Manual del TESQL (Embedded SQL)

BASE100

BASE 100, S.A.
www.base100.com

Índice

1. INTRODUCCIÓN 3

2. EL COMANDO TESQL 4

3. FICHEROS CABECERA 5

4. VARIABLES HOST 6

5. INSTRUCCIONES TSQL EMBEBIDAS EN C 8

5.1 SINTAXIS DE UN PROGRAMA EMBEDDED C 8
5.2 ESTRUCTURAS DE DATOS ASOCIADAS A LAS INSTRUCCIONES TSQL 8
5.2.1 Estructura sql_ca 8
5.2.2 Estructura sql_da 9

6. INSTRUCCIONES TSQL DE MANTENIMIENTO DINÁMICO. USO DE LA ESTRUCTURA SQL_DA 10

6.1 SENTENCIAS SELECT NO PARAMETRIZADAS. MANEJO DE CURSORES 10
6.2 SENTENCIAS SELECT PARAMETRIZADAS 11
6.3 SENTENCIAS PARAMETRIZADAS DISTINTAS DE SELECT 12

7. LISTA DE INSTRUCCIONES TSQL 13

7.1 DE EJECUCIÓN DIRECTA 13
7.1.1 Mantenimiento de datos 13
7.1.2 Definición de datos 13
7.1.3 Privilegios para acceder a los datos 14
7.2 DE MANTENIMIENTO DEL CURSOR 14
7.3 DE MANTENIMIENTO DINÁMICO 14
7.4 LA SENTENCIA ASSUME 14

8. ANEXO. LA LIBRERÍA LESQL 15

8.1 UTILIDADES CON CHAR 15
8.2 MANEJO DE MENSAJES 17
8.3 OPERACIONES CON DECIMALES 18
8.4 UTILIDADES CON STRINGS PARA TSQL 20
8.5 MANEJO DE FECHAS 22
8.6 MANEJO DE VARIABLES MONEY 24
8.7 CONVERSIONES CON DECIMALES (DEC_T) 25

© Copyright BASE 100, S.A. Todos los derechos reservados. Ninguna parte de este documento puede ser reproducida ni transmitida por medio alguno sin permiso previo por escrito del titular del copyright. Todos los productos citados en este documento son marcas registradas o marcas comerciales registradas de sus respectivos propietarios.

1. Introducción

El paquete ESQL/C (Embedded SQL) es un software orientado a la programación de aplicaciones en lenguaje C que necesiten el manejo de bases de datos. Está compuesto por el preprocesador “t_esql”, una librería de funciones C y ficheros cabecera, que permiten:

- Incluir instrucciones TSQL en programas C.
- Realizar conversiones y manipulaciones con los tipos DECIMAL, DATE, MONEY y CHAR.
- Utilizar las funciones de utilidad del lenguaje C.

2. El comando tesql

Una vez que se tiene el programa fuente escrito en C con sus instrucciones TSQL, el comando **tesql** permite compilarlo para generar el código ejecutable. Para ello, lo que hace este comando es llamar primero a un preprocesador ("t_esql") que genera un código C listo para compilar, y a continuación llama al compilador C creando un fichero objeto que será montado con la librería tesql para obtener el ejecutable correspondiente.

La sintaxis del comando es la siguiente:

```
tesql [opciones] fichero1 [fichero2 [...]]
```

Las opciones posibles son las siguientes:

-l	No genera las directivas del compilador # line…
-C	Realiza sólo la fase de preprocesamiento, sin compilar ni montar.
-v	Permite ver la versión del preprocesador "t_esql", sin realizar preprocesamiento ni compilación.

Los ficheros fuentes para el tesql deberán tener la extensión ".ec".

Los ficheros "fichero1", "fichero2", etc., representan los módulos fuente sin preprocesar (*.ec) y/o los módulos fuente en C. El fichero ejecutable resultado de la compilación y el montaje llevará el nombre del último módulo especificado.

3. Ficheros cabecera

Los programas C que contienen sentencias ESQL/C pueden incluir los siguientes ficheros cabecera:

sqlca.h	Contiene la estructura donde se guardan los mensajes de error.
sqlda.h	Contiene las estructuras que almacenan los punteros a los valores y descripciones de variables definidas dinámicamente.
sqltypes.h	Contiene las definiciones correspondientes a los tipos C y TSQL.
decimal.h	Contiene la definición del tipo dec_t, que permite declarar variables compatibles con los tipos DECIMAL y MONEY del TSQL.

La sintaxis para incluir estos ficheros es:

```
$ include    sqlca;
$ include    sqlda;
$ include    sqltypes;
# include    <decimal.h>
```

El preprocesador ESQL/C lee el nombre del fichero, lo procesa y lo incluye en el fichero fuente C que se obtiene a la salida.

4. Variables host

Las variables host son variables C normales que se usan en las instrucciones TSQL. Cuando se usan dentro de una instrucción TSQL deben ir precedidas por el carácter **"$"**.

Usaremos este término tanto cuando estas variables vayan a recoger un valor de una sentencia SELECT mediante la cláusula INTO…, como cuando se utilicen para pasar valores al SQL en una cláusula USING…

En este último caso, las variables host serán sustituidas por el SQL en las posiciones donde aparezca un signo "?".

Las variables host son declaradas como variables C normales, excepto que la declaración debe estar precedida por el carácter **"$"**.

