INFORMIX–4GL

SQL-BASED
APPLICATION DEVELOPMENT LANGUAGE

REFERENCE MANUAL
Volume Two

June 1987

Part No. 200-501-0003-0
Rev. A
PRINTED IN U.S.A.

INFORMIX-4GL
Version 1.10

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

NOTE: ANY REFERENCES IN DOCUMENTATION TO "RELATIONAL DATABASE, INC." OR "RDS" SHALL MEAN INFORMIX SOFTWARE, INC.

Table of Contents

VOLUME ONE

Introduction

Chapter 1. INFORMIX-4GL Programming

Chapter 2. Using RDSQL

Chapter 3. Form Building and Compiling

Chapter 4. Report Writing

Chapter 5. 4GL Function Library

Chapter 6. Programmer's Environment

VOLUME TWO

Chapter 7. INFORMIX-4GL Statement Syntax

Appendix A. Demonstration Database and Application

Appendix B. System Catalogs

Appendix C. Environment Variables

Appendix D. Reserved Words

Appendix E. The *upscol* Utility

Appendix F. The *bcheck* Utility

Appendix G. The *mkmessage* Utility

Appendix H. The *sqlconv* Utility

Appendix I. The *dbupdate* Utility

Chapter 7

INFORMIX-4GL Statement Syntax

Chapter 7 Table of Contents

Types of Statements

Twelve types of **INFORMIX-4GL** statements are available:

- Program Definition Statements

FUNCTION	REPORT
MAIN	

- Variable Definition Statements

DEFINE	GLOBALS

- Assignment Statements

INITIALIZE	LET

- Program Flow Statements

CALL	IF
CASE	LABEL
CONTINUE	RETURN
DEFER	RUN
EXIT	SLEEP
FOR	WHENEVER
FOREACH	WHILE
GOTO	

- Screen Interaction Statements

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

- Report Statements

FINISH REPORT	START REPORT
OUTPUT TO REPORT	

- Data Definition Statements

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

- Data Manipulation Statements

DELETE	SELECT
INSERT	UPDATE

- Cursor Manipulation Statements

CLOSE	FLUSH
DECLARE	OPEN
FETCH	PUT

- Dynamically Defined Statements

EXECUTE	PREPARE

- Data Access Statements

GRANT	SET LOCK MODE
LOCK TABLE	UNLOCK TABLE
REVOKE	

- Data Integrity Statements

BEGIN WORK	CREATE AUDIT
COMMIT WORK	DROP AUDIT
ROLLFORWARD DATABASE	RECOVER TABLE
ROLLBACK WORK	
START DATABASE	VALIDATE

Definition of Statements

The following section describes the INFORMIX-4GL statements. The statements appear in alphabetical order.

ALTER INDEX

Overview

Use the ALTER INDEX statement to cluster a table in the order of an existing index or to release an index from the clustering attribute.

Syntax

ALTER INDEX *index-name* TO [NOT] CLUSTER

Explanation

ALTER INDEX	are required keywords.
index-name	is the **RDSQL** identifier of an existing index.
TO	is a required keyword.
NOT	is an optional keyword.
CLUSTER	is a required keyword.

Notes

1. The TO CLUSTER option causes **RDSQL** to reorder the rows in the physical table to agree with the order of *index-name*. Reordering causes the entire file to be rewritten and requires sufficient disk space to maintain two copies of the table. The process may take a long time.

2. Since there can be only one clustered index per table, you must use the NOT option to release the cluster attribute from one index before assigning it to another. The NOT option does not affect the physical table; it merely drops the cluster attribute on *index-name* from the system catalogs.

3. When **RDSQL** executes ALTER INDEX with the TO CLUSTER option, it locks the table in EXCLUSIVE mode. If some other process is using the table to which *index-name* belongs, **RDSQL** cannot execute ALTER INDEX with the TO CLUSTER option and returns an error.

4. If you execute this statement within a transaction, it cannot be rolled back. It can, however, be rolled forward.

5. As rows are added and deleted you can expect the benefit of an earlier clustering to disappear. You can recluster the table by issuing another ALTER INDEX TO CLUSTER statement on the clustered index.

6. You do not need to drop a cluster index before issuing another ALTER INDEX TO CLUSTER statement on a currently clustered index.

Examples

The following example creates two indexes on the **orders** table and clusters the physical table in ascending order on the **customer_num** column. Later, the example clusters the physical table in ascending order on the **order_num** column.

```
CREATE UNIQUE INDEX ix_ord
       ON orders (order_num)

CREATE CLUSTER INDEX ix_cust
       ON orders (customer_num)

. . .

ALTER INDEX ix_cust TO NOT CLUSTER

ALTER INDEX ix_ord TO CLUSTER
```

Related Statements

CREATE INDEX

ALTER TABLE

Overview

Use the ALTER TABLE statement to add a column to a table, to delete a column from a table, or to modify the data type of a column.

Syntax

```
ALTER TABLE table-name
        {ADD (newcol-name newcol-type
                [BEFORE oldcol-name], ...)
        |DROP (oldcol-name, ...)
        |MODIFY (oldcol-name newcol-type [NOT NULL] , ...)}, ...
```

Explanation

ALTER TABLE	are required keywords.
table-name	is the name of an existing table.
ADD	is a keyword you use to add a column to *table-name*.
newcol-name	is the name of the column you want to add.
newcol-type	is either the data type of the column you are adding or of the column whose data type you are modifying.
BEFORE	is an optional keyword you use to indicate where you want *newcol-name* placed in the list of columns. The default is at the end of the list of columns.
oldcol-name	is the name of an existing column.

DROP	is a keyword you use to drop a column from *table-name*.
MODIFY	is a keyword you use to change the data type of an existing column.
NOT NULL	are optional keywords.

Notes

1. You may use one or more of the ADD clause, the DROP clause, or the MODIFY clause and may place them in any order. Use a comma to separate the clauses. The actions are performed in the order specified. If any of the actions fail, the entire operation is canceled.

2. You can only add a SERIAL column to an empty table.

3. You cannot use a NOT NULL clause in an ALTER TABLE statement when you add a new column, since **RDSQL** inserts a NULL value into the new column for all existing rows.

4. If a column is currently designated as NOT NULL, you can modify it to allow NULL values by specifying the same column name and data type in the MODIFY clause of an ALTER TABLE statement. If a column allows NULL values but currently does not contain any NULLs, you can change it to a NOT NULL column by specifying the same column name and data type in the MODIFY clause along with the NOT NULL keywords.

5. You must own *table-name*, have DBA privilege, or be granted ALTER permission to use ALTER TABLE.

6. Do not use the ALTER TABLE statement within a transaction; it cannot be rolled back.

Examples

```
ALTER TABLE items
    ADD (item_weight DECIMAL(6,2)
         BEFORE total_price)

ALTER TABLE items
    DROP (total_price)

ALTER TABLE items
    MODIFY (manu_code CHAR(4))
```

Related Statements

RENAME COLUMN, RENAME TABLE

BEGIN WORK

Overview

Use the BEGIN WORK statement to start a transaction (a sequence of database operations that are terminated by the COMMIT WORK or ROLLBACK WORK statement).

Syntax

BEGIN WORK

Explanation

BEGIN WORK are required keywords.

Notes

1. See the section "Transactions" in Chapter 2 for a full description of transactions.

2. Each row affected by an UPDATE, DELETE, or INSERT statement during a transaction is locked and remains locked throughout the transaction. A transaction that contains a large number of such statements, or that contains statements affecting a large number of rows, may exceed the limits placed by your operating system on the maximum number of simultaneous locks. If you encounter this error, you may need to lock the entire table immediately after beginning the transaction.

   See the section "Locking" in Chapter 2 for a more detailed description of table-level and row-level locking in **INFORMIX-4GL**.

Related Statements

COMMIT WORK, ROLLBACK WORK

CALL

Overview

Use the CALL statement to invoke a function.

Syntax

CALL *function*([*argument-list*]) [RETURNING *variable-list*]

Explanation

CALL	is a required keyword.
function	is the name of a function.
argument-list	is a list of zero or more expressions, separated by commas and enclosed in parentheses. The parentheses are required even if there are no arguments.
RETURNING	is an optional keyword.
variable-list	is a list of one or more program variables, separated by commas.

Notes

1. You can use the CALL statement to call both **INFORMIX-4GL** functions and C functions. See Chapter 1 for the rules on using C functions in **INFORMIX-4GL** programs.

2. The arguments *arg-list* will be passed by value.

3. You may define **INFORMIX-4GL** functions in the same source file as the MAIN routine, or you may compile them separately and link them later to the MAIN routine.

Examples

```
CALL statistics(rec.*) RETURNING mean, std_dev
```

Related Statements

DEFINE, FUNCTION

CASE

Overview

Use the CASE statement to select a sequence of statements depending on the current value of an expression.

Syntax

```
CASE [(expr)]
    WHEN {expr | Boolean-expr}
        statement
        ...
        [EXIT CASE]
        ...
    WHEN {expr | Boolean-expr}
        statement
        ...
        [EXIT CASE]
        ...
    ...
    [OTHERWISE]
        statement
        ...
        [EXIT CASE]
        ...
END CASE
```

Explanation

CASE	is a required keyword.
expr	is an expression that returns an INTEGER, a SMALLINT, a DECIMAL, or a CHAR(1) value.
WHEN	is a required keyword.
Boolean-expr	is an expression that is either true or false.
statement	is an **INFORMIX-4GL** statement.

EXIT CASE — is an optional statement that causes program control to pass to the statement following the END CASE keywords.

OTHERWISE — is an optional keyword introducing a sequence of statements to be executed if none of the WHEN clauses is executed.

END CASE — are required keywords that terminate the CASE statement.

Notes

1. The CASE statement is equivalent to a set of nested IF statements.

2. If you use the OTHERWISE option, it must be the last in the list.

3. If the optional parenthesized expression following the CASE keyword is missing, you must follow the WHEN keyword with a Boolean expression. If there is an expression following the CASE keyword, you must follow the WHEN keyword with an expression that evaluates to the same type.

4. There is an implied EXIT CASE statement at the end of each sequence of statements following a WHEN clause. Program control will pass to the sequence of statements following the END CASE statement.

Examples

```
LABEL question:
   ...
   CASE
      WHEN answer MATCHES "[Yy]"
            CALL process()
      WHEN answer MATCHES "[Nn]"
            CALL abort()
      OTHERWISE
            CALL retry()
   END CASE
```

Related Statements

IF, EXIT

CLEAR

Overview

Use the CLEAR statement to clear the whole screen, a window, all fields in a screen form, or a set of fields.

Syntax

CLEAR {SCREEN | WINDOW *window-name* | FORM | *field-list*}

Explanation

CLEAR	is a required keyword.
SCREEN	is the keyword you use to clear the whole screen.
WINDOW	is the keyword you use to clear a window.
window-name	is the name of the window you want to clear.
FORM	is the keyword you use to clear the values in all screen fields of a form.
field-list	is a list of one or more field names representing fields that you want to clear.

Notes

1. The CLEAR statement does not change the value of any variable; it simply clears the display from the region indicated.

2. The CLEAR SCREEN statement makes the screen the current window and clears it.

3. The CLEAR WINDOW statement clears the specified window, retaining any border. (The specified window need not be the current window; the current window remains unchanged.) For example, if you specify screen as the *window-name* in a CLEAR WINDOW statement, INFORMIX-4GL clears the screen except for the area occupied by any windows.

4. CLEAR FORM and CLEAR *field-list* apply to the form in the current window.

Examples

```
CLEAR fname, lname, address1,
    city, state, zipcode

CLEAR FORM

CLEAR SCREEN

CLEAR WINDOW win1

CLEAR WINDOW screen
```

CLOSE

Overview

Use the CLOSE statement when you no longer need to refer to the active set of a SELECT cursor or when you want to flush the insert buffer and close an INSERT cursor.

Syntax

CLOSE *cursor-name*

Explanation

CLOSE	is a required keyword.
cursor-name	is the name of a cursor that has been DECLAREd for a SELECT or an INSERT statement.

Notes

1. If *cursor-name* is associated with a SELECT statement, the CLOSE statement puts the cursor in a closed state and leaves the active set undefined.

2. After you CLOSE a SELECT cursor, you cannot execute a FETCH statement until you reopen it.

3. If *cursor-name* is associated with an INSERT statement, the CLOSE statement flushes any rows in the buffer into the database (writes to disk) and closes the cursor.

4. After you CLOSE an INSERT cursor, you cannot execute a PUT or FLUSH statement until you reOPEN the cursor.

5. The global variables **status** (whose value is taken from SQLCA.SQLCODE) and SQLCA.SQLERRD[3] indicate the result of each FLUSH and CLOSE statement for an INSERT cursor. If **RDSQL** successfully inserts the buffered rows into the database, it sets **status** to zero and sets SQLCA.SQLERRD[3] to the number of rows inserted into the database. If **RDSQL** encounters an error while inserting the buffered rows into the database, it sets **status** to a negative number (specifically, the number of the error message) and sets SQLCA.SQLERRD[3] to the number of rows successfully inserted into the database. Buffered rows following the last successfully inserted row are discarded. In this case, **RDSQL** does not close the cursor.

6. Although the COMMIT WORK and ROLLBACK WORK statements close all open cursors, you should not use them for this purpose. Explicitly CLOSE each INSERT cursor before committing the work so you can check that the insertion was successful.

7. **RDSQL** does not provide a global variable containing the total number of rows successfully inserted into the database with an insert cursor. If you want to know the total number of inserts performed, you must set a counter in your program and increment it upon each PUT statement.

Examples

```
CLOSE query_cursor

CLOSE icurs
```

Related Statements

DECLARE, FETCH, FLUSH, OPEN, PUT

CLOSE DATABASE

Overview

Use the CLOSE DATABASE statement to close the current database and to leave the program with no current database.

Syntax

```
CLOSE DATABASE
```

Explanation

CLOSE DATABASE are required keywords.

Notes

1. Following the CLOSE DATABASE statement, the only legal RDSQL statements are CREATE DATABASE, DATABASE, and DROP DATABASE.

2. The CLOSE DATABASE statement is useful when you want to drop the only remaining database.

Examples

```
CLOSE DATABASE
```

Related Statements

CREATE DATABASE, DROP DATABASE

CLOSE FORM

Overview

Use the CLOSE FORM statement to release the memory required for a screen form.

Syntax

CLOSE FORM *form-name*

Explanation

CLOSE FORM are required keywords.

form-name is an **INFORMIX-4GL** identifier that you assigned to a screen form in an OPEN FORM statement.

Notes

1. After you execute the CLOSE FORM statement, *form-name* is no longer associated with a screen form. Executing a subsequent DISPLAY FORM statement will give an error message.

2. If you execute a new OPEN FORM statement with the same *form-name*, **INFORMIX-4GL** will close the existing form before opening the new one.

3. When you execute the OPEN FORM statement, the compiled form is loaded into and kept in memory until you execute a CLOSE FORM statement for that form. If you have displayed another form and wish to regain the memory used by the first form, you may execute CLOSE FORM on the old form.

4. The CLOSE FORM statement affects memory use only and does not affect the logic of the program. Since allocating memory and reading a form from the disk takes time, you should leave open forms that you use repeatedly.

Examples

```
CLOSE FORM order_entry
```

Related Statements

DISPLAY FORM, OPEN FORM

CLOSE WINDOW

Overview

Use the CLOSE WINDOW statement to close a window.

Syntax

CLOSE WINDOW *window-name*

Explanation

CLOSE WINDOW	are required keywords.
window-name	is the name of the window you want to close.

Notes

1. When you close a window, **INFORMIX-4GL** frees all resources used by the window, including forms, and restores the underlying display.

2. When you close the current window, the next window on the stack becomes the current window. When you close any other window, **INFORMIX-4GL** simply removes the window from the stack, leaving the current window unchanged. In both cases, **INFORMIX-4GL** restores the underlying display.

3. If you close a window that is currently being used for input, **INFORMIX-4GL** generates a runtime error. For example, closing the current window in the middle of a DISPLAY ARRAY, INPUT, INPUT ARRAY, or MENU statement produces a runtime error.

4. You cannot issue a CLOSE WINDOW *screen* command.

Examples

```
CLOSE WINDOW win1
```

Related Statements

CLEAR WINDOW, CURRENT WINDOW, OPEN WINDOW, OPTIONS

COMMIT WORK

Overview

Use the COMMIT WORK statement to commit all modifications made to the database since the last BEGIN WORK statement.

Syntax

```
COMMIT WORK
```

Explanation

COMMIT WORK are required keywords.

Notes

1. Use the COMMIT WORK statement when you are satisfied that the changes to the database made since the last BEGIN WORK statement are what you want.

2. The COMMIT WORK statement closes all open cursors. Do not use the COMMIT WORK statement within a FOREACH loop.

3. The COMMIT WORK statement releases all row and table locks.

4. See the section "Transactions" in Chapter 2 for details.

Related Statements

BEGIN WORK, ROLLBACK WORK

CONSTRUCT

Overview

Use the CONSTRUCT statement to create a CHAR variable that contains the Boolean expression constructed from a screen-generated *query-by-example*.

Syntax

```
CONSTRUCT {BY NAME char-variable ON column-list |
        char-variable ON column-list
        FROM {field-list | screen-record[[n]].* } [, ...]}
```

Explanation

CONSTRUCT	is a required keyword.
BY NAME	are optional keywords.
char-variable	is an identifier of a CHAR type program variable.
ON	is a required keyword.
column-list	is a list of one or more database column names, separated by commas. You can use the syntax *table.**.
FROM	is an optional keyword.
field-list	is a list of one or more screen field names. You can use the syntax *screen-record.** or *screen-record*[*n*].*.

Notes

1. The CONSTRUCT statement allows the user to use a screen form to enter query-by-example search parameters with the syntax described after the following table:

Symbol	Name	Data Types	Pattern
=	equal to	all	=x
>	greater than	all	>x
<	less than	all	<x
>=	greater than or equal to	all	>=x
<=	less than or equal to	all	<=x
<>	not equal to	all	<>x
:	range	all	x:y
*	wildcard	CHAR	*x, x*, *x*
?	single-character wildcard	CHAR	?x, x?, ?x?, x??
|	or	all	a|b...

Explanation

- The equal sign (=) is the default query symbol for non-character columns.

- The default query symbol for a character column depends upon the value the user enters in the corresponding character field. If the user enters a character value that contains no wildcard characters, **INFORMIX-4GL** uses the equal sign as the default query symbol:

  char-column="*value*"

  If the user enters a character value that contains a wildcard character (either an asterisk or a question mark), **INFORMIX-4GL** uses MATCHES as the default query symbol:

  column matches "*value*"

- Enter the equal sign by itself to search for a database row that contains a null column; enter = * to find a row that contains a column with only an asterisk.

- **x** is any search value with the appropriate data type for the search field. Enter the search value immediately after any one of the first six query symbols in the table above. Do not leave a space between the query symbol and the search value.

- The symbols >, <, >=, and <= imply an ordering of the data in the column. For CHAR data, greater than means later in ASCII ordering (a>A>1). For DATE data, greater than means after.

- **x:y** creates a search for all values between **x** and **y**, inclusive. **y** must be higher in value than **x**. The search criterion 1:10 would find all rows with a value in that column from 1 through 10.

- * is the wildcard character. It represents zero or more characters. An *ts* search value in the field corresponding to the **lname** column of the **customer** table would find two names, Watson and Albertson. An S* search value in the same display field would find Sadler and Sipes. An *er search value would find the four names Sadler, Miller, Jaeger, and Baxter.

- The question mark (?) is the single-character wildcard. The user can use the question mark to find values that match a pattern where the number of characters is fixed. For example, the user can enter Eriks?n to search for names like "Erikson" and "Eriksen." Similarly, the user can enter New??n to search for names like "Newton," "Newman," and "Newson."

- The symbol | between values **a** and **b** signifies the logical OR. The user entry

  ```
  102|105|118
  ```

  in the field corresponding to the column **customer_num** retrieves any of those three numbers.

2. You can use the BY NAME option when the field names on the screen form have the same names as the corresponding column names in *column-list*. If you do not, you must specify a screen record or name the fields explicitly in *field-list*.

3. The CONSTRUCT statement is terminated when the user enters **ESCAPE** or the key specified as the ACCEPT KEY in the OPTIONS statement. For single-item CONSTRUCTs, a **RETURN** is equivalent to pressing the ACCEPT KEY unless the INPUT WRAP option is in effect. For multiple-item CONSTRUCTs, a **RETURN** after the last item is equivalent to pressing the ACCEPT KEY unless INPUT WRAP is in effect.

4. By default, both ESCAPE and INTERRUPT are exits from the CONSTRUCT statement. If the DEFER INTERRUPT statement has been executed, an INTERRUPT causes **INFORMIX-4GL** to set the global variable **int_flag** to nonzero and to terminate the CONSTRUCT statement. Otherwise, an INTERRUPT causes an immediate program stop.

5. In addition to the **RETURN**, **ESCAPE**, INTERRUPT, and **ARROW** keys, the user can employ the following keys for editing during a CONSTRUCT statement:

CONTROL-A	toggles between insert and typeover mode.
CONTROL-D	deletes characters from the current cursor position to the end of the field.
CONTROL-H	moves the cursor nondestructively one space to the left inside a field. It is equivalent to pressing the **LEFT ARROW** key.
CONTROL-L	moves the cursor nondestructively one space to the right inside a field. It is equivalent to pressing the **RIGHT ARROW** key.
CONTROL-R	redisplays the screen.

CONTROL-X deletes the character beneath the cursor.

6. The user can query for only those fields displayed on the screen that you have specified in the FROM clause. The number of fields in the FROM clause must be the same as the number of columns in the ON clause. The order of fields in the FROM clause must match the order of columns in the ON clause. **INFORMIX-4GL** constructs *char-variable* by associating the column name in the ON clause with the search condition the user entered into the corresponding field in the FROM clause.

7. The UPSHIFT and DOWNSHIFT attributes work during a CONSTRUCT statement. The COMMENTS attribute works with the following restriction: if a field that displays a comment is too short to hold the search criteria the user enters, **INFORMIX-4GL** opens a work space on the comment line, erasing any comment that is displayed.

8. If the column names in a CONSTRUCT BY NAME statement are associated with field names in a screen array, the construct takes place in the first row of the screen array.

   If you want to use screen-array field names in the FROM clause of a CONSTRUCT statement, you must specify (with the *screen-record*[*n*].*field-name* notation) the row in which the construct takes place.

9. When you use *screen-record.** or *table.** as "shorthands" for explicit lists, be sure that the order of the fields implied in the *screen-record.** notation corresponds to the order of the columns implied in the *table-record.** notation. The order of the fields in *screen-record.** depends on its definition in the screen form. The order of the columns in *table.** depends on the order in the **syscolumns** system catalog at the time you compile your program. If you have altered the order or number of the columns in *table* since you compiled your program, you may need to alter your program and the forms that depend on it.

10. The information stored in *char-variable* can be used in the WHERE clause of a PREPAREd SELECT statemen to retrieve a set of rows from the database.

Examples

```
CONSTRUCT query_1
   ON order_num, customer_num, order_date,
      ship_date
   FROM order_num, customer_num, order_date,
      ship_date

LET s1 = "select * from orders where ", query_1

PREPARE s_1 FROM s1

DECLARE cursor_1 CURSOR FOR s_1

FOREACH cursor_1 INTO order_rec.*
   ...

END FOREACH
```

Related Statements

DECLARE, PREPARE, SELECT

CONTINUE

Overview

Use the CONTINUE statement to cause a FOR, FOREACH, or WHILE statement to start a new cycle immediately if the conditions permit or to return to the menu from an option in the MENU statement.

Syntax

```
CONTINUE {FOR | FOREACH | MENU | WHILE}
```

Explanation

CONTINUE	is a required keyword.
FOR	is a required keyword in a FOR statement.
FOREACH	is a required keyword in a FOREACH statement.
MENU	is a required keyword in a MENU statement.
WHILE	is a required keyword in a WHILE statement.

Related Statements

EXIT

CREATE AUDIT

Overview

Use the CREATE AUDIT statement to create an audit trail file and to start writing the audit trail.

Syntax

CREATE AUDIT FOR *table-name* IN "*pathname*"

Explanation

CREATE AUDIT FOR	are required keywords.
table-name	is the name of the table for which you want to create an audit trail file.
IN	is a required keyword.
pathname	is the full pathname for the audit trail file. It must be enclosed in quotation marks.

Notes

1. You create audit trails to keep a record of all modifications of a table. An audit trail is a complete history of all additions, deletions, and updates to the table. **RDSQL** can use the audit trail to reconstruct the table from a backup copy made at the time the audit trail is created. (See the RECOVER TABLE statement.) See the section "Audit Trails" in Chapter 2 for more information.

2. If an audit trail file for the same table exists with a different pathname, **RDSQL** displays an error message.

3. Make a backup copy of your database files as soon as you run the CREATE AUDIT statement, but before you make any further changes to the database. (See the RECOVER TABLE statement for an example.) If possible, put the audit trail file on a different physical device from the one that holds your data, so that a failure of one does not damage the data on the other.

4. Audit trails slow **RDSQL** slightly because each alteration of the table is recorded in the audit trail file, as well as in the database files.

5. You must own *table-name* or have DBA status to use the CREATE AUDIT statement.

6. You must set execute permission for all directories below **root** in *pathname* for each class of user (owner, owner's group, and public) that accesses your database.

7. You cannot create an audit file for a view.

Examples

On a UNIX system:

```
create audit for orders in "/dev/safe"
```

On a DOS system:

```
create audit for orders in "D:\KEEP\SAFE"
```

Related Statements

DROP AUDIT, RECOVER TABLE

CREATE DATABASE

Overview

Use the CREATE DATABASE statement to create a new database. **RDSQL** will create the system catalogs that will contain the data dictionary describing the structure of the database. The database you create automatically becomes the current database.

Syntax

CREATE DATABASE *database-name* [WITH LOG IN "*pathname*"]

Explanation

CREATE DATABASE	are required keywords.
database-name	is the name you want to assign to the database. *database-name* may be a program variable of type CHAR containing the name of the database you want to create.
WITH LOG IN	are optional keywords.
pathname	is the full pathname of the transaction log file. *pathname* must be enclosed in quotation marks.

Notes

1. **RDSQL** creates a subdirectory in the current directory with the name *database-name.dbs*. All of the system catalogs, data, and index files will be placed in this subdirectory, except for tables that you explicitly instruct **RDSQL** to create elsewhere.

2. A database name can be up to eight (DOS) or ten (UNIX) characters in length and can contain only letters, digits, and underscores(_). The first character must be a letter. If you store more than one database in a single directory, the database names must be unique.

3. For a user to have access to a database on UNIX systems, the user must have execute (search) permission for each directory in the full pathname of *database-name.dbs*, as well as appropriate database privileges (see the GRANT statement).

4. See Appendix B for a detailed description of the system catalogs.

5. The WITH LOG IN clause creates a transaction log file. Without this file, you may not use the BEGIN WORK, COMMIT WORK, or the ROLLBACK WORK statements. You can use the START DATABASE statement to assign a log file to an existing database. See the section "Transactions" in Chapter 2 for further details.

Examples

On a UNIX system:

```
CREATE DATABASE stores
    WITH LOG IN "/s/log/stores.log"
```

On a DOS system:

```
CREATE DATABASE stores
    WITH LOG IN "\LOG\STORES.LOG"
```

Related Statements

DROP DATABASE, GRANT, START DATABASE

CREATE INDEX

Overview

Use the CREATE INDEX statement to create an index for one or more columns in a table and optionally to cluster the physical table in the order of the index. When there is more than one column listed, the concatenation of the set of columns is treated as a single composite column for the purpose of indexing.

Syntax

```
CREATE [UNIQUE | DISTINCT] [CLUSTER] INDEX index-name
  ON table-name (column-name [ASC | DESC], ...)
```

Explanation

CREATE INDEX	are required keywords.
UNIQUE	is a keyword you use to prevent duplicate entries in the column or composite column to which the index applies.
DISTINCT	is a keyword that is synonymous with UNIQUE.
CLUSTER	is an optional keyword that causes the physical table to be ordered according to the order of the index.
index-name	is the **RDSQL** identifier you want to assign to the index. You must assign a different identifier to each index in the database.
ON	is a required keyword.

table-name	is the name of the table containing the column or columns you want to index.
column-name	is the name of a column you want to index. To create an index that applies to several columns, enter a list of column names, separated by commas. All the columns you specify must belong to the same table.
ASC	is a keyword that specifies an index that RDSQL maintains in ascending order. ASC is the default.
DESC	is a keyword that specifies an index that RDSQL maintains in descending order.

Notes

1. When RDSQL executes the CREATE INDEX statement, it locks *table-name* in EXCLUSIVE mode. If another process is using *table-name*, RDSQL cannot execute CREATE INDEX and returns an error.

2. You may include up to eight columns in a composite index.

3. The total length of all columns indexed in a single CREATE INDEX statement cannot exceed 120 bytes.

4. See the section "Indexing Strategy" in Chapter 2 for a discussion of indexing strategy.

5. The CREATE CLUSTER INDEX statement fails if a CLUSTER index already exists.

6. Only one index on a particular sequence of columns is allowed (ASC and DESC are considered to provide uniqueness).

7. If you use the CREATE INDEX statement in a transaction, it cannot be rolled back. However, it can be rolled forward.

Examples

```
CREATE UNIQUE INDEX i_ordnum
    ON orders (order_num)

CREATE CLUSTER INDEX i_ordnum2
    ON orders (order_num DESC)
```

Related Statements

ALTER INDEX, DROP INDEX

CREATE SYNONYM

Overview

Use the CREATE SYNONYM statement to provide an alternative name for a table or view.

Syntax

CREATE SYNONYM *synonym* FOR *table-name*

Explanation

CREATE SYNONYM	are required keywords.
synonym	is an **INFORMIX-4GL** identifier.
FOR	is a required keyword.
table-name	is the name of a table or view.

Notes

1. A synonym is effective only for the user who creates it.

2. A user has no privileges under a synonym that were not granted for the table to which it applies.

3. When a synonym is created in an **INFORMIX-4GL** program, the owner of the synonym is the person who runs the program.

4. Once created, a synonym persists until the owner of the synonym executes the DROP SYNONYM statement. This property distinguishes the synonym from the alias that you can use in the FROM clause of a SELECT statement. The alias persists only until the completion of the SELECT statement.

5. Do not use the CREATE SYNONYM statement in a transaction; it cannot be rolled back.

6. The synonym name cannot be the same as a table in the database.

Examples

```
CREATE SYNONYM c FOR customer
```

Related Statements

DROP SYNONYM, SELECT

CREATE TABLE

Overview

Use the CREATE TABLE statement to create a new table in the current database.

Syntax

CREATE [TEMP] TABLE *table-name*
(*column-name datatype* [NOT NULL], ...)
[IN "*pathname*"]

Explanation

CREATE TABLE	are required keywords.
TEMP	is an optional keyword.
table-name	is the identifier you want to assign to the table. The first eight (DOS) or ten (UNIX) characters must be unique within a database.
column-name	is the identifier you want to assign to each column.
datatype	specifies the data type for each column. (See the following list for valid data types.)
NOT NULL	are optional keywords.
IN	is an optional keyword.

pathname specifies the full pathname in which you want to store the database table, with no extension to the filename. The *pathname* cannot be longer than 64 characters, and must be contained within quotes (").

In UNIX systems, a pathname is of the form

[/*directory-name*/...]*filename*

and in DOS systems the form of a pathname is

[*directory-name*\\...]*filename*

(that is, with double-backslash separators).

The following are the valid data types in **RDSQL**:

CHAR(*n*) is a character string of length *n* (where 1 <= *n* <= 32,767).

SMALLINT is a whole number from −32,767 to +32,767.

INTEGER is a whole number from −2,147,483,647 to +2,147,483,647.

DECIMAL[(*m*[,*n*])] is a decimal floating-point number with a total of *m* (<= 32) significant digits (precision) and *n* (<= *m*) digits to the right of the decimal point (scale). See the section "Database Data Types" in Chapter 2 for more details.

SMALLFLOAT is a binary floating-point number corresponding to the "float" data type in the C language. On some small machines, **INFORMIX-4GL** implements SMALLFLOAT columns as DECIMAL(8).

FLOAT	is a binary floating-point number corresponding to the "double" data type in the C language. On some small machines, **INFORMIX-4GL** implements FLOAT columns as DECIMAL(16).
MONEY[(*m*[,*n*])]	is a DECIMAL type number, displayed with leading **$**. MONEY(*m*) = DECIMAL(*m*,2) and MONEY = DECIMAL(16,2). See the section "Database Data Types" in Chapter 2 for more details.
SERIAL[(*n*)]	is a unique sequential integer assigned automatically by **RDSQL**. You may assign an initial value *n*. The default starting integer is 1.
DATE	is a date entered as a character string in one of the formats described in the following notes.

Notes

1. Table names must be unique within a database.

2. Column names must be unique within each table, but you can use duplicate names in different tables in the same database. See "RDSQL Identifiers" in Chapter 2 for guidelines on table names and column names.

3. Temporary tables created with the TEMP option last only for the duration of the program.

4. Users with CONNECT privilege may create temporary tables.

5. The default value for a column is NULL, unless you include the NOT NULL keywords after the data type of the column.

6. If you designate a column as NOT NULL, you must enter a value into this column when performing an INSERT or UPDATE to the table.

7. By default, all users who have been granted CONNECT privilege to the database have all types of access to the new table except ALTER. To further restrict access, use the REVOKE statement to take *all* access away from PUBLIC (everyone on the system); then use the GRANT statement to designate the types of access you want to give to specific users.

8. You can specify only one SERIAL column in a table.

9. Enter DATE data types in the sequence of month, day, and year, with any non-numeric character, including a blank, as a separator. Represent the month as the number of the month (January = 1 or 01, February = 2 or 02, and so on). Represent the day as the day of the month (1 or 01, 2 or 02, and so on). The year is stored as a four-digit number (0001 to 9999). If you enter two digits *yy* for the year, **RDSQL** assumes the year is *19yy*.

   The following are all acceptable representations of June 1, 1986: 06/01/86, 6.1.86, and 6-1-1986.

10. The DATE type is actually stored as the integer number of days since December 31, 1899. You can sort DATE columns and make chronological comparisons between two DATE columns.

11. The file space requirements in bytes for each data type are

SERIAL	4
SMALLINT	2
INTEGER	4
SMALLFLOAT	4
FLOAT	8
CHAR(n)	n
DECIMAL(m,n)	1 + m/2
MONEY(m,n)	1 + m/2
DATE	4

12. Do not use the CREATE TABLE statement within a transaction; it cannot be rolled back.

13. If the *pathname* in the IN clause specifies a filename that is different from the *table-name*, always use the *table-name* (rather than the filename) to refer to the table in subsequent **INFORMIX-4GL** statements.

14. The *pathname* in an IN clause can specify any valid directory, and is not restricted to the drive or directory that contains the current database. This enables a database to include tables in different disk volumes, on more than one drive, on different network nodes, or on virtual drives. Use this feature if your database is becoming too large for your current disk volume.

Examples

The following statements create the tables in the **stores** database.

```
CREATE DATABASE stores

CREATE TABLE customer
    (
    customer_num        SERIAL(101),
    fname               CHAR(15),
    lname               CHAR(15),
    company             CHAR(20),
    address1            CHAR(20),
    address2            CHAR(20),
    city                CHAR(15),
    state               CHAR(2),
    zipcode             CHAR(5),
    phone               CHAR(18)
    )

CREATE TABLE orders
    (
    order_num           SERIAL(1001),
    order_date          DATE,
    customer_num        INTEGER,
    ship_instruct       CHAR(40),
    backlog             CHAR(1),
    po_num              CHAR(10),
    ship_date           DATE,
    ship_weight         DECIMAL(8,2),
    ship_charge         MONEY(6),
    paid_date           DATE
    )

CREATE TABLE items
    (
    item_num            SMALLINT,
    order_num           INTEGER,
    stock_num           SMALLINT,
    manu_code           CHAR(3),
    quantity            SMALLINT,
    total_price         MONEY(8)
    )
```

```
CREATE TABLE stock
    (
    stock_num           SMALLINT,
    manu_code           CHAR(3),
    description         CHAR(15),
    unit_price          MONEY(6),
    unit                CHAR(4),
    unit_descr          CHAR(15)
    )

CREATE TABLE manufact
    (
    manu_code           CHAR(3),
    manu_name           CHAR(15)
    )

CREATE TABLE state
    (
    code                CHAR(2),
    sname               CHAR(15)
    )
```

Related Statements

CREATE DATABASE, DROP DATABASE, DROP TABLE, GRANT, REVOKE

CREATE VIEW

Overview

Use CREATE VIEW to create a new view based upon existing tables and views in the database.

Syntax

CREATE VIEW *view-name* [(*column-list*)]
AS *SELECT-statement* [WITH CHECK OPTION]

Explanation

CREATE VIEW	are required keywords.
view-name	is an **RDSQL** identifier.
column-list	is a list of one or more identifiers that name the columns of *view-name*.
AS	is a required keyword.
SELECT-statement	is an **RDSQL** SELECT statement.
WITH CHECK OPTION	are optional keywords.

Notes

1. Except for the statements in the following list, you can use a view in any **INFORMIX-4GL** statement where you can use a table.

ALTER TABLE	LOCK TABLE
CREATE AUDIT	RECOVER TABLE
CREATE INDEX	RENAME TABLE
DROP AUDIT	

   The view behaves like a table having the name *view-name* and consisting of the set of rows and columns returned by the *SELECT-statement* each time the **INFORMIX-4GL** statement is executed using the view. The view reflects changes to the underlying tables with one exception. If the view is defined with a SELECT * clause, it contains only the columns in the underlying tables at the time the view is created. New columns added subsequently to the underlying tables using the ALTER TABLE statement will not appear in the view. See the section "Views" in Chapter 2 for more information.

2. If you do not specify *column-list* for *view-name*, the view will inherit the column names of the underlying tables. If the *SELECT-statement* returns an expression, the corresponding column in the view is called a *virtual* column. You must provide a name for virtual columns. You must also provide a column name when the *select-list* has duplicate column names when the table prefixes are stripped. For example, when both **orders.order_num** and **items.order_num** appear in the *select-list*, you must provide two separate column names to label them in the CREATE VIEW statement.

3. Data types of the columns of the view are inherited from the tables from which they come. Data types of virtual columns are determined from the nature of the expression.

4. *view-name* must be unique among all the tables and views that already exist in the database.

5. You can define a view in terms of other views, except that you must abide by the restrictions on queries listed in the section "Querying through Views," in Chapter 2.

6. *SELECT-statement* cannot have an ORDER BY clause nor a UNION operator, and it cannot include program variables.

7. You must have SELECT privilege on all columns from which the view is derived.

8. The WITH CHECK OPTION clause instructs **RDSQL** to ensure that all modifications to the underlying tables made through the view satisfy the definition of the view.

9. You cannot update a view if the view definition involves joins, the GROUP BY clause, the DISTINCT keyword, or an aggregate function. You also cannot update virtual columns or INSERT rows through a view that contains virtual columns.

10. Do not use the CREATE VIEW statement within a transaction; it cannot be rolled back.

Examples

```
CREATE VIEW palo_alto AS
   SELECT * FROM customer
       WHERE city = "Palo Alto"
```

Related Statements

CREATE TABLE, DROP VIEW

CURRENT WINDOW

Overview

Use the CURRENT WINDOW statement to make a window the current or topmost window.

Syntax

CURRENT WINDOW IS *window-name*

Explanation

CURRENT WINDOW IS	are required keywords.
window-name	is the name of the window that you want to be the current window.

Notes

1. A window becomes completely visible when it becomes the current window. In the process, other inactive windows may be obscured.

2. All input and output is done in the current window.

3. If *window-name* contains a screen form, the screen form becomes the current form.

4. The terminal screen is the current window when a program starts.

5. If you specify screen as the *window-name*, the entire screen becomes the current window.

6. The DISPLAY ARRAY, INPUT, INPUT ARRAY, and MENU statements run in the current window. When you change the current window while one of these statements is active and then resume the statement, the original window is restored as the current window. For example, you can use an ON KEY clause in an INPUT statement to allow the user to open a new window by pressing a specific key during input. When the user presses the designated key, **INFORMIX-4GL** executes the statements in the ON KEY clause and then resumes input from the window that was current before the ON KEY break.

7. The context of each window includes the values for the PROMPT, MESSAGE, FORM, and COMMENT lines. When a window becomes the current window, these values are restored.

8. When working with multiple windows, **INFORMIX-4GL** maintains a list or "stack" of all open windows. It adds the current window to its window list whenever you open a new window. The new window then becomes the current window. When you close a window, **INFORMIX-4GL** removes it from its window list. The topmost window (of those that remain) becomes the current window.

9. When you specify a current window, **INFORMIX-4GL** adjusts the window list by moving the new current window to the top and closing the gap in the list left by this window.

Examples

```
CURRENT WINDOW IS win1

CURRENT WINDOW IS screen
```

Related Statements

CLEAR WINDOW, CLOSE WINDOW, OPEN WINDOW, OPTIONS

DATABASE

Overview

Use the DATABASE statement to select an accessible database as the current database.

Syntax

DATABASE *database-name* [EXCLUSIVE]

Explanation

DATABASE is a required keyword.

database-name is the name of the database you want to select. It may also be a program variable that evaluates to the name of a database.

EXCLUSIVE is an optional keyword.

Notes

1. If you want to specify a database that does not reside in your current directory nor in a directory specified by the DBPATH environment variable (see Appendix C), you must follow the DATABASE keyword with a program variable that evaluates to the full pathname of the database (excluding the **.dbs** extension).

2. In an INFORMIX-4GL program, the DATABASE statement can serve two purposes, one procedural and the other non-procedural. It makes the named database the current database (procedural), and it tells the compiler where to find information about variables defined LIKE columns in a table (non-procedural).

To serve the non-procedural purpose, the DATABASE statement must occur outside of any routine and precede the GLOBALS statements when you use indirect data typing using the LIKE clause. *database-name* must be explicitly expressed and not given as a program variable. You may not use the EXCLUSIVE keyword in this context. If you use the DATABASE statement in this non-procedural way, **INFORMIX-4GL** begins the MAIN routine making the database you name the current database.

Ordinarily, you use only one database, and the preceding procedure is enough. If you do not have global variables defined LIKE database columns, but still want to interact with a database, you can use the DATABASE statement in a purely procedural way. In this case, it must occur within a routine and must follow any DEFINE statements within that routine. *database-name* may be a program variable, and you may use the EXCLUSIVE keyword.

3. If you close one database and open another in a program, you cannot define variables LIKE columns in the second database.

4. The EXCLUSIVE option opens the database in an exclusive mode and allows only the current user access to the database. To allow others access to the database, you must execute the CLOSE DATABASE statement and then reopen the database.

5. Do not use the DATABASE statement within a transaction session; it cannot be rolled back.

Examples

```
DATABASE stores
```

Related Statements

CREATE DATABASE, DROP DATABASE,
CLOSE DATABASE

DECLARE

Overview

Use the DECLARE statement to assign a cursor name to a SELECT or INSERT statement.

Syntax

```
DECLARE cursor-name [SCROLL] CURSOR FOR
    {SELECT-statement [FOR UPDATE [OF column-list]] |
        INSERT-statement | statement-id}
```

Explanation

DECLARE	is a required keyword.
cursor-name	is an **RDSQL** identifier.
SCROLL	is an optional keyword and can be used only with a SELECT statement or a statement_id that represents a PREPAREd SELECT statement.
CURSOR FOR	are required keywords.
SELECT-statement	is a SELECT statement.
FOR UPDATE	are optional keywords.
OF	is an optional keyword.
column-list	is a list of column names from the tables listed in the FROM clause of the SELECT statement.
INSERT-statement	is an INSERT statement.
statement-id	is the identifier of an INSERT or SELECT statement that has been previously PREPAREd.

Notes

1. You must DECLARE a SELECT cursor before it can be used in an OPEN, FETCH, FOREACH, DELETE, UPDATE, or CLOSE statement. You must DECLARE an INSERT cursor before it can be used in an OPEN, PUT, FLUSH, or CLOSE statement.

2. You must include the SCROLL keyword in the DECLARE statement for a SELECT cursor if you are going to issue a FETCH PREVIOUS, FETCH LAST, FETCH FIRST, FETCH CURRENT, FETCH RELATIVE, or FETCH ABSOLUTE statement.

3. If you use a *statement-id* to identify a SELECT or INSERT statement, you must have previously PREPAREd the statement.

4. *cursor-name* has meaning from the point at which it is DECLAREd to the end of the source file. This means that the DECLARE statement for a cursor must physically appear before any statement that makes reference to the cursor. It is not a global variable that can be used in a separate source file.

5. You cannot use the FOR UPDATE clause in the DECLARE statement for a SELECT cursor that scrolls or includes an ORDER BY clause.

6. You must use the FOR UPDATE clause in the DECLARE statement for a non-scrolling SELECT cursor if you will later use either the UPDATE or the DELETE statement with the WHERE CURRENT OF *cursor-name* option.

7. If you do not specify any columns in the FOR UPDATE clause of a DECLARE statement, you can update any column in a subsequent UPDATE WHERE CURRENT OF statement. If you do specify one or more columns in the FOR UPDATE clause, you can update only those columns in a subsequent UPDATE WHERE CURRENT OF statement. When you specify the column names in the

FOR UPDATE clause, **INFORMIX-4GL** can usually perform the updates more quickly.

8. The columns in an OF *column-list* clause need not be in the *select-list* of the SELECT clause.

9. When you DECLARE a cursor FOR UPDATE, each FETCH executed on that cursor locks the FETCHed row in exclusive mode. The lock is released when you execute the next FETCH statement or when you CLOSE the cursor. See the section "Locking" in Chapter 2 for a more detailed description of table-level and row-level locking. When you execute the actions within a transaction, the locked rows are released only when you issue a COMMIT WORK or ROLLBACK WORK statement.

10. **INFORMIX-4GL** evaluates the variables in a DECLARE statement at the time the cursor is OPENed, except for those variables that include subscripts. In the latter case, **INFORMIX-4GL** evaluates the subscript when the cursor is DECLAREd and the variable when the cursor is OPENed.

    In the following example, **INFORMIX-4GL** selects rows where the value in the **customer_num** column equals 106:

```
LET a = 101
DECLARE q_curs CURSOR FOR
   SELECT * FROM orders WHERE customer_num = a
LET a = 106
OPEN q_curs
```

    In this example, **INFORMIX-4GL** selects rows where the value in the **customer_num** column equals a[5]:

```
LET i = 5
DECLARE q_curs CURSOR FOR
   SELECT * FROM orders WHERE customer_num = a[i]
LET i = 2
OPEN q_curs
```

11. You cannot DECLARE a cursor for an INSERT statement that contains an embedded SELECT statement.

12. If you DECLARE a cursor for an INSERT statement that includes a VALUES clause containing only constants, RDSQL does not create a buffer but merely keeps count of the number of inserts. Such inserts are never flushed as the result of a PUT statement. Flushing occurs when you issue a FLUSH or CLOSE cursor statement.

Examples

```
DECLARE scurs CURSOR FOR
   SELECT * FROM customer


DECLARE ucurs CURSOR FOR
   SELECT * FROM customer WHERE customer_num > 110
   FOR UPDATE OF fname, lname


DECLARE icurs CURSOR FOR
   INSERT INTO stock
      VALUES (stock_no, man_code, descr, u_price,
      unit, u_desc)


DECLARE s_curs SCROLL CURSOR FOR
   SELECT * FROM orders
   WHERE customer_num = 104
```

Related Statements

CLOSE, FETCH, FLUSH, FOREACH, OPEN, PREPARE, PUT, DELETE, SELECT, UPDATE

DEFER

Overview

Use the DEFER statement to keep INFORMIX-4GL from terminating your program whenever a user presses the INTERRUPT key (usually DEL or CONTROL-C) or the QUIT key (usually CONTROL-\).

Syntax

DEFER {INTERRUPT | QUIT}

Explanation

DEFER	is a required keyword.
INTERRUPT	is an optional keyword.
QUIT	is an optional keyword.

Notes

1. In the absence of the DEFER statement, your program will stop immediately whenever INTERRUPT or QUIT is pressed.

2. The DEFER statement sets a global flag (**int_flag** for INTERRUPT and **quit_flag** for QUIT) to non-zero whenever INTERRUPT or QUIT is pressed by the user. It is the programmer's responsibility to reset the flags to zero.

3. If the DEFER INTERRUPT statement has been executed and the user subsequently enters an INTERRUPT during an INPUT statement, program control will leave the INPUT statement and **int_flag** will be set. This applies to the CONSTRUCT, INPUT ARRAY, and PROMPT statements, as well.

4. The DEFER INTERRUPT statement may occur only once in a program and then only in the MAIN routine. After being executed, it remains in effect for the duration of the program. This applies to the DEFER QUIT statement, as well.

Examples

```
DEFER INTERRUPT
```

Related Statements

WHENEVER

DEFINE

Overview

Use the DEFINE statement to define the identifiers used in your program and to set aside adequate memory for each program variable.

Syntax

```
DEFINE  variable-list {type
                      | LIKE table.column
                      | RECORD {LIKE table.*
                               | variable-list type [, ... ]
                               END RECORD}[, ... ]
```

Explanation

DEFINE — is a required keyword.

variable-list — is a list of one or more program variable identifiers.

type — is one of the data types listed here and defined in Chapter 1.

SMALLINT	SERIAL
INTEGER	CHAR[(*b*)]
DECIMAL[(*m*[, *n*])]	DATE
SMALLFLOAT	MONEY[(*m*[, *n*])]
FLOAT	ARRAY[*i*, *j*, ...] OF *type*
RECORD	

LIKE — is a keyword you use to type a variable indirectly.

table.column — is the full identifier for a column in the current database. The DATABASE statement must precede DEFINE statements using indirect typing.

RECORD	is a data type that describes a set of variables of possibly differing database data types.
table	is the name of a database table. The DATABASE statement must precede DEFINE statements using indirect typing.
END RECORD	are optional keywords.

Notes

1. Elements of record variables are addressed as *record_name.column_name.*

2. See the "Data Types" section in Chapter 1 for a full discussion of variable types.

Examples

```
DEFINE p_customer RECORD LIKE customer.*,
       p_orders RECORD
            order_num LIKE orders.order_num,
            order_date LIKE orders.order_date,
            po_num LIKE orders.po_num,
            ship_instruct LIKE orders.ship_instruct
            END RECORD,
       p_stock ARRAY[30] OF RECORD
            s_num LIKE stock.stock_num,
            m_code LIKE stock.manu_code
            END RECORD,
       stock_tot SMALLINT
```

Related Statements

GLOBALS, LET

DELETE

Overview

Use the DELETE statement to delete one or more rows from a table.

Syntax

DELETE FROM *table-name*
[WHERE {*condition* | CURRENT OF *cursor-name*}]

Explanation

DELETE FROM	are required keywords.
table-name	is the name of the table from which you want to delete rows.
WHERE	is a keyword.
condition	is a condition of a standard WHERE clause (see the SELECT statement for further information). **RDSQL** deletes all rows that satisfy the condition in the WHERE clause.
CURRENT OF	are keywords.
cursor-name	is the **RDSQL** identifier of a previously DECLAREd and positioned cursor.

Notes

1. If you do not use the BEGIN WORK and COMMIT WORK or ROLLBACK WORK statements, each **RDSQL** statement that you execute is treated as a single transaction if your database has transactions. Because each row affected by the DELETE statement within a transaction is locked for the duration of the transaction, a single DELETE statement that affects a large number of rows locks the rows until the entire operation is completed. If the number of rows affected is very large, you may exceed the limits placed by your operating system on the maximum number of simultaneous locks. If this occurs, you may want either to reduce the scope of the DELETE statement or to lock the entire table before executing the statement.

   See the section "Locking" in Chapter 2 for a more detailed description of table-level and row-level locking in **INFORMIX-4GL**.

2. To use the WHERE CURRENT OF option, you must have previously DECLAREd the *cursor-name* with the FOR UPDATE option.

3. If you use the CURRENT OF option, DELETE removes the row of the active set at the current position of the cursor. The cursor is left pointing between the remaining rows and cannot be used for DELETE or UPDATE until it is repositioned with a FETCH statement.

4. If your database has transactions and you use the WHERE CURRENT OF clause, you must execute the DELETE statement within a transaction.

Examples

```
DELETE FROM items
    WHERE order_num = onum

DELETE FROM orders
    WHERE CURRENT OF query_cursor
```

These statements remove all items belonging to the order number set in the program variable **onum** from the **items** table and remove the row from the **orders** table pointed to by the cursor **query_cursor**.

Related Statements

DECLARE, INSERT, UPDATE

DISPLAY

Overview

Use the DISPLAY statement to display data on the screen.

Syntax

```
DISPLAY {BY NAME variable-list |
        variable-list [TO {field-list | screen-record[[n]].*} [, ...]
                      | AT row, column]}
        [ATTRIBUTE(attribute-list)]
```

Explanation

DISPLAY	is a required keyword.
BY NAME	are optional keywords you use to have **INFORMIX-4GL** match variable names to screen field names.
variable-list	is a list of one or more program variables, separated by commas.
TO	is an optional keyword.
field-list	is a list of one or more screen field names.
screen-record	is the identifier of a collection of field names defined in a form specification as a SCREEN RECORD.
n	is an optional integer that specifies the row of a screen array (beginning with 1) where the *variable-list* should be displayed. Enclose *n* in brackets.

AT	is an optional keyword.
row	is an integer variable or constant indicating a row of the screen or current window.
column	is an integer variable or constant indicating a column of the screen or current window.
ATTRIBUTE	is an optional keyword.
attribute-list	is a list of one or more screen display attributes.

Notes

1. The DISPLAY statement (without the BY NAME, TO, or AT keywords) can be used to display *variable-list* on the next line. You can use CLIPPED, USING, and COLUMN to control the format of your DISPLAY statements.

2. Changing the data stored in program variables has no effect on the screen until this statement is executed.

3. The DISPLAY BY NAME option selects the screen fields based on the identity of the program variable name and the screen field name. **INFORMIX-4GL** uses only the suffix portion of the name of the program variable and the screen field. **INFORMIX-4GL** returns an error for this option unless the suffixes are unique and unambiguous.

4. The DISPLAY TO *screen-record*[*n*].* option permits the display of program variables to the n^{th} row of a screen array. Values displayed in a screen array can be moved with the SCROLL statement.

5. **INFORMIX-4GL** displays numeric variables right justified and character variables left justified in the screen field.

6. If a displayed character value does not fit in the field, **INFORMIX-4GL** truncates the value. If a displayed numeric value does not fit in the field, **INFORMIX-4GL** fills the field with asterisks (*).

7. The DISPLAY AT statement can be used to display *variable-list* at a specified location on the screen or in the current window. You can use CLIPPED and USING to control the format of DISPLAY AT statements.

8. The coordinates start with row 1 and column 1 in the upper left corner of the screen or current window. The row coordinates increase as you go down the screen or current window and the column coordinates increase as you move from left to right. On a standard terminal screen, the lower right corner has the coordinates (24, 80).

9. An error occurs if either *row* or *column* exceeds the dimensions of the screen or current window; **INFORMIX-4GL** sets **status** to a negative value.

10. If, when you use the AT option, the last element of *variable-list* is a NULL CHAR value, **INFORMIX-4GL** clears to the end of the line. Specifically,

```
DISPLAY "" AT n,1
```

clears the n^{th} line of the screen or current window.

11. If your program includes a DISPLAY statement followed by a DISPLAY AT statement, **INFORMIX-4GL** clears the screen or current window before the second display.

12. The attributes listed in *attribute-list* apply to all the field names in *field-list.*

13. If you use the ATTRIBUTE clause, none of the default attributes listed in **syscolatt** or in the form specification file for fields in *field-list* apply.

14. The following list shows the allowed screen attributes in the ATTRIBUTE clause:

WHITE = NORMAL	REVERSE
YELLOW = BOLD	BLINK
MAGENTA = BOLD	UNDERLINE
RED = BOLD	
CYAN = DIM	
GREEN = DIM	
BLUE = DIM	
BLACK = INVISIBLE	

On color terminals, NORMAL is interpreted as WHITE, BOLD as RED, DIM as BLUE, and INVISIBLE as BLACK. If you have a **colornames** file, you can use the color names listed there (see Chapter 3).

Note: The termcap entry for some terminals includes an *sg#1* setting. On these terminals, the first character of *variable-list* is deleted if you use the DISPLAY AT statement with any display attribute to display text starting in the first column of a line. To be safe, make sure that the first character of the *variable-list* is a blank when using this statement with a display attribute.

Examples

```
DISPLAY BY NAME lname, fname
    ATTRIBUTE(REVERSE, BLUE)

DISPLAY "There are ", num,
    " items in the list" AT 12,1

DISPLAY add_cust.* TO sc_addr[4].*
```

Related Statements

INPUT, DISPLAY FORM

DISPLAY ARRAY

Overview

Use the DISPLAY ARRAY statement to display a program array to a screen array and to permit scrolling through the array.

Syntax

```
DISPLAY ARRAY record-array TO screen-array.*
        [ATTRIBUTE (attribute-list)]
        {ON KEY (key-list)
          statement
            ...
        [EXIT DISPLAY]
        ...
END DISPLAY | [END DISPLAY]}
```

Explanation

DISPLAY ARRAY are required keywords.

record-array	is a program array name. Usually, *record-array* is an array of records.
TO	is a required keyword.
screen-array	is the name of a screen record, defined in a form specification file, that corresponds to the fields in a row of a screen array.
ATTRIBUTE	is an optional keyword.
attribute-list	is a list of one or more screen-display attributes.
ON KEY	are optional keywords.

key-list	is usually a list of one or more function or CONTROL key designations. It may also include **ESCAPE** (if you have specified another key as the ACCEPT KEY in the OPTIONS statement) or INTERRUPT (if you have executed a DEFER INTERRUPT statement).
statement	is an **INFORMIX-4GL** statement.
EXIT DISPLAY	is a statement that causes **INFORMIX-4GL** to exit the DISPLAY ARRAY statement.
END DISPLAY	is a statement that terminates a DISPLAY ARRAY statement. It is required only when an ON KEY clause is used or when the DISPLAY ARRAY statement appears as the last statement in a clause and is followed by an ON KEY clause.

Notes

1. You must call **set_count()** with the number of filled rows in *record-array* prior to executing DISPLAY ARRAY.

2. The user can use the **DOWN ARROW** key to move the cursor down one row at a time and to scroll at the bottom of the screen array, the **UP ARROW** key to move the cursor up one row at a time and to scroll at the top of the screen array, **F3** to scroll to the next page, and **F4** to scroll to the previous page. The function keys can be changed in the OPTIONS statement to other keys and, on UNIX systems, must be defined properly in the **termcap** file.

3. The user can exit the DISPLAY ARRAY statement by pressing **ESCAPE** or the key specified as the ACCEPT KEY in the OPTIONS statement. The program should tell the user to do this.

4. The following conditions require that an END DISPLAY statement appear in your program:

   - The DISPLAY ARRAY statement includes one or more ON KEY clauses

   - The DISPLAY ARRAY statement appears as the last statement in a clause (within an INPUT statement, for example) and is followed by an ON KEY clause

5. By default, **INFORMIX-4GL** displays numeric variables right justified and character variables left justified in the screen field.

6. If a displayed character value is larger than the field, **INFORMIX-4GL** truncates the value. If a displayed numeric value is larger than the field, **INFORMIX-4GL** fills the field with asterisks (*).

7. The attributes listed in *attribute-list* apply to all the fields in *screen-array*.

8. If you use the ATTRIBUTE clause, none of the default attributes listed in **syscolatt** or in the form specification file for the fields in *screen-array* will apply.

9. The following list shows the screen attributes allowed in the ATTRIBUTE clause:

WHITE = NORMAL	REVERSE
YELLOW = BOLD	BLINK
MAGENTA = BOLD	UNDERLINE
RED = BOLD	
CYAN = DIM	
GREEN = DIM	
BLUE = DIM	
BLACK = INVISIBLE	

   On color terminals, NORMAL is interpreted as WHITE, BOLD as RED, DIM as BLUE, and INVISIBLE as BLACK. If you have a **colornames** file, you can use the color names listed there (see Chapter 3).

10. INFORMIX-4GL passes control to the statements following an ON KEY clause when the user presses a key in *key-list*. After executing the statements in the ON KEY clause, INFORMIX-4GL resumes the display with the cursor in the same location as before the ON KEY break, unless NEXT FIELD and EXIT DISPLAY are implemented.

11. The notation for functions keys is F1, F2, F3,... F36. The notation for CONTROL keys is CONTROL-*X* where *X* is any letter except A, D, H, L, R, or X. The notation for ESCAPE is ESC or ESCAPE. The notation for the INTERRUPT key is INTERRUPT.

12. By default, both ESCAPE and INTERRUPT are exits from the DISPLAY ARRAY statement. If the DEFER INTERRUPT statement has been executed, an INTERRUPT causes INFORMIX-4GL to set **int_flag** to non-zero and terminates the DISPLAY ARRAY statement (unless INTERRUPT has been redefined in an ON KEY clause). Otherwise, an INTERRUPT causes an immediate program stop.

13. You can include the following keys in a *key-list* under the stated conditions:

    - ESCAPE if you have specified another key as the ACCEPT KEY in the OPTIONS statement.

    - F3 if you have specified another key as the NEXT KEY in the OPTIONS statement.

    - F4 if you have specified another key as the PREVIOUS KEY in the OPTIONS statement.

    - INTERRUPT if you have executed a DEFER INTERRUPT statement. (When the user presses the INTERRUPT key under these conditions, INFORMIX-4GL executes the statements in the ON KEY clause and sets **int_flag** to non-zero, but does not terminate the DISPLAY ARRAY statement.)

Do not use the following keys in a *key-list*:

- **CONTROL-A, CONTROL-D, CONTROL-H, CONTROL-L, CONTROL-R,** or **CONTROL-X** since these CONTROL keys are reserved for editing functions in the CONSTRUCT, INPUT, and INPUT ARRAY statements.

- Other keys that have special meaning for your operating system.

14. Do not execute PROMPT, INPUT, or INPUT ARRAY statements within the ON KEY clause of a DISPLAY ARRAY statement. However, you can call a function that executes one of these statements.

15. If you execute a statement in an ON KEY clause that changes the value of **arr_curr**, **scr_line**, or **arr_count**, the new value remains for the duration of the clause (unless changed again). After **INFORMIX-4GL** executes the last statement in the clause, it restores the original value. For example, if the DISPLAY ARRAY statement in the following ON KEY clause changes the value of **arr_curr** from 1 to 4, the value remains 4 for the rest of the ON KEY clause. After **INFORMIX-4GL** executes the MESSAGE statement, it resets **arr_curr** to 1.

```
DISPLAY ARRAY p_items TO s_items.*
   ON KEY (F6)
      DISPLAY ARRAY p_stock TO s_stock.*
      END DISPLAY
      LET pa_curr = arr_curr()
      MESSAGE "The current value of arr_curr is: ", pa_curr
END DISPLAY
```

Examples

```
DISPLAY ARRAY pa_array TO sc_array.*


DISPLAY ARRAY p_items TO s_items.*
   ON KEY (CONTROL-E)
      MESSAGE "Highlight an item and ",
          "press the ACCEPT key."
END DISPLAY


INPUT BY NAME p_customer.*
   AFTER FIELD company
      ...
      DISPLAY  ARRAY pa_array TO sc_array.*
      END DISPLAY
   ON KEY (CONTROL-B)
      ...
END INPUT
```

Related Statements

DISPLAY

DISPLAY FORM

Overview

Use the DISPLAY FORM statement to display a pre-compiled screen form.

Syntax

DISPLAY FORM *form-name* [ATTRIBUTE(*attribute-list*)]

Explanation

DISPLAY FORM	are required keywords.
form-name	is an **INFORMIX-4GL** identifier that has been associated with a screen form in an OPEN FORM statement.
ATTRIBUTE	is an optional keyword.
attribute-list	is a list of one or more screen attributes that will apply to the screen form delimiters and any text outside of display fields.

