Introducción al SQL embebido

Bases de Datos II

Índice

1. Introducción

- SQL embebido está pensado para intercalar instrucciones en el código de un programa escrito en un lenguaje de alto nivel (C, C++, Fortran, Cobol..).
- El intercambio de información con el SGBD se realiza a través de variables del lenguaje, a las que se les denomina variables *anfitrionas*.
- Idea: escribir un programa que manipule una BD con el lenguaje SQL usando las estructuras de control y las variables del lenguaje correspondiente.

2. Estructura de un programa

```
Programa
  Declaración de variables,

    Declaración de variables anfitrionas,

    Fin de Declaración de variables anfitrionas,

  Fin de declaración de variables
 Comienzo del código,
     Instrucciones propias del lenguaje
     Conexión a la BD
     Instrucciones en SQL
     Desconexión de la BD
     Instrucciones propias del lenguaje
  Fin del código del programa
```

Acceso A la BD

3. Declaración de variables anfitrionas (1)

- Se deben declarar en una sección especial:

```
EXEC SQL BEGIN DECLARE SECTION;
    ..declaraciones de variables en C
EXEC SQL END DECLARE SECTION;
```

- Las variables deben tener un tipo apropiado al uso que se va a hacer de ellas.
- Se pueden utilizar los tipos de datos de C típicos y el pseudotipo VARCHAR para manipular cadenas:

VARCHAR primera[20] ⟷

```
Struct{
   unsigned short len;
   char arr[20];
} primera;
```


3. Declaración de variables anfitrionas (2)

- Ejemplo de declaración de variables:

```
EXEC SQL BEGIN DECLARE SECTION;
  float salario,comision;
  Short ind_var;
  VARCHAR nombre[30];
  int n_emp;
EXEC SQL END DECLARE SECTION;
```

- Variables indicadoras: están asociadas a variables anfitrionas y se declaran en la sección de variables anfitrionas:

```
:host_variable INDICATOR :ind_variable
```

:comision INDICATOR :ind_var ⟷ :comision:ind_var

Nota: variables anfitrionas y Variables indicadoras van Precedidas de ':'

3. Declaración de variables anfitrionas (3)

- Valores posibles para las variables indicadoras:

0	Éxito
-1	Valor nulo devuelto, insertado o actualizado.
>0	Valor devuelto truncado

```
EXEC SQL BEGIN DECLARE SECTION;
            int n_emp;
            float salario,comision;
            short ind_com;
EXEC SQL END DECLARE SECTION;
n_emp = 5;
EXEC SQL SELECT sal, comm INTO :salario,:comision:ind_com FROM emp
WHERE idemp =:n_emp;
if (ind_com == -1)  //Comision es NULL
            pagar =salario;
else
            pagar = salario + comision;
```


4. Sentencias SQL básicas

- Sentencias que devuelven una única tupla o ninguna.
- Son las sentencias de SQL: select, update, insert, delete:

```
EXEC SQL SELECT enombre, sal INTO :emp_name, :salario
            FROM emp WHERE idemp=:n_emp;
```

```
EXEC SQL INSERT INTO emp (idemp,enombre)
            VALUES (:n_emp,:salario)
```

```
EXEC SQL UPDATE emp SET sal = :salario, com = :comision
            WERE idemp = : n_emp;
```

```
EXEC SQL DELETE FROM emp WHERE iddepart = :n_depart;
```

4. Otras sentencias SQL

- Sentencias de conexión a la BD:

```
EXEC SQL CONNECT :nombreusuario IDENTIFIED BY :clave;

EXEC SQL CONNECT :nombreclave;
```

"esther/miclave"

- Cuando la consulta puede devolver más de una tupla es necesario utilizar cursores. Pasos:
  - Definición del cursor,
  - Apertura del cursor,
  - Manejo del cursor,
  - Cerrar el cursor.

4. Sentencias SQL : cursores

- Definición del cursor:

```
EXEC SQL DECLARE ncursor CURSOR for especificacion_cursor;
```

Sentencia SELECT

- Abrir cursor: cuando un cursor se abre su posición actual pasa a ser delante de la primera tupla:

```
EXEC SQL OPEN ncursor;
```

- FETCH: permite mover el cursor y leer la tupla en la que queda colocado

```
EXEC SQL FETCH ncursor INTO listadevariables;
```

Variables anfitrionas

- Cerrar el cursor:

```
EXEC SQL CLOSE ncursor;
```

Ejemplo con cursores

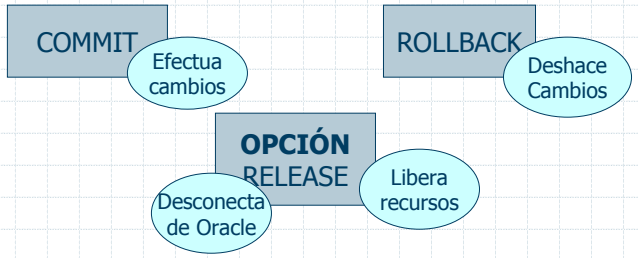
```
//defino un cursor
EXEC SQL DECLARE emp_cursor CURSOR FOR
   SELECT enombre FROM emp WHERE idemp = :nemp;

//Abro el cursor
EXEC SQL OPEN emp_cursor;
//Recojo datos y proceso
//PARA FINAL DEL BUCLE
EXEC SQL WHENEVER NOT FOUND DO break;
for(;;)
{ //Recojo datos
  EXEC SQL FETCH emp_cursor INTO :emp_nombre;
  //Añado carácter final de cadena
  emp_nombre.arr[emp_nombre.len]='\0';
  printf("%s",emp_nombre.arr);
}

EXEC SQL CLOSE emp_cursor;
EXEC SQL COMMIT WORK REALEASE;
```


5. Detalles a tener en cuenta en C++

- Cuatro consideraciones importantes:
  - Emisión de código del precompilador. ← CODE=CPP
  - Capacidad de análisis (PARSE). ← PARSE= NONE|PARTIAL|FULL
  - Extensión del fichero de salida. ← CPP_SUFFIX= CC || C
  - Localización de los ficheros de cabecera. ← SYS_INCLUDE

6. Soporte de transacciones

- Sentencias especiales para manejar transacciones:



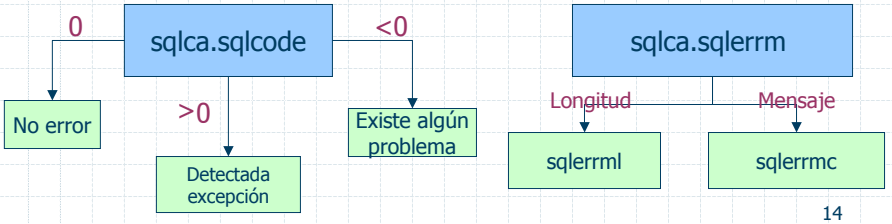

```
EXEC SQL COMMIT RELEASE;
EXEC SQL ROLLBACK RELEASE;
```


7. Control de errores

- Para el control de los errores producidos durante la ejecución de un programa se utiliza la estructura sqlca (Area de comunicación SQL).

```
#include <sqlca.h>
```

- Oracle actualiza la estructura después de cada nueva sentencia SQL. Para saber si se han producido errores, dos alternativas:
  - Comprobar la estructura sqlca explicitamente.
  - Comprobar implícitamente (sentencia WHENEVER).
- Los campos más utilizados de la estructura son:



7. Control de errores

- De forma explícita implica revisar el valor del campo sqlcode cada vez que se ejecute una sentencia SQL.
- De forma implicita implica la utilización de la sentencia WHENEVER:

```
EXEC SQL WHENEVER <CONDICION> <ACCIÓN>
```

Condiciones:
SQLWARNING
SQLERROR
NOT FOUND

Acciones:
CONTINUE
DO
DO BREAK
DO CONTINUE
GOTO etiqueta
STOP

7. Consejos para el control de errores

- La sentencia WHENEVER debe estar antes de cualquier sentencia SQL.
- Evitar bucles infinitos: si una sentencia WHENEVER SQLERROR GOTO va a una función que ejecuta una sentencia SQL esta puede fallar de nuevo y producirse un bucle.

```
EXEC SQL WHENEVER SQLERROR GOTO sql_error;

Sql_error:
        EXEC SQL WHENEVER SQLERROR CONTINUE;
        EXEC SQL ROLLBACK WORK RELEASE;
```

- Cuando un FETCH no devuelve datos:

```
EXEC SQL WHENEVER NOT FOUND DO break;
While(1)
{
         EXEC SQL FETCH...
}
EXEC SQL CLOSE mi_cursor;
```

8. Métodos de SQL dinámico

- Existen 4 métodos para definir sentencias SQL dinámicas:

1	Consultas no, no variables anfitrionas.	'delete from emp where dptno =20'
2	Consultas no, número conocido de variables anfitrionas.	'delete from emp where Idemp= :n_emp'
3	Consultas si, número conocido de elementos a seleccionar y v. Anfitrionas.	'select ename, empno from emp where deptno =:dept_number'
4	Consultas con un desconocido número de v. anfitrionas y elementos a seleccionar	'select <no conocido> from emp Where deptno = 20'

8. Ejemplos de los tres primeros métodos

Método 1

```
Cadena = "insert into mitabla values('test')";
EXEC SQL EXECUTE IMMEDIATE :cadena;
```

Variable anfitriona

Método 2

```
Cadena = "delete from emp where empno = :n"
EXEC SQL PREPARE sent_sql FROM :cadena;
Emp_no =3;
EXEC SQL EXECUTE sent_sql USING :emp_no;
EMP_NO=5;
EXEC SQL EXECUTE sent_sql USING :emp_no
```

Guardasitio

Variable anfitriona

Método 3

```
Consulta ="select idemp,trab from emp where sal < :salario";
EXEC SQL PREPARE sent_sql FROM :consulta;
EXEC SQL DECLARE mi_curso FOR sent_sql;
EXEC SQL OPEN mi_cursor USING :mi_salario;
EXEC SQL FETH mi_cursor INTO :no_emp, :trab;
EXEC SQL CLOSE mi_cursor
```

Guardasitio

Anfitriona entrada

Anfitriona: salida

```
#include <stdio.h>
#include <string.h>
#include <sqlca.h>
// Para el control de errores
void control_errores();
main()
{
  EXEC SQL BEGIN DECLARE OPTION;
    VARCHAR usuclave[30]; //como me conecto
    VARCHAR consulta[100]; //donde almaceno la consulta
    VARCHAR vnombre[30];  //variable anfitriona
    int vnodept; //otra variable anfitriona
  EXEC SQL END DECLARE OPTION;
//Fijo el control de errores antes de empezar con SQL
  EXEC SQL WHENEVER SQLERROR DO control_errores();
//Me conecto a la BD con un login/password
  strcpy(usuclave.arr,"esther/miclave"); //
  usuclave.len=strlen(usuclave.arr);
  EXEC SQL CONNECT :usuclave;
  puts("Conectado con Oracle"); //cout <<"Concectado con Oracle";
  //Guardo la consulta que se va a realizar en la variable consulta
  strcpy(consulta.arr,"select nombre from emp where deptno = :v1");
  consulta.len=strlen(consulta.arr);
```



```
//Preparo la sentencia para mandarla a Oracle
EXEC SQL PREPARE sentsql FROM :consulta;
//Esta consulta necesita cursor, por lo tanto defino el cursor
EXEC SQL DECLARE micursor CURSOR FOR senntsql;
//Abro el cursor
EXEC SQL OPEN micursor USING :vnodept;
//Control del final de datos consulta
EXEC SQL WHENEVER NOT FOUND DO break;
while(1) //Cuidado!! Bucle infinito
{
  EXEC SQL FETCH micursor INTO :vnombre;
  //Arreglo la variable VARCHAR vnombre para manipularla en C/C++
  vnombre.arr[vnombre.len]='\0';
  //Imprimo el resultado de la consulta.... o lo que quiera
  printf("%s\n",vnombre.arr); // cout << vnombre.arr;
}
//Imprimo el número de filas devueltas
printf("\nNumero de filas devuleltas: %d\n",sqlca.sqlerrd[2]);
EXEC SQL CLOSE micursor;
EXEC SQL COMMIT RELEASE; //Termino la transacción y me desconecto de Oracle
}
```

```
//Función de control de errores
void control_errores()
{
 //IMPRIMO MENSAJE DE ERROR:
 sqlca.sqlerrm.sqlerrmc[sqla.sqlerrm.sqlerrml]='\0';
 printf("%s\n",sqlca.sqlerrm.sqlerrmc); //cout << sqlca.sqlerrm.sqlerrmc
 EXEC SQL WHENEVER SQLERROR CONTINUE;
//Deshago cualquier posible cambio y me desconecto de Oracle
 EXEC SQL ROLLBACK RELEASE;
}
```