

Cursores

1.	Cursores.....	3
1.1	Cursores.....	3
1.1.1	Sentenza DECLARE ... CURSOR.....	3
1.1.2	Sentenza OPEN	3
1.1.3	Sentenza FETCH	4
1.1.4	Sentenza CLOSE	4
1.1.5	Utilización de cursores	5
1.1.6	Cursores e sentenzas preparadas	7
2.	Tarefas.....	9
2.1	Tarefa 1. Identificar os pasos e sentenzas para utilizar cursores	9
	Solución	9
2.2	Tarefa 2. Utilizar cursores en programas almacenados	9
	Solución	9

1. Cursores

1.1 Cursores

Os cursores serven para procesar fila a fila o resultado dunha consulta SELECT, dentro dos programas almacenados. Cando a sentenza SELECT devolve unha única fila, pódense asignar os valores das columnas a variables utilizando a sentenza SELECT ... INTO, e non é necesario o uso de cursores.

Estas estruturas só deben utilizarse cando non se teña outra forma de manipular conxuntos de datos cunha soa sentenza.

Cada SXBDR ten a súa forma de declarar un cursor e incluso os seus propios tipos de cursores (Oracle PL/SQL, Transact, etc).

MySQL permite utilizar cursores dende a versión 5.0. Os cursores en MySQL son de só lectura, é dicir, pódese ler a información que conteñen, pero non se pode escribir neles.

Outra característica dos cursores en MySQL é que a información que conteñen só pode ser procesada de maneira secuencial fila a fila, empezando pola primeira fila, e non é posible acceder a unha fila directamente sen pasar polas anteriores. A cada fila pódese aplicar un bloque de sentenzas SQL. Para utilizar cursores en MySQL hai que seguir catro pasos:

- Declaración.
- Apertura.
- Lectura.
- Peche.

1.1.1 Senteza DECLARE ... CURSOR

A sentenza DECLARE ... CURSOR permite declarar un cursor, asignarlle un nome, e relacionalo cunha sentenza SELECT. Sintaxe:

```
DECLARE nome_cursor CURSOR FOR sentenza_SELECT
```

O cursor debe declarase (definirse) antes de usarse. No proceso de declaración non se recuperan datos, simplemente asóciase a unha sentenza SELECT. A declaración dos cursores ten que ir sempre despois da declaración de variables e de condicións, pero antes da declaración de manipuladores. Nun programa almacenado pódense declarar máis dun cursor.

1.1.2 Senteza OPEN

A sentenza OPEN permite abrir un cursor para poder procesar a información resultante da consulta SELECT. Sintaxe:

```
OPEN nome_cursor
```

A operación de apertura ten que facerse despois de declarar o cursor e antes de facer ningunha operación con el. Esta operación recupera os datos que resultan de executar a sentenza SELECT que ten asociada.

Cando se abre o cursor, créase un punteiro que sinala á primeira fila de resultados. Este

punteiro é utilizado polo sistema para sinalar a fila que se vai a ler cando se execute unha operación de lectura.

1.1.3 Sentenza FETCH

A sentenza FETCH permite ler as filas obtidas pola consulta SELECT asociada a un cursor. Cada vez que se executa esta sentenza, móvense os valores das columnas correspondente a unha fila do conxunto de resultados, a un conxunto de variables e avanza o punteiro á fila seguinte. Sintaxe:

```
FETCH nome_cursor INTO nome_variable [, nome_variable ] ...
```

Despois da palabra INTO teñen que especificarse tantas variables como columnas devolve a sentenza SELECT e teñen que estar definidas previamente co mesmo tipo de dato que a columna da que vai recibir o valor.

A lectura de resultados é sempre secuencial. A primeira vez que se executa unha sentenza FETCH, despois de abrir o cursor, accédese á primeira fila obtida pola consulta, a segunda vez que se executa accédese á segunda fila, e así sucesivamente.

