PROGRAMAÇÃO EM INFORMIX E-SQL/C

Título: Programação E-SQL/C
Autor: Sérgio Ferreira

ÍNDICE

INTRODUÇÃO

O Informix ESQL/C (Embebed SQL and tools for C) é uma linguagem que permite ao programador criar aplicações em C que utilizem as facilidades da linguagem SQL (structured query language) para acesso a bases de dados relacionais.O ESQL/C permite a transação de dados de uma linguagem tradicional (C) paraa base de dados por intermédio do RDSQL.

O ESQL/C é ainda composto de conjuntos de funções que permitem a inclusão de funções em C nos utilitários ACE (report writer) e perform (form generator) do produto informix SQL.

O Informix ESQL/C consiste num pré-processador e num conjunto de livrarias. O pré-processador transforma um source de C com RDSQL embebido (geralmente de chamado ESQL/C) num source de C (Kernnigham & Ritchie) compatível com o seu compilador C. As rotinas incluidas na biblioteca (Llibsql.a no Xenix) são linkadas na fase da linkagem.

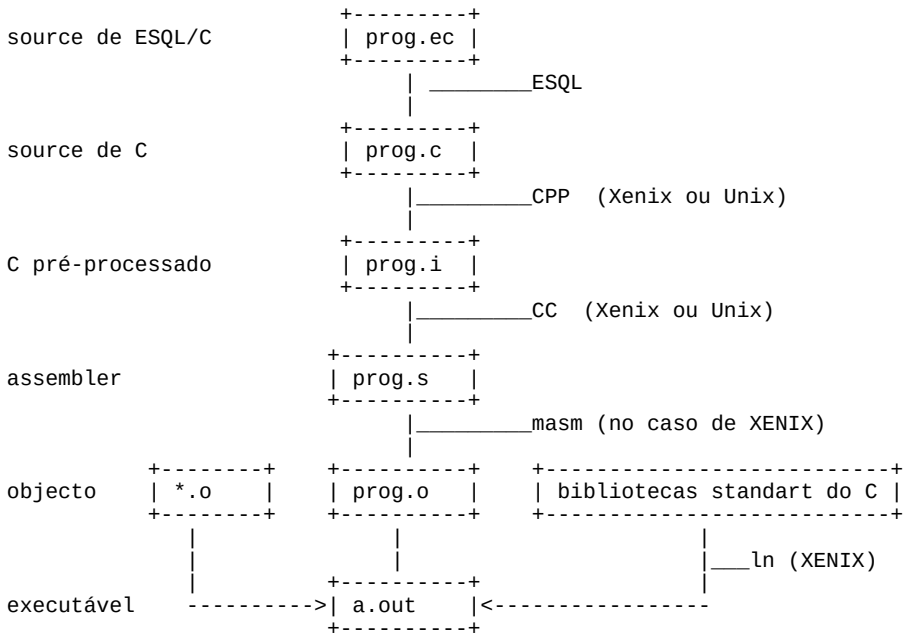


Figura 1 - Fases compilação programa ESQL/C.

ESTRUTURA de um PROGRAMA de ESQL/C

Estrutura Básica de um Programa

Tal como no C (K & R) a estrutura de um programa consiste em uma ou mais funções, declarações globais e primitivas de controlo de pré-processador (#include #ifdef, etc). A diferença está na utilização de includes próprios ($include) para ESQL/C, na utilização de instrucções de SQL ('$' no inicio da linha), na utilização de variáveis usadas nas instrucções (usando o caracter '$' no inicio da linha) e na linkagem de funções pré-definidas.

Como exemplo apresenta-se na figura 1 um programa que faz uma busca na tabela livros, ao livro cujo número é 1493.

```
#include <stdio.h>
$include sqlca;
main()
{
    $long numero;
    $char nome[61];
    $DATABASE livros;
    printf ( "entrou no programa );
    $SELECT numero, nome
            INTO $numero, $nome
            FROM livros
            WHERE livros.numero = 1493;
    printf("%ld %s/n",numero,nome);
}

%esql ex1.ec
%a.out
entrou no programa
1493 O Crime do Padre Amaro
```

Figura 2 - Exemplo programa ESQL/C

Ficheiros de Include

Para icluir outros ficheiros de ESQL/C no programa utiliza-se a sintaxe:

$include nome_do_ficheiro

Tal como no C, o ficheiro cujo o nome se indicou é copiado para a zona onde se encontra a instrucção de include.

Se o nome do ficheiro não se encontrar entre nenhum caracter, o ESQL/C vai á sua procura no directório standard do informix (talvez /usr/informix/incl) e junta-lhe a extensão .h. Para utilizar outro directório é necessário colocar o nome entre aspas:

$include "nome_do_ficheiro"

O pré-processamento do ESQL/C é efectuado antes do pré-processamento do C, assim, sempre que o ficheiro a incluir for de ESQL/C deve ser utilizada a instrucção de $include em vez de #include.

Apresenta-se seguidamente alguns exemplos da utilização da instrucção $include.

```
% cat ex2.eh
$long numero;
$char nome[61];
```

Figura 3 - Ficheiro incluir

```
% cat ex2.ec
#include <stdio.h>
#include "ex2.eh";
main()
{
        $DATABASE livros;
        $SELECT numero, nome
                INTO $numero, $nome
                FROM livros
                WHERE livros.numero = 633;
        printf("%ld %s ,numero,nome);

}
% esql ex2.ec
file "ex2.ec", line 10: 'numero' not declared
1 error found
%
```

Figura 4 - Inclusão ficheiro ESQL/C com

Foi dectectado erro pois houve uma inclusão de um ficheiro de ESQL/C com #include. Quando o compilador de ESQL/C chega á linha 10, como ainda não foi incluindo o ficheiro ex2.eh, a variável número não é ainda conhecida, pelo que é enviada mensagem de erro.

```
% cat ex3.ec
#include <stdio.h>
$include "ex2.eh";
main()
{
        $DATABASE livros;
        $SELECT numero, nome
                INTO $numero, $nome
                FROM livros
                WHERE livros.numero = 633;
        printf("%ld %s ,numero,nome);

}
% esql ex3.ec
% a.out
633 Casa da malta
%
```

Figura 5 - Alteração compilação exemplo anterior

Desta forma o programa já funciona correctamente.