Notes

1. DISPLAY FORM displays the screen form starting on the third line of the terminal screen or window. You can change the starting line for all windows (including the screen) by using the OPTIONS statement or for a specific window by using an ATTRIBUTE clause in the appropriate OPEN WINDOW statement.

2. The following list shows the screen attributes allowed in the ATTRIBUTE clause:

WHITE = NORMAL	REVERSE
YELLOW = BOLD	BLINK
MAGENTA = BOLD	UNDERLINE
RED = BOLD	
CYAN = DIM	
GREEN = DIM	
BLUE = DIM	
BLACK = INVISIBLE	

   On color terminals, NORMAL is interpreted as WHITE, BOLD as RED, DIM as BLUE, and INVISIBLE as BLACK. If you have a **colornames** file, you can use the color names listed there (see Chapter 3).

3. **INFORMIX-4GL** issues an error if a window is not large enough to display a form. See Chapter 13 of the *INFORMIX-4GL User Guide* and the OPEN WINDOW statement in this chapter for more information about displaying a form in a window.

4. A DISPLAY FORM statement is not required if you opened and displayed a form using the WITH FORM option of the OPEN WINDOW statement.

Examples

```
DISPLAY FORM order_entry

DISPLAY FORM inventory ATTRIBUTE(BLUE)
```

Related Statements

CLOSE FORM, OPEN FORM

DROP AUDIT

Overview

Use the DROP AUDIT statement to delete an audit trail file.

Syntax

DROP AUDIT FOR *table-name*

Explanation

DROP AUDIT FOR are required keywords.

table-name is the name of the table whose audit trail file you want to delete.

Notes

1. Use the DROP AUDIT statement to remove the old audit trail file when you have made a backup of your database files. Use the CREATE AUDIT statement to start a new audit trail, and then back up the table. See the section "Audit Trails" in Chapter 2 for more information.

2. You must own *table-name* or have DBA status to use the DROP AUDIT statement.

Examples

```
DROP AUDIT FOR orders
```

Related Statements

CREATE AUDIT, RECOVER TABLE

DROP DATABASE

Overview

Use the DROP DATABASE statement to delete an entire database, including all system catalogs, indexes, data, and the transaction log.

Syntax

DROP DATABASE {*database-name* | *char-variable*}

Explanation

DROP DATABASE	are required keywords.
database-name	is the name of the database you want to delete.
char-variable	is a program variable of type CHAR containing the name of the database you want to delete.

Notes

1. You must own all the tables in the database or have DBA status to run the DROP DATABASE statement.

2. The DROP DATABASE statement does not delete the database directory if there are any files in the database directory other than those created for database tables and their indexes.

3. You cannot drop the current database. You must execute the CLOSE DATABASE statement first.

4. You cannot execute the DROP DATABASE statement within a transaction; it cannot be rolled back.

5. For databases with transactions, the DROP DATABASE statement deletes the transaction log.

Examples

```
DROP DATABASE stores
```

Related Statements

CREATE DATABASE, CLOSE DATABASE

DROP INDEX

Overview

Use the DROP INDEX statement to delete an index.

Syntax

DROP INDEX *index-name*

Explanation

DROP INDEX are required keywords.

index-name is the name of the index you want to delete.

Notes

1. You must be the owner of the index or have DBA privilege to use the DROP INDEX statement.

2. Do not execute the DROP INDEX statement within a transaction; it cannot be rolled back.

Examples

```
DROP INDEX i_ordnum
```

Related Statements

CREATE INDEX

DROP SYNONYM

Overview

Use the DROP SYNONYM statement to delete a previously defined synonym.

Syntax

DROP SYNONYM *synonym*

Explanation

DROP SYNONYM are required keywords.

synonym is an **INFORMIX-4GL** identifier.

Notes

1. You must be the owner of the synonym or have DBA status to use the DROP SYNONYM statement.
2. When you compile a program containing a synonym, the synonym is replaced by the real table identifier in the compiled program. If you subsequently drop the synonym, the compiled program will still run.
3. Do not execute the DROP SYNONYM statement within a transaction; it cannot be rolled back.

Examples

```
DROP SYNONYM cust
```

Related Statements

CREATE SYNONYM

DROP TABLE

Overview

Use the DROP TABLE statement to delete a table, along with its associated indexes and data.

Syntax

DROP TABLE *table-name*

Explanation

DROP TABLE are required keywords.

table-name is the name of the table you want to delete.

Notes

1. When you delete a table, you also delete the data stored in it, the indexes on columns, any synonyms assigned to it, and any authorizations you have granted on the table. You also delete all views based on the table.
2. You cannot drop any of the system catalog tables.
3. You must be the owner of a table or have DBA privilege to use the DROP TABLE statement.
4. Do not execute the DROP TABLE statement within a transaction; it cannot be rolled back.

Examples

```
DROP TABLE customer
```

Related Statements

CREATE TABLE

DROP VIEW

Overview

Use the DROP VIEW statement to delete a view from the database.

Syntax

DROP VIEW *view-name*

Explanation

DROP VIEW are required keywords.

view-name is the identifier of a view.

Notes

1. You must be the owner of the view or have DBA status to use the DROP VIEW statement.

2. When you drop *view-name*, you also drop all views that have been defined in terms of *view-name*.

3. Do not execute the DROP VIEW statement within a transaction session; it cannot be rolled back.

4. See the section "Views" in Chapter 2 for more information.

Examples

```
DROP VIEW cust1
```

Related Statements

CREATE VIEW, DROP TABLE

ERROR

Overview

Use the ERROR statement to display an error message on the Error Line (by default, the bottom line of the screen) and to ring the terminal bell.

Syntax

ERROR *display-list* [ATTRIBUTE (*attribute-list*)]

Explanation

ERROR	is a required keyword.
display-list	is a list of one or more program variables and/or string constants (enclosed in quotation marks), separated by commas.
ATTRIBUTE	is an optional keyword.
attribute-list	is a list of one or more screen attributes, separated by commas.

Notes

1. The string generated by substituting values for the variables in *display-list* must fit on a single display line. The string is displayed in a window on the Error Line.

2. The position of the Error Line can be changed by the OPTIONS statement. The location of the Error Line is relative to the screen rather than to the current window.

3. *display-list* can contain the CLIPPED and USING functions.

4. REVERSE is the default attribute for the ERROR display. You can alter the default attribute with the ATTRIBUTE clause.

5. The following list shows the screen attributes allowed in the ATTRIBUTE clause:

WHITE = NORMAL	REVERSE
YELLOW = BOLD	BLINK
MAGENTA = BOLD	UNDERLINE
RED = BOLD	
CYAN = DIM	
GREEN = DIM	
BLUE = DIM	
BLACK = INVISIBLE	

On color terminals, NORMAL is interpreted as WHITE, BOLD as RED, DIM as BLUE, and INVISIBLE as BLACK. If you have a **colornames** file, you can use the color names listed there (see Chapter 3).

Examples

```
ERROR "There is no match for ", pattern
```

Related Statements

DISPLAY, MESSAGE, OPTIONS

EXECUTE

Overview

Use the EXECUTE statement to run a previously PREPAREd statement.

Syntax

EXECUTE *statement-id* [USING *input-list*]

Explanation

EXECUTE	is a required keyword.
statement-id	is an **RDSQL** identifier named in a previously PREPAREd statement.
USING	is an optional keyword.
input-list	is a list of program variables to be substituted as values for the question marks (?) in the statement indicated by *statement-id*. Use this option when you know the number and data types of the PREPAREd statement.

Notes

1. Once you have PREPAREd an **RDSQL** statement, you may EXECUTE it as often as you desire.
2. To use the USING clause, you must know the number of the parameters of the PREPAREd statement. The data type of each variable in *input-list* must be compatible with the value expected in the PREPAREd statement for the corresponding parameter.

3. You may not EXECUTE a PREPAREd SELECT statement. Use DECLARE with a FOREACH loop or the OPEN, FETCH, and CLOSE statements to execute a PREPAREd SELECT statement.

Examples

```
LET s1 = "UPDATE orders SET po_num = ?, order_date = ?"
PREPARE statement_1 FROM s1
EXECUTE statement_1 USING purchase_num, order_date
```

Related Statements

DECLARE, PREPARE

EXIT

Overview

Use the EXIT statement to terminate the program; to break out of a FOR, a FOREACH, or a WHILE loop; to leave the CASE statement; to leave the INPUT or INPUT ARRAY statement; or to leave a menu.

Syntax

```
EXIT {CASE | DISPLAY | FOR | FOREACH | INPUT | MENU
      | PROGRAM [(integer-expr)] | WHILE}
```

Explanation

EXIT	is a required keyword.
CASE	is an optional keyword.
DISPLAY	is an optional keyword.
FOR	is an optional keyword.
FOREACH	is an optional keyword.
INPUT	is an optional keyword.
MENU	is an optional keyword.
PROGRAM	is an optional keyword.
integer-expr	is an expression that evaluates as an integer.
WHILE	is an optional keyword.

Notes

1. You can use the CASE option only within a CASE statement, the DISPLAY option only within a DISPLAY ARRAY statement, the FOR option only within a FOR statement, the FOREACH option only within a FOREACH statement, the INPUT option only within an INPUT or INPUT ARRAY statement, the MENU option only following a COMMAND clause of a MENU statement, and the WHILE option only within a WHILE statement. In each case, program control passes to the first statement following the END CASE, END DISPLAY, END FOR, END FOREACH, END INPUT, END MENU, or END WHILE statements, respectively.

2. The PROGRAM option can be used anywhere and causes immediate program termination.

3. The optional integer argument of EXIT PROGRAM is available in the UNIX operating system as a status code.

Related Statements

CONTINUE

FETCH

Overview

Use the FETCH statement to move the cursor to a new row in the active set and to retrieve the values from that row.

Syntax

FETCH [NEXT | {PREVIOUS | PRIOR} | FIRST | LAST | CURRENT | RELATIVE *m* | ABSOLUTE *n*] *cursor-name*
[INTO *variable-list*]

Explanation

FETCH	is a required keyword.
NEXT	is an optional keyword indicating the next row in the active list. NEXT is the default.
PREVIOUS	is an optional keyword indicating the prior row in the active list.
PRIOR	is an optional keyword that is synonymous with PREVIOUS.
FIRST	is an optional keyword indicating the first row of the active list.
LAST	is an optional keyword indicating the last row of the active list.
CURRENT	is an optional keyword indicating the current row of the active list.

RELATIVE *m*	is an optional keyword indicating the *m*th row relative to the current cursor position in the active list. *m* can be either an integer or a program variable, and can be either positive or negative.
ABSOLUTE *n*	is an optional keyword indicating the *n*th row in the active list. *n* can be either an integer or a program variable.
cursor-name	is an **RDSQL** identifier of a previously DECLAREd cursor.
INTO	is an optional keyword.
variable-list	is a list of program variables that contains the column values of the row pointed to by *cursor-name*.

Notes

1. FETCH NEXT is the default condition.

2. You must DECLARE a SCROLL cursor before issuing a FETCH PRIOR, FETCH FIRST, FETCH LAST, FETCH CURRENT, FETCH RELATIVE *m*, or FETCH ABSOLUTE *n* statement.

3. If the SELECT statement associated with the cursor has an INTO clause, there must be no INTO clause in any FETCH statement referring to that cursor. If the SELECT statement has no INTO clause, the FETCH statement must have one.

4. You can FETCH into a program array element only by using an INTO clause in the FETCH statement. Do not refer to an array element in the *SELECT-statement* of a DECLARE statement.

5. When the cursor points to the last row in the active set, a subsequent FETCH NEXT causes **INFORMIX-4GL** to return a "not found" code (**status** = NOTFOUND).

6. If you issue a FETCH PRIOR (PREVIOUS) statement when the cursor points to the first row in the active set, **INFORMIX-4GL** returns a "not found" code (**status** = NOTFOUND).

7. If you execute a FETCH ABSOLUTE *n* statement where no *n*th row exists in the active set, **INFORMIX-4GL** returns a "not found" code (**status** = NOTFOUND).

8. If you execute a FETCH RELATIVE *m* statement where no *m*th row exists in the active set, **INFORMIX-4GL** returns a "not found" code (**status** = NOTFOUND).

Examples

```
FETCH query_curs INTO cnum, lname

FETCH PREVIOUS q_curs INTO orders.*

FETCH LAST q_curs INTO orders.*

FETCH ABSOLUTE 8 q_curs INTO orders.*

FETCH RELATIVE -10 q_curs INTO orders.*
```

Related Statements

OPEN, CLOSE, DELETE, UPDATE

FINISH REPORT

Overview

Use the FINISH REPORT statement to cause **INFORMIX-4GL** to finish processing a report.

Syntax

FINISH REPORT *report-name*

Explanation

FINISH REPORT	are required keywords.
report-name	is the identifier of a report.

Notes

1. You must use the FINISH REPORT statement to let **INFORMIX-4GL** know that there are no more statements to be included in the report processing.

Examples

```
FINISH REPORT cust_ords
```

Related Statements

OUTPUT TO REPORT, START REPORT

FLUSH

Overview

Use the FLUSH statement to force **RDSQL** to insert the buffered rows into the database without closing the cursor.

Syntax

FLUSH *cursor-name*

Explanation

FLUSH	is a required keyword.
cursor-name	is the name of a cursor that has been DECLAREd for an INSERT statement.

Notes

1. The global variables **status** (whose value is taken from SQLCA.SQLCODE) and SQLCA.SQLERRD[3] indicate the result of each FLUSH statement. If **RDSQL** successfully inserts all the buffered rows into the database, it sets **status** to zero and sets SQLCA.SQLERRD[3] to the number of rows inserted. If **RDSQL** is unsuccessful in its attempt to insert the rows into the database, it sets **status** to a negative number (specifically, the number of the error message) and sets SQLCA.SQLERRD[3] to the number of rows successfully inserted into the database.

2. You can use the FLUSH statement to force the insertion. You cannot delay insertion by not using the FLUSH statement. **RDSQL** automatically flushes the buffer when it is full.

3. Insert cursors that contain only constants in the values clause are not buffered. **RDSQL** keeps a count of the number of rows to be inserted into the database, and the database is updated only when you issue a FLUSH or CLOSE statement.

4. If you exit a program without closing the cursor, the buffer is left unflushed. Rows inserted into the buffer and remaining since the last flush are lost. Do not expect the end of program to close the cursor and flush the buffer.

Examples

```
FLUSH icurs
```

Related Statements

CLOSE, DECLARE, OPEN, PUT

FOR

Overview

Use the FOR statement to cause a sequence of statements to be executed a specified number of times.

Syntax

```
FOR integer-var = integer-expr TO integer-expr
                  [STEP integer-expr]
    statement
    ...
    [CONTINUE FOR]
    ...
    [EXIT FOR]
    ...
END FOR
```

Explanation

FOR	is a required keyword.
integer-var	is a program variable of type INTEGER or SMALLINT that serves as a counter.
integer-expr	is an expression that evaluates to an INTEGER or a SMALLINT.
TO	is a required keyword.
STEP	is an optional keyword.
statement	is an **INFORMIX-4GL** statement.
CONTINUE FOR	is an optional statement.
EXIT FOR	is an optional statement.
END FOR	are required keywords.

Notes

1. The FOR statement repeats the sequence of statements up to the END FOR as *integer-var* takes on the values of the first *integer-expr* TO the second *integer-expr* in STEPs of the third *integer-expr*. The default STEP is 1.

2. The CONTINUE FOR statement interrupts the sequence and causes the program control to return to the top of the sequence and to increment and test the counter *integer-var*.

3. The EXIT FOR statement interrupts the sequence and causes the program control to jump to the first statement following the END FOR keywords.

4. If *integer-var* is greater than the TO *integer-expr* upon entry and the STEP value is positive, none of the statements up to END FOR is executed.

5. The STEP value may be negative.

Examples

```
DEFINE order_total MONEY(8),
       i INTEGER

LET order_total = 0.00
FOR i = 1 TO ARR_COUNT()
   LET order_total = order_total
                     + p_items[i].total_price
END FOR
```

Related Statements

CONTINUE, EXIT, FOREACH, WHILE

FOREACH

Overview

Use the FOREACH statement to cause a sequence of statements to be executed for each row returned from a query.

Syntax

```
FOREACH cursor-name [INTO variable-list]
    statement
    ...
    [CONTINUE FOREACH]
    ...
    [EXIT FOREACH]
    ...
END FOREACH
```

Explanation

FOREACH	is a required keyword.
cursor-name	is the name of a cursor that previously was DECLAREd.
INTO	is an optional keyword.
variable-list	is a list of one or more program variables, separated by commas.
CONTINUE FOREACH	is an optional statement.
EXIT FOREACH	is an optional statement.
END FOREACH	are required keywords.

Notes

1. The FOREACH statement repeats the sequence of statements up to END FOREACH for each row returned by the query associated with *cursor-name*. The FOREACH statement OPENs the cursor and performs successive FETCHes until the active list of the cursor is exhausted.

2. The INTO clause is required only if there is no INTO clause in the SELECT statement associated with *cursor-name*, and vice versa. It lists the variables into which **INFORMIX-4GL** places the values returned by the query.

3. When fetching into a program array, you must place the INTO clause on the FOREACH statement and not on the *SELECT-statement* of a DECLARE statement.

4. The CONTINUE FOREACH statement interrupts the sequence and causes program control to return to the top of the sequence and to fetch the next row of the query.

5. The EXIT FOREACH statement interrupts the sequence and causes program control to jump to the first statement following the END FOREACH keywords.

6. If the query returns no rows, none of the statements up to the END FOREACH is executed, and program control passes immediately to the first statement following END FOREACH. If you need to know whether any rows were returned, you can set up a flag or a counter as in the following example.

7. The FOREACH statement performs an implied OPEN statement. You cannot use the FOREACH statement if the OPEN statement must have a USING clause to define unknown parameters in the SELECT statement.

8. If your database has transactions, the FOREACH statement must appear inside a transaction.

Examples

```
PROMPT "Enter cut-off date for query: "
   FOR o_date

DECLARE q_curs CURSOR FOR
   SELECT order_num, o.customer_num, company
      FROM orders o, customer c
      WHERE o.customer_num = c.customer_num
         AND order_date < o_date



LET counter = 0

FOREACH q_curs INTO ord_num, cust_num, comp
    LET counter = counter + 1
    CALL scan(ord_num, cust_num, comp)
END FOREACH

IF counter = 0 THEN
   ERROR "No orders before ", o_date
END IF
```

Related Statements

CONTINUE, EXIT, FETCH, FOR, OPEN, WHILE

FUNCTION

Overview

Use the FUNCTION statement to define a function.

Syntax

```
FUNCTION function-name ([argument-list])
    statement
    ...
    [RETURN expr-list]
    ...
END FUNCTION
```

Explanation

FUNCTION	is a required keyword.
function-name	is a program identifier for the name of the function.
argument-list	is a list of one or more variables, separated by commas.
statement	is an **INFORMIX-4GL** statement.
RETURN	is an optional keyword that causes program control to return to the calling program.
expr-list	is a list of zero or more expressions that yield the values returned by *function-name*.
END FUNCTION	are required keywords that terminate the FUNCTION statement.

Notes

1. All arguments are passed by value.

2. You must DEFINE the arguments to the function, as well as other variables used locally within the function.

3. When passing an entire record as an argument to a function, use the .* notation. In the FUNCTION statement, however, use only a record name that you define within the FUNCTION routine. Thus you call the following function with the notation:

```
CALL example(orders.*)
...
FUNCTION example(r)
   DEFINE r RECORD LIKE orders.*
   ...
END FUNCTION
```

4. Variables DEFINEd within a function are local to the function.

5. If *function-name* returns a single value, it may be used in an expression in place of a variable. Otherwise, it must be CALLed.

6. The number and data type of expressions in *expr-list* must be the same, and in the same order, as the number and type of program variables in the RETURNING clause of the CALL statement that calls the function.

Related Statements

CALL

GLOBALS

Overview

Use the GLOBALS statement to DEFINE one or more variables to be global variables or to refer to the file where global variables are DEFINEd.

Syntax

```
GLOBALS {"filename" |
    DEFINE-statement
    ...
END GLOBALS}
```

Explanation

GLOBALS	is a required keyword.
filename	is the pathname of the file where the global variables are defined.
DEFINE-statement	is a DEFINE statement for the global variable.
END GLOBALS	are required keywords that terminate the GLOBALS statement when variables are DEFINEd.

Notes

1. You may have at most one GLOBALS statement where global variables are defined. This statement may be in a separate file or in the file containing the MAIN program.

2. The GLOBALS *filename* statement must occur earlier in every file than any function that makes reference to a global variable.

3. The GLOBALS statement must lie outside of the MAIN and any FUNCTION or REPORT routines.

4. If any global variable is DEFINEd LIKE a database column, the DATABASE statement naming the database must precede the GLOBALS statement.

Examples

```
DATABASE stores
GLOBALS
   DEFINE p_customer RECORD LIKE customer.*
   ...
END GLOBALS


GLOBALS "d4_globals.4gl"
```

Related Statements

DEFINE

GOTO

Overview

Use the GOTO statement to transfer program control unconditionally to a designated point.

Syntax

GOTO *label-id*

Explanation

GOTO is a required keyword.

label-id is a program identifier.

Notes

1. Enter labels into the program preceded by the keyword LABEL and with an appended colon:

```
IF customer_num < 0 THEN
   GOTO abort
END IF
statement
   . . .
LABEL abort:
   statement
```

2. Following a GOTO *label* statement, program control is transferred to the statement following LABEL *label-id*:.

3. *label-id* must be in the function, report, or main routine in which the GOTO is used. You may not transfer into or out of a function or a report with a GOTO statement.

Related Statements

LABEL

GRANT

Overview

Use the GRANT statement to specify user access privileges to a database or to the tables and views in a database. Access privileges are relevant only on multi-user systems.

Syntax

GRANT *tab-privilege* ON *table-name* TO {PUBLIC | *user-list*}
[WITH GRANT OPTION]

GRANT *db-privilege* TO {PUBLIC | *user-list*}

Explanation

GRANT — is a required keyword.

tab-privilege — is one or more of the following table-level access types (multiple privileges must be separated by commas):

ALTER	Add or delete columns or modify data types of columns
DELETE	Delete rows
INDEX	Create indexes
INSERT	Insert rows
SELECT [(*cols*)]	Retrieve data from specified columns

	UPDATE [(*cols*)]	Change values in specified columns
	ALL [PRIVILEGES]	All of the above

SELECT and UPDATE take column names as arguments, allowing you to specify columns that the user may select or update. Separate column names with commas.

The keyword PRIVILEGES following ALL is optional.

ON — is a required keyword.

table-name — is the name of the table or view for which you are granting access privileges.

TO — is a required keyword.

PUBLIC — is the keyword you use to specify access privileges for all users.

user-list — is a list of login names (UNIX systems) for the users to whom you are granting access privileges. You can enter one login name or a series of login names, separated by commas.

WITH GRANT OPTION — are optional keywords.

db-privilege — is one of the following database-level access types:

CONNECT	Allows access to database tables without permission to create permanent tables and indexes.

RESOURCE	Allows access to database tables with permission to create permanent tables and indexes.
DBA	Allows full database administrator privileges.

Notes

1. You can grant privileges only on tables or views you create or on tables or views for which you have been given GRANT OPTION.

2. If you do not enter any column names, the SELECT or UPDATE access that you grant applies to all columns.

3. When you GRANT table-level privileges to another user using the WITH GRANT OPTION phrase, you give that user the power to GRANT the same privileges to another user.

4. The CONNECT privilege allows the recipient to interact with the existing tables of the database with all the table-level access privileges except ALTER. The CONNECT privilege alone forbids the recipient from creating tables (except temporary tables) and indexes. The CONNECT recipient may create views.

5. The RESOURCE privilege includes the CONNECT privilege and adds the permission to create tables and indexes.

6. The DBA privilege includes the RESOURCE privilege, as well as the ability to alter system catalogs, to drop, start, and to roll forward the database, and to grant and revoke CONNECT, RESOURCE, and DBA privileges to and from other users.

7. When you create a database, you are the Database Administrator and have DBA privileges.

8. Do not execute the GRANT statement within a transaction; it cannot be rolled back.

9. The most restrictive privileges always take precedence. For example, if you grant RESOURCE privileges to a user but do not grant INDEX privileges at the table level, that user is not able to create indexes for that table. Similarly, if you grant a user CONNECT privileges and table-level INDEX privileges, that user is still prevented from using the CREATE INDEX statement.

10. You can grant DELETE, INSERT, or UPDATE privileges only on a simple view. You can grant SELECT privilege on a simple or complex view.

Examples

When you grant CONNECT privileges to PUBLIC, all table-level privileges, except ALTER, are granted to everyone. The situation is the same as if you ran the following statements for each of the tables in the database:

```
GRANT ALL ON customer TO PUBLIC

REVOKE ALTER ON customer FROM PUBLIC
```

To restrict access privileges on the table level, you must revoke all privileges and then grant those you want:

```
REVOKE ALL ON customer FROM PUBLIC

GRANT ALL ON customer TO joe, mary
GRANT SELECT (fname, lname, company, city)
    ON customer TO PUBLIC
```

Related Statements

REVOKE

IF

Overview

Use the IF statement to execute one or more statements conditionally.

Syntax

```
IF Boolean-expr
    THEN
    statement
     ...
    [ELSE
    statement
     ...]
END IF
```

Explanation

IF	is a required keyword.
Boolean-expr	is an expression that can be true or false.
THEN	is a required keyword.
statement	is any **INFORMIX-4GL** statement, including another IF statement.
ELSE	is an optional keyword.
END IF	are required keywords.

Notes

1. If *Boolean-expr* is true, the statements following THEN down to an optional ELSE (if present) or to END IF (if there is no ELSE) are executed.

2. If *Boolean-expr* is false, the statements following THEN down to an optional ELSE (if present) or to END IF (if there is no ELSE) are skipped. If an ELSE is present, the statements following the ELSE are executed.

3. If *Bool-expr* evaluates as unknown because the expression contains NULL values, the IF statement behaves as though it were false.

Examples

```
IF p_index <= 1 THEN
   ERROR "No more stock items in this direction"
ELSE
   LET p_index = p_index - 1
   DISPLAY dp_stock[p_index].* TO s_stock.*
END IF
```

Related Statements

CASE

INITIALIZE

Overview

Use the INITIALIZE statement to initialize a program variable.

Syntax

INITIALIZE *variable-list* [LIKE *column-list* | TO NULL]

Explanation

INITIALIZE	is a required keyword.
variable-list	is a list of one or more program variables, separated by commas.
LIKE	is an optional keyword.
column-list	is a list of one or more column names, separated by commas.
TO NULL	are optional keywords.

Notes

1. There must be as many entries in *column-list* as there are variables in *variable-list*.

2. You must use a table name prefix in the designation of column names.

3. You may use the * notation in either list.

4. The INITIALIZE statement uses default values stored in the database dictionary **syscolval** to initialize individual program variables.

5. If there is no default value for a column in **syscolval**, the variable corresponding to that column will be assigned a NULL value.

6. The TO NULL option requests initialization with the appropriate NULL value for each variable.

7. You may use the **upscol** utility to create and to update the values in **syscolval** (see Appendix E).

Examples

```
INITIALIZE var1, var2, var3
   LIKE tab1.col1, tab1.col2, tab1.col3

INITIALIZE v_cust.* LIKE customer.*

INITIALIZE v_orders.* TO NULL
```

Related Statements

LET

INPUT

Overview

Use the INPUT statement to assign values to program variables from data entered by the user into screen fields.

Syntax

```
INPUT {BY NAME variable-list [WITHOUT DEFAULTS] |
       variable-list [WITHOUT DEFAULTS]
       FROM {field-list | screen-record[[n]].*} [,...]}
       [HELP help-number]
              [{BEFORE FIELD field-sublist
              | AFTER {FIELD field-sublist | INPUT}
              | ON KEY (key-list)}
                     statement
                     ...
                     [NEXT FIELD field-name]
                     ...
                     [EXIT INPUT]
                     ...
              ...
END INPUT]
```

Explanation

INPUT	is a required keyword. The INPUT statement allows the user to change the data displayed on the screen.
BY NAME	are optional keywords you use to have **INFORMIX-4GL** match variable names to screen field names.
variable-list	is a list of one or more program variables, separated by commas.
WITHOUT DEFAULTS	are optional keywords.

FROM	is a required keyword.
field-list	is a list of one or more field names, separated by commas.
screen-record	is the identifier of a collection of field names defined in a form specification as a SCREEN RECORD.
HELP	is an optional keyword.
help-number	is an integer that identifies the help message for this INPUT statement in the help file set in the OPTIONS statement.
BEFORE	is an optional keyword.
FIELD	is an optional keyword.
field-sublist	is a list of one or more fields either explicitly or implicitly referred to in the INPUT statement.
AFTER	is an optional keyword.
INPUT	is an optional keyword.
ON KEY	are optional keywords.
key-list	is usually a list of one or more function or CONTROL key designations. The list can also include ESCAPE (if you have specified another key as the ACCEPT KEY in the OPTIONS statement) and INTERRUPT (if you have executed a DEFER INTERRUPT statement).
statement	is any **INFORMIX-4GL** statement.
NEXT FIELD	is an optional statement that causes the cursor to move immediately to the next field.

field-name is the field name to which the cursor should move.

EXIT INPUT is an optional statement directing INFORMIX-4GL to leave the INPUT statement immediately.

END INPUT is a statement that terminates an INPUT statement. It is required only when a BEFORE, an AFTER, or an ON KEY clause is used.

Notes

1. When you execute the INPUT statement with the WITHOUT DEFAULTS clause, INFORMIX-4GL displays the current values of *variable-list* in the screen fields. This is appropriate when you are requesting input prior to updating an existing row of a table.

2. When you execute the INPUT statement omitting the WITHOUT DEFAULTS clause, INFORMIX-4GL displays on the form the default values from the form specification (or from **syscolval** if no defaults were set in the form) and initializes the variables in *variable-list* accordingly. INFORMIX-4GL assigns NULL values to all variables for which no default has been set. This is appropriate when you are requesting input prior to inserting a new row in a table.

3. The INPUT BY NAME option selects the screen fields based on the identity of the program variable name and the screen field name. INFORMIX-4GL uses only the suffix portion of the name of the program variable and the screen field. You must use the FROM *field-list* option if the suffixes are not unique and unambiguous.

4. If you use the FROM option, the number of variables in *variable-list* must be the same as the number of field names in *field-list* or *screen-record* and of the corresponding data type.

5. The INPUT statement is terminated when the user enters **ESCAPE** or the key you specified as the ACCEPT KEY in the OPTIONS statement. For single-item INPUTs, a **RETURN** is equivalent to pressing the ACCEPT KEY. For multiple-item INPUTs, a **RETURN** after the last item is equivalent to pressing the ACCEPT KEY.

   You can use the AFTER FIELD clause on the last field to override the terminating power of the **RETURN** by setting NEXT FIELD to the first field. This has the effect of wrapping the *field-list* into a loop. Alternatively, you can set the INPUT WRAP option in the OPTIONS statement to achieve the same effect.

6. The user triggers the AFTER INPUT clause (and the set of statements that follow that clause) by attempting to terminate the INPUT statement (see the previous note). If there is an AFTER INPUT clause, the program control passes to the statements following that clause, rather than to the statements following the END INPUT clause. This feature allows the programmer to perform data validity checks before allowing the INPUT statement to terminate.

7. **INFORMIX-4GL** passes control to the statements following a BEFORE clause when the cursor enters a field in *field-list*. It passes control to the statements following an AFTER clause when the cursor leaves a field in *field-list* (after the user has pressed **RETURN**, indicating that data has been entered into the field). It passes control to the statements following an ON KEY clause after the user presses a key in *key-list*.

8. If there is no NEXT FIELD statement in the sequence of statements following a BEFORE or AFTER clause, the cursor moves to the next field in the direction that the user indicated (forward for **RIGHT ARROW**, **DOWN ARROW**, **TAB**, or **RETURN** and backward for **LEFT ARROW** or **UP ARROW**).

9. If the user triggers an ON KEY clause while entering data into a field, **INFORMIX-4GL** suspends the input of the current field while it executes the statements in the ON KEY clause. It preserves the input buffer that contains the characters the user typed before triggering the ON KEY clause, restores them when the ON KEY clause returns, and resumes the input in the same field with the cursor at the end of the buffered list of characters. If you want to use the ON KEY clause to fill the field with another value, be sure to move to a new field with the NEXT FIELD statement to prevent the INPUT statement from ignoring the new value.

10. The notation for function keys is F1 through F36. The notation for CONTROL keys is CONTROL-*X*, where *X* is any letter except A, D, H, L, R, or X. The notation for **ESCAPE** is ESC or ESCAPE. The notation for the INTERRUPT key is INTERRUPT. For UNIX systems, Appendix N describes how to verify that the termcap entry for your terminal allows **INFORMIX-4GL** to recognize function keys.

11. By default, both ESCAPE and INTERRUPT are exits from the INPUT statement. If the DEFER INTERRUPT statement has been executed, an INTERRUPT causes **INFORMIX-4GL** to set the global variable **int_flag** to nonzero and terminate the INPUT statement (unless the function of INTERRUPT has been redefined in an ON KEY clause). Otherwise, an INTERRUPT causes an immediate program stop.

12. You can use the following keys in a *key-list* under the stated conditions:

    - ESCAPE if you have specified another key as the ACCEPT KEY in the OPTIONS statement.

- INTERRUPT if you have executed a DEFER INTERRUPT statement. (When the user presses the INTERRUPT key under these conditions, INFORMIX-4GL executes the statements in the ON KEY clause and sets **int_flag** to nonzero, but does not terminate the INPUT statement.)

Do not use the following keys in a *key-list*:

- CONTROL-A, CONTROL-D, CONTROL-H, CONTROL-L, CONTROL-R, or CONTROL-X since these keys are reserved for editing functions in the INPUT statement.

- Other keys that may have special meaning on your system.

13. The function **infield**(*field*) from the function library returns true if the current field is *field* and false otherwise. It can be used to make field-dependent responses when the user presses a key specified in the *key-list* of an ON KEY clause. If you call **infield**(*field*) outside the INPUT statement, it returns a value corresponding to the field that was current when INPUT was terminated.

14. Do not execute PROMPT, INPUT, or INPUT ARRAY statements within the BEFORE, AFTER, or ON KEY clauses of an INPUT statement. However, you can call a function that executes one of these statements.

15. In addition to the RETURN, ESCAPE, INTERRUPT, and ARROW keys, the user can employ the following keys for editing during an INPUT statement:

CONTROL-A	toggles between insert and typeover mode.
CONTROL-D	deletes characters from the current cursor position to the end of the field.
CONTROL-H	moves the cursor nondestructively one space to the left inside a field.

CONTROL-L	moves the cursor nondestructively one space to the right inside a field.
CONTROL-R	redisplays the screen.
CONTROL-X	deletes the character beneath the cursor.

Examples

```
INPUT p_addr.* FROM sc_addr.*
    BEFORE FIELD fname
        MESSAGE "Enter first name of customer"
    BEFORE FIELD lname
        MESSAGE "Enter last name of customer"
    AFTER INPUT
        IF check_zip(p_addr.zipcode, p_addr.city)
                = FALSE THEN
            ERROR "Bad zip code for ", p_addr.city
            NEXT FIELD zipcode
        END IF
    ON KEY (F1)
        IF infield(city) THEN
            LET p_addr.city = "San Francisco"
            DISPLAY p_addr.city TO city
            LET p_addr.state = "CA"
            DISPLAY p_addr.state TO state
            NEXT FIELD zipcode
        END IF
END INPUT
```

Related Statements

DISPLAY, DISPLAY FORM, EXIT, INPUT ARRAY

INPUT ARRAY

Overview

Use the INPUT ARRAY statement to permit the user to enter data onto a screen array and to store the data in a program record array.

Syntax

```
INPUT ARRAY record-array [WITHOUT DEFAULTS]
        FROM screen-array.* [HELP help-number]
                [{BEFORE {ROW | INSERT| DELETE
                        | FIELD field-sublist}[, ...]
                | AFTER {ROW | INSERT | DELETE |
                        FIELD field-sublist | INPUT}[, ...]
                | ON KEY (key-list)}
                        statement
                        ...
                        [NEXT FIELD field-name]
                        ...
                        [EXIT INPUT]
                        ...
                ...
END INPUT]
```

Explanation

INPUT ARRAY	are required keywords.
record-array	is a program array name. Usually, *record-array* is an array of records.
WITHOUT DEFAULTS	are optional keywords.
FROM	is a required keyword.

screen-array	is the name of a screen record that corresponds to the fields in a row of a screen array.
HELP	is an optional keyword.
help-number	is an integer that identifies the help message for this INPUT ARRAY statement in the help file set in the OPTIONS statement.
BEFORE	is an optional keyword.
ROW	is an optional keyword.
INSERT	is an optional keyword.
DELETE	is an optional keyword.
FIELD	is an optional keyword.
field-sublist	is a list of one or more fields taken from *screen-array*.*.
AFTER	is an optional keyword.
INPUT	is an optional keyword.
ON KEY	are optional keywords.
key-list	is usually a list of one or more function or CONTROL key designations. The list may also include ESCAPE (if you have specified another key as the ACCEPT KEY in the OPTIONS statement) or INTERRUPT (if you have executed a DEFER INTERRUPT statement).
statement	is any **INFORMIX-4GL** statement.

NEXT FIELD	is an optional statement that causes the cursor to move immediately to the next field.
field-name	is a field name to which the cursor should move.
EXIT INPUT	is an optional statement directing INFORMIX-4GL to leave the INPUT ARRAY statement immediately.
END INPUT	is a statement that terminate an INPUT ARRAY statement. It is required only when a BEFORE, an AFTER, or an ON KEY clause is used.

Notes

1. When you execute the INPUT ARRAY statement with the WITHOUT DEFAULTS clause, INFORMIX-4GL displays in *screen-array* the values in *record-array*. You must call **set_count**() with the number of rows in *program-array* before the INPUT ARRAY statement. (See a later note for the definition of **set_count**().) This is appropriate when you are requesting input prior to updating an existing row of a table.

2. When you execute the INPUT ARRAY statement omitting the WITHOUT DEFAULTS clause, INFORMIX-4GL initializes the first row of *screen-array* and *record-array* with the default values from the form specification (or from **syscolval** if no defaults were set in the form). INFORMIX-4GL assigns NULL values for all variables for which no default has been set. As the cursor moves into a blank line of *screen-array*, INFORMIX-4GL initializes that line and the corresponding row of *record-array*. This is appropriate when you are requesting input prior to inserting new rows in a table.

3. The user can insert rows into the middle of existing rows of *record-array* by pressing the insert key (the default is the **F1** function key; you may alter the default with the OPTIONS statement). The user can insert rows at the bottom of existing rows in *record-array* without pressing the insert key.

4. If the user attempts to insert rows beyond the defined size of the *record-array*, **INFORMIX-4GL** displays a message that the array is full.

5. The user can delete the current row and move the following rows up to fill the gap by pressing the **F2** function key. You may alter this default with the OPTIONS statement.

6. The user can move the cursor and scroll the displayed rows using the **ARROW** keys and the function keys **F3** and **F4**.

RIGHT ARROW moves the cursor non-destructively (does not blank out the current contents) one space to the right inside a screen field. At the end of the screen field, it causes a jump to the first character position of the next screen field.

LEFT ARROW moves the cursor non-destructively one space to the left inside a screen field. At the end of the screen field, it causes a jump to the first character position of the previous screen field.

DOWN ARROW moves the cursor down one row on the screen. If the cursor was on the last row of the screen array before the user pressed **DOWN ARROW**, the displayed data moves up one row.

UP ARROW moves the cursor up one row on the screen. If the cursor was on the first row before the user pressed **UP ARROW**, the displayed data moves down one row. If the first program array row is already on the first screen array row, **UP ARROW** does nothing.

F3 scrolls the display to the next full page of program array rows. This key may be reset using the OPTIONS statement.

F4 scrolls the display to the previous full page of program array rows. This key may be reset using the OPTIONS statement.

7. The INPUT ARRAY statement is terminated when the user enters **ESCAPE** or the key you specified as the ACCEPT KEY in the OPTIONS statement. If there is an AFTER INPUT clause, the program control passes to the statements following that clause, rather than to the statements following the END INPUT clause. This feature allows the programmer to perform data validity checks before allowing the INPUT ARRAY statement to terminate. Unlike the INPUT statement, the **RETURN** does not terminate the statement.

8. The number of variables in a row of *record-array* must be the same as the number of fields in a row of *screen-array* and of the corresponding data type.

9. **INFORMIX-4GL** executes BEFORE, AFTER, and ON KEY clauses in the following order:

```
BEFORE ROW
BEFORE INSERT, DELETE
BEFORE FIELD
ON KEY
AFTER FIELD
AFTER INSERT, DELETE
AFTER ROW
AFTER INPUT
```

10. INFORMIX-4GL passes control to the statements following a BEFORE ROW clause when

    - The cursor moves into a new form row.
    - An Insert fails due to lack of space.
    - Insert is aborted with an INTERRUPT.
    - The user performs a Delete (F2).

11. INFORMIX-4GL passes control to the statements following a BEFORE INSERT clause when

    - The user presses the Insert key (F1).
    - A row is added automatically at the end of an array.

12. INFORMIX-4GL passes control to the statements following a BEFORE DELETE clause when the user presses the Delete key (F2).

13. INFORMIX-4GL passes control to the statements following a BEFORE FIELD clause when the cursor enters a field.

14. INFORMIX-4GL passes control to the statements following an ON KEY clause when the user presses a key named in *key-list*.

15. INFORMIX-4GL passes control to the statements following an AFTER FIELD clause when the cursor leaves a field.

16. INFORMIX-4GL passes control to the statements following an AFTER INSERT clause when the user inserts a row and leaves it (without necessarily entering data into it).

17. INFORMIX-4GL passes control to the statements following an AFTER DELETE clause when the user presses the Delete key (F2) and the row has been deleted.

18. INFORMIX-4GL passes control to the statements following an AFTER ROW clause when

    - The cursor leaves the row by any means.
    - An Insert is complete.

19. INFORMIX-4GL passes control to the statements following an AFTER INPUT clause when the user presses ESCAPE or the key specified as the ACCEPT KEY in the OPTIONS statement.

20. If there is no NEXT FIELD statement in the sequence of statements following a BEFORE or AFTER clause, the cursor moves to the next field in the direction that the user indicated (forward for RIGHT ARROW or RETURN and backward for LEFT ARROW).

21. If the user triggers an ON KEY clause while entering data into a field, INFORMIX-4GL suspends the input of the current field while it executes the statements in the ON KEY clause. It preserves the input buffer that contains the characters the user typed before triggering the ON KEY clause, restores them when the ON KEY clause returns, and resumes the input in the same field with the cursor at end of the buffered list of characters. If you want to use the ON KEY clause to fill the field with another value, be sure to move to a new field with the NEXT FIELD statement to prevent the INPUT ARRAY statement from ignoring the new value.

22. The notation for function keys is F1, ..., F36. The notation for CONTROL keys is CONTROL-*X*, where *X* is any letter except A, D, H, L, R, or X. The notation for ESCAPE is ESC or ESCAPE. The notation for the INTERRUPT key is INTERRUPT. For UNIX systems, Appendix N describes how to verify that the termcap entry for your terminal allows INFORMIX-4GL to recognize the function keys.

23. By default, both ESCAPE and INTERRUPT are exits from the INPUT ARRAY statement. If the DEFER INTERRUPT statement has been executed, an INTERRUPT causes INFORMIX-4GL to set **int_flag** to nonzero and terminate the INPUT ARRAY statement (unless the function of INTERRUPT has been redefined in an ON KEY clause). Otherwise, an INTERRUPT causes an immediate program stop.

24. You can include the following keys in a *key-list* under the stated conditions:

    - ESCAPE if you have specified another key as the ACCEPT KEY in the OPTIONS statement.
    - F1 if you have specified another key as the INSERT KEY in the OPTIONS statement.
    - F2 if you have specified another key as the DELETE KEY in the OPTIONS statement.
    - F3 if you have specified another key as the NEXT KEY in the OPTIONS statement.
    - F4 if you have specified another key as the PREVIOUS KEY in the OPTIONS statement.
    - INTERRUPT if you have executed a DEFER INTERRUPT statement. (When the user presses the INTERRUPT key under these conditions, INFORMIX-4GL executes the statements in the ON KEY clause and sets **int_flag** to nonzero, but does not terminate the INPUT ARRAY statement.)

    You cannot use the following keys in a *key-list*:

    - CONTROL-A, CONTROL-D, CONTROL-H, CONTROL-L, CONTROL-R, or CONTROL-X since these keys are reserved for editing functions in the INPUT ARRAY statement.
    - Other keys that may have special meaning on your operating system.

25. In addition to the RETURN, ESCAPE, INTERRUPT, function, and ARROW keys, the user may employ the following keys for editing during an INPUT ARRAY statement:

    CONTROL-A toggles between insert and typeover mode.

CONTROL-D deletes characters from the current cursor position to the end of the field.

CONTROL-H moves the cursor non-destructively one space to the left inside a field.

CONTROL-L moves the cursor non-destructively one space to the right inside a field.

CONTROL-R redisplays the screen.

CONTROL-X deletes the character beneath the cursor.

26. The function **infield**(*field*) from the function library returns true if the current field is *field* and false otherwise. It can be used to make field-dependent responses when the user presses a key in the *key-list* of an ON KEY clause. If you call **infield**(*field*) outside the INPUT ARRAY statement, it returns a value corresponding to the field that was current when INPUT ARRAY was terminated.

27. Do not execute PROMPT, INPUT, or INPUT ARRAY statements within the BEFORE, AFTER, or ON KEY clauses of an INPUT ARRAY statement. However, you can call a function that executes one of these statements.

28. Four functions are required to keep track of the relative state of the cursor, the program array, and the screen array. The functions are defined as follows:

arr_curr( ) returns the current *record-array* row. This is the row where the cursor is at the beginning of the [BEFORE | AFTER] ROW block, not the row the cursor moves to after execution of the block.

arr_count( ) returns the total number of filled rows in *record-array*.

scr_line( ) returns the current row of *screen-array*. The current row is defined in the same way that the current row for **arr_curr( )** is defined.

set_count( ) takes an argument that is the total number of filled rows in *record-array*. You must call this function before executing the INPUT ARRAY WITHOUT DEFAULTS so that the program knows the initial value of **arr_count( )**.

29. If you execute a statement in a BEFORE, AFTER, or ON KEY clause that changes the value of **arr_curr**, **scr_line**, or **arr_count**, the new value remains for the duration of the clause (unless changed again). After **INFORMIX-4GL** executes the last statement in the clause, **INFORMIX-4GL** restores the original value. For example, if the DISPLAY ARRAY statement in the following INPUT ARRAY statement changes the value of **arr_curr** from 1 to 4, the value remains 4 for the rest of the ON KEY clause. After **INFORMIX-4GL** executes the MESSAGE statement, it resets **arr_curr** to 1.

```
INPUT ARRAY p_items TO s_items.*
ON KEY (F6)
   DISPLAY ARRAY p_stock TO s_stock.*
   END DISPLAY
   LET pa_curr = arr_curr()
   MESSAGE "The current value of arr_curr is: ", pa_curr
END INPUT
```

Examples

The following program fragment assumes that one column of the screen array (**sc_array**) contains the row number (**row_num**) of the program array (**pa_array**) being displayed there. The next column of **sc_array** is called **first_data**. The program recalculates and displays the row number after the user inserts new rows or deletes old ones.

```
DEFINE pa_total  INTEGER      # total number of rows
                              # in program array
DEFINE pa_curr   INTEGER      # current program array
                              # row number
DEFINE sc_curr   INTEGER      # current screen array
                              # row number
DEFINE sc_total  INTEGER      # total number of rows
                              # in screen array
DEFINE k         SMALLINT
...
INPUT ARRAY pa_array FROM sc_array.*
    AFTER INSERT, DELETE
        LET pa_curr = arr_curr()
        LET pa_total = arr_count()
        LET sc_curr = scr_line()
        FOR k = pa_curr TO pa_total
            LET pa_array[k].row_num = k
            IF sc_curr <= sc_total THEN
                DISPLAY k TO sc_array[sc_curr].row_num
                LET sc_curr = sc_curr + 1
            END IF
        END FOR
END INPUT
```

Related Statements

DISPLAY ARRAY, EXIT, INPUT

INSERT

Overview

Use the INSERT statement to insert one or more new rows into an existing table.

Syntax

```
INSERT INTO table-name [(column-list)]
        {VALUES (value-list) | SELECT-statement}
```

Explanation

INSERT INTO	are required keywords.
table-name	is the name of the table to which you want to add rows.
column-list	a list of the names of the columns into which you want to insert data. You can enter one column name or a series of column names, separated by commas.
VALUES	is a keyword.
value-list	are the values that you want to insert into the columns you specified. You can enter one or more program variables or constants, separated by commas.
SELECT-statement	is a valid **RDSQL** SELECT statement.

Notes

1. RDSQL inserts data into the columns in the specified table in the order in which you enter column names. It inserts the first value you enter into the first listed column, the second value into the second listed column, and so on.

2. Entering column names is optional. If you omit them, RDSQL assumes the values are listed in the order in which the columns are listed in the **syscolumns** systems catalog. Unless you have subsequently used the ALTER TABLE statement to change the order, the order is the same as when the table was created.

3. If you have previously defined a RECORD type program variable LIKE *table-name*, you can use the program variable in place of a list of values in an INSERT statement.

4. When you execute an INSERT statement, INFORMIX-4GL inserts a single row into the database (unless a SELECT statement appears after the VALUES clause). If you DECLARE a cursor for an INSERT statement and use the OPEN, PUT, FLUSH, and CLOSE statements, INFORMIX-4GL buffers the rows in memory and writes to disk only when the buffer is full.

5. RDSQL inserts the rows of data that result from the SELECT statement into the table, just as though you had entered them with the VALUES keyword.

6. You cannot use *table-name* in the FROM clause of the SELECT statement. The data must be selected from other tables.

7. Do not include an INTO TEMP clause or an ORDER BY clause in the SELECT statement.

8. Although the values you insert do not have to be of the same data type as the columns themselves, they must be compatible. You can insert only CHAR data into CHAR columns and only numeric or character representation of numeric data into numeric columns.

9. Enter a zero (0) for a SERIAL column in the INSERT statement if you want **RDSQL** to insert the next SERIAL value for the table. Enter a nonzero value for a SERIAL column that does not duplicate a value already in the table, if you want **RDSQL** to use that value. An error occurs if you enter a nonzero value for a SERIAL column that duplicates a value already in the table and if a unique index is defined on the column; **status** is set to a negative value.

10. You can use program variables in the list of values.

11. All constant CHAR and DATE values must be enclosed in quotation marks.

12. When you create a database with transactions, each **RDSQL** statement you execute is treated as a single transaction, even if you do not use the BEGIN WORK and COMMIT or ROLLBACK WORK statements. Because each row affected by the INSERT statement within a transaction is locked for the duration of the transaction, a single INSERT statement that affects a large number of rows locks those rows until the entire operation is completed. If the number of rows affected is very large, you may exceed the limits placed by your operating system on the maximum number of simultaneous locks. If this occurs, you may want either to reduce the scope of the INSERT statement or to lock the entire table before executing the statement.

    See the section "Locking" in Chapter 2 for a more detailed description of table-level and row-level locking in **INFORMIX-4GL.**

Caution! **RDSQL** makes every possible effort to perform data conversion, including converting the character string "123" into the integer 123. If the data cannot be converted, however, INSERT stops. Unless you have created the database with transactions, all changes made to that point remains, but subsequent rows from the SELECT statement will not be inserted. Data conversion also fails if the target data type cannot hold the value offered. For example, you cannot insert the integer 123456 into a SMALLINT.

Examples

```
DEFINE ps_customer RECORD LIKE customer.*
    ...
INSERT INTO customer VALUES (ps_customer.*)

INSERT INTO customer
    VALUES (0, f_name, l_name, comp,
        addr1, addr2, "Palo Alto",
            "CA", zip, phone)
```

Related Statements

DELETE, SELECT

LABEL

Overview

Use the LABEL statement to indicate the position in a program to which the GOTO statement refers.

Syntax

LABEL *label-id*:

Explanation

LABEL	is a required keyword.
label-id	is an **INFORMIX-4GL** identifier.
:	is a required punctuation.

Notes

1. The LABEL statement must be in the same routine (MAIN, FUNCTION, or REPORT) as the GOTO statement with the same *label-id*. You may not transfer out of a routine with a GOTO statement.

Examples

```
        IF customer_num < 0 THEN
          GOTO abort
        END IF
        statement
        . . .
  label abort:
        statement
```

Related Statements

GOTO

LET

Overview

Use the LET statement to assign a value to a program variable.

Syntax

LET *variable* = *expr*

Explanation

LET is a required keyword.

variable is the identifier of a simple program variable.

expr is an expression.

Notes

1. *variable* can be an element of an ARRAY if that element is a simple variable.

2. As an exception to the .* notation, you may make assignment to a record variable from another record variable using the statement

   ```
   LET x.* = y.*
   ```

   This is "shorthand" for a sequence of LET statements assigning values of elements of **y** to elements of **x**.

Examples

```
LET a = b + c

LET d[index] = "This is a string"

LET newstr = mystr[2,6]
```

Related Statements

INITIALIZE

LOCK TABLE

Overview

Use the LOCK TABLE statement to prohibit access to a table by other users.

Syntax

LOCK TABLE *table-name* IN {SHARE | EXCLUSIVE} MODE

Explanation

LOCK TABLE	are required keywords.
table-name	is the name of the table you want to lock.
IN	is a required keyword.
SHARE	is the keyword you use to give other users read access to the table, but to prevent them from modifying any of the data it contains.
EXCLUSIVE	is the keyword you use to prevent other users from having *any* access to the table.
MODE	is a required keyword.

Notes

1. Only one lock can apply to a table at any given time. That is, if a user locks a table (in either SHARE or EXCLUSIVE mode), no other user can lock that table in either mode until the first user unlocks it.

2. If your database has transactions, you must issue a BEGIN WORK statement before you can issue the LOCK TABLE statement.

3. You can use the LOCK TABLE statement immediately after a BEGIN WORK statement to override row-level locking during the transaction. Normally, each row of a table affected by a statement within a transaction is locked for the duration of the transaction. During transactions that affect a large number of rows, you may exceed the limit placed by your operating system on the maximum number of locks. If you lock the entire table at the beginning of the transaction, however, INFORMIX-4GL does not lock each row in the table. You may want to use this strategy when executing a transaction that affects a large number of rows or every row in a table.

   See the section "Locking" in Chapter 2 for further explanation of table-level and row-level locking in INFORMIX-4GL.

Examples

```
LOCK TABLE orders IN EXCLUSIVE MODE
```

Related Statements

UNLOCK TABLE

MAIN

Overview

Use the MAIN keyword to introduce the main program.

Syntax

```
MAIN
    statement
    ...
END MAIN
```

Explanation

MAIN	is a required keyword.
statement	is any **INFORMIX-4GL** statement except MAIN.
END MAIN	are required keywords that terminate the main program.

Notes

1. Every **INFORMIX-4GL** program must have a main program and may have one or more functions and reports.

Related Statements

FUNCTION, REPORT

MENU

Overview

Use the MENU statement to create a menu screen, to define a user's menu options, to designate help numbers, and to define what statements should be executed for each option.

Syntax

```
MENU "menu-name"
        COMMAND {KEY(key-list) |
                [KEY(key-list)] "menu-option"
                        ["helpline"] [HELP help-number]}
                statement
                ...
                [CONTINUE MENU]
                ...
                [EXIT MENU]
                ...
                [NEXT OPTION "menu-option"]
                ...
    ...
END MENU
```

Explanation

MENU	is a required keyword.
menu-name	is a character string giving the title of the menu.
COMMAND	is a required keyword.
KEY	is an optional keyword.
key-list	is a list of one or more letters, function key identifiers, or CONTROL key identifiers, separated by commas. You do not need to put quotation marks around single, printable characters.

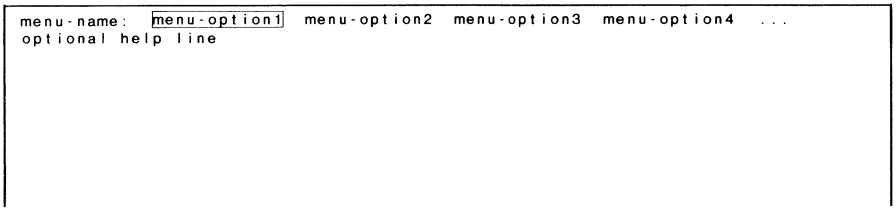
menu-option	is a single-word label for a menu option. *menu-option* must be enclosed in quotation marks.
helpline	is a one-line character string describing the *menu-option*. *helpline* must be enclosed in quotation marks.
HELP	is an optional keyword.
help-number	is the number of the help message in the help file designated in the OPTIONS statement that corresponds to *menu-option*.
statement	is an **INFORMIX-4GL** statement you want executed when the user selects the preceding option. There may be several statements for each option.
CONTINUE MENU	is an optional statement that returns program control to the current MENU statement.
EXIT MENU	is an optional statement that causes program control to move to the first statement following the END MENU keywords.
NEXT OPTION	are optional keywords that precede the option of the menu that you want highlighted when you return to the menu.
END MENU	are required keywords that terminate the MENU statement.

Notes

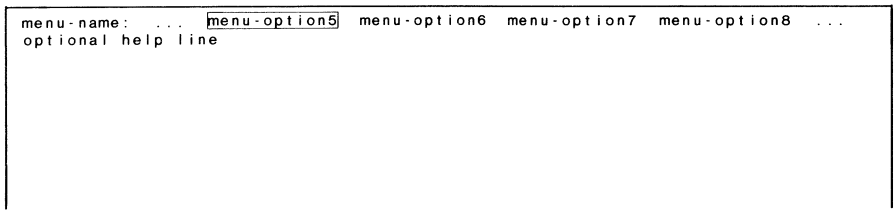
1. You must define at least two options (COMMAND clauses) for each menu.

2. The menu screen displays in a ring menu each of the single-word *menu-option*s in the order of the COMMAND clauses.

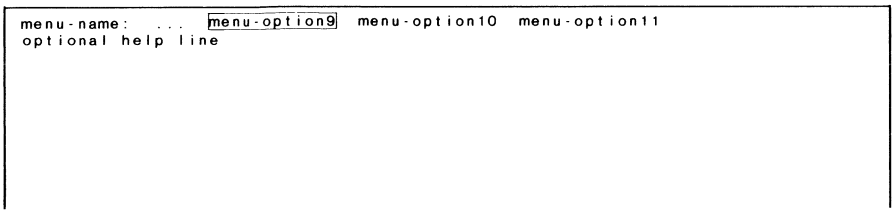
3. When **INFORMIX-4GL** displays a menu, it adds a colon (:) and a space after the menu name, as well as a space before and after each menu option. If the length of the menu exceeds the number of characters that the screen or a window can display on a single line, **INFORMIX-4GL** displays the first "page" of options followed by an ellipsis (...) indicating that additional options exist. For example,



If the user presses the **SPACEBAR** or **RIGHT ARROW** key to move past the rightmost option (**menu-option4** in this case), **INFORMIX-4GL** displays the next "page" of menu options. In the following example, the ellipses at each end indicates that more menu options exist in both directions.



If the user moves the highlight to the right past menu-option8 in this example, **INFORMIX-4GL** displays a page of menu options like the following:



Since no ellipsis appears to the right of the menu, the user has come to the last page of the menu options. The user can display the previous page of menu options again by using the **LEFT ARROW** key to move the highlight past the leftmost option in the example. The user can display the first page of menu options by using the **RIGHT ARROW** key to move the highlight past the rightmost option in the example.

The **UP ARROW** key moves the highlight to the first item on the previous page; the **DOWN ARROW** key moves the highlight to the first item on the subsequent page.

4. *help-number* refers to the number of the help message in the help file set by the OPTIONS statement.

5. A runtime error occurs if you specify a *help-number* for an option and that number does not occur in the help file or if the help file does not exist.

6. You will incur a runtime error if the menu cannot fit on the screen or in the current window.

7. **INFORMIX-4GL** truncates any *helpline* that exceeds the width of the screen or current window.

8. The user chooses an option by typing one of the letters in *key-list.* If the KEY clause is not present, the user chooses an option by typing the first letter of *menu-option.*

9. After the user chooses an option, **INFORMIX-4GL** executes the statements immediately following the COMMAND clause.

10. After **INFORMIX-4GL** executes all the statements for an option, it redisplays the menu, and the user can choose another option.

11. You can execute a CONTINUE MENU statement anywhere within the statements following the COMMAND clause. Use of this statement causes the menu to reappear so that the user can choose another option.

12. The notation for function keys is F1 through F36. The notation for CONTROL keys is CONTROL-*X*, where *X* is any letter except A, D, H, L, R, or X. The notation for **ESCAPE** is ESC or ESCAPE. The notation for the INTERRUPT key (often **DEL** or **CONTROL-C**) is INTERRUPT.

13. Unless you use the KEY clause, the initial letters of each *menu-option* should be different, regardless of case. The values within the *key-list* must be unambiguous; each option must be uniquely defined.

14. **INFORMIX-4GL** produces a runtime error if a menu option exceeds the length of the screen or window.

15. You can add a "hidden" option to your menu by including a KEY *key-list* choice in the list of menu COMMANDs. This is demonstrated in the following example.

Examples

```
MENU "TOP LEVEL"
     COMMAND "Add" "Add a row to the database" HELP 12
          . . .
     COMMAND "Find" "Find a row in the database" HELP 13
          . . .
     COMMAND "Change" "Update a row in the database" HELP 14
          . . .
     COMMAND "Delete" "Delete a row from the database" HELP 15
          . . .
     COMMAND key ("!")
          CALL bang()
          . . .
     COMMAND "Exit" "Return to operating system" HELP 16
          EXIT PROGRAM
END MENU
```

These statements produce the following menu:

```
TOP LEVEL:  Add  Find  Change  Delete  Exit
Add a row to the database
```

MESSAGE

Overview

Use the MESSAGE statement to display a character string on the Message Line.

Syntax

MESSAGE *display-list* [ATTRIBUTE (*attribute-list*)]

Explanation

MESSAGE	is a required keyword.
display-list	is a list of one or more program variables and/or string constants (enclosed in quotation marks), separated by commas.
ATTRIBUTE	is an optional keyword.
attribute-list	is a list of one or more screen attributes, separated by commas.

Notes

1. INFORMIX-4GL generates the message by replacing the variables in *display-list* with their values and concatenating the resulting strings.

2. The default Message Line is the same line used to display the *helpline* in menus. See the OPTIONS statement for information about resetting this line to a different position.

3. The default attribute for the MESSAGE display is the normal display. You may alter the default attribute with the ATTRIBUTE clause.

4. The following list shows the screen attributes allowed in the ATTRIBUTE clause:

WHITE = NORMAL	REVERSE
YELLOW = BOLD	BLINK
MAGENTA = BOLD	UNDERLINE
RED = BOLD	
CYAN = DIM	
GREEN = DIM	
BLUE = DIM	
BLACK = INVISIBLE	

On color terminals, NORMAL is interpreted as WHITE, BOLD as RED, DIM as BLUE, and INVISIBLE as BLACK. If you have a **colornames** file, you can use the color names listed there (see Chapter 3).

Examples

```
MESSAGE "Enter the order data."
```

OPEN

Overview

Use the OPEN statement to establish the search criteria for a SELECT cursor and initialize the system for subsequent fetches, or to set up an insert buffer for an INSERT cursor that references program variables.

Syntax

OPEN *cursor-name* [USING *variable-list*]

Explanation

OPEN	is a required keyword.
cursor-name	is an **RDSQL** identifier of a previously DECLAREd cursor.
USING	is an optional keyword.
variable-list	is a list of program variables, separated by commas, corresponding to the "?" parameters in the query associated with a SELECT cursor.