Para poder recorrer todas as filas que devolve a sentenza SELECT é necesario executar un número variable de veces a sentenza FETCH, en función do número de filas que se devolvan. Para facer isto, o lóxico é utilizar unha estrutura repetitiva (WHILE ou REPEAT) e incluír a sentenza FETCH dentro do bucle, que debe finalizar cando xa non queden filas do resultado por ler.

Cando xa non quedan filas por ler no cursor, e se executa unha sentenza FETCH, prodúcese un erro asociado ao código SQLSTATE '02000', ou ben á condición de erro 'NOT FOUND', polo que para establecer a condición para que finalice a execución do bucle hai que definir un manipulador asociado a ese código de erro.

Exemplo de manipulador para controlar o final do bucle:

```
declare continue handler for SQLSTATE '02000' set vFinal = 1;
```

Esta sentenza de declaración de manipulador indica que se ocorre o erro con código SQLSTATE '02000' non finalice a execución do programa almacenado (por ser tipo 'continue'), e que se lle asigne o valor 1 á variable *vFinal*, que ten que estar previamente declarada. Esta variable pode ser utilizada para establecer a condición para finalizar o bucle, é dicir, o bucle debe finalizar cando a variable tome o valor 1.

Outra forma de declarar o manipulador que produce o mesmo resultado cá anterior é:

```
declare continue handler for NOT FOUND set vFinal = 1;
```

1.1.4 Sentenza CLOSE

A sentenza CLOSE permite pechar un cursor e limpar os espazos de memoria que utiliza.

```
CLOSE nome_cursor
```

O cursor péchase automaticamente no momento que finaliza o programa almacenado que o contén, aínda que non se execute a sentenza CLOSE. Mentres o programa non finalice, un cursor pode ser pechado e aberto as veces que se necesite.

1.1.5 Utilización de cursores

O proceso de utilización de cursores require facer unha serie de operacións que son obrigatorias sempre, e polo tanto pódese facer un esqueleto coas sentenzas SQL necesarias para utilizar cursores.

Cando se utiliza MySQL Workbench para editar os procedementos almacenados que utilizan cursores, pódese crear un fragmento de código (*snippet*) e gardalo para utilizalo cada vez que se escribe o código dun cursor.

Exemplo de contido do snippet esqueleto_cursor:

```
begin
  -- Declaración de variables
  declare vFinal integer default 0;
  /* aquí decláranse tamén as variables para almacenar os datos contidos
  nas columnas da consulta*/
  -- Declaración do cursor
  declare nomeCursor cursor for
    sentenza_SELECT;
  -- Declaración do manipulador para controlar o final da lectura do cursor
  declare continue handler for sqlstate '02000' set vFinal=1;
  -- Abrir cursor: recupéranse os datos da consulta
  open nomeCursor;
  -- Bucle para procesar os datos asociados ao cursor, ata que non queden filas
  /*Sábase que non quedan filas coando se produce o erro código sqlstate '02000
  nese momento asígnaselle á variable vFinal o valor 1*/
  repeat
    -- Lectura dos datos dunha fila do cursor e asignación a variables
    fetch nomeCursor into lista_variables;
    if vFinal = 0 then
      -- Aquí van as sentenzas para procesar unha fila do cursor
    end if;
    -- Condición de final da execución do bucle: cando vFinal tome o valor 1
  until vFinal = 1 end repeat;
  -- Peche do cursor, para liberar memoria
  close nomeCursor;
end
```

