PROGRAMAÇÃO EM INFORMIX 4GL

Título: Programação Informix 4GL
Autor: Sérgio Ferreira

INDICE

1. A LINGUAGEM 4GL

O Informix 4GL surge como um dos produtos da família Informix destinados a explorar bases de dados relacionais Informix.

A família Informix é constituída por um gestor de base de dados com duas versões diferentes do baixo nível (standard com C-ISAM e ON-LINE) e um conjunto de ferramentas destinadas a construir aplicações sobre essas bases de dados. Essas ferramentas são:

- Informix SQL, que inclui um gerador de écrans, gestor de menus e gerador de listagens. (Todos estes utilitários são bastante limitados.)
- SQL embebido em linguagens de 3ª geração: Informix ESQL/C, ESQL/COBOL, ESQL/FORTRAN, ESQL/ADA.
- Informix 4GL. Uma linguagem própria da Informix Corporation (ex-RDMS) COM características de desenvolvimento acelerado de programas (estimado em 10 a 20 vezes mais rápido que as linguagens de 3ª geração).

Todos os produtos Informix se caracterizam por aceder ás bases de dados através de uma linguagem comum chamada RDSQL (extensão do standard SQL-Structured Query Language). Esta filosofia permite uma maior facilidade na formação de programadores bem como uma separação efectiva entre o nível físico e lógico da exploração da base de dados tal como é definido na teoria das bases de dados relacionais.

Na linguagem 4GL juntam-se alguns dos mecanismos das linguagens de terceira geração, um gestor de écrans (FORM'S), um gestor de menus, um gestor de janelas, um gestor de input/output e um gerador de relatórios (REPORT), todos eles interagindo com a base de dados através da linguagem RDSQL. É também possível aceder a subrotinas escritas em C (ou outra linguagem à qual o C tenha acesso) e a recursos do sistema operativo.

A linguagem 4GL apresenta-se em duas versões totalmente compatíveis, uma compilada para programas executáveis em código específico das diferentes máquinas, outra compilada para um código intermédio independente do S.O.e do computador.

Na versão compilada do 4GL é constituído por um compilador, um conjunto de livrarias e algumas facilidades interactivas para os programadores. O compilador transforma um source de 4GL num source de ESQL/C, que por sua vez é transformado em C compatível com o compilador de C disponível na máquina/sistema operativo em que está instalada a base de dados.

Na versão com geração de p_código (Rapid Development System) o programa de 4GL dá origem directamente a um programa "executável" (sufixo 4go ou 4gi) que terá de ser interpretado por um programa RUN-TIME.

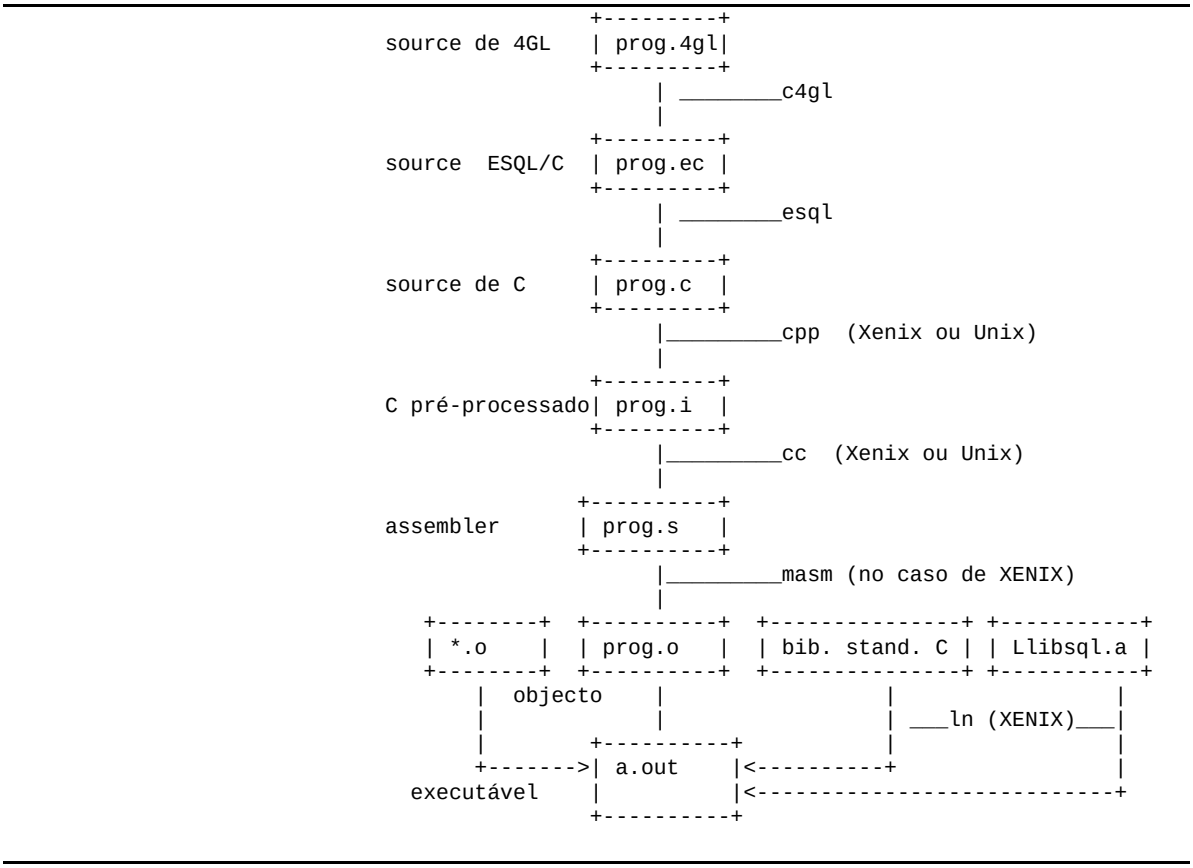


Figura 1 Fases da compilação de um programa de 4GL com a versão C

A diferença dos dois métodos de compilação está na rapidez com que são executadas diferentes fases do desenvolvimento. Os programas compilados com o Rapid-Development-System têm uma fase de compilação muito mais rápida (ex: 30s para 15m) do que a versão C. Em contrapartida os programas RDS sofrem alguma degradação no tempo de execução (15 a 20%) relativamente aos da outra versão.

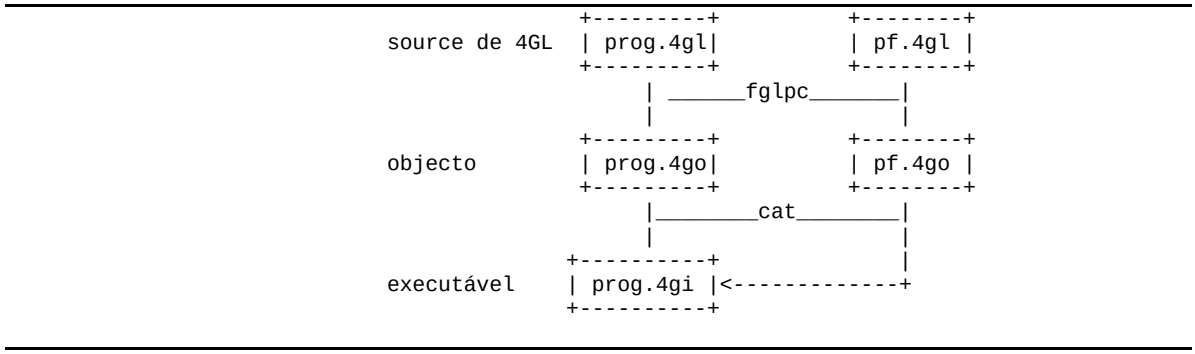


Figura 2 Fases de compilação de um programa 4GL com a versão RDS

Outra diferença que é da maior importância para os programadores, é a possibilidade de os programas desenvolvidos com a versão RDS serem integrados com um programa de detecção de erros chamado **Interactive Debuger.**

2. RDSQL - BREVE REVISÃO DAS NOÇÕES

No inicio da década de 70, F.F. Codd desenvolveu, na IBM, uma linguagem estruturada de buscas, SQL-Structured Query Language. Igualmente com o nome SQL foi desenvolvida pelo ANSI-American National Standards Institute, uma linguagem de manipulação de dados de uma base de dados, isto é, "Standard SQL".

Na figura seguinte podemos observar as instruções do Standard-SQL mais os comandos adicionados pela Informix Software, Inc.

ALTER INDEX*	DROP SYNONYM*	START DATABASE*
ALTER TABLE*	DROP TABLE*	UNION
BEGIN WORK*	DROP VIEW*	UNLOCK TABLE*
CLOSE DATABASE*	EXECUTE*	UPDATE
COMMIT WORK	GRANT	UPDATE STATISTICS*
CREATE DATABASE*	INSERT	WHENEVER
CREATE INDEX*	LOCK TABLE*	
CREATE SYNONYM*	PREPARE*	Cursor Manipulation:
CREATE TABLE	RENAME COLUMN*	CLOSE
CREATE VIEW	RENAME TABLE*	DECLARE
DATABASE*	REVOKE*	FETCH
DELETE	ROLLBACK WORK	FLUSH*
DROP DATABASE*	ROLLFORWARD DATABASE*	OPEN
DROP INDEX*	SELECT	PUT*

Figura 3 Instruções do Standard-SQL e as adicionadas pela Informix Software, Inc

Este capítulo pretende ser uma pequena revisão dos conceitos de RDSQL adquiridos no curso de Informix-SQL, complementando com alguns comandos novos para que nos possamos iniciar no estudo do Informix-4GL. Podemos observar que a maioria dos comandos da figura anterior estão já documentados no manual de Informix-SQL. Vamos passar a uma breve descrição destes:

ALTER INDEX índice **TO [NOT] CLUSTER**

Quando uma coluna tem um índice CLUSTER, isto é tem a ordenação física da tabela coincidente com a ordenação dos índices, (temporariamente, isto é, enquanto os dados da tabela não forem alterados).Esta instrução serve para alterar o índice que passa a ser CLUSTER ou deixa de ser CLUSTER.

*Instrucções adicionadas pela Informix Software, Inc

ALTER TABLE tabela

```
{ ADD (nova_coluna tipo_nova_coluna
[ BEFORE velha_coluna ][,...])
| DROP (velha_coluna [,...])
| MODIFY (velha_coluna novo_tipo_coluna [NOT NULL][,...])
}[,...]
```

Esta instrução serve para fazer alterações estruturais, na tabela *tabela.*

BEGIN WORK

Serve para iniciar uma transacção.

CLOSE DATABASE

Serve para fechar a base de dados, isto é fechar todos os ficheiros da base de dados que eventualmente estejam abertos.

COMMIT WORK

Termina uma transacção com sucesso, ou seja, todas as instruções de inserção ou alteração de dados a tabelas entre as instruções **BEGIN WORK** e **COMMIT WORK** serão realizadas.

CREATE DATABASE base dados [**WITH LOG IN** "pathname"]

Criar a base de dados com o nome *basedados.* Quando se cria a base de dados pode-se simultaneamente criar o ficheiro de transacções, onde ficam registados todas a alterações feitos sobre a base de dados.

CREATE [**UNIQUE** |**DISTINCT**] [**CLUSTER**] **INDEX** índice **ON** tabela (coluna [**ASC**/**DESC**][,...])

Quando se cria um índice tem de se dizer para que tabela e a que coluna este se refere. Pode-se ainda definir se é único, ou **CLUSTER**.

CREATE SYNONYM sinonimo **FOR** tabela

Criar um sinónimo para uma tabela, isto é, um nome alternativo. Este sinónimo apenas é válido para a pessoa que o criou.

CREATE TABLE tabela (coluna tipo [**NOT NULL**][,...])

[**IN** "pathname"]

Permite-nos criar uma tabela com as respectivas colunas. Caso não se queira que a tabela fique no directório basedados.dbs, pode-se definir para que directório vai a tabela que está a ser criada com a opção "IN pathname".

CREATE VIEW view [(lista_de_colunas)]

AS SELECT procedimento [**WITH CHECK OPTION**]

Permite-nos criar uma VIEW que tem como conteúdo o resultado do SELECT. Quando se cria a VIEW com "WITH CHECK OPTION" garante-se que o utilizador da VIEW apenas poderá ter acesso aos dados que resultam da execução da instrução SELECT.

DATABASE base dados [**EXCLUSIVE**]

A instrução DATABASE serve para activar a base de dados em que se quer trabalhar. Quando se activa a base de dados com a opção "EXCLUSIVE" temos a garantia de que mais ninguém lhe pode mexer.

DELETE FROM tabela [**WHERE** condição]

Para apagar linhas de dados de uma tabela; caso haja condição os dados apagados são os que obedecem a esta.

DROP DATABASE base dados

Apaga todas as tabelas que constituem a base de dados e remove o directório da base de dados a não ser que neste directório existam outros ficheiros que não pertençam a esta base de dados.

DROP INDEX índice

DROP SYNONYM sinonimo

DROP TABLE tabela

DROP VIEW view

Estas instruções apagam respectivamente índice, sinónimo, tabela e view.

GRANT {lista_previlégios_tabela **ON** tabela | privilégios base_dados} **TO** {**PUBLIC** | lista_utilizadores} [**WITH GRANT OPTION**]

Serve para dar permissões de acesso dos utilizadores às tabelas de base de dados. Quando se põe a opção "WITH GRANT OPTION" isto quer dizer que os utilizadores para quem se está a fazer GRANT podem fazer o mesmo para outros utilizadores.

INSERT INTO Tabela [(lista_colunas)]

{**VALUES** (lista_de_valores) | **SELECT** procedimentos}

Permite inserir dados numa tabela; estes dados podem-se inserir directamente ou serem o resultado de um SELECT.

LOCK TABLE tabela **IN** {**SHARE** |**EXCLUSIVE**} **MODE**

Esta instrução serve para vedar o acesso dos utilizadores enquanto se procede a uma determinada tarefa. O acesso pode ser totalmente vedado se se fizer o **LOCK** com a opção "**EXCLUSIVE**"; ou parcialmente vedado no caso de se usar a opção "**SHARE**" em que outros utilizadores podem fazer consultas à tabela em questão.

RENAME COLUMN tabela.velha_coluna **TO** nova_coluna

Permite mudar o nome de uma coluna de uma tabela.

RENAME TABLE nome_velho_tabela **TO** nome_novo_tabela

Permite alterar o nome de uma tabela.

REVOKE { lista_previlégios_tabela **ON** tabela | previlégios_base_dados } **FROM** {**PUBLIC** | lista_utilizadores }

Retira autorizações a utilizadores.

ROLLBACK WORK

Termina uma transacção com insucesso; todas as instruções de **INSERT**, **UPDATE** à tabela, entre as instruções **BEGIN WORK e ROLLBACK WORK** não se concretizam.

ROLLFORWARD DATABASE base_dados

Recuperação de dados de uma base de dados (quando esta tem problemas) a partir do ficheiro de transacções.

SELECT [**ALL** | **DISTINCT** | **UNIQUE**} lista_seleccionada

[**INTO** lista de variáveis]

FROM {tabela [alias_tabela]}

| **OUTER** tabela [alias_tabela]

[**WHERE** condição]

[**GROUP BY** lista colunas]

[**HAVING** condição]

[**ORDER BY** coluna [**ASC** | **DESC**][,...]]

[**INTO TEMP** tabela_tempo]

Esta instrução permite seleccionar linhas de uma ou mais tabelas; agrupadas ou não; ordenadas ou não; obedecendo ou não a um conjunto de condições.

START DATABASE basedados **WITH LOG IN** "pathname"

Inicialização de uma base de dados criando o ficheiro de transacção. Este procedimento tem de ser precedido da instrução "**CLOSE DATABASE**" no caso de se ter criado inicialmente a base de dados sem o ficheiro de transacção.

UNION

Serve para combinar duas ou mais instruções **SELECT** por forma a constituir uma única instrução **SELECT**.

SELECT procedimento **UNION** [**ALL**] **SELECT** procedimento [...]

UNLOCK TABLE tabela

Serve para destrancar o acesso a uma tabela que tinha sido vedado pela instrução "**LOCK TABLE**".

UPDATE tabela

SET { coluna = expressão [,...] |

{(lista_coluna) | tabela.***** | ***** } =

{(lista_expressão) | record.*****}}

[**WHERE** condição]

Permite alterar os dados de uma tabela, obedecendo ou não a uma determinada condição.

UPDATE STATISTICS [**FOR TABLE** tabela]

Quando se faz a inserção de dados numa tabela com a instrução "**LOAD**", é necessário correr esta instrução para se obter informação sobre o estado da tabela actualizado, isto é, proceder à actualização da tabela do dicionário da base de dados.

As instruções que aparecem na figura anterior mas que não foram aqui referidas irão ser explicadas no capítulo " Interação do Informix 4GL com as bases de dados".

3. ESTRUTURA DE UM PROGRAMA

Um programa é constituido por funções e declarações. A primeira função a ser executada é a função "MAIN", e, tem de existir em todos os programas.

As várias funções podem estar distribuidas por vários ficheiros a que se convencionou chamar módulos.

```
% cat hello.4gl

MAIN
 DISPLAY "hello world"
END MAIN

% c4gl -o hello.4ge hello.4gl
% hello.4ge
hello world
%
```

Figura 3 O mais pequeno programa de 4GL

3.1. FLUXO DO PROGRAMA

Tal como nas linguagens de terceira geração um programa é executado de acordo com instruções de controlo de fluxo, em todas as funções excepto nos reports (os reports possuiem uma estrutura declarativa). Para controlar o fluxo do programa foram implementadas diversas instruções de controlo de fluxo das linguagens tradicionais (while, if, case, etc.), e, algumas com instruções especificas desta linguagem (Menus, prompts, etc.), algumas das quais não procedimentais (ex: MENU, INPUT
etc.).

```
% cat here.4gl
MAIN
 DISPLAY "hello world"
 DISPLAY "here i am"
END MAIN

% c4gl -o here.4ge here.4gl
% here.4ge
hello world
here i am
%
```

Figura 4 Fluxo do programa

A frase "here i am" apareceu depois da frase "hello world" porque a 2ª instrução **DISPLAY** foi executada depois da primeira, e exprime o facto de a linguagem ser de estrutura procedimental (ou sequencial).

O 4GL é ao mesmo tempo declarativo e procedimental. Como exemplo de declarativo temos os reports e as instruções de entrada/saida formatada, e, como exemplo de procedimental a estrutura geral das funções.

```
% cat hellorep.4gl
MAIN
 START REPORT hello
        OUTPUT TO REPORT hello()
 FINISH REPORT hello
END MAIN

REPORT hello()
 FORMAT
        PAGE TRAILER
                PRINT "here i am"
        PAGE HEADER
                PRINT "hello world"
END REPORT

% c4gl hellorep.4gl
```

Figura 5 Fluxo num report

Quando este programa for compilado e executado a frase "here i am" aparecerá primeiro do que "hello world", pelo simples facto de que foi declarado ao programa que no inicio de página deverá aparecer "here i am" e no fim da página deverá aparecer "hello world". Este facto exprime a estrutura declarativa (não procedimental) dos REPORT(s) do 4GL.

3.2. TIPOS E CONVERSÕES DE TIPOS

Como linguagem de gestão de base de dados, o 4GL reconhece as estruturas representáveis na base de dados, e, algumas de maior complexidade. Os tipos de dados complexos, são declarados por composição de tipos elementares. Em 4GL os tipos de dados elementares existentes são:

SMALLINT Numeros inteiros pequenos, uma variável deste tipo pode receber numeros inteiros entre -32767 e 37767.

INTEGER Numeros inteiros entre aproximadamente -2 milhares de milhões e 2 milhares de milhões.

DECIMAL[(m,[,n])] Numeros decimais com m algarismos na parte inteira e n casas decimais (máximo 16 algarismos).

SMALLFLOAT Numero com virgula flutuante, representado em duas partes: uma mantissa entre 0 e 1, e um expoente.

FLOAT Numero com virgula flutuante de dupla precisão.Ocupa o dobro da memória de um SMALLFLOAT e permiite representar numeros maiores e com maior precisão.

MONEY[(m[,n])] Com tratamento especial na edição com vista a representar quantidades numerárias.

CHAR(n) Alfanumérico de comprimento n.

DATE Tipo equivalente a uma data, é configurável através da variável de ambiente **DBDATE**. Internamente, este tipo é armazenado como inteiro, no formato juliano (nº de dias desde 31 de dezembro de 1899).

Existem ainda dois tipos, compostos dos acima descritos

ARRAY tipo Representa uma variável composta por vários elementos do mesmo tipo.

RECORD Representa uma variável composta por diferentes variáveis de tipos diferentes. 'E possivel representar ARRAY(s) de RECORD(s), mas não se pode ter um ARRAY como elemento de um RECORD.

3.3. VARIÁVEIS

Uma linguagem, deve possuir sempre alguma forma de armazenar em memória informação intermédia necessária á execução dos programas. Tal como nas linguagens de terceira geração, no 4GL existem variáveis.

