



Procedimientos almacenados y funciones

En muchas ocasiones una aplicación cliente requiere que la base de datos le entregue el resultado de operaciones ejecutadas sobre los datos. Para ello podemos definir en la base de datos código del lado del servidor a través de procedimientos almacenados y funciones de servidor.

Esta página se ha dejado en blanco intencionalmente.

Capítulo 14

Procedimientos almacenados y funciones

Contenido

- *Procedimientos almacenados*
 - ✓ *Definición de procedimiento almacenado*
 - ✓ *Proceso de creación y ejecución de procedimientos almacenados*
 - ✓ *Ventajas de los procedimientos almacenados*
 - ✓ *Creación de un procedimiento almacenado*
 - ✓ *Procedimiento que no recibe ni devuelve parámetros*
 - ✓ **Ejercicio 127:** *Procedimiento sin parámetros – La instrucción CREATE PROCEDURE*
 - ✓ *Procedimiento que recibe parámetros*
 - ✓ **Ejercicio 128:** *Procedimiento que recibe un parámetro*
 - ✓ **Ejercicio 129:** *Procedimiento que recibe más de un parámetro*
 - ✓ *Procedimiento que recibe y entrega parámetros*
 - ✓ **Ejercicio 130:** *Procedimiento que recibe y entrega parámetros*
 - ✓ **Ejercicio 131:** *Ranking de los N productos más solicitados – Uso de TOP*
 - ✓ *Obtención de la definición de un procedimiento almacenado*
 - ✓ *Procedimientos almacenados anidados*
 - ✓ *Pautas para crear procedimientos*
 - ✓ *Modificación de un procedimiento almacenado – La instrucción ALTER PROCEDURE*
 - ✓ *Eliminación de un procedimiento almacenado – La instrucción DROP PROCEDURE*

- ✓ *Pautas al utilizar los parámetros de entrada*
 - *Paso de los parámetros por referencia*
 - *Paso de los parámetros por posición*
- ✓ *Pautas al utilizar los parámetros de salida*
- *Manejo de los mensajes de error*
 - ✓ *La instrucción RAISERROR*
 - ✓ **Ejercicio 132:** *Mensaje de error personalizado*
 - ✓ **Ejercicio 133:** *Mensaje de error del sistema*
 - ✓ **Ejercicio 134:** *Mensaje de error con parámetros sustituibles*
- *Funciones definidas por el usuario*
 - ✓ *Definición de función definida por el usuario*
 - ✓ *Tipos de funciones definidas por el usuario*
 - ✓ *Creación de función – La instrucción CREATE FUNCTION*
 - ✓ **Ejercicio 135:** *Creación de función*
 - ✓ *Función escalar*
 - ✓ **Ejercicio 136:** *Función escalar*
 - ✓ *Función tabla evaluada multisentencia*
 - ✓ **Ejercicio 137:** *Función tabla evaluada multisentencia*
 - ✓ *Función tabla evaluada en línea*
 - ✓ **Ejercicio 138:** *Función tabla evaluada en línea*
 - ✓ *Creación de función con enlace de esquema*
 - ✓ *El operador APPLY*
 - *Formas de APPLY*
 - ✓ **Ejercicio 139:** *Autojoin*
 - ✓ **Ejercicio 140:** *Uso del operador APPLY*

Procedimientos almacenados y funciones

Procedimientos almacenados

En muchas ocasiones una aplicación cliente requiere que la base de datos le entregue el resultado de operaciones ejecutadas sobre los datos. Para ello podemos definir en la base de datos código del lado del servidor a través de procedimientos almacenados y funciones de servidor.

Definición de procedimiento almacenado

Los procedimientos almacenados son instrucciones precompiladas de Transact-SQL almacenadas en una base de datos SQL Server. Debido a que los procedimientos almacenados son precompilados, por lo general ofrecen mejor desempeño que cualquier consulta.

Un procedimiento almacenado es un objeto que se almacena en la base de datos, y que ejecuta una serie de sentencias SQL cuando el procedimiento es invocado.

Los procedimientos almacenados en SQL Server son similares a los procedimientos en otros lenguajes de programación ya que ellos:

- Contienen declaraciones que realizan las operaciones en la base de datos, incluyendo la habilidad de llamar a otros procedimientos almacenados.
- Aceptan parámetros de entrada.
- Devuelven un valor de estado cuando para indicar el éxito o fracaso del procedimiento.
- Pueden retornar múltiples valores en forma de parámetros de salida.

Proceso de creación y ejecución de procedimientos almacenados

Creación

Cuando un procedimiento almacenado se crea, se analizan las declaraciones en él para verificar la sintaxis. SQL Server entonces almacena el nombre del procedimiento almacenado en la tabla de sistema sysobjects, y el texto del procedimiento almacenado en la tabla de sistema syscomments en la base de datos actual. Un mensaje de error es retornado si se encuentra un error en la sintaxis; en este caso, el procedimiento almacenado no se crea.