Exemplo de utilización de cursores:

```
/*
u704exemplo_demoCursor1.sql

NOME RUTINA: traballadores.sp_demoCursor1 (procedemento)
DATA CREACIÓN: 23/11/2015
AUTOR: Grupo licenza 2015
TAREFA A AUTOMATIZAR:   - Utilizando un cursor, que conta os empregados agrupados por
                        departamento, actualizar a columna depEmpregados (número de
                        empregados do departamento) na táboa departamento no caso que
                        o número de empregados contados non coincida co contido actual
                        da columna depEmpregados.
                        - Contar o número de departamentos que foron actualizados.
PARAMETROS REQUERIDOS: - Non require parámetros.
RESULTADOS PRODUCIDOS: - Mostrar na pantalla a data na que se fai a actualización, e
                        o número de departamentos que foron actualizados.
                        - Columna depEmpregados actualizada na táboa departamento.

*/
drop procedure traballadores.sp_demoCursor1;
```

```

delimiter //
create procedure traballadores.sp_demoCursor1()
begin
-- Declaración de variables:
-- vDepartamento e vContador almacenan os valores das columnas da consulta
declare vDepartamento, vContador integer;
-- vFinal utilízase para controlar o momento que non hai máis filas
-- vModificados é un contador dos departamentos actualizados
declare vFinal, vModificados integer default 0;
-- Declaración de cursores
declare cursorEmpregados cursor for
select empDepartamento, count(*)
    from traballadores.empregado
    group by empDepartamento;
-- Declaración do manipulador para controlar o final da lectura do cursor
declare continue handler for sqlstate '02000' set vFinal=1;
-- Abrir cursor: recupéranse os datos da consulta
open cursorEmpregados;
-- Bucle para procesar os datos asociados ao cursor, ata que non queden filas
repeat
-- Lectura dos datos dunha fila do cursor e asignación a variables
fetch cursorEmpregados into vDepartamento, vContador;
if vFinal = 0 then
-- Procesamento dunha fila do cursor
if vContador <> (select depEmpregados
                from traballadores.departamento
                where depNumero=vDepartamento)
then
-- contar departamento modificado
set vModificados = vModificados + 1;
-- modificar os datos do departamento
update traballadores.departamento set depEmpregados=vContador
    where depNumero=vDepartamento;
end if;
end if;
-- Condición de final da execución do bucle: cando vFinal tome o valor 1
until vFinal = 1 end repeat;
-- Peche do cursor, para liberar memoria
close cursorEmpregados;
-- Mostra número de departamentos modificados
select now() as 'Data actualización', vModificados as 'Departamentos actualizados';
end
//
delimiter ;

```

Este procedemento almacenado utiliza o cursor *cursorEmpregados* asociado a unha consulta sobre a táboa *empregado* que devolve o número do departamento e o contador de empregados que traballan nese departamento. Cando se le unha fila do cursor, compárase o valor da columna *depEmpregados* da táboa *departamento* co valor que devolve o contador da consulta asociada ao cursor. Se non coinciden os valores, actualízase o contido da columna *depEmpregados* co resultado do contador, e contabilízase a actualización feita empregando unha variable tipo contador. Ao finalizar o proceso, móstrase a data e hora do sistema e o número de departamentos que foron actualizados.

Pódense executar as seguintes sentenzas para facer unha proba do funcionamento do procedemento e ver o resultado na zona *Result Grid* en Workbench.

```

update traballadores.departamento set depEmpregados=0;
call sp_demoCursor1();

```

Result Grid	Filter Rows:	Export:
Data actualización	Departamentos actualizados	
2015-11-23 13:04:47	9	

1.1.6 Cursores e sentenzas preparadas

A combinación do uso de cursores e de sentenzas preparadas proporciona aos administradores de bases de datos unha ferramenta moi potente para automatizar tarefas propias da administración.

