



**Carrera:** Técnico Superior en Programación

**Materia:** Laboratorio de Computación III

**Tema:** Procedimientos almacenados

---

## Procedimientos almacenados

Los procedimientos almacenados son rutinas que se encuentran almacenadas en el servidor de base de datos bajo un nombre que los identifica.

Los mismos pueden devolver resultados de tablas, mensajes, lanzar excepciones o bien ejecutar sentencias de manipulación y definición de datos.

La sintaxis básica para la creación de un procedimiento almacenado es la siguiente:

```
CREATE PROCEDURE sp_NombreProcedimiento [(lista_de_parámetros)] AS
BEGIN
[CONSULTA SQL]
END
```

Ejemplo:

```
CREATE PROCEDURE sp_ObtenerTodosLosRegistros
AS
BEGIN
    SELECT * FROM TABLA_X
END
```

La sintaxis para la modificación de un procedimiento almacenado existente es la siguiente:

```
ALTER PROCEDURE sp_NombreProcedimiento [(lista_de_parámetros)] AS
BEGIN
[CONSULTA SQL]
END
```

Ejemplo:

```
ALTER PROCEDURE sp_ObtenerTodosLosRegistros
AS
BEGIN
    SELECT * FROM TABLA_X WHERE estado = 1
END
```

La sintaxis para la eliminación de un procedimiento almacenado es la siguiente:

```
DROP PROCEDURE sp_NombreProcedimiento
```

Por último, la ejecución o llamada a un procedimiento almacenado se realiza mediante la sentencia EXEC

```
EXEC sp_NombreProcedimiento
```

Ejemplo:

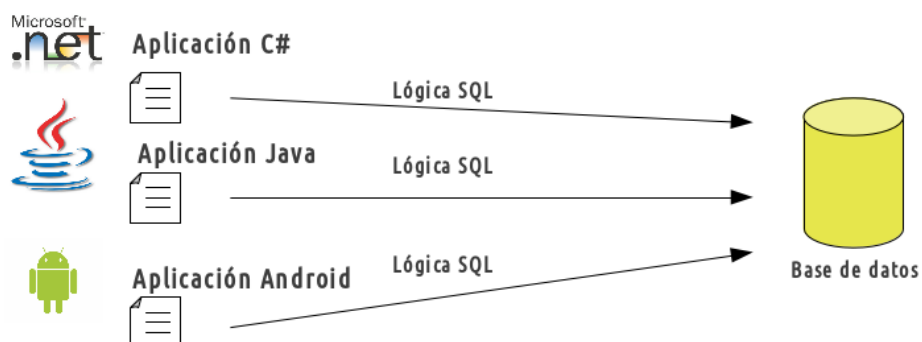
```
EXEC sp_ObtenerTodosLosRegistros
```

Como los procedimientos almacenados son muy similares a las funciones, pueden recibir parámetros. En este caso, vamos a ver el código para crear un procedimiento almacenado que recibe parámetros para su ejecución.

```
CREATE PROCEDURE sp_ObtenerRegistros(  
    @pEstado BIT  
)  
AS  
BEGIN  
    SELECT * FROM TABLA_X WHERE estado = @pEstado  
END  
  
CREATE PROCEDURE sp_InsertarRegistro(  
    @pNombre VARCHAR(20),  
    @pFecha DATETIME  
)  
AS  
BEGIN  
    INSERT INTO TABLA_X(campo1, campo2) VALUES(@pNombre, @pFecha)  
END
```