Ejemplo:

```
$ char  *pcar;            /* puntero a un carácter */
$ short hostshort;        /* variable tipo short */
$ long hostlong;          /* variable tipo long */
```

Estas variables host están declaradas con tipos de datos C y asociadas con tipos de datos TSQL, de manera que el manejo de los datos extraídos de la base de datos se realiza a través de estas variables. La relación entre los tipos C y los tipos TSQL es la siguiente:

TSQL	C
CHAR(n)	char array[n + 1]
SMALLINT	short int
INTEGER	long int
SERIAL	long int
DECIMAL	dec_t
MONEY	dec_t
DATE	long int

Ejemplos:

1)

```
$ char  nombre[16];
$ char  apellidos[16];
main()
{
$ database almacen;
$ select nombre, apellidos into $nombre, $apellidos
  from clientes where nombre = "JUAN";
```

2)

```
$ char * value1 = "A";
$ char * value2 = "Z";
$ char * query = "select nombre, apellidos from clientes
        where apellidos > ? and apellidos < ?";
  …
  …
$ prepare ident from query;
  …
  …
$ declare democursor cursor for ident;
  …
  …
$ open democursor using $value1, $value2;
  …
  …
```

5. Instrucciones TSQL embebidas en C

5.1 Sintaxis de un programa Embedded C

Un programa Embedded C es un programa C normal en el que se añade una serie de líneas que comienzan con el símbolo **"$"**. Estas líneas están compuestas por directivas include, declaraciones de variables y sentencias TSQL. La sintaxis de una sentencia TSQL es de este tipo:

```
$ instruccion_TSQL;
```

Las variables host pueden declararse tanto en el programa principal como dentro de funciones, teniendo en cuenta que para que una variable declarada dentro de un bloque (conjunto de instrucciones comprendido entre dos llaves "{" y "}") sea local a ese bloque se deberá poner el símbolo **"$"** delante de ambas llaves.

Ejemplo:

```
funcion()
${
$ short num;
$ database base1;
$ select clave into $num from tabla;
  …
$}
```

5.2 Estructuras de datos asociadas a las instrucciones TSQL

5.2.1 Estructura sql_ca

ESQL/C devuelve un código en la estructura sql_ca después de ejecutar cada instrucción TSQL, indicando cómo se ha realizado la instrucción.

La estructura sql_ca está definida en el fichero cabecera sqlca.h y se muestra a continuación:

```
struct sql_ca
        {
        long sqlcode;
        long sqlntuples;        /* número de tuplas procesadas */
        long sqliserrno;        /* error devuelto por el TISAM */
        long sqlofferr;         /* offset en la sentencia donde se produce el error */
        short sqlwarning;       /* si se ha truncado algo, en el caso de que el tamaño de la
                                variable host sea más pequeño que el campo de la base de datos */
                                /* si hay un update o delete sin la cláusula where */
                                /* si el número de variables en la cláusula INTO es distinto del
                                número de columnas de la SELECT */
        }
```

Los valores que puede tomar sqlcode son:

0	Operación realizada satisfactoriamente.
<0	Número de error devuelto por el servidor SQL.

Los valores que puede tomar sqlierr son:

0	Operación realizada satisfactoriamente por el TISAM.
n	Número de error devuelto por TISAM.

5.2.2 Estructura sql_da

Esta estructura permite recoger los datos obtenidos mediante una instrucción de la base de datos sin necesidad de utilizar variables host. Es utilizada normalmente en aquellas instrucciones en las que no se puede conocer de antemano qué campos son los que se van a extraer de la base de datos.

Veamos a continuación la definición de la estructura sql_da, que se encuentra en el fichero cabecera sqlda.h:

```
struct sql_var
    {
        short sqltype;          /* tipo de variable */
        short sqllen;           /* longitud en bytes */
        char *sqldata;          /* puntero a la posición de memoria donde TSQL almacenará el
                                   dato extraído en un FETCH */
        char *sqlname;          /* nombre de la columna de la cual se ha extraído el dato
                                   depositado en la variable host */
    }
struct sql_da
    {
        short sqld;             /* Número de variables host */
        struct sql_var *sqlvar; /* Puntero al vector de estructuras sql_var */
    }
```

Se definen además, para el uso de esta estructura, las siguientes macros:

```
#define     DESCR  struct sql_var
#define     DHEAD  struct sql_da
```

6. Instrucciones TSQL de mantenimiento dinámico. Uso de la estructura sql_da

Hay ocasiones en que las instrucciones TSQL definidas dinámicamente necesitan la estructura sql_da, como por ejemplo:

- Instrucciones SELECT donde se desconoce el número o el tipo de datos de los miembros de la lista de nombres de columnas.
- Instrucciones SELECT donde se desconoce el número o el tipo de datos de los parámetros de la cláusula WHERE.
- Instrucciones parametrizadas distintas de SELECT.

Estas instrucciones de mantenimiento dinámico permiten poner comodines "?" en las cláusulas de condición (por ejemplo, "campo1 > ?") para más tarde introducir estos valores en instrucciones "EXECUTE… USING" u "OPEN cursor USING…".

Ejemplo:

```
$ prepare qid for "select * from tabla where campo1 > ?";
    …
$ execute qid using $num;
```

6.1 Sentencias SELECT no parametrizadas. Manejo de cursores

En las sentencias SELECT descritas anteriormente, los valores devueltos por la query son puestos en las variables host de la cláusula INTO. Cuando se crea una sentencia SELECT dinámicamente después de compilar el programa, no se puede usar una cláusula INTO, ya que las variables host no están disponibles directamente. Si además se obtiene más de una tupla en la respuesta, se debe utilizar la estructura cursor junto con una estructura sql_da para poder ir tratando las tuplas una a una.