Notes

1. If *cursor-name* is associated with a SELECT statement, the OPEN statement examines the content of the program variables and, using these values for the parameters in the SELECT statement, establishes the search criteria for determining the logical set of rows that satisfies the WHERE clause. This set of rows is called the *active set*. It leaves the cursor in an open state and pointing before the first row of the active set.

2. The active set is a dynamic collection of rows. Rows meeting the SELECT search criteria and qualified for fetching depend upon the activity in the table.

3. Once the active set for a SELECT cursor is determined, the program variables are not reexamined until you reopen the cursor.

4. If a SELECT cursor is already open, an OPEN statement closes the cursor and reopens it, creating a new active set dependent on the current values of the program variables.

5. The FOREACH statement performs an implied OPEN statement. You cannot use the FOREACH statement if the OPEN statement for a SELECT cursor must have a USING clause to supply values for "?" parameters in a prepared SELECT statement.

6. If *cursor-name* is associated with an INSERT statement (rather than a SELECT statement), the OPEN statement cannot include a USING clause.

7. If you reopen an INSERT cursor that is already open, **RDSQL** flushes the INSERT buffer (that is, **RDSQL** inserts any rows currently in the INSERT buffer into the database table). The global variable SQLCA.SQLERRD[3] is set to the number of rows successfully inserted into the database.

Examples

```
DECLARE s_curs CURSOR FOR
    SELECT * FROM orders
OPEN s_curs

DECLARE q_cursor CURSOR FOR
    SELECT o.order_num, SUM(total_price)
    FROM orders o, items i
    WHERE o.order_date > "06/04/86"
        AND o.customer_num = 110
        AND o.order_num = i.order_num
    GROUP BY o.order_num

OPEN q_cursor
```

Related Statements

CLOSE, DECLARE, FETCH, FLUSH, FOREACH, PREPARE, PUT

OPEN FORM

Overview

Use the OPEN FORM statement to associate an INFORMIX-4GL identifier with a pre-compiled screen form.

Syntax

OPEN FORM *form-name* FROM "*form-file*"

Explanation

OPEN FORM	are required keywords.
form-name	is an INFORMIX-4GL identifier.
FROM	is a required keyword.
form-file	is the pathname of a compiled screen form (omitting the extension **.frm**). *form-file* must be enclosed in quotation marks.

Notes

1. You must open a form before you may display it.

2. When you execute the OPEN FORM statement, the compiled form is loaded into and kept in memory until you execute a CLOSE FORM statement for that form. If you have displayed another form and wish to regain the space used by the first form, you may execute CLOSE FORM on the old form. The CLOSE FORM statement is a memory-management feature only and does not affect the logic of the program.

Examples

```
OPEN FORM order_form FROM "order:form"
```

Related Statements

CLOSE FORM, DISPLAY FORM

OPEN WINDOW

Overview

Use the OPEN WINDOW statement to create and open a window at a specified origin.

Syntax

```
OPEN WINDOW window-name AT row, column
  WITH { integer ROWS, integer COLUMNS |
       FORM "form-file" }
  [ ATTRIBUTE (attribute-list) ]
```

Explanation

OPEN WINDOW	are required keywords.
window-name	is the name of the window you want to create.
AT	is a required keyword.
row	is an integer or integer variable between one and the maximum number of rows allowed by your terminal (usually 24). *row* indicates where **INFORMIX-4GL** displays the first row of the window.
column	is an integer or integer variable between one and the maximum number of columns allowed by your terminal (usually 80). *column* indicates where **INFORMIX-4GL** displays the first column of the window.
WITH	is a required keyword.
integer	is an integer or integer variable.

ROWS	is an optional keyword.
COLUMNS	is an optional keyword.
FORM	is an optional keyword.
form-file	is the name of a compiled form (excluding the **.frm** extension).
ATTRIBUTE	is an optional keyword.
attribute-list	is a list of one or more window display attributes.

Notes

1. When you open a window, **INFORMIX-4GL** saves the current window and makes the new window the current window.

2. *window-name* is global to the program in which it is OPENed.

3. You can explicitly specify the window dimensions by including a WITH *integer* ROWS, *integer* COLUMNS clause in the OPEN WINDOW statement.

   If you include a WITH FORM clause in the OPEN WINDOW statement, **INFORMIX-4GL** automatically opens a window sized to *form-file* and displays the form. **INFORMIX-4GL** determines the width of the window from the rightmost character of the screen form and calculates the length of the window as follows:

   FORM LINE (relative to the first line of the window) −1
   + form length
   + 1 for the COMMENT LINE

4. The WITH FORM clause allows you to conveniently open a window that displays a single form. You cannot close a form that was displayed using the WITH FORM clause; the CLOSE WINDOW statement does this automatically.

If you need to display more than one form in a window, or want a window larger than the one **INFORMIX-4GL** creates when you use the WITH FORM clause, you must explicitly specify the window dimensions with a WITH *integer* ROWS, *integer* COLUMNS clause and open, display, and close the form(s) yourself.

5. When you OPEN a window, **INFORMIX-4GL** uses the values specified in the most recently executed OPTIONS statement for the PROMPT, MESSAGE, FORM, and COMMENT lines. These values are relative to the first or last line of the newly opened window. You can change the values for the PROMPT, MESSAGE, FORM, and COMMENT lines (without affecting the most recently executed OPTIONS statement) by including an ATTRIBUTE clause in the OPEN WINDOW statement.

6. If a window is not large enough to contain the specified value for a PROMPT, MESSAGE, FORM, or COMMENT line, **INFORMIX-4GL** increases the value to FIRST or decreases it to LAST, as appropriate.

7. The attributes you can include in an ATTRIBUTE clause along with their default values as follows:

Attribute	**Default**
BORDER	no border
color	The default foreground color on your terminal
REVERSE	no reverse video
PROMPT LINE *line-value*	FIRST (1)
MESSAGE LINE *line-value*	FIRST + 1 (2)
FORM LINE *line-value*	FIRST + 2 (3)
COMMENT LINE *line-value*	LAST − 1 (for the screen) LAST (for all windows except the screen)

For the PROMPT, MESSAGE, and COMMENT lines, *line-value* can be an integer, program variable, FIRST plus an optional integer, or LAST minus an optional integer. For the FORM line, *line-value* can be an integer, program variable, or FIRST plus an optional integer.

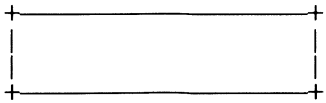
8. Since the error line is relative to the screen rather than the current window, you cannot change its location in the ATTRIBUTE clause of an OPEN WINDOW statement.

9. INFORMIX-4GL displays system error messages and text associated with the ERROR statement in a borderless window on the error line. INFORMIX-4GL opens the window as necessary and closes it at the next keystroke to erase the message.

10. If you specify a color in the ATTRIBUTE clause of an OPEN WINDOW statement, it becomes the default color for anything displayed in the window except a menu. You can override the default color for a particular display by specifying a different color in the ATTRIBUTE clause of a DISPLAY, DISPLAY ARRAY, or DISPLAY FORM statement.

11. Use the BORDER attribute if you want INFORMIX-4GL to draw a border around your window.

    On DOS systems, INFORMIX-4GL uses graphics characters to draw the border. On UNIX systems, INFORMIX-4GL uses characters defined in the system **termcap** file to draw the border. If you choose, you can specify alternate border characters in this file. If no characters are defined, INFORMIX-4GL uses the hyphen (-) for horizontal lines, the vertical bar (|) for vertical lines, and the plus sign (+) for corners, as follows:



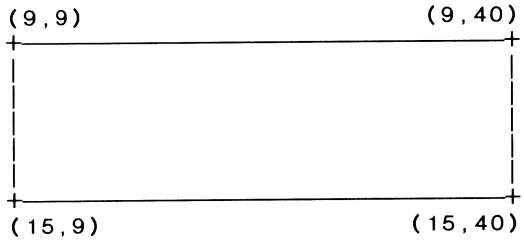
See Appendix N, "Termcap Changes for INFORMIX-4GL," and the manual that comes with your terminal, for information about making changes to your **termcap** file.

Note: Some terminals may not support the features described in Appendix N.

12. When you use the BORDER attribute, **INFORMIX-4GL** draws a border outside the window area you specify. For example, if you open the following window

```
OPEN WINDOW w1 AT 10,10
   WITH 5 ROWS, 30 COLUMNS
   ATTRIBUTE (BORDER)
```

INFORMIX-4GL displays a border with coordinates like those in the following example:



Note: The termcap entry for some terminals includes an *sg#1* setting. On these terminals **INFORMIX-4GL** an additional column to the left and to the right of the window. These two columns are reserved whether you specify a border or not. **INFORMIX-4GL** uses a total of four extra columns for bordered windows on these terminals: two columns to the left of the window and two columns to the right of the window.

13. If a window and its border (if any) exceed the physical limits of the screen, **INFORMIX-4GL** generates a runtime error.

14. You can specify the following foreground colors for *color* in an ATTRIBUTE clause:

WHITE	BLUE
YELLOW	BLACK
MAGENTA	DIM
RED	INVISIBLE
CYAN	BOLD
GREEN	NORMAL

15. If the text you display with the following statements does not fit in the current window, **INFORMIX-4GL** truncates it:

PROMPT	
MESSAGE	
DISPLAY [AT]	
COMMENTS	(FORM4GL attribute)

16. Windows are stacked in the order they are opened. See the CURRENT WINDOW and CLOSE WINDOW statements for more information about how these statements affect the window stack.

Examples

```
OPEN WINDOW w1 AT 5, 5
  WITH FORM "custform"

OPEN WINDOW w2 AT 10, 12
  WITH 5 ROWS, 40 COLUMNS
  ATTRIBUTE (BORDER, PROMPT LINE 3)
```

Related Statements

CLEAR WINDOW, CLOSE WINDOW, CURRENT WINDOW, OPTIONS

OPTIONS

Overview

Use the OPTIONS statement to modify default layouts for screens and default keys for screen operations and program aids (like help messages).

Syntax

```
OPTIONS { MESSAGE LINE message |
          PROMPT LINE prompt |
          COMMENT LINE comment |
          ERROR LINE error |
          FORM LINE form |
          INPUT {WRAP | NO WRAP} |
          INSERT KEY insert-key |
          DELETE KEY delete-key |
          NEXT KEY next-key |
          PREVIOUS KEY previous-key |
          ACCEPT KEY accept-key |
          HELP FILE "help-file" |
          HELP KEY help-key}[, ...]
```

Explanation

OPTIONS	is a required keyword.
MESSAGE LINE	are optional keywords that refer to the Message Line.
message	is an integer expression indicating on which line of the current window the Message Line appears. The default is FIRST + 1 (line 2 of the window).
PROMPT LINE	are optional keywords that refer to the Prompt Line.

prompt	is an integer expression indicating on which line of the current window the Prompt Line appears. The default is the FIRST window line.
COMMENT LINE	are optional keywords that refer to the Comment Line.
comment	is an integer expression indicating on which line of the current window the comment appears. The default is LAST − 1 for the screen and LAST for all other windows.
ERROR LINE	are optional keywords that refer to the Error Line.
error	is an integer expression indicating on which line of the screen the error message appears. The default is LAST.
FORM LINE	are optional keywords that refer to the first line of a form.
form	is an integer expression indicating on which line of the current window the form begins. The default is FIRST + 2 (line 3 of the window).
INPUT WRAP	are optional keywords indicating that the cursor wraps around the list of input fields during the execution of an INPUT or CONSTRUCT statement until the ACCEPT key is pressed. The default is INPUT NO WRAP.
INPUT NO WRAP	are optional keywords indicating that the INPUT or CONSTRUCT statement terminates upon a **RETURN** after the last field. This is the default.

INSERT KEY	are optional keywords.
insert-key	is **ESCAPE** or a function or CONTROL key designation for the key that opens up a line for data insertion in the INPUT ARRAY statement. The default is **F1**.
DELETE KEY	are optional keywords.
delete-key	is **ESCAPE** or a function or CONTROL key designation for the key that deletes a line in the INPUT ARRAY statement. The default is **F2**.
NEXT KEY	are optional keywords.
next-key	is **ESCAPE** or a function or CONTROL key designation for the key that scrolls to the next page in the INPUT ARRAY or DISPLAY ARRAY statement. The default is **F3**.
PREVIOUS KEY	are optional keywords.
previous-key	is **ESCAPE** or a function or CONTROL key designation for the key that scrolls to the previous page in the INPUT ARRAY or DISPLAY ARRAY statement. The default is **F4**.
ACCEPT KEY	are optional keywords.
accept-key	is **ESCAPE** or a function or CONTROL key designation for the key that terminates the INPUT, INPUT ARRAY, DISPLAY ARRAY, and CONSTRUCT statements. The default is **ESCAPE**.
HELP FILE	are optional keywords.

help-file	is the pathname of the file containing help messages.
HELP KEY	are optional keywords that refer to the function or CONTROL key that elicits help messages.
help-key	is **ESCAPE** or a function key or CONTROL key designation for the key that causes help messages to be displayed. The default is **CONTROL-W**.

Notes

1. You use the OPTIONS statement to change the defaults listed above. (If you list more than one item in an OPTIONS statement, make sure you separate the items with commas.)

2. You can issue the OPTIONS statement more than once. The values set in the last OPTIONS statement encountered at run time prevail.

3. All values for *message*, *prompt*, *comment*, and *form* are relative to the first line of the screen or current window. The value for *error* is relative to the screen rather than to the current window.

4. The value for *message*, *prompt*, *comment*, or *error* can be one of the following

   integer
   FIRST [+ *integer*]
   LAST [− *integer*]

   where FIRST is the first line of the current window (line 1) and LAST is the last line of the current window. The value for *form* can be *integer* or FIRST [+ *integer*].

5. The notation for function keys is F1 through F36. The notation for CONTROL keys is CONTROL-*X*, where *X* is any letter except A, D, H, L, R, or X. The notation for ESCAPE is ESC or ESCAPE.

6. INFORMIX-4GL uses the CONTROL keys CONTROL-A, CONTROL-D, CONTROL-H, CONTROL-L, CONTROL-R, and CONTROL-X to perform screen-editing functions. You cannot use them for the INSERT KEY, DELETE KEY, NEXT KEY, PREVIOUS KEY, ACCEPT KEY, or HELP KEY. In addition, you may not be able to use other keys, depending on your operating system. (For example, CONTROL-C is usually the INTERRUPT key for DOS systems.)

Examples

```
OPTIONS MESSAGE LINE 23,
        PROMPT LINE LAST-2,
        FORM LINE FIRST,
        NEXT KEY CONTROL-N,
        PREVIOUS KEY CONTROL-P
```

Related Statements

DISPLAY, DISPLAY ARRAY, ERROR, INPUT, INPUT ARRAY, MENU, MESSAGE, PROMPT

OUTPUT TO REPORT

Overview

Use the OUTPUT TO REPORT statement to pass a single row of data to a report.

Syntax

OUTPUT TO REPORT *report-name* (*expr-list*)

Explanation

OUTPUT TO REPORT	are required keywords.
report-name	is the identifier of a report.
expr-list	is a list of one or more expressions, separated by commas.

Notes

1. Ordinarily, you will use the OUTPUT TO REPORT statement within a loop that passes data to a report.

2. The number of expressions in *expr-list* should agree with the number and type of arguments in the REPORT statement.

Examples

```
OUTPUT TO REPORT rept1 (v_pers.*)
```

Related Statements

FINISH REPORT, REPORT, START REPORT

PREPARE

Overview

Use the PREPARE statement to preprocess an **RDSQL** statement for later execution.

Syntax

PREPARE *statement-name* FROM *string-spec*

Explanation

PREPARE	is a required keyword.
statement-name	is an **RDSQL** identifier.
FROM	is a required keyword.
string-spec	is either a string constant enclosed in quotation marks or a CHAR type program variable. *string-spec* must contain an **RDSQL** statement.

Notes

1. The statement described in *string-spec* cannot contain program variables. Use a question mark (?) as a place holder for an input value that will be supplied in an EXECUTE, OPEN, or PUT statement. Do not use a question mark as a placeholder for an **RDSQL** identifier such as a database name, table name, column name, or user name.

2. If you are preparing a SELECT statement for use with the DECLARE statement, *string-spec* may include a SELECT statement followed by a FOR UPDATE clause.

3. Do not use the PREPARE statement on the following statements: CLOSE, DECLARE, EXECUTE, FETCH, OPEN, PREPARE, and WHENEVER.

4. Do not PREPARE a SELECT statement with an INTO clause.

5. *statement-name* is global to the source file in which it is PREPAREd. You can refer to it by name in two or more different functions contained in the same module. It is not, however, like a global variable that can be used in a separate source file.

6. Within a module, *statement-name* can apply to only one RDSQL statement. Do not PREPARE another statement with the same *statement-name*.

Examples

```
LET select_2 = "select * from orders ",
               "where customer_num = ? and ",
                     "order_date > ?"
PREPARE query_2 FROM select_2
```

Related Statements

DECLARE, EXECUTE, FOREACH, OPEN

PROMPT

Overview

Use the PROMPT statement to query the user and to accept a value entered by the user.

Syntax

```
PROMPT display-list FOR [CHAR] variable
                [HELP help-number]
                [ON KEY (key-list)
                        statement
                        ...
                ...
END PROMPT]
```

Explanation

PROMPT	is a required keyword.
display-list	is a list of one or more program variables or string constants, separated by commas.
FOR	is a required keyword.
CHAR	is an optional keyword.
variable	is the program variable that will contain the value typed in by the user.
HELP	is an optional keyword.
help-number	is an integer that identifies the help message for this PROMPT statement in the help file designated in the OPTIONS statement.
ON KEY	are optional keywords.

key-list	is a list of one or more function or CONTROL key designations. It can also include ESCAPE (if you have specified another key as the ACCEPT KEY in the OPTIONS statement) or INTERRUPT (if you have executed a DEFER INTERRUPT statement).
statement	is an **INFORMIX-4GL** statement.
END PROMPT	is a statement that terminates a PROMPT statement. It is required only when an ON KEY clause is used.

Notes

1. **INFORMIX-4GL** displays the string generated by replacing the variables in *display-list* with their current values on the prompt line if an open form is displayed. The prompt occurs at the current cursor position if no form is displayed and

   - It is preceded by a DISPLAY statement with no AT clause.
   - It is the first printing statement in the program.
   - The screen is cleared.

2. The PROMPT statement returns the value entered by the user in *variable*. For a string *variable*, the value returned may include spaces.

3. The use of the CHAR option causes PROMPT to accept a single character input without requiring a carriage return.

4. If **INFORMIX-4GL** cannot convert the value entered by the user to the data type of *variable*, it returns a negative error code in the global variable **status** and the value of *variable* is undetermined.

5. You can use the following keys in a *key-list* under the stated conditions:

   - Function keys.
   - CONTROL keys (except as noted below)
   - ESCAPE (if you have specified another key as the ACCEPT KEY in the OPTIONS statement).
   - INTERRUPT if you have executed a DEFER INTERRUPT statement. (When the user presses the INTERRUPT key under these conditions, **INFORMIX-4GL** executes the statements in the ON KEY clause and sets **int_flag** to nonzero.)

   You cannot use the following keys in a *key-list*:

   - **CONTROL-A, CONTROL-D, CONTROL-H, CONTROL-L, CONTROL-R,** or **CONTROL-X** since these CONTROL keys are reserved for editing functions in the CONSTRUCT, INPUT, and INPUT ARRAY statements.
   - Other keys that may have special meaning on your operating system.

6. **INFORMIX-4GL** terminates PROMPT and passes control to the statements following an ON KEY clause when the user presses a key specified in *key-list*. In this case, the value in *variable* is undetermined. After completing the ON KEY clause, **INFORMIX-4GL** passes control to the statements following END PROMPT.

7. The notation for function keys is F1 through F36. The notation for CONTROL keys is CONTROL-*X*, where *X* is any letter except A, D, H, L, R, or X. The notation for **ESCAPE** is ESC or ESCAPE. The notation for the INTERRUPT key (often **DEL** or **CONTROL-C**) is INTERRUPT.

8. Do not execute PROMPT, INPUT, or INPUT ARRAY statements within the ON KEY clause of a PROMPT statement. However, you can call a function that executes one of these statements.

Examples

```
PROMPT "Enter the Customer Number: "
       FOR v.cust_no
    ON KEY (CONTROL-E)
       GOTO stop_now
END PROMPT
```

PUT

Overview

Use the PUT statement to store a row in the INSERT buffer for later insertion into the database table.

Syntax

PUT *cursor-name*
  [FROM *variable-list*]

Explanation

PUT	is a required keyword.
cursor-name	is the name of a cursor that has been DECLAREd for an INSERT statement.
FROM	is an optional keyword.
variable-list	is a list of program variables, separated by commas, corresponding to the "?" parameters in the prepared INSERT statement associated with *cursor-name*.

Notes

1. You can execute the PUT statement only if *cursor-name* has been DECLAREd for an INSERT statement and is in an open state. Such a cursor is referred to as an insert cursor.

2. The PUT statement puts a row in the buffer created when *cursor-name* was OPENed. When you flush the buffer (by executing a series of PUT statements, a CLOSE statement, or a FLUSH statement) **RDSQL** inserts the buffered rows into the database table as a block.

3. RDSQL does not create an insert buffer for a cursor associated with an INSERT statement that contains only constants in the VALUES clause. If you execute a PUT statement for such a cursor, RDSQL increments a counter that keeps track of the number of rows to be inserted into the database. The database is updated only when you issue a FLUSH or CLOSE statement.

4. You close a cursor by issuing a CLOSE statement. Exiting a program without closing an insert cursor leaves the buffer unflushed. Rows inserted into the buffer since the last flush are lost. You cannot rely on the end of program to close the cursor and flush the buffer.

5. The global variables **status** (whose value is received from SQLCA.SQLCODE) and SQLCA.SQLERRD[3] indicate the result of each PUT statement.

   If RDSQL simply puts a row in the insert buffer, it sets **status** and SQLCA.SQLERRD[3] to zero. If, as the result of a PUT statement, RDSQL successfully inserts a block of rows into the database, it sets **status** to zero and sets SQLCA.SQLERRD[3] to the number of rows inserted. If, as the result of a PUT statement, RDSQL is unsuccessful in its attempt to insert an entire block of rows into the database, it sets **status** to a negative number (specifically, the number of the error message) and sets SQLCA.SQLERRD[3] to the number of rows successfully inserted into the database.

6. Whenever the buffer is flushed, **SQLCA.SQLERRD[3]** is set to the number of rows successfully inserted into the database. If an error occurs during the flushing of a buffer, the buffered rows following the last successfully inserted row are discarded.

7. If your database has transactions, you must execute a PUT statement within a BEGIN WORK—COMMIT WORK or a BEGIN WORK—ROLLBACK WORK block. COMMIT WORK and ROLLBACK WORK automatically close all cursors.

8. If *cursor-name* has been DECLAREd for a prepared INSERT statement that includes "?" parameters, you must use the PUT statement with a FROM clause. After the FROM keyword, you can list the variable(s) containing the value(s) that RDSQL substitutes for the "?" parameters in the prepared INSERT statement.

Examples

```
DECLARE icurs CURSOR FOR
    INSERT INTO manufact VALUES (m_code, m_name)
OPEN icurs
PUT icurs


PREPARE ins_stmt FROM
    "INSERT INTO manufact VALUES (?, ?)"
DECLARE ins_curs CURSOR FOR ins_stmt
OPEN ins_curs
PUT ins_curs FROM m_code, m_name
```

Related Statements

CLOSE, DECLARE, FLUSH, OPEN

RECOVER TABLE

Overview

In the event of a system failure, use the RECOVER TABLE statement to restore a database table from a backup copy and an audit trail file.

Syntax

RECOVER TABLE *table-name*

Explanation

RECOVER TABLE	are required keywords.
table-name	is the name of the table you want to recover.

Notes

1. Once you have recovered the table, use the DROP AUDIT statement to remove the contents of the audit trail file. Run the CREATE AUDIT statement to start a new audit trail file, then back up the table. See the section "Audit Trails" in Chapter 2 for more information.

2. RECOVER TABLE checks that the audit trail and *table-name* have consistent record numbers for rows where changes have taken place. If RECOVER TABLE finds inconsistencies, it stops restoring the table.

3. You must own *table-name* or have DBA status to use the RECOVER TABLE statement.

Examples

The following RDSQL statements give a template for the recovery of a table. They assume that your audit trail began from the last backup.

```
{restore table from last backup}

RECOVER TABLE customer

DROP AUDIT FOR customer

CREATE AUDIT FOR customer IN "/dev/safe"

{make a backup of the recovered table}
```

Related Statements

CREATE AUDIT, DROP AUDIT

RENAME COLUMN

Overview

Use the RENAME COLUMN statement to change the name of a column.

Syntax

RENAME COLUMN *table.oldcolumn* TO *newcolumn*

Explanation

RENAME COLUMN	are required keywords.
table	is the name of the table containing the column whose name you want to change. This is a required item in the statement.
oldcolumn	is the name of the column you want to rename.
TO	is a required keyword.
newcolumn	is the new name you want to assign to the column. *newcolumn* must satisfy the requirements for an **RDSQL** identifier and cannot duplicate another column name in the table.

Notes

1. You may RENAME a column of a table only when you own the table, have DBA privilege, or have been granted ALTER permission.

2. Do not use the RENAME COLUMN statement in a transaction; it cannot be rolled back.

Examples

```
RENAME COLUMN customer.customer_num TO c_num
```

Related Statements

ALTER TABLE, CREATE TABLE, INSERT, DROP TABLE, RENAME TABLE

RENAME TABLE

Overview

Use the RENAME TABLE statement to change the name of a table.

Syntax

RENAME TABLE *oldname* TO *newname*

Explanation

RENAME TABLE	are required keywords.
oldname	is the name of the table you want to rename.
TO	is a required keyword.
newname	is the new name you want to assign to the table.

Notes

1. You may RENAME a table only when you own the table, have DBA privilege, or have been granted ALTER permission on the table.

2. Do not execute the RENAME TABLE statement within a transaction; it cannot be rolled back.

Examples

This example moves the **quantity** column to the third place.

```
CREATE TABLE newtab
        (item_num       SMALLINT,
         order_num      INTEGER,
         quantity       SMALLINT,
         stock_num      SMALLINT,
         manu_code      CHAR(4),
         total_price    MONEY(8)
        )

INSERT INTO newtab
       SELECT item_num, order_num,
           quantity, stock_num, manu_code,
           total_price FROM items

DROP TABLE items

RENAME TABLE newtab TO items
```

Related Statements

ALTER TABLE, CREATE TABLE, INSERT, DROP TABLE, RENAME COLUMN

REPORT

Overview

Use the REPORT routine to provide the format specifications for a report.

Syntax

```
REPORT report-name (variable-list)
                [DEFINE-statement]
                    ...
                [OUTPUT
                        output-statement
                        ...]
                [ORDER [EXTERNAL] BY sort-list]
                FORMAT
                        format-statement
                        ...
                        4gl-statement
                        ...
END REPORT
```

Explanation

REPORT	is a required keyword.
report-name	is an **INFORMIX-4GL** identifier.
variable-list	is a list of zero or more variables, separated by commas.
DEFINE-statement	is a DEFINE statement giving the data type for the variables in *variable-list*.
OUTPUT	is an optional keyword.
output-statement	is an output statement described in Chapter 4.

ORDER BY	are optional keywords.
EXTERNAL	is an optional keyword.
sort-list	is a list of one or more variables from those in *variable-list.*
FORMAT	is a required keyword.
format-statement	is a FORMAT statement described in Chapter 4.
4gl-statement	is an arbitrary **INFORMIX-4GL** statement.
END REPORT	are required keywords that terminate the REPORT statement.

Notes

1. If *variable-list* contains the name of a record, you must DEFINE the record in *DEFINE-statement.* Do not append the .* to the name of the record in *variable-list.*

2. See Chapter 4 for a discussion of the OUTPUT, ORDER BY, and FORMAT sections of the REPORT routine.

3. If **INFORMIX-4GL** statements occur in the control blocks of the FORMAT section, they are executed during the report-printing phase. If the data is sorted outside the report, report printing takes place with each OUTPUT TO REPORT statement. This is called a one-pass report. If the data is sorted inside the report, report printing takes place with the FINISH REPORT statement. This is called a two-pass report.

   If **INFORMIX-4GL** statements occur in the OUTPUT TO REPORT loop as well as in the report, **INFORMIX-4GL** alternately executes the **INFORMIX-4GL** statements in the

OUTPUT TO REPORT loop and the **INFORMIX-4GL** statements in the report during a one-pass report. In contrast, **INFORMIX-4GL** repeatedly executes all the **INFORMIX-4GL** statements in the OUTPUT TO REPORT loop *before* executing the **INFORMIX-4GL** statements in the report during a two-pass report.

Examples

The simplest report presents the output of a query:

```
DECLARE simp_curs CURSOR FOR
   SELECT * FROM CUSTOMER

START REPORT simple

FOREACH simp_curs INTO cust.*
   OUTPUT TO REPORT simple(cust.*)
END FOREACH

FINISH REPORT simple

...

REPORT simple (x)

   DEFINE x RECORD LIKE customer.*

   FORMAT
      EVERY ROW

END REPORT
```

Related Statements

FINISH REPORT, OUTPUT TO REPORT, START REPORT

RETURN

Overview

Use the RETURN statement to leave a FUNCTION routine and to return values to the calling routine.

Syntax

RETURN [*expr-list*]

Explanation

RETURN	is a required keyword.
expr-list	is an optional list of one or more expressions, separated by commas.

Notes

1. The RETURN statement can occur only within a FUNCTION routine and directs **INFORMIX-4GL** to exit the function and to return to the calling routine (MAIN, FUNCTION, or REPORT).

2. The expressions in *expr-list* must match in number and type the argument list in the RETURNING clause of the CALL statement.

Related Statements

FUNCTION

REVOKE

Overview

Use the REVOKE statement to remove another user's access privileges for a table. Access privileges are relevant only on multi-user systems.

Syntax

REVOKE {*tab-privilege* ON *table-name* | *db-privilege*}
FROM {PUBLIC | *user-list*}

Explanation

REVOKE — is a required keyword.

tab-privilege — is one or more of the following table-level access privileges, separated by commas:

ALTER	Adds or deletes columns or modifies data types of columns
DELETE	Deletes rows
INDEX	Creates indexes
INSERT	Inserts rows
SELECT	Retrieves data
UPDATE	Changes column values
ALL [PRIVILEGES]	All of the above

ON — is a required keyword.

table-name is the name of the table for which you are revoking access privileges.

db-privilege is one of the following database-level access types:

CONNECT	allows access to database tables without permission to create permanent tables and indexes.
RESOURCE	allows access to database tables with permission to create permanent tables and indexes.
DBA	allows full database administrator privileges.

FROM is a required keyword.

PUBLIC is the keyword you use to revoke access *privilege* from all users.

user-list is a list of login names for the users whose access privilege you are revoking. You can enter one login name or a series of login names, separated by commas.

Notes

1. Do not execute the REVOKE statement within a transaction; it cannot be rolled back.

2. You can revoke database-level access privileges only if you have DBA status.

3. You can revoke only table-level access privileges that you have granted to another user.

4. You cannot revoke privileges from yourself.

5. Although you can grant UPDATE and SELECT privileges for specific columns, you cannot revoke these privileges column by column. If you revoke UPDATE or SELECT privileges from a user, **RDSQL** automatically revokes all UPDATE and SELECT privileges you have ever granted to that user for *table-name*. You can then re-grant privileges for specific columns.

6. Only a DBA recipient can revoke the DBA privilege from another recipient. If the database creator grants DBA privileges to another user, that person can revoke the DBA privilege from the database creator.

7. If you revoke the DBA or RESOURCE privilege from one or more users, they are left with the CONNECT privilege. To revoke all database privileges from users with DBA or RESOURCE status, you must revoke CONNECT as well as DBA or RESOURCE.

Examples

```
REVOKE ALL ON orders FROM PUBLIC

REVOKE DELETE, UPDATE
    ON customer FROM john, mary

REVOKE CONNECT FROM enid, felix
```

Related Statements

GRANT

ROLLBACK WORK

Overview

Use the ROLLBACK WORK statement to undo all database modifications since the last BEGIN WORK statement.

Syntax

```
ROLLBACK WORK
```

Explanation

ROLLBACK WORK are required keywords.

Notes

1. If you use the ROLLBACK WORK statement in a routine that is called by a WHENEVER statement, be sure to specify WHENEVER ERROR CONTINUE and WHENEVER WARNING CONTINUE before the ROLLBACK WORK statement. This will avoid looping if the ROLLBACK WORK statement fails with an error or warning.

2. See the "Transactions" section in Chapter 2 for further details.

3. The ROLLBACK WORK statement releases all row and table locks.

4. The ROLLBACK WORK statement closes all open cursors, although using it for this purpose is not recommended.

Related Statements

BEGIN WORK, COMMIT WORK

ROLLFORWARD DATABASE

Overview

Use the ROLLFORWARD DATABASE statement to cause RDSQL to apply the transactions in the transaction log file to a backup copy of your database, recovering all completed transactions.

Syntax

ROLLFORWARD DATABASE *database-name*

Explanation

ROLLFORWARD DATABASE	are required keywords.
database-name	is the name of a database.

Notes

1. See the section "Transactions" in Chapter 2 for more information.

Related Statements

BEGIN WORK, COMMIT WORK, START DATABASE, ROLLBACK WORK

RUN

Overview

Use the RUN statement to execute a system program.

Syntax

RUN *command-line* [RETURNING *integer-variable*
| WITHOUT WAITING]

Explanation

RUN	is a required keyword.
command-line	is an expression that evaluates to a command line for your operating system. In particular, it may be a character string enclosed in quotation marks.
RETURNING	is an optional keyword for UNIX systems only.
integer-variable	is an INTEGER-type program variable that will receive the value returned by the program executed by the RUN statement.
WITHOUT WAITING	are optional keywords for UNIX systems only.

Notes

1. On UNIX systems, RUN spawns a child process described by *command-line*. The WITHOUT WAITING option instructs **INFORMIX-4GL** to continue immediately to the next **INFORMIX-4GL** statement, while the RETURNING option instructs **INFORMIX-4GL** to await the return value before continuing to the next **INFORMIX-4GL** statement. If neither optional clause is present, **INFORMIX-4GL** waits until the child process is completed (and ignores the return code) before continuing to the next **INFORMIX-4GL** statement.

2. On DOS systems, RUN causes your **INFORMIX-4GL** program to pause while the operating system executes *command-line*. After executing *command-line*, the operating system returns program control to your **INFORMIX-4GL** program.

Examples

```
RUN "date_script" RETURNING error_val

RUN "isql -qr myscript"

RUN charval[i]
```

SCROLL

Overview

Use the SCROLL statement to move rows of a screen record through a screen array.

Syntax

SCROLL {*field-list* | *screen-record*.*}[, ...]
{UP | DOWN} [BY *integer*]

Explanation

SCROLL	is a required keyword.
field-list	is a list of one or more screen field names, separated by commas.
screen-record	is the name of a screen record.
UP	is an optional keyword indicating that the data on the screen should move upwards.
DOWN	is an optional keyword indicating that the data on the screen should move downwards.
BY	is an optional keyword.
integer	is an INTEGER constant or variable.

Notes

1. The BY clause determines the number of lines upward or downward that the data will move. The default is 1.

2. It is the programmer's responsibility to keep track of what data is left on the screen.

Examples

```
SCROLL sc_item UP BY 2
```

Related Statements

DISPLAY ARRAY, INPUT ARRAY

SELECT

Overview

Use the SELECT statement to query the current database.

The SELECT statement can include up to eight clauses. Only the SELECT clause and the FROM clause are required.

Syntax

```
SELECT clause  [INTO clause] FROM clause
               [WHERE clause]
               [GROUP BY clause]
               [HAVING clause]
               [ORDER BY clause]
               [INTO TEMP clause]
```

See "The SELECT Statement" section later in this chapter for detailed descriptions of these clauses.

SET LOCK MODE

Overview

Use the SET LOCK MODE statement to determine whether subsequent RDSQL calls wait for a locked row to become unlocked.

Syntax

```
SET LOCK MODE TO [NOT] WAIT
```

Explanation

SET LOCK MODE	are required keywords.
TO	is a required keyword.
NOT	is an optional keyword.
WAIT	is a required keyword.

Notes

1. The TO NOT WAIT option causes RDSQL to return an error if a statement attempts to alter or delete a row (or to SELECT a row FOR UPDATE) that another process has locked. This is the default situation; that is, if you have not issued a SET LOCK MODE statement previously. The NOT option is relevant, therefore, only when you have previously executed SET LOCK MODE TO WAIT and want to return to the default state.

2. The TO WAIT option causes RDSQL to wait on an attempt to alter or delete a row (or to SELECT a row FOR UPDATE) that another process has locked until the locked row becomes unlocked.

3. Use the SET LOCK MODE TO WAIT statement with extreme caution. If the locking process fails and does not remove the lock, your statement could wait indefinitely.

4. This feature is available only on systems that have record-level locking and applies only to row-level locking. *Any* attempt by another user to access a row in a table locked IN EXCLUSIVE MODE produces an error.

Related Statements

LOCK TABLE

SLEEP

Overview

Use the SLEEP statement to cause the program to suspend operation for a period of time.

Syntax

SLEEP *integer-expr*

Explanation

SLEEP	is a required keyword.
integer-expr	is an expression that evaluates to INTEGER type.

Notes

1. The SLEEP statement causes the program to suspend operation for *integer-expr* seconds.

START DATABASE

Overview

Use the START DATABASE statement to start a new transaction log file.

Syntax

START DATABASE *database-name* WITH LOG IN "*pathname*"

Explanation

START DATABASE	are required keywords.
database-name	is the name of a database.
WITH LOG IN	are required keywords.
pathname	is the full pathname of the transaction log file. *pathname* must be enclosed in quotation marks.

Notes

1. Use the START DATABASE statement only if you change the name of your transaction log file or if you want to start recording transactions in a database that was created without transactions.

2. See the section "Transactions" in Chapter 2 for full details.

Related Statements

BEGIN WORK, COMMIT WORK, CREATE DATABASE, ROLLBACK WORK, ROLLFORWARD DATABASE

START REPORT

Overview

Use the START REPORT statement to begin processing a report.

Syntax

START REPORT *report-name*
[TO {*filename* | PRINTER | PIPE *program*}]

Explanation

START REPORT	are required keywords.
report-name	is the identifier of a report.
TO	is an optional keyword.
filename	is either a CHAR type variable or a quoted string constant containing the name of a system file.
PRINTER	is an optional keyword.
PIPE	is an optional keyword.
program	is either a CHAR variable or a string constant containing the command line for a system program.

Notes

1. Usually, you will execute the START REPORT statement just before a loop in which you process the report data using the OUTPUT TO REPORT statement.

2. If you use the TO clause, **INFORMIX-4GL** ignores any REPORT TO statement in the OUTPUT section of *report-name*.

3. If you indicate *filename*, **INFORMIX-4GL** puts the report output there.

4. If you use the TO PRINTER option, **INFORMIX-4GL** sends the report output to your printer. The default printer command on UNIX systems is **lp** (or **lpr**; be sure to check with your system administrator) and on DOS systems is **lpt1**. You may change the default by setting the DBPRINT environment variable (see Appendix C).

5. The TO PIPE option applies only to the UNIX operating system where the report output is piped to *program*.

Related Statements

FINISH REPORT, OUTPUT TO REPORT, REPORT

UNLOCK TABLE

Overview

Use the UNLOCK TABLE statement to unlock a table that you previously locked with the LOCK TABLE statement.

Syntax

UNLOCK TABLE *table-name*

Explanation

UNLOCK TABLE	are required keywords.
table-name	is the name of the table you want to unlock.

Notes

1. If the database has transactions, the UNLOCK TABLE statement cannot be used. Any locks placed on the table will be released when the COMMIT WORK or ROLLBACK WORK statement is processed.

Related Statements

LOCK TABLE

UPDATE

Overview

Use the UPDATE statement to change the values in one or more columns of one or more rows in a table.

Syntax

```
UPDATE table-name SET {column-name = expr[, ... ] |
      {(column-list) | [table-name.]*} = {(expr-list) | record-name.* }}
   [WHERE {condition | CURRENT OF cursor-name}]
```

Explanation

UPDATE	is a required keyword.
table-name	is the name of the table that contains the row(s) you want to update.
SET	is a required keyword.
column-name	is the name of a column you want to update.
expr	is any combination of column names, constants, program variables, arithmetic operators, or an RDSQL subquery that returns a single row of one value.
column-list	is a list of the names of columns you want to update.
*	refers to all columns in *table-name*. (You can substitute *table-name.** if you prefer.)

expr-list is a list of expressions that represent values corresponding to the columns in *column-list* or the columns represented by the asterisk notation. In *expr-list* you can specify multiple values using a record name with the asterisk (*) or THRU notation. The list may also include an **RDSQL** subquery that returns a single row of multiple values.

record-name is the name of a program variable of type RECORD.

WHERE is an optional keyword.

condition is a condition for a standard WHERE clause made up of a search condition that compares the values in one column to the values in another column, to a program variable, or to a constant. (For further information, refer to the explanation of WHERE clauses in the section "The SELECT Statement" at the end of this chapter.)

CURRENT OF are keywords.

cursor-name is the **RDSQL** identifier of a previously DECLAREd cursor.

Notes

1. *expr* can be a SELECT statement in parentheses that adheres to standard rules for subqueries. The SELECT statement can return no more than one value except when included in an *expr-list*.

2. You cannot use a SELECT statement that retrieves data from *table-name*.

3. The number of column names included in the *column-list* must equal the number of values produced in the *expr-list*.

4. Although the value returned by *expr* does not have to be of the same data type as *column-name*, it must be compatible. You can put only CHAR data into CHAR columns and only numeric or character representations of numeric data into numeric columns.

5. If you use the CURRENT OF option in the WHERE clause, **RDSQL** updates the current row of the active set and leaves the cursor on the same row.

6. If you do not specify any columns in the FOR UPDATE clause of a DECLARE statement, you can update any column in a subsequent UPDATE WHERE CURRENT OF statement. If you do specify one or more columns in the FOR UPDATE clause, you can update only those columns in a subsequent UPDATE WHERE CURRENT OF statement. When you specify the column names in the FOR UPDATE clause, **RDSQL** can usually perform the updates more quickly.

7. SERIAL columns cannot be updated. If you want to use the [*table-name*].* notation and *table-name* contains a SERIAL column, you can only execute the following form of the UPDATE statement:

```
UPDATE table-name
  SET [table-name.]* = record-name.*
```

When **INFORMIX-4GL** executes this form of the UPDATE statement, it automatically skips any SERIAL column and the corresponding value in the expression list produced by *record-name.**.

8. For databases with transactions, all RDSQL statements executed within a BEGIN WORK and COMMIT WORK/ROLLBACK WORK block are treated as a single transaction. Because each row affected by an UPDATE statement within a transaction is locked for the duration of the transaction, a single UPDATE statement that affects a large number of rows locks those rows until the entire operation is completed. If the number of rows affected is very large, you may exceed the limits placed by your operating system on the maximum number of simultaneous locks. If this occurs, you may want either to reduce the scope of the UPDATE statement or to lock the entire table before executing the statement. See the section "Locking" in Chapter 2 for a more detailed description of table-level and row-level locking in RDSQL.

Caution! If RDSQL encounters an error while performing an UPDATE, the operation stops. Unless you have created the database with transactions, all database changes made up to the point where the error is encountered remain, but subsequent rows are not updated.

A data conversion error is an example of an error that stops an UPDATE operation. Common data conversion errors include attempting to insert numeric data into a CHAR column, or attempting to insert numeric values that exceed the limits of the data type of the column. For example, you cannot insert the integer 123456 into a SMALLINT column.

If you omit the WHERE clause, RDSQL assumes you want to update every row in the table.

Examples

```
UPDATE stock
    SET unit_price = unit_price * 1.04
    WHERE manu_code = "HRO"


UPDATE customer SET * = p_customer.*
    WHERE customer_num = p_customer.customer_num


UPDATE customer
    SET (fname, company, address2) =
            ("Marie", "Marie's Sports",
             "P. O. Box 3621")
    WHERE customer_num = 103


UPDATE table1
    SET (col1, col2, col3) =
            ((select min (ship_charge),
              max (ship_charge) from orders),
            "07/01/1986")
    WHERE col4 = 1001
```

Related Statements

SELECT, DELETE, INSERT

UPDATE STATISTICS

Overview

Use the UPDATE STATISTICS statement to cause the number of rows in a table to be recorded in the **systables** catalog.

Syntax

UPDATE STATISTICS [FOR TABLE *table-name*]

Explanation

UPDATE STATISTICS	are required keywords.
FOR TABLE	are optional keywords you use when you want to update the statistics for a single table.
table-name	is the name of the table for which you want the statistics updated.

Notes

1. UPDATE STATISTICS is effective only when there is a current database.

2. **RDSQL** uses the data generated by UPDATE STATISTICS to optimize searching strategy. When you have modified a table extensively, use UPDATE STATISTICS to improve the efficiency of queries.

3. **RDSQL** does not update the statistics unless you execute the UPDATE STATISTICS statement.

4. If you do not use the FOR TABLE clause, UPDATE STATISTICS will update all the tables in the current database.

VALIDATE

Overview

Use the VALIDATE statement to determine whether the values of a list of variables conform to the allowed range of values in **syscolval** for a corresponding list of columns.

Syntax

VALIDATE *variable-list* LIKE *column-list*

Explanation

VALIDATE	is a required keyword.
variable-list	is a list of one or more variables, separated by commas.
LIKE	is a required keyword.
column-list	is a list of one or more column names, separated by commas.

Notes

1. If the values of the components of *variable-list* do not conform entirely with the INCLUDE values of **syscolval**, INFORMIX-4GL will set the **status** variable to a negative value. You must test the variables individually to detect the non-conforming component.

2. You must use a table-name prefix in the designation of the column names.

Examples

```
VALIDATE p_customer.* LIKE customer.*
```

WHENEVER

Overview

Use the WHENEVER statement to trap errors and warnings that result during the execution of other statements.

Syntax

WHENEVER {ERROR | WARNING}
    {GOTO *label* | CALL *function-name* | CONTINUE | STOP}

Explanation

WHENEVER	is a required keyword.
ERROR	is an optional keyword indicating that an error has occurred during an **INFORMIX-4GL** statement (**status**< 0).
WARNING	is an optional keyword indicating that a warning has occurred during an **RDSQL** statement, for example, the truncation of a CHAR column value when moved from the database to a program variable: **SQLCA.SQLAWARN[2]** = W.
GOTO	is an optional keyword.
label	is a statement label to which program control transfers when the trap is triggered.
CALL	is an optional keyword.
function-name	is the name of a function to which program control transfers when the trap is triggered.

CONTINUE is an optional keyword indicating that INFORMIX-4GL should take no action. Use this option to turn off a previously set option.

STOP is an optional keyword instructing INFORMIX-4GL to exit from the program immediately.

Notes

1. The WHENEVER statement is shorthand for putting an IF statement after every RDSQL statement and form-related INFORMIX-4GL statement and testing for an error or warning.

2. The default for ERROR is STOP and that for WARNING is CONTINUE. In the default situation, INFORMIX-4GL will test for errors, but not warnings, after every INFORMIX-4GL statement.

3. There may be several WHENEVER statements in a program and, if they refer to the same keyword, the last one encountered rules.

4. The scope of a WHENEVER ERROR statement is the file in which it occurs and from the position of the WHENEVER ERROR statement to the next WHENEVER ERROR statement in the file (or to the end of the file, if there are no more WHENEVER ERROR statements). The scope of the WHENEVER WARNING statement is similar.

5. Since INFORMIX-4GL provides useful information (like source-file line numbers where the error occurred) when it terminates a program because of an error, you may want to use the default during program development and insert trapping at a later stage.

Examples

```
WHENEVER ERROR CALL error_recovery
WHENEVER WARNING stop
```

Related Statements

DEFER

WHILE

Overview

Use the WHILE statement to execute a group of statements while a condition is true.

Syntax

```
WHILE Boolean-expr
        statement
        ...
        [EXIT WHILE]
        ...
        [CONTINUE WHILE]
        ...
END WHILE
```

Explanation

WHILE	is a required keyword.
Boolean-expr	is an expression that can be either true or false.
statement	is an **INFORMIX-4GL** statement (including another WHILE statement).
EXIT WHILE	is an optional statement.
CONTINUE WHILE	is an optional statement.
END WHILE	are required keywords that terminate a WHILE statement.

Notes

1. The CONTINUE WHILE statement interrupts the sequence and causes the program control to return to the top of the sequence and to test the *Boolean-expr*.

2. The EXIT WHILE statement interrupts the sequence and causes the program control to jump to the first statement following the END WHILE keywords.

3. If *Boolean-expr* is false on entry to the WHILE statement, program control passes directly to the statement following END WHILE.

Related Statements

CONTINUE, EXIT, FOR

The SELECT Statement

Overview

Use the SELECT statement to query the current database.

The SELECT statement is made up of the following eight clauses. Only the SELECT clause and the FROM clause are required. If the INTO clause is present, it must precede the FROM clause.

Syntax

```
SELECT clause  [INTO clause] FROM clause
               [WHERE clause]
               [GROUP BY clause]
               [HAVING clause]
               [ORDER BY clause]
               [INTO TEMP clause]
```

These have the following syntax:

```
SELECT [ALL | DISTINCT | UNIQUE] select-list
```

```
INTO variable-list
```

```
FROM {table-name [table-alias] |
   OUTER table-name [table-alias] |
   OUTER (table-expr) } [,...]
```

WHERE *condition*

A *condition* is a collection of one or more *search conditions* connected by the logical operators AND, OR, or NOT. A search condition can be any of the following:

1. Comparison Condition

   a. *expr rel-op expr*

   b. *expr* [NOT] BETWEEN *expr* AND *expr*

   c. *expr* [NOT] IN (*value-list*)

   d. *column-name* [NOT] LIKE "*string*"

   e. *column-name* [NOT] MATCHES "*string*"

   f. *column-name* IS [NOT] NULL

2. Join Condition (a comparison condition among columns of the joined tables)

3. Condition with Subquery

   a. *expr rel-op* {ALL | ANY | SOME} (*SELECT-statement*)

   b. *expr* [NOT] IN (*SELECT-statement*)

   c. [NOT] EXISTS (*SELECT-statement*)

GROUP BY *column-list*

HAVING *condition*

ORDER BY *column-name* [ASC | DESC][, ...]

INTO TEMP *table-name*

Explanation

The following pages explain each of the syntax elements. A few basic concepts are defined here.

1. An *expression* consists of a column name, a program variable, or a constant, or any combination of these connected by the arithmetic operators:

Operator	Operation
+	addition
−	subtraction
*	multiplication
/	division

Note: Unlike **INFORMIX-4GL** statements, **RDSQL** statements may not contain expressions that use the exponentiation (**) or modulus (mod) operators.

The result of the operation must make sense. For example, you cannot divide 16 by Jones.

Column names in expressions must have an "at sign" (@) in front of them if there is danger of confusion with program variables with the same identifier.

RDSQL has two functions that can be used wherever a constant can be used. TODAY always returns your system's date. USER returns a string containing the current user's name. On UNIX systems, this is the login name, while on multi-user DOS systems, this is the machine name.

An expression may also be one of the aggregate or one of the date functions. You may not include both an aggregate function and a column in an expression. The aggregate and date functions that you can use in RDSQL statements are defined at the end of this chapter.

A CHAR column may have subscripts so that only a portion of the column value is involved in the expression. The notation for subscripting a column is *column-name*[*m*, *n*], where you want the *m*th through the *n*th characters of *column-name*. *m* must be less than or equal to *n*.

2. A *relational operator* is one of the following:

Operator	Operation
=	equal
!= or < >	not equal
>	greater than
>=	greater than or equal
<	less than
<=	less than or equal

For CHAR expressions, greater than means "after" in ASCII collating order, where lowercase letters are after uppercase letters, and both are after numerals. For the ASCII collating sequence of all the characters, see the ASCII chart in Appendix M.

For DATE expressions, greater than means later in time.

Notes

1. The clauses of the SELECT statement are explained in detail on the following pages. Briefly, the SELECT clause names a list of columns or expressions to be retrieved, the INTO clause names the program variables to receive the data, the FROM clause names a list of tables; the WHERE clause sets conditions on the rows; the GROUP BY clause groups rows together; the HAVING clause sets conditions on the groups; ORDER BY orders the selected rows; and INTO TEMP puts the results into a temporary table.

2. If the SELECT statement returns no rows, **INFORMIX-4GL** returns a "no rows found" code (**status** = NOTFOUND = 100). See Chapter 1 for a full explanation.

3. If a SELECT statement returns more than one row or if it is dynamically defined, you must use a cursor to fetch one row at a time (see Chapter 2).

4. It is sometimes helpful to think of the SELECT statement as follows:

   When you list more than one table in the FROM clause, **RDSQL** behaves as though it were creating a composite table that is the Cartesian product of all the tables in the FROM clause. That is, the rows of the new table are constructed by taking all the possible combinations of rows from all the tables listed in the FROM clause. If there is a WHERE clause, **RDSQL** eliminates from this new table all rows that do not meet the conditions of the WHERE clause. This modified table is returned to the SELECT clause, where all columns not listed are eliminated. The resulting table is what the SELECT statement returns.

SELECT Clause

Overview

Use the SELECT clause to specify the data you want to retrieve from one or more tables in a database.

Syntax

SELECT [ALL | DISTINCT | UNIQUE] *select-list*

Explanation

SELECT	is a required keyword.
ALL	is a keyword that causes **RDSQL** to select all rows that satisfy the WHERE clause, without eliminating duplicates. This keyword is the default.
DISTINCT	is a keyword that causes **RDSQL** to eliminate duplicate rows from the query results.
UNIQUE	is a keyword that is synonymous with DISTINCT.
select-list	is a list of column names and/or expressions separated by commas. A column name must be unambiguous; use its table name as a prefix if there can be confusion.

Notes

1. If the SELECT statement does not include a WHERE clause, every row will be returned.

2. You can use DISTINCT or UNIQUE only once in a query.

3. You may use the asterisk (*) in the *select-list* to select all columns from all the tables in the FROM clause. You could produce the same result by listing every column name in the *select-list*.

4. To select all the columns from a single table, use the notation *tablename.**.

5. You can supply a *display label* for the column name or an expression in the *select-list* by following the column name or expression with a legal identifier. If you create a temporary table with the INTO TEMP clause, the column names of the temporary table are the display labels if they have been defined.

6. If you specify an aggregate function and a column in the *select-list*, the column must be used in the GROUP BY list (see GROUP BY for further explanation).

Examples

The following examples will use INTO, FROM, and WHERE clauses, whose syntax will be defined later.

```
SELECT customer_num, @lname, city
    INTO cnum, lname, town
    FROM customer
```

This example selects columns **customer_num**, **lname**, and **city**; the FROM clause indicates that these columns are taken from the **customer** table. The values returned are placed in the program variables **cnum**, **lname**, and **town**, respectively. Since **lname** is both a program variable name and a column name, the column reference is prefixed with the at sign.

```
SELECT COUNT(*)
    INTO num
    FROM orders
    WHERE customer_num = 101
```

This statement counts the number of rows in **orders** in which the **customer_num** column contains the value 101. In other words, it counts the number of orders made by the customer whose identifying number is 101. The number is placed in the variable **num**.

```
SELECT AVG(total_price)
    FROM items
    WHERE order_num = 1021
```

This statement computes the average of the **total_price** values in those rows of **items** that contain an **order_num** column equal to 1021.

```
SELECT a+b abtotal, c*d cdprod
    FROM tablez
    INTO TEMP x
```

This example illustrates the use of display labels. It selects the sum of columns **a** and **b** from **tablez** and gives the sum the label **abtotal**. Similarly, the product of columns **c** and **d** is labeled **cdprod**.

INTO Clause

Use the INTO clause to specify the program variables to receive the data that is retrieved by the SELECT statement.

Syntax

INTO *variable-list*

Explanation

INTO	is a required keyword.
variable-list	is a list of program variables that should agree in order and type with the corresponding columns or expressions in the *select-list*.

Notes

1. If the SELECT statement stands alone (not in a DECLARE statement), it must be a singleton SELECT (returning exactly one row) and must have an INTO clause.

2. If the SELECT statement returns more than one row, you must use a cursor to FETCH the rows one at a time (see Chapter 2). You may put the INTO clause in the FETCH statement, rather than in the SELECT statement. You may use the FOREACH statement in place of the FETCH statement.

3. If the number of variables in *variable-list* differs from the number of items in the *select-list*, RDSQL returns a warning by setting **SQLCA.SQLAWARN[4]** to W. The actual number of variables transferred is the lesser of the two numbers.

4. If possible, RDSQL converts the data type of each selected item to match that of the receiving variable. If the conversion is not possible, an error occurs and a negative value is returned in **status**. In this case, the value in the program variable is unpredictable. See Chapter 1 for a discussion of data conversion.

Examples

```
DECLARE q_curs CURSOR FOR
     SELECT @lname, @company
     INTO lname, company
     FROM customer
OPEN q_curs
FETCH q_curs
```

or equivalently

```
DECLARE q_curs CURSOR FOR
     SELECT @lname, @company
     FROM customer
OPEN q_curs
FETCH q_curs
     INTO lname, company
```

Another alternative is

```
DECLARE q_curs CURSOR FOR
     SELECT @lname, @company
     FROM customer
FOREACH q_curs INTO lname, company
...
```

FROM Clause

Overview

Use the FROM clause to specify the table or tables you want to select data from.

Syntax

```
FROM {table-name [table-alias] |
    OUTER table-name [table-alias] |
    OUTER (table-expr)} [,...]
```

Explanation

FROM	is a required keyword.
OUTER	is an optional keyword.
table-name	is the name of a table that contains data you are searching for.
table-alias	is an optional alias for *table-name*.
table-expr	is one or more of the options in the syntax shown above (for example `tab1, outer tab2`). See Appendix L, "Complex Outer Joins," for a discussion of this syntax.

Notes

1. Use the optional keyword OUTER to form outer joins. See the "Outer Joins" section in this chapter and Appendix L for a detailed description of outer joins.

2. You can supply an alias for a table name by following the table name with a space and an RDSQL identifier. This feature is especially useful when performing self-joins (see the "WHERE Clause" section of this chapter).

Examples

The following example selects customers who have placed orders.

```
SELECT fname, lname, order_num
    FROM customer, orders
    WHERE customer.customer_num =
 orders.customer_num
```

The following example selects all customers whether or not they have placed orders.

```
SELECT fname, lname, order_num
    FROM customer, OUTER orders
    WHERE customer.customer_num =
        orders.customer_num
```

WHERE Clause

Overview

Use the WHERE clause to specify search criteria and join conditions on the data you want to select.

Syntax

WHERE *condition*

Explanation

WHERE is a required keyword.

condition is a collection of one or more *search conditions* connected by the logical operators AND, OR, or NOT. A search condition can be any of the following:

- Comparison Condition
- Join Condition
- Condition with Subquery

Comparison Condition

A comparison condition can have one of the following forms:

a. *expr rel-op expr*

b. *expr* [NOT] BETWEEN *expr* AND *expr*

c. *expr* [NOT] IN (*values list*)

d. *column-name* [NOT] LIKE "*string*"

e. *column-name* [NOT] MATCHES "*string*"

f. *column-name* IS [NOT] NULL

These forms are explained in the following pages. Any one of these conditions can be preceded by the keyword NOT, in which case, the negative of the condition must be satisfied.

Syntax

---

expr rel-op expr

Explanation

expr is an expression.

rel-op is a relational operator.

Examples

```
SELECT fname, lname, company
    FROM customer
    WHERE city[1,3] = "San"

SELECT order_num, company
    FROM orders o, customer c
    WHERE o.order_date > "6/12/86"
        AND o.customer_num = c.customer_num
```

If a column value is NULL in a given row, the WHERE clause will not locate that row if you use relational operators. For example, if **paid_date** has a NULL value, you cannot use either statement (a) or (b) to retrieve that row:

```
(a)  SELECT customer_num, order_date
        FROM orders
        WHERE paid_date = ""
```

```
(b)  SELECT customer_num, order_date
        FROM orders
        WHERE NOT paid_date != ""
```

You must use the syntax that follows:

```
     SELECT customer_num, order_date
        FROM orders
        WHERE paid_date IS NULL
```

Syntax

expr [NOT] BETWEEN *expr* AND *expr*

Explanation

expr	is an expression.
NOT	is a keyword that indicates that the expression to the left lies outside the range.
BETWEEN	is a keyword that indicates that the value of the expression to its left lies in the inclusive range of the values of the two expressions to its right.
AND	is a required keyword.

Examples

```
SELECT stock_num, manu_code
    FROM stock
    WHERE unit_price BETWEEN
        lowprice AND hiprice

SELECT UNIQUE customer_num, stock_num, manu_code
    FROM orders, items
    WHERE order_date
        BETWEEN "6/1/86" AND "9/7/86"
        AND orders.order_num = items.order_num

SELECT fname, lname
    FROM customer
    WHERE zipcode NOT
        BETWEEN "94100" AND "94199"
```

Syntax

expr [NOT] IN (*value-list*)

Explanation

expr is an expression.

NOT is an optional keyword.

IN is a required keyword.

value-list is a list of values enclosed in parentheses.

Notes

1. The search condition is satisfied when the expression to the left is included in the list of items. The NOT option produces a search condition that is satisfied when *expr* is not in the list of items.

2. *value-list* may contain program variables, constants, or the special keyword constants TODAY and USER.

Examples

```
SELECT lname, fname, company
    FROM customer
    WHERE state IN ("CA", "WA", "OR")

SELECT item_num, total_price
    INTO inum, tprice
    FROM items
    WHERE manu_code IN
        ("HRO", "HSK")
```

Syntax

---

column-name [NOT] LIKE "*string*"

Explanation

column-name	is the name of a column.
NOT	is an optional keyword.
LIKE	is a required keyword.
string	is a pattern of characters enclosed in quotation marks.

Notes

1. The search condition is successful when the value of the column on the left matches the pattern specified by *string*. You can use wildcard characters in place of other characters in the string:

   % A percent sign matches zero or more characters.

   _ An underscore matches any single character.

2. The NOT option makes the search condition successful when the column on the left does not match the pattern specified by *string*.

Examples

```
SELECT fname, lname
    FROM customer
    WHERE lname LIKE "%son"
```

finds every customer representative whose last name ends in *son*.

```
SELECT stock_num, manu_code, unit_price
    FROM stock
    WHERE description LIKE "%ball%"
```

obtains stock number, manufacturer's code, and unit price for all stock items with *ball* anywhere in their description.

Syntax

column-name [NOT] MATCHES "*string*"

Explanation

column-name	is the name of a column.
NOT	is an optional keyword.
MATCHES	is a required keyword.
string	is a pattern of characters enclosed in quotation marks.

Notes

1. The search condition is successful when the value of the column on the left matches the pattern specified by *string*. You can use three wildcard characters in place of other characters in the string:

*	matches zero or more characters
?	matches any single character
[...]	matches any of the enclosed characters, including character ranges as in [a-z]. A caret (^) as the first character within the brackets matches any character that is *not* listed. [^abc] matches any character that is not **a** or **b** or **c**.
\	escapes special significance of the next character (used to match * or ? by writing * or \?)

2. The NOT option makes the search condition successful when the column on the left does not match the pattern specified by *string*.

3. The MATCHES comparison is an RDS extension to retain compatibility with earlier versions of INFORMIX.

4. Values used in a MATCHES search must be type CHAR.

Examples

```
SELECT fname, lname
    FROM customer
    WHERE lname MATCHES "Richard*"
```

selects rows in which the first seven letters of the last name are *Richard* (thus matching Richard, Richards, Richardson, and any others).

```
SELECT customer_num, company
    FROM customer
    WHERE city matches "[A-J]*"
```

provides the customer number and company name for all customers in cities that start with the letters *A* through *J*.

