INFORMIX-4GL

Quick Syntax

Version 6.0
April 1994
Part No. 000-7609

Published by INFORMIX® Press

Informix Software, Inc.
4100 Bohannon Drive
Menlo Park, CA 94025

ACKNOWLEDGMENTS

The following people contributed to this version of *INFORMIX-4GL Quick Syntax*:

Documentation Team: Adam Barnett, Kaye Bonney, Lisa Braz, Tom Houston, Liz Knittel, Jerry Pope

Technical Contributors: Jonathan Leffler, Sally Cox

Table of Contents

INFORMIX-4GL Quick Syntax

Introduction

This Guide presents a quick reference to the material listed in the following table. For a full discussion of each topic, refer to the corresponding documentation.

Topic	Reference Documentation	Section
Data types	*INFORMIX-4GL Reference*	Chapter3 (DEFINE)
Built-in functions and operators	*INFORMIX-4GL Reference*	Chapter 4
4GL statements	*INFORMIX-4GL Reference*	Chapter 3
SQL statements		
4.1 servers:	*Informix Guide to SQL: Reference*	Chapter 6
6.0 servers:	*Informix Guide to SQL: Syntax*, Version 6.0	Chapter 1
Stored Procedure Language (SPL)		
6.0 servers:	*Informix Guide to SQL: Syntax*, Version 6.0	Chapter 2
4GL forms	*INFORMIX-4GL Reference*	Chapter 5
Reports	*INFORMIX-4GL Reference*	Chapter 6
SQLCA record structure	*INFORMIX-4GL Reference*	Chapter 2
NewEra Debugger commands	*Guide to the INFORMIX-4GL Interactive Debugger*	Chapter 9
Environment variables	*INFORMIX-4GL Reference*	Appendix D

This Guide shows syntax that must be prepared before you can include it in a **4GL** program. You must prepare any SQL statement introduced later than the 4.1 server release. These statements are indicated in this manual by the following icon:

6.0

To use these statements, such as CREATE TRIGGER, you must:

1. Store the SQL statement as a character string.
2. Set up the statement for execution by means of the PREPARE statement (see page 79).
3. Process the statement by means of the EXECUTE statement (see page 75).

Syntax Conventions

Syntax diagrams describe the format of SQL, SPL, and 4GL statements and **Debugger** commands, including alternative forms of them, required and optional parts of them, and so forth. Syntax diagrams have their own conventions, which are defined in detail and illustrated in this section.

Each syntax diagram displays the sequences of required and optional elements that are valid in a statement or command. Briefly:

- All keywords are shown in uppercase letters for ease of identification, though you need not enter them that way.
- Words for which you must supply values are in italics.
- All boldface characters are literals.

Each diagram begins at the upper left with a keyword and ends at the upper right with a vertical line. Between these points, you can trace any path that does not stop or back up. Each path describes a valid form of the statement. Except for separators in loops (see page 8), which the path approaches counterclockwise from the right, the path always approaches elements from the left and continues to the right.

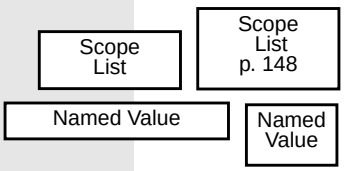
Along a path, you may encounter the following elements:

KEYWORD — You must spell a word in uppercase letters exactly as shown; however, you can use either uppercase or lowercase letters when you enter it.

(.,;+*-/) — All other characters are literal symbols that you must enter exactly as shown.

' ' " " — Single and double quotes are literal symbols that you must enter as shown.

variable — A word in italics represents a value that you must supply. The nature of the value is explained fully in the appropriate reference manual.



A reference in a *box* represents a subdiagram on the same page (if no page number is supplied) or on a specified page. Imagine that the subdiagram is spliced into the diagram at this point. (A synonym for "subdiagram" is "segment.")

If the term "(subset)" appears in the box below the name of the segment being referenced, you should refer to the appropriate reference manual for further clarification.

The aspect ratio of the box is not significant. That is, the same segment can be represented by boxes of different shapes, as in these symbols for the Named Value segment.

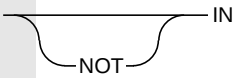
OL A code in an icon is a signal warning you that this path is valid only for certain database servers or under certain conditions. The codes indicate the products or conditions that support the path. The following codes are used:

- **6.0** Path requires the statement to be prepared (by using PREPARE).
- **SE** Path is valid only for INFORMIX-SE.
- **OL** Path is valid only for INFORMIX-OnLine Dynamic Server.
- **+** Path is an Informix extension to ANSI standard SQL.
- **NLS** Path is valid only if you are using NLS.

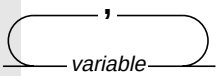
A shaded option is the default. Even if you do not explicitly type the option, it will be in effect unless you choose another option.

Syntax enclosed in a pair of arrows indicates that this is a subdiagram.

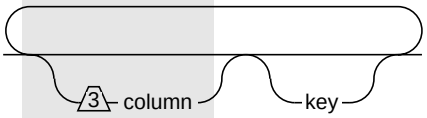
The vertical line is a terminator and indicates that the statement is complete.



A branch below the main line indicates an optional path.

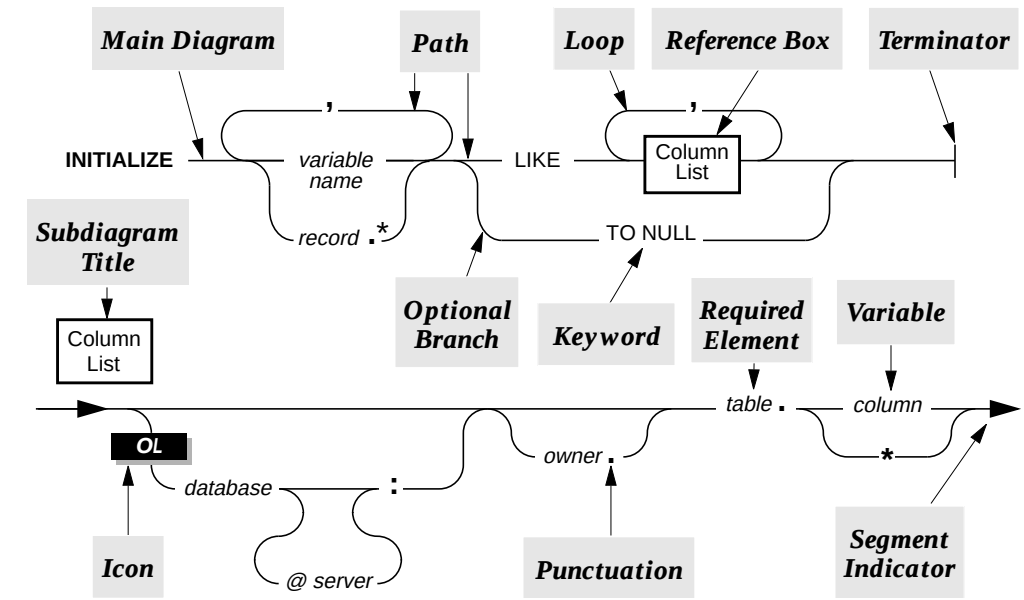


A loop indicates a path that can be repeated. Punctuation included in the loop indicates the *separator symbol* for items in a list; when no symbol is shown in the loop, as in the example below, a blank space or LINEFEED is the separator.



A gate (3) in an option indicates that you can only use that option the number of times indicated, even though it is within a larger loop.

The grey labels and arrows in the following illustration identify the elements of a syntax diagram for the INITIALIZE statement of 4GL.



Elements of a syntax diagram

Basics

Data Types

Data Type	Kind of Values Stored
ARRAY OF *type*	Arrays of values of any other single data type.
[1]BYTE	Any kind of binary data.
CHAR	Character strings of up to 32,767 ASCII characters.
CHARACTER	(This keyword is a synonym for CHAR.)
DATE	Points in time, specified as calendar dates.
DATETIME	Points in time, specified as calendar dates and time-of-day.
DEC	(This keyword is a synonym for DECIMAL.)
DECIMAL	Fixed point numbers, of a specified scale and precision.
DOUBLE PRECISION	(These keywords are a synonym for FLOAT.)
FLOAT	Floating-point numbers, of up to 32-digit precision.
INT	(This keyword is a synonym for INTEGER.)
INTEGER	Whole numbers, from -2,147,483,647 to +2,147,483,647.
INTERVAL	Spans of time in years and months, or else in smaller time units.
MONEY	Currency amounts, with definable scale and precision.
NUMERIC	(This keyword is a synonym for DECIMAL.)
REAL	(This keyword is a synonym for SMALLFLOAT.)
RECORD	Ordered sets of values, of any combination of 4GL data types.
[2]SERIAL	Same as INTEGER. Automatically assigned by the engine.
SMALLFLOAT	Floating-point numbers, of up to 16-digit precision.
SMALLINT	Whole numbers, from -32,767 to +32,767.
[1]TEXT	Character strings of any length.
[1]VARCHAR	Character strings of varying length, no greater than 255.

[1]*INFORMIX-OnLine Dynamic Server or other 4GL statements only.* [2]*SQL only.*

4GL Arithmetic Operators

Operator Symbol	Operator Name	Name of Result	Precedence
**	exponentiation	power	12
mod	modulus	integer remainder	12
*	multiplication	product	11
/	division	quotient	11
+	addition	sum	10
-	subtraction	difference	10

Boolean Operators

AND
OR
NOT

4GL Relational Operators

Operator Symbol	Operator Name	Operator Symbol	Operator Name
<	Less than	!= *or* <>	Not equal to
<=	Not greater than	>=	Not less than
= *or* ==	Equal to	>	Greater than

Global Constants and Variables

FALSE
INT_FLAG
NOTFOUND
NULL
SQLCODE
STATUS
TRUE
QUIT_FLAG

SQLCA Record:
- SQLCODE
- SQLERRM
- SQLERRP
- SQLERRD
- SQLAWARN

Built-In Functions

ARG_VAL(*int-expr*)
ARR_COUNT( )
ARR_CURR( )
†AVG(*int-field*)
†COUNT(*)
DOWNSHIFT(*char-expr*)
ERR_GET(*int-expr*)
ERR_PRINT(*int-expr*)
ERR_QUIT(*int-expr*)
ERRORLOG(*char-expr*)
FGL_DRAWBOX(*height, width,line,left-offset[,color]*)
FGL_GETENV(*char-expr*)
FGL_KEYVAL(*char-expr*)
FGL_LASTKEY( )
LENGTH(*char-expr*)
†MAX(*int-field*)
†MIN(*int-field*)
NUM_ARGS( )
†PERCENT(*)
SCR_LINE( )
SET_COUNT(*int-expr*)
SHOWHELP(*int-expr*)
SQLEXIT( )
STARTLOG(*char-expr*)
†SUM(*int-field*)
UPSHIFT(*char-expr*)

†*Valid only in REPORT blocks or in some SQL statements. Also, may be preceded by GROUP.*

Color numbers and their meanings that can be used in FGL_DRAWBOX( ) are:

Number	Color	Number	Color	Number	Color
0	White	3	Red	6	Blue
1	Yellow	4	Cyan	7	Black
2	Magenta	5	Green		

Operators

ASCII *int-expr*
char-expr CLIPPED
COLUMN *integer*
CURRENT
CURRENT *qualifier*
DATE
DATE (*date-expression*)
DAY(*date-expression*)
EXTEND(*time-value*)
EXTEND(*time-value, qualifier*)
FIELD_TOUCHED(*field-list*)
GET_FLDBUF(*field-list*)
expression IS NOT NULL
expression IS NULL
INFIELD(*field*)
LENGTH(*char-expr*)
†LINENO
MDY(*int-expr, int-expr, int-expr*)
MONTH(*date-expression*)
ORD(*string-expr*)
†PAGENO
int-expr SPACE
int-expr SPACES
TIME
TODAY
int-expr UNITS *time-keyword*
expression USING *format-string*
WEEKDAY (*date-expression*)
†*char-expr* WORDWRAP
YEAR (*date-expression*)

†Valid only in REPORT blocks.

Library Functions

Calling C Functions from 4GL

Popping Numbers

```
extern  void  popint(int  *iv)
extern  void  popshort(short  *siv)
extern  void  poplong(long  *liv)
extern  void  popflo(float  *fv)
extern  void  popdub(double  *dfv)
extern  void  popdec(dec_t  *decv)
```

Popping Characters

```
extern void popquote(char *qv, int len)
extern void popvchar(char *qv, int len)
```

Popping Dates and Times

```
extern void popdate(long *datv)
extern void popdtime(dtime_t *dtv, int qual)
extern void popinv(intrvl_t *iv, int qual)
```

Popping BYTE and TEXT

```
extern void poplocator(loc_t **blob)
```

Returning Values

```
extern void retint(int iv)
extern void retshort(short siv)
extern void retlong(long lv)
extern void retflo(float fv)
extern void retdub(double dfv)
extern void retdec(dec_t *decv)

extern void retquote(char *str0)
extern void retvchar(char *vc)

extern void retdate(long date)
extern void retdtime(dtime_t *dtv)

extern void retinv(intrvl_t *inv)
```

Pushing Values

```
extern void pushint(int iv)
extern void pushshort(short siv)
extern void pushlong(long liv)
extern void pushflo(float fv)
extern void pushdub(double dfv)
extern void pushdec(dec_t *decv, unsigned decp)
```

Calling 4GL Functions from C

fgl_start(*filename, argc, argv*) char *filename; int argc; char *argv[];	*initialize resources for the 4GL environment*
fgl_call(*funcname, nparams*) char *funcname; int nparams;	*call the 4GL function*
fgl_exitfm( )	*reset terminal to character mode*
fgl_end( )	*free 4GL resources*

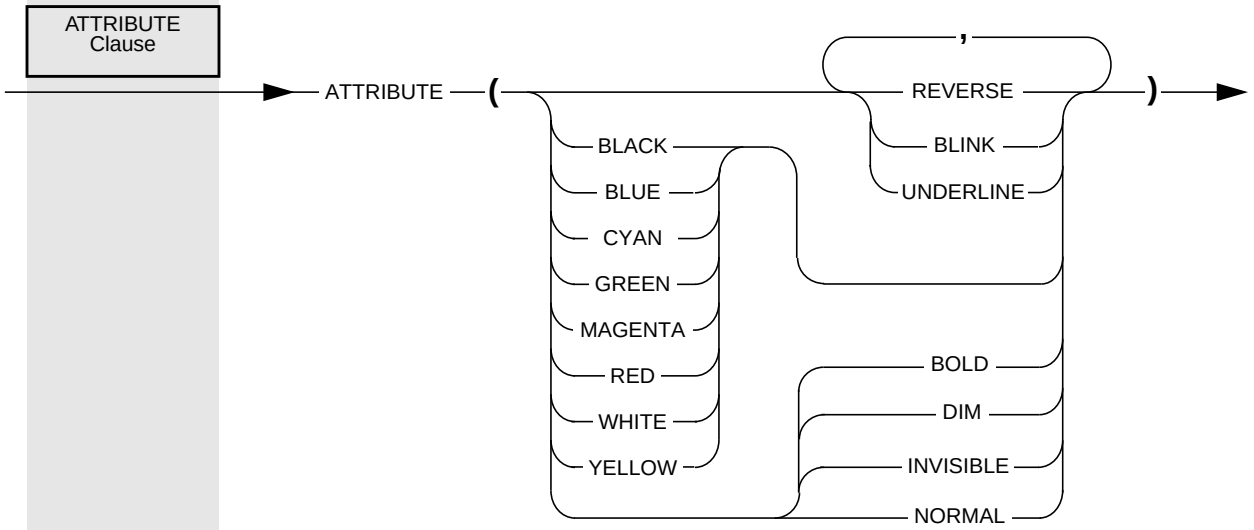
Decimal Functions

deccvasc(cp, len, np) char *cp; int len; dec_t *np;	*convert C char type to DECIMAL type*
dectoasc(np, cp, len, right) dec_t *np; char *cp; int len; int right;	*convert DECIMAL type to C char type*
deccvint(integer, np) int integer; dec_t *np;	*convert C int type to DECIMAL type*
dectoint(np, ip) dec_t *np; int *ip;	*convert DECIMAL type to C int type*
deccvlong(lng, np) long lng; dec_t *np;	*convert C long type to DECIMAL type*
dectolong(np, lngp) dec_t *np; long *lngp;	*convert DECIMAL type to C long type*
deccvflt(flt, np) float flt; dec_t *np;	*convert C float type to DECIMAL type*
dectoflt(np, fltp) dec_t *np; float *fltp;	*convert DECIMAL type to C float type*

Basics
4GL
Forms
Reports
SQL
SQLCA
Debugger
Variables
Keys

```
deccvdbl(dbl, np)                          convert C double type to DECIMAL type
    double dbl;
    dec_t *np;

dectodbl(np, dblp)                         convert DECIMAL type to C double type
    dec_t *np;
    double *dblp;

decadd(n1, n2, result)                     add two decimal numbers
    dec_t  *n1;                            (result = n1 + n2)
    dec_t  *n2;
    dec_t  *result;

decsub(n1, n2, result)                     subtract two decimal numbers
    dec_t  *n1;                            (result = n1 - n2)
    dec_t  *n2;
    dec_t  *result;

decmul(n1, n2, result)                     multiply two decimal numbers
    dec_t  *n1;                            (result = n1 * n2)
    dec_t  *n2;
    dec_t  *result;

decdiv(n1, n2, result)                     divide two decimal numbers
    dec_t  *n1;                            (result = n1 / n2)
    dec_t  *n2;
    dec_t  *result;

int deccmp(n1, n2)                         compare two decimal numbers
    dec_t *n1;
    dec_t *n2;

deccopy(n1, n2)                            copy a decimal number
    dec_t *n1;
    dec_t *n2;

char *dececvt(np, ndigit, decpt, sign)     convert decimal value to ASCII string
    dec_t *np;
    int ndigit;
    int *decpt;
    int *sign;

char *decfcvt(np, ndigit, decpt, sign)     convert decimal value to ASCII  string
    dec_t *np;
    int ndigit;
    int *decpt;
    int *sign;
```

Display Attributes



The ATTRIBUTE clause is used in these **4GL** statements:

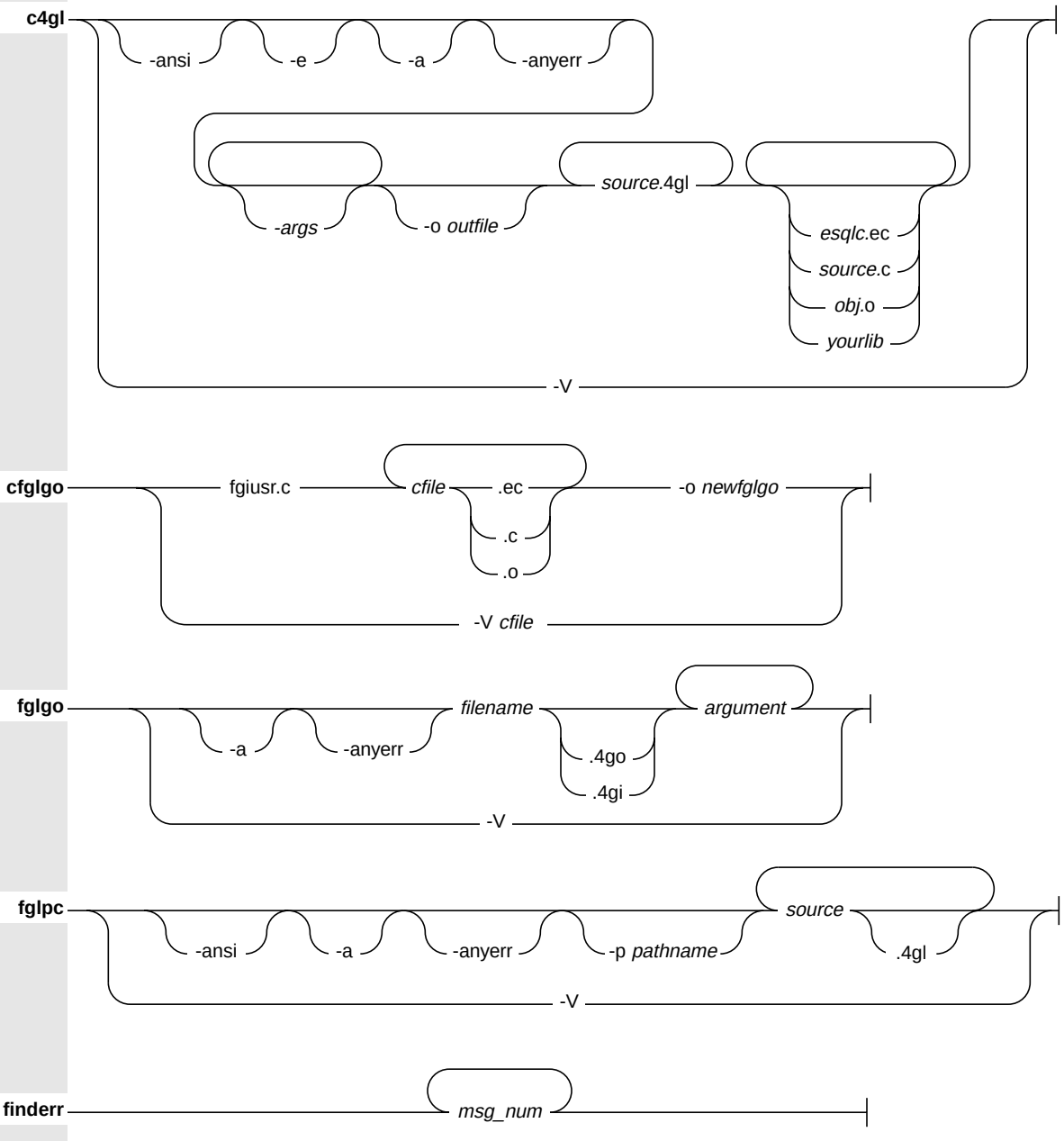
CONSTRUCT	DISPLAY FORM	INPUT ARRAY
DISPLAY	ERROR	MESSAGE
DISPLAY ARRAY	INPUT	PROMPT

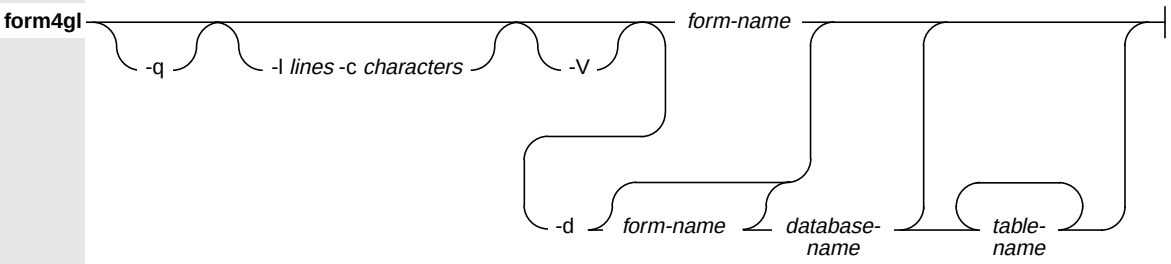
For all ATTRIBUTE clauses and field attributes the following table shows the effects of the color attributes on a monochrome monitor, as well as the effects of the intensity attributes on a color monitor:

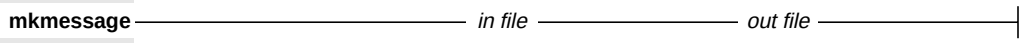
Color Attribute	Monochrome Display	Intensity Attribute	Color Display
WHITE	NORMAL	NORMAL	WHITE
YELLOW	BOLD	BOLD	RED
MAGENTA	BOLD	DIM	BLUE
RED	BOLD		
CYAN	DIM		
GREEN	DIM		
BLUE	DIM		
BLACK	DIM		

Basics
4GL
Forms
Reports
SQL
SQLCA
Debugger
Variables
Keys

Command-Line Syntax







Basics
4GL
Forms
Reports
SQL
SQLCA
Debugger
Variables
Keys

4GL Statements

INFORMIX-4GL supports the SQL language, but it is sometimes convenient to distinguish between SQL statements and other 4GL statements:

- SQL statements operate on tables in the database.
- Other 4GL statements operate on variables in memory.

The SQL statements of 4GL can be divided into these functional categories.

Note: *Not all of these SQL statements listed on this and the next page are directly supported by* **4GL**. *If the statement or any part of its syntax is preceded by a 6.0 icon in its syntax diagram later in this chapter, the statement must be prepared (by using the PREPARE statement). Preparing SQL statements is described in Chapter 3 of the INFORMIX-4GL Reference.*

Types of SQL Statements

SQL Data Definition Statements

ALTER INDEX	CREATE TABLE	DROP TABLE
ALTER TABLE	CREATE VIEW	DROP VIEW
CLOSE DATABASE	DATABASE	RENAME COLUMN
CREATE DATABASE	DROP DATABASE	RENAME TABLE
CREATE INDEX	DROP INDEX	
CREATE SYNONYM	DROP SYNONYM	

SQL Data Manipulation Statements

INSERT	LOAD	UNLOAD
DELETE	SELECT	UPDATE

SQL Cursor Manipulation Statements

CLOSE	FETCH	OPEN
DECLARE	FLUSH	PUT

SQL Query Optimization Information Statements

SET EXPLAIN	SET OPTIMIZATION	UPDATE STATISTICS

SQL Data Access Statements

GRANT	REVOKE	SET LOCK MODE
LOCK TABLE	OL SET ISOLATION	UNLOCK TABLE

SQL Data Integrity Statements

	BEGIN WORK	SE	DROP AUDIT	SE	ROLLFORWARD DATABASE
SE	CHECK TABLE	SE	RECOVER TABLE	OL	SET CONSTRAINTS
	COMMIT WORK	SE	REPAIR TABLE	OL	SET LOG
SE	CREATE AUDIT		ROLLBACK WORK	SE	START DATABASE

Note: *The data integrity statements marked with the* SE *symbol are supported only by the INFORMIX-SE engine. Statements marked* OL *can only be used with the INFORMIX-OnLine Dynamic Server engine.*

Other Types of 4GL Statements

SQL Dynamic Management Statements

EXECUTE
FREE
PREPARE

4GL Definition and Declaration Statements

DEFINE
FUNCTION
MAIN
REPORT

4GL Program Flow Control Statements

CALL
CASE
CONTINUE
DATABASE
END
EXIT
FINISH REPORT
FOR
FOREACH
GOTO
IF
LABEL
OUTPUT TO REPORT
RETURN
RUN
START REPORT
WHILE

4GL Compiler Directives

DATABASE
DEFER
GLOBALS
WHENEVER

4GL Storage Manipulation Statements

INITIALIZE
LET
LOCATE
VALIDATE

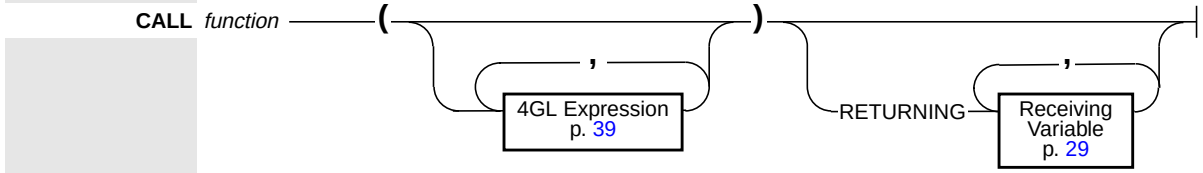
4GL Screen Interaction Statements

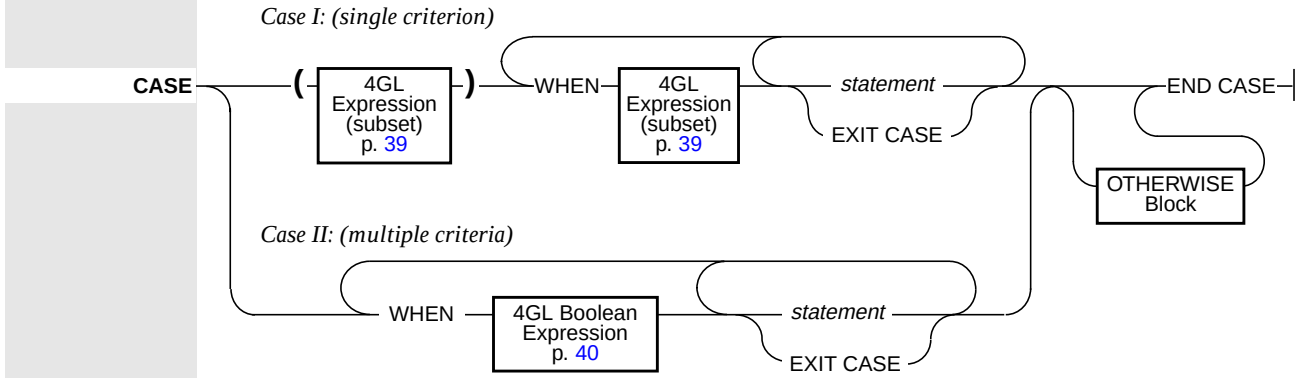
CLEAR
CLOSE FORM
CLOSE WINDOW
CONSTRUCT
CURRENT WINDOW
DISPLAY
DISPLAY ARRAY
DISPLAY FORM
ERROR
INPUT
INPUT ARRAY
MENU
MESSAGE
OPEN FORM
OPEN WINDOW
OPTIONS
PROMPT
SCROLL
SLEEP

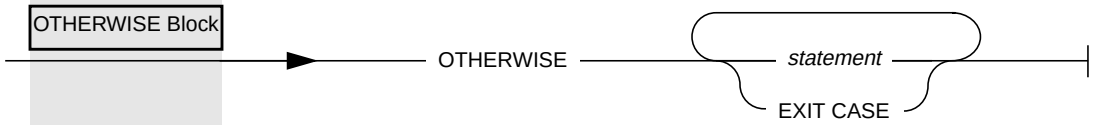
4GL Report Execution Statements

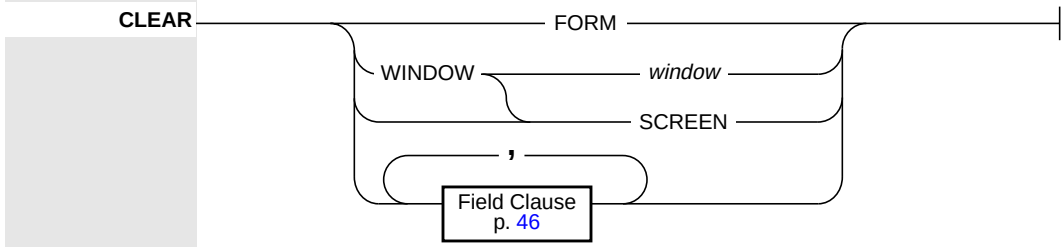
NEED
PAUSE
PRINT
SKIP

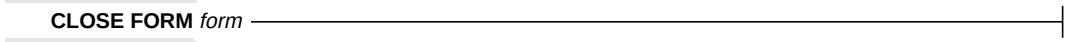
Most 4GL statements are not sensitive to whether the SE or the OnLine engine supports the application. Only the OnLine engine, however, can store values in BYTE, TEXT, or VARCHAR columns, or can accept *database:* or *database@system:* as qualifiers to names of tables, views, or synonyms.

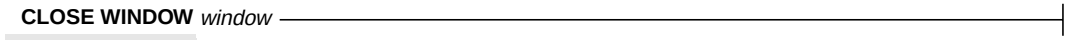

CALL function
(
)
,
4GL Expression
p. 39
RETURNING
,
Receiving
Variable
p. 29

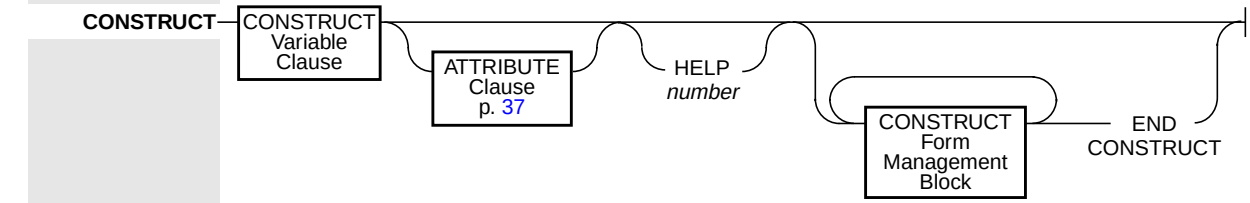

Case I: (single criterion)
CASE
(
4GL
Expression
(subset)
p. 39
)
WHEN
4GL
Expression
(subset)
p. 39
statement
EXIT CASE
END CASE
OTHERWISE
Block
Case II: (multiple criteria)
WHEN
4GL Boolean
Expression
p. 40
statement
EXIT CASE

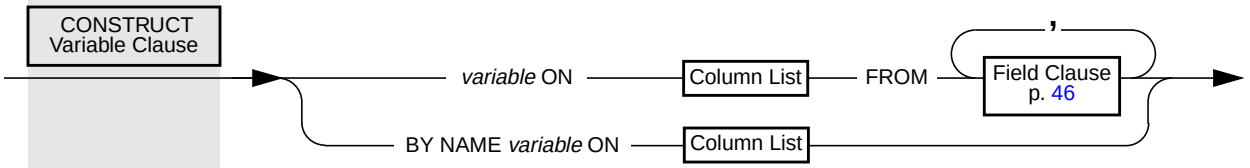

OTHERWISE Block
OTHERWISE
statement
EXIT CASE

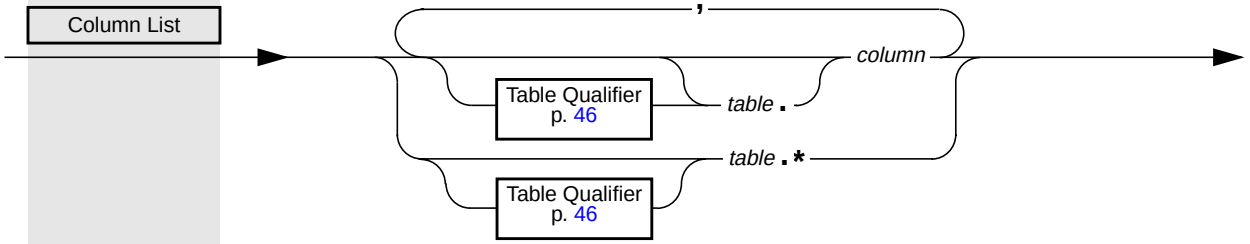

CLEAR
FORM
WINDOW
window
SCREEN
,
Field Clause
p. 46

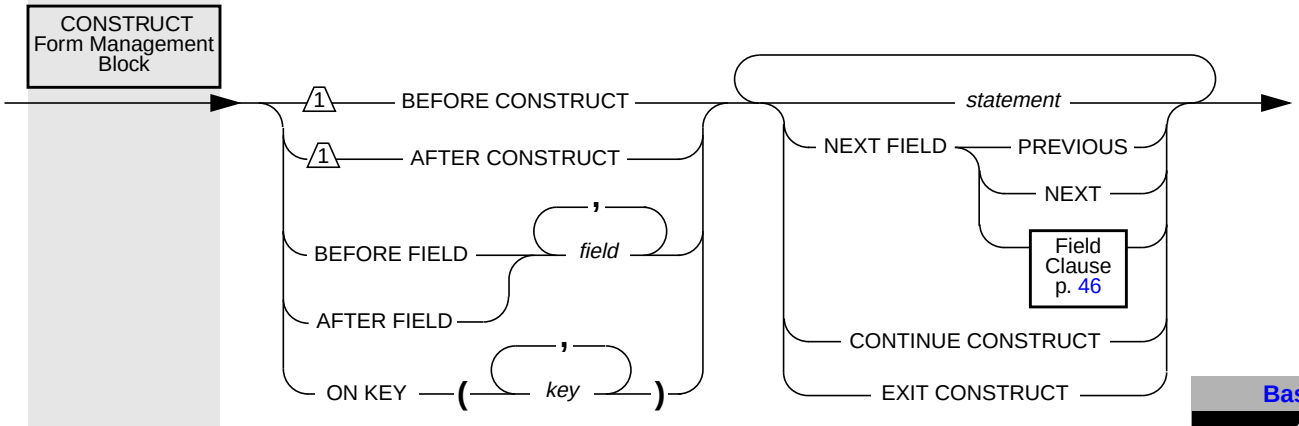

CLOSE FORM form

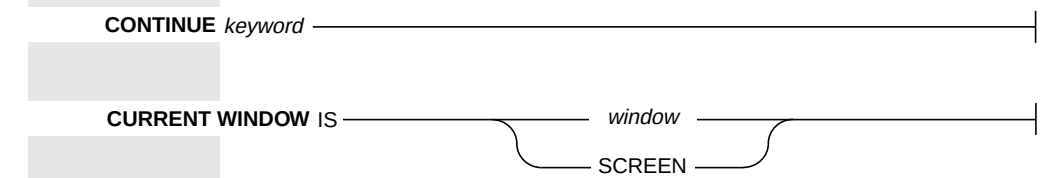

CLOSE WINDOW window

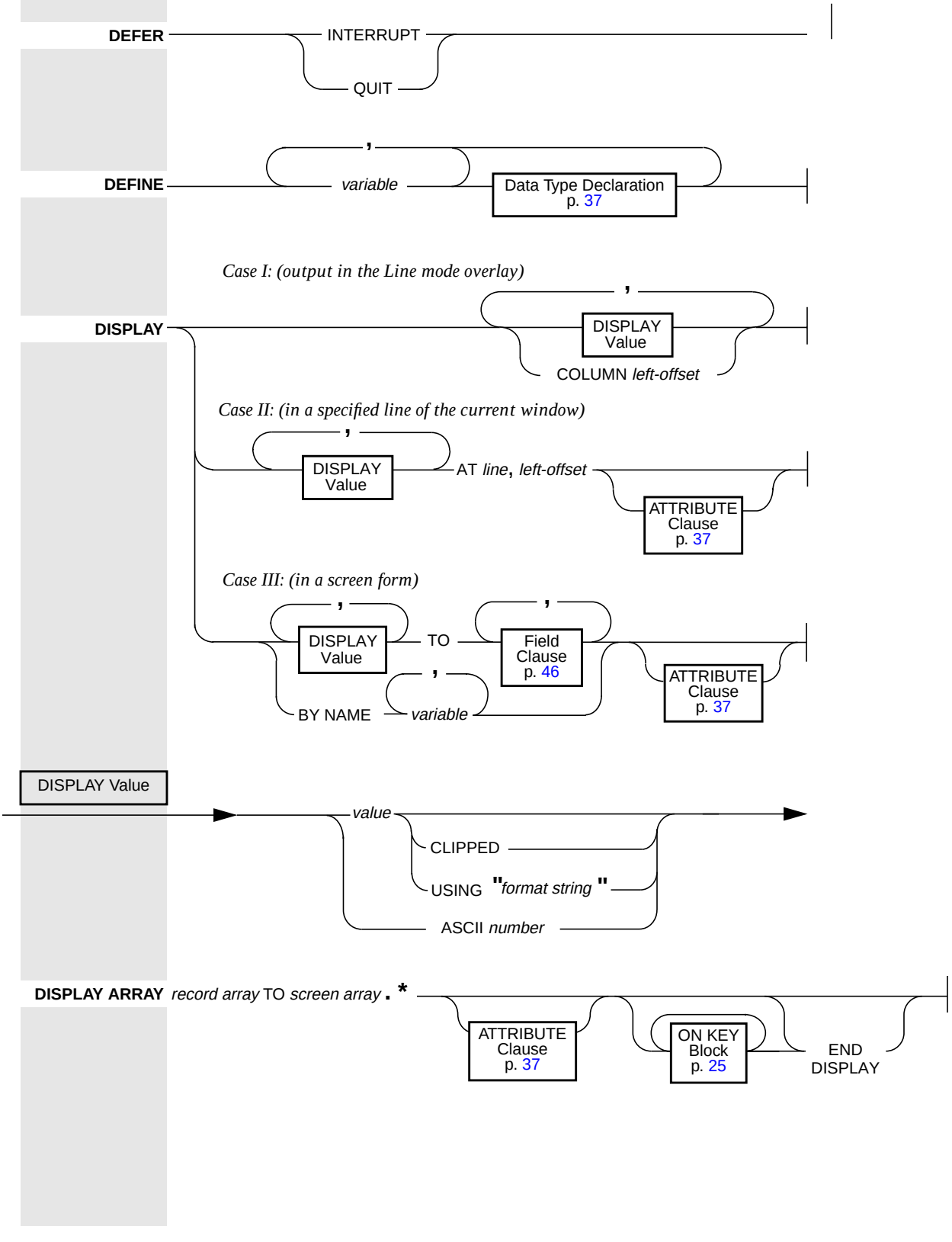

CONSTRUCT
CONSTRUCT Variable Clause
ATTRIBUTE Clause p. 37
HELP number
CONSTRUCT Form Management Block
END CONSTRUCT

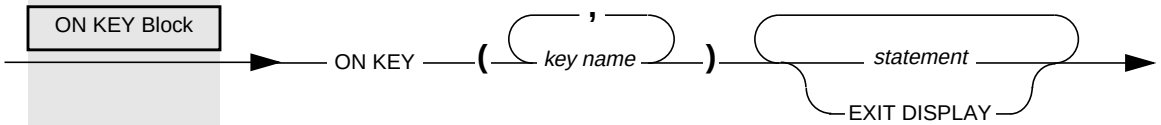

CONSTRUCT Variable Clause
variable ON
Column List
FROM
Field Clause p. 46
BY NAME variable ON
Column List

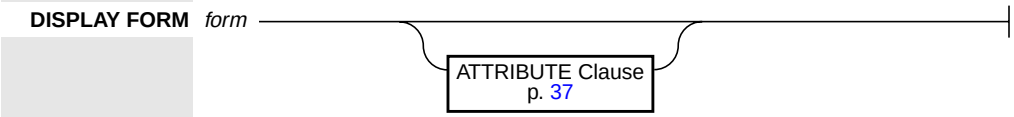

Column List
Table Qualifier p. 46
table .
column
table .*
Table Qualifier p. 46

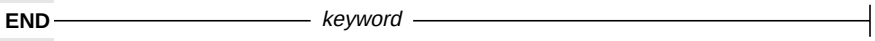

CONSTRUCT Form Management Block
1
BEFORE CONSTRUCT
statement
1
AFTER CONSTRUCT
NEXT FIELD
PREVIOUS
NEXT
BEFORE FIELD
field
Field Clause p. 46
AFTER FIELD
CONTINUE CONSTRUCT
ON KEY
key
EXIT CONSTRUCT


CONTINUE keyword
CURRENT WINDOW IS
window
SCREEN


DEFER
INTERRUPT
QUIT
DEFINE
,
variable
Data Type Declaration
p. 37
Case I: (output in the Line mode overlay)
DISPLAY
,
DISPLAY Value
COLUMN left-offset
Case II: (in a specified line of the current window)
,
DISPLAY Value
AT line, left-offset
ATTRIBUTE Clause
p. 37
Case III: (in a screen form)
,
DISPLAY Value
TO
,
Field Clause
p. 46
ATTRIBUTE Clause
p. 37
BY NAME
,
variable
DISPLAY Value
value
CLIPPED
USING "format string"
ASCII number
DISPLAY ARRAY record array TO screen array . *
ATTRIBUTE Clause
p. 37
ON KEY Block
p. 25
END DISPLAY

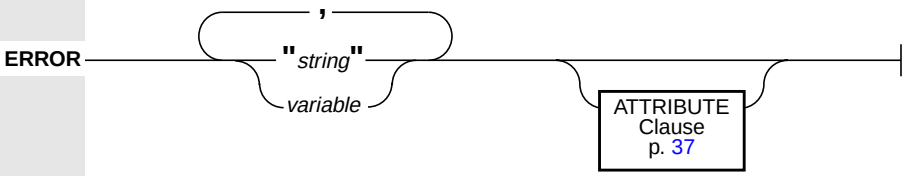




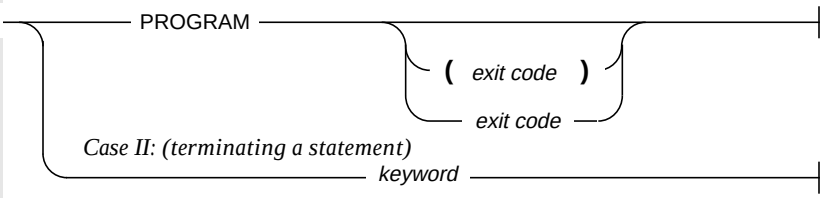


These are valid *keywords* in the END statement:

CASE	FOREACH	INPUT	PROMPT
CONSTRUCT	FUNCTION	MAIN	REPORT
DISPLAY	GLOBALS	MENU	WHILE
FOR	IF		



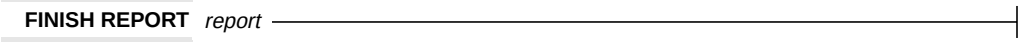
Case I: (terminating a program)

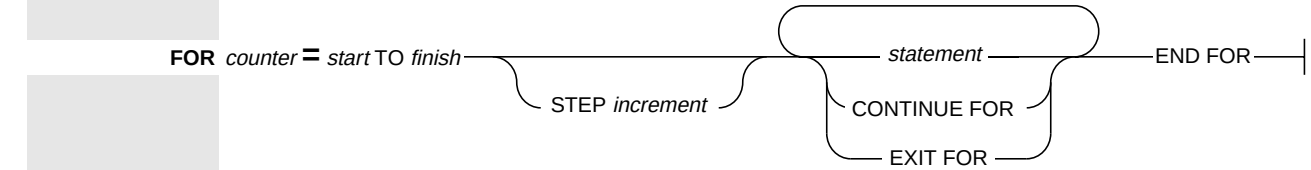


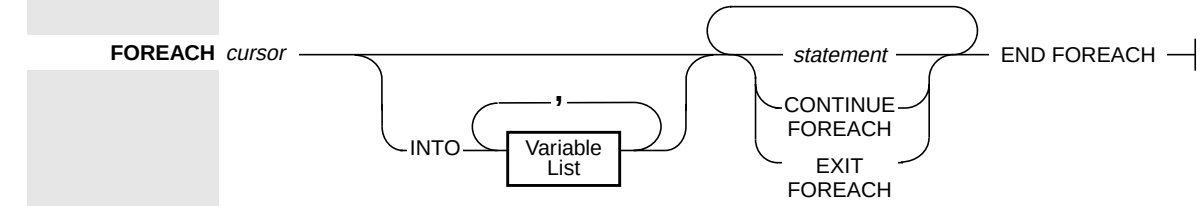
Case II: (terminating a statement)

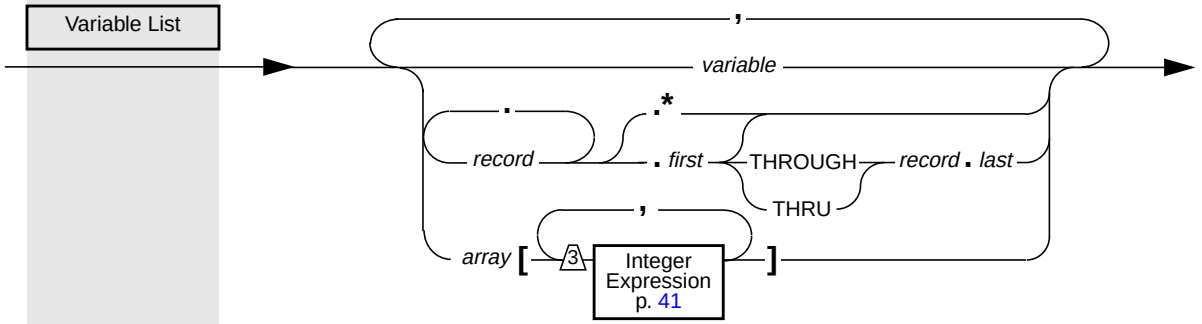
These are valid *keywords* in Case II of the EXIT statement:

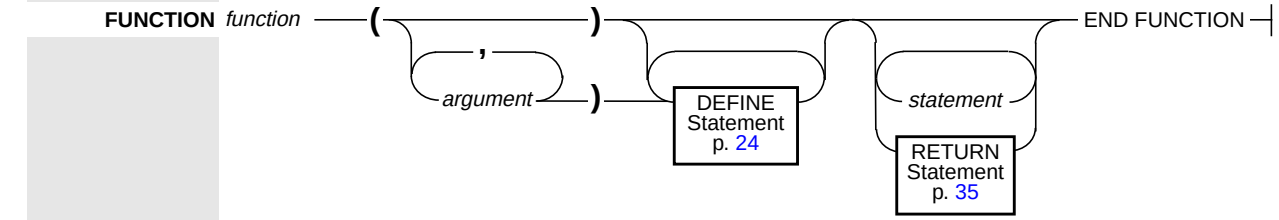
CASE	FOR	MENU
CONSTRUCT	FOREACH	PROMPT
DISPLAY	INPUT	WHILE



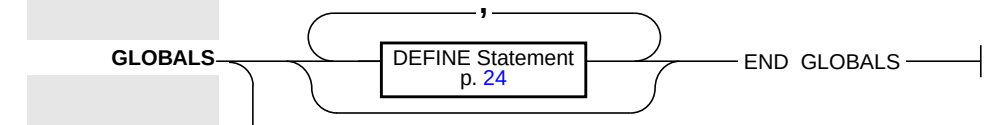




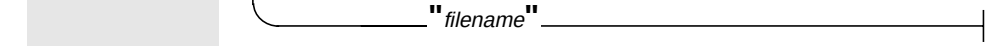


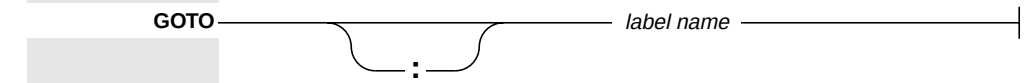


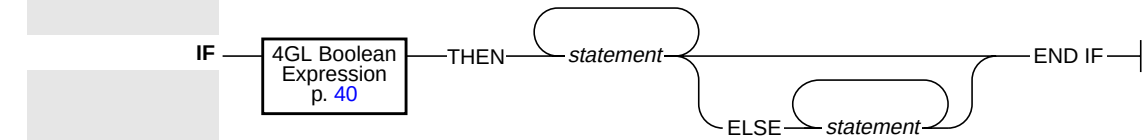
Case I: (declaring and exporting variables)