Uma variável pode ser de qualquer dos tipos definidos anteriormente.

A declaração de variáveis é sempre efectuada depois da palavra chave "DEFINE". Uma variável declara-se da seguinte forma:

```
DEFINE nome_da_variavel TIPO
ou
DEFINE nome_da_variavel  LIKE tabela.coluna
```

No segundo caso cria-se uma nova variável cuja estrutura condiz com a estrutura de uma coluna de uma tabela da base de dados. Sempre que uma variável for declarada usando a instrução LIKE, deve existir no inicio do módulo uma instrução DATABASE que indica qual a base de dados a ser usada para obter as colunas com vista á declaração por equivalência.

Se a variável for um record deve-se apenas declarar da seguinte forma:

```
nome_do_record RECORD lista_de_variaveis END RECORD
ou
nome_do_record RECORD LIKE tabela.* END RECORD
```

No segundo caso é criada uma estrutura com elementos identicos a todas as colunas de uma tabela da base de dados corrente.

Um elemento de uma variável tipo RECORD acede-se através do operador ".", assim pr_x.c1 representa o elemento c1 do record pr_x. A notação ".*" pode ser usada em alguns contextos para indicar todos os elementos de um RECORD, ex: pr_x.* indica todos os elementos do RECORD pr_x.

Se a variável for um array deve ser declarada da seguinte forma:

```
nome_da_variavel ARRAY[m,n,...] OF tipo
```

Figura 6 Definição genérica de um declaração de array

Os elementos das variáveis tipo ARRAY acedem-se através do operador [] e dá indicação do numero do elemento a que se quer aceder, ex: pa_x[10] indica o 10º elemento do ARRAY pa_x.

As variáveis podem ser locais a uma função ou globais a um módulo ou programa

3.3.1. VARIÁVEIS LOCAIS

```
DEFINE lista_de_variveis tipo [, ...]
```

Figura 7 Definição genérica de uma variável local

As variáveis locais têm que ser declaradas no inicio de cada função, e, só são conhecidas dentro da função em que foram declaradas.

```
% cat localvars.4gl

DATABASE livros

MAIN
DEFINE
        x INTEGER,
        pr_livros RECORD LIKE livros.*

LET x = 1
SELECT livros.*
        INTO pr_livros.*
        FROM livros
        WHERE numero = x

DISPLAY pr_livros.nome

END MAIN

% c4gl localvars.4gl
% a.out
Artificial Inteligence using C

%
```

Figura 8 Programa com variáveis locais

Se declarar duas variáveis com o mesmo nome em duas funções distintas, cada variável só vai ser conhecida na função de declaração. Efectivamente as variáveis com o mesmo nome, locais a funções diferentes são variáveis diferentes.

3.3.2. VARIÁVEIS GLOBAIS

```
GLOBALS
        {"nome_de_ficheiro"|
        instrucao DEFINE
END GLOBALS}
```

Figura 9 Definição genérica de uma declaração global

Como se pode ver acima, a instrução GLOBALS tem duas formas possiveis.Na primeira indica-se o nome de um ficheiro que contém um conjunto de instruções DEFINE e uma instrução DATABASE.A vantagem deste sistema é o conhecimento das variáveis globais em todos os módulos sem necessidade de declará-las em cada um deles.

Na outra forma, a instrução GLOBALS é constituida por uma instrução define.

```
% cat globalvars.4gl

DATABASE livros
GLOBALS
DEFINE
        x INTEGER,
        pr_livros RECORD LIKE livros.*
END GLOBALS

MAIN

LET x = 1
SELECT livros.*
        INTO pr_livros.*
        FROM livros
        WHERE numero = x

DISPLAY pr_livros.nome

END MAIN

% c4gl globalvars.4gl
% a.out
Artificial Inteligence using C
%
```

Figura 9 Programa com variáveis globais

No programa dado como exemplo não havia grande necessidade de declarar as variáveis como globais. Á medida que fôr programando vai sentindo as necessidades deste tipo de variáveis.

3.3.3. AFECTAÇÕES, COMPARAÇÕES E INICIALIZAÇÕES.

Uma das operações básicas a efectuar sobre variáveis é a afectação.

No 4gl para afectar uma variável com uma constante, expressão ou conteudo de outra variável utiliza-se o seguinte formato:

```
LET variável = { expressão | variável | constante }
```

Figura 10 Definição genérica da afectação

No 4gl, ao contrário de outras linguagens (C por exemplo), pode-se afectar uma variável de qualquer tipo básico. As variáveis compostas só podem ser afectadas através dos seus elementos constituintes.

```
% cat afecta.4gl

DATABASE livros

MAIN
DEFINE
        x INTEGER,
        y CHAR(20),
        z FLOAT,
        pr_livros RECORD LIKE livros.*,
        pr_livros2 RECORD LIKE livros.*

LET x = 69
LET y = "Uma string"
LET z = 120.596

DISPLAY "X tem ", x,", Y tem ", y,"e Z = ",z

SELECT *
        INTO pr_livros.*
        FROM livros
        WHERE @numero = 1

LET pr_livros2.* = pr_livros.*
DISPLAY pr_livros2.numero, pr_livros.nome

END MAIN
% c4gl afecta.4gl
% a.out
X tem       69, Y tem Uma string       e Z =     120.60
        1Artificial Inteligence Using C
%
```

Figura 11 Afectações de variáveis

Conforme se viu afectaram-se variáveis de diversos tipos elementares e compostos. Nos tipos compostos o número e tamanho das variáveis elementares que dele fazem parte t^em de ser iguais.

Nesta linguagem, não há diferença entre o operador "=" usado na afectação e na comparação. Este facto deve-se a que a identificação destas operações é efectuada por intermédio do contexto em que as instruções estão inseridas.

Quando se afecta uma variável com uma expressão, esta é automáticamente convertida para o tipo da variável, se a conversão resulta em erro, o programa notifica com **erro de execução.** Por exemplo na instrução LET i="10" se a variável i for um inteiro, a expressão "10" é automáticamente convertida a inteiro.

As comparações são todas feitas por intermédio do sinal de igual mesmo que sejam comparações de strings.

```
% cat comp.4gl
MAIN
DEFINE
        y CHAR(20)

LET y = "Uma string"

IF y > "UMA STRING" THEN
        DISPLAY "Uma string maior que UMA STRING"
END IF
IF y < "uma string" THEN
        DISPLAY "Uma string menor que UMA STRING"
END IF
IF y = "Uma string" THEN
        DISPLAY "Uma string igual a UMA STRING"
END IF
END MAIN
% c4gl comp.4gl
% a.out
Uma string maior que UMA STRING
Uma string menor que UMA STRING
Uma string igual a UMA STRING
%
```

Figura 12 Vários exemplos de comparações

Os operadores de comparação, no 4GL são:

>	→ maior que
<	→ menor que
>=	→ maior ou igual que
<=	→ menor ou igual que
=	→ igual a
!=	→ diferente de (not igual)
MATCHES	→ comparação usando metacarateres (*, ?, [])
NOT MATCHES	→ negação de MATCHES
LIKE	→ comparação usando metacaracteres (-, %)
NOT LIKE	→ negação de LIKE
BETWEEN	→ intervalo de valores
NOT BETWEEN	→ negação de BETWEEN

Figura 13 Tabela de operadores de comparação

Em qualquer programa é, muitas vezes necessário proceder a inicializações de variáveis.

Uma das formas de o fazer é afectando a variável com o valor pretendido, na zona do programa em que tal se pretende fazer. Outra forma é utilizando a instrução INITIALIZE.

```
INITIALIZE lista_de_variaveis
  {LIKE lista_de_colunas | TO NULL}
```

Figura 14 Definição genérica da instrução INITIALIZE

Se utilizar a cláusula LIKE lista_de_colunas, a instrução inicializa as variáveis com os valores correspondentes a cada coluna e que estão registadas na tabela syscolval (ver utilitário upscol), ou seja as variáveis ficam com os valores de default definidos com o utilitário upscol.

Se evocar a função com o par^ametro TO NULL, as variáveis dadas ficam inicializadas a NULL.

3.3.4.CONCATENAÇÃO DE VARIÁVEIS CHAR E O OPERADOR,

No 4GL é muito simples concatenar strings, basta separá-las por uma virgula (o operador de concatenação de strings é a virgula).

```
% cat concat.4gl
MAIN
DEFINE
        x CHAR(20),
        y CHAR(50)

LET x = "String 1       "
LET y = "Concatenou ",x CLIPPED," com outras duas strings"
DISPLAY y
END MAIN
% c4gl concat.4gl
% a.out
Concatenou String 1 com outras duas strings
%
```

Figura 13 Exemplo de concatenação de strings

3.4. FUNÇÕES

Uma função é um conjunto de instruções que são executadas conjuntamente e que pode receber um conjunto de par^ametros e devolver valores.

O 4GL apesar de ser uma linguagem de 4ª geração possui esta capacidade das linguagens estruturadas.

```
FUNCTION nome_da_funcao([arg1,arg2,...,argn])
instrução
...
[RETURN lista_expressões]
...
END FUNCTION
```

Figura 15 Definição genérica de uma função

Uma função é identificada por um nome, começa com a palavra chave FUNCTION e termina com as palavras chave END FUNCTION, excepto a função MAIN que não tem nome e é delimitada por MAIN...END MAIN.

```
% cat hellofunc.4gl
MAIN
        CALL hello()
END MAIN

FUNCTION hello()
        DISPLAY "hello world"
END FUNCTION

% c4gl -o hellofunc.4ge hellofunc.4gl
% hellofunc.4ge
hello world
%
```

Figura 15 Uma função muito simples

Uma função, para além de poder possuir variáveis locais, pode ainda ter variáveis de entrada (parâmetros), e, ter associada valores a devolver á função que a evocou (valores de retorno). Tanto os par^ametros como os valores de retorno podem ser de qualquer tipo permitido pelo 4gl.A chamada á função faz-se com a instrução CALL e para definir em que variáveis vêm os valores devolvidos usa-se a cláusula RETURNING com uma lista de variáveis a devolver.

```
CALL função([argumentos]) [RETURNING lista_de_variáveis]
```

Figura 16 Definição genérica da instrução CALL

Se uma função devolver apenas um valor pode-se chamar com a instrução LET.

```
LET variavel = função([argumentos])
```

Figura 17 Chamar uma função com instrução LET

Os parâmetros de uma função são sempre declarados. Esta declaração faz com que a linguagem verifique todas as chamadas ás funções e no caso de estas não serem executadas com o mesmo número de parâmetros que a declaração da função resulta um erro de compilação do programa.

```
% cat max.4gl
MAIN
        DEFINE maxval INTEGER
        LET maxval = maximo(4,8,5)
        DISPLAY "O maior numero e: ",maxval
END MAIN

FUNCTION    maximo(n1,n2,n3)
DEFINE
        n1 INTEGER,
        n2 INTEGER,
        n3 INTEGER
IF n1 > n2 THEN
        LET n2 = n1
END IF
IF n2 > n3 THEN
        RETURN n2
END IF
RETURN n3
END FUNCTION
% c4gl max.4gl
% a.out
O maior numero e:  8
%
```

Figura 18 Função com vários tipos de par^ametros de entrada

Para retornar determinado valor usa-se a palavra chave RETURN.

Ao contrário de algumas linguagens de terceira geração (p/ex. o C), uma função de 4GL não necessita de ser declarada como de determinado tipo, ou seja: se pretender que a função devolva o número inteiro 10, só terá que fazer:

```
% cat retfunc1.4gl

MAIN
        DEFINE x INTEGER
        LET x = FUNC()
        DISPLAY "X tem",x
END MAIN

FUNCTION FUNC()
        RETURN(10)
END FUNCTION

% c4gl retfunc1.4gl
% a.out
X tem 10
%
```

Figura 19 Função que retorna um número inteiro

Se pretender devolver uma string poderá fazer:

```
% cat retfunc2.4gl

MAIN
        DEFINE x CHAR(100)
        LET x = FUNC()
        DISPLAY x
END MAIN

FUNCTION FUNC()
        RETURN("Esta string vai ser devolvida")
END FUNCTION

% c4gl retfunc2.4gl
% a.out
Esta string vai ser devolvida
%
```

Figura 20 Função que retorna uma string

Há que ter cuidado para que a variável com que se afecta a função seja do mesmo tipo do valor (ou variável) que se devolveu dentro da função.

```
% cat retfunc3.4gl

MAIN
DEFINE
        x INTEGER
        y CHAR(100)

        CALL FUNC() RETURNING(x,y)
        DISPLAY x
        DISPLAY y
END MAIN

FUNCTION FUNC()
        RETURN(100,"Esta string vai ser devolvida")
END FUNCTION

% c4gl retfunc3.4gl
% a.out
100
Esta string vai ser devolvida
%
```

Figura 21 Função com devolução de vários valores

3.5. MÓDULOS

Para um programa muito complexo torna-se dificil a coexistencia de todas as funções num unico ficheiro, pois a legibilidade fica grandemente afectada (é dificil ler um ficheiro muito grande).

Determinadas tarefas podem ter de ser executadas várias vezes, em programas diferentes, como por exemplo a validação de um digito de controle (check digit), do número de contribuinte, que é igual ao do B.I. Seria bom se fosse possivel reutilizar uma função em vários programas.

A um programa separado em módulos facilita muito a tarefa de manutenção, pois em caso de alteração é apenas necessário recompilar os módulos que sofreram alterações.

A linguagem 4GL permite a divisão de um programa em módulos, que são compilados separadamente e ligados (link) em seguida.

Em cada módulo existem declarações globais e funções. Num programa constituido por vários módulos, só deve existir a função MAIN num deles, e um nome de função só deve existir uma vez em todos os módulos.

```
% cat mod1.4gl
MAIN
        CALL hello()
END MAIN
% cat mod2.4gl
FUNCTION hello()
        DISPLAY "Hello world"
END FUNCTION
% c4gl mod1.4gl mod2.4gl
% a.out
Hello world
%
```

Figura 22 Programa dividido em dois módulos separados

No exemplo anterior compilaram-se os dois módulos simultâneamente, no entanto poder-se-ia compilar cada um deles com a opcao -c para gerar um objecto (.o), e no final ligar (linkar) os dois [*]. Procedendo desta forma você pode compilar apenas o módulo que alterou e linkar com o objecto já existente. Se estiver a trabalhar em RDS o processo de ligação de dois módulos é feito concatenando-os com o comando cat (no UNIX).

Quando se trabalha com muitos módulos é conveniente utilizar um programa gestor de compilações (make do unix ou o "program compile" do 4gl), pois a tarefa de construção do programa pode complicar-se de tal modo que se perde o controle sobre as alterações já feitas e os passos necessários á reconstrução do programa.

3.6. CONVERSÕES DE TIPOS

Como já foi dito quando se falou de afectações, as expressões no 4GL são sempre convertidas para o tipo necessário ao contexto onde estão incluidas, não existem por isso operadores de conversão de tipo como noutras linguagens, a única excepção é o operador DATE(...) que converte uma expressão para um tipo

*Usando o comando ld (link), no caso da versão C do 4GL

DATE. Esta situação torna por vezes problemática a conversão de tipos, podendo no entanto tornear-se o problema declarando uma variável do tipo para o qual se quer converter a expressão e afectando a variável com a expressão.

4. CONTROLO DE SEQUÊNCIA

A construção de um programa implica a tomada de decisões, a repetição de instruções parâmetrizadas, etc. No 4GL encontram-se alguns dos mecanismos de controlo e repetição existentes nas linguagens de terceira geração.

4.1. IF...THEN...ELSE...END IF

Esta instrução existe na quase generalidade das linguagens. A decisão é efectuada partir de uma expressão booleana.

Em português esta instrução poderia ser explicada por:

```
Se esta expressão é verdadeira
Então faz isto
Senão faz aquilo
```

Em que "faz isto" e "faz aquilo" seria um conjunto de instruções a executar em cada um dos casos.

```
IF expressão_booleana THEN
        instrução
[ ELSE
        instrução
END IF
```

Figura 23 Definição geral da instrução IF...THEN...ELSE

Uma expressão booleana pode por exemplo, ser um conjunto de comparações entre expressões, sendo possivel efectuar operações booleanas entre expressões, nomeadamente:

AND A expressão é verdadeira se e só se as duas sub-expressões o forem.

```
IF a = "casado" AND b = "solteiro" THEN
        CALL casa_com(a,b)
END IF
```

Figura 24 Utilização de AND numa expressão

OR A expressão é verdadeira se alguma das sub-expressões o fôr.

```
IF cao = "morde" OR cao = "grande" THEN
        CALL correr_mais_depressa()
END IF
```

Figura 25 Utilização de OR numa expressão

NOT A expressão é verdadeira se a sub-expressão não fôr.

```
IF   NOT ha_cerveja THEN
        CALL bebe_vinho()
END IF
```

Figura 26 Utilização de NOT numa expressão

4.2. WHILE

Esta instrução de repetição pode ser explicada por:

Enquanto esta_expressão_for_verdadeira faz_isto

Faz_isto é um conjunto de instruções a serem executadas se a expressão booleana for verdadeira.

```
WHILE expressao_booleana
        { instrução | EXIT WHILE | CONTINUE WHILE }
END WHILE
```

Figura 27 Definição genérica da instrução WHILE

Uma expressão booleana usada no while é identica á usada no IF e a qualquer uma que se utilize num programa.

```
WHILE esta != "bebado"
        CALL bebe_mais_cerveja()
END WHILE
```

Figura 28 Ciclo WHILE

Existem duas instruções que podem ser usadas dentro de um WHILE:

"EXIT WHILE" e "CONTINUE WHILE".

EXIT WHILE, serve para abandonar o ciclo independentemente da expressão booleana do WHILE.

CONTINUE WHILE, quando evocada, provoca um reposicionamento no inicio do ciclo (com o correspondente teste á guarda do ciclo) sem cumprir as instruções que se encontravam abaixo.

4.3. FOR

Por vezes é necessário cumprir um ciclo um determinado número de vezes incrementando uma variável com um valor.

```
FOR variavel_inteira = expressão1 TO expressao2
        [ STEP expressão3 ]
        { instrução | CONTINUE FOR | EXIT FOR }
END FOR
```

Figura 28 Definição genérica do ciclo FOR

A instrução acima definida inicializa a variavel_inteira com o resultado da expressão1 e cumpre as instruções até que a expressão2 seja falsa, incrementando a variável_inteira com o resultado de expressão3. Se o programador, por alguma razão desejar sair do ciclo, pode utilizar a instrução "EXIT FOR". Se, por outro lado já não pretender executar as instruções abaixo pode utilizar a instrução "CONTINUE FOR". Estas duas ultimas instruções são em tudo análogas ás existentes no ciclo WHILE, incidindo no entanto sobre o "FOR".

4.4. CASE

Por vezes é necessário efectuar uma decisão multipla, isso pode fazer-se com recurso a várias instruções IF encaixadas, a instrução CASE surge como uma forma mais simples de resolver esta questão.

```
CASE [(expressão)]
        WHEN {expressão | expressão booleana}
                instrução
        ...
                [EXIT CASE]
        ....
        [OTHERWISE]
                instrução
                ...
                [EXIT CASE]
        ...
END CASE
```

Figura 29 Definição genérica da instrução CASE

A instrução CASE pode ser utilizada de duas formas diversas:

4.4.1. SELEÇÃO EFECTUADA POR EXPRESSÕES

```
CASE (expr)
        WHEN valor
                instrução
                ...

        OTHERWISE
                instrução
                ...
END CASE
```

Figura 30 Forma geral de CASE com seleção por expressões

A expressão é comparada com os vários valores sendo executadas as instruções associadas ao primeiro valor igual ao resultado da expressão.Se todos os valores forem diferentes do resultado da expressão, são executadas as expressões associadas á cláusula OTHERWISE (se esta existir).

```
% cat case1.4gl
DATABASE livros

MAIN
DEFINE
        x   LIKE livros.numero,
        ans CHAR(1),
        nome LIKE livros.nome

LET ans = "s"
WHILE 1
        PROMPT "Qual o numero do livro que pretende procurar: "     FOR x

        SELECT livros.nome
                INTO nome
                FROM livros
                WHERE numero = x

        DISPLAY nome

        PROMPT "Quer escolher mais livros (S/N) "
        FOR CHAR ans
        CASE (ans)
                WHEN "s"
                        EXIT CASE
                WHEN "n"
                        EXIT WHILE
                OTHERWISE
                        DISPLAY "OPCAO INVALIDA"
                        EXIT WHILE
        END CASE
END WHILE
END MAIN
% c4gl case1.4gl
% a.out
Qual o numero do livro que pretende procurar: 34
Unix system programing
Quer escolher mais livros (S/N) g
OPCAO INVALIDA
%
```

Figura 31 Utilização do CASE utilizando expressões para fazer a seleção

Na execução deste programa, quando se perguntou ao utilizador se desejava procurar mais livros, ele respondeu <g> quando as opções eram <s> ou <n>, pelo que o programa entrou no OTHERWISE do case.

