Universidad Nacional del Litoral

Facultad de Ingeniería y Ciencias Hídricas
Departamento de Informática

Bases de Datos

Guía de Trabajo Nro. 6

T-SQL: Procedimientos almacenados

Msc. Lic. Hugo Minni
2008

Stored procedures

Los *stored procedures* (*SP* en adelante) son *batches* que SQL Server almacena como un objeto de la base de datos y que pueden luego invocarse para su ejecución. Los SPs proveen una serie de ventajas, tales como:
- Mayor control sobre los datos.
- Acceso directo a complejas operaciones con datos.
- Mejor performance.

Los SP pueden recibir y retornar *parámetros*. Los *parámetros de entrada* permiten escribir procedimientos más generales. Los *parámetros de salida* generalmente retornan valores a un SP invocante o algún programa de aplicación cliente.

Determinar si un SP existe

```
sp_helptext @objname = 'sp_Prueba'
```

Creación y ejecución de SPs

La sintaxis básica de creación de SPs con *parámetros de entrada* es la siguiente:

```
CREATE PROC[EDURE] <Nombre-SP>
   [ (@Nombre-Parametro Tipo-de-dato [,...]) ]
   AS
      <Lógica de datos>
RETURN
```

La sintaxis para ejecutar SPs es la siguiente:

```
[EXEC[UTE]] Nombre-parametro [@Nombre-Parametro =] Valor-del-parametro [,...]
```

La cláusula `EXECUTE` solo es opcional cuando la sentencia de ejecución del SP es la primer sentencia de un batch.

1) Cree un SP (`sp_ObtenerPrecio`) sin parámetros de entrada que liste el precio de la publicación con código `"PS2091"`. Ejecútelo.

Modificar Stored Procedures

```
Alter Procedure spErrorNoFatal
   (@Col2 Int = Null)
   As
      Insert Prueba
         Values (@Col2)
      If @@Error <> 0
         Begin
            Print 'Error No fatal'
         End
```

Ambito de los SPs

Si un SP se crea en una base de datos de usuario, éste estará solo disponible en esa base de datos.
Para hacer que un SP esté disponible en todas las bases de datos hay que:
- Hacer que su nombre comience con `sp_`
- Crearlo en la base de datos `master`.

Obtener información de SPs

Los SP son objetos que pertenecen a la base de datos en la que están definidos. Varias tablas del sistema hacen referencia a los SPs, incluyendo:

- `sysobjects` (Una fila por SP)
- `syscolumns` (Una fila por parámetro involucrado en un SP)
- `sysdepends` (Una fila por tabla, vista o procedure referenciado en un SP)
- `syscomments` (Una fila por bloque de 255 caracteres de código fuente de un SP)

2) El SP del sistema `sp_help` brinda información sobre objetos de la base de datos actual. Ejecútelo transfiriendo el nombre de su SP recién creado como parámetro a fin de confirmar su creación.

3) La tabla del sistema `sysobjects` en `Pubs` posee información sobre todos los objetos de la base de datos. Consulte por las columnas `name`, `ID` y `type` para el nombre `sp_ObtenerPrecio`.

4) Obtenga todos los SP definidos en la base de datos `Pubs`. Para ello liste la columna `name` en `sysobjects` para columna `type = "P"`.

5) El SP del sistema `sp_helptext` lista el código fuente de cualquier SP que se le pase como parámetro. Liste el código fuente de `sp_ObtenerPrecio`.
Liste también el del SP del sistema `sp_help` que utilizó recientemente.

En general es conveniente almacenar el código fuente de los SPs (también definiciones de triggers, vistas y tablas) como *scripts* corrientes a modo de copia de resguardo en unidades de disco. Esto permite recuperar el código fuente de los mismos en el evento de una caída del servidor y regenerar en forma exacta los objetos perdidos.

Eliminación de SPs

Los SPs se pueden eliminar por medio de la `sentencia DROP PROC[EDURE] <Nombre-SP>`.

Parámetros de entrada

Para que un SP pueda recibir un parámetro, este debe ser declarado como parte de la sentencia `CREATE PROCEDURE`. El parámetro puede ser utilizado luego como una variable en el procedure.

