Cursores e Variáveis

Cursores são conjuntos de linhas de alguma tabela que são processadas separadamente. São definidos dentro de programas armazenados (procedimentos e funções. Cursores são *read-only*, e ao serem processados devolvem todas as linhas que os formam, sem permitir saltos. Veja um exemplo

```
CREATE PROCEDURE curdemo()
BEGIN
  DECLARE done INT DEFAULT FALSE;
  DECLARE a CHAR(16);
  DECLARE b, c INT;
  DECLARE cur1 CURSOR FOR SELECT id,data
    FROM test.t1;
  DECLARE cur2 CURSOR FOR SELECT i
    FROM test.t2;
  DECLARE CONTINUE HANDLER FOR
    NOT FOUND SET done = TRUE;
  OPEN cur1;
  OPEN cur2;
  read_loop: LOOP
    FETCH cur1 INTO a, b;
    FETCH cur2 INTO c;
    IF done THEN
      LEAVE read_loop;
    END IF;
    IF b < c THEN
      INSERT INTO test.t3 VALUES (a,b);
    ELSE
      INSERT INTO test.t3 VALUES (a,c);
    END IF;
  END LOOP;
  CLOSE cur1;
  CLOSE cur2;
END;
```

Eis alguns comandos envolvidos:

CLOSE O comando `CLOSE cursorname` encerra o processamento de um cursor previamente aberto. Um erro ocorre se o cursor não estiver aberto. Entretanto, se ele não for explicitamente fechado, ele o será no final do bloco `BEGIN... END` no qual ele foi declarado.

DECLARE O formato é `DECLARE nome CURSOR FOR comandoSELECT`. Ele declara um cursor e o associa a um SELECT que vai trazer as linhas que formarão o cursor e que serão percorridas depois. O comando para recuperar as linhas do cursor é `FETCH`. O número de colunas recuperadas pelo comando SELECT deve coincidir com o número de variáveis de saída definidas no comando FETCH. O comando SELECT não pode conter a cláusula INTO. A declaração do cursor deve aparecer antes das declarações de *handler* e depois das declarações de variáveis e de condições.

FETCH O fomato é `FETCH [[NEXT] FROM ] cursorname INTO var1 [, var2]...` Este comando traz a próxima linha do comando SELECT assciado ao cursor (que deve estar aberto) e avança o apontador do cursor. Se a linha existe, as colunas da tabela são associadas às variáveis citadas. Se nenhuma linha mais há, volta uma condição de `NO DATA` que pode ser recuperada via um *handler* ou pela condição `NOT FOUND`.

OPEN Com formato `OPEN cursorname` este comando abre um cursor previamente declarado

DECLARE ... CONDITION Declara uma condição de erro que pode ser referida à frente em um `DECLARE ... HANDLER`

DECLARE ... HANDLER Continuação da declaração de manuseio de condições. Veja o exemplo e a literatura.

BEGIN... END Esta construção é usada para escrever comandos compostos dentro de programas armazenados (procedimentos, funções, triggers e eventos). Seu formato é

```
[rótulo-início:] BEGIN
   [lista-comandos]
END [rótulo-fim]
```

Eventualmente, pode ser necessário reescrever o delimitador, através do comando DELIMITER. Os rótulos são nomes de até 16 caracteres. Eles podem aparecer também nos comandos LOOP e END LOOP, REPEAT e END REPEAT, além de WHILE e END WHILE. O comando DECLARE (de variáveis, de condições, de *handler*s e de cursores) só é permitido dentro de um BEGIN... END e deve ser o primeiro do bloco.