Syntax

column-name IS [NOT] NULL

Explanation

column-name	is the name of a column.
IS NULL	are required keywords.
NOT	is an optional keyword.

Examples

```
SELECT order_num, customer_num
   FROM orders
   WHERE paid_date IS NULL
```

lists those order numbers and customer numbers where the order has not been paid.

You can link any number of the above described conditions together using the logical operators AND or OR. For example:

```
SELECT order_num, total_price
   FROM items
   WHERE total_price > loprice AND
      manu_code LIKE "H%"

SELECT lname, customer_num
   FROM customer
   WHERE zipcode BETWEEN
      "93500" AND "95700"
      OR state NOT IN
      ("CA", "WA", "OR")
```

Join Conditions

Overview

You join two tables when you create a relationship in the WHERE clause between at least one column from one table and at least one column from the other. The effect of the join is to create a temporary composite table in which each pair of rows (one from each table) satisfying the join condition is linked together to form a single row.

Syntax

The critical elements of a SELECT statement with a join between two tables *table1* and *table2* follow:

SELECT clause FROM *table1*, *table2* WHERE *condition*

Explanation

SELECT clause	is any valid SELECT clause involving columns from *table1* or *table2*.
FROM	is a required keyword.
table1	is one of the tables in the join.
table2	is the other table in the join.
WHERE	is a required keyword.
condition	is any of the comparison conditions that involves columns from both *table1* and *table2*.

Notes

1. When columns from different tables have the same name, you must distinguish them by prefixing the table identifier and a period—**table.column**.

2. A *multiple join* is a join of more than two tables. Its structure is similar to that shown previously, except that you have a join condition for more than one pair of tables in the FROM clause.

3. You can also join a table to itself in a *self-join*. To do so, you must list the table name twice in the FROM clause, assigning it two different aliases. Use the aliases to refer to each of the "two" tables in the WHERE clause.

4. An *outer join* occurs when every row from *table1* is taken whether or not the join condition is met. If the join condition is not met, the columns from *table2* in the *select-list* are set to NULL values. You indicate an outer join by inserting the keyword OUTER before the table name *table2*.

Examples

A two-table join:

```
SELECT order_num, lname, fname
    FROM customer, orders
    WHERE customer.customer_num
        = orders.customer_num
```

lists the order number and first and last names of the customer's representative for each order.

A multiple-table join:

```
SELECT UNIQUE company, stock_num, manu_code
   FROM customer c, orders o, items i
   WHERE c.customer_num = o.customer_num
      and o.order_num = i.order_num
```

yields the company name of the customer who ordered an item identified with the stock number and manufacturer code.

Self-join:

```
SELECT x.stock_num, x.manu_code,
       y.stock_num, y.manu_code
   FROM stock x, stock y
   WHERE x.unit_price > 2.5 * y.unit_price
```

finds pairs of stock items whose unit prices differ by a factor greater than two and one-half. **x** and **y** are each aliases for the **stock** table.

Outer Join:

```
SELECT company, order_num
   FROM customer c, OUTER orders o
   WHERE c.customer_num = o.customer_num
```

will list all the customers' company name and order numbers, if the customer has placed an order. If not, the company name will still be listed. See the "Outer Joins" section in Chapter 2 and Appendix L for information about outer joins.

Subqueries

Overview

The search condition in a SELECT statement can also

- Compare an expression to the result of another SELECT statement
- Determine whether an expression is included in the results of another SELECT statement
- Ask whether there are any rows selected by another SELECT statement

SELECT statements within a WHERE clause are called *subqueries*. A subquery may return a single value, no values, or a set of values, but it must have only a single column or expression in its *select-list* and must not contain an ORDER BY clause. The subquery can be dependent upon the current row being evaluated by the outer SELECT statement (correlated subqueries).

Syntax

WHERE *expr rel-op* {ALL | [ANY | SOME]} (*SELECT-statement*)

WHERE *expr* [NOT] IN (*SELECT-statement*)

WHERE [NOT] EXISTS (*SELECT-statement*)

Explanation

WHERE is a required keyword.

expr is an expression.

rel-op is a relational operator.

ALL is a keyword that denotes that the subquery may return zero, one, or more values and that the search condition is true if the comparison is true for each of the values returned. If the subquery returns no value, the search condition is true.

ANY is a keyword that denotes that the subquery may return zero, one, or more values and that the search condition is true if the comparison is true for at least one of the values returned. If the subquery returns no value, the search condition is false.

SOME is an alias for ANY.

IN is a keyword that asks whether *expr* is among the values returned by the following *SELECT-statement.*

EXISTS is a keyword that asks whether there are any rows returned by the following *SELECT-statement.* The search condition is true if the subquery returns one or more rows.

NOT is an optional keyword that reverses the truth value of the search condition.

Notes

1. The keywords ANY and ALL may be omitted in a comparison if you know the subquery will return exactly one value. In this case, the search condition is true if the comparison is true for the expression and the value returned by the subquery. **status** will be set to a negative number if the subquery returns other than a single value.

2. *expr* IN (*SELECT-statement*) is equivalent to *expr* =ANY (*SELECT-statement*).

3. *expr* NOT IN (*SELECT-statement*) is equivalent to *expr* !=ALL (*SELECT-statement*).

Examples

```
SELECT order_num
    FROM items
    WHERE stock_num = 9 AND quantity =
        (SELECT MAX(quantity) FROM items
         WHERE stock_num = 9)
```

This subquery returns a single value (the maximum number of volleyball nets ordered) to the outer-level query. The entire SELECT statement lists the order numbers for orders that include the maximum number of volleyball nets.

```
SELECT UNIQUE order_num
    FROM items
    WHERE total_price >ALL
        (SELECT total_price
            FROM items
            WHERE order_num = 1011)
```

This query lists the order numbers of all orders containing an item whose total price is greater than the total price on all the items in order number 1011.

```
SELECT UNIQUE customer_num
    FROM orders
    WHERE order_num NOT IN
        (SELECT order_num
            FROM items
          WHERE stock_num = 1)
```

This query lists all customer numbers corresponding to customers who have placed orders that do not include baseball gloves (**stock_num** = 1).

```
SELECT order_num, stock_num, manu_code, total_price
    FROM items x
    WHERE total_price >
        (SELECT 2 * MIN(total_price)
            FROM items
            WHERE order_num = x.order_num)
```

This query (using a correlated subquery) lists all items whose total price is at least twice the minimum total price for all items on the same order.

GROUP BY Clause

Overview

Use the GROUP BY clause to produce a single row of results for each group. A group is a set of rows having the same values for each column listed.

Syntax

GROUP BY *group-list*

Explanation

GROUP BY	are required keywords.
group-list	is a column name or a list of column names, separated by commas, that determines the group. The query result contains a single row for each set of rows that satisfies the WHERE clause and contains a unique value or set of values in the column or columns indicated by *group-list*.

Notes

1. Using a GROUP BY clause restricts what you can enter in the SELECT clause. The *select-list* may include aggregate functions for any column and/or the name of any column that you also list in the GROUP BY clause. You may not, however, list any column in the *select-list* that you do not also list in *group-list*.

2. You cannot list a column in *group-list* that participates in *select-list* only as a part of an expression.

3. In the place of column names in *group-list*, you can enter one or more integers that refer to the position of items in the *select-list*.

Examples

```
SELECT order_num, COUNT(*), SUM(total_price)
    FROM items
    GROUP BY order_num
```

obtains the number of items and total price of all items for each order.

```
SELECT order_num, COUNT(*), SUM(total_price)
    FROM items
    GROUP BY 1
```

returns the same information as the previous example.

HAVING Clause

Overview

Use the HAVING clause to apply one or more qualifying conditions to groups.

Syntax

HAVING *condition*

Explanation

HAVING is a required keyword.

condition is a condition as defined for the WHERE clause.

Notes

1. Each condition compares one aggregate property of the group either with another aggregate property of the group or with a constant.

2. The HAVING clause generally complements a GROUP BY clause. If you use HAVING without GROUP BY, the HAVING clause applies to all rows that satisfy the WHERE clause. Without a GROUP BY clause, all rows that satisfy the WHERE clause make up a single group.

Examples

```
SELECT order_num, AVG(total_price)
    FROM items
    GROUP BY order_num
    HAVING COUNT(*) > 2
```

This query returns average total price per item on all orders that have more than two items.

ORDER BY Clause

Overview

Use the ORDER BY clause to sort query results by the values contained in one or more columns. You can sort only by a column that you select in the SELECT clause.

Syntax

ORDER BY *column-name* [ASC | DESC][, ...]

Explanation

ORDER BY	are required keywords.
column-name	is the name of a column by which you want to sort the query results.
ASC	is a keyword that specifies that the results should be in ascending order. ASC is the default.
DESC	is a keyword that specifies that the results should be in descending order.

Notes

1. You can ORDER BY up to eight columns.
2. You can only ORDER BY columns that are named explicitly or implicitly in the *select-list*.
3. The total length of the data in the columns included in the ORDER BY clause cannot be greater than 120 bytes.

4. In the place of column names, you can enter one or more integers that refer to the position of items in the *select-list*. In this way, you can ORDER BY an expression.

Examples

```
CORRECT:     SELECT order_date, ship_date
                FROM orders
                ORDER BY order_date

CORRECT:     SELECT *
                FROM orders
                ORDER BY order_date

INCORRECT:   SELECT order_date, ship_date
                FROM orders
                ORDER BY customer_num
```

In the first two examples, **order_date** is either explicitly or implicitly listed in the *select-list*. In the third example, even though **customer_num** is in the **orders** table, it was not contained in the *select-list*.

```
SELECT customer_num, fname, lname, company
   FROM customer
   ORDER BY 3, 2
```

performs a nested sort, the first level being the **lname** column, the second level being the **fname** column.

INTO TEMP Clause

Overview

Use the INTO TEMP clause to create a temporary table that contains the query results. The temporary table disappears when your program ends.

Syntax

INTO TEMP *table-name*

Explanation

INTO TEMP	are required keywords.
table-name	is the RDSQL identifier you want to assign to the temporary table.

Notes

1. You will save time if you use a temporary table when the same query results are required several times.

2. An INTO TEMP clause in a SELECT statement often gives you clearer and more easily understood statements.

3. The column names of the temporary table are those named in the *select-list*. If you list a display label for a column or expression, the column name in the temporary table is the display label.

4. There are no indexes associated with *table-name*.

5. It is an error to use an INTO clause with the INTO TEMP clause. If you do, no results will be returned to the program variables, and **status** will be set to a negative value.

6. The temporary table persists until you exit your **INFORMIX-4GL** program or issue the DROP TABLE statement for it.

UNION Operator

Overview

The UNION operator is a keyword placed between two SELECT statements that let you combine the queries into a single query.

Syntax

SELECT-statement UNION [ALL] *SELECT-statement*
[UNION [ALL] *SELECT-statement* ...]

Explanation

UNION	is a keyword that selects all rows from both queries, removes duplicates, and returns what is left.
ALL	is an optional keyword that leaves the duplicates.

Notes

1. The UNION operator can be placed between each member of a sequence of more than two queries.

2. It is possible to write single queries that are equivalent to the compound queries constructed using the UNION operator. The advantage of the UNION operator is that it makes the process easier.

3. There are restrictions on the queries that you may connect with UNION operators.

   - The number of the items in the *select-list* of each query must be the same, and corresponding items in each *select-list* must have identical data types.

   - Corresponding items need not have the same identifier.

   - You may not use an INTO statement in a query unless you are sure that the compound query will return exactly one row and you are not using a cursor. In this rare case, the INTO clause must be in the first SELECT statement.

   - If you use an ORDER BY clause, it must follow the last SELECT statement, and you must refer by integer, not by identifier, to the item to be ordered. Ordering takes place after the set operation is complete.

4. UNION operators may not occur inside a subquery and may not be used in the definition of a view.

5. The column names (or display labels) of the resulting table are the same as those from the first SELECT statement.

6. You may put the results of a UNION into a temporary table by putting an INTO TEMP clause in the final SELECT statement.

Examples

The following example selects those items (identified by a stock number and a manufacturing code) that have a unit price of less than $100.00 or that have been ordered in quantities greater than 3.

```
SELECT DISTINCT stock_num, manu_code
    FROM stock
    WHERE unit_price < 100.00

UNION

SELECT stock_num, manu_code,
    FROM items
    WHERE quantity > 3
    ORDER BY 1
```

Aggregate and Date Functions

You may use the aggregate and the date functions anywhere in an **RDSQL** expression that a constant may appear. These functions are:

Aggregate Functions	**Date Functions**
COUNT( )	DATE( )
SUM( )	DAY( )
AVG( )	MDY( )
MAX( )	MONTH( )
MIN( )	WEEKDAY( )
	YEAR( )

They are defined on the following pages.

Aggregate Functions

Overview

The aggregate functions take on values that depend on the set of rows returned by the WHERE clause of a SELECT statement. In the absence of a WHERE clause, the aggregate functions take on values that depend on all the rows formed by the FROM clause.

Syntax	Function
COUNT(*)	returns the number of rows that satisfy the WHERE clause.
COUNT(DISTINCT *x*)	returns the number of unique values in column *x* that satisfy the WHERE clause.
SUM([DISTINCT] *x*)	returns the sum of all values in column *x* that satisfy the WHERE clause. You can apply SUM only to numeric columns. If you use the keyword DISTINCT, the sum is only over distinct values in column *x*.
AVG([DISTINCT] *x*)	returns the average of all values in column *x* that satisfy the WHERE clause. You can apply AVG only to numeric columns. If you use the keyword DISTINCT, the average is only over distinct values in column *x*.
MAX(*x*)	returns the highest value contained in column *x* from a row that satisfies the WHERE clause.

MIN(x)	returns the lowest value contained in column x from a row that satisfies the WHERE clause.

Notes

1. In the functions SUM(x), AVG(x), MAX(x), and MIN(x), x may be an expression instead of a column. In this case, the function is evaluated over the values of the expression as computed for each row that satisfies the WHERE clause. The expression may not contain another aggregate function.

2. You may use the DISTINCT keyword only with columns, not with expressions.

3. In a *select-list* you may use the keyword DISTINCT only once: either to eliminate duplicate query results or to eliminate duplicates from the argument of an aggregate function.

4. NULL values affect the aggregate functions as follows: COUNT(*) counts all rows, even if the value of every column in the row is NULL. COUNT(DISTINCT x), AVG (x), SUM (x), MAX (x), and MIN (x) ignore rows with NULL values for x and return the appropriate values based on the rest of the rows. If x contains only NULL values, however, then COUNT(DISTINCT x) returns zero, and the other four aggregate functions return NULL for that column.

DATE()

Overview

The DATE function returns a type DATE value corresponding to the expression you call it with.

Syntax

DATE(*expr*)

Explanation

DATE	is a required keyword.
expr	is a required expression that can be converted to a type DATE value.

Examples

The following example uses DATE to convert a string to a date.

```
WHERE end_date > DATE ("12/13/1986")
```

date(100) returns the 100^{th} day after December 31, 1899.

DAY()

Overview

The DAY function returns the day of the month when you call it with a type DATE expression.

Syntax

DAY(*date-expr*)

Explanation

DAY	is a required keyword.
date-expr	is a required expression of type DATE.

Examples

```
SELECT order_num, DAY (order_date) FROM orders
```

MDY()

Overview

The MDY function returns a type DATE value when you call it with three expressions that evaluate to integers representing the month, date, and year.

Syntax

MDY(*expr1*, *expr2*, *expr3*)

Explanation

MDY	is a required keyword.
expr1	is an expression that evaluates to an integer representing the number of the month (1–12).
expr2	is an expression that evaluates to an integer representing the number of the day of the month (1–28, 29, 30, or 31, depending on the month).
expr3	is an expression that evaluates to a four-digit integer representing the year.

Notes

1. The value of *expr3* cannot be the abbreviation for the year. 86 is in the first century.

MONTH()

Overview

The MONTH function returns an integer corresponding to its type DATE argument.

Syntax

MONTH(*date-expr*)

Explanation

MONTH	is a required keyword.
date-expr	is a required expression of type DATE.

Examples

```
SELECT order_num, MONTH (order_date) FROM orders
```

WEEKDAY()

Overview

The WEEKDAY function returns an integer that represents the day of the week when you call it with a type DATE expression.

Syntax

WEEKDAY(*date-expr*)

Explanation

WEEKDAY	is a required keyword.
date-expr	is a required expression of type DATE.

Notes

1. WEEKDAY returns an integer in the range 0-6. 0 represents Sunday, 1 represents Monday, and so on.

Examples

```
SELECT order_num, WEEKDAY (order_date) FROM orders
```

YEAR()

Overview

The YEAR function returns an integer that represents the year when you call it with a type DATE expression.

Syntax

YEAR(*date-expr*)

Explanation

YEAR	is a required keyword.
date-expr	is a required expression of type DATE.

Examples

```
SELECT order_num, YEAR (order_date) FROM orders
```

List of Appendixes

Appendix A

Demonstration Database and Application

Demonstration Database and Application

This appendix contains five sections.

- The first section describes the structure of each table in the **stores** database, presents the RDSQL statement used to create each table, and notes any indexes on columns in each table.

- The second section presents a map of the **stores** database with potential join columns identified.

- The third section discusses the join columns that link the six tables in the database and presents figures illustrating these relationships.

- The fourth section shows the data contained in each table in the **stores** database.

- The final section contains the form specifications, INFORMIX-4GL source code modules, and help message source code for the demonstration application.

Structure of the Tables

The **stores** database contains information about a fictitious sporting goods distributor that services stores in the Western United States. The database is made up of the following six tables:

- customer
- orders
- items
- stock
- manufact
- state

customer

The **customer** table contains information about 18 stores that order sporting goods from the distributor. This information includes the name of the store representative and the store name, address, and telephone number. The columns of the **customer** table are as follows:

```
customer_num        serial(101)
fname               char(15)
lname               char(15)
company             char(20)
address1            char(20)
address2            char(20)
city                char(15)
state               char(2)
zipcode             char(5)
phone               char(18)
```

The **customer_num** column is indexed as unique. The **zipcode** column is indexed to allow duplicate values.

orders

The **orders** table contains information about orders placed by the distributor's customers. This information includes the order number, date the order was made, the customer number, shipping instructions, whether a backlog exists, the customer's purchase order number, date the order was shipped, weight of the order, shipment charge, and the date the customer paid for the order. The columns of **orders** are as follows:

order_num	serial(1001)
order_date	date
customer_num	integer
ship_instruct	char(40)
backlog	char(1)
po_num	char(10)
ship_date	date
ship_weight	decimal(8,2)
ship_charge	money(6)
paid_date	date

The **order_num** column is indexed and must contain unique values; the **customer_num** column is indexed but allows duplicates.

items

The **items** table keeps track of individual items in an order. For example, some orders only contain one item, while other orders contain as many as five items. Information in the **items** table includes the item number, order number, stock number, manufacturer code, quantity, and the total price for each item that has been ordered. The columns of the **items** table are as follows:

item_num	smallint
order_num	integer
stock_num	smallint
manu_code	char(3)
quantity	smallint
total_price	money(8)

The **order_num** column is indexed and allows duplicate values. A multiple-column index for the **stock_num** and **manu_code** columns also permits duplicate values.

stock

The distributor offers customers 15 different types of sporting goods. For example, the distributor offers baseball gloves from three different manufacturers and basketballs from one manufacturer.

The **stock** table is a catalog of the items sold by the distributor. For each item in stock, the **stock** table lists a stock number identifying the type of item, a code identifying the manufacturer, a description of the item, its unit price, the unit by which the item must be ordered, and a description of the unit (for example, a case containing 10 baseballs).

The **stock** table contains the following columns:

stock_num	smallint
manu_code	char(3)
description	char(15)
unit_price	money(6)
unit	char(4)
unit_descr	char(15)

A multiple-column index for the **stock_num** and **manu_code** columns allows only unique values.

manufact

The distributor handles sporting goods from five manufacturers. Information about these manufacturers is kept in the **manufact** table. This information consists of an identification code and the manufacturer's name. The columns in the **manufact** table are as follows:

manu_code	char(3)
manu_name	char(15)

The **manu_code** column is indexed and must contain unique values.

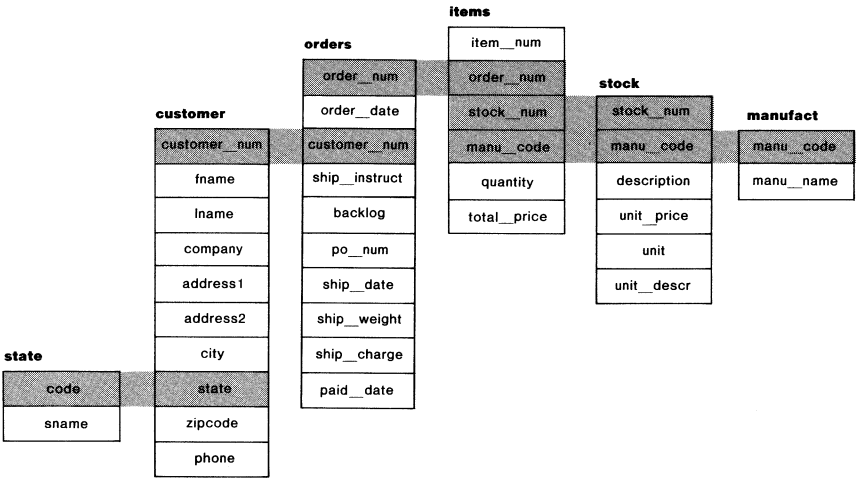
state

The **state** table contains the names and abbreviations of the states. It includes the following columns:

code	char(2)
sname	char(15)

The **code** column is indexed and must contain unique values.

Map of the **stores** *Database*



Join Columns That Link the Database

The six tables of the **stores** database are linked together by join columns. This section identifies these columns and describes how you can use them to retrieve data from several tables at once and display it as if it were stored in a single table. Figures show the relationships among the tables and how information stored in one table supplements information in other tables.

Join Columns in the *customer* and *orders* Tables

The **customer** table and the **orders** table are joined by the **customer_num** column, as shown in the following example:

customer table (detail)

customer_num	fname	lname
101	Ludwig	Pauli
102	Carole	Sadler
103	Philip	Currie
104	Anthony	Higgins

orders table (detail)

order_num	order_date	customer_num
1001	01/20/1986	104
1002	06/01/1986	101
1003	10/12/1986	104
1004	04/12/1986	106

Figure A-1. Tables Joined by the **customer_num** Column

For example, the **customer** table contains a **customer_num** column that holds a number identifying the customer, along with columns for name, company, address, and telephone number. The row with information about Anthony Higgins contains the number 104 in the **customer_num** column. The **orders** table also contains a **customer_num** column that stores the number of the customer who placed a particular order. According to Figure A-1, customer 104 (Anthony Higgins) has placed two orders, since his customer number appears in two rows of the **orders** table.

The join relationship lets you select information from both tables. This means you can retrieve Anthony Higgins's name and address and information about his orders at the same time.

Join Columns in the *orders* and *items* Tables

The **orders** and **items** tables are linked by an **order_num** column that contains an identification number for each order. If a particular order includes several items, the same order number appears in several rows of the **items** table. Figure A-2 shows this relationship.

orders table (detail)

order_num	order_date	customer_num
1001	01/20/1986	104
1002	06/01/1986	101
1003	10/12/1988	104

items table (detail)

item_num	order_num	stock_num	manu_code
1	1001	1	HRO
1	1002	4	HSK
2	1002	3	HSK
1	1003	9	ANZ
2	1003	8	ANZ
3	1003	5	ANZ

Figure A-2. Tables Joined by the **order_num** Column

Join Columns in the *items* and *stock* Tables

The **items** table and the **stock** table are joined by two columns: the **stock_num** column stores a stock number for an item, and the **manu_code** column stores a code that identifies the manufacturer. You need both the stock number and the manufacturer code to uniquely identify an item. For example, the item with the stock number 1 and the manufacturer code HRO is a Hero baseball glove, while the item with the stock number 1 and the manufacturer code HSK is a Husky baseball glove.

The same stock number and manufacturer code may appear in more than one row of the **items** table if the same item belongs to separate orders, as shown in Figure A-3.

items table (detail)

item_num	order_num	stock_num	manu_code
1	1001	1	HRO
1	1002	4	HSK
2	1002	3	HSK
1	1003	9	ANZ
2	1003	8	ANZ
3	1003	5	ANZ
1	1004	1	HRO

stock table (detail)

stock_num	manu_code	description
1	HRO	baseball glove
1	HSK	baseball glove
1	SMT	baseball glove

Figure A-3. Tables Joined by Two Columns

Join Columns in the *stock* and *manufact* Tables

The **stock** table and the **manufact** table are joined by the **manu_code** column. The same manufacturer code may appear in more than one row of the **stock** table if the manufacturer produces more than one piece of equipment. This relationship is illustrated in Figure A-4.

stock table (detail)

stock_num	manu_code	description
1	HRO	baseball glove
1	HSK	baseball glove
1	SMT	baseball glove
2	HRO	baseball

manufact table (detail)

manu_code	manu_name
NRG	Norge
HSK	Husky
HRO	Hero

Figure A-4. Tables Joined by the **manu_code** Column

Join Columns in the *state* and *customer* Tables

The **state** table and the **customer** table are joined by a column that contains the state code. This column is called **code** in the **state** table and **state** in the **customer** table. If several customers live in the same state, the same state code will appear in several rows of the **customer** table, as shown in Figure A-5.

customer tal
(detail)

customer_num	fname	lname	...	state
101	Ludwig	Pauli	...	CA
102	Carole	Sadler	...	CA
103	Philip	Currie	...	CA

code	sname
AK	Alaska
AL	Alabama
AR	Arkansas
AZ	Arizona
CA	California

state table
(detail)

Figure A-5. Tables Joined by the **State-Code Column**

Joining lets you rearrange your view of a database whenever you want. It provides flexibility that lets you create new relationships between tables without redesigning the database. You can easily expand the scope of a database by adding new tables that join the existing tables. As you read through this manual, you will see programs that set up the join relationships across tables of the **stores** database. You can refer to the preceding figures whenever you need to review these relationships.

Data in the **stores** *Database*

The data in the **stores** database is displayed in the tables that follow.

customer Table

customer_num	fname	lname	company	address1	address2	city	state	zipcode	phone
101	Ludwig	Pauli	All Sports Supplies	213 Erstwild Court		Sunnyvale	CA	94086	408-789-8075
102	Carole	Sadler	Sports Spot	785 Geary St		San Francisco	CA	94117	415-822-1289
103	Philip	Currie	Phil's Sports	654 Poplar	P. O. Box 3498	Palo Alto	CA	94303	415-328-4543
104	Anthony	Higgins	Play Ball!	East Shopping Cntr.	422 Bay Road	Redwood City	CA	94026	415-368-1100
105	Raymond	Vector	Los Altos Sports	1899 La Loma Drive		Los Altos	CA	94022	415-776-3249
106	George	Watson	Watson & Son	1143 Carver Place		Mountain View	CA	94063	415-389-8789
107	Charles	Ream	Athletic Supplies	41 Jordan Avenue		Palo Alto	CA	94304	415-356-9876
108	Donald	Quinn	Quinn's Sports	587 Alvarado		Redwood City	CA	94063	415-544-8729
109	Jane	Miller	Sport Stuff	Mayfair Mart	7345 Ross Blvd.	Sunnyvale	CA	94086	408-723-8789
110	Roy	Jaeger	AA Athletics	520 Topaz Way		Redwood City	CA	94062	415-743-3611
111	Frances	Keyes	Sports Center	3199 Sterling Court		Sunnyvale	CA	94085	408-277-7245
112	Margaret	Lawson	Runners & Others	234 Wyandotte Way		Los Altos	CA	94022	415-887-7235
113	Lana	Beatty	Sportstown	654 Oak Grove		Menlo Park	CA	94025	415-356-9982
114	Frank	Albertson	Sporting Place	947 Waverly Place		Redwood City	CA	94062	415-886-6677
115	Alfred	Grant	Gold Medal Sports	776 Gary Avenue		Menlo Park	CA	94025	415-356-1123
116	Jean	Parmelee	Olympic City	1104 Spinosa Drive		Mountain View	CA	94040	415-534-8822
117	Arnold	Sipes	Kids Korner	850 Lytton Court		Redwood City	CA	94063	415-245-4578
118	Dick	Baxter	Blue Ribbon Sports	5427 College		Oakland	CA	94609	415-655-0011

orders Table

order_num	order_date	customer_num	ship_instruct	backlog	po_num	ship_date	ship_weight	ship_charge	paid_date
1001	01/20/86	104	ups	n	B77836	02/01/86	20.40	10.00	03/22/86
1002	06/01/86	101	po on box; deliver back door only	n	9270	06/06/86	50.60	15.30	07/03/86
1003	10/12/86	104	via ups	n	B77890	10/13/86	35.60	10.80	11/04/86
1004	04/12/86	106	ring bell twice	y	8006	04/30/86	95.80	19.20	
1005	12/04/86	116	call before delivering	n	2865	12/19/86	80.80	16.20	12/30/86
1006	09/19/86	112	after 10 am	y	Q13557		70.80	14.20	
1007	03/25/86	117		n	278693	04/23/86	125.90	25.20	
1008	11/17/86	110	closed Monday	y	LZ230	12/06/86	45.60	13.80	12/21/86
1009	02/14/86	111	door next to supersaver	n	4745	03/04/86	20.40	10.00	04/21/86
1010	05/29/86	115	deliver 776 Gary if no answer	n	429Q	06/08/86	40.60	12.30	07/22/86
1011	03/23/86	104	ups	n	B77897	04/13/86	10.40	5.00	06/01/86
1012	06/05/86	117		n	278701	06/09/86	70.80	14.20	
1013	09/01/86	104	via ups	n	B77930	09/18/86	60.80	12.20	10/10/86
1014	05/01/86	106	ring bell, kick door loudly	n	8052	05/10/86	40.60	12.30	07/18/86
1015	07/10/86	110	closed Mon	n	MA003	08/01/86	20.60	6.30	08/31/86

items Table

item_num	order_num	stock_num	manu_code	quantity	total_price
1	1001	1	HRO	1	250.0
1	1002	4	HSK	1	960.0
2	1002	3	HSK	1	240.0
1	1003	9	ANZ	1	20.0
2	1003	8	ANZ	1	840.0
3	1003	5	ANZ	5	99.0
1	1004	1	HRO	1	960.0
2	1004	2	HRO	1	126.0
3	1004	3	HSK	1	240.0
4	1004	1	HSK	1	800.0
1	1005	5	NRG	10	280.0
2	1005	5	ANZ	10	198.0
3	1005	6	SMT	1	36.0
4	1005	6	ANZ	1	48.0
1	1006	5	SMT	5	125.0
2	1006	5	NRG	5	190.0
3	1006	5	ANZ	5	99.0
4	1006	6	SMT	1	36.0
5	1006	6	ANZ	1	48.0
1	1007	1	HRO	1	250.0
2	1007	2	HRO	1	126.0
3	1007	3	HSK	1	240.0
4	1007	4	HRO	1	480.0
5	1007	7	HRO	1	600.0
1	1008	8	ANZ	1	840.0
2	1008	9	ANZ	5	100.0
1	1009	1	SMT	1	450.0
1	1010	6	SMT	1	36.0
2	1010	6	ANZ	1	48.0
1	1011	5	ANZ	5	99.0
1	1012	8	ANZ	1	840.0
2	1012	9	ANZ	10	200.0
1	1013	5	ANZ	1	19.8
2	1013	6	SMT	1	36.0
3	1013	6	ANZ	1	48.0
4	1013	9	ANZ	2	40.0
1	1014	4	HSK	1	960.0
2	1014	4	HRO	1	480.0
1	1015	1	SMT	1	450.0

stock table

stock_num	manu_code	description	unit_price	unit	unit_descr
1	HRO	baseball gloves	250.00	case	10 gloves/case
1	HSK	baseball gloves	800.00	case	10 gloves/case
1	SMT	baseball gloves	450.00	case	10 gloves/case
2	HRO	baseball	126.00	case	24/case
3	HSK	baseball bat	240.00	case	12/case
4	HSK	football	960.00	case	24/case
4	HRO	football	480.00	case	24/case
5	NRG	tennis racquet	28.00	each	each
5	SMT	tennis racquet	25.00	each	each
5	ANZ	tennis racquet	19.80	each	each
6	SMT	tennis ball	36.00	case	24 cans/case
6	ANZ	tennis ball	48.00	case	24 cans/case
7	HRO	basketball	600.00	case	24/case
8	ANZ	volleyball	840.00	case	24/case
9	ANZ	volleyball net	20.00	each	each

manufact Table

manu_code	manu_name
ANZ	Anza
HSK	Husky
HRO	Hero
NRG	Norge
SMT	Smith

state Table

code	sname	code	sname
AK	Alaska	MT	Montana
AL	Alabama	NB	Nebraska
AR	Arkansas	NC	North Carolina
AZ	Arizona	ND	North Dakota
CA	California	NH	New Hampshire
CT	Connecticut	NJ	New Jersey
CO	Colorado	NM	New Mexico
DE	Delaware	NV	Nevada
FL	Florida	NY	New York
GA	Georgia	OH	Ohio
HI	Hawaii	OK	Oklahoma
IA	Iowa	OR	Oregon
ID	Idaho	PA	Pennsylvania
IL	Illinois	RI	Rhode Island
IN	Indiana	SC	South Carolina
KS	Kansas	SD	South Dakota
KY	Kentucky	TN	Tennessee
LA	Louisiana	TX	Texas
MA	Massachusetts	UT	Utah
MD	Maryland	VA	Virginia
ME	Maine	VT	Vermont
MI	Michigan	WA	Washington
MN	Minnesota	WI	Wisconsin
MO	Missouri	WV	West Virginia
MS	Mississippi	WY	Wyoming

Demonstration Application

The following pages contain the form specifications, INFORMIX-4GL source code modules, and help message source file for the **demo4.4ge** demonstration application. The application is not complete, and some of the functions called by the menus are merely "dead ends."

File Name	Description
custform.per	Form for displaying customer information
orderform.per	Form for entering an order
state_list.per	Form for displaying a list of states
stock_sel.per	Form for displaying a list of stock items
d4_globals.4gl	Module containing global definitions
d4_main.4gl	Module containing MAIN routine
d4_cust.4gl	Module handling the **Customer** option
d4_orders.4gl	Module handling the **Orders** option
d4_stock.4gl	Module handling the **Stock** option
d4_report.4gl	Module handling the **Report** option
d4_demo.4gl	Module handling hidden sample source code option
helpdemo.src	Source file for help messages

custform.per

```
DATABASE stores

SCREEN
{

                              Customer Form

        Number       :[f000        ]
        Owner Name   :[f001             ][f002             ]
        Company      :[f003                  ]
        Address      :[f004                  ]
                      [f005                  ]
        City         :[f006             ] State:[a0] Zipcode:[f007 ]
        Telephone    :[f008                ]

}

TABLES
customer

ATTRIBUTES
f000 = customer.customer_num, NOENTRY;
f001 = customer.fname;
f002 = customer.lname;
f003 = customer.company;
f004 = customer.address1;
f005 = customer.address2;
f006 = customer.city;
a0 = customer.state, UPSHIFT;
f007 = customer.zipcode;
f008 = customer.phone, PICTURE = "###-###-#### XXXXX";
```

orderform.per

```
DATABASE stores

SCREEN
{
 ------------------------------------------------------------------------------
                                ORDER FORM
 ------------------------------------------------------------------------------
 Customer Number:[f000       ] Contact Name:[f001           ][f002           ]
    Company Name:[f003                    ]
         Address:[f004                    ][f005                ]
            City:[f006                 ] State:[a0] Zip Code:[f007  ]
       Telephone:[f008                 ]
 ------------------------------------------------------------------------------
 Order No:[f009       ]    Order Date:[f010       ]    PO Number:[f011       ]
        Shipping Instructions:[f012                                        ]
 ------------------------------------------------------------------------------
 Item No.  Stock No.  Code   Description     Quantity    Price       Total
 [f013   ] [f014   ] [a1 ] [f015           ] [f016   ] [f017      ] [f018      ]
 [f013   ] [f014   ] [a1 ] [f015           ] [f016   ] [f017      ] [f018      ]
 [f013   ] [f014   ] [a1 ] [f015           ] [f016   ] [f017      ] [f018      ]
 [f013   ] [f014   ] [a1 ] [f015           ] [f016   ] [f017      ] [f018      ]
                Running Total including Tax and Shipping Charges:[f019       ]
}

TABLES
customer orders items stock

ATTRIBUTES
f000 = customer.customer_num;
f001 = customer.fname;
f002 = customer.lname;
f003 = customer.company;
f004 = customer.address1;
f005 = customer.address2;
f006 = customer.city;
a0 = customer.state, UPSHIFT;
f007 = customer.zipcode;
f008 = customer.phone, PICTURE = "###-###-#### XXXXX";

f009 = orders.order_num;
f010 = orders.order_date, DEFAULT = TODAY;
f011 = orders.po_num;
f012 = orders.ship_instruct;

f013 = items.item_num, NOENTRY;
f014 = items.stock_num;
a1 = items.manu_code, UPSHIFT;
f015 = stock.description, NOENTRY;
f016 = items.quantity;
f017 = stock.unit_price, NOENTRY;
f018 = items.total_price, NOENTRY;
f019 = formonly.t_price TYPE MONEY;

INSTRUCTIONS
SCREEN RECORD s_items[4](items.item_num, items.stock_num, items.manu_code,
        stock.description, items.quantity, stock.unit_price, items.total_price)
```

state_list.per

```
DATABASE stores

SCREEN
{
   State Selection

[a0] [f000            ]
[a0] [f000            ]
[a0] [f000            ]
[a0] [f000            ]
[a0] [f000            ]
[a0] [f000            ]
[a0] [f000            ]
}

TABLES
state

ATTRIBUTES
a0 = state.code;
f000 = state.sname;

INSTRUCTIONS
DELIMITERS "  "
SCREEN RECORD s_state[7](state.*)
```

stock_sel.per

```
DATABASE stores

SCREEN
{
   [f018][f019][f020          ][f021          ][f022     ][f023          ]
   [f018][f019][f020          ][f021          ][f022     ][f023          ]
   [f018][f019][f020          ][f021          ][f022     ][f023          ]
}

TABLES
stock

ATTRIBUTES
f018 = FORMONLY.stock_num;
f019 = FORMONLY.manu_code;
f020 = FORMONLY.manu_name;
f021 = FORMONLY.description;
f022 = FORMONLY.unit_price;
f023 = FORMONLY.unit_descr;

INSTRUCTIONS
DELIMITERS "  "
SCREEN RECORD s_stock[3] (FORMONLY.stock_num THRU FORMONLY.unit_descr)
```

d4__globals.4gl

```
DATABASE stores

GLOBALS
   DEFINE
      p_customer RECORD LIKE customer.*,
      p_orders RECORD
            order_num LIKE orders.order_num,
            order_date LIKE orders.order_date,
            po_num LIKE orders.po_num,
            ship_instruct LIKE orders.ship_instruct
         END RECORD,
      p_items ARRAY[10] OF RECORD
            item_num LIKE items.item_num,
            stock_num LIKE items.stock_num,
            manu_code LIKE items.manu_code,
            description LIKE stock.description,
            quantity LIKE items.quantity,
            unit_price LIKE stock.unit_price,
            total_price LIKE items.total_price
         END RECORD,
      p_stock ARRAY[30] OF RECORD
            stock_num LIKE stock.stock_num,
            manu_code LIKE manufact.manu_code,
            manu_name LIKE manufact.manu_name,
            description LIKE stock.description,
            unit_price LIKE stock.unit_price,
            unit_descr LIKE stock.unit_descr
         END RECORD,
      p_state ARRAY[50] OF RECORD LIKE state.*,
      state_cnt, stock_cnt INTEGER,
      print_option CHAR(1)
END GLOBALS
```

d4__main.4gl

```
GLOBALS
   "d4_globals.4gl"

MAIN

   DEFER INTERRUPT
   OPTIONS
   HELP FILE "helpdemo"
   LET print_option = "s"
   CALL get_states()
   CALL get_stocks()

   CALL ring_menu()
   MENU "MAIN"
      COMMAND "Customer" "Enter and maintain customer data" HELP 101
         CALL customer()
         CALL ring_menu()
      COMMAND "Orders" "Enter and maintain orders" HELP 102
         CALL orders()
         CALL ring_menu()
      COMMAND "Stock" "Enter and maintain stock list" HELP 103
         CALL stock()
         CALL ring_menu()
      COMMAND "Reports" "Print reports and mailing labels" HELP 104
         CALL reports()
         CALL ring_menu()
      COMMAND key("!")
         CALL bang()
         CALL ring_menu()
         NEXT OPTION "Customer"
      COMMAND key("X")
         CALL demo()
         CALL ring_menu()
         NEXT OPTION "Customer"
      COMMAND "Exit" "Exit program and return to operating system" HELP 105
         CLEAR SCREEN
         EXIT PROGRAM
   END MENU
END MAIN


FUNCTION bang()
   DEFINE cmd CHAR(80),
      x CHAR(1)

   CALL clear_menu()
   LET x = "!"
   WHILE x = "!"
      PROMPT "!" FOR cmd
      RUN cmd
      PROMPT "Type RETURN to continue." FOR CHAR x
   END WHILE
END FUNCTION
```

```
FUNCTION mess(str, mrow)
   DEFINE str CHAR(80),
      mrow SMALLINT

   DISPLAY " ", str CLIPPED AT mrow,1
   SLEEP 3
   DISPLAY "" AT mrow,1
END FUNCTION


FUNCTION ring_menu()

   DISPLAY "---------------------------------------- ",
      "Type Control-W for MENU HELP -------" AT 4,2 ATTRIBUTE(MAGENTA)
END FUNCTION


FUNCTION clear_menu()

   DISPLAY "" AT 1,1
   DISPLAY "" AT 2,1
END FUNCTION


FUNCTION get_states()

   DECLARE c_state CURSOR FOR
      SELECT * FROM state
      ORDER BY sname
   LET state_cnt = 1
   FOREACH c_state INTO p_state[state_cnt].*
      LET state_cnt = state_cnt + 1
      IF state_cnt > 50 THEN
         EXIT FOREACH
      END IF
   END FOREACH
   LET state_cnt = state_cnt - 1
END FUNCTION


FUNCTION get_stocks()

   DECLARE stock_list CURSOR FOR
      SELECT stock_num, manufact.manu_code,
         manu_name, description, unit_price, unit_descr
      FROM stock, manufact
      WHERE stock.manu_code = manufact.manu_code
      ORDER BY stock_num
   LET stock_cnt = 1
   FOREACH stock_list INTO p_stock[stock_cnt].*
      LET stock_cnt = stock_cnt + 1
      IF stock_cnt > 30 THEN
         EXIT FOREACH
      END IF
   END FOREACH
   LET stock_cnt = stock_cnt - 1
END FUNCTION
```

d4__cust.4gl

```
GLOBALS
    "d4_globals.4gl"


FUNCTION customer()

   OPTIONS
      FORM LINE 7
   OPEN FORM customer FROM "custform"
   DISPLAY FORM customer
      ATTRIBUTE(MAGENTA)
   CALL ring_menu()
   CALL fgl_drawbox(3,30,3,43)
   CALL fgl_drawbox(3,61,8,7)
   CALL fgl_drawbox(11,61,8,7)
   LET p_customer.customer_num = NULL
   MENU "CUSTOMER"
      COMMAND "One-add" "Add a new customer to the database" HELP 201
         CALL add_customer(FALSE)
      COMMAND "Many-add" "Add several new customer to database" HELP 202
         CALL add_customer(TRUE)
      COMMAND "Find-cust" "Look up specific customer" HELP 203
         CALL query_customer(23)
         IF p_customer.customer_num IS NOT NULL THEN
            NEXT OPTION "Update-cust"
         END IF
      COMMAND "Update-cust" "Modify current customer information" HELP 204
         CALL update_customer()
         NEXT OPTION "Find-cust"
      COMMAND "Delete-cust" "Remove a customer from database" HELP 205
         CALL delete_customer()
         NEXT OPTION "Find-cust"
      COMMAND "Exit" "Return to MAIN Menu" HELP 206
         CLEAR SCREEN
         EXIT MENU
   END MENU
   OPTIONS
      FORM LINE 3
END FUNCTION


FUNCTION add_customer(repeat)
   DEFINE repeat INTEGER

   CALL clear_menu()
   MESSAGE "Press F1 or CTRL-F for field help; ",
      "F2 or CTRL-Z to return to menu"
   IF repeat THEN
      WHILE input_cust()
         ERROR "Customer data entered" ATTRIBUTE (GREEN)
      END WHILE
      CALL mess("Multiple insert completed - current screen values ignored", 23)
   ELSE
      IF input_cust() THEN
         ERROR "Customer data entered" ATTRIBUTE (GREEN)
      ELSE
         CLEAR FORM
         LET p_customer.customer_num = NULL
         ERROR "Customer addition aborted" ATTRIBUTE (RED, REVERSE)
      END IF
   END IF
END FUNCTION
```

```
FUNCTION input_cust()

    DISPLAY "Press ESC to enter new customer data" AT 1,1
    INPUT BY NAME p_customer.*
       AFTER FIELD state
          CALL statehelp()
          DISPLAY "Press ESC to enter new customer data", "" AT 1,1
       ON KEY (F1, CONTROL-F)
          CALL customer_help()
       ON KEY (F2, CONTROL-Z)
          LET int_flag = TRUE
          EXIT INPUT
    END INPUT
    IF int_flag THEN
       LET int_flag = FALSE
       RETURN(FALSE)
    END IF
    LET p_customer.customer_num = 0
    INSERT INTO customer VALUES (p_customer.*)
    LET p_customer.customer_num = SQLCA.SQLERRD[2]
    DISPLAY BY NAME p_customer.customer_num ATTRIBUTE(MAGENTA)
    RETURN(TRUE)
END FUNCTION


FUNCTION query_customer(mrow)
    DEFINE where_part CHAR(200),
       query_text CHAR(250),
       answer CHAR(1),
       mrow, chosen, exist SMALLINT

    CLEAR FORM
    CALL clear_menu()

    MESSAGE "Enter criteria for selection"
    CONSTRUCT where_part ON customer.* FROM customer.*
    MESSAGE ""
    IF int_flag THEN
       LET int_flag = FALSE
       CLEAR FORM
       ERROR "Customer query aborted" ATTRIBUTE(RED, REVERSE)
       LET p_customer.customer_num = NULL
       RETURN (p_customer.customer_num)
    END IF
    LET query_text = "select * from customer where ", where_part CLIPPED,
       " order by lname"
    PREPARE statement_1 FROM query_text
    DECLARE customer_set SCROLL CURSOR FOR statement_1

    OPEN customer_set
    FETCH FIRST customer_set INTO p_customer.*
```

```
    IF status = NOTFOUND THEN
       LET exist = FALSE
    ELSE
       LET exist = TRUE
       DISPLAY BY NAME p_customer.*
       MENU "BROWSE"
          COMMAND "Next" "View the next customer in the list"
             FETCH NEXT customer_set INTO p_customer.*
             IF status = NOTFOUND THEN
                ERROR "No more customers in this direction"
                   ATTRIBUTE(RED, REVERSE)
                FETCH LAST customer_set INTO p_customer.*
             END IF
             DISPLAY BY NAME p_customer.*  ATTRIBUTE(CYAN)
          COMMAND "Previous" "View the previous customer in the list"
             FETCH PREVIOUS customer_set INTO p_customer.*
             IF status = NOTFOUND THEN
                ERROR "No more customers in this direction"
                   ATTRIBUTE(RED, REVERSE)
                FETCH FIRST customer_set INTO p_customer.*
             END IF
             DISPLAY BY NAME p_customer.*  ATTRIBUTE(CYAN)
          COMMAND "First" "View the first customer in the list"
             FETCH FIRST customer_set INTO p_customer.*
             DISPLAY BY NAME p_customer.*  ATTRIBUTE(CYAN)
          COMMAND "Last" "View the last customer in the list"
             FETCH LAST customer_set INTO p_customer.*
             DISPLAY BY NAME p_customer.*  ATTRIBUTE(CYAN)
          COMMAND "Select" "Exit BROWSE selecting the current customer"
             LET chosen = TRUE
             EXIT MENU
          COMMAND "Quit" "Quit BROWSE without selecting a customer"
             LET chosen = FALSE
             EXIT MENU
       END MENU
    END IF
    CLOSE customer_set

    IF NOT exist THEN
       CLEAR FORM
       CALL mess("No customer satisfies query", mrow)
       LET p_customer.customer_num = NULL
       RETURN (FALSE)
    END IF
    IF NOT chosen THEN
       CLEAR FORM
       LET p_customer.customer_num = NULL
       CALL mess("No selection made", mrow)
       RETURN (FALSE)
    END IF
    RETURN (TRUE)
END FUNCTION
```

```
FUNCTION update_customer()

    CALL clear_menu()
    IF p_customer.customer_num IS NULL THEN
       CALL mess("No customer has been selected; use the Find-cust option",23)
       RETURN
    END IF
    MESSAGE "Press F1 or CTRL-F for field-level help"
    DISPLAY "Press ESC to update customer data; DEL to abort" AT 1,1
    INPUT BY NAME p_customer.* WITHOUT DEFAULTS
       AFTER FIELD state
          CALL statehelp()
          DISPLAY "Press ESC to update customer data; DEL to abort", "" AT 1,1
       ON KEY (F1, CONTROL-F)
          CALL customer_help()
    END INPUT
    IF NOT int_flag THEN
       UPDATE customer SET customer.* = p_customer.*
          WHERE customer_num = p_customer.customer_num
       CALL mess("Customer data modified", 23)
    ELSE
       LET int_flag = FALSE
       SELECT * INTO p_customer.* FROM customer
          WHERE customer_num = p_customer.customer_num
       DISPLAY BY NAME p_customer.*
       ERROR "Customer update aborted" ATTRIBUTE (RED, REVERSE)
    END IF
END FUNCTION


FUNCTION delete_customer()
    DEFINE answer CHAR(1),
       num_orders INTEGER

    CALL clear_menu()
    IF p_customer.customer_num IS NULL THEN
       ERROR "No customer has been selected; use the Find-customer option"
          ATTRIBUTE (RED, REVERSE)
       RETURN
    END IF

    SELECT COUNT(*) INTO num_orders
       FROM orders
       WHERE customer_num = p_customer.customer_num
    IF num_orders THEN
       ERROR "This customer has active orders and can not be removed"
          ATTRIBUTE (RED, REVERSE)
       RETURN
    END IF

    PROMPT " Are you sure you want to delete this customer row? "
       FOR CHAR answer
    IF answer MATCHES "[yY]" THEN
       DELETE FROM customer
          WHERE customer_num = p_customer.customer_num
       CLEAR FORM
       CALL mess("Customer entry deleted", 23)
       LET p_customer.customer_num = NULL
    ELSE
       ERROR "Deletion aborted" ATTRIBUTE (RED, REVERSE)
    END IF
END FUNCTION
```

```
FUNCTION customer_help()
   CASE
      WHEN infield(customer_num) CALL showhelp(1001)
      WHEN infield(fname) CALL showhelp(1002)
      WHEN infield(lname) CALL showhelp(1003)
      WHEN infield(company) CALL showhelp(1004)
      WHEN infield(address1) CALL showhelp(1005)
      WHEN infield(address2) CALL showhelp(1006)
      WHEN infield(city) CALL showhelp(1007)
      WHEN infield(state) CALL showhelp(1008)
      WHEN infield(zipcode) CALL showhelp(1009)
      WHEN infield(phone) CALL showhelp(1010)
   END CASE
END FUNCTION


FUNCTION statehelp()
   DEFINE idx INTEGER

   SELECT COUNT(*) INTO idx
      FROM state
      WHERE code = p_customer.state
   IF idx = 1 THEN
      RETURN
   END IF

   DISPLAY "Move cursor using F3, F4, and arrow keys; press ESC to select state"
      AT 1,1
   OPEN WINDOW w_state AT 8,37
      WITH FORM "state_list"
      ATTRIBUTE (BORDER, RED, FORM LINE 2)

   CALL set_count(state_cnt)
   DISPLAY ARRAY p_state TO s_state.*
   LET idx = arr_curr()

   CLOSE WINDOW w_state
   LET p_customer.state = p_state[idx].code
   DISPLAY BY NAME p_customer.state ATTRIBUTE (MAGENTA)
   RETURN
END FUNCTION
```

d4__orders.4gl

```
GLOBALS
   "d4_globals.4gl"


FUNCTION orders()

   OPEN FORM order_form FROM "orderform"
   DISPLAY FORM order_form
      ATTRIBUTE(MAGENTA)
   MENU "ORDERS"
      COMMAND "Add-order" "Enter new order to database and print invoice"
            HELP 301
         CALL add_order()
      COMMAND "Update-order" "Enter shipping or payment data" HELP 302
         CALL update_order()
      COMMAND "Find-order" "Look up and display orders" HELP 303
         CALL get_order()
      COMMAND "Delete-order" "Remove an order from the database" HELP 304
         CALL delete_order()
      COMMAND "Exit" "Return to MAIN Menu" HELP 305
         CLEAR SCREEN
         EXIT MENU
   END MENU
END FUNCTION


FUNCTION add_order()
   DEFINE pa_curr, s_curr, num_stocks INTEGER,
      file_name CHAR(20),
      query_stat INTEGER

   CALL clear_menu()
   LET query_stat = query_customer(2)
   IF query_stat IS NULL THEN
      RETURN
   END IF
   IF NOT query_stat THEN
      OPEN WINDOW cust_w AT 3,5
         WITH 19 ROWS, 72 COLUMNS
         ATTRIBUTE(BORDER, YELLOW)
      OPEN FORM o_cust FROM "custform"
      DISPLAY FORM o_cust
         ATTRIBUTE(MAGENTA)
      CALL fgl_drawbox(3,61,4,7)
      CALL fgl_drawbox(11,61,4,7)
      CALL add_customer(FALSE)
      CLOSE FORM o_cust
      CLOSE WINDOW cust_w
      IF p_customer.customer_num IS NULL THEN
         RETURN
      ELSE
         DISPLAY by name p_customer.*
      END IF
   END IF

   MESSAGE "Enter the order date, PO number and shipping instructions."
   INPUT BY NAME p_orders.order_date, p_orders.po_num, p_orders.ship_instruct
   IF int_flag THEN
      LET int_flag = FALSE
      CLEAR FORM
      ERROR "Order input aborted" ATTRIBUTE (RED, REVERSE)
      RETURN
   END IF
```

```
INPUT ARRAY p_items FROM s_items.* HELP 311
   BEFORE FIELD stock_num
      MESSAGE "Press ESC to write order"
      DISPLAY "Enter a stock number or press CTRL-B to scan stock list"
         AT 1,1
   BEFORE FIELD manu_code
      MESSAGE "Enter the code for a manufacturer"
   BEFORE FIELD quantity
      DISPLAY "" AT 1,1
      MESSAGE "Enter the item quantity"
   ON KEY (CONTROL-B)
      IF INFIELD(stock_num) OR INFIELD(manu_code) THEN
         LET pa_curr = arr_curr()
         LET s_curr = scr_line()
         CALL get_stock() RETURNING
            p_items[pa_curr].stock_num, p_items[pa_curr].manu_code,
            p_items[pa_curr].description, p_items[pa_curr].unit_price
         DISPLAY p_items[pa_curr].stock_num TO s_items[s_curr].stock_num
         DISPLAY p_items[pa_curr].manu_code TO s_items[s_curr].manu_code
         DISPLAY p_items[pa_curr].description TO s_items[s_curr].description
         DISPLAY p_items[pa_curr].unit_price TO s_items[s_curr].unit_price
         NEXT FIELD quantity
      END IF
   AFTER FIELD stock_num, manu_code
      LET pa_curr = arr_curr()
      IF p_items[pa_curr].stock_num IS NOT NULL
         AND p_items[pa_curr].manu_code IS NOT NULL
      THEN
         CALL get_item()
      END IF
   AFTER FIELD quantity
      MESSAGE ""
      LET pa_curr = arr_curr()
      IF p_items[pa_curr].unit_price IS NOT NULL
         AND p_items[pa_curr].quantity IS NOT NULL
      THEN
         CALL item_total()
      ELSE
         ERROR
         "A valid stock code, manufacturer, and quantity must all be entered"
            ATTRIBUTE (RED, REVERSE)
         NEXT FIELD stock_num
      END IF
   AFTER INSERT, DELETE
      CALL renum_items()
      CALL order_total()
   AFTER ROW
      CALL order_total()
END INPUT
IF int_flag THEN
   LET int_flag = FALSE
   CLEAR FORM
   ERROR "Order input aborted" ATTRIBUTE (RED, REVERSE)
   RETURN
END IF

WHENEVER ERROR CONTINUE
BEGIN WORK
INSERT INTO orders (order_num, order_date, customer_num,
      ship_instruct, po_num)
   VALUES (0, p_orders.order_date, p_customer.customer_num,
      p_orders.ship_instruct, p_orders.po_num)
IF status < 0 THEN
   ROLLBACK WORK
   ERROR "Unable to complete update of orders table"
      ATTRIBUTE(RED, REVERSE, BLINK)
   RETURN
END IF
LET p_orders.order_num = SQLCA.SQLERRD[2]
DISPLAY BY NAME p_orders.order_num
IF NOT insert_items() THEN
   ROLLBACK WORK
   ERROR "Unable to insert items" ATTRIBUTE(RED, REVERSE, BLINK)
   RETURN
END IF
```

```
    COMMIT WORK
    WHENEVER ERROR STOP
    CALL mess("Order added", 23)
    LET file_name = "inv", p_orders.order_num USING "<<<<&",".out"
    CALL invoice(file_name)
    CLEAR FORM
END FUNCTION

FUNCTION update_order()

    ERROR "This option has not been implemented" ATTRIBUTE (RED)
END FUNCTION


FUNCTION delete_order()

    ERROR "This option has not been implemented" ATTRIBUTE (RED)
END FUNCTION


FUNCTION order_total()
    DEFINE order_total MONEY(8),
        i INTEGER

    LET order_total = 0.00
    FOR i = 1 TO ARR_COUNT()
        IF p_items[i].total_price IS NOT NULL THEN
            LET order_total = order_total + p_items[i].total_price
        END IF
    END FOR
    LET order_total = 1.1 * order_total
    DISPLAY order_total TO t_price
        ATTRIBUTE(GREEN)
END FUNCTION


FUNCTION item_total()
    DEFINE pa_curr, sc_curr INTEGER

    LET pa_curr = arr_curr()
    LET sc_curr = scr_line()
    LET p_items[pa_curr].total_price =
        p_items[pa_curr].quantity * p_items[pa_curr].unit_price
    DISPLAY p_items[pa_curr].total_price TO s_items[sc_curr].total_price
END FUNCTION


FUNCTION renum_items()
    DEFINE pa_curr, pa_total, sc_curr, sc_total, k INTEGER

    LET pa_curr = arr_curr()
    LET pa_total = arr_count()
    LET sc_curr = scr_line()
    LET sc_total = 4
    FOR k = pa_curr TO pa_total
        LET p_items[k].item_num = k
        IF sc_curr <= sc_total THEN
            DISPLAY k TO s_items[sc_curr].item_num
            LET sc_curr = sc_curr + 1
        END IF
    END FOR
END FUNCTION
```

```
FUNCTION insert_items()
   DEFINE idx INTEGER

   FOR idx = 1 TO arr_count()
      IF p_items[idx].quantity != 0 THEN
         INSERT INTO items
            VALUES (p_items[idx].item_num, p_orders.order_num,
               p_items[idx].stock_num, p_items[idx].manu_code,
               p_items[idx].quantity, p_items[idx].total_price)
         IF status < 0 THEN
            RETURN (FALSE)
         END IF
      END IF
   END FOR
   RETURN (TRUE)
END FUNCTION


FUNCTION get_stock()
   DEFINE idx integer

   OPEN WINDOW stock_w AT 7, 3
      WITH FORM "stock_sel"
      ATTRIBUTE(BORDER, YELLOW)
   CALL set_count(stock_cnt)
   DISPLAY
      " Use cursor using F3, F4, and arrow keys; press ESC to select a stock item"
      AT 1,1
   DISPLAY ARRAY p_stock TO s_stock.*
   LET idx = arr_curr()
   CLOSE WINDOW stock_w
   RETURN p_stock[idx].stock_num, p_stock[idx].manu_code,
      p_stock[idx].description, p_stock[idx].unit_price
END FUNCTION


FUNCTION get_order()
   DEFINE idx, exist, chosen INTEGER,
      answer CHAR(1)

   CALL clear_menu()
   CLEAR FORM
   IF NOT query_customer(2) THEN
      RETURN
   END IF
   DECLARE order_list CURSOR FOR
      SELECT order_num, order_date, po_num, ship_instruct
         FROM orders
         WHERE customer_num = p_customer.customer_num
   LET exist = FALSE
   LET chosen = FALSE
   FOREACH order_list INTO p_orders.*
      LET exist = TRUE
      CLEAR orders.*
      FOR idx = 1 TO 4
         CLEAR s_items[idx].*
      END FOR
      DISPLAY p_orders.* TO orders.*
      DECLARE item_list CURSOR FOR
         SELECT item_num, items.stock_num, items.manu_code,
                description, quantity, unit_price, total_price
            FROM items, stock
            WHERE order_num = p_orders.order_num
               AND items.stock_num = stock.stock_num
               AND items.manu_code = stock.manu_code
            ORDER BY item_num
      LET idx = 1
```

```
        FOREACH item_list INTO p_items[idx].*
            LET idx = idx + 1
            IF idx > 10 THEN
                ERROR "More than 10 items; only 10 items displayed"
                    ATTRIBUTE (RED, REVERSE)
                EXIT FOREACH
            END IF
        END FOREACH
        CALL set_count(idx - 1)
        CALL order_total()
        MESSAGE "Press ESC when you finish viewing the items"
        DISPLAY ARRAY p_items TO s_items.*
            ATTRIBUTE(CYAN)
        MESSAGE ""
        IF int_flag THEN
            LET int_flag = FALSE
            EXIT FOREACH
        END IF
        PROMPT " Enter 'y' to select this order ",
            "or RETURN to view next order: " FOR CHAR answer
        IF answer MATCHES "[yY]" THEN
            LET chosen = TRUE
            EXIT FOREACH
        END IF
    END FOREACH

    IF NOT exist THEN
        ERROR "No orders found for this customer" ATTRIBUTE (RED)
    ELSE
        IF NOT chosen THEN
            CLEAR FORM
            ERROR "No order selected for this customer" ATTRIBUTE (RED)
        END IF
    END IF
END FUNCTION

FUNCTION get_item()
    DEFINE pa_curr, sc_curr INTEGER

    LET pa_curr = arr_curr()
    LET sc_curr = scr_line()
    SELECT description, unit_price
        INTO p_items[pa_curr].description,
            p_items[pa_curr].unit_price
        FROM stock
        WHERE stock.stock_num = p_items[pa_curr].stock_num
            AND stock.manu_code = p_items[pa_curr].manu_code
    IF status THEN
        LET p_items[pa_curr].description = NULL
        LET p_items[pa_curr].unit_price = NULL
    END IF
    DISPLAY p_items[pa_curr].description, p_items[pa_curr].unit_price
        TO s_items[sc_curr].description, s_items[sc_curr].unit_price
    IF p_items[pa_curr].quantity IS NOT NULL THEN
        CALL item_total()
    END IF
END FUNCTION
```

```
FUNCTION invoice(file_name)
   DEFINE x_invoice RECORD
          order_num              LIKE orders.order_num,
          order_date             LIKE orders.order_date,
          ship_instruct          LIKE orders.ship_instruct,
          backlog                LIKE orders.backlog,
          po_num                 LIKE orders.po_num,
          ship_date              LIKE orders.ship_date,
          ship_weight            LIKE orders.ship_weight,
          ship_charge            LIKE orders.ship_charge,
          item_num               LIKE items.item_num,
          stock_num              LIKE items.stock_num,
          manu_code              LIKE items.manu_code,
          quantity               LIKE items.quantity,
          total_price            LIKE items.total_price,
          description            LIKE stock.description,
          unit_price             LIKE stock.unit_price,
          unit                   LIKE stock.unit,
          unit_descr             LIKE stock.unit_descr,
          manu_name              LIKE manufact.manu_name
       END RECORD,
       file_name CHAR(20),
       msg CHAR(40)

   DECLARE invoice_data CURSOR FOR
      SELECT o.order_num,order_date,ship_instruct,backlog,po_num,ship_date,
          ship_weight,ship_charge,
          item_num,i.stock_num,i.manu_code,quantity,total_price,
          s.description,unit_price,unit,unit_descr,
          manu_name
      FROM orders o,items i,stock s,manufact m
      WHERE
          ((o.order_num=p_orders.order_num) AND
          (i.order_num=p_orders.order_num) AND
          (i.stock_num=s.stock_num AND
          i.manu_code=s.manu_code) AND
          (i.manu_code=m.manu_code))
      ORDER BY 9
   CASE (print_option)
      WHEN "f"
          START REPORT r_invoice TO file_name
          CALL clear_menu()
          MESSAGE "Writing invoice -- please wait"
      WHEN "p"
           START REPORT r_invoice TO PRINTER
          CALL clear_menu()
          MESSAGE "Writing invoice -- please wait"
      WHEN "s"
           START REPORT r_invoice
   END CASE
   FOREACH invoice_data INTO x_invoice.*
      OUTPUT TO REPORT r_invoice (p_customer.*, x_invoice.*)
   END FOREACH
   FINISH REPORT r_invoice
   IF print_option = "f" THEN
      LET msg = "Invoice writen to file ", file_name CLIPPED
      CALL mess(msg, 23)
   END IF
END FUNCTION
```

```
REPORT r_invoice (c, x)
   DEFINE c RECORD LIKE customer.*,
      x RECORD
         order_num       LIKE orders.order_num,
         order_date      LIKE orders.order_date,
         ship_instruct   LIKE orders.ship_instruct,
         backlog         LIKE orders.backlog,
         po_num          LIKE orders.po_num,
         ship_date       LIKE orders.ship_date,
         ship_weight     LIKE orders.ship_weight,
         ship_charge     LIKE orders.ship_charge,
         item_num        LIKE items.item_num,
         stock_num       LIKE items.stock_num,
         manu_code       LIKE items.manu_code,
         quantity        LIKE items.quantity,
         total_price     LIKE items.total_price,
         description     LIKE stock.description,
         unit_price      LIKE stock.unit_price,
         unit            LIKE stock.unit,
         unit_descr      LIKE stock.unit_descr,
         manu_name       LIKE manufact.manu_name
      END RECORD,
      sales_tax, calc_total MONEY(8,2)

OUTPUT
   LEFT MARGIN 0
   RIGHT MARGIN 0
   TOP MARGIN 1
   BOTTOM MARGIN 1
   PAGE LENGTH 48

FORMAT
   BEFORE GROUP OF x.order_num
      SKIP TO TOP OF PAGE
      SKIP 1 LINE
      PRINT 10 SPACES,
         "   W E S T   C O A S T   W H O L E S A L E R S ,   I N C ."
      PRINT 30 SPACES," 1400 Hanbonon Drive"
      PRINT 30 SPACES,"Menlo Park, CA  94025"
      SKIP 1 LINES
      PRINT "Bill To:", COLUMN 10,c.fname CLIPPED, " ", c.lname CLIPPED;
      PRINT COLUMN 56,"Invoice No.         ",x.order_num USING "&&&&&"
      PRINT COLUMN 10,c.company
      PRINT COLUMN 10,c.address1 CLIPPED;
      PRINT COLUMN 56,"Invoice Date: ", x.order_date
      PRINT COLUMN 10,c.address2 CLIPPED;
      PRINT COLUMN 56,"Customer No.        ", c.customer_num USING "#####&"
      PRINT COLUMN 10,c.city CLIPPED,", ",c.state CLIPPED,"   ",
         c.zipcode CLIPPED;
      PRINT COLUMN 56,"PO No. ",x.po_num
      PRINT COLUMN 10,c.phone CLIPPED;
      PRINT COLUMN 56,"Backlog Status: ",x.backlog
      SKIP 1 LINES
      PRINT COLUMN 10,"Shipping Instructions: ", x.ship_instruct
      PRINT COLUMN 10,"Ship Date: ",x.ship_date USING "ddd. mmm dd, yyyy";
      PRINT "   Weight: ", x.ship_weight USING "######.&&"
      SKIP 1 LINES
      PRINT "---------------------------------------";
      PRINT "---------------------------------------"
      PRINT "   Stock                                Unit         ";
      PRINT "                          Item "
      PRINT " #  Num Man    Description      Qty   Cost   Unit ";
      PRINT " Unit Description      Total"
      SKIP 1 LINES
      LET calc_total = 0.00
```

```
    ON EVERY ROW
        PRINT x.item_num USING "#&","  ",
            x.stock_num USING "&&", "  ",x.manu_code;
        PRINT "  ",x.description,"  ",x.quantity USING "###&", "  ";
        PRINT x.unit_price USING "$$$&.&&","  ",x.unit, "  ",x.unit_descr,"  ";
        PRINT x.total_price USING "$$$$$$$&.&&"
        LET calc_total = calc_total + x.total_price

    AFTER GROUP OF x.order_num
        SKIP 1 LINES
        PRINT "----------------------------------------";
        PRINT "----------------------------------------"
        PRINT COLUMN 50, "         Sub-total: ",calc_total USING "$$$$$$$&.&&"
        LET sales_tax = 0.065 * calc_total
        LET x.ship_charge = 0.035 * calc_total
        PRINT COLUMN 45, "Shipping Charge (3.5%): ",
            x.ship_charge USING "$$$$$$$&.&&"
        PRINT COLUMN 50, " Sales Tax (6.5%): ",sales_tax USING "$$$$$$$&.&&"
        PRINT COLUMN 50, "                   ----------"
        LET calc_total = calc_total + x.ship_charge + sales_tax
        PRINT COLUMN 50, "            Total: ",calc_total USING "$$$$$$$&.&&"
        IF print_option = "s" THEN
            PAUSE "Type RETURN to continue"
        END IF
END REPORT
```