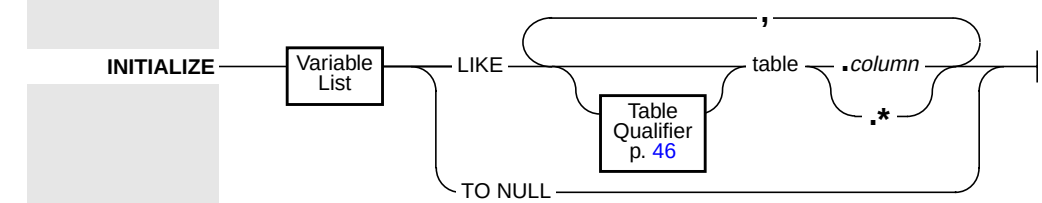


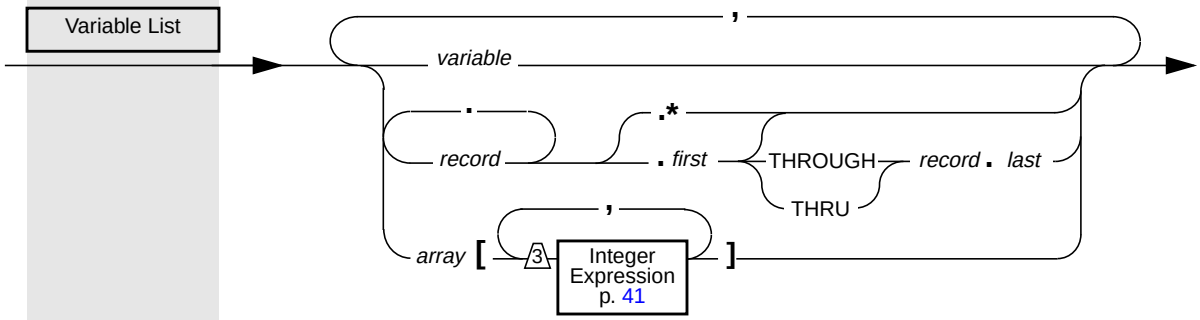
Case II: (importing variables)

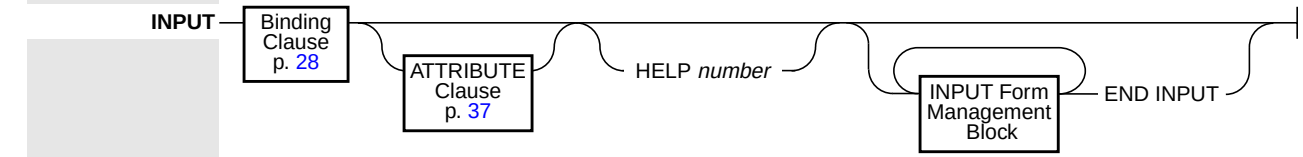


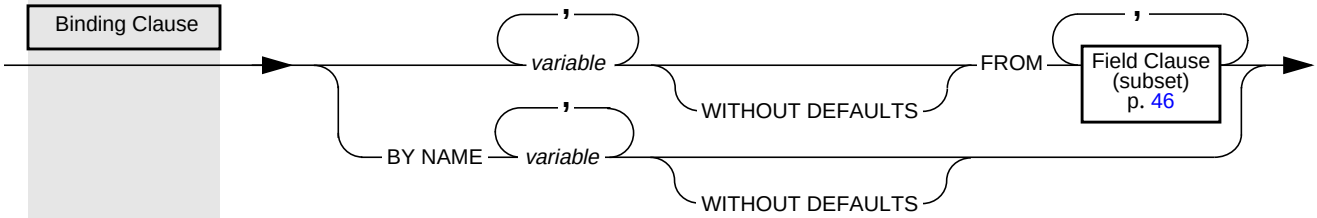

GOTO
:
label name

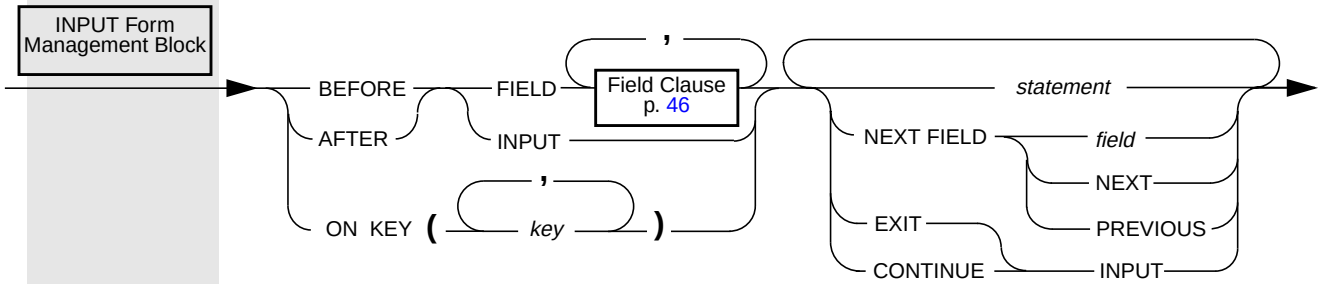

IF
4GL Boolean Expression p. 40
THEN
statement
ELSE
statement
END IF


INITIALIZE
Variable List
LIKE
,
table
.column
.*
Table Qualifier p. 46
TO NULL


Variable List
,
variable
.
record
.*
. first
THROUGH
THRU
record . last
,
array [
3
Integer Expression p. 41
]


INPUT
Binding Clause p. 28
ATTRIBUTE Clause p. 37
HELP number
INPUT Form Management Block
END INPUT



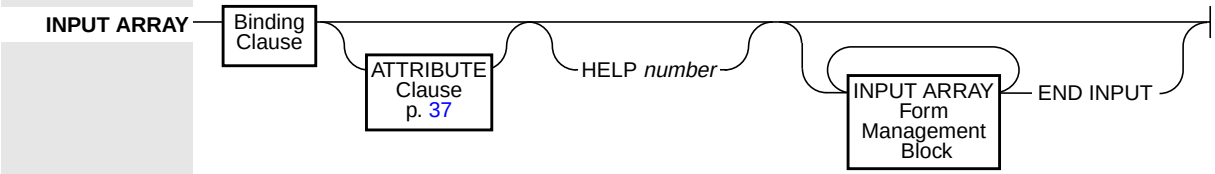


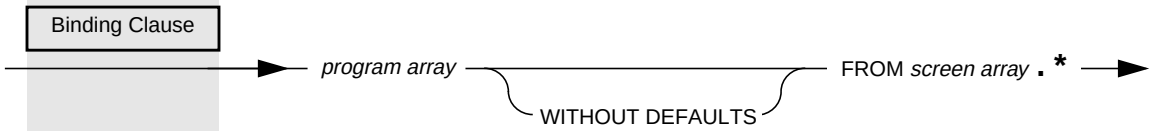
Acceptable values of *key* (in lowercase or uppercase letters) for the ON KEY block are:

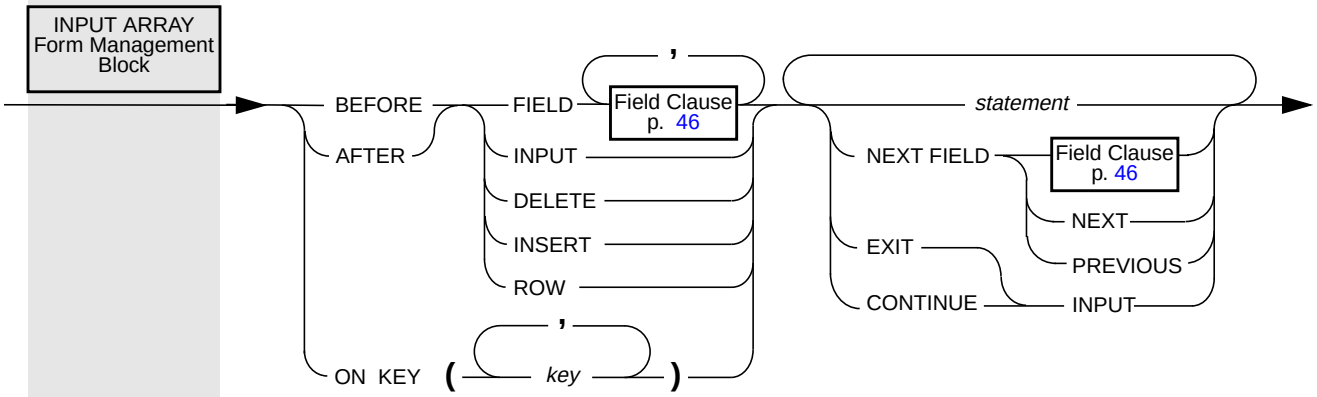
ACCEPT	HELP	NEXT *or* NEXTPAGE	RETURN *or* ENTER
DELETE	INSERT	PREVIOUS *or* PREVPAGE	RIGHT
DOWN	INTERRUPT		TAB
ESC *or* ESCAPE	LEFT		UP
F1 *through* F64			
CONTROL-*char* (except A, D, H, I, J, L, M, R, or X)			

Built-in functions that access field buffers and keystroke buffers:

Built-In Funtion	**Description**
FIELD_TOUCHED(*field*)	Returns TRUE when the user has made a change to screen *field*.
GET_FLDBUF(*field-list*)	Returns the character values of the contents of one or more fields.
FGL_LASTKEY( )	Returns an INTEGER value corresponding to the most recent keystroke.
INFIELD(*field*)	Returns TRUE if *field* is the name of the current screen field.



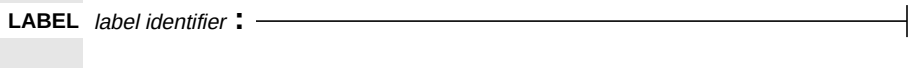


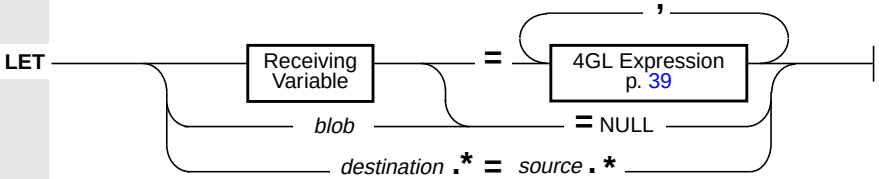


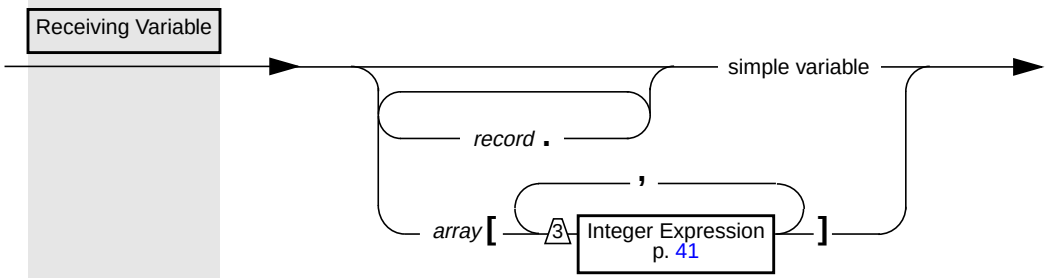
For acceptable values of *key*, see p. 28. For built-in functions that access field buffers and keystroke buffers, see p. 28.

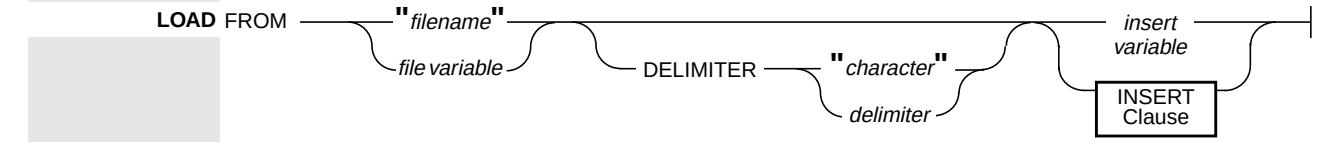
Built-in functions that keep track of the relative states of the screen cursor, the program array, and the screen array:

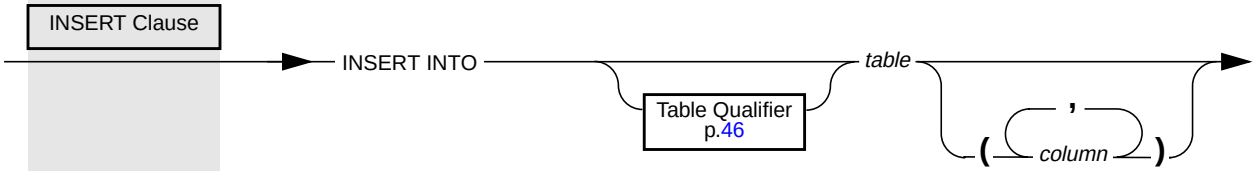
Function	Description
ARR_CURR( )	Returns the number of the *current record* of the program array.
ARR_COUNT( )	Returns the current number of records in the program array.
SCR_LINE( )	Returns the number of the current line within the screen array.
SET_COUNT(*filled-rows*)	Sets the initial value of ARR_COUNT( ) to *filled-rows*.

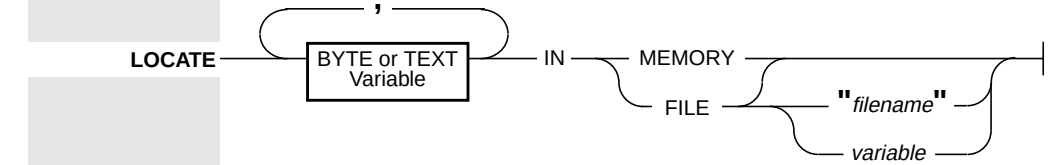


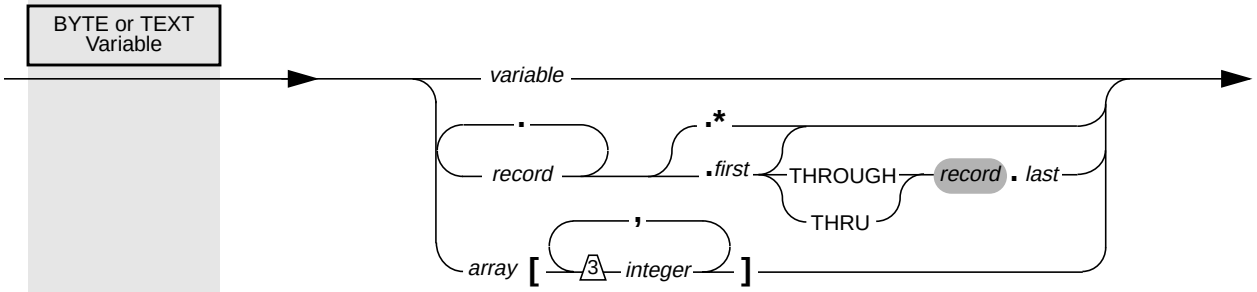


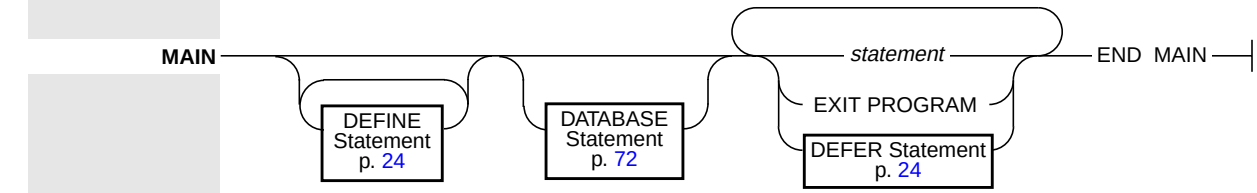


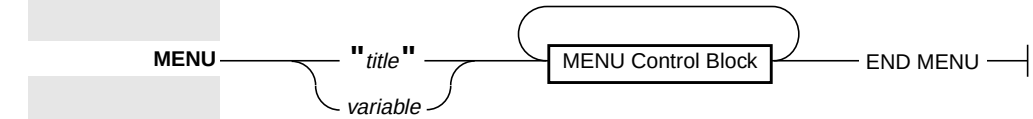

LOAD FROM
"filename"
file variable
DELIMITER
"character"
delimiter
insert variable
INSERT Clause

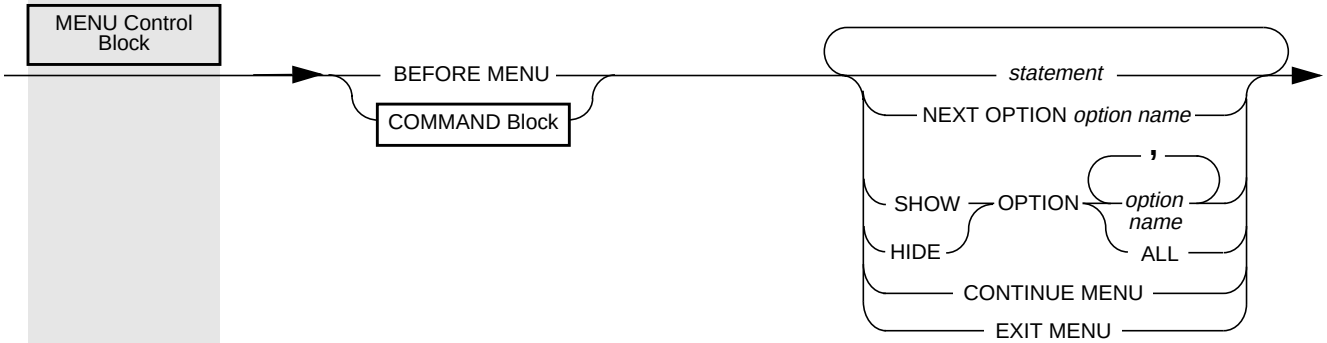

INSERT Clause
INSERT INTO
Table Qualifier p.46
table
(
,
column
)

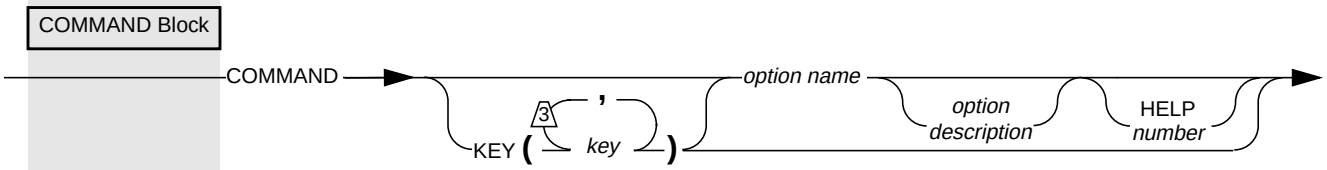

LOCATE
,
BYTE or TEXT Variable
IN
MEMORY
FILE
"filename"
variable

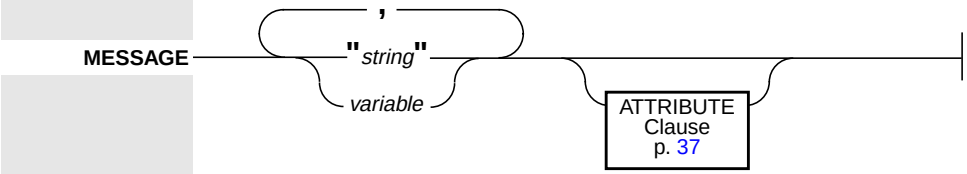

BYTE or TEXT Variable
variable
.
record
.*
.first
THROUGH
THRU
record
. last
array
[
,
3
integer
]

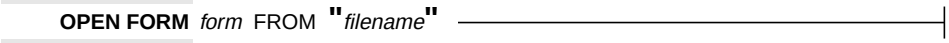

MAIN
DEFINE Statement p. 24
DATABASE Statement p. 72
statement
EXIT PROGRAM
DEFER Statement p. 24
END MAIN

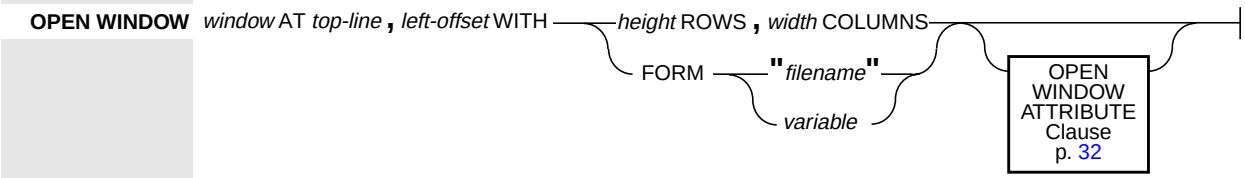

MENU
"title"
variable
MENU Control Block
END MENU


MENU Control Block
BEFORE MENU
COMMAND Block
statement
NEXT OPTION option name
SHOW
HIDE
OPTION
option name
ALL
CONTINUE MENU
EXIT MENU

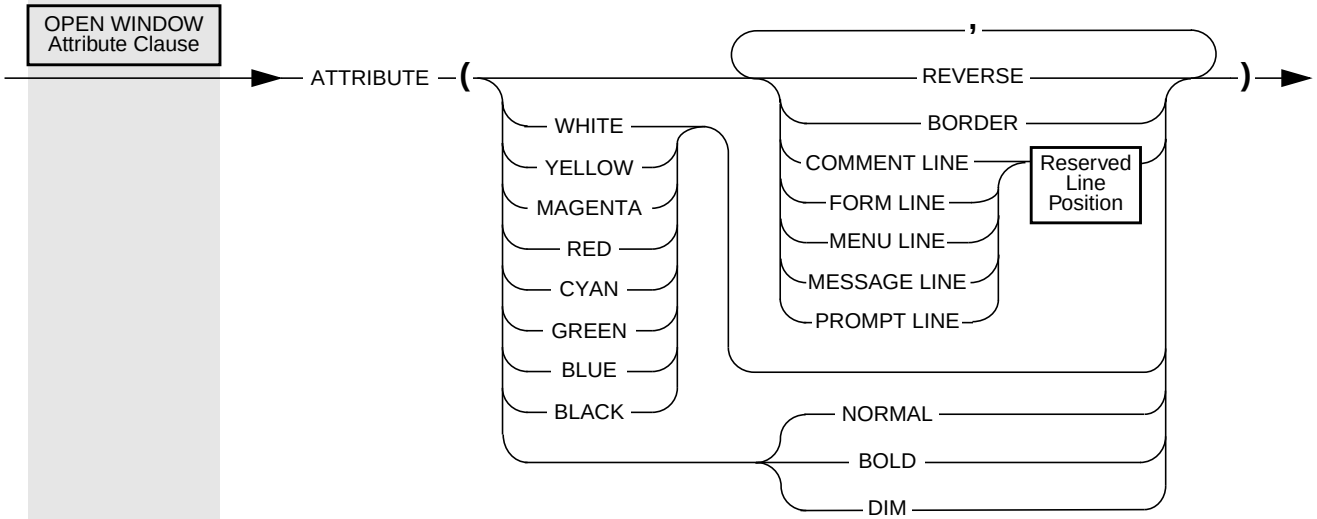

COMMAND Block
COMMAND
KEY
key
option name
option description
HELP number


MESSAGE
string
variable
ATTRIBUTE Clause p. 37


OPEN FORM form FROM filename

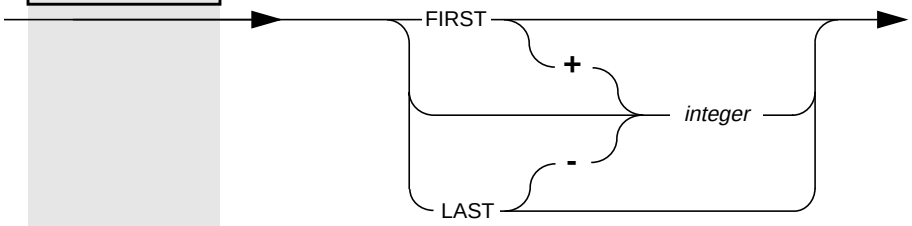

OPEN WINDOW window AT top-line , left-offset WITH
height ROWS , width COLUMNS
FORM
filename
variable
OPEN WINDOW ATTRIBUTE Clause p. 32

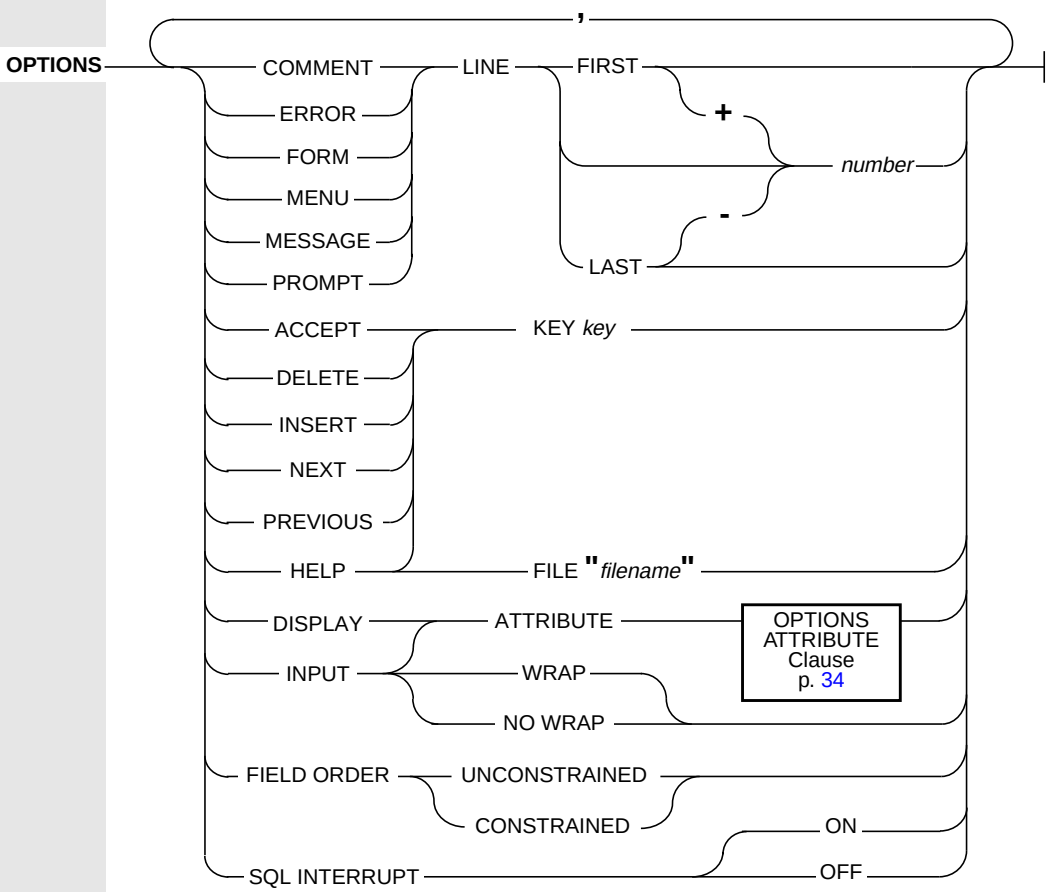
OPEN WINDOW Attribute Clause



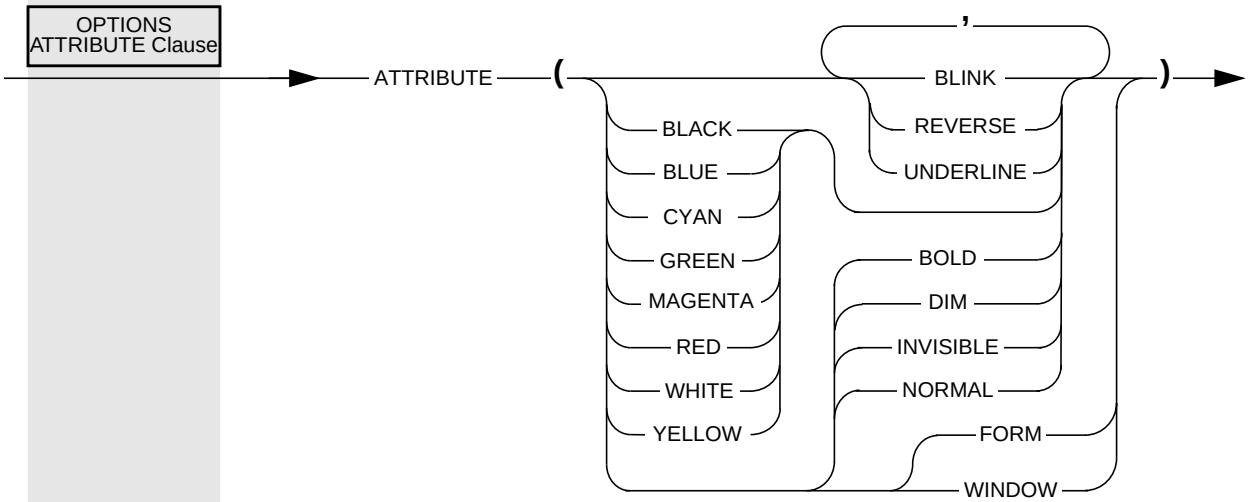
Attribute	Default Setting
color	Default foreground color on your terminal
REVERSE	No reverse video
BORDER	No border
PROMPT LINE *line value*	FIRST (=1)
MESSAGE LINE *line value*	FIRST + 1 (=2)
MENU LINE *line value*	FIRST (=1)
FORM LINE *line value*	FIRST + 2 (=3)
COMMENT LINE *line value*	LAST - 1 (*for the 4GL screen*) LAST (*for all other 4GL windows*)

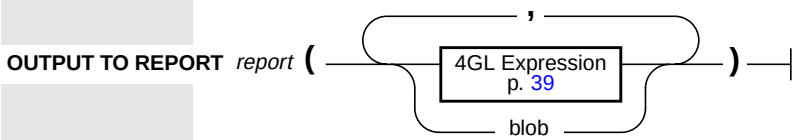
Reserved Line Position

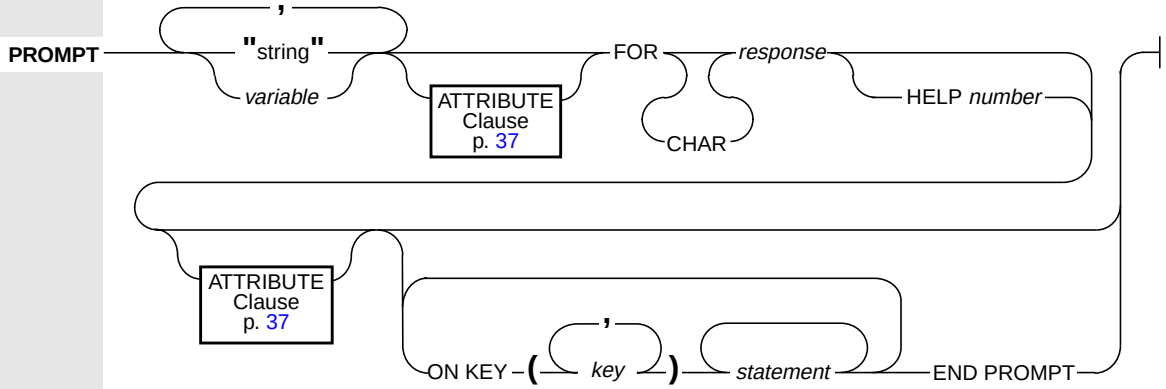




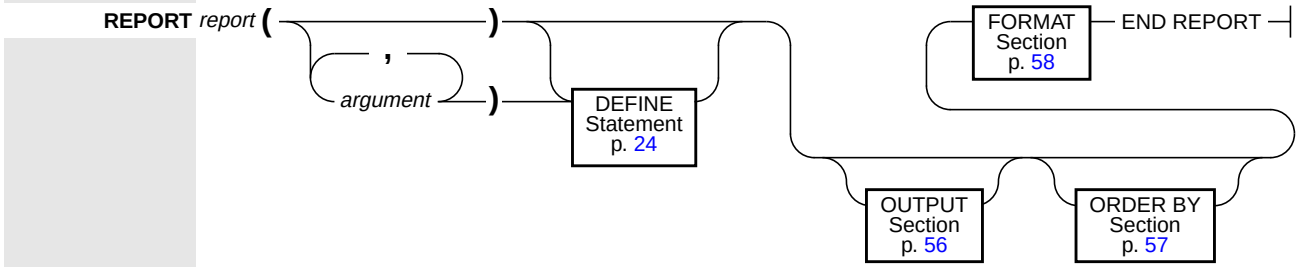
Clause	Default
COMMENT LINE	LAST - 1 for the 4GL screen LAST for all other 4GL windows
ERROR LINE	LAST line of the 4GL screen
FORM LINE	FIRST + 2 or line 3 of the current 4GL window
MENU LINE	FIRST line of the 4GL window
MESSAGE LINE	FIRST + 1 or line 2 of the current 4GL window
PROMPT LINE	FIRST line of the 4GL window
ACCEPT KEY	ESCAPE
DELETE KEY	F2
INSERT KEY	F1
NEXT KEY	F3
PREVIOUS KEY	F4
HELP KEY	CONTROL-W
HELP FILE	None

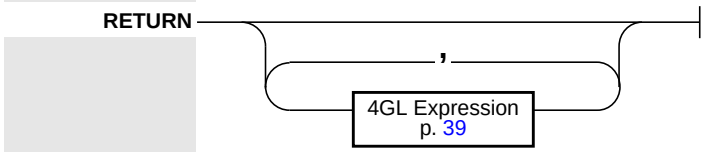


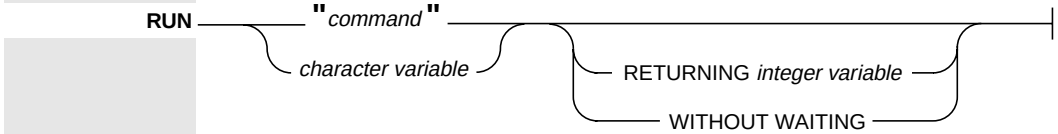


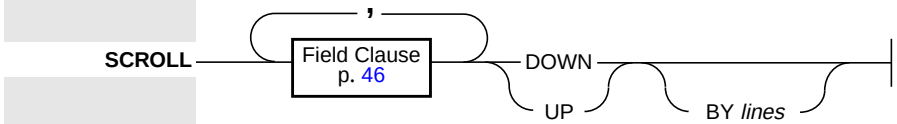


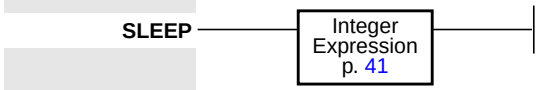
For values for *key*, see p. 28.

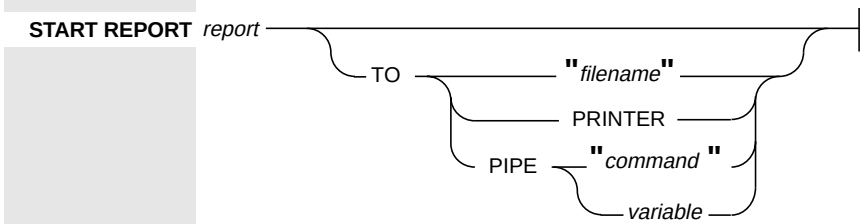

REPORT report ( ) , argument)
DEFINE Statement p. 24
FORMAT Section p. 58
END REPORT
OUTPUT Section p. 56
ORDER BY Section p. 57

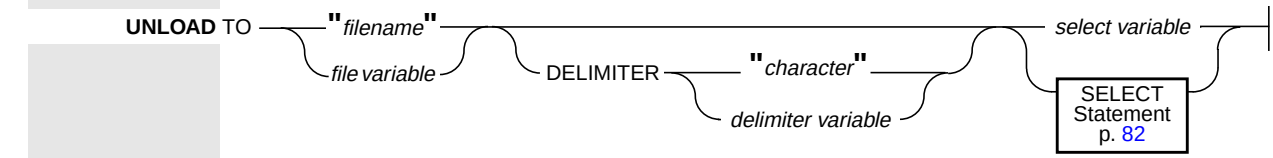

RETURN
,
4GL Expression p. 39

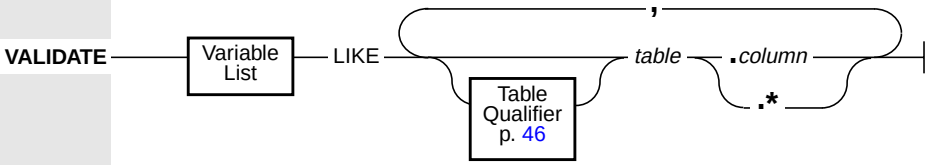

RUN
"command"
character variable
RETURNING integer variable
WITHOUT WAITING

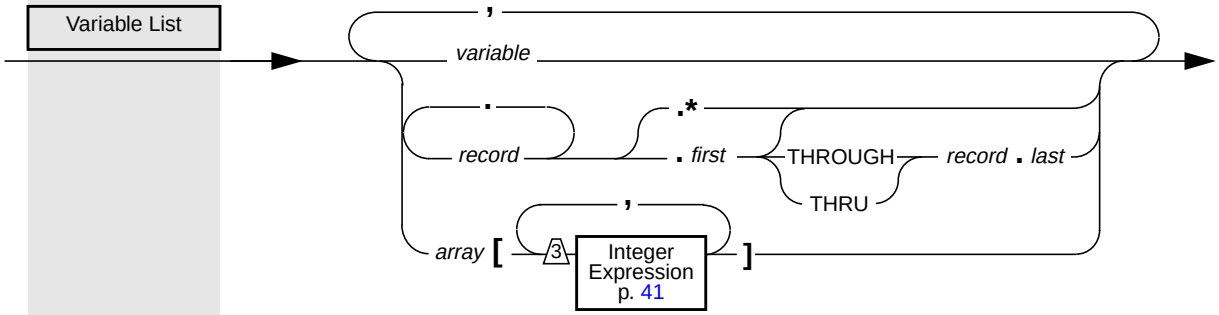

,
SCROLL
Field Clause p. 46
DOWN
UP
BY lines

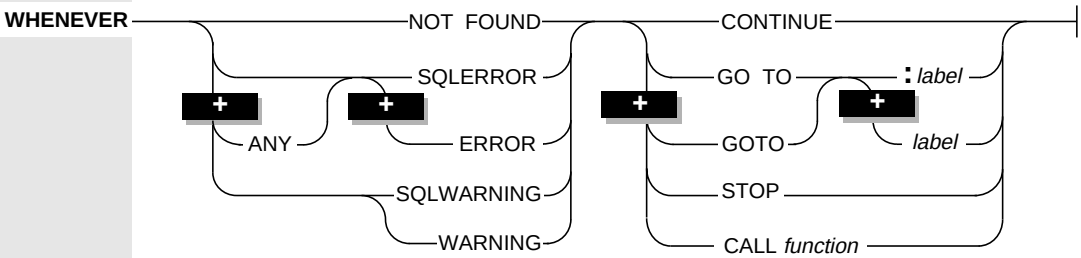

SLEEP
Integer Expression p. 41

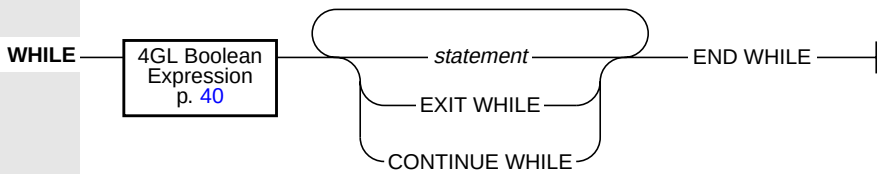

START REPORT report
TO
"filename"
PRINTER
PIPE
"command"
variable


UNLOAD TO
"filename"
file variable
DELIMITER
"character"
delimiter variable
select variable
SELECT Statement p. 82

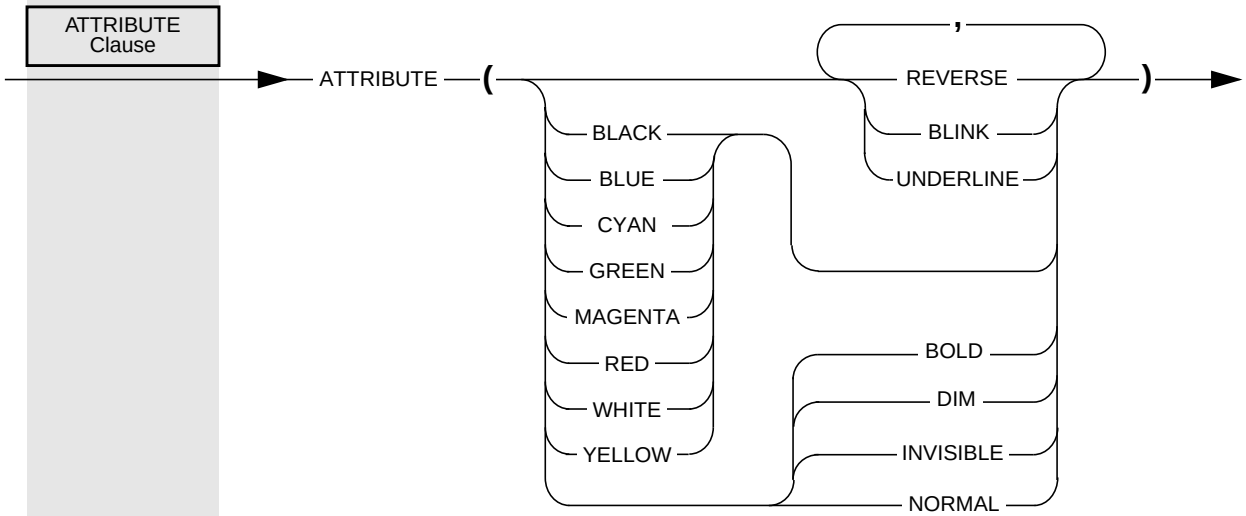

VALIDATE
Variable List
LIKE
,
table
.column
.*
Table Qualifier p. 46

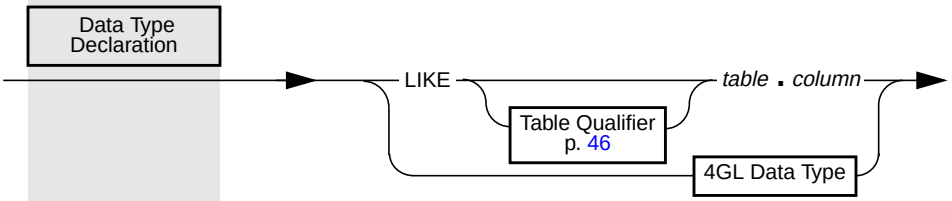

Variable List
,
variable
.
record
.*
. first
THROUGH
THRU
record . last
,
array [
3
Integer Expression p. 41
]