Existem quatro ficheiros de include standard no ESQL/C:

sqlca: Refere-se á estrutura das mensagens de erro. Para verificar o sucesso de uma instrucção de RDSQL deve incluir sempre o ficheiro SQLCA.

sqlstmtype: Constantes inteiras correspondentes a instruções de RDSQL.

sqltypes: Definições das correspondencias entre C e RDSQL.

Variáveis Host

O informix ESQL/C é um pré-processador que converte C com RDSQL embebido em em C standard (Kerningham & Ritchie).

De todo um programa, só as instrucções RDSQL e outras operações directamente ligadas ao RDSQL são convertidos: tudo o resto permanece idêntico.

Para informar o ESQL/C que deve haver uma conversão é necessária a colocação do caracter '$' no inicio de cada linha de código RDSQL.

Algumas instrucções de RDSQL necessitam de utilizar variáveis (para as comunicações com a base de dados). Para que sejam efectuados as devidas inicializações a essas variáveis, essa declaração deve ser efectuada com o caracter '$' no inicio da linha. A estas variáveis dá-se o nome de variáveis host.

Uma variável host funciona exactamente como uma variável normal de C. A única diferença consiste na declaração e na permissão para estas serem utilizadas em instrucções de RDSQL.

Uma variável host pode ser utilizada, afectada e enviada como parametro de uma função de forma idêntica a qualquer variável normal de C.

Qualquer tipo standard de C pode ser declarado (inclusivamente estruturas).

Não podem ser usados tipos definidos com typedef para a declaração de variáveis host.

Na verdade uma variável host, durante a compilação pelo ESQL/C, é transformada numa variável normal de C (local ou global). A diferenciação serve apenas para que o ESQL/C saiba que deve efectuar algumas afectações ou inicializações que necessita para a comunicação com a base de dados.

A figura seguinte apresenta um programa com uma variável host declarada com um tipo definido pela instrucção typedef.

```
% cat ex4.ec
#include <stdio.h>
$include sqlca;
/*
 * Leitura do autor com o numero 1
 */
main()
{
        typedef struct {
                long numero;
                char nome[71];
        }AUTOR;
        $AUTOR aut_buff;
        $DATABASE livros;
        $SELECT numero, nome
                INTO $aut_buff.numero, $aut_buff.nome
                FROM autores
                WHERE autores.numero = 1;
        if ( !sqlca.sqlcode )
                printf("%ld %s ,aut_buff.numero,aut_buff.nome);
        else
                printf("Houve erro na leitura da base de dados );

}
% esql ex4.ec
file "ex4.ec", line 15: Invalid type or storage class áutor'
file "ex4.ec", line 20: áut_buff.numero' not declared
2 errors found
%
```

Figura 6 - Programa esql com utilização typedef numa variável host

O erro aparece devido á utilização de typedefs. O ESQL/C só conhece os tipos standard do C ou estruturas.

```
% cat ex5.ec
#include <stdio.h>
$include sqlca;
/*
 * Leitura do autor com o numero 1
 */
main()
{
        $struct {
                long numero;
                char nome[71];
        }aut_buff;
        $DATABASE livros;
        $SELECT numero, nome
                INTO $aut_buff.numero, $aut_buff.nome
                FROM autores
                WHERE autores.numero = 1;
        if ( !sqlca.sqlcode )
                printf("%ld %s ,aut_buff.numero,aut_buff.nome);
        else
                printf("Houve erro na leitura da base de dados );
}
% esql ex5.ec
% a.out
1 ECA DE QUEIROZ
%
```

Figura 7 - Correcção exemplo anterior

Neste caso o compilador já não dá erros e o programa é executado com sucesso.

A representação interna de valores nulos (na base de dados) varia com a máquina e o tipo de dados, por isso aconselha-se a utilização das funções risnull e rsetnull para testes ou afectações de variáveis com valores nulos.

```
% cat ex6.ec
#include <stdio.h>
$include sqltypes;
$include sqlca;
main()
{
        $long numero;
        $char nome[61];
        $char edicao[6];
        $DATABASE livros;
        $SELECT numero, nome, edicao
                        INTO $numero, $nome, $edicao
                        FROM livros
                        WHERE livros.numero = 383;
        if ( risnull(SQLCHAR,edicao) )
           printf("%ld %s  campo da edicao esta a null ,numero,nome);
        else
           printf("%ld %s %s ,numero,nome,edicao);
}
% esql ex6.ec
% a.out
383 O trigo e o Joio
o campo da edição está a null
%
```

Figura 8 - Teste linhas devolvidas com NULLs

Variáveis Indicator

Algumas instruções RDSQL (INTO em SELECT ou FETCH) podem devolver um valor NULO numa variável host. Esta situação acontece quando a variável host corresponde a uma coluna da base de dados que permite a inserção de NULOS.

Para facilitar o teste da devolução de valores NULOS deve-se declarar uma variável indicator associada á variavel host.

Uma variável indicator é uma variável host do tipo short.

A variável que indicator está associada chama-se variável main.

Quando uma instrução de ESQL/C devolve um valor NULO afecta (caso exista), com um valor negativo a variável indicator. Neste caso, o valor da variável host não é válido. Se não for definida variável indicator deve-se testar a variável sqca.sqlcode, para verificar a devolução ou não de um valor NULO.