Si el parámetro va a ser utilizado para definir –por ejemplo- una condición en una cláusula `WHERE`, el mismo debe coincidir en su tipo de dato con el tipo de dato de la columna a comparar. La performance de los SPs es superior si el tipo de dato del parámetro es exactamente el mismo que el de la columna interviniente en la cláusula `WHERE`.

6) Reescriba el SP del `Ejercicio 1` a fin de que proporcione el precio de cualquier publicación para la cual se proporcione un código. (`sp_ObtenerPrec2`). Ejecútelo a fin de consultar el precio de la publicación `PS1372`.

Especificación de parámetros

Si el SP maneja más de un parámetro, estos pueden ser transferidos al mismo bajo dos modalidades:

Por posición

```
EXECUTE <Nombre-SP>
   valor-de-parametro1,
   valor-de-parametro2,
   valor-de-parametro3,...
```

Por Nombre

```
EXECUTE <Nombre-SP>
   @Nombre-Parametro = valor-de-parametro,
   @Nombre-Parametro = valor-de-parametro, ...
```

Los dos estilos de especificación de parámetros se pueden combinar en una misma invocación, pero una vez que se utiliza la modalidad de *especificación por nombre*, el resto de los parámetros deben ser especificados por nombre.

La especificación de parámetros por nombre es mucho mejor desde el punto de vista de la claridad del código, sobre todo cuando el SP posee gran cantidad de parámetros. Sin embargo, la ejecución de los SP es más veloz cuando los parámetros son especificados por posición. Se puede obtener lo mejor de ambos mundos especificando los parámetros por posición y documentando las llamadas complejas de la siguiente forma:

```
EXECUTE <Nombre-SP>
   Valor-de-Parametro1,     -- @Nombre-Parametro
   Valor-de-Parametro2,     -- @Nombre-Parametro
   Valor-de-Parametro3      -- @Nombre-Parametro
```

Los parámetros de tipo `char` o `varchar` no necesitan comillas salvo que:
- Incluyan signos de puntuación.
- Consistan en una palabra reservada
- Incluyan solo números

Las fechas deben especificarse como strings entre comillas ya que incluyen el símbolo `"/"`.

7) Cree un SP (`sp_VerVenta`) que obtenga la fecha de venta para aquellas ventas para las que se especifique el código de almacén (`stor_id`), número de orden y cantidad. Ejecútela para los siguientes parámetros: código de almacén `7067`, número de orden `P2121` y cantidad `40`, primero especificados *por posición* (documentados) y luego *por nombre*.

Parámetros Opcionales.

Los parámetros de entrada especificados hasta el momento han sido todos *requeridos*. En otras palabras, la ejecución falla si se omite alguno. Es posible hacer que un parámetro de entrada sea opcional si se especifica un valor *default* en el cuerpo del procedure en el caso de su omisión. Para ello, la declaración del parámetro debe poseer la siguiente sintaxis:

```
(@Nombre-Parametro Tipo-de-dato = Valor-por-omisión)
```

Los parámetros requeridos deben ir siempre primero en la lista de parámetros, caso contrario debemos hay que pasarlos igual como null.

8) Reescriba el SP del `Ejercicio 6` a fin de que la especificación del código de publicación sea opcional y establecido a `NULL` por omisión. (`sp_ObtenerPrec3`)
Ejecútelo proporcionando el código de publicación `PS1372` y sin proporcionar parámetros.

9) Reescriba el SP del `Ejercicio 8` de manera tal que –si el usuario omite el parámetro- se le notifique esta situación por medio de un mensaje informativo con la `forma "El SP sp_ObtenerPrec4 requiere del paramtero title_id"` y se finalice la ejecución del mismo.
Ejecútelo con y sin parámetros.

Parámetros de salida

Los *parámetros de salida* se definen agregando la cláusula `OUTPUT` a continuación de una definición de parámetro:

```
(@Nombre-Parametro Tipo-de-dato OUTPUT)
```

Los parámetros de salida son valores que un SP retorna a otro SP externo (*outer procedure*) o aplicación cliente invocante. Estos valores están destinados a un programa, no al usuario.