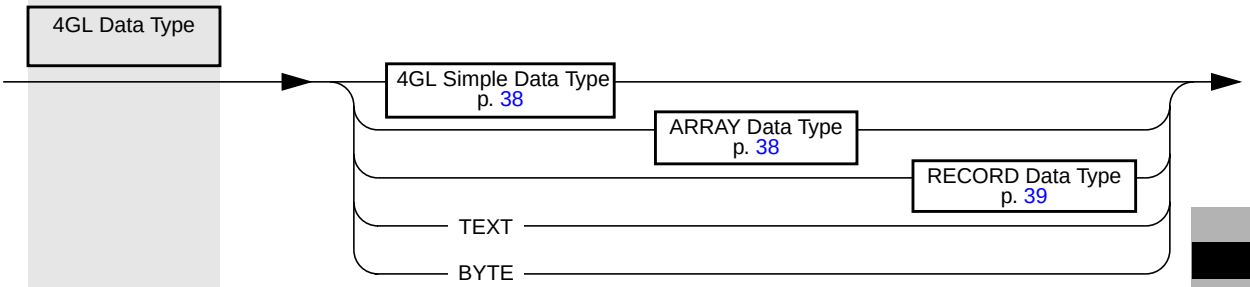

WHENEVER
NOT FOUND
SQLERROR
+
ANY
+
ERROR
SQLWARNING
WARNING
CONTINUE
GO TO
: label
+
GOTO
+
label
STOP
CALL function


WHILE
4GL Boolean Expression p. 40
statement
END WHILE
EXIT WHILE
CONTINUE WHILE

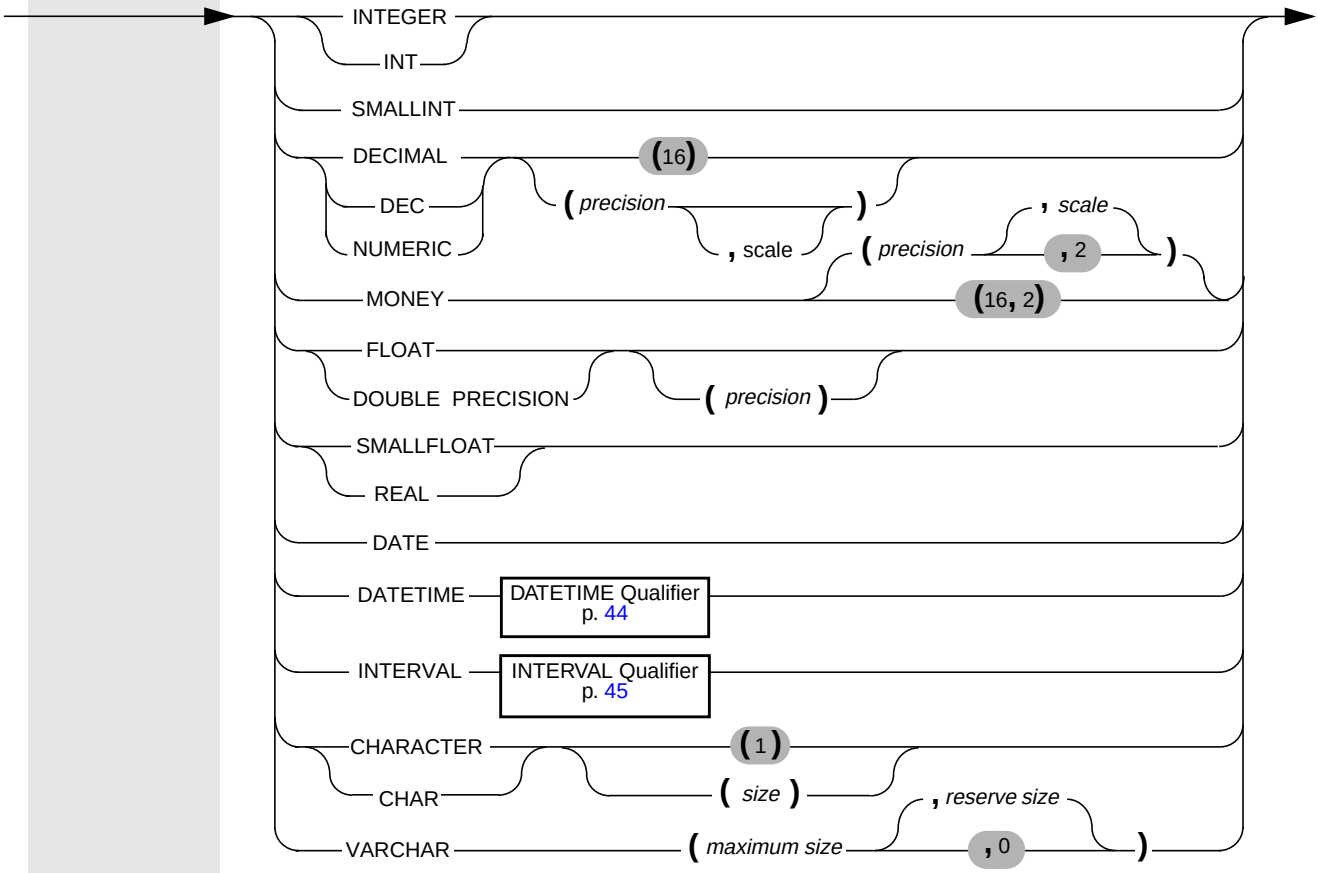
4GL Statement Segments



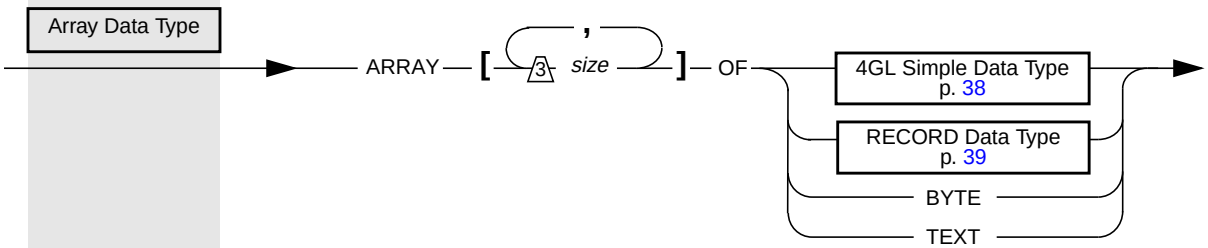


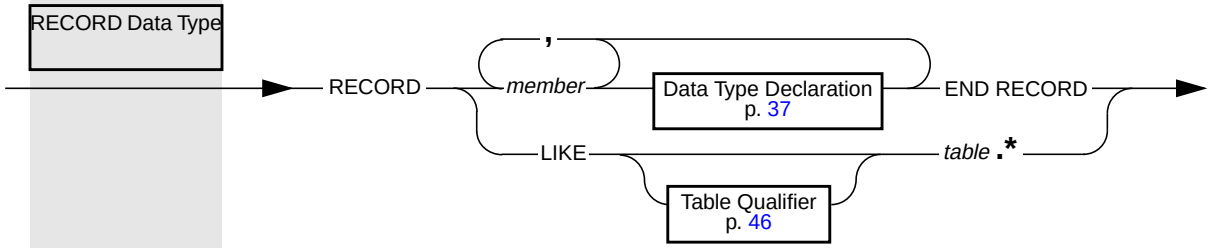


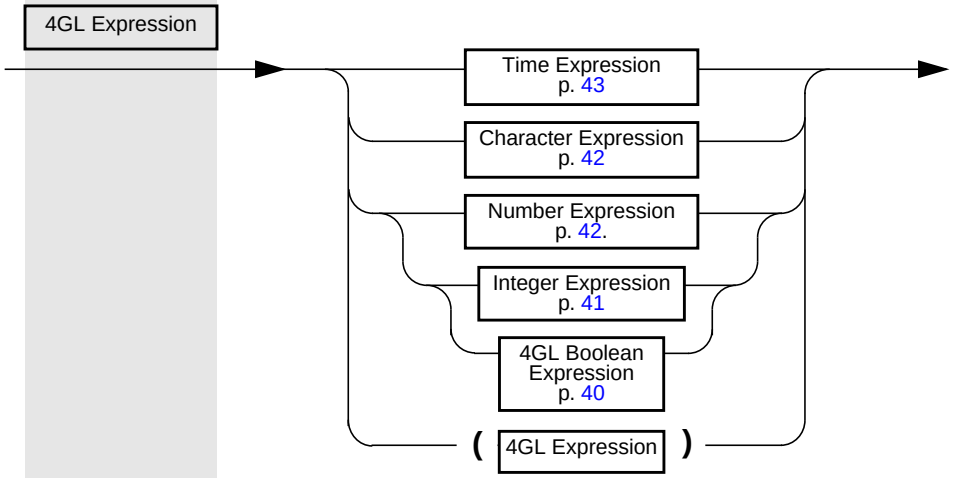
4GL Simple Data Type

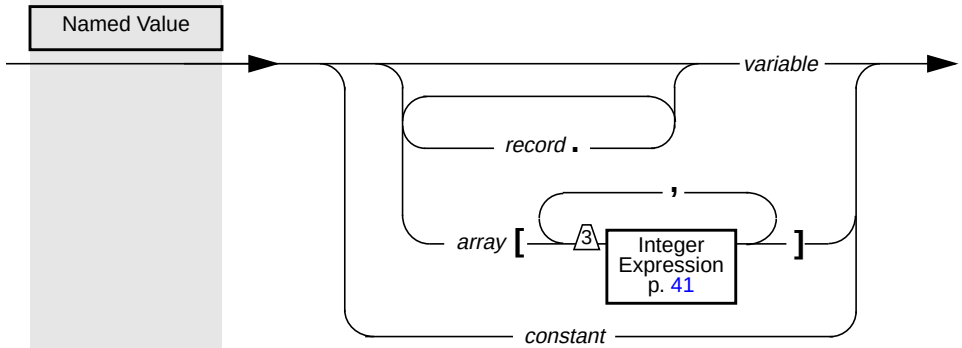


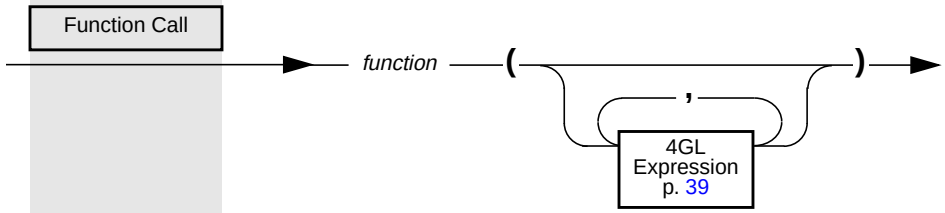
Array Data Type

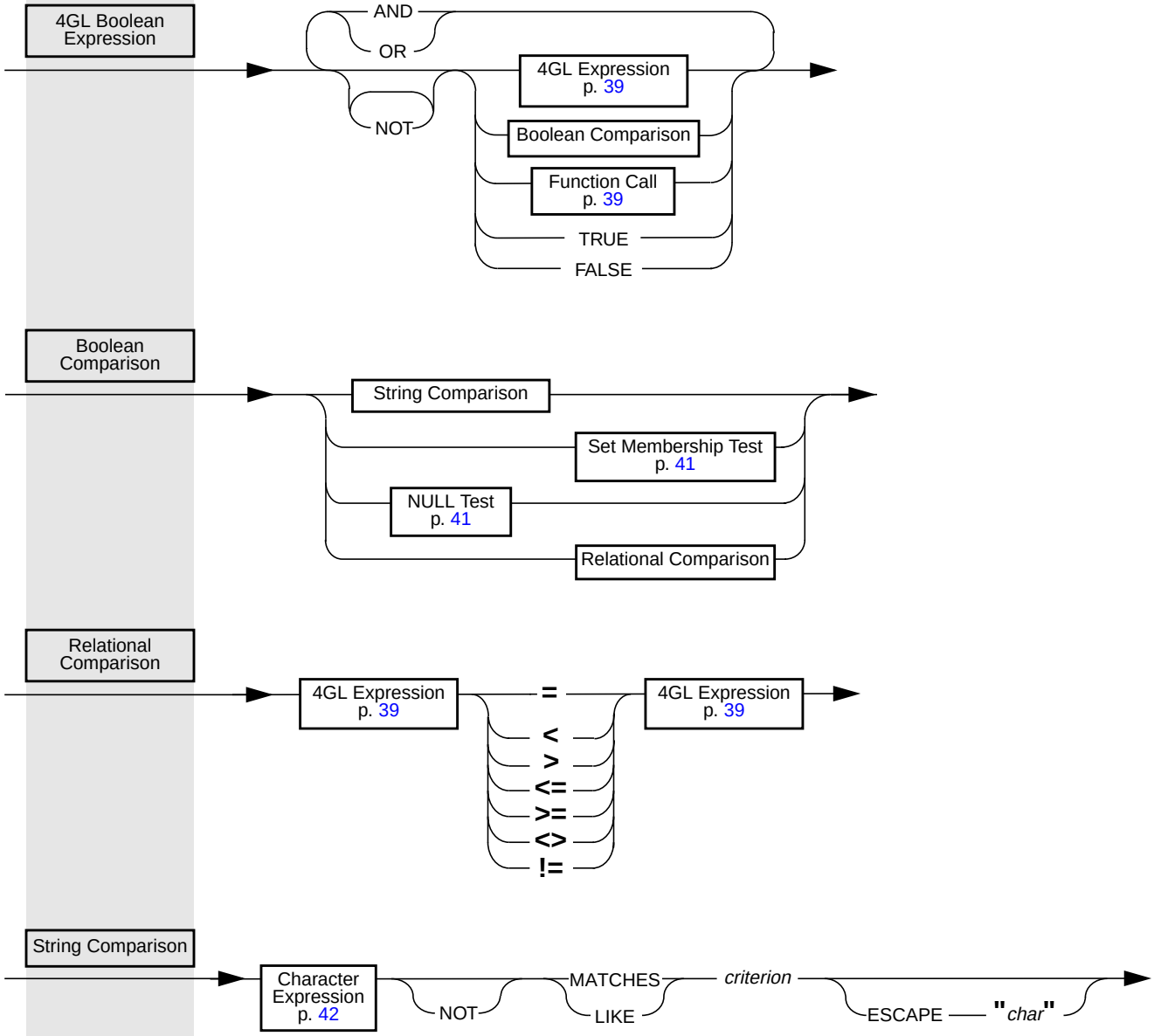



RECORD Data Type
RECORD
,
member
Data Type Declaration
p. 37
END RECORD
LIKE
table .*
Table Qualifier
p. 46


4GL Expression
Time Expression
p. 43
Character Expression
p. 42
Number Expression
p. 42.
Integer Expression
p. 41
4GL Boolean
Expression
p. 40
(4GL Expression)


Named Value
variable
record .
,
array [3 Integer
Expression
p. 41]
constant


Function Call
function ( )
,
4GL
Expression
p. 39

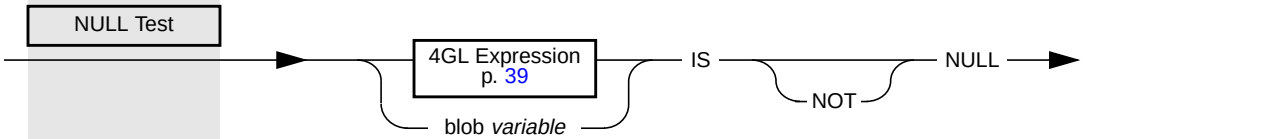

4GL Boolean Expression
AND
OR
NOT
4GL Expression p. 39
Boolean Comparison
Function Call p. 39
TRUE
FALSE
Boolean Comparison
String Comparison
Set Membership Test p. 41
NULL Test p. 41
Relational Comparison
Relational Comparison
4GL Expression p. 39
=
<
>
<=
>=
<>
!=
4GL Expression p. 39
String Comparison
Character Expression p. 42
NOT
MATCHES
LIKE
criterion
ESCAPE
"char"

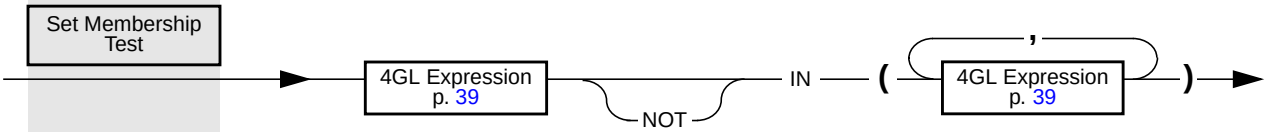
MATCHES

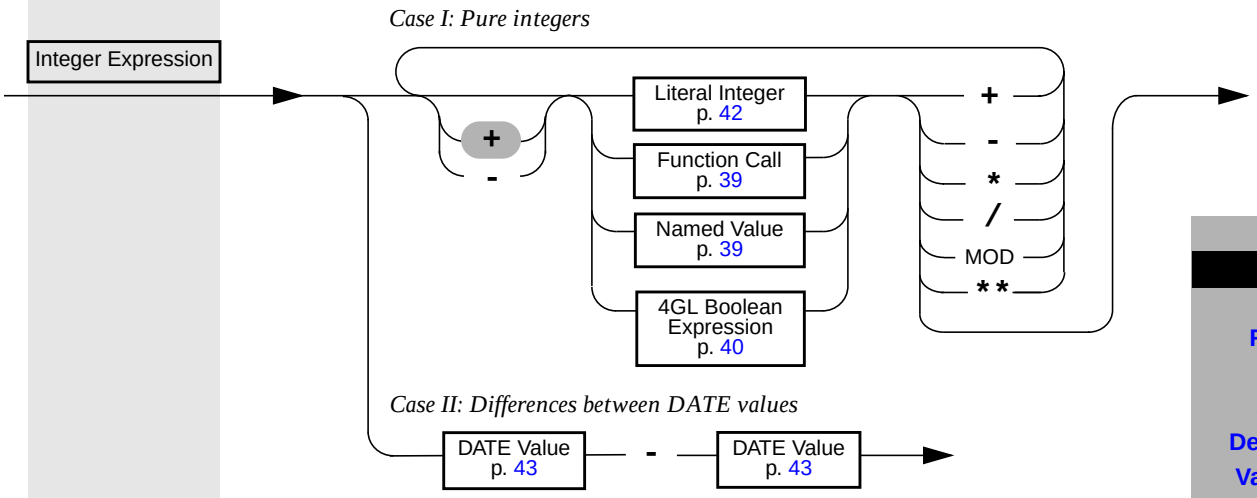
Wildcard	Effect
*	Matches a string of zero or more characters.
?	Matches any single character.
[ ]	Matches any of the enclosed characters.
-	Between characters in brackets means a range in the ASCII collating sequence.
^	As the first character in the brackets, matches any character that is not listed.
\	Treats the next character as a literal.

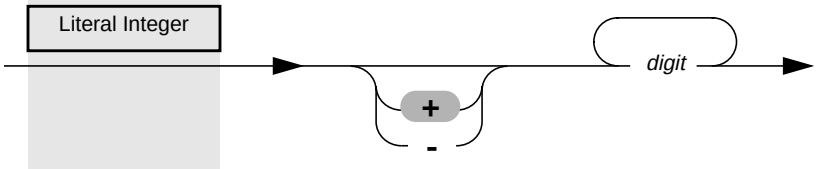
LIKE

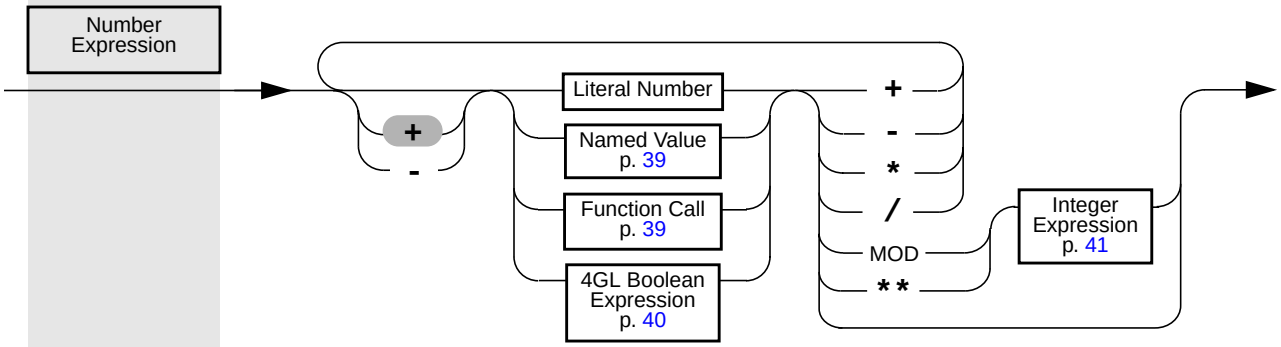
Wildcard	Effect
%	Matches a string of zero or more characters.
_	Matches any single character.
\	Treats the next character as a literal.

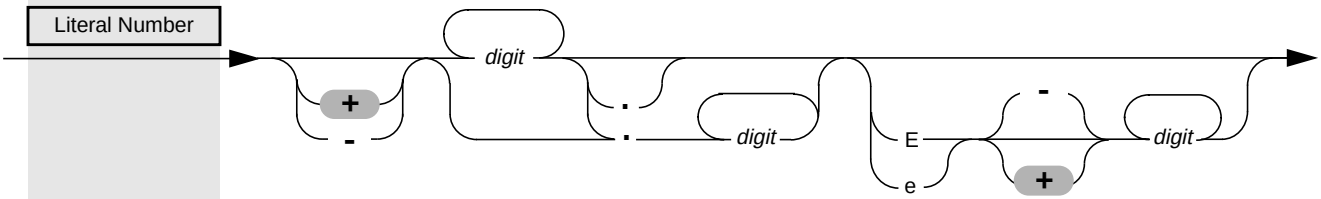


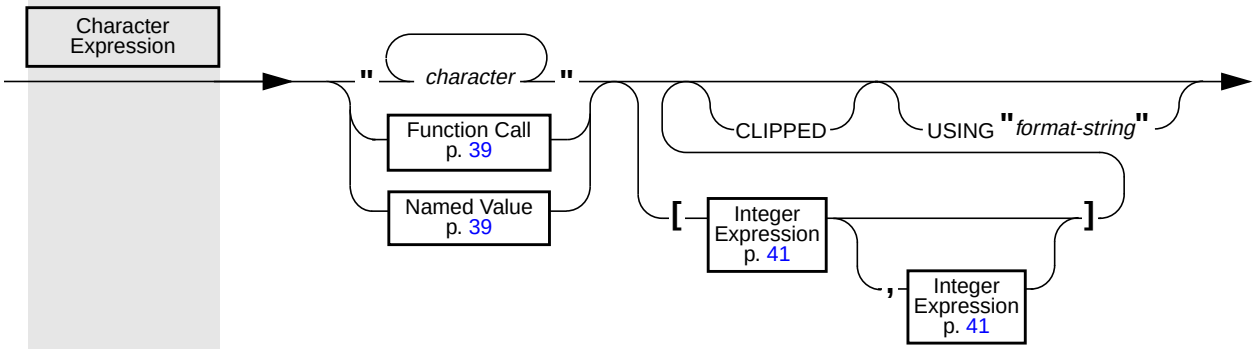


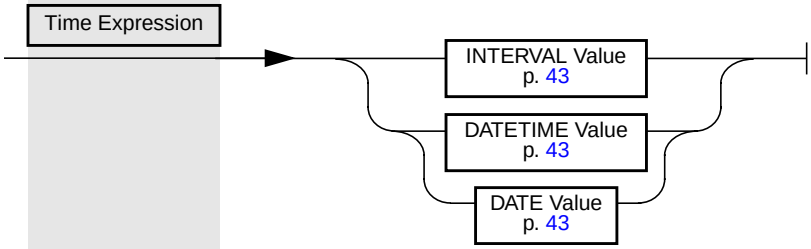


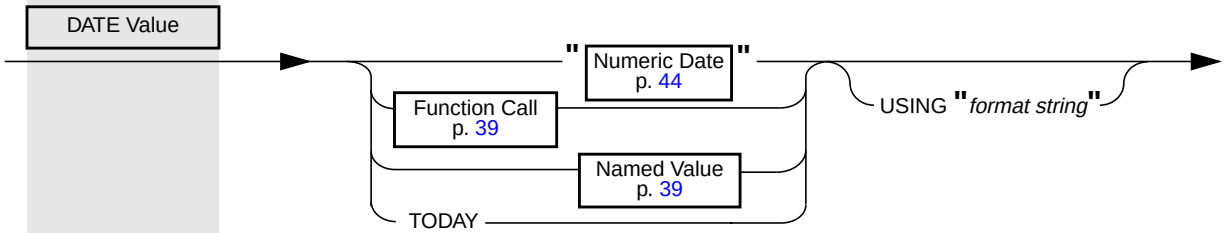

Literal Integer
+
-
digit

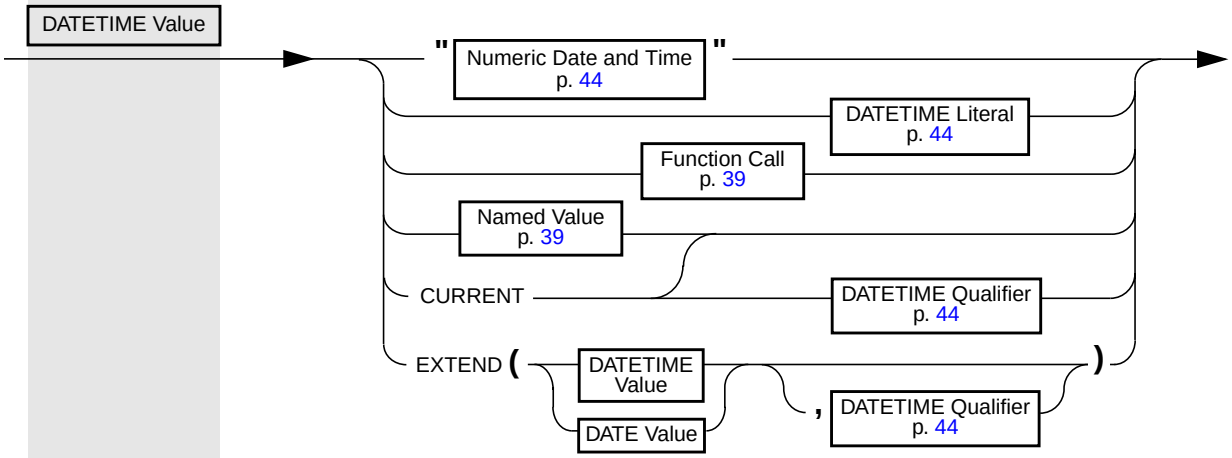

Number Expression
+
-
Literal Number
Named Value p. 39
Function Call p. 39
4GL Boolean Expression p. 40
+
-
*
/
MOD
**
Integer Expression p. 41

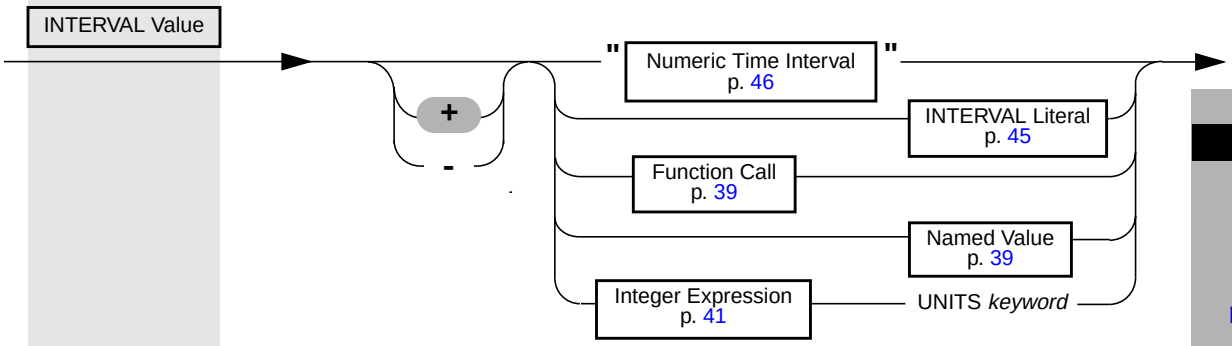

Literal Number
+
-
digit
.
.
digit
E
e
-
+
digit


Character Expression
"
character
"
Function Call p. 39
Named Value p. 39
CLIPPED
USING "format-string"
[
Integer Expression p. 41
,
Integer Expression p. 41
]


Time Expression
INTERVAL Value
p. 43
DATETIME Value
p. 43
DATE Value
p. 43

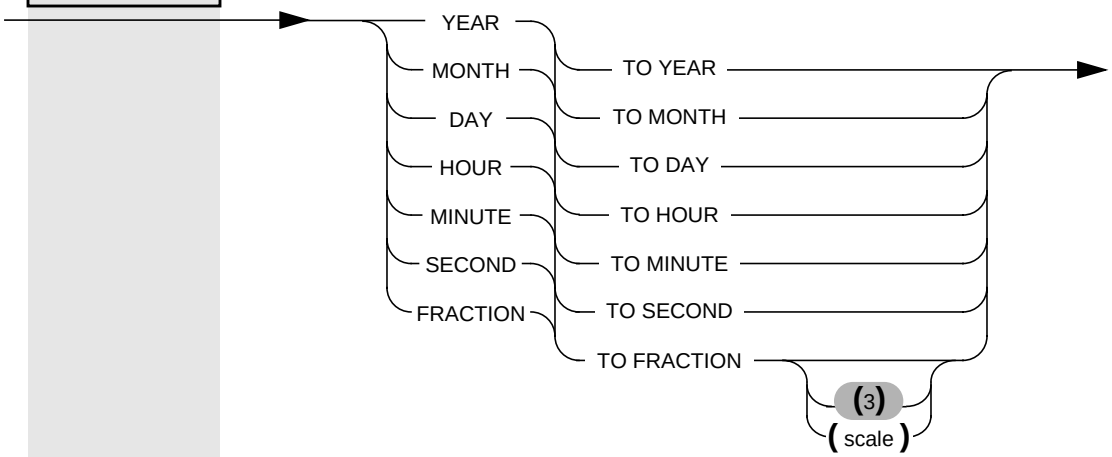

DATE Value
"
Numeric Date
p. 44
"
Function Call
p. 39
Named Value
p. 39
TODAY
USING "format string"


DATETIME Value
"
Numeric Date and Time
p. 44
"
DATETIME Literal
p. 44
Function Call
p. 39
Named Value
p. 39
CURRENT
DATETIME Qualifier
p. 44
EXTEND (
DATETIME Value
DATE Value
,
DATETIME Qualifier
p. 44
)

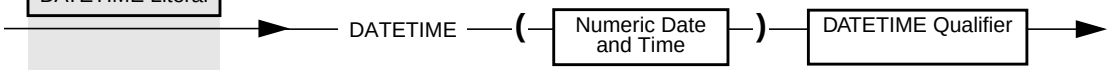

INTERVAL Value
+
-
"
Numeric Time Interval
p. 46
"
INTERVAL Literal
p. 45
Function Call
p. 39
Named Value
p. 39
Integer Expression
p. 41
UNITS keyword

Numeric Date

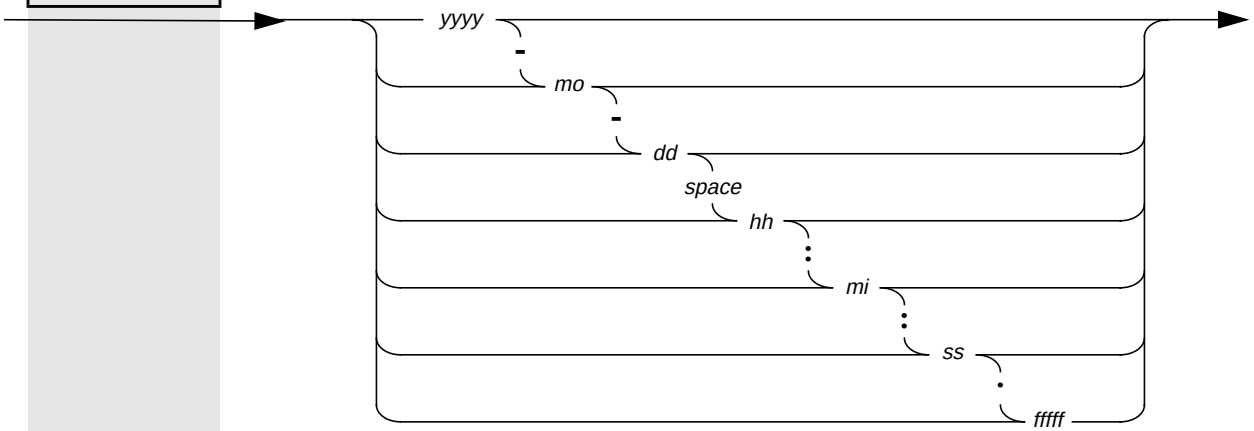
DATETIME Qualifier

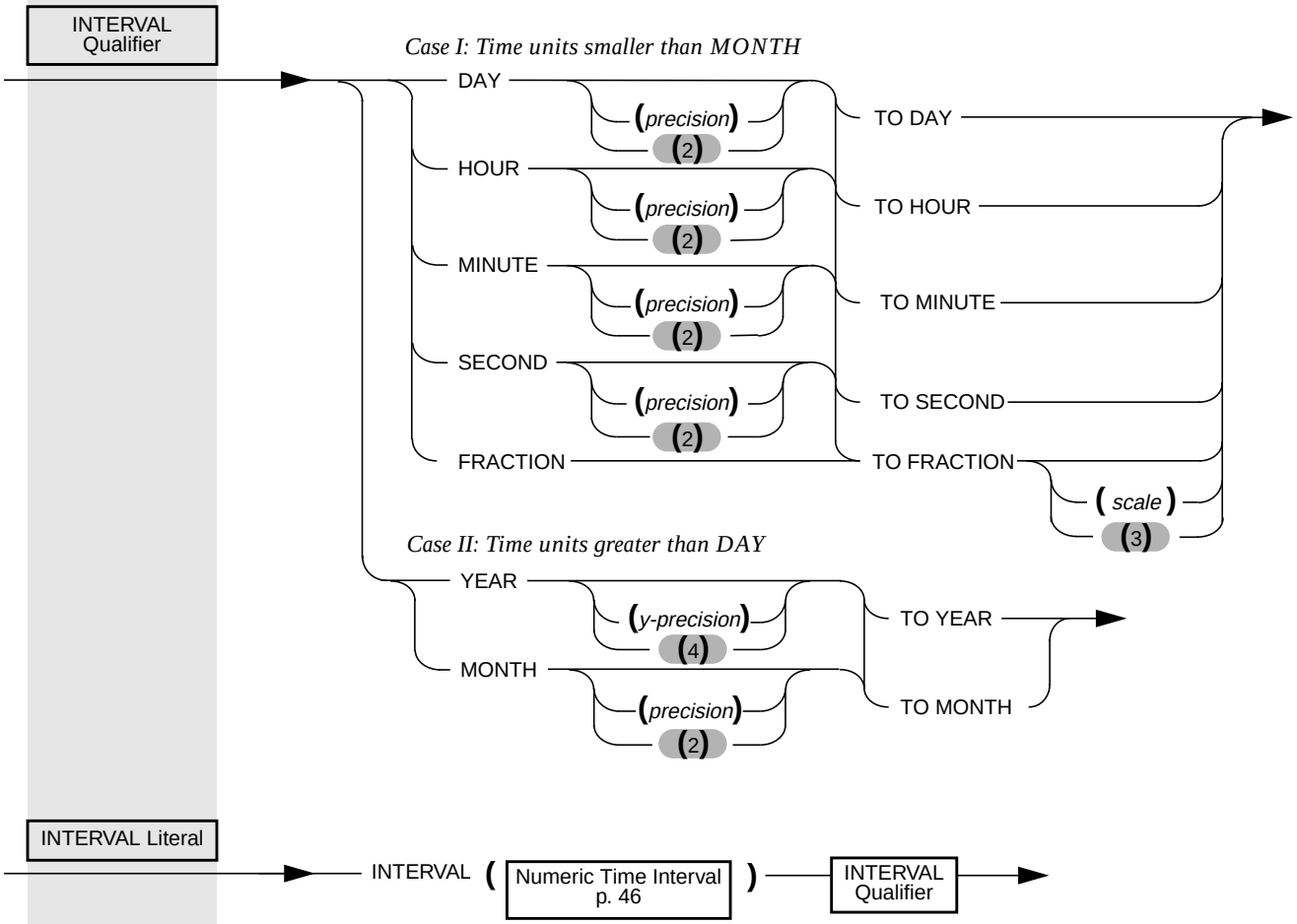


DATETIME Literal



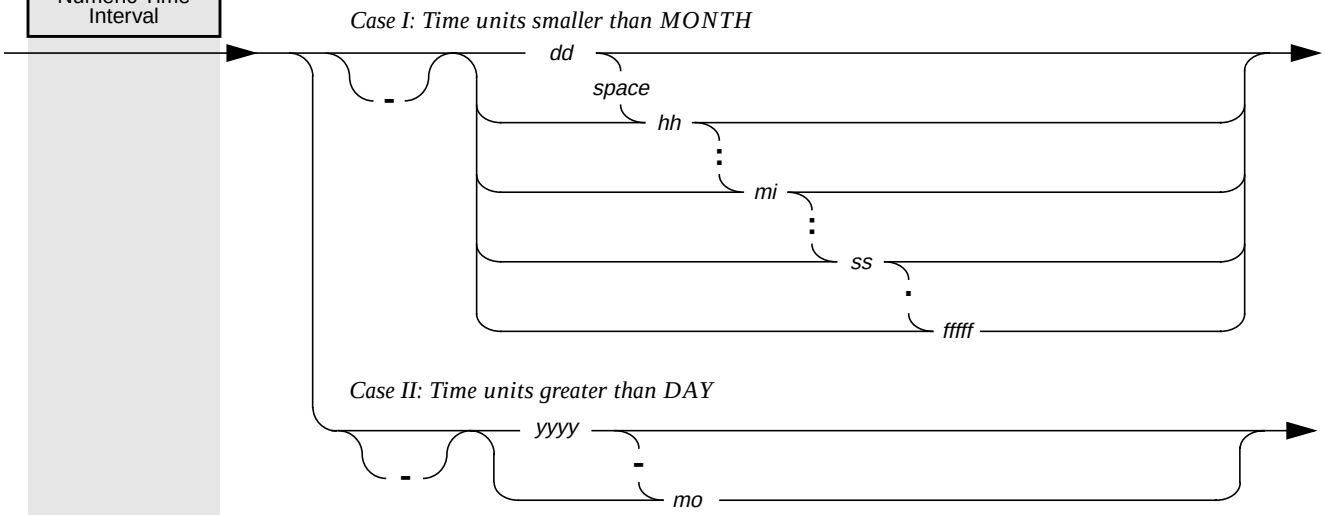
Numeric Date and Time




INTERVAL Qualifier
Case I: Time units smaller than MONTH
DAY
HOUR
MINUTE
SECOND
FRACTION
(precision)
(2)
TO DAY
TO HOUR
TO MINUTE
TO SECOND
TO FRACTION
(scale)
(3)
Case II: Time units greater than DAY
YEAR
MONTH
(y-precision)
(4)
(precision)
(2)
TO YEAR
TO MONTH

INTERVAL Literal
INTERVAL
(
Numeric Time Interval p. 46
)
INTERVAL Qualifier

Numeric Time Interval



Field Clause

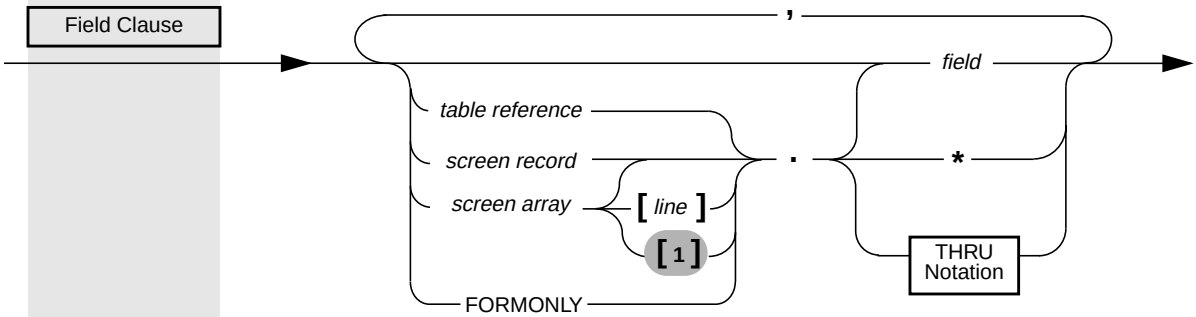
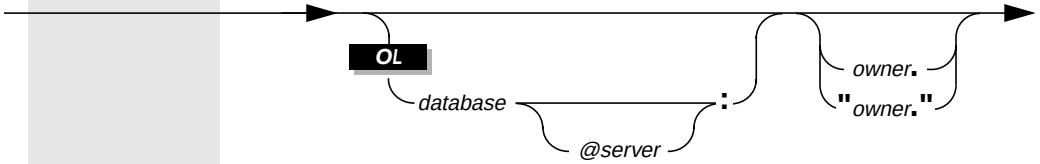
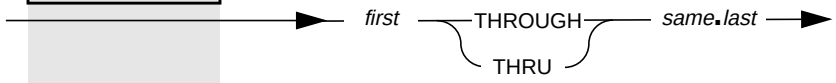


Table Qualifier

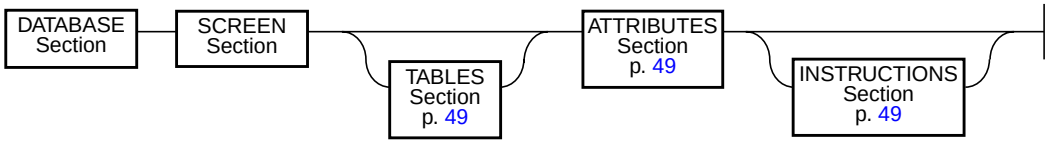


THRU Notation

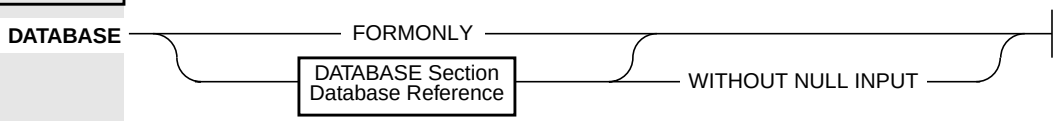


4GL Forms

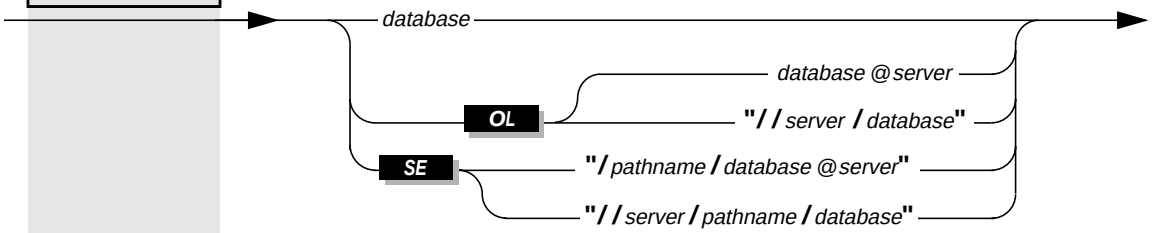
Form Specification Syntax



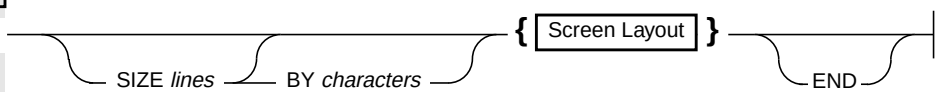
DATABASE Section



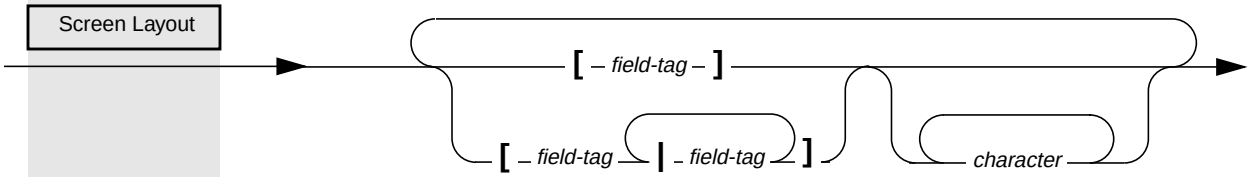
DATABASE Section Database Reference



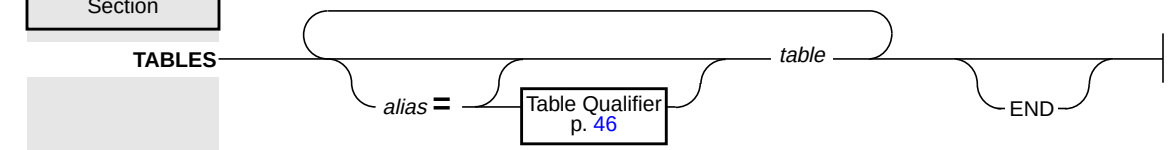
SCREEN Section



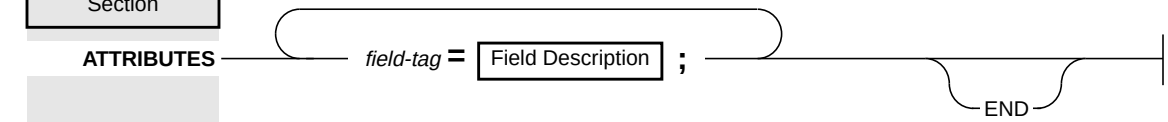
Screen Layout



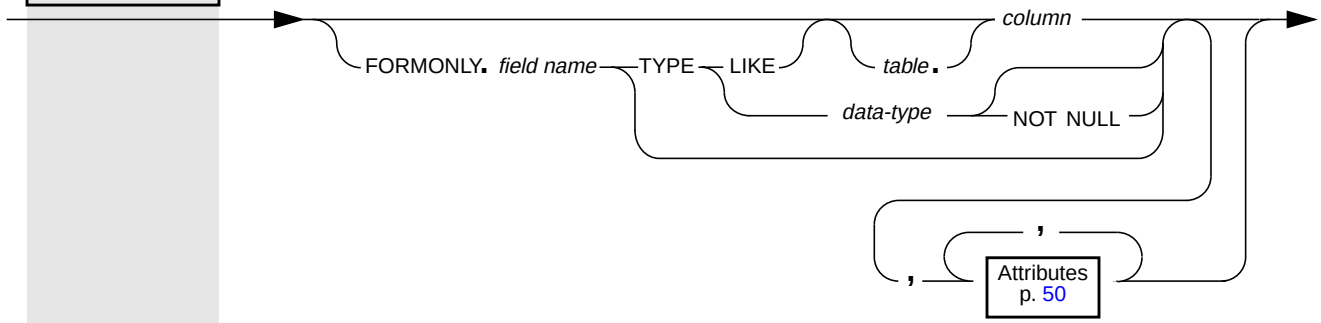
TABLES Section



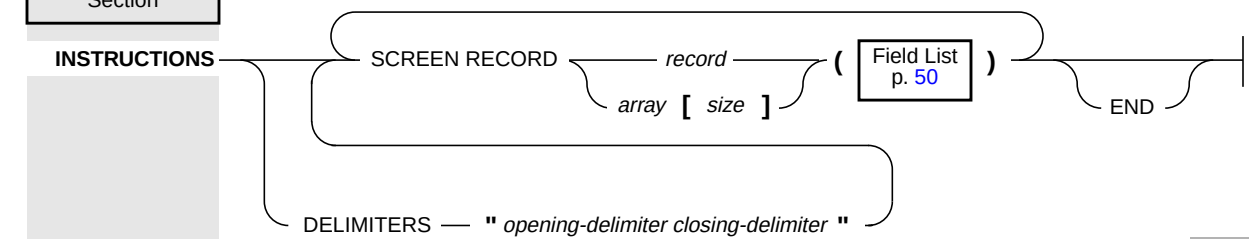
ATTRIBUTES Section

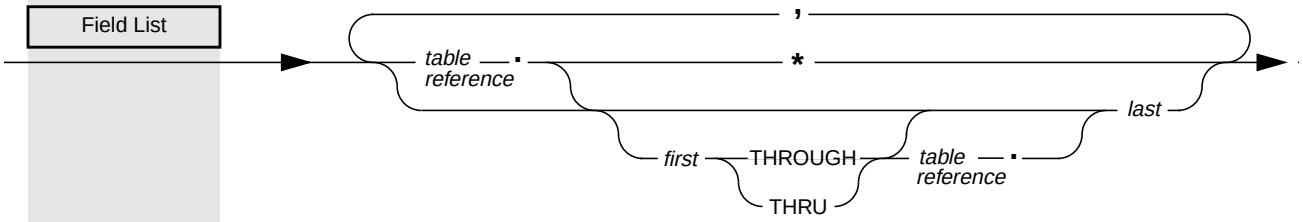


Field Description

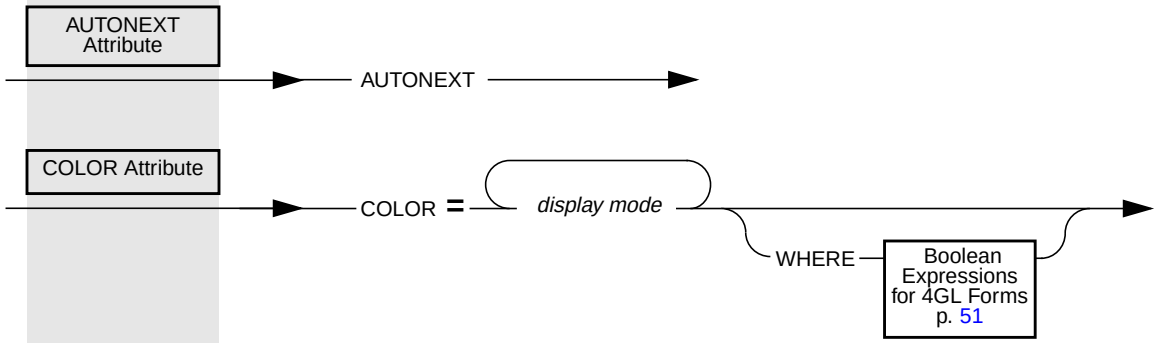


INSTRUCTIONS Section



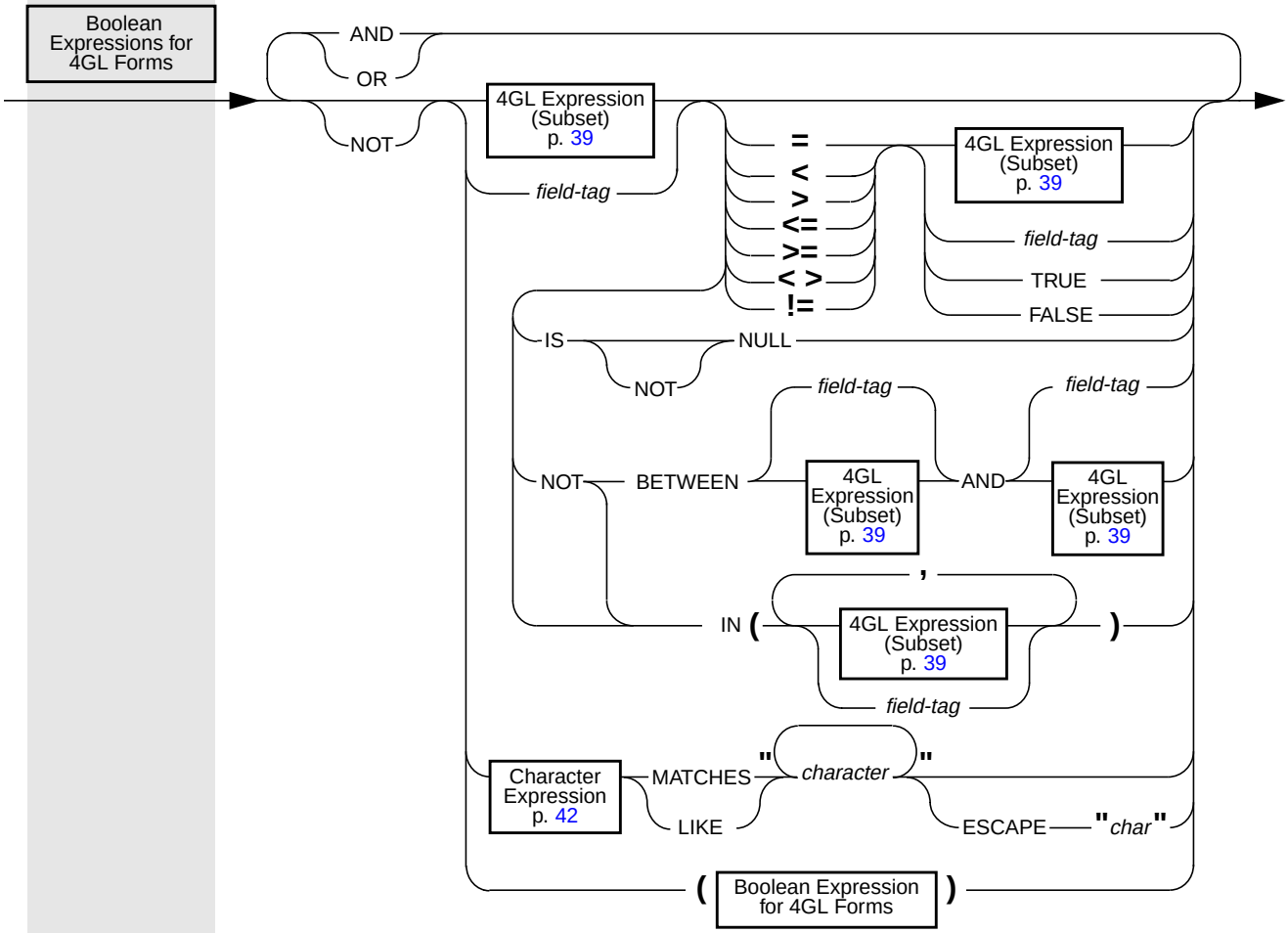


Attributes



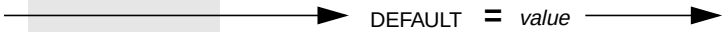
Values for *display mode* consists of zero or one *color* and zero or more *intensities*:

Color Keywords		Intensity Keywords
BLACK	MAGENTA	REVERSE
BLUE	RED	LEFT
CYAN	WHITE	BLINK
GREEN	YELLOW	UNDERLINE



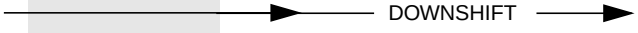
COMMENTS Attribute

DEFAULT Attribute



DISPLAY LIKE Attribute

DOWNSHIFT Attribute



FORMAT Attribute

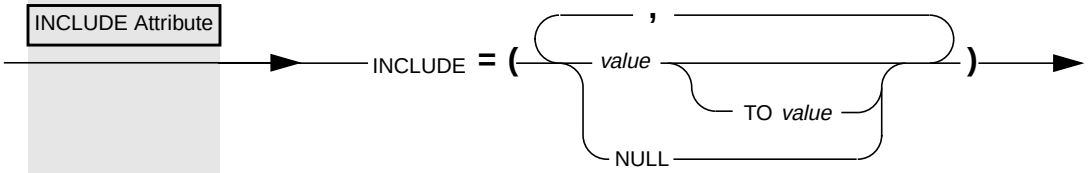
For DATE data types, *format-string* consists of:

Special Characters	Meaning
mm	2-digit representation of the month
mmm	3-letter abbreviation of the month
dd	2-digit representation of the day of the month
ddd	3-letter abbreviation of the day of the week
yy	2-digit representation of the year, discarding the leading digits
yyyy	4-digit representation of the year

All other characters are literals.

For DECIMAL, SMALLFLOAT, or FLOAT data types, *format-string* consists of pound signs (#) to represent digits and a decimal point. If you are using NLS, the period is a placeholder for the decimal separator and the comma is a placeholder for the thousands separator.