A variável indicator associada á variável main na instrucção é utilizada colocando-se o caracter ':' imediatamente a seguir á variável main seguindo-se a variável indicator, ou seja:

```
$instrucção $clausulas variável_host: variável_indicator.
```

Apresenta-se em seguida um exemplo de utilização de uma variável indicator.

```
% cat ex7.ec
#include <stdio.h>
$include sqltypes;
$include sqlca;
main()
{
        $long numero;
        $char nome[61];
        $char edicao[6];
        $short indicadora;
        $DATABASE livros;
        $SELECT numero, nome, edicao
                        INTO $numero, $nome, $edicao:indicadora
                        FROM livros
                        WHERE livros.numero = 383;
        if ( indicadora < 0 )
           printf("%ld %s  o campo da edicao esta a null  ",numero,nome);
        else
           printf("%ld %s %s  ",numero,nome,edicao);
}
% esql ex7.ec
% a.out
383 O trigo e o Joio
o campo da edição esta a null
%
```

Figura 9 - Teste devolução valor nulo com uma variável indicator.

Se for declarada uma variável host do tipo string e se tentar utilizar esta variável host na leitura de uma linha do tipo char em que a dimensão seja maior que a da variável em causa (a linha não cabe na string host), o RDSQL truncará a string e afecta a variável indicator com o número de bytes da string antes de ser truncada. O facto de a string ter sido truncada é assinalado na estrutura sqlca, por intermédio da afectação com o valor 'w' ao elemento sqlca.sqlwarn.sqlwarn1.

A variável indicator é afectada com zero se não houver devolução de NULOS ou nenhuma string for truncada.

Apresenta-se em seguida um exemplo de um "SELECT" em que uma coluna do tipo char tem maior dimensão que a string host que vai ser afectada.

```
% cat ex8.ec
#include <stdio.h>
$include sqltypes;
$include sqlca;
main()
{
        $long numero;
        $char nome[6];
        $short indicadora;
        $DATABASE livros;
        $SELECT numero, nome
                        INTO $numero, $nome:indicadora
                        FROM autores
                        WHERE numero = 2;
        if ( indicadora < 0 )



           printf("%ld o campo do nome esta a null ,numero);
        else if ( indicadora > 0 )
           printf("%ld %s  nome foi truncado! A coluna tinha %d ",
                                numero,nome,indicadora);
        else
           printf("%ld %s ,numero,nome);
}
% esql ex8.ec
% a.out
2 FERNA
O nome foi truncado! A coluna tinha 70
%
```

Figura 10 - Exemplo que uma coluna cabe numa string host.

Conforme se pode ver na figura pretende-se colocar uma coluna com 70 numa string host com 6 e a variável indicator associada foi afectada, com um valor negativo.

Em vez de usar a palavra NULL num instrucção INSERT em que se pretende inserir uma coluna com NULL, pode-se usar uma variável indicator contendo um valor negativo.

```
% cat ex9.ec
#include <stdio.h>
$include sqltypes;
$include sqlca;
main()
{
  $char traducao[61];
  $short trad_ind;
  $char sala[6];
  $char estante[6];
  $char prateleira[6];
  $short col_ind;
  $char observacoes[51];
  $short obs_ind;
  trad_ind = -1;
  col_ind = -1;
  obs_ind = -1;
  $DATABASE livros;
  $INSERT INTO livros
    VALUES (1111,
                              "Viagens na minha terra",
                              $traducao:trad_ind,
            1 ,
                              500 ,
                              "BERTR" ,
                              1989 ,
                              NULL ,
                              $sala:col_ind ,
                              $estante:col_ind ,
                              $prateleira:col_ind ,
                              $observacoes:obs_ind);
  if ( sqlca.sqlcode )
    fprintf(stderr,"Erro na insercao na base de dados %d ",
                              sqlca.sqlcode);
  else
    printf("Insercao bem sucedida );
}
% esql ex9.ec
% a.out
Inserção bem sucedida
%
```

Figura 11 - Inserção base dados com campo NULL afectando varáveis indicator.

CURSORES, GESTÃO ERROS, GESTÃO DINÂMICA INSTRUÇÕES

Gestão Cursores

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

```
#include <stdio.h>
$include sqltypes;
$include sqlca;
main()
{
    $long numero;
    $char nome[61];
    $char edicao[6];
    $DATABASE livros;
    $SELECT numero, nome, edicao
            INTO $numero, $nome, $edicao
            FROM livros
            WHERE livros.edicao = "BERTR";
    if(sqlca.sqlcode)
        fprintf(stderr,"Erro num SELECT a base dados %d ,sqlca.sqlcode);
    else
        printf("%d %s %s ,numero,nome,edicao);
}
%esql ex10.ec
%a.out
Erro num SELECT a base dados -284
%
```

Figura 12 - Leitura com instrução SELECT que origem mais que uma linha.

Se consultar o manual de ESQL/C verificará que erro corresponde a uma busca que não retornou exactamente uma linha.

Este erro acontece porque as variáveis host da clausula INTO são referentes apenas a uma linha da tabela livros, e a instrução deu origem a mais do que uma linha.

Para eliminar esse problema o ESQL/C possui um mecanismo chamado "cursor".

Um cursor é uma lista de indicadores de linhas de uma tabela. 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.

Um cursor permite aceder ás linhas, uma de cada vez.

A linha activa 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.