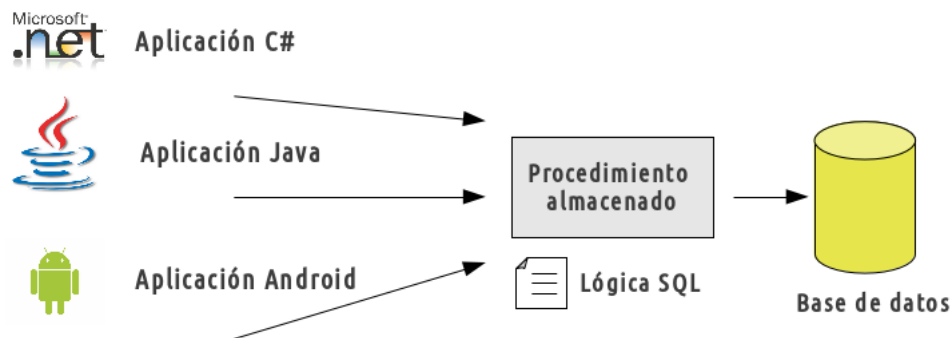
## Ventajas

Las ventajas de utilizar procedimientos almacenados son muchas. En principio, hay que tener en cuenta que nuestro sistema desarrollado bien podría estar en otro servidor distinto al que contiene la base de datos, sin embargo, como los procedimientos almacenados se encuentran físicamente en la base de datos posee un acceso inmediato y directo a los datos que necesita manipular de manera que sólo deberá enviar los resultados de la operación al usuario. De ésta manera se ahorra una gran cantidad de comunicaciones de datos entrantes y salientes. Otra de las ventajas fundamentales de los procedimientos almacenados es que la lógica del procedimiento se encuentra en la base de datos por lo que se evita tener que incorporar dicha lógica embebida en el código de la aplicación.



En la imagen superior se puede notar como la lógica de la consulta SQL se encuentra en el código fuente de la aplicación. De manera que una actualización en dicha lógica significará la modificación y recompilación de cada una de las aplicaciones. En caso de las aplicaciones de

escritorio y celular, probablemente, se deberá generar una nueva versión de la misma para ser reemplazada por la anterior incorporando la modificación.



En cambio, en la siguiente imagen se puede observar como es el procedimiento almacenado quien incorpora la lógica de la consulta SQL. Cualquier cambio que deba hacerse en dicha lógica se podrá reemplazar sólo en el procedimiento almacenado, abstrayéndola de las aplicaciones y sus lenguajes de programación.

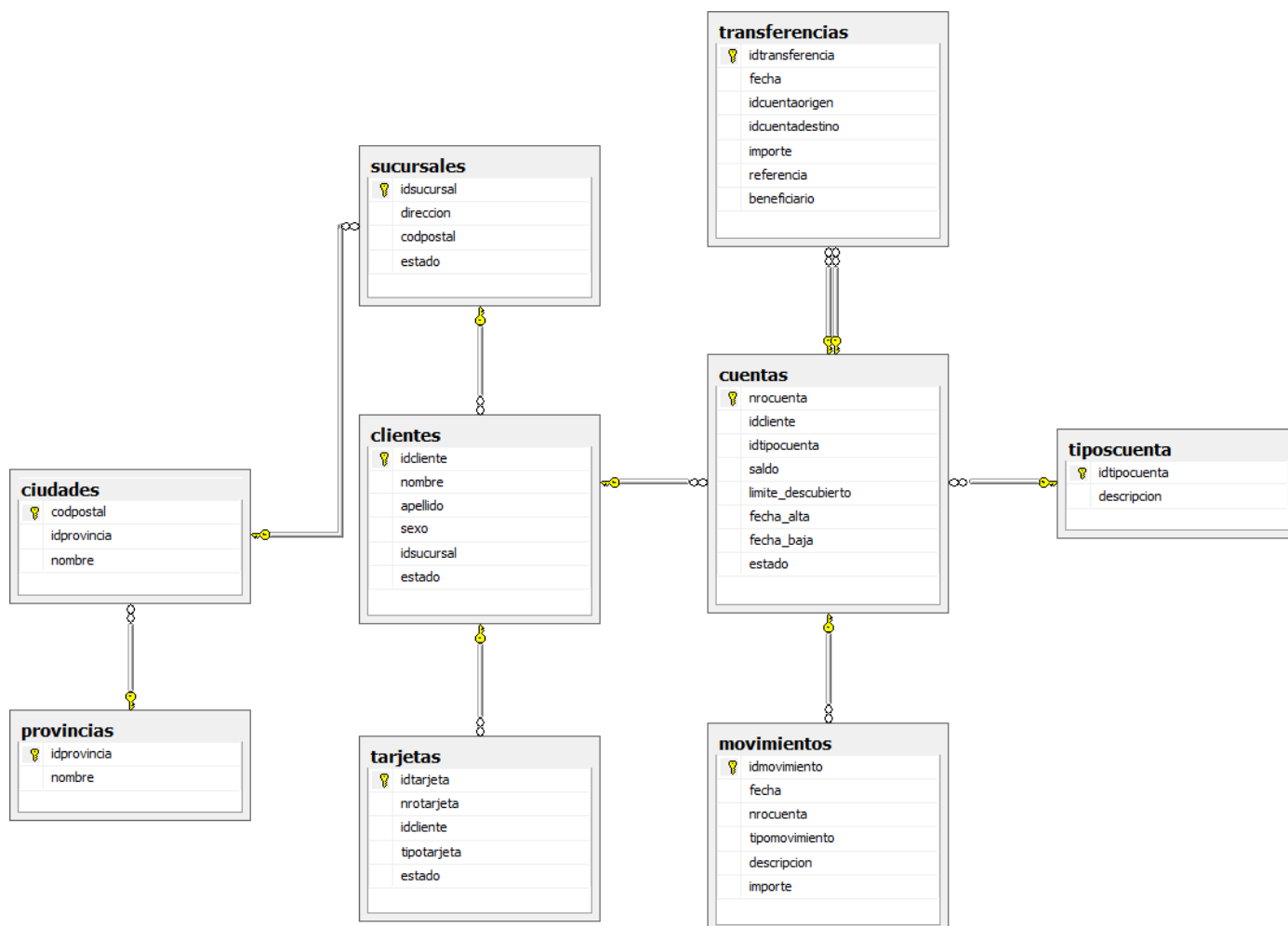
En el caso de los datos que devuelve, será cuestión de actualizar no sólo el procedimiento almacenado sino también la aplicación que deberá evaluar dichos datos.

