be opened before it can be used. The `OPEN DATABASE` command is used to open the database. You can also specify the path and password for the database using this command.

For example:

```
OPEN DATABASE "abc";
OPEN DATABASE "/usr/pps/db/pps";
OPEN DATABASE "/usr/sad/sad" AS "SECRETARY";
OPEN DATABASE "DB1" AS "ALL", "DB2" AS "ALL";
```

Before you open any additional databases, you must first close all open databases. The `CLOSE DATABASE` command is used to do this.

6.17.1 Multiple Databases

When several databases are open, conflicts in field and table names can occur. For example, a field called “NR” can occur in several databases. The same conflict can occur with table names.

If the same field name occurs in several databases, it is important that you always reference the item using both the table name and the field name, so that the correct field is used (e.g. orders.part_no).

If the same table name occurs in several databases, **SQL/R** joins the table name and the database name using an underline character (_). For example, the table CUSTOMERS in database DB1 is referenced as CUSTOMERS_DB1.

6.17.2 The QIF File

When a database is opened, **SQL/R** checks for the existence of a file called *database name*`.qif`. If this batch file is found, the commands in the batch file are executed. Therefore you can use this `.qif` batch file to automatically execute certain (e.g. `FIELD`) commands, when the database is opened.

The QIF file name must be in the following format:

`databasename.qif` oder `DATABASENAME.QIF`

The name must be either all upper case or all lower case.

SQL/R searches for the existence of a QIF file in the following search order:

database path
path specified through environment variable QPATH
local directory

6.18 The REPORT Command

```
REPORT SELECT  [ CALCULATE field_calc [, ... ] ]
               [ INTO { TERMINAL
                        PRINTER
                        [ ASCII | DIF ] FILE "filename" } ]
               [ report_fmt ]
               [ USING "report_form" ] ;

field_calc =  [ { SUM
                  AVG
                  MIN
                  MAX
                  COUNT } ( field_ref [, ... ] ) ["row label"] ]
              BREAK ON { ( field_ref [, ... ] )
                         REPORT }  [ SKIP [n]
                                     PAGE [n] ]

report_fmt =  [ TITLE AS "report title" ]
              [ DATE AS { TODAY | "date string" } ]
              [ LENGTH = num ]
              [ WIDTH = num ]
```

The REPORT command is used to format the results produced by a SELECT command. A report is created according to the user requirements. The optional rules of the REPORT command allow you to execute the following functions:

- Calculate field values including subtotals and totals
- Direct the output to various output devices
- Format the output using various options
- Create and use specific format files to define the output

6.18.1 The CALCULATE Rule and the BREAK ON Rule

The CALCULATE rule is used to perform calculations on the item values of the data fields retrieved using the SELECT command. The results of the calculations are further processed in the report.

The following calculations can be used:

`SUM`	= the sum of all values
`AVG`	= the average of all values
`MAX`	= the maximum value
`MIN`	= the minimum value
`COUNT`	= the number of values

The calculations are specified using arithmetic functions and the fields. The list of fields is enclosed in a set of parentheses. These fields are referenced using either the field name or the position number of the field as it appears in the previous `SELECT` command. You can execute several calculations within one `REPORT` command. The individual calculations are separated by commas. The results of these calculations are displayed in one line in the order that they appeared in the `REPORT` command.

The `BREAK ON` rule allows you to define which fields are used for the calculations. You specify the field references the same way as in the `CALCULATE` rule. All results of the `SELECT` command are grouped by identical values for the fields defined in the `BREAK ON` command. Each calculation is performed using the values of one of these groups. When the value of a field in the `BREAK ON` field list changes, a break occurs and the results of the calculation are reported. After the results are reported, the calculations are performed on the next group and these results are displayed. To perform a calculation on all the results produced by the `SELECT` command, use the `BREAK ON REPORT` rule.

You can also use the `BREAK ON` rule to define a line-break or page-break. The `SKIP[`n`]` option advances the report output by n lines. Similarly the `PAGE[`n`]` option advances the report output n pages. If no number is specified after the `SKIP` or `PAGE` option a default value of one is used. The `BREAK ON` rule allows you to define line and page breaks without performing any calculations.

If the `BREAK ON` rule contains a list of field references, then the `SELECT` command should be ordered by these fields.

When the results of the calculations are reported, the calculation function used is displayed at the end of the line. This function name can be replaced with your own text which must be listed directly after the `CALCULATE` rule.

For example:

The SUM(3) statement is a short way of specifying a total. The “3” indicates the third field in the SELECT command (amount) counting from left to right.

```
REPORT
 SELECT company, orderno, amount, month
  FROM orders ORDER BY company, month
```

```
CALCULATE
  SUM(3)
    BREAK ON ( month, company ) SKIP 3,
  SUM(3) "Sales per Company"
    BREAK ON ( company ) SKIP 3,
  SUM(3) "Total Sales"
    BREAK ON REPORT PAGE,
  COUNT(orderno) "Number of Orders"
    BREAK ON ( company),
  COUNT(orderno) "Number of Orders"
    BREAK ON REPORT;
```

6.18.2 Output Devices

The default output device is the output device defined using the `SET OUTPUT` command. This is generally the screen display. The `INTO` rule redirects the output for a particular `REPORT` command.

The output devices are described in the `SET OUTPUT` command section (see page 127).

6.18.3 Number of Lines per Page

The number of lines per page for a specific report can be defined using the `SET LENGTH` command. This allows you to override the default page length.

If the report is output to a screen display, you must hit the RETURN key after each page to display the next page.

6.18.4 Output Width

The `WIDTH` rule overrides the default line width for a specific `REPORT` command. The function of this command is similar to the `SET LENGTH` command. Output lines which are longer than the defined value are right-truncated.