Exemplo de cursor combinado con sentenzas preparadas:

```

/*
u704exemplo_demoCursor2.sql

NOME RUTINA: utilidades.analizartaboas (procedemento)
DATA CREACIÓN: 23/11/2015
AUTOR: Grupo licenza 2015
TAREFA A AUTOMATIZAR: - Analizar todas as táboas dunha base de datos que se pasa
                        como parámetro, utilizando a sentenza:
                        'analyze table nome_táboa'

PARAMETROS REQUERIDOS: - IN: nome da base de datos
RESULTADOS PRODUCIDOS: - Executar a sentenza 'analyze table nome_táboa' para cada
                        táboa.
                        - Saída por pantalla do resultado da análise.

*/

drop procedure if exists utilidades.analizartaboas;
delimiter //
create procedure utilidades.analizartaboas(pBaseDatos varchar(64))
begin
    -- Declaración de variables:
    -- vTaboa almacena os nomes da táboas que se obteñen na consulta
    declare vTaboa varchar(64) character set utf8;
    -- vFinal utilízase para controlar o momento que non hai máis filas
    declare vFinal integer default 0;
    -- Declaración de cursores
    declare cursorTaboas cursor for
        select table_name
        from information_schema.tables
        where table_schema=pBaseDatos;
    -- Declaración de manipuladores:
    -- Declaración do manipulador para controlar o final da lectura do cursor
    declare continue handler for sqlstate '02000' set vFinal=1;
    -- Abrir cursor: recupéranse os datos da consulta
    open cursorTaboas;
    -- Comprobar se existe a base de datos
    if (select count(*)
        from information_schema.schemata
        where schema_name = pBaseDatos) = 0
    then
        select 'A base de datos non existe';
    else
        -- No caso de que exista a base de datos:
        -- Bucle para procesar os datos asociados ao cursor, ata que non queden filas
        repeat
            -- Lectura dos datos dunha fila do cursor e asignación a variables
            fetch cursorTaboas into vTaboa;
            if vFinal = 0 then

```

```

-- Procesamento dunha fila do cursor: analizar a táboa
    set @sentenza = concat('analyze table ',pBaseDatos,'.',vTaboa);
    prepare analizar from @sentenza;
    execute analizar;

end if;
-- Condición de final da execución do bucle: cando vFinal tome o valor 1
until vFinal = 1 end repeat;
-- borrado da sentenza preparada, para liberar memoria
drop prepare analizar;
-- Peche do cursor, para liberar memoria
close cursorTaboas;
end if;
end
//
delimiter ;

```

Este tipo de procedementos almacenados utilizan a información do diccionario de datos para poder automatizar operacións sobre obxectos das bases de datos. No caso do exemplo execútase a sentenza ANALYZE TABLE para facer un mantemento dos índices e mellorar o rendemento das consultas. A execución desta sentenza produce o bloqueo da táboa mentres se fai o análise, e mostra como saída unha mensaxe informando do estado da operación. Como a sintaxe da sentenza non admite utilizar variables para poñer o nome da táboa, hai que recorrer ás sentenzas preparadas.

Exemplo de execución do procedemento almacenado:

```
call utilidades.analizartaboas('traballadores');
```

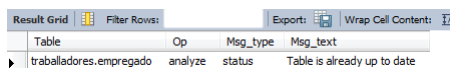


Table	Op	Msg_type	Msg_text
traballadores.empleado	analyze	status	Table is already up to date

Pódese automatizar a execución do procedemento almacenado, para que se execute cada certo tempo, creando un evento que fixe o calendario de execución.

Exemplo:

```

-- habilitar o planificador de eventos
set global event_scheduler = on;
-- crear o evento
delimiter //
create event traballadores.analizarTaboas
    on schedule every 1 month
    starts now() + interval 15 day
do
begin
    call utilidades.analizartaboas('traballadores');
end
//
delimiter ;

```


2. Tarefas

As tarefas propostas son as seguintes:

- Tarefa 1. Identificar os pasos para utilizar cursores.
- Tarefa 2. Utilizar cursores en programas almacenados.

2.1 Tarefa 1. Identificar os pasos e sentenzas para utilizar cursores

A tarefa consiste en nomear e explicar brevemente os pasos e sentenzas necesarios para poder utilizar cursores.

Solución

- Declarar: Asignar un nome ao cursor e asociarlle unha sentenza SELECT. Faise coa sentenza CREATE....CURSOR.
- Abrir: Recuperar datos da consulta. Faise coa sentenza OPEN.
- Ler: Ler fila a fila o resultado da consulta. Faise coa sentenza FETCH.
- Pechar: Pechar o acceso ao cursor e liberar memoria. Pode facerse coa sentenza CLOSE.