INCLUDE Attribute



INVISIBLE Attribute

NOENTRY Attribute

PICTURE Attribute

A *format-string* can include these three special symbols:

Symbol	Meaning
A	Any letter
#	Any digit
X	Any character

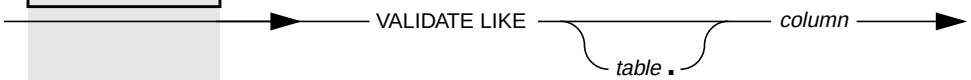
PROGRAM Attribute

REQUIRED Attribute

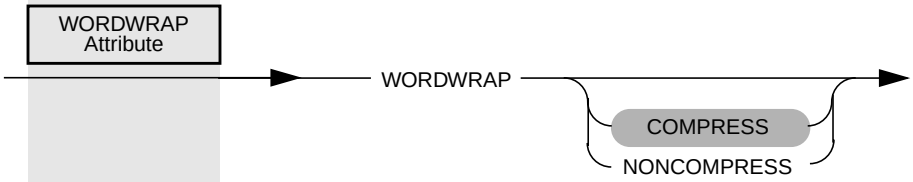
REVERSE Attribute

UPSHIFT Attribute

VALIDATE LIKE Attribute

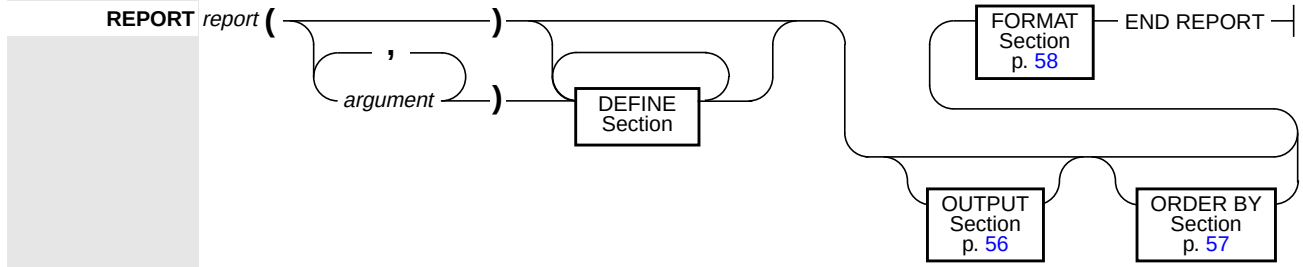


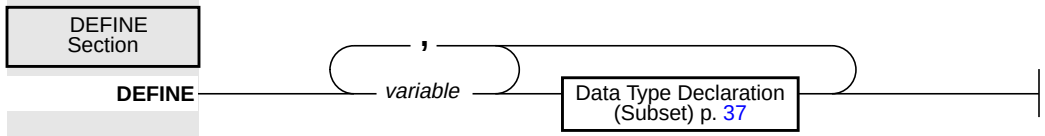
VERIFY Attribute

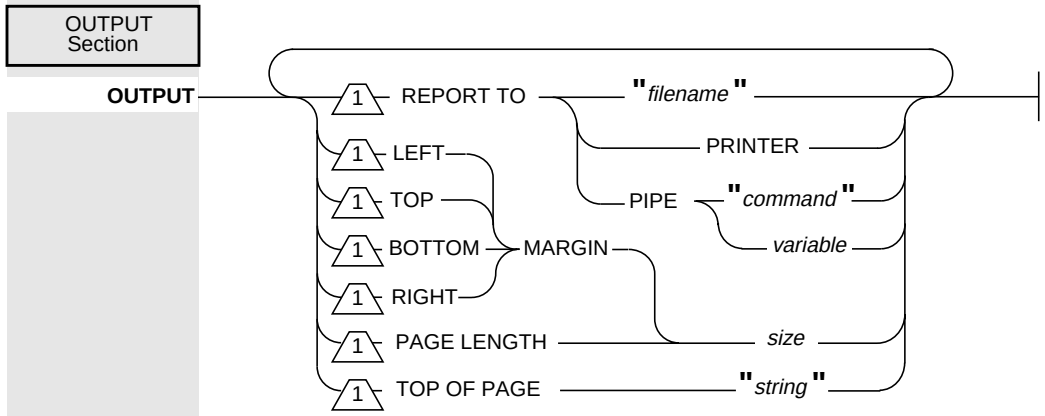

WORDWRAP Attribute
WORDWRAP
COMPRESS
NONCOMPRESS

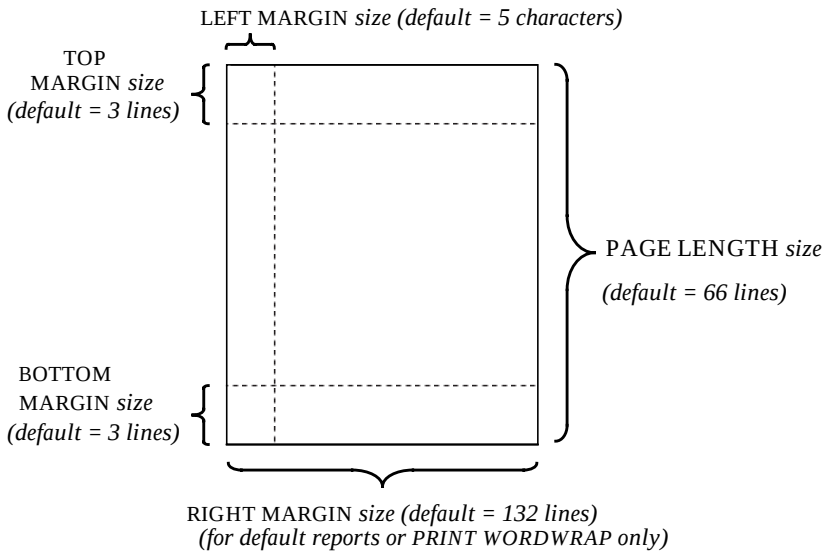
Reports

Report Specification Syntax

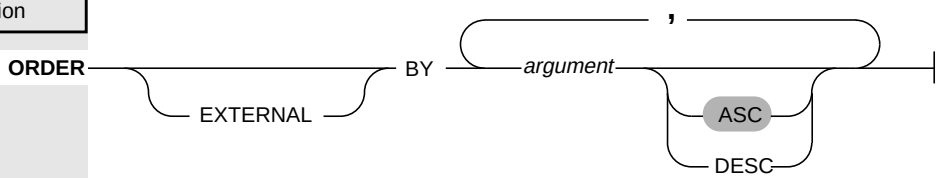






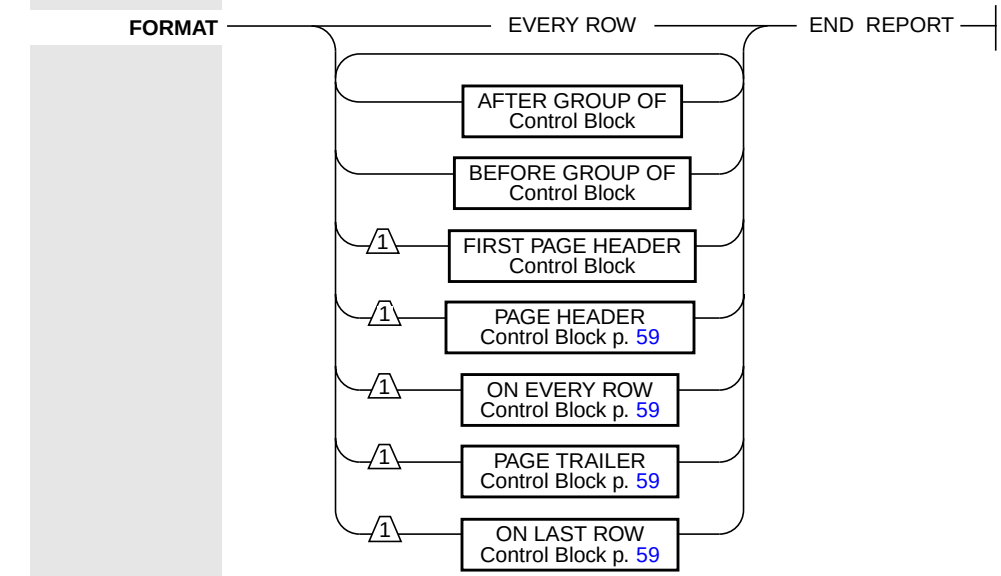


ORDER BY Section



Basics
4GL
Forms
Reports
SQL
SQLCA
Debugger
Variables
Keys

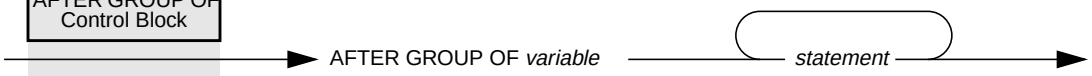
FORMAT Section



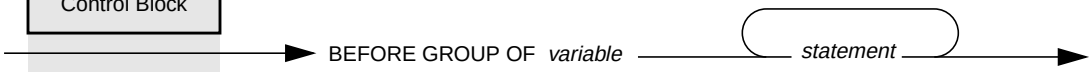
Following is the execution sequence of report control blocks.

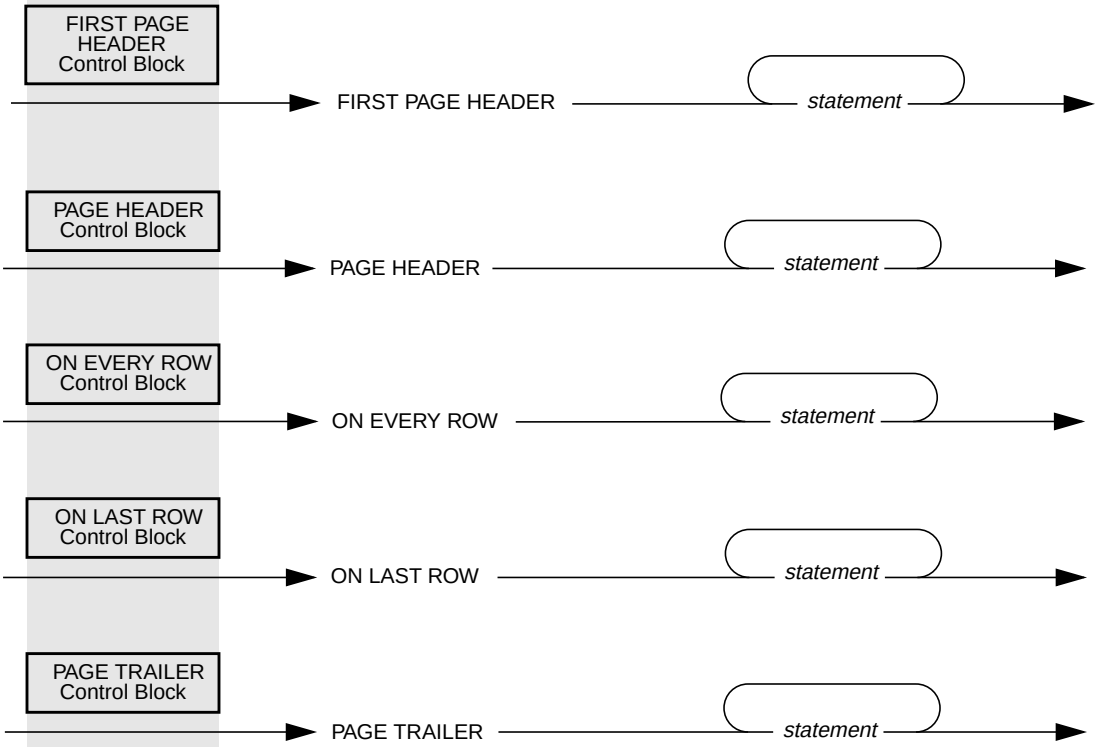
```
BEFORE GROUP OF a          {1}
    BEFORE GROUP OF b      {2}
        BEFORE GROUP OF c  {3}
            ON EVERY ROW   {4}
        AFTER GROUP OF c   {3}
    AFTER GROUP OF b       {2}
AFTER GROUP OF a           {1}
```

AFTER GROUP OF Control Block

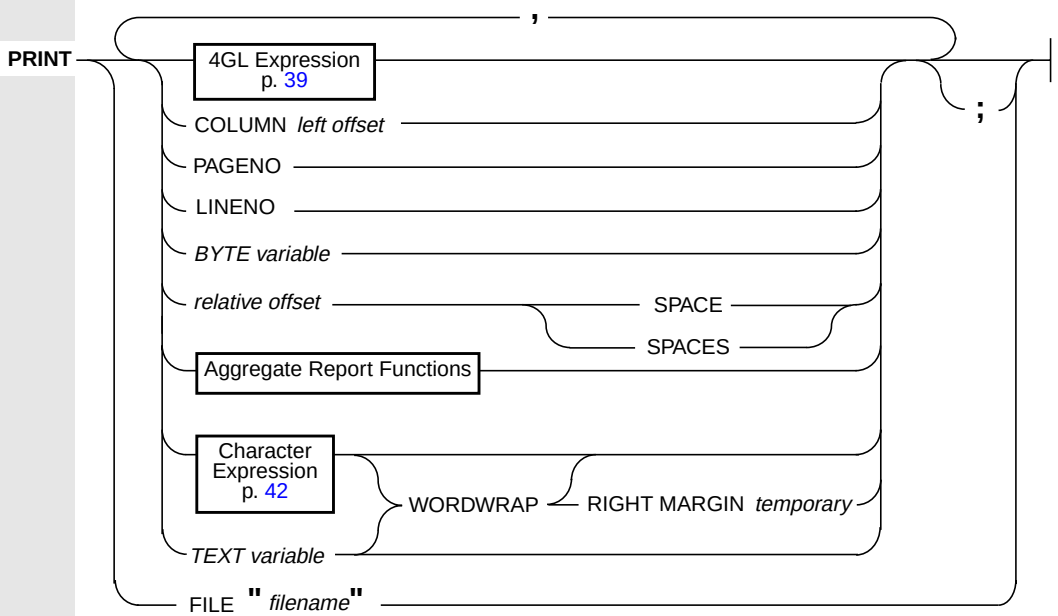


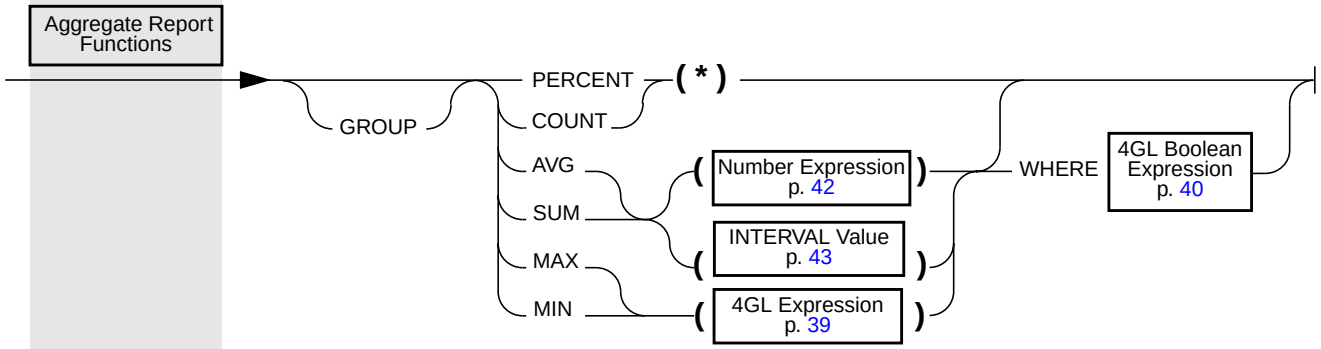
BEFORE GROUP OF Control Block

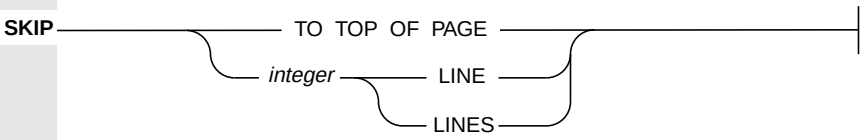




Report Execution Statements


PRINT
,
4GL Expression
p. 39
COLUMN left offset
PAGENO
LINENO
BYTE variable
relative offset
SPACE
SPACES
Aggregate Report Functions
Character Expression
p. 42
WORDWRAP
RIGHT MARGIN temporary
TEXT variable
FILE " filename"
;

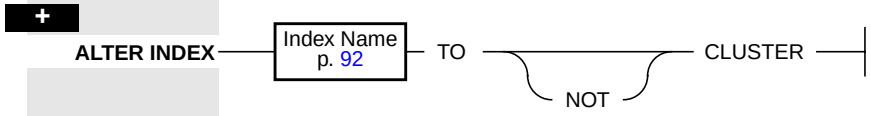

Aggregate Report Functions
PERCENT
(*)
GROUP
COUNT
AVG
SUM
MAX
MIN
(Number Expression p. 42)
(INTERVAL Value p. 43)
(4GL Expression p. 39)
WHERE
4GL Boolean Expression
p. 40

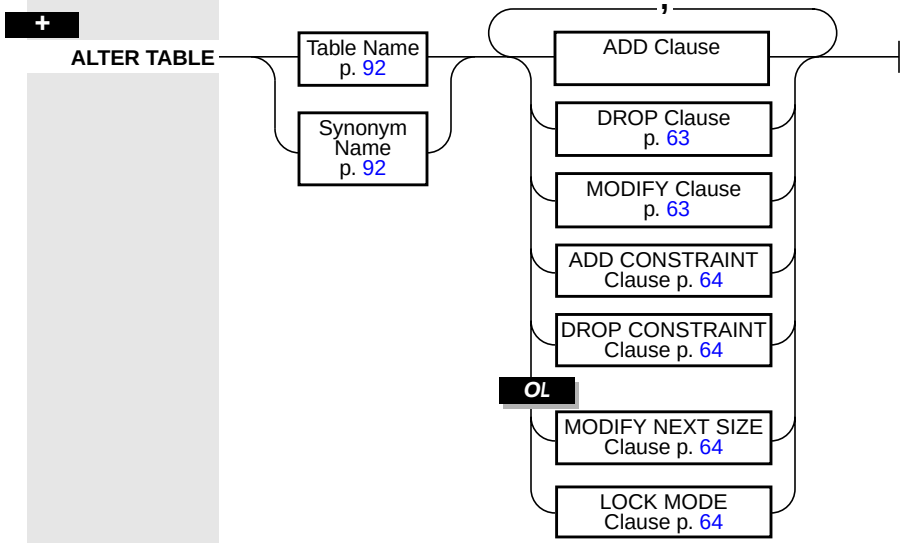

SKIP
TO TOP OF PAGE
integer
LINE
LINES

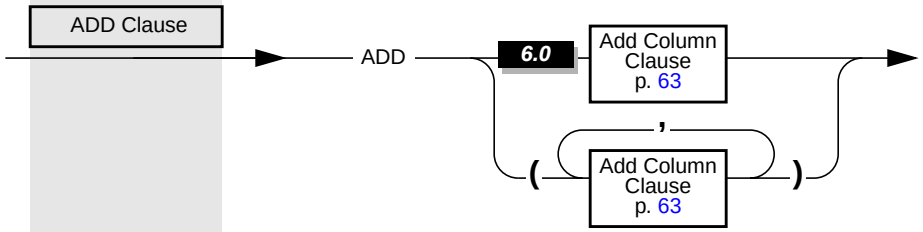
SQL Statements

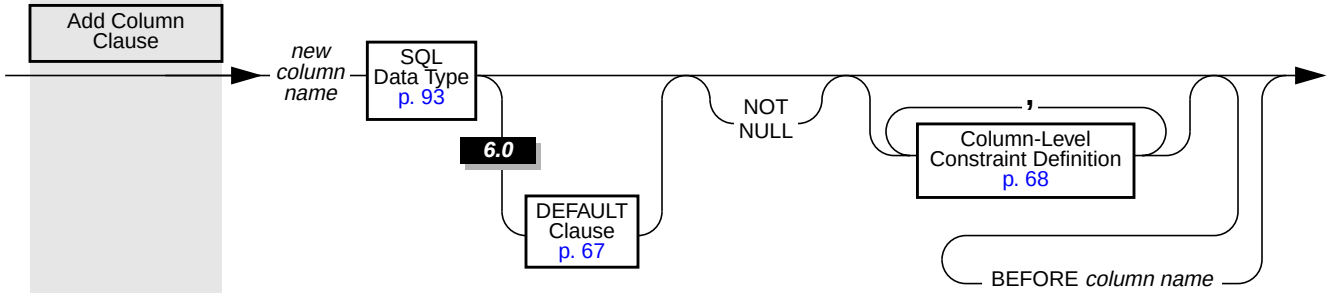
The **4GL** source compiler does not recognize SQL statements identified in this Guide by the 6.0 icon nor SQL statements containing a clause identified by the 6.0 icon. To compile **4GL** source code containing such statements, you must do the following:

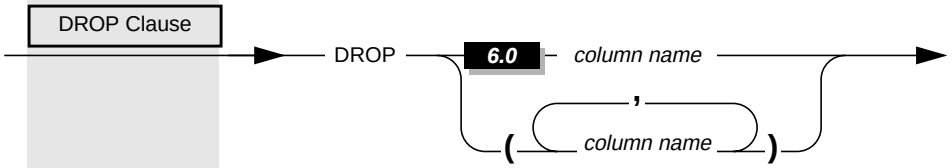
1. Store the 6.0 SQL statement as a character string.
2. Set up the statement for execution by means of the PREPARE statement (see p. 79).
3. Process the statement by means of the EXECUTE statement (see p. 75).

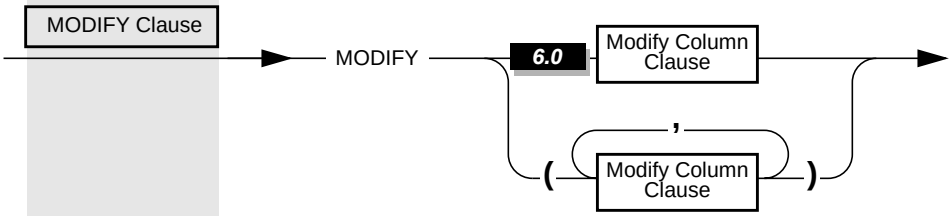


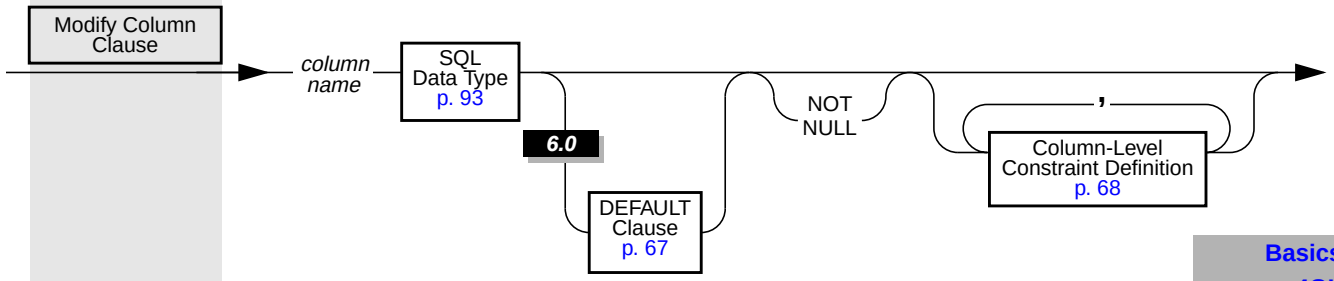




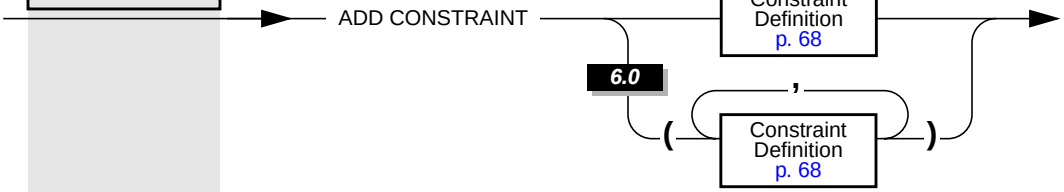

Add Column Clause
new column name
SQL Data Type p. 93
6.0
DEFAULT Clause p. 67
NOT NULL
,
Column-Level Constraint Definition p. 68
BEFORE column name


DROP Clause
DROP
6.0
column name
,
(
column name
)

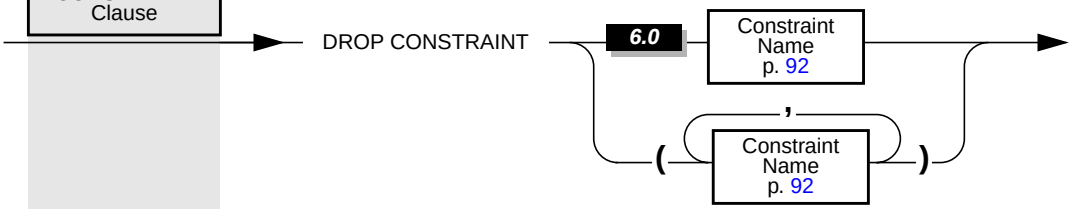

MODIFY Clause
MODIFY
6.0
Modify Column Clause
,
(
Modify Column Clause
)


Modify Column Clause
column name
SQL Data Type p. 93
6.0
DEFAULT Clause p. 67
NOT NULL
,
Column-Level Constraint Definition p. 68

ADD CONSTRAINT Clause



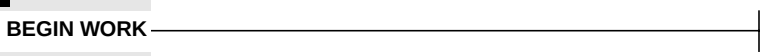
DROP CONSTRAINT Clause

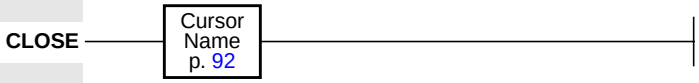


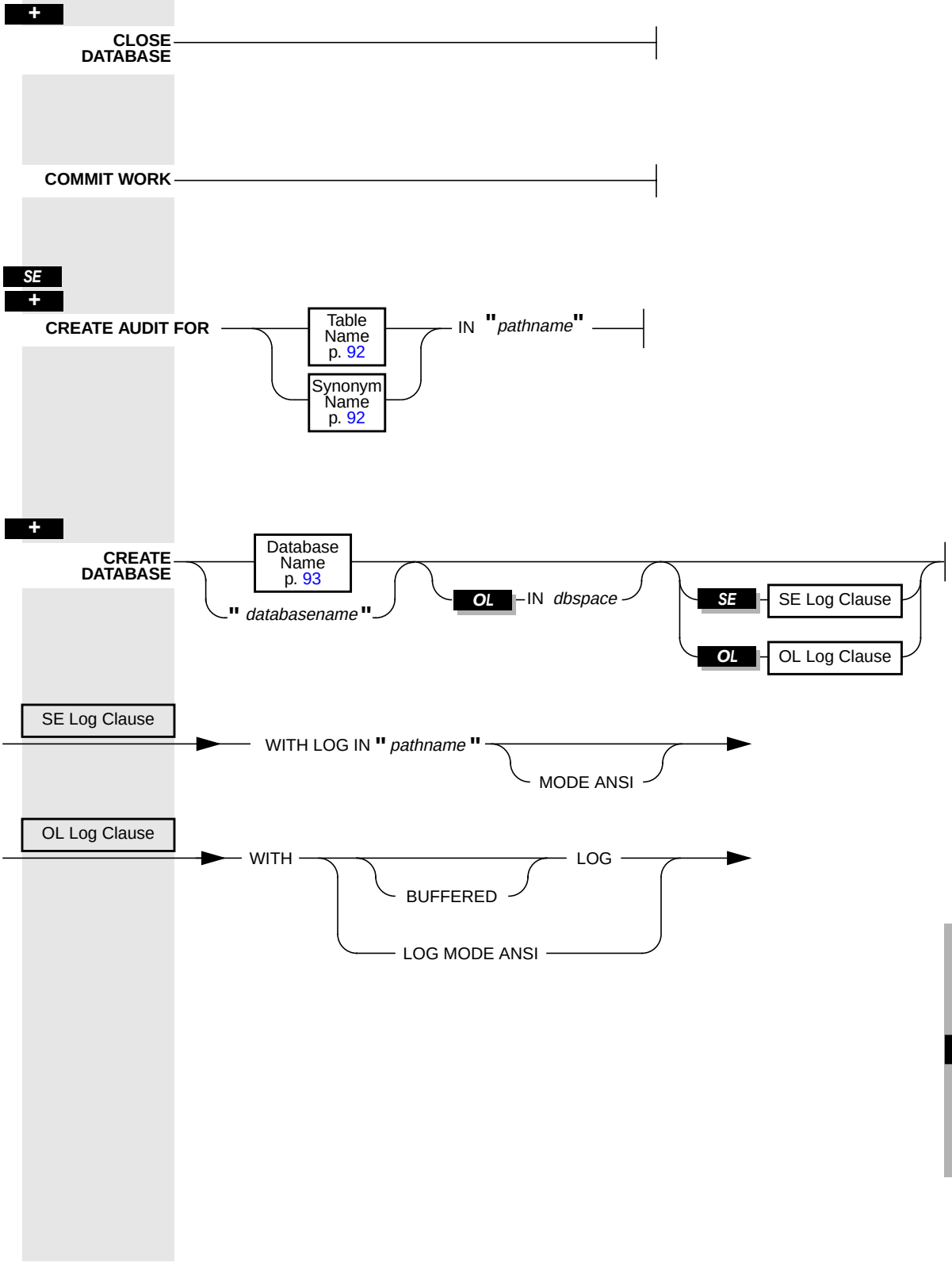
MODIFY NEXT SIZE Clause

LOCK MODE Clause

+

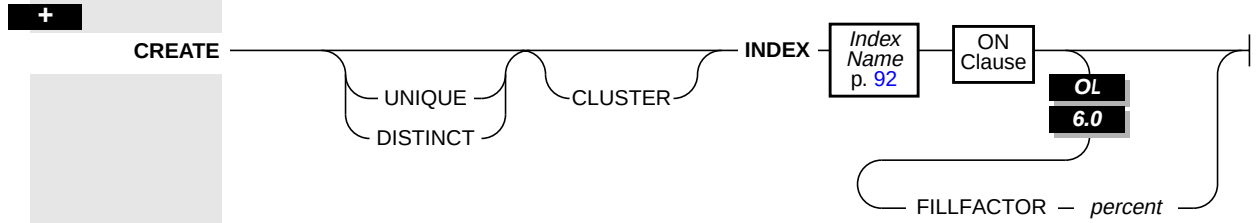


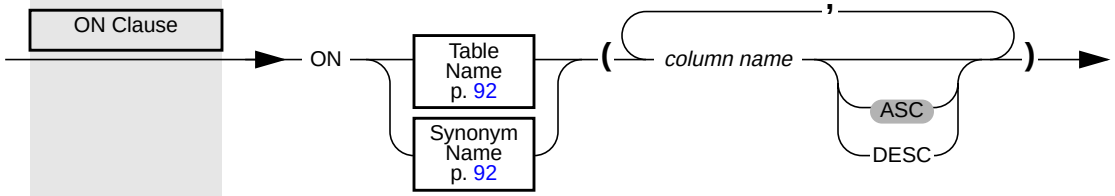




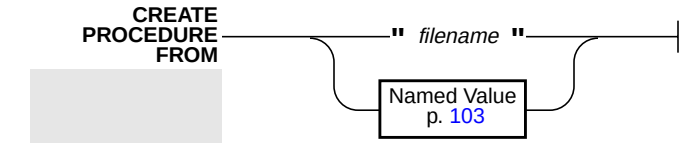
Basics
4GL
Forms
Reports
SQL
SQLCA
Debugger
Variables
Keys

+

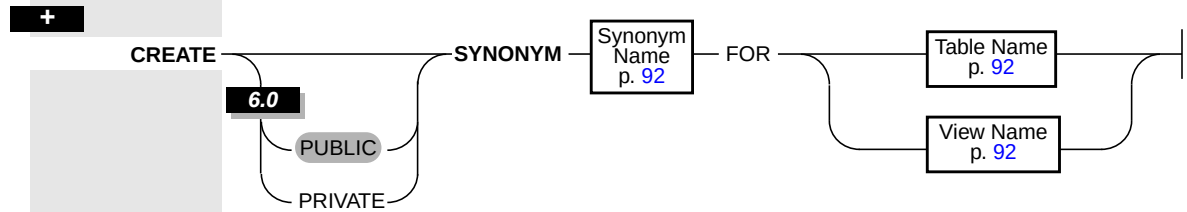

CREATE
UNIQUE
DISTINCT
CLUSTER
INDEX
Index Name p. 92
ON Clause
OL
6.0
FILLFACTOR — percent

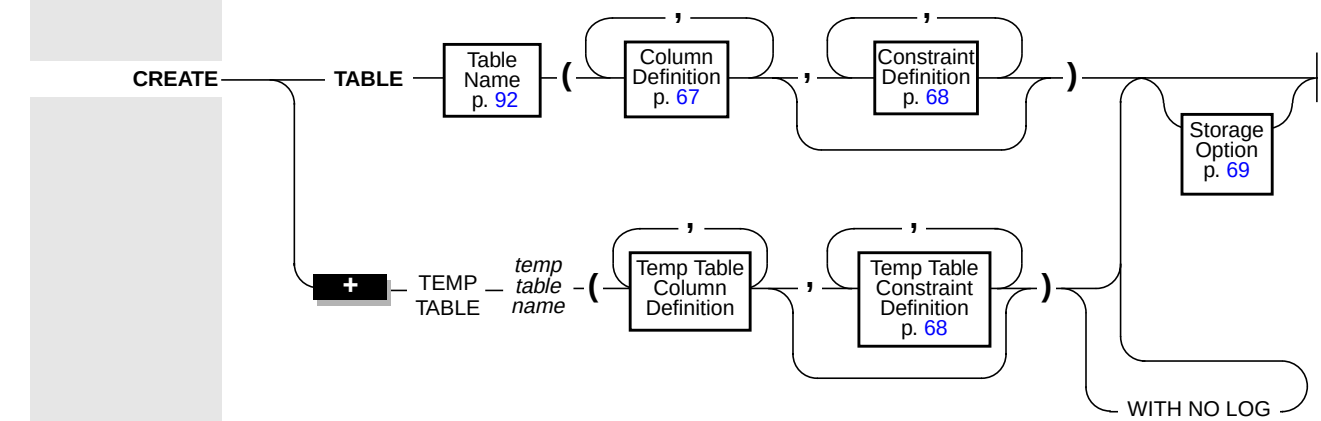

ON Clause
ON
Table Name p. 92
Synonym Name p. 92
(
column name
,
ASC
DESC
)

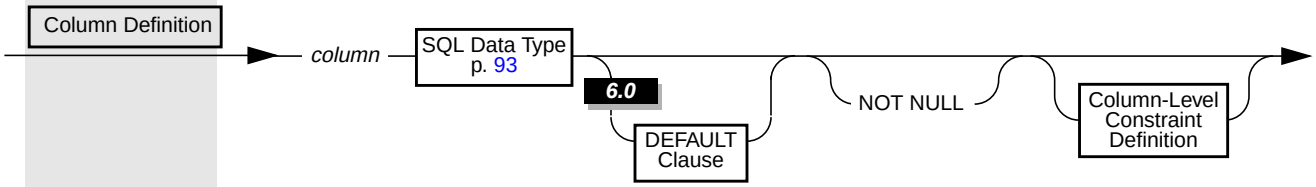
6.0
+

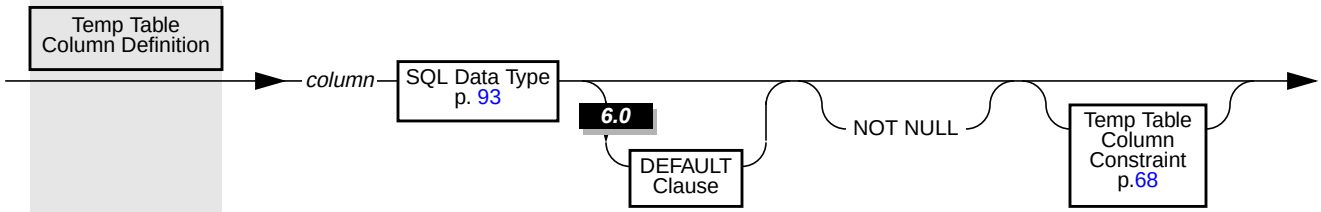

CREATE PROCEDURE FROM
" filename "
Named Value p. 103

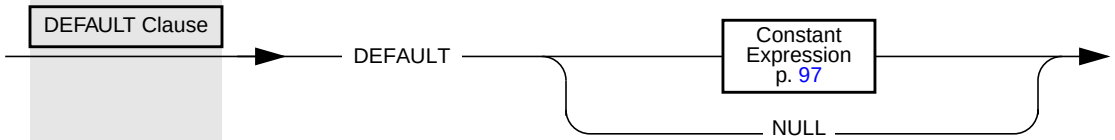
+


CREATE
6.0
PUBLIC
PRIVATE
SYNONYM
Synonym Name p. 92
FOR
Table Name p. 92
View Name p. 92

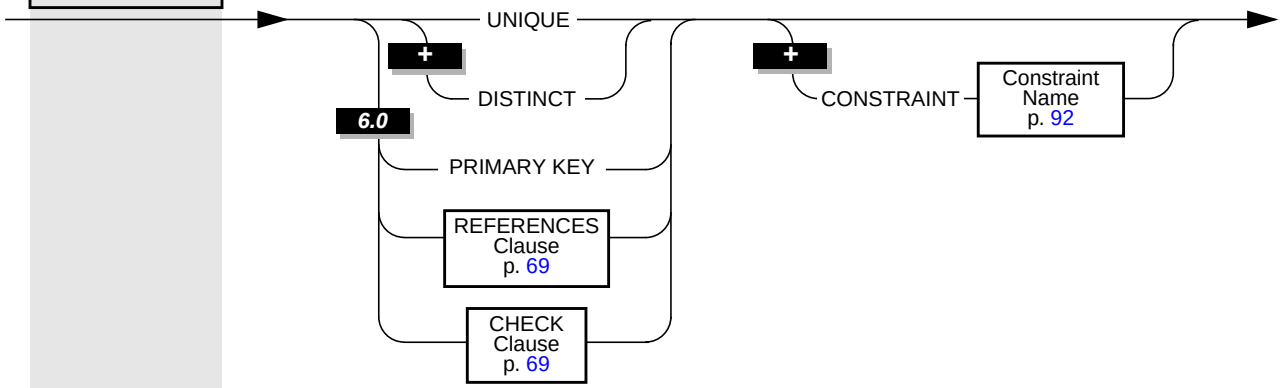

CREATE
TABLE
Table Name p. 92
(
Column Definition p. 67
,
,
Constraint Definition p. 68
)
Storage Option p. 69
+
TEMP TABLE
temp table name
(
Temp Table Column Definition
,
,
Temp Table Constraint Definition p. 68
)
WITH NO LOG


Column Definition
column
SQL Data Type p. 93
6.0
DEFAULT Clause
NOT NULL
Column-Level Constraint Definition

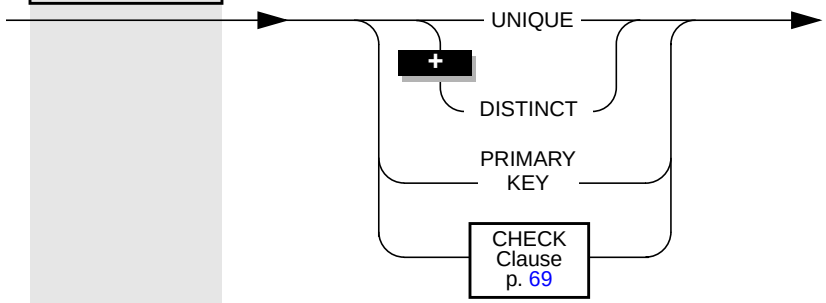

Temp Table Column Definition
column
SQL Data Type p. 93
6.0
DEFAULT Clause
NOT NULL
Temp Table Column Constraint p.68


DEFAULT Clause
DEFAULT
Constant Expression p. 97
NULL

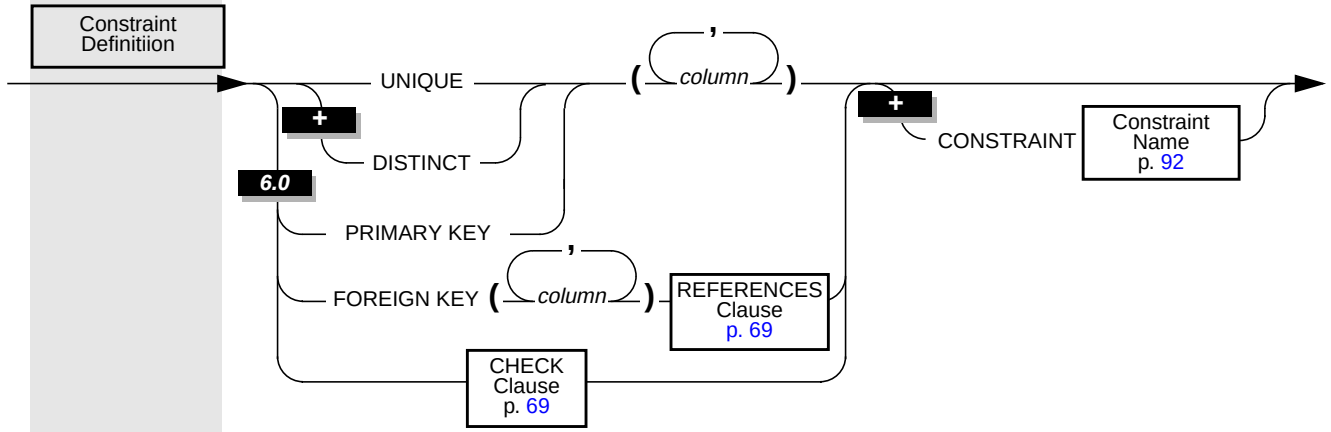
Column-Level Constraint Definition

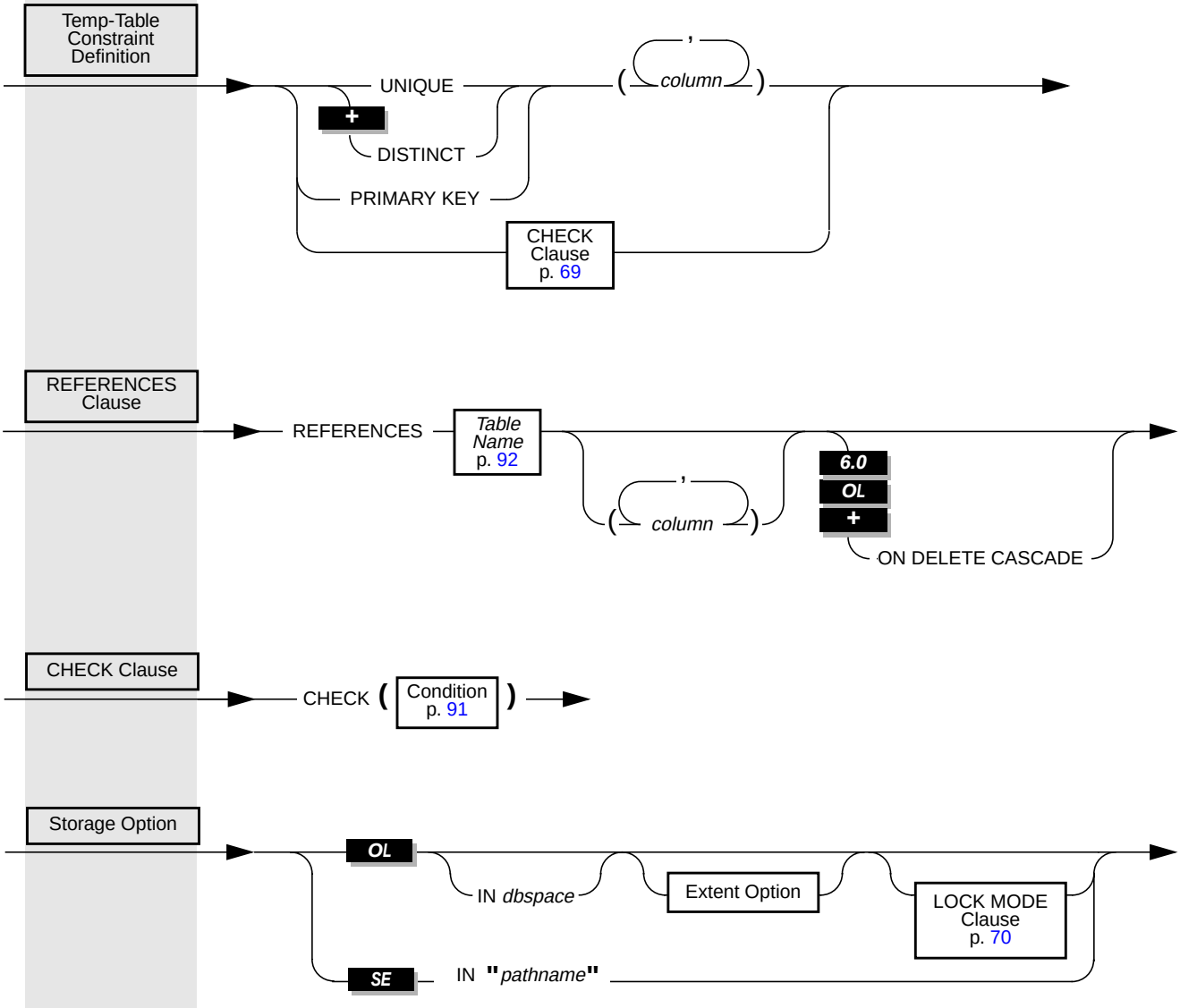


Temp-Table Column Constraint

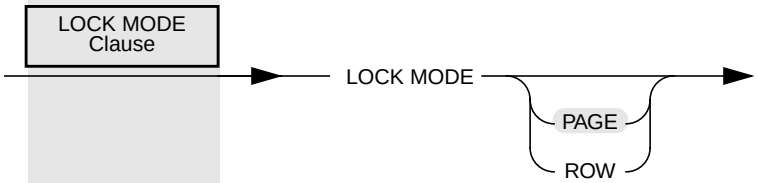


Constraint Definitiion

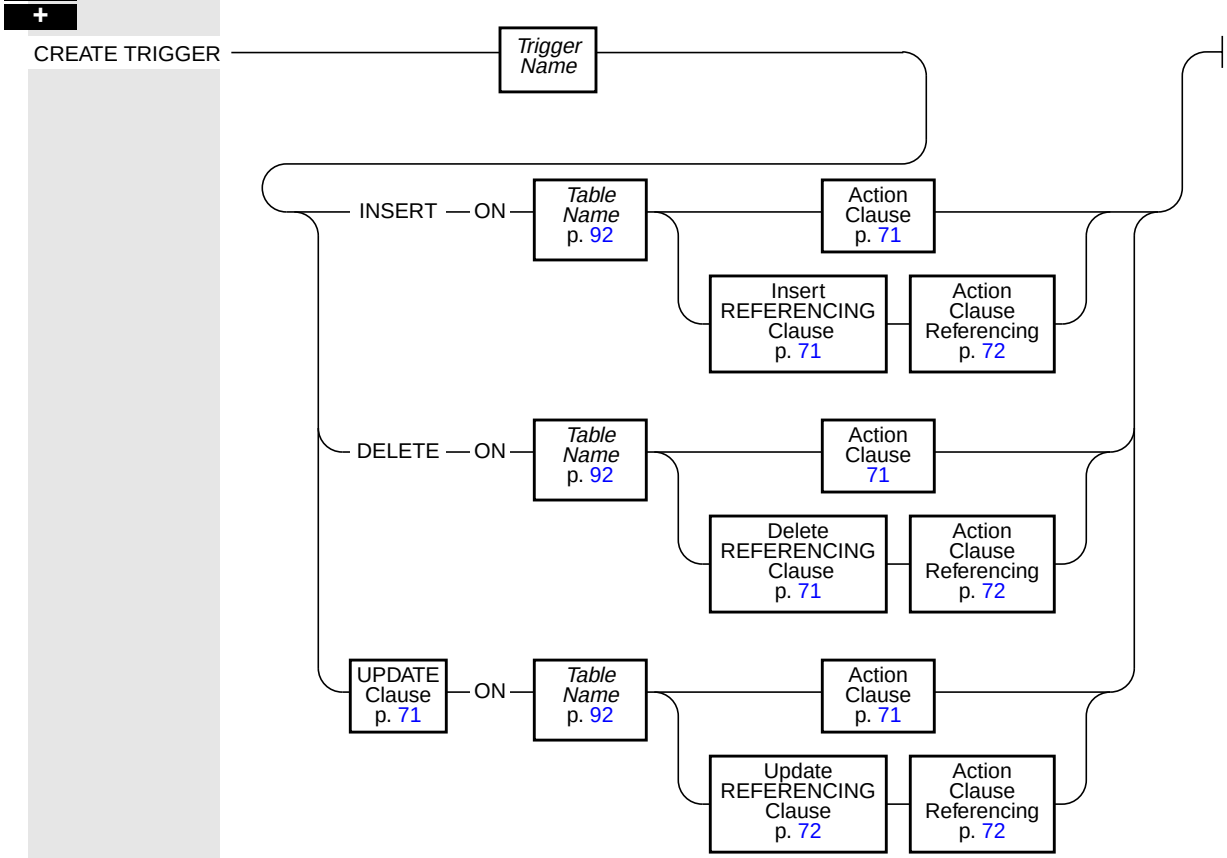


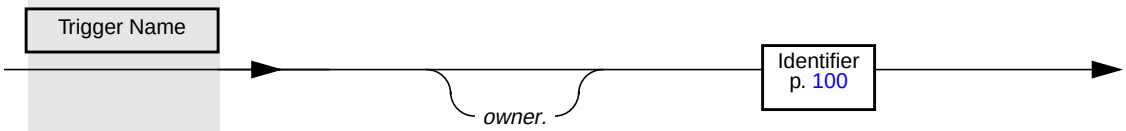


Basics
4GL
Forms
Reports
SQL
SQLCA
Debugger
Variables
Keys

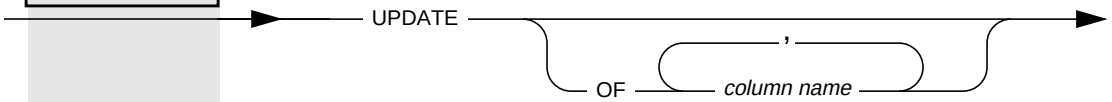

LOCK MODE Clause
LOCK MODE
PAGE
ROW

6.0 +

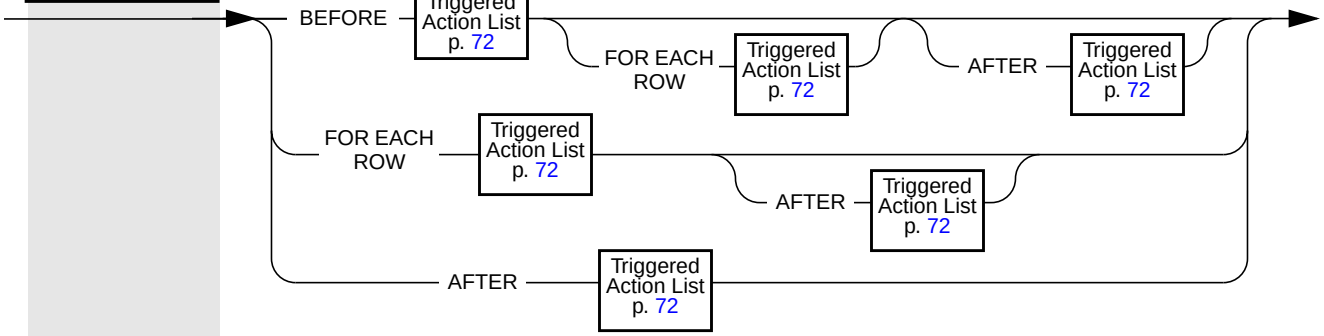

CREATE TRIGGER
Trigger Name
INSERT
ON
Table Name p. 92
Action Clause p. 71
Insert REFERENCING Clause p. 71
Action Clause Referencing p. 72
DELETE
ON
Table Name p. 92
Action Clause 71
Delete REFERENCING Clause p. 71
Action Clause Referencing p. 72
UPDATE Clause p. 71
ON
Table Name p. 92
Action Clause p. 71
Update REFERENCING Clause p. 72
Action Clause Referencing p. 72