Los pasos a seguir son:

1. Declarar un puntero a una estructura sql_da (por ejemplo desc):

   ```
   DHEAD  *desc;
   ```

2. Hacer un PREPARE de la sentencia SELECT (que se encuentra almacenada en la tira string-spec) y darle un identificador a la sentencia (por ejemplo qid):

   ```
   $ prepare qid from string-spec;
   ```

3. Ejecutar la sentencia:

   ```
   $ describe qid into desc;
   ```

   Esta instrucción hace que "desc->sqlvar" apunte a una secuencia de estructuras "sql_var" parcialmente inicializadas; "desc->sqld" contiene el número de estructuras "sql_var". Por cada componente de la "select_list" se crea una estructura "sql_var" y se rellenan los campos "sqltype", "sqllen" (para datos de tipo CHAR) y "sqlname". Es decir, que las tuplas obtenidas de la base de datos se irán cargando en esta estructura.

4. Ubicar memoria para los valores esperados y asignar punteros a sqldata en cada estructura sql_var. Se deben asignar el tamaño adecuado dependiendo del tipo de datos, y en el caso de variables de tipo CHAR, la longitud de la variable.

5. Declarar un cursor para el identificador de la sentencia PREPARE.

```
$ declare q_cursor cursor for qid;
```

6. Abrir el cursor.

```
$ open q_cursor;
```

7. Recorrer el cursor con la sentencia FETCH de la siguiente manera:

```
$ fetch q_cursor using descriptor desc;
```

   Esta sentencia asigna valores a las variables apuntadas por los distintos “sqldata”.

8. Cerrar el cursor.

```
$ close q_cursor;
```

Ejemplo:

```
$ short cl;
  tstrcopy("select nombre from tabla where clave < 10", query);
$ prepare qid from $query;
$ declare democursor cursor for qid;
$ open democursor;
$ describe qid into demodesc;
    demodesc->sqlvar[0].sqldata = (char *) &cl;
    for (;;){
$ fetch democursor using descriptor demodesc;
        if (sqlca.sqlcode) break;
        printf("%d\n", cl);
        }
$ close democursor;
```

6.2 Sentencias SELECT parametrizadas

En este caso las variables de entrada de la cláusula WHERE son suministradas en tiempo de ejecución. La sentencia DESCRIBE sólo examina la “select_list”, pero no los parámetros de la cláusula WHERE. Si se conoce el número de estos parámetros y su tipo se debe utilizar un número igual de variables host cuando se abra el cursor.

```
$ open q_cursor using $h1,$h2,...;
```

6.3 Sentencias parametrizadas distintas de SELECT

El funcionamiento es prácticamente igual a las sentencias SELECT parametrizadas, excepto que se usa EXECUTE en lugar de OPEN, indicando los parámetros en variables host.

```
$ execute qid using $h1,$h2,...;
```

Previamente se habrá hecho un PREPARE de la sentencia utilizando el identificador “qid”.

7. Lista de instrucciones TSQL

7.1 De ejecución directa

7.1.1 Mantenimiento de datos

DELETE	Ejecuta la instrucción DELETE del tsql con la posibilidad de pasarle valores de variables de programa.
INSERT	Ejecuta la instrucción INSERT del tsql con la posibilidad de pasarle valores de variables de programa.
SELECT	Ejecuta una instrucción SELECT del tsql con la posibilidad de recoger valores en variables de programa. Esto se realiza con la cláusula INTO, cuya sintaxis es:

```
SELECT select_list [INTO $var1, $var2…, $varn] FROM…
```

Ejemplo:

```
$ select nombre into $char1 from tabla;
```

UPDATE	Ejecuta la instrucción UPDATE del tsql con la posibilidad de pasarle valores de variables de programa.

7.1.2 Definición de datos

CREATE DATABASE	Permite crear una base de datos.
DATABASE	Permite cambiar la base de datos en curso.
CLOSE DATABASE	Cierra la Base de datos en curso.
DROP DATABASE	Borra la base de datos indicada.
CREATE TABLE	Crea una tabla.
ALTER TABLE	Modifica la definición de una tabla.
RENAME TABLE	Cambia el nombre de una tabla.
DROP TABLE	Borra una tabla.
RENAME COLUMNA	Cambia el nombre de una columna de la tabla.
CREATE INDEX	Crea un índice para una tabla con una o más columnas.
DROP INDEX	Borra un índice previamente creado.
UPDATE STATISTICS	Actualiza los catálogos del sistema para determinar e insertar el número de filas que contienen las tablas de la Base de Datos en curso. El tsql utilizará posteriormente esta información para optimizar las búsquedas.

7.1.3 Privilegios para acceder a los datos

GRANT	Para conceder permisos.
REVOKE	Para retirar permisos.
LOCK TABLE	Restringe el acceso a una tabla o permite el acceso a otros usuarios sólo para operaciones de lectura.
UNLOCK TABLE	Devuelve el acceso a una tabla previamente bloqueada.

7.2 De mantenimiento del cursor

CLOSE nomcursor	Cierra un cursor abierto.
DECLARE nomcursor CURSOR FOR qid	Definición de un cursor.
FETCH nomcursor [INTO $var1, $var2, …, $varn]	Se posiciona y lee la siguiente fila del cursor.
OPEN nomcursor [USING $var1, $var2, …, $varn]	Abre un cursor previamente definido con la instrucción DECLARE.

7.3 De mantenimiento dinámico

EXECUTE qid USING $var1, $var2, …, $varn	Ejecuta una instrucción de tsql previamente preparada mediante PREPARE.
PREPARE qid FROM sentencia_sql	Prepara una instrucción de tsql para su posterior ejecución.
DESCRIBE qid INTO desc	Almacena en una estructura de tipo DHEAD, la información relativa a las variables host que se necesitan para una query previamente preparada mediante PREPARE.