Neste caso a seleção do caso pretendido é efectuada quando o resultado da expressão existente junto da palavra chave CASE é identico ao resultado de uma expressão junto a uma das palavras chave WHEN. De notar que, se existirem duas expressões que tenham como resultado igual valor, é executada a que aparece em 1º lugar.

4.4.2. SELEÇÃO EFECTUADA POR EXPRESSÃO BOOLEANA

A instrução CASE com expressões booleanas toma a seguinte forma:

```
CASE
        WHEN expressão_booleana1
                instrução
                ...
        WHEN expressão_booleana2
                instrução
                .
        OTHERWISE
                instrução
                ...
END CASE
```

Figura 31 Instrução CASE com seleção por expressão booleana

As expressões booleanas são avaliadas na ordem em que aparecem e é executado o grupo de instruções associado á primeira expressão booleana verdadeira. Se todas as expressões forem falsas são executadas as instruções associadas á cláusula OTHERWISE.

Em qualquer das formas da instrução CASE pode-se incluir no bloco de instruções uma instrução EXIT CASE que provoca a interrupção do CASE neste ponto.

5. A BIBLIOTECA 4GL

Este capítulo descreve as funções que estão disponíveis na biblioteca do INFORMIX-4GL. Qualquer das funções apresentadas de seguida podem ser chamdas num programa, que o compilador do INFORMIX-4GL automáticamente as linka ao programa.

Passamos de seguida à descrição das funções da biblioteca:

ARG_VAL(expressão)

Esta função recebe como argumento uma variável tipo INTEGER, devolve o argumento correspondente a esse número que lhe foi passado pela linha de comando.

A função ARG_VAL(n) devolve o n-ésimo argumento da linha de comando,do tipo CHAR.

O valor da *expressão*tem de estar entre **0** e **NUM_ARGS()**

do número de argumentos passados pela linha de comando. **ARG_VAL(0)** corresponde ao nome do programa.

As funções ARG_VAL e NUM_ARGS permitem passar dados ao programa a partir da linha de comando quando este é executado.

Suponhamos que o nosso programa de 4GL compilado se chama **teste.4ge,** ao executá-lo da linha de comando fazemos

```
teste.4ge maria pedro catarina
```

Figura 32Execução de um programa de 4GL, com argumentos

O programa que se segue é um exemplo, recebe os nomes num ARRAY de variaveis do tipo CHAR

ARR_COUNT()

Esta função devolve o número de linhas introduzidas num array de um programa durante ou após a instrução INPUT ARRAY.

Quando se executa uma instrução com a cláusula BEFORE, AFTER ou ON KEY, o valor de ARR_CURR, SCR_LINE ou ARR_COUNT, é alterado, e o valor resulta da execução das cláusulas.

```
FUNCTION inserir_livros()
DEFINE contador SMALLINT

FOR contador = 1 TO arr_count()
        INSERT INTO livros
        VALUES (p_livros[contador].liv_num,
                        p_livros.[contador].liv_nome,
                        p_livros.[contador].liv_editor,
                        p_livros.[contador].livªno)
END FOR

END FUNCTION
```

Figura 33 Exemplo de utilização da função ARR_COUNT

ARR_CURR()

Esta função devolve a linha do array do programa que corresponde à linha do array corrente do écran durante o imediatamente após a instrução INPUT ARRAY ou DISPLAY ARRAY.

A linha corrente do array do écran corresponde á linha onde está localizado o cursor no inicio de uma cláusula BEFORE ROW or AFTER ROW.

A primeira linha tanto do array do programa ou do écran é numerada com 1.

```
INPUT ARRAY p_livros TO s_livros.*
        ON KEY(F6)
        DISPLAY ARRAY p_estante TO p_estante.*
        END DISPLAY
        LET pa_curr = ARR_CURR()
        MESSAGE "O valor corrente de ARR_CURR é: ", pa_curr
END INPUT
```

Figura 34 Exemplo da utilização da função ARR_CURR

DOWNSHFIT(frase)

Esta função devolve uma string em que todos os caracteres que lhe tenham sido passados em maiusculas são convertidos em minusculas.

Se o caracteres passados á função como argumentos forem numéricos, este não serão alterados.

A função DOWNSHIFT pode ser usada numa expressão ou para afectar uma variável com o valor devolvido pela função.

```
LET d_str = DOWNSHIFT(str)

DISPLAY d_str
```

Figura 35 Exemplo de utilização da função DOWNSHIFT

UPSHFIT(frase)

Esta função devolve uma string em que todos os caracteres que lhe tenham sido passados em minusculas são convertidos em maisculas.

Se o caracteres passados á função como argumentos forem numéricos, este não serão alterados.

A função UPSHIFT pode ser usada numa expressão ou para afectar uma variável com o valor devolvido pela função.

```
LET d_str = UPSHIFT(str)

DISPLAY d_str
```

Figura 36 Exemplo de utilização da função UPSHIFT

ERR_GET(expressão)

Esta função devolve uma variável do tipo CHAR que contém a mensagem de erro, correspondente ao código enviado como argumento.

Geralmente a **expressão**é a variável global STATUS.

A utilização desta função é muito importante durante as operações de debug.

```
IF STATUS < 0 THEN
        LET errtext = ERR_GET(STATUS)
END IF
```

ERR_PRINT(expressão)

Esta função escreve a mensagem de erro do INFORMIX-4GL, correspondente ao seu argumento, na ERROR LINE (linha de erro).

Mais uma vez, em geral a **expressão** é a variável global STATUS.

A utilização desta função é muito importante durante as operações de debug.

```
IF STATUS < 0 THEN
        CALL ERR_PRINT(STATUS)
END IF
```

Figura 37 Exemplo de utilização da função ERR_PRINT

ERR_QUIT(expressão)

Esta função escreve a mensagem de erro do INFORMIX-4GL, correspondente ao seu argumento, na ERROR LINE (linha de erro) e termina a execução do programa.

```
CALL ERR_QUIT(expressão)
```

Figura 38 Formato da função ERR_QUIT

Geralmente a **expressão** é a variável global STATUS.

A utilização desta função é muito importante durante as operações

de debug.

```
IF STATUS < 0 THEN
        CALL ERR_QUIT(STATUS)
END IF
```

Figura 39 Exemplo da utilização da função ERR_QUIT

6. I/O BÁSICO

6.1. PROMPTS

Em 4GL, uma das formas do programa comunicar com o utilizador, é por intermédio da instrução PROMPT. Esta instrução envia uma mensagem para o écran, e, recebe um caracter ou sequencia de caracteres que foram digitados pelo utilizador no seu terminal.

A definição genérica de uma instrução PROMPT é a seguinte:

```
PROMPT mensagem FOR [CHAR] variável
        [HELP numero_do_help]
        [ON KEY (tecla)
                instrução
        ...
        ...
END PROMPT]
```

Figura 40 Definição genérica da instrução PROMPT

As mensagens assinaladas são um conjunto de uma ou mais variáveis, ou string(s) constantes, separados por virgulas.

Se a instrução for evocada com a palavra chave CHAR, o programa não espera pelo <NEW LINE>, o que equivale a dizer que aceita o caracter e sai da instrução prompt assim que é premida uma tecla.

Mais tarde serão focadas outras capacidades desta instrução.

```
% cat prompt.4gl

DATABASE livros

MAIN
DEFINE
        x   LIKE livros.numero,
        ans CHAR(1),
        nome LIKE livros.nome

LET ans = "s"
WHILE ( ans = "s" OR ans = "S" )
        PROMPT "Qual o numero do livro que pretende procurar: "
        FOR x

        SELECT livros.nome
                INTO nome
                FROM livros
                WHERE numero = x

        DISPLAY nome

        PROMPT "Quer escolher mais livros (S/N) "
        FOR CHAR ans
END WHILE
END MAIN
% c4gl prompt.4gl
% a.out
Qual o numero do livro que pretende procurar: 15
Microsoft C compiler
Quer escolher mais livros (S/N) n
%
```

Figura 41 Utilização da instrução PROMPT

O programa proposto quando é executado pede ao utilizador que digite o número do livro do qual pretende saber o nome, faz uma busca na base de dados para descobrir o nome, e, pergunta se o utilizador pretende ou não continuar a executar essa tarefa.

No programa, quando é executada a primeira instrução PROMPT,como não existe a palavra CHAR, é permitido digitar vários digitos, e terminar com <NEW LINE>. Na segunda instrução PROMPT, tal já não se passa, pois assim que foi premida a tecla <n> o programa continuou a sua execução.

6.2. MENUS

Uma forma tipica de perguntar ao utilizador que tarefa pretende executar de um conjunto de n é a utilização de menus.

No 4GL é bastante fácil construir menus horizontais. Nestes menus cada opção é identificada por uma pequena palavra e uma linha de explicação. Uma opção encontra-se sempre destacada sendo selecionada

premindo a tecla <RETURN> pode mudar-se a opção selecionada com a barra de espaços, as setas ou a primeira letra da opção desejada. Podem também criar-se opções invisiveis. Se as opções não couberem todas no menu, este é prolongado com "..." significando que se pode aceder a outras opções que não são visiveis.

Para construir um menu usa-se a instrução MENU cuja definição genérica é:

```
MENU "nome_do_menu"
        COMMAND {KEY(lista_de_teclas) |
                        [KEY(lista_de_teclas) "opcao"
                        ["linha_de_help"] [HELP numero]
                instrucao
                ...
                [CONTINUE MENU]
                ...
                [EXIT MENU]
                ...
                [NEXT OPTION "opcao"]
                ...
END MENU
```

Figura 42 Definição genérica de uma instrução MENU

As cláusulas CONTINUE MENU, NEXT OPTION, e EXIT MENU podem ser colocadas dentro de outras instruções desde que estejam dentro de uma instrução MENU. Estas cláusulas permitem uma grande optimização do comportamento do menu para o utilizador pois pode haver posicionamento na opção que lógicamente deveria ser escolhida.

Se for utilizada a cláusula de HELP, quando pressiona a tecla de help, o programa chama a função show_help e envia o help com o número pretendido para o écran.

A cláusula KEY é optima para definir opções escondidas, e/ou assignar uma nova tecla de escolha rápida a uma opção.

6.3. DISPLAY AT

Se se pretender enviar pequenas mensagens ao utilizador utiliza-se, no 4gl a instrução DISPLAY AT.

```
DISPLAY lista_de_variaveis
        [AT linha_do_ecrançoluna_do_ecran]
        [ATTRIBUTE (lista_deªtributos)]
```

Figura 42 Definição da instrução DISPLAY AT

lista_de_variaveis

É um conjunto de uma ou mais variáveis do programa ou string(s) constantes separadas por virgulas.

linha_do_ecran e coluna_do_ecran

São as coordenadas no écran do local onde vai aparecer a mensagem pretendida

lista de atributos

É um conjunto de atributos possiveis a utilizar quando se enviar a mensagem para o écran.

Os atributos possiveis são:

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

Figura 44 Atributos do écran possiveis

A instrução DISPLAY é bastante mais complexa, no capítulo sobre gestão de écrans voltaremos a falar sobre ela.

Esta instrução envia para o écran uma mensagem constituida pelo conjunto de variáveis e strings definidas na instrução. Em alguns dos exemplos anteriores, esta instrução já foi usada na sua forma mais simples.

```
% cat display.4gl
MAIN
DISPLAY "Isto vai aparecer no écran na linha 10 coluna 3"
        AT 10,3
        ATTRIBUTE(BOLD)
END MAIN
% c4gl display.4gl
% a.out
APARECEU A FRASE DO DISPLAY NA linha 10 coluna 3
%
```

Figura 45 Programa utilizando DISPLAY AT com ATTRIBUTES"

7. GESTÃO DE ERROS

No 4GL existem vários tipos de erros, depois de compilar com sucesso o seu programa, pode ainda estar sujeito a erros.

É possivel que o programa ainda possua erros de lógica (bugs), ou que devido a uma má utilização do programa, surja uma situação de erro.

O utilizador de um programa, sem conhecimentos de programação nem acesso aos sources deve receber uma mensagem de erro que o ajude a corrigir a sua utilização ou remeter a correção para os programadores.

7.1. COMO APANHAR O ERRO

No 4GL existe a instrução WHENEVER que permite alterar a forma como o programa se comporta ao ser confrontado com uma situação de erro ou aviso (warning).

```
WHENEVER {ERROR | WARNING}
        {GOTO label | CALL nome_de_função | CONTINUE | STOP}
```

Figura 46 Definição genérica da instrução WHENEVER

A primeira opção nas cláusulas informa sobre a situção (se ocorrer) para a qual se vai tomar determinada acção. As situações são: erros e avisos. Numa situação de erro pediu-se ao programa que efectuasse determinada tarefa que ele não pode efectuar dessa forma e nesse contexto, a instrução WHENEVER é utilizada com a cláusula ERROR. Numa situação de aviso (warning) a tarefa pode ser cumprida ainda que não seja uma situação usual, a instrução WHENEVER é evocada com a cláusula WARNING.

A segunda cláusula diz que acção tomar quando confrontado com uma situação.As acções possiveis são:

GOTO label

Quando fôr encontrada esta situação, o programa deve continuar a correr a partir da instrução assinalada por uma label. A label é assinalada colocando um nome de label onde se pretende e o caracter ":".

```
FUNCTION     exp()
WHENEVER ERROR GOTO erro

SELECT *
        INTO pr_livros.*
        FROM livros

RETURN


LABEL erro: DISPLAY "ERRO"
END FUNCTION
```

Figura 47 Whenever com GOTO

CALL nome_de_função.

Quando for encontrada uma situação de erro, é executada a função nome_de_função.

```
FUNCTION     exp()
WHENEVER ERROR GOTO CALL erro()

SELECT *
        INTO pr_livros.*
        FROM livros

RETURN
END FUNCTION

FUNCTION     erro()
  DISPLAY "ERRO"
END FUNCTION
```

Figura 48 Exemplo de WHENEVER com CALL

STOP

Quando é utilizada a cláusula STOP, o programa ao ser confrontado com uma situação de erro ou aviso cujo código de erro seja negativo, o programa é abortado e envia para o écran uma pequena explicação do erro ocorrido.

CONTINUE

Quando é utilizada a cláusula CONTINUE o programa numca pára em situação de erro do RDSQL, pelo que para haver controlo da execução do programa, tem que testar a variável status ou estrutura SQLCA.

7.2. A VARIÁVEL STATUS

Existe em qualquer programa de 4GL uma variável denominada status, pré-definida como global, que é afectada a 0 se não existiu erro e diferente de 0 se ocorreu algum erro na execução de uma instrução.

7.3. A ESTRUTURA SQLCA

Por vezes a variável status não chega para identificar o erro ou aviso.Existe um RECORD chamado SQLCA, global, que identifica os tipos de erros e avisos. Geralmente usa-se a estrutura SQLCA como adicional ao status que identifica a exist^encia de erro.

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

Figura 49 RECORD SQLCA

O significado de cada uma das variáveis é:

SQLCODE:

Indica o resultado de uma instrução de RDSQL.Toma os valores:

0 — instrução bem sucedida.

NOTFOUND — busca bem sucedida mas não devolveu nenhuma linha.

SQLERRM: não é utilizado

SQLERRP: não é utilizado

SQLERRD:

É um array de 6 inteiros com o seguinte significado:

sqlerrd [1]: não é utilizado

sqlerrd [2]: Valor retornado pelo C-ISAM

(Ver erros do C-isam)

sqlerrd [3]: número de linhas processadas

sqlerrd [4]: não é utilizado

sqlerrd [5]: caracter da string em foi encontrado erro

sqlerrd [6]: é o ROWID da última linha

SQLWARN:

E um array de 8 caracteres que assinalam várias situações que necessitam de aviso durante a execucão de uma instrução RDSQL.

sqlwarn[1]:

Se contiver:

Caracter W: Um ou mais dos outros elementos do array contêm 'W'

Espaço: Não é necessário verificar os outros elementos do array.

sqlwarn[2]:

Se contiver:

Caracter W: Os dados de uma ou mais colunas foi truncado para caber dentro de uma variável.

Espaço: Nao houve truncagem de dados.

sqlwarn[3]:

Se contiver:

Caracter W: Foi encontrado um erro numa função agregada (SUM, AVG, MAX ou MIN) A NULL.

Espaço: Não houve problemas.

sqlwarn[4]:

Se contiver:

Caracter W: O número de items de um SELECT é diferente do número de variáveis da cláusula INTO.

Espaço: Não houve erro.

sqlwarn[5]: Nao é utilizado

sqlwarn[6]: Nao é utilizado

sqlwarn[7]: Nao é utilizado

sqlwarn[8]: Nao é utilizado

8. GESTÃO DE ECRANS EM 4GL

Uma das principais tarefas quando se desenvolve uma aplicação é a interação com o utilizador. Esta tarefa é geralmente muito demorada e extensa.

O 4GL, possui vários mecanismos que facilitam a criação rápida e poderosa de écrans de interação com o utilizador.Concretamente existe um gestor de janelas; um compilador de entradas de dados; instruções de acesso ás entradas de dados.

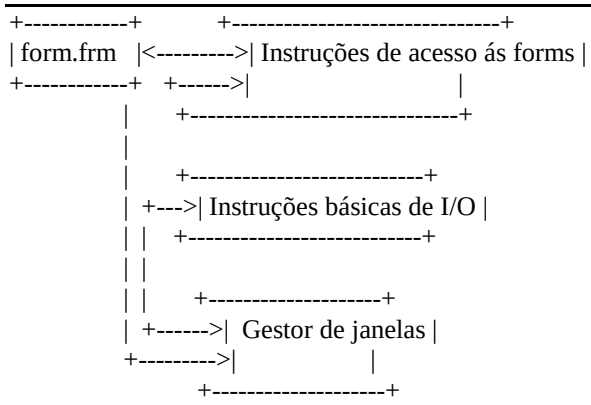


Figura 8.1 Interação entre os módulos de gestão de écrans

O módulo "form.frm" é um programa (habitualmente conhecido como form) em que se define o formato do écran de entrada de dados a utilizar, compilado com o programa form4gl.

As instruções de acesso ás forms fazem parte da linguagem 4GL, e têm como objectivo o acesso á form. As principais são: OPEN FORM; DISPLAY; DISPLAY ARRAY; INPUT; INPUT ARRAY; etc.

Algumas instruções de I/O básico já foram focadas, elas são: DISPLAY AT; PROMPT; MENU.

A interação com o écran é sempre efectuada por intermédio do gestor de janelas. Muitas das vezes o programador não se apercebe deste facto.Existem algumas instruções e opções para controlar este gestor, das quais se pode destacar: OPEN WINDOW; CURRENT WINDOW; CLOSE WINDOW; OPTIONS: MESSAGE LINE; etc.

As forms e instruções de INPUT, por si só não acedem nem escrevem em tabelas. Nas forms associam-se colunas de tabelas aos campos para que os programas possam definir os tipos de variáveis internas que vão utilizar, ou declaram-se directamente os campos a partir dos tipos básicos.

O fluxo de informação de um écran para a base de dados poderia ser descrito pelo seguinte esquema:

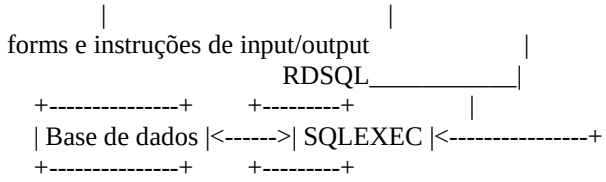


Figura 8 Comunicação dos écrans com a base de dados

A criação de uma entrada de dados análoga á gerada por umperform do informix SQL é feita construindo a form, e o programa 4GL que a vai gerir, com os seus menus, comunicações com a base de dados etc.

8.1. FORM

As forms servem para desenhar e processar écrans.

Uma form é escrita num ficheiro, por exemplo através dum editor de texto, e depois compilada pelo programa form4gl. A versão compilada pode então ser executada sempre que necessário pelo programa em 4gl.

Assim, a form tem como suporte físico dois ficheiros: o programa fonte com a extensão ".per" e o programa compilado com a extensão ".frm".

Uma form é um programa que se compila com o comando "form4gl" e gera um executável com a extensão .frm. A execução do .frm é feita pelo programa de 4GL que, durante a execução abre e manipula a form por intermédio de instruções próprias.

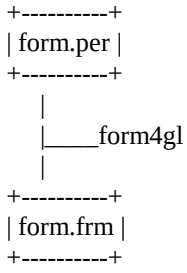


Figura 8.1 Compilação de uma form

Neste capítulo optámos por utilizar o user-menu em vez da compilação pela linha de comando.

8.1.1. O MENU FORM

Entre no menu INFORMIX-4GL e escolha a opção Form --> entrou no menu Form.

