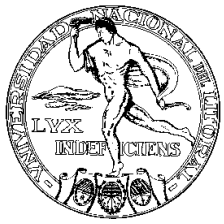


Universidad Nacional del Litoral

Facultad de Ingeniería y Ciencias Hídricas

Departamento de Informática

Bases de Datos

SQL: Guía de Trabajo Nro. 4

Stored Procedures

Parte 3: Ejercicios

Msc. Lic. Hugo Minni

2020

Ejercicios

Ejercicio 1.
Escriba un SP T-SQL (`obtenerPrecio`) que proporcione el precio de cualquier publicación para la cual se proporcione un código.
Testee su funcionamiento con un código de publicación. Por ejemplo, PS1372.

Guía para resolver el ejercicio

T-SQL

Entrada al procedimiento:
El procedimiento requiere de un parámetro de entrada.

Salida del procedimiento:
Como salida debe devolver un valor escalar.

Recordemos que T-SQL nos permite retornar directamente la salida de una sentencia `SELECT`, cualquiera sea la forma que la relación resultado tenga, así que podemos valernos de este tipo de solución.

Ejecución:
Finalmente testeamos la ejecución del procedimiento proporcionando el parámetro de entrada. Recordemos que podemos especificar el parámetro por posición o por nombre.

PL/pgSQL

Salida del procedimiento:

Recordemos que una función PL/pgSQL NO PERMITE el OUTPUT directo de una sentencia `SELECT`.
Podemos retornar el precio como una relación unaria. En tal caso deberemos establecer como valor de retorno de la función un `setof` del tipo de dato adecuado.
(Ver "*5.1. Retornar un relación unaria*" en la "*Guía de Trabajo Nro. 4 - Parte 2*").

Ejercicio 2.

Escriba una función PL/pgSQL que dado un código de almacén (`stor_id`) y un número de factura (`ord_num`), retorne la fecha de dicha venta.

Ejecútela para los siguientes parámetros: código de almacén `7067`, número de orden `P2121`.

Guía para resolver el ejercicio

PL/pgSQL

Recordemos la tabla de Ventas en Pubs:

Sales		
stor_id	char(4)	<pk>
ord_num	varchar(20)	<pk>
title_id	varchar(6)	<pk>
ord_date	datetime	
qty	smallint	
payterms	varchar(12)	

Para el código de almacén `7067 y` número de orden `P2121` tenemos:

```
SELECT * FROM sales
WHERE Stor_id = '7067' AND Ord_num = 'P2121'
```

Data Output | Explain | Messages | History

	stor_id character(4)	ord_num character varying(20)	ord_date date	qty smallint	payterms character varying(12)	title_id character varying(6)
1	7067	P2121	1992-06-15	40	Net 30	TC3218
2	7067	P2121	1992-06-15	20	Net 30	TC4203
3	7067	P2121	1992-06-15	20	Net 30	TC7777

Para recuperar una única fecha podemos usar DISTINCT.

Podemos resolver la salida usando setof.

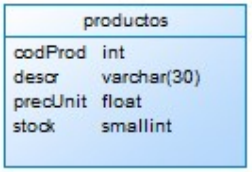
Ejercicio 3.

Recordemos el esquema de tablas que creamos en la Guía de Trabajo Nro. 2 para realizar ejercicios de manipulación de datos:

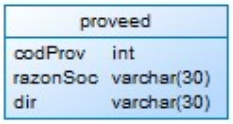
Sobre SQL Server:

```
CREATE TABLE cliente
  (
   codCli     int           NOT NULL,
   ape        varchar(30)   NOT NULL,
   nom        varchar(30)   NOT NULL,
   dir        varchar(40)   NOT NULL,
   codPost    char(9)       NULL DEFAULT 3000
  )
```

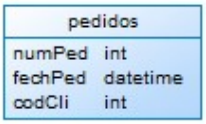

```
CREATE TABLE productos
  (
   codProd    int           NOT NULL,
   descr      varchar(30)   NOT NULL,
   precUnit   float         NOT NULL,
   stock      smallint      NOT NULL
  )
```