If no `USING` rule has been defined, the default line width is used to center the report title and to right-justify the page number.

6.18.5 Output Format

There are two methods for formatting the output produced by a query:

- Using a form file is described in the next section (the `USING` rule).
- Formatting the page heading of an individual report

You can define a report title by using the `TITLE AS` rule. The title is centered at the top of each page according to the page width. The individual lines of a multiple line heading are separated by a slash (/).

For example:

```
The command
    TITLE AS "Order Status/All Product Groups/Sorted by Customers"

produces the following heading:

                      Order Status
                   All Product Groups
                   Sorted by Customers
```

You can use the `DATE AS` rule to display either the current date (`DATE AS TODAY` using the predefined date format) or a specific date format.

The date format can be either a specific date or time format (see Appendix B) or a combination of user defined text and a date (e.g. “Today is %m/%d/%y”).

The output always begins in column one of the first line of each page.

For example:

```
REPORT
SELECT company, orderno, amount
   FROM orders
   ORDER BY company
 TITLE AS "Order Status/Sorted by Customers"
 INTO ASCII FILE "status.out"
 DATE AS "Date: %x"
```

```
LENGTH = 72
WIDTH = 80;
```

6.18.6 The Use of Form Files

The USING rule is used to specify the text file containing the output format specifications. This text file, called a “form file”, specifies how the results of the REPORT command will be appear in the output list.

The form file consists of sections, separated by lines, beginning with two percent signs (%%). The rest of the line is ignored.

The first section defines the title, date, and page number. This information appears at the top of each page.

The second section contains the formatting instructions for the output lines resulting from the SELECT command.

To allow for calculations, the form file contains a "break" section for each BREAK ON rule. This break section defines the output of the calculated values. These break sections must appear in the same order in the form file as the corresponding BREAK ON rule lines appear in the REPORT command.

You can also define an “end” section in the form file. The text defined in the end section will appear at the bottom of each output page.

Field values are defined in the form file by using either the field name or the field position number of the SELECT list. Field values are preceded by the @ character (e.g. @2 or @custno).

The field contents of the SELECT command can be referenced in the first and second sections. The values in the heading section are always the actual values at the time that the heading is printed. The values in the second section are printed separately for each record resulting from the SELECT command.

The field references of the "break" sections must be consistent with the field references of either the corresponding function or break conditions of the BREAK ON rule. The break section is output whenever the corresponding BREAK ON occurs (i.e. when a new group is output). A corresponding SKIP or PAGE command is executed as defined in the section.

In addition, there are three special functions related to form files. These functions are the $page, $date and $*n* functions.

A `$page` reference specifies that a sequential page number is output for each page. The `$page` reference can be used in either the page header or the page footer. Page numbers can be up to four characters in length and are left justified.

The `$date` reference specifies that a date is output. This reference can appear in any section of the form file. The date format can be defined with the `DATE AS` *date_format* rule, provided that the date format is defined *before* the `USING` rule is specified. If no date format is specified with the `DATE AS` rule, then the date format defined with the `SET DATE` command is used.

A `$`n reference represents the argument specified either in the command line or the `RUN` command. This is important for specifying condition values, ranges and comments.

First, the `REPORT` command:

```
REPORT
SELECT name, street, zip, city, orderno, ordate, amount
   FROM orders
   ORDER BY 1, orderno
CALCULATE
   COUNT(5) BREAK ON (1),
   SUM(7) BREAK ON (1) PAGE
USING "sales.frm"
```

These commands use the form file "`sales.frm`". The contents of this form file are shown here:

```
$date           List of Orders               $page
             Sorted by Customers

Customername...: @1      Street........: @2
                         Zip City......: @3    @4

   Order no      Date        Sales amount in $
   -------------------------------------------
%% End of the page heading section
   @orderno      @6          @amount
%% End of the detail line section
```

```
   Number of Orders: @5
%% End of BREAK section relating to COUNT(5)
   Value of Orders:  @7
%% End of BREAK section relating to SUM(7)
```

6.19 The RUN Command

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

The RUN command is used to execute a QRF (Query Routine File) within **SQL/R**. These QRF files are ASCII text files containing one or more **SQL/R** commands. They are executed as batch files and use the following naming convention:

name.qrf   or   NAME.QRF

The file name must be either all lower case or all upper case.

SQL/R searches for the QRF file in the following search order:

database path
environment variable QPATH
local directory

The use of the RUN command is optional; only the QRF file name is needed to execute the QRF file.

Arguments can be specified as text strings enclosed in quotation marks. When multiple arguments are used, these arguments are separated by commas. In the .qrf file, arguments are referenced by a $\$n$, where n represents the number of the n-th argument in the list. When a QRF file is executed, each $\$n$ is replaced by the corresponding string passed as a parameter.

QRF files can be nested, but secondary QRF files can only reference the arguments used by the primary QRF file.

The following example shows the contents of the QRF file "test.qrf":

```
SELECT number, name, city FROM customers
   WHERE city LIKE "$1";
```

This QRF file can be executed as shown here:

```
RUN test ( "Dallas*" );
```

This produces the following **SQL/R** command:

```
SELECT number, name, city FROM customers
WHERE city LIKE "Dallas*";
```

6.20 The SELECT Command

```
SELECT   [ ALL | DISTINCT ]
         {* | expression ["alternate_heading"] [, . . . ] }
         [ FROM view_name ]
         [ WHERE cond_expression ]
         [ GROUP BY col_ref [, . . . ]  [ HAVING cond_expression] ]
         [ ORDER BY col_ref [ASC | DESC] [, . . . ]] ;
```

The SELECT command is used to select specific data from a table or view. Each output line corresponds to a retrieved table or view entry. Each column contains the result of a data field or expression of a table or view entry.