Ejecución (Primera vez o recompilación)

La primera vez que un procedimiento almacenado se ejecuta, o si el procedimiento almacenado debe ser recompilado, el procesador de consultas lee el procedimiento almacenado en un proceso llamado resolución.

Los procedimientos almacenados son recompilados automáticamente en las circunstancias siguientes:

- Siempre que las versiones del esquema son cambiadas, por ejemplo, cuando una tabla o índice es modificada.
- Cuando el entorno en que el procedimiento almacenado fue compilado es diferente al entorno en el que se está ejecutando.
- Cuando las estadísticas han cambiado para un índice o una tabla que es referenciada en el procedimiento almacenado.

Optimización

Cuando un procedimiento almacenado ha pasado con éxito las etapas de la fase de resolución, SQL Server analiza las sentencias Transact-SQL en el procedimiento almacenado, y crea un plan que contiene el método más rápido para acceder a los datos. Para la optimización se toma en cuenta lo siguiente:

- La cantidad de datos en las tablas.
- La presencia y naturaleza de los índices de la tabla y la distribución de datos en las columnas afectadas por el índice.

- Los operadores de comparación y, los valores de comparación que se usan en la cláusula condicional WHERE.
- La presencia de joins y las cláusulas UNION, GROUP BY, u ORDER BY.

Recopilación

La recopilación se refiere al proceso de analizar el procedimiento almacenado y crear un plan de ejecución del mismo. Una vez determinado el plan de ejecución, éste es colocado en el caché de procedimientos, y luego el procedimiento es ejecutado.

Ventajas de los procedimientos almacenados

- La lógica de la aplicación puede ser compartida con otras aplicaciones. Los procedimientos almacenados pueden encapsular la lógica de negocios en una sola ubicación, y estar disponible para muchas aplicaciones.
- Proporciona un mecanismo de seguridad. Se puede otorgar a un usuario permiso para ejecutar un procedimiento almacenado aun cuando no tenga permiso para acceder a las tablas referenciadas por el procedimiento almacenado.
- Mejora el rendimiento del sistema y reduce el tráfico en la red. El uso de lógica condicional determina qué sentencias se ejecutarán y cuáles no.

Creación de un procedimiento almacenado

La creación de un procedimiento almacenado es similar a la creación de una vista. Debe crear el procedimiento almacenado en la base de datos actual. Primero, escriba y pruebe las sentencias Transact-SQL que quiere incluir en el procedimiento almacenado. Entonces, si recibe los resultados que espera, cree el procedimiento almacenado.

Para crear los procedimientos almacenados utilice la sentencia CREATE PROCEDURE. Considere lo siguiente al crear los procedimientos almacenados:

- Los procedimientos almacenados pueden hacer referencia a las tablas, las vistas, y otros procedimientos almacenados, así como a las tablas temporales.
- Si un procedimiento almacenado crea una tabla temporal local, la tabla temporal sólo existe con mientras se está ejecutando el procedimiento almacenado, y desaparece cuando la ejecución del procedimiento almacenado se completa.

- Una sentencia CREATE PROCEDURE no puede combinarse con otras declaraciones de SQL en un solo batch.
- La definición del CREATE PROCEDURE puede incluir cualquier número y tipo de sentencias Transact-SQL, con la excepción de la declaraciones de creación de los siguientes objetos: CREATE DEFAULT, CREATE PROCEDURE, CREATE RULE, CREATE TRIGGER, y CREATE VIEW. Pueden crearse otros objetos de la base de datos dentro de un procedimiento almacenado.
- Para ejecutar la sentencia CREATE PROCEDURE, se debe ser un miembro del rol **sysadmin**, **db_owner**, o **db_ddladmin**, o se debe tener el permiso CREATE PROCEDURE.

Procedimiento que no recibe ni devuelve parámetros – La instrucción CREATE PROCEDURE

Sintaxis

```
CREATE PROCEDURE nombreProcedimiento
AS
sentenciasSQL
```

Ejercicio 127: Procedimiento sin parámetros

Crear un procedimiento almacenado que genere la Lista de Precios de **MarketPERU**.