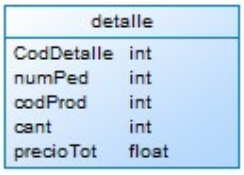

```
CREATE TABLE proveed
  (
   codProv    int           IDENTITY(1,1),
   razonSoc   varchar(30)   NOT NULL,
   dir        varchar(30)   NOT NULL
  )
```




```
CREATE TABLE pedidos
  (
   numPed     int           NOT NULL,
   fechPed    datetime      NOT NULL,
   codCli     int           NOT NULL
  )
```




```
CREATE TABLE detalle
  (
   codDetalle       int           NOT NULL,
   numPed           int           NOT NULL,
   codProd          int           NOT NULL,
   cant             int           NOT NULL,
   precioTot        float         NULL
  )
```



Sobre PostgreSQL:

```
CREATE TABLE cliente
  (
   codCli     int           NOT NULL,
   ape        varchar(30)   NOT NULL,
   nom        varchar(30)   NOT NULL,
   dir        varchar(40)   NOT NULL,
   codPost    char(9)       NULL DEFAULT 3000
  )


CREATE TABLE productos
  (
   codProd    int           NOT NULL,
   descr      varchar(30)   NOT NULL,
   precUnit   float         NOT NULL,
   stock      smallint      NOT NULL
  )


CREATE TABLE proveed
  (
   codProv    SERIAL,
   razonSoc   varchar(30)   NOT NULL,
   dir        varchar(30)   NOT NULL
  )


CREATE TABLE pedidos
  (
   numPed     int           NOT NULL,
   fechPed    date          NOT NULL,
   codCli     int           NOT NULL
  )



CREATE TABLE detalle
  (
   codDetalle       int           NOT NULL,
   numPed           int           NOT NULL,
   codProd          int           NOT NULL,
   cant             int           NOT NULL,
   precioTot        float         NULL
  )
```

Trabajaremos con las tablas Productos y Detalle, en T-SQL.

Cargue el siguiente lote de prueba en la tabla de Productos:

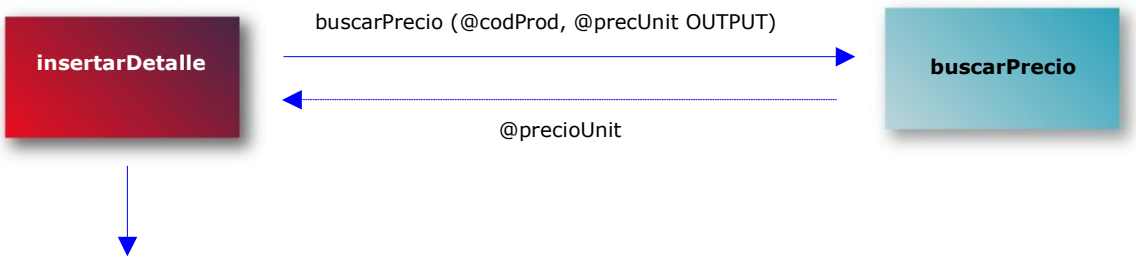
```
(10, "Articulo 1", 50, 20)
(20, "Articulo 2", 70, 40)
```

El valor que almacena en la columna `precioTot` de la tabla `Detalle` se calcula en función de la cantidad pedida de un producto (columna `cant`) y su precio unitario (columna `precUnit` en la tabla `productos`).

Se desea crear un procedimiento almacenado (`insertarDetalle`) que reciba como parámetros código de detalle (codDetalle), número de Pedido (numPed), código de producto (codProd) y cantidad vendida (cant) e inserte una nueva fila en la tabla detalle.

Para obtener el valor correspondiente a la columna `precioTot`, el procedimiento principal debe invocar a un procedimiento auxiliar (`buscarPrecio`) que retorne el precio unitario correspondiente al producto recibido como parámetro en `insertarDetalle`.

El siguiente esquema ilustra la idea:




```
INSERT Detalle Values(....)
```

Guía para resolver el ejercicio

T-SQL

Siempre nos conviene desarrollar los procedimientos auxiliares (o de utilidad) antes desarrollar el procedimiento principal.

En este caso comenzamos desarrollando el procedimiento **buscarPrecio**.

Salida del procedimiento:
Si bien T-SQL nos permite retornar directamente la salida de una sentencia `SELECT`, esta solución está pensada para cuando el resultado debe ser enviado a una aplicación cliente en un entorno cliente/servidor (por ejemplo, hacia una aplicación Java).
Aquí debemos implementar la solución a través de un OUTPUT parameter.

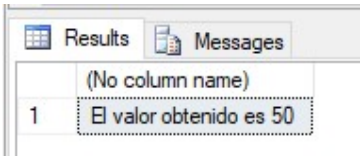
Recordemos que, si no necesitamos un return value, podemos finalizar el procedure con una simple sentencia `RETURN` sin especificar valor alguno.
Esta sentencia `RETURN` sin valor es lo mismo que especificar `RETURN 0` (finalización normal).

Siempre que desarrollemos un procedimiento debemos probarlo.
En T-SQL lo probamos usando un batch.
Si el procedimiento posee OUTPUT parameters como en este caso, debemos definir variables auxiliares en el batch que **reciban** a estos parámetros de salida.

En nuestro ejemplo, podemos probar el procedimiento buscarPrecio de la siguiente manera:

```
DECLARE  @PrecioObtenido FLOAT
EXECUTE buscarPrecio 10, @PrecioObtenido OUTPUT
SELECT 'El valor obtenido es ' + CONVERT(VARCHAR, @PrecioObtenido)
```

Teniendo en cuenta los datos insertados en la tabla `Producto`, para el producto 10 debemos obtener el precio 50.



Una vez que comprobamos que el procedimiento auxiliar funciona, comenzamos a desarrollar el procedimiento principal.

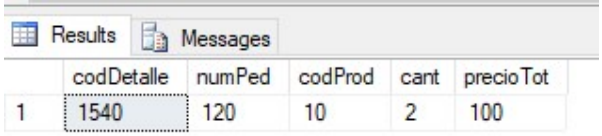
En el procedimiento *insertarDetalle* definimos los parámetros de entrada y utilizamos una sintaxis prácticamente idéntica a la del batch para invocar al procedimiento buscarPrecio.

Una vez obtenido el precio, solo resta calcular el precio total y realizar el `INSERT`.

Finalmente, podemos probar el procedure *insertarDetalle* con los siguiente parámetros:

```
CodDetalle    1540
NumPed        120
CodProd       10
Cant          2
```

Deberíamos obtener en la tabla *Detalle* una nueva fila como la siguiente:



Ejercicio 4

Queremos mejorar el procedure *insertarDetalle*

Si sucediese que el código de producto (*codProd*) que recibe el procedure no exista en la tabla de productos, queremos que esta situación sea indicada en un mensaje y que la sentencia `INSERT` no se ejecute.

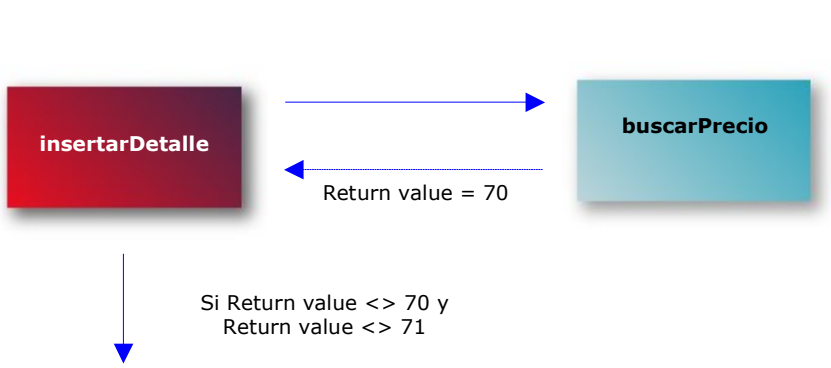
De manera análoga, si sucediese que el producto no tuviese definido precio (en este caso eso no sería posible ya que la columna es NOT NULL), queremos que esta situación sea indicada en un mensaje y que la sentencia `INSERT` tampoco se ejecute.

Guía para resolver el ejercicio

Ver "*4.1.1. Return value personalizado*" en la "*Guía de Trabajo Nro. 4 - Parte 2*"

T-SQL

Podemos ubicar este código de validación en nuestro procedure auxiliar *buscarPrecio* y valernos de diferentes return values personalizados para indicar al procedure invocante lo que está sucediendo. Por ejemplo:



Return values definidos por el desarrollador

RETURN **70** significa: El producto no existe.
RETURN **71** significa: El producto no tiene precio definido.

```
INSERT Detalle Values(....)
```

Una forma rápida de averiguar si el producto existe es intentar recuperar el precio del producto e inmediatamente consultar la variable del sistema `@@rowcount.`

También podemos hacer una consulta con EXISTS, etc., como ya aprendimos.

Una forma rápida de averiguar si el producto posee precio nulo es intentar recuperar el precio del producto y consultar si la variable obtenida `IS NULL.`
Por supuesto, podemos también realizar una consulta adicional para averiguar esto. Incluso podríamos programar dos procedimientos almacenados extra que lleven a cabo estas validaciones atómicas.

El procedure principal debe recuperar este valor de retorno como ya aprendimos:

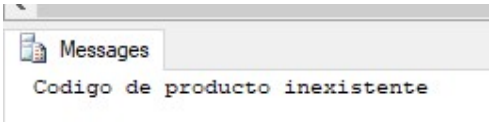
```
EXECUTE <@Variable-local-de-tipo-Int> = <Nombre-SP>
```

...y llevar a cabo el `INSERT` solamente si el valor de retorno es diferente de 70 y 71.

Finalmente testeamos `insertarDetallev2` proporcionando un código de producto inexistente:

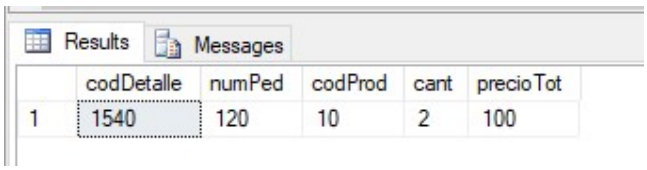
```
CodDetalle    1540
NumPed        120
CodProd       99
Cant          2
```

Debemos obtener un mensaje como el siguiente:



... y la inserción no debe llevarse a cabo:

```
SELECT * FROM detalle
```



PL/pgSQL

Recordemos que podemos obtener la cantidad de filas afectadas por una operación usado `GET DIAGNOSTICS` con la keyword `ROW_COUNT`:

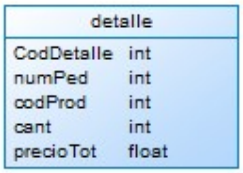
```
GET DIAGNOSTICS vCantFilas = ROW_COUNT;
```

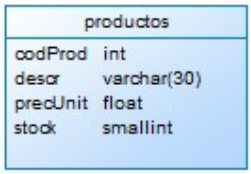
En PL/pgSQL no tenemos el recurso de RETURN VALUE, pero podemos utilizar OUTPUT parameters que cumplan esa función.

Ejercicio 5.

Seguimos trabajando con el esquema de tablas que creamos en la Guía de Trabajo Nro. 2 para realizar ejercicios de manipulación de datos.

Trabajaremos nuevamente con las tablas *Productos* y *Detalle*, en T-SQL.





Vamos a insertar un nuevo producto:

```
INSERT productos
   VALUES (100, 'Articulo 3', 30, 10)
```

Queremos registrar el pedido de cinco unidades de este producto.

Para ello debemos disminuir su stock en la tabla *Productos* e insertar inmediatamente los datos del pedido en la tabla *Detalle*

Nuestro detalle a insertar sería entonces:

```
CodDetalle    1200
NumPed        1108
CodProd       100
Cant          5
```

Debemos considerar que estas dos operaciones deben tener éxito o fracasar **juntas**.
No puede suceder que descontemos 5 unidades de stock y no registremos el pedido.
Tampoco puede suceder que registremos el pedido y no descontemos las 5 unidades de stock.

Escriba un batch T-SQL que lleve a cabo la operación transaccionada.

A fin de facilitar la realización del ejercicio, especifique un valor NULL para la columna *precioTot*.

Guía para resolver el ejercicio

T-SQL

Ver "*8. Demarcación de transacciones en SQL*" en la "*Guía de Trabajo Nro. 4 - Parte* 2".

Debemos obtener como resultado una nueva fila de Detalle:

Results | Messages

	codDetalle	numPed	codProd	cant	precioTot
1	1540	120	10	2	100
2	1200	1108	100	5	NULL

...y a la vez el stock disminuido en 5 unidades:

Results | Messages

	codProd	descr	precUnit	stock
1	10	Articulo 1	50	20
2	20	Articulo 2	70	40
3	100	Articulo 3	30	5