d4__stock.4gl

```
GLOBALS
   "d4_globals.4gl"


FUNCTION stock()
   MENU "STOCK"
      COMMAND "Add-stock" "Add new stock items to database" HELP 401
         CALL add_stock()
      COMMAND "Find-stock" "Look up specific stock item" HELP 402
         CALL query_stock()
      COMMAND "Update-stock" "Modify current stock information" HELP 403
         CALL update_stock()
      COMMAND "Delete-stock" "Remove a stock item from database" HELP 404
         CALL delete_stock()
      COMMAND "Exit" "Return to MAIN Menu" HELP 405
         CLEAR SCREEN
         EXIT MENU
   END MENU
END FUNCTION


FUNCTION add_stock()
   ERROR "This option has not been implemented" ATTRIBUTE (RED)
END FUNCTION


FUNCTION query_stock()
   ERROR "This option has not been implemented" ATTRIBUTE (RED)
END FUNCTION


FUNCTION update_stock()
   ERROR "This option has not been implemented" ATTRIBUTE (RED)
END FUNCTION


FUNCTION delete_stock()
   ERROR "This option has not been implemented" ATTRIBUTE (RED)
END FUNCTION
```

d4__report.4gl

```
GLOBALS
   "d4_globals.4gl"


FUNCTION reports()
   CALL ring_menu()
   MENU "REPORTS"
      COMMAND "Labels" "Print mailing labels from customer list"
            HELP 501
         CALL print_labels()
         CLEAR SCREEN
         CALL ring_menu()
      COMMAND "Accounts-receivable" "Print current unpaid orders" HELP 502
         CALL print_ar()
         CLEAR SCREEN
         CALL ring_menu()
      COMMAND "Backlog" "Print backlogged orders" HELP 503
         CALL print_backlog()
         CLEAR SCREEN
         CALL ring_menu()
      COMMAND "Stock-list" "Print stock available" HELP 504
         CALL print_stock()
         CLEAR SCREEN
         CALL ring_menu()
      COMMAND "Options" "Change the report output options" HELP 505
         CALL update_options()
         CALL ring_menu()
      COMMAND "Exit" "Return to MAIN Menu" HELP 506
         CLEAR SCREEN
         EXIT MENU
   END MENU
END FUNCTION
```

```
FUNCTION print_labels()
   DEFINE where_part CHAR(200),
      query_text CHAR(250),
      msg CHAR(50),
      file_name CHAR(20)

   OPTIONS
      FORM LINE 7
   OPEN FORM customer FROM "custform"
   DISPLAY FORM customer
      ATTRIBUTE(MAGENTA)
   CALL fgl_drawbox(3,30,3,43)
   CALL fgl_drawbox(3,61,8,7)
   CALL fgl_drawbox(11,61,8,7)
   CALL clear_menu()
   DISPLAY "CUSTOMER LABELS:" AT 1,1
   MESSAGE "Use query-by-example to select customer list"
   CONSTRUCT BY NAME where_part ON customer.*
   IF int_flag THEN
      LET int_flag = FALSE
      ERROR "Label print request aborted"
      RETURN
   END IF
   MESSAGE ""
   LET query_text = "select * from customer where ", where_part CLIPPED,
      " order by zipcode"
   PREPARE label_st FROM query_text
   DECLARE label_list CURSOR FOR label_st
   CASE (print_option)
      WHEN "f"
         PROMPT " Enter file name for labels >" FOR file_name
         IF file_name IS NULL THEN
            LET file_name = "labels.out"
         END IF
         MESSAGE "Printing mailing labels to ", file_name CLIPPED,
            " -- Please wait"
         START REPORT labels_report TO file_name
      WHEN "p"
         MESSAGE "Printing mailing labels -- Please wait"
         START REPORT labels_report TO PRINTER
      WHEN "s"
         START REPORT labels_report
         CLEAR SCREEN
   END CASE
   FOREACH label_list INTO p_customer.*
      OUTPUT TO REPORT labels_report (p_customer.*)
      IF int_flag THEN
         LET int_flag = FALSE
         EXIT FOREACH
      END IF
   END FOREACH
   FINISH REPORT labels_report
   IF int_flag THEN
      LET int_flag = FALSE
      ERROR "Label printing aborted" ATTRIBUTE (RED, REVERSE)
      RETURN
   END IF
   IF print_option = "f" THEN
      LET msg = "Labels printed to ", file_name CLIPPED
      CALL mess(msg, 23)
   END IF
   CLOSE FORM customer
   OPTIONS
      FORM LINE 3
END FUNCTION
```

```
REPORT labels_report (rl)
   DEFINE rl RECORD LIKE customer.*

OUTPUT
   TOP MARGIN 0
   BOTTOM MARGIN 0
   PAGE LENGTH 6

FORMAT
   ON EVERY ROW
   SKIP TO TOP OF PAGE
   PRINT rl.fname CLIPPED, 1 SPACE, rl.lname
   PRINT rl.company
   PRINT rl.address1
   IF rl.address2 IS NOT NULL THEN
      PRINT rl.address2
   END IF
   PRINT rl.city CLIPPED, ", ", rl.state, 2 SPACES, rl.zipcode
   IF print_option = "s" THEN
      PAUSE "Type RETURN to continue"
   END IF
END REPORT


FUNCTION print_ar()
   DEFINE r RECORD
          customer_num      LIKE customer.customer_num,
          fname             LIKE customer.fname,
          lname             LIKE customer.lname,
          company           LIKE customer.company,
          order_num         LIKE orders.order_num,
          order_date        LIKE orders.order_date,
          ship_date         LIKE orders.ship_date,
          paid_date         LIKE orders.paid_date,
          total_price       LIKE items.total_price
      END RECORD,
   file_name CHAR(20),
   msg CHAR(50)

   DECLARE ar_list CURSOR FOR
   SELECT customer.customer_num,fname,lname,company,
          orders.order_num,order_date,ship_date,paid_date,
          total_price
      FROM customer,orders,items
      WHERE customer.customer_num=orders.customer_num AND
          paid_date IS NULL AND
          orders.order_num=items.order_num
      ORDER BY 1,5

   CALL clear_menu()
   CASE (print_option)
      WHEN "f"
          PROMPT " Enter file name for AR Report >" FOR file_name
          IF file_name IS NULL THEN
             LET file_name = "ar.out"
          END IF
          MESSAGE "Printing AR REPORT to ", file_name CLIPPED,
             " -- Please wait"
          START REPORT ar_report TO file_name
      WHEN "p"
          MESSAGE "Printing AR REPORT -- Please wait"
          START REPORT ar_report TO PRINTER
      WHEN "s"
          START REPORT ar_report
          CLEAR SCREEN
          MESSAGE "Printing AR REPORT -- Please wait"
   END CASE
```

```
    FOREACH ar_list INTO r.*
       OUTPUT TO REPORT ar_report (r.*)
       IF int_flag THEN
          LET int_flag = FALSE
          EXIT FOREACH
       END IF
    END FOREACH
    FINISH REPORT ar_report
    IF int_flag THEN
       LET int_flag = FALSE
       ERROR "AR REPORT printing aborted" ATTRIBUTE (RED, REVERSE)
       RETURN
    END IF
    IF print_option = "f" THEN
       LET msg = "AR REPORT printed to ", file_name CLIPPED
       CALL mess(msg, 23)
    END IF
END FUNCTION


REPORT ar_report (r)
    DEFINE r RECORD
           customer_num      LIKE customer.customer_num,
           fname             LIKE customer.fname,
           lname             LIKE customer.lname,
           company           LIKE customer.company,
           order_num         LIKE orders.order_num,
           order_date        LIKE orders.order_date,
           ship_date         LIKE orders.ship_date,
           paid_date         LIKE orders.paid_date,
           total_price       LIKE items.total_price
       END RECORD,
    name_text CHAR(80)

OUTPUT
    PAGE LENGTH 22
    LEFT MARGIN 0

FORMAT
    PAGE HEADER
       PRINT 15 SPACES,"West Coast Wholesalers, Inc."
       PRINT 6 SPACES,
          "Statement of ACCOUNTS RECEIVABLE - ",
          TODAY USING "mmm dd, yyyy"
       SKIP 1 LINES
       LET name_text = r.fname CLIPPED," ",r.lname CLIPPED,"/",
          r.company CLIPPED
       PRINT 29 - length(name_text)/2 SPACES, name_text
       SKIP 1 LINES
       PRINT " Order Date   Order Number     Ship Date              Amount"
       PRINT "-------------------------------------------------------------"

    BEFORE GROUP OF r.customer_num
       SKIP TO TOP OF PAGE
```

```
    AFTER GROUP OF r.order_num
        NEED 3 LINES
        PRINT " ",r.order_date,7 SPACES,r.order_num USING "###&",8 SPACES,
            r.ship_date,"  ",
            GROUP SUM(r.total_price) USING "$$$$,$$$,$$$.&&"

    AFTER GROUP OF r.customer_num
        PRINT 42 SPACES,"----------------"
        PRINT 42 SPACES,GROUP SUM(r.total_price) USING "$$$$,$$$,$$$.&&"

    PAGE TRAILER
        IF print_option = "s" THEN
            PAUSE "Type RETURN to continue"
        END IF
END REPORT


FUNCTION update_options()
    DEFINE po CHAR(2)

    DISPLAY "Current print option:" AT 8,25
    LET po = " ", print_option
    DISPLAY po AT 8,46 ATTRIBUTE(CYAN)
    MENU "REPORT OPTIONS"
        COMMAND "File" "Send all reports to a file"
            LET print_option = "f"
            EXIT MENU
        COMMAND "Printer" "Send all reports to the printer"
            LET print_option = "p"
            EXIT MENU
        COMMAND "Screen" "Send all reports to the terminal screen"
            LET print_option = "s"
            EXIT MENU
        COMMAND "Exit"
            EXIT MENU
    END MENU
    DISPLAY "" AT 8,1
END FUNCTION


FUNCTION print_backlog()
    ERROR "This option has not been implemented" ATTRIBUTE (RED)
END FUNCTION


FUNCTION print_stock()
    ERROR "This option has not been implemented" ATTRIBUTE (RED)
END FUNCTION
```

d4__demo.4gl

```
FUNCTION demo()

    CALL ring_menu()
    MENU "DEMO"
        COMMAND "Menus" "Source code for MAIN Menu"
            CALL showhelp(2001)
        COMMAND "Windows" "Source code for STATE CODE Window"
            CALL showhelp(2007)
        COMMAND "Forms" "Source code for new CUSTOMER data entry"
            CALL showhelp(2006)
        COMMAND "Detail-Scrolling"
          "Source code for scrolling of new ORDER line-items"
            CALL showhelp(2003)
        COMMAND "Scroll-Cursor" "Source code for customer record BROWSE/SCROLL"
            CALL showhelp(2008)
        COMMAND "Query_language" "Source code for new order insertion using SQL"
            CALL showhelp(2004)
        COMMAND "Construct_query"
            "Source code for QUERY-BY-EXAMPLE selection and reporting"
            CALL showhelp(2002)
        COMMAND "Reports" "Source code for MAILING LABEL report"
            CALL showhelp(2005)
        COMMAND "Exit" "Return to MAIN MENU"
            CLEAR SCREEN
            EXIT MENU
    END MENU
END FUNCTION
```

helpdemo.src

.101
The Customer option presents you with a menu that allows you to:

- Add new customers to the database
- Locate customers in the database
- Update customer files
- Remove customers from the database

.102
The Orders option presents you with a menu that allows you to:

- Enter a new order and print an invoice
- Update an existing order
- Look up and display orders
- Remove orders from the database

.103
The Stock option presents you with a menu that allows you to:

- Add new items to the list of stock
- Look up and display stock items
- Modify current stock descriptions and values
- Remove items from the list of stock

.104
The Reports option presents you with a menu that allows you to:

- Select and print mailing labels sorted by zip code
- Print a report of current accounts receivable
- Print a report of backloged orders
- Print a list of current stock available
- Change the report output options

.105
The Exit option leaves the program and returns you to the operating system.

.201
The One-add option enables you to enter data on new customers to the database. You may get assistance on what input is appropriate for each field by pressing the function key F1 when the cursor is in the field. When you have entered all the data you want for a given customer, press ESC to enter the data in the database. If you want to abort a given entry and not write it to the database, press the INTERRUPT key (usually DEL or CTRL-C).

.202
The Many-add option enables you to enter data on new customers to the database. You may get assistance on what input is appropriate for each field by pressing the function key F1 when the cursor is in the field. When you have entered all the data you want for a given customer, press ESC to enter the data in the database. If you want to abort a given entry and not write it to the database, press the INTERRUPT key (usually DEL or CTRL-C). After each entry, the cursor will move to the beginning of the form and await the entry of the next customer. If you have no more customers to add, press CTRL-Z to return to the CUSTOMER Menu.

.203
The Find-cust option allows you to select one or more customers and to display their data on the screen by using query-by-example input. Use the RETURN or arrow keys to move through the form. Enter the criteria you want the program to use in searching for customers. Your options include:

- o Literal values
- o A range of values (separated by ":")
- o A list of values (separated by "|")
- o Relational operators (for example ">105")
- o Wildcards like ? and * to match single or any number of characters

.204
The Update-cust option enables you to alter data on old customers in the database. You must first select a current customer row to deal with by using the Find-cust option. You may get assistance on what input is appropriate for each field by pressing the function key F1 when the cursor is in the field. When you have altered all the data you want for a given customer, press ESC to enter the data in the database. If you want to abort the changes and not write them to the database, press the INTERRUPT key (usually DEL or CTRL-C).

.205
The Delete-cust option enables you to remove customers from the database. You must first select a current customer row to deal with by using the Find-cust option. For your protection, you will be asked to confirm that the record should be deleted. Once deleted, it cannot be restored. Customers with active orders can not be deleted.
.206
The Exit option of the CUSTOMER Menu takes you back to the MAIN Menu.

.301
The Add-order option enables you to add a new order for an existing customer. You must first select the desired customer using query-by-example selection criteria. You will then enter the order date, PO number, and shipping instructions. The detail line items are then entered into a scrolling display array. Up to ten items may be entered using the four line screen array. After the new order is entered, an invoice is automatically generated and displayed on the screen.
.302
The Update-order option is currently not implemented.
.303
The Find-order option enables you to browse through and select an existing order. You must first select the desired customer (or customers) who's orders you wish to scan. For each customer selected, each corresponding order will be displayed on the screen for examination. You may either select an invoice, skip to the next invoice, or cancel processing.
.304
The Delete-order option is currently not implemented.
.305
The Exit option of the ORDER Menu returns you to the MAIN Menu.

.311
You may enter up to ten line items into the scrolling screen array. A number of standard functions are available for manipulating the cursor in a screen array.

- o F1Insert new line in the screen array
- o F2Remove the current line from the screen array
- o F3Page down one page in the screen array
- o F4Page up one page in the screen array
- o ESC Exit input array
- o CTRL-B When in the Stock Number or Manufacturer Code fields, a window will open in the middle of the screen and display a scrolled list of all items in stock, identified by the stock number and manufacturer. Using F3, F4, and the up and down arrow keys, move the cursor to the line that identifies the desired item and hit ESC. The window will disappear and the selected information will automatically appear in the proper line.
- o etc... The arrow-keys, and the standard field editing keys are available

The item_total field will be displayed in reverse-video green for total amounts over $500.
.401
The Add-stock option is currently not implemented.
.402
The Find-stock option is currently not implemented.
.403
The Update-stock option is currently not implemented.
.404
The Delete-stock option is currently not implemented.
.405
The Exit option of the STOCK Menu returns you to the MAIN Menu.

.501
The Labels option enables you to create a list of mailing labels generated using a query-by-example specification. You will be prompted for the output file name.
.502
The Accounts-receivable option enables you to create a report summarizing all unpaid orders in the database. You will be prompted for the output file name.
.503
The Backlog option is currently not implemented.
.504
The Stock-list option is currently not implemented.
.505
The Options option enables you to change the destination of any report generated during the current session. The default option is to display all reports on the terminal screen. The other options are to print all reports to either the printer or an operating system file.
.506
The Exit option of the REPORT Menu returns you to the MAIN Menu.

.1001
The Number field on the Customer Form contains the serial integer assigned to the customer row when the data for the customer is first entered into the database. It is a unique number for each customer. The lowest value of this field is 101.

```
.1002
The first section following the Name label should contain the first name of the
contact person at the customer's company.
.1003
The second section following the Name label should contain the last name of the
contact person at the customer's company.
.1004
This field should contain the name of the customer's company.
.1005
The first line of the Address section of the form should contain the mailing
address of the company.
.1006
The second line of the Address section of the form should be used only when
there is not sufficient room in the first line to contain the entire mailing
address.
.1007
The City field should contain the city name portion of the mailing address of
the customer.
.1008
Enter the two-character code for the desired state. If no code is entered, or
the entered code is not in the program's list of valid entries, a window will
appear on the screen with a scrolling list of all states and codes. Using the
F3, F4, up and down arrow keys, move the cursor to the line containing the
desired state. After typing ESC, the window will disappear and the selected
state code will appear in the customer entry screen.
.1009
Enter the five digit Zip Code in this field.
.1010
Enter the telephone number of the contact person at the customer's company.
Include the Area Code and extension using the format "###-###-#### #####".

.2001
The following is the INFORMIX-4GL source for the main menu. Note that only
the text is specified by the MENU statement; the structure and runtime menu
functions are built-in.

    OPTIONS
        HELP FILE "helpdemo"
    OPEN FORM menu_form FROM "ring_menu"
    DISPLAY FORM menu_form
    MENU "MAIN"
        COMMAND "Customer" "Enter and maintain customer data" HELP 101
            CALL customer()
            DISPLAY FORM menu_form
        COMMAND "Orders" "Enter and maintain orders" HELP 102
            CALL orders()
            DISPLAY FORM menu_form
        COMMAND "Stock" "Enter and maintain stock list" HELP 103
            CALL stock()
            DISPLAY FORM menu_form
        COMMAND "Reports" "Print reports and mailing labels" HELP 104
            CALL reports()
            DISPLAY FORM menu_form
        COMMAND "Exit" "Exit program and return to operating system" HELP 105
            CLEAR SCREEN
            EXIT PROGRAM
    END MENU
```

.2002
The following is the INFORMIX-4GL source code for mailing-label selection and printing. The CONSTRUCT statement manages the query-by-example input and builds the corresponding SQL where-clause.

```
CONSTRUCT BY NAME where_part ON customer.*
LET query_text = "select * from customer where ", where_part CLIPPED,
    " order by zipcode"
PREPARE mail_query FROM query_text
DECLARE label_list CURSOR FOR mail_query
PROMPT "Enter file name for labels >" FOR file_name
MESSAGE "Printing mailing labels to ", file_name CLIPPED," -- Please wait"

START REPORT labels_report TO file_name
FOREACH label_list INTO p_customer.*
    OUTPUT TO REPORT labels_report (p_customer.*)
END FOREACH
FINISH REPORT labels_report
```

See the source code option REPORT for the corresponding report routine.

.2003
The following is the INFORMIX-4GL source code for order entry using scrolled input fields. Only the INPUT ARRAY statement in needed to utilize the full scrolling features. Some additional code has been added merely to customize the array processing to this application.

```
DISPLAY "Press ESC to write order" AT 1,1
INPUT ARRAY p_items FROM s_items.* HELP 311
    BEFORE FIELD stock_num
        MESSAGE "Enter a stock number."
    BEFORE FIELD manu_code
        MESSAGE "Enter the code for a manufacturer."
    AFTER FIELD stock_num, manu_code
        LET pa = arr_curr()
        LET sc = scr_line()
        SELECT description, unit_price
            INTO p_items[pa].description,
                p_items[pa].unit_price
            FROM stock
            WHERE stock_num = p_items[pa].stock_num AND
                stock_manu = p_items[pa].menu_code
        DISPLAY p_items[pa].description, p_items[pa].unit_price
            TO stock[sc].*
        CALL item_total()
    AFTER FIELD quantity
        CALL item_total()
    AFTER INSERT, DELETE, ROW
        CALL order_total()
END INPUT
```

See the source code option QUERY-LANGUAGE for the SQL statements that insert the order information into the database.

.2004
The following is the INFORMIX-4GL source code that uses SQL to insert the entered order information into the database. Note that the use of transactions ensures that database integrity is maintained, even if an intermediate operation fails.

```
    BEGIN WORK
    LET p_orders.order_num = 0
    INSERT INTO orders VALUES (p_orders.*)
    IF status < 0 THEN
       ROLLBACK WORK
       MESSAGE "Unable to complete update of orders table"
       RETURN
    END IF
    LET p_orders.order_num = SQLCA.SQLERRD[2]
    DISPLAY BY NAME p_orders.order_num
    FOR i = 1 to arr_count()
        INSERT INTO items
           VALUES (p_items[counter].item_num, p_orders.order_num,
              p_items[counter].stock_num,     p_items[counter].manu_code,
              p_items[counter].quantity,      p_items[counter].total_price)
        IF status < 0 THEN
           ROLLBACK WORK
           Message "Unable to insert items"
           RETURN FALSE
        END IF
    END FOR
    COMMIT WORK
```

.2005
The following is the INFORMIX-4GL source code that generates the mailing-label report. See the source code option CONSTRUCT for the report calling sequence.

```
    REPORT labels_report (rl)
       DEFINE rl RECORD LIKE customer.*
       OUTPUT
          TOP MARGIN 0
          PAGE LENGTH 6
       FORMAT
          ON EVERY ROW
             SKIP TO TOP OF PAGE
             PRINT rl.fname CLIPPED, 1 SPACE, rl.lname
             PRINT rl.company
             PRINT rl.address1
             IF rl.address2 IS NOT NULL THEN
                PRINT rl.address2
             END IF
             PRINT rl.city CLIPPED, ", ", rl.state, 2 SPACES, rl.zipcode
    END REPORT
```

.2006
The following is the INFORMIX-4GL source code that manages a simple form for data entry. Note the use of special key definitions during data entry.

```
    OPEN FORM cust_form FROM "customer"
    DISPLAY FORM cust_form

    MESSAGE "Press F1 or CTRL-F for field help;",
       "F2 or CTRL-Z to return to CUSTOMER Menu"
    DISPLAY "Press ESC to enter new customer data or DEL to abort entry"
    INPUT BY NAME p_customer.*
       AFTER FIELD state
          CALL statehelp()
       ON KEY (F1, CONTROL-F)
          CALL customer_help()
       ON KEY (F2, CONTROL-Z)
          CLEAR FORM
          RETURN
    END INPUT
```

.2007
The following is the INFORMIX-4GL source code that opens a window in the customer entry screen, displays the list of valid state names and codes, saves the index into the p_state array for the selected state, closes the window, and returns the index to the calling routine.

```
   OPEN WINDOW w_state AT 8,40
      WITH FORM "state_list"
      ATTRIBUTE (BORDER, RED, FORM LINE 2)

   CALL set_count(state_cnt)
   DISPLAY ARRAY p_state TO s_state.*
   LET idx = arr_curr()

   CLOSE WINDOW w_state
   RETURN (idx)
```

.2008
The following is the INFORMIX-4GL source code that allows the user to browse through the rows returned by a "scroll" cursor.

```
   DECLARE customer_set SCROLL CURSOR FOR
      SELECT * FROM customer
         ORDER BY lname
   OPEN customer_set
   FETCH FIRST customer_set INTO p_customer.*
   IF status = NOTFOUND THEN
      LET exist = FALSE
   ELSE
      LET exist = TRUE
      DISPLAY BY NAME p_customer.*
      MENU "BROWSE"
         COMMAND "Next" "View the next customer in the list"
            FETCH NEXT customer_set INTO p_customer.*
            IF status = NOTFOUND THEN
               ERROR "No more customers in this direction"
               FETCH LAST customer_set INTO p_customer.*
            END IF
            DISPLAY BY NAME p_customer.*
         COMMAND "Previous" "View the previous customer in the list"
            FETCH PREVIOUS customer_set INTO p_customer.*
            IF status = NOTFOUND THEN
               ERROR "No more customers in this direction"
               FETCH FIRST customer_set INTO p_customer.*
            END IF
            DISPLAY BY NAME p_customer.*
         COMMAND "First" "View the first customer in the list"
            FETCH FIRST customer_set INTO p_customer.*
            DISPLAY BY NAME p_customer.*
         COMMAND "Last" "View the last customer in the list"
            FETCH LAST customer_set INTO p_customer.*
            DISPLAY BY NAME p_customer.*
         COMMAND "Select" "Exit BROWSE selecting the current customer"
            LET chosen = TRUE
            EXIT MENU
         COMMAND "Quit" "Quit BROWSE without selecting a customer"
            LET chosen = FALSE
            EXIT MENU
      END MENU
   END IF
   CLOSE customer_set
```

Appendix B

System Catalogs

System Catalogs

Information about the database is maintained in the system catalogs. The system catalogs are

systables	describes database tables.
syscolumns	describes columns in tables.
sysindexes	describes indexes on columns.
systabauth	identifies table-level privileges.
syscolauth	identifies column-level privileges.
sysdepend	describes how views depend on tables.
syssynonyms	lists synonyms for tables.
sysusers	identifies database-level privileges.
sysviews	defines views.

The following listing gives a brief description of the system catalogs.

The SYSTABLES Catalog

Column Name	Type	Explanation
tabname	char(18)	name of table
owner	char(8)	owner of table
dirpath	char(64)	directory path for the table file
tabid	serial	internal table identifier
rowsize	smallint	row size
ncols	smallint	number of columns
nindexes	smallint	number of indexes
nrows	integer	number of rows
created	date	date created
version	integer	table version number
tabtype	char(1)	table type (T = table, V = view, L = transaction log)
audpath	char(64)	audit filename (full pathname)

Index Name	Type	Columns
tabname	unique	tabname, owner
tabid	unique	tabid

The SYSCOLUMNS Catalog

Column Name	Type	Explanation
colname	char(18)	column name
tabid	integer	table identifier
colno	smallint	column number
coltype	smallint	column type
collength	smallint	column length (physical)

Index Name	Type	Columns
column	unique	tabid, colno

The SYSINDEXES Catalog

Column Name	Type	Explanation
idxname	char(18)	index name
owner	char(8)	owner of index
tabid	integer	table identifier
idxtype	char(1)	index type (U = unique, D = dups)
clustered	char(1)	clustering
part1	smallint	column number
part2	smallint	column number
part3	smallint	column number
part4	smallint	column number
part5	smallint	column number
part6	smallint	column number
part7	smallint	column number
part8	smallint	column number

Index Name	Type	Columns
idxtab	dupls	tabid
idxname	unique	idxname, tabid

The SYSTABAUTH Catalog

Column Name	Type	Explanation
grantor	char(8)	grantor of permission
grantee	char(8)	grantee (receiver) of permission
tabid	integer	table identifier
tabauth	char(7)	authorization type

Index Name	Type	Columns
tabgtor	unique	tabid, grantor, grantee
tabgtee	dupls	tabid, grantee

The SYSCOLAUTH Catalog

Column Name	Type	Explanation
grantor	char(8)	grantor of permission
grantee	char(8)	grantee (receiver) of permission
tabid	integer	table identifier
colno	smallint	column number
colauth	char(2)	authorization type

Index Name	Type	Columns
colgtor	unique	tabid, grantor, grantee, colno
colgtee	dupls	tabid, grantee

The SYSDEPEND Catalog

Column Name	Type	Explanation
btabid	integer	tabid of base table or view
btype	char(1)	base object type (table or view)
dtabid	integer	tabid of dependent table
dtype	char(1)	dependent object type (only view now)

Index Name	Type	Columns
btabid	dups	btabid
dtabid	dups	dtabid

The SYSSYNONYMS Catalog

Column Name	Type	Explanation
owner	char(8)	user name of owner
synonym	char(18)	synonym identifier
created	date	date synonym created
tabid	integer	table identifier

Index Name	Type	Columns
synonym	unique	owner, synonym
syntabid	dupls	tabid

The SYSUSERS Catalog

Column Name	Type	Explanation
username	char(8)	user login identifier
usertype	char(1)	D = DBA, R = RESOURCE, C = CONNECT
priority	smallint	reserved for future use
password	char(8)	reserved for future use

Index Name	Type	Columns
users	unique	username

The SYSVIEWS Catalog

Column Name	Type	Explanation
tabid	integer	table identifier
seqno	smallint	sequence number
text	char(64)	portion of SELECT statement

Index Name	Type	Columns
view	unique	tabid, seqno

Appendix C

Environment Variables

Environment Variables

INFORMIX-4GL makes the following assumptions about the user's environment:

- The editor used is the predominant editor for the operating system (usually **vi** for UNIX systems and **edlin** for DOS systems).
- The database worked with is in the current directory.
- If the computer is running UNIX, the program that sends files to the printer is **lp**. If the computer is running DOS, the name of the printer device is **lpt1**.
- On UNIX systems, you use **/tmp** to store temporary files. On DOS systems, you use the current directory to store temporary files.
- The INFORMIX-4GL program and its associated files are located in **/usr/informix** (on a UNIX system) or **\informix** (on a DOS system).

Setting Environment Variables

You can change any of the assumptions by setting one or more of the environment variables recognized by **INFORMIX-4GL**. These environment variables are listed by operating system in later sections of this appendix.

Setting Environment Variables on UNIX Systems

You may set environment variables at the system prompt or in your **.profile** (Bourne shell) or **.login** (C shell) file. If you set an environment variable at the system prompt, you will have to assign it again the next time you log onto the system. If you set a variable in your **.profile** (Bourne shell) or **.login** (C shell) file, UNIX will assign it automatically every time you log onto the system.

If you are using the Bourne shell, enter the following two commands to set the **ABCD** environment variable to *value*:

```
ABCD=value
export ABCD
```

If you are using the C shell, enter the following command to set the **ABCD** environment variable to *value*:

```
setenv ABCD value
```

Setting Environment Variables on DOS Systems

You may set environment variables at the system prompt or in your **AUTOEXEC.BAT** file. If you set an environment variable at the system prompt, you will have to assign it again the next time you start the system. If you set a variable in your **AUTOEXEC.BAT** file, DOS will assign it automatically every time you start the system.

Enter the following command to set the **ABCD** environment variable to *value*:

```
set ABCD=value
```

UNIX Environment Variables

The environment variables recognized by **INFORMIX-4GL** are as follows:

DBDELIMITER specifies the field delimiter used by the **dbload** utility in unloaded data files. The vertical bar (| = ASCII 124) is the default.

For example, if you are using the C shell, enter the following command to set the field delimiter to a plus sign:

```
setenv DBDELIMITER +
```

DBDATE specifies the format you want to use for DATE values. Through DBDATE, you can specify

- The order of the month, day, and year in a date
- Whether the year should be printed with two digits (Y2) or four digits (Y4)
- The separator between the month, day, and year

The default value for DBDATE is

```
MDY4/
```

where M represents the month, D represents the day, Y4 represents a four-digit year, and the separator is a (/) slash (mm/dd/yyyy).

Other acceptable values for the separator are a hyphen (-) or a period (.). The slash appears if you attempt to use any values other than the hyphen or period.

The separator value must always be specified last. The number of digits specified for the year must always follow the Y. To make the date appears as mm.dd.yy, you would set the DBDATE environment variable for the C shell as follows:

```
setenv DBDATE MDY2.
```

For the Bourne shell, you would use

```
DBDATE=MDY2.
export DBDATE
```

Suppose you wanted the date to appear in European format (dd-mm-yyyy). For the C shell, you would set the DBDATE environment variable as follows:

```
setenv DBDATE DMY4-
```

For the Bourne shell, you would use

```
DBDATE=DMY4-
export DBDATE
```

DBEDIT — names the text editor you want to use to create program files, form specification files, and command files from within the **INFORMIX-4GL** Programmer's Environment. For most systems, the default is **vi**.

DBLANG — specifies the subdirectory of **$INFORMIXDIR** that contains the message (**.iem**) files used by your program. The default is **$INFORMIXDIR/msg.**

If you want to use a message directory other than **$INFORMIXDIR/msg**, perform the following steps:

1. Use the **mkdir** command to create the appropriate subdirectory in **$INFORMIXDIR**. Set the user and group to **informix** and the access permission for this directory to 755.

2. Set the DBLANG environment variable to the new subdirectory. If you are using the Bourne shell, enter

```
DBLANG=dirname
export DBLANG
```

   If you are using the C shell, enter

```
setenv DBLANG dirname
```

3. Copy the **.iem** files to the message directory specified by **$INFORMIXDIR/$DBLANG**. All files in the message directory should have user and group **informix** and 644 access permission.

4. Run your program.

DBMONEY applies to MONEY values. It has the format

```
[front]{. | ,}[back]
```

where *front* is the optional symbol that precedes the MONEY value, the comma or the period is the optional symbol that separates the integral from the fractional part of the MONEY value, and *back* is the optional symbol that follows the MONEY value. The *front* and *back* symbols may be up to seven characters long and may contain any characters except commas or periods.

The default value for DBMONEY is

```
$.
```

where the dollar sign precedes the MONEY value, and the period (.) separates the integral from the fractional part of the MONEY value. (No *back* symbol appears.)

Suppose you wanted to represent MONEY values in deutsche marks. For the C shell, you would set the DBMONEY environment variable as follows:

```
setenv DBMONEY DM,
```

For the Bourne shell, you would use

```
DBMONEY=DM,
export DBMONEY
```

where DM is the currency symbol, and the comma separates the integral from the fractional part of the MONEY value.

Whenever you make a change to the *back* symbol, you must also supply the *front* symbol and the value separator (comma or period). Similarly, if you change the the value separator from a comma to a period, you must also supply the *front* symbol.

DBPATH

specifies a list of directories (in addition to the current directory) for **INFORMIX-4GL** to search for databases and associated files. If you have not selected a database, **INFORMIX-4GL** looks to DBPATH as well as the current directory to determine the list of available databases.

Once you have selected a database, INFORMIX-4GL looks first in the current directory and then in the parent directory of the directory containing the database (recall that each database is contained in a separate directory) for the associated forms, reports, and command files.

For example, if you want INFORMIX-4GL to search for database files in Kevin's and Debby's directories, as well as in your current directory, you would specify

```
DBPATH=/usr/Kevin:/usr/Debby
export DBPATH
```

Make sure you enter a colon between the directory names.

DBPRINT — specifies the print program for your computer. For most UNIX systems, the default program is **lp**.

DBTEMP — specifies the directory into which INFORMIX-4GL should place its temporary files. The default is the **/tmp** directory.

INFORMIXDIR — specifies the directory containing the INFORMIX-4GL files. The default is the **/usr/informix** directory.

DOS Environment Variables

The DOS environment variables recognized by **INFORMIX-4GL** are as follows:

DBDELIMITER specifies the field delimiter used by the **dbload** utility for unloaded data files. The vertical bar (| = ASCII 124) is the default.

To change the field delimiter to a semicolon, for example, you can enter

```
set DBDELIMITER=;
```

DBDATE specifies the format you want to use for DATE values. Through DBDATE, you can specify

- The order of the month, day, and year in a date
- Whether the year should be printed with two digits (Y2) or four digits (Y4)
- The separator between the month, day, and year

The default value for DBDATE is

```
MDY4/
```

where M represents the month, D represents the day, Y4 represents a four-digit year, and the separator is a (/) slash (mm/dd/yyyy). Other acceptable values for the separator are a hyphen (-) or a period (.). The slash appears if you attempt to use any values other than the hyphen or period.

Always specify the separator value last. Always follow the Y with the number of digits specified for the year. To make the

date appear as mm.dd.yy, set the DBDATE environment variable as follows:

```
set DBDATE=MDY2.
```

Suppose you wanted the date to appear in European format (dd-mm-yyyy). You would set the DBDATE environment variable as follows:

```
set DBDATE=DMY4-
```

DBEDIT

names the text editor you want to use to create program files, form specification files, and command files from within the **INFORMIX-4GL** Programmer's Environment. The default editor is **edlin**. To change the default editor, you can enter

```
set DBEDIT=program-name
```

where *program-name* is the name of the new editor.

DBLANG

specifies the subdirectory of **$INFORMIXDIR** that contains the message (**.iem**) files used by your program. The default is **$INFORMIXDIR\msg**.

If you want to use a message directory other than $INFORMIXDIR\msg, perform the following steps:

1. Use the **mkdir** command to create the appropriate subdirectory in **$INFORMIXDIR**.

2. Set the DBLANG environment variable to the new subdirectory by using the following command:

```
set DBLANG=dirname
```

3. Copy the **.iem** files to the message directory specified by **$INFORMIXDIR\$DBLANG**. All files in the message directory should have user and group **informix** and 644 access permission.

4. Run your program.

DBMONEY applies to MONEY values. It has the format

[*front*]{. | ,}[*back*]

where *front* is the optional symbol that precedes the MONEY value, the comma or the period is the optional symbol that separates the integral from the fractional part of the MONEY value, and *back* is the optional symbol that follows the MONEY value. The *front* and *back* symbols may be up to seven characters long and may contain any characters except commas or periods.

The default value for DBMONEY is

$.

where the dollar sign precedes the MONEY value, and the period separates the integral from the fractional part of the MONEY value.

If you want to represent MONEY values in deutsche marks, for example, you would set the DBMONEY variable as follows

```
set DBMONEY=DM,
```

where DM is the currency symbol, and the comma separates the integral from the fractional part of the MONEY value.

Whenever you make a change to the *back* symbol, you must also supply the *front* symbol and the value separator (comma or period). Similarly, if you change the the value separator from a comma to a period, you must also supply the *front* symbol.

DBPATH

specifies a list of directories for **INFORMIX-4GL** to search for databases and associated files. If you have not selected a database, **INFORMIX-4GL** looks to DBPATH to determine the list of available databases. If you have selected a database, **INFORMIX-4GL** looks first in the current directory and then in the directory where your database is located for associated forms, reports, and command files.

For example, if you want **INFORMIX-4GL** to search for database files in Kevin's directory on C: and Debby's directory on D:, you can enter

```
set DBPATH=c:\kevin;d:\debby
```

(When you set the DBPATH variable, make sure you enter a semicolon between directory names.)

Make sure you set the DBPATH environment variable before you execute the **startsql** command.

DBPRINT specifies the name of the printer device. The default is **lpt1**. To change the default to **lpt2**, for example, you can enter

```
set DBPRINT=lpt2
```

DBTEMP specifies the directory where **INFORMIX-4GL** should place its temporary files. The default is the current directory.

INFORMIXDIR specifies the directory containing the **INFORMIX-4GL** files. The default is **\informix**.

Appendix D

Reserved Words

Reserved Words

The following words are **INFORMIX-4GL** reserved words and cannot be used as program identifiers or database column names.

ABSOLUTE
ACCEPT
ADD
AFTER
ALL
ALTER
AND
ANY
ARRAY
AS
ASC
ASCENDING
ASCII
AT
ATTRIBUTE
ATTRIBUTES
AUDIT
AVERAGE
AVG
BEFORE
BEGIN
BETWEEN
BORDER
BOTTOM
BUFFERED
BY
CALL
CASE
CHAR
CHECK
CLEAR
CLIPPED
CLOSE
CLUSTER
COLUMN
COLUMNS
COMMAND
COMMENT
COMMIT
COMMITTED
CONNECT
CONSTRUCT
CONTINUE
COUNT
CREATE
CURRENT
CURSOR
DATABASE
DATE
DAY
DBA
DECIMAL
DECLARE
DEFAULTS
DEFER
DEFINE
DELETE
DESC
DESCENDING
DESCRIBE
DESCRIPTOR
DIRTY
DISPLAY
DISTINCT
DOUBLE
DOWN
DROP
ELSE
END
ERROR
EVERY
EXCLUSIVE
EXECUTE
EXISTS
EXIT
EXTENT
EXTERNAL
FALSE
FETCH
FIELD
FILE
FINISH
FIRST
FLOAT
FLUSH
FOR
FOREACH
FORM
FORMAT
FREE
FROM
FUNCTION
GLOBALS
GOTO
GRANT
GROUP
HAVING
HEADER
HELP

IF
IN
INDEX
INFIELD
INITIALIZE
INPUT
INSERT
INTEGER
INTERRUPT
INTO
IS
ISAM
ISOLATION
KEY
LABEL
LAST
LEFT
LENGTH
LET
LIKE
LINE
LINENO
LINES
LOCK
LOG
MAIN
MARGIN
MATCHES
MAX
MDY
MENU
MESSAGE
MIN
MOD
MODE
MODIFY
MONEY
MONTH
NAME
NEED
NEXT
NO
NOCR
NOT
NOTFFOUND
NULL
OF
OFF
ON
OPEN
OPTION
OPTIONS
OR
ORDER
OTHERWISE
OUTER
OUTPUT
PAGE
PAGENO
PAUSE
PERCENT
PIPE
PREPARE
PREVIOUS
PRINT
PRINTER
PRIOR
PRIVILEGES
PROGRAM
PROMPT
PUBLIC
PUT
QUIT
READ
READONLY
RECORD
RECOVER
REGISTER
RELATIVE
RENAME
REPEATABLE
REPORT
RESOURCE
RETURN
RETURNING
REVOKE
RIGHT
ROLLBACK
ROLLFORWARD
ROW
ROWS
RUN
SCROLL
SELECT
SERIAL
SET
SHARE
SHORT
SIZE

SKIP
SLEEP
SMALLFLOAT
SMALLINT
SPACE
SPACES
STABILITY
START
STATISTICS
STEP
STOP
STRING
SUM
SYNONYM
TABLE
TEMP
THEN
THROUGH
THRU
TIME
TO
TODAY
TOP
TOTAL
TRAILER
TRUE
UNION
UNIQUE
UNLOCK
UP
UPDATE
USING
VALIDATE
VALUES
VIEW
WAIT
WAITING
WARNING
WEEKDAY
WHEN
WHENEVER
WHERE
WHILE
WINDOW
WITH
WITHOUT
WORK
WRAP
YEAR

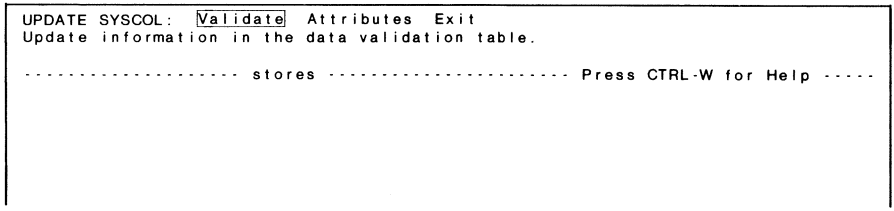
Appendix E

The *upscol* Utility

The *upscol* Utility

The **upscol** utility program allows you to create and modify the **syscolval** and **syscolatt** tables, which contain default information for fields in screen forms that correspond to database columns. For a full description of these tables and their use by INFORMIX-4GL, see Chapter 3.

You invoke the **upscol** utility by typing the command upscol on the command line in response to the system prompt. After you select a database (*db-name*), the following menu appears:



The options in the UPDATE SYSCOL Menu are

Validate Update the information in **syscolval**.

Attribute Update the information in **syscolatt**.

Exit Return to the operating system.

If you select either **Validate** or **Attribute**, **upscol** checks whether the corresponding table exists and, if not, asks whether you want to create it. In the text that follows, the corresponding table is called *syscol*. If you choose not to create it, you will return to the UPDATE SYSCOL Menu.

In the next step, **upscol** asks for the name of the table (*tab-name*) and the name of the column (*col-name*) whose defaults you want to modify in *syscol*. The selected table and column names appear, along with the database name, on the dividing line beneath the next menu:

```
ACTION: Add  Update  Remove  Next  Query  Table  Column  Exit
Add an entry to the data validation [or screen display attribute] table.

--------- db-name:tab-name:col-name ---------- Press CTRL-W for Help ---------
```

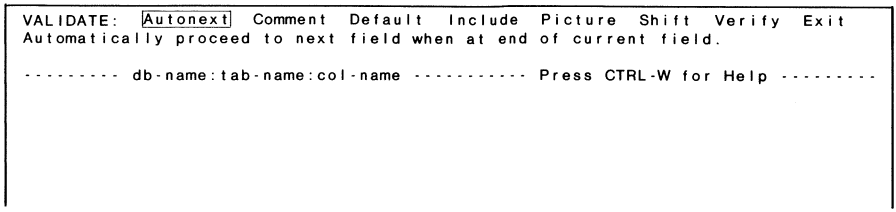
upscol displays the first row of **syscol** that relates to *tab-name* and *col-name* in the work area beneath this menu. If no such entries exist, a message stating this appears on the error line.

The options in the ACTION Menu are

Add	Add new rows to the **syscol** table.
Update	Update the currently displayed row.
Remove	Remove the currently displayed row (after a prompt for verification).
Next	Display the next row of **syscol**.
Query	Restart the display at the first row of **syscol** for *tab-name* and *col-name*.
Table	Select a new database table and column.
Column	Select a new column within *tab-name*.
Exit	Return to the UPDATE SYSCOL Menu.

Adding or Updating Under the **Validate** *Option*

When you select **Add** in the ACTION Menu after choosing the **Validate** option in the UPDATE SYSCOL Menu, the VALIDATE Menu appears:



The options are attribute names and their selection has the following effects:

Autonext Produces a menu with three options, **Yes**, **No**, and **Exit**. **Exit** returns you to the VALIDATE Menu. The default is **No**.

Comment Produces a prompt to enter a comment line. No quotation marks are required around the comment, but it must fit on a single screen line.

Default Produces a prompt to enter the DEFAULT attribute, formatted as described in Chapter 3. Quotation marks are required where necessary to avoid ambiguity.

Include Produces a prompt to enter the INCLUDE attribute, formatted as described in Chapter 3. Quotation marks are required where necessary to avoid ambiguity.

Picture Produces a prompt to enter the PICTURE attribute, formatted as described in Chapter 3. No quotation marks are required.

Shift	Produces a menu with four options, **Up**, **Down**, **None**, and **Exit**. **Up** corresponds to the UPSHIFT attribute and **Down** to the DOWNSHIFT attribute. **Exit** returns you to the VALIDATE Menu. The default is **None**.
Verify	Produces a menu with three options, **Yes**, **No**, and **Exit**. **Exit** returns you to the VALIDATE Menu. The default is **No**.
Exit	Returns you to the ACTION Menu.

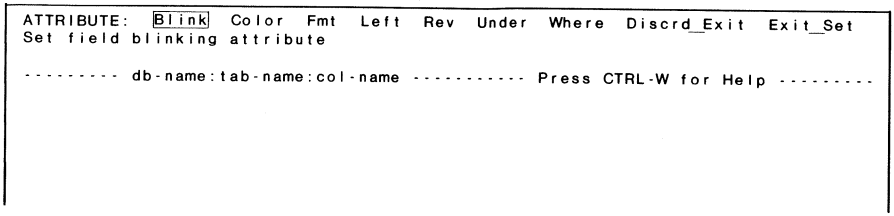
upscol adds or modifies a row of **syscolval** after you complete each of these options except **Exit**.

Note: No checking is done by **upscol** to ensure the validity or meaningfulness of your entries. Errors, if any, are revealed when you compile a form specification file with FORM4GL.

The **Update** option on the ACTION Menu takes you immediately to the ATTRIBUTE Menu or prompt corresponding to the current attribute for the current column. You may look at another attribute for the current column by using the **Next** option, start through the list again by using the **Query** option, remove the current attribute with the **Remove** option, and select a new column or table with the **Column** or **Table** options.

Adding or Updating Under the **Attribute** *Option*

When you select **Add** or **Update** in the ACTION Menu after choosing the **Attribute** option in the UPDATE SYSCOL Menu, the ATTRIBUTE Menu appears:



If you are adding a new row to **syscolatt**, a default row is displayed in the work area below the menu. If you are updating an existing row of **syscolatt**, the current row appears. Since no entry is made in **syscolatt** until you select **Exit_Set**, you may alter all the attributes before deciding to modify **syscolatt** (**Exit_Set**) or to abort the changes (**Discrd_Exit**).

The options in the ATTRIBUTE Menu are display-attribute names, and their selection has the following effects:

Blink — Produces a menu with three options, **Yes**, **No**, and **Exit**. The default is **No**.

Color — Produces a menu with the available colors (color terminals) or intensities (monochrome terminals) for display of **tab-name.col-name**. The colors displayed are those in the local **colornames** file. If no such file exists locally, **upscol** looks in $INFORMIXDIR/incl. If the file does not exist there, **upscol** uses the default color list (see Chapter 3). You can toggle back and forth among the colors or intensities using CONTROL–N.

Fmt Prompts you for the format string to be used when **tab-name.col-name** is displayed.

Left Produces a menu with three options, **Yes**, **No**, and **Exit**. **Yes** causes numeric data to be left justified within the screen field. The default is **No**.

Rev Produces a menu with three options, **Yes**, **No**, and **Exit**. **Yes** causes the field to be displayed in reverse video. The default is **No**.

Under Produces a menu with three options, **Yes**, **No**, and **Exit**. **Yes** causes the field to be displayed with underlining. The default is **No**.

Where Prompts for the values and value ranges under which these attributes will apply. See Chapter 3 for allowable syntax.

Discrd_Exit Discards the indicated changes and returns to the ACTION Menu.

Exit_Set Enters the indicated changes into the **syscolatt** table and returns to the ACTION Menu.

upscol adds or modifies a row of **syscolatt** after you complete each of these options except **Discrd_Exit**.

Note: No checking is done by **upscol** to ensure the validity or meaningfulness of your entries. Errors, if any, are revealed when you compile a form specification file with FORM4GL.

Appendix F

The *bcheck* Utility

The *bcheck* Utility

The **bcheck** program is a C-ISAM utility program that checks and repairs C-ISAM index files. It is distributed with INFORMIX-4GL.

bcheck compares an index (**.idx**) file to a data (**.dat**) file to see whether the two are consistent. If they are not, **bcheck** asks whether you want to delete and rebuild the corrupted indexes.

When running **bcheck** from the operating system command line, you must specify the table name used by the INFORMIX-4GL system catalogs. For example, the **customer** table in the **stores** database is identified in the system catalogs as **custome100** and not as **customer.**

You can list the contents of the database directory to determine the appropriate "table name."

In the following example, **bcheck** is run from the command line on the **customer** file and finds no errors.

```
bcheck -n custome100

BCHECK  C-ISAM B-tree Checker version 3.00.00
Copyright (C) 1981-1986 Informix Software, Inc.
Software Serial Number RDS#R000000


C-ISAM File: custome100

Checking dictionary and file sizes.
Index file node size = 1024
Current C_ISAM index file node size = 1024
Checking data file records.
Checking indexes and key descriptions.
Index 1 = unique key
    0 index node(s) used -- 1 index b-tree level(s) used
Index 2 = unique key  (0,4,2)
    1 index node(s) used -- 1 index b-tree level(s) used
Index 3 = duplicates  (111,5,0)
    1 index node(s) used -- 1 index b-tree level(s) used
Checking data record and index node free lists.
4 index node(s) used, 0 free -- 18 data record(s) used, 4 free
```

For each index, **bcheck** prints a group of up to eight numbers. These numbers indicate the position of the key in each record.

You can also use **bcheck** with the following options:

-i Check index file only
-l List entries in B+ trees
-n Answer no to all questions
-y Answer yes to all questions
-q Suppress printing of the program banner
-s Resize the index file node size

The **bcheck** command syntax is as follows:

bcheck -[**i** | **l** | **y** | **n** | **q** | **s**] *file-name*

Unless you use the **-n** or **-y** option, **bcheck** is interactive, waiting for you to respond to each error it finds.

Use the **-y** option with caution. Do not run **bcheck** using the **-y** option if you are checking the files for the first time.

Here is a sample run in which **bcheck** finds errors. The **-n** option is selected, so that each question **bcheck** asks is automatically answered "no."

```
BCHECK  C-ISAM B-tree Checker version 3.00.00
Copyright (C) 1981-1986 Informix Software, Inc.
Software Serial Number RDS#R000000


C-ISAM File: customer100

Checking dictionary and file sizes.
Index file node size = 1024
Current C-ISAM index file node size = 1024
Checking data file records.
Checking indexes and key descriptions.
Index 1 = unique key
    0 index node(s) used -- 1 index b-tree level(s) used

ERROR:  3 bad data record(s)
Delete index  ? no

Index 2 = unique key  (0,4,2)
    1 index node(s) used -- 1 index b-tree level(s) used

ERROR:  3 bad data record(s)
Delete index  ? no

Index 3 = duplicates  (111,5,0)
    1 index node(s) used -- 1 index b-tree level(s) used

ERROR:  3 bad data record(s)
Delete index  ? no

Checking data record and index node free lists.

ERROR:  3 missing data record(s)
Fix data record free list  ? no

4 index node(s) used, 0 free -- 18 data record(s) used, 4 free
```

Since **bcheck** finds errors, you must delete and rebuild the corrupted indexes. The **-y** option is used to answer "yes" to all questions asked by **bcheck**:

```
BCHECK  C-ISAM B-tree Checker version 3.00.00
Copyright (C) 1981-1986 Informix Software, Inc.
Software Serial Number RDS#R000000


C-ISAM File: custome100

Checking dictionary and file sizes.
Checking data file records.
Checking indexes and key descriptions.
Index 1 = unique key
    1 index node(s) used -- 1 index b-tree level(s) used

ERROR:  3 bad data record(s)
Delete index  ? yes

Remake index ? yes
Index 2 = unique key  (0,4,2)
    1 index node(s) used -- 1 index b-tree level(s) used

ERROR:  3 bad data record(s)
Delete index  ? yes

Remake index ? yes

Index 3 = duplicates  (111,5,0)
    1 index node(s) used -- 1 index b-tree level(s) used

ERROR:  3 bad data record(s)
Delete index  ? yes

Remake index ? yes

Checking data record and index node free lists.

ERROR:  3 missing data record(s)
Fix data record free list  ? yes

Recreate data record free list
Recreate index 3
Recreate index 2
Recreate index 1

4 index node(s) used, 0 free -- 18 data record(s) used, 4 free
```

Appendix G

The *mkmessage* Utility

Overview

When executing an INFORMIX-4GL program, the user may request help whenever the program is waiting for user input. This can occur while making a menu selection, while inputting data to a form, or while responding to a prompt. You can supply help messages that are displayed whenever the user presses the help key (set in the OPTIONS statement). These messages can be specific to the menu option currently highlighted or to the INPUT, INPUT ARRAY, or PROMPT statement. INFORMIX-4GL handles clearing and redisplaying the screen.

The *mkmessage* Utility

INFORMIX-4GL looks for the appropriate help message in the help file that you set in an OPTIONS statement, using the HELP FILE option. You may have several help files, but only one can be in effect at a time.

The **mkmessage** utility program converts a simply formatted help message text file into a form that can be read by INFORMIX-4GL programs. The syntax for the message source file, which is identified here by the extension **.src**, is very simple. Each help message should be preceded by a line with nothing on it but a period and a unique number. The period should be in the first column. The message starts on the next line and continues until the next numbered line. You will use the number to identify the help message in your INFORMIX-4GL programs (see the INPUT, INPUT ARRAY, MENU, and PROMPT statements).

All blank lines between two numbered lines are considered part of the message that belongs to the first of the two numbers. Lines beginning with # are considered comment lines and are ignored by **mkmessage**. For an example, see the **helpdemo.src** file from the Demonstration Application in Appendix A.

Once you have created your source file, you can process it for use by INFORMIX-4GL with this syntax:

---

mkmessage *helpfile.src helpfile*

You may name the output file anything you wish; it need not be as shown above. Use the output help file name in the OPTIONS statement to identify the help file.

If you want to use help messages from the help file on a field-by-field basis in an INPUT statement, you must make use of the **infield()** and **showhelp()** library functions supplied with INFORMIX-4GL. The example that follows illustrates the use of these functions:

```
OPTIONS
    HELP FILE "stores.hlp",
    HELP KEY F1
...
INPUT pr_fname, pr_lname, pr_phone
    FROM fname, lname, phone HELP 101
    ON KEY (F1)
        CASE
            WHEN INFIELD(lname)
                CALL SHOWHELP(111)
            WHEN INFIELD(fname)
                CALL SHOWHELP(112)
            WHEN INFIELD(phone)
                CALL SHOWHELP(113)
            OTHERWISE
                CALL SHOWHELP(101)
        END CASE
END INPUT
```

Appendix H

The *sqlconv* Utility

The *sqlconv* Utility

The **sqlconv** utility is provided for the Database Administrator who wants to use INFORMIX-4GL with an existing INFORMIX database (Version 3.0 or higher). From the INFORMIX database, **sqlconv** aids a user in creating a new RDSQL -compatible database. It leaves the old database intact.

This appendix discusses the steps necessary to convert an INFORMIX database to an RDSQL -compatible database. The new database can be used with INFORMIX-4GL on either DOS or UNIX systems.

The appendix is divided into two sections. Each section describes methods you can use to accomplish the conversion. The method you should use depends on the disk space constraints of your system.

Caution! **sqlconv** will not convert INFORMIX database permissions. You must grant new permissions on tables and fields after your new INFORMIX-4GL database has been created and loaded.

RDSQL reserved words are not the same as INFORMIX reserved words. If you receive a syntax error while running your INFORMIX-4GL programs, make sure your table and field names are not among the new reserved words. For a list of reserved words, see the appendix "Reserved Words" in this manual.

Make sure all INFORMIX composite fields are indexed. INFORMIX-4GL takes no action on unindexed composite fields. Indexed composite fields will have composite indexes created for them.

If you have used the LOCATION option to spread an INFORMIX database across a number of directories, converting the database using **sqlconv** places all of these files in the new **.dbs** directory.

You cannot specify a new starting number for a serial field.

If There Is No Shortage of Disk Space

You can convert an entire **INFORMIX** database to an **RDSQL** -compatible database for use with **INFORMIX-4GL** at one time if there is no shortage of disk space.

To convert an entire database at once, you must have available on the disk at least three times the amount of space required by all the tables in the database plus additional space for the **INFORMIX-4GL** system files. The following steps outline the process of converting an entire database at once:

1. Make certain that **INFORMIX** and **INFORMIX-4GL** are included in your search path. On DOS systems, **INFORMIX** and **INFORMIX-4GL** must be installed in the same directory.