Para a correcta utilização de um cursor geralmente as acções são:

1. Declarar o cursor. Esta acção é efectuada com o auxilio da instrucção DECLARE e associa o nome escolhido para o cursor com determinado SELECT.

```
$DECLARE cursor_livros CURSOR FOR
SELECT numero, nome, edicao
FROM livros
WHERE numero < 1000;
```

2. Abrir o cursor. Esta acção é efectuada com a instrucção OPEN. é com a execução desta instrução que é executada a instrução SELECT associada ao cursor. é gerado um conjunto activo de linhas. O cursor fica posicionado imediatamente antes da primeira linha.

```
$OPEN cursor_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 clausula INTO. Se o cursor estiver posicionado na ultima linha, é afectada a variável sqlca.sqlcode com o valor SQLNOTFOUND.

```
for(;;)
{
$FETCH cursor_livros
INTO $numero, $nome, $edicao;
if ( sqlca.sqlcode )
break;

printf("%ld %s %s
,numero,nome,edicao);
}
```

4. Fechar o cursor. Acção efectuada pela instrucção 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 executado o SELECT associado ao cursor.

```
$CLOSE cursor_livros;
```

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

```
#include <stdio.h>
$include sqlca;
main()
{
    $long numero;
```

```
    $char nome[61];
    $char edicao[6];
    /* Escolha da base de dados
     */
    $DATABASE livros;
    printf ( "entrou no programa );
    /* Este select vai ler todas as linhas da tabela livros
     * com numero de livro menor que 1000
     */
    $DECLARE cursor_livros CURSOR FOR
        SELECT numero, nome, edicao
            FROM livros
            WHERE numero < 1000;

    /* Abertura do cursor
     */
    $OPEN cursor_livros;
    for(;;)
    {
        $FETCH cursor_livros
            INTO $numero, $nome, $edicao;
        if ( sqlca.sqlcode )   /* Se houve erro de RDSQL sai do ciclo */
            break;
        printf("%ld %s %s ,numero,nome,edicao);
    }

}
%esql  ex11.ec
%a.out
entrou no programa
379 Retalhos da vida de um medico                                   BERTR
382 A noite e a madrugada                                           BERTR
383 O trigo e o Joio
633 Casa da malta                                                   BERTR
%
```

Figura 13 - Leitura dos livros com usando cursor

Um cursor pode ser declarado para alteração de determinadas colunas nas linhas seleccionadas.

Forma de utilização de cursores com instrução UPDATE associada:

1. Declarar o cursor associando-lhe uma instrução UPDATE.

```
$DECLARE cursor_livros FOR
SELECT numero, nome, edicao
FROM livros
WHERE numero < 1000
FOR UPDATE OF edicao
```

2. Abrir o cursor da forma descrita anteriormente.

```
$OPEN cursor_livros;
```

3. Evocar a instrução UPDATE, alteração das colunas definidas na clausula SET do elemento corrente do cursor.

```
$UPDATE
SET
WHERE CURRENT OF cursor_livros
```

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

```
#include <stdio.h>
$include sqlca;
/*
 * Este programa altera a edicao dos livros com numero < 1000
 * com a coluna a NULL
 */
main()
{

    $long numero;
    $char nome[61];
    $char edicao[6];
    $short ind_edicao;
    /* Escolha da base de dados
     */
    $DATABASE livros;
    printf ( "entrou no programa );
    /* Este select vai ler todas as linhas da tabela livros
     * com numero de livro menor que 1000
     */
    $DECLARE cursor_livros CURSOR FOR
        SELECT numero, nome, edicao
            FROM livros
            WHERE numero < 1000
            FOR UPDATE OF edicao;

    /* Abertura do cursor
     */
    $OPEN cursor_livros;
    for(;;)
    {
        $FETCH cursor_livros
            INTO $numero, $nome, $edicao:ind_edição;
        if ( ind_edição < 0 )
        {
                /* Pedir o nome da editora ao utilizador */
            printf("%ld %s ual a editora para este livro : ",numero,nome);
            gets(edicao);
            $UPDATE livros
                SET edicao = $edicao
                WHERE CURRENT OF cursor_livros;
        }
        if ( sqlca.sqlcode )
            break;
    }

}
%esql ex12.ec
%a.out
entrou no programa
383 O trigo e o Joio
Qual a editora para este livro : BERTR
%
```

Figura 14 - Cursor para alteração FOR UPDATE

O mecanismo descrito anteriormente só permite o posicionamento sequencial do principio para o fim do 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).

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 instrucção 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
- Erimeiro elemento - FETCH FIRST
- Último elemento - FETCH LAST
- Elemento corrente n do conj. activo - FETCH RELATIVE
- Elemento n do conjunto activo - FETCH ABSOLUTE

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

```
#include <stdio.h>
$include sqlca;
main()
{
    $long numero;
    $char nome[61];
    $char edicao[6];
    /* Escolha da base de dados
     */
    $DATABASE livros;
    printf ( "entrou no programa );
    /* Este select vai ler todas as linhas da tabela livros
     * com numero de livro menor que 1000
     */
    $DECLARE cursor_livros SCROLL CURSOR FOR
        SELECT numero, nome, edicao
            FROM livros
            WHERE numero < 1000;
    /* Abertura do cursor
     */
    $OPEN cursor_livros;
    if ( sqlca.sqlcode )
        fprintf(stderr,"ERRO %ld ,sqlca.sqlcode);
    $FETCH LAST cursor_livros
            INTO $numero, $nome, $edicao;
    for(;;)
    {
        $FETCH PREVIOUS cursor_livros
            INTO $numero, $nome, $edicao;
        if ( sqlca.sqlcode )
            break;
        printf("%ld %s %s ,numero,nome,edicao);
    }
}
%esql ex14.ec
%a.out
%
```

Figura 15 - Programa que usa scroll cursor

Cursores inserção

Para optimizar a inserção numa base de dados pode-se utilizar um mecanismo de ESQL/C chamado cursor de inserção (insert cursor). Este mecanismo consiste no armazenamento num buffer dos dados recebidos, e na sua escrita na base de dados quando o buffer está cheio ou é forçada a escrita.

Manuseamento proposto de cursores de inserção :

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

```
$DECLARE cursor_livros
FOR INSERT INTO livros(numero, nome, adicao)
VALUES ($numero, $nome, $adicao)
```

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

```
$OPEN cursor_livros
```

3. Ler os dados por intermédio da linguagem C (scanf, gets, etc.) para dentro de determinada variáveis (host).

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

```
$PUT cursor_livros
```

5. Escrever o buffer na base de dados.

$FLUSH cursor_livros

6. Fechar o cursor.

$CLOSE cursor_livros

O ESQL/C 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 variavel sqlca.sqlerr[2], que deve conter o número de linhas inseridas na base de dados.