2.2 Tarefa 2. Utilizar cursores en programas almacenados

A tarefa consiste en crear un procedemento almacenado na base de datos *tendaBD*, ao que se lle pasen como parámetros de entrada dúas datas, e mostre o *id* do empregado, e a suma dos importes das vendas do empregado que fixo vendas por un maior importe entre esas datas.

Solución

```
/*
u7a4tarefa2.sql
-----
NOME RUTINA: tendabd.vendasEmpregado (procedemento)
DATA CREACIÓN: 23/11/2015
AUTOR: Grupo licenza 2015
TAREFA A AUTOMATIZAR:  - Utilizar un cursor, que selecciona o id dos empregados e
                        a suma dos importes de todas as vendas feitas por eles
                        entre dúas datas que se pasan como parámetros de entrada,
                        tendo en conta a información contida na táboa detalle_vendas.
                        - Compara a suma dos importe de todos os empregados, e
                        mostrar o id e o importe total das vendas do empregado que
                        fixo vendas por un importe maior entre esas datas.
PARAMETROS REQUERIDOS:  - data_inicio e data_final.
RESULTADOS PRODUCIDOS:  - Mostrar en pantalla o id e o importe total das vendas do
                        empregado que fixo vendas por un importe maior entre esas
                        datas.
*/
```

```

delimiter //
create procedure tendabd.vendasEmpregado(data_inicio date, data_final date)
begin
    -- Declaración de variables:
    -- vEmpId e vSumaVendas almacenan os valores das columnas da consulta
    -- vEmpIdMax e vSumaMax almacenan os datos do empregado co maior importe de vendas
    declare vEmpId, vEmpIdMax smallint;
    declare vSumaVendas, vSumaMax decimal(11,2) default 0;
    -- vFinal utilízase para controlar o momento que non hai máis filas
    declare vFinal, vModificados integer default 0;
    -- Declaración de cursores
    declare cursorVendasEmpregado cursor for
    select ven_empregado,
           sum(dev_cantidad*dev_prezo_unitario-
              (dev_cantidad*dev_prezo_unitario)*dev_desconto/100)
    from tendabd.vendas join tendabd.detalle_vendas on ven_id = dev_venta
    where ven_data between data_inicio and data_final
    group by ven_empregado;
    -- Declaración de manipuladores:
    -- Declaración do manipulador para controlar o final da lectura do cursor
    declare continue handler for not found set vFinal=1;
    -- Abrir cursor: recupéranse os datos da consulta
    open cursorVendasEmpregado;
    -- Bucle para procesar os datos asociados ao cursor, ata que non queden filas
    repeat
        -- Lectura dos datos dunha fila do cursor e asignación a variables: gárdase
        -- o id do empregado e a suma dos importes das vendas que fixo entre as datas
        fetch cursorVendasEmpregado into vEmpId, vSumaVendas;
        if vFinal = 0 then
            -- se o empregado actual ten mellores resultados que o mellor ata o momento
            -- gárdanse os seus datos nas variables vEmpIdMax e vSumaMax
            if vSumaVendas>=vSumaMax then
                set vSumaMax = vSumaVendas;
                set vEmpIdMax = vEmpId;
            end if;
        end if;
    until vFinal = 1 end repeat;
    -- Peche do cursor, para liberar memoria
    close cursorVendasEmpregado;
    -- Mostra o código e total acumulado das vendas do mellor empregado
    select vEmpIdMax as 'Código Empleado', vSumaMax as 'Total vendas';
end//
delimiter ;

```

Exemplo de execución do procedemento almacenado:

```
call tendabd.vendasEmpregado('2015-05-01', '2015-05-31');
```

Result Grid		Filter Rows:	Export:
Código Empleado	Total vendas		
42	111948.57		

Este procedemento está feito con fins educativos, para ver o funcionamento dun cursor. Pódese obter a información solicitada escribindo unha consulta complexa.