De ésta manera la performance de nuestras aplicaciones mejora marcadamente. Además, cada vez que enviamos una consulta SQL embebida desde nuestra aplicación, el motor de base de datos deberá, antes de ejecutarla, chequear la sintaxis. Mediante la utilización de Stores Procedures, sólo deberá chequear la sintaxis de la llamada al procedimiento y sus parámetros, pero no deberá chequear la sintaxis del código SQL del mismo ya que se encuentra precompilado en la base de datos.

Otra de las ventajas es el manejo de la seguridad y encapsulamiento de los datos. Si le permitimos a un usuario que se conecta a nuestra base de datos sólo a ejecutar ciertos procedimientos almacenados, nos aseguraremos de que no tendrá la posibilidad de leer o modificar otras tablas o elementos que no debería acceder.

Por ejemplo, un usuario de base de datos podría ejecutar un procedimiento almacenado que realiza un update sobre una tabla X pero, sin utilizar el Stored Procedure, no tener acceso a dicha tabla.

## Un caso práctico



La siguiente base de datos almacenará la información de un banco, para ello cuenta con registros de sucursales y clientes. También se almacenan las tarjetas que poseen los clientes y sus cuentas en el banco. Cada cuenta a su vez puede sufrir movimientos, tales como Débitos o Créditos, además de por supuesto realizar transferencias entre cuentas. Utilizaremos la siguiente base de datos para ejemplificar el funcionamiento de los procedimientos almacenados.

Supongamos que queremos obtener un listado de clientes de una determinada sucursal. Esto quiere decir que, por ejemplo, voy a querer obtener un listado de clientes que pertenezca a la sucursal 1.

Lo primero que hay que determinar es el tipo de consulta que será nuestro procedimiento. En este caso sabemos que este procedimiento devolverá el resultado de una consulta del tipo SELECT.

Luego hay que determinar si el procedimiento recibirá parámetros. En este caso, sería muy útil que nuestro procedimiento reciba un número que represente el código de sucursal.

Por último, hay que definir un nombre para nuestro procedimiento almacenado. Para este ejemplo le pondremos *sp\_ClientesPorSucursal*.

Creación del Procedimiento:

```

CREATE PROCEDURE sp_ClientesPorSucursal(
    @idSucursal INT
)
AS

```

```
BEGIN
    SELECT * FROM CLIENTES WHERE idsucursal = @idSucursal
END
```

Ejecución del procedimiento:

```
EXEC sp_ClientesPorSucursal 1
```

Veamos paso a paso el código para cada una de las tareas que realizamos en el primer ejemplo. Primero creamos nuestro objeto de base de datos mediante la sentencia CREATE. Como nuestro objeto es un procedimiento almacenado lo indicamos mediante la palabra reservada PROCEDURE e indicamos que el nombre del procedimiento es sp\_ClientesPorSucursal. Luego definimos los parámetros que recibe el Stored Procedure. En este caso es sólo uno y se indica anteponiendo un arroba (@) al nombre de la variable donde lo queremos recibir acompañado de su tipo de dato.