```
#include <stdio.h>
$include sqlca;
main()
{
    $long numero;
    $char nome[61];
    $char edicao[6];
    char resp;
    /* Escolha da base de dados
     */
    $DATABASE livros;
    if ( sqlca.sqlcode )
        {
        fprintf(stderr,"Erro na abertura da base de dados (%ld) ,
                                                    sqlca.sqlcode);
        exit(1);
        }
    printf ( "entrou no programa );
    $DECLARE cursor_livros CURSOR FOR
        INSERT INTO livros (numero, nome, edicao)
                    VALUES ($numero, $nome, $edicao);
    /* Abertura do cursor
     */
    $OPEN cursor_livros;
    if ( sqlca.sqlcode )
        {
        fprintf(stderr,"Erro na abertura do cursor de insercao (%ld) ,
                                                    sqlca.sqlcode);
        exit(1);



        }
    resp='s';
    while ( resp == 's' || resp == 'S' )
        {
```

```
        printf("NUMERO: ");
        scanf("%ld",&numero);
        printf(" OME: ");
        scanf("%60s",nome);
        printf(" DICAO: ");
        scanf("%6s",edicao);
        printf(" n );

        $PUT cursor_livros;
        if ( sqlca.sqlcode )
            fprintf(stderr,"O livro nao foi bem introduzido (%ld) ,
                                                    sqlca.sqlcode);
        printf("Deseja continuar (S ou N): ");
        fflush(stdin);
        scanf("%c",&resp);
        }
    $CLOSE cursor_livros;
    if ( sqlca.sqlcode )
        fprintf(stderr,"O livro nao foi bem introduzido );
}
%esql
%a.out
%
```

Figura 16 - Cursor com inserção

A estrutura SQLCA

Durante um programa de ESQL/C, quando se executa determinada instrução do RDSQL é necessário testar o sucesso ou não da mesma.

Todos as instrucções de RDSQL (excepto DECLARE) devolvem um valor ao ESQL/C por intermédio da estrutura sqlca no ficheiro de include sqlca.h:

```
struct sqlca {
 long sqlcode;
char sqlerrm [72];
 char sqlerrp [8];
 long sqlerrd [6];
 struct sqlcaw_s{
  char sqlwarn0;
  char sqlwarn1;
  char sqlwarn2;
  char sqlwarn3;
  char sqlwarn4;
  char sqlwarn5;
  char sqlwarn6;
  char sqlwarn7;
    }sqlwarn;
    }sqlca;
```

O significado de cada um destes campos é:

sqlcode: Indica o resultado de uma instrucção de RDSQL.

Toma os valores:

0 - instrucção bem sucedida.
SQLNOTFOUND - 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[0]: não é utilizado
sqlerrd[1]: Valor retornado pelo
Ver erros do C-isam
sqlerrd[2]: número de linhas processadas
sqlerrd[3]: não é utilizada
sqlerrd[4]:
sqlerrd[5]: é o ROWID da última linha

sqlwarn: É uma estrutura que contem 8 caracteres que assinalam várias situações que necessitam de aviso durante a execução de uma instrucção RDSQL.

sqlwarn0:

Se contiver:
Caracter W: Um ou mais dos outros elementos da estrutura contêm 'W'
Espaço: Não é necessário verificar os outros elementos da estrutura.

sqlwarn1:

Se contiver:
Caracter W: Os dados de uma ou mais colunas foi truncado para caber dentro de uma variavel host. Pode verificar qual das colunas foi truncada se examinar a respectiva variavel indicator.
Espaço: Nao houve truncagem de dados.

sqlwarn2:

Se contiver:
Caracter W : O numero de items na lista de colunas da instrução SELECT não é o mesmo que o numero de variáveis host na clausula INTO.
Espaço : Não houve problemas no SELECT.

sqlwarn3:

Se contiver:
Caracter W: Foi pedido o DESCRIBE de uma instrução UPDATE ou DELETE que foi preparada (PREPARE) sem clausula WHERE. A verificação desta variavel evita mudanças radicais numa tabela.
Espaço : Não houve erro no describe.

sqlwarn4: Nao é utilizado

sqlwarn5: Nao é utilizado

sqlwarn6: Nao é utilizado

sqlwarn7: Nao é utilizado

Gestão de Instrucções 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 instrucções de RDSQL a utilizar. Algumas dessas situações podem ser as seguintes :

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

As instrucções PREPARE e EXECUTE

Para executar uma instrucção dinamica seguem-se os seguintes passos:

1. Colocação da instrucção dentro de uma string (declarada como host).

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

```
$PREPARE del1 FROM delete1;
```

3. Execução da instrucção. é efectuada pela instrucção EXECUTE.

```
$EXECUTE del1
```

4. Apresenta-se seguidamente o exemplo de uma remoção na tabela de livros.