Considere las tablas creadas en la Guía de ejercicios Nro. 2:

```
productos
codprod      int          not null,
descr        varchar(30)  not null,
precUnit     money        not null,
Stock        smallint     not null

Detalle
CodDetalle   Int          not null,
NumPed       Int          not null,
CodProd      Int          Not Null
Cant         Int          Not Null,
PrecioTot    money        Null
```

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

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

El precio total de la tabla `Detalle` se calcula en función de la cantidad pedida de un producto y su precio unitario.
Si se necesita insertar un nuevo detalle de pedido en la tabla `detalle`, el precio total puede calcularse en función del precio unitario en la tabla `Productos`. Un SP (`sp_BuscaPrecio`) podría obtener el precio unitario y proveerlo como parámetro de salida a un SP invocante (`sp_InsertaDetalle`) a fin de que implemente finalmente el alta en la tabla `detalle`.

El siguiente código -por ejemplo- implementa el procedimiento que busca el precio del artículo pedido:

```
CREATE PROCEDURE sp_BuscarPrecio
   (@CodProd int,                  -- Parametro de entrada
    @PrecUnit money  OUTPUT)        -- Parametro de salida
   AS
      SELECT @PrecUnit = PrecUnit
         FROM Productos
         WHERE CodProd = @Codprod
      RETURN
```

A fin de ejecutar este SP interactivamente hace falta definir una variable que reciba el parámetro de salida retornado por `sp_BuscarPrecio`:

```
DECLARE  @PrecioObtenido MONEY
EXECUTE sp_BuscarPrecio 10, @PrecioObtenido OUTPUT
SELECT @PrecioObtenido "Par. de salida"
```

El procedimiento invocante (*outer procedure*) completo se lista a continuación:

```
CREATE PROCEDURE sp_InsertaDetalle
   (@CodDetalle Int,        -- Parametro de entrada a sp_InsertaDetalle
    @NumPed Int,          -- Parametro de entrada a sp_InsertaDetalle
    @CodProd int,           -- Parametro de entrada a sp_InsertaDetalle y al inner proc
    @Cant  Int)             -- Parametro de entrada a sp_InsertaDetalle
    AS
      DECLARE  @PrecioObtenido MONEY   --Parametro de salida del inner procedure
      EXECUTE sp_BuscarPrecio @CodProd, @PrecioObtenido OUTPUT
      INSERT Detalle Values(@CodDetalle, @NumPed, @CodProd, @Cant,
                            @Cant * @PrecioObtenido)
      If @@RowCount = 1
         PRINT "Se inserto una fila"
      RETURN
```

11) Ejecute `sp_InsertaDetalle` a fin de insertar el siguiente detalle:

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

Status de retorno de un SP

El *status de retorno* de un SP describe el *estado* del procedimiento al finalizar el mismo. Sirve para indicar al programa invocante el motivo por el cual el procedimiento finalizó su ejecución. Bajo condiciones normales, un SP finaliza su ejecución cuando alcanza el final del código del mismo o cuando ejecuta una sentencia `RETURN`. **Esta finalización normal retorna un status de `0`.**

Microsoft reserva un bloque de números de `-1` a `-99` para identificar *status de error*. Solo los primeros 14 valores poseen significado:

Valor	Significado
-1	Falta objeto
-2	Error de tipo de dato
-3	El proceso fue elegido como víctima de deadlock.
-4	Error de permisos
-5	Error de sintaxis
-6	Error de usuario de miscelánea
-7	Error de recursos (sin espacio, por ejemplo)
-8	Problema interno no fatal.
-9	El sistema ha alcanzado su límite.
-10	Inconsistencia interna fatal.
-11	Inconsistencia interna fatal.
-12	Tabla o índice corrupto.
-13	Base de datos corrupta.
-14	Error de hardware.

El valor del status de retorno de un SP puede capturarse mediante la siguiente sintaxis:

```
EXECUTE <@Variable-local-de-tipo-Int> = <Nombre-SP>
   Valor-de-Parametro1,
   Valor-de-Parametro2,
   Valor-de-Parametro3
```

12) Ejecute nuevamente `sp_InsertaDetalle` con los datos del `Ejercicio 11` pero omitiendo el valor de `Cantidad`. Capture y muestre el *status de retorno* del SP.