O menu Form apresenta as seguintes sete opções:

MODIFY

Esta opção permite fazer alterações a forms já existentes. Tal como na opção RUN aparece um écran com a lista das forms existentes para seleccionar uma delas. Ao escolher uma form aparece-lhe o source da form, num editor do sistema, pronto a ser alterado. Depois de concluidas as alterações e de ter saído do editor, aparece o menu MODIFY FORM com as seguintes opções:

COMPILE:

Compila a form. Se a form tiver erros estes são colocados num ficheiro juntamente com a form, podendo então corrigi-la e compilá-la de novo. Estes procedimentos podem ser executados ciclicamente até obter uma compilação da form sem erros.

SAVE-AND-EXIT:

Grava as alterações à form no ficheiro fonte, sem executar compilação.

DISCARD-AND-EXIT:

Não grava as alterações à form no ficheiro fonte, mantendo neste a versão anterior ao inicio das alterações. Também não executa compilação.

GENERATE

Cria automàticamente uma form com um formato por defeito (default). Quando esta opção é seleccionada aparecem écrans para selecção da base de dados, se ainda não foi seleccionada, e para escolha das tabelas que constituirão a form. Veremos mais em pormenor esta opção na secção seguinte "Criação e Compilação duma Form Default".

NEW

Permite criar uma form num novo ficheiro através dum editor. Quando terminar a edição da form o INFORMIX-4GL comportar-se-á de forma igual à da opção MODIFY.

COMPILE

Compila uma form já existente. Apresenta um écran para seleccionar a form a compilar. Se a compilação falhar comportar-se-á de forma igual à da opção MODIFY.

EXIT

Retorna ao menu INFORMIX-4GL.

8.2. O PERFORM DEFAULT.

Uma das caracteristicas mais simpáticas do form4gl é a que permite gerar automáticamente forms, associadas a uma ou mais tabelas.

8.2.1. CRIAÇÃO DUMA FORM DEFAULT

Primeiro vamos posicionar-nos no menu FORM e seleccionar a opção GENERATE. --> Aparece o écran NEW FORM, para a introdução do nome da form a gerar.

O nome a digitar será o nome com que a form será conhecida a partir daqui, e fica á escolha do programador.

Digite "autores" e faça RETURN. --> Aparece o écran CHOOSE TABLE com a lista das tabelas da base de dados livros, para que possa seleccionar as tabelas que farão parte da form.

Seleccione a tabela autores. --> Aparece o menu CREATE FORM, que pede para indicar se pretende ou não seleccionar mais tabelas.

Como só se pretende uma form sobre a tabela autores seleccione a opção TABLE-SELECTION-COMPLETE. --> Cria e compila a form "autores" e regressa ao menu FORM.

8.2.2. DESCRIÇÃO DUM PERFORM DEFAULT

Seleccione a opção MODIFY e a form "autores". --> Aparece um editor (de acordo com a variável de sistema DBEDIT ou o usado anteriormente na mesma sessão de trabalho), com o seguinte source de uma form.

```
database livros
screen
{
numero      [f000     ]
nome        [f001                    ]
            [f002                    ]
}
end
tables
autores
attributes
f000 = autores.numero;
f001 = autores.nome[1,35];
f002 = autores.nome[36,70];
end
```

Figura 8.4

For_____
_____
_____
_____
_____
_____
_____
_____
_____
_____

_____

_____

_____ntém a ligação entre a variável do écran (representada pelo seu código de identificação) e uma coluna da tabela autores.

Estas secções são o esqueleto de qualquer form. Manipulando e inserindo instruções nestas secções conseguem-se criar e explorar as capacidades das forms.

Como exercício tente alterar a apresentação e a colocação das variáveis e do texto livre da form gerada anteriormente. Para isso altere apenas as instruções da screen section, desenhando um écran a seu gosto, mantendo os códigos e comprimentos máximos das variáveis. Utilize a opção MODIFY.

8.3. O FORM4GL, A INSTRUÇÃO INPUT E AS SUAS CAPACIDADES

Neste capítulo vamos descrever a maior parte das instruções do form4gl e as instruções do 4GL que permitem a interação com as forms. Estas irão sendo introduzidas durante as várias fases do processo de desenvolvimento de um écran complexo, com várias tabelas, validações, mensagens de erro, etc. Seleccionaram-se as instruções normalmente utilizadas nos tipos de écrans que fazem parte de qualquer aplicação.

O écran que se vai construir começa por ser uma proposta genérica, sendo desenvolvida e aperfeiçoada por etapas. A ordem de apresentação destas é a que normalmente se utiliza na prática, salvo nalgumas situações para facilitar a exposição.

A proposta de partida e' a seguinte:

- Pretende-se um único écran para trabalhar com as tabelas livros, escritos, temas e editores da base de dados livros, nomeadamente para poder visualizar simultâneamente um livro (livros) os autores do livro (escritos) e os temas focados (temas). As entradas referentes a escritos, e temas devem aparecer em janelas diferentes.
- Os diversos campos devem ser validados (tipo de dados correcto, valores admissiveis, etc.). Sempre que exista um erro deve ser assinalado com uma mensagem descrevendo o erro e o cursor deve posicionar-se no campo em questão.
- As colunas que contenham códigos a procurar numa tabela devem possuir facilidades de busca na tabela de descodificação.

Uma form de 4GL possui alguns limites e definições de que convém falar:

- Todas as linhas para além de nº de linhas de um écran menos 4 são ignoradas.
- Todos os écrans para além do 1º são ignorados. No nosso caso vão existir três forms: uma para os livros, uma para os autores do livro em causa e uma para os temas focados.

8.3.1. DESENHO DA FORM LIVROS

Como ponto de partida vamos gerar a form de defeito livros para a tabela livros, obtendo-se o esqueleto do programa com o comprimento e a ligação das colunas às tabelas já definidos.

Para já não vamos efectuar validação do código de editor com a respectiva descodificação.

Crie o PERFORM default mencionado no parágrafo anterior.

Visualize-o usando a opção MODIFY, obtendo o seguinte programa:

```
database livros
screen
{
numero          [f000      ]
nome            [f001                    ]
                [f002                    ]
traducao        [f003                    ]
                [f004                    ]
volumes         [f005  ]
paginas         [f006  ]
edicao          [f007 ]
ano             [f008  ]
data_entrada    [f009      ]
sala            [f010 ]
estante         [f011 ]
prateleira      [f012 ]
observacoes     [f013                                ]
}
end
tables
livros
attributes
f000 = livros.numero;
f001 = livros.nome[1,30];
f002 = livros.nome[31,60];
f003 = livros.traducao[1,30];
f004 = livros.traducao[31,60];
f005 = livros.volumes;
f006 = livros.paginas;
f007 = livros.edicao;
f008 = livros.ano;
f009 = livros.data_entrada;
f010 = livros.sala;
f011 = livros.estante;
f012 = livros.prateleira;
f013 = livros.observacoes;
end
```

A seguir vamos trabalhar sobre a secção Screen para obter o desenho do écran pretendido, considerando que:

Todos os campos devem estar presentes num único écran.

Uma solução para o écran pretendido poderá ser a que se apresenta na screen section seguinte:

```
SCREEN
{


NUMERO DO LIVRO   [f000  ]
NOME DO LIVRO     [f001                          ]
TRADUCAO          [f004                        ]
VOLUMES           [f007  ]        PAGINAS          [f008  ]
EDICAO            [f009 ]
ANO               [f010]          DATA DE ENTRADA [f011    ]
COLOCACAO:  Sala [f012 ] Estante [f013 ]  Prateleira [f014 ]
OBSERVACOES       [f015                          ]
}
```

Figura 8.5 Screen section da form livros

8.3.2. ESPECIFICAÇÃO DAS TABELAS

As tabelas intervenientes são indicadas na secção "tables" do programa.

No nosso caso não é necessário alterar a tables section da form default criada anteriormente, que será:

```
  TABLES
    livros
```

Figura 8.6 TABLES section da form livros

Para já não se vai efectuar nenhuma alteração na attribute section, vamos antes optar por fazer o esqueleto do programa de 4GL que vai gerir a form.

8.3.3. PROGRAMA DE GESTÃO DA FORM LIVROS - VERSÃO 1

Sempre que se pretenda manusear uma form deve-se abri-la.

A abertura de uma form faz-se por intermédio da instrução OPEN FORM.

```
OPEN FORM nome_da_form FROM "ficheiro_da_form"
```

Figura 8.7 Definição genérica da instrução OPEN FORM

O Nome da form é dado pelo programador e é com esse nome que se passa a a referenciar o écran dentro do programa. "ficheiro_da_form" é o ficheiro em que se encontra a formcompilada retirando a extensão ".frm".

```
OPEN livros FROM "livros"
```

Figura 8.7 Abertura da form livros

Neste caso foi resolvido identificar a form livros com o mesmo nome do ficheiro.

Na form livros vão-se editar todas as colunas da tabela livros. O programa deve ter variáveis para todas as colunas, podemos declarar facilmente um record que contenha todas as colunas da tabela livros.

```
DEFINE
        pr_livros LIKE livros.*
```

Figura 8.9 Declaração do record pr_livros

Antes de qualquer acção sobre a entrada de dados, deve-se mostrar esta no ecran sem qualquer dado. Para efectuar esta acção usa-se a instrução DISPLAY FORM.

```
DISPLAY FORM nome_da_form ATTRIBUTE(lista_de atributos)
```

Figura 8.10 Definição genérica da instrução DISPLAY FORM

Este programa vai ter um menu que permite as seguintes opções sobre a tabela de livros:

NOVO

Adiciona um novo livro á tabela de livros.

MUDAR

Muda a linha presente no écran, se existir corrente.

PROCURAR

Procura um livro por número.

REMOVER

Remove, se existir, a linha presente no écran.

FIM

Sai do programa.

Para criar menus utiliza-se a instruçào MENU já estudada anteriormente.

```
MENU "LIVROS"
        COMMAND "Novo" "Adiciona um novo livro"
                CALL novo()
        COMMAND "Procurar" "Procurar um livro pelo numero"
                CALL procurar()
        COMMAND "Mudar" "Permite mudar o livro corrente"
                CALL mudar()
        COMMAND "Remover" "Remove o livro corrente"
                CALL remover()
        COMMAND "Fim" "Sai do programa"
END MENU
```

Figura 8.11 Menu da entrada de dados para livros

A função novo recebe os dados de um livro do teclado e insere-o na base de dados.

A instrução que se utiliza para editar uma form (para receber dados por intermédio de uma form), é a instrução INPUT.

Esta instrução tem como parâmetros um conjunto de nomes de campos (que correspondem aos campos que se pretende editar) que foram préviamente identificados na secção attributes da form e um conjunto de variáveis que também podem ser definidas de várias formas. É também possivel executar instruções e controlar o movimento do cursor condicionalmente á posição do cursor.

```
INPUT [BY NAME lista_de_variaveis
                    [WHITHOUT DEFAULTS]
                    |lista_de_variaveis [WITHOUT DEFAULTS]
          FROM {lista_campos|screen_record[[n]].*}[,...}}
          [HELP numero_do_help]
          [{BEFORE FIELD sublista_de_campos
          |AFTER {FIELD sublista_de_campos|INPUT}
          |ON KEY(lista_de_teclas)}
                    instrução
                    ...
                    [NEXT FIELD nome_do_campo]
                    ...
                    [EXIT INPUT]
          ...
          ...
END INPUT
```

Figura 8.12 Definição genérica de uma instrução INPUT

Nesta função vamos utilizar a instrução INPUT na sua forma mais simples.

```
FUNCTION     novo()
  INPUT BY NAME pr_livros.*

WHENEVER ERROR CONTINUE
INSERT INTO livros VALUES(pr_livros.*)
IF  status = 0 THEN
          ERROR "Livro inserido"
ELSE
          ERROR "Erro na insercao do livro",status
END IF
WHENEVER ERROR STOP
END FUNCTION
```

Figura 8.12 Introdução de novo livro

A lista de campos da cláusula FROM, neste caso está implicita no caracter '*' do record. Isto quer dizer que vão ser editados todos os campos que existem na variável composta pr_livros. Como a variável foi declarada LIKE livros.*, vão ser editados todas as colunas da tabela livros.

A instrução INSERT é efectuada como foi definida no RDSQL tendo na cláusula VALUES, as variáveis que contém os valores pretendidos.

As instruções WHENEVER aparecem para que o programa não pare a sua execução, e seja enviado um erro de SQL quando houver erro no INSERT (ver tratamento de erros).

A função procurar, recebe o número do livro que se pretende procurar, procura-o na base de dados, e mostra no écran todos os dados correspondentes. Note que o número do livro é introduzido na form.

```
FUNCTION     procurar()

CLEAR FORM

INPUT BY NAME pr_livros.numero

SELECT *
        INTO pr_livros.*
        FROM livros
        WHERE numero = pr_livros.numero

IF status != 0 THEN
        ERROR "Livro nao existente"
ELSE
        DISPLAY BY NAME pr_livros.*
END IF

END FUNCTION
```

Figura 8.13 Função que procura um livro com determinadonúmero"

A instrução CLEAR FORM serve para limpar todos os campos existentes na form corrente

Se a busca foi bem sucedida faz-se um display de todas as colunas da tabela.

A função mudar altera um livro que tenha sido préviamente procurado ou inserido, ou seja, se este estiver no écran.

Esta função aceita os valores sem limpar variáveis e sem limpar a form, validando a existência de um livro válido.

```
FUNCTION     mudar()
IF pr_livros.numero IS NULL THEN
        ERROR "Nao ha nenhum livro para mudar"
ELSE
        INPUT BY NAME pr_livros.* WITHOUT DEFAULTS
        UPDATE livros
                SET livros.* = pr_livros.*
                ERE livros.numero = pr_livros.numero
END IF
END FUNCTION
```

Figura 8.14 Alteração de um livro

É feito um teste á variável pr_livros.numero para garantir que esta está preenchida com um número de livro existente.

A instrução INPUT aparece com a cláusula WITHOUT DEFAULTS. Se esta cláusula for omitida faz com que a instrução INPUT, quando chamada, coloque nos campos, os defaults definidos nas forms ou no

ficheiro de defaults syscolval (ver utilitário upscol). Se a cláusula existir, os valores presentes nos campos durante o INPUT são os valores que estão dentro das variáveis.

A instrução UPDATE está descrita no RDSQL, as variáveis do 4GL colocam-se na cláusula SET.

A função remover, remove o livro corrente.

Esta função retira da base de dados o livro correspondente aos dados presentes no écran.

```
FUNCTION     remover()
IF pr_livros.numero IS NULL THEN
        ERROR "Nao ha nenhum livro para remover"
ELSE
        DELETE FROM livros
                WHERE @numero = pr_livros.numero
        IF status != 0 THEN
                ERROR "Erro na remocao"
        ELSE
                INITIALIZE pr_livros.* TO NULL
                CLEAR FORM
                ERROR "Livro removido"
        END IF
END IF
END FUNCTION
```

Figura 8.15 Remoção de um livro

O teste da existência de livro corrente é feito da mesma forma que na função mudar.

As instruções INITIALIZE e CLEAR FORM aparecem para garantir a não existência de livros correntes depois de uma remoção.

8.3.4. DEFINIÇÃO DE ATRIBUTOS (ATTRIBUTES SECTION)

Continuando o nosso exercício, vamos agora desenvolver a Attributes Section. Esta define o comportamento e o aspecto de cada um dos campos do écran definidos na Screen Section. Qualquer campo da Screen Section deve ser descrito na Attributes Section.

Na Attributes Section define-se:

- Como o form4gl apresenta o campo no écran.
- Limites para os valores a entrar no campo do écran.
- O tipo do campo por duas formas: a coluna e tabela a que corresponde ou tipo elementar do 4GL.
- O nome do campo para as instruções de 4GL que tratam do manuseamento de écrans. Se o campo fôr formonly, pode-se escolher o nome, senão o nome é o da coluna a que corresponde.

Se qualquer um dos atributos descrito for utilizado, quando se faz um INPUT sem a cláusula WHITHOUT DEFAULTS, os defaults utilizados são os dos attributes, caso contrário serão utilizados os defaults da tabela syscolval.

A seguir iremos descrever todos os atributos disponiveis e aplicar, na form livros, os de uso mais comum.

8.3.4.1. AUTONEXT

Este atributo faz com que o cursor, durante uma instrução de INPUT ou CONSTRUCT, salte para o campo seguinte quando o campo actual estiver totalmente preenchido.

É muito útil e na form livros iremos usá-lo nas datas. Exemplo:

```
f011 = livros.data_entrada, autonext;
```

8.3.4.2. COMMENTS

Este atributo associa uma mensagem a um campo. Esta mensagem aparecerá na linha correspondente do écran sempre que o cursor se encontre no campo respectivo.

Na form livros iremos aplicá-lo ao campo data_entrada com a seguinte mensagem:

"Introduza a data em que o livro entrou na biblioteca"

Exemplo:

```
f011 = livros.data_entrada, autonext,
        comments = "Introduza a data em que o livro entrou na          biblioteca";
```

8.3.4.3. DEFAULT

Insere um valor inicial no campo do écran a que está associado, quando se evoca a instrução INPUT.

Na form livros iremos aplicá-lo aos campos volumes, sala e data_entrada. Neste último campo (a data da entrada do livro) vai-se inserir a data do dia, escrevendo "today" como valor inicial. Exemplo:

```
f011 = livros.data_entrada, autonext, default = today;
```

8.3.4.4. FORMAT

Permite controlar o formato de display dos campos a que está associado, desde que tenham tipo de dados DECIMAL, FLOAT, SMALLFLOAT ou DATE.

Os caracteres de controlo do formato são os seguintes:

###.##

Para campos numéricos (decimal, float e smallfloat). Indica o número de campos à esquerda e à direita do ponto decimal.

mm

Para datas. Representação do mês através de dois **dígitos.**

mmm

Para datas. Representação do nome do mês através de abreviatura com três letras (em inglês).

dd

Para datas. Representação do dia através de dois dígitos.

ddd

Para datas. Representação do nome do dia através de abreviatura com três letras (em inglês).

yy

Para datas. Representação do ano através de dois dígitos.

yyyy

Para datas. Representação do ano através de quatro dígitos.

As datas podem ter os separadores "-" ou "/" à escolha.

Na form livros iremos usar o atributo "format" apenas nas datas e com o formato "yyyy/mm/dd". Exemplo:

```
f011 = livros.data_entrada, autonext, default = today,
        format = "yyyy/mm/dd";
```

8.3.4.5. INCLUDE

Permite especificar valores ou intervalos de valores admissiveis a introduzir no campo associado a este atributo.

Pode especificar uma série de valores separados por virgulas ou um valor mínimo e um valor máximo separados pela palavra "to".

Na form livros vamos usar o atribute include nos campos:

volumes: Admitindo que um livro não pode ter mais de 100 volumes (1 a 100).

Exemplo:

```
f007 = livros.volumes, include = (1 to 100);
```

8.3.4.6. NOENTRY

Permite evitar a inserção de dados quando executa uma instrução INPUT

8.3.4.7. REQUIRED

Este atributo torna a inserção obrigatória na coluna a que for associado, durante a execução da instrução INPUT.

Na form livros vamos tornar a inserção obrigatória nas seguintes colunas: livros.numero, livros.nome. Exemplo:

```
f000 = livros.numero,required;
```

8.3.4.8. REVERSE

Este atributo mostra os campos, a que está associado, com fundo luminoso (reverse video).

Na form livros vamos usar em reverse video o campo número. Exemplo:

```
f000 = livros.numerlo, reverse;
```

8.3.4.9. PICTURE

Permite definir a forma dum campo do écran, com tipo de dados alfanumérico.

A forma do campo define-se, no atributo "picture", combinando om qualquer outro caracter os seguintes caracteres:

A Representa qualquer letra.

Representa qualquer dígito.

X Representa qualquer caracter.

8.3.4.10. VERIFY

Este atributo usa-se quando se pretende introduzir, num determinado campo, o mesmo valor duas vezes para reduzir a probabilidade de erro na entrada dos dados.

8.3.4.11. DOWNSHIFT

Transforma as letras maiúsculas, dum campo alfanumérico, em letras minúsculas.

8.3.4.12. UPSHIFT

Transforma as letras minúsculas, dum campo alfanumérico, em letras maiúsculas.

8.3.4.13. A Attributes Section da form livros

Apresenta-se a seguir a Attributes Section do ecran proposto como o objectivo de alcançar o que foi sendo descrito ao longo dos últimos capítulos.

```
ESCREVER AQUI A SECÇÃO DE ATTRIBUTES
```

8.3.5. A INSTRUCTION SECTION

A Instruction Section é a última secção da form e é opcional.

No 4GL a única operação que podemos efectuar nesta secção é a instrução DELIMITERS e a definição de SCREEN RECORDS.

8.3.5.1. Alteração dos Caracteres de Enquadramento - DELIMITERS

Com o comando "delimiters" pode alterar os caracteres de enquadramento do campo do ecran, quando este é visualizado através da execução da form pelo PERFORM. Os caracteres default são os parêntesis rectos "[...]". Exemplo:

```
delimiters "{}"
```

em que "{" é o caracter de abertura e "}" o caracter de fecho.