☞ Para você fazer

Defina um banco de dados de nome D94 e crie duas tabelas: A primeira, chamada CLIE, deve ter os atributos CLIERGCL, decimal (4) deve conter o RG (fictício, só com 4 números – para ficar fácil de digitar e testar), além de CLIENOCL (nome, com char(15)), CLIEENCL (endereço, com char(20)), além de CLIEMUCL (município com 15 caracteres). Outra tabela deve ser TRAN (transações), formada pelos seguintes atributos: TRANRGCL decimal(4) com o RG dos clientes que efetuaram transações. Depois TRANDATR do tipo DATE com a data da transação, e TRANVATR, decimal (8,2) contendo o valor da transação. Você deve criar estas duas tabelas e colocar os valores constantes do arquivo D94-001.myd que está no AVA. A seguir defina um programa armazenado que gere uma carta para cada um dos RGs que estão no arquivo CLIE As linhas da carta devem ser criadas numa tabela chamada WORA, que contém linhas de 80 caracteres. A correção do código gerado deve ser vista fazendo um SELECT * FROM WORK que deve imprimir as linhas válidas da carta. Devolva todo o código gerado anexo a esta folha (impresso ou manuscrito, você decide) para a correção. Não esqueça de colocar o código do exercício e do aluno – anexos ao código de barras. A carta deve conter:

```
Curitiba, (data de hoje)
Prezado(a) Sr(a) (nome do cliente)
(endereço)
(município)

Tem a presente a finalidade de informar sobre
as seguintes transações:
(data) (valor)
(...)  (...)
VALOR TOTAL (total dos valores)
```

Uma possível solução

```
create database if not exists d94;
use d94;
drop table clie;
drop table tran;
drop table wora;
create table if not exists clie (cliergcl
   decimal(4) primary key, clienocl char(15),
   clieencl char(20), cliemucl char(15));
create table if not exists tran (tranrgcl
   decimal(4),trannutr decimal(2), trandatr
   date, tranvatr decimal(8,2));
create table if not exists wora
   (worklinh char(80) );
insert into clie values(123,'joao da silva',
   'rua 7 n. 4', 'curitiba');
insert into clie values(124,'maria da silva',
   'rua XV n. 4', 'curitiba');
insert into clie values(125, 'andre zebra',
   'rua 21 n. 8', 'londrina');
insert into tran values(123,01,'2016-08-29'
   ,100.00);
insert into tran values(123,02,'2016-08-27'
   ,200.00);
insert into tran values(123,03,'2016-08-25'
   ,400.00);
insert into tran values(124,01,'2016-08-29'
   ,1000.00);
insert into tran values(125,10,'2016-08-20'
   ,200.00);
insert into tran values(125,11,'2016-08-21'
   ,800.00);
insert into tran values(125,12,'2016-08-21'
   ,1000.00);
select * from clie;
select * from tran;
select * from wora;
delimiter //
drop procedure if exists carta2//
create procedure carta2()
begin
   declare aa decimal(4);
   declare nome char(30);
   declare ende char(50);
   declare muni char(30);
   declare feito int default false;
declare cursor2 CURSOR for select cliergcl,
   clienocl, clieencl, cliemucl from clie;
declare continue handler for not found set
   feito = true;
open cursor2;
read_loop2: loop
fetch cursor2 into aa,nome,ende,muni;
if feito then leave read_loop2;
end if;
insert into wora values
   (concat('Curitiba ',curdate()));
insert into wora values
   (concat('Prezado ',nome));
insert into wora values (ende);
insert into wora values (muni);
insert into wora values ('temos o prazer de
   informar as seguintes transacoes');
insert into wora values ('que existem em
   nossos arquivos:');
call carta1(aa);
end loop;
close cursor2;
end//

drop procedure if exists carta1//
create procedure carta1(aa decimal(4))
begin
 declare soma decimal(10,2);
 declare a date;
 declare b decimal(8,2);
 declare done int default false;
 declare cursor1 cursor for select trandatr,
    tranvatr from tran where aa=tranrgcl;
 declare continue handler for not found
    set done = true;
 set soma = 0;
 open cursor1;
 read_loop: loop
    fetch cursor1 into a,b;
 if done then leave read_loop;
 end if;
 set soma = soma + b;
 insert into wora values
    (concat(a,' ',format(b,2)));
 end loop;
 close cursor1;
 insert into wora values(concat('Total ',soma));
 insert into wora values('****************');
end//
delimiter ;
```

Após definir isto tudo, salvando com o nome de carta.sql (por exemplo), há que se fazer `source ...carta.sql;` e depois basta chamar `call carta2;` e finalmente recuperar-se-á o conteúdo das cartas fazendo `select * from wora;`.

Descubra 3 coisas:

1. Quantas cartas foram geradas ?
2. Qual o valor da carta com o maior valor ?
3. Qual o valor da carta com o menor valor ?

1	2	3

- 1 - /