Para una explicación detallada de cada una de estas sentencias, excepto los apartados B y C, consultar el Manual General del T2 v. 3.0.

7.4 La sentencia ASSUME

Esta sentencia comprueba en tiempo de preprocesamiento que existe la base de datos database_name y que los nombres de las columnas utilizadas corresponden con las definidas en las tablas de esa base de datos. Esta sentencia se ubica en la cabecera del programa.

Su sintaxis es la siguiente:

```
$ assume database database_name;
```

8. Anexo. La Librería lesql

8.1 Utilidades con CHAR

```
                                    allblank(s,len)
                                    charcmp (s, t, len)
                                    charcopy(s, len, t)
                                    charfill(c, len, s)
                                    charlen(s, len)
                                    ldchar(buf, len, s)
            doublelddbl(s)
            double                  ldfloat(s)
            long                    ldlong(s)
                                    stchar(s,buf,len)
                                    stdbl (d, s)
                                    stfloat(f, s)
                                    stlong(l, s)
```

allblank (char *s, short len)

Descripción: Comprueba si todos los caracteres de un char son blancos.

Parámetros	:	s	-	(I)	Char.
	:	len	-	(I)	Longitud del char.
Retorna	:	TRUE	:	si son blancos	
	:	FALSE	:	en caso contrario	

Las constantes TRUE y FALSE están definidas en “stddef.h”.

Ejemplo : allblank(s, 15);

charcmp(char *s, char *t, short len)

Descripción: Compara alfanuméricamente dos cadenas de “len” caracteres.

Parámetros	:	s	-	(I)	primera cadena
	:	t	-	(I)	segunda cadena
	:	len	-	(I)	longitud a comparar
Retorna	:	>0:	La primera es mayor que la segunda		
	:	<0:	La primera es menor que la segunda		
	:	=0:	Las dos son iguales		
Ejemplo	:	charcmp(str1, str2, 4)			
	:	con	str1 = "Mari " y str2 = "Marisa "		
	:	——> 0			

charcopy (char *s, short len, char *t)

Descripción: Copia los “len” primeros caracteres del char s en el char t.

Parámetros	:	s	-	(I)	“string” a copiar
	:	t	-	(O)	“string” donde copiar
	:	len	-	(I)	longitud del char
Ejemplo	:	charcopy("hola y adiós", 6, str2)			
	:	——> str2 = “hola y”			

charfill (char c, short len, char *s)

Descripción: Llena con el carácter “c” un char de longitud “len”.

Parámetros	:	c	-	(I)	carácter de relleno
	:	len	-	(I)	longitud del char
	:	s	-	(O)	char
Ejemplo	:	charfill(" ", 4, str1)			
	:	——> str1 = " "			

charlen (char *s, short len)

Descripción: Devuelve la longitud significativa del char.

Parámetros	:	s	-	(I)	cadena de caracteres
	:	len	-	(I)	longitud de la cadena
Retorna	:	Longitud de la cadena sin los “blancos” del final			
Ejemplo	:	charlen("María Sánchez ", 16)			
	:	——> 13			

ldchar (char *buf, short len, char *s)

Descripción: Copia el char “buf” en el string “s”, eliminando los caracteres blancos del final de “buf”.

Parámetros	:	buf	-	(I)	puntero al char
	:	len	-	(I)	longitud del char
	:	s	-	(O)	“string” de salida
Ejemplo	:	ldchar(str1, 15, s)			
	:	con	str1 = "Marisa Ruiz "		
	:	——>	s = "Marisa Ruiz"		

double lddbl(char *s)

Descripción: Carga un double desde un buffer.

Parámetros	:	s	-	(I)	Buffer
Retorna	:	el double con el valor cargado desde el buffer			
Ejemplo	:	lddouble(str)			

double ldfloat(char *s)

Descripción: Carga un float desde un buffer.

Parámetros	:	s	-	(I)	Buffer
Retorna	:	el double con el valor cargado desde el buffer			
Ejemplo	:	ldfloat(str)			

long ldlong(char *s)

Descripción: Carga un long desde un buffer.

Parámetros	:	s	-	(I)	Buffer
Retorna	:	el long con el valor cargado desde el buffer			
Ejemplo	:	ldlong(str)			

stchar (char *s, char *buf, short len)

Descripción: Carga un string en un char de longitud “len”, rellenando con blancos al final del char si es necesario.

Parámetros	:	s	-	(I)	string
	:	buf	-	(O)	puntero al char
	:	len	-	(I)	longitud del char
Ejemplo	:	stchar("Marisa Ruiz", str1, 15)			
	:	——>	str1 = "Marisa Ruiz "		

stdbl (double d, char *s)

Descripción: Almacena un double en un buffer.

Parámetros	:	d	-	(I)	double
	:	s	-	(O)	Buffer
Ejemplo	:	stdbl(4.6e95, str)			

stfloat (double f, char *s)

Descripción: Almacena un float en un buffer.

Parámetros	:	f	-	(I)	double (con el valor del float)
	:	s	-	(O)	Buffer
Ejemplo	:	stfloat(10.45, str)			

stlong (long l, char *s)

Descripción: Almacena un long en un buffer.

Parámetros	:	l	-	(I)	long
	:	s	-	(O)	Buffer
Ejemplo	:	stlong(10L, str)			

8.2 Manejo de mensajes

```
char *   getdblang()
char *   getmsgdir()
char *   gettransdir()
         msgpathname(s,sysmsg,pathname)
         tgetmsg(num, len, s)
         treadmsg(num, len, buf)
```

char *getdblang()

Descripción: Devuelve el valor de la variable de entorno de usuario DBLANG. Esta variable determina el directorio del que hay que tomar los mensajes y textos del sistema. Por defecto es el subdirectorio “msg” del directorio contenido en la variable TRANSDIR.