8.4. OUTRAS CAPACIDADES DA INSTRUÇÃO INPUT E OUTROS ATTRIBUTES DAS FORMS

No capítulo anterior a instrução INPUT foi utilizada na sua forma mais simples. Para além da cláusula onde se definem as variáveis, a instrução INPUT pode ser utilizada com mais cláusulas, nomeadamente:

HELP numero

Quando a tecla de help fôr pressionada, durante a edição de uma form, é automáticamente executada a função show_help, que apresenta o help relativo ao número especificado (ver helps).

BEFORE FIELD nome

Antes da edição da coluna cujo nome é especificado o programa cumpre as instruções especificadas a seguir á cláusula.

AFTER FIELD nome

Depois da edição da coluna cujo nome é especificado, o programa cumpre as instruções especificadas a seguir á cláusula.

AFTER INPUT

Depois da edição da form o programa cumpre as instruções especificadas a seguir á cláusula.

ON KEY tecla

Se fôr pressionada uma das teclas definidas nesta cláusula, o programa cumpre as instruções definidas a seguir.

Quando se definiu o programa falou-se na validação do campo editora na tabela editores, com envio para o écran do nome da editora. Vamos agora alterar o programa e form de modo a cumprir essa tarefa.

A form tem que possuir um novo campo onde será visualizado o nome da editora, assim:

```
EDICAO          [f009 ]   [f9d                          ]
```

Figura 8.15 Campo para visualização do nome de editora

Tem também de se introduzir os atributos para este campo:

```
f9d  = FORMONLY.editor TYPE LIKE editores.nome, noentry;
```

Figura 8.16 Atributo do novo campo

O atributo FORMONLY identifica um campo com um nome permitindo dizer qual o tipo para esse campo. Neste caso definimos o tipo com LIKE , no entanto é possivel definir o tipo através dos tipos elementares.

```
campo = FORMONLY.nome_do_campo
  [TYPE [tipo_de_dados | LIKE tabela.coluna]]
  [NOT NULL][,lista_de_atributos]
```

Figura 8.17 Definição genérica do atributo FORMONLY

Utilizou-se também o atributo noentry para forçar a instrução INPUT a não editar este campo.

O programa de 4GL, ainda não sabe como fazer a descodificação do código de editor. Para conseguir efectuar esta função, utiliza-se a cláusula AFTER FIELD, faz-se uma consulta á tabela de editores e envia-se o nome do editor para o écran.

```
INPUT BY NAME pr_livros.*
        AFTER FIELD edicao
                SELECT nome
                        INTO editor
                        FROM editores
                        WHERE codigo = pr_livros.edicao

                IF status = NOTFOUND THEN
                        ERROR "Editor nao existente"
                        NEXT FIELD edicao
                ELSE
                        DISPLAY BY NAME editor
                END IF

END INPUT
```

Figura 8.18 Alteração das instruções INPUT para efectuar as descodificações

Quando a variável status é afectada com NOTFOUND, quer dizer que não foi encontrada a editora cujo código foi introduzido, pelo que é enviada uma mensagem de erro e indica-se que o próximo campo a editar é o mesmo campo (não sai do campo enquanto não for introduzido código correcto).

8.4.1. O COMANDO FORM4GL

O processo de criação duma form através do sistema operativo pode ser constituido pelas mesmas etapas que usámos no desenvolvimento da form f_livros, ou seja a criação duma form default, através do comando form4gl, e posterior alteração através dum editor de texto.

O comando form4gl tem a estrutura dum comando UNIX, com três opções disponíveis.

-s

Esta opção compila a form fonte existente no ficheiro cujo nome segue a opção no comando. Note-se que o verdadeiro nome do ficheiro é constituído pelo nome existente no comando seguido pela extensão ".per". No caso de existirem incorrecções, cria um ficheiro com o mesmo nome da form fonte mas com extensão ".err", e que contém as mensagens que indicam os pontos onde se verificam as incorrecções. Pode-se editar este ficheiro e corrigir os erros e, depois de se apagarem as mensagens de erro, gravá-lo no ficheiro com extensão ".per" e compilá-lo de novo.

-v

Esta opção também compila a form fonte mas verifica se o comprimento dos campos definidos na SCREEN SECTION têm o mesmo comprimento das colunas a que são associados na ATTRIBUTES SECTION, e apresenta as respectivas mensagens no ficheiro de erros.

-d

Esta opção cria uma form default tal como a opcão GENERATE do menu FORM. Pode especificar o nome da form, a base de dados e as tabelas que compoêm a form escrevendo-as por esta ordem e a

seguir à opção **-d** do comando. No entanto se escrever apenas o comando seguido da opção -d e pressionar RETURN o comando pedir-lhe-á que introduza esses elementos e aceita-os sempre que pressionar RETURN. Para indicar que não pretende especificar mais tabelas pressione RETURN pela segunda vez.

Exemplos:

```
form4gl -d f_livros livros autores livros
```

Cria e compila a form f_livros com as tabelas livros e autores da base de dados livros.

```
form4gl -v f_livros
```

Compila a form f_livros.

9. INTERACÇÃO DO 4GL COM AS BASES DE DADOS

9.1. GESTÃO DE CURSORES

A linguagem 4GL, como linguagem de programação serve-se das variáveis para armazenar dados durante a execução do programa (valores digitados pelo utilizador, resultados de cálculos intermédios, etc.). Tal como na maioria das linguagens, as variáveis de 4GL têm uma dimensão fixa e determinada á partida. A caracteristica de linguagem de acesso a uma base de dados (através do RDSQL) faz com que o 4GL tenha também a possibilidade de manusear entes de uma base de dados (tabelas, colunas, views, etc). A instrução **tabela**, ou seja um conjunto de linhas de dimensão variável e desconhecida á priori. Pela razão anterior é impossivel ao construir um programa declarar variáveis para armazenar o resultado de uma instrução SELECT, excepto em casos muito particulares, por esta razão tornou-se necessário introduzir a noção de cursor com vista a resolver estas questões.

Uma instrução SELECT pode dar origem a mais que uma linha da base de dados (depende da cláusula WHERE).

```
% cat errselect.4gl

DATABASE livros

MAIN
DEFINE
        numero LIKE livros.numero,
        nome   LIKE livros.nome,
        edicao LIKE livros.edicao

        SELECT numero, nome, edicao
                INTO numero, nome, edicao
                FROM livros
                WHERE livros.edicao = "BERTR"

END MAIN

% c4gl errselect.4gl
% a.out
Program stopped at "errselect.4gl" line number 13.
SQL statement error number -201.
ISAM error number 0.
A syntax error has ocurred.
%
```

Figura 9.1 Leitura com instrução SELECT que dá origem a mais do que uma linha.

Se consultar o manual de 4GL verificará que o erro corresponde a um erro de sintaxe. Este erro aconteceu porque uma busca não devolveu exactamente uma linha.

Um cursor é uma forma de aceder individualmente ás linhas de resultado de uma instrução SELECT. Existe um segundo tipo de cursores associados a instruções INSERT de que se falará mais adiante. Para aceder a uma linha pertencente a esse cursor usa-se a instrução FETCH.

Ao conjunto de linhas seleccionados como uma instrução SELECT para um cursor chama-se *conjunto* activo para essa instrução SELECT.

Á linha activa em determinado momento chama-se *linha corrente.*

Um cursor pode estar em dois estados: **aberto** ou **fechado.** Quando está aberto, o cursor tem associado um conjunto activo e aponta para uma linha corrente. Quando um cursor está fechado, deixa de estar associado a um conjunto activo, continuando no entanto associado á instrução SELECT para a qual foi declarado.

As acções que se podem realizar sobre um cursor são:

1. Declarar o cursor. Esta acção é efectuada com o auxilio da instrução DECLARE, e associa-o a uma instrução SELECT.

```
DECLARE cr_livros CURSOR FOR
        SELECT numero, nome, edicao
        FROM livros
        WHERE numero < 5;
```

2. Abrir o cursor. Esta acção é efectuada com a instrução OPEN. Esta provoca a execução da instrução SELECT associada ao cursor e a correspondente busca na base de dados é efectuada nessa altura. A partir desta instrução até ao cursor ser fechado fica-lhe associado um conjunto de linhas (conjunto activo) que corresponde ao resultado da instrução SELECT. O cursor fica posicionado imediatamente antes da primeira linha.

```
OPEN cr_livros
```

3. Aceder uma ou mais vezes ao cursor. Os acessos são feitos com o auxilio da instrução FETCH. Esta avança a linha corrente para a linha seguinte e escreve os dados desta nas variáveis definidas na cláusula INTO. Se o cursor estiver posicionado na ultima linha, é afectada a variável sqlca.sqlcode com o valor NOTFOUND(=100).

```
WHILE 1
        FETCH cr_livros INTO numero, nome, edicao;

        IF  sqlca.sqlcode != 0 THEN
                EXIT WHILE
        END IF
        DISPLAY numero, nome, editor
END WHILE
```

4. Fechar o cursor. Acção efectuada pela instrução CLOSE. Colocação do cursor em estado fechado. Não é possível a evocação de outra instrução sobre o cursor que não seja OPEN. O cursor por ser fechado não deixa de ter uma instrução SELECT associada, pelo que ao evocar a instrução OPEN é executada novamente a instrução SELECT associado ao cursor.

```
CLOSE cr_livros
```

Um cursor pode também ser fechado através de uma instrução de fim de transação (COMMIT WORK ou BEGIN WORK), já que uma das acções destas instruções é fecharem todos os cursores que se encontrarem abertos na altura.

Apresenta-se um exemplo de acesso a uma tabela da base de dados por intermédio de um cursor.

```
% cat cursor1.4gl

DATABASE livros

MAIN
DEFINE
        numero LIKE livros.numero,
        nome   LIKE livros.nome,
        aux    SMALLINT

DECLARE cr_livros CURSOR FOR
        SELECT @numero, @nome
                FROM livros
                WHERE @numero < 5

LET aux = 1
OPEN cr_livros

WHILE aux = 1
        FETCH cr_livros INTO numero, nome
        IF  sqlca.sqlcode != 0 THEN
                EXIT WHILE
        END IF
        DISPLAY numero, nome
END WHILE

CLOSE cr_livros

END MAIN
% c4gl cursor1.4gl
% a.out
1Artificial Inteligence Using C
2C Power User's Guide
3Microprocessors
4The Complete Reference
%
```

Figura 9.2 Leitura dos livros com número < 5 usando um cursor

Quando se declara um cursor, pode-se anunciar a intenção de alterar os valores de algumas coluna das linhas lidas. Essa intenção anuncia-se através da declaração "FOR UPDATE OF ..." a seguir á instrução SELECT, como se pode ver na seguinte figura.

```
DECLARE nome_cursor CURSOR FOR
        instrução_select
        FOR UPDATE OF lista_de_colunas
```

Figura 9.3 Definição genérica da declaração de um cursor FOR UPDATE

A Alteração faz-se da seguinte forma:

1. Declarar um cursor "FOR UPDATE"

```
DECLARE cr_livros FOR
        SELECT numero, nome, edicao
                FROM livros
                WHERE numero < 5
        FOR UPDATE OF edicao
```

2. Abrir o cursor da forma descrita anteriormente.

```
OPEN cr_livros;
```

3. Posicionar o cursor na linha que se quer alterar:

```
FETCH cr_livros INTO ...
```

Evocar a instrução UPDATE, alteração das colunas definidas na cláusula SET do elemento corrente do cursor

```
UPDATE
        SET
        WHERE CURRENT OF cr_livros
```

Depois da instrução UPDATE o cursor continua posicionado na linha corrente.

```
% cat cursupd.4gl

DATABASE livros

MAIN
DEFINE
        numero LIKE livros.numero,
        nome   LIKE livros.nome,
        edicao LIKE livros.edicao,
        ans    CHAR(1),
        aux    SMALLINT

DECLARE cr_livros CURSOR FOR
        SELECT @numero, @nome, @edicao
                FROM livros
                WHERE @numero < 5
        FOR UPDATE OF edicao

OPEN cr_livros

WHILE aux = 1
        FETCH cr_livros INTO numero, nome, edicao

        IF  edicao IS NULL THEN
                PROMPT nome CLIPPED,
                        "-Qual a editora para este livro: "
                        FOR edicao
                UPDATE livros
                        SET edicao = edicao
                        WHERE CURRENT OF cr_livros

                PROMPT "Quer alterar mais livros (s/n):"
                        FOR CHAR ans
                IF ans != "s" THEN
                        EXIT WHILE
                END IF
        END IF

        IF  sqlca.sqlcode != 0 THEN
                EXIT WHILE
        END IF
END WHILE
END MAIN
% c4gl cursupd.4gl
% a.out
Artificial Inteligence using C-Qual a editora para este livro: BERTR
Quer alterar mais livros (s/n): n
%
```

Figura 9.4 Cursor para alteração (FOR UPDATE).

9.1.1. CURSORES COM SCROLL

O mecanismo descrito anteriormente só permite o posicionamento sequencial do principio para o fim no conjunto activo e, quando chegar ao fim só é possível voltar ao principio fechando e voltando a abrir o cursor (não garantindo que seja igual ao anterior pois a base de dados pode ter sido actualizada entretanto).

Para evitar estes problemas usa-se os cursores com scroll (ou scroll cursor). O seu funcionamento é idêntico a um cursor normal sendo declarado, aberto, acedido e fechado (DECLARE, OPEN, FETCH e CLOSE). A principal diferença entre os dois tipos de cursores reside no facto de os cursores de SCROLL não poderem ser utilizados com a cláusula FOR UPDATE. A instrução FETCH exige no entanto outra palavra que define a forma do posicionamento no cursor.

As formas de acesso são:

```
- Elemento corrente                    - FETCH CURRENT
- Elemento anterior                    - FETCH PREVIOUS
- Elemento seguinte                    - FETCH NEXT
- Primeiro elemento                    - FETCH FIRST
- Último elemento                      - FETCH LAST
- Elemento corrente n do conj. activo (n positivo ou negativo
                                       - FETCH RELATIVE
- Elemento n do conj. activo (n positivo)
                                       - FETCH ABSOLUTE
```

Figura 9.5 Formas de acesso a um cursor com scroll

Um cursor com scroll não pode, no entanto ser declarado FOR UPDATE.

9.1.2. CURSORES DE INSERÇÃO

Para optimizar a inserção numa base de dados pode-se utilizar um mecanismo de 4GL chamado cursor de inserção (insert cursor). Através deste mecanismo os dados vão sendo colocados numa área de memória temporária e só são efectivamente escritos na base de dados quando esta área se encontrar cheia (automáticamente) ou quando o programa assim determine (instrução FLUSH).

Manuseamento proposto de cursores de inserção:

1. Declarar um cursor associando-o a uma instrução INSERT.

```
DECLARE cr_livros CURSOR
        FOR INSERT INTO livros(numero, nome, adicao)
        VALUES (numero, nome, adicao)
```

2. Abrir o cursor. Identico a um cursor normal.

```
OPEN cursor_livros
```

3. Ler os dados para dentro de determinada variáveis.

4. Escrever as variáveis no buffer com o auxilio da instrução PUT.

```
PUT cr_livros
```

5. Fechar o cursor

```
CLOSE cursor_livros
```

O 4GL força a escrita do buffer na base de dados sempre que o limite do buffer seja atingido ou o cursor seja fechado.

Depois de uma instrução FLUSH deve-se verificar a variável status que se fôr diferente de zero indica erro na inserção na base de dados. Se pretender saber o numero de linhas inseridas deve consultar a variável sqlca.sqlerr[3], que deve conter o numero de linhas inseridas na base de dados.

9.2. GESTÃO DE INSTRUÇÕES DINÂMICAS DE RDSQL

Geralmente as aplicações são escritas de forma a executar tarefas pré-determinadas sobre bases de dados de estrutura constante.

Existem situações em que o programador não sabe, na fase da compilação, as instruções de RDSQL a utilizar. Algumas dessas situações podem ser as seguintes:

- Programas interactivos, onde os utilizadores introduzem os parâmetros de uma instrução pelo teclado.
- Programas em que o utilizador introduz as instruções de RDSQL pelo teclado.
- Programas que devem trabalhar com bases de dados em que a estrutura pode variar.
- Programas que recebem os parâmetros ou as instruções a partir de ficheiros de configuração.

Uma instrução de RDSQL, apesar de ser testada a sua sintaxe durante a compilação, é interpretada em execução, como tal é efectuado todo o parsing necessário para reconhecer a instrução e as suas cláusulas, todas estas tarefas são relativamente necessárias.

Em 4GL pode-se gerar dinamicamente instruções de RDSQL, quer isto dizer que as instruções podem geradas numa string, preparadas, e posteriormente executadas. A uma instrução preparada é dada uma identificação que permite a sua utilização. Uma instrução preparada, quando executada já não é efectuado todo o parsing da instrução, aconselhando-se utilizar instruções preparadas quando utilizar ciclos a fim de diminuir o tempo de execução do programa.

9.2.1. AS INSTRUÇÕES PREPARE E EXECUTE

Para executar uma instrução dinâmica seguem-se os seguintes passos:

1. Colocação de uma string com as instruções pretendidas dentro de uma variável tipo CHAR.

2. Definição de um identificador de instrução a partir da string que contém a instrução. Esta tarefa e excutada pela instrução PREPARE. Há validação da sintaxe da instrução contido na string.

```
PREPARE del1 FROM delete1
```

Figura 9.6 Instrução PREPARE

3. Execução da instruçao. 'E efectuada pela instrução EXECUTE.

```
EXECUTE del1
```

Figura 9.7 Instrução EXECUTE

As instruções RDSQL enviadas ao PREPARE em run-time não se podem referir a variaveis do 4gl, visto que o nesta fase o programa já foi compilado.

Para resolver este problema coloca-se o caracter '?' no local onde se deveria encontrar essa variável. Para assignar a variável ao ponto de interrogação usa-se a cláusula USING var1 var2 ... varn na instrução EXECUTE. As variaveis ficam assignadas ao ponto de interrogação correspondente á sua posição.

9.2.2. A INSTRUÇÃO DECLARE NAS INSTRUÇÕES DINAMICAS

Se a instrução dinamica for um SELECT, não se pode utilizar a instrução EXECUTE, devido á possivel ambiguidade relativa ao numero de linhas. Neste caso as fases de actuação são:

1. Preparar a instrução com PREPARE.

```
PREPARE id_de_expressão FROM expressão_char
```

2. Declarar o cursor para essa instrução.

```
DECLARE CURSOR FOR state_id
```

3. Usar o cursor da mesma forma que se usa com instruções definidas antes da compilação.
   Se for usado o caracter '?' para substituir variaveis é necessário usar a cláusula USING na abertura do cursor.

```
OPEN CURSOR USING var1 var2 ... varn
```

10. O GESTOR DE JANELAS

É comum, num programa a utilização de pequenas partes do écran para efectuar algumas tarefas a que se costuma chamar janelas.

10.1. OPEN WINDOW

Para abrir uma janela no écran utiliza-se a instrução OPEN WINDOW. As cláusulas desta instrução definem o tamanho e o posicionamento da janela no écran, pode associar uma form á janela, e define um atributo para a janela.

Assim que é executada a instrução OPEN, a janela aparece no écran.

```
OPEN WINDOW nome_da_janela AT linha,coluna
WITH { numero_de_linhas ROWS numero_de_colunas COLUMNS
|FORM "ficheiro_da_form"}
[ATTRIBUTE (lista_deªtributos)]
```

Figura 10.1 Definição genérica da instrução OPEN WINDOW

Na cláusula AT diz-se quais as coordenadas do canto superior esquerdo da janela. A cláusula WITH serve para dizer qual o tamanho da janela e pode ser feita de duas maneiras: dizendo o numero de linhas e de colunas que se pretende para a janela ou associando a uma form. No caso de se associar a janela a uma form, o 4GL atribui como tamanho da janela o tamanho da form, e abre a form (se escolher esta opção não necessita de chamar a instrução OPEN FORM). A cláusula ATTRIBUTE é opcional e associa um atributo á janela. Os atributos podem ser os definidos no capítulo da instrução DISPLAY AT mais o atributo border, que faz com que a janela crie uma caixa com os caracteres gráficos do terminal (se definidos no termcap) á volta da janela.

10.2. CLOSE WINDOW

Para retirar a janela do écran utiliza-se a instrução CLOSE WINDOW. Assim que esta instrução é executada, a janela desaparece do écran. Se a janela estiver associada a uma form, quando se fechar a janela, a form é fechada automáticamente.

```
CLOSE WINDOW nome_da_janela
```

Figura 10.2 Definição genérica da instrução CLOSE WINDOW."

10.3. CURRENT WINDOW

Durante alguns processamentos, você pode pretender mudar de janela de trabalho (ou janela corrente). A instrução CURRENT WINDOW transforma em janela corrente a janela da qual se passa o nome.

```
CURRENT WINDOW IS nome_da_janela
```

Figura 10.3 Definição genérica da instrução CURRENT WINDOW

A janela da qual se dá o nome tem de estar aberta. Se á janela especificada estiver associada uma form, essa form passrá a ser a form corrente.