Nótese que las variables en SQL Server se utilizan anteponiendo un arroba y el listado de parámetros se indican entre paréntesis y con sus tipos de datos a la derecha del nombre de la variable. En caso de ser muchos parámetros se los separa por comas.

Luego se indica con la palabra reservada AS la/s consulta/s que conformarán al procedimiento.

Se recomienda utilizar el bloque BEGIN ... END para indicar el inicio y fin del mismo.

En cuanto a la ejecución, se realiza mediante la palabra reservada EXEC seguido del nombre del procedimiento almacenado y su posterior pasaje de parámetros. Si fuesen más de uno se indican separados por coma. En este caso no se los indica entre paréntesis.

Ahora supongamos que queremos realizar un procedimiento almacenado que inserte un cliente en la base de datos.

Debemos tener en cuenta la cantidad y el tipo de parámetros necesarios para poder realizar el INSERT correctamente y definir un nombre para el procedimiento, en este caso se podría llamar *sp\_IngresarCliente*.

Creación del procedimiento:

```
CREATE PROCEDURE sp_IngresarCliente(
    @Apellido VARCHAR(50),
    @Nombre VARCHAR(50),
    @sexo CHAR,
    @idSucursal INT
)
AS
BEGIN
    INSERT INTO clientes (apellido, nombre, sexo, idSucursal, estado)
    VALUES(@Apellido, @Nombre, @sexo, @idSucursal, 1)
END
```

Ejecución del procedimiento:

```
EXEC sp_IngresarCliente 'López', 'Germán', 'M', 1
EXEC sp_IngresarCliente 'Iraola', 'Silvina', 'F', 1
```

El procedimiento almacenado anterior, permite realizar la inserción de un registro de la tabla clientes. Nótese el pasaje de múltiples parámetros al hacer la llamada al Stored Procedure.

A continuación, realizaremos un procedimiento almacenado para guardar datos en la tabla de movimientos. La misma registrará los débitos y créditos de una cuenta y actualizará el saldo de la misma. Habrá que tener en cuenta que si la cuenta es del tipo 'Caja de Ahorro' no se podrá realizar un débito mayor al saldo. Es decir, este no podrá quedar negativo. En cambio, si la cuenta es del tipo 'Cuenta corriente' se podrá dejar el saldo negativo pero no podrá ser mayor al límite de descubierto que permite el banco.

Esto quiere decir que deberemos aplicar programación a nuestro procedimiento almacenado. Particularmente necesitaremos aplicar una estructura condicional para ejecutar nuestras consultas. Y una estructura Try-catch para el manejo de errores. Y la declaración de variables auxiliares para simplificar las consultas.