2. Set the INFORMIXDIR environment variable to point to the **INFORMIX-4GL** directory.

3. Change your current directory to the directory that contains your **INFORMIX** database.

4. Create a backup copy of the **INFORMIX** database.

5. Enter

   ```
   sqlconv -4 databasename
   ```

   where *databasename* is the name of the **INFORMIX** database you want to convert. Do not include a filename extension. This command generates an **INFORMER** script file (indicated by a **.uld** extension), an **INFORMIX-4GL** program (indicated by a **.4gl** extension), and a **dbload** command file (indicated by a **.cmd** extension).

6. Enter

   informer *databasename databasename*.**uld**

   This command unloads the database files (in ASCII format) to *unload* files (indicated by a **.unl** extension) for each table in the database.

7. Enter

   c4gl *databasename*.**4gl** -o *databasename*.**4ge**

   This command compiles the **INFORMIX-4GL** program created in Step 5, and creates a **.c** file, a **.ec** file (C programs), and a **.4ge** executable file.

8. Enter

   databasename.**4ge**

   This command runs the **INFORMIX-4GL** program you compiled in Step 7 and re-creates the database, tables, and indexes in **RDSQL** format.

9. Enter

   dbload -d *databasename* -c *databasename*.**cmd** -l errlog

   This command loads the data from the **.unl** files (generated by **informer)** into the appropriate tables and creates an *errlog* file, which contains diagnostic information about any rows that were not successfully loaded. For more information on **dbload**, see the appendix "The **dbload** Utility" in this manual.

10. The final step in the conversion procedure is to remove all the old database files, the **.uld**, **.4gl**, **.4ge**, **.ec**, **.cmd**, **.unl**, and **.c** files from your directory. Do not remove your forms and reports. You can update these later.

If There Is a Shortage of Disk Space

The following method is more economical in terms of disk space, but it is a more involved and time-consuming process than the method discussed in the previous section. Use this method if you do not have at least three times the amount of space required by the database.

The following steps outline the conversion of the sample INFORMIX database, **payroll**, to an RDSQL -compatible database for use with INFORMIX-4GL

1. Make certain that INFORMIX and INFORMIX-4GL are included in your search path. On DOS systems, INFORMIX and INFORMIX-4GL must be installed in the same directory.

2. Set the INFORMIXDIR environment variable to point to the INFORMIX-4GL directory.

3. Change your current directory to the directory that contains your INFORMIX database.

4. Create a backup copy of the INFORMIX database.

5. Enter

   ```
   sqlconv -4 payroll
   ```

   Do not include a filename extension. This command generates an INFORMER script file (indicated by a **.uld** extension), an INFORMIX-4GL program (indicated by a **.4gl** extension), and a **dbload** command file (indicated by a **.cmd** extension).

6. Enter

   ```
   c4gl payroll.4gl -o payroll.4ge
   ```

   This command compiles the INFORMIX-4GL program you created in Step 5 and creates a **.c** file, a **.ec** file (C programs), and a **.4ge** executable file.

7. Enter

```
payroll.4ge
```

This command runs the INFORMIX-4GL program you compiled in Step 6 and re-creates the database, tables, and indexes in RDSQL format, but it does not load the data.

8. On UNIX systems, enter

```
cat payroll.uld
```

On DOS systems, enter

```
type payroll.uld
```

The statements in the **payroll.uld** file outline the first steps of the conversion operation. You must execute each of these statements separately. You may find it helpful to print a copy of the **payroll.uld** file for easy reference.

9. Enter

```
informer payroll
```

At the INFORMER prompt, enter the first line of the **payroll.uld** file exactly as it appears. This creates the unload file for the first table. Exit INFORMER.

10. The **.cmd** file describes the form of the data and contains the INSERT INTO statements indicating how this data is to be placed in the database files. The INSERT INTO statements are necessary to load the data into the newly created database. You must execute each of the statements separately. To do this, create a copy of the **payroll.cmd** file for each INSERT INTO statement and make sure you include the **.cmd** extension to the filename of each new file. In this instance, we have named the files **one.cmd** and **two.cmd**.

11. Edit the **one.cmd** file using your system editor, and remove all but the first INSERT INTO statement from the file. Exit the file.

12. Execute the first INSERT INTO statement in the **one.cmd** file by entering

```
dbload -d payroll -c one.cmd -l errlog
```

This command loads the data from the **.unl** file into the appropriate table and creates an *errlog* file, which contains diagnostic information about any rows that were not successfully loaded. A statement appears on the screen indicating how many rows were loaded into the file. For more information on **dbload**, see the appendix "The **dbload** Utility" in this manual.

13. Before you can perform the same operations on any other database tables, you must free additional disk space by erasing the INFORMIX versions of the recently created INFORMIX-4GL files. To erase these files, enter

```
dbstatus payroll
```

14. At the **dbstatus** prompt, enter

```
erase file filename
```

where *filename* is the name of the file referred to in the **.cmd** file created in Step 11. Exit **dbstatus.**

15. On UNIX systems, erase the unload file for this same file by entering from the command line

```
rm filename.unl
```

On DOS systems, erase the unload file for this same file by entering from the command line

```
erase filename.unl
```

16. Continue to unload each table, one at a time, from the **INFORMIX** database and then load it into the **INFORMIX-4GL** database. Do this by repeating Steps 9 through 15. Remember to make a copy of the **.cmd** file for each INSERT INTO statement it contains. Each copy must have a unique name and must end in the **.cmd** extension. The correct filename must be included on the command line each time you run the **dbload** command.

    Repeat these steps until all load statements in the **payroll.cmd** file have been executed. This operation loads the actual data into the newly created database.

17. Check the contents of the data files in the newly created **INFORMIX-4GL** database to make sure you have successfully converted your **INFORMIX** database.

18. When all tables in the database have been converted, erase the **.uld**, **.ec**, **.cmd**, **.c**, **.4gl**, and **.4ge** files, and drop the **INFORMIX** database.

Caution! You cannot rerun sqlconv after you have erased a table. The new scripts generated by the command do not contain the information necessary to convert the table.

Appendix I

The *dbupdate* Utility

The *dbupdate* Utility

Several significant changes occurred in the transition from an RDSQL Version 1 database to an RDSQL Version 2 database. These changes include expanding the number of system catalogs and slightly changing their content (see Appendix B), and introducing NULL values.

Version 2.1 of INFORMIX-SQL and INFORMIX-ESQL/C and Version 1.1 of INFORMIX-4GL use the RDSQL Version 2 database. You cannot use a Version 1 database with the Version 2.1 products or with INFORMIX-4GL unless you modify the old databases with the **dbupdate** utility.

Using dbupdate

To convert your old databases to an RDSQL Version 2 database, execute the following command:

---

dbupdate [**-b** | **-n**] *old-database-name new-database-name*

---

dbupdate creates a new RDSQL-compatible database in the current directory with the name *new-database-name*, and copies the data from the old system catalogs to the new system catalogs, making the appropriate changes.

If you do not use the **–b** or **–n** option, **dbupdate** converts the value of all CHAR type columns with blank data to NULL and, for each numeric column, asks you whether it should convert zero values to NULL values.

The **–b** option causes **dbupdate** to leave blank data in CHAR columns as blanks. The **–n** option alters the system catalogs to define all columns as NOT NULL and does not touch the data files. The **–n** option includes the **–b** option.

In addition to these changes, **dbupdate** corrects a bug in the representation of negative DECIMAL values.

When **dbupdate** finishes, you have two database directories (the new and the old) with two separate system catalogs, but the data and index files are shared (linked). To complete the update, you should remove the old database directory.

Caution! On DOS systems, executing the **dbupdate** command loads the database agent. Be certain that the database agent is not already loaded in memory when you execute this command.

To find out if the database agent is loaded, enter **set** from the command line. This displays a list of your DOS environment variables. If the database agent is loaded, the variable **sqlcaddr** is listed. If **sqlcaddr** is listed, you must unload the database agent using the **exit** command before you run **dbupdate.**

No NULL Databases

You may want to avoid dealing with NULL values and their three-valued logic. You can do this if you carefully adhere to the following rules:

- When converting the **RDSQL** Version 1 database to an **RDSQL** Version 2 database, select the **–n** option of **dbupdate.**
- When creating new tables, define all columns as NOT NULL.
- In all form specification files, add the WITHOUT NULL INPUT clause in the DATABASE section.
- In form specification files, specify all **formonly** fields as NOT NULL.

The last two rules mean you must recompile all your old form specification files.

Appendix J

The *dbload* Utility

The *dbload* Utility

The **dbload** utility provides a method for transferring information (raw data existing in ASCII form) from an operating-system file into an **RDSQL** Version 2 database. It allows for the easy and efficient transfer of database files created for other **RDSQL** databases—or created on an entirely different database system—to your **RDSQL** Version 2 database. Error logging provides diagnostic information about rows that are not successfully loaded.

When running **dbload**, you have the option of specifying the point at which each *N* number of rows is committed to the database—or of aborting the entire process and making no changes to the database.

These are the primary features of **dbload**:

- Data from one or more system files can be loaded into one or more database tables.
- Rows that cannot be loaded into the database are trapped and placed (with diagnostic information) into a user-specified file.
- Loading continues until a user-specified total number of rows cannot be entered into the database.
- The loading process can be optionally terminated without making changes to the existing database.
- Loading can begin at a specified line in the system file.
- Selected fields or columns can be loaded from the system file and entered in specified columns of a table.
- Both fixed- and variable-length data files can be loaded.
- NULL values can be indicated, and a different indicator can be used with each field.

- Constants (values not in the system file) can be entered into the database file.

To use **dbload**, first create a command file that tells **dbload** what actions you want carried out.

This command file

- Describes the form of the data in the system files

- Contains one or more RDSQL INSERT INTO statements indicating how this data is to be placed into the selected database files

You invoke **dbload** from the operating system command line by indicating (at a minimum) the name of the **dbload** program and an option. If you provide no additional information on the command line, **dbload** prompts you for the information (database name, command filename, and error-log filename) it must have to run.

The syntax of the basic **dbload** command is

---

dbload { **-d** *dbname* | **-c** *comfile* | **-l** *errlog* } ...

-d *dbname* is the name of the RDSQL database receiving the new information.

-c *comfile* is the name of the command file containing the **dbload** command statements.

-l *errlog* is the name of the file selected to receive the error-logging information.

dbload can be used with several options. Options must be placed on the command line. The following options are available:

-s The **-s** option instructs **dbload** to perform a syntax check on the command file only and load no data. The command file is displayed on the screen, and errors are indicated where they are found. You can redirect this output to a system file with the > redirect symbol.

You will normally want to run **dbload** with the **-s** option in advance of actually loading the data.

-n *num* **dbload** displays a message when each set of *num* rows is loaded into the database.

For databases with transactions, the **-n** *num* indicates the number of rows that must be successfully read by **dbload** before being committed to the database. Until this number is reached, no rows are committed to the database. The default is 100 rows.

-e *num* The **-e** *num* indicates the total number of bad rows that will be processed before **dbload** aborts the program. The default is 10 total rows.

For example, if *num* is set to 10, **dbload** aborts upon reaching the eleventh error. **dbload** aborts upon encountering the first bad row if *num* is set to 0.

-p For databases with transactions, the **-p** option instructs **dbload** to ask upon reaching the abort point whether to commit to the database the rows read since the last set of rows was committed. The default is to commit the lines to the database without asking for verification.

-i *num* The **-i** *num* option instructs **dbload** to ignore the first *num* number of rows in the system data file when executing your request. For example, this option is useful if the **dbload** program aborts after

line 240 of the system file data, and you want to begin loading with line 241. It is useful also if the system file contains header information preceding the data.

An example operating-system command line using all options (except **-s**) is

```
dbload -d stores -c comfile -l logfile
                 -n 50 -e 5 -p -i 20
```

In the example command line:

-d stores	stores is the name of the database
-c comfile	comfile is the name of the command file
-l logfile	logfile is the name of the error-log file
-n 50	**dbload** commits each set of 50 good rows to the database
-e 5	**dbload** allows 5 bad rows before aborting
-p	**dbload** prompts for instructions about committing rows read, but not yet committed to the database, upon reaching the abort point
-i 20	**dbload** ignores the first 20 lines of the data file

To abort the **dbload** program, press the interrupt key (DEL on UNIX systems, CTRL-C on DOS systems). Upon interrupt, rows read but not committed to the database are discarded.

Suggestion: The speed with which the **dbload** utility loads data is greatly affected by the presence of indexes. Before running **dbload** you may want to drop any indexes on the tables receiving the data. Then add new indexes once the program has finished.

The format of an example command file follows. A discussion of each line in this file follows the example.

```
FILE datafile1 (fld1 1 - 10 : 13 : 5 - 22  NULL = "str1" ,
                fld2 10 - 21 : 28 - 32 ,
                fld3 8 - 10 : 33 - 50 : 29 - 33  NULL = "str2" ,
                .
                .
                .
                fldN 9 : 16 - 19  NULL = "string") ;

INSERT INTO tab1 (col1, col2, col9, . . . . , colN) ;

INSERT INTO tab2
     VALUES  (fld1, fld3, "kevin", . . . . , fldN) ;

INSERT INTO tab3 ;              {no column or values list provided}

FILE "datafile.2" DELIMITER "|" Num ;     {variable length fields}

INSERT INTO tab1
     VALUES  (f01, f02, "kevin", "234", . . . . , f0N) ;

INSERT INTO tab4 ;
```

The preceding command file contains seven statements. Each statement is described below.

```
FILE datafile1 (fld1 1 - 10 : 13 : 5 - 22  NULL = "str1" ,
```

datafile1 is the name of the operating system data file. *fld1*, *fld2*, *fld3*, through *fldN* are the field names you select to identify each fixed-length data field.

In this example, *fld1* consists of the characters in positions 1 through 10, 13, and 5 through 22 of every row of *datafile1*. (All rows end with a **NEWLINE** character.) Character positions can be repeated in each field and across fields. Character positions 5 through 10 and 13 appear twice in *fld1*, and characters 10, 13, and 21 appear in fields *fld1* and *fld2*.

In *fld1* the NULL value is defined as *"str1."* **dbload** places a NULL value in the column wherever *"str1"* is encountered.

fld2 consists of the characters in positions 10 through 21 and 28 through 32.

fld3 consists of the characters in positions 8 through 10, 33 through 50, and 29 through 33. The NULL value in this field is defined as *"str2"*.

The field-definition process continues until the last field is reached. Here, *fldN* contains characters in positions 9 and 16 through 19. The NULL value is defined as *"string"*.

Characters and character sequences are separated within each field definition by a colon. Field definitions are separated from one another by a comma. The end of the set of field definitions is indicated with a right parenthesis and a semicolon. All statements end with a semicolon.

```
INSERT INTO tab1 (col1, col2, col9, . . . , colN) ;
```

A standard RDSQL INSERT statement follows. *col1*, *col2*, and so on are the actual database column names. Since no value list is provided, **dbload** takes the value list from the preceding FILE statement. **dbload** inserts *fld1* data into *col1*; *fld2* into *col2*; and *fld3* into *col9* of table *tab1*. (In this instance, columns 4 through 8 are skipped.) This process continues until *fldN* is inserted into *colN*.

```
INSERT INTO tab2
    VALUES  (fld1, fld3, "kevin", . . . , fldN) ;
```

Since no column list is provided, **dbload** checks the system catalogs for the column list for table *tab2*. The values to be loaded into each column are specifically stated. Here, *fld1* will be loaded as column1; *fld3*, as column2; and the constant *"kevin,"* as column3. The process continues until *fldN* appears in the final column.

```
INSERT INTO tab3 ;      {no column or values list provided}
```

Since no column or value list is provided, **dbload** checks the system catalogs for the column list for table *tab3*, and the value list is taken from the prior FILE statement (appearing at the start of the command file).

In this example, the value *fld1* is placed in column1; *fld2* in column2; *fld3* in column3; and *fldN* in columnN.

Note: This statement requires that the field list in the FILE statement have a one-to-one correspondence with the columns in the system catalogs. If this correspondence is not present,

dbload will not load the rows. Instead, you will receive an error message on each row. **dbload** aborts the loading process when the total number of bad rows exceeds the user-specified limit (if indicated) or the default limit of 10 bad rows.

A comment {no column or values list provided} is enclosed in braces.

```
FILE "datafile.2" DELIMITER "|" Num ; {variable length fields}
```

Columns in *datafile.2* are of variable length. The DELIMITER clause in this statement tells **dbload** that the "|" character separates each column. The same delimiter must be used throughout the file and must appear within quotation marks. *Num* is the number of columns that make up one row in the data file. Fields are automatically assigned the names *f01*, *f02*, *f03*, and so on. Any field that contains no characters is assigned a NULL value (CHAR fields are padded with blanks).

Because the filename contains an extension (.2), the filename must be enclosed in quotes ("*datafile.2*").

```
INSERT INTO tab1
    VALUES  (f01, f02, "kevin", "234", . . . , f0N) ;
```

Since no column list is provided, **dbload** checks the system catalogs for this information. The values to be loaded into each column of *tab1* are specifically stated. Here, *f01* will be loaded as column1; *f02*, as column2; the constant *"kevin,"* as column3; the value *"234"*, as column4; and so on, until the value *f0N* is placed in the last column.

You must reference fields in a variable length data file with the letter "f" followed by a two digit number: *f01*, *f02*, *f10*, and so on. The format *fld01* or *f3* is incorrect.

```
INSERT INTO tab4 ;
```

Since no column or value list is provided, **dbload** checks the system catalogs for the column list for table *tab4*. The value list is taken from the previous FILE statement. The file list in the FILE statement must have a one-to-one correspondence with the columns in the system catalogs. In this instance, the value *f01* is placed in column1; *f02* in column2; *f3* in column3; and so on, until the value *f0N* is placed in columnN.

The complete operating-system command line syntax is

---

dbload { **-d** *dbname* | **-c** *cfile* | **-l** *lfile* } ...
[-s] [-n *num***] [-e** *num***] [-p] [-i** *num***]**

-d *dbname* is the name of the database receiving the new information.

-c *cfile* is the name of the **dbload** command file.

-l *lfile* is the name of the error logging file.

-s instructs **dbload** only to check the syntax of the *cfile* statement—no data is loaded into the database.

-n *num* gives the number of rows (*num*) read by **dbload** before committing the rows to the database (default = 100 rows).

-e *num* is the total number of bad rows (*num*) that are processed before **dbload** aborts (default = 10 rows; **dbload** aborts on bad row 11.)

-p instructs **dbload** about what to do upon reaching the abort point. **dbload** asks about committing rows read, but not yet committed to the database, since the last commit was performed (default = commit).

-i *num* instructs **dbload** to ignore the first *num* number of rows in the data file.

You must include at least one of the above options. If you do not include the first three options, **dbload** prompts for the information (database name, command filename, and error logging filename) it must have.

Appendix K

DECIMAL Functions for C

DECIMAL Functions for C

The data type DECIMAL is a machine-independent method for the representation of numbers of up to thirty-two significant digits, with or without a decimal point, and exponents in the range −128 to +126. INFORMIX-4GL provides routines that facilitate the conversion of DECIMAL type numbers to and from every data type allowed in the C Language.

DECIMAL type numbers consist of an exponent and a mantissa (or fractional part) in base 100. In normalized form, the first digit of the mantissa must be greater than zero.

When used within a C program, DECIMAL type numbers are stored in a C structure of the type shown below.

```
#define DECSIZE 16

struct decimal
       {
       short dec_exp;
       short dec_pos;
       short dec_ndgts;
       char  dec_dgts[DECSIZE];
       };

typedef struct decimal dec_t;
```

The **decimal** structure and the typedef **dec_t** shown above can be found in the header file **decimal.h**. Include this file in all C source files that use any of the decimal routines:

```
#include <decimal.h>
```

The **decimal** structure has four parts:

dec__exp holds the exponent of the normalized DECIMAL type number. This exponent represents a power of 100.

dec__pos holds the sign of the DECIMAL type number (1 when the number is zero or greater; 0 when less than zero).

dec__ndgts contains the number of base 100 significant digits of the DECIMAL type number.

dec__dgts is a character array that holds the significant digits of the normalized DECIMAL type number (dec__dgts[0] != 0). Each character in the array is a one-byte binary number in base 100. **dec__ndgts** contains the number of significant digits in **dec__dgts**.

DECIMAL-Type Routines

All operations on DECIMAL-type numbers should take place through the routines provided in the **INFORMIX-4GL** library and described in the following pages. Any other operations, modifications, or analysis of DECIMAL-type numbers can produce unpredictable results.

The following C function calls are available in **INFORMIX-4GL** to treat DECIMAL-type numbers:

deccvasc()	convert C char type to DECIMAL type
dectoasc()	convert DECIMAL type to C char type
deccvint()	convert C int type to DECIMAL type
dectoint()	convert DECIMAL type to C int type
deccvlong()	convert C long type to DECIMAL type
dectolong()	convert DECIMAL type to C long type
deccvflt()	convert C float type to DECIMAL type
dectoflt()	convert DECIMAL type to C float type
deccvdbl()	convert C double type to DECIMAL type
dectodbl()	convert DECIMAL type to C double type
decadd()	add two decimal numbers
decsub()	subtract two decimal numbers
decmul()	multiply two decimal numbers
decdiv()	divide two decimal numbers
deccmp()	compare two decimal numbers
deccopy()	copy a decimal number
dececvt( )	convert decimal value to ASCII string (corresponds to ecvt(3) on UNIX systems)
decfcvt( )	convert decimal value to ASCII string (corresponds to fcvt(3) on UNIX systems)

DECCVASC

Overview

Use **deccvasc** to convert a value held as a printable character in a C **char** type into a DECIMAL-type number.

Syntax

```
deccvasc(cp, len, np)
      char *cp;
      int len;
      dec__t *np;
```

Explanation

cp points to a string that holds the value you want to convert.

len is the length of the string.

np is a pointer to a **dec__t** structure to receive the result of the conversion.

Notes

1. **deccvasc** ignores leading spaces in the character string.

2. The character string can have a leading plus (+) or minus (−) sign, a decimal point (.), and numbers to the right of the decimal point.

3. The character string can contain an exponent preceded by either *e* or *E*. The exponent can be preceded by a plus or minus sign.

Return Codes

0	Function was successful.
−1200	Number is too large to fit into a DECIMAL type (overflow).
−1201	Number is too small to fit into a DECIMAL type (underflow).
−1213	String has non-numeric characters.
−1216	String has bad exponent.

Examples

```
#include <decimal.h>

char input[80];
dec_t number;
        .
        .
        .
/* get input from terminal */
getline(input);

/* convert input into decimal number */
deccvasc(input, 32, &number);
```

DECTOASC

Overview

Use **dectoasc** to convert a DECIMAL type number to a printable ASCII string.

Syntax

```
dectoasc(np, cp, len, right)
    dec_t *np;
    char  *cp;
    int   len;
    int   right;
```

Explanation

np is a pointer to the decimal structure whose associated decimal value you want to convert to an ASCII string.

cp is a pointer to the beginning of the character buffer to hold the ASCII string.

len is the maximum length in bytes of the string buffer.

right is an integer indicating the number of decimal places to the right of the decimal point.

Notes

1. If *right* equals −1, the number of decimal places is determined by the decimal value of **np*.

2. If the number does not fit into a character string of length *len*, **dectoasc** converts the number to exponential notation. If the number still does not fit, **dectoasc** fills the string with asterisks. If the number is shorter than the string, it is left-justified and padded on the right with blanks.

Return Codes

0	Conversion was successful.
−1	Conversion was not successful.

Examples

```
#include <decimal.h>

char input[80];
char output[16];
dec_t number;
      .
      .
      .

/* get input from terminal */
getline(input);

/* convert input into decimal number */
deccvasc(input, 32, &number);

/* convert number to printable string */
dectoasc(&number, output, 16, 1);

/* print the value just entered */
printf("You just entered %s", output);
```

DECCVINT

Overview

Use **deccvint** to convert a C type **int** into a DECIMAL type number.

Syntax

```
deccvint(integer, np)
     int integer;
     dec__t *np;
```

Explanation

integer is the integer you want to convert.

np is a pointer to a **dec__t** structure that receives the result of the conversion.

Examples

```
#include <decimal.h>

dec__t decnum;

/* convert the integer value -999
 * into a DECIMAL type number
 */
deccvint(-999, &decnum);
```

DECTOINT

Overview

Use **dectoint** to convert a DECIMAL type number into a C type **int**.

Syntax

```
dectoint(np, ip)
      dec__t *np;
      int *ip;
```

Explanation

np is a pointer to a decimal structure whose value is converted to an integer.

ip is a pointer to the integer.

Return Codes

0	Conversion was successful.
−1200	The magnitude of the DECIMAL type number > 32767.

Examples

```
#include <decimal.h>

dec_t mydecimal;
int   myinteger;

/* convert the value in
 * mydecimal into an integer
 * and place the results in
 * the variable myinteger.
 */
dectoint(&mydecimal, &myinteger);
```

DECCVLONG

Overview

Use **deccvlong** to convert a C type **long** value into a DECIMAL-type number.

Syntax

```
deccvlong(lng, np)
      long lng;
      dec__t *np;
```

Explanation

lng is a pointer to a long integer.

np is a pointer to a **dec__t** structure that receives the result of the conversion.

Examples

```
#include <decimal.h>

dec_t mydecimal;
long  mylong;

/* Set the decimal structure
 * mydecimal to 37.
 */
deccvlong(37L, &mydecimal);

mylong = 123456L;
/* Convert the variable mylong into
 * a DECIMAL type number held in
 * mydecimal.
 */
deccvlong(mylong, &mydecimal);
```

DECTOLONG

Overview

Use **dectolong** to convert a DECIMAL-type number into a C type **long**.

Syntax

```
dectolong(np, lngp)
      dec__t *np;
      long  *lngp;
```

Explanation

np is a pointer to a decimal structure.

lngp is a pointer to a long where the result of the conversion will be placed.

Return Codes

0	Conversion was successful.
−1200	The magnitude of the DECIMAL type number > 2,147,483,647.

Examples

```
#include <decimal.h>

dec_t mydecimal;
long  mylong;

/* convert the DECIMAL type value
 * held in the decimal structure
 * mydecimal to a long pointed to
 * by mylong.
 */
dectolong(&mydecimal, &mylong);
```

DECCVFLT

Overview

Use **deccvflt** to convert a C type **float** into a DECIMAL-type number.

Syntax

```
deccvflt(flt, np)
        float flt;
        dec__t *np;
```

Explanation

flt is a floating-point number.

np is a pointer to a **dec__t** structure that receives the result of the conversion.

Examples

```
#include <decimal.h>

dec_t mydecimal;
float myfloat;

/* Set the decimal structure
 * myfloat to 3.14159.
 */
deccvflt(3.14159, &mydecimal);

myfloat  = 123456.78;

/* Convert the variable myfloat into
 * a DECIMAL type number held in
 * mydecimal.
 */
deccvflt(myfloat, &mydecimal);
```

DECTOFLT

Overview

Use **dectoflt** to convert a DECIMAL-type number into a C type **float**.

Syntax

```
dectoflt(np, fltp)
      dec__t *np;
      float *fltp;
```

Explanation

np is a pointer to a decimal structure.

fltp is a pointer to a floating-point number to receive the result of the conversion.

Notes

1. The resulting floating-point number has eight significant digits.

Examples

```
#include <decimal.h>

dec_t mydecimal;
float myfloat;

/* convert the DECIMAL type value
 * held in the decimal structure
 * mydecimal to a floating point number pointed to
 * by myfloat.
 */
dectoflt(&mydecimal, &myfloat);
```

DECCVDBL

Overview

Use **deccvdbl** to convert a C type **double** into a DECIMAL-type number.

Syntax

```
deccvdbl(dbl, np)
      double dbl;
      dec__t *np;
```

Explanation

dbl is a double-precision, floating-point number.

np is a pointer to a **dec__t** structure that receives the result of the conversion.

Examples

```
#include <decimal.h>

dec_t mydecimal;
double  mydouble;

/* Set the decimal structure
 * mydecimal to 3.14159.
 */
deccvdbl(3.14159, &mydecimal);

mydouble = 123456.78;

/* Convert the variable mydouble into
 * a DECIMAL type number held in
 * mydecimal.
 */
deccvdbl(mydouble, &mydecimal);
```

DECTODBL

Overview

Use **dectodbl** to convert a DECIMAL-type number into a C type **double**.

Syntax

```
dectodbl(np, dblp)
      dec__t *np;
      double *dblp;
```

Explanation

np is a pointer to a decimal structure.

dblp is a pointer to a double-precision, floating-point number that receives the result of the conversion.

Notes

1. The resulting double-precision number receives a total of 16 significant digits.

Examples

```
#include <decimal.h>

dec_t mydecimal;
double mydouble;

/* convert the DECIMAL type value
 * held in the decimal structure
 * mydecimal to a double pointed to
 * by mydouble.
 */
dectodbl(&mydecimal, &mydouble);
```

DECADD, DECSUB, DECMUL, and DECDIV

Overview

The decimal arithmetic routines take pointers to three decimal structures as parameters. The first two decimal structures hold the operands of the arithmetic function. The third decimal structure holds the result.

Syntax

```
decadd(n1, n2, result)/* result = n1 + n2 */
   dec__t *n1;
   dec__t *n2;
   dec__t *result;

decsub(n1, n2, result)/* result = n1 - n2 */
   dec__t *n1;
   dec__t *n2;
   dec__t *result;

decmul(n1, n2, result)/* result = n1 * n2 */
   dec__t *n1;
   dec__t *n2;
   dec__t *result;

decdiv(n1, n2, result)/* result = n1 / n2 */
   dec__t *n1;
   dec__t *n2;
   dec__t *result;
```

Explanation

n1 is a pointer to the decimal structure of the first operand.

n2 is a pointer to the decimal structure of the second operand.

result is a pointer to the decimal structure of the result of the operation.

Notes

1. *result* can use the same pointer as either *n1* or *n2*.

Return Codes

0	Operation was successful.
−1200	Operation resulted in overflow.
−1201	Operation resulted in underflow.
−1202	Operation attempts to divide by zero.

DECCMP

Overview

Use **deccmp** to compare two DECIMAL type numbers.

Syntax

```
int deccmp(n1, n2)
    dec__t *n1;
    dec__t *n2;
```

Explanation

n1 is a pointer to the decimal structure of the first number.

n2 is a pointer to the decimal structure of the second number.

Return Codes

0	The two values are the same.
−1	The first value is less than the second.
+1	The first value is greater than the second.

DECCOPY

Overview

Use **deccopy** to copy one **dec__t** structure to another.

Syntax

```
deccopy(n1, n2)
      dec__t *n1;
      dec__t *n2;
```

Explanation

n1 is a pointer to the source **dec__t** structure.

n2 is a pointer to the destination **dec__t** structure.

DECECVT and DECFCVT

Overview

These functions convert a DECIMAL value to an ASCII string.

Syntax

```
char *dececvt(np, ndigit, decpt, sign)
      dec__t *np;
      int ndigit;
      int *decpt;
      int *sign;

char *decfcvt(np, ndigit, decpt, sign)
      dec__t *np;
      int ndigit;
      int *decpt;
      int *sign;
```

Explanation

np is a pointer to a **dec__t** structure that contains the number you want to convert.

ndigit is, for **dececvt**, the length of the ASCII string; for **decfcvt**, it is the number of digits to the right of the decimal point.

decpt points to an integer that is the position of the decimal point relative to the beginning of the string. A negative value for **decpt* means to the left of the returned digits.

sign is a pointer to the sign of the result. If the sign of the result is negative, **sign* is nonzero; otherwise, the value is zero.

Notes

1. **dececvt** converts the decimal value pointed to by *np* into a null-terminated string of *ndigit* ASCII digits and returns a pointer to the string.

2. The low-order digit of the DECIMAL number is rounded.

3. **decfcvt** is identical to **dececvt**, except that *ndigit* specifies the number of digits to the right of the decimal point instead of the total number of digits.

Examples

In the following example, *np* points to a **dec__t** structure containing 12345.67 and ***decpt** points to an integer containing a 5:

```
ptr = dececvt (np,4,&decpt,&sign);   = 1235
ptr = dececvt (np,10,&decpt,&sign);  = 1234567000
ptr = decfcvt (np,1,&decpt,&sign);   = 123457
ptr = decfcvt (np,3,&decpt,&sign);   = 12345670
```

In this example, *np* points to a **dec__t** structure containing a 0.001234 and ***decpt** points to an integer containing a −2:

```
ptr = dececvt (np,4,&decpt,&sign);   = 1234
ptr = dececvt (np,10,&decpt,&sign);  = 1234000000
ptr = decfcvt (np,1,&decpt,&sign);   =
ptr = decfcvt (np,3,&decpt,&sign);   = 1
```

Appendix L

Complex Outer Joins

Complex Outer Joins

When more than two tables participate in an outer join, there are many ways in which to relate them. For three tables, there are five logically distinct joins, of which four involve outer joins. These distinct outer joins are listed later in this appendix using the notation to be used in SELECT statements and a graphic representation that can be generalized for any number of tables. The detailed rules for the graphic representation are located at the end of this section. Briefly, the subservient tables are placed on a lower level than dominant tables.

The two-table outer join described in Chapter 2 is graphically represented as follows:

```
select a, b
from tab1, outer tab2
where a = b

tab1 ——— ∅        Level 1
          |
        tab2      Level 2
```

In the examples that follow, the tilda ~ symbol represents any Boolean relation between the columns of two tables.

Example 1

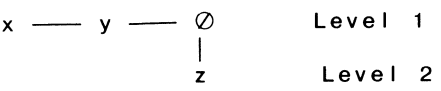
```
from x, y, z
where ...

x ——— y ——— z        Level 1
```

This is the standard inner join and all three tables are treated on the same level. There are no restrictions on the WHERE clause. It need not exist. Before the WHERE clause is applied, a full Cartesian product is made among all three tables. (**Note**: A Cartesian product of two tables appends every row of the second table to every row of the first. A Cartesian product of three tables is the Cartesian product of the third with the Cartesian product of the first two, and so on.)

Example 2

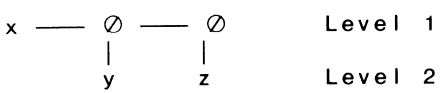
```
from x, y, outer z
where (x or y) ~ z
```



Example 2 illustrates the simplest extension of the outer join from two tables to three tables. **x** and **y** are joined at the same level and are shown connected together. **z** is the subservient table and is placed down one level. The WHERE clause must relate a Level 1 table with a Level 2 table. Other relationships are also permitted. Before the WHERE clause is applied, **RDSQL** takes the Cartesian product of **x** and **y** and then takes the outer join of the result with **z**.

Example 3

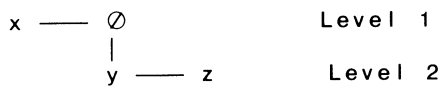
```
from x, outer y, outer z
where x ~ y and x ~ z
```



Although both **y** and **z** are at the same level, they are not connected. Here, a relationship is required between **x** and **y** and also between **x** and **z**. No relationship is permitted between **y** and **z** since they are not connected at the second level. The outer join is performed between **x** and **y** and then the outer join between the resulting table and **z**.

Example 4

```
from x, outer (y, z)
where x ~ (y or z)
```



In Example 4, a relationship is required between **x** and either **y** or **z** or both. In addition, there may be a relationship between **y** and **z** since they are connected at the second level. To perform this join, **RDSQL** starts at the top and works down. For each row of **x**, **RDSQL** attempts to find a row of **y** that satisfies the WHERE clause. If none is found, **RDSQL** substitutes NULL values for the columns of **y** and **z**. If a row of **y** satisfies the WHERE clause, **RDSQL** seeks a row of **z** that satisfies the WHERE clause. It substitutes NULL values for both **y** and **z** if no row in **z** is found.

Example 5

```
from x, outer (y, outer z)
where x ~ y and y ~ z

x ---- ∅              Level 1
       |
       y ---- ∅       Level 2
              |
              z       Level 3
```

Example 5 shows a three-level outer join. There must always be a relationship between adjacent levels. In this case, **x** must be related to **y**, and **y** must be related to **z**. **RDSQL** performs this join by picking a row from **x** and looking for a pair from **y** and **z** that satisfies (**y**, outer **z**).

If each of the tables consists of a single column with the values in the following table, then the previously described joins will result in the subsequent tables.

x.a	y.b	z.c
1	2	3
2	3	4
3	4	5
5		

Figure L-1. Initial Tables

Example 1

```
select *
from x, y, z
where x.a = z.c
```

a	b	c
3	2	3
3	3	3
3	4	3
5	2	5
5	3	5
5	4	5

Figure L-2. Example 1 Results

Example 2

```
select *
from x, y, outer z
where x.a = z.c
```

a	b	c
1	2	-
1	3	-
1	4	-
2	2	-
2	3	-
2	4	-
3	2	3
3	3	3
3	4	3
5	2	5
5	3	5
5	4	5

Figure L-3. Example 2 Results

Example 3

```
select *
from x, outer y, outer z
where x.a = y.b and x.a = z.c
```

a	b	c
1	-	-
2	2	-
3	3	3
5	-	5

Figure L-4. Example 3 Results

Example 4

```
select *
from x, outer (y, z)
where x.a = z.c
```

a	b	c
1	-	-
2	-	-
3	2	3
3	3	3
3	4	3
5	2	5
5	3	5
5	4	5

Figure L-5. Example 4 Results

Example 5

```
select *
from x, outer (y, outer z)
where x.a = y.b and y.b = z.c
```

a	b	c
1	-	-
2	2	-
3	3	3
5	-	-

Figure L-6. Example 5 Results

The following steps are useful when considering how **RDSQL** performs an outer join:

1. Draw the graph corresponding to the FROM clause. Replace each keyword OUTER with the symbol ∅ and put the table names to which the keyword applies in the next lower level.

2. Ensure that a condition exists in the WHERE clause relating a table on each level through each ∅ to a table in the next level below.

3. Form a Cartesian product of all tables connected on the same level, applying whichever conditions of the WHERE clause apply only to that level.

4. Starting with the first level, take a row at a time from the resulting table and attempt to satisfy the WHERE clause with the resulting table of the next lower level, replacing the columns of lower level table with NULL values if the WHERE clause cannot be satisfied.

As an example, the following FROM clause results in the accompanying graph:

```
from x, outer (y, outer (z, a, b)), outer c, d

    x ---- ∅ ---------- ∅ ---- d          Level 1
           |            |
           y ---- ∅     c                 Level 2
                  |
                  z ---- a ---- b        Level 3
```

A Cartesian product is made between **x** and **d** (call it **txd**) and between **z**, **a**, and **b** (call it **tzab**), since these sets of tables are connected on the same level. **y** and **c** are both on Level 2, but they are not connected on that level. For each row of **txd**, RDSQL attempts to find a row of **y** that satisfies the WHERE clause. If it succeeds, it searchs for a row of **tzab** that satisfies the WHERE clause. In both cases, NULL values are substituted if no satisfactory row is found. RDSQL then searches for a row of **c** that satisfies the WHERE clause, substituting NULL values if unsuccessful.

Appendix M

ASCII Character Set

Num	Char	Num	Char	Num	Char
0	^@	43	+	86	V
1	^A	44	,	87	W
2	^B	45	-	88	X
3	^C	46	.	89	Y
4	^D	47	/	90	Z
5	^E	48	0	91	[
6	^F	49	1	92	\
7	^G	50	2	93	]
8	^H	51	3	94	^
9	^I	52	4	95	_
10	^J	53	5	96	`
11	^K	54	6	97	a
12	^L	55	7	98	b
13	^M	56	8	99	c
14	^N	57	9	100	d
15	^O	58	:	101	e
16	^P	59	;	102	f
17	^Q	60	<	103	g
18	^R	61	=	104	h
19	^S	62	>	105	i
20	^T	63	?	106	j
21	^U	64	@	107	k
22	^V	65	A	108	l
23	^W	66	B	109	m
24	^X	67	C	110	n
25	^Y	68	D	111	o
26	^Z	69	E	112	p
27	esc	70	F	113	q
28	^\	71	G	114	r
29	^]	72	H	115	s
30	^^	73	I	116	t
31	^_	74	J	117	u
32		75	K	118	v
33	!	76	L	119	w
34	"	77	M	120	x
35	#	78	N	121	y
36	$	79	O	122	z
37	%	80	P	123	{
38	&	81	Q	124	|
39	'	82	R	125	}
40	(	83	S	126	~
41	)	84	T	127	del
42	*	85	U		

^x = CONTROL-X

Appendix N

Termcap Changes for INFORMIX-4GL

Termcap Changes for INFORMIX-4GL

INFORMIX-4GL programs may use function keys or expect color or intensity attributes in screen displays. These, and other keyboard and screen options, are terminal-dependent. INFORMIX-4GL uses the information in the **termcap** file in **/etc** to determine terminal-dependent characteristics.

Because local **termcap** files do not always exist or are not complete, RDS distributes with INFORMIX-4GL terminal characteristics for many popular terminals. These terminal characteristics include color- and intensity-change descriptions. You may want to set your TERMCAP environment variable to **$INFORMIXDIR/etc/termcap** or replace the termcap entry for your terminal in **/etc/termcap** with the corresponding entry in the file **$INFORMIXDIR/etc/termcap**.

If **$INFORMIXDIR/etc/termcap** does not describe your terminal, you should know how to modify the termcap file accordingly. The following pages sketch the changes that you or your system administrator can make.

Note: Some terminals cannot support color or graphics characters. Please read this appendix and the user guide that comes with your terminal to determine whether or not the changes described in this appendix are applicable to your terminal.

Extending Your Function Key Definitions

INFORMIX-4GL recognizes function keys F1 through F36. These correspond to the termcap variables **k0** through **k9**, followed by **kA** through **kZ**. The termcap entry for these variables is the sequence of ASCII characters that your terminal sends when you press the function keys or any other keys that you choose to use as function keys. For the Wyse 50 and the Televideo 950, the first several function keys send the characters shown in Figure N-1.

Function Key	Termcap Entry
F1	k0=^A@^M
F2	k1=^AA^M
F3	k2=^AB^M
F4	k3=^AC^M
F5	k4=^AD^M
F6	k5=^AE^M
F7	k6=^AF^M
F8	k7=^AG^M

Figure N-1. Example Termcap Entries for Function Keys

In addition to the function keys, you may define other keys corresponding to Insert Line (**ki**), Delete Line (**kj**), Next Page (**kf**), and Previous Page (**kg**). If these are defined in your **termcap** file, INFORMIX-4GL uses them. Otherwise, INFORMIX-4GL uses F1 through F4. You may also use the OPTIONS statement to name other function keys or CONTROL keys for these operations.

Specifying Characters for Window Borders

On DOS systems, INFORMIX-4GL uses graphics characters to draw the border of a window. On UNIX systems, INFORMIX-4GL uses characters defined in the **termcap** file to draw the border. If no characters are defined in this file, INFORMIX-4GL uses the hyphen (-) for horizontal lines, the vertical bar (|) for vertical lines, and the plus sign (+) for corners.

If the **termcap** file provided with INFORMIX-4GL does not define border characters for your terminal type, or you want to specify alternate border characters, you or your system administrator must modify the **termcap** file. Perform the following steps to modify the entry for your terminal type in the **termcap** file:

1. Determine the escape sequences for turning graphics mode on and off. This information should be located in the manual that comes with your terminal. For example, on Wyse 50 terminals, the escape sequence for entering graphics mode is CTRL-B, and the escape sequence for leaving graphics mode is CTRL-C.

   Note: Terminals without a graphics mode do not have this escape sequence. The procedure for specifying alternate border characters on a non-graphics terminal is included in the following steps.

2. Identify the ASCII equivalents for the six graphics characters required by INFORMIX-4GL to draw the border. (The ASCII equivalent of a graphics character is the key you would press in graphics mode to obtain the indicated character.)

   The following table shows the graphics characters and the ASCII equivalents for a Wyse 50 terminal:

	graphics character	ASCII equivalent
upper left corner	┌	2
lower left corner	└	1
upper right corner	┐	3
lower right corner	┘	5
horizontal	─	z
vertical	|	6

   (Again, this information should be located in the manual that comes with your terminal.)

3. Edit the **termcap** entry for your terminal. For example, the termcap entry in **$INFORMIXDIR/etc/termcap** for the Wyse 50 terminal starts with `w5 | wy50 | wyse50`.

4. Use the format

   termcap-variable=value

   to enter values for the following **termcap** variables:

   gs The escape sequence for entering graphics mode. In the **termcap** file, ESCAPE is represented as a backslash (\ ) followed by the letter *E*; CTRL is represented as a caret (^).

   ge The escape sequence for leaving graphics mode.

gb The concatenated, ordered list of ASCII equivalents for the six graphics characters used to draw the border. Use the following order:

upper left corner
lower left corner
upper right corner
lower right corner
horizontal lines
vertical lines

When you insert this information in the **termcap** entry, use a colon (:) to separate one **termcap** variable from another and use the backslash (\ ) at the end of a line if the **termcap** entry continues on the next line.

For example, if you are using a Wyse 50 terminal, you would insert the following information in the **termcap** entry for the Wyse 50:

```
...
:gs=EH^B:    # sets gs to ESC H CTRL B
:ge=EH^C:    # sets ge to ESC H CTRL C
:gb=2135z6:  # sets gb to the ASCII equivalents of
               # graphics characters for upper left,
               # lower left, upper right, lower right,
               # horizontal, and vertical
...
```

If you prefer, you can enter this information in a linear sequence:

```
. . . :gs=\EH^B:ge=\EH^C:gb=2135z6: . . .
```

For terminals without graphics capabilities, you must enter a blank value for the **gs** and **ge** variables, and alternate ASCII equivalents for the characters you wish INFORMIX-4GL to use in drawing the border of a window.

INFORMIX-4GL uses the graphics characters in the **termcap** file when you specify a border for a window in an OPEN WINDOW statement.

Adding Color and Intensity

There is no standard way to add color and intensity characteristics to the **termcap** file. INFORMIX-4GL recognizes the method outlined in the following pages and based on the parameterized string concepts used in the UNIX System V **terminfo** file. You should supplement this appendix by reading the UNIX manual entry for **terminfo(4)** (if available).

You may write your INFORMIX-4GL program either for a monochrome or a color terminal and then run the program on either. If you set up the **termcap** files as described here, the color attributes and the intensity attributes will be related as shown in Figure N-2.

#	color terminal	monochrome terminal
0	WHITE	NORMAL
1	YELLOW	BOLD
2	MAGENTA	BOLD
3	RED	BOLD†
4	CYAN	DIM
5	GREEN	DIM
6	BLUE	DIM†
7	BLACK	INVISIBLE

Figure N-2. Color-Monochrome Correspondence

The background for colors is BLACK in all cases. The † signifies that, if the keyword BOLD is indicated as the attribute, the field will be RED on a color terminal or, if the keyword DIM is indicated as the attribute, the field will be BLUE on a color terminal.

You may change the color names from the default list by associating different numbers with different color names in a file named **colornames** in your current directory or in **$INFORMIXDIR/incl** (See the next section for the format of the **colornames** file.)

In either color or monochrome mode, you may add the REVERSE, BLINK, or UNDERLINE attributes if your terminal supports them. You may not combine intensity with colors.

INFORMIX-4GL uses a new parameterized string variable **ZA** in the **termcap** file to determine color assignments. Unlike other **termcap** variables that you set equal to a literal sequence of ASCII characters, **ZA** is a function string that depends upon four parameters:

Parameter 1	Color number (see Figure N-2)
Parameter 2	0 = Normal; 1 = Reverse
Parameter 3	0 = No-Blink; 1 = Blink
Parameter 4	0 = No-Underline; 1 = Underline

Depending on the parameters sent to it, INFORMIX-4GL uses **ZA** to determine an appropriate sequence of characters to send to the terminal. The function string is based on a "stack machine." Typically, the instructions in the string push one or more of the parameters onto the stack, manipulate them with constants, and print out the resulting characters. Often more complex operations are necessary and, by storing the display attributes in static stack machine registers (named **a** through **z**), you can achieve terminal-specific optimizations.

Figure N-3 shows the allowed operations:

Operation	Meaning
%d	write pop() as in C's **printf** to the screen
%2d	write pop() like %2d
%3d	write pop() like %3d
%c	write pop() like %c
%s	write pop() like %s
%l	pop() a string and push() its length
%p[1-9]	push i^{th} parameter
%P[a-z]	store variable
%g[a-z]	get variable and push on stack
%'c'	push char constant
%{nn}	push integer constant
%S[a-z]	store static variable
%G[a-z]	get static variable and push
%+ %- %* %/ %m	arith. operators: push(pop() op pop())
%& %| %^	bit operators: push(pop() op pop())
%= %> %<	logical operators: push(pop() op pop())
%! %~	unary operators: push(op pop())
%? expr %t thenpart %e elsepart %;	if-then-else; the else part is optional. else-if's are possible (c's are conditions): %? c1 %t...%e c2 %t...%e c3 %t...%e...%; nested if's allowed.

all other characters are written to the terminal;
use '%%' to write '%'.

Figure N-3. Stack Machine Operations

To illustrate, consider the monochrome Qume terminal. Figure N-4 shows the escape sequences for various display characteristics:

ESC G 0	Normal
ESC G 1	Blank (invisible)
ESC G 2	Blink
ESC G 3	Blank
ESC G 4	Reverse
ESC G 5	Reverse and Blank
ESC G 6	Reverse and Blink
ESC G 7	Reverse, Blink, and Blank
ESC G 8	Underline
ESC G 9	Underline and Blank
ESC G :	Underline and Blink
ESC G ;	Underline, Blink, and Blank
ESC G <	Underline and Reverse
ESC G =	Underline, Reverse, and Blank
ESC G >	Underline, Reverse, and Blink
ESC G ?	Underline, Reverse, Blink, and Blank

Figure N-4. Qume Display Escape Sequences

The characters after G form an ASCII sequence from the character 0 (zero) through ?. You can generate the character by starting with 0 and adding 1 for Blank, 2 for Blink, 4 for Reverse, and 8 for Underline.

In addition to these characteristics, the Qume supports half intensity with the sequence ESC) and a return to full intensity with ESC (.

You can construct the **termcap** entry in stages, as outlined in the following display. pi refers to the i^{th} parameter. The designation for ESC is \E.

```
ZA =
%'0'                              #push '0' (normal) on the stack
%?%p1%{7}%=%t%{1}%+%;             #if p1 = 7 (invisible), add 1
%?%p1%{3}%>%t\E)%\E(%;            #if p1 > 3, print eE) (dim);
                                  #  else print \E( (normal)
%?%p2%t%{4}%+%;                   #if p2 is set, add 4 (reverse)
%?%p3%t%{2}%+%;                   #if p3 is set, add 2 (blink)
%?%p4%t%{8}%+%;                   #if p4 is set, add 8 (underline)
\EG%c                             #print \EG and whatever character
                                  #  is on top of the stack
```

You must concatenate these lines as a single string with no NEWLINEs embedded. The actual **termcap** entry for the Qume follows (where you can assume the single line has wrapped):

```
ZA=%'0'%?%p1%{7}%=%t%{1}%+%;%?%p1%{3}%>%t\E)%\E(%;
%?%p2%t%{4}%+%;%?%p3%t%{2}%+%;%?%p4%t%{8}%+%;\EG%c:
```

The next example is for the ID Systems Corporation ID231, a color terminal. To set color and other characteristics on this terminal, you must enclose a character sequence between a lead-in sequence (ESC [0) and a terminating character (m). The first in the sequence is a two-digit number that determines whether the assigned color is in the background (30) or in the foreground (40). The next is another two-digit number that is the other of 30 or 40, incremented by the color number. These characters are followed by 5 if there is blinking and also by 4 for underlining. The following code sets up the entire escape sequence:

```
ZA =
\E[0;                          #print lead-in
%?%p1%{0}%=%t%{7}              #encode color number (translate
%e%p1%{1}%=%t%{3}              #    from Figure N-2 to the number
%e%p1%{2}%=%t%{5}              #    for the ID231)
%e%p1%{3}%=%t%{1}              #
%e%p1%{4}%=%t%{6}              #
%e%p1%{5}%=%t%{2}              #
%e%p1%{6}%=%t%{4}              #
%e%p1%{7}%=%t%{0}%;            #
%?%p2%t30;%{40}%+%2d           #if p2 is set, print '30' and
                               #  '40' + color number (reverse)
%e40;%{30}%+%2d%;              #  else print '40' and
                               #  '30' + color number (normal)
%?%p3%t;5%;                    #if p3 is set, print 5 (blink)
%?%p4%t;4%;                    #if p4 is set, print 4 (underline)
m                              #print 'm' to end character
                               #  sequence
```

When you concatenate these strings, the **termcap** entry is as follows:

```
ZA =\E[0;%?%p1%{0}%=%t%{7}%e%p1%{1}%=%t%{3}%e%p1%{2}%=
%t%{5}%e%p1%{3}%=%t%{1}%e%p1%{4}%=%t%{6}%e%p1%{5}%=%t%
{2}%e%p1%{6}%=%t%{4}%e%p1%{7}%=%t%{0}%;%?%p2%t30;%{40}
%+%2d%e40;%{30}%+%2d%;%?%p3%t;5%;%?%p4%t;4%;m
```

The last example, for the Envision 215, optimizes the output by using static variables to remember the current state and printing characters only when necessary. In logical segments, this reads

```
ZA=
%?%p3%t%{1}%;                  # if p3 (blink) is set, pop(1)
%?%p4%t%{2}%+%;                # if p4 (underline) is set, pop(2),
%Sz                            #    add, and store in z;
%?%Gz%Ga%=%!%t                 # if attribute changed, then print
      \E[0                     #    attributes:
      %?%p3%t;5%;              #       \E[05m  (blink)
      %?%p4%t;4%;              #       \E[04m  (underline)
      m%;                      #
  %Gz%Sa                       # save attributes in static a
  %?%p1%{0}%=%t%{7}            # encode color number (substitute
  %e%p1%{1}%=%t%{3}            #    values for Envision 215)
  %e%p1%{2}%=%t%{5}            #
  %e%p1%{3}%=%t%{1}            #
  %e%p1%{4}%=%t%{6}            #
  %e%p1%{5}%=%t%{2}            #
  %e%p1%{6}%=%t%{4}            #
  %e%p1%{7}%=%t%{0}%;          #
  %'0'%+%Sc                    # add to '0';store in static c
  %p1%{7}                      # push p1 (foreground) and 7
                               #   (background)
%?%p2%t%Sx%Sy%Gc%'0'           # foreground in x / background in y
                               # push color then '0' (black)
   %e%Sy%Sx%'0'%Gc%;           # background in x / foreground in y
                               # push '0' (black) then color
%?%Gx%Gf%=%!%t\Ea%c            # if foreground changed, output
  %e%Sd%;                      #  black if reversed, color if not
%?%Gy%Gb%=%!%t\Eb%c            # if background changed, output
  %e%Sd%;                      #  color if reversed, black if not
%Gx%Sf%Gy%Sb                   # save foreground and background
```

Concatenated, these strings produce the following string:

```
ZA=%?%p3%t%{1}%;%?%p4%t%{2}%+%;%Sz%?%Gz%Ga%=%!%t\E[0%?
%p3%t;5%;%?%p4%t;4%;m%;%Gz%Sa%?%p1%{0}%=%t%{7}%e%p1%{1
}%=%t%{3}%e%p1%{2}%=%t%{5}%e%p1%{3}%=%t%{1}%e%p1%{4}%=
%t%{6}%e%p1%{5}%=%t%{2}%e%p1%{6}%=%t%{4}%e%p1%{7}%=%t%
{0}%;%'0'%+%Sc%p1%{7}%?%p2%t%Sx%Sy%Gc%'0'%e%Sy%Sx%'0'%
Gc%;%?%Gx%Gf%=%!%t\Ea%c%e%Sd%;%?%Gy%Gb%=%!%t\Eb%c%e%Sd
%;%Gx%Sf%Gy%Sb
```

In addition to the **ZA** variable, there are two other **termcap** entries that you can use. **ZG** is the number of character positions on the screen occupied by the attributes of **ZA**. It is analogous to the **SG** entry.

The entry, if present, indicates that you are using the standard color scheme and that curses can assume that all spaces are the same. This condition enhances the optimization of color performance.

Format of colornames File

Create a **colornames** file if you want to change the default assignment of the names of colors. A **colornames** file changes the keywords that you use to write INFORMIX-4GL programs; it does not affect the colors produced by your terminal. The format for the **colornames** file follows:

```
        name            number
        ...             ...
```

Explanation

name is the identifier of a color.

number is an integer from 0 to 7.

Notes

1. Each color name and number must be on a separate line. They should be separated by one or more spaces or tabs.

2. *name* must be unique in the **colornames** file. You cannot assign the same name to more than one number.

3. Unless you redefine them in **colornames** to have a different number, the default color-name keywords of Chapter 3 retain their meaning even when you assign another name to that color number.

Examples

If you created a **colornames** file to set up the default assignment of names to color numbers, **colornames** would look as follows:

```
WHITE          0
YELLOW         1
MAGENTA        2
RED            3
CYAN           4
GREEN          5
BLUE           6
BLACK          7
```

If you wanted to change CYAN to AQUA and MAGENTA to ORANGE as color names, set **colornames** as follows:

```
AQUA           4
ORANGE         2
```

You could use either CYAN or AQUA in your INFORMIX-4GL program and get the same color. Similarly, use of MAGENTA or ORANGE produces the same color.

If you want to change the meaning of the default color names, you can reassign them in **colornames**:

```
RED            2
```

In this case when you use RED in a program, the color you get is the same as has been assigned to MAGENTA. If you have not assigned a name to number 3, you are not able to get the color that RED originally represented.

Appendix O

The *dbschema* Utility

The *dbschema* Utility

The Database Administrator (DBA) can use the **dbschema** utility to quickly produce an RDSQL command file containing the CREATE TABLE, CREATE INDEX, and CREATE VIEW statements required to replicate an entire database or a selected table. In addition, **dbschema** can produce all CREATE SYNONYM and GRANT statements in effect for the database or a selected table or view.

By default, **dbschema** produces all CREATE TABLE, CREATE VIEW, CREATE INDEX, CREATE SYNONYM, and GRANT statements in effect for the entire database. By including the appropriate command line option, you can limit the output to a selected table or view, or produce synonyms and permissions for a particular user.

The syntax for the **dbschema** utility appears as follows:

---

dbschema [**-t** *tabname*] [**-s** *sname*] [**-p** *pname*] **-d** *database* [*filename*]

-t *tabname* identifies *tabname* as the table or view for which **dbschema** outputs the CREATE TABLE and CREATE INDEX statements or the CREATE VIEW statement. If *tabname* is all, the RDSQL statements for all database tables and views are output.

-s *sname* identifies *sname* as the user for which **dbschema** outputs the CREATE SYNONYM statements. If *sname* is all, the RDSQL statements for all users are output. If you include the **-t** option, **dbschema** produces the CREATE SYNONYM statements only for the indicated *table-name*.

-p *pname* identifies *pname* as the user for which **dbschema** outputs permissions. If *pname* is all, the RDSQL statements necessary to grant permissions to all users are output. If you include the **-t** option, **dbschema** produces the GRANT statements only for the indicated *table-name*.

-d *database* is the name of the database.

filename is the name of the file receiving the RDSQL statements. If *filename* is not provided, **dbschema** writes to the terminal screen.

Notes

1. The default command line appears as follows and produces the RDSQL statements necessary to replicate an entire database:

```
dbschema -d database
```

2. When you include the **-t** option, **dbschema** produces RDSQL statements only for the indicated *table-name*. **dbschema** uses the *table-name* to filter the output. If you request CREATE SYNONYM and GRANT statements, these are provided only for *table-name*.

3. Most statements appear with comments. Depending upon the information requested, **dbschema** indicates the owner of the table, owner of the synonym, or grantor of a permission.

4. All SERIAL fields included in CREATE TABLE statements output by **dbschema** have a starting value of one, regardless of their original starting value.

5. The *database* must exist in your current directory or a directory cited in your DBPATH environment variable.

Examples

The following statement outputs the **RDSQL** statements relating to the **customer** table in the **stores** database to the *c_all* file.

```
dbschema -t customer -s nancy -p bettyjo -d stores c_a
```

The output consists of the following statements:

- The CREATE TABLE and CREATE INDEX statements replicating the **customer** table
- All CREATE SYNONYM statements executed by the user *nancy* on the **customer** table
- All permissions granted to the user *bettyjo* on the **customer** table

The output from the **dbschema** statement follows:

```
{enid is owner of table customer}
create table customer
(
custnum serial not null,
fname char(15)
lname char(15),
company char(20),
address1 char(20),
address2 char(20),
city char(15),
state char(2),
zipcode char(5),
phone char(18)
);

{enid is owner of index c_num_ix}
create unique index c_num_ix on customer (custnum);

{enid is owner of index zip_ix}
create index zip_ix on customer (zipcode);

revoke all on customer from public;

{nancy is owner of synonym cust}
create synonym cust for customer;

{felix is the grantor}
grant all on customer to bettyjo;
```

The next **dbschema** statement outputs the **RDSQL** statements relating to all tables in the database to the **s_out** file.

```
dbschema -t all -s felix -p felix -d stores s_out
```

The output consists of the following statements:

- The CREATE TABLE, CREATE VIEW, and CREATE INDEX statements that replicate all tables, views, and indexes in the **stores** database
- All CREATE SYNONYM statements executed by the user *felix*
- All permissions granted to the user *felix*

INFORMIX-4GL Error Messages

This section contains the text of the INFORMIX-4GL error messages. It also describes how the system responds to each error and suggests corrective actions.

All INFORMIX-4GL errors include an error number. With the error number you can quickly locate the message in this section. Error messages with negative numbers appear in order, starting with −100. Error messages with positive numbers (if any) are placed at the back of the section.

-100 **Description of Error**: C-ISAM error: there is already a record with the same value in a unique key.

System Action Taken: The statement was not processed.

Corrective Action: Check that you did not attempt to add a duplicate value to a column with a unique index via *iswrite*, *isrewrite*, *isrewcurr*, or *isaddindex*.

-101 **Description of Error**: C-ISAM error: file is not open.

System Action Taken: The statement was not processed.

Corrective Action: Check that the C-ISAM file has been opened using the *isopen* call, or that you have not tried to write to a C-ISAM file opened for read only.

-102 **Description of Error**: C-ISAM error: illegal argument to C-ISAM function.

System Action Taken: The statement was not processed.

Corrective Action: Check that one of the arguments to the C-ISAM call is not outside of the range of acceptable values for that argument.

-103 **Description of Error**: C-ISAM error: illegal key descriptor (too many parts or too long).

System Action Taken: The statement was not processed.

Corrective Action: Check that one or more of the elements that make up the key description is not outside of the range of acceptable values for that element. (There is a maximum of 8 parts and 120 characters to each key descriptor.)

-104 **Description of Error**: C-ISAM error: too many files open.

System Action Taken: The statement was not processed.

Corrective Action: The maximum number of files that may be open at one time would be exceeded if this request were processed. Reduce the number of open files by breaking your program up into two or more parts. On UNIX systems (and most DOS systems), the maximum number of open files per process is 20. If running under DOS, check that your **CONFIG.SYS** file contains the line

```
FILES=20
```

-105 **Description of Error**: C-ISAM error: bad ISAM file format.

System Action Taken: The statement was not processed.

Corrective Action: The format of the C-ISAM file has been corrupted. Run the **bcheck** utility on the file, and it will try to repair damaged indexes. If **bcheck** cannot repair the file, you will need to reload your data from a backup medium.

-106 **Description of Error**: C-ISAM error: non-exclusive access.

System Action Taken: The statement was not processed.

Corrective Action: You must first open the file with exclusive access when you add or delete an index.

-107 **Description of Error**: C-ISAM error: record is locked.

System Action Taken: The statement was not processed.

Corrective Action: The record or file requested by this call cannot be accessed because it has been locked by another user. Wait a moment and re-enter your request.