Trigger Name
owner.
Identifier p. 100

UPDATE Clause

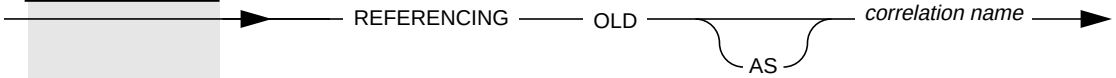


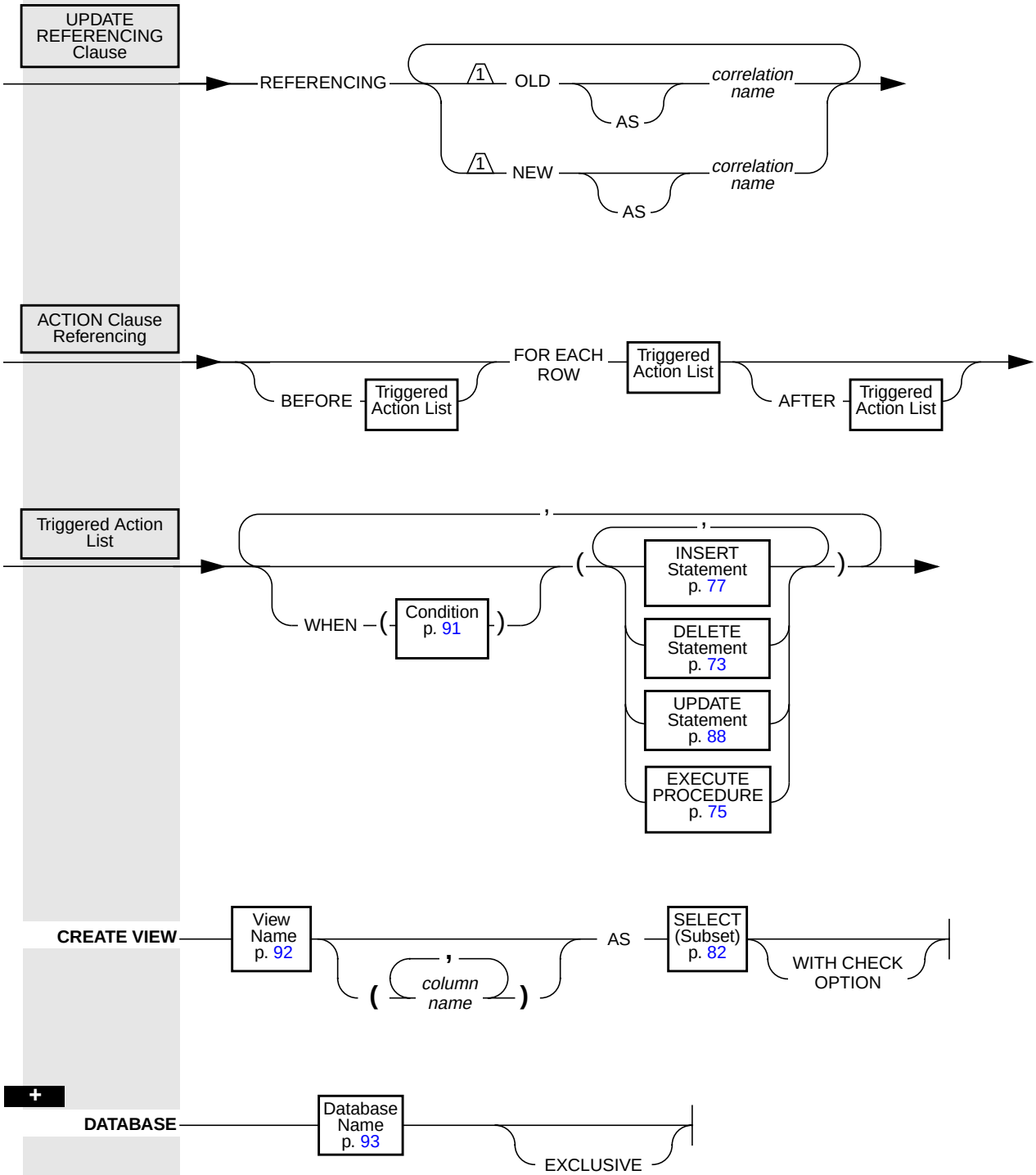
Action Clause

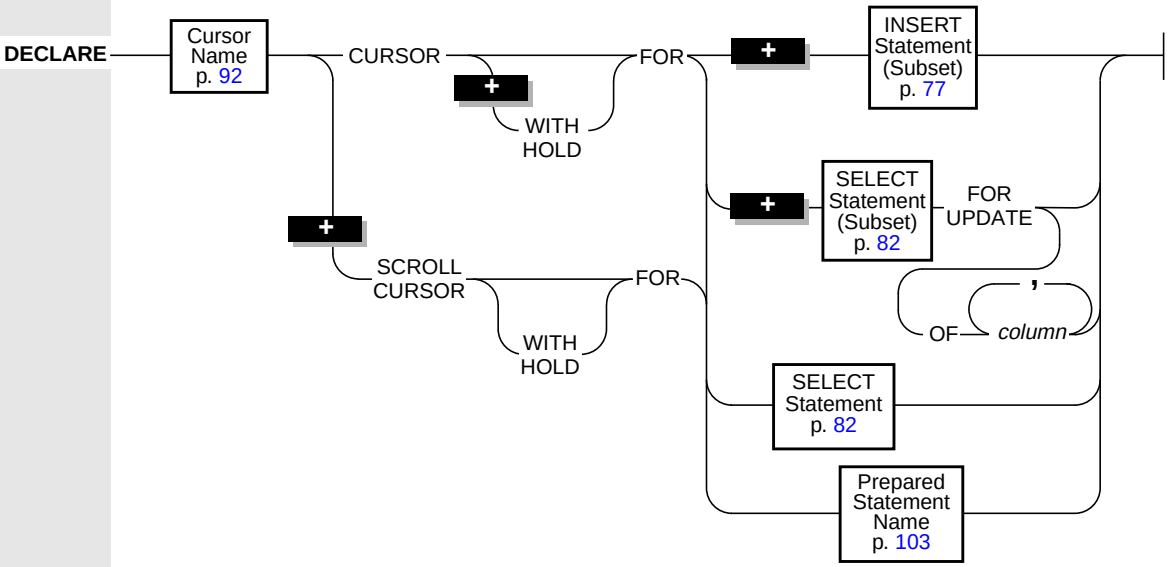


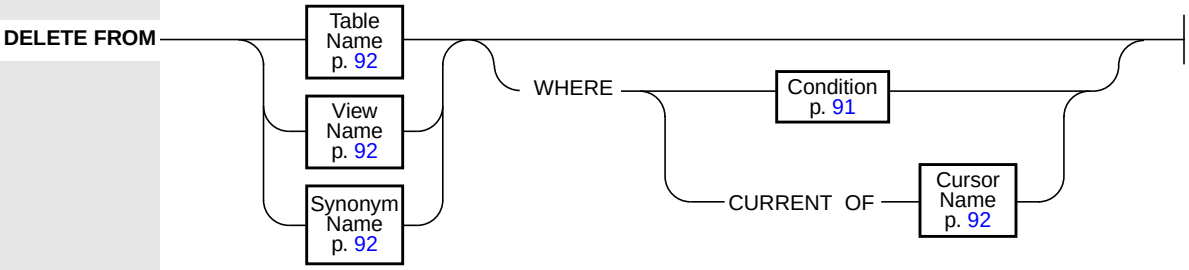
INSERT REFERENCING Clause

DELETE REFERENCING Clause

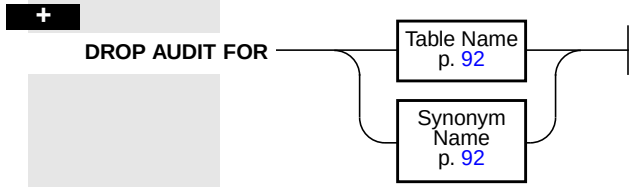


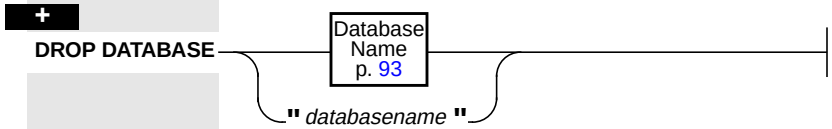

UPDATE REFERENCING Clause
REFERENCING
1
OLD
AS
correlation name
1
NEW
AS
correlation name
ACTION Clause Referencing
BEFORE
Triggered Action List
FOR EACH ROW
Triggered Action List
AFTER
Triggered Action List
Triggered Action List
,
,
WHEN
Condition p. 91
INSERT Statement p. 77
DELETE Statement p. 73
UPDATE Statement p. 88
EXECUTE PROCEDURE p. 75
CREATE VIEW
View Name p. 92
,
column name
AS
SELECT (Subset) p. 82
WITH CHECK OPTION
+
DATABASE
Database Name p. 93
EXCLUSIVE

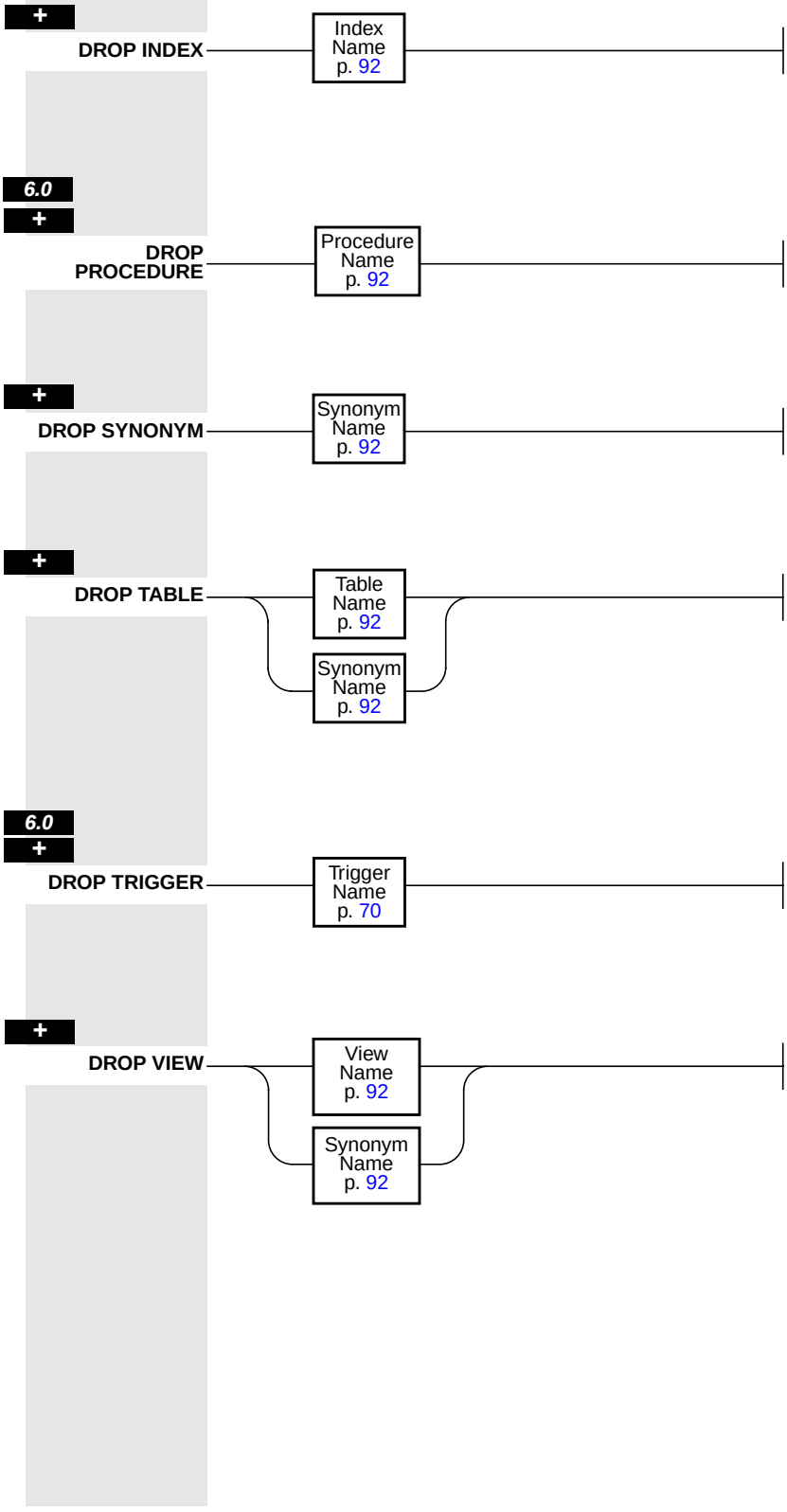

DECLARE
Cursor Name p. 92
CURSOR
WITH HOLD
FOR
INSERT Statement (Subset) p. 77
SELECT Statement (Subset) p. 82
FOR UPDATE
OF
column
SCROLL CURSOR
WITH HOLD
FOR
SELECT Statement p. 82
Prepared Statement Name p. 103

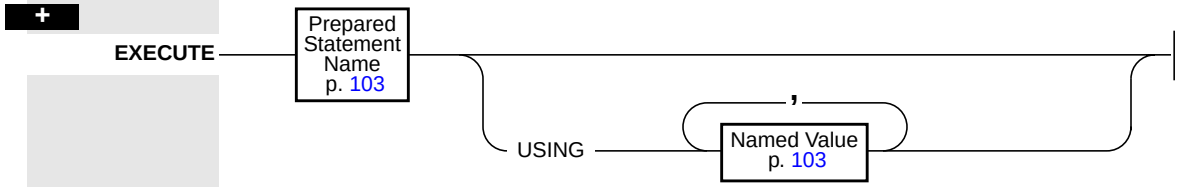

DELETE FROM
Table Name p. 92
View Name p. 92
Synonym Name p. 92
WHERE
Condition p. 91
CURRENT OF
Cursor Name p. 92

SE

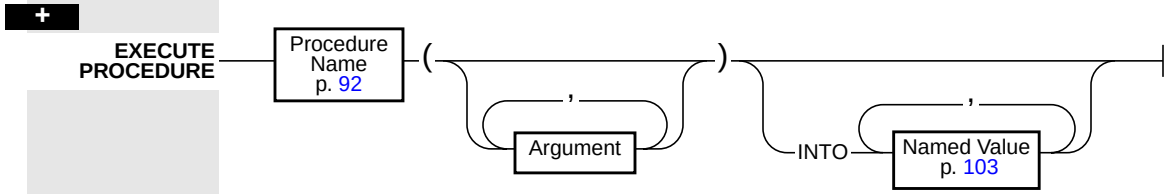

DROP AUDIT FOR
Table Name p. 92
Synonym Name p. 92


DROP DATABASE
Database Name p. 93
" databasename "

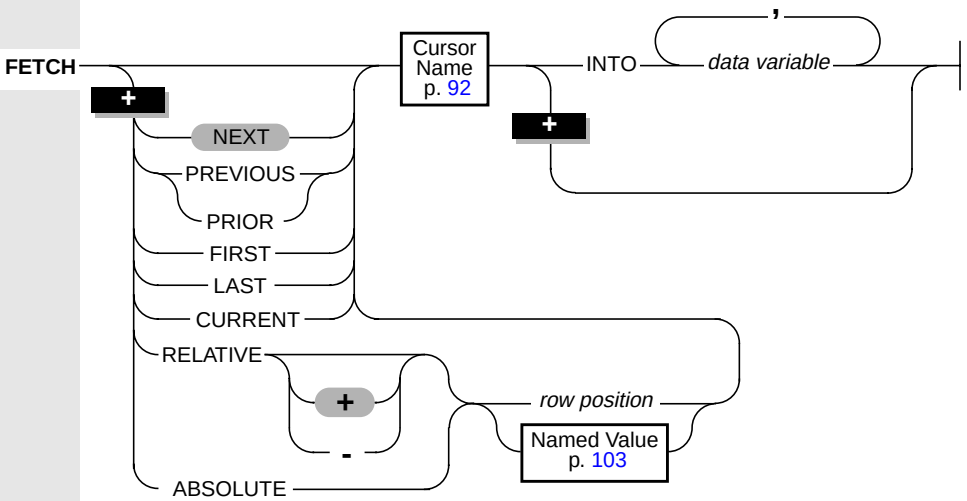

+
DROP INDEX
Index Name p. 92
6.0
+
DROP PROCEDURE
Procedure Name p. 92
+
DROP SYNONYM
Synonym Name p. 92
+
DROP TABLE
Table Name p. 92
Synonym Name p. 92
6.0
+
DROP TRIGGER
Trigger Name p. 70
+
DROP VIEW
View Name p. 92
Synonym Name p. 92

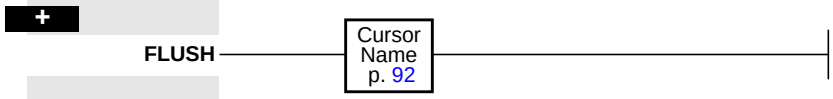


6.0

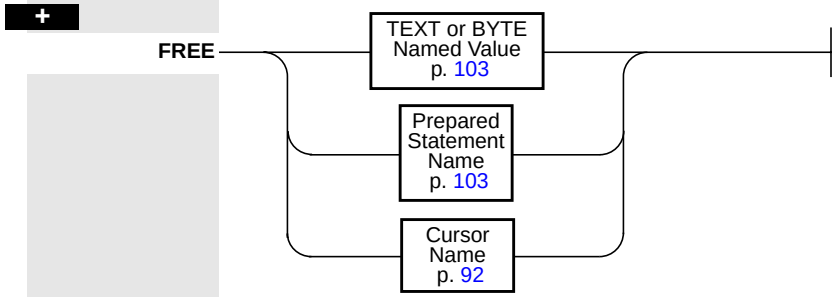


A procedure that returns no values must be executed by using PREPARE and EXECUTE. A procedure that returns values must be handled by using PREPARE and DECLARE, and then either a FOREACH loop or OPEN, FETCH, or CLOSE.



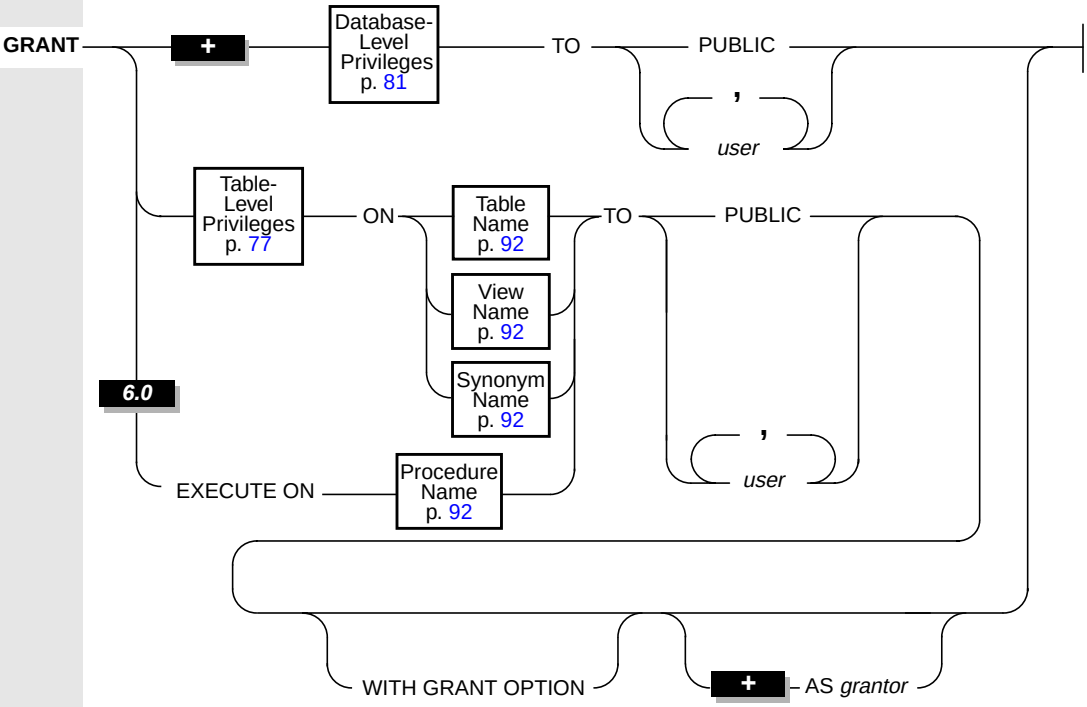


Basics
4GL
Forms
Reports
SQL
SQLCA
Debugger
Variables
Keys



The **4GL** compiler treats the name of the object to be freed in the order shown in the diagram. In other words, the compiler looks first for a TEXT or BYTE variable having the given name; if one exists, that is the object that is freed. If no TEXT or BYTE variable having that name exists, the compiler then looks for a prepared statement or a cursor having that name and frees that.

When a TEXT or BYTE variable has the same name as a prepared statement or cursor, you cannot free resources allocated to the prepared statement or to the cursor.



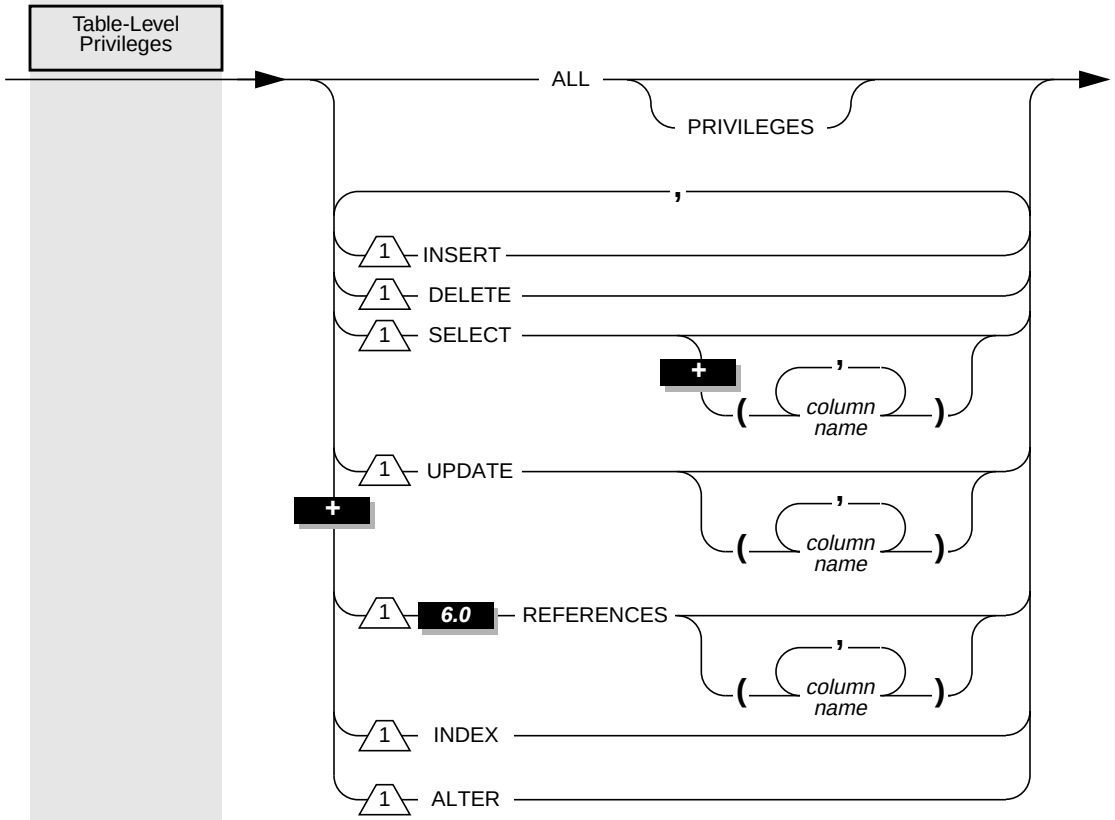
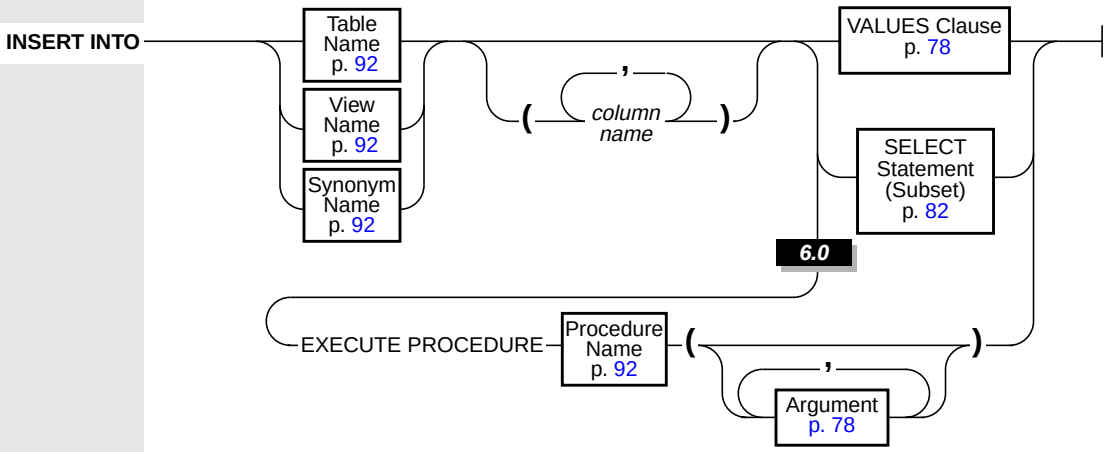
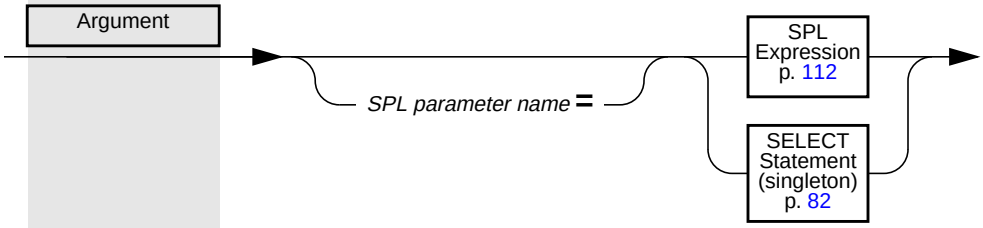
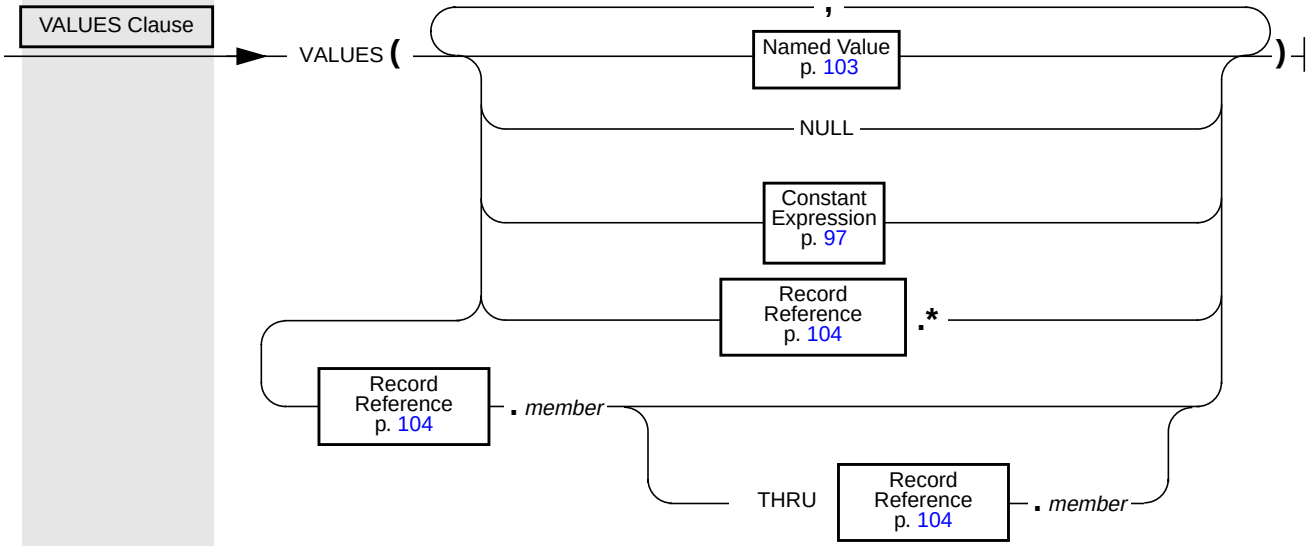
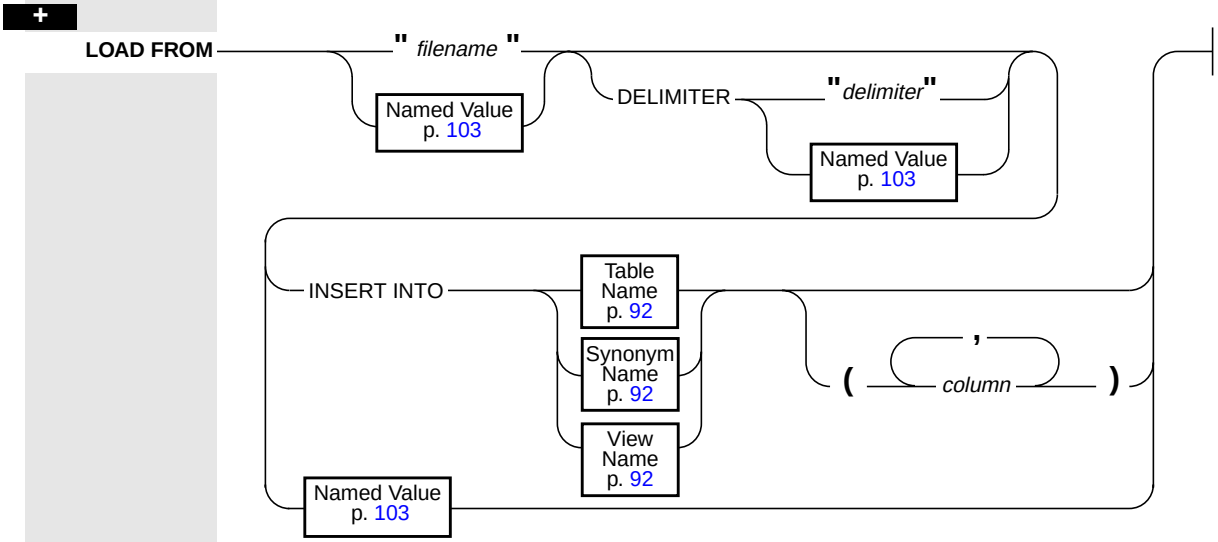


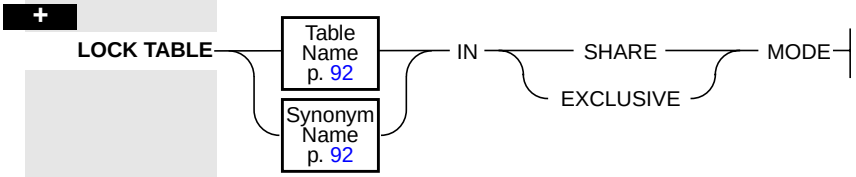
Table-Level Privileges
ALL
PRIVILEGES
INSERT
DELETE
SELECT
UPDATE
REFERENCES
6.0
INDEX
ALTER
column name

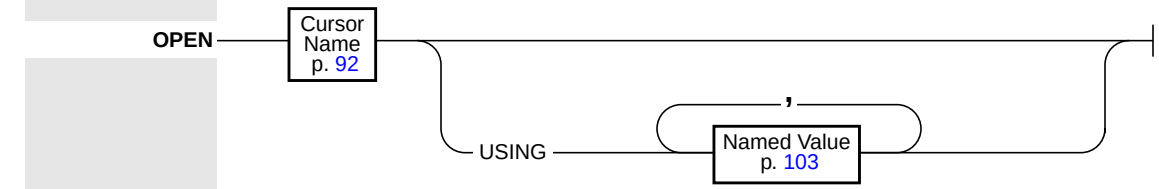

INSERT INTO
Table Name p. 92
View Name p. 92
Synonym Name p. 92
column name
VALUES Clause p. 78
SELECT Statement (Subset) p. 82
6.0
EXECUTE PROCEDURE
Procedure Name p. 92
Argument p. 78

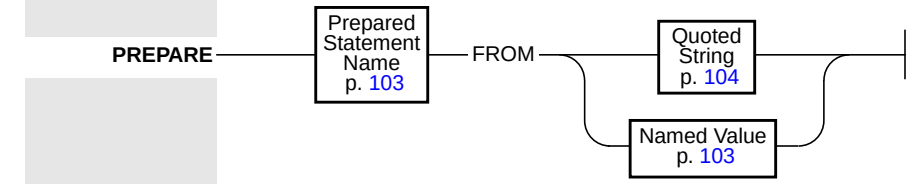

Argument
SPL parameter name =
SPL Expression p. 112
SELECT Statement (singleton) p. 82

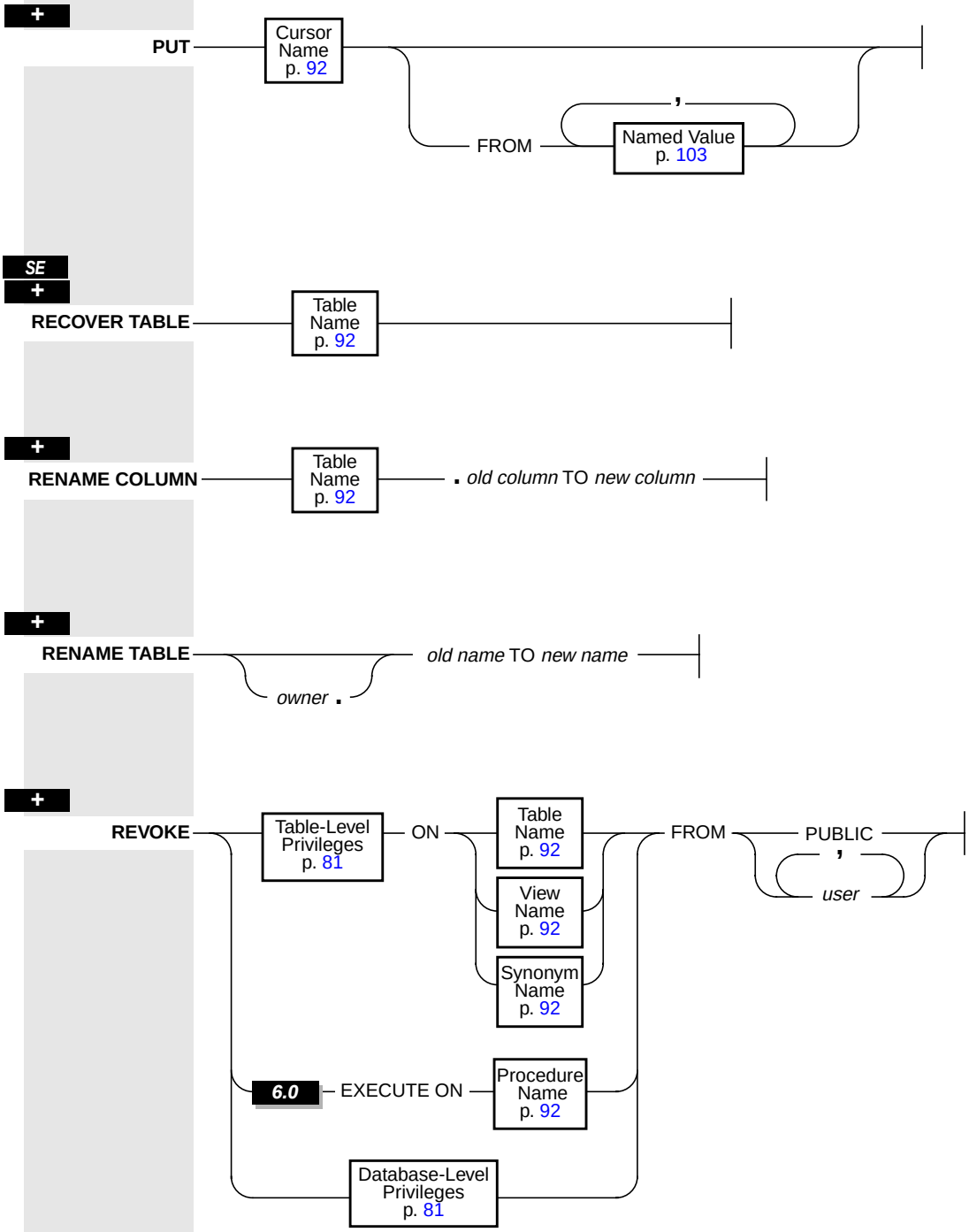

VALUES Clause
VALUES (
,
Named Value p. 103
)
NULL
Constant Expression p. 97
Record Reference p. 104
.*
Record Reference p. 104
. member
THRU
Record Reference p. 104
. member


LOAD FROM
" filename "
Named Value p. 103
DELIMITER
" delimiter "
Named Value p. 103
INSERT INTO
Table Name p. 92
Synonym Name p. 92
View Name p. 92
(, column)
Named Value p. 103


LOCK TABLE
Table Name p. 92
Synonym Name p. 92
IN
SHARE
EXCLUSIVE
MODE


OPEN
Cursor Name p. 92
USING
, Named Value p. 103


PREPARE
Prepared Statement Name p. 103
FROM
Quoted String p. 104
Named Value p. 103


+
PUT
Cursor Name p. 92
,
FROM
Named Value p. 103
SE
+
RECOVER TABLE
Table Name p. 92
+
RENAME COLUMN
Table Name p. 92
. old column TO new column
+
RENAME TABLE
owner .
old name TO new name
+
REVOKE
Table-Level Privileges p. 81
ON
Table Name p. 92
View Name p. 92
Synonym Name p. 92
FROM
PUBLIC
,
user
6.0
EXECUTE ON
Procedure Name p. 92
Database-Level Privileges p. 81

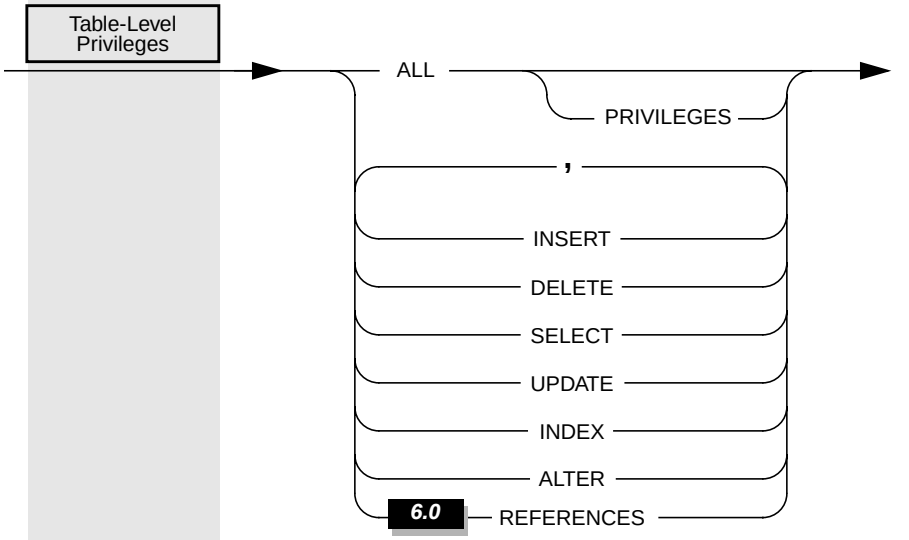
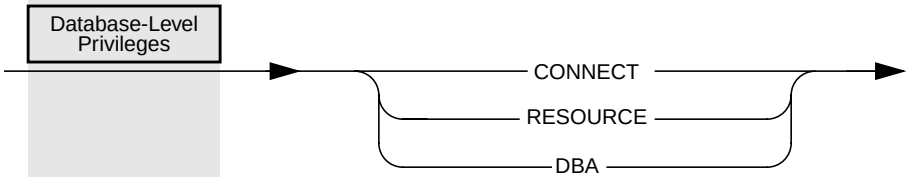
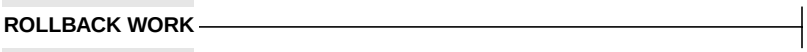


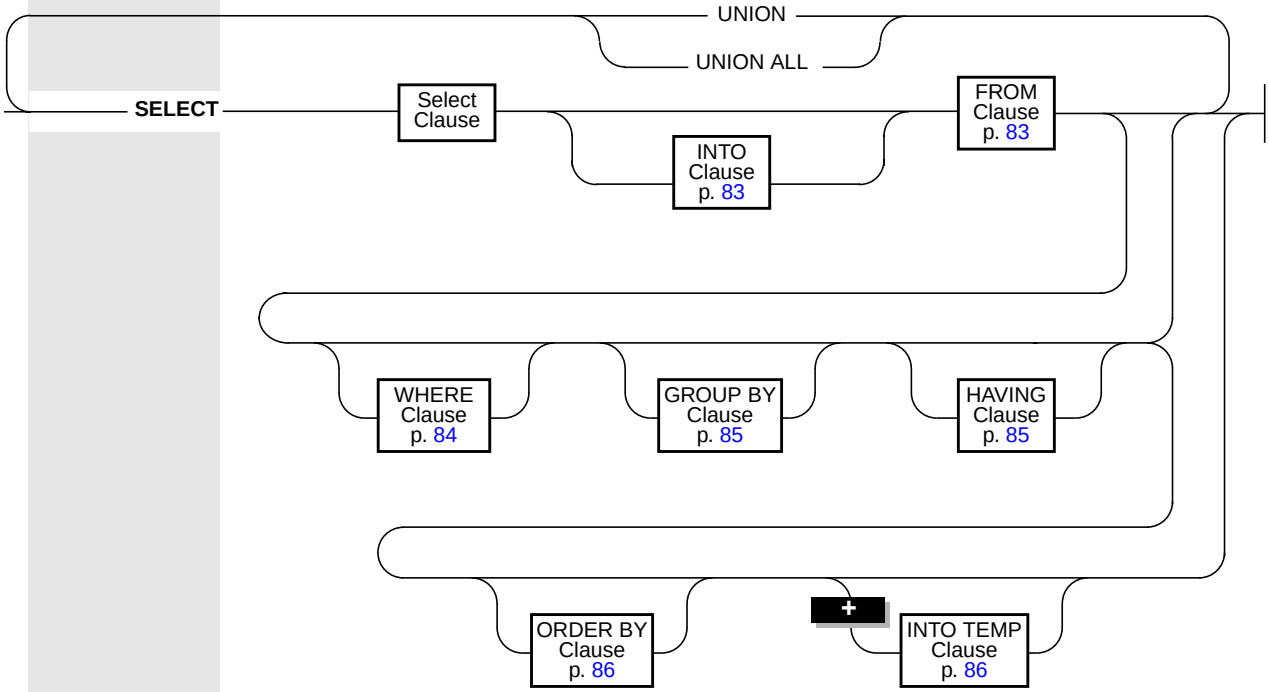
Table-Level Privileges
ALL
PRIVILEGES
,
INSERT
DELETE
SELECT
UPDATE
INDEX
ALTER
6.0
REFERENCES

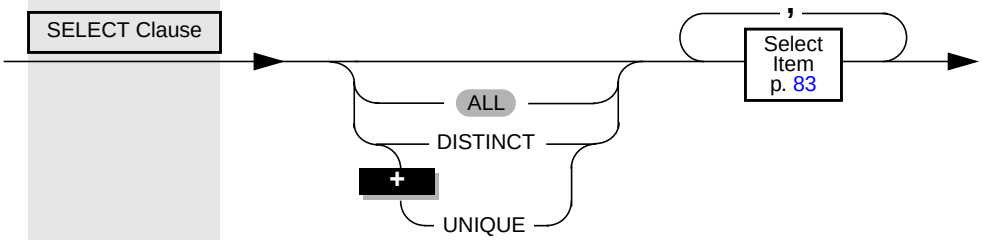

Database-Level Privileges
CONNECT
RESOURCE
DBA

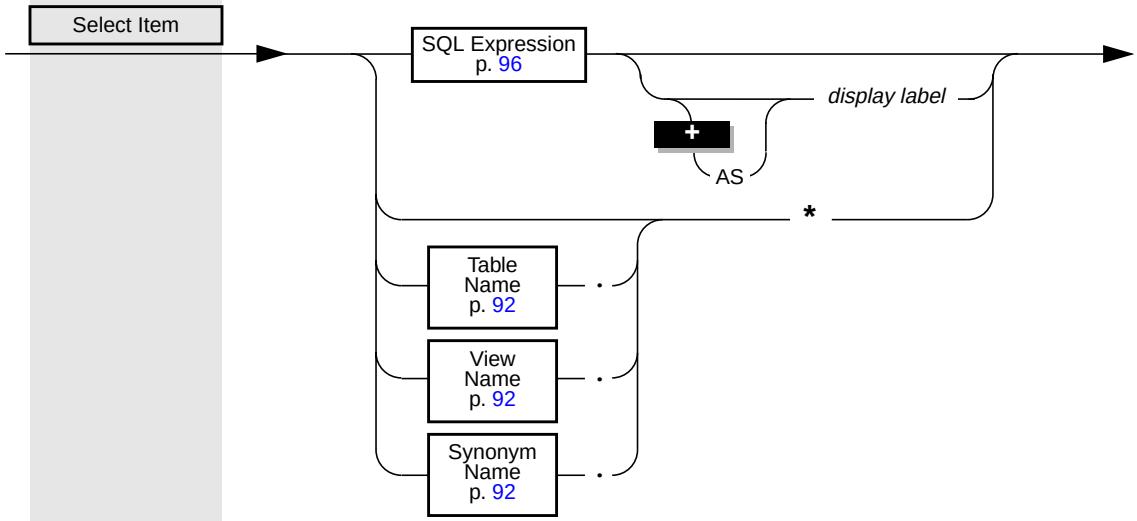

ROLLBACK WORK

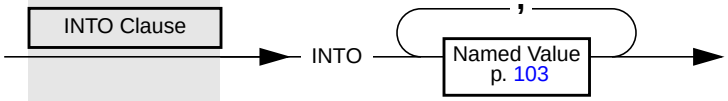
SE
+

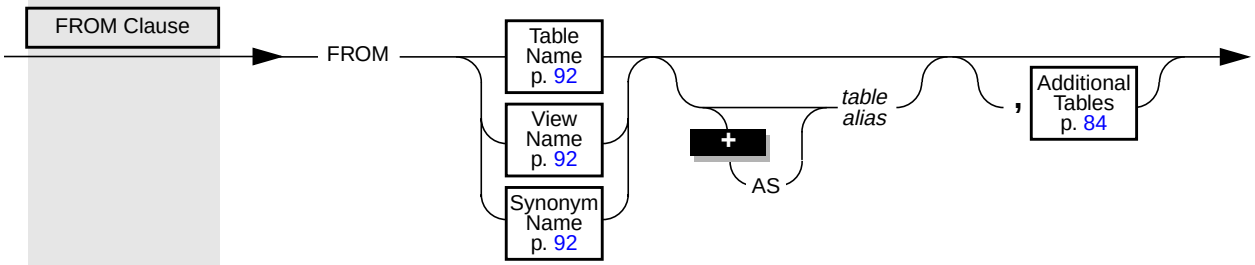

ROLLFORWARD DATABASE
Database Name
p. 93

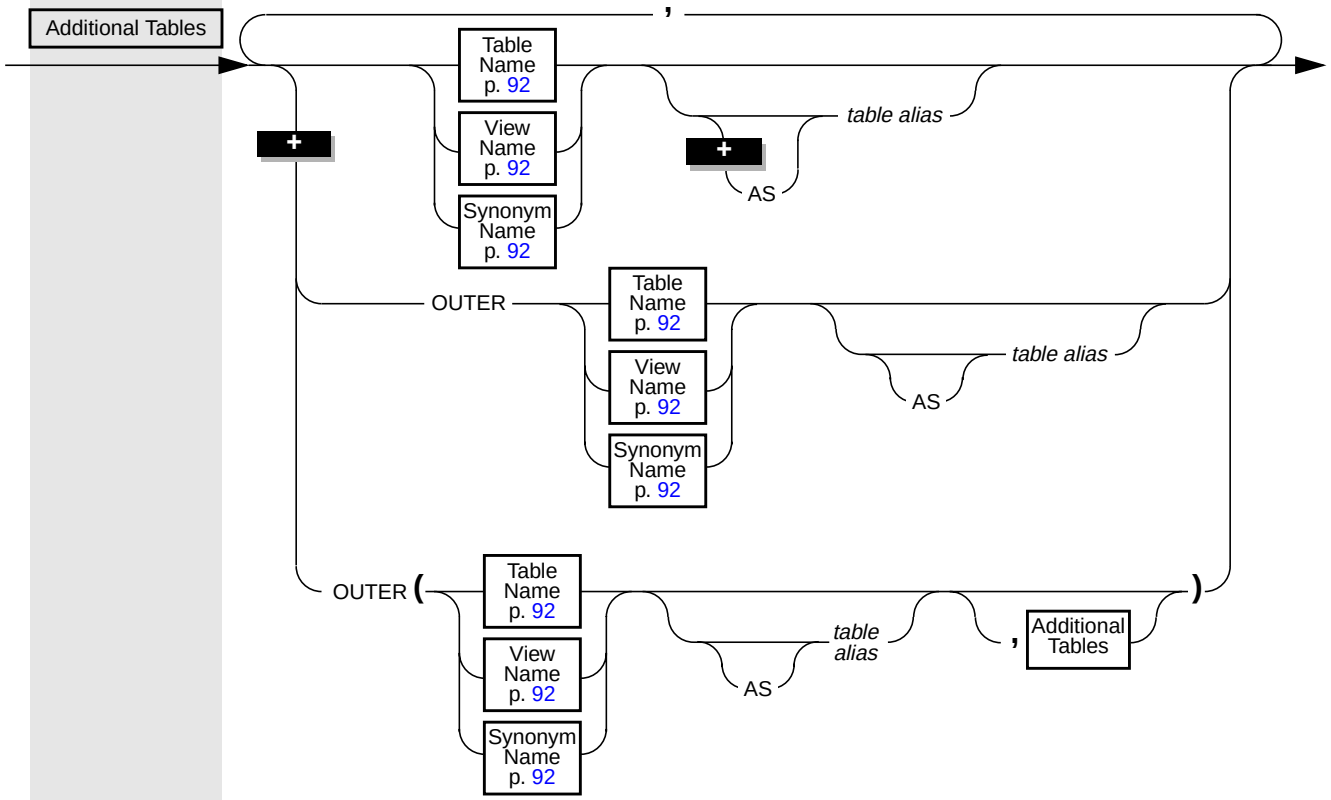

UNION
UNION ALL
SELECT
Select Clause
INTO Clause p. 83
FROM Clause p. 83
WHERE Clause p. 84
GROUP BY Clause p. 85
HAVING Clause p. 85
ORDER BY Clause p. 86
+
INTO TEMP Clause p. 86

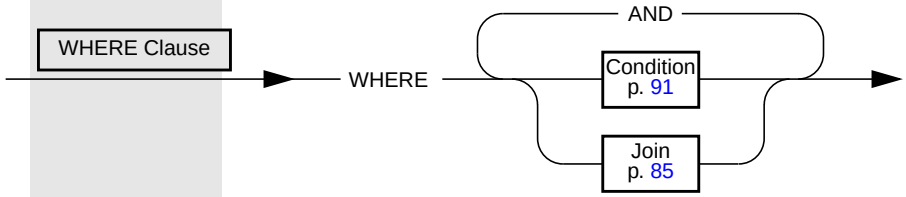

SELECT Clause
,
Select Item p. 83
ALL
DISTINCT
+
UNIQUE

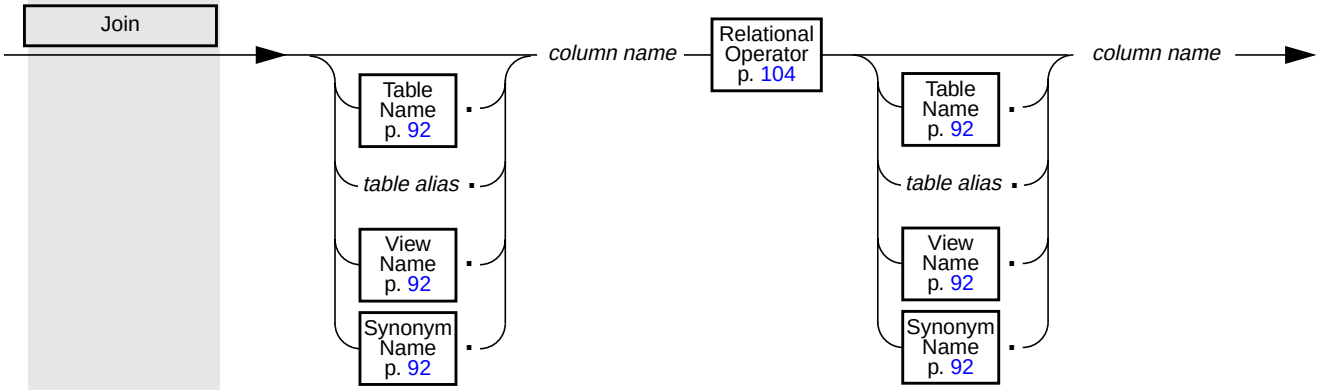

Select Item
SQL Expression
p. 96
display label
+
AS
*
Table
Name
p. 92
View
Name
p. 92
Synonym
Name
p. 92
.