The selected data fields are listed after the SELECT or SELECT DISTINCT command and are separated by commas. An expression can be the name of a data field, alias, or expression (consisting of data fields used in a view). The actual construction of arithmetic expressions and character strings are described on pages 95 and 98.

A SELECT command can contain a maximum number of 64 data fields. To select all the fields of a view, you can use an asterisk (*) instead of listing all the fields. The fields of an array are accessed using an index. If only the name of the array is specified, by default the first field is used.

The SELECT command output appears in page format according to the page length defined with the SET LENGTH command. The page heading contains the page number and column headings, where the column headings consist of either a specified heading or the *alternate_heading*s. The output width of a column is controlled by the maximum width defined by the field length or column heading.

6.20.1 The DISTINCT Rule

The DISTINCT rule specifies that duplicate output lines are reported only once. Use of this rule results in a longer processing time, because the output is first sorted to locate all duplicate entries. The DISTINCT rule can also be used in calculations. It is important that only one DISTINCT rule is used in each SELECT command.

6.20.2 The FROM Rule

The FROM rule is used to define the data table used for the selection. This table can be either a data record or a virtual table which has been created using the CREATE VIEW

command.

```
SELECT custno, name, city FROM customers
               WHERE city LIKE "Dallas*";
```

Access to the table is sequential, regardless of whether the table is a data record or the first table of a view. Access to all subsequent tables in the view depends on the path definition of the `CREATE VIEW` command. You can specify that the access is performed using a key field. This will optimize the actions of the `WHERE` command which sorts, groups, and evaluates the entries.

6.20.3 The WHERE Rule

The `WHERE` rule is used to set conditions for the data selection. The actual format of the conditional expressions which contain the comparison of data fields and constants, is described on page 100.

```
SELECT custno, name, city FROM customers
WHERE name IN ("Brown","Smith","Jones");
```

6.20.4 The GROUP BY Rule

The `GROUP BY` rule is used to group output lines which contain identical values in the specified columns (col_ref). The other columns of each output line have either a constant value or are the result of a calculation (e.g. `SUM`, `MAX`). These lines are the result produced by the `SELECT` command. The column reference can be either a field name, an alias, or a number which refers to the position of the field in the list of the expressions.

```
SELECT city, SUM(sales), AVG(sales)
FROM customers
GROUP BY city;
```

6.20.5 The HAVING Rule

The HAVING rule is similar to the WHERE rule in that both are used to define which result lines fit certain conditions (filtering). The conditions specified with the HAVING rule are processed after the action of the GROUP BY command. The conditional expressions are described starting on page 100.

```
SELECT city, SUM(sales), AVG(sales)
FROM customers
GROUP BY city
HAVING SUM(sales) > 100000.00;
```

6.20.6 The ORDER BY Rule

The ORDER BY rule is used to sort the results of the SELECT command. The results are sorted by the values contained in the columns defined using the ORDER BY rule. In addition, you can define whether the output is sorted in ascending (ASC) or descending (DESC) order. The default is ascending order. The columns are referenced in the same way as in the GROUP BY rule. The columns are processed in the order in which they appear in the command (from left to right).

```
SELECT * FROM customers ORDER BY name;

SELECT city, SUM(sales), AVG(sales)
FROM customers
GROUP BY city
HAVING SUM(sales) > 100000.00
ORDER BY SUM(sales);

SELECT custno, name, city, sales
FROM customers
ORDER BY 3, sales DESC;
```

6.21 SET Commands

6.21.1 SET DATE

`SET DATE` = "$date_fmt$" `;`

The `SET DATE` command controls the definition of a standard date format. The format string $date_format$ contains either a specific date and time format (see Appendix B) or user defined text and the date (e.g. “Today is %m/%d/%y”).

This standard format is the default date format when no other date format is specified in a report.

The default date format is the american date format:

MM/DD/YY

6.21.2 SET LENGTH

`SET LENGTH` = $lines$ `;`

The `SET LENGTH` command is used to set the number of lines per page. The default is 24 lines (screen output) per page. This value determines where the page breaks appear in the output. The page breaks apply to the results of the `SELECT` and `REPORT` commands as well as the `SHOW` commands (`SHOW FIELD`, `SHOW VIEW`, and `SHOW MACRO`.

If the output is sent to the screen, the `<RETURN>` key is used to view the output page-by-page. If the output is sent to a printer or file, an automatic form feed is executed.

To temporarily override this default page length, use the `LENGTH` rule within the `REPORT` command.

To avoid diaplying a page header and page number, set the page length to zero using the `SET LENGTH` = 0 expression.

6.21.3 SET LOCALE

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

$$category = \left\{ \begin{array}{l} \text{ALL} \\ \text{COLLATE} \\ \text{CTYPE} \\ \text{MONETARY} \\ \text{NUMERIC} \\ \text{TIME} \end{array} \right\}$$

The `SET LOCALE` command is used to set a local condition (e.g. language)

The defaults shown here define the **SQL/R** environment.

Scope	Environment	Action Target
ALL	LANG	all subsequent
COLLATE	LC_COLLATE	not currently used
CTYPE	LC_CTYPE	characters typzierung
MONETARY	LC_MONETARY	MONEY output
NUMERIC	LC_NUMERIC	mumric outpur
TIME	LC_TIME	date field output.