```
#include <stdio.h>
$include sqlca;
main()
{
     $long numero;
     $char nome[61];
     $char edicao[6];
     $char delete1[100];
     char resp;
     /* Escolha da base de dados
      */
     $DATABASE livros;
     if ( sqlca.sqlcode )
             {
             fprintf(stderr,"Erro na abertura da base de dados (%ld) ,

                     sqlca.sqlcode);
             exit(1);
             }
     printf ( "PROGRAMA PARA APAGAR LIVROS DA TABELA DE LIVROS );
     printf("NUMERO DO LIVRO A APAGAR: ");
     scanf("%ld",&numero);
     $SELECT nome, edicao INTO $nome, $edicao FROM livros
             WHERE numero = $numero;
     if ( sqlca.sqlcode == SQLNOTFOUND)
             {
             fprintf(stderr,"Livro %ld inexistente ,numero);
             exit(1);
             }
     else if ( sqlca.sqlcode )
             {
             fprintf(stderr,"Erro na abertura da base de dados (%ld) ,

                     sqlca.sqlcode);
             exit(1);
             }
     printf("LIVRO: );
     printf("NUMERO: %ld ,numero);
     printf("NOME: %s ,nome);
     printf("EDICAO: %s n",edicao);
     printf("QUER MESMO REMOVER (S ou N): ");
     fflush(stdin);
     scanf("%c",&resp);
     if ( resp == 's' || resp == 'S' )
             {
             sprintf(delete1,"%s%ld","DELETE FROM livros WHERE numero = ", numero);
             $PREPARE del1 FROM $delete1;
             if ( sqlca.sqlcode )
                     {
                     fprintf(stderr,"Erro na preparacao da intrucao de remocao (%ld)
,

                             sqlca.sqlcode);
                     exit(1);
                     }
             $EXECUTE del1;
             if ( sqlca.sqlcode )
                     fprintf(stderr,"O livro nao foi bem removido (%ld) ,

                             sqlca.sqlcode);

             }
     }
```

Figura 17- Remoção dinâmica

As instrucções RDSQL enviadas ao PREPARE em run-time não se podem referir a variáveis host, 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 variavel. Para assignar a variavel ao ponto de interrogação usa-se a clausula USING $var1 $var2 ... $varn na instrucção EXECUTE. As variáveis ficam assignadas ao ponto de interrogação correspondente á sua posição.

```
#include <stdio.h>
$include sqlca;
main()
{
     $long numero;
     $char nome[61];
     $char edicao[6];
     $char insert1[100];
     char resp;
     /* Escolha da base de dados
      */
     $DATABASE livros;
     if ( sqlca.sqlcode )
             {
             fprintf(stderr,"Erro na abertura da base de dados (%ld) ,

                     sqlca.sqlcode);
             exit(1);
             }
     printf ( "entrou no programa );
     sprintf(insert1,"%s%s","INSERT INTO livros ( numero, nome, edicao)",
                                          " VALUES ( ?, ?, ?)");
     $PREPARE ins1 FROM $insert1;
     if ( sqlca.sqlcode )
             {
             fprintf(stderr,"Erro na preparacao da intrucao de insercao (%ld) ,

                     sqlca.sqlcode);
             exit(1);
             }
     resp='s';
     while ( resp == 's' || resp == 'S' )
             {
             printf("NUMERO: ");
             scanf("%ld",&numero);
             printf(" OME: ");
             scanf("%60s",nome);
             printf(" DICAO: ");
             scanf("%6s",edicao);
             printf(" n );

             $EXECUTE ins1 USING $numero, $nome, $edicao;
             if ( sqlca.sqlcode )
                     fprintf(stderr,"O livro nao foi bem introduzido (%ld) ,

                     sqlca.sqlcode);
             printf("Deseja continuar (S ou N): ");
             fflush(stdin);
             scanf("%c",&resp);
             }

}
```

Figura 18 - Utilização dos operadores

A instrucção DECLARE nas instrucções dinamicas

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

1. Preparar a instrucção com PREPARE.

```
$PREPARE
```

2. Declarar o cursor para essa instrucção.

```
DECLARE CURSOR FOR state_id
```

3. Usar o cursor da mesma forma que se usa com instrucções definidas antes da compilação.

```
OPEN CURSOR
while()
{
FETCH
}
```

Se for usado o caracter '?' para substituir variáveis é necessário usar a clausula USING na abertura do cursor.

```
OPEN CURSOR USING $var1 $var2
    ...  $varn
    while()
        {
        FETCH
        }
```

A instrução DESCRIBE

Se pretendermos trabalhar com diversas tabelas (de uma ou mais bases de dados), das quais nao conhecemos nada sobre a estrutura antes da fase de compilação, deparamos com o problema de não saber quantas variáveis host se deve declarar nem de que tipo as declarar.

A instrucção DESCRIBE resolve esse problema. Esta instrucção fornece informação acerca da memória necessária para receber uma linha da tabela em causa. A partir desta informação, pode-se pedir ao sistema operativo que nos forneça a memória necessária (por exemplo utilizando a função malloc).

A instrucção DESCRIBE é apenas usada para a instrucção SELECT. A estrutura na qual a instrução DESCRIBE escreve é a estrutura sqlda, e a sua definição está contida no ficheiro de include sqlda.h, a sua estrutura é a seguinte :