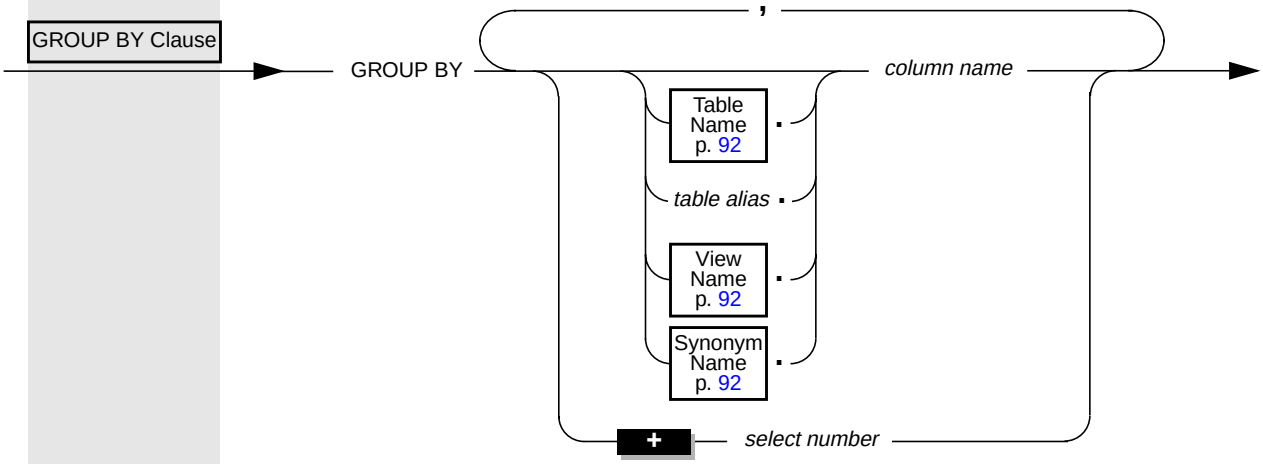

INTO Clause
INTO
,
Named Value
p. 103

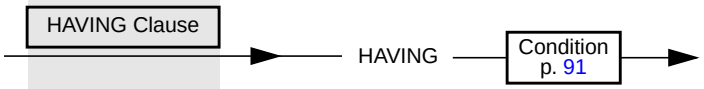

FROM Clause
FROM
Table
Name
p. 92
View
Name
p. 92
Synonym
Name
p. 92
+
AS
table
alias
,
Additional
Tables
p. 84


Additional Tables
,
Table Name p. 92
View Name p. 92
Synonym Name p. 92
table alias
AS
OUTER
Table Name p. 92
View Name p. 92
Synonym Name p. 92
table alias
AS
OUTER
(
Table Name p. 92
View Name p. 92
Synonym Name p. 92
table alias
AS
,
Additional Tables
)

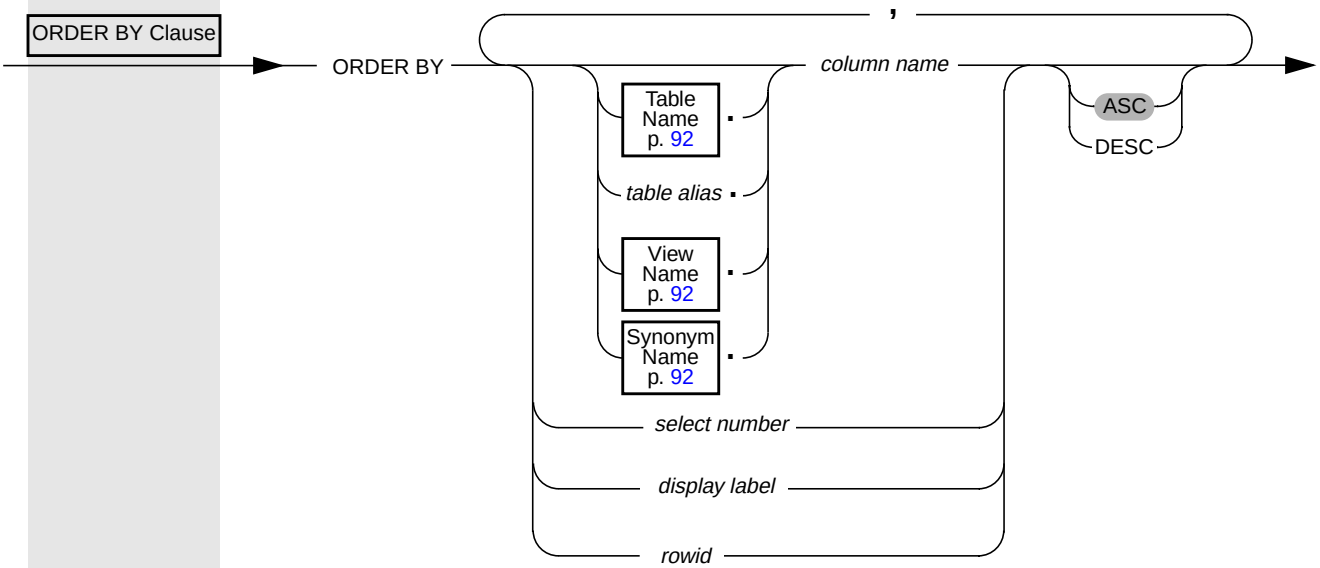

WHERE Clause
WHERE
AND
Condition p. 91
Join p. 85


Join
Table Name p. 92
table alias
View Name p. 92
Synonym Name p. 92
column name
Relational Operator p. 104
Table Name p. 92
table alias
View Name p. 92
Synonym Name p. 92
column name

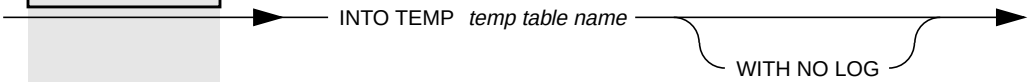

GROUP BY Clause
GROUP BY
,
Table Name p. 92
table alias
View Name p. 92
Synonym Name p. 92
column name
+
select number


HAVING Clause
HAVING
Condition p. 91

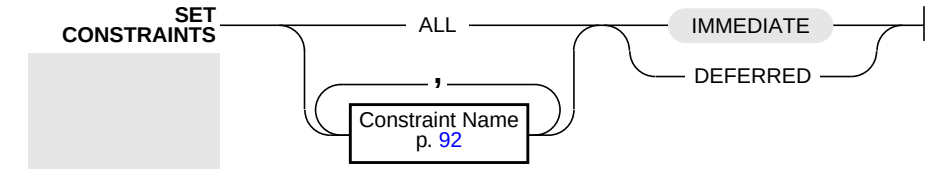
ORDER BY Clause



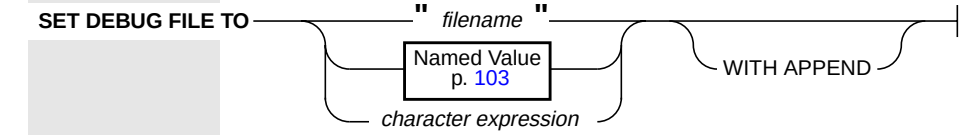
INTO TEMP Clause

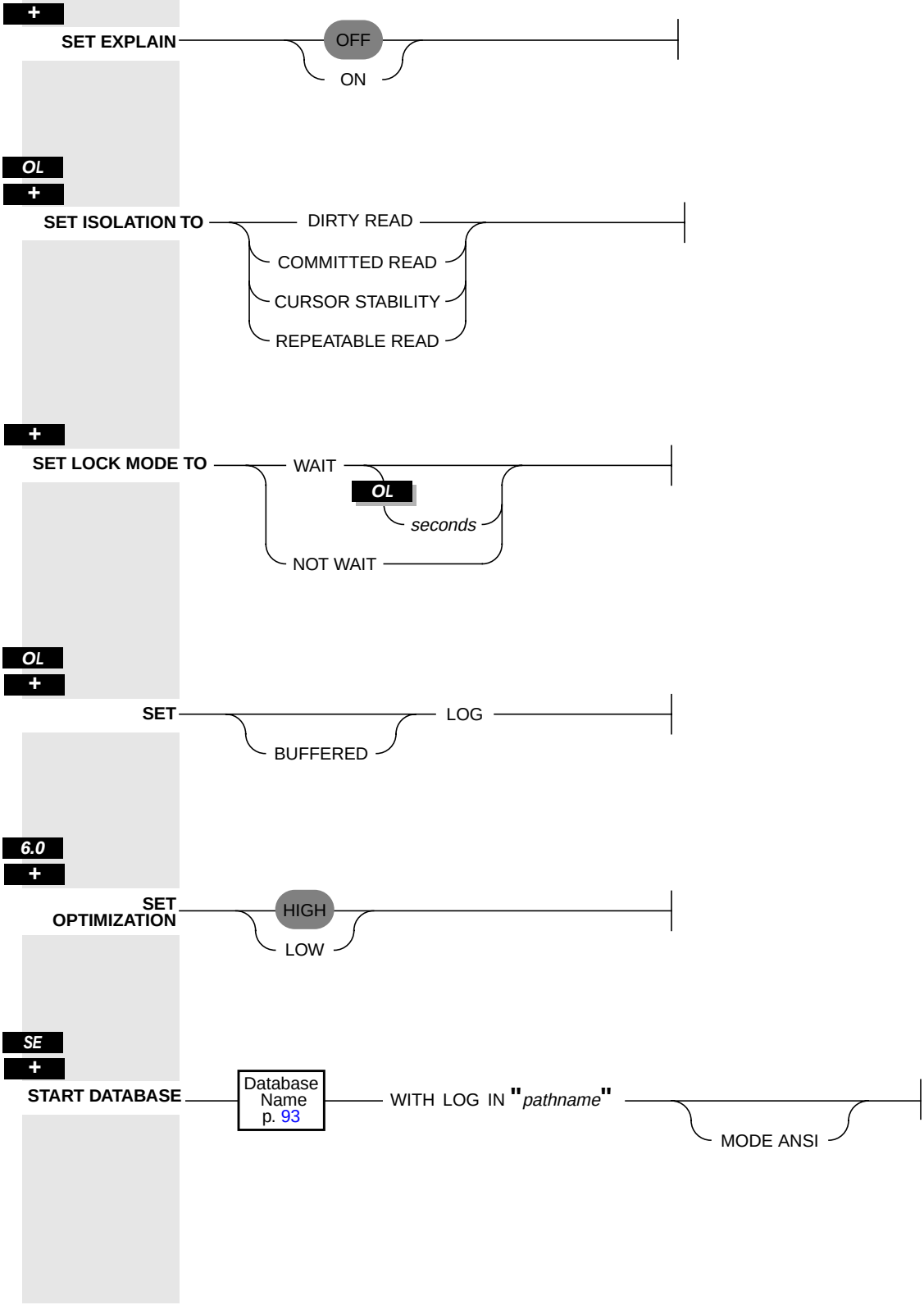


6.0
OL
+

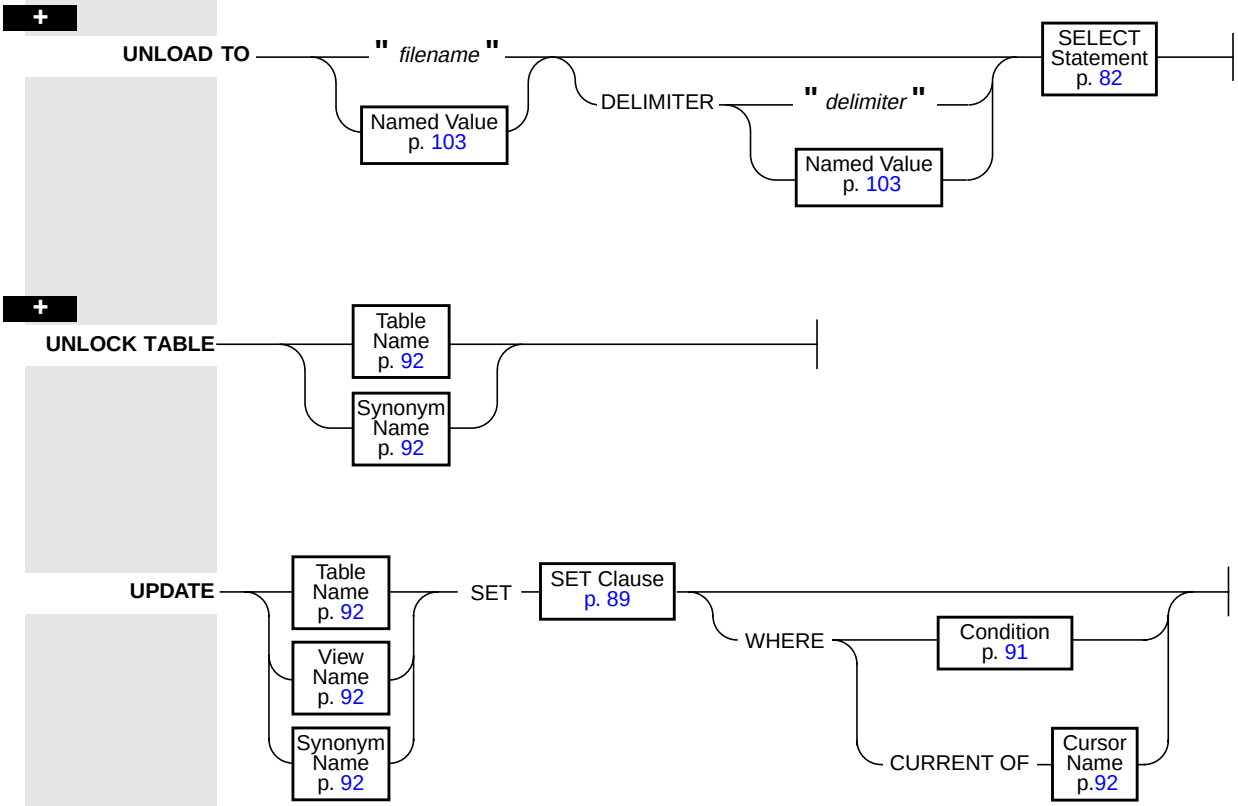


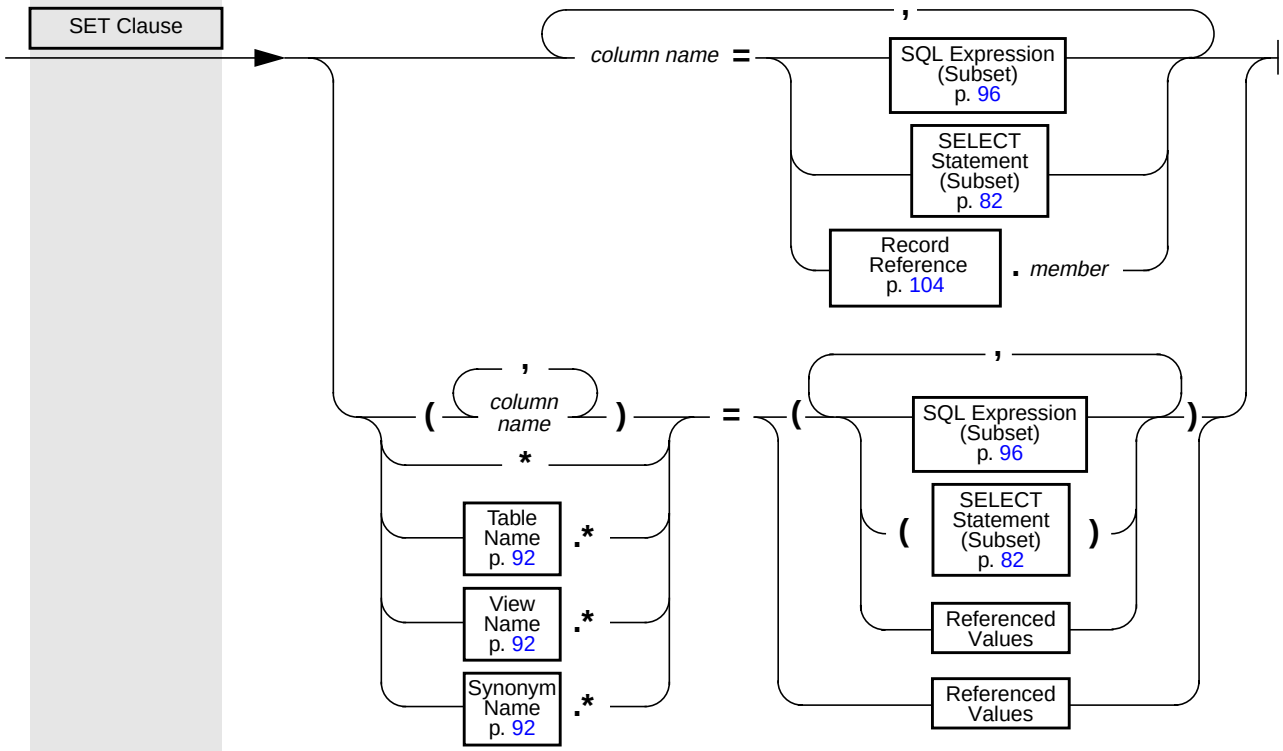
6.0
+

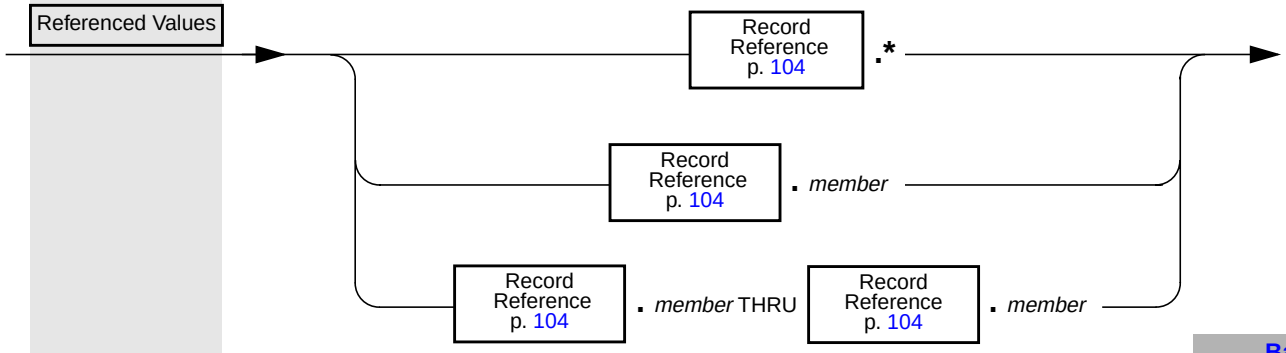


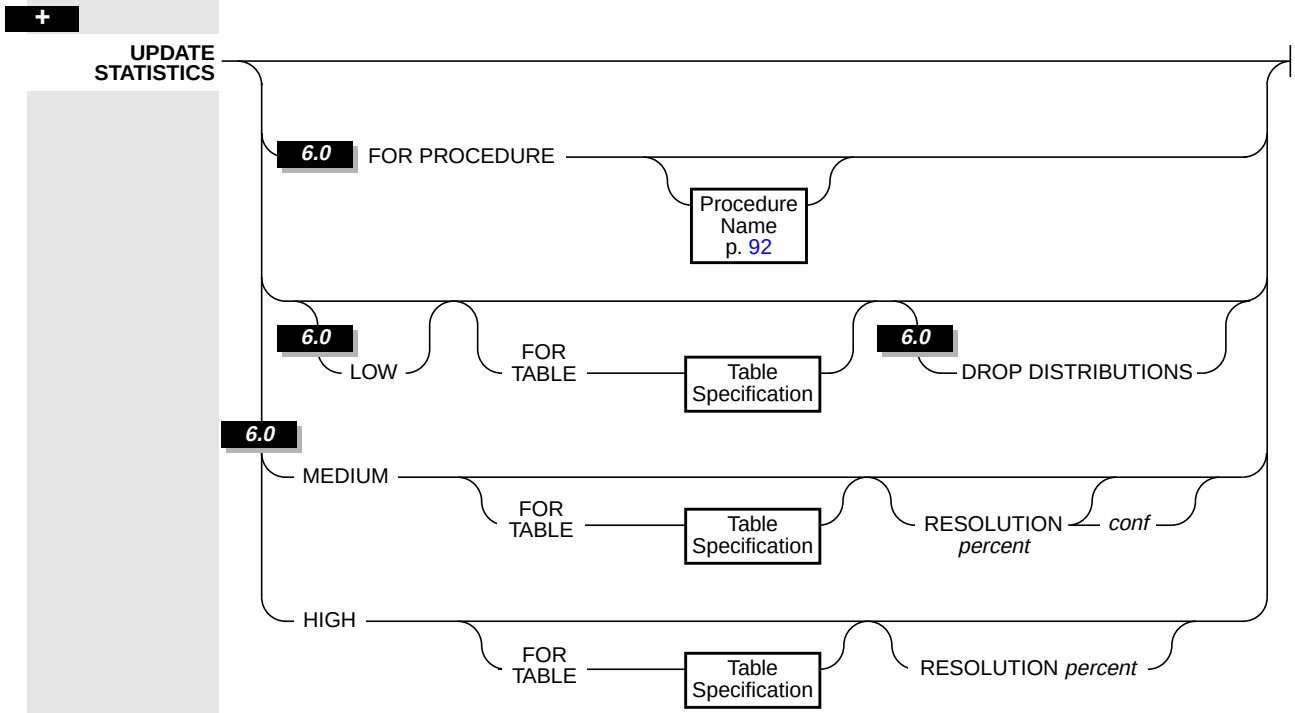


Basics
4GL
Forms
Reports
SQL
SQLCA
Debugger
Variables
Keys


+
UNLOAD TO
" filename "
Named Value p. 103
DELIMITER
" delimiter "
Named Value p. 103
SELECT Statement p. 82
+
UNLOCK TABLE
Table Name p. 92
Synonym Name p. 92
UPDATE
Table Name p. 92
View Name p. 92
Synonym Name p. 92
SET
SET Clause p. 89
WHERE
Condition p. 91
CURRENT OF
Cursor Name p.92


SET Clause
column name
=
SQL Expression (Subset) p. 96
SELECT Statement (Subset) p. 82
Record Reference p. 104
. member
(
column name
)
*
Table Name p. 92
View Name p. 92
Synonym Name p. 92
.*


Referenced Values
Record Reference p. 104
.*
Record Reference p. 104
. member
Record Reference p. 104
. member THRU
Record Reference p. 104
. member


+
UPDATE STATISTICS
6.0
FOR PROCEDURE
Procedure Name p. 92
6.0
LOW
FOR TABLE
Table Specification
6.0
DROP DISTRIBUTIONS
6.0
MEDIUM
FOR TABLE
Table Specification
RESOLUTION
conf
percent
HIGH
FOR TABLE
Table Specification
RESOLUTION percent

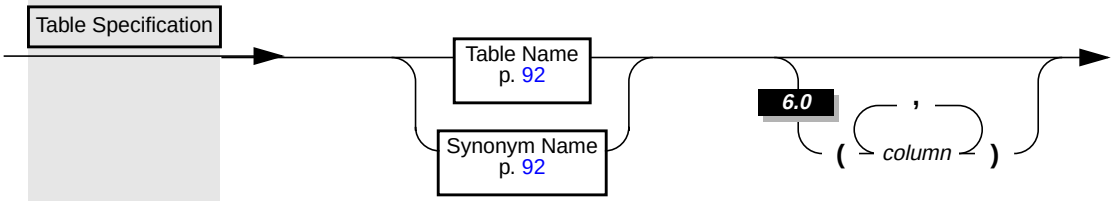
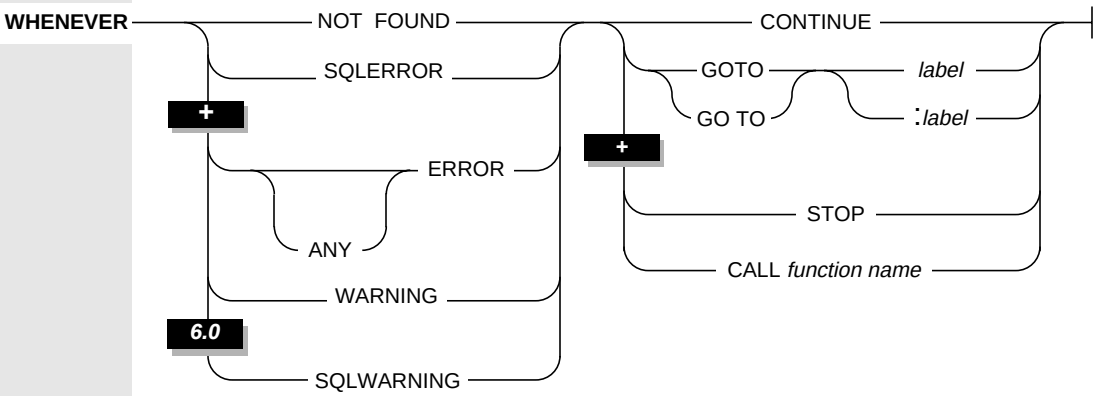
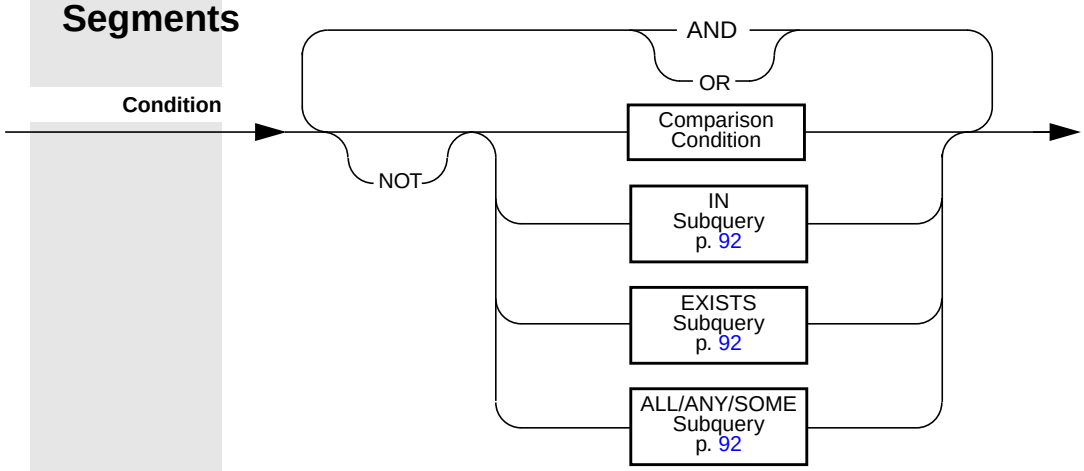


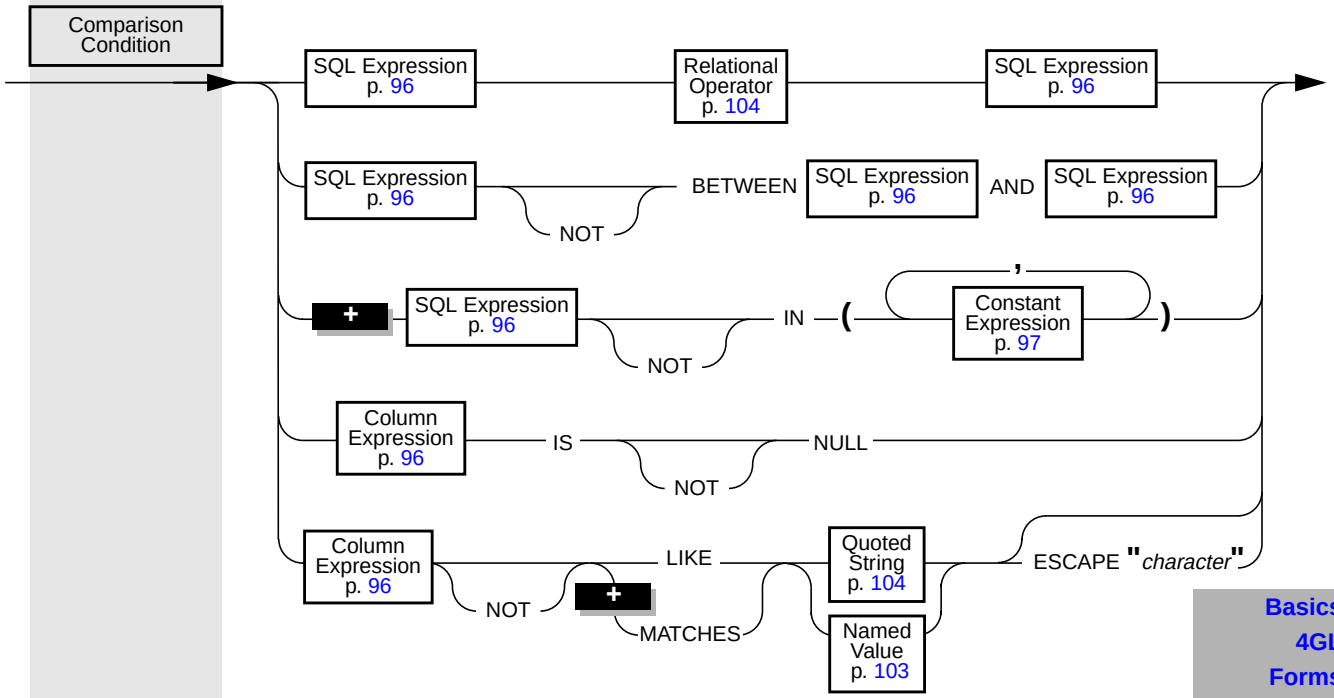
Table Specification
Table Name p. 92
Synonym Name p. 92
6.0
,
(column)

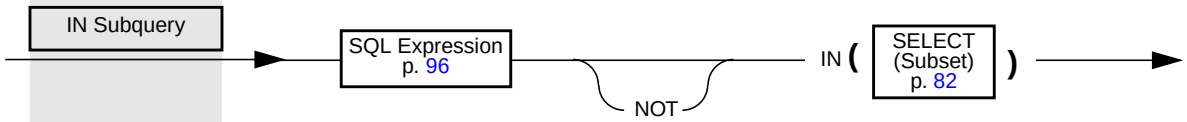

WHENEVER
NOT FOUND
CONTINUE
SQLERROR
GOTO
label
GO TO
:label
+
ERROR
ANY
+
STOP
CALL function name
WARNING
6.0
SQLWARNING

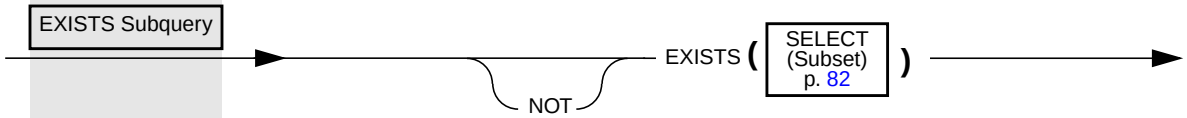
SQL Segments

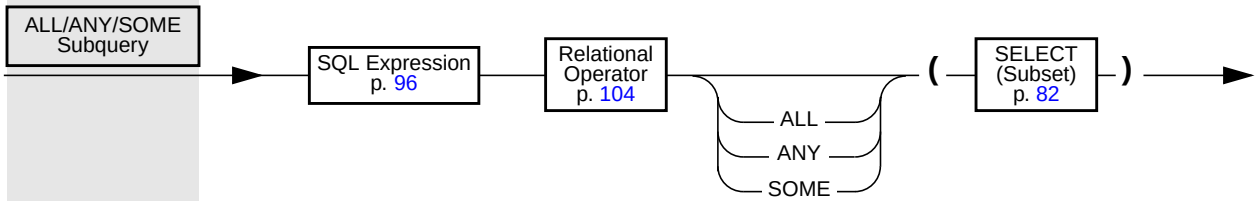
Condition











Constraint Name

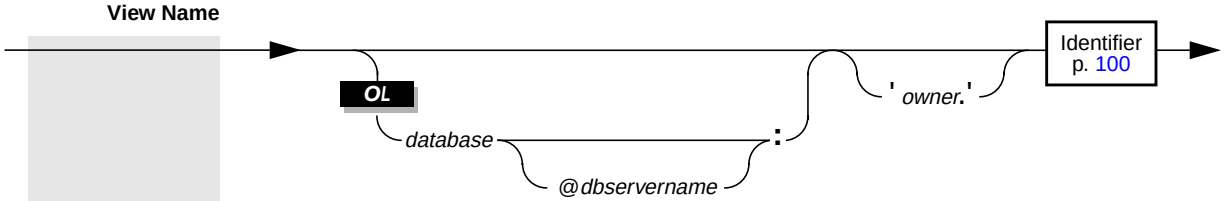
Index Name

Procedure Name

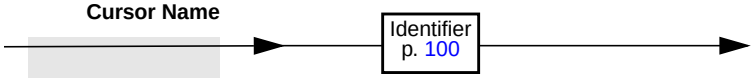
Synonym Name

Table Name

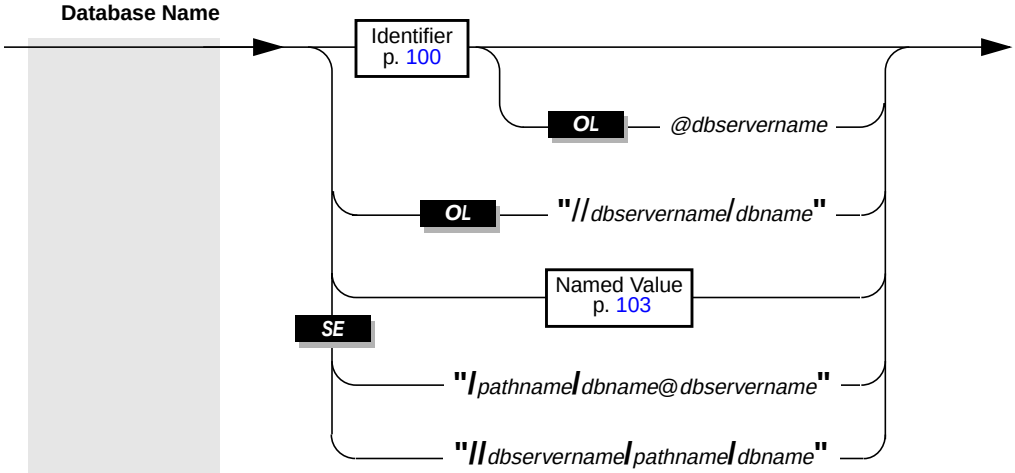
View Name



Cursor Name



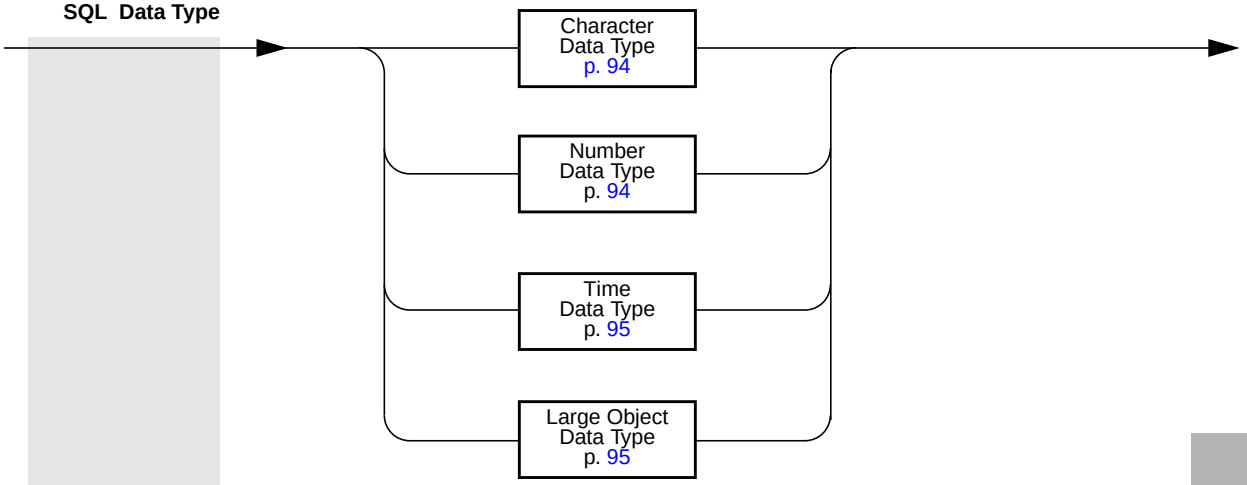
Database Name



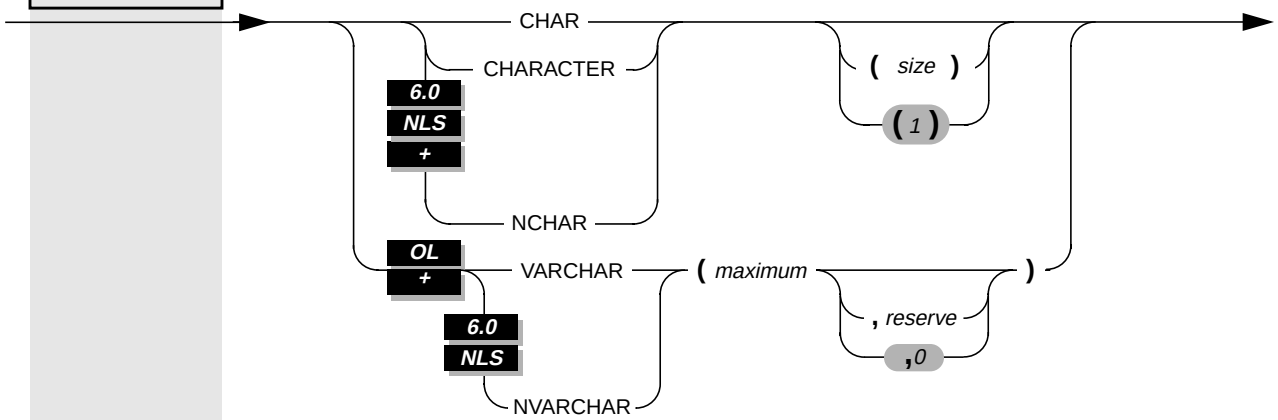
For **SE** engines, database identifiers can have up to ten characters in UNIX.

When the identifier for a database is also the name of a **4GL** variable, the compiler uses the variable. To override this compiler action, quote the database identifier.

