1 Introducción

1.1 que es embedded SQL

SQL inmerso es un método para insertar instrucciones SQL dentro de un lenguaje de programación y permitir el acceso a bases de datos mediante programas. En el caso de PostgreSQL el lenguaje de alto nivel utilizado es C.
Este conjunto de instrucciones es compilado por el preprocesador SQL que genera un código en C con unas librerías especiales mediante el cual podemos acceder a las bases de datos.

2 Sintaxis

El termino SQL inmerso se refiere a las declaraciones de SQL insertadas en una aplicación. Las instrucciones para manipular las Bases de Datos son las comunes de SQL como INSERT para insertar filas a nuestra tabla, SELECT para seleccionar los datos que busquemos o UPDATE para modificar el contenido de nuestra Base de Datos, pero utilizadas dentro de un programa en lenguaje de alto nivel.

De todas formas no todas las instrucciones de PostgreSQL estan presentes en SQL inmerso, solo hay unas cuantas instrucciones que permiten la manipulacion y transeferencia de datos desde los programas a la base de datos.

2.1 Estructura Basica

Para poder utilizar de manera satisfactoria las instrucciones de SQL y que funcionen de manera correcta es necesario que esten todas encabezadas por la sentencia “EXEC SQL” finalizando con un punto y coma para que asi el precompilador ecpg traduzca correctamente la instrucción a nuestro lenguaje.

La mayoria de las instrucciones son iguales a SQL o añadiendo algun parametro extra. Ahora se muestran dos ejemplos:

COMMIT; ← Instrucción de SQL

EXEC SQL COMMIT; ← Sentencia en nuestro programa en C

SELECT nombre FROM tabla; ← Instrucción de SQL

EXEC SQL SELECT nombre INTO :nombre_variable FROM tabla; ← Sentencia en nuestro programa en C.

2.2 Declaracion de variables de intercomunicacion

Las variables de intercomunicacion son las encargadas del intercambio de datos entre el programa y la base de datos PostgreSQL. El programa utiliza estas variables para enviar los datos a la base de datos y la base de datos utiliza las variables de intercomunicacion para enviar datos al programa que hemos realizado.

Para la declaracion de variables usamos el mismo metodo que para la declaracion de variables en C con la unica diferencia de que esta declaracion de variables debe estar precedida por la instrucción:

EXEC SQL BEGIN DECLARE SECTION;

y finalizada por :

EXEC SQL END DECLARE SECTION;

Las variables declaradas entre estas sentencias se pueden usar en todo el programa aunque no pertenezcan a instrucciones de SQL pero no al reves, las variables declaradas fuera, es decir, como variables de C normales no las podremos usar en las instrucciones de SQL.

```
EXEC SQL BEGIN DECLARE SECTION;
        int num;
        char name[20];
END DECLARE SECTION;
```

2.3 Uso de Variables de Intercomunicacion

Las variables de intercomunicacion pueden ser usadas en cualquier instruccion de un programa, tanto en instrucciones tipicas de C como “scanf(“%d”, nombre_variable”)”, como en instrucciones de SQL inmerso como EXEC SQL SELECT nombre INTO :nombre_variable FROM tabla;.

Para que las variables puedan ser usadas en instrucciones de SQL es necesario que las variables esten precedidas por dos puntos “:” en los programas. La comunicación entre la base de datos y el programa se produce debido que las variables con “:” delante se interpretan como variables de intercomunicacion.

Estas variables son las unicas que se pueden utilizar para comunicar el programa con la Base de Datos ya que todas las variables declaradas fuera de las sentencias del apartado anterior seran interpretadas como variables del programa normales y no de intercomunicacion.

Variables de entrada y de salida

Las variables de intercomunicacion se utilizan en dos tipos de instrucciones: en instrucciones como SELECT en las que el programa los datos y las variables los almacenan o en instrucciones como INSERT en las que las variables envian los datos a la base de datos. En los casos de instrucciones SELECT hay que indicar que se han de almacenar datos en esas variables. Para eso se utiliza la instrucción INTO para especificarlo.

EXEC SQL SELECT name, cod_empresa INTO :name_emp, :cod_emp FROM empresas;

Las variables de intercomunicacion tambien las se pueden utilizar para indicar el nombre de la base de datos a la que el programa se va a conectar. Se puede pedir al usuario que indique el nombre de la base de datos y a traves de esta variable host conectarse a ella.

EXEC SQL CONNECT :database;

Sin embargo esto no funciona para crear o modificar tablas, no se puede dar el nombre de una tabla a una variable host y luego utilizar esta variable pata modificar una tabla.

2.4 Indicadores

A cada variable de intercomunicacion se le puede asociar una variable indicador. Esta variable indicador lo que hace es informarnos sobre el valor de la variable de intercomunicacion. Se pueden usar los indicadores para detectar valores nulos o incorrectos de la variable de intercomunicacion.

Esta variable indicadora se declara como un short en la seccion de declaracion de variables host. Para asociarla a una variable hosr se usa la palabra especial INDICATOR despues de usar la variable de intercomunicacion a la que se quiere asociar, aunque tambien puede omitirse la palabra INDICATOR. Un ejemplo podria ser este:

:host_variable INDICATOR :indicator_variable;
:host_variable:indicator_variable;

2.5 Instrucciones Basicas

INSERT

La instrucción INSERT sigue el mismo mecanismo que una insercion en SQL. Solo varia que al inicio es necesario EXEC SQL y para finalizar EXEC SQL COMMIT; Para poder utilizar los valores de las variables se necesitan los dos puntos previos al nombre de la variable.

EXEC SQL INSERT INTO tabla(col1, col2, ..., coln) VALUES(:var1, :var2, ..., :varm);

EXEC SQL
INSERT INTO programa(cod_programa, nombre_programa, dia_emision, horario, audiencia, cod_emisora) VALUES(:cod_prog, :nombre_prog, :d_e, :h, :au, :cod_emi);

Este ejemplo inserta una nueva fila en la tabla programas con los valores indicados en las variables previamente declaradas.

UPDATE

Para modificar valores de una tabla la estructura es igual que usando SQ, con la instrucción UPDATE . La única diferencia es que hay que referenciar las variables mediante dos puntos y encabezar la función con EXEC SQL.

EXEC SQL UPDATE tabla SET col1:=var1, col2:=var2, ..., coln:=varn WHERE condicion;

Un ejemplo seria este:

EXEC SQL UPDATE emisoras SET telefono=:tel, frecuencia=:emi WHERE cod_emisora= :cod_emi;

DELETE

Se utiliza la instrucción DELETE para eliminar una fila de nuestra tabla.
También sigue la misma estructura que las instrucciones anteriores. Primero EXEC SQL y final EXEC SQL COMMIT. Las variables se referencian mediante :.

EXEC SQL DELETE FROM tabla WHERE condicion;

EXEC SQL DELETE FROM programa WHERE cod_programa= :cod_prog;

3. Uso de SQL Inmerso

En esta sección se vera como se usa el SQL inmerso. Primero como conectar a una base de datos y luego las maneras de realizar las consultas, tanto de manera dinámica como estática, es decir, pidiendo al usuario que entre una consulta el mismo por el terminal o pidiéndole solo unos valores con las consultas definidas dentro del programa.

3.1 Conexión a la Base de Datos

Para conectar a una base de datos usamos la instrucción

EXEC SQL CONNECT nombre_de_la_base_de datos;

Se puede indicar el login y el password en la instrucción y pedirle al usuario que lo introduzca el mismo o indicandolo dentro del programa.

Es conveniente conectar a la base de datos una sola vez en cada programa en la funcion main y desconectar con la instrucción

EXEC SQL DISCONNECT;

al final de la funcion principal. Si esto no sucede se ralentiza el programa y resulta poco optimo en terminos de velocidad y accesos a la base de datos.

3.2 SQL Estatico

En esta sección se vera la manera de realizar operaciones con SQL inmerso de manera satisfactoria como consultas, inserciones de filas o modificación de valores. Primero comentaremos las operaciones básicas y luego se trataran las consultas de una forma mas detallada.

El lenguaje de programación C, de tercera generación, solo esta preparado para tratar variables atómicas, por ejemplo una casilla, y se ve incapaz de procesar de forma correcta los datos que le puedan llegar de un lenguaje de una generación posterior o relacional, como seria el caso de las consultas que devuelven mas de una fila como resultado.

Las consultas las se dividen en dos tipos distintos, consultas atómicas y no atómicas. Las consultas atómicas son las que devuelven un solo valor como resultado y las no atómicas las que devuelven valores compuestos.

3.2.1 Consultas Atómicas

Cuando una consulta devuelve un valor atómico es directamente almacenable en una variable. Si devuelve una sola fila se almacena en tantas variables como tenga la fila. Para esto se usa la sentencia INTO entre las sentencias SELECT y FROM.
INTO indica las variables en las que se quieren almacenar el resultado de la consulta que hayamos hecho, por ejemplo, si se pide el nombre de un cliente se tendra que almacenar en una variable y la instrucción seria la siguiente:

EXEC SQL SELECT nombre_cliente INTO :nb_cliente FROM ...

La variable nb_cliente debería haber sido declarada al inicio de la función y los dos puntos antes de la variable son para referenciar al nombre de la variable y que el precompilador sepa que es una variable y no otra cosa.
Ahora se indica un ejemplo de consulta simple:

```
EXEC SQL BEGIN DECLARE SECTION;
    int cod_emi=1;
    char nb_emi[20];
    int tel=0;
    char emi[2], db[15];
EXEC SQL END DECLARE SECTION;

EXEC SQL SELECT nombre_emisora, telefono, frecuencia
    INTO :nb_emi, :tel, :emi  FROM emisoras
    WHERE cod_emisora =:cod_emi;
```

Este ejemplo consultaría la tabla emisoras de la base de datos que contenga la variable db y devolvería los datos pedidos que se almacenan en nb_emi, tel y emi.

Con este patrón también se pueden hacer consultas más complejas utilizando JOIN o HAVING o alguna instrucción similar, siempre y cuando la consulta devuelva un solo valor:

```
EXEC SQL SELECT nombre_emisora, nombre_programa, horario INTO :nb_emi, :nb_prog, :h
        FROM emisoras, programa
        WHERE programa.cod_emisora=emisoras.cod_emisora and
    programa.audiencia<14;
```

3.2.2 Consultas NO Atómicas

Para este tipo de consultas el método anterior no sirve. Cuando una consulta devuelve múltiples valores no es posible almacenar estos valores en variables que solo aceptan valores atómicos. Teniendo múltiples valores como los que devuelven las consultas complejas debemos almacenarlos y luego recorrer los resultados de manera secuencial mediante cursores.

En las consultas con cursores se utilizan estos elementos:

DECLARE CURSOR para declarar el cursor que se va a utilizar e indicar la select que se va a realizar, OPEN que lo abre para que se pueda utilizar y recoger los datos que ha producido la consulta, FETCH que sirve para pasar a la siguiente posicion del cursor y CLOSE que sirve para cerrar el cursor abierto.

La estructura utilizada es el CURSOR y tiene un modo de utilizacion y unas instrucciones caracteristicas.

Habiendo declarado anteriormente todas las variables en las que almacenaremos los resultados se pasa a declarar el cursor. Para esto se utiliza la instrucción EXEC SQL DELCLARE seguida por el nombre del cursor y luego por la palabra CURSOR FOR. Despues de esto se introduce nuesra consulta y la declaracion finaliza.
Una vez declarado el cursor pasa a la obtención de los datos de la consulta. Para esto se abre el cursor mediante la instrucción “EXEC SQL OPEN nombre_cursor;” y después se pasa a coger los datos proporcionados por el cursor mediante la instrucción FETCH que proporciona los resultados fila a fila.

DECLARE CURSOR

La sentencia DECLARE CURSOR se utiliza para declarar el cursor con un nombre y asignarle la operación de selección que se va a utilizar. La estructura que tiene es la siguiente: DECLARE CURSOR nombre_del_cursor FOR SELECT loquesea FROM ... WHERE [HAVING GROUP BY...]

La SELECT que se haga dentro del CURSOR no puede contener ninguna sentencia INTO ya que no se va a guardar los resultados en la declaración, sino más adelante cuando se utilice la instrucción FETC para recoger los datos que interesen.

La declaración del cursor, como cualquier otra declaración debe preceder a otras instrucciones.

```
EXEC SQL DECLARE interval CURSOR
FOR SELECT * FROM programa
WHERE ((:int1<= audiencia) and (: int2>= audiencia))
ORDER BY dia_emision, horario;
```

Se pueden declarar tantos cursores como se necesiten aunque deben tener distinto nombre.

OPEN

Se utiliza la sentencia OPEN para abrir un cursor.

```
EXEC SQL OPEN emp_cursor;
```

El cursor debe estar ya declarado. Cuando se utiliza la sentencia open se ejecuta la consulta y el programa se posiciona en la primera fila del resultado de esta consulta. Puede ser que mientras se utiliza el cursor se produzcan cambios en la base de datos pero el programa no se dara cuenta. Para poder ver los cambios que se realicen mientras se utiliza el cursor sera necesario que lo se cierre y lo se vuelva a abrir. Un cursor se puede abrir y cerrar tantas veces como se quiera.

FETCH

Se utiliza la instrucción FECTH para obtener filas de la consulta que se habia hecho anteriormente.

```
EXEC SQL FETCH NEXT FROM cursor
INTO :var1, :var2, ..., var_num_col;
```

Las filas que se obtienen las se pasan a las variables de intercomunicacion que se han indicado en la instrucción FETCH mas adelante mediante INTO. Hay que referenciar la instrucción FETCH a un CURSOR declarado anteriormente indicandolo en la instrucción. Para poder usar FETCH se debe haber abierto el cursor antes.

Para pasar a la siguiente columna se utiliza la instrucción NEXT justo después de FETCH. Mientras no se utilice la instrucción NEXT el cursor no avanzara a la siguiente posicion.

```
EXEC SQL FETCH FROM interval
                    INTO :c_p, :n_p, :d_e, :h, :au, :cod_emi;
```

```
EXEC SQL FETCH FROM interval
                                        INTO :c_p1, :n_p1, :d_e1, :h1, :au1, :cod_emi1;
```

CLOSE

Se utiliza CLOSE para cerrar un cursor declarado y abierto previamente. La instrucción es:

EXEC SQL CLOSE nombre_cursor;

Se libera el espacio usado por el cursor y se borra su contenido.

Ejemplo

Un sejemplo podria ser este, en el que la consulta que se quiere realizar
SELECT * FROM programa WHERE ((:int1<= audiencia) and (:int2>= audiencia))
ORDER BY dia_emision, horario;
devuelve mas de una fila como resultado

```
                                EXEC SQL BEGIN DECLARE SECTION;
                                int c_p;
                                    char n_p[30];
                                char d_e[2];
                                      char h[4];
                                      float au;
                                    int cod_emi;
                                    char c[5], db[15];
                                    float int1=0,int2=0;
                                EXEC SQL END DECLARE SECTION;

                                EXEC SQL DECLARE interval CURSOR
                                FOR SELECT *
                                          FROM programa
                                          WHERE ((:int1<= audiencia) and (:int2>= audiencia))
                                              ORDER BY dia_emision, horario;
                                    EXEC SQL OPEN interval;
                                    do
                                    {
                                              EXEC SQL FETCH NEXT FROM interval
                                                        INTO :c_p, :n_p, :d_e, :h, :au, :cod_emi;
                                              if (sqlca.sqlcode != 0)
                                              {
                                                        if (i==0)
                                                        {
                                                        printf("\nNo  hay  ningun  programa  con  los
intervalos dados\n");
                                                        }
                                              break;
                                              }
                                              else
                                              {
                                              printf("\n\tCodigo De Programa: %d", c_p);
                                              printf("\n\tNombre del programa: %s", n_p);
                                              printf("\n\tDia de emision: %s", d_e);
                                              printf("\n\tHorario: %s", h);
                                    printf("\n\tAudiencia: %f", au);
                                    printf("\n\tcodigo emisora: %d\n", cod_emi);
                                              i++;
```

```
                    printf("\nPulsa c y ENTER para ver el siguiente
programa\n");
                    scanf("%s",c);
                    }
            } while (1);
        EXEC SQL CLOSE interval;
```

Finalmente se cierra el cursor mediante la instrucción EXEC SQL CLOSE nombre_del_cursor;

3.3 SQL Dinamico

SQL Inmerso Dinámico permite la ejecución de instrucciones SQL no conocidas en tiempo de compilación al contrario de SQL Inmerso Estático. Mientras que en SQL Estático la consulta sé precompila y se prueban los posibles errores sintácticos en tiempo de compilación en SQL Dinámico esto no sucede así.

3.3.1 Sintaxis

Cuando se hacen las consultas hay que almacenar esta consulta en una variable de tipo string y luego pasársela a la Base de Datos. Esta consulta sigue el mismo patrón que cualquier consulta de SQL con la única diferencia de que NO hay que escribir un punto y coma al finalizar la instrucción.
Una vez se tiene la instrucción se le pasa a la Base de Datos y esta devolverá el resultado.
Las consultas se diferencian por la manera de pasar los parámetros a la Base de Datos: se puede hacer pasando toda una consulta en un string (no valido para SELECTS y consultas que devuelvan valores) o indicando los valores fuera de la secuencia de instrucciones.
Los resultados dependen de las consultas que se hagan. Se puede obtener un código de error que indicara si la instrucción se ha ejecutado satisfactoriamente o no y si se ha hecho una SELECT devolverá los diversos resultados de la consulta. Así se podrán dividir las consultas en tres tipos: SELECTS y consultas que devuelven valores, consultas se escriben directamente y consultas en las que se indican valores.

3.3.2 Consultas que se escriben directamente

En este tipo de consultas se pasa directamente la consulta a través de una variable string de intercomunicación. Para esto se pide al usuario que escriba la instrucción y el programa la recoge siempre sin el punto y coma al final del string que contiene la instrucción.
Para enviar la instrucción a la Base de Datos se utiliza la sentencia :

EXEC SQL EXECUTE IMMEDIATE :nombre_de_la_variable_que_contiene_la_instruccion;

Con este método no se pueden realizar SELECTS ya que no se espera que se devuelva ningún dato de la Base de Datos a excepción del código de error sqlca.sqlcode.
Un ejemplo podría ser el siguiente :

```
printf("\nintroduce el nombre de la base d datos: ");

scanf("%s", &db);

EXEC SQL CONNECT TO :db;


if (sqlca.sqlcode!=0)

{

        printf("\nERROR en la conexion");

        exit(-1);
```

```
}
else printf("\nBase de datos %s abierta\n", db);

printf("\nEscribe una consulta:    ");
strcpy(tablenm, “UPDATE tabla SET col1=’23’ WHERE codicion”);
printf("\n%s", tablenm);
EXEC SQL EXECUTE IMMEDIATE :tablenm;
EXEC SQL COMMIT;
printf("\n--> %d <--\n", sqlca.sqlcode);

EXEC SQL DISCONNECT;
```

3.3.3 Consultas en las que se indican variables

Este tipo de consultas es similar al anterior con la única diferencia de que no es necesario indicar los valores que hay que poner. Estos valores son sustituidos en las consultas por unos indicadores que luego serán sustituidos por las variables de intercomunicación que a nosotros nos interese.
Para hacer esto primero se pide al usuario que introduzca la consulta que desee y se almacena temporalmente mediante la instrucción EXEC SQL PREPARE sql_cons FROM :instruccion;
Dentro de la variable instrucción debería haber una cosa semejante a esta: UPDATE nombre_tabla SET param1=:var1, param2=:var2..., paramn=:varn
La variable sql_cons no tiene que ser ni declarada ni inicializada, solo indicar que existe cuando se la necesite. Es una especie de declaración cursor.
Una vez hecho esto cuando sea necesario se le pasan las variables que el usuario le ha indicado en la instrucción, para hacer esto se utiliza EXEC SQL EXECUTE sql_cons USING :cod_prog1, :cod_prog1, ..., cod:progn;
Detrás del la palabra USING se deben poner todas las variables de intercomunicación que el usuario ha indicado en la instrucción (no es necesario que tengan el mismo nombre).

```
EXEC SQL CONNECT TO :db;
if (sqlca.sqlcode!=0)
{
        printf("\nERROR en la conexion");
        exit(-1);
}
else printf("\nBase de datos %s abierta\n", db);

printf("\nEscribe una consulta:    ");
strcpy(tablenm, “UPDATE tabla SET col1=:var1 WHERE condicion;
tablenm.arr[i-1]='\0';
```

```
tablenm.len = strlen(tablenm.arr);

printf("\n%s", tablenm.arr);


printf("\n\tEscribe el codigo del programa: ");

cod_prog=1;

scanf("%d", cod_prog);

printf("\n--> cod_prog: %d <--\n", cod_prog);


EXEC SQL PREPARE sql_cons FROM :tablenm;

printf("\n--> %d <--\n", sqlca.sqlcode);


EXEC SQL EXECUTE sql_cons USING :cod_prog;


EXEC SQL COMMIT;
```

3.3.4 SELECTS y consultas que devuelven valores múltiples

Para este tipo de consultas se necesita declarar cursores donde almacenar temporalmente los resultados que se obtengan. La manera de realizar estas consultas es muy similar a la anterior, solo se necesita una variable donde almacenar las respuestas a las consultas.
Primero se pide al usuario que introduzca su consulta y se almacena de la misma manera que en el punto anterior:

```
SELECT nombre_cli, direccion FROM clientes WHERE cod_cli=:var1;
EXEC SQL PREPARE sql_cons FROM :instruccion;
```

Después, para controlar las respuestas que de la Base de Datos se debe declarar un cursor para esta instrucción:

```
EXEC SQL DECLARE cursor_cons CURSOR FOR sql_cons;
```

Y finalmente se abre el cursor para la consulta indicando detrás de la palabra USING las variables que se han utilizado al hacer el SELECT:

```
EXEC SQL OPEN cursor_cons USING :cod_cli;
```

Para obtener los datos que devuelve la base de datos como resultado se utiliza la instrucción normal de cursores

```
EXEC SQL FETCH NEXT FROM cursor_cons
INTO :nb_cli, :dir;
```

Un ejemplo podría ser este:

```
EXEC SQL CONNECT TO :db;

if (sqlca.sqlcode!=0)
```

```
{
        printf("\nERROR en la conexion");
        exit(-1);
}
else printf("\nBase de datos %s abierta\n", db);

printf("\nEscribe una consulta:    ");
strcpy(tablenm, “SELECT * FROM tabla WHERE condicion<:=variable”);
tablenm.len = strlen(tablenm.arr);
printf("\n%s", tablenm.arr);

printf("\n\tEscribe el codigo del programa: ");
cod_prog=1;
scanf("%d", cod_prog);

EXEC SQL PREPARE sql_cons FROM :tablenm;
printf("\n--> %d <--\n", sqlca.sqlcode);

EXEC SQL DECLARE cursor_cons CURSOR FOR sql_cons;
printf("\n--> %d <--\n", sqlca.sqlcode);

EXEC SQL OPEN cursor_cons USING :cod_prog;
printf("\n--> %d <--\n", sqlca.sqlcode);


do
{
        EXEC SQL FETCH NEXT FROM cursor_cons
        INTO :cod_emi, :nombre_emi, :s, :f, :tel;
        printf("\n--> %d <--\n", sqlca.sqlcode);
        if (sqlca.sqlcode != 0)
        {
                if (i==0)
```

```
                    {
                    printf("\nNo hay ningun programa con los intervalos dados\n");
                    }
            break;
            }
            else
            {
                    printf("\n\tCodigo De emisora: %d", cod_emi);
                    printf("\n\tNombre del emisora: %s", nombre_emi);
                    printf("\n\tfrecuencia: %s", f);
            printf("\n\ttelefono: %d", tel);
            printf("\n\tsintonia: %f", s);
            printf("\n\tcodigo emisora: %d\n", cod_emi);
                    i++;
                    printf("\nPulsa c y ENTER para ver el siguiente programa\n");
                    scanf("%s",&c);
            }
    } while (1);


    printf("\n--> %d <--\n", sqlca.sqlcode);


    printf("\nnombre emisora: %s, telefono: %d, freq: %s", nb_emi, tel, emi);


    EXEC SQL DISCONNECT;
```

3.4 Indicadores y utilizacion de valores nulos

Las variables indicador se asocian a las variables host normales y permiten saber la situacion de estas variables.

Las variables host se asocian a las variables host en las instrucciones VALUES, SET o INTO dependiendo de si enviamos datos a la base de datos o los estamos recibiendo.

La variable indicadora puede contener cuando enviamos datos

-1	La Base de Datos asignara un valor nulo a la columna ignorando el valor de la variable de

	intercomunicacion.
>=0	La Base de Datos asignara el valor que tiene la variable de intercomunicacion a la columna.

La variable indicadora puede contener cuando recibimos datos

-1	La variable de la columna es un valor NULO
0	La variable de la columna NO es un valor NULO
>0	La BD asigna un valor truncado a la variable de intercomunicacion. El valor del indicador es la longitud original de la variable host.

Insercion de valores nulos

En la base de datos se pueden insertar valores nulos como si fuesen valores normale. Esto puede ser necesario para algunas consultas o algun control de datos especiales.

EXEC SQL INSERT INTO emisoras (nombre, cod_emi, telefono) VALUES (:nb_emi, c_e, NULL);

Deteccion de valores nulos

Al realizar una consulta sera necesario comprovar que variables han recogido un valor nulo consultando que valor tiene el indicador en un if posterior.

```
            EXEC SQL SELECT nombre_emisora, telefono, frecuencia
INTO :nb_emi, :tel:vindi, :emi FROM emisoras
                        WHERE cod_emisora =:cod_emi;
If(vindi==-1)then {...}
```

3.5 Errores

En las instrucciones de acceso a bases de datos se pueden producir errores por multiples causas que deben ser contempladoes en el programa, como por ejemplo que la base de datos no exista cuando intentamos conectarnos a ella. Estos errores se identifican por un codigo en la variable sqlca.sqlcode. Estos codigos estan definidos en el fichero ecpgerrno.h y cada codigo contiene una pequeña definicion del tipo de error que ha sucedido.

Tambien hay unas instrucciones capaces de detectar los errores en tiempo de ejecucion y que nos permiten actuar en consecuencia si detectan un error.

3.5.1 Lista de errores, fichero ecpgerrno.h

En este apartado se muestran todos los errores y se describen los mas comunes. Para acceder en el programa a estos errores se puede hacer a traves de la variable sqlca.sqlcode que contiene el valor retornado por la funcion de SQL Inmerso una vez ejecutada en el programa.

Nombre	Valor	Explicacion	Instrucciones que lo provocan
ECPG_NO_ERROR	0	Este es el valor que devuelve una operación de SQL inmerso cuando todo ha ido bien y la consulta se ha producido de manera satisfactoria	todas
ECPG_NOT_FOUND	100	Es el valor devuelto cuando la consulta se ha realizado correctamente pero no habia coincidencias en la tabla que se correspondiesen con nuestra consulta	Selects y operaciones que contengan alguna consulta
ECPG_OUT_OF_MEMORY	-ENOMEM	No hay sufuciente memoria	todas
ECPG_UNSUPPORTED	-200	El preprocesador ha generado algo que la librería no conoce.	Todas ?
ECPG_TOO_MANY_ARGUMENTS	-201	Este error sucede cuando escribimos demasiados argumentos en una instruccion, incluso cuando ponemos argumentos no esperados en sentencias de SQL dinamico.	todas
ECPG_TOO_FEW_ARGUMENTS	-202	Este error se presenta cuando se escriben menos argumentos de los esperados en la instrucción	todas
ECPG_TOO_MANY_MATCHES	-203	Este error se da cuando al hacer consultas el programa no esta preparado para atender todos los resultados que nos devuelve la BD	Selects
ECPG_INT_FORMAT	-204	El programa espera que la BD devuelva un entero, en cambio devuelve algo	Operaciones en las que la base de datos nos devuelve valores

		diferente a un int	
ECPG_UINT_FORMAT	-205	El programa espera un Unsigned Int pero recibe otro tipo de dato	Operaciones en las que la base de datos nos devuelve valores
ECPG_FLOAT_FORMAT	-206	El programa espera un FLOAT pero recibe otro tipo de dato	Operaciones en las que la base de datos nos devuelve valores
ECPG_CONVERT_BOOL	-207	This means that the host variable is of a bool type and the field in the Postgres database is neither 't' nor 'f'.	?
ECPG_EMPTY	-208	Postgres returned PGRES_EMPTY_QUERY, probably because the query indeed was empty.	?
ECPG_MISSING_INDICATOR	-209	?	?
ECPG_NO_ARRAY	-210	?	?
ECPG_DATA_NOT_ARRAY	-211	?	?
ECPG_NO_CONN	-220	Se intenta realizar operaciones con una BD pero no se esta conectado.	todas
ECPG_NOT_CONN	-221	El programa intenta acceder a una conexión que existe pero que no esta abierta	todas
ECPG_INVALID_STMT	-230	Este error se produce cuando en sql dinamico cuando se ejecuta una consulta que no contiene nada, una consulta que no esta preparada, o en general cuando se intenta ejecutar un comando de SQL incorrecto	todas
ECPG_UNKNOWN_DESCRIPTOR	-240	?	dynamic SQL related
ECPG_INVALID_DESCRIPTOR_INDEX	-241	?	dynamic SQL related
ECPG_UNKNOWN_DESCRIPTOR_ITEM	-242	?	dynamic SQL related
ECPG_VAR_NOT_N	-243	?	dynamic SQL related

UMERIC			
ECPG_VAR_NOT_CHAR	-244	?	dynamic SQL related
ECPG_PGSQL	-400	se ha producido cuando he comparado una variable int con una variable char, en sql dinamico cuando se pasa una instruccion no valida	Backend error messages
ECPG_TRANS	-401	No se puede comenzar, COMMIT o ROLLBACK la transaccion	Transacciones
ECPG_CONNECT	-402	No se puede conectar a la BD, la conexión con la BD no funciona	todas
ECPG_NOTICE_UNRECOGNIZED	-600	NOTICE: (transaction aborted): queries ignored until END	backend notices
ECPG_NOTICE_QUERY_IGNORED	-601	NOTICE: PerformPortalClose: portal "*" not found	backend notices
ECPG_NOTICE_UNKNOWN_PORTAL	-602	NOTICE: BEGIN: already a transaction in progress	backend notices
ECPG_NOTICE_IN_TRANSACTION	-603	NOTICE: AbortTransaction and not in in-progress state NOTICE: COMMIT: no transaction in progress	backend notices
ECPG_NOTICE_NO_TRANSACTION	-604	NOTICE: BlankPortalAssignName: portal * already exists	backend notices
ECPG_NOTICE_PORTAL_EXISTS	-605	?	backend notices

3.5.2 gestion de errores en tiempo de ejecucion

Normalmente se pueden producir muchos errores cuando se ejecuta una consula, como errores estaticos, nombres mal escritos.

Para estos errores se dispone de la instrucción WHENEVER a la que se puede acceder si se produce algun error. Se puede poner la instrucción WHENEVER justo despues de una consulta y mediante varias opciones decidir como continuar la ejecucion del programa.

El formato es el siguiente:

EXEC SQL WHENEVER <condition> <action>;

En la opción condition se debe indicar el tipo de error para el que se quiere que haya una respuesta especial.

SQLERROR indica que se ha producido un error de ejecución.
NOT FOUND indica que no se ha encontrado ninguna fila en la SELECT realizada.

En la opción <action> se debe indicar lo que se quiere que se realice en caso de error. Tenemos tres opciones:

· CONTINUE con lo que se intentara ejecutar la siguiente instrucción del programa
· DO <instruction> con lo que se podrá ejecutar una instrucción como printf("error"); o una función creada por el programador siempre que tenga un valor de retorno.
· GOTO <label_name> con la que se puede saltar a una etiqueta del programa previamente declarada

4. Transacciones

Hasta ahora se han visto las instrucciones de manera aislada pero lo normal es utilizar SQL para transacciones que agrupan instrucciones para pasar de un estado consistente de la base de datos a otro estado consistente.

Como empezar y finalizar transacciones

Se realizan transacciones en cada instrucción de SQL que se ejecutan. Unas se guardan al momento en la base de datos como las operaciones CREATE TABLE pero otras como INSERT o UPDATE hay que decir explicitamente que se guarde de forma permanente en la Base de Datos o que deshaga los cambios previamente efectuados.

COMMIT

Se usa la sentencia COMMIT para terminar una transaccion salvando los cambios que se han realizado. Mientras los cambios no estan guardados la transaccion no finaliza y nadie puede acceder a los datos en la BD. La sentencia COMMIT hace permanentes los cambios realizados, los hace visibles a los otros usuarios permitiendo su manipulaciony finaliza la transaccion.

Existe una opcion en el preprocesador ecpg que permite no usar la instrucción COMMIT y aun asi que se guarden los cambios producidos en las transacciones realizadas en el programa. Escribiendo ecpg –t filename.sqc permitimos que se guarden automaticamente las transacciones que se realizan.

La forma mas correcta de usar COMMIT es poniendo la instrucción justo despues de realizar la transaccion habiendo comprovado que no se ha producido ningun error en la transaccion para que asi la BD este el minimo tiempo ocupada y se guarden los cambios rapidamente.

EXEC SQL UPDATE <table> SET <instruction> WHERE <query>;

If (ERROR) EXEC SQL ROLLBACK;

ELSE EXEC SQL COMMIT;

EXEC SQL COMMIT;

ROLLBACK

Se usa ROLLBACK para deshacer los cambios que estan pendientes de salvar en la base de datos. Por ejemplo si se produce un error en alguna instrucción de una transaccion se usa la sentencia ROLLBACK para evitar que se guarden de forma permanente. Al escribir EXEC SQL ROLLBACK se deshacen los cambios producidos y se finaliza la transaccion que esta activa actualmente.

La forma mas correcta de usar ROLLBACK seria ponerlo despues de de una instrucción de control de errores como WHENEVER o alguna sentencia de control de flujo como if usando la variable sqlca.sqlcode para poder deshacer los cambios en el caso de que se produjese algun tipo de error.

EXEC SQL UPDATE <table> SET <instruction> WHERE <query>;

If (ERROR) EXEC SQL ROLLBACK;

ELSE EXEC SQL COMMIT;

EXEC SQL COMMIT;

5. El precompilador

5.1 que es

El precompilador se llama ecpg. Se instala por defecto en el directorio */postgres/bin*. Al invocarlo pasándole como parámetro el fichero.sqc que contiene el programa, lo traduce a un fichero.c compilable por el compilador de C que este instalado en la maquina que se use, como por ejemplo gcc. Seguidamente hay que unir el programa objetivo creado con las librerías de PostgreSQL que se encuentran en el directorio de instalación de PostgreSQL, *gcc -g -I /usr/local/pgsql/include fichero.c -L /usr/local/pgsql/lib -lecpg -lpq*.

5.2 Opciones de compilación

```
ecpg [ -v ] [ -t ] [ -I include-path ] [ -o outfile ]  file1 [ file2 ]
[ ... ]
```

ecpg acepta los siguientes para metros de entrada:

-v Imprime información sobre la compilación

-t Desactiva las autotransacciones

-I *path* Especifica la dirección de las definiciones de las librerías. Inicialmente el camino es */usr/local/include*, el camino inicial de Postgres y sus includes que se define en tiempo de compilación */usr/local/pgsql/lib*, y */usr/include*.

-o Especifica a ecpg donde escribir el archivo salida. Si no se especifica nada se creara el mismo archivo pero con extensión .c

file los ficheros a procesar

Compilación y linkaje

Asumiendo que los archivos de Postgres están en el directorio */usr/local/pgsql* :

gcc -g -I /usr/local/pgsql/include [-o *file*] *file*.c -L /usr/local/pgsql/lib -lecpg -lpq

Ejemplo

Programa en SQL Inmerso

```
#include <string.h>
#include <stdio.h>
#include <stdlib.h>

EXEC SQL INCLUDE sqlca;

main()
{

int i;
//char db[15];


        EXEC SQL BEGIN DECLARE SECTION;
        int c_p;
                char n_p[30];
        char d_e[2];
              char h[4];
              float au;
                int cod_emi;
                char c[5], db[15];
                float int1=0,int2=0;
        EXEC SQL END DECLARE SECTION;

        printf("introduce el nombre de la base d datos: ");
        scanf("%s", &db);
        EXEC SQL CONNECT TO :db;
        if (sqlca.sqlcode!=0)
        {
                printf("\nERROR en la conexion");
                exit(-1);
        }
```

```
else printf("\nBase de datos %s abierta\n", db);

i=0;
printf("Introduce el intervalo menor de audiencia\n");
scanf("%f",&int1);
printf("%f\n",int1);
printf("Introduce el intervalo mayor de audiencia\n");
scanf("%f",&int2);
printf("%f\n",int2);

EXEC SQL DECLARE interval CURSOR
FOR SELECT *
        FROM programa
        WHERE ((:int1<= audiencia) and (:int2>= audiencia))
           ORDER BY dia_emision, horario;
    EXEC SQL OPEN interval;
    do
    {
        EXEC SQL FETCH NEXT FROM interval
            INTO :c_p, :n_p, :d_e, :h, :au, :cod_emi;
        if (sqlca.sqlcode != 0)
        {
            if (i==0)
            {
            printf("\nNo hay ningun programa con los intervalos dados\n");
            }
        break;
        }
        else
        {
        printf("\n\tCodigo De Programa: %d", c_p);
        printf("\n\tNombre del programa: %s", n_p);
        printf("\n\tDia de emision: %s", d_e);
```

```
                printf("\n\tHorario: %s", h);
                printf("\n\tAudiencia: %f", au);
                printf("\n\tcodigo emisora: %d\n", cod_emi);
                        i++;
                        printf("\nPulsa c y ENTER para ver el siguiente programa\n");
                        scanf("%s",c);
                        }
                } while (1);
        EXEC SQL CLOSE interval;


        EXEC SQL DISCONNECT;

}
```

Programa despues de usar el preprocesador ecpg

```
/* Processed by ecpg (2.8.0) */
/* These three include files are added by the preprocessor */
#include <ecpgtype.h>
#include <ecpglib.h>
#include <ecpgerrno.h>
#line 1 "cursores.sqc"
#include <string.h>
#include <stdio.h>
#include <stdlib.h>


#line 1 "/usr/local/pgsql/include/sqlca.h"
#ifndef POSTGRES_SQLCA_H
#define POSTGRES_SQLCA_H

#ifndef DLLIMPORT
```

```
#ifdef __CYGWIN__
#define DLLIMPORT __declspec (dllimport)
#else
#define DLLIMPORT
#endif /* __CYGWIN__ */
#endif /* DLLIMPORT */

#define SQLERRMC_LEN        70

#ifdef __cplusplus
extern          "C"
{
#endif

        struct sqlca
        {
                char            sqlcaid[8];
                long            sqlabc;
                long            sqlcode;
                struct
                {
                        int                     sqlerrml;
                        char            sqlerrmc[SQLERRMC_LEN];
                }                       sqlerrm;
                char            sqlerrp[8];
                long            sqlerrd[6];
                /* Element 0: empty                                     */
                /* 1: OID of processed tuple if applicable              */
                /* 2: number of rows processed                  */
                /* after an INSERT, UPDATE or                   */
                /* DELETE statement                             */
                /* 3: empty                                     */
                /* 4: empty                                     */
```

```
                /* 5: empty                                             */
                char            sqlwarn[8];
                /* Element 0: set to 'W' if at least one other is 'W'   */
                /* 1: if 'W' at least one character string              */
                /* value was truncated when it was                      */
                /* stored into a host variable.                         */

                /*
                 * 2: if 'W' a (hopefully) non-fatal notice occured
                 *//* 3: empty */
                /* 4: empty                                             */
                /* 5: empty                                             */
                /* 6: empty                                             */
                /* 7: empty                                             */

                char            sqlext[8];
        };

        extern DLLIMPORT struct sqlca sqlca;


#ifdef __cplusplus
}

#endif

#endif

#line 4 "cursores.sqc"


main()
{
```

```
int i;

        /* exec sql begin declare section */

#line 14 "cursores.sqc"
  int  c_p   ;

#line 15 "cursores.sqc"
  char  n_p [ 30 ]   ;

#line 16 "cursores.sqc"
  char  d_e [ 2 ]   ;

#line 17 "cursores.sqc"
  char  h [ 4 ]   ;

#line 18 "cursores.sqc"
  float  au   ;

#line 19 "cursores.sqc"
  int  cod_emi   ;

#line 20 "cursores.sqc"
```

```
  char  c [ 5 ]   ,  db [ 15 ]   ;

#line 21 "cursores.sqc"
  float  int1  = 0 ,  int2  = 0 ;
/* exec sql end declare section */
#line 22 "cursores.sqc"



        printf("introduce el nombre de la base d datos: ");
        scanf("%s", &db);
        { ECPGconnect(__LINE__, db , NULL,NULL , NULL, 0); }
#line 26 "cursores.sqc"

        if (sqlca.sqlcode!=0)
        {
                printf("\nERROR en la conexion");
                exit(-1);
        }
        else printf("\nBase de datos %s abierta\n", db);

        i=0;
        printf("Introduce el intervalo menor de audiencia\n");
        scanf("%f",&int1);
        printf("%f\n",int1);
        printf("Introduce el intervalo mayor de audiencia\n");
        scanf("%f",&int2);
        printf("%f\n",int2);
```

```
/* declare interval  cursor for select  *  from programa where ( ( ? <= audiencia  ) and ( ? >= audiencia  ) )  order by dia_emision  , horario    */

#line 46 "cursores.sqc"


                { ECPGdo(__LINE__, NULL, "declare interval  cursor for select  *  from programa where ( ( ? <= audiencia  ) and ( ? >= audiencia  ) )  order by dia_emision  , horario
",
        ECPGt_float,&(int1),1L,1L,sizeof(float),

        ECPGt_NO_INDICATOR, NULL , 0L, 0L, 0L,

        ECPGt_float,&(int2),1L,1L,sizeof(float),

        ECPGt_NO_INDICATOR, NULL , 0L, 0L, 0L, ECPGt_EOIT, ECPGt_EORT);}

#line 47 "cursores.sqc"


                do

                {


{ ECPGdo(__LINE__, NULL, "fetch next from interval", ECPGt_EOIT,

        ECPGt_int,&(c_p),1L,1L,sizeof(int),

        ECPGt_NO_INDICATOR, NULL , 0L, 0L, 0L,

        ECPGt_char,(n_p),30L,1L,30*sizeof(char),

        ECPGt_NO_INDICATOR, NULL , 0L, 0L, 0L,

        ECPGt_char,(d_e),2L,1L,2*sizeof(char),

        ECPGt_NO_INDICATOR, NULL , 0L, 0L, 0L,

        ECPGt_char,(h),4L,1L,4*sizeof(char),

        ECPGt_NO_INDICATOR, NULL , 0L, 0L, 0L,

        ECPGt_float,&(au),1L,1L,sizeof(float),

        ECPGt_NO_INDICATOR, NULL , 0L, 0L, 0L,

        ECPGt_int,&(cod_emi),1L,1L,sizeof(int),

        ECPGt_NO_INDICATOR, NULL , 0L, 0L, 0L, ECPGt_EORT);}

#line 51 "cursores.sqc"


                        if (sqlca.sqlcode != 0)

                        {

                                if (i==0)
```

```
                                        {
                                        printf("\nNo hay ningun programa con los intervalos dados\n");
                                        }
                                break;
                                }
                                else
                                {
                                printf("\n\tCodigo De Programa: %d", c_p);
                                printf("\n\tNombre del programa: %s", n_p);
                                printf("\n\tDia de emision: %s", d_e);
                        printf("\n\tHorario: %s", h);
                        printf("\n\tAudiencia: %f", au);
                        printf("\n\tcodigo emisora: %d\n", cod_emi);
                                i++;
                                printf("\nPulsa c y ENTER para ver el siguiente programa\n");
                                scanf("%s",c);
                                }
                        } while (1);
                { ECPGdo(__LINE__, NULL, "close interval", ECPGt_EOIT, ECPGt_EORT);}
#line 73 "cursores.sqc"




                { ECPGdisconnect(__LINE__, "CURRENT");}
#line 76 "cursores.sqc"



}
```