Se a janela estiver debaixo de outras, quando evocar a instrução ela sobrepõe-se a todas que estejam no espaço utilizado pela janela pedida.

Existe uma janela especial, chamada screen, que está sempre aberta, e onde são utilizadas as forms não associadas a janelas, feitos os displays, etc.

10.4. OPTIONS

Um programa de 4GL assume valores por defeito para o funcionamento de diversas instruções. Para alterar esses defeitos usa-se a instrução OPTIONS.

```
OPTIONS { MESSAGE LINE linha |
          PROMPT LINE linha  |
          COMMENT LINE linha |
          ERROR LINE   linha |
          FORM LINE    linha |
          INPUT {WRAP | NO WRAP} |
          INSERT KEY tecla     |
          DELETE KEY tecla     |
          NEXT KEY   tecla     |
          PREVIOUS KEY tecla   |
          ACCEPT KEY tecla     |
          HELP FILE "ficheiro"  |
          HELP KEY tecla }
[,...]
```

Figura 10.4 Definição genérica da instrução OPTIONS

11. GESTÃO DE ÉCRANS – 2ª PARTE

11.1. INSTRUÇÃO CONSTRUCT

Durante a execução de um programa, muitas vezes o utilizador necessita de fazer uma consulta a várias colunas de uma ou mais tabelas (para fazer alterações em écran, gerar listagens parciais, etc.). Não é viável fazer um programa que preveja todas as consultas que se pretendam efectuar.

A instrução CONSTRUCT edita a form corrente, e a partir dos dados que o utilizador insere nas diversas colunas, constroi uma instrução SELECT numa variável tipo CHAR que posteriormente pode ser preparada e executada.

```
CONSTRUCT {BY NAME variavel_char ON list_de_colunas
        |variavel_char ON lista_de_colunas
        FROM {lista_de_campos|record_ecran[[n]].*}{[,...]}
```

Figura 11.1 Definição genérica da instrução CONSTRUCT

Esta instrução relativamente á cláusula BY NAME, funciona de forma análoga á instrução INPUT, ou seja, se fôr utilizada não é necessário indicar o nome das variáveis internas ás forms e que se pretendem editar.

11.2. INSTRUÇÃO DISPLAY ARRAY

Por vezes uma linha de uma tabela cabe numa linha do écran, e pretende-se introduzir na tabela várias linhas de seguida (por exemplo vários temas para um mesmo livro). Esta tarefa pode ser executada seleccionando várias vezes a opção novo do menu, no entanto seria muito mais prático introduzir todos os temas de seguida, visualizando parte dos anteriores para evitar enganos.

Para tratar este tipo de écrans o 4GL possui as instruções DISPLAY ARRAY e INPUT ARRAY, associadas a novas definições nas forms.

```
DISPLAY ARRAY array_de_records TO array_do_ecran.*
        [ATTRIBUTE(lista_deªtributos)]
        {ON KEY (lista_de_teclas)
                instrução
                ...
                [EXIT DISPLAY]
                 ...
END DISPLAY
```

Figura 11.2 Definição genérica da instrução DISPLAY ARRAY

A instrução DISPLAY ARRAY envia um array de records (do programa) para o écran, por intermédio de um array de linhas no écran.

O array de records pode ser de maior dimensão que o array do écran. Neste caso o utilizador pode provocar o deslocamento do array de records no array do écran (scroll) para cima e para baixo, com as teclas das setas ou se pretender posicionar-se página a página, utilizando as teclas F3 e F4.

Tal como na instrução DISPLAY é possivel testar a pressão de uma tecla especial, associar atributos á visualização e sair da instrução após determinada situação.

Quando é executada esta instrução, ela fica á espera de instruções do utilizador, pelo que, para sair deverá ser pressionada a tecla <ESC> (se esta estiver definida como accept key).

Geralmente, com esta instrução utilizam-se as funções do 4GL SET_COUNT e ARR_CURR. A função SET_COUNT tem sempre de ser executada antes da instrução, e serve para informar a instrução de quantas linhas se encontram preenchidas no array de records. A função ARR_CURR informa qual a linha do array em que o utilizador se encontrava posicionado antes de pressionar a accept key.

Propomos como exercício, a criação de uma função que, quando o utilizador digitar um código de editora inválido, abra uma janela com os códigos válidos e que permita o posicionamento e escolha do código pretendido.

11.3. FORM PARA UTILIZAÇÃO DE DISPLAY ARRAY

Para a utilização das instruções acima definidas tem de se construir uma form que permita estas capacidades. Assim, na secção SCREEN escrevem-se o numero de linhas que se pretende editar de cada vez, cada coluna de cada linha declarada deve ter o mesmo nome de label.

```
SCREEN
{
codigo                  nome
------------------------------------------------------------
[f000 ] [f001                                   ]
[f000 ] [f001                                   ]
[f000 ] [f001                                   ]
[f000 ] [f001                                   ]
[f000 ] [f001                                   ]
[f000 ] [f001                                   ]
[f000 ] [f001                                   ]
[f000 ] [f001                                   ]
}
END
```

Figura 11.3 Secção screen da form de descodificação de editores

Como em qualquer form cada label tem de estar declarada na secção attributes.

```
ATTRIBUTES
        f000 = editores.codigo;
        f001 = editores.nome;
END
```

Figura 11.4 Secção de attributes da form de descodificação de editores

Os campos com a mesma label, têm que corresponder a um SCREEN ARRAY. Um SCREEN ARRAY declara-se na secção instructions e associa o array ás colunas que se pretende editar.

```
INSTRUCTIONS
  DELIMITERS "  "
  SCREEN RECORD sr_editores[8] (editores.codigo,editores.nome)
```

Figura 11.5 Instruction section

Colocaram-se os delimiters a espaços para dar a noção ao utilizador de posicionamento num menu.

11.4. GESTÃO DA FORM COM DISPLAY ARRAY

A zona do programa em que se faz a descodificação deverá ser alterada de forma a que se for introduzido um código inválido, receba um código válido da função que visualiza.

```
AFTER FIELD edicao
CALL descodifica_editor()
IF status = NOTFOUND THEN
        LET pr_livros.edicao = ve_editor()
        CALL descodifica_editor()
END IF
DISPLAY BY NAME pr_livros.edicao, editor
```

Se o valor introduzido fôr inválido (a variável status não tiver 0) é chamada a função ve_editor que devolve um código de editor válido. Como é garantida a validade do código de editor pode-se fazer display do código e nome do editor.

A função descodifica_editor abre uma janela com a form descrita atrás onde serão visualizados os códigos possiveis.

```
OPEN WINDOW w_editores AT 5, 5
        WITH FORM "f_dedit"
        ATTRIBUTE(BORDER)
```

Para evocar a instrução DISPLAY ARRAY, é necessário declarar um record no programa para fazer display.

```
DEFINE
        pa_editores ARRAY[40] OF RECORD LIKE editores.*,
        numero_editores INTEGER,
        linha_corrente  INTEGER
```

O array de records serve para colocar os elementos do cursor. A variável numero_editores serve para verificar o numero de editores obtidos. A variável linha_corrente serve para determinar o elemento do array que foi escolhido.

Para visualizar os dados, tem agora que escrevê-los no array de records.

```
LET numero_editores = 1

DECLARE cr_editores CURSOR FOR
        SELECT *
        FROM editores

OPEN cr_editores

FOREACH cr_editores INTO pa_editores[numero_editores].*
IF  status != 0 THEN
        EXIT FOREACH
END IF

IF  numero_editores >= 40 THEN
        ERROR "Editores a mais"
        EXIT FOREACH
END IF

LET numero_editores = numero_editores + 1
END FOREACH
```

Nesta versão são apenas visualizados os primeiros quarenta editores. Numa versão em que fosse previsto um numero infinito de editores, teria de se declarar o cursor como scroll cursor, e consoante o numero e os pedidos, ir preenchendo o array de records.

```
CALL set_count(numero_editores-1)

DISPLAY ARRAY pa_editores TO sr_editores.*

LET linha_corrente = arr_curr()

CLOSE WINDOW w_editores
```

Para que a instrução DISPLAY ARRAY funcione correctamente tem de ser informada de quantas linhas se deve visualizar, tarefa que é executada pela função set_count.

A instrução DISPLAY ARRAY é evocada informando sobre qual o array de records que vai ser visualizado e em que array de écran.

A função arr_curr() é chamada para que se saiba qual o elemento do arrray que foi escolhido pelo utilizador, para que se possa devolver o código pretendido.

```
RETURN pa_editores[linha_corrente].codigo
```

11.5. INSTRUÇÃO INPUT ARRAY

A instrução INPUT ARRAY, tal como a instrução DISPLAY ARRAY serve para gerir forms com arrays de écran.

Esta instrução permite a introdução de dados através da form para dentro de um array de records.

```
INPUT ARRAY array_de_records [WHITHOUT DEFAULTS]
        FROM array_ecran.* [HELP numero_help]
        [{BEFORE {ROW | INSERT | DELETE
                | FIELD lista_de_campos}[,...]
                |AFTER ROW | INSERT | DELETE
                | FIELD lista_de_campos | INPUT}[,...]
                |ON KEY(lista_teclas)
                        instrução
        [NEXT FIELD nome_campo]
                ...
        [EXIT INPUT]
                ...
END INPUT]
```

Figura 11.6 Definição genérica da instrução INPUT ARRAY

As cláusulas WHITHOUT DEFAULTS, AFTER e BEFORE: INSERT; FIELD e INPUT funcionam da mesma forma que na instrução INPUT, assim como as cláusulas ON KEY, NEXT FIELD, EXIT INPUT.

As cláusulas AFTER ROW e BEFORE ROW permitem executar instruções antes ou depois da edição de uma linha de écran.

Como exercício propõe-se a criação de uma entrada de dados para os temas focados em cada livro.

11.5.1. DEFINIÇÃO DA FORM

Uma form para ser gerida pela instrução INPUT ARRAY é, em tudo identica a uma form para DISPLAY ARRAY. Não nos vamos, portanto alargar na definição desta form.

```
DATABASE livros

SCREEN
{
 codigo |              tema
 --------+-----------------------------------------------------
 [f000 ] | [f001                                              ]
 [f000 ] | [f001                                              ]
 [f000 ] | [f001                                              ]
 [f000 ] | [f001                                              ]
 [f000 ] | [f001                                              ]
 [f000 ] | [f001                                              ]
}
END

TABLES
  tabela_temas

ATTRIBUTES
  f000 = tabela_temas.codigo;
  f001 = tabela_temas.nome,NOENTRY;
END

INSTRUCTIONS
  SCREEN RECORD sr_temas[6] (tabela_temas.codigo, tabela_temas.nome)
```

Figura 11.7 Form de temas

O campo nome foi definido como NOENTRY pois vai servir apenas para descodificar a editora cujo código foi introduzido.

11.5.2. PROGRAMA DE GESTÃO DA FORM DE TEMAS

Ao menu do programa de entrada de dados para livros adicionou-se uma nova opção denominada temas.

```
COMMAND "Temas" "introducao de temas focados neste livro"
IF pr_livros.numero IS NULL THEN
        ERROR "Nao ha nenhum livro para introduzir temas"
ELSE
        CALL temas(pr_livros.numero)
END IF
```

Figura 11.8 Menu da entrada de dados para livros

Faz-se um teste para verificar se existe algum livro corrente, caso contrário não faz sentido a introdução de temas de um livro.

A função temas vai conter os processamentos necessários á correcta introdução dos temas com visualização dos temas já introduzidos.

```
FUNCTION     temas(livro)
DEFINE
        livro LIKE livros.numero,
        pr_idx     INTEGER,
        sr_idx     INTEGER,
        pa_temas ARRAY[15] OF RECORD LIKE tabela_temas.*
```

Figura 11.9 Declaração das variáveis da função temas

O parâmetro livro vem (quando a função é chamada) com o numero do livro de que se vai alterar os temas. As variáveis pr_idx e sr_idx servem como indices de acesso aos arrays respectivamento do programa e do écran. A variável pa_temas é o array de records onde vão ser colocados os dados relativos a cada editor.

É possivel que já existam alguns temas para o livro em causa, convém visualizar os temas existentes para eventual alteração.Se preenchermos o array de records com os editores existentes, estes irão ser colocados na form quando se chamar a instrução INPUT ARRAY.

```
DECLARE cr_temas CURSOR FOR
        SELECT temas.tema,tabela_temas.nome
                FROM temas,tabela_temas
                WHERE temas.livro = livro AND
                        tabela_temas.codigo = temas.tema

LET pr_idx = 1

FOREACH cr_temas INTO pa_temas[pr_idx].*
     LET pr_idx = pr_idx + 1
     IF pr_idx >= 15 THEN
          ERROR "TEMAS A MAIS"
          EXIT FOREACH
     END IF
END FOREACH
```

Figura 11.10 Preenchimento do array de records com os editores já existentes

A forma como é preenchido o record é identica á utilizada quando se falou na instrução DISPLAY ARRAY.

A abertura da janela com a form é feita com a instrução OPEN WINDOW.

```
OPEN WINDOW w_temas AT 5,7
        WITH FORM "f_temas" ATTRIBUTE(BORDER)
```

Figura 11.11 Abertura da janela da form

Tal como no DISPLAY ARRAY é necessário informar a instrução de quantas linhas do array se encontram preenchidas. Utiliza-se também a função set_count().

```
CALL set_count(pr_idx-1)

INPUT ARRAY pa_temas WITHOUT DEFAULTS
        FROM sr_temas.*
```

Figura 11.12 Inicio da instrução INPUT ARRAY

Na instrução diz-se que vamos receber os dados para o array de records pa_temas, do array de écran sr_temas. A cláusula WITHOUT DEFAULTS é utilizada para que apareçam no écran os temas já existentes no array de records.

```
AFTER FIELD codigo
    LET pr_idx = arr_curr()
    LET sr_idx = scr_line()
    LET pa_temas[pr_idx].nome = get_temas(pa_temas[pr_idx].codigo)
    IF pa_temas[pr_idx].nome IS NULL THEN
        ERROR "TEMA INVALIDO"
    ELSE
        DISPLAY pa_temas[pr_idx].nome TO sr_temas[sr_idx].nome
    END IF
```

Figura 11.13 Cláusula AFTER FIELD do INPUT ARRAY

Esta cláusula aparece para que depois de introduzir um código, se verifique a existência desse código na tabela de temas e se descodifique para o campo nome.

A função arr_curr() informa a variável pr_idx de qual o elemento do array de records (do programa) que se está a editar.

A função scr_line() informa qual o elemento do array do écran que nos encontramos a editar (não esqueça que pode ter mais elementos no array de records do que no array do écran pois as instruções fazem scroll automáticamente).

A função get_temas vai á tabela de temas ver se o código existe e devolve o seu nome ou NULL se não existir.

Se o código de tema for válido envia-se para o écran o nome devolvido na linha de écran corrente, com a instrução DISPLAY TO.

Quando o utilizador termina a introdução de editores (é premida a accept key), vai-se limpar todos os temas referentes ao livro corrente e substituir por aqueles que se encontram no array de records (os que já existiam e os que foram introduzidos entretanto).

```
AFTER INPUT
    DELETE FROM temas WHERE temas.livro = livro
    FOR pr_idx = 1 TO arr_count()
        INSERT INTO temas
            VALUES(livro,pa_temas[pr_idx].codigo)
        IF  status != 0 THEN
            ERROR "Erro na insercao"
        END IF
    END FOR
```

Figura 11.14 Actualização da base de dados com os novos editores

Para detectar o fim de introdução utiliza-se a cláusula AFTER INPUT. A função arr_count() é utilizada para nos informar do numero de linhas preenchidas no array do écran.

11.5.2.1. RELATÓRIOS (REPORT) EM INFORMIX-4GLO código de INFORMIX-4gl que permite gerar um relatório (ou uma listagem) é o REPORT.Vamos ver quais as vantagens e a potência do gerador de relatórios (REPORT) do INFORMIX-4GL, para extrair a informação da base de dados. O Report recebe como argumento um conjunto de dados, linha a linha, e devolve a informação formatada.O estudo dos reports de INFORMIX-4GL pode-se dividir em 2 partes: • O código do report propriamente dito• O código que o procede - interacção com o INFORMIX-4GL (REPORT DRIVER).Este colhe a informação, processa-a e depois envia-a linha a linha para o report.A construção tipica de um "REPORT DRIVER" consiste numa instrução SELECT que recolhe a informação da base de dados e associada a um ciclo FOREACH envia para o REPORT linha a linha o resultado do SELECT.11.5.2.1. INTERACÇÃO COM O INFORMIX-4GLVamos considerar como exemplo a geração de um report com a informação da tabela de livros:DATABASE livros GLOBALS "gl_livros.4gl"FUNCTION rd_livros(){ REPORT com a informação da TABELA DE LIVROS}DECLARE cr_liv CURSOR FORSELECT *INTO pr_liv.*FROM livrosORDER BY numeroDISPLAY "Inicio do REPORT; espere um momento "AT 15,1START REPORT r_livFOREACH cr_livOUTPUT TO REPORT r_liv (pr_liv.*)END FOREACHFINISH REPORT r_livDISPLAY "Terminou o report. Encontra-se no ficheiro livros.out "AT 15,1SLEEP 3DISPLAY "" AT 15,1END FUNCTIONFigura 12.1 Exemplo da interacção do REPORT com INFORMIX-4GLProsseguindo com o programa temos as instruções seguintes:**START REPORT** Instrucção que permite inicializar o report.**FOREACH**Executa um ciclo de forma a percorrer todas as linhas resultantes da instrução SELECT, executando a instrução que se segue.**OUTPUT TO REPORT** É a instrução que chama o report, própriamente dito, e que lhe passa linha a linha, as linhas resultantes da instrução SELECT (percorre um cursor), sob a forma de argumentos.**END FOREACH** Termina o ciclo.**FINISH REPORT** Termina o report.De seguida o programa dá ao utilizador uma mensagem de que terminou o report e informa-o do ficheiro onde este se encontra.Com excepção das instruções "FOREACH" e "END FOREACH" que já foram apresentadas anteriormente, iremos aprofundar um pouco os restantes.11.5.2.1. START REPORTEsta instrução é usada em geral antes antes de um ciclo que processa um report com a instrucção "OUTPUT TO REPORT".START REPORT report [TO {ficheiro | PRINTER | PIPE programa}Figura 12.2 Sintaxe da instrução START REPORTSe se usar a cláusula "TO", o INFORMIX-4GL ignora a instrução "REPORT TO" na secção "OUTPUT" do *report.*Se usar a cláusula "TO *ficheiro*", o INFORMIX-4GL escreve o resultado do report nesse ficheiro.Se usar a cláusula "TO PRINTER", o INFORMIX-4GL, envia o resultado do report para a impressora. A impressora por defeito é a escolhida pelo programa "lp", (no S.O.UNIX), esta pode ser alterada reinicializando a variável de ambiente DBPRINT.11.5.2.1. OUTPUT TO REPORTEsta instrução serve para passar ao report um linha de dados.OUTPUT TO REPORT report(linha_dados)Figura 12.3 Sintaxe da instrução OUTPUT TO REPORTEm geral a instrução "OUTPUT TO REPORT" é utilizada dentro de um ciclo que passa dados para o *report.*A *linha_dados* pode ser uma ou mais colunas e/ou uma ou mais expressões, separadas por virgúlas. O número de elementos na *linha_dados* tem que estar de acordo com o número de argumentos da função REPORT chamada.11.5.2.1. FINISH REPORTEsta instrução indica ao INFORMIX-4GL que o report terminou.FINISH REPORT reportFigura 12.4 Sintaxe da instrução FINISH REPORTÉ necessário usar esta instrução para que o INFORMIX-4GL saiba que terminou o processamento para o *report.*11.5.2.1. ESTRUTURA DO REPORT Vamos comecar por considerar o REPORT do exemplo anterior:

```
DATABASE  bibliografia

REPORT  r_uv (rr)
{
r_uv  é o report que escreve a informacão da tabela de livros

 este report é enviado para o ficheiro livros.out
}

DEFINE
        rr RECORD LIKE  livros.*

OUTPUT
     REPORT TO "livros.out"
     LEFT MARGIN 0
     TOP MARGIN 0
     BOTTOM MARGIN 0
     PAGE LENGH 8

FORMAT
     ON EVERY ROW
          PRINT rr.numero,1 SPACE, rr.nome CLIPPED
          PRINT rr.autor, 1 SPACE, rr.editor

END report
```

Figura 12.5 Exemplo de um REPORT

A primeira instrução do report é a declaração de base de dados. É necessário definir a base de dados para poder entender a cláusula "LIKE" de instrução "DEFINE".

Um report inicia-se com a instrução "REPORT" que inclui entre parentesis a lista de argumentos. Esta lista tem de corresponder exactamente á lista de argumentos enviada pela instrução "OUTPUT TO REPORT"; o nome das variáveis não tem de ser exactamente o mesmo,que é enviado pela instrução "OUTPUT TO REPORT", no entanto se o tipo das variáveis não for o mesmo, tem de poder ser directamente convertivel. (Por exemplo podemos passar uma frase entre ("") a uma variável do tipo "CHAR", mas não podemos passar uma frase com caracteres alfanuméricos a uma variável do tipo "INTEGER".