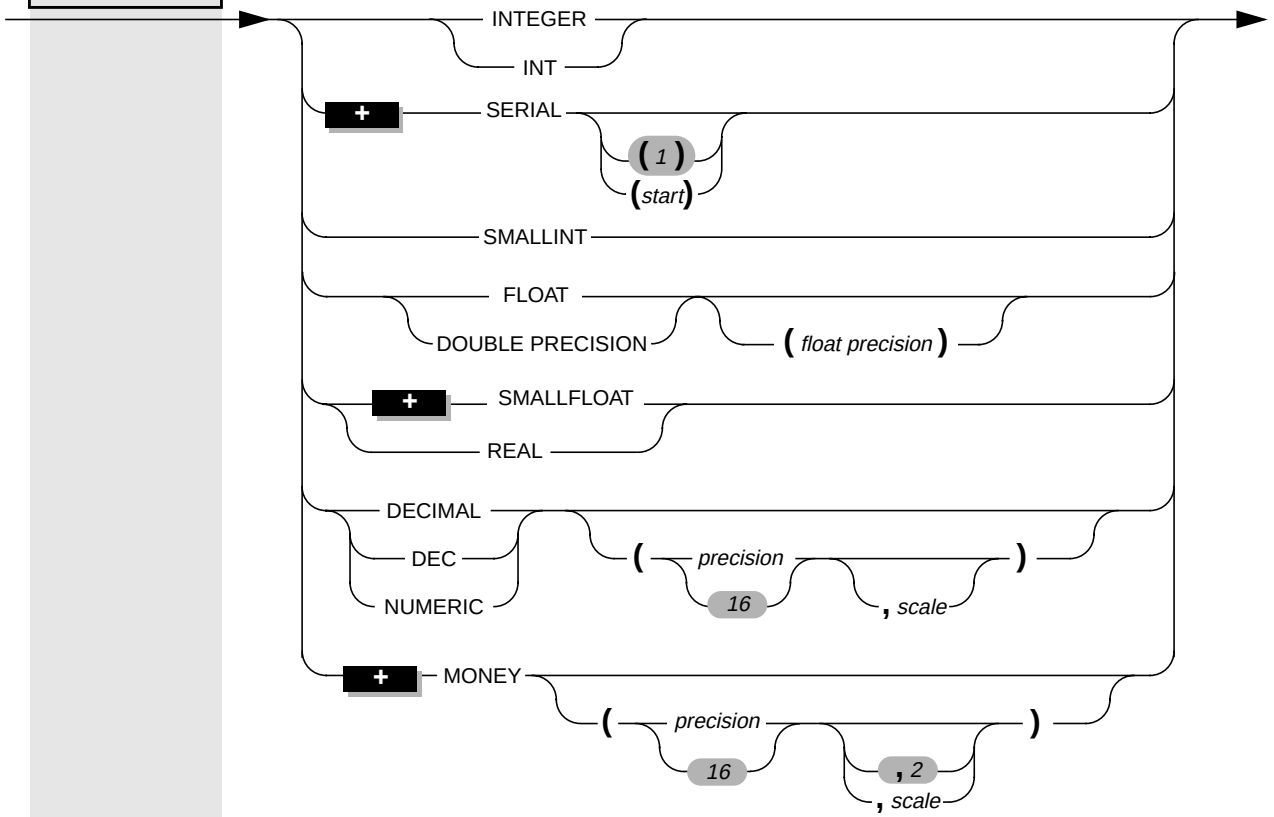
SQL Data Type

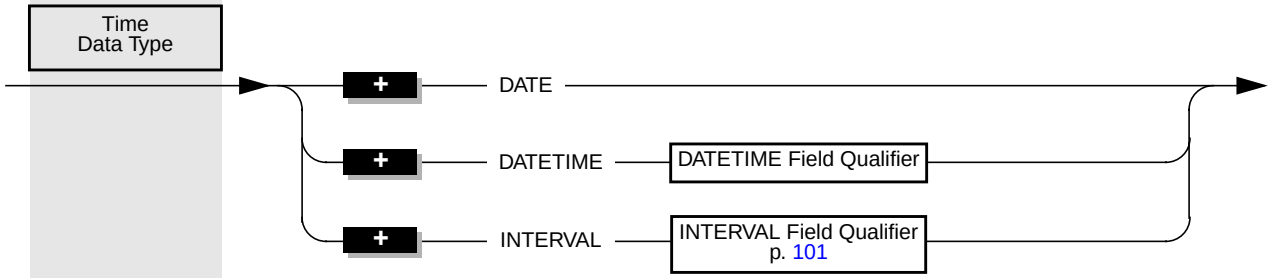


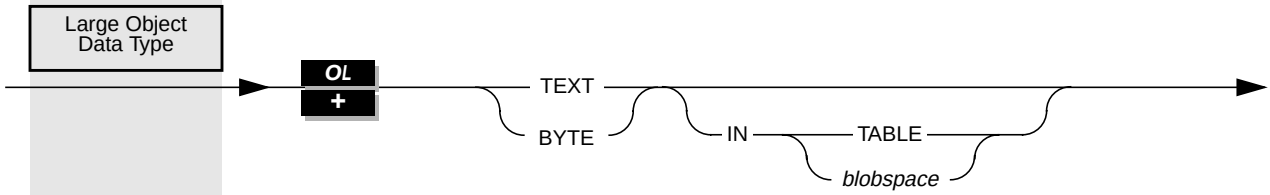
Character Data Type



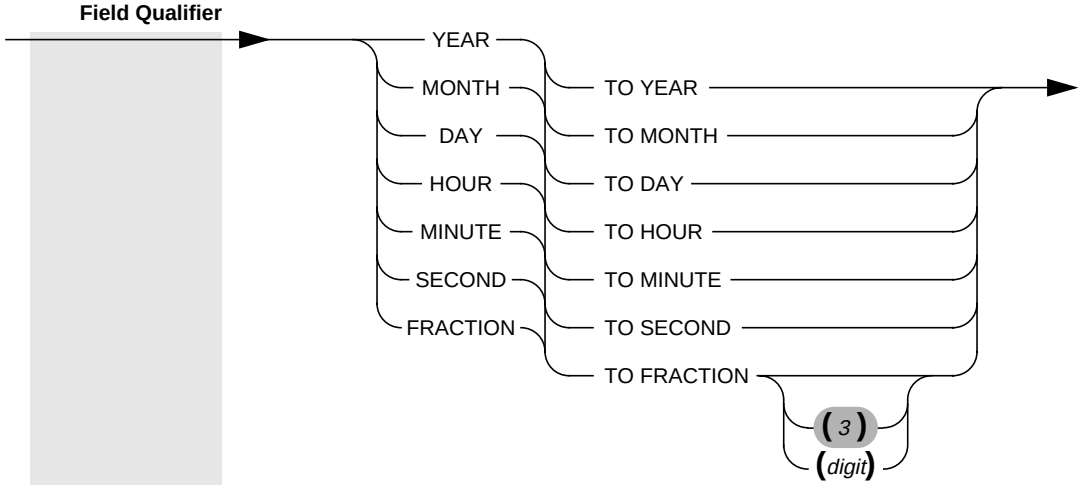
Number Data Type

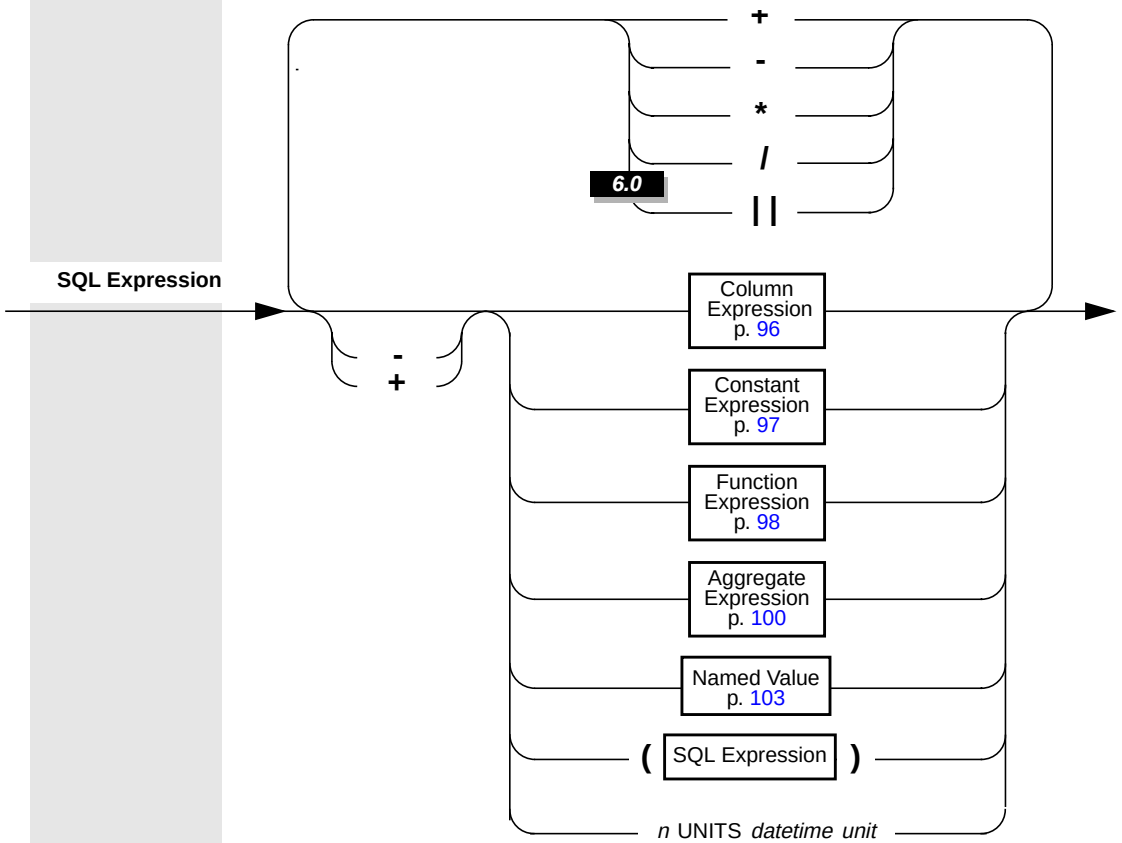


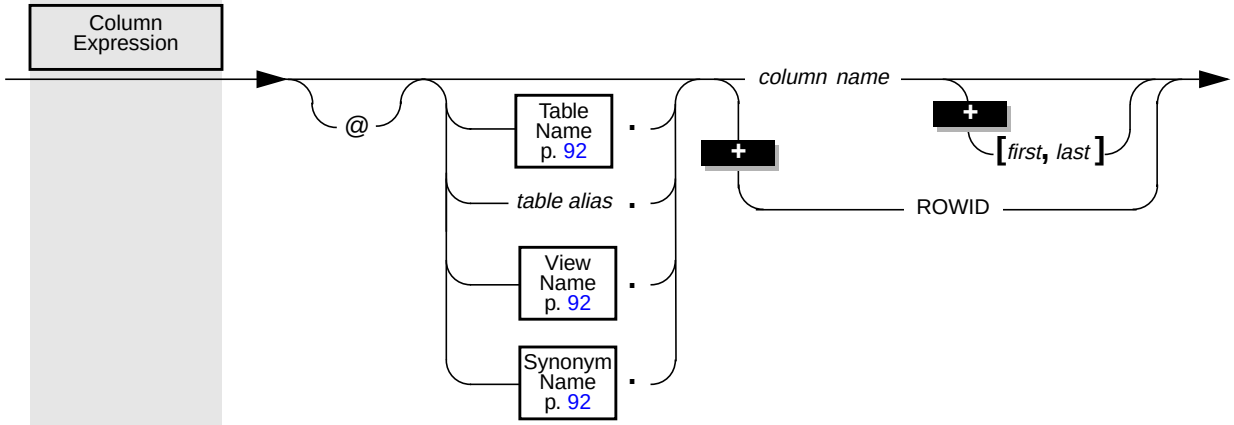




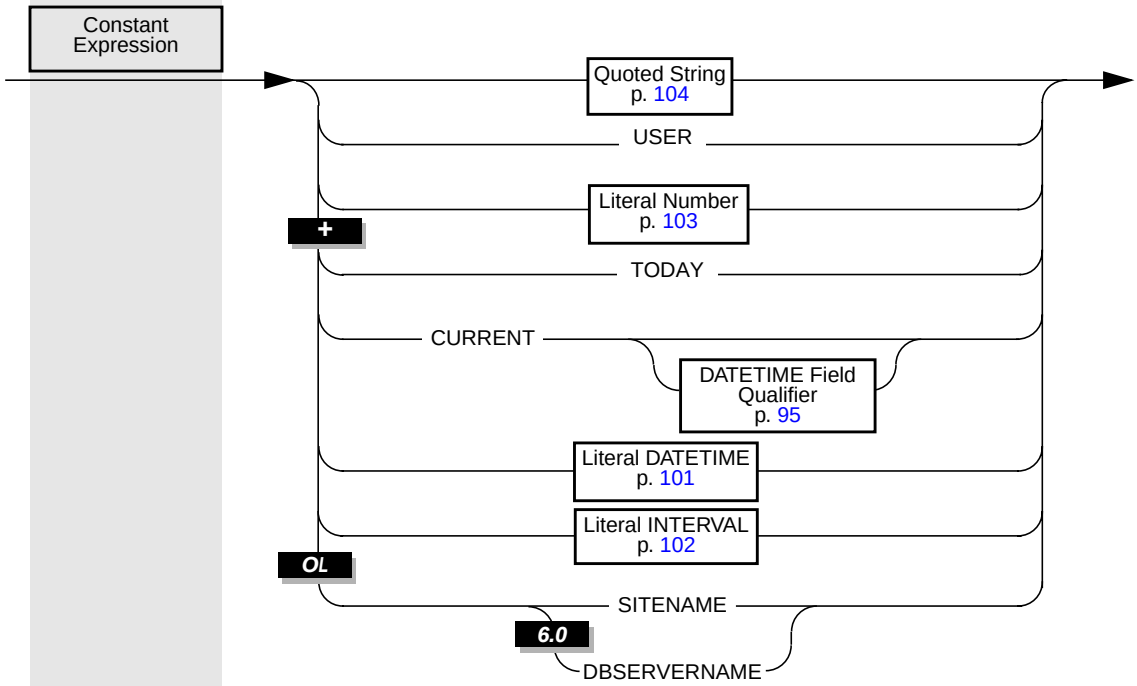
DATETIME Field Qualifier

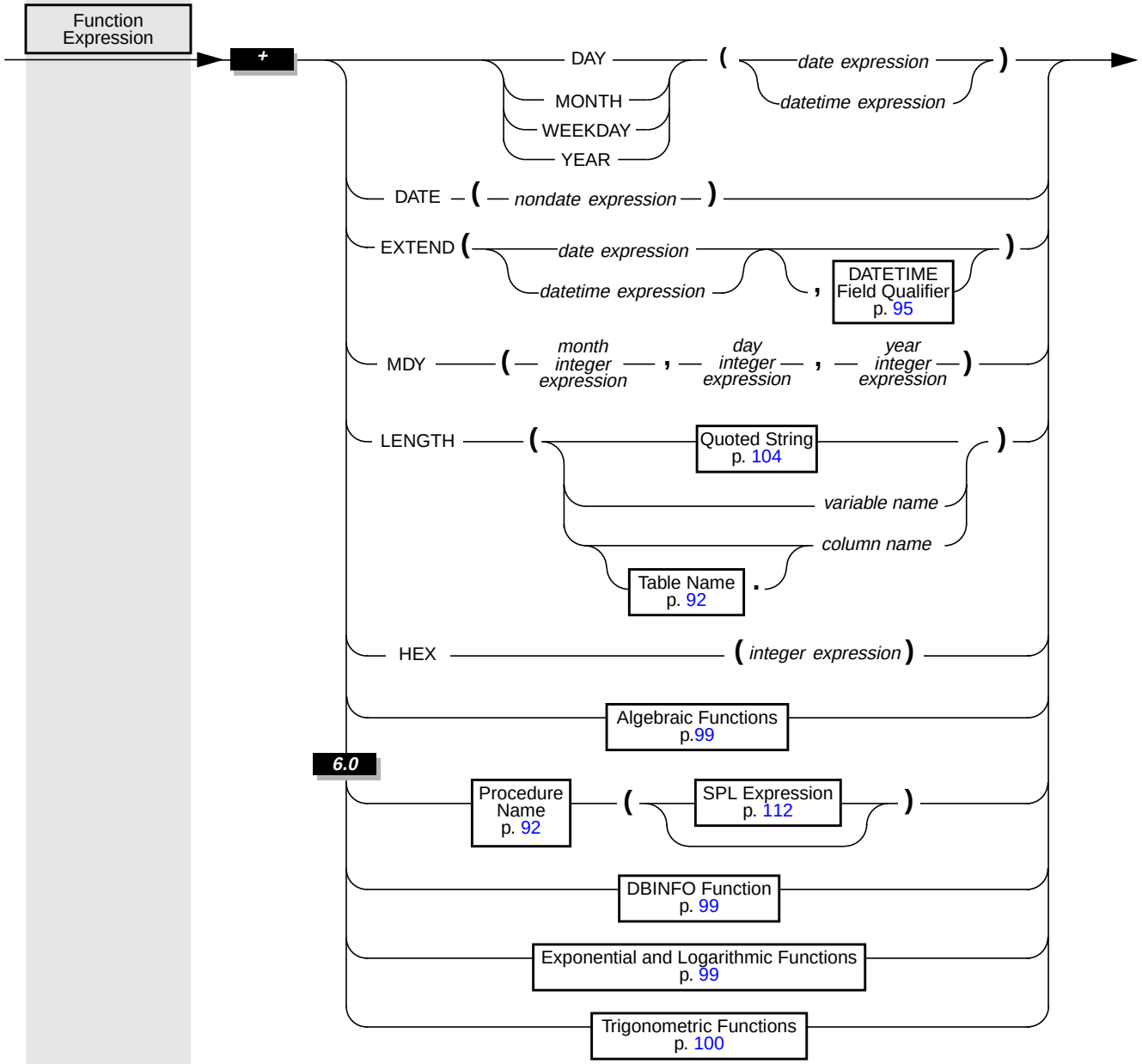




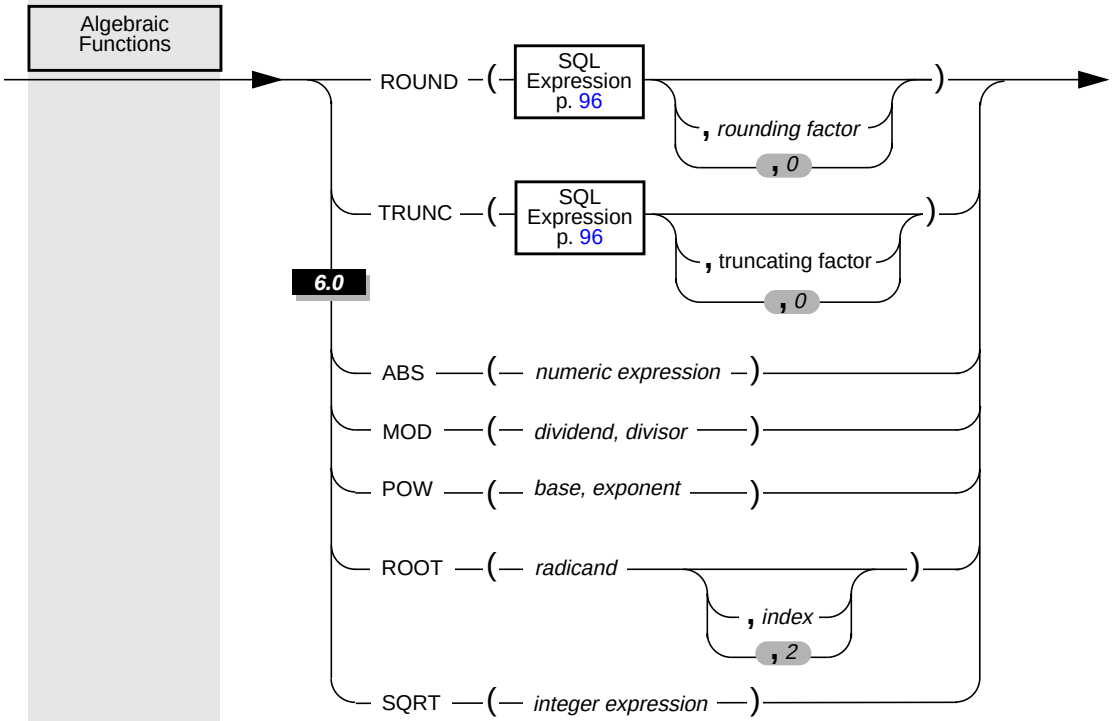


When you refer to a column whose name is identical to that of a variable, you must prefix the column name with an @ symbol; otherwise the **4GL** compiler treats it as a variable.

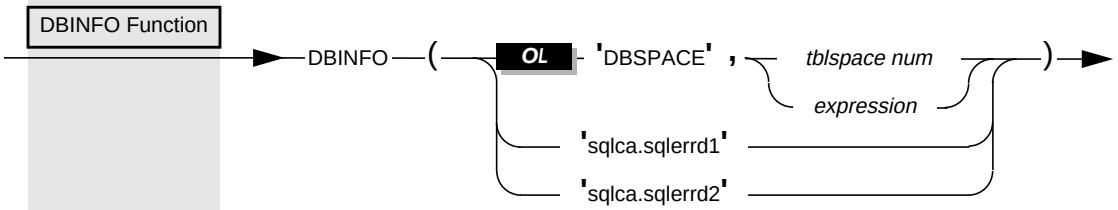

Constant Expression
Quoted String p. 104
USER
Literal Number p. 103
+
TODAY
CURRENT
DATETIME Field Qualifier p. 95
Literal DATETIME p. 101
Literal INTERVAL p. 102
OL
SITENAME
6.0
DBSERVERNAME


Function Expression
+
DAY
MONTH
WEEKDAY
YEAR
(date expression)
datetime expression
DATE (nondate expression)
EXTEND (date expression , DATETIME Field Qualifier p. 95)
datetime expression
MDY (month integer expression , day integer expression , year integer expression)
LENGTH (Quoted String p. 104)
variable name
column name
Table Name p. 92 .
HEX (integer expression)
Algebraic Functions p.99
6.0
Procedure Name p. 92 (SPL Expression p. 112)
DBINFO Function p. 99
Exponential and Logarithmic Functions p. 99
Trigonometric Functions p. 100

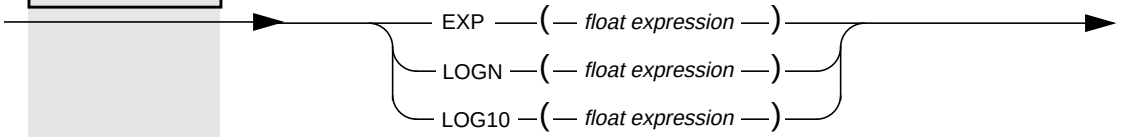
Algebraic Functions



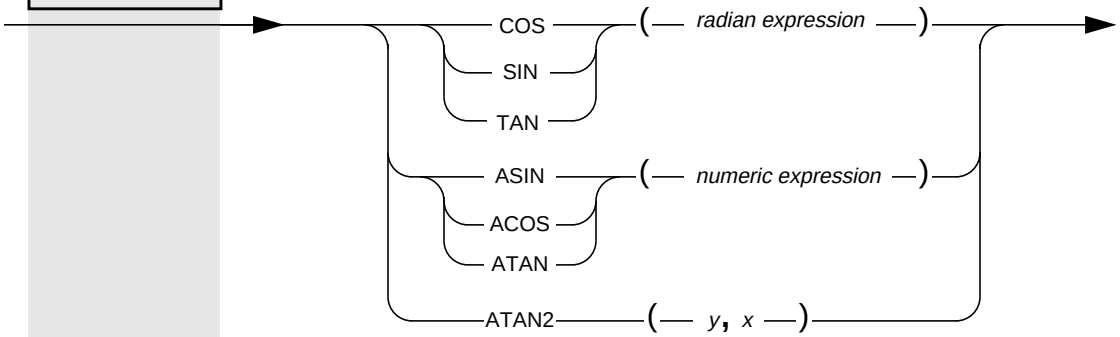
DBINFO Function



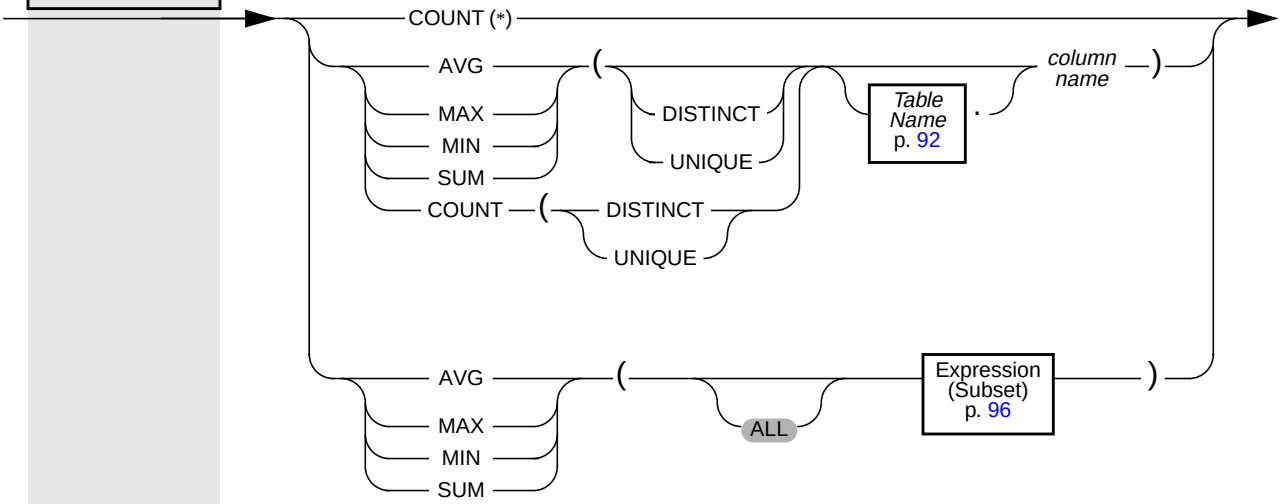
Exponential and Logarithmic Functions



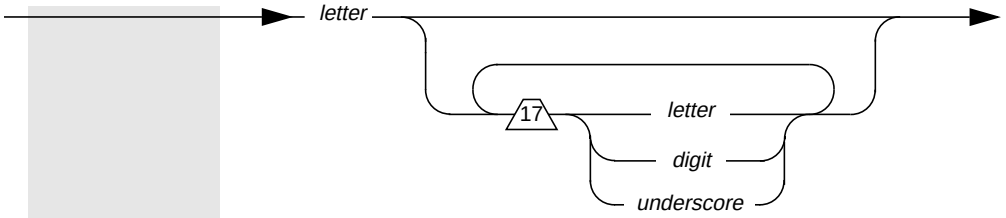
Trigonometric Functions



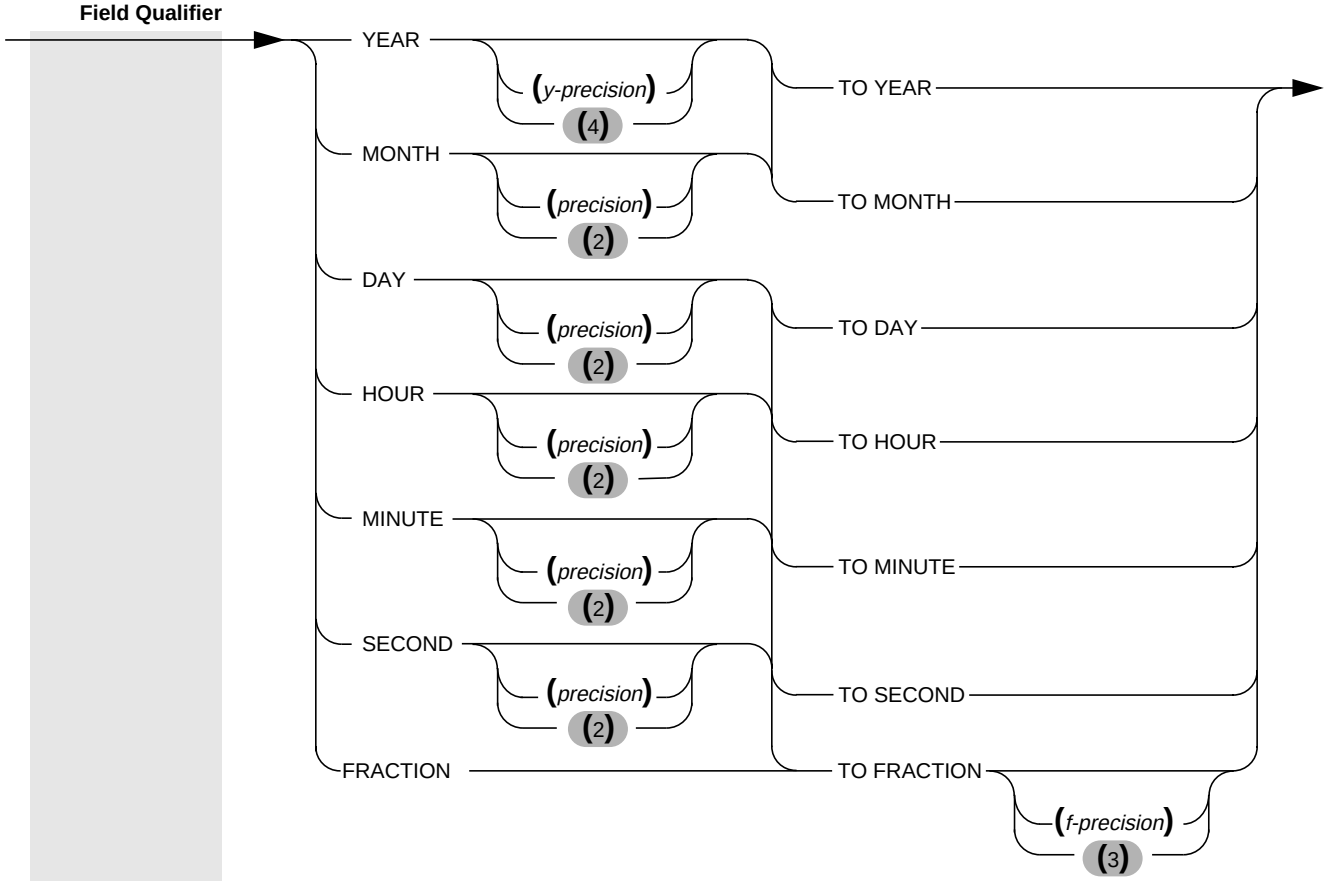
Aggregate Expression



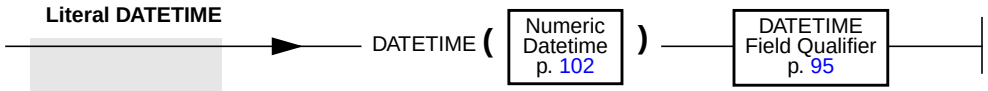
Identifier



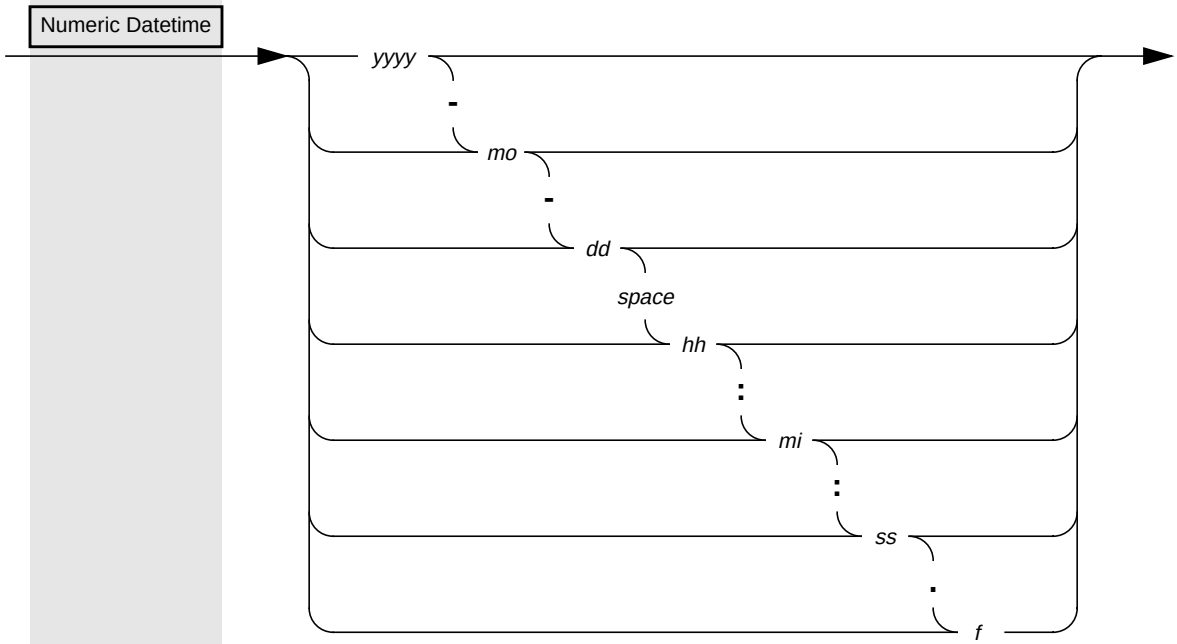
INTERVAL Field Qualifier



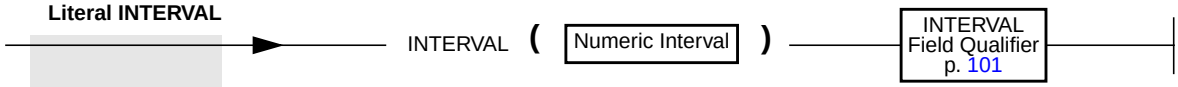
Literal DATETIME

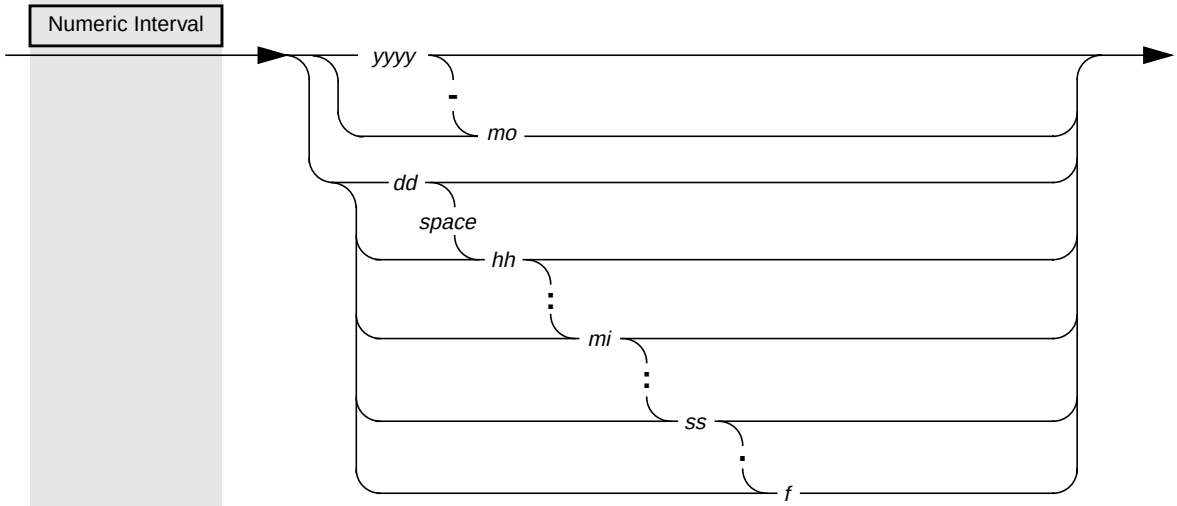


Basics
4GL
Forms
Reports
SQL
SQLCA
Debugger
Variables
Keys

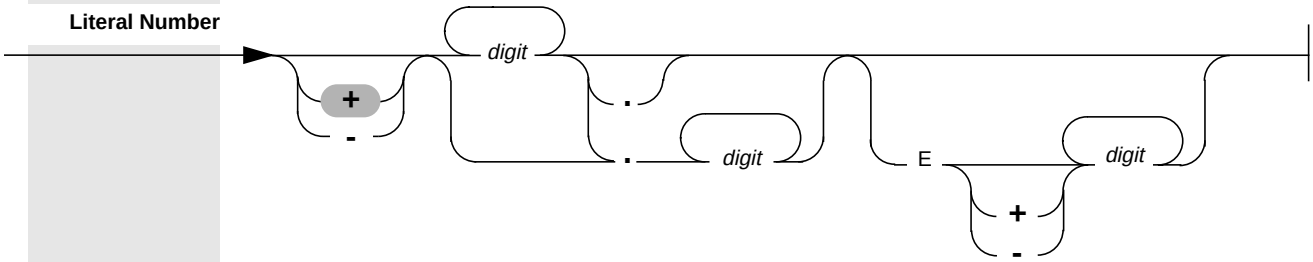


Literal INTERVAL

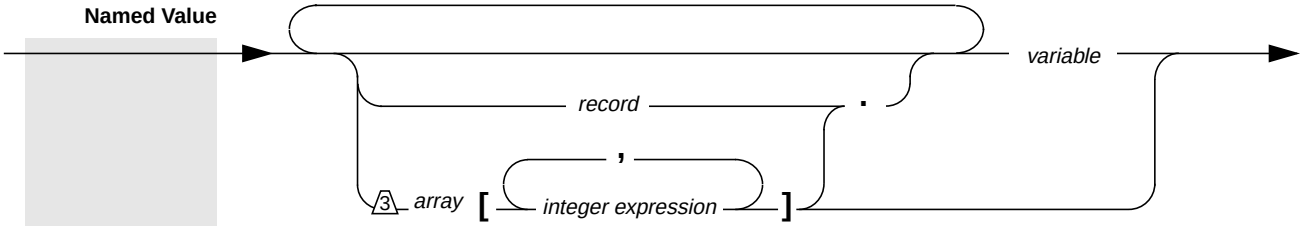




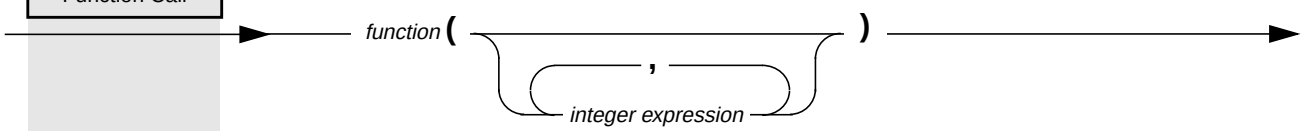
Literal Number



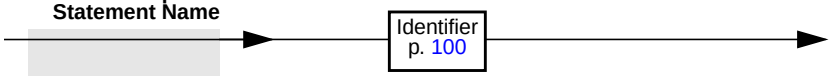
Named Value



Function Call

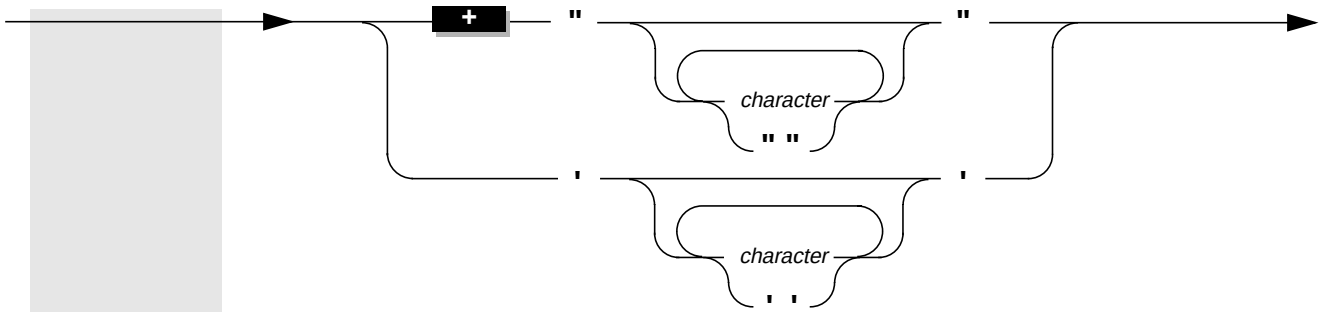


Prepared Statement Name

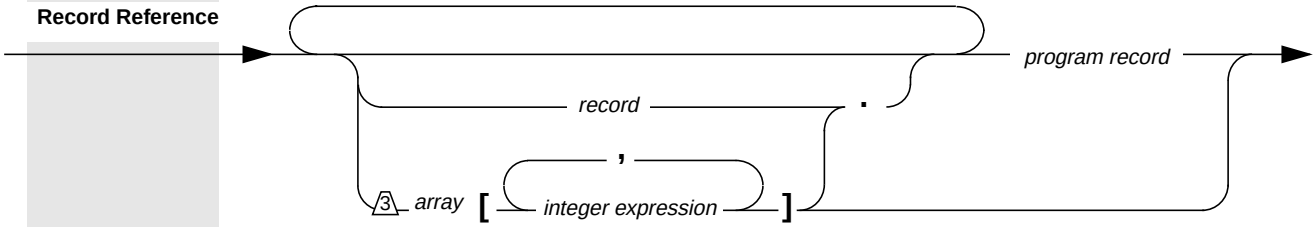


Basics
4GL
Forms
Reports
SQL
SQLCA
Debugger
Variables
Keys

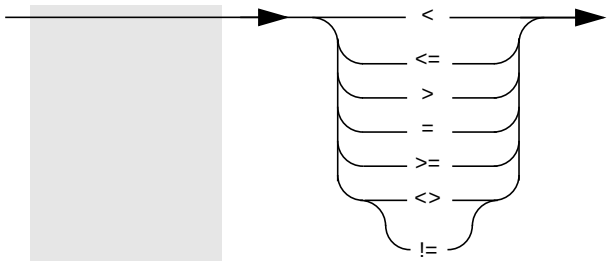
Quoted String



Record Reference



Relational Operator



Stored Procedure Language Statements

It is important to recognize that Stored Procedure Language (SPL) statements are not part of **4GL**. This means that you cannot include these statements within a **4GL** program. Doing so causes compile errors.

To create a stored procedure from a **4GL** program, do the following:

1. Put the text of the CREATE PROCEDURE statement in a file.
2. Use a PREPARE statement to prepare a CREATE PROCEDURE FROM statement that refers to the text file created in Step 1.
3. Use an EXECUTE statement to execute the prepared statement, which then compiles the stored procedure.

Refer to the *Informix Guide to SQL: Reference*, Version 6.0 for a full description of the CREATE PROCEDURE statement.

You may explicitly invoke stored procedures from within your **4GL** program by preparing and executing the following SQL statements:

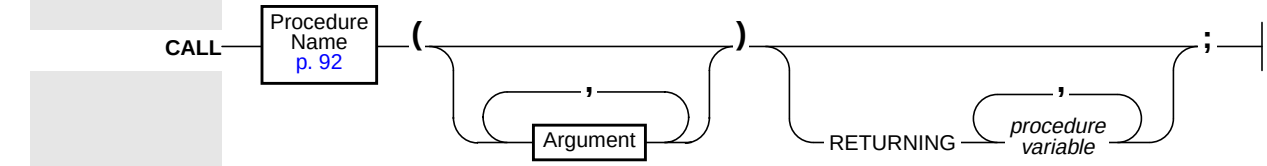
- CREATE PROCEDURE FROM
- DROP PROCEDURE
- EXECUTE PROCEDURE
- GRANT
- INSERT INTO
- REVOKE
- SET DEBUG FILE TO
- UPDATE STATISTICS

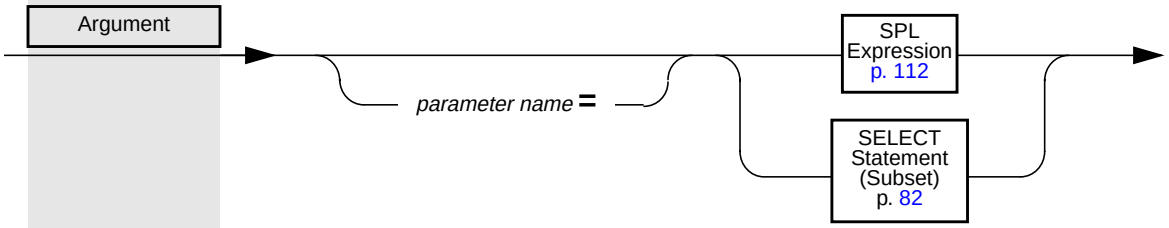
Refer to the *Informix Guide to SQL: Reference*, Version 6.0 for a description of working with dynamic SQL.

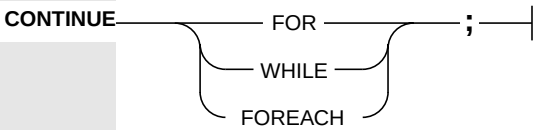
Also, you may implicitly invoke a stored procedure through a reference to that procedure within the context of an SQL expression. For example, the reference to **avg_price( )** in the following SELECT statement implicitly invokes the stored procedure having the name **avg_price**:

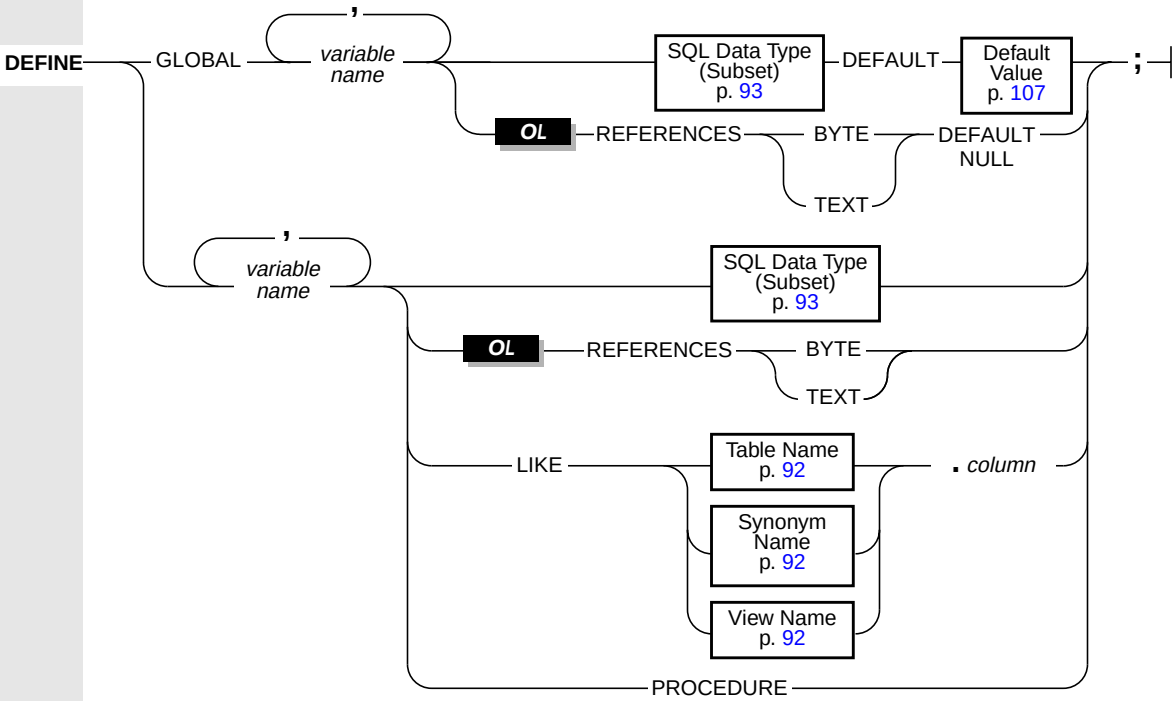
```
select
    manu_code, unit_price,
         (avg_price(1) - unit_price) variance
from stock
where stock_num = 1
```

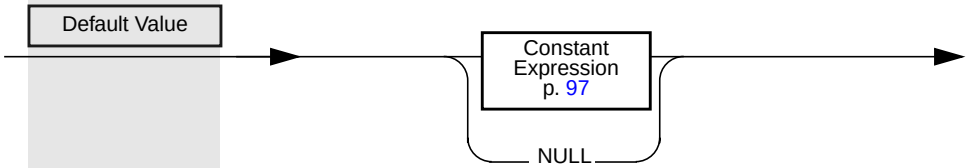
Such implicit references to stored procedures do not require the statement to be prepared.