6.21.4 SET OUTPUT

$$\texttt{SET OUTPUT} = \left\{ \begin{array}{l} \texttt{TERMINAL} \\ \texttt{PRINTER} \\ [\,\texttt{ASCII}\,|\,\texttt{DIF}\,]\,\texttt{FILE}\ \text{"filename"} \end{array} \right\} \texttt{;}$$

The `SET OUTPUT` command is used to define the output device (and firmat). Possible devices are screen, printer, or (disk)file. The default device is `TERMINAL` (stdout). The printer used depends on the `SET PRINTER` rule. Output sent to a disk file is stored as it would appear on the screen, namely in the page headings and page numbers. In addition, supported output formats included ASCII and DIF. This enables you to export the data to another application.

6.21.5 SET PRINTER

`SET PRINTER` $= device$ `;`

The `SET PRINTER` command is used to define the default printer. If no printer is specified, the default printer is `lp`. The printer defined with the `SET PRINTER` command is the printer that is used whenever the `SET OUTPUT = PRINTER` or `REPORT ... INTO PRINTER` expression is used. To send output directly to the printer (without using the spooler), use the `SET OUTPUT = FILE` $file_name$ to define the device file.

```
SET PRINTER = "lp -dlj -onb -ol72";
SET PRINTER = FILE "output";
SET PRINTER = ASCII FILE "output";
```

6.21.6 SET WIDTH

`SET WIDTH` = *columns* `;`

The `SET WIDTH` command is used to define the number of columns for an output page. The default is 80 columns. Output lines which are longer than 80 columns are right-truncated. The title centering and page number position for a page are based on this value.

6.22 SHOW Commands

6.22.1 SHOW DATE

`SHOW DATE;`

The `SHOW DATE` command displays the current date format. This format can be changed using the `SET DATE` command.

6.22.2 SHOW FIELD

`SHOW FIELD` { * | *field_name* } `;`

The `SHOW FIELD` command displays information about data fields and aliases for the database currently open. If an asterisk (*) is used with the `SHOW FIELD` command, all database items and aliases and their corresponding table names are displayed. Also displayed is the description that was defined with the `DESCRIBE AS` rule of the `FIELD` statement. The `SHOW FIELD` *field_name* displays all the relevant information about a field, including the following:

- the alias and its corresponding field or expression
- the field description defined using the `DESCRIBE AS` rule of the `FIELD` statement
- the database definition
- the output format
- an indication of the activity of the coded-value-translation
- lists of tables and views from which *field_name* can be selected

6.22.3 SHOW LENGTH

`SHOW LENGTH;`

The `SHOW LENGTH` command displays the number of lines configured for a page. This page length is set using either the `SET LENGTH` command or the `LENGTH` rule within the `REPORT` command. A page length defined with the `LENGTH` rule will temporarily (within the `REPORT` command) override the page length defined with the `SET LENGTH` command.

The page length value is used by **SQL/R** to control the page breaks.

6.22.4 SHOW LOCALE

`SHOW LOCALE;`

The `SHOW LOCALE` command displays either the values set with the `SET LOCALE` command or the default values. The default values depend on the user environment.

6.22.5 SHOW MACRO

`SHOW MACRO` { * | "*macro_name*" }`;`

The `SHOW MACRO` command followed by a "*macro_name*" displays the definition and description of a macro. The "*macro_name*" is a character string enclosed in quotation marks. If the `SHOW MACRO` command is followed by an asterisk (`*`), then all the macros are listed.

6.22.6 SHOW OUTPUT

`SHOW OUTPUT;`

The `SHOW OUTPUT` command displays the name of the output device which was defined using the `SET OUTPUT` command. The default device is the screen. The output device can be redefined using the `SET OUTPUT` command. In addition, you can use the `INTO` rule of the `REPORT` command to define a different output device for a specific report.

6.22.7 SHOW PRINTER

`SHOW PRINTER;`

The `SHOW PRINTER` command displays the same of the default printer. Output is sent to this printer whenever the `SET OUTPUT = PRINTER` or a `REPORT` ... `INTO PRINTER` statement is used. You can use the `SET PRINTER` command to redefine the default printer.

6.22.8 SHOW VIEW

SHOW VIEW { * | *view_name* };

The SHOW VIEW command displays information about all record types and views which are defined for the currently in use database, or which were produced using the CREATE VIEW command. If the SHOW VIEW command is followed by an asterisk (*), then all record types and views are displayed with the description defined using the CREATE VIEW command.

When a record type or view is specified, then detailed information is provided about the following:

- view type
- description
- name and type of all fields in the record

6.22.9 SHOW WIDTH

```
SHOW WIDTH;
```

The SHOW WIDTH command displays the number of columns in a page. The default is 80 columns. This value can be changed using the SET WIDTH command. Output lines which are wider than the defined width are right-truncated. The report title and page number position are centered using the value of the page width. The page width for a specific report can be changed using the WIDTH rule of the REPORT command.

Quick Reference Guide

```
CLOSE DATABASE ;

CREATE VIEW view_name PATH occur_spec path_group
 [ DESCRIBE AS "description" ] ;

occur_spec     = [ OCCURRENCE occur_name OF ] record_name
                 [ occur_name =             ]

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

path_element   = { (path_element path_group)                                      }
                 { occur_spec WHERE field_name = [occur_name.] field_name          }

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 ]
          [ DESCRIBE AS "description" ] ;

         { (length)
         { INT( length )
         { LONG( length )
         { FLOAT( length, decimal places )
format = { DOUBLE( length, decimal places )
         { FIXED( length, decimal places )
         { MONEY( length [, decimal places ] )
         { DATE   [ ( "date_format" [, length ] ) ]
         {        [ FROM { SYSDATE | YYYY } ]
         { TIME [ ( length ) ]
```

```
HELP [ { identifier | "string"} ] ;

OPEN DATABASE "database name" [ AS "password" ] [, . . . ] ;

REPORT SELECT  [ CALCULATE field_calc [, . . . ] ]
               [ INTO { TERMINAL
                        PRINTER
                        [ ASCII | DIF ] FILE "filename" } ]
               [ report_fmt ]
               [ USING "report_form" ] ;

field_calc =   [ { SUM
                   AVG
                   MIN
                   MAX
                   COUNT } ( field_ref [, . . . ] ) ["row label"] ]
               BREAK ON { ( field_ref [, . . . ] )
                          REPORT } [ SKIP [n]
                                     PAGE [n] ]

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 col_ref [, . . . ] [ HAVING cond_expression] ]
        [ ORDER BY col_ref [ ASC | DESC ] [, . . . ]] ;

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