Retorna	:	String con el valor de DBLANG
Ejemplo	:	p = getdblang() … con DBLANG=""
	:	—> p = "msg"

char *getmsgdir()

Descripción: Devuelve el valor de la variable de entorno de MSGDIR.

Retorna	:	String con el valor de MSGDIR
Ejemplo	:	p = getmsgdir() … con MSGDIR=""
	:	—> p = ""

char *gettransdir()

Descripción: Devuelve el valor de la variable de entorno de usuario TRANSDIR. Esta variable indica qué directorio contiene los programas que componen el T2. Por defecto es “/usr/trans”.

Retorna	:	String con el valor de TRANSDIR
Ejemplo	:	p = gettransdir() … con TRANSDIR=""
	:	—> p = "/usr/trans"

msgpathname (char *s, short sysmsg, char *pathname)

Descripción: Devuelve el nombre con el path para un fichero de mensajes.

Parámetros	:	s	-	(I)	nombre del fichero (sin path)	
	:	sysmsg	-	(I)	1:	fichero en el directorio por defecto
					0:	fichero en otro directorio
	:	pathname-		(O)		nombre del fichero (con path)
Retorna	:	0	:	SUCCESS		
Ejemplo	:	msgpathname("jjj.oms", 1, buf)				
	:	—> buf = "/usr/trans/msg/jjj.oms"				

tgetmsg (short num, short len, char *s)

Descripción: Devuelve en la variable “s” el mensaje cuyo número es “num”, con una longitud de “len” caracteres. El mensaje devuelto acabará con el carácter “new line”.

Parámetros	:	num	-	(I)	Número de mensaje
	:	len	-	(I)	longitud
	:	s	-	(O)	“string” donde se guarda el mensaje.
Retorna	:	-1	:	FAIL	
	:	0	:	SUCCESS	
Ejemplo	:	tgetmsg(157, 60, str)			

treadmsg (short num, short len, char *s)

Descripción: Lee un mensaje eliminando el carácter “new line” final.

Parámetros	:	num	-	(I)	Número de mensaje
	:	len	-	(I)	longitud
	:	s	-	(O)	“string” donde leer el mensaje
Retorna	:	-1	:	FAIL	
	:	0	:	SUCCESS	
Ejemplo	:	treadmsg(157, 60, str)			

8.3 Operaciones con decimales

```
decadd(n1, n2, res)
deccmp(p, q)
deccopy(p, q)
decdiv(n1, n2, res)
decmul(n1, n2, res)
decsub(n1, n2, res)
lddec(buf, len, p)
stdec(p, len, s)
```

decadd (dec_t *n1, dec_t *n2, dec_t *res)

Descripción: Suma 2 decimales (dec_t) (res = n1 + n2).

Parámetros	:	n1	-	(I)	sumando 1
	:	n2	-	(I)	sumando 2
	:	res	-	(O)	resultado
Retorna	:	-1200	:	número demasiado grande	
	:	-1201	:	número demasiado pequeño	
	:	0	:	SUCCESS	
Ejemplo	:	decadd(&d1, &d2, &dres)			
Comentarios	:	ver dec_t_load(..)			

deccmp(dec_t *n1, dec_t *n2)

Descripción: Compara 2 decimales (dec_t).

Parámetros	:	n1	-	(I)	decimal 1
	:	n2	-	(I)	decimal 2
Retorna	:	1	:	n1 > n2	
	:	-1	:	n1 < n2	
	:	0	:	n1 == n2	
	:	-2	:	n1 o n2 es un valor nulo	
Ejemplo	:	deccmp(&d1, &d2)			

deccopy (dec_t *p, dec_t *q)

Descripción: Copia un decimal en otro.

Parámetros	:	p	-	(I)	decimal origen
	:	q	-	(0)	decimal destino
Ejemplo	:	deccopy(&d1, &d2)			

decdiv (dec_t *n1, dec_t *n2, dec_t *res)

Descripción: Divide 2 decimales (dec_t) (res = n1n2).

Parámetros	:	n1	-	(I)	numerador
	:	n2	-	(I)	denominador
	:	res	-	(O)	resultado
Retorna	:	-1200	:	número demasiado grande	
	:	-1201	:	número demasiado pequeño	
	:	-1202	:	intento de división por 0	
	:	0	:	SUCCESS	
Ejemplo	:	decdiv(&d1, &d2, &dres)			
Comentarios	:	ver dec_t_load(..)			

decmul (dec_t *n1, dec_t *n2, dec_t *res)

Descripción: Multiplica 2 decimales (dec_t) (res = n1 x n2).

Parámetros	:	n1	-	(I)	sumando 1
	:	n2	-	(I)	sumando 2
	:	res	-	(O)	resultado
Retorna	:	-1200	:	número demasiado grande	
	:	-1201	:	número demasiado pequeño	
	:	0	:	SUCCESS	
Ejemplo	:	decadd(&d1, &d2, &dres)			
Comentarios	:	ver dec_t_load(..)			

decsub (dec_t *n1, dec_t *n2, dec_t *res)

Descripción: Resta 2 decimales (dec_t) (res = n1 - n2).

Parámetros	:	n1	-	(I)	minuendo
	:	n2	-	(I)	sustraendo
	:	res	-	(O)	resultado
Retorna	:	-1200	:	número demasiado grande	
	:	-1201	:	número demasiado pequeño	
	:	0	:	SUCCESS	
Ejemplo	:	decsub(&d1, &d2, &dres)			
Comentarios	:	ver dec_t_load(..)			

lddec (char *buf, short len, dec_t *p)

Descripción: Carga de un buffer con longitud "len" un "dec_t".