Tal como em "FUNCTION" ou "MAIN", a instrução que se segue é "DEFINE". Neste caso usamos a cláusula "LIKE" para definir as variáveis . O report define "rr" como "RECORD" da tabela de livros. As variáveis do report são sempre do mesmo tipo das variáveis seleccionadas, daí que se for necessário efectuar alguma alteracão na tabela em qestão, (Ex: mudar a dimensão de uma variável tipo "CHAR"), não é necessário alterar o report, apenas tem que ser compilado.

Continuando no exemplo do REPORT, segue-se a secção de "OUTPUT". Nesta secção que é opcional define-se para onde vai ser enviado o REPORT, e qual o formato da folha, isto é, a margem de cima, margem de baixo, margem esquerda e o número de linhas na página.

A secção "FORMAT" é a mais importante. Esta secção pode ter as subsecções "PAGE HEADER" para o cabeçalho, "PAGE TRAILLER" para o rodapé, e "ON EVERY ROW" para todas as linhas. No nosso

exemplo apenas temos a ultima, onde se vai definir como vão ser escritas as linhas, usando as instruções seguintes:

PRINT - escreve os elementos do "RECORD" rr.

1 SPACE - para deixar um espaco em branco.

CLIPPED - truncar os espacos em branco no fim de string.

Finalmente o report termina com a instrução "END REPORT".

Depois de analizar este exemplo, vamos ver com mais profundidade as diferentes seccões.

O report é composto por secções que são constituidas por blocos e/ou por instruções. As secções são as seguintes e devem de ser escritas segundo a ordem apresentada:

```
REPORT report (ListaArgumentos)
    [DEFINE secção]
    [OUTPUT secção]
    [ORDER BY secção]
    FORMAT secção
END REPORT
```

Figura 12.6 Secções que constituem um REPORT

As secções "DEFINE","OUTPUT" e "ORDER BY" são opcionais; a secção "FORMAT" é obrigatória.

11.6. SECÇÃO DEFINE

A rotina REPORT necessita da secção "DEFINE" quando se lhe passam argumentos ou quando é necessário o uso de variáveis locais no report. Tem de haver lista de argumentos nas seguintes condições:

- Quando há secção "ORDER BY" no report. Neste caso é necessário passar todos os valores para cada linha do report.
- Quando se usam funções de agregação, que dependem das linhas do report, em qualquer parte do report, é necessário passa-las na lista de argumentos.
- Quando se usa o bloco de controlo "EVERY ROW" na secção "FORMAT", neste caso é obrigatório passar os valores de cada linha do report.
- Quando se usa o bloco de controlo "AFTER GROUP OF", tem de se passar pelos menos o parâmetro que é referido.

A secção "DEFINE" obedece às regras já vistas no capítulo sobre variáveis.

11.6.1. SECÇÃO OUTPUT

A rotina REPORT pode ou não conter esta secção. A secção "OUTPUT" controla a dimensão das margens e o comprimento da página. Permite definir para onde vai o resultado do report, isto é, écran, ficheiro, impressora ou para outro processo através de uma pipe.

A secção "OUTPUT" é constituida pela palavra "OUTPUT" seguida de uma ou mais instruções:

```
OUTPUT
    [REPORT TO instrução]
    [LEFT MARGIN valor]
    [RIGHT MARGIN valor]
    [TOP MARGIN valor]
    [BOTTOM MARGIN valor]
    [PAGE LENGHT valor]
```

Figura 12.7 Formato da secção OUTPUT

```
REPORT TO {"ficheiro" | PIPE programa | PRINTER}
```

Se a instrução "START REPORT" tiver a cláusula "TO" indicando para onde enviar o resultado do report, o INFORMIX-4GL ignora a instrução "REPORT TO" na secção "OUTPUT".

Se o *ficheiro* for uma variável, é necessário passá-la como argumento da rotina "REPORT".

A instrução "REPORT TO PRINTER" envia o resultado do report para o programa definido pela variável DBPRINT. Se esta variável nào estiver definida então o INFORMIX-4GL envia o resultado para o programa "lp".

Se se quizer enviar o resultado do report para outra impressora que não seja a do sistema, pode-se usar a instrução "REPORT TO ficheiro", que escreve num ficheiro podendo depois ser imprimido. Pode também usar-se a instrução "REPORT TO PIPE" para enviar directamente o resultado para um programa que manda para a impressora correcta.

```
OUTPUT
    REPORT TO PIPE "more"
```

Figura 12.7 Exemplo da instrução REPORT TO da secção OUTPUT

Este exemplo envia o resultado do report para o utilitário "more". Se se omitir a instrução "REPORT TO" na rotina REPORT, e se a instrução "START REPORT" não tiver cláusula "TO" o resultado do report vai para o écran.

11.6.2. SECÇÃO ORDER BY

A secção "ORDER BY" é opcional na rotina REPORT. Especifica quais as variáveis porque queremos ver ordenado o nosso relatório, e a ordem com que o INFORMIX-4GL processará os blocos de controlo na secção "FORMAT". É necessário o uso desta secção quando:

- se usar blocos de controlo
- se enviam as linhas sem qualquer ordenação
- já foram ordenadas as linhas enviadas, mas, no entanto pretende-se especificar a ordem exacta com que serão processados os blocos de controlo. Neste caso usa-se a cláusula "EXTERNAL" para que as linhas não sejam novamente ordenadas.

A secção "ORDER BY" tem o formato seguinte:

```
ORDER [EXTERNAL] BY lista_colunas
```

Figura 12.8 Formato da secção ORDER BY

11.6.3. A SECÇÃO FORMAT

Esta secção é obrigatória. É a secção "FORMAT" que vai determinar a aparência do nosso relatório. Usa os dados que são passados à rotina "REPORT" através da sua lista de argumentos ou dados fornecidos por variáveis globais para cada linha do relatório.

A secção "FORMAT" tem inicio com a palavra "FORMAT" e termina quando a rotina "REPORT" termina, isto é, com "END REPORT".

Existem dois tipo de secção "FORMAT". O primeiro, e o mais simples, contém unicamente a subsecção "EVERY ROW", e não pode conter qualquer outra subsecção ou bloco de controlo da secção "FORMAT".

```
FORMAT
    EVERY ROW
END REPORT
```

Figura 12.8 Exemplo da mais simples secção FORMAT

O outro tipo, mais complexo, da secção "FORMAT" pode ser constituído da seguinte forma:

```
FORMAT
    [PAGE HEADER bloco_de_controlo]
    [PAGE TRAILER bloco_de_controlo]
    [FIRST PAGE HEADER bloco_de_controlo]
    [ON EVERY ROW bloco_de_controlo]
    [ON LAST ROW bloco_de_controlo]
    [BEFORE GROUP OF n blocos_de_controlo]
    [AFTER GROUP OF n blocos_controlo]
END REPORT
```

Figura 12.9 Exemplo da secção FORMAT

A ordem de escrita dos blocos de controlo é arbitrária. O número de blocos "BEFORE ..." ou "AFTER ..." que se podem utilizar é igual ao número de colunas exitentes na secção "ORDER BY". Vejamos as subsecções que constituem a secção "FORMAT":

EVERY ROW

Esta instrução usa-se quando se pretende gerar um relatório rápidamente. O relatório consiste unicamente nos dados que são passados como argumentos à rotina "REPORT".

Ao usar esta instrução não pode usar qualquer bloco de controlo.

```
REPORT minimo(x_livros)
DEFINE x_livros RECORD LIKE livros.*
FORMAT
  EVERY ROW
END REPORT
```

Figura 12.10 Exemplo de um REPORT muito simples, usando a instrução EVERY ROW

Os blocos de controlo:

Cada bloco de controlo é opcional mas, se não se usar "EVERY ROW", é necessário incluir pelos menos um bloco de controlo na rotina "REPORT". Cada bloco de controlo inclui pelo menos uma instrução.

Quando se usa "BEFORE GROUP OF", "AFTER GROUP OF" e "ON EVERY ROW" numa única rotina "REPORT", o INFORMIX-4GL processa em primeiro lugar todos os blocos "BEFORE GROUP OF", depois o bloco "ON EVERY ROW", e finalmente todos os blocos "AFTER GROUP OF". A ordem com que o INFORMIX-4GL processa os blocos "BEFORE GROUP OF" e os blocos "AFTER GROUP OF" depende da ordem definida para as variáveis na secção "ORDER BY".

Suponhamos que temos:

```
ORDER BY a, b, c
```

Figura 12.11 Um exemplo da instrução ORDER BY

Então na secção "FORMAT" teriamos:

```
BEFORE GROUP OF a
    BEFORE GROUP OF b
```

```
        BEFORE GROUP OF c
            ON EVERY ROW
        AFTER GROUP OF c
    AFTER GROUP OF b
AFTER GROUP OF a
```

Figura 12.13 Um exemplo da ordem de execução dos blocos de controlo

AFTER GROUP OF

Este bloco de controlo define as acções (ou instruções) que se devem executar depois de ser processado um determinado grupo de linhas. Um grupo de linhas define-se como um conjunto de linhas que têm o mesmo valor para uma determinada coluna que figure na secção "ORDER BY" da rotina "REPORT" ou na cláusula "ORDER BY" da instrução "SELECT".

```
AFTER GROUP OF variavel
    instruções
```

Figura 12.14 Formato do bloco de controlo AFTER GROUP OF

O INFORMIX-4GL processa as instruções do bloco "AFTER ..." cada vez que a variável mudar valor.

Pode-se ter tantos blocos "AFTER ..." quantas as variáveis da secção ou da cláusula "ORDER BY".

A ordem com que são escritos estes blocos não tem qualquer importância. quando o INFORMIX-4GL terminar o processamento do relatório executará todos os blocos "AFTER ..." antes de executar o bloco "ON LAST ROW".

As funções agregadas de grupo dos "REPORTs" só podem ser usadas nos blocos de controlo "AFTER ...".

BEFORE GROUP OF

Este bloco de controlo define as acções (ou instruções) que se devem executar antes de ser processado um determinado grupo de linhas.

```
BEFORE GROUP OF variavel
    instruções
```

Figura 12.15 Formato do bloco de controlo BEFORE GROUP OF

O INFORMIX-4GL processa as instruções do bloco "BEFORE ..." cada vez que a variável mudar valor.

Pode-se ter tantos blocos "BEFORE ..." quantas as variáveis da secção ou da cláusula "ORDER BY".

Pode-se ter um ou mais blocos "BEFORE ..." para cada variável da secção ou cláusula "ORDER BY".

Estes blocos podem ser usados para limpar variáveis de acumulação de totais ou de controlo.

O INFORMIX-4GL ao gerar um relatório começa por executar as instrucções destes blocos e só depois executa o bloco "ON EVERY ROW".

Pode se usar os blocos "BEFORE ..." para se iniciar uma nova página para cada grupo de informação com a instrução "SKIP TO TOP OF PAGE"

FIRST PAGE HEADER

Este bloco de controlo inclui as acções que se devem executar para definir o cabeçalho da primeira página do relatório.

```
FIRST PAGE HEADER
    instruções
```

Figura 12.16 Formato do bloco FIRST PAGE HEADER

A instrução "TOP MARGIN" da secção "OUTPUT" influencia a linha onde vai ser escrito este cabeçalho.

Não se pode usar a instrução "SKIP TO TOP OF PAGE" neste bloco.

Se se usar a instrução "IF THEN ELSE" neste bloco, então o número de instruções "PRINT" tem de ser igual a seguir a "THEN" e a seguir a "ELSE".

Não se pode usar a instrução "PRINT ficheiro", para poder despejar o conteúdo deste de um ficheiro de texto no "FIRST PAGE HEADER".

Pode-se usar este bloco de controlo para inicializar variáveis, pois o INFORMIX-4GL executa-o antes de qualquer outro.

ON EVERY ROW

Este bloco define as instruções que o INFORMIX-4GL passa para cada linha de dados como argumento à rotina "REPORT".

```
ON EVERY ROW
    instruções
```

Figura 12.17 Formato do bloco ON EVERY ROW

Se houver blocos "BEFORE ..." na rotina "REPORT", estes serão processados antes do bloco "ON EVERY ROW".

Se houver blocos "AFTER ..." na rotina "REPORT", estes serão processado depois do bloco "ON EVERY ROW".

ON LAST ROW

Este bloco de controlo define as instruções que se devem executar depois de processada a ultima linha passada à rotina "REPORT".

```
ON LAST ROW
    instruções
```

Figura 12.18 Formato do bloco ON LAST ROW

O INFORMIX-4GL executa este bloco depois dos blocos "ON EVERY ROW" e "AFTER GROUP OF".

Pode-se usar o bloco "ON LAST ROW" para apresentar totalizações.

PAGE HEADER

Este bloco especifica a informação que vai aparecer no cabeçalho de cada página do relatório.

```
PAGE HEADER
    instruções
```

Figura 12.19 Formato do bloco PAGE HEADER

A instrução "TOP MARGIN" da secção "OUTPUT" influencia a linha em que vai ser escrita este cabeçalho.

Não se pode usar a instrução "SKIP TO TOP OF PAGE" neste bloco.

O número de linhas do "PAGE HEADER" tem de ser o mesmo para todas as páginas. Tem de obedecer ás regras seguintes:

- não se pode usar a instrução "SKIP valor_inteiro LINES" dentro de um ciclo dentro do bloco "PAGE HEADER"
- não se pode usar a instrução "NEED"
- se se usar a instrução "IF THEN ELSE" neste bloco, então o número de instruções "PRINT" tem de ser igual a seguir a "THEN" e a seguir a "ELSE"
- se usar uma das instruções "CASE", "FOR" ou "WHILE" que contenha a instrução "PRINT", dentro do bloco "PAGE HEADER", deve ser terminada com ";".
- não se pode usar a instrução "PRINT ficheiro", para poder despejar deste de um ficheiro de texto no "PAGE HEADER".

Pode usar neste bloco a expressão "PAGENO" para paginar o relatório automáticamente, no cabeçalho.

PAGE TRAILER

Este bloco inclui as instruções que se devem executar para definir o rodapé de cada página.

```
PAGE TRAILER
    instruções
```

Figura 12.20 Formato do bloco PAGE TRAILER

Não se pode incluir a instrução "SKIP TO TOP OF PAGE" neste bloco.

A instrução "BOTTOM MARGIN", da secção "OUTPUT", define a linha em que termina este bloco.

O número de linhas do "PAGE TRAILER" tem de ser o mesmo para todas as páginas. Tem de obedecer ás regras seguintes:

- não se pode usar a instrução "SKIP valor_inteiro LINES" dentro de um ciclo dentro do bloco "PAGE TRAILER"
- não se pode usar a instrução "NEED"
- se se usar a instrução "IF THEN ELSE" neste bloco, então o número de instruções "PRINT" tem de ser igual a seguir a "THEN" e a seguir a "ELSE".
- se usar uma das instruções "CASE", "FOR" ou "WHILE" que contenha a instrução "PRINT", dentro do bloco "PAGE TRAILER", deve ser terminada com ";".
- não se pode usar a instrução "PRINT ficheiro", para poder despejar deste de um ficheiro de texto no "PAGE TRAILER".
- Pode usar neste bloco a expressão "PAGENO" para paginar o relatório automáticamente, no rodapé.

Instrucções

Os blocos de controlo da "FORMAT SECTION" determinam quando uma ou várias acções ocorrem, enquanto que as INSTRUÇÕES determinam qual a acção a efectuar. Pode considerar-se dois tipos de instruções:

Simples: constituida por uma única instrução.

Compostas: constituidas por um grupo de instruções encabeçadas pela palavra BEGIN e finalizadas pela palavra END.

FOR A instrução FOR define um ciclo (loop), executando consecutivamente uma instrução simples ou composta, incrementando o indice e controlando a condição de saída, e quando esta se verifica passa o controlo do programa para a instrução seguinte ao fim do ciclo.

```
FOR i=1 TO 10 STEP 2
        LET a = a * a + 2
        LET b = a * 2
END FOR
```

Figura 12.21 Exemplo da instrução FOR

*NOTA:*Não se pode usar incrementos negativos.

IF THEN ELSE Esta instrução executa um teste a uma condição e processa, em alternativa, 2 instruções (simples ou compostas) conforme a condição se verifica ou não.

```
IF paid_date IS NULL THEN
        LET pg1 = "fal"
        LET pgc1 = "ta   "
ELSE
        LET pg1 = paid_date - ship_date
        LET pgc1 = " dias"
END IF
```

Figura 12.22 Exemplo da instrução IF-THEN-ELSE

*NOTA:*Pode-se usar até 128 IF-THEN-ELSE embebidos.

LET A instrução LET assigna um valor a uma variável.

```
LET pg1 = paid_date - ship_date
LET pgc1 = " dias"
```

Figura 12.23 Exemplo da instrução LET

NEED Esta instrução faz com que a próxima linha seja impressa na página seguinte, se não estiver disponível na página corrente, o número de linhas especificado na instrução NEED

```
BEFORE GROUP OF num_livro
    SKIP 1 LINE
    NEED 2 LINES
    PRINT COLUMN 01, "LIVRO:",
        COLUMN 10, num_livro USING "<<<<",
        COLUMN 16, nome CLIPPED
```

Figura 12.24 Exemplo da instrução NEED

PAUSE Esta instrução faz com que um relatório destinado a um écran possa ser fixado para ser visualizado. Pressione RETURN para avançar pelo relatório.

```
PAGE TRAILER
    PAUSE "Pressione RETURN para continuar"
```

Figura 12.25 Exemplo da instrução PAUSE

*NOTA:*Esta instrução não tem nenhum efeito se o relatório for dirigido para uma impressora, um ficheiro ou uma pipe.

PRINT Esta instrução imprime informação no écran, numa impressora ou num ficheiro, conforme for especificado na OUTPUT SECTION.

```
PRINT COLUMN 10, "Encomenda:",
      COLUMN 21, ord USING "<<<<",
      COLUMN 29, "Total:",
      COLUMN 36, tot1 USING "$$$$$####&.&&",
      COLUMN 53, "Pagamento em:",
      COLUMN 67, pg1, pgc1 CLIPPED
```

Figura 12.26 Exemplo da instrução PRINT

NOTAS:Cada instrução PRINT imprime o seu output numa linha. Pode-se, com um único PRINT escrever várias linhas, separando-as com ";".

Conforme o tipo de dados, no PRINT, as colunas ocupam espaços previamente fixados. Para controlar o comprimento dos campos no relatório utilize as palavras CLIPPED e USING.

SKIP Esta instrução salta o número de linhas em branco especificadas a seguir a SKIP.

SKIP TO TOP OF PAGE Esta instrução faz com que a próxima linha seja escrita no início da próxima página.

WHILE A instrução WHILE define um ciclo, executando consecutivamente uma instrução simples ou composta, enquanto for verdadeira a condição que segue a palavra WHILE. Quando esta se verifica passa, o controlo do programa para a instrução seguinte ao fim do ciclo.

```
WHILE PAGENO = 1
      LET a = a * a + 2
      LET b = a * 2
END WHILE
```

Figura 12.27 Exemplo da instrução WHILE

FUNÇÕES AGREGADAS As funções agregadas possibilitam a execução de operações sobre colunas das linhas de todo resultado do SELECT, podendo-se ainda escolher entre estas as que satisfaçam determinado critério.

Existe ainda a hipótese de definir funções agregadas só para grupos de quebra, desde que se utilize a palavra GROUP.

A estrutura da instrução para definir uma função agregada é a seguinte:

```
[GROUP]
<função agregada>
OF <coluna ou expressão aritmética envolvendo coluna>
[WHERE <expressão lógica>]
```

Figura 12.28 Formato da instrução para definir funções agregadas

*NOTAS:*GROUP e WHERE são opcionais.

GROUP só se pode usar num bloco AFTER.

Estão disponiveis as seguintes funções:

COUNT Conta o número total de linhas do resultado ou do grupo e que satisfaçam a cláusula WHERE, se existir.

PERCENT Executa um função de contagem semelhante a COUNT mas dá o resultado em percentagem do número total de linhas seleccionadas a considerar.

TOTAL Acumula o valor da coluna especificada para todas as linhas do resultado ou do grupo e que satisfaçam a cláusula WHERE, se existir.

AVERAGE ou AVG Calcula o valor médio da coluna especificada para todas as linhas do resultado ou do grupo e que satisfaçam a cláusula WHERE, se existir.

MIN Calcula o valor mínimo da coluna especificada para todas as linhas do resultado ou do grupo e que satisfaçam a cláusula WHERE, se existir.