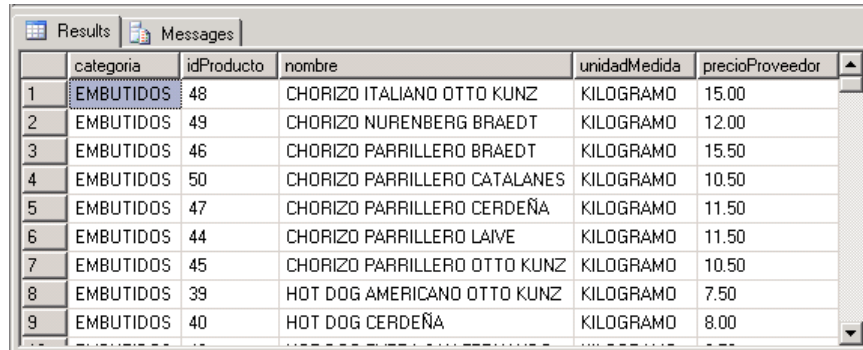
En el **Code Editor** digite y ejecute el siguiente código:

```
USE MarketPERU
go

CREATE PROCEDURE usp_ListaPrecios
AS
SELECT Categoria.categoria, Producto.idProducto,
       Producto.nombre, Producto.unidadMedida,
       Producto.precioProveedor
FROM Producto INNER JOIN Categoria
       ON Producto.idCategoria = Categoria.idCategoria
ORDER BY 1, 3
go
```


Para ejecutar el procedimiento

```
EXEC usp_ListaPrecios  
go
```



	categoria	idProducto	nombre	unidadMedida	precioProveedor
1	EMBUTIDOS	48	CHORIZO ITALIANO OTTO KUNZ	KILOGRAMO	15.00
2	EMBUTIDOS	49	CHORIZO NURENBERG BRAEDT	KILOGRAMO	12.00
3	EMBUTIDOS	46	CHORIZO PARRILLERO BRAEDT	KILOGRAMO	15.50
4	EMBUTIDOS	50	CHORIZO PARRILLERO CATALANES	KILOGRAMO	10.50
5	EMBUTIDOS	47	CHORIZO PARRILLERO CERDEÑA	KILOGRAMO	11.50
6	EMBUTIDOS	44	CHORIZO PARRILLERO LAIVE	KILOGRAMO	11.50
7	EMBUTIDOS	45	CHORIZO PARRILLERO OTTO KUNZ	KILOGRAMO	10.50
8	EMBUTIDOS	39	HOT DOG AMERICANO OTTO KUNZ	KILOGRAMO	7.50
9	EMBUTIDOS	40	HOT DOG CERDEÑA	KILOGRAMO	8.00

Procedimiento que recibe parámetros

Sintaxis

```
CREATE PROCEDURE nombreProcedimiento  
    @parámetro1 tipo_dato [ = valor ] ,  
    @parámetro2 tipo_dato [ = valor ] , ...  
AS  
sentenciasSQL
```

Ejercicio 128: Procedimiento que recibe un parámetro

Crear un procedimiento que entregue la lista de productos del proveedor X, donde X es el código del proveedor.

```
USE MarketPERU
go

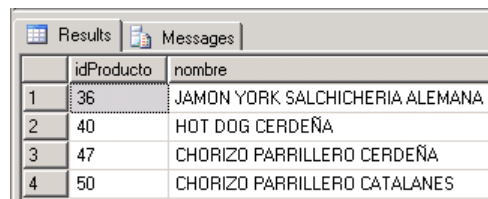
CREATE PROCEDURE usp_ListaProveedor
    @proveedor int
AS
SELECT idProducto, nombre
FROM Producto
WHERE idProveedor = @proveedor
go
```

Para ejecutar el procedimiento

```
DECLARE @codProveedor int
SET @codProveedor = 3
EXEC usp_ListaProveedor @codProveedor
go
```

También,

```
EXEC usp_ListaProveedor 3
go
```



	idProducto	nombre
1	36	JAMON YORK SALCHICHERIA ALEMANA
2	40	HOT DOG CERDEÑA
3	47	CHORIZO PARRILLERO CERDEÑA
4	50	CHORIZO PARRILLERO CATALANES

Ejercicio 129: Procedimiento que recibe más de un parámetro

Crear un procedimiento que entregue una lista de los productos del proveedor X para la categoría Y.

```
USE MarketPERU
go

CREATE PROCEDURE usp_ListaProveedorCategoria
    @proveedor int, @categoria int
AS
SELECT idProducto, nombre
FROM Producto
WHERE idProveedor = @proveedor AND idCategoria = @categoria
go
```

Para ejecutar el procedimiento

```
DECLARE @codPro int, @codCat int
SET @codPro = 15
SET @codCat = 1
EXEC usp_ListaProveedorCategoria @codPro, @codCat
go
```

También,

```
EXEC usp_ListaProveedorCategoria 15, 1
go
```

También,

```
EXEC
usp_ListaProveedorCategoria
    @categoria=1,
    @proveedor=15
go
```

Results			Messages
	idProducto	nombre	
1	2	CARAMELOS SURTIDO DE FRUTAS	
2	5	CHUPETES LOLY AMBROSOLI	
3	6	FRUNA SURTIDA DONOFRIO	
4	7	CHOCOLATE DOÑA PEPA FIELD	
5	8	CHOCOLATE CUA CUA FIELD	
6	9	MELLOW'S FAMILIAR FIELD	
7	10	WAFER CHOCOLATE FIELD