Parámetros	:	buf	-	(I)	buffer
	:	len	-	(I)	longitud con la que se cargó el dec.
	:	p	-	(I)	puntero al decimal
Retorna	:	-1200	:	número demasiado grande	
	:	-1201	:	número demasiado pequeño	
	:	0	:	SUCCESS	
Ejemplo	:	lddec(buf+5, 10, &d1)			

stdec (dec_t *p, short len, char *s)

Descripción: Carga un decimal en un buffer con longitud len.

Parámetros	:	p	-	(I)	puntero al decimal
	:	len	-	(I)	longitud que disponemos en buf
	:	buf	-	(0)	buffer
Ejemplo	:	stdec(&d1, 10, buf+5)			

8.4 Utilidades con strings para TSQL

```
lowcase(s)
tatoi(s, pint)
tatol(s, plong)
tstrcat(s, t)
tstrcmp(s, t)
tstrcopy(s, t)
tstrlen(s)
tstrncat(s,n,t)
upcase(s)
```

lowcase (char *s)

Descripción: Convierte todos los caracteres de la cadena "s" a minúsculas.

Parámetros	:	s	-	(IO)	"string"
Ejemplo	:	lowcase(str);			

tatoi (char *s, short *pint)

Descripción: Convierte un número en formato ascii a un short.

Parámetros	:	s	-	(I)	cadena de caracteres de entrada
	:	pint	-	(O)	puntero al short resultado
Retorna	:	-1213	:	conversión errónea	
	:	-1215	:	valor fuera de rango	
	:	0	:	SUCCESS	
Ejemplo	:	tatoi("-130", &l)			

tatol (char *s, long *plong)

Descripción: Convierte un número en formato ascii a un long.

Parámetros	:	s	-	(I)	cadena de caracteres de entrada
	:	plong	-	(O)	puntero al long resultado
Retorna	:	-1213	:	conversión errónea	
	:	-1215	:	valor fuera de rango	
	:	0	:	SUCCESS	
Ejemplo	:	tatol("130000", &l)			

tstrcat (char *s, char *t)

Descripción: Concatena el string “s” al final del string “t”.

Parámetros	:	s	-	(I)	“string” que concatena
	:	t	-	(IO)	“string” al que concatena
Ejemplo	:	tstrcat(str1, str2)			
	:	con	str1 = "qué tal" y str2 = "hola "		
	:	——> str2 = "hola qué tal"			

tstrcmp (char *s, char *t)

Descripción: Compara alfanuméricamente dos cadenas de caracteres.

Parámetros	:	s	-	(I) primer “string”
	:	t	-	(I) segundo “string”
Retorna	:	>0	:	El primero es mayor que el segundo
	:	<0	:	El primero es menor que el segundo
	:	=0	:	Los dos son iguales
Ejemplo	:	tstrcmp("Mari ", "María ")		
	:	——> <0		

tstrcopy (char *s, char *t)

Descripción: Copia la cadena “s” en la cadena “t”.

Parámetros	:	s	-	(I)	“string” a copiar
	:	t	-	(O)	“string” donde copiar
Ejemplo	:	tstrcopy(str1, str2)			

tstrlen (char *s)

Descripción: Devuelve la longitud de una cadena de caracteres.

Parámetros	:	s	-	(I)	cadena de caracteres
Retorna	:	Longitud de la cadena			
Ejemplo	:	tstrlen("Marisa ")			
	:	——> 7			

tstrncat (char *s, short n, char *t)

Descripción: Concatena “n” caracteres de la cadena “s” al final de la cadena “t”.

Parámetros	:	s	-	(I)	“string” que concatena
	:	n	-	(I)	número de caracteres a concatenar
	:	t	-	(IO)	“string” al que concatena y resultado
Ejemplo	:	tstrncat(str1, str2, 5)			
	:	con	str1 = "qué tal" y str2 = "hola "		
	:	——> str2 = "hola qué t”			

upcase (char *s)

Descripción: Convierte todos los caracteres de la cadena “s” a mayúsculas.

Parámetros	:	s	-	(IO)	“string”
Ejemplo	:	upcase(str);			

8.5 Manejo de fechas

```
char *    getdbdate()
          getusdbdate(str)
          tatodte(str, pl)
          tdtedweek(l)
          tdtetoa(lng, str)
          tdtetofmt(lng, fmt, str)
          tdteto3i(l, mdy)
          tfmttodte(str, fmt, pl)
          t3itodte(mdy, pl)
          tleapyear(year)
          ttoday(pl)
```

Comentarios: Las fechas vendrán en distintos formatos: long, ascii, short [3]. Por defecto asociaremos tipo fecha = “long” (abreviatura dte).

char *getdbdate()

Descripción: Devuelve el valor de la variable de entorno de usuario DBDATE, que indica el formato que se desea utilizar para las fechas. En DBDATE se puede indicar:

- El orden del mes, día y año en una fecha.
- Si el año se mostrará con dos dígitos (Y2) o con cuatro (Y4).
- El separador entre mes, día y año.

El valor por defecto de DBDATE es DMY4/.

Retorna : “string” con el valor de DBDATE

Ejemplo : p = getdbdate() … con DBDATE=""

: —> p = “DMY4”

getusdbdate (char *str)

Descripción: Devuelve el valor de la variable de entorno de usuario DBDATE en formato de ‘using’.

