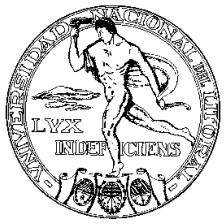


Universidad Nacional del Litoral

Facultad de Ingeniería y Ciencias Hídricas

Departamento de Informática

Bases de Datos

SQL: Guía de Trabajo Nro. 4

Stored Procedures: Consultas

2023

¿Como utilizo las consultas SQL en el contexto de un programa?

Como hemos estado viendo hasta aquí los procedures T-SQL y las funciones PL/pgSQL son programas bastante parecidos a los de otros lenguajes de programación. Su poder sin embargo radica en su vínculo tan estrecho con SQL y las posibilidades que brinda la interacción entre ambos.

Veremos cómo se lleva a cabo esta interacción.

1. Queries de los que no almacenamos su resultado

La primer categoría que analizaremos es la de consultas de las cuales no necesitamos almacenar su resultado.

Estas consultas se utilizan mucho en estructuras condicionales.

Ejemplo 1

Ver *"2.1. Subqueries que producen valores escalares"* en la *"Guía de trabajo Nro. 3 - Parte 2"*

Podemos utilizar SQL en una estructura condicional de la siguiente forma:

```
IF 1 <= (SELECT COUNT(*)
            FROM titles
            WHERE pub_id = '1389')

   PRINT 'Posee más de una publicación'
   ...

--END IF
```

Ejemplo 2

Ver "*2.2. Condiciones que involucran relaciones*" en la "*Guía de trabajo Nro. 3 - Parte 2*"

Podemos utilizar SQL en una estructura condicional de la siguiente forma:

```
IF 'D4482' IN (SELECT ord_num
                  FROM Sales)

   PRINT 'Existe la factura'
   ...

--END IF
```

Ejemplo 3

Ver "*2.2.6. El cuantificador EXISTS*" en la "*Guía de trabajo Nro. 3 - Parte 2*"

Podemos utilizar SQL en una estructura condicional de la siguiente forma:

```
IF NOT EXISTS (SELECT *
                  FROM titles
                  WHERE pub_id = '3389')

   PRINT 'No existe'
   ...

--END IF
```

Tanto T-SQL como PL/pgSQL soportan queries en condiciones.

2. Queries de los que almacenamos su resultado

2.1. Queries que retornan un valor escalar

Ver *"2.1. Subqueries que producen valores escalares"* en la *"Guía de trabajo Nro. 3 - Parte 2"*

Los queries que retornan un valor escalar se pueden usar al lado derecho de las sentencias de asignación, encerrados entre paréntesis.

```
SET @price = (SELECT price
                  FROM titles
                  WHERE title_id = @title_id);
```

```
price1 := (SELECT price
                  FROM titles
                  WHERE title_id = title_id1);
```

Más allá de la asignación a una variable, todo valor retornado por una consulta de este tipo puede en realidad ser considerado como un valor más en nuestro programa, que como tal puede ser usado en el contexto de cualquier expresión.

Las variables para almacenar un valor escalar deben ser declaradas por supuesto, con su tipo de dato correspondiente.
En el caso de PL/pgSQL tenemos la alternativa de utilizar lo que se denomina **anchored declarations**.

Anchored declarations

Las Anchored Declarations son declaraciones de variables especiales en las que se "asocia" la declaración de la variable **a un objeto de la base de datos**.

Cuando asociamos de esta manera un tipo de dato estamos indicando que queremos establecer como tipo de dato de nuestra **variable al tipo de dato de una estructura de datos previamente definida**, que puede ser una **tabla** de la base de datos, una **columna** de una tabla, o un TYPE predefinido.

T-SQL no soporta anchored declarations.

Scalar anchoring

Un scalar anchoring define una variable en base a una columna de una tabla.

Se especifican a través del atributo `%TYPE`.

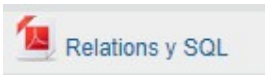
Ejemplo 4

```
DECLARE
      precio titles.price%TYPE;

   BEGIN
      precio := (SELECT price
                     FROM titles
                     WHERE title_id = prmTitle_id);

      RAISE NOTICE 'La publicación % vale $%', prmTitle_id, precio;
      ...
   END
```

2.2. Sentencias SELECT Single-row

Sentencias SQL single row

En el documento "*Relations y SQL*" vimos las sentencias SQL single row.



Ahora veremos como se recuperan los valores que retornan estas sentencias desde el código T-SQL y PL/pgSQL.

En T-SQL recuperamos los valores de una sentencia **SELECT single-row** anteponiendo una variable y un signo "=" a la columna o expresión:

```
DECLARE
   @price FLOAT,
   @type CHAR(12)

SELECT @price = price, @type = type
   FROM titles
   WHERE title_id = @title_id;
```

En PL/pgSQL recuperamos los valores de una sentencia **SELECT single-row** con una cláusula `INTO` que especifica las variables donde deben ser ubicados los componentes de esa única tupla. Estas variables pueden ser tipos anclados.

Si la sentencia `SELECT` no recuperara ninguna tupla, las variables asumirían el valor `NULL`.

```
CREATE OR REPLACE FUNCTION Ejemplo25
    (
     IN prmTitle_id VARCHAR(6)
    )
    RETURNS void

    AS
    $$
    DECLARE
       price1 titles.price%TYPE;
       type1 titles.type%TYPE;
    BEGIN
       SELECT price, type INTO price1, type1
          FROM titles
          WHERE title_id = prmTitle_id;

       RAISE NOTICE 'La publicación % es de tipo % y vale $%', prmTitle_id,
                    trim(both from type1), price1;

       RETURN;
    END;
    $$
    LANGUAGE plpgsql

  SELECT Ejemplo25 ('TC7777');
```

Data Output | Messages | Notifications

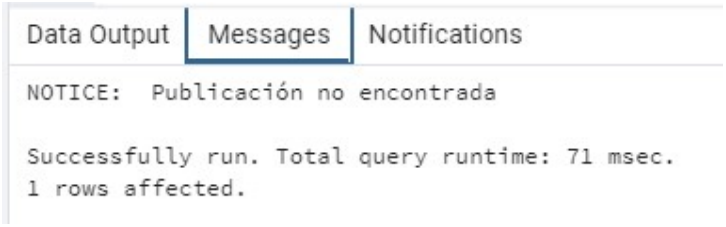
```
NOTICE:  La publicación TC7777 es de tipo trad_cook y vale $14.99

Successfully run. Total query runtime: 77 msec.
1 rows affected.
```

En cualquier caso los tipos de datos entre los atributos recuperados y las variables deben ser compatibles, de otra forma los datos pueden perder precisión o resultar truncados.

La variable FOUND

PostgreSQL proporciona una variable especial llamada `FOUND` que nos indica si un sentencia SQL encontró o no tuplas como resultado de su ejecución.

```
CREATE OR REPLACE FUNCTION Ejemplo26
    (
     IN prmTitle_id VARCHAR(6)
    )
    RETURNS void

    AS
    $$
    DECLARE
       price1 titles.price%TYPE;
          type1 titles.type%TYPE;
    BEGIN
       SELECT price, type INTO price1, type1
          FROM titles
          WHERE title_id = prmTitle_id;

       IF NOT FOUND THEN
          RAISE NOTICE 'Publicación no encontrada';
       END IF;
       RETURN;
    END;
    $$
    LANGUAGE plpgsql


  SELECT Ejemplo26 ('PC6565');
```



2.3. Sentencias SELECT Single-row que retornan una tupla completa

Sentencias SQL single row que retornan una tupla completa

Ver "*3.1. Sentencias SQL single row que retornan una tupla completa*" en el documento "*Relations y SQL*"

Record anchoring
Table-based record

Un record anchoring define una variable –que poseerá una estructura de record- en base al schema de una tupla completa de una tabla.
Cada campo corresponde a –y tiene el mismo nombre que- una columna en la tabla.

Se especifican anexando al nombre de la tabla el atributo `%ROWTYPE`. Por ejemplo:

```
DECLARE
   titleRec titles%ROWTYPE;
```

Luego de recuperados los datos, podemos acceder a cada uno de los componentes de la estructura de record usando una sintaxis de punto. Por ejemplo:

```
titleRec.price
```

Ejemplo 5

```
DECLARE
   titleRec titles%ROWTYPE;

BEGIN
   SELECT * INTO titleRec
      FROM titles
      WHERE title_id = prmTitle_id;

   RAISE NOTICE 'La publicación % es de tipo % y vale $%', prmTitle_id,
                trim(both from titleRec.type), titleRec.price;

   ...

END
```

Cuando la variable es un record asociado a una tupla completa, en la sentencia `SELECT` solo podemos recuperar la totalidad de las columnas de la tupla. No podemos recuperar una projection de tuplas.

2.4. Sentencias SELECT Single-row que retornan una Projection de una tupla

Ver "*3.2. Sentencias SQL single row que retornan una Projection de una tupla*" en el documento "*Relations y SQL*"

En este caso, y solamente para PL/pgSQL, tenemos una alternativa más usando declaraciones ancladas.

Tenemos que hacer un paso adicional: tenemos que definir una estructura de record que pueda recibir de la tupla sólo los atributos que necesitamos.

PostgreSQL llama a esto un **composite-type.**

Se crean con la sentencia `CREATE TYPE`. Luego de la cláusula `AS`, definimos la estructura del type de manera similar a como hacemos con la definición de una tabla:

```
CREATE TYPE publisherCT
   AS (
       pub_id CHAR(4),
       totalPrice numeric
      );
```

Este `TYPE` se define a nivel schema, fuera de la función PL/pgSQL.

Una vez creado el type, podemos "anclar" variables al mismo, como si se tratase prácticamente de una tabla.

El siguiente ejemplo ilustra este caso:

Ejemplo 6

```
CREATE TYPE titleCT
   AS (
       type char(12),
       price numeric
      );
```

```
...
DECLARE
   titleRec titleCT%ROWTYPE;
BEGIN
   SELECT type, price INTO titleRec
      FROM titles
      WHERE title_id = prmTitle_id;

   RAISE NOTICE 'La publicación % es de tipo % y vale $%',
     prmTitle_id, trim(both from titleRec.type), titleRec.price;
   RETURN;
END;
...
```

Data Output | Messages | Notifications

```
NOTICE:  La publicación BU1032 es de tipo business y vale $19.99

Successfully run. Total query runtime: 86 msec.
1 rows affected.
```

2.5. Records

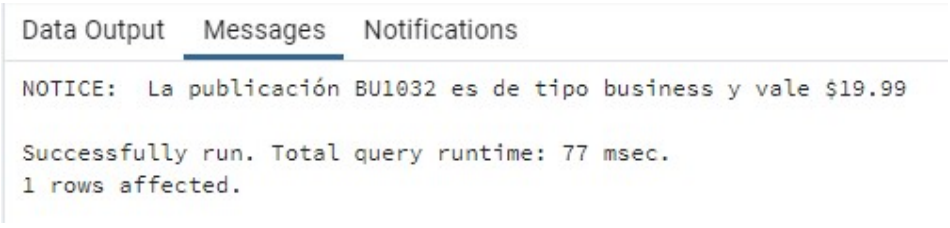
Sin embargo la solución más práctica es usar un tipo de dato **RECORD** datatype:

El tipo de dato `RECORD` no adhiere a un formato de relación determinado. Adopta la forma en el momento de la asignación.

El siguiente ejemplo ilustra su uso para una Projection de una tupla:

Ejemplo 7

```
CREATE OR REPLACE FUNCTION Ejemplo29
    (
     IN prmTitle_id VARCHAR(6)
    )
    RETURNS void

    AS
    $$
    DECLARE
       titleRec RECORD;
    BEGIN
       SELECT type, price INTO titleRec
          FROM titles
          WHERE title_id = prmTitle_id;

       RAISE NOTICE 'La publicación % es de tipo % y vale $%',
                    prmTitle_id, trim(both from titleRec.type),
                    titleRec.price;
       RETURN;
    END;
    $$
    LANGUAGE plpgsql
```



3. SELECT sobre expresiones

En T-SQL podemos consultar el valor de cualquier variable o expresión usando directamente la sentencia `SELECT`:

```
SELECT @<nombre-variable>
```

Esto es muy útil cuando estamos programando batches.

También, como veremos más adelante, T-SQL permite que la salida de un stored procedure sea una sentencia `SELECT`, así que tenemos la posibilidad de retornar cualquier valor que querramos disparando directamente un `SELECT` de este tipo.

SQL dinámico

Muchos motores de bases de datos permiten definir sentencias SQL a partir de una cadena de caracteres construida a partir de elementos variables.

Ejecutamos una cadena de caracteres como una sentencia SQL a través del comando `EXEC` (o `EXECUTE`):

```
EXEC (@Nombre-de-variable)
```

En el siguiente ejemplo disparamos un `SELECT` sobre una tabla y columna de salida cuyos nombres conocemos en tiempo de ejecución:

```
DECLARE @Cad Varchar(100)
DECLARE @nomTabla Varchar(30) = 'titles'
DECLARE @nomColumna VARCHAR(30) = 'title_id'

SET @Cad = '        SELECT ' + @nomColumna;
SET @Cad = @Cad + '    FROM ' + @nomTabla;
EXEC (@Cad)
```

Ejecutamos la cadena de caracteres como una sentencia SQL a través del comando `EXECUTE`:

```
EXECUTE Nombre-de-variable
```

El siguiente es el mismo ejemplo de SQL dinámico en una función PL/pgSQL:

```
 ...
DECLARE
    cad Varchar(100);
    nomTabla Varchar(30) := 'titles';
    nomColumna VARCHAR(30) := 'title_id';
 BEGIN

    cad := 'SELECT ' || nomColumna;
    cad = cad || ' FROM ' || nomTabla;

    RETURN QUERY EXECUTE cad;
 END
 ...
```