```
             { ALL       }
             { COLLATE   }
category =   { CTYPE     }
             { MONETARY  }
             { NUMERIC   }
             { TIME      }
```

```
SET DATE = "date_fmt" ;
```

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

```
     { LENGTH    = lines     }
SET  { PRINTER   = "device"  }  ;
     { WIDTH     = columns   }
```

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

B

Date and Time Formats

A date format is a formatting command consisting of text and format codes. A format code is preceded by a % character:

Code	Length	Description
%a	2	day-of-week (short alphabetic notation)
%A	10	day-of-week (alphabetic)
%b	5	month (short alphabetic notation)
%B	10	month (alphabetic)
%c	*	date and time
%d	2	day-of-month (01–31)
%H	2	hour (24 hour clock) (00–23)
%I	2	hour (12 hour clock) (01–12)
%j	3	day-of-year (001-366)
%m	2	month (numeric notation) (01–12)
%M	2	minutes (00-59)
%p	2	AM or PM (if necessary)
%S	2	seconds (00–59)
%U	2	week-of-year (00-53)
		(the first sunday of a year is the first day of week 1)
%w	1	day-of-week (numeric) (0(sunday)–6)
%W	2	week-of-year (00-53)
		(the first monday of a year is the first day of week 1)
%x	*	date
%X	*	time
%y	2	year (last two digits only) (00-99)
%Y	4	year (4 digits)
%Z	4	time zone (if necessary)
%%	1	%-character

In the previous table, the specifications for the column length are the default length used by **SQL/R** if no other values are specified. These values are not required to correspond with the actual lengths. If the actual length is longer than the speciified length, the output is right-truncated.

The codes having a length marked with an asterisk (*) in the table have lengths which are dependent on the work environment.

In addition, it is also possible to include length and adjustment specifications between the "%" character and the format code. These specifications are shown here:

[– | 0] n
: The n represents a number specifying the minimum field length of the formatted output. This output is then left or right justified. By default, the output is right justified with leading spaces. If the option – is used, the resulting output is left justified with trailing spaces. If the zero 0 option is used, the resulting output is right justified with leading zeros.

$.p$
: For numeric output, (%d, %H, %I, %j, %m, %M, %S, %U, %w, %W, %y, %Y), the $.p$ represents the **minimum** number of characters. If the result has fewer digits than the minimum, leading zeros are added.
If the output produces a character string, (%a, %A, %b, %B, %c, %p, %x, %X, %Z, %%) then $.p$ represents the **maximum** number of characters. If the result has more characters than the maximum specified, the result is right-truncated.

Examples:

Format	Result	Comment
%A	September	no length specified
%.3A	Sep	maximum length = 3 characters
%d.%m.%y	08.05.92	no length specified
%.1m/%.1d/%y	5/18/92	minimum length = 1 (month, day)
%3d.%-3m.%05y	8.5 .00092	day and month use a minimum length of 3, day is right justified, month is left justified, minimum length for year is 5 characters with leading zeros.

If the work environment has been defined with a `LOCALE` "TIME=german" command, the following formats are pre-defined:

Code	Format	Example
%c	%a., %d. %b %Y, %H:%M:%S	Fr., 08. Mai 1992, 10:28:05
%x	%a., %d. %b %Y	Fr., 08. Mai 1992
%X	%H:%M:%S	10:28:05

If the work environment has been defined with a `LOCALE` "TIME=american" command, the following formats are pre-defined:

Code	Format	Example
%c	%a, %b %1d, %Y, %I:%M:%S %p	Mon, May 8, 1992, 10:28:05 AM
%x	%a, %b %1d, %Y	Mon, May 8, 1992
%X	%I:%M:%S %p	10:28:05 PM

Date format consists of a maximum of 70 characters. This 70 character maximum applies to both the format codes and the resulting text.

The time formats are only significant in conjunction with date variables defined using the system format for defining dates (where the date format is calculated by counting the number of seconds since Jan 1, 1970). This also applies to fields defined using the `FIELD ... DISPLAY AS DATE ... FROM SYSDATE` statement or `REPORT ... DATE AS` statement.

C

Differences between SQL/R and standard SQL

The **SQL/R** language is based on standard SQL. However, there are differences which are the result from the distinct goals of the two languages. These differences are described here:

- **SQL/R** only reads data from a database. Database changes or deletions are not possible.
- **SQL/R** supports the use of Arrays. Standard SQL does not support the use of Arrays.
- The standard SQL functions `CHAR`, `LENGTH`, `DATE`, `DAYS`, `TIME`, `HOUR`, `MINUTE`, and `SECOND` are not supported with **SQL/R**.
- **SQL/R** contains the additional functions `UPPER`, `LOWER`, `TRIM` and `STRLEN`.
- The `CREATE VIEW` command is handled differently by the **SQL/R** language and standard SQL. Both the syntax and action of the command are different.
- The `SELECT` command of **SQL/R** does not include the full functionality of the standard SQL SELECT command. There is no `UNION` option and no subselect. In addition, it is not possible to access several tables within one `SELECT` command. To do this using **SQL/R**, you use the `CREATE VIEW` command to create a view before using the `SELECT` command. The `ORDER BY` rule can only be used for columns which are listed (referenced) within the `SELECT` command. Sorting of fields which are not produced is also not possible.
- **SQL/R** contains a number of functions which are not included in standard SQL. These functions are designed especially for formatting lists. These functions are provided using the `REPORT`, `FIELD`, `RUN`, `SET`, and `SHOW` commands.

D

Work Environment

Using environment variables you can define the work environment; specifically to adjust programs to your needs. The following section describes the environmental variables used by **SQL/R**.

Environmental variables are HP-UX Shell variables which can be accessed by other programs. The commands described in the following section are used to set the environmental variables.

For example:

```
LANG=american
export LANG
```

These commands set the HP-UX shell variable `LANG` to the value `american` and gives other programs access to this variable.

SQL/R uses the following environmental variables:

Variable	Short Description
TERM	terminal type
LINES	number of lines (if different)
COLUMNS	number of columns (if different)
LANG	language and language environment
LC_COLLATE	collating sequence (if different)
LC_CTYPE	character type (if different)
LC_MONETARY	output format for `MONEY` (if different)
LC_NUMERIC	numeric output format (if different)
LC_TIME	date/time output format (if different)
QPATH	list of directories containing files for **SQL/R**
TZ	time zone
LPDEST	output device for lp (alternate to standard printer)
TMPDIR	directory for temporary files

These environmental variables are described in detail in the following sections. For additional information, use the following HP-UX shell command:

```
man 5 environ
```

Description of the environmental variables:

QPATH
: QPATH contains a list of directories which **SQL/R** searches for the qif, qrf and form files, if the pathname was not specified in the program. For example if a filename was specified without a leading slash (/).

  The directories in the list are separated by a colon (:).

  For example: `/sqlr:/usr/sqlr/sample`

  searches in the directories `/sqlr` and `/usr/sqlr/sample`.

LANG
: The LANG variable sets the defaults for language and character set (for example, the use of characters unique to a specific language). The values for LANG are specified in english (see `lang(5)`).

  If no LANG value is specified, a default of english (with no special characters) is used.

  The Editor program, screen messages and function key labels are determined by the LANG variable.

LC_. . .

LC_COLLATE, LC_CTYPE, LC_MONETARY, LC_NUMERIC and LC_TIME. These LC_. . . variables allow you to specify the country-related defaults which deviate from the values predetermined by the LANG variable.

If these variables are not set, then an appropriate default value is provided by the LANG variable. You can also set default values for these variables using the `SET LOCALE` command within **SQL/R**.

LC_COLLATE, LC_CTYPE, LC_MONETARY, LC_NUMERIC and LC_TIME can be set using the following format:

```
language [@modifier]
```

The @modifier field allows you to set a different value for a specific variable while keeping the remaining default values for that language. An example would be setting a different collating sequence. You can use the man pages of `nlsinfo(1)` to obtain a list of the possible values.

For example, to configure german screen messages, but use the dutch names for the months you set the following variables to the values shown here:

```
LANG=german
LC_TIME=dutch
```

Variable	Changes/Defines
LC_COLLATE	Collating sequence. This variable is used to set the collating sequence. Note: It is currently not used.
LC_CTYPE	Character type. This variable is used to define which characters are treated as alphabetic characters and how lower and upper case characters can be changed.
LC_MONETARY	Monetary output format. This variable is used to define how monetary amounts are displayed. For example, how many decimal places are displayed and how money is grouped.
LC_NUMERIC	Numeric output format. This variable is used to define the numeric output format. For example, whether a period or a comma preceeds the decimal places.
LC_TIME	Date field output format. This variable is used to define the output format for date information such as day and month names.

TERM — The TERM variable defines the terminal type. This is required because **SQL/R** supports specific terminal types.

COLUMNS — The COLUMNS variable defines the number of columns for the terminal display. If no value is specified, a default value of 80 characters per line is used.

LINES — The LINES variable defines the number of lines for the terminal display. If no value is specified, a default value of 24 lines is used.

TZ — The TZ variable defines the time zone.

LPDEST — The LPDEST variable is used to define the name of the default printer used by the `lp` command. This printer is used if no alternate printer was defined using option `-d`.

If no value is specified, the standard printer for that system is used.

TMPDIR — The TMPDIR variable defines the directory used for temporary files. If no value is specified, the directory `/tmp` is used.

E

HP Eloquence Format Numbers

The HP Eloquence format numbers are defined for a database by either the `schema` or the `dbmods` utility. These numbers are then used by HP Eloquence QUERY to evaluate and format data. When **SQL/R** opens a database it translates these numbers to the corresponding format.

The HP Eloquence format numbers are *cumulative* codes. For each group or attribute, a code value is added.

Group	Value	Comments
Query Write inhibit		
No write inhibit (default)	0	(ignored)
Write inhibit	1	(ignored)
Item type		
Date type	2	`DATE` (FROM 1972)
Currency	4	`MONEY`
Undefined	6	(ignored)
Spacing		
Default	0	(ignored)
Comma every 3 digits	8	(ignored)
Post decimals		
Default	0	(ignored)
FIXED 0	16	(1)
FIXED 1	32	(1.0)
FIXED 2	48	(1.00)
FIXED 4	80	(1.000)
FIXED 3	64	(1.0000)
FIXED 5	96	(1.00000)
FIXED 6	112	(1.000000)

For the item types `MONEY` and `DATE`, all further entries are ignored.

The number of decimal places (post decimals) are recognized for floating point decimal data types (float, double) only.

Examples:

Date = 2
Money = 4
Value with 2 decimal places = 48

F

Glossary

This appendix provides definitions and explanations for many of the terms and expressions used in this manual.

ARGUMENT

An independent variable

ARITHMETICAL EXPRESSION

Contains arithmetical operations and operators which result in a single numeric value

ARITHMETICAL OPERATOR

A symbol used to represent a mathematical operation. For example:

+ = Addition
− = Subtraction
* = Multiplication
/ = Division

ARITHMETICAL OVERFLOW

Represents a condition that occurs when the result of a calculation exceeds the defined boundaries of the value range.

ASCII

Acronym for "American Standard Code for Information Interchange". This is a common standard for information exchange.

BYTE

Represents a standardized unit of data. A byte consists of 8 bits. A byte is required to store one ASCII character.

CHARACTER SET

Defines all the possible characters which can be used in a data field. The possible characters are defined by the data type of the field.

CHARACTER STRING

A sequence of characters. Character strings are enclosed in quotation marks.

COLUMN

A data item (field) of a data structure within a database.

COLUMN NAME

The unique name assigned to a column or field within a database table.

COMMAND

Generally, an instruction to the operating system. The term “STATEMENT” is another term for an **SQL/R** instruction.

COMPARISON / RELATIONAL OPERATORS

Symbols such as $=$, $>$ and $<$ that indicate the relationship between two values.

CONSTANT

A fixed, constant value. The opposite of a variable.

DATABASE

A collection of related data which is stored together. A database is used to store the data of one or more applications in an optimal form without disadvantageous or unnecessary redundancy. The data is stored independently of the application programs which use the data. The programs have a common, controlled access to the database by using a database language such as SQL. Depending on the database language used, you can add, modify, or delete database entries.

DATABASE DEFINITION

A description of the storage format, tables and columns of an individual database.

DATA TYPES

All the available types used to produce a column. **SQL/R** supports the following data types:

CHAR, SHORT, INT, LONG, FLOAT, DOUBLE, DATE, FIXED, MONEY and TIME

DEFAULT

The attribute, value, option, or setting used if no other value is specified.

DEFINE

Represents an **SQL/R** command. The `DEFINE` rule can be used with other **SQL/R** commands to define short notations and place holders (macros).

EXIT

The **SQL/R** command which is used to end an **SQL/R** process. All commands after the `EXIT` command in a file are ignored.

EXPRESSION

This is either an operand or a combination of operands and operators which results in a single value.

FIELD

Another representation of columns in a database table, also referred as ITEMs or DATA FIELDs.

FIELD COMMAND

An **SQL/R** command that has several uses. For example, the `FIELD` command can be used to define an alternate name for fields and expressions or to specify the output format of data fields. It can also be used to set values in reference to coded data fields.

GROUP BY

A rule within the `SELECT` command which is used to create groups.

HAVING

A rule within the `SELECT` command which is used to filter out selected individual results of the `GROUP BY` rule. This rule can only be used in combination with the `GROUP BY` rule.

HELP

Displays information about the meaning of an identifier, such as a FIELD, RECORDS or MACRO. You can get additional information about each of these identifiers by using the appropriate `SHOW` command along with the identifier.