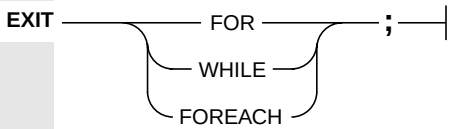

CALL
Procedure Name p. 92
(
,
Argument
)
RETURNING
,
procedure variable
;

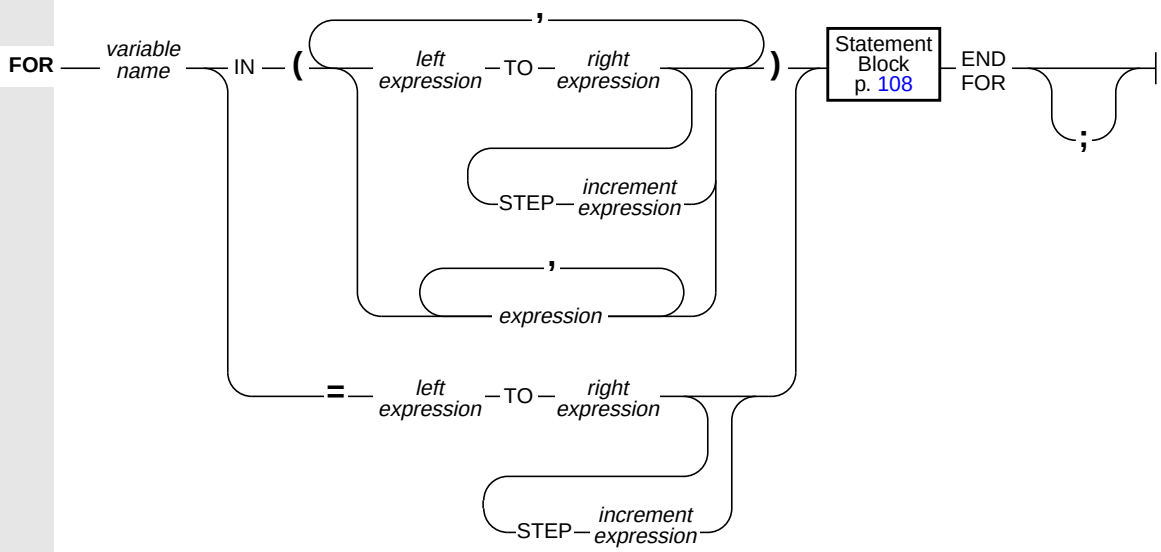

Argument
parameter name =
SPL Expression p. 112
SELECT Statement (Subset) p. 82


CONTINUE
FOR
WHILE
FOREACH
;

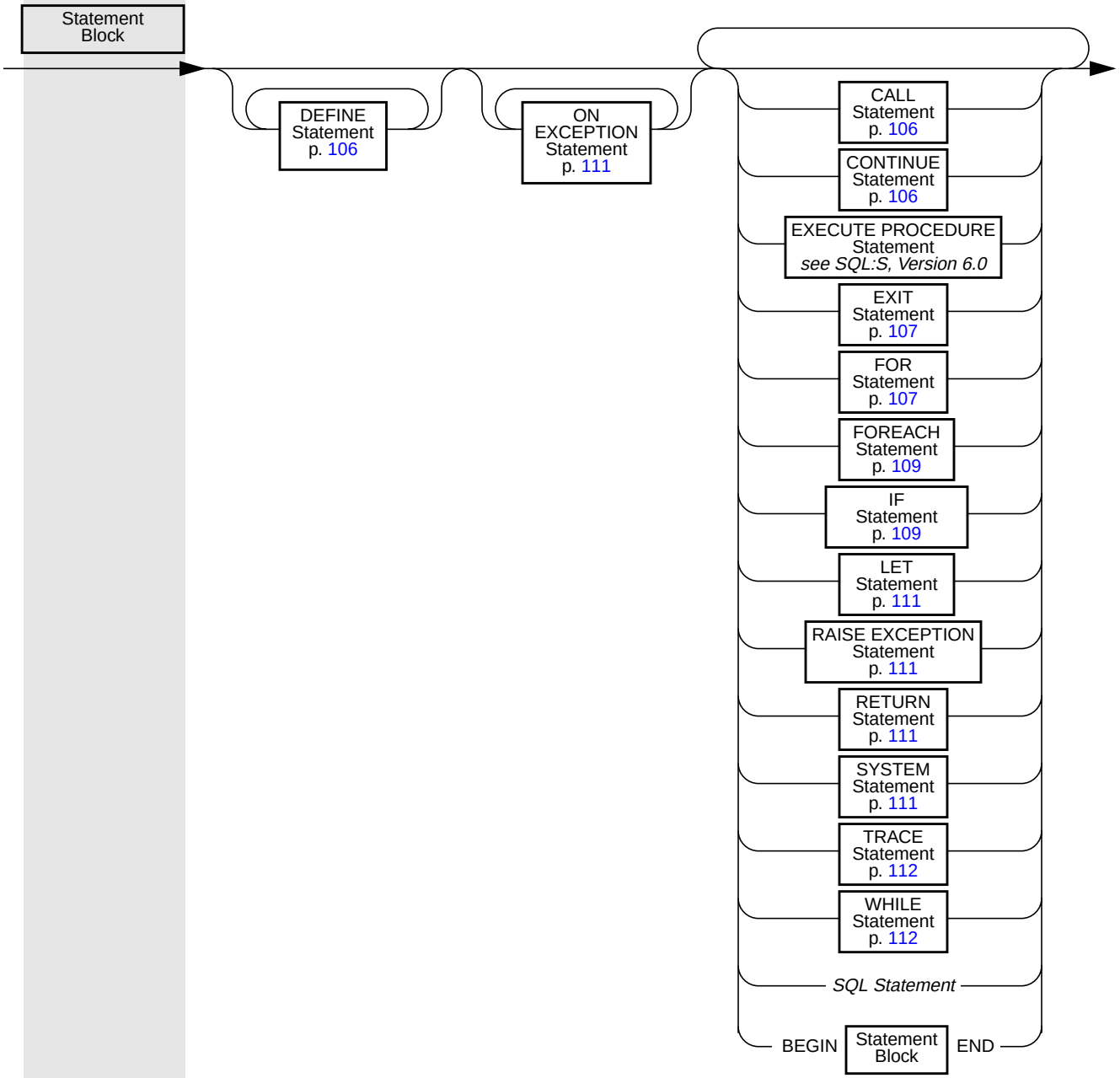

DEFINE
GLOBAL
,
variable name
SQL Data Type (Subset) p. 93
DEFAULT
Default Value p. 107
;
OL
REFERENCES
BYTE
TEXT
DEFAULT NULL
,
variable name
SQL Data Type (Subset) p. 93
OL
REFERENCES
BYTE
TEXT
LIKE
Table Name p. 92
Synonym Name p. 92
View Name p. 92
. column
PROCEDURE

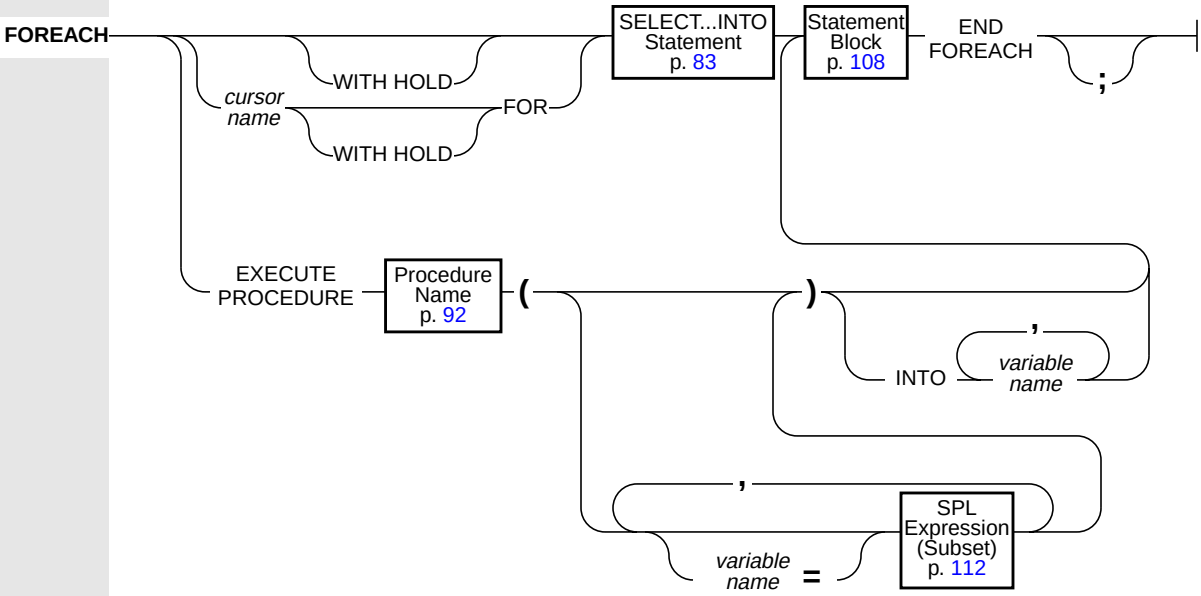


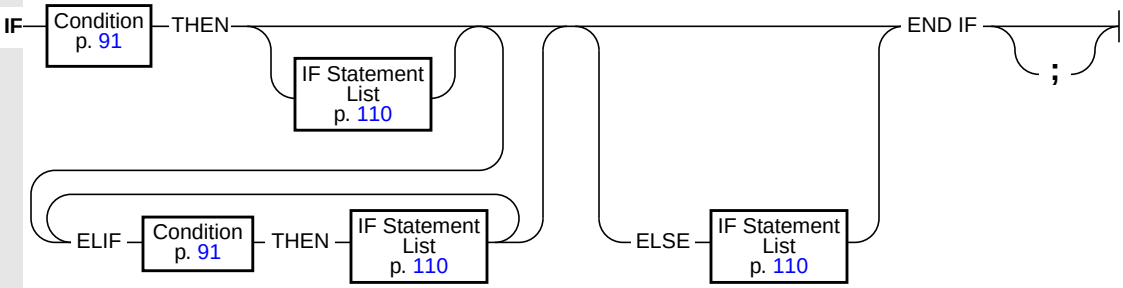


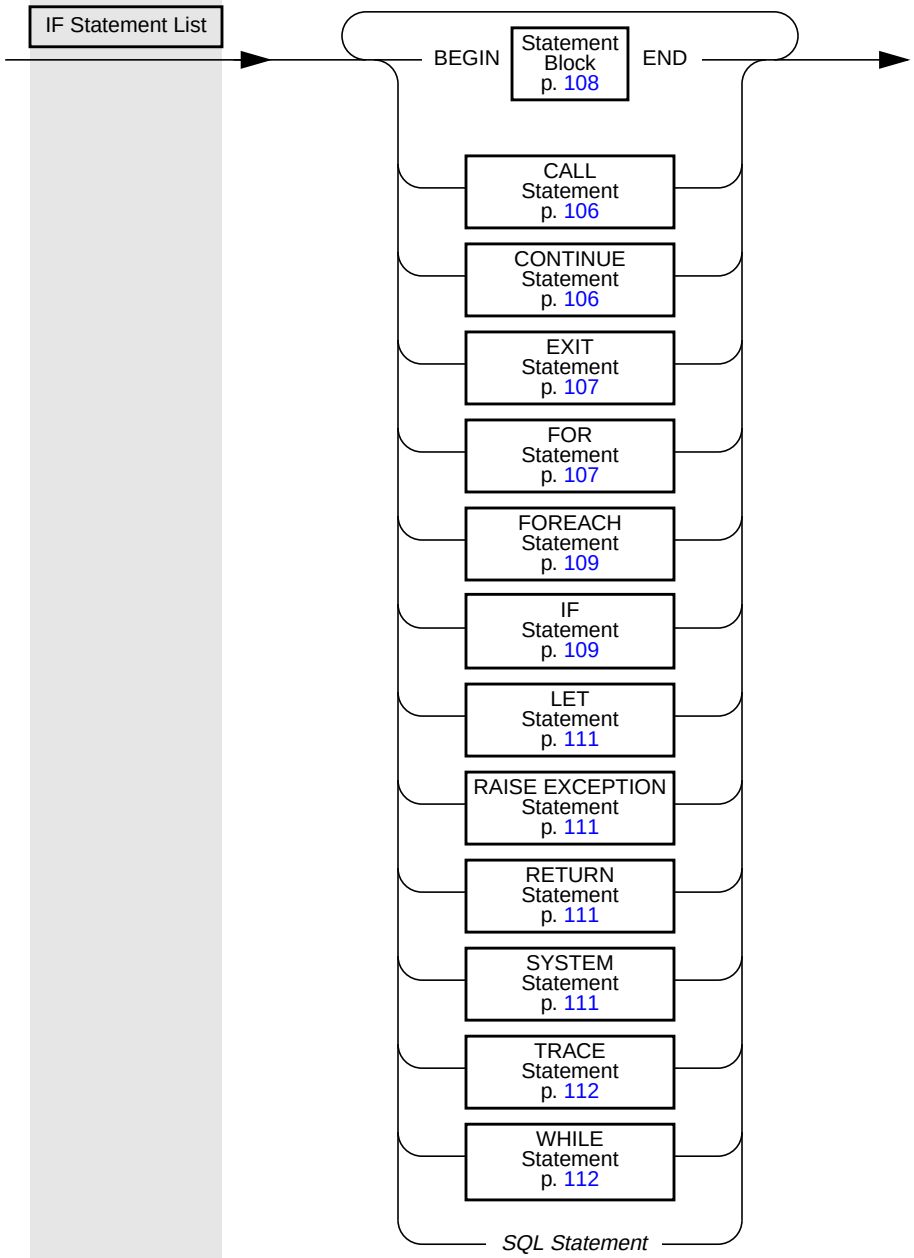


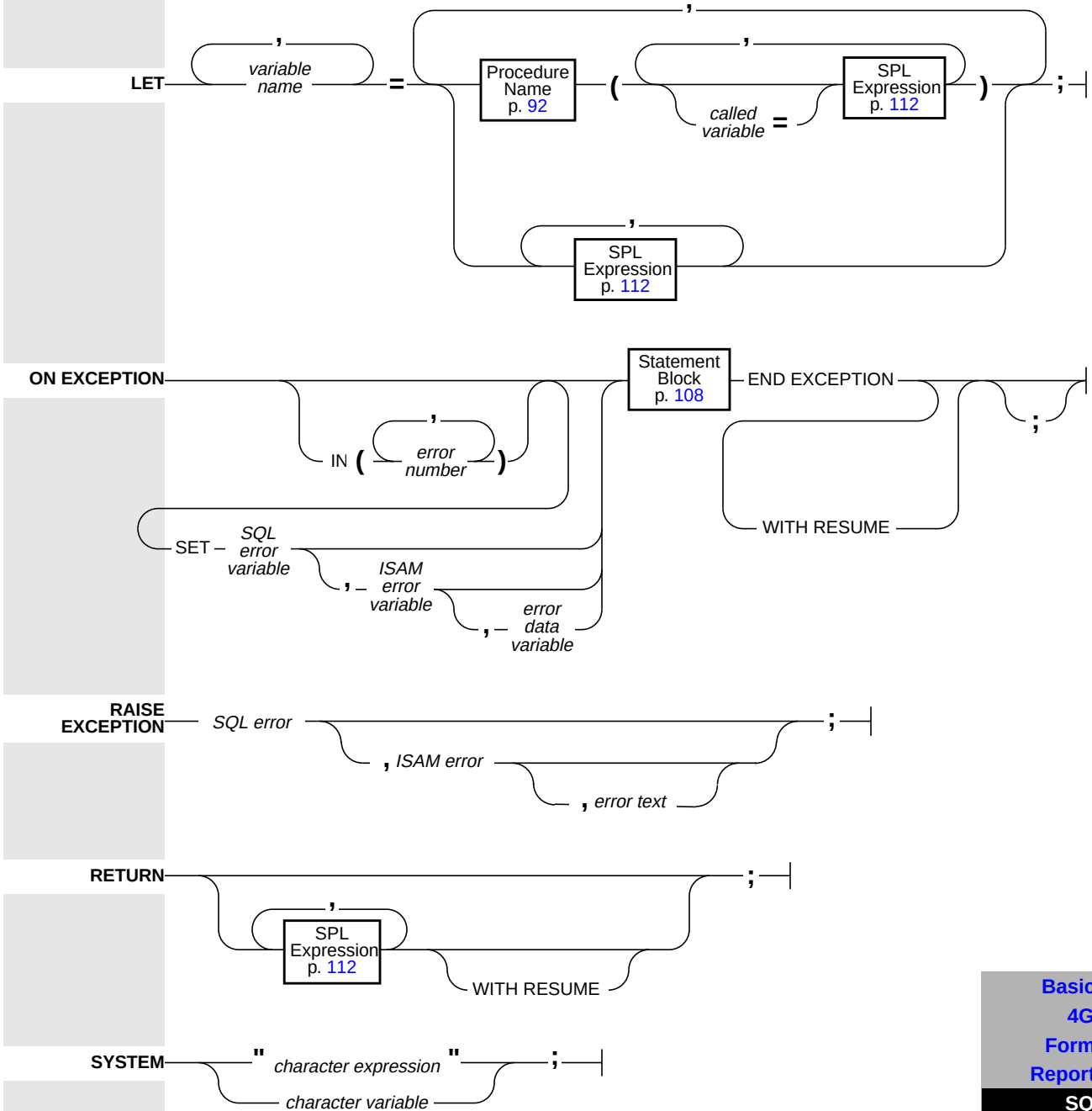
Basics
4GL
Forms
Reports
SQL
SQLCA
Debugger
Variables
Keys

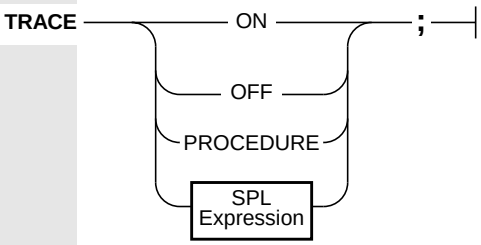

Statement Block
DEFINE Statement p. 106
ON EXCEPTION Statement p. 111
CALL Statement p. 106
CONTINUE Statement p. 106
EXECUTE PROCEDURE Statement see SQL:S, Version 6.0
EXIT Statement p. 107
FOR Statement p. 107
FOREACH Statement p. 109
IF Statement p. 109
LET Statement p. 111
RAISE EXCEPTION Statement p. 111
RETURN Statement p. 111
SYSTEM Statement p. 111
TRACE Statement p. 112
WHILE Statement p. 112
SQL Statement
BEGIN
Statement Block
END

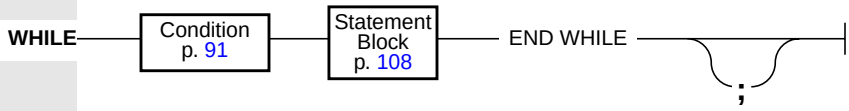

FOREACH
WITH HOLD
cursor name
FOR
WITH HOLD
SELECT...INTO Statement p. 83
Statement Block p. 108
END FOREACH
;
EXECUTE PROCEDURE
Procedure Name p. 92
(
)
INTO
,
variable name
,
variable name
=
SPL Expression (Subset) p. 112

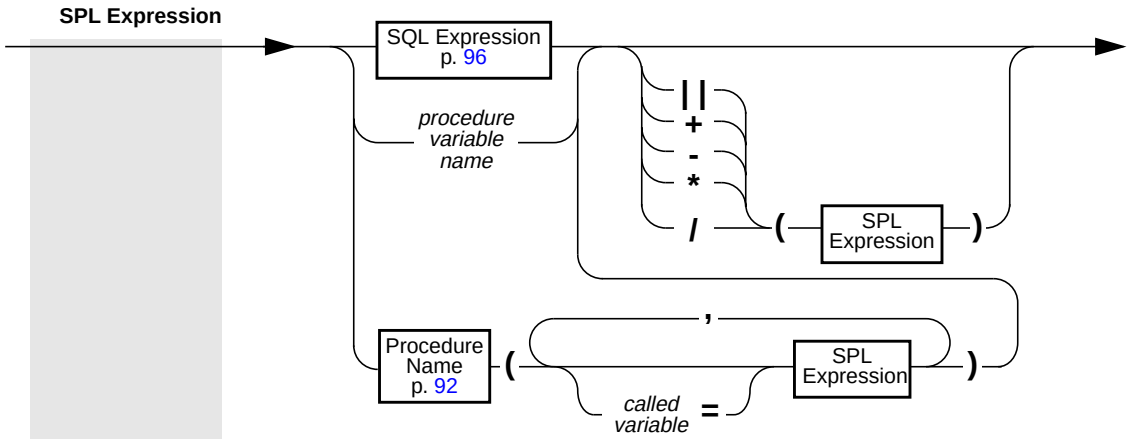

IF
Condition p. 91
THEN
IF Statement List p. 110
ELIF
Condition p. 91
THEN
IF Statement List p. 110
ELSE
IF Statement List p. 110
END IF
;


IF Statement List
BEGIN
Statement Block p. 108
END
CALL Statement p. 106
CONTINUE Statement p. 106
EXIT Statement p. 107
FOR Statement p. 107
FOREACH Statement p. 109
IF Statement p. 109
LET Statement p. 111
RAISE EXCEPTION Statement p. 111
RETURN Statement p. 111
SYSTEM Statement p. 111
TRACE Statement p. 112
WHILE Statement p. 112
SQL Statement


LET
variable name
=
Procedure Name p. 92
(
called variable
=
SPL Expression p. 112
)
;
SPL Expression p. 112
ON EXCEPTION
IN (error number)
Statement Block p. 108
END EXCEPTION
;
WITH RESUME
SET
SQL error variable
ISAM error variable
error data variable
RAISE EXCEPTION
SQL error
ISAM error
error text
;
RETURN
SPL Expression p. 112
WITH RESUME
;
SYSTEM
"
character expression
"
character variable
;


TRACE
ON
;
OFF
PROCEDURE
SPL Expression


WHILE
Condition p. 91
Statement Block p. 108
END WHILE
;


SPL Expression
SQL Expression p. 96
procedure variable name
||
+
-
*
/
(
SPL Expression
)
Procedure Name p. 92
(
,
called variable
=
SPL Expression
)

SQLCA Record

```
DEFINE  SQLCA  RECORD
 SQLCODE  INTEGER,
 SQLERRM  CHAR(71),
 SQLERRP  CHAR(8),
 SQLERRD  ARRAY [6] OF INTEGER,
 SQLAWARN CHAR (8)
END  RECORD
```

Summary of fields:

Result Code	Details of Statement Execution	Special Conditions
STATUS or SQLCA.SQLCODE	SQLCA.SQLERRD[1] through SQLCA.SQLERRD[6]	SQLCA.SQLAWARN[1] through SQLCA.SQLAWARN[8]

SQLCODE indicates the result of executing an SQL statement.
It is set as follows:

- To zero for a successful execution of most statements.
- To NOTFOUND (defined as 100) for a successfully executed query that returns zero rows or for a FETCH that seeks beyond the end of an active set. (However, in an ANSI-compliant database, when an INSERT INTO / SELECT statement or a DELETE, UPDATE, or SELECT INTO TEMP statement fails to access any rows, the value of SQLCA.SQLCODE is set to NOTFOUND rather than 0.)
- To a negative value for an unsuccessful execution.

INFORMIX-4GL sets the global variable STATUS equal to SQLCODE after each SQL statement. However, any subsequent **4GL** statement can reset STATUS.

SQLERRM contains parameters for the error message.

SQLERRP is reserved for future use.

SQLERRD is an array of six integers:

SQLERRD[1] is the estimated number of rows returned.

SQLERRD[2] is the SERIAL value returned or an error code.

SQLERRD[3] is the number of rows processed.

SQLERRD[4] is a weighted sum of disk accesses and total rows processed, the estimated CPU cost of the query.

SQLERRD[5] is the offset of error into the SQL statement.

SQLERRD[6] is the rowid of the last row processed.

SQLAWARN is a character string of length eight whose individual characters signal various warning conditions (as opposed to errors) following the execution of an SQL statement. The characters are blank if no problems or exceptional conditions are detected.

- SQLAWARN[1] is set to W if one or more of the other warning characters has been set to W. If SQLAWARN[1] is blank, you do not have to check the remaining warning characters.
- SQLAWARN[2] is set to W if one or more data items were truncated to fit into a character variable or if a DATABASE statement selected a database with transactions.
- SQLAWARN[3] is set to W if an aggregate function (SUM, AVG, MAX, or MIN) encountered a null value in its evaluation or if a DATABASE statement selected an ANSI-compliant database.
- SQLAWARN[4] is set to W if a DATABASE statement selected an **INFORMIX-OnLine Dynamic Server<Default ¶ Fo>** database or when the number of items in the *select-list* of a SELECT clause is not the same as the number of program variables in the INTO clause. In the latter case, the number of values **INFORMIX-4GL** returns is the smaller of these two numbers.
- SQLAWARN[5] is set to W if float-to-decimal conversion is used.
- SQLAWARN[6] is set to W when your program executes an Informix extension to ANSI-compliant standard syntax and the DBANSIWARN environment variable is set or the **-ansi** option is specified.
- SQLAWARN[7] is reserved for future use.
- SQLAWARN[8] is reserved for future use.

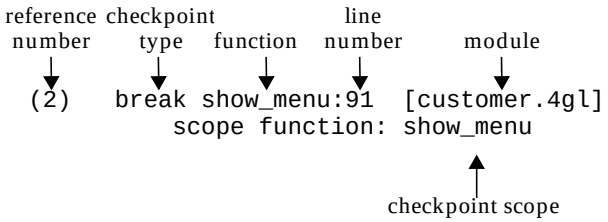
Interactive Debugger Commands

The following table lists the **Debugger** commands, options, and accelerators.

Command	Option	Shortest Form
?		?
/		/
ALIAS		al
APPLICATION	DEVICE	ap ap d
BREAK	IF	b b if
CALL		ca
CLEANUP	ALL	cl cl a
CONTINUE		co
DATABASE		da
DISABLE	ALL	di di all
DUMP	ALL GLOBALS	du du a du g
ENABLE	ALL	en en all
EXIT		ex
FUNCTIONS		f
GROW	SOURCE COMMAND	g g s g c
HELP	ALL	h h a
LET		le
LIST	BREAK TRACE	li li b li t
NOBREAK	ALL	nob nob all
NOTRACE	ALL	not not all
PRINT		p
READ		re
RUN		ru
STEP	INTO NOBREAK	s s i s n
TIME DELAY	SOURCE COMMAND	ti ti s ti c

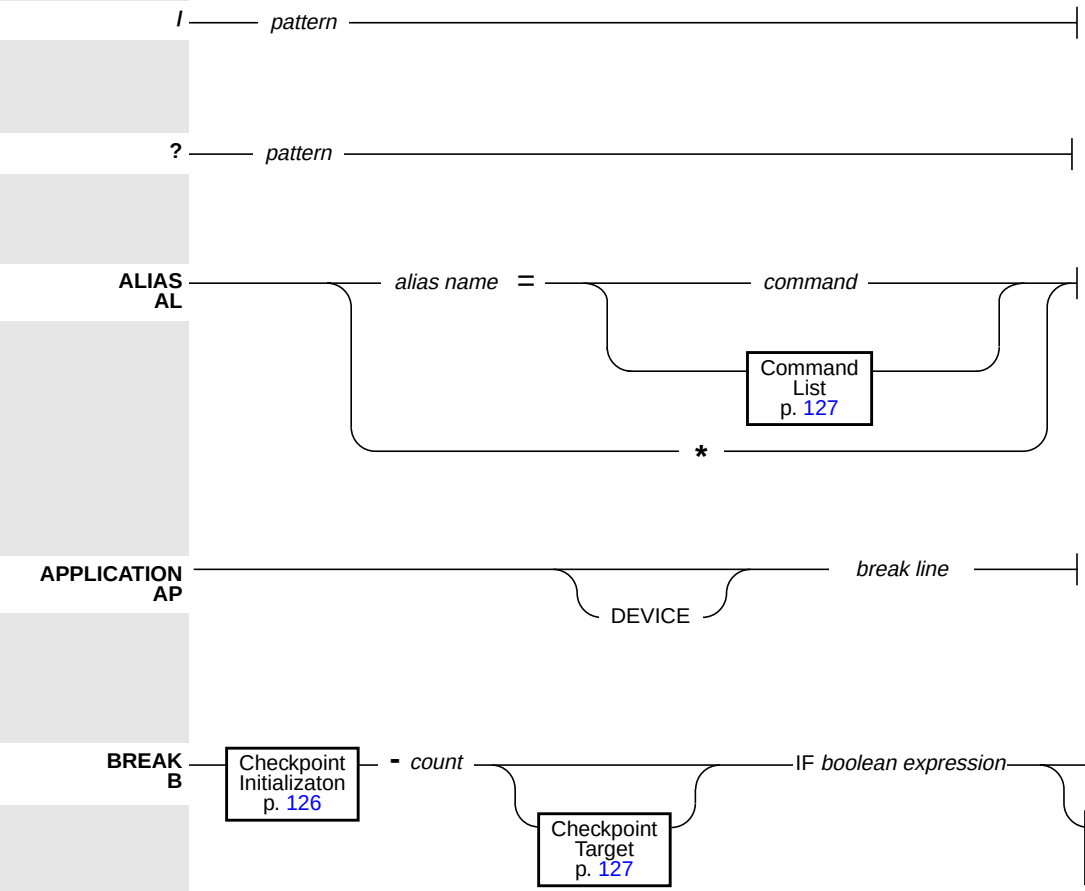
Command	Option	Shortest Form
TRACE	FUNCTIONS	tr tr functions
TURN	ON OFF	tu tu on tu of
USE		us
VARIABLE	ALL GLOBALS	va va all va globals
VIEW	LINE	vi vi l
WHERE		wh
WRITE	ALIAS BREAK TRACE	wr wr a wr b wr t

After setting a breakpoint (or tracepoint), the Output text region displays:



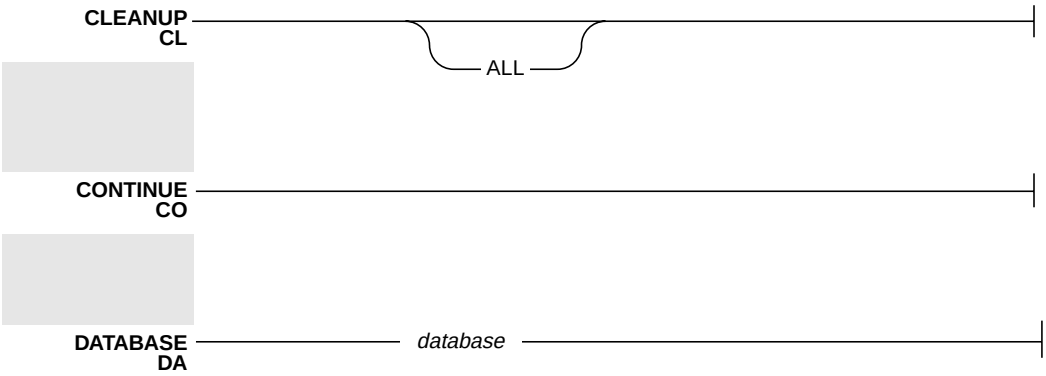
Command-Line Syntax

To:	Type:
Escape	! command
Interrupt	CONTROL-D or Del key
Redraw	CONTROL-R
Screen	CONTROL-P
Toggle	CONTROL-T
Search for characters	/ pattern ? pattern



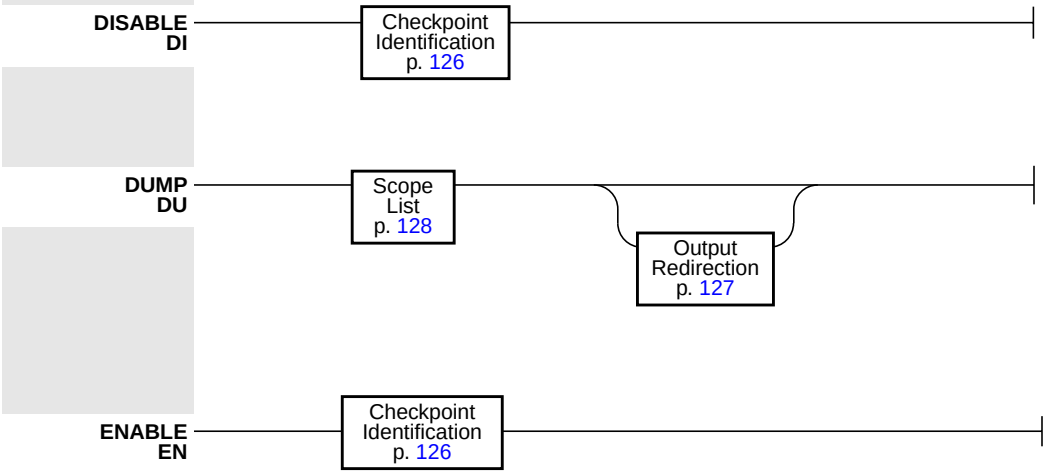
These built-in operators can be used in a *boolean expression*:

CURRENT	ENTEND( )	TODAY
DATE( )	INTERVAL( )	UNITS
DATETIME( )	MDY( )	WEEKDAY( )
DAY( )	MONTH( )	YEAR( )

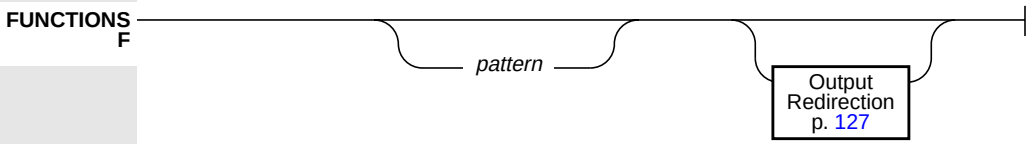


To select a remote database, use one of the following formats:

Remote Database Engine	Format of *database* name
INFORMIX-OnLine Dynamic Server	*database@servername* or *//servername/database*
INFORMIX-SE	*//servername/path/database*



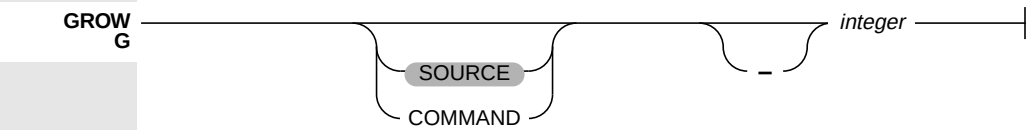
Basics
4GL
Forms
Reports
SQL
SQLCA
Debugger
Variables
Keys

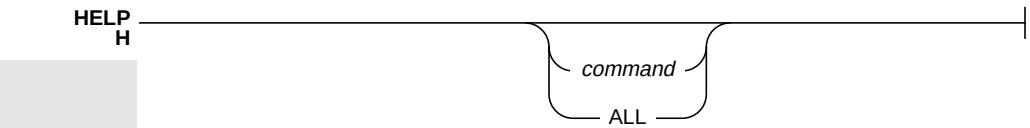


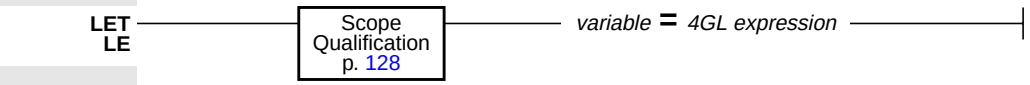
A *pattern* is a string of no more than 50 characters and blanks or up to 80 characters if enclosed in quotation marks.

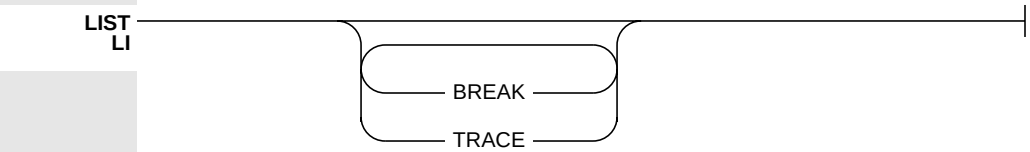
You can use the following wildcard characters within the pattern:

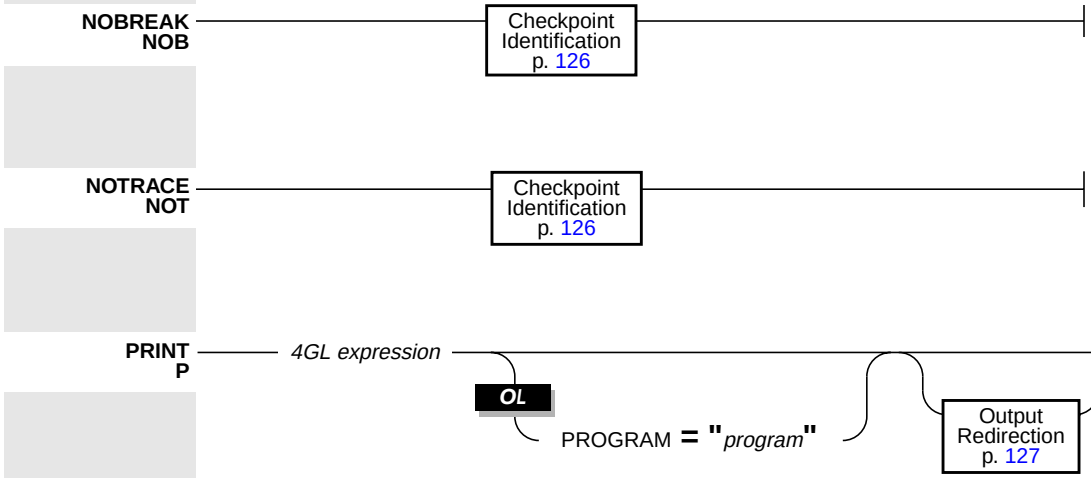
Pattern	Matches
?	any single character
*****	zero or more characters
[*characters*]	one or more unseparated characters
[*character-character*]	characters within the range in ASCII collating sequence





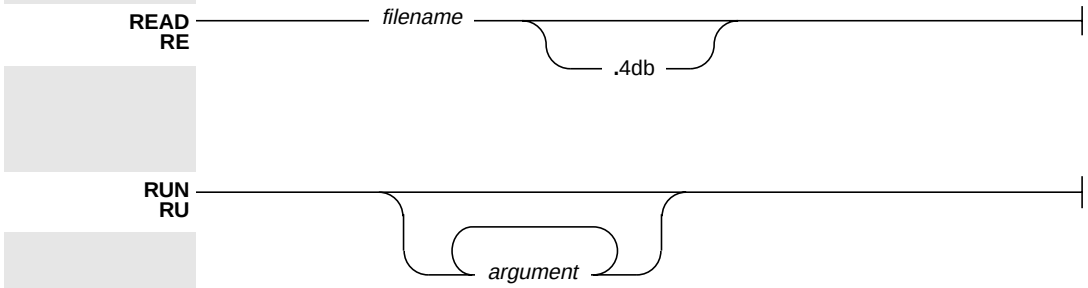




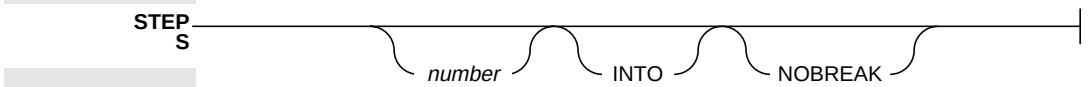


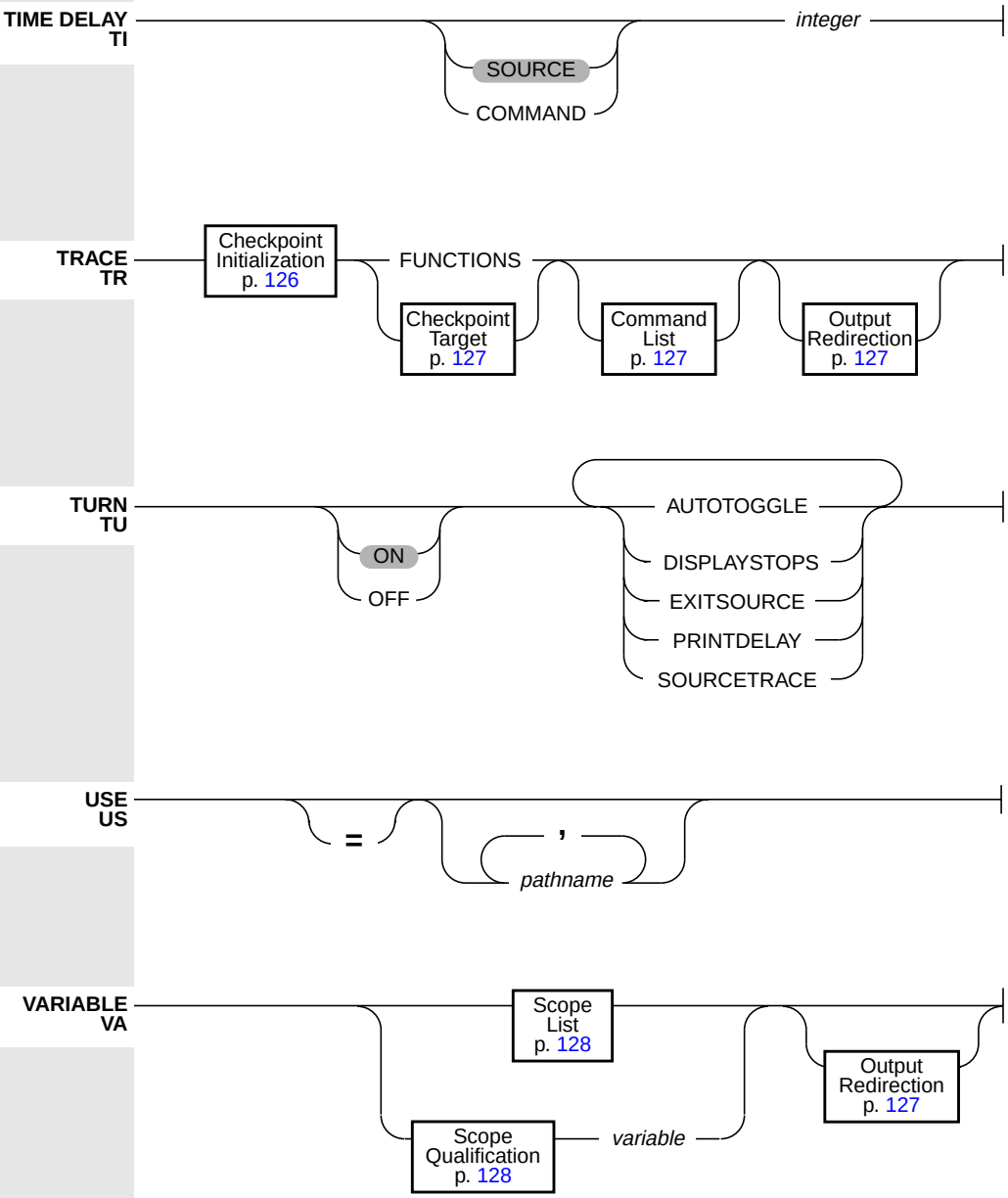
These built-in operators can be used within a *4GL expression*:

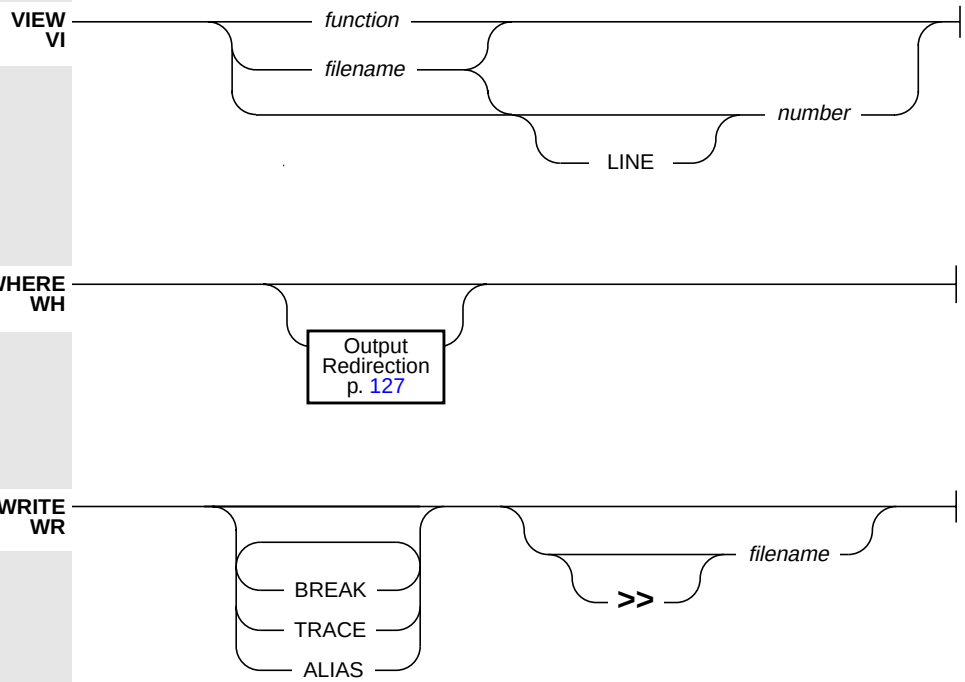
CURRENT	ENTEND( )	TODAY
DATE( )	INTERVAL( )	UNITS
DATETIME( )	MDY( )	WEEKDAY( )
DAY( )	MONTH( )	YEAR( )



To pass arguments to the p-code runner, include them on the **Debugger** command line.

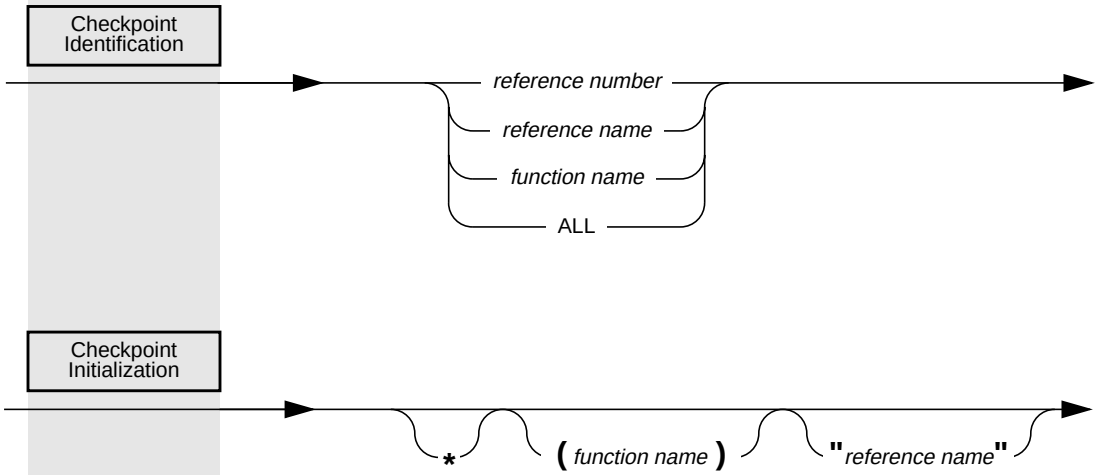



TIME DELAY
TI
integer
SOURCE
COMMAND
TRACE
TR
Checkpoint Initialization p. 126
FUNCTIONS
Checkpoint Target p. 127
Command List p. 127
Output Redirection p. 127
TURN
TU
AUTOTOGGLE
ON
OFF
DISPLAYSTOPS
EXITSOURCE
PRINTDELAY
SOURCETRACE
USE
US
=
'
pathname
VARIABLE
VA
Scope List p. 128
Scope Qualification p. 128
variable
Output Redirection p. 127



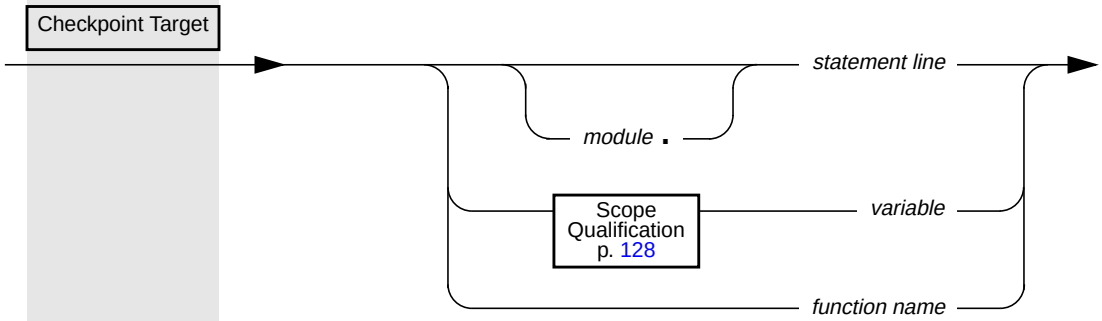
Basics
4GL
Forms
Reports
SQL
SQLCA
Debugger
Variables
Keys

4GL Interactive Debugger Command Segments



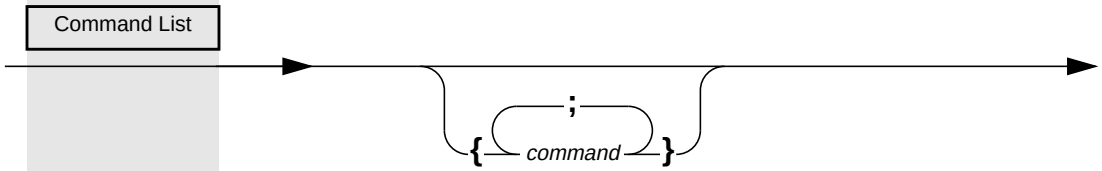
A *reference name* must:

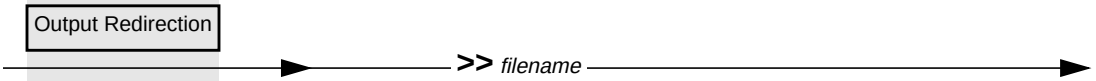
- Be unique among other checkpoint names.
- Begin with an alphabetic character.
- Contain only letters, numbers, or the underscore (_) character.



Type	Valid Entities	Triggered When
statement	executable statements within **4GL** functions	Execution completes previous statement.
variable	†active variables	Variable is assigned a different value.
function	**4GL** functions (including **MAIN**), **4GL** reports, **ESQL/C** functions, C functions	Function is called.

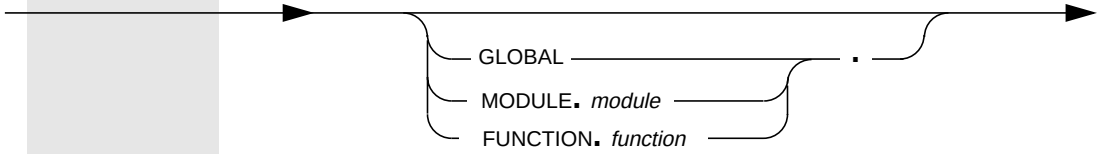
†You cannot set a checkpoint on an entire array or record structure.





Scope List

Scope Qualification



Environment Variables

INFORMIX Environment Variable	Restrictions	*INFORMIX-4GL Reference* Page
DBANSIWARN		D-8
DBDATE		D-9
DBDELIMITER		D-11
DBEDIT		D-11
DBFORMAT		D-14
DBLANG		D-18
DBMONEY		D-21
DBPATH		D-23
DBPRINT		D-26
DBREMOTECMD	**OnLine** only	D-27
DBSPACETEMP	**OnLine** only	D-28
DBTEMP	**SE** only	D-29
DBUPSPACE		D-29
ENVIGNORE		D-30
INFORMIXCONRETRY		D-30
INFORMIXCONTIME		D-31
INFORMIXDIR		D-32
INFORMIXSERVER		D-33
INFORMIXSHMBASE	**OnLine** only	D-33
INFORMIXSTACKSIZE	**OnLine** only	D-34
INFORMIXTERM		D-34
ONCONFIG	**OnLine** only	D-36
PSORT_DBTEMP	**OnLine** only	D-36
PSORT_NPROCS	**OnLine** only	D-37
SQLEXEC		D-38
SQLRM	SQL APIs only	D-38
SQLRMDIR	SQL APIs only	D-39

NLS Environment Variable	*INFORMIX-4GL Reference* Page
COLLCHAR	E-18
DBAPICODE	E-23
DBNLS	E-16
LANG	E-25
LC_COLLATE	E-27
LC_CTYPE	E-29
LC_MONETARY	E-31
LC_NUMERIC	E-35

UNIX Environment Variable	*INFORMIX-4GL Reference* Page
PATH	D-40
TERM	D-41
TERMCAP	D-41
TERMINFO	D-42

Default Key Assignments

Logical command keys at runtime and their default assignments:

Key Name	Purpose of Key	Default Keystroke
Accept	Selects the current menu option in a MENU statement; terminates input during CONSTRUCT, INPUT and INPUT ARRAY; terminates DISPLAY ARRAY.	Escape
Interrupt	Represents the external interrupt signal; available when interrupts are deferred with the DEFER statement.	CONTROL-C
Insert	Requests insertion of a new line during INPUT ARRAY, starting execution of a BEFORE INSERT block.	F1
Delete	Requests deletion of the current line during INPUT ARRAY, starting execution of a BEFORE DELETE block.	F2
Next	Causes scrolling to the next page (group of lines) during DISPLAY ARRAY and INPUT ARRAY.	F3
Previous	Causes scrolling to the previous page (group of lines) during DISPLAY ARRAY and INPUT ARRAY.	F4
Help	Starts the display of the specified HELP message from the current help file.	CONTROL-W
Quit	Terminates the program unless DEFER QUIT is specified.	CONTROL-\

Effect of special keys on interactive **4GL** statements and within **4GL** menus:

Key Name	Use in CONSTRUCT, INPUT, and INPUT ARRAY	Use in MENU
CONTROL-A	Switches between overtype and insert modes.	None.
CONTROL-D	Deletes from the cursor to the end of the field.	None.
CONTROL-H (backspace)	During text entry, moves the cursor left one position (nondestructive backspace).	Moves highlight to next option left.
CONTROL-I or TAB	Cursor moves to next field; except in a WORDWRAP field, inserts a tab or skips to a tab depending on mode.	None.
CONTROL-J (Linefeed)	Cursor moves to next field; except in a WORDWRAP field, inserts a newline or moves down one line depending on mode.	Moves the highlight to the next option right.
CONTROL-L	During text entry, moves the cursor right one position.	Moves the highlight to the next option right.

Key Name	Use in CONSTRUCT, INPUT, and INPUT ARRAY	Use in MENU
CONTROL-M or RETURN	Completes entry of the current field. Cursor moves to next field if any; else same as Accept.	Accepts the option that is currently highlighted.
CONTROL-N	Cursor moves to beginning of current field.	None.
CONTROL-R	Causes the screen to be redrawn.	Causes the screen to be redrawn.
CONTROL-X	Deletes the character under the cursor.	None.
Left Arrow	Same as Backspace.	Same as Backspace.
Right Arrow	Same as CONTROL-L.	Same as CONTROL-L.
Up Arrow	Moves to previous field; except in a WORDWRAP field moves up one line in field and in an INPUT ARRAY moves to the corresponding field in the previous row.	Moves the highlight to the next option left.
Down Arrow	Moves to next field; except in a WORDWRAP field moves down one line in field and in an INPUT ARRAY moves to the corresponding field in the next row.	Moves the highlight to the next option right.

Default function key assignments in the **Debugger**:

Default Key	Equivalent Command
F1	HELP
F2	STEP
F3	STEP INTO
F4	CONTINUE
F5	RUN
F6	LIST BREAK TRACE
F7	LIST
F8	DUMP
F9	EXIT