Creación del procedimiento:

```
CREATE PROCEDURE spRegistrarMovimiento(
    @nroCuenta VARCHAR(20),
    @tipoMovimiento CHAR,
    @descripcion VARCHAR(50),
    @importe DECIMAL(10, 2)
)
AS
BEGIN
    --Comenzamos con el manejo de errores
    BEGIN TRY

        --Verificamos si el tipo de movimiento es Extracción
        IF @tipoMovimiento = 'E' OR @tipoMovimiento = 'e'
        BEGIN
            --Declaramos las variables
            DECLARE @saldo DECIMAL(10, 2)
            DECLARE @tipoCuenta INT
            DECLARE @descubierto DECIMAL(10, 2)

            --Le asignamos valores que provienen de una consulta SQL
            SELECT @saldo = C.saldo, @tipoCuenta = C.idTipoCuenta, @descubierto =
            C.limite_descubierto FROM cuentas C WHERE C.nroCuenta = @nroCuenta

            --Verificamos si se puede hacer la Extracción
            IF @saldo - @importe < 0 AND @tipoCuenta = 1
            BEGIN
                RAISERROR ('NO SE PUEDE REALIZAR EL MOVIMIENTO, SALDO ES INSUFICIENTE', 16, 1)
            END
            IF @saldo - @importe < (0 - @descubierto)
            BEGIN
                RAISERROR ('NO SE PUEDE REALIZAR EL MOVIMIENTO, SALDO ES INSUFICIENTE', 16, 1)
            END
            END
        END

        --Registramos el movimiento en la tabla de Movimientos
        INSERT INTO movimientos (fecha, nrocuenta, descripcion, tipomovimiento,
```

```
importe) VALUES(GETDATE(), @nroCuenta, @descripcion, @tipoMovimiento, @importe)
```

```
--Si es Extracción el importe debe restarse
```

```
IF @tipoMovimiento = 'E' OR @tipoMovimiento = 'e'
```

```
BEGIN
```

```
SET @importe = @importe * -1
```

```
END
```

```
--Actualizamos el saldo de la cuenta
```

```
UPDATE cuentas SET saldo = saldo + @importe WHERE nroCuenta = @nroCuenta
```

```
END TRY
```

```
BEGIN CATCH
```

```
    PRINT ERROR_MESSAGE()
```

```
END CATCH
```

```
--Finaliza el manejo de errores
```

```
END
```


Antes de comenzar con la ejecución de los procedimientos almacenados, veamos el contenido de la tabla Cuentas. La tabla Movimientos no tiene registros.

```
SELECT * FROM Cuentas
```

	nrocuenta	idcliente	idtipocuenta	saldo	limite_descubierto	fecha_alta	fecha_baja	estado
1	1	1	1	400.00	0.00	2000-01-01 00:00:00.000	NULL	1
2	2	2	2	300.00	1000.00	2009-01-01 00:00:00.000	NULL	1

Veamos qué pasa si intentamos debitar \$500 de la cuenta nro 1 que posee \$400 y es del tipo Caja de Ahorro.


```
EXEC spRegistrarMovimiento '1', 'E', 'EXTRACCION EN CAJERO AUTOMATICO', 550
```

 Mensajes

NO SE PUEDE REALIZAR EL MOVIMIENTO, SALDO ES INSUFICIENTE

Veamos qué pasa si intentamos debitar \$1400 de la cuenta nro 2 que posee \$300 de saldo y es del tipo Cuenta corriente con \$1000 de límite para girar en descubierto.

```
EXEC spRegistrarMovimiento '2', 'E', 'EXTRACCION EN SUCURSAL', 1400
```

 Mensajes

NO SE PUEDE REALIZAR EL MOVIMIENTO, SALDO ES INSUFICIENTE

```
EXEC spRegistrarMovimiento '1', 'E', 'EXTRACCION EN CAJERO AUTOMATICO', 120
```

```
EXEC spRegistrarMovimiento '2', 'D', 'DEPOSITO', 33
```

Las dos últimas consultas funcionarán correctamente y si realizamos una selección de datos de

las tablas Cuentas y Movimientos veremos los siguientes datos:

```
SELECT * FROM Cuentas
```

```
SELECT * FROM Movimientos
```

	nrocuenta	idcliente	idtipocuenta	saldo	limite_descubierto	fecha_alta	fecha_baja	estado
1	1	1	1	280.00	0.00	2000-01-01 00:00:00.000	NULL	1
2	2	2	2	333.00	1000.00	2009-01-01 00:00:00.000	NULL	1

	idmovimiento	fecha	nrocuenta	tipomovimiento	descripcion	importe
1	1	2012-05-23 20:07:22.287	1	E	EXTRACCION EN CAJERO AUTOMATICO	120.00
2	2	2012-05-23 20:07:22.287	2	D	DEPOSITO	33.00

Veamos algunas cuestiones acerca del procedimiento almacenado anterior, recibe los parámetros para realizar el movimiento (extracción o depósito) y debe de evaluar si se puede realizar o no. En caso de ser un depósito no habrá problemas y lo único que debe hacer el Stored Procedure es insertar los valores en la tabla de movimientos y actualizar el saldo de la cuenta. En caso de ser una extracción hay que determinar si se tiene el saldo suficiente, dicho proceso variará en caso de ser una cuenta corriente (permite girar en descubierto) o una caja de ahorro. En cualquiera de los casos si el débito es mayor al permitido deberá impedir el proceso y finalizar el procedimiento.

