CAPÍTULO 10°: SQL EN MODO PROGRAMACIÓN

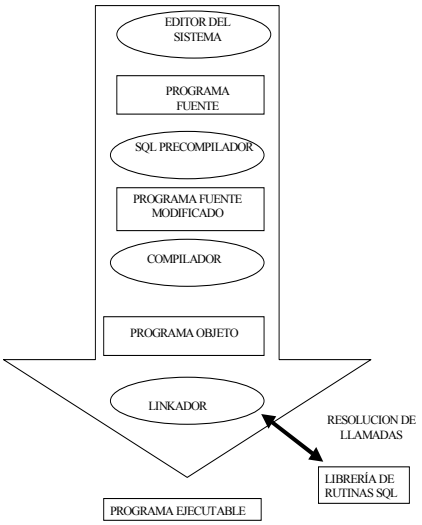
1 **SQL Embebido** .-

SQL también puede utilizarse para acceder a las tablas de una base de datos desde un programa de aplicación escrito en lenguajes como Pascal, Cobol, PL/I, Ensamblador, Basic, C , …

El lenguaje de programación que contiene sentencias SQL se denomina lenguaje *Anfitrión* y al SQL que forma parte del programa se llama lenguaje *Embebido*.

No existe limitación alguna respecto al tipo o número de sentencias SQL que se pueden embeber en un programa pudiéndose utilizar todas las sentencias y directivas vistas anteriormente tanto en Lenguaje de Definición de Datos (DDL) como en Lenguaje de Manipulación de Datos (DML).

Los programas con SQL embebido deben ser procesados por un *precompilador* antes de utilizar el compilador normal del lenguaje correspondiente. La misión del *precompilador* consiste en traducir las sentencias SQL en llamadas a rutinas de biblioteca que se encargan de realizar las acciones correspondientes sobre la Base de Datos. Por ello, dichas sentencias deben aparecer en el programa precedidas por un carácter o comando especial que permite al precompilador reconocerlas.




Igualmente, si dentro de la sentencia SQL embebida se necesitan utilizar variables definidas en el programa anfitrión, éstas se deben distinguir mediante un símbolo especial que las preceda. Estos símbolos o comandos especiales serán diferentes dependiendo del lenguaje

de programación que se utilice. Por este motivo, se ha optado aquí por una codificación en Pseudocódigo, usando EXEC-SQL y END-EXEC para delimitar las sentencias SQL embebidas y el símbolo ":" para distinguir las variables anfitrión dentro de ellas.

2 **Ejemplo de Aplicación** .-

Como ejemplo de aplicación se desarrollara un programa que permita cómodamente a un operador realizar una reserva de plaza para un vuelo y vender el billete correspondiente.

La base de datos sobre la que actuará la aplicación esta formada por tres tablas:

*Vuelos*

| **Num_Vuelo** | **Origen** | **Destino** | **Hora_Salida** | **Tipo_Avion** |
|---|---|---|---|---|

*Reservas*

| **Num_Vuelo** | **Fecha_Salida** | **Plazas_Libres** |
|---|---|---|

*Aviones*

| **Tipo** | **Capacidad** | **Longitud** | **Envergadura** | **Veloc_Crucero** |
|---|---|---|---|---|

Las operaciones podrían incluir obtener el número de plazas libres de un vuelo y fecha determinados, averiguar todos los vuelos que hay entre un origen y un destino dados, reservar una plaza, etc. Cada una de ellas podría ser una *opción de menú* de manera que cuando el operador seleccione una opción determinada, se ejecute una subrutina del programa encargada de esta tarea.

Así, si el operador selecciona la opción "Obtener número de plazas" se iniciaría una subrutina que deberá hacer lo siguiente:

- Pedir al operador las ciudades *origen* y *destino* del vuelo así como la fecha y hora de éste.
- Hacer la consulta correspondiente para recuperar el número de plazas.
- Presentar los resultados en pantalla.

El primer y tercer paso se realizarán mediante el lenguaje anfitrión elegido, en cuanto al segundo, la sentencia SELECT correspondiente sería:

```
SELECT RESERVAS.NUM_VUELO, PLAZAS_LIBRES
```

FROM VUELOS, RESERVAS

WHERE ORIGEN = :origen AND DESTINO = :destino AND FECHA_SALIDA = :fecha

AND HORA_SALIDA = :hora AND RESERVAS.NUM_VUELO = VUELOS.NUM_VUELO

Las variables :origen, :destino, :fecha y :hora son los parámetros de entrada de la sentencia SELECT y contendrán los valores leídos en el apartado 1. Están precedidas por el signo ":" como se ha indicado anteriormente.

El resultado de la consulta son dos valores RESERVAS.NUM_VUELO y PLAZAS_LIBRES. A diferencia de lo que ocurre con SQL interactivo, el resultado de un SELECT de SQL embebido no se muestra automáticamente en pantalla sino que debe ser recogido por variables del programa en el que se encuentra inmerso. Estas variables se especifican en la cláusula INTO (exclusiva de SQL embebido) y corresponden a los parámetros de salida de la consulta, esto es, a los valores que devuelve.

La cláusula INTO va inmediatamente antes de la cláusula FROM dentro de una sentencia SELECT. En ella se especifican las variables a las que se deben asignar los valores devueltos.

Debe haber tantas variables como columnas se recuperen en la sentencia SELECT. En el caso anterior, la sentencia SELECT quedaría así:

SELECT RESERVAS.NUM_VUELO, PLAZAS_LIBRES

INTO :vuelo, :plazas

FROM VUELOS, RESERVAS

WHERE ORIGEN = :origen AND DESTINO = :destino AND FECHA_SALIDA = :fecha

AND HORA_SALIDA = :hora AND RESERVAS.NUM_VUELO = VUELOS.NUM_VUELO

Esto significa que el valor devuelto para la columna RESERVAS.NUM_VUELO debe ser asignado a la variable :vuelo y el de la columna PLAZAS_LIBRES a la variable :plazas.

La subrutina completa, en Pseudocódigo tendría el aspecto siguiente:

VARIABLES

Origen, Destino, Fecha, Hora, Vuelo : STRING

```
    Plazas : ENTERO

COMIENZO
    ESCRIBIR "Introduzca la ciudad de Origen:"
    LEER Origen
    ESCRIBIR "Introduzca la ciudad de Destino:"
    LEER Destino
    ESCRIBIR "Introduzca la Fecha:"
    LEER Fecha
    ESCRIBIR "Introduzca la Hora:"
    LEER Hora
EXEC-SQL
    SELECT RESERVAS.NUM_VUELO, PLAZAS_LIBRES
    INTO :Vuelo, :Plazas
    FROM VUELOS, RESERVAS
    WHERE ORIGEN = :Origen AND DESTINO = :Destino AND FECHA_SALIDA = :Fecha
    AND HORA_SALIDA = :Hora AND RESERVAS.NUM_VUELO = VUELOS.NUM_VUELO
END-EXEC
    ESCRIBIR "Número de vuelo: ", Vuelo
    ESCRIBIR "Número de Plazas Libres: ", Plazas
FIN
```

Debe tenerse en cuenta las siguientes consideraciones:

.- Las variables del lenguaje utilizado aparecen precedidas del símbolo ":" *exclusivamente* dentro de las sentencias SQL pero no en el resto del código.

.- Los tipos de las variables han de ser compatibles con los tipos de datos de las columnas cuyos valores van a recoger. Son particularmente importantes las funciones de conversión de tipos integradas en los lenguajes anfitrión utilizados, para establecer dicha compatibilidad.

.- La solución expresada anteriormente sólo es válida en el caso en que exista *una única fila que satisfaga las condiciones impuestas por la cláusula WHERE* ya que las variables anfitrión sólo pueden recibir un único valor, dado que son variables simples.

Si el resultado de la cláusula WHERE es de más de una fila, es preciso definir una nueva estructura de datos que almacene un número de filas a priori indeterminado. Esta nueva estructura es el *CURSOR* y funciona como un puntero que apunta a cada fila resultado.

3 **Manejo de Cursores** .-

El manejo del cursor se realiza en cuatro etapas:

1.- *Declarar el cursor.-* Como cualquier otro tipo de variable un cursor necesita ser descrito. Para ello se le asocia a la sentencia SQL que lo va a utilizar. La sintaxis es:

DECLARE <nombre-cursor> CURSOR FOR <sentencia-Select>

2.- *Abrir el cursor.-* El Sistema Gestor de Base de Datos se encarga de construir la tabla resultante de la consulta asociada al cursor. El cursor queda apuntando a una posición anterior a la primera fila. La sentencia requerida es:

OPEN <nombre-cursor>

3.- *Recuperar la fila.-* El SGBD avanza el cursor una fila y se la devuelve al programa dentro de las variables anfitrión que se especifiquen. Esta operación se ejecutará cuantas veces sea necesario hasta que se recuperen todas las filas. La sentencia a utilizar es:

FETCH <nombre-cursor> INTO <lista-de-variables>

Cuando se realiza un FETCH y no quedan mas filas que recuperar la sentencia actualiza una variable especial, llamada SQLCODE asignándole el valor de 100. Esto permite detectar el final del cursor.

4.- *Cerrar el cursor.-* Cuando se ha recuperado la última fila del cursor hay que cerrarlo:

CLOSE <nombre-cursor>

Como ejemplo de manejo de cursores se desarrollará una subrutina correspondiente a otra opción de menú de la aplicación anterior: Aquella que permite averiguar todos los vuelos que hay entre un origen y un destino dados. Las acciones necesarias para la subrutina son:

1.- Pedir al operador el origen y el destino.

2.- Realizar la consulta correspondiente.

3.- Recuperar la primera fila seleccionada.

4.- Finalizar si la fila pedida no existe,

5.- Escribir la fila recuperada en la pantalla.

6.- Recuperar la siguiente fila.

7.- Volver al paso 4.

8.- Finalizar

```
VARIABLES

Origen, Destino, Hora, Vuelo : STRING

EXEC-SQL

    DECLARE Lista_Vuelos CURSOR FOR

        SELECT Num_Vuelo, Hora_Salida

        FROM VUELOS

        WHERE Origen = :Origen AND Destino = :Destino

END-EXEC

COMIENZO

ESCRIBIR "Introduzca la Ciudad de Origen: "

LEER Origen

ESCRIBIR "Introduzca la Ciudad de Destino: "
```

LEER Destino

EXEC-SQL OPEN Lista_Vuelos END-EXEC

EXEC-SQL FETCH Lista_Vuelos INTO :Vuelo, :Hora END-EXEC

MIENTRAS SQLCODE <> 100 HACER

ESCRIBIR Vuelo, Hora

EXEC-SQL FETCH Lista_Vuelos INTO :Vuelo, :Hora END-EXEC

FIN MIENTRAS

EXEC-SQL CLOSE Lista_Vuelos END-EXEC

FIN

4 **Utilización de sentencias SQL de actualización** .-

Como ejemplo de uso de sentencias de actualización con SQL embebido se desarrolla una subrutina que realice reservas de plazas. Dicha subrutina deberá:

1.- Pedir al operador el número de vuelo, la fecha y el número de billetes a reservar.

2.- Obtener el número de plazas libres para dicho vuelo y fecha

3.- Realizar la reserva, si hay suficientes plazas, emitiendo en caso contrario un mensaje.

El Pseudocódigo de la subrutina podría ser:

VARIABLES

Vuelo, Fecha : STRING

Billetes, Plazas : ENTERO

COMIENZO

ESCRIBIR "Introduzca el Número de Vuelo: "

LEER Vuelo

ESCRIBIR "Introduzca la fecha: "

```
        LEER Fecha
        ESCRIBIR "Introduzca el Número de Billetes Solicitados: "
        LEER Billetes
    EXEC-SQL
        SELECT Plazas_Libres INTO :Plazas
                FROM RESERVAS
                WHERE Num_Vuelo = :Vuelo AND Fecha_Salida = :Fecha
    END-EXEC
    SI Plazas >= Billetes ENTONCES
        EXEC-SQL
                UPDATE RESERVAS
                        SET Plazas_Libres = :Plazas - :Billetes
                        WHERE Num_Vuelo = :Vuelo AND Fecha_Salida = :Fecha
        END-EXEC
        ESCRIBIR "Reserva Realizada"
    SI NO
        ESCRIBIR "No hay Suficientes Plazas para Reservar"
    FIN SI
    FIN
```

Con los ejemplos anteriores se han mostrado las principales características y las aplicaciones mas habituales de SQL embebido. La inclusión de otras sentencias SQL dentro de un programa de aplicación se realiza de forma similar.

---