If you are certain that the table is not in use and your system uses *tablename.lok* files, you may need to empty the contents of this file. (This file contains information about which rows of the table are being used at any one time. It is normally emptied when a user finishes accessing the table. On occasion, the file is not emptied and, as a result, no one will be able to use the table.) You can copy the "file" **/dev/null** into this file to remove all locks on rows in the table. Contact your System Administrator about this action.

-108 **Description of Error**: C-ISAM error: key already exists.

System Action Taken: The statement was not processed.

Corrective Action: You have attempted to add an index that previously has been defined. You will need to delete this existing index before defining another.

-109 **Description of Error**: C-ISAM error: the key is the file's primary key.

System Action Taken: The statement was not processed.

Corrective Action: An attempt was made to delete the primary key column. The primary key may not be deleted by the *isdelindex* call.

-110 **Description of Error**: C-ISAM error: end or beginning of the file.

System Action Taken: The statement was not processed.

Corrective Action: The beginning or end of the file was reached.

-111 **Description of Error**: C-ISAM error: no record found.

System Action Taken: The statement was not processed.

Corrective Action: No record could be found that contained the requested value in the specified position. Edit your request and re-enter.

-112 **Description of Error**: C-ISAM error: there is no current record.

System Action Taken: The statement was not processed.

Corrective Action: An attempt to access a record in the current list has been made, but there is no current list. You must first perform a query and obtain a current list.

-113 **Description of Error**: C-ISAM error: the file is locked.

System Action Taken: The statement was not processed.

Corrective Action: The table you wish to alter is currently being used by another user in exclusive mode. Wait until the table is no longer being used before entering your request.

If you are certain that the table is not in use, you should run the RDSQL UNLOCK TABLE command to unlock the table. Also, if your system contains *tablename.lok* files, you may need to empty the contents of this file. (This file contains information about which rows of the table are being used at any one time. It is normally emptied when a user finishes accessing the table. On occasion, the file is not emptied and, as a result, no one will be able to use the table.) You can copy the "file" **/dev/null** into this file to remove all locks on rows in the table. Be certain no processes are accessing the locked table before emptying the *tablename.lok* file. Contact your System Administrator about this action.

-114 **Description of Error**: C-ISAM error: the file name is too long.

System Action Taken: The statement was not processed.

Corrective Action: Reduce the filename length to eight or fewer characters (if using DOS) or ten or fewer characters (if using UNIX).

-116 **Description of Error**: C-ISAM error: cannot allocate memory.

System Action Taken: The statement was not processed.

Corrective Action: Insufficient memory is available to run your request. (INFORMIX-4GL has run out of addressable data space.) You need to reduce the complexity of your statement or form.

-118 **Description of Error**: Cannot read transaction log record.

System Action Taken: The statement containing the error was not processed.

Corrective Action: The transaction log record is corrupted and cannot be used. You should CLOSE the DATABASE, execute a START DATABASE WITH LOG IN statement, and backup the database.

-119 **Description of Error**: Bad log record.

System Action Taken: The statement containing the error was not processed.

Corrective Action: The transaction log record is corrupted and cannot be used. You should CLOSE the DATABASE, execute a START DATABASE WITH LOG IN statement, and backup the database.

-120 **Description of Error**: Cannot open log file.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Check that the log file exists, that you are using the correct pathname, and that you have operating system read and write permissions on the file.

-121 **Description of Error**: Cannot write log file record.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Check that you have operating system write permission on the file and that sufficient disk space is available to add to the file.

-122 **Description of Error**: BEGIN WORK encountered in a database without transactions.

System Action Taken: The statement containing the error was not processed.

Corrective Action: You can only use the BEGIN WORK statement in a database with transactions. Make sure that you have created or started the database with transactions.

-123 **Description of Error**: No shared memory.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Check that the Database Administrator has set up the shared memory partition.

-124 **Description of Error**: No BEGIN WORK found.

System Action Taken: The statement containing the error was not processed.

Corrective Action: You must execute a BEGIN WORK statement before you can execute a COMMIT WORK or ROLLBACK WORK statement.

-125 **Description of Error**: Cannot use NFS.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Do not attempt to access remote files on the network using NFS.

-126 **Description of Error**: Audit trail exists.

System Action Taken: The statement containing the error was not processed.

Corrective Action: You cannot specify a new audit trail on a table without first dropping the current one.

-200 **Description of Error**: Identifier is too long.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Identifiers must be 18 characters or less. Select a new identifier of the appropriate length.

-201 **Description of Error**: A syntax error has occurred.

System Action Taken: The statement containing the syntax error was not processed.

Corrective Action: Check that you have not misspelled an RDSQL statement, placed key words out of sequence, or included an INFORMIX-4GL reserved word in your query.

-202 **Description of Error**: An illegal character has been found in the statement.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Remove the illegal character (often a non-printable control character) and resubmit the statement.

-203 **Description of Error**: An illegal integer has been found in the statement.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Integers must be whole numbers from -2,147,483,647 to 2,147,483,647. Check that you have not included a number with a fractional portion or a number outside of the range of acceptable whole numbers. Check also that you have not inadvertently entered a letter in place of a number (for example, 125p3 instead of 12503).

-204 **Description of Error**: An illegal floating-point number has been found in the statement.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Check that you have not inadvertently entered a letter in place of a number (for example, 125b3 in place of 12503).

-205 **Description of Error**: Cannot use ROWID for views.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Restructure your statement so that the virtual column is not used in defining the view.

-206 **Description of Error**: The specified table name is not in the database.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Check the spelling of the table name in your statement. Check the *systables* system catalog for a list of all database tables.

-207 **Description of Error**: Cannot update cursor declared on more than one table.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Check that you have not attempted to use a FOR UPDATE clause with cursors on multiple tables. Restructure your update statement, perhaps using multiple cursors.

-208 **Description of Error**: Memory allocation failed during query processing.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Reduce the complexity of your query or program.

-209 **Description of Error**: Incompatible database format.

System Action Taken: The statement was not processed.

Corrective Action: You are attempting to use INFORMIX-4GL with a database built by INFORMIX-SQL or INFORMIX-ESQL/C 1.1. You must first run **dbupdate** on your database, which will prepare the database for INFORMIX-4GL.

-210 **Description of Error**: Pathname too long.

System Action Taken: The statement containing the error was not processed.

Corrective Action: INFORMIX-4GL will accept a pathname of up to 70 total characters. Reduce the length of the pathname.

-211 **Description of Error**: Cannot read system catalog *catalog-name*.

System Action Taken: See below for a list of system actions.

Corrective Action: Check the C-ISAM error for information about the source of the problem. Depending upon the content of the statement and the system catalog cited in the error message, the following actions have occurred:

For a CREATE TABLE statement: the **systabauth** catalog was not read, the table was created, but no authorizations are granted to PUBLIC.

For a DROP TABLE statement: if the **systables** catalog was not read, then no action was taken; if the **sysviews** catalog was not read, then the table was dropped but any views built on the table may not have been dropped.

For a DROP VIEW statement: the **sysviews** catalog was not read, and no action was taken.

For a DROP INDEX statement: the **sysindexes** or **systables** catalogs was not read, and the index was not dropped.

For a DROP SYNONYM statement: the **syssynonym** catalog was not read, and the synonym was not dropped.

For a DROP DATABASE statement: the **systables** catalog was not read, and the database was not dropped.

For a START DATABASE statement: the **systables** catalog was not read, and the database was not started.

For a DATABASE statement: the **systables** or **sysusers** catalog was not read, and the database was not selected.

-212 **Description of Error**: Cannot add index.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Check the C-ISAM error for information about the source of the problem.

-213 **Description of Error**: Statement interrupted by user.

System Action Taken: The statement was not processed.

Corrective Action: INFORMIX-4GL has received an interrupt signal (probably due to the user pressing the DEL or CONTROL-C key). Resubmit your statement.

-214 **Description of Error**: Cannot remove file for table *tablename*.

System Action Taken: If this is a DROP DATABASE statement, then some tables may have been dropped from the database. If this is a DROP TABLE statement, then some system entries for the table may have been dropped from the database.

Corrective Action: INFORMIX-4GL cannot remove one or more of the entries in the system catalogs for the table. Check the C-ISAM error for information about the source of the problem. Check with the System Administrator about remedial actions.

-215 **Description of Error**: Cannot open file for table *tablename*.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Check the C-ISAM error for information about the source of the problem.

-216 **Description of Error**: Cannot remove ISAM index on file.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Check the C-ISAM error for information about the source of the problem.

-217 **Description of Error**: Column *column-name* not found in any table in the query.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Correct the spelling of the column name or that column name exists in database table. Check for the presence of required commas and quotes.

-218 **Description of Error**: Synonym *name* not found.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Check the spelling of the synonym. If needed, check the **syssynonyms** system catalog for a list of available synonyms.

-219 **Description of Error**: Wildcard matching may not be used with non-character types.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Wildcards (*, ?) and characters enclosed in brackets [ ] can be used only with CHAR data types. Check the data type for the offending column.

-220 **Description of Error**: There is no FROM clause in the query.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Must include a FROM clause in the query. Check that you do not have an illegal character ($, #, &, etc., or a CONTROL character) in the line prior to the FROM keyword.

-221 **Description of Error**: Cannot build temporary file for new table *table-name*.

System Action Taken: The statement containing the error was not processed.

Corrective Action: C-ISAM cannot access the temporary directory (usually, **/tmp** on UNIX systems, or the current directory on DOS systems) or the disk may be out of space. Check the C-ISAM error for information about the source of the problem.

-222 **Description of Error**: Cannot write to temporary file for new table *table-name*.

System Action Taken: The statement containing the error was not processed.

Corrective Action: The disk may be out of space. Check the C-ISAM error for information about the source of the problem.

-223 **Description of Error**: Duplicate table name *table-name* in the FROM clause.

System Action Taken: The statement did not run.

Corrective Action: Remove the redundant table name from the statement or use an alias to rename one of the tables.

-224 **Description of Error**: Cannot open log file.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Check the C-ISAM error for information about the source of the problem.

-225 **Description of Error**: Cannot create file for system catalog *catalog-name.*

System Action Taken: The CREATE DATABASE statement was not completed. Some system catalogs may have been created.

Corrective Action: Check the C-ISAM error for information about the source of the problem.

-226 **Description of Error**: Cannot create index for system catalog *catalog-name.*

System Action Taken: The CREATE DATABASE statement was not completed. Some system catalogs may have been created.

Corrective Action: Check the C-ISAM error for information about the source of the problem.

-227 **Description of Error**: Cannot use ORDER BY clause when selecting into temporary table.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Remove the ORDER BY clause from your statement. Place an index on the column you wish to order by after creating the temporary table.

-228 **Description of Error**: Cannot have negative characters.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Check that you have not included a negative CHAR data type (for example, a **–a** or **–p**) in your statement.

-229 **Description of Error**: Could not open or create a temporary file.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Check the C-ISAM error for information about the source of the problem.

-230 **Description of Error**: Could not read a temporary file.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Check the C-ISAM error for information about the source of the problem.

-231 **Description of Error**: Cannot perform aggregate function with distinct on expression.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Select the expression into a temporary table and then perform an aggregate distinct on the temporary table.

-232 **Description of Error**: A SERIAL column *column-name* may not be updated.

System Action Taken: The statement containing the error was not processed.

Corrective Action: The values appearing in a SERIAL column are provided by INFORMIX-4GL and may not be updated.

-233 **Description of Error**: Cannot read record that is locked by another user.

System Action Taken: Your request was not processed.

Corrective Action: Another user has locked the record. Wait a moment and re-enter your request.

-234 **Description of Error**: Cannot insert into virtual column *column-name*.

System Action Taken: The statement containing the error was not processed.

Corrective Action: The specified column is derived from an expression or an aggregate function. Redefine the view.

-235 **Description of Error**: Character column size too big. The maximum size is 32767.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Redefine the size of the column to 32767 characters or less.

-236 **Description of Error**: Number of columns in INSERT does not match number of VALUES.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Check that the number of columns in the table or in the column list matches the number of values in the VALUES clause or the SELECT clause.

-237 **Description of Error**: Cannot begin work.

System Action Taken: Your request was not processed.

Corrective Action: Check the C-ISAM error number for information about the source of the problem. Contact your System Administrator or Database Administrator if you need assistance with interpreting the C-ISAM error number.

-238 **Description of Error**: Cannot COMMIT WORK.

System Action Taken: Your request was not processed.

Corrective Action: Your log file may be corrupted. Check the C-ISAM error number for information about the source of the problem. Contact your System Administrator or Database Administrator if you need assistance with interpreting the C-ISAM error number.

-239 **Description of Error**: Could not insert new row—duplicate value in a UNIQUE INDEX column.

System Action Taken: The statement containing the error was not processed.

Corrective Action: The row contains a value which already exists in the column (indexed as unique) of an existing row. Enter a new value for the column or remove the unique index on the column.

-240 **Description of Error**: Could not delete a row.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Check the C-ISAM error for information about the source of the problem.

-241 **Description of Error**: Cannot ROLLBACK WORK.

System Action Taken: Your request was not processed.

Corrective Action: Check the C-ISAM error number for information about the source of the problem. Contact your System Administrator or Database Administrator if you need assistance interpreting the C-ISAM error number.

-242 **Description of Error**: Could not open database table *table-name.*

System Action Taken: The statement containing the error was not processed.

Corrective Action: Check the C-ISAM error for information about the source of the problem.

-243 **Description of Error**: Could not position within a table *table-name.*

System Action Taken: The statement containing the error was not processed.

Corrective Action: Check the C-ISAM error for information about the source of the problem.

-244 **Description of Error**: Could not do a physical-order read to fetch next row.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Check the C-ISAM error for information about the source of the problem.

-245 **Description of Error**: Could not position within a file via an index.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Check the C-ISAM error for information about the source of the problem.

-246 **Description of Error**: Could not do an indexed read to get the next row.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Check the C-ISAM error for information about the source of the problem.

-247 **Description of Error**: ROLLFORWARD database failed.

System Action Taken: The statement was not processed.

Corrective Action: Check the C-ISAM error for information about the source of the problem. Contact your System Administrator or Database Administrator if you need assistance with interpreting the meaning of the C-ISAM error.

-248 **Description of Error**: Value of the program variable in the WHERE clause is NULL.

System Action Taken: The statement was not processed.

Corrective Action: Use IS [NOT] NULL for NULL operation or initialize the program variable.

-249 **Description of Error**: Virtual column must have explicit name.

System Action Taken: The statement containing the error was not processed.

Corrective Action: When selecting into a temporary table or creating a view, each temporary or view column based on an expression must be given a unique name. Check also that distinct names are provided.

-250 **Description of Error**: Cannot read record from file for update.

System Action Taken: The statement containing the error was not processed.

Corrective Action: The record may be locked by another user. Check the C-ISAM error for information about the source of the problem.

-251 **Description of Error**: Column number *number* too big.

System Action Taken: The statement containing the error was not processed.

Corrective Action: The number of the column noted in your ORDER BY or GROUP BY statement exceeds the total number of columns in the SELECT statement.

-252 **Description of Error**: Cannot get system information for table.

System Action Taken: Some statistics may be updated.

Corrective Action: Check the C-ISAM error for information about the source of the problem.

-253 **Description of Error**: Identifier too long—maximum length is 18.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Check the spelling or length of the table name.

-254 **Description of Error**: Too many or too few host variables given.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Check that the number of program variables in the fetch is equal to the number used when defining the cursor.

-255 **Description of Error**: Not in transaction.

System Action Taken: The statement containing the error was not processed.

Corrective Action: The statement must be executed within a transaction. First start a transaction, then perform the statement.

-256 **Description of Error**: Transaction not available.

System Action Taken: The statement containing the error was not processed.

Corrective Action: INFORMIX-4GL cannot perform a transaction operation (BEGIN WORK, ROLLBACK WORK, COMMIT WORK) on the database because a transaction log was never created for the database. Ask your Database Administrator to create a transaction log for the database.

-257 **Description of Error**: System limit on cursors exceeded, maximum is *num*.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Reduce the number of cursors used in the program.

-258 **Description of Error**: System error—invalid statement ID received by the **sqlexec** process.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Make sure that the cursor or object is defined after the DATABASE statement. Call the Technical Support Department at RDS if you need additional assistance.

-259 **Description of Error**: Cursor not open.

System Action Taken: The statement containing the error was not processed.

Corrective Action: First open a cursor, then attempt the fetch.

-260 **Description of Error**: Cannot execute a SELECT statement that is PREPARED—must use cursor.

System Action Taken: The statement was not processed.

Corrective Action: Use a cursor for the SELECT statement.

-261 **Description of Error**: Could not create file for table *table*.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Check the C-ISAM error for information about the source of the problem.

-262 **Description of Error**: There is no current cursor.

System Action Taken: The statement containing the error was not processed.

Corrective Action: You cannot perform an UPDATE or DELETE action when no current row exists. First perform a fetch, then attempt this action.

-263 **Description of Error**: Could not lock row for UPDATE.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Check the C-ISAM error for information about the source of the problem.

-264 **Description of Error**: Could not write to a temporary file.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Check the C-ISAM error for information about the source of the problem.

-265 **Description of Error**: Load or insert cursors must be run within a transaction.

System Action Taken: The statement containing the error was not processed.

Corrective Action: On databases with transactions, you must execute a BEGIN WORK statement before using an insert cursor.

-266 **Description of Error**: There is no current row for UPDATE/DELETE cursor.

System Action Taken: The statement containing the error was not processed.

Corrective Action: You cannot perform an UPDATE or DELETE action when no current row exists. First perform a fetch, then attempt this action.

-267 **Description of Error**: The cursor has been previously released and is unavailable.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Make sure that the cursor is open before you make reference to it.

-268 **Description of Error**: Cannot use SELECT DISTINCT with UNION ALL.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Rewrite your statement.

-269 **Description of Error**: Cannot add column *column-name* that does not accept nulls.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Rewrite your statement.

-270 **Description of Error**: Could not position within a temporary file.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Check the C-ISAM error for information about the source of the problem.

-271 **Description of Error**: Could not insert new row into the table.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Check the C-ISAM error for information about the source of the problem.

-272 **Description of Error**: No SELECT permission.

System Action Taken: The statement was not processed.

Corrective Action: Request permission to SELECT from the owner of the table.

-273 **Description of Error**: No UPDATE permission.

System Action Taken: The statement was not processed.

Corrective Action: Request permission to UPDATE from the owner of the table.

-274 **Description of Error**: No DELETE permission.

System Action Taken: The statement was not processed.

Corrective Action: Request permission to DELETE from the owner of the table.

-275 **Description of Error**: No INSERT permission.

System Action Taken: The statement was not processed.

Corrective Action: Request permission to INSERT from the owner of the table.

-276 **Description of Error**: Cursor not found.

System Action Taken: The statement was not processed.

Corrective Action: You have called for a cursor that has not been declared in the current session. (The current session runs from the issuance of a DATABASE statement to a CLOSE DATABASE statement, or the issuance of the next DATABASE statement.) Declare your cursor within the current (database) session.

-277 **Description of Error**: UPDATE table *table-name* is not the same as the cursor table.

System Action Taken: The statement was not processed.

Corrective Action: Either declare a cursor on the table used in the UPDATE statement, or update the table used in the cursor. Check the spelling of the table name and cursor name.

-278 **Description of Error**: Too many ORDER BY columns; maximum is 8.

System Action Taken: The statement was not processed.

Corrective Action: Reduce the number of columns included in the ORDER BY clause to 8 or less.

-279 **Description of Error**: Cannot GRANT or REVOKE database privileges for table or view.

System Action Taken: The statement was not processed.

Corrective Action: Database privileges (CONNECT, RESOURCE, and DBA) cannot be granted on individual tables.

-280 **Description of Error**: Total size of ORDER BY columns exceeds 120 bytes.

System Action Taken: The statement was not processed.

Corrective Action: Reduce the number of columns included in the ORDER BY clause so that the total number of characters is less than or equal to 120 (perhaps delete a CHAR column of 30 or more characters).

-281 **Description of Error**: Could not add index to a temporary table.

System Action Taken: The statement was not processed.

Corrective Action: Check the C-ISAM error for information about the source of the problem.

-282 **Description of Error**: Found a quote for which there is no matching quote.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Check that all quoted strings are properly terminated with a quote.

-283 **Description of Error**: Found a non-terminated comment ("{" with no matching "}").

System Action Taken: The statement was not processed.

Corrective Action: Check that all comments are properly enclosed in braces. (Comments cannot be nested.)

-284 **Description of Error**: A subquery has returned not exactly one value.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Check data for the subquery. Restructure the subquery by adding more components in the WHERE clause so that only one value is returned.

-285 **Description of Error**: Invalid cursor received by **sqlexec**.

System Action Taken: The statement was not processed.

Corrective Action: First issue a PREPARE statement within the current session, then re-enter your original statement.

-286 **Description of Error**: An expression may not include ANY or ALL.

System Action Taken: The statement containing the error was not processed.

Corrective Action: ANY and ALL can only be used in association with a subquery.

-287 **Description of Error**: Cannot add SERIAL column *column-name* to the table.

System Action Taken: The statement containing the error was not processed.

Corrective Action: A SERIAL column does not accept NULL values. Add the column to the table as type INTEGER, UPDATE it so that there are no NULLS, and then MODIFY it to type SERIAL.

-288 **Description of Error**: Table *table-name* not locked by current user.

System Action Taken: The statement containing the error was not processed.

Corrective Action: You cannot unlock a table locked by another user. Only the user who locked the table (or the DBA) can unlock the table.

-289 **Description of Error**: Cannot lock table *table-name* in share mode.

System Action Taken: The statement containing the error was not processed.

Corrective Action: The table is already locked in exclusive mode. Wait until the table is unlocked before reexecuting your request.

-290 **Description of Error**: Cursor not declared with FOR UPDATE clause.

System Action Taken: The statement containing the error was not processed.

Corrective Action: You must first declare a cursor with FOR UPDATE if the cursor is used for updating the database.

-291 **Description of Error**: Table *table-name* is already locked.

System Action Taken: The statement containing the error was not processed.

Corrective Action: You must first unlock the table before executing your request.

-292 **Description of Error**: An implied INSERT column *column-name* does not accept NULLs.

System Action Taken: The statement containing the error was not processed.

Corrective Action: **RDSQL** will not allow you to insert a NULL value in a column that does not allow NULLS. Check to see if a non-NULL column is included in the column list in the INSERT statement.

-293 **Description of Error**: IS [NOT] NULL predicate may be used only with simple columns.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Restructure your query.

-294 **Description of Error**: The column *column-name* must be in the GROUP BY list.

System Action Taken: The statement was not processed.

Corrective Action: All non-aggregate columns in the SELECT list must be included in the GROUP BY list. Restructure your statement to include all columns that are not aggregate functions.

-295 **Description of Error**: The GROUP BY column number *number* is too large.

System Action Taken: The statement containing the error was not processed.

Corrective Action: The number of the column noted in your ORDER BY or GROUP BY statement exceeds the total number of columns in the SELECT statement.

-297 **Description of Error**: The SELECT list may not contain a subquery.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Remove the subquery from the SELECT list in the statement.

-298 **Description of Error**: COUNT(DISTINCT ...) may be used only with a simple column.

System Action Taken: The statement was not processed.

Corrective Action: You cannot include expressions with the COUNT(DISTINCT ...) function. Restructure the query.

-299 **Description of Error**: A query may not contain more than one DISTINCT.

System Action Taken: The statement was not processed.

Corrective Action: Restructure your query to include only one DISTINCT.

-300 **Description of Error**: There are too many GROUP BY columns (maximum is 8).

System Action Taken: The request was not completed.

Corrective Action: Reduce the number of columns in the statement to 8 or less.

-301 **Description of Error**: The total size of the GROUP BY columns exceeds 120 characters.

System Action Taken: The request was not completed.

Corrective Action: The total number of characters in all columns listed in the GROUP BY list exceeds 120 characters. Reduce the column list (perhaps exclude any columns with character data types greater than 30 characters.)

-302 **Description of Error**: No GRANT option.

System Action Taken: The statement containing the error was not processed.

Corrective Action: You do not have permission to grant access privileges to the table. Only the table owner or a user with GRANT permission can do this.

-303 **Description of Error**: Expression mixes columns with aggregates.

System Action Taken: The statement was not processed.

Corrective Action: Restructure your query so that columns and aggregates are not included in the same expression.

-304 **Description of Error**: HAVING can only have expressions with aggregates.

System Action Taken: The statement was not processed.

Corrective Action: The HAVING clause can only take on aggregates. Restructure your query.

-305 **Description of Error**: Subscripted column *column-name* is not of type CHAR.

System Action Taken: The request was not completed.

Corrective Action: Remove the subscript delimiter from the non-character column name in the request.

-306 **Description of Error**: Subscript out of range.

System Action Taken: The request was not completed.

Corrective Action: The range of the subscript delimiter exceeds the range of the column data type. Check the size of the data type and reduce the subscript range.

-307 **Description of Error**: Illegal subscript definition.

System Action Taken: The statement did not run.

Corrective Action: Check that you have not reversed the order of the subscript delimiters ([3,8] is a valid subscript; [8,3] is invalid) or included a negative number as a subscript delimiter.

-308 **Description of Error**: Column type must be the same for each UNION statement.

System Action Taken: The statement was not processed.

Corrective Action: Check that each column in the UNION statement are of the same data type.

-309 **Description of Error**: ORDER BY column *column-name* must be in SELECT list.

System Action Taken: The statement did not run.

Corrective Action: Check that columns included in the ORDER BY clause appear in the SELECT list.

-310 **Description of Error**: Table *table-name* already exists in database.

System Action Taken: The request was not completed.

Corrective Action: Select an alternate name for the table.

-311 **Description of Error**: Cannot open system catalog *catalog-name.*

System Action Taken: The statement was not processed.

Corrective Action: Check the C-ISAM error for information about the source of the problem.

-312 **Description of Error**: Cannot update system catalog *catalog-name.*

System Action Taken: The statement was not processed.

Corrective Action: Check the C-ISAM error for information about the source of the problem.

-313 **Description of Error**: Not owner of table *table-name.*

System Action Taken: The statement containing the error was not processed.

Corrective Action: Only the owner of the table (or the Database Administrator) can remove the table.

-314 **Description of Error**: Table *table-name* currently in use.

System Action Taken: The request was not completed.

Corrective Action: The table you wish to drop is currently being used by another user. Wait until the table is no longer being used before executing your request.

-315 **Description of Error**: No CREATE INDEX permission.

System Action Taken: The request was not completed.

Corrective Action: Permission not granted for you to create an index on the table.

-316 **Description of Error**: Index *index-name* already exists in database.

System Action Taken: The request was not completed.

Corrective Action: An index currently exists for the table. You must drop the existing index before creating a new one.

-317 **Description of Error**: Must have the same number of selected columns in each UNION statement.

System Action Taken: The request was not completed.

Corrective Action: Check the number of columns selected in each SELECT statement.

-318 **Description of Error**: File with the same name as specified log file already exists.

System Action Taken: The request was not completed.

Corrective Action: Select a different name for the log file.

-319 **Description of Error**: Index does not exist in database file.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Check the spelling of the index name or check the sysindexes system catalog for the correct index name.

-320 **Description of Error**: Not owner of index *index-name.*

System Action Taken: The statement containing the error was not processed.

Corrective Action: Only the owner of the index (or the Database Administrator) can remove the index.

-321 **Description of Error**: Cannot group by aggregate column *column-name.*

System Action Taken: The statement was not processed.

Corrective Action: Check the column number used in the GROUP BY clause.

-322 **Description of Error**: Cannot alter view *view-name.*

System Action Taken: The request was not completed.

Corrective Action: Views cannot be altered. You must drop and then recreate the view.

-323 **Description of Error**: Cannot grant permission on temporary table.

System Action Taken: The request was not completed.

Corrective Action: Permissions can only be granted on permanent tables.

-324 **Description of Error**: Ambiguous column *column-name.*

System Action Taken: The statement containing the error was not processed.

Corrective Action: A column name exists in more than one table cited in your query. Prepend each column name with the appropriate table name.

-325 **Description of Error**: Log file must be given a full pathname starting with a "/".

System Action Taken: The request was not completed.

Corrective Action: Provide the full pathname for the log file.

-326 **Description of Error**: Expecting CHAR type host variable.

System Action Taken: The statement was not processed.

Corrective Action: The cursor is on a CHAR column and your program is attempting to pass it a non-CHAR data type. Restructure your statement.

-327 **Description of Error**: Cannot unlock table *tablename* within a transaction.

System Action Taken: The statement containing the error was not processed.

Corrective Action: You cannot execute an UNLOCK TABLE statement within a transaction.

-328 **Description of Error**: Column *column-name* already exists in table.

System Action Taken: The request was not completed.

Corrective Action: Select a new column name for the column.

-329 **Description of Error**: Database not found or no system permission.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Check the spelling of the database name. Check that the database name exists in your current directory or a directory included in your DBPATH environment variable. Check the C-ISAM error for information about the source of the problem.

-330 **Description of Error**: Cannot create database.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Check that you have not entered the name of an existing database. Select an alternate name for the database. Check the C-ISAM error for information about the source of the problem.

-331 **Description of Error**: Cannot drop database directory.

System Action Taken: All database files in the database directory are deleted, but the directory remains.

Corrective Action: Remove any non-database files present in the database directory, then remove the directory. Check the C-ISAM error for information about the source of the problem.

-332 **Description of Error**: Cannot access audit trail name information.

System Action Taken: The request was not completed.

Corrective Action: Re-execute your request. If you again receive the error, the audit trail file has been corrupted. You may need to drop and restart the audit trail.

-333 **Description of Error**: The audit trail file already exists with a different name.

System Action Taken: The request was not completed.

Corrective Action: You must first drop the existing audit trail file (issue a DROP AUDIT statement) before creating a new audit trail.

-334 **Description of Error**: Cannot create audit trail.

System Action Taken: The statement containing the error was not processed.

Corrective Action: You must indicate the full pathname of the file receiving the audit trail. Check that you have permission to write to a file in the selected directory. Contact your System Administrator if you need help with this action.

-335 **Description of Error**: There is no audit trail for the specified table.

System Action Taken: The request was not completed.

Corrective Action: INFORMIX-4GL is unable to recover the table as no audit trail was created.

-336 **Description of Error**: Cannot create or drop audit on temporary table *table-name*.

System Action Taken: The request was not processed.

Corrective Action: You cannot place an audit trail on a temporary table.

-337 **Description of Error**: Cannot create view on temporary table *table-name*.

System Action Taken: The request was not processed.

Corrective Action: You cannot create a view on a temporary table.

-338 **Description of Error**: Cannot drop audit trail.

System Action Taken: The audit trail was not dropped (possible operating system error.)

Corrective Action: Re-execute your request. If the problem reoccurs, check the C-ISAM error message for information about the source of the problem. Contact your System Administrator if you require assistance with this action.

-339 **Description of Error**: The audit trail file name must start with '/'.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Edit your statement to include the full pathname of the audit trail file.

-340 **Description of Error**: Cannot open audit trail file.

System Action Taken: The request was not completed (operating system error).

Corrective Action: Check that you have operating system read permission to the file. Contact your System Administrator if you need help with this action.

-341 **Description of Error**: Could not read a row from audit trail file.

System Action Taken: The request was not completed (possible operating system error).

Corrective Action: Re-execute your request. If you again receive the error, the audit trail file has been corrupted. You may need to drop and restart the audit trail.

-343 **Description of Error**: Row from audit trail was added to a different position than expected.

System Action Taken: The request was not completed (possible operating system error).

Corrective Action: Re-execute your request. If you again receive the error, the audit trail file has been corrupted. You may need to drop and restart the audit trail.

-344 **Description of Error**: Cannot delete row—row in table does not match row in audit trail.

System Action Taken: The request was not completed (possible operating system error).

Corrective Action: Re-execute your request. If you again receive the error, the audit trail file has been corrupted. You may need to drop and restart the audit trail.

-345 **Description of Error**: Cannot update row—row in table does not match row in audit trail.

System Action Taken: The request was not completed (possible operating system error).

Corrective Action: Re-execute your request. If you again receive the error, the audit trail file has been corrupted. You may need to drop and restart the audit trail.

-346 **Description of Error**: Could not update a row in the table.

System Action Taken: The statement was not processed.

Corrective Action: Check the C-ISAM error for information about the source of the problem.

-347 **Description of Error**: Could not open table for exclusive access.

System Action Taken: The statement was not processed.

Corrective Action: Check the C-ISAM error for information about the source of the problem.

-348 **Description of Error**: Could not read a row from the table.

System Action Taken: The statement was not processed.

Corrective Action: Check the C-ISAM error for information about the source of the problem.

-349 **Description of Error**: Database not selected yet.

System Action Taken: The statement containing the error was not processed.

Corrective Action: You must first select the database before executing a statement that refers to a database.

-350 **Description of Error**: Index already exists on column.

System Action Taken: The request was not completed.

Corrective Action: Adding an index on the column is not necessary, as the column is already indexed.

-351 **Description of Error**: Database contains tables owned by other users.

System Action Taken: The statement was not processed.

Corrective Action: You can only drop a database if you own all tables in the database or have Database Administrator status. Contact your Database Administrator if you need help with this action.

-352 **Description of Error**: Column not found.

System Action Taken: The request was not completed.

Corrective Action: Check the spelling of the column name.

-353 **Description of Error**: No table or view specified when granting or revoking table privileges.

System Action Taken: The request was not completed.

Corrective Action: You must include the name of the table or view on which a privilege is granted or revoked in your **RDSQL** statement.

-354 **Description of Error**: Incorrect database or cursor name format.

System Action Taken: The statement was not processed.

Corrective Action: A database name must be ten characters or less on UNIX systems or eight characters or less on DOS systems. A cursor name must be 18 characters or less on UNIX and DOS systems. A database or cursor name must begin with a letter, and contain letters, numbers, or underscores. Check that you have not included an illegal character in the name.

-355 **Description of Error**: Cannot rename file for table.

System Action Taken: The statement was not processed.

Corrective Action: Check the C-ISAM error for information about the source of the problem.

-356 **Description of Error**: Table *table-name* specified in both main query and subquery.

System Action Taken: The statement was not processed.

Corrective Action: The statement is ambiguous because a column cannot be identified uniquely. Use an alias to rename the offending table.

-357 **Description of Error**: Dependent table for view *view-name* has been altered.

System Action Taken: The statement was not processed.

Corrective Action: A table upon which the view is constructed has been modified (for example, a column has been dropped, a datatype has been modified, or a column has been added to the middle of the table). Drop the view and create a new view.

-358 **Description of Error**: You must close the current database before you execute CREATE, START, or ROLLFORWARD.

System Action Taken: The statement containing the error was not processed.

Corrective Action: You can only use the CREATE DATABASE, START DATABASE, and ROLLFORWARD DATABASE statements when there is no current database. Execute a CLOSE DATABASE statement before executing one of these statements.

-359 **Description of Error**: Cannot drop current database.

System Action Taken: The statement containing the error was not processed.

Corrective Action: First execute a CLOSE DATABASE statement before executing a DROP DATABASE statement.

-360 **Description of Error**: Cannot modify table or view used in subquery.

System Action Taken: The statement was not processed.

Corrective Action: If allowed, your statement could reduce to a looping program. Edit your statement.

-361 **Description of Error**: Column size too large.

System Action Taken: The statement was not processed.

Corrective Action: Reduce the size of the column. You cannot have a CHAR column larger than 32767 characters.

-362 **Description of Error**: Can have only one column of SERIAL type.

System Action Taken: The action was not completed.

Corrective Action: You may not have more than one column of SERIAL type in a table. Select an alternate data type for the column.

-363 **Description of Error**: Cursor not on SELECT statement.

System Action Taken: The action was not completed.

Corrective Action: Use EXECUTE to execute prepared objects on non-SELECT statements.

-364 **Description of Error**: Column *column-name* not declared FOR UPDATE OF.

System Action Taken: The action was not completed.

Corrective Action: Include the column in the FOR UPDATE OF list.

-365 **Description of Error**: Cursor must be on simple SELECT for FOR UPDATE.

System Action Taken: The action was not completed.

Corrective Action: Check that the cursor does not include more than one table or involve aggregates.

-366 **Description of Error**: The scale exceeds the maximum precision specified.

System Action Taken: The request was not completed.

Corrective Action: The problem is located in a DECIMAL or MONEY column: the scale (number of digits to the right of the decimal point) exceeds the precision (total number of digits).

```
DECIMAL(m,n)  where n>m
MONEY(m,n)    where n>m
MONEY(m)      where m<2
```

-367 **Description of Error**: Sums and averages cannot be computed for character columns.

System Action Taken: The request was not completed.

Corrective Action: Check that you have not included a column of character data type in the aggregate function statement.

-368 **Description of Error**: Incompatible sqlexec module.

System Action Taken: The request was not completed.

Corrective Action: Check that the correct version of **sqlexec** has been installed. Contact your Database Administrator if you need help with this action.

-369 **Description of Error**: Invalid serial number. Please consult your installation instructions.

System Action Taken: The request was not completed.

Corrective Action: Check that the correct version of **sqlexec** has been installed. Contact your Database Administrator if you need help with this action.

-370 **Description of Error**: Cannot drop last column.

System Action Taken: The request was not completed.

Corrective Action: Only one column remains in the table. Use the DROP TABLE statement to remove the table.

-371 **Description of Error**: Cannot create unique index on column with duplicate data.

System Action Taken: The request was not completed.

Corrective Action: The column contains duplicate data.

-372 **Description of Error**: Cannot alter table with audit trail on.

System Action Taken: The statement containing the error was not processed.

Corrective Action: You must first drop the audit trail before making any changes to the table. After making the changes you may want to re-establish an audit trail.

-373 **Description of Error**: DBPATH too long.

System Action Taken: The request was not completed.

Corrective Action: Reduce the length of your DBPATH environment variable.

-374 **Description of Error**: Can only use column number in ORDER BY clause with UNION.

System Action Taken: The request was not completed.

Corrective Action: Restructure the query, using ordinal numbers for the ORDER BY columns.

-375 **Description of Error**: Cannot create log file for transaction.

System Action Taken: The request was not completed.

Corrective Action: Check the C-ISAM error for information about the source of the problem.

-376 **Description of Error**: Log file already exists.

System Action Taken: The request was not completed.

Corrective Action: Select an alternate name for the log file.

-377 **Description of Error**: Must terminate transaction before closing database.

System Action Taken: The request was not completed.

Corrective Action: Issue a COMMIT WORK or ROLLBACK statement before closing the database.

-378 **Description of Error**: Record currently locked by another user.

System Action Taken: The request was not completed.

Corrective Action: Wait until the record is unlocked before submitting the statement. Check the C-ISAM error for information about the source of the problem.

-379 **Description of Error**: Cannot revoke privilege on columns.

System Action Taken: The request was not completed.

Corrective Action: First revoke UPDATE or SELECT privilege on the table, then grant the revoked privileges.

-380 **Description of Error**: Cannot erase log file.

System Action Taken: The request was not completed.

Corrective Action: Check the C-ISAM error for information about the source of the problem.

-381 **Description of Error**: Cannot grant to someone who has granted you the same privilege before.

System Action Taken: The request was not completed.

Corrective Action: The name of the individual who granted you permission to use the table must be removed from your user list.

-382 **Description of Error**: Same number of columns must be specified for view and select clause.

System Action Taken: The request was not completed.

Corrective Action: Check the number of columns in the view definition and the selected columns.

-383 **Description of Error**: View column for aggregate or expression must be explicitly named.

System Action Taken: The request was not completed.

Corrective Action: Provide a name for all virtual columns.

-384 **Description of Error**: Cannot modify non-simple view.

System Action Taken: The request was not completed.

Corrective Action: Can only modify views built on a single table.

-385 **Description of Error**: Data value out of range.

System Action Taken: The request was not completed.

Corrective Action: Check the view definition for valid data ranges.

-386 **Description of Error**: Column contains null values.

System Action Taken: The request was not completed.

Corrective Action: The table contains NULL values in a column being altered to disallow NULLS. Remove NULL values from the column.

-387 **Description of Error**: No connect permission.

System Action Taken: The request was not completed.

Corrective Action: Contact the Database Administrator and request CONNECT permission.

-388 **Description of Error**: No resource permission.

System Action Taken: The request was not completed.

Corrective Action: Contact the Database Administrator and request RESOURCE permission.

-389 **Description of Error**: No DBA permission.

System Action Taken: The request was not completed.

Corrective Action: Contact the Database Administrator and request DBA permission.

-390 **Description of Error**: Synonym already used as table name or synonym.

System Action Taken: The request was not completed.

Corrective Action: Select a different synonym. Check the **syssynonym** system catalog for a list of existing synonyms.

-391 **Description of Error**: Cannot insert a NULL into column *column-name.*

System Action Taken: The request was not completed.

Corrective Action: Check that a column that does not allow NULL values is omitted from the insert column list.

-392 **Description of Error**: System error—unexpected NULL pointer encountered.

System Action Taken: The request was not completed.

Corrective Action: Contact the Technical Support Department at RDS.

-393 **Description of Error**: A condition in the where clause results in a two-sided outer join.

System Action Taken: The request was not completed.

Corrective Action: A two-sided outer join is not allowed. Restructure your query.

-394 **Description of Error**: View *view-name* not found.

System Action Taken: The request was not completed.

Corrective Action: Check the spelling of the view name. Check the **sysviews** system catalog for a list of existing views.

-395 **Description of Error**: The where clause contains an outer cartesian product.

System Action Taken: The request was not completed.

Corrective Action: Check the syntax of the statement.

-396 **Description of Error**: Illegal join between a nested outer table and a preserved table.

System Action Taken: The request was not completed.

Corrective Action: Check the syntax of the statement.

-397 **Description of Error**: System catalog corrupted.

System Action Taken: The request was not completed.

Corrective Action: Contact the Database Administrator for help with this error.

-398 **Description of Error**: Cursor manipulation must be within a transaction.

System Action Taken: The request was not completed.

Corrective Action: Perform a BEGIN WORK statement before any cursor manipulations.

-399 **Description of Error**: Cannot access log file.

System Action Taken: The request was not completed.

Corrective Action: You may not edit the log file.

-400 **Description of Error**: Fetch attempted on unopen cursor.

System Action Taken: Your statement did not run.

Corrective Action: Check that the cursor was properly opened using an OPEN CURSOR statement.

-401 **Description of Error**: Fetch attempted on NULL cursor.

System Action Taken: Your statement did not run.

Corrective Action: Check that the cursor was properly opened using an OPEN CURSOR statement.

-402 **Description of Error**: Address of a host variable is NULL.

System Action Taken: Your statement did not run.

Corrective Action: Check the addresses of each program variable (one or more have a NULL value).

-403 **Description of Error**: The size of a received row disagrees with the expected size.

System Action Taken: Your statement did not run.

Corrective Action: Check that you are using the proper library in the program.

-404 **Description of Error**: A NULL control block has been passed as an argument.

System Action Taken: Your statement did not run.

Corrective Action: Check that you are using the proper library in the program.

-405 **Description of Error**: The address of a host variable is not properly aligned.

System Action Taken: Your statement did not run.

Corrective Action: Check that each program variable is aligned with the proper address boundary for variables of that type.

-406 **Description of Error**: Memory allocation failed.

System Action Taken: Your statement did not run.

Corrective Action: Exit to the operating system command line, re-enter the program you were using, and resubmit your program.

-407 **Description of Error**: Error number zero received from the **sqlexec** process.

System Action Taken: Your statement did not run.

Corrective Action: Exit to the operating system command line, re-enter the program you were using, and resubmit your program.

-408 **Description of Error**: Invalid message type received from the **sqlexec** process.

System Action Taken: Your statement did not run.

Corrective Action: Exit to the operating system command line, re-enter the program you were using, and resubmit your program.

-409 **Description of Error**: **sqlexec** was not found or was not executable by the current user.

System Action Taken: Your statement did not run.

Corrective Action: Check that your INFORMIXDIR environment variable is properly set. Contact your System Administrator if you need help with this action.

-410 **Description of Error**: PREPARE statement failed or was not executed.

System Action Taken: Your statement was not executed.

Corrective Action: Check that your PREPARE statement was successfully executed (a failure is often due to a syntax error).

-412 **Description of Error**: Command pointer is NULL.

System Action Taken: Your statement was not executed.

Corrective Action: The statement executed prior to the current statement returned an error that was not trapped. Re-execute the prior statement(s) and include a response to the error code returned.

-413 **Description of Error**: Insert attempted on unopened cursor.

System Action Taken: The statement containing the error was not processed.

Corrective Action: You must first open the cursor before executing a PUT statement.

-414 **Description of Error**: Insert attempted on NULL cursor.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Make sure you have correctly declared the cursor for insert. Check the error code returned from the DECLARE statement.

-415 **Description of Error**: Data conversion error discovered during PUT operation.

System Action Taken: The statement containing the error was not processed.

Corrective Action: The program variable is incompatible with the data type of a column in the database. Choose an appropriate program variable or restrict the size of the data in the program variable.

-416 **Description of Error**: USING option with open statement is invalid for insert cursor.

System Action Taken: The statement containing the error was not processed.

Corrective Action: You should use the FROM option with the PUT statement or the USING option with the EXECUTE statement.

-417 **Description of Error**: FLUSH can only be used on an insert cursor.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Make sure that you are using the correct cursor.

-420 **Description of Error**: Cannot execute remote **sqlexec**.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Check the setting of the SQLHOST environment variable, that the **sqlexec** file exists on the specified node, or that the specified node is accessible to you.

-421 **Description of Error**: Unknown service for execution of remote **sqlexec**.

System Action Taken: The statement containing the error was not processed.

Corrective Action: The **/etc/services** file does not contain an **sql** entry. Check with your System Administrator about adding the required entry.

-422 **Description of Error**: FLUSH attempted on unopened cursor.

System Action Taken: The statement containing the error was not processed.

Corrective Action: You must first open an insert cursor before executing the FLUSH statement.

-500 **Description of Error**: Clustered index *index-name* already exists in the table.

System Action Taken: The statement containing the error was not processed.

Corrective Action: A table can have only one clustered index. You must first alter the existing cluster index to NOT CLUSTER before creating a new clustered index.

-501 **Description of Error**: Index *index-name* is already not clustered.

System Action Taken: The statement containing the error was not processed.

Corrective Action: You do not need to alter the index to NOT CLUSTER.

-502 **Description of Error**: Cannot cluster index.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Check the C-ISAM error code for more information about the source of the problem.

-503 **Description of Error**: Too many tables locked.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Either unlock one or more tables, or open the database in exclusive mode.

-504 **Description of Error**: Cannot lock a view.

System Action Taken: The statement containing the error was not processed.

Corrective Action: You cannot execute the LOCK TABLE command on a view.

-505 **Description of Error**: Number of columns in UPDATE does not match number of VALUES.

System Action Taken: The statement containing the error was not processed.

Corrective Action: The number of columns in the UPDATE statement must equal the number of values. Rewrite your RDSQL statement.

-506 **Description of Error**: Do not have permission to update all columns.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Obtain permission from the owner of the table. You can query the **systables** system catalog for this information.

-507 **Description of Error**: Cursor not found.

System Action Taken: The statement containing the error was not processed.

Corrective Action: This cursor has not been defined. Check the spelling of *cursor-name*.

-508 **Description of Error**: You cannot rename a temporary table.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Remove the RENAME TABLE statement from your program.

-509 **Description of Error**: You can not rename a column in a temporary table.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Remove the RENAME COLUMN statement from your program.

-510 **Description of Error**: Cannot create synonym for temporary table *table-name.*

System Action Taken: The statement containing the error was not processed.

Corrective Action: Remove the CREATE SYNONYM statement from your program.

-511 **Description of Error**: Cannot modify system catalog *catalog-name.*

System Action Taken: The statement containing the error was not processed.

Corrective Action: You do not have alter permission on any of the system catalogs. Do not attempt to modify these files.

-512 **Description of Error**: Cannot open database *database-name* in exclusive mode.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Another user is currently using the database. Wait until the database is no longer in use.

-516 **Description of Error**: System error-temporary output file not yet created.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Check the C-ISAM error for the source of the problem.

-802 **Description of Error**: Cannot open file for run (operating system error).

System Action Taken: Your statement was not executed.

Corrective Action: Check that the file exists (if it is not found in your current directory you will need to include a full pathname). Check that you have operating system read permission to access the file. Contact your System Administrator if you need help with this action.

-804 **Description of Error**: Comment has no end.

System Action Taken: Your statement containing the error was not processed.

Corrective Action: Comments must be enclosed within a pair of { and } braces. Edit your statement to include a matching right brace } to the left brace { denoting the start of your comment. (Note: comments may not be nested.)

-809 **Description of Error**: **RDSQL** syntax error has occurred.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Check that you have not misspelled an **RDSQL** statement, placed key words out of sequence, or included an **INFORMIX-4GL** reserved word in your query.

-816 **Description of Error**: Cannot read file (check file permissions).

System Action Taken: The statement did not run (operating system error).

Corrective Action: Check that you have operating system read permission to access the file. Contact your System Administrator if you need help with this action.

-817 **Description of Error**: Cannot write file (check file permissions).

System Action Taken: The statement did not run (operating system error).

Corrective Action: Check that you have operating system write permission in the designated directory. Contact your System Administrator if you need help with this action.

-824 **Description of Error**: Missing values clause on insert statement.

System Action Taken: The statement did not run.

Corrective Action: The INSERT INTO statement requires a VALUES clause. Check that you have included a VALUES clause (with a values list) in your statement.

-825 **Description of Error**: Program not found.

System Action Taken: The statement did not run (operating system error).

Corrective Action: INFORMIX-4GL could not locate a necessary program. Check that your INFORMIXDIR environment variable is properly set. Contact your System Administrator you need help with this action.

-826 **Description of Error**: Fork system call failed.

System Action Taken: The statement did not run (operating system error).

Corrective Action: INFORMIX-4GL was unable to fork a necessary process. Pause a moment and re-enter your request. If you receive this error message again, contact your System Administrator.

-827 **Description of Error**: Database not found.

System Action Taken: The statement was not processed.

Corrective Action: Check the spelling of the database name. Check that the database name exists in your current directory or a directory specified in your DBPATH environment variable.

-829 **Description of Error**: Form not found.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Check the spelling of the form name. Make sure the the form exists in your current directory or a directory specified in your DBPATH environment variable.

-832 **Description of Error**: Error(s) found in form specifications.

System Action Taken: The form compile was not completed.

Corrective Action: Edit the form specification file following the prompts provided by FORM4GL. Identified error messages may be located in this appendix.

-834 **Description of Error**: **form4gl** could not compile the form.

System Action Taken: The form compile was not completed.

Corrective Action: Attempt a second compile of the form, or run **FORM4GL** outside of **INFORMIX-4GL**.

-836 **Description of Error**: Insert statement has no values clause.

System Action Taken: The statement did not run.

Corrective Action: The INSERT INTO statement requires a VALUES clause. Check that you have included a VALUES clause (with a values list) in your statement.

-837 **Description of Error**: There is not enough memory available.

System Action Taken: The statement did not run.

Corrective Action: There is insufficient data space in memory to run your request. Save your program, exit from **INFORMIX-4GL**, and then re-enter **INFORMIX-4GL** and run your program again. If this does not work, you may need to reduce the complexity of your program.

-839 **Description of Error**: Table not found.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Check the spelling of the table name. Make sure the *table-name.dat*, *table-name.idx*, and *table-name.lok* files are located in the *database-name.dbs* directory.

-840 **Description of Error**: Name is too long.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Database names must be eight (in DOS) or ten (in UNIX) characters or less. Select a new name of the appropriate length.

-841 **Description of Error**: Name must start with a letter and contain letters, digits, or underscores.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Select a name beginning with a letter and containing only letters, digits, and underscores.

-842 **Description of Error**: Cannot read temp file.

System Action Taken: The statement did not run (operating system error).

Corrective Action: Check that you have operating system read permission to access the file. Contact your System Administrator you need help with this action.

-843 **Description of Error**: Cannot write temp file.

System Action Taken: The statement did not run (operating system error).

Corrective Action: Check that you have operating system write permission to create a file in the current directory, the **/tmp** directory, or the directory indicated in your DBTEMP environment variable. Contact your System Administrator if you need help with this action.

-846 **Description of Error**: Number of values in load file is not equal to the number of columns.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Check that the number of values in the load file equals the number of columns in the table.

-847 **Description of Error**: Error in load file line *line-no.*

System Action Taken: The statement containing the error was not processed.

Corrective Action: See the accompanying error message for information about the source of the problem. Check the load file.

-1101 **Description of Error**: Variable address is NULL.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Call RDS Technical Support.

-1102 **Description of Error**: Field name is not found in form.

System Action Taken: The statement containing the error was not processed.

Corrective Action: A field name listed in an INPUT, INPUT ARRAY, DISPLAY, DISPLAY ARRAY, or CONSTRUCT statement does not appear in the form specification file of the screen form that is currently displayed. Change the field name in the program or form specification file as appropriate and recompile.

-1103 **Description of Error**: This value is not among the valid possibilities.

System Action Taken: The system issues this warning during an INPUT or INPUT ARRAY statement when the user tries to enter data outside the range(s) specified for the field via the INCLUDE attribute in the form specification.

Corrective Action: The user must enter a value that appears in the INCLUDE list.

-1104 **Description of Error**: The two entries were not the same—please try again.

System Action Taken: The system issues this warning during an INPUT or INPUT ARRAY statement when the user does not enter the same value twice in a display field that has been assigned the VERIFY attribute.

Corrective Action: The user must enter the same value twice before the value will be accepted.

-1105 **Description of Error**: You cannot use this editing feature because a picture exists.

System Action Taken: The system issues this warning during an INPUT or INPUT ARRAY statement when the user tries to use CONTROL-A, CONTROL-D, or CONTROL-X in a display field that has been assigned the PICTURE attribute.

Corrective Action: The user should not use the CONTROL-A, CONTROL-D, or CONTROL-X keys when entering data in a field that has been assigned the PICTURE attribute.

-1106 **Description of Error**: Error in field.

System Action Taken: The system issues this warning when the user tries to enter a value in a display field that cannot be converted to the data type of the corresponding program variable in the INPUT or INPUT ARRAY statement.

Corrective Action: The user must enter a value in a display field that is compatible with the data type of the corresponding program variable in the INPUT or INPUT ARRAY statement.

-1107 **Description of Error**: Field subscript out of bounds.

System Action Taken: The statement containing the error was not processed.

Corrective Action: The subscript of a screen array in an INPUT, INPUT ARRAY, DISPLAY, DISPLAY ARRAY, or CONSTRUCT statement is larger than the corresponding subscript in the SCREEN RECORD statement that defines the screen array in the form specification file. Change the subscript in your program or make the appropriate modifications to your form specification file.

-1108 **Description of Error**: Record not in form.

System Action Taken: The statement containing the error was not processed.

Corrective Action: The screen record in an INPUT, INPUT ARRAY, DISPLAY, DISPLAY ARRAY, or CONSTRUCT statement does not appear in the form specification file. Change the screen record name in your program or in your form specification file.

-1109 **Description of Error**: List and record field counts differ.

System Action Taken: The statement containing the error was not processed.

Corrective Action: The number of program variables or columns is not the same as the number of fields in an INPUT, INPUT ARRAY, DISPLAY, DISPLAY ARRAY, or CONSTRUCT statement. Check the *variable-list* or *column-list* and the *field-list* to make sure they contain the same number of items.

-1110 **Description of Error**: Form file not found.

System Action Taken: The statement containing the error was not processed.

Corrective Action: The form file specified in the OPEN FORM statement could not be found. Make sure that your program can access the form file specified in the OPEN FORM statement.

-1111 **Description of Error**: Field table offset out of bounds.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Call RDS Technical Support.

-1112 **Description of Error**: Form file is incompatible.

System Action Taken: The statement containing the error was not processed.

Corrective Action: The form file specified in an OPEN FORM statement has been corrupted or was compiled with an older version of **FORM4GL**. Recompile the form specification file with the appropriate version of **FORM4GL**.

Note: Do not use an extension with the form name in an OPEN FORM statement.

-1113 **Description of Error**: Memory allocation error.

System Action Taken: The program stops.

Corrective Action: Divide your program into smaller programs.

-1114 **Description of Error**: There is not an open form.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Make sure you successfully execute an OPEN FORM statement before a DISPLAY FORM statement.

-1116 **Description of Error**: Default value from form field cannot be converted to input variable type.

System Action Taken: The statement containing the error was not processed.

Corrective Action: The default value in a display field is not compatible with the data type of the corresponding program variable in an INPUT or INPUT ARRAY statement. Change the default value for the display field or change the data type of the corresponding variable in the INPUT or INPUT ARRAY statement.

-1117 **Description of Error**: Cannot convert date value to string.

System Action Taken: The statement containing the error was not processed.

Corrective Action: A DATE value in the database cannot be converted to a string.

-1119 **Description of Error**: NEXT FIELD name not found.

System Action Taken: The statement containing the error was not processed.

Corrective Action: The field name in the NEXT FIELD clause of an INPUT or INPUT ARRAY statement does not appear in the *field-list* of the statement. Change the NEXT FIELD clause or the *field-list* as appropriate.

-1120 **Description of Error**: Message file not found.

System Action Taken: The statement containing the error was not processed.

Corrective Action: The message file specified in the HELP FILE clause of the OPTIONS statement either could not be found or could not be read. Make sure your program can access the message file specified in the OPTIONS HELP FILE statement.

-1121 **Description of Error**: Message number not found in message file.

System Action Taken: The statement containing the error was not processed.

Corrective Action: A message number in an INPUT, PROMPT, or MENU statement does not appear in the corresponding message file. Change the message number in your program or add a new message to the file and recompile.

-1122 **Description of Error**: Incompatible message file.

System Action Taken: The statement containing the error was not processed.

Corrective Action: The message file specified in the HELP FILE clause of the OPTIONS statement could not be read. Make sure the OPTIONS HELP FILE statement specifies a compiled message file that is readable.

-1123 **Description of Error**: No help file specified.

System Action Taken: The statement containing the error was not processed.

Corrective Action: The user pressed the help key before a OPTIONS HELP FILE statement was executed. Make sure you execute the appropriate OPTIONS HELP FILE statement before allowing the user to request help during an INPUT, PROMPT, or MENU statement.

-1124 **Description of Error**: This field requires an entered value.

System Action Taken: The system issues this warning during an INPUT or INPUT ARRAY statement when the user tries to end input without entering a value in a display field that has been assigned the REQUIRED attribute.

Corrective Action: The user must enter a value in the required field.

-1125 **Description of Error**: Please type again for verification.

System Action Taken: The system issues this message during an INPUT or INPUT ARRAY statement after the user enters a value once in a display field that has been assigned the VERIFY attribute.

Corrective Action: The user should enter the same value again.

-1126 **Description of Error**: Cannot insert another row—the input array is full.

System Action Taken: The system issues this warning during an INPUT ARRAY statement when the user tries to insert a row after the program array is full.

Corrective Action: The user should select another editing function or end input. If the user receives this warning frequently, the user should have the applications designer increase the size of the program array.

-1127 **Description of Error**: Cannot delete row—it has no data.

System Action Taken: The system issues this warning during an INPUT ARRAY statement when the user selects the delete function while the cursor is on the blank row below the last row of the program array.

Corrective Action: The user should select another editing function or end input.

-1128 **Description of Error**: There are no more rows in the direction you are going.

System Action Taken: The system issues this warning during an INPUT ARRAY or DISPLAY ARRAY statement when the user presses the **UP ARROW** or Previous Page key while the cursor is at the beginning of the program array or when the user presses the **DOWN ARROW** or Next Page key while the cursor is at the end of the program array.

Corrective Action: The user should select another scrolling or editing function.

-1129 **Description of Error**: Field in BEFORE/AFTER clause not found in form.

System Action Taken: The statement containing the error was not processed.

Corrective Action: A field name in the BEFORE or AFTER clause of an INPUT or INPUT ARRAY statement does not appear in the form specification file of the screen form that is currently displayed. Change the field name in the program or form specification file as appropriate and recompile.

-1130 **Description of Error**: You cannot have multiple BEFORE clauses for the same field.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Combine the BEFORE clauses for the same field in INPUT and INPUT ARRAY statements.

-1131 **Description of Error**: You cannot have multiple AFTER clauses for the same field.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Combine the AFTER clauses for the same field in INPUT and INPUT ARRAY statements.

-1132 **Description of Error**: The destination string of the CONSTRUCT statement is not large enough to contain the constructed string.

System Action Taken: The statement containing the error was not processed.

Corrective Action: The character variable for storing the user's search criteria in a CONSTRUCT statement is not large enough. Increase the size of the character variable in the appropriate DEFINE statement and recompile your program.

-1133 **Description of Error**: The NEXT OPTION name is not in the menu.

System Action Taken: The statement containing the error was not processed.

Corrective Action: The NEXT OPTION name is not a menu option. Change the menu option in the NEXT OPTION statement or the menu as appropriate and recompile your program.

-1134 **Description of Error**: There is no termcap entry for this function key.

System Action Taken: The statement containing the error was not processed.

Corrective Action: The specified key in the HELP KEY, INSERT KEY, DELETE KEY, NEXT KEY, or PREVIOUS KEY clause of an OPTIONS statement does not appear in the termcap file. Add the code for the specified function key to the termcap file. See Appendix N of the *INFORMIX-4GL Reference Manual* for details.

-1135 **Description of Error**: The row or column number in DISPLAY AT exceeds the limits of your screen or window.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Change the row or column number in the DISPLAY AT statement so that it is within the limits of your screen or window.

-1136 **Description of Error**: Window is too large to fit on the screen.

System Action Taken: The statement containing the error was not processed.

Corrective Action: The window dimensions specified in the WITH clause of the OPEN WINDOW statement exceed the limits of your screen. Reduce the size of the window dimensions, or change the originating location of the window. If you are attempting to open the window WITH FORM, you must reduce the size of the form. (When sizing a window to fit a form, INFORMIX-4GL truncates trailing blank spaces but prints leading blank spaces. You might remove any leading blank spaces.)

-1137 **Description of Error**: Cannot open window.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Your INFORMIX-4GL program exceeds the data space limit of your system. Reduce the complexity of the program by closing one or more windows or forms, or reducing the number of global variables.

-1138 **Description of Error**: Border does not fit on screen. Window is too large.

System Action Taken: The statement containing the error was not processed.

Corrective Action: The dimensions of the bordered window exceed the limits of your screen. Note that the border appears outside the dimensions of the window. Reduce the size of the window dimensions. (Unix system note: if the **termcap** entry for your terminal includes an sg#1 setting, **INFORMIX-4GL** reserves an additional column to the left and to the right of the window when computing the required window size.)

-1139 **Description of Error**: Form line cannot be set using LAST keyword.

System Action Taken: The statement containing the error was not processed.

Corrective Action: You cannot use LAST or LAST-*integer* to set the FORM line. You must indicate the FORM line relative to FIRST or a literal value.

-1141 **Description of Error**: Cannot close window with active INPUT, DISPLAY ARRAY, or MENU statement.

System Action Taken: The statement containing the error was not processed.

Corrective Action: You must complete the INPUT, DISPLAY ARRAY, or MENU statement before closing the current window.

-1142 **Description of Error**: Window is too small to display this form.

System Action Taken: The statement containing the error was not processed.

Corrective Action: The window dimensions specified in the WITH clause of the OPEN WINDOW statement are too small for the form. Either open the window WITH FORM or increase the dimensions in the WITH clause.

-1143 **Description of Error**: Window is already open.

System Action Taken: The statement containing the error was not processed.

Corrective Action: You cannot issue an OPEN WINDOW statement for a window that is already open.

-1144 **Description of Error**: Cannot open window. Window origin is not on the screen.

System Action Taken: The statement containing the error was not processed.

Corrective Action: The originating location specified in the AT clause of the OPEN WINDOW statement exceeds the limits of the screen. Specify a new location.

-1145 **Description of Error**: Cannot open ERROR window.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Your INFORMIX-4GL program exceeds the data space limits of your system. Reduce the complexity of the program by closing one or more windows or forms, or reducing the number of global variables.