INDEX

A collection of data about the position of records within a table. These index keys enable faster access to the data.

LINE

A horizontal entry in a database table. The terms RECORD or DATA LINE are also used.

MATHEMATICAL FUNCTION

Functions used on the columns of a data record. For example: AVG, COUNT, MIN, MAX, SUM.

OBJECT

An object is a table, view, or index.

ORDER BY

A rule used wthin the `SELECT` command to specify the sort order of the `SELECT` command results.

PARAMETER

Information or data given to a command or function which affects the results of the command or function. Parameters can be specified by either a user or a program.

RECORD

A database entry. A record is a row in a database table. A record consists of fields.

REPORT

The `REPORT` command and the `SELECT` command are the most important **SQL/R** commands. The `REPORT` command displays the results produced by the `SELECT` command.

RESULT TABLE

A quantity of result lines which are produced by a `SELECT` command.

RULE

A syntactically separate part of an **SQL/R** command. This part is identified during the syntax analysis of the entire command.

SELECT

The `SELECT` command is the most important **SQL/R** command because it is used to define the data to be retrieved from the database. Rules are part of the `SELECT` command and used to further define the data to be returned by the `SELECT` command.

SET

Used to set defaults such as page length and width.

SHOW

Used to display detailed information about objects such as fields, records, and views.

SQL

Abbreviation for "Structured Query Language". This is a general term for a database query language such as INGRES or INFORMIX. Structured query languages are used to create and use relational databases.

STATEMENT

An instruction used in a high level language such as **SQL/R**. Examples are the `SELECT` and `REPORT` commands.

STRING

A sequence of characters (character string).

TABLE

A relative (relational) object in which data is stored. A table contains horizontal lines (also called RECORDs or data lines) and vertical columns (also called FIELDs).

VALUE

Is a measurable item assigned to a constant, a variable, or a parameter.

VARIABLE

A data unit, such as a number, which is defined in a high level language and used to assign a value. Examples are: single characters or a data line structure.

VIEW

An **SQL/R** command used to define a logical table which presents a specific view of existing physical tables of a database.

WHERE

A rule within the `SELECT` command. This rule is used to establish conditions for the desired results of a `SELECT` command.

Index

*, 101, 123
.qif, 112
.qrf, 121
;, 10
$, 66
$date, 63, 118, 119
$n, 66, 67
$page, 63, 118, 119
&, 55

A
Alias, 89
ALL, 95, 97, 123, 134
AND, 16–18, 21, 79, 80, 100, 101, 103, 133
Array, 88
ASC, 22, 123, 125, 134
ASCII, 114, 127, 134, 135
AVG, 27, 95, 96, 114, 115, 134

B
BETWEEN, 16, 21, 100, 101
BREAK ON, 60, 73, 74, 114, 115, 118, 134
BREAK ON REPORT, 60, 73, 115

C
CALCULATE, 58, 60, 63, 71–73, 84, 97, 114, 115, 134
CENTER, 108, 110, 133
CLOSE DATABASE, 102, 112, 133
COUNT, 27, 28, 60, 74, 95, 97, 114, 115, 134
CREATE VIEW, 75–81, 84, 88, 89, 103, 123, 124, 131, 133, 139

D
Data Types, 91
Dataset, ⟶ Table
DATE, 91, 108, 110, 133, 135, 138, 144
DATE AS, 58, 63, 114, 117, 119, 134, 138
DATE AS TODAY, 117
DAY, 95, 97
DEFINE, 106, 133, 148
DESC, 22, 72, 123, 125, 134
DESCRIBE AS, 103, 106, 108, 129, 133
DIF, 114, 127, 134, 135
DISPLAY AS, 65, 71, 84, 108–111, 133, 138
DISPLAY AS DATE, 94, 111
DISPLAY AS FIXED, 111
DISPLAY AS TIME, 111
DISTINCT, 24, 28, 95–97, 123, 134
DOUBLE, 108, 133

E
Entry, ⟶ Record
EXIT, 71, 74, 107, 133, 148

F
Field, 88
FIELD, 55, 65, 71, 72, 76, 82, 84, 89, 91, 94, 108–110, 112, 129, 133, 135, 138, 139, 148
Field Reference, 88
FILE, 114, 127, 134, 135
FIXED, 91, 108, 110, 111, 133
FLOAT, 108, 133
FROM, 12, 14, 53, 108, 111, 122, 123, 133, 134
FROM SYSDATE, 111, 138

G
GROUP, 56
GROUP BY, 12, 32, 33, 71, 74, 84, 97, 123–125, 134, 148

H
HAVING, 32–34, 100, 123, 125, 134
HELP, 107, 134

I

IF, 71, 72, 95, 98–100
IN, 16, 20, 100, 101
INT, 108, 109, 111, 133
INTO, 114, 116, 130, 134
INTO PRINTER, 61, 127, 130
Item, ⟶ Field

L
LEFT, 108, 110, 133
LENGTH, 60, 114, 126, 129, 134, 135
LIKE, 16, 29, 31, 100, 101, 122
LOCALE, 93, 135, 138
LONG, 108, 109, 111, 133
LOWER, 98, 99, 139
lp, 127, 143

M
MACRO, 135
MAX, 27, 95, 96, 114, 115, 124, 134
MIN, 27, 95, 96, 114, 115, 134
MONEY, 91, 92, 108, 110, 111, 133, 141, 144
MONTH, 95, 97

N
NOT, 16, 100

O
Occurrence, 88
OCCURRENCE, 103, 133
OPEN DATABASE, 13, 52, 86, 87, 102, 112, 134
OR, 16, 18, 20, 100, 101
ORDER BY, 12, 22, 53, 56, 84, 123, 125, 134, 139
OUTPUT, 135

P
PAGE, 73, 114, 115, 118, 134
PAGE[n], 73
Path, 89
PATH, 76, 103, 133
PRINTER, 114, 127, 134, 135

R
Record, 88
REPORT, 57, 58, 60, 62, 63, 71, 81, 84, 97, 114–116, 118, 119, 126, 127, 129, 130, 132, 134, 138, 139, 149, 150
REPORT SELECT, 114, 134
Reserved Words, 90
RIGHT, 108, 110, 133
RUN, 87, 119, 121, 134, 139

S
SELECT, 12, 15, 16, 23–25, 32, 53, 55, 56, 62, 63, 74, 75, 81, 84, 97, 99, 100, 108, 114, 115, 118, 122–126, 134, 139, 148–150
SELECT DISTINCT, 123
SET, 135, 139
SET DATE, 58, 63, 111, 119, 126, 129, 135
SET LENGTH, 116, 123, 126, 129
SET LOCALE, 126, 127, 130, 134, 142
SET OUTPUT, 116, 127, 130, 135
SET OUTPUT = FILE, 127
SET OUTPUT = PRINTER, 127, 130
SET PRINTER, 61, 127, 130
SET WIDTH, 128, 132
SHORT, 109, 111
SHOW, 107, 126, 135, 139, 148
SHOW DATE, 129
SHOW FIELD, 108, 126, 129
SHOW LENGTH, 129
SHOW LOCALE, 130
SHOW MACRO, 126, 130
SHOW OUTPUT, 130
SHOW PRINTER, 130
SHOW VIEW, 126, 131
SHOW WIDTH, 132
SKIP, 73, 114, 115, 118, 134
sqlr, 86
sqlred, 86
sqlrexec, 66, 67, 71, 86, 87
STRLEN, 95, 97, 139
SUBSTR, 98, 99
SUM, 27, 32, 56, 60, 73, 74, 95, 96, 114, 115, 124, 134

SYSDATE, 108, 133

T
Table, 88
TERMINAL, 114, 127, 134, 135
TIME, 91, 108, 110, 133
TITLE AS, 58, 62, 114, 117, 134
TO, 80, 103, 133
TODAY, 58, 114, 134
TRIM, 98, 99, 139

U
UPPER, 98, 99, 139
USING, 62, 67, 84, 114, 116–119, 134

V
VALUES ARE, 82, 108, 109, 133
VIEW, 135

W
WHERE, 12, 16, 26, 32, 33, 53, 66, 84, 100, 103, 122–125, 133, 134
WIDTH, 114, 116, 132, 134, 135

X
XOR, 100, 101

Y
YEAR, 95, 97