MAX Calcula o valor máximo da coluna especificada para todas as linhas do resultado ou do grupo e que satisfaçam a cláusula WHERE, se existir.

```
AFTER GROUP OF cust
    LET totcli = GROUP TOTAL OF total_price
    LET totger = totger + totcli
    PRINT COLUMN 01, "TOTAL CLIENTE:",
        COLUMN 17, totcli USING "$$$$$####&.&&"
    PRINT GROUP TOTAL OF total_price
        WHERE total_price > 500 USING "$$$$$####&.&&"
```

Figura 12.29 Exemplos de utilização de funç`oes agregadas

CLIPPED Esta instrução suprime nos campos alfanuméricos, todos os espaços à direita do último caracter diferente de espaço.

```
BEFORE GROUP OF cust
    SKIP 1 LINE
    NEED 2 LINES
    PRINT COLUMN 01, "CLIENTE:",
        COLUMN 10, cust USING "<<<<",
        COLUMN 16, company CLIPPED
```

Figura 12.30 Exemplo da instrução CLIPPED

COLUMN Esta instrução permite imprimir, o campo que se segue, a partir da coluna especificada.

```
BEFORE GROUP OF cust
    SKIP 1 LINE
    NEED 2 LINES
    PRINT COLUMN 01, "CLIENTE:",
        COLUMN 10, cust USING "<<<<",
        COLUMN 16, company CLIPPED
```

Figura 12.31 Exemplo da instrução COLUMN

LINENO Esta expressão contém o número da linha que o REPORT está a imprimir actualmente.

```
IF LINENO = 24 THEN
    SKIP TO TOP OF PAGE
```

Figura 12.32 Exemplo da instrução LINENO

NOTA: Não utilize LINENO em blocos page header ou trailer porque não funcionará correctamente.

PAGENO Esta expressão contém o número da página que o REPORT está a imprimir actualmente.

```
PAGE HEADER
PRINT "RESUMO DAS ENCOMENDAS POR CLIENTE DE  ",
inic, "  A  ", fim,
COLUMN 70, "pagina ", PAGENO USING "<<<"
```

Figura 12.33 Exemplo da instrução PAGENO

TIME Esta instrução devolve uma frase com o valor da hora corrente que tem o formato seguinte "hh:mm:ss", isto é, horas, minutos e segundos.

USING Esta expressão permite definir o formato de impressão dos campos tipo numérico e data, a que está associada.

```
PRINT  "RESUMO DAS ENCOMENDAS POR CLIENTE DE  ",
    inic, "  A  ", fim,
    COLUMN 70, "pagina ", PAGENO USING "<<<"

PRINT COLUMN 10, "Encomenda:",
    COLUMN 21, ord USING "<<<<",
    COLUMN 29, "Total:",
    COLUMN 36, tot1 USING "$$$$$####&.&&",
    COLUMN 53, "Pagamento em:",
    COLUMN 67, pg1, pgc1 CLIPPED
```

Figura 12.34 Exemplos da instrução USING

NOTAS: O conjunto de caracteres que definem o formato têm de estar entre " ".

USING pode ser usado nas instruções PRINT e LET.

Se, por acaso um campo tiver comprimento superior ao espaço definido para o imprimir, este será preenchido a asteriscos.

A seguir apresentam-se e descrevem-se os principais caracteres para definição do formato:

Numéricos

"*****" Imprime asteriscos em vez de zeros não significativos.

"**&**" Mantém os zeros não significativos.

"**#**" Imprime espaços em vez de zeros não significativos.

"**<**" Imprime um número encostado à esquerda.

"**,**" Este caracter é impresso na posição em que for escrito. Se não existirem números à sua esquerda não é impresso.

"**.**" Este caracter é impresso na posição em que for escrito. Só pode existir um ponto por campo.

"**-**" Este caracter é impresso quando o número for negativo.

"**+**" Este caracter imprime "+" quando o número for positivo e "-" quando for negativo.

"**$**" Este caracter é impresso antes do número.

Datas

"**dd**" Imprime o dia do mês como um número (01 a 31).

"**ddd**" Imprime o dia da semana usando uma abreviatura com 3 letras.

"**mm**" Imprime o mês como um número (01 a 12).

"**mmm**" Imprime o mês usando uma abreviatura com 3 letras.

"**yy**" Imprime o ano como um número de 2 digitos (01 a 99). "yyyy" Imprime o ano como um número de 4 digitos (0001 a 9999).

12. AMBIENTE INFORMIX-4GL

Há três variáveis de ambiente que têm de estar correctamente definidas para que o INFORMIX-4GL possa funcionar. Duas destas variáveis (PATH, TERM) são variáveis gerais do UNIX, a outra INFORMIXDIR, é específica dos produtos da Informix Inc. Há um conjunto de variáveis de ambiente específicas do INFORMIX que lhe afectam o comportamento, cujo nome começa sempre pelas letras "DB", algumas variáveis do S.O. UNIX afectam também o comportamento do INFORMIX.

INFORMIXDIR

Num primeiro passo para a utilização do INFORMIX-SQL, é necessário saber onde foram instalados os programas. Todos os programas distribuidos com o INFORMIX-SQL estão armazenados num conjunto de subdirectórios abaixo de um directório principal (Ex: "/usr/informix" nas máquinas UNIX, informix nas máquinas DOS). Se o produto foi instalado noutro directório (por ex: /u0/informix), então é necessário dizê-lo ao INFORMIX-SQL, definindo a variável INFORMIXDIR.

Se estiver a usar um sistema UNIX e Bourne shell (sh) ou Korn shell (ksh) então a sintaxe é:

```
INFORMIXDIR=/u0/informix
export INFORMIXDIR
```

Usando o C shell (csh) a sintaxe é:

```
setenv INFORMIXDIR /u0/informix
```

No caso de se estar num sistema DOS (MS-DOS ou PC-DOS) então a sintaxe é:

```
set INFORMIXDIR=C:\u0\informix
```

Para que não seja necessário proceder a esta operação em cada sessão de trabalho pode modificar-se o ficheiro .profile (sh ou ksh) ou .login (csh) ou AUTOEXEC.BAT (DOS). Esta técnica aplica-se a todas as variáveis de ambiente.

NOTA: Em algumas máquinas se o INFORMIX-SQL for instalado no seu local por defeito (/usr/informix em UNIX), então não é necessário definir a variável INFORMIXDIR. No entanto aconselha-se a que se defina sempre esta variável.

PATH

A segunda variável a definir serve para ter a certeza de que o interpretador dos comandos ou o shell saibam onde estão os comandos do INFORMIX, esta é uma variável do S.O. UNIX, para mais pormenores deve consultar-se a documentação do Sistema Operativo. Todos os comandos estão

instalados em "$INFORMIXDIR/bin" (isto é, "/usr/informix/bin"). Então a forma de inicializar a variável PATH é, por exemplo:

```
PATH=$INFORMIXDIR/bin:$PATH
export PATH
```

TERM e TERMCAP

Tal como a maioria dos produtos para o sistema UNIX, o INFORMIX-SQL trabalha com quase todo o tipo de terminais. Os produtos Informix usam o ficheiro de descrição de terminais **termcap** normal do sistema UNIX. No caso de o ficheiro "/etc/termcap" não existir ou de se pretender usar características do terminal não definidas na descrição standard do UNIX, é necessário inicializar convenientemente a variável de ambiente TERMCAP, do seguinte modo:

```
TERMCAP=$INFORMIXDIR/etc/termcap
export TERMCAP
```

pois vem com o INFORMIX-SQL um ficheiro onde estão definidos a maioria dos terminais. Este encontra-se no directório "etc" abaixo do directório onde foi instalado o INFORMIX-SQL.

DBDATE

Consideremos o seguinte formato de data "06/09/88", para os americanos lê-se 9 de Junho de 1988; para os europeus lê-se 6 de Setembro de 1988. Pode-se definir como é se quer que o INFORMIX-SQL interprete esta data, para isso é necessário inicializar correctamente a variável DBDATE. O formato por defeito é:

```
DBDATE=mdy2/
export DBDATE
```

Esta variável deve conter uma sequência das letras "m", "d" e "y" por qualquer ordem para indicar qual a ordem em que aparecem respectivamente o mês, o dia e o ano, um caracter separador dos dias meses e anos (usalmente "/" ou "-"), e o número de digitos do ano (2 ou 4).

O exemplo acima indica que os elementos introduzidos numa variável tipo data estão com a ordem mês, dia, ano, que se tem 2 digitos para o ano e que o caracter separador é "/".

Pode-se inicializar a variável DBDATE de qualquer uma das formas apresentadas a seguir:

DBDATE	Ex: para escrever 1 de Fevereiro de 2003
mdy2/	02/01/03
dmy2/	01/02/03
dmy4/	01/02/2003
mdy4/	02/01/2003
y4md-	2003-02-01
y2md/	03/02/01

DBMONEY

O INFORMIX-SQL é um produto americano, como tal os valores monetários são representados como dolares "$", e o indicador de casa decimal por defeito é ".". Para usar outro formato nas colunas MONEY é necessário incializar a variável DBMONEY, especificando o separador das partes inteira e decimal, e sufixo ou prefixo indicador de unidade. Por exemplo na alemanha onde se usa "Deutsch Marks", fariamos:

```
DBMONEY=,DM
export DBMONEY
```

assim teriamos para escrever doze mil Deutsch Marks apareceria "123.000,00DM".

DBPATH

Esta variável de ambiente serve para indicar ao INFORMIX-SQL onde vai procurar bases de dados, forms (écrans de entrada de dados), reports (relatórios), ou scripts de RDSQL.

DBPRINT

Quando se envia um report para uma impressora, este é enviado para um programa que se encarrega das tarefas de gestão de impressões (por ex: lpstat no UNIX), o nome do tal programa pode ser indicado na variável DBPRINT. Em qualquer altura se pode mudar a impressora de destino por defeito, mudando a variável DPRINT.

Se quizermos por exemplo enviar para a impressora "beta" com as opções "-onb", fazemos:

```
DPRINT="lp -s -dbeta -onb"
export DBPRINT
```

DBEDIT

Esta variável de ambiente serve para definir o editor de texto que é chamado por defeito. Por exemplo:

```
DBEDIT=vi
export DBEDIT
```

DBTEMP

Por defeito o UNIX cria os ficheiros temporários no directório "/tmp". Para indicar outro directório deve colocar-se o nome deste na variável DBTEMP.

DBDELIMETER

Esta variável serve para definir os delimitadores, dos campos usados pelos comandos LOAD e UNLOAD.

No caso desta variável não estar definida o INFORMIX assume o caracter "|".

SQLEXEC

Esta variável indica qual o programa de acesso à base de dados (database engine ou database mechanism) usado. Esta variável só pode ser usada quando se utilize o INFORMIX-TURBO ou ON-LINE.

Se na mesma máquina coexistirem os dois métodos de acesso à base de dados deve inicializar-se esta variável com:

```
SQLEXEC=sqlexec
export SQLEXEC
```

para a versão standard e

```
SQLEXEC=sqlturbo
export SQLEXEC
```

13. INTERACTIVE DEBUGGER

Uma das tarefas mais dificeis e de que os programadores menos apreciam é a fase de testes de um programa.

Quanto existem erros internos ao programa (geralmente conhecidos como bugs), geralmente recorre-se a técnicas de tracejamento ás variáveis, utilizando instruções básicas (DISPLAY, PROMPT etc.). Depois de testado o programa é necessário retirar todo o debug (instruções colocadas para descobrir bugs) do programa, para gerar a versão utilizador.

Estas técnicas são geralmente muito trabalhosas e multiplicam o tempo de desenvolvimento e manutenção de um programa.

Para permitir a fácil manutenção e descoberta de erros, foram criados programas de testes chamados **debugger(s)**

Estes programas, geralmente tomam conta da execução do nosso programa, e permitem o controlo completo do programa, durante a execução.

No 4GL foi criado o programa "INTERACTIVE DEBUGGER", que só funciona se possuir a versão RDS do 4GL.

13.1. COMO EXECUTAR O I.D.

O I.D. executa-se a partir do menu do r4gl, escolhendo a opção Debug do menu de módulos, ou executando o comando fgldb pelo sistema operativo, enviando como argumento o nome do programa de que se pretende fazer debug.

13.2. ECRANS DO I.D.

O I.D. tem dois écrans:

- O écran da aplicação. Neste écran é visualizado o programa tal como se apresenta quando é executado normalmente.
- O écran de DEBUG. 'E constituido por duas janelas: a janela do source; a janela de comandos.Na janela do source é visualizado o source do programa utilizado. Na janela de comandos, comanda-se toda a execução e parâmetrização do debugger.

14. CONVENÇÕES UTILIZADAS NO MANUAL

Neste manual são utilizadas convenções para a definição da sintaxe dos comandos. Como nem todos os utilizadores poderiam conhecer o standard usualmente conhecido como BNF (Bachus Normal Form) que são de uso comum em textos de informática e foram introduzidos nos anos 60 na especificação da linguagem ALGOL, adopta-se as convenções utilizadas pela Informix. Assim:

LETRAS MAIUSCULAS

São palavras reservadas da linguagem que devem ser digitadas da forma como estão definidas. Pode também utilizar letras minusculas.

letras minusculas

Representam variáveis, que podem ser identificadores ou expressões.

[]

O que encontrar entre estes dois caracteres é opcional na instrução em causa.

{}

Significa que tem escolher uma das opções propostas.

|

Significa "ou", isto é escolha entre várias opções.

...

Repetição opcional da cláusula anterior.

15. ALGUMAS CONVENÇÕES PROPOSTAS

Um programa deve ser legivel, tanto pelos programadores que os escrevem, como pelos que lhe vão dar assistencia. A utilização das seguintes convenções, propostas por Mark Sobbel no seu livro podem facilitar muito a legibilidade dos programas. proponho que adoptem as seguintes convenções:

Nomes de identificadores

Prefixo	Tipo do identificador
cr_	cursor
f_	form
i_	indice de uma tabela
pa_	Array dentro de um programa
pr_	Record de um programa
rr_	Record de um report
r_	Report
rd_	Report driver - Função de controlo de um report.
sr_	Record de um écran (screen record).
w_	Janela (window)

Tipo de letra (maiusculas ou minusculas)

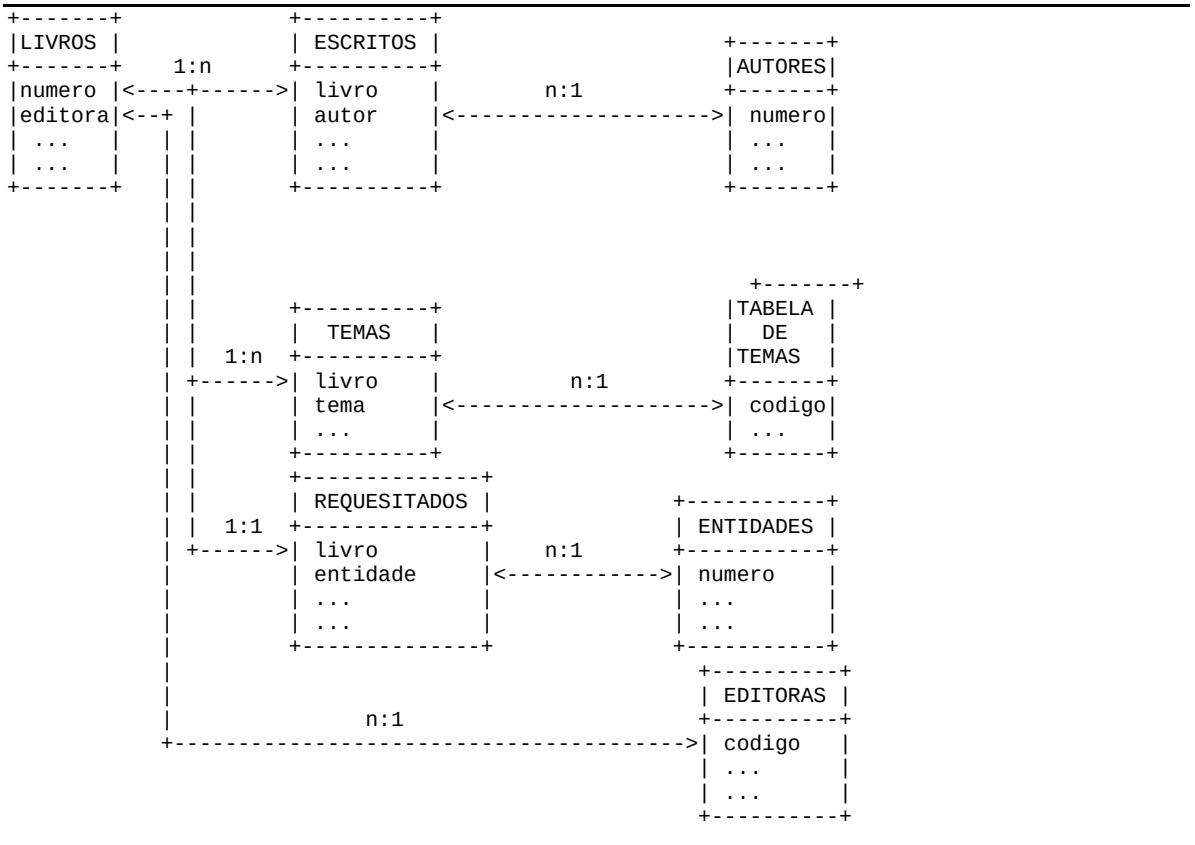
Maiusculas	Minusculas
Palavras chave	Identificadores criados pelo utilizador
Nome dos menus	Opções dos menus

16. BASE DE DADOS DE EXEMPLO DESTE MANUAL

Pretende-se uma base de dados para gerir uma biblioteca.

Numa primeira aproximação verifica-se que é necessário armazenar vários dados sobre cada livro, nomeadamente: nome; autor ou autores; numero de volumes; tema ou temas do livro. Chama-se a atenção para o facto de um livro poder ter sido escrito por mais de um autor ou incidir sobre vários temas. O nome de um autor ou de um tema geralmente são bastante extensos (69 caracteres ou mais), e são utilizados pelo menos uma vez por livro. Aconselha-se portanto o uso de uma tabela que relacione um código reduzido com o nome.
Pretende-se também que esteja prevista a possível requisição de um livro por eventuais utilizadores da biblioteca.

16.1. ESTRUTURA DA BASE DE DADOS



A tabela **livros** contém uma linha para cada livro introduzido e contém toda a informação relativa a um livro que seja única por livro. A sua estrutura é a seguinte:

NOME	TIPO	DESCRIÇÃO
numero	integer	Número do livro
nome	char(60)	Nome do livro
traducao	char(60)	Nome do tradutor
volumes	smallint	Número de volumes
paginas	smallint	Número de páginas
edicao	char(5)	Número da edição
ano	smallint	Ano de edição
data_entrada	date	Data da compra
sala	char(5)	Sala onde está arrumado
estante	char(5)	Estante onde está arrumado
prateleira	char(5)	Prateleira onde está arrumado
observacoes	char(50)	Observações sobre o livro

Estrutura da tabela livros

A tabela **autores** relaciona o nome de cada autor com um código reduzido. Não é permitida a repetição nem do código, nem do livro. A sua estrutura é a seguinte:

NOME	TIPO	DESCRIÇÃO
número	serial	Número sequêncial do autor
nome	char(70)	Nome do autor

Estrutura da tabela autores

A tabela **tabela_temas** identifica determinado tema por um código. Não é permitida a repetição de nomes e de códigos. A sua estrutura é a seguinte:

NOME	TIPO	DESCRIÇÃO
codigo	char(5)	Código de identifica,ão do tema
nome	char(50)	Descrição do tema

Estrutura da tabela tabela_temas

A tabela **editoras** identifica uma editora por um código de editora. Tal como nas tabelas anteriores não há repetições do código e nome de editora. A sua estrutura é:

NOME	TIPO	DESCRIÇÃO
codigo	char(5)	Código de identificação da editora
nome	char(50)	Nome da editora

Estrutura da tabela editoras

A tabela **escritos** relaciona determinados livros com determinados autores, permitindo que um livro possua mais de um autor, não permitindo linhas com os mesmos códigos de livro e autor.A sua estrutura é:

NOME	TIPO	DESCRIÇÃO
autor	integer	Número do autor
livro	integer	Número do livro

Estrutura da tabela escritos

A tabela **temas** funciona da mesma forma que a tabela escritos, fornecendo a informação sobre os temas focados por determinado livro. A sua estrutura é:

NOME	TIPO	DESCRIÇÃO
livro	integer	Número do livro
tema	char(5)	Código de identificação do tema

Estrutura da tabela temas

A tabela **requesitados** tem informação acerca de quem, e quais os livros que estão requesitados por determinada entidade. A sua estrutura é:

NOME	TIPO	DESCRIÇÃO
livro	integer	Identificação numérica do livro
entidade	integer	Identificação da entidade requesitante

Estrutura da tabela requesitados

A tabela **entidades** tem os dados gerais de cada entidade.

NOME	TIPO	DESCRIÇÃO
numero	integer	Número da entidade
nome_reduzido	char(20)	Nome de rápido acesso
nome	char(50)	Nome da entidade
morada	char(60)	Morada da entidade
cod_post	char(4)	Código postal

Estrutura da tabela entidades