-1146 **Description of Error**: PROMPT message is too long to fit in the window.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Reduce the length of the PROMPT statement or enlarge the window. You will receive a runtime error if the window does not accommodate the text of the PROMPT message along with the characters entered by the user in response to the prompt. (Note: if necessary, INFORMIX-4GL truncates MESSAGE and COMMENT text. INFORMIX-4GL does not truncate the text of a PROMPT statement.)

-1147 **Description of Error**: You cannot CLOSE, CLEAR, or make CURRENT an unopened window.

System Action Taken: The statement containing the error was not processed.

Corrective Action: You must first execute an OPEN WINDOW statement before executing a CLOSE WINDOW, CLEAR WINDOW, or CURRENT WINDOW statement. In addition, you cannot execute these statements once the window is CLOSEd.

-1148 **Description of Error**: Size of a window may not be negative.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Only positive integers or variables that evaluate to positive integers can be used in the WITH clause of the OPEN WINDOW statement.

-1149 **Description of Error**: An unknown code has been detected in the form. Be sure you have linked your 4GL program with the correct 4GL libraries.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Relink your INFORMIX-4GL program with the most recent version of the INFORMIX-4GL libraries. Rebuild your form with FORM4GL.

-1150 **Description of Error**: Window is too small to display this menu.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Increase the dimensions of the window to accommodate the menu. At a minimum, the window must be two rows long and large enough to display the menu title and a colon, the longest option, two sets of ellipses, and 6 spaces.

-1200 **Description of Error**: Number is too large for a DECIMAL data type.

System Action Taken: The statement containing the error was not processed.

Corrective Action: The range of numbers allowed as a decimal data type has been exceeded. Allowable decimal numbers range from $-.1 \times 10^{-128}$ to $.1 \times 10^{126}$ (with 32 significant digits). Check the size of the number.

-1201 **Description of Error**: Number is too small for a DECIMAL data type.

System Action Taken: The statement containing the error was not processed.

Corrective Action: The range of numbers allowed as a decimal data type has been exceeded. Allowable decimal numbers range from $-.1 \times 10^{-128}$ to $.1 \times 10^{126}$ (with 32 significant digits). Check the size of the number.

-1202 **Description of Error**: An attempt was made to divide by zero.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Check that you are not attempting to divide a numerical column type by a character column type (for example, 16/Jones) or that the value of the divisor does not equal zero.

-1203 **Description of Error**: Values used in a MATCH must both be type CHARACTER.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Check that the values included in your MATCH condition are both CHAR types. Use an alternate comparison condition for non-CHAR types.

-1204 **Description of Error**: Invalid year in date.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Acceptable years are 0001 to 9999. If two digits are used, **RDSQL** assumes the year is 19xx. Check the value entered in the date field.

-1205 **Description of Error**: Invalid month in date.

System Action Taken: The statement containing the error was not processed.

Corrective Action: In **RDSQL** statements such as INSERT a month can be represented as the number of the month (1 through 12). In display fields a month can be represented either as the number of the month (1 through 12) or as the first three letters of the name of the month. Check the value entered in the date column or field.

-1206 **Description of Error**: Invalid day in date.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Acceptable days are 01 through 31. Check the value entered in the date column or field.

-1208 **Description of Error**: There is no conversion from non-character values to characters values.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Numeric values cannot be converted to character values in **RDSQL** statements such as INSERT and UPDATE. Make sure that your **RDSQL** statements do not attempt this kind of conversion.

-1209 **Description of Error**: Without any delimiters, this date must contain exactly 6 or 8 digits.

System Action Taken: The statement containing the error was not processed.

Corrective Action: You must enter either 6 or 8 digits when specifying a data value to represent the date.

-1210 **Description of Error**: Date could not be converted to month/day/year format.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Dates must be presented as month, day, and year (August 4, 1986 is allowed; 4 August, 1986 is not). Check the sequence of characters entered in the date field.

-1211 **Description of Error**: Out of memory.

System Action Taken: The statement containing the error was not processed.

Corrective Action: You have exceeded the data space capacity on your machine. Reduce the complexity of your form.

-1212 **Description of Error**: Date conversion format string must contain a month, day, and year component.

System Action Taken: The statement containing the error was not processed.

Corrective Action: The FORMAT string used to format a DATE field must contain month, day, and year components. One of these is missing.

-1213 **Description of Error**: Character to numeric conversion error.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Check that the values in the character string contain only ASCII characters representing numeric data types. (A table of ASCII codes is included as an appendix to this manual.)

-1214 **Description of Error**: Value too large to fit in a SMALLINT.

System Action Taken: The statement was not processed.

Corrective Action: Acceptable SMALLINT values are whole numbers between -32,767 and 32,767. If a larger number is required, you need to use the **RDSQL** ALTER TABLE statement to modify the column to INTEGER type.

-1215 **Description of Error**: Value too large to fit in an INTEGER.

System Action Taken: The statement was not processed.

Corrective Action: Acceptable INTEGER values are whole numbers between -2,147,483,647 and 2,147,483,647. If a larger number is required, you need to use the **RDSQL** ALTER TABLE statement to modify the column to DECIMAL type.

-1216 **Description of Error**: Illegal exponent.

System Action Taken: The statement was not processed.

Corrective Action: Check that the exponent is an integer with a value not exceeding 32767.

-1217 **Description of Error**: The format string is too large.

System Action Taken: The statement was not processed.

Corrective Action: Reduce the size of the FORMAT string (used to format a DATE field) in the form specification.

-1218 **Description of Error**: String to date conversion error.

System Action Taken: The statement was not processed.

Corrective Action: Check the format of the DATE data type in the DBDATE environment variable. The format is illegal.

-1226 **Description of Error**: Decimal or money value exceeds maximum precision.

System Action Taken: The statement was not processed.

Corrective Action: Increase the precision of the DECIMAL or MONEY field.

-2014 **Description of Error**: There was an incorrect number of arguments on the operating system command line. At least one argument is expected.

System Action Taken: The operating system command was not executed.

Corrective Action: FORM4GL requires that you include a filename on the command line (unless you use the **-d** option to FORM4GL). Carefully re-enter the command, including the filename as an argument, or use the **-d** option to FORM4GL.

-2015 **Description of Error**: An open comment symbol, {, was found inside an already open comment on line *lineno*, character *charposition*. This could be due to a failure to close the previously opened comment, which was begun on line *lineno*, character *charposition*.

System Action Taken: The compile was not completed.

Corrective Action: Comments must be enclosed within a pair of { and } braces. Insert a close comment symbol where appropriate. (Note: comments may not be nested.)

-2016 **Description of Error**: A comment has been opened, but not closed. The last comment begun was opened on line *lineno*, character *charposition*.

System Action Taken: The compile was not completed.

Corrective Action: Comments must be enclosed within a pair of { and } braces. Insert a close comment symbol where appropriate.

-2018 **Description of Error**: A grammatical error has been found on line *lineno*, character *charposition*. The construct is not understandable in its context.

System Action Taken: The compile was not completed.

Corrective Action: Check the grammatical content of the statement (placement of commas, braces, etc).

-2019 **Description of Error**: This integer exceeds the maximum size allowed.

System Action Taken: The compile was not completed.

Corrective Action: Allowable INTEGER values are whole numbers that range from -2,147,483,647 to 2,147,483,647. Check the value of the number (number of digits and location of the decimal point). If a larger number is required, you will need to use the **RDSQL** ALTER TABLE statement to modify the column to DECIMAL type.

-2020 **Description of Error**: The table *tablename* could not be opened. The operating system was asked to open it for writing.

System Action Taken: The compile was not completed (operating system error).

Corrective Action: Check that you have operating system write permission to create a file in the designated directory. Contact your System Administrator if you need help with this action.

-2022 **Description of Error**: This identifier exceeds the maximum length for identifiers, which is 50.

System Action Taken: The compile was not completed.

Corrective Action: Check that all field names, field labels, and identifiers are less than or equal to 50 characters in length.

-2023 **Description of Error**: This quoted string exceeds the maximum length for quoted strings, which is 80.

System Action Taken: The compile was not completed.

Corrective Action: Reduce the number of characters in the quoted string to 80 or less.

-2025 **Description of Error**: The comment close symbol, }, has been found on line *lineno*, character *charposition*, even though no comment has been opened.

System Action Taken: The compile was not completed.

Corrective Action: Comments must be enclosed within a pair of { and } braces. Remove the close comment symbol if it is unnecessary or insert an open comment symbol where appropriate.

-2027 **Description of Error**: An illegal (invisible, control) character has been found on line *lineno*, character *charposition*. It has been replaced by a blank in the listing, but it is still in the source (input) table, and should be removed before attempting to compile again.

System Action Taken: The compile was successful.

Corrective Action: Remove the illegal character from the form specification file before attempting the next compile. If running under UNIX you can use the **od -c** *form-name* command to generate octal code that can help isolate the error. See the *UNIX Programmers Manual* for more information about this command.

-2030 **Description of Error**: A typographical error has been found on line *lineno*, character *charposition*.

System Action Taken: The compile was not completed.

Corrective Action: Edit the form specification file where indicated and correct the error.

-2032 **Description of Error**: The number above could not be successfully converted to either an INTEGER or a DOUBLE or a LONG.

System Action Taken: The compile was not completed.

Corrective Action: **FORM4GL** could not convert the number provided. Acceptable LONG values are whole numbers between -2,147,483,647 and 2,147,483,647. Check that the number does not exceed these values (if a fixed point number) or that it does not contain an error (if a decimal number).

-2040 **Description of Error**: The table name *table-name* exceeds the maximum length of 10 characters.

System Action Taken: The compile was not completed.

Corrective Action: Check the spelling of *table-name*.

-2041 **Description of Error**: The source table *table-name* cannot be opened. This is probably because the table does not exist.

System Action Taken: The compile was not completed.

Corrective Action: Check the spelling of *table-name*.

-2044 **Description of Error**: A table needed by **FORM4GL** is locked by another user.

System Action Taken: The compile was not completed.

Corrective Action: Wait a few minutes and try to recompile, or ask the user of the table to unlock the table. If you are certain no one is using the file, you can run an UNLOCK TABLE statement.

-2045 **Description of Error**: A table needed by **FORM4GL** is exclusively locked by another user.

System Action Taken: The compile was not completed.

Corrective Action: Wait a few minutes and try to recompile, or ask the user of the table to unlock the table. If you are certain no one is using the file, you can run an UNLOCK TABLE statement.

-2800 **Description of Error**: The first line of the specification must be the keyword DATABASE followed by the database name.

System Action Taken: The compile was not completed.

Corrective Action: Check the spelling of the first line of the form specification file, or check that the keyword DATABASE is followed by the database name or the FORMONLY keyword.

-2810 **Description of Error**: The name *database-name* is not an existing database name.

System Action Taken: The compile was not completed.

Corrective Action: Check the spelling of *database-name*. Check that *database-name* exists in your current directory or a directory included in your DBPATH environment variable.

-2811 **Description of Error**: The temporary table *table-name* could not be opened for writing.

System Action Taken: The compile was not completed (operating system error).

Corrective Action: Check that you have operating system write permission to create the file. Contact your System Administrator if you need help with this action.

-2812 **Description of Error**: The temporary table *table-name* could not be read.

System Action Taken: The compile was not completed (operating system error).

Corrective Action: Check that you have operating system read permission to access the file. Contact your System Administrator if you need help with this action.

-2820 **Description of Error**: The label name between brackets is incorrectly given or the label is missing.

System Action Taken: The compile was not completed.

Corrective Action: Check that the label name exists and is correctly spelled.

-2830 **Description of Error**: A left bracket has been found on this line, with no right bracket to match it.

System Action Taken: The compile was not completed.

Corrective Action: A set of brackets [ ] is used to delimit the field size of each field. Insert a right square bracket where appropriate into the form specification file. Note that a display field cannot be split across lines. For long character fields you can use subscripts to create more than one display field.

-2840 **Description of Error**: The label *label-name* was not defined in the form.

System Action Taken: The compile was not completed.

Corrective Action: Check that *label-name* is included in both the SCREEN and ATTRIBUTES sections of the form specification file, or delete the unnecessary *label-name*. (This error often accompanies errors -2820 and -2975.)

-2850 **Description of Error**: The name *column-name* is not a column name in this database.

System Action Taken: The compile was not completed.

Corrective Action: Check the spelling of the *column-name* column name.

-2856 **Description of Error**: The TODAY attribute may be assigned only to date columns.

System Action Taken: The compile was not completed.

Corrective Action: Check that the field in which the TODAY attribute has been used is a DATE column. If appropriate, remove the TODAY attribute from the ATTRIBUTES section of the form.

-2859 **Description of Error**: The column *column-name* is a member of more than one table—you must specify the table name.

System Action Taken: The compile was not completed.

Corrective Action: The column name *column-name* appears in more than one table used in the form. You must specify the table from which *column-name* is to be accessed, using the format *table-name.column-name.*

-2860 **Description of Error**: There is a column/value type mismatch for *column-name.*

System Action Taken: The compile was not completed.

Corrective Action: Check that the value provided as the DEFAULT attribute or listed in the INCLUDE list matches the data type of the column (for example, DATE for a date column, or INTEGER for an integer).

-2861 **Description of Error**: You have exceeded the maximum of 20 tables.

System Action Taken: The compile was not completed.

Corrective Action: A maximum of 20 (this number may be larger on some systems) tables can be in use at any one time. Reduce the number of tables included in the form.

-2862 **Description of Error**: The table *table-name* cannot be found in the database.

System Action Taken: The compile was not completed.

Corrective Action: Check the spelling of *table-name*.

-2863 **Description of Error**: The column *column-name* does not exist among the specified tables.

System Action Taken: The compile was not completed.

Corrective Action: Check the spelling of the column name or check that *column-name* does exist in one of the specified tables.

-2864 **Description of Error**: The table *table-name* is not among the specified tables.

System Action Taken: The compile was not completed.

Corrective Action: Check the spelling of *table-name* in the ATTRIBUTES section of the file. Check that the table is specified in the TABLES section of your form.

-2865 **Description of Error**: The column *column-name* does not exist in the table *table-name*.

System Action Taken: The compile was not completed.

Corrective Action: Check the spelling of *column-name*.

-2870 **Description of Error**: The subscripted column size does not match the space allocated in the display field.

System Action Taken: The compile was not completed.

Corrective Action: Check that the space provided in the display field is greater than or equal to the subscripted column size.

-2880 **Description of Error**: The word "screen" has been left out.

System Action Taken: The compile was not completed.

Corrective Action: The keyword SCREEN must appear in all form specification files, and must begin in the first column on the line. Check the spelling and location of the SCREEN keyword or edit your statement to include it.

-2890 **Description of Error**: A screen definition must begin with a left brace {.

System Action Taken: The compile was not completed.

Corrective Action: Each screen layout must be enclosed within a pair of braces { }. Edit your statement to include the necessary left brace. Note that the left brace must appear in the first character position on the line.

-2892 **Description of Error**: The column *column-name* name appears more than once. If you wish a column to be duplicated in a form, use the same display field label.

System Action Taken: The compile was not completed.

Corrective Action: The column name *column-name* appears more than once in the ATTRIBUTES section of the form specification file. Use the same display field label (in the SCREEN section) to denote the repeated column, and remove any duplicate column names from the ATTRIBUTES section.

-2893 **Description of Error**: The display field label *field-tag* appears more than once in this form, but the lengths are different.

System Action Taken: The compile was not completed.

Corrective Action: A display field label may appear more than once in the SCREEN section, but in every instance the display fields must have identical lengths. Edit the field delimiters so that they are of equal length.

-2895 **Description of Error**: Display field length of number does not match the database column length of number. This is a warning only.

System Action Taken: The compile was completed.

Corrective Action: Check that the display field length (included in the SCREEN section) is equal to the table column size. (This error occurs only in character fields and with the **-v** option to **FORM4GL**.)

-2921 **Description of Error**: The database *database-name* is not compatible with the current version of RDSQL.

System Action Taken: The compile was not completed.

Corrective Action: The database was created under a prior version of INFORMIX. You must first convert the database, using **dbconvert,** before attempting to compile.

-2930 **Description of Error**: Portions of the column *column-name* are displayed on the screen more than once.

System Action Taken: The compile was not completed.

Corrective Action: Check the subscripting of *column-name* (present in the ATTRIBUTES section of the form specification file). Subscripts cannot overlap (for example, [25–49] and [50–75] are acceptable; [25–50] and [50–75] are unacceptable, as character 50 would have to appear twice).

-2931 **Description of Error**: There is an error in the format specification.

System Action Taken: The compile was not completed.

Corrective Action: You may only use the FORMAT attribute with a DECIMAL, SMALLFLOAT, FLOAT or DATE column to control the format of the display. Check the format specifications for errors.

-2932 **Description of Error**: Formats may be specified only for DECIMAL, SMALLFLOAT, FLOAT or DATE columns.

System Action Taken: The compile was not completed.

Corrective Action: You may only use the FORMAT attribute with a DECIMAL, SMALLFLOAT, FLOAT or DATE column to control the format of the display. Check that you have not specified the format on a CHAR, INTEGER, or SMALLINT column type.

-2933 **Description of Error**: The format width is larger than the allocated display width.

System Action Taken: The compile was not completed.

Corrective Action: Check that the format of a DECIMAL, SMALLFLOAT, FLOAT, or DATE column matches the length of the display field in the form.

-2934 **Description of Error**: The format width is less than the allocated display width. This is a warning only.

System Action Taken: The compile was completed.

Corrective Action: Check that the format of a DECIMAL, SMALLFLOAT, FLOAT, or DATE column matches the length of the display field in the form. (Note: Until this error is corrected any data displayed in the field may be truncated.)

-2940 **Description of Error**: The column *column-name* appears both with and without subscripts.

System Action Taken: The compile was not completed.

Corrective Action: Only column-names that are subscripted can appear more than once in the ATTRIBUTES section. Remove subscripts from *column-name* and all redundant references to it, or add subscripts to each.

-2943 **Description of Error**: You have exceeded the pseudo machine capacity.

System Action Taken: The compile was not completed.

Corrective Action: Reduce the complexity and/or number of the instructions in the form.

-2950 **Description of Error**: The column *column-name* has no section that starts at 1. Remember that the first subscript is one, not zero.

System Action Taken: The compile was not completed.

Corrective Action: Edit the form specification file so that the subscript to the column *column-name* begins with 1 (and not 0).

-2951 **Description of Error**: The left and right delimiters must be specified in a two-character string.

System Action Taken: The compile was not completed.

Corrective Action: When changing the delimiters that INFORMIX-4GL uses to enclose the display fields in the SCREEN section of a form, you must specify both the left and right delimiters with one character each.

-2952 **Description of Error**: In order to use a picture, the picture length must be the same as the display field length.

System Action Taken: The compile was not completed.

Corrective Action: Edit the file so that the length of the picture specified with the PICTURE attribute equals the display field length in the SCREEN section.

-2955 **Description of Error**: The name *field-tag* is not a displayed field in this form.

System Action Taken: The compile was not completed.

Corrective Action: The display field *field-tag* has been specified in the ATTRIBUTES section of the form specification file, but the *field-tag* is not included in the SCREEN section of the form. Delete the *field-tag* from the ATTRIBUTES section or include it in the SCREEN section.

-2971 **Description of Error**: This column is not a character column, and therefore cannot be subscripted.

System Action Taken: The compile was not completed.

Corrective Action: Remove subscripting from any non-CHARACTER columns in your form.

-2975 **Description of Error**: The display field label *field-tag* has not been used.

System Action Taken: The compile was not completed.

Corrective Action: A display field label *field-tag* present in the SCREEN section of the form specification file does not correspond to a column in the ATTRIBUTES section. Delete the display field label *field-tag* if it is unnecessary or include a reference to it in the ATTRIBUTES section.

-2988 **Description of Error**: **FORM4GL** has run out of memory.

System Action Taken: The compile was not completed.

Corrective Action: You have exceeded the data space limit on your machine. Reduce the complexity of the form specification file.

-2992 **Description of Error**: The display label *field-tag* has already been used.

System Action Taken: The compile was not completed.

Corrective Action: Each display label tag must be unique. Select a new display label tag.

-2996 **Description of Error**: The unanticipated error number *error-number* has occurred.

System Action Taken: The compile was not completed (operating system error).

Corrective Action: *error-number* is an operating system error number. If running under UNIX, check the *UNIX Programmers Manual* for error information relating to *error-number*. Contact your System Administrator if you need assistance with this action. You might also contact the Technical Support Department of the company that supplied your operating system for advice about this error number.

-2997 **Description of Error**: See error number *errno*.

System Action Taken: The compile was not completed.

Corrective Action: Locate the indicated error message in this appendix.

-2998 **Description of Error**: Operating system error *error-number*: *string*.

System Action Taken: The compile was not completed (operating system error).

Corrective Action: If the error number is between 1–99 and you are using a UNIX-based system, check your *UNIX Programmers Manual* for the error corresponding to the number indicated. Contact your System Administrator if you need assistance with this action. If you are using a DOS system, call RDS Technical Support.

-4300 **Description of Error**: This statement contains too many levels of function call nesting.

System Action Taken: The statement containing the error was not processed.

Corrective Action: A CALL statement can contain only four levels of nested functions. Reduce the number of nested functions in your CALL statement so that it does not exceed four.

-4301 **Description of Error**: The program has too many levels of WHILE, FOR, MENU, and/or CASE statements.

System Action Taken: The statement containing the error was not processed.

Corrective Action: A program can contain only 25 levels of WHILE, FOR, MENU, and/or CASE statements (in any combination). Reduce the number of nested WHILE, FOR, MENU, and/or CASE statements so that it does not exceed 25.

-4302 **Description of Error**: The record description is nested too deep.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Only five levels of nested records can appear in a record definition. Reduce the number of nested records so that it does not exceed five.

-4304 **Description of Error**: A different database has already been declared. If your program uses a global definition file, it must contain the same database name as this one.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Make sure the database specified in a global definition file is the same as the database specified in the file containing the main program.

-4305 **Description of Error**: The database *database-name* cannot be found or opened. If the database exists, check the database permissions on the database. In addition, check the system permissions on the database directory and its ascendant directories.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Check the spelling of the database name. Check that the database name exists in your current directory or in a directory included in your DBPATH environment variable. Make sure you have at least CONNECT permission for the database as well as the appropriate operating system permissions for the database directory.

-4306 **Description of Error**: The GLOBALS file *filename* cannot be opened for reading.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Check that the GLOBALS file exists and that you have operating system read permission for the file. If the GLOBALS file is not in the current directory, you must specify the pathname of the file from root or from the current directory.

-4307 **Description of Error**: The number of variables and/or constants in the display list does not match the number of form fields in the display destination.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Check the *variable-list* and the *field-list* in your DISPLAY or DISPLAY ARRAY statement to make sure they contain the same number of items.

-4308 **Description of Error**: The number of input variables does not match the number of form fields in the screen input list.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Check the *variable-list* and the *field-list* in your INPUT or INPUT ARRAY statement to make sure they contain the same number of items.

-4309 **Description of Error**: Printing cannot be done within a loop or CASE statement contained in report headers or trailers.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Do not include PRINT statements within FOR, WHILE, or CASE statements that appear in FIRST PAGE HEADER, PAGE HEADER, and PAGE TRAILER control blocks.

-4310 **Description of Error**: Files cannot be printed within report headers or trailers.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Do not include PRINT FILE statements within FIRST PAGE HEADER, PAGE HEADER, and PAGE TRAILER control blocks.

-4311 **Description of Error**: The variable *variable-name* was not defined as a record. It cannot be used in this fashion.

System Action Taken: The statement containing the error was not processed.

Corrective Action: You can use the THRU, THROUGH, or .* notations only with records. Define the variable as a record or remove the THRU, THROUGH, or .* notation, as appropriate.

-4312 **Description of Error**: The NEED statement is allowed only within reports.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Make sure that the NEED statement only occurs within the FORMAT section of a report.

-4313 **Description of Error**: The NEED statement cannot be used within report headers or trailers.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Remove any NEED statements from FIRST PAGE HEADER, PAGE HEADER, and PAGE TRAILER control blocks.

-4314 **Description of Error**: The program cannot exit a menu at this point because it is not within a MENU statement.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Make sure that the EXIT MENU keywords appear only within a COMMAND clause of the MENU statement.

-4315 **Description of Error**: The program cannot exit a FOREACH statement at this point because it is not within a FOREACH statement.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Make sure that the EXIT FOREACH keywords appear only within a FOREACH statement.

-4316 **Description of Error**: The program cannot exit a WHILE statement at this point because it is not within a WHILE statement.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Make sure that the EXIT WHILE keywords appear only within a WHILE statement.

-4317 **Description of Error**: The program cannot exit a FOR statement at this point because it is not within a FOR statement.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Make sure that the EXIT FOR keywords appear only within a FOR statement.

-4318 **Description of Error**: The program cannot exit a CASE statement at this point because it is not within a CASE statement.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Make sure that the EXIT CASE keywords appear only within a CASE statement.

-4319 **Description of Error**: The symbol *variable-name* has been defined more than once.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Make sure that each variable is defined only once in your GLOBALS, MAIN, FUNCTION, or REPORT statement.

-4320 **Description of Error**: The symbol *table-name* is not the name of a table in the specified database.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Check the spelling of *table-name* and make sure it is a table in the specified database.

-4321 **Description of Error**: An array may have the maximum of three dimensions.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Make sure the array has no more than three dimensions.

-4322 **Description of Error**: The symbol *column-name* is not the name of a column in the specified database.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Check the spelling of *column-name* and make sure that it appears in a table of the specified database.

-4323 **Description of Error**: The variable *variable-name* is too complex a type to be used in an assignment statement.

System Action Taken: The statement containing the error was not processed.

Corrective Action: You can assign a value only to a variable that has a simple type. If you are assigning a value to an element of an array or record, make sure it has a simple type.

-4324 **Description of Error**: The variable *variable-name* is not a character type, and cannot be used to contain the result of concatenation.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Change the data type of the variable to CHAR so that it can store the string that results from concatenation.

-4325 **Description of Error**: The source and destination records in this record assignment statement are not compatible in types and/or lengths.

System Action Taken: The statement containing the error was not processed.

Corrective Action: In the statement `LET a.* = b.*`, each element in record `a` must have a data type which is compatible with the data type of the corresponding element in record `b`.

-4326 **Description of Error**: A NULL value may not be applied to substrings.

System Action Taken: The statement containing the error was not processed.

Corrective Action: A NULL value may be assigned to a string but not to a substring. Remove the subscript(s) if you want to assign a NULL value to the string.

-4327 **Description of Error**: The variable *variable-name* is not of type INTEGER or SMALLINT. It cannot be used as a loop index.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Change the type of the variable to INTEGER or SMALLINT if you want to use it as a loop index.

-4328 **Description of Error**: The variable *variable-name* has too complex a type to be used as the destination of a return from a function.

System Action Taken: The statement containing the error was not processed.

Corrective Action: You must return values to variables that have simple types. If you have specified a record in the RETURNING clause of a CALL statement, remember to include the THRU, THROUGH, or .* shorthand after the record name and make sure that all the record elements referenced in this way have simple types.

-4329 **Description of Error**: The variable *variable-name* is not a record. Only record variables may be expanded using the .* or THROUGH shorthand.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Change the variable to a record or eliminate the THROUGH, THRU, or .* notation as appropriate.

-4330 **Description of Error**: RETURN statements can be executed only within functions.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Make sure that the RETURN keyword appears only within a FUNCTION statement.

-4331 **Description of Error**: Only variables of type INTEGER or SMALLINT may be used to index display fields.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Change the type of the variable to INTEGER or SMALLINT if you want to use it to index display fields.

-4332 **Description of Error**: The LET statement must have at least one source expression.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Make sure that one or more valid expressions appears to the right of the equal sign (=) in a LET statement.

-4333 **Description of Error**: The function *function-name* has already been called with a different number of parameters.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Make sure you specify the same number of parameters each time you call a function.

-4334 **Description of Error**: The variable *variable-name* in its current form is too complex to be used in this statement.

System Action Taken: The statement containing the error was not processed.

Corrective Action: You can use only variables that have simple types. If you have specified a record in the statement, remember to include the THRU, THROUGH, or .* shorthand after the record name and make sure that all the record elements referenced in this way have simple types.

-4335 **Description of Error**: The symbol *variable-name* is not an element of the record *record-name*.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Make sure that the variable is listed as an element of the record in the appropriate DEFINE statement.

-4336 **Description of Error**: The parameter *variable-name* has not been defined within the function or report.

System Action Taken: The statement containing the error was not processed.

Corrective Action: You must use the DEFINE statement to define all parameters passed as arguments to a function or report.

-4338 **Description of Error**: The symbol *variable-name* has already been defined once as a parameter.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Make sure that each parameter is defined only once in a function or report.

-4339 **Description of Error**: 4GL has run out of data space memory.

System Action Taken: The program stops and running transactions are rolled back.

Corrective Action: Divide your program into smaller modules or reduce the complexity of the program.

-4340 **Description of Error**: The variable *variable-name* is too complex a type to be used in an expression.

System Action Taken: The statement containing the error was not processed.

Corrective Action: You can use only variables that have simple types in expressions. If you have specified an element of a record or array, make sure it has a simple type.

-4341 **Description of Error**: Aggregate functions are only allowed in reports and SELECT statements.

System Action Taken: The statement containing the error was not processed.

Corrective Action: You can use aggregate functions like SUM and AVG only in reports and in expressions that appear in SELECT statements.

-4342 **Description of Error**: PAGENO and LINENO are allowed only in reports.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Make sure that the PAGENO and LINENO statements only occur within the FORMAT section of a report.

-4343 **Description of Error**: Subscripting cannot be applied to the variable *variable-name* because it is not a character or array variable.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Check the spelling of the variable name. Define *variable-name* as a character or array variable or remove the subscript, as appropriate.

-4344 **Description of Error**: The variable *variable-name* cannot be used with substrings because it is not a character variable.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Check the data type of the variable. If *variable-name* is not a character variable, remove the subscript(s) or change the type of the variable, as appropriate.

-4345 **Description of Error**: The variable *variable-name* has already had substrings applied to it.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Rewrite the statement so that only one substring (represented by the notation [*a*, *b*]) appears after the variable name.

-4346 **Description of Error**: Subscripts may contain only INTEGER or SMALLINT variables.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Check the spelling of the variable name(s). Make sure each variable is defined as an INTEGER or SMALLINT if you want to use it as a subscript.

-4347 **Description of Error**: The variable *variable-name* is not a record. It cannot reference record elements.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Check the spelling of the variable name. Make sure that the variable is defined as a record before you add a suffix (*.variable-name*) to it.

-4348 **Description of Error**: This type of aggregate must be applied to an expression, not '*'. Only PERCENT and COUNT aggregates use '*'.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Use an expression instead of an asterisk (*) with the SUM, AVG, MIN, and MAX aggregates.

-4349 **Description of Error**: The PERCENT and COUNT report aggregates cannot be used with an expression.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Use an asterisk (*) instead of an expression with the PERCENT and COUNT aggregates.

-4350 **Description of Error**: The program cannot continue a FOR loop at this time because it is not within a FOR loop.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Make sure the CONTINUE FOR keywords appear only within a FOR statement.

-4351 **Description of Error**: The program cannot continue a WHILE loop at this time because it is not within a WHILE loop.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Make sure the CONTINUE WHILE keywords appear only within a WHILE statement.

-4352 **Description of Error**: The program cannot continue a FOREACH loop at this time because it is not within a FOREACH loop.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Make sure the CONTINUE FOREACH keywords appear only within a FOREACH statement.

-4356 **Description of Error**: A page header has already been specified within this report.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Change the FORMAT section of your report so that it contains only one PAGE HEADER control block.

-4357 **Description of Error**: A page trailer has already been specified within this report.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Change the FORMAT section of your report so that it contains only one PAGE TRAILER control block.

-4358 **Description of Error**: A first page header has already been specified within this report.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Change the FORMAT section of your report so that it contains only one FIRST PAGE HEADER control block.

-4359 **Description of Error**: An ON EVERY ROW clause has already been specified within this report.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Change the FORMAT section of your report so that it contains only one ON EVERY ROW control block.

-4360 **Description of Error**: An ON LAST ROW clause has already been specified within this report.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Change the FORMAT section of your report so that it contains only one ON LAST ROW control block.

-4361 **Description of Error**: Group aggregates can occur only in AFTER GROUP clauses.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Make sure the GROUP COUNT, GROUP PERCENT, GROUP SUM, GROUP AVG, GROUP MIN, and GROUP MAX aggregates appear only in AFTER GROUP control blocks.

-4362 **Description of Error**: The report cannot skip to the top of page while in a header or trailer.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Remove SKIP TO TOP OF PAGE statements from FIRST PAGE HEADER, PAGE HEADER, and PAGE TRAILER control blocks.

-4363 **Description of Error**: The report cannot skip lines while in a loop within a header or trailer.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Remove SKIP statements from FOR and WHILE loops that appear in FIRST PAGE HEADER, PAGE HEADER, and PAGE TRAILER control blocks.

-4364 **Description of Error**: There are a non-matching number of variables and database columns in this statement.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Make sure that the number of variables in the statement is the same as the number of database columns.

-4365 **Description of Error**: Deferments of interrupt or quit may be executed only in the main program.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Make sure that DEFER INTERRUPT and DEFER QUIT statements appear only in the MAIN section of your program.

-4366 **Description of Error**: There are a non-matching number of variables and database columns in this statement.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Make sure that the number of variables in the statement is the same as the number of database columns.

-4367 **Description of Error**: Interrupt has already been deferred once in the main program. Each main program may defer interrupt only once.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Make sure that the DEFER INTERRUPT statement appears only once and then only in the MAIN section of your program. (Once deferred, interrupts cannot be reactivated.)

-4368 **Description of Error**: Quit has already been deferred once in the MAIN section of your program. Each main program may defer quit only once.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Make sure that the DEFER QUIT statement appears only once and then only in the MAIN section of your program. (Once deferred, quits cannot be reactivated.)

-4369 **Description of Error**: The symbol *variable-name* does not represent a defined variable.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Make sure you define the variable in a DEFINE statement in the appropriate part of your program.

-4370 **Description of Error**: The variable *variable-name* cannot be used in validation.

System Action Taken: The statement containing the error was not processed.

Corrective Action: You can validate only variables that have simple types. If you have specified a record in the statement, remember to include the THRU, THROUGH, or .* notation after the record name, and make sure that all record elements referenced in this way have simple types.

-4371 **Description of Error**: Cursors must be uniquely declared within one program module.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Make sure that each cursor is declared only once in a program file.

-4372 **Description of Error**: The cursor *cursor-name* has not yet been declared in this program module. It must be declared before it can be used.

System Action Taken: The statement containing the error was not processed.

Corrective Action: You must use the DECLARE statement to declare a cursor in each module before you can use it in statements such as FOREACH, OPEN, and FETCH.

-4373 **Description of Error**: A grammatical error has been found on line *line-number*, character *character-number*. The construct is not understandable in its context.

System Action Taken: The statement containing the error was not processed.

Corrective Action: When INFORMIX-4GL encounters a grammatical error, it inserts a marker in the **.err** file just past the point where the parser detected the error. Check the syntax of the marked statement.

-4375 **Description of Error**: The page length is too short to cover the specified page header and trailer lengths.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Make sure that the total number of lines required for the page header and trailer do not exceed the default page length or the length specified in the PAGE LENGTH statement. Change the page header and trailer or the page length as appropriate.

-4376 **Description of Error**: The temporary file *filename* cannot be created for writing.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Check that you have permission to write a file in **/tmp** or the directory specified by the DBTEMP environment variable. Check that there is enough space in the directory where the temporary file will reside. Contact your System Administrator if you need help with these actions.

-4377 **Description of Error**: The output file *filename* cannot be created or opened.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Check that you have permission to write a file in the directory where the output file will be created. Contact your System Administrator if you need help with this action.

-4378 **Description of Error**: No input file was specified.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Make sure you specify an input filename.

-4379 **Description of Error**: The input file *filename* cannot be opened.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Check the spelling of the input filename. Check that the file exists in the current directory. Check that you have operating system read permission for the input file.

-4380 **Description of Error**: The listing file *filename* cannot be created.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Check that you have operating system write permission in the directory where the listing file will be created. Contact your System Administrator if you need help with this action.

-4381 **Description of Error**: The input file *filename* has an invalid extension. The filename must have **.4gl** as the extension.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Rename the input file so that it has the extension **.4gl**.

-4382 **Description of Error**: Record variables that contain array type elements may not be referenced by the ".*" or THROUGH shorthand, or used as a function parameter.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Rewrite your statement so that record variables with array components do not appear with the THRU, THROUGH, or .* shorthands or as function parameters.

-4383 **Description of Error**: The elements *element-name1* and *element-name2* do not belong to the same parent record.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Check the spelling of the element names and make sure that both elements belong to the same record.

-4384 **Description of Error**: The symbol *element-name* does not represent the element of any record.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Check the spelling of *element-name* and make sure it belongs to the specified record.

-4385 **Description of Error**: Report aggregates may not be nested.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Rewrite your statement so that report aggregates are not nested.

-4386 **Description of Error**: There are too many ORDER BY fields in this report. The maximum number is eight.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Reduce the number of ORDER BY fields to eight or less.

-4387 **Description of Error**: The right margin must be greater than the left margin.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Check that the RIGHT MARGIN value is greater than the LEFT MARGIN value.

-4388 **Description of Error**: There is one BEFORE GROUP OF clause and one AFTER GROUP OF clause allowed for each report input parameter.

System Action Taken: The statement containing the error was not processed.

Corrective Action: You can use only one BEFORE GROUP OF control block and one AFTER GROUP OF control block for each report parameter. You must combine multiple BEFORE GROUP OF or AFTER GROUP OF control blocks for the same parameter into a single control block.

-4389 **Description of Error**: There are too many levels of nesting of IF statements in this report.

System Action Taken: The statement containing the error was not processed.

Corrective Action: You have exceeded the maximum of five levels of nested IF statements. Remove one or more statements.

-4391 **Description of Error**: When doing INPUT BY NAME or INPUT ARRAY, the BEFORE/AFTER field names can be specified only by the field name suffix. Screen array and screen record elements are not allowed.

System Action Taken: The statement containing the error was not processed.

Corrective Action: You cannot include a prefix when specifying a field name in a BEFORE FIELD or AFTER FIELD clause of an INPUT or INPUT ARRAY statement. (A prefix consists of *table-name*, *formonly*, *screen-record*, or *screen-record*[*n*] followed by a period.)

-4392 **Description of Error**: The 4GL compiler has run out of data space memory to contain the 4GL program symbols. If the program module is very large, dividing it into separate modules may alleviate the situation.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Divide your program into smaller programs or reduce the complexity of your program.

-4393 **Description of Error**: The MENU statement has exceeded the maximum number of selections.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Reduce the number of COMMAND clauses in the MENU statement so that it does not exceed the limit of 25.

-4394 **Description of Error**: The MENU statement has two or more selections using the *key-name* key.

System Action Taken: The statement containing the error was not processed.

Corrective Action: The user selects a menu option by typing one of the letters in a key list (if a KEY clause for the menu option is present) or by typing the first letter of the menu option (if a KEY clause for the option is not present). Rewrite the MENU statement so that the letter for selecting each menu option is unique.

-4395 **Description of Error**: There are too many subscripts specified with a database column name.

System Action Taken: The statement containing the error was not processed.

Corrective Action: You can use no more than two subscripts with database columns of type CHAR.

-4396 **Description of Error**: The MENU declaration at line *lineno* is not terminated.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Make sure you conclude the MENU statement with the END MENU keywords.

-4397 **Description of Error**: The IF statement at line *lineno* is not terminated.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Make sure you conclude the IF statement with the END IF keywords.

-4398 **Description of Error**: The CASE statement at line *lineno* is not terminated.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Make sure you conclude the CASE statement with the END CASE keywords.

-4399 **Description of Error**: The WHILE statement at line *lineno* is not terminated.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Make sure you conclude the WHILE statement with the END WHILE keywords.

-4400 **Description of Error**: The FOR statement at line *lineno* is not terminated.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Make sure you conclude the FOR statement with the END FOR keywords.

-4401 **Description of Error**: A concatenation operation has created a string too long to fit in the destination string variable.

System Action Taken: The statement containing the error was not processed.

Corrective Action: When possible, use the CLIPPED keyword to eliminate trailing blanks from the strings you want to concatenate. If the resultant string still exceeds the length of the character variable, increase the size of the character variable.

-4402 **Description of Error**: In this type of statement, subscripting may be applied only to array variables to select individual array elements.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Make sure the variable is defined as an array before you use it with subscripts in this type of statement.

-4403 **Description of Error**: The number of dimensions for the variable *variable-name* does not match the number of subscripts.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Rewrite the statement so that the number of subscripts after the array name is the same as the number of dimensions in the array definition.

-4405 **Description of Error**: The function has exceeded the maximum number of allowed parameters.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Reduce the number of parameters in the FUNCTION statement so that it does not exceed the limit of 50.

-4406 **Description of Error**: There is an unmatched quote in the above line.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Check that all strings begin and end with a quote.

-4407 **Description of Error**: There is an unprintable character in the above line.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Remove the unprintable character (usually a control character). You may have to retype the line.

-4408 **Description of Error**: There is a quoted string that is too long in the above line.

System Action Taken: The statement containing the error was not processed.

Corrective Action: A quoted string cannot exceed 80 characters. Reduce the length of the quoted string or, if appropriate, divide it into shorter strings separated by commas.

-4409 **Description of Error**: There is an invalid character in the above line.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Remove the invalid character (often a non-printable control character).

-4410 **Description of Error**: There is a number that is too long in the above line.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Make sure that no number is longer than 50 characters.

-4411 **Description of Error**: There is an alphanumeric identifier that is too long in the above line.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Make sure that an alphanumeric identifier is no longer than 50 characters.

-4412 **Description of Error**: Values from the RUN command can be returned only to INTEGER or SMALLINT variables.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Make sure that the variables that appear in the RETURNING clause of a RUN statement are defined as INTEGER or SMALLINT.

-4413 **Description of Error**: The label *label-name* has already been defined within this MAIN program or function.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Make sure that each label is defined only once in the MAIN program or function.

-4414 **Description of Error**: The label *label-name* has been used but has never been defined within the above main program or function.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Make sure you define each label with the LABEL statement before using it in the MAIN program or function.

-4415 **Description of Error**: An ORDER BY or GROUP item specified within a report must be one of the report parameters.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Make sure that only report parameters appear in ORDER BY, BEFORE GROUP OF, or AFTER GROUP OF statements in a report. You cannot use global or local variables.

-4416 **Description of Error**: There is an error in the validation string: *"string"*.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Change the appropriate DEFAULT or INCLUDE value in the **syscolval** table.

-4417 **Description of Error**: This type of statement can be used only in a report.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Make sure statements like PRINT, SKIP, and NEED appear only in a REPORT statement.

-4418 **Description of Error**: The variable used in the INPUT ARRAY statement must be an array.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Check the spelling of the variable name and make sure it has been defined as an array.

-4419 **Description of Error**: The variable used in the CONSTRUCT statement must be a character variable.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Make sure the variable that appears after the CONSTRUCT keyword has been defined as a large CHAR variable.

-4420 **Description of Error**: The number of lines printed in the IF part of an IF-THEN-ELSE statement of a header or trailer clause must equal the number of lines printed in the ELSE part.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Add or remove lines as necessary so that number of lines printed in the IF part is the same as the number of lines printed in the ELSE part of the IF-THEN-ELSE statement.

-4421 **Description of Error**: You may not use an INPUT statement within another INPUT statement or PROMPT statement, even if it is enclosed within a conditional or looping statement.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Remove any INPUT statements that appear within an INPUT or PROMPT statement.

-4422 **Description of Error**: You may not use a CONSTRUCT statement within another INPUT statement. This includes situations when CONSTRUCT is enclosed within a conditional or looping statement. You must call a function that executes the CONSTRUCT statement.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Move any CONSTRUCT statement within an INPUT statement to a function and call that function from the INPUT statement.

-4423 **Description of Error**: The CLIPPED and USING options for the DISPLAY statement may not be used when displaying to a form field.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Remove all references to CLIPPED and USING from DISPLAY TO or DISPLAY BY NAME statements.

-4424 **Description of Error**: The variable *variable-name* has not been defined as a record.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Make sure the variable is defined as a record before using it with the THRU, THROUGH, or **.*** notation.

-4425 **Description of Error**: The variable *variable-name* has not been defined LIKE the table *table-name.*

System Action Taken: The statement containing the error was not processed.

Corrective Action: Define the variable with the RECORD LIKE keywords if you want to use it in an UPDATE statement.

-4426 **Description of Error**: The PRINT statement may be used only within reports. If you wish to print without screen positioning, use the DISPLAY statement without any field or screen destination.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Replace all PRINT statements that appear outside of reports with DISPLAY statements. Use the DISPLAY TO or DISPLAY BY NAME statement to display information in a display field on a screen form, the DISPLAY AT statement to display information at a specified row and column on the screen, or the DISPLAY statement to display information without screen positioning.

-4427 **Description of Error**: The COLUMN feature for the DISPLAY statement may be used only when displaying without screen or field destination.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Remove the COLUMN function from the display list of any DISPLAY AT, DISPLAY TO, or DISPLAY BY NAME statement.

-4428 **Description of Error**: You may not use a PROMPT statement within an INPUT or PROMPT statement, even if it is enclosed within a conditional or looping statement.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Remove any PROMPT statements that appear within an INPUT or PROMPT statement.

-4429 **Description of Error**: Report and function parameters cannot be arrays.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Remove any arrays from the parameter lists of functions or reports.

-4430 **Description of Error**: Record parameters for a report cannot contain elements that are arrays.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Rewrite your program so that record parameters for reports do not contain array elements.

-4431 **Description of Error**: The number of expanded report parameters has exceeded the maximum allowed.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Make sure that the total number of parameters (including elements that result when INFORMIX-4GL expands a record that uses the THRU, THROUGH, or .* notation) is less than the maximum of 150.

-4432 **Description of Error**: An element in a GROUP clause must be a member of the ORDER BY clause.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Rewrite the statement so that each element in a BEFORE GROUP OF or AFTER GROUP OF clause also appears in the ORDER BY clause.

-4433 **Description of Error**: A variable used in the above statement must be of type CHAR.

System Action Taken: The statement containing the error was not processed.

Corrective Action: The *filename* in the REPORT TO "filename" statement must evaluate to a CHAR variable, or (on UNIX systems only) the *program* in the REPORT TO PIPE "program" statement must evaluate to a CHAR variable.

-4434 **Description of Error**: The limits of the INFORMIX-4GL Demo Version have been exceeded. Please call Relational Database Systems, Inc. at (415) 322-4100 for licensing information.

System Action Taken: The statement containing the error was not processed.

Corrective Action: A program compiled using the demonstration version of **INFORMIX-4GL** can contain only one module with no more than 150 **INFORMIX-4GL** statements. Check that you have not exceeded the statement limit or called a function that is not included in the module.

Please contact your RDS Sales Representative for information about a full **INFORMIX-4GL** development license.

-4435 **Description of Error**: An acceptable hyphenated key format is "control-x", where x is any letter except a, d, h, l, r, or x.

System Action Taken: The statement containing the error was not processed.

Corrective Action: You cannot redefine a function key as one of the editing keys for an INPUT, INPUT ARRAY, or CONSTRUCT statement. Select an alternate key.

-4436 **Description of Error**: There are too many variables to VALIDATE or INITIALIZE in one statement.

System Action Taken: The statement containing the error was not processed.

Corrective Action: You cannot VALIDATE or INITIALIZE more than 100 variables in a statement. Reduce the number of variables in the statement.

-4437 **Description of Error**: All table names in the SELECT list must be the same as the table names in the FROM clause.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Check that you have not misspelled the name of a table in the SELECT list.

-4438 **Description of Error**: You cannot SELECT into a substring of a character variable.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Remove the substring from the CHAR variable in the INTO clause of the SELECT statement.

-4439 **Description of Error**: You cannot SELECT into record *record-name* because element *element* is a record or an array.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Edit the SELECT statement in your program.

-4440 **Description of Error**: *element-name* precedes *element-name* in record *record-name* and must also precede it when used with the THROUGH shorthand.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Re-order the elements used with the THROUGH keyword to match the order of elements in the record.

-4441 **Description of Error**: The ISAM cursor *cursor-name* has not yet been declared in this program module. It must be declared before it can be used.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Check the spelling of *cursor_name* and that you have physically declared the cursor before making reference to it.

-4442 **Description of Error**: *fetch-direction* is not a recognized row selector in the ISAM FETCH statement.

System Action Taken: The statement containing the error was not processed.

Corrective Action: *fetch-direction* must evaluate to one of the following: FIRST, LAST, CURRENT, RELATIVE *integer*, ABSOLUTE *integer*, NEXT, PRIOR, PREVIOUS, or ROWID *integer*. Check the spelling of *fetch-direction*.

-4443 **Description of Error**: Only constants and variables of type INTEGER or SMALLINT may be used to specify the size and/or position of windows.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Check that the values in the WITH and AT clauses of the OPEN WINDOW statement evaluate to type INTEGER or SMALLINT.

-4444 **Description of Error**: Too many colors specified for window.

System Action Taken: The statement containing the error was not processed.

Corrective Action: You can include only one color in the ATTRIBUTE clause of the OPEN WINDOW statement.

-4445 **Description of Error**: You may not open or close window SCREEN.

System Action Taken: The statement containing the error was not processed.

Corrective Action: You cannot execute an OPEN WINDOW screen or CLOSE WINDOW screen statement. Edit your INFORMIX-4GL program.

-4500 **Description of Error**: A numeric conversion error has occurred due to incompatibility between a calling program and its function parameters or between a variable and its assigned expression.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Make sure that the data types of variables or function parameters are compatible with the types of values assigned to them.

-4501 **Description of Error**: A parameter count mismatch has occurred between the calling function and the called function.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Make sure that the number of parameters in the calling statement is the same as the number of parameters in the called function.

-4502 **Description of Error**: The 4GL program has run out of runtime data space memory.

System Action Taken: The program stops and running transactions are rolled back.

Corrective Action: Divide your program into smaller modules. Reduce the size of arrays and character variables. Make sure that no function returns a character string longer than 512 characters.

-4503 **Description of Error**: A function has not returned the correct number of values expected by the calling function.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Make sure that the number of parameters after the RETURN keyword in the called function is the same as the number of parameters after the RETURNING keyword in the calling statement.

-4504 **Description of Error**: A validation error has occurred as a result of the VALIDATE command.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Make sure that the values of the variables in a VALIDATE statement conform to the values allowed for the corresponding columns in the **syscolval** table. You should trap this error.

-4505 **Description of Error**: A report output file cannot be opened.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Check that you have operating system write permission in the designated directory. Contact your System Administrator if you need help with this action.

-4506 **Description of Error**: A report output pipe cannot be opened.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Check the spelling of the name of the program receiving the output. Check that the program is available on your system. Check that the program exists in a directory accessed in your PATH environment variable. Contact your System Administrator if you need help with these actions.

-4507 **Description of Error**: A report output file cannot be written to.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Check that you have operating system write permission in the designated directory. Contact your System Administrator if you need help with this action.

-4508 **Description of Error**: A report PRINT FILE source file cannot be opened for reading.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Check that the specified file exists. Check that you have operating system read permission for the file. Contact your System Administrator if you need help with this action.

-4509 **Description of Error**: An array variable has been referenced outside of its specified dimensions.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Check that the number of subscripts for the array corresponds to the number of dimensions specified in the array definition. Check that the subscript(s) do not exceed the value(s) specified in the array definition. Compile with the **-a** option to check array dimensions.

-4511 **Description of Error**: A temporary table needed for a report could not be created in the selected database. The user must have permission to create tables in the selected database.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Check that the user has CONNECT privilege for the selected database. Check the error number that appears with this error for more information about the source of the problem.

-4512 **Description of Error**: A database index could not be created for a temporary database table needed for a report.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Check the error number that appears with this error for more information about the source of the problem.

-4513 **Description of Error**: A number used as a DISPLAY AT location or SCROLL count must be positive.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Make sure that you use only positive integer expressions in DISPLAY AT or SCROLL statements.

-4514 **Description of Error**: A row could not be inserted into a temporary report table.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Check the error number that appears with this error for more information about the source of the problem.

-4515 **Description of Error**: A row could not be fetched from a temporary report table.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Check the error number that appears with this error for more information about the source of the problem.

-4516 **Description of Error**: A character variable has referenced subscripts that are out of range.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Make sure that a subscript for a character variable does not exceed the number of characters specified in the variable definition.

-4517 **Description of Error**: Strings of length > 512 cannot be returned from function calls.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Make sure that a character string returned by a function does not exceed 512 characters.

-4518 **Description of Error**: The 4GL program cannot allocate any more space for temporary string storage.

System Action Taken: The statement containing the error was not processed.

Corrective Action: INFORMIX-4GL cannot allocate more than 512 characters for temporary string storage. This error may occur if the nested functions in a CALL statement return strings whose total length exceeds 512 characters.

-4600 **Description of Error**: No form by specified name found.

System Action Taken: INFORMIX-4GL prompts you to enter the name of an existing form.

Corrective Action: Check the spelling of the form name. Check that the form exists in the current directory or in one of the directories specified by DBPATH.

-4601 **Description of Error**: No 4GL module by specified name found.

System Action Taken: INFORMIX-4GL prompts you to enter the name of an existing module.

Corrective Action: Check the spelling of the module name. Check that the module exists in the current directory or in one of the directories specified by DBPATH.

-4602 **Description of Error**: No 4GL program by specified name found.

System Action Taken: INFORMIX-4GL prompts you to enter the name of an existing program.

Corrective Action: Check the spelling of the program name. Check that the program exists in the 4GL program database.

-4603 **Description of Error**: No executable 4GL program by specified name found.

System Action Taken: INFORMIX-4GL prompts you to enter the name of an existing executable program.

Corrective Action: Check the spelling of the program name. Check that the executable program exists in the current directory or in one of the directories specified by DBPATH.

-4604 **Description of Error**: Error(s) found in 4GL module.

System Action Taken: INFORMIX-4GL prompts you to correct the errors in the module.

Corrective Action: Select the **Correct** option and correct the errors indicated by the error messages.

-4608 **Description of Error**: The compilation of the program was not successful.

System Action Taken: INFORMIX-4GL prompts you to correct the errors in the module.

Corrective Action: Check that you have used the correct syntax for the statements in your program.

-4609 **Description of Error**: Insufficient memory is available to complete program compilation.

System Action Taken: The program was not compiled.

Corrective Action: Divide your program into smaller modules.

-4611 **Description of Error**: There is no 4GL source available for this program.

System Action Taken: INFORMIX-4GL prompts you to specify one or more program modules.

Corrective Action: Check the program definition file for the names of the modules that make up the program.

-4612 **Description of Error**: Data validation table does not currently exist for this database.

System Action Taken: INFORMIX-4GL prompts you to create the data validation table.

Corrective Action: If the **syscolval** table exists, make sure it resides in the current directory or in a directory specified by DBPATH. Otherwise, select the **Yes** option to create it.

-4613 **Description of Error**: Screen display attribute table does not currently exist for this database.

System Action Taken: INFORMIX-4GL prompts you to create the display attribute table.

Corrective Action: If the **syscolatt** table exists, make sure it resides in the current directory or in a directory specified by DBPATH. Otherwise, select the **Yes** option to create it.

-4614 **Description of Error**: A program already exists by this name.

System Action Taken: INFORMIX-4GL prompts you to enter another name for the program.

Corrective Action: Make sure that the name of each program is unique.

-4615 **Description of Error**: Invalid program name.

System Action Taken: INFORMIX-4GL prompts you to enter a valid program name.

Corrective Action: A program name cannot exceed ten characters on UNIX systems or eight characters on DOS systems. Make sure that the program name begins with a letter. The rest of the name may contain any combination of letters, numbers, and underscores (_).

-4616 **Description of Error**: The 4GL program database does not exist. Please create via PROGRAM section.

System Action Taken: INFORMIX-4GL prompts you to create the **syspgm4gl** database.

Corrective Action: If the **syspgm4gl** program database does not exist, create it using the **Program** option of the Programmer's Environment. If **syspgm4gl** does exist, make sure it resides in the current directory or in a directory specified by the DBPATH environment variable.

1203 **Description of Error**: Cannot find message file.

System Action Taken: The statement was not processed.

Corrective Action: RDSQL cannot locate a needed message file. Check that both the INFORMIXDIR and DBLANG environment variables are set with the appropriate pathname. Contact your System Administrator if you need help with this action.

1204 **Description of Error**: Type of terminal is unknown to the system.

System Action Taken: The statement was not processed.

Corrective Action: Check the setting of the TERM environment variable. Contact your System Administrator if you need help with this action.

2002 **Description of Error**: You have entered the wrong number of parameters. The syntax is as follows: **form4gl -d** *form-name database-name table1 table2 ...* (maximum of 14 tables).

System Action Taken: The form was not generated.

Corrective Action: Check the sequence of parameters present on the command line.

2005 **Description of Error**: Database *database-name* not found or not correct format.

System Action Taken: The requested action was not completed (operating system error).

Corrective Action: Check that the DBPATH environment variable includes the full pathname of the directory holding the database. Contact your System Administrator if you need help with this action.

2008 **Description of Error**: The table *table-name* does not exist in the database.

System Action Taken: The requested action was not completed.

Corrective Action: The table name included in the TABLE section of the form specification file is not found in the database specified in the DATABASE section. Check the spelling of the table name.

2009 **Description of Error**: You have not selected any database tables.

System Action Taken: The requested action was not completed.

Corrective Action: You must include one or more table names in the TABLES section of the form specification file.

2010 **Description of Error**: There is not enough memory to use the default form option for this particular choice of tables.

System Action Taken: The requested action was not completed.

Corrective Action: You have exceeded the data space limits of your machine. Reduce the number of tables included in the form.

2011 **Description of Error**: Default FORM4GL has run out of memory.

System Action Taken: The compile was not completed.

Corrective Action: You have exceeded the data space limits of your machine. Reduce the number of tables included in the form.

2012 **Description of Error**: Form output table *form-name* could not be opened.

System Action Taken: The compile was not completed.

Corrective Action: You may have exceeded the open file limit of 14 data files (this number includes the output file). Reduce the number of tables included in the form.

2017 **Description of Error**: The default form has exceeded its label limit.

System Action Taken: The compile was not completed.

Corrective Action: A form can use up to 26 one-character fields, 260 two-character fields, and 1000 fields of three characters or more. Reduce the number of tables in the default form.

2018 **Description of Error**: The default form has exceeded its limit of 260 two-character labels.

System Action Taken: The compile was not completed.

Corrective Action: The total number of two-character fields contained in the tables used to generate the default form exceeds the 260 limit. You must delete one or more of the tables containing two-character fields.

2019 **Description of Error**: The default form has exceeded its limit of 26 one-character labels.

System Action Taken: The compile was not completed.

Corrective Action: The total number of one-character fields contained in the tables used to generate the default form exceeds the 26 limit. You must delete one or more of the tables containing one-character fields.

2020 **Description of Error**: The following tables are involved: *table-name*.

System Action Taken: Check accompanying message for indication of system action.

Corrective Action: The indicated tables are involved in the specified error(s).

INFORMIX-TURBO and MS-NET Error Messages

This section contains the text of error messages that relate primarily to **INFORMIX-TURBO** and the MS-NET versions of INFORMIX products. A description of how the system responds to each error and the suggested corrective action is included. These messages are appropriate if you have **INFORMIX-TURBO** or an MS-NET version of an INFORMIX product on your system.

-127 **Description of Error**: No primary key.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Run **bcheck** to determine the source of the problem. Repair your table if necessary.

-128 **Description of Error**: No logging.

System Action Taken: The statement containing the error was not processed.

Corrective Action: You have attempted to execute a statement that requires logging. Start your database with transactions and reexecute the statement.

-129 **Description of Error**: Too many users.

System Action Taken: The statement containing the error was not processed.

Corrective Action: The shared memory parameter for the maximum number of users was exceeded. Either ask your System Administrator to adjust the parameter, or try the statement again later.

-130 **Description of Error**: Dbspace not found.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Check the name of your dbspace. Run the Turbo Monitor to view existing dbspaces.

-131 **Description of Error**: No free disk space.

System Action Taken: The statement containing the error was not processed.

Corrective Action: There is not enough contiguous disk space to complete the operation. Consult your System Administrator for assistance.

-132 **Description of Error**: The row size is too big.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Make the fields of the table smaller or create multiple tables with fewer fields.

-133 **Description of Error**: File has audit trail.

System Action Taken: The statement containing the error was not processed.

Corrective Action: You cannot cluster a file with an audit trail. If you want to cluster the file, you must first drop the audit trail.

-134 **Description of Error**: No more locks.

System Action Taken: The statement containing the error was not processed.

Corrective Action: The shared memory parameter for the maximum number of locks was exceeded. Either lock the entire table, wait until more locks are available, or ask your System Administrator to adjust the parameter.

-513 **Description of Error**: Statement not available in **INFORMIX-TURBO**.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Refer to the *Informix-Turbo Programmer's Manual.* Make sure that you are using the appropriate database server (**sqlexec** for non-**INFORMIX-TURBO** or **sqlturbo** for **INFORMIX-TURBO**).

-514 **Description of Error**: Only DBA can create a table, view or index for another user.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Ask the DBA to perform the action or to grant you the DBA privilege.

-515 **Description of Error**: Statement not available using non-**INFORMIX-TURBO** database server.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Refer to the reference manual for your front-end product. Make sure that you are using the appropriate database server (**sqlexec** for non-**INFORMIX-TURBO** or **sqlturbo** for **INFORMIX-TURBO**).

-517 **Description of Error**: The size of the index fields is too large or there are too many parts in the index.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Reduce the number of fields or the size of the fields in your index.

-518 **Description of Error**: Cannot have extent size equal to zero.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Check your CREATE or ALTER TABLE statement for valid extent sizes.

-519 **Description of Error**: Cannot update column to illegal value.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Check the values in your UPDATE or INSERT statement.

-520 **Description of Error**: Cannot open database partition.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Either the database does not exist or the shared memory parameter for the maximum number of open partitions was exceeded. Use the Turbo Monitor to determine the source of the problem. Ask the System Administrator to adjust the parameter.

-521 **Description of Error**: Cannot lock system catalog *catalog-name.*

System Action Taken: The statement containing the error was not processed.

Corrective Action: You are not allowed to lock system catalogs. Check the spelling of the table to be locked.

-522 **Description of Error**: Table *table-name* not selected in query.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Check the spelling of the table in the query.

-523 **Description of Error**: You can only recover, repair, or drop a table.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Check to make sure that the table in the query is not a view.

-524 **Description of Error**: Lock table can only be used within a transaction.

System Action Taken: The statement containing the error was not processed.

Corrective Action: You must start logging for the database before executing the statement.

-525 **Description of Error**: Cannot create database - no system permission.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Make sure you have write and execute permission on the current directory.

-526 **Description of Error**: Updates are not allowed on a scroll cursor.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Do not use a scroll cursor with a statement that is declared SELECT FOR UPDATE.

-527 **Description of Error**: Lock mode is not available on this system.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Modify your program to not WAIT FOR LOCK.

-844 **Description of Error**: Statement is too long (maximum 2,048 characters).

System Action Taken: The statement containing the error was not processed.

Corrective Action: Reduce the length of the statement to less than 2,048 characters.

-900 **Description of Error**: Cannot read network user authorization file.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Check that the network user authorization file exists on the server, that it is accessible through the network, and that you have read permission.

-901 **Description of Error**: User not found in network user authorization file.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Add the user to the network authorization file.

-902 **Description of Error**: User not authorized or too many entries in authorization file.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Requires a version configured for more users. Contact Informix Software for assistance.

-904 **Description of Error**: Authorization file not on licensed INFORMIX-SQL server.

System Action Taken: The statement containing the error was not processed.

Corrective Action: Reset the INFORMIXDIR variable to the directory that contains the licensed software.

Index

This index covers both the User Guide and the Reference Manual. Page numbers that end in U can be found in the User Guide, while those that end in R are in the Reference Manual.