Ahí es donde se incorporan las nuevas estructuras, pero antes de comenzar a hablar de ellas, realizaremos una mención acerca de las variables en SQL Server.

## Variables

Las variables que son declaradas en un procedimiento almacenado son locales al mismo, esto quiere decir que sólo el Stored Procedure podrá 'ver' las variables declaradas en él.

Son declaradas mediante la palabra reservada DECLARE y su sintaxis es:

```
DECLARE @nombre_variable TIPO_DE_DATO
```

Para asignarle valores se podrá hacerlo mediante el uso de la instrucción SET o bien mediante un SELECT.

En nuestro procedimiento almacenado podemos ver como declaramos las variables @saldo, @tipoCuenta y @descubierto. Las mismas necesitan valores pero no serán establecidos manualmente por una instrucción sino que dichos datos provendrán de una consulta SELECT. Esto quiere decir, que en este caso, queremos obtener el saldo, tipo de cuenta y limite de descubierto de una cuenta en particular. Cuenta cuyo valor lo tenemos en otra variable, denominada @nroCuenta y que se envía como parámetro al procedimiento.

Es por eso que se ejecuta la siguiente instrucción: `SELECT @saldo = C.saldo, @tipoCuenta = C.idTipoCuenta, @descubierto = C.limite_descubierto FROM cuentas C WHERE C.nroCuenta = @nroCuenta.`

Por otro lado, podemos ver como también se puede asignar valores a variables mediante la intrucción SET como figura en nuestro procedimiento `SET @importe = @importe * -1.`

En el procedimiento almacenado spRegistrarMovimiento figuran dos grandes estructuras. La decisión simple (IF) y el manejo de errores (TRY...CATCH).



## Decisión simple

La decisión simple en SQL Server se utiliza de la misma manera que en los demás lenguajes de programación. La misma puede o no tener un caso falso (else). La sintaxis de la misma es la siguiente:

```
IF condicion
BEGIN
    -- Instrucciones por el lado verdadero
END
ELSE
BEGIN
    -- Instrucciones por el lado falso
END
```

En nuestro procedimiento almacenado, utilizamos la decisión simple para determinar, por ejemplo, el tipo de movimiento que se desea registrar. Es por eso que se utiliza el siguiente fragmento de código:

```
IF @tipoMovimiento = 'E' OR @tipoMovimiento = 'e'
BEGIN
SET @importe = @importe * -1
END
```

En este caso, si el tipo de movimiento es una extracción. El importe es multiplicado por -1 para que al sumarlo al saldo este se debite en lugar de acreditarse. Transformando el movimiento en un débito de dinero.

## Try - Catch

Por último, una de las estructuras utilizadas en el procedimiento almacenado es la del TRY ... CATCH. Ésta tiene el mismo funcionamiento que en .NET donde se incorpora el código de nuestro proceso dentro del bloque TRY y se establecen las acciones que deben realizarse en el bloque CATCH en el caso que ocurra algún error.

La estructura general del TRY-CATCH es la siguiente:

```
BEGIN TRY
    --GRUPO DE SENTENCIAS SQL
END TRY
BEGIN CATCH
    --GRUPO DE SENTENCIAS SQL EN CASO DE ERROR
END CATCH
```

## La función RAISERROR

La función RAISERROR está pensada para el manejo de errores, por lo que se le puede asignar un mensaje de error, un nivel de severidad y un número de error.

Los mismos pueden ser utilizados por los clientes desarrollados en lenguajes de programación que manejen captura de errores tales como C#, VB.NET, C++, etc.

La sintaxis es RAISERROR(*cadena\_mensaje, severidad, estado*)

Donde:

- *cadena\_mensaje* es el mensaje de error que se quiere enviar, de hasta 399 caracteres.
- *severidad* es el nivel de severidad definido por el usuario para el error y puede tomar valores entre 0 y 18. Las severidades entre 19 y 25 sólo pueden ser utilizadas por el administrador de la base de datos (sysadmin).
- *estado* es un entero entre 1 y 127. Si el mismo tipo de error definido por el usuario se utiliza en varios lugares, se puede utilizar distintos estados para cada una de las locaciones para identificar mejor dónde se ubica el error.