```
struct sqlvar_struct {
short sqltype;
short sqllen;
char *sqldata;
short *sqlind;
char *sqlname;
char *sqlformat;
};
struct sqlda {
short sqld;
struct sqlvar_struct *sqlvar;
};
```

Quando se vai utilizar a instrucção DESCRIBE deve-se declarar uma variavel host do tipo sqlda *.

O siginificado de cada um dos elementos da estrutura é o seguinte :

sqld: Contem o numero de elementos sqlvar_struct contidos na variavel sqlvar, ou seja o número de colunas ou variáveis que se vai utilizar no SELECT.

sqlvar: Contêm os pointers para as estruturas sqlvar_struct (em número igual a sqld). Para aceder a cada uma delas pode utilizar as facilidades da aritmética de pointer(s).

Elementos estrutura sqlvar_struct:

sqltype: Diz qual é o data type a ser transferido.
Existem constantes definidas no ficheiro de include sqltypes.h com os possíveis valores de que esta variável é afectada.

sqllen: Tamanho em bytes necessários para uma variável do tipo CHAR.

sqldata: Pointer para a memória onde irão ficar os dados. Atenção que a memória ainda não vem inicializada, há que o fazer com as funções do C.

sqlind: Pointer para a memória onde vai ficar uma variável indicator para associar variável main. Este elemento, tal como anterior também vem inicializado a NULL, e por isso deve ser inicializado com o endereço da memória que se vai utilizar. Se não pretender utilizar esta variável, deve deixar o seu valor com NULL.

sqlname: Pointer para uma string que contém o nome da coluna que vai ser escrita nesta estrutura. Esta variável vem inicializada da intrucção DESCRIBE.

sqlformat: não é utilizado

Instrucções SELECT parametrizadas definidas dinamicamente

Uma instrucção SELECT não parametrizada definida dinâmicamente é uma instrucção SELECT que foi definida em run-time (com o auxilio da instrucção PREPARE), e para a qual não se sabe quantas e quais as variáveis host a utilizar (SELECT de bases de dados com estrutura de tabelas variável etc.).

Para a eficaz utilização do SELECT aconselha-se as seguintes fases :

1. Declarar um pointer (host) para a estrutura sqlda.

```
$struct sqlda *sqlda_ptr;
```

2. Preparar (PREPARE) a instrucção SELECT e fornecer-lhe um identificador de instrucção.

```
$PREPARE instr_id FROM instrucção;
```

3. Descrever a instrucção para dentro do pointer para a estrutura sqlda.

```
$DESCRIBE instr_id INTO sqlda_ptr;
```

4. Alocar memória para cada uma das variáveis necessárias e inicializar os pointer(s) com essa memória (sqlda e eventualmente sqlind) em cada uma das estruturas sqlvar_struct.

5. Declarar um cursor para um identificador de instrucção já preparado.

```
$DECLARE cursor_livros CURSOR FOR instr_id;
```

6. Executar os acessos pretendidos ao cursor (instrucção FETCH) usando a clausula USING DESCRIPTOR. O argumento desta clausula é o pointer para a estrutura sqlda.

```
$FETCH cursor_livros USING DESCRIPTOR sqlda_ptr;
```

COMPILAÇÃO DOS PROGRAMAS ESQL/C'

Sistemas UNIX

Para compilar programas de ESQL/C em sistemas UNIX ou XENIX, basta que o programa possua a extenção .ec e que se evoque o comando esql. A sintaxe de compilação é :

```
esql [-e] [-outros_args ...] [-o
nome_executavel]
ficheiros_esql.ec ...
[outros_progs.c...]
[outros_objs.o ...][-llivrarias ...]
```

As opções de compilação têm o seguinte significado :

e: faz com que seja apenas chamado o ESQL/C produzindo apenas um programa C.

outros_args: O comando esql passa ao compilador de C (cc) todos os argumentos que não conhecer.

o nome_executavel: é o nome do ficheiro onde se pretende colocar o executável a que o programa dá origem.

outros_progs.c: Outros programas em C que se pretenda compilar, e linkar com o nosso programa ESQL/C. O comando esql passa directamente para o compilador de C todos os ficheiros com extensão .c.

outros_objs.o: Outros programas já compilados (objectos) que se pretenda linkar com o nosso programa de ESQL/C. O comando esql passa directamente ao cc todos os ficheiros com extensão .o.

lbibliotecas: bibliotecas onde se pretendam ir buscar objectos. Este argumento tambem é enviado directamente ao cc.

FUNÇÕES C NO ACE E PERFORM

Este capitulo faz a inclusão de funções em C no gerador de relatórios ACE ou no gerador de ecrans PERFORM.

Se não possuir o produto INFORMIX SQL ou não está familiarizado com a programação em ACE e PERFORM pode saltar este capítulo.

O PERFORM e o ACE criam relatorios e entradas de dados sem modificações dos dados. Por vezes é necessário tratar os dados recibidos de uma base de de dados (por exemplo uma estatistica) ou validas os dados a introduzir.

Como estas linguagens não possuem uma versatilidade necessaria, permiteuma versatilidade necessária, permite a chamada a funções C.

Utilizações Funções C no ACE:

Uma função C que se utiliza durante um relatorio deve ser declarado na secção DEFINE de um ficheiro de definição relatorio, utilizada na secção FORMAT e linkada ao view-time da ACE.

A utilização de uma função C na ACE deve ter a seguinte forma:

1. Construir a função num ficheiro.

2. Compilar e linkar a função ao run-time.

```
cace cprogram.c ecprogram.ec
```

3. Declarar as funções no programa perform. Esta declaração é efectuada na secção DEFINE.

```
DEFINE
function my-func
END
```

4. Evocar a Função efectuada na secção FORMAT. Na evocação passa-se uma listas de expressões que vão ser enviadas como parametros da funções.

```
CALL my-func
```

Uma expressão pode ser qualquer coisa desde uma constante alfabetica ou numérica até nomes de colunas variaveis ACE, parametros ACE, funções da ACE, strings, operadores lógicos e aritiméticos ou palavras chaves.

Uma função C pode tambem ser usada como expressão, devendo retornar um valor e não devendo ser chamada com a palavra CALL.

Estrutura de um programa C para utilização no ACE

As funções para serem utilizadas no ACE devem:

1. incluir o ficheiro "ctools.h"

```
#include "ctools.h"
```

O ficheiro ctools.h foi desenvolvido para facilitar a comunicação de funções C e a ACE.

2. Declarar as funções que vêm a utlizar sem o tipo value ptv:

```
valueptr função1();
valueptr função2();
```

O tipo valueptr está definido no ficheiro ctools.h

3. Declarar e inicializar o array da estrutura user funcs[ ]:

```
struact ufunc userfunc[]=
{
        "funçaõ", função,
        "função", função,
                0,0
};
```

O tipo ufunc está definida no ficheiro ctools.h.

As strings usadas no array são os nomes a utilizar no programa em ACE e funçoa 1 e função 2 são as pointer's para as funções definidas no programas em C.

Este array serve para fazer a ligação entre o nome usado no programa em ACE e o identificador da função (que é um pointer para função)

4. Definir as funções a utilizar:

- As funções devem ser declaradas do tipo valueptr do valueptr.

- Todos os argumentos do função têm que ser declarados do tipo valllueptr.

```
valueptr função1 (argumento)
valueptr argumento;
{
:
;
}
```

- Para retornar valores deve chamar uma função, conforme o tipo de valor a retornar:

```
strreturn (s,len);
```

Compilação de funções para utilizar com o ACE

O ACE é um programa semi-intrepertado, como tal, o programa executável proprio acego que dados ficheiros que resultado da compilação com sacego.

Para que o acego consiga executar uma função C, esta tem de ser compilada e linkada a ele proprio (acego).

Como não se pode linkar uma função a um executavel é fornecida o programa acego na forma objecto, que se linka com a nossa função C e cria uma nova versão de acego.

Compilação em Sistemas UNIX:

Para simplificar a compilação nos sistemas unix é fornecida um shell script utilizada para compilar e linkar uma função. O seu nome é cace e a sua sintaxe de utilização é:

```
cace cprogram .[c/ec] -o newacego
```

O cprogram é o programa a linkar, que tanto pode ser de C normal (extenção) ou de ESQL/C (extensão.ec). newacego é o nome do executavel que nos vai servir para executar os programas ace compilados pelo comando aceprep.

Os passos para incluir uma função de C num ACE podem ser:

1. Criar a função

```
vi ex.c
vi ex.ec
```

2. Criar um report

```
vi ace.ace
```

3. Compilar um relatorio

```
aceprep ace.ace
```

4. Compilar a nova função e linkar ao novo executavel

cace

5. Executar a ace

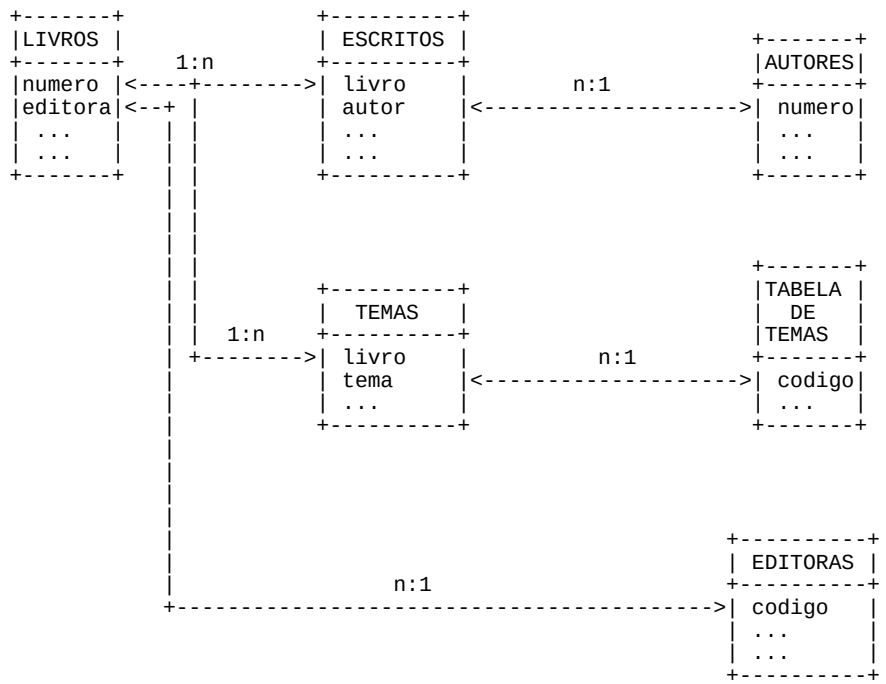
newacego - yrevf

BASE DADOS EXEMPLO DESTE MANUAL

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

A estrutura proposta será:



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:

```
create table livros
    (
     numero integer not null,
     nome char(60) not null,
     traducao char(60),
     volumes smallint,
     paginas smallint,
     edicao char(5),
     colocacao char(5),
     observacoes char(50)
    );
create unique index ix102_1 on livros (numero);
create unique index ix102_2 on livros (nome);
```

Figura 19 - Estrutura 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:

```
create table autores
    (
     numero serial not null,
     nome char(70) not null
    );
create unique index ix101_1 on autores (numero);
```

Figura 20 - Estrutura 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:

```
create table tabela_temas
    (
     codigo char(5) not null,
     nome char(50) not null
    );
create unique index ix114_1 on tabela_temas (codigo);
```

Figura 21 - Estrutura tabela tabela_temas

A tabela editorasidentifica 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:

```
create table editores
    (
```

```
     codigo char(5) not null,
     nome char(50) not null
    );
create unique index ix110_1 on editores (codigo);
```

Figura 22 - Estrutura 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:

```
create table escritos
    (
     autor integer not null,
     livro integer not null
    );
create unique index ix112_2 on escritos (livro);
```

Figura 23 - Estrutura tabela escritos

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

```
create table temas
    (
     livro integer not null,
     tema char(5) not null
    );
create index ix115_1 on temas (livro);
create index ix115_2 on temas (tema);
```

Figura 24 - Estrutura tabela temas