Seteo de valores de retorno personalizados

Si un SP interno -invocado por otro SP- genera un determinado error que no depende del sistema (errores de reglas de negocios, por ejemplo) el desarrollador puede notificar de esta situación al SP invocante retornando un código de error personalizado que el SP invocante pueda interpretar.
Estos códigos de error pueden ser cualquier valor de tipo `Int` fuera de los reservados (`-99` a `-1`) y se especifican a continuación de la(s) cláusula(s) `RETURN` en el SP invocado.

En el siguiente ejemplo se ha redefinido el SP de inserción de detalles de pedidos a fin de que contemple la posibilidad de que no se encuentre el producto a pedir o que su precio sea `NULL` (aunque en este caso la definición de la tabla no lo permite).

```
CREATE PROCEDURE sp_BuscarPrecio2
   (@CodProd int,                  -- Parametro de entrada
    @PrecUnit money  OUTPUT)        -- Parametro de salida
   AS
      SELECT @PrecUnit = PrecUnit
         FROM Productos
         WHERE CodProd = @Codprod
      IF @@RowCount = 0
         RETURN 70             -- No se encontro el producto
       IF @PrecUnit IS NULL
         RETURN 71             -- El producto existe pero su precio es NULL
      RETURN 0                 -- El producto existe y su precio no es NULL
```

```
CREATE PROCEDURE sp_InsertaDetalle2
   (@CodDetalle Int,        -- Parametro de entrada a sp_InsertaDetalle2
    @NumPed Int,          -- Parametro de entrada a sp_InsertaDetalle2
    @CodProd int,           -- Parametro de entrada a sp_InsertaDetalle2 y al inner proc
    @Cant  Int)             -- Parametro de entrada a sp_InsertaDetalle2
    AS
      DECLARE  @PrecioObtenido MONEY   --Parametro de salida del inner procedure
      DECLARE @StatusRetorno Int
      EXECUTE @StatusRetorno = sp_BuscarPrecio2 @CodProd, @PrecioObtenido OUTPUT
      IF @StatusRetorno != 0
         BEGIN
            IF @StatusRetorno = 70
               RAISERROR ("Codigo de producto inexistente", 16, 1)
            ELSE
               IF @StatusRetorno = 71
                  RAISERROR ("El producto %d no posee precio", 16, 1, @CodProd)
               ELSE
                  RAISERROR ("Error en el SP sp_BuscarPrecio2", 16, 1)
                  RETURN 99
         END
      INSERT Detalle Values(@CodDetalle, @NumPed, @CodProd, @Cant,
                            @Cant * @PrecioObtenido)
      IF @@Error != 0
         RETURN 77
      If @@RowCount = 1
         PRINT "Se inserto una fila"

      RETURN 0
```

La sentencia `RAISERROR` –a diferencia de `PRINT`- se utiliza exclusivamente para retornar mensajes de error críticos al usuario. `RAISERROR` modifica el valor de `@@error`.

13) Ejecute `sp_InsertaDetalle2` a fin de insertar el siguiente detalle, correspondiente a un producto inexistente:

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

SP en INSERTSs

En la Guía de Ejercitación Nro. 2 se revisó un tipo de sentencia `INSERT` con la siguiente sintaxis:

```
INSERT <tabla-destino>
   SELECT *
      FROM <tabla-origen>
      WHERE <condicion>
```

en donde el lote de datos a insertar era obtenido de un *result set* resultado de un acceso a datos. SQL Server permite que este lote de datos pueda ser proporcionado también por un SP:

```
INSERT <tabla-destino>
   EXECUTE Nombre-SP
```

14) Cree en pubs una tabla -llamada `Editoriales`- análoga a la tabla `Publishers` en `Pubs`, pero

únicamente con las columnas `pub_id` y `pub_name` y sin datos. Para la creación de la estructura puede utilizar un `Select pub_id, pub_name into Editoriales` para una condición que jamás sea verdadera.

15) Inserte en la tabla `Editoriales` todas las filas de la tabla `Publishers` en `Pubs`. Implemente la inserción a través de la estructura `INSERT..EXECUTE` expuesta.