Parámetros	:	str	-	(O)	formato ‘using’
Retorna	:	1218	:	error de conversión de fecha	
	:	0	:	SUCCESS	
Ejemplo	:	getusdbdate(str) … con DBDATE=MDY4			
	:	——> str = "mmdd/yyyy"			

tatodte (char *str, long *pl)

Descripción: Convierte una fecha en formato ascii a un DATE.

Parámetros	:	str	-	(I)	fecha en formato ascii
	:	pl	-	(O)	fecha en formato DATE
Retorna	:	1204	:	año incorrecto (t3itodte)	
	:	1205	:	mes incorrecto (t3itodte)	
	:	1206	:	día incorrecto (t3itodte)	
	:	1218	:	error de conversión de fecha	
	:	0	:	SUCCESS	
Ejemplo	:	tatodte("08-22-89", &ll)			

tdtedweek (long l)

Descripción: Devuelve el día de la semana de una fecha en formato DATE (long).

Parámetros	:	l	-	(I)	fecha
Retorna	:	0	:	Domingo	
	:	1	:	Lunes	
	:	…			
	:	6	:	Sábado	
Ejemplo	:	tdtedweek(&lfec)			

tdtetoa (long lng, char *pvres)

Descripción: Convierte una fecha de DATE (long) a ascii.

Parámetros	:	lng	-	(I)	Fecha en formato DATE (long)
	:	pvres	-	(O)	Fecha resultado en ascii
Retorna	:	1218	:	error de conversión de fecha	
	:	0	:	SUCCESS	
Ejemplo	:	tdtetoa(lfec, str);			

tdtetofmt (long lng, char *fmt, char *str)

Descripción: Convierte una fecha en formato DATE (long) a un “string” con un formato dado.

Parámetros	:	lng	-	(I)	fecha
	:	fmt	-	(I)	formato
	:	str	-	(O)	“string” resultado
Retorna	:	-1210	:	fecha incorrecta	
	:	0	:	SUCCESS	
Ejemplo	:	tdtetofmt(3L, "dd-mmm-yyyy", str)			
	:	—> en str "03-Jan-1900"			

tdteto3i (long l, short *mdy)

Descripción: Convierte una fecha en formato DATE (long) a un array de 3 short que contendrá el mes, día y año.

Parámetros	:	l	-	(I)	fecha en formato DATE (long)
	:	mdy	-	(O)	fecha en formato 3i (short [3])
Retorna	:	0	:	SUCCESS	
Ejemplo	:	tdteto3i(l, fec3i)			

tfmttodte (char *str, char *fmt, long *pl)

Descripción: Convierte un “string” fecha en un formato dado a una fecha en formato DATE (long).

Parámetros	:	str	-	(I)	“string” - fecha
	:	fmt	-	(I)	formato
	:	pl	-	(O)	fecha en DATE
Retorna	:	1204	:	año incorrecto	
	:	1205	:	mes incorrecto	
	:	1206	:	día incorrecto	
	:	0	:	SUCCESS	
Ejemplo	:	tfmttofmt("03-Jan-1900", "dd-mmm-yyyy", &l)			

t3itodte (short *mdy, long *pl)

Descripción: Convierte un array de 3 short que contienen el mes, día y año a un DATE (long).

Parámetros	:	mdy	-	(I)	fecha en formato 3i (short [3])
	:	pl	-	(O)	fecha en formate DATE (long)
Retorna	:	1204	:	año incorrecto	
	:	1205	:	mes incorrecto	
	:	1206	:	día incorrecto	
	:	0	:	SUCCESS	
Ejemplo	:	t3itodte(&fec3i, &l)			

tleapyear (short year)

Descripción: Comprueba si un año es bisiesto o no.

Parámetros	:	year	-	(I) año
Retorna	:	0	:	FALSE el año no es bisiesto
	:	1	:	TRUE el año si es bisiesto
Ejemplo	:	tleapyear(1984) — ret —> TRUE		

ttoday (long *pl)

Descripción: Devuelve la fecha del sistema en formato DATE (long).

Parámetros	:	pl	-	(O)	Resultado
Retorna	:	0	:	SUCCESS	
Ejemplo	:	ttoday(&lfec)			

8.6 Manejo de variables money

```
getdbmoney()
mnyformat(str, len, ps)
mnylpref()
mnylsuf()
mnyunformat(str, len, ps)
```

Comentarios: El tipo money es un decimal o “dec_t” con el formato dado en DBMONEY.

getdbmoney()

Descripción: Devuelve el valor de la variable de entorno de usuario DBMONEY inicializando las cuatro variables globales: mnyprefch (front characters), mnysufch (back characters), mnysepch (separator character) y mnyfrach (decimal character).

Longitud máxima DBMONEY = 7

Retorna	:	0	: SUCCESS
Ejemplo	:	getdbmoney() … con DBMONEY=,Pts	
	:	——>	mnyprefch =""
	:		mnysufch ="Pts"
	:		mnysepch ='.'
	:		mnyfrach =','

mnyformat (char *str, short len, char *ps)

Descripción: Devuelve el “string” money de entrada en formato DBMONEY.

Parámetros	:	str	-	(I)	“string” money
	:	len	-	(I)	longitud del string
	:	ps	-	(O)	“string” en DBMONEY
Retorna	:	0	:	SUCCESS	
Ejemplo	:	mnyformat("2345.78",7,str) … con DBMONEY=,Pts			
	:	——> str = "2345,78Pts"			

mnylpref ()

Descripción: Devuelve la longitud del “string” - prefijo especificado en el formato DBMONEY.

Retorna : longitud del prefijo

Ejemplo : mnylpref() … con DBMONEY=Pts,

: ——> 3

mnylsuf ()

Descripción: Devuelve la longitud del “string” sufijo especificado en el formato DBMONEY.

Retorna : longitud del sufijo

Ejemplo : mnylsuf() … con DBMONEY=,Pt

: ——> 2

mnyunformat (char *str, short len, char *ps)

Descripción: Devuelve el “string” money de entrada sin formato DBMONEY.

Parámetros	:	str	-	(I)	“string” money en DBMONEY
	:	len	-	(I)	longitud del string
	:	ps	-	(O)	“string” money
Retorna	:	0	:	SUCCESS	

Ejemplo : mnyunformat("Pts2345,78",10,str) … con DBMONEY=Pts,

: ——> str = "2345,78"

8.7 Conversiones con decimales (dec_t)

```
deczero(p)
tatodec(s, len, p)
tdbltodec(d, p)
tdectoa(p, len, right, s)
tdectora(p, len, right, s)
tdectodbl(p, pd)
tdectoflt(p, pf)
tdectoi(p, ip)
tdectol(p, pl)
tflttodec(f, p)
titodec(i, p)
tltodec(l, p)
```

deczero (dec_t *p)

Descripción: Carga un 0 en un decimal (dec_t).

Parámetros	:	p	-	(O)	puntero al decimal
Ejemplo	:	deczero(&dec)			

tatodec (char *s, short len, dec_t *p)

Descripción: Convierte un número ascii en decimal (dec_t).

Parámetros	:	s	-	(I)	puntero al ascii
	:	len	-	(I)	longitud total de “s”
	:	p	-	(O)	puntero al decimal de salida
Retorna	:	-1200	:	número demasiado grande	
	:	-1201	:	número demasiado pequeño	
	:	-1213	:	error de conversión carácter a decimal	
	:	0	:	SUCCESS	

Ejemplo : tatodec("1234.35e23", 10, &d1)

tdbltodec (double d, dec_t *p)

Descripción: Convierte un double en un decimal (dec_t).

Parámetros	:	d	-	(I)	double
	:	p	-	(O)	puntero al decimal
Retorna	:	-1200	:	número demasiado grande	
	:	-1201	:	número demasiado pequeño	
	:	0	:	SUCCESS	
Ejemplo	:	tdbltodec((double)2.34e114, &d1)			

tdectoa (dec_t *p, short len, short right, char *s)

tdectora (dec_t *p, short len, short right, char *s)

Descripción: Convierten un decimal (dec_t) a formato ascii. TDECTORA proporciona un decimal por defecto (right=-1) .

Parámetros	:	p	-	(I)	puntero al decimal
	:	len	-	(I)	longitud total en “s”
	:	right	-	(I)	número de decimales a la derecha de la coma.

Si es -1, el número de decimales serán los que indique la estructura dec_t.

	:	s	-	(O)	puntero al “string” de salida
Retorna	:	-1	:	FAIL (en “s” aparecerá relleno de “*”)	
	:	0	:	SUCCESS	
Ejemplo	:	tdectoa(&dec1, 10, 3, str)			
	:	tdectora(&dec1, 10, 3, str)			

tdectodbl (dec_t *p, double *pd)

Descripción: Convierte un decimal (dec_t) en un double.

Parámetros	:	p	-	(I)	puntero al decimal
	:	pd	-	(O)	puntero al double
Retorna	:	-1200	:	número demasiado grande	
	:	0	:	SUCCESS	
Ejemplo	:	tdectodbl(&dec1, &dbl1)			

tdectoflt (dec_t *p, double *pf)

Descripción: Convierte un decimal (dec_t) en un float.

Parámetros	:	p	-	(I)	puntero al decimal
	:	pf	-	(O)	puntero al float
Retorna	:	-1200	:	número demasiado grande	
	:	0	:	SUCCESS	
Ejemplo	:	tdectoflt(&dec1, &flt)			

tdectoi (dec_t *p, short *ip)

Descripción: Convierte un decimal (dec_t) en un short.

Parámetros	:	p	-	(I)	puntero al decimal
	:	ip	-	(O)	puntero al short
Retorna	:	-1200	:	número demasiado grande	
	:	0	:	SUCCESS	
Ejemplo	:	tdectoi(&dec1, &j)			

tdectol (dec_t *p, long *pl)

Descripción: Convierte un decimal (dec_t) en un long.

Parámetros	:	p	-	(I)	puntero al decimal
	:	pl	-	(O)	puntero al long
Retorna	:	-1200	:	número demasiado grande	
	:	0	:	SUCCESS	
Ejemplo	:	tdectol(&dec1, &l1)			

tflttodec (float f, dec_t *p)

Descripción: Convierte un float en un decimal (dec_t).

Parámetros	:	f	-	(I)	float
	:	p	-	(O)	puntero al decimal
Retorna	:	-1200	:	número demasiado grande	
	:	-1201	:	número demasiado pequeño	
	:	0	:	SUCCESS	
Ejemplo	:	tflttodec((float)1078.45, &d1)			

titodec (short i, dec_t *p)

Descripción: Convierte un short en un decimal (dec_t).

Parámetros	:	i	-	(I)	short
	:	p	-	(O)	puntero al decimal
Retorna	:	-1200	:	número demasiado grande	
	:	-1201	:	número demasiado pequeño	
	:	0	:	SUCCESS	
Ejemplo	:	titodec(10, &d1)			

tltodec (long l, dec_t *p)

Descripción: Convierte un long en un decimal (dec_t).

Parámetros	:	l	-	(I)	long
	:	p	-	(O)	puntero al decimal
Retorna	:	-1200	:	número demasiado grande	
	:	-1201	:	número demasiado pequeño	
	:	0	:	SUCCESS	
Ejemplo	:	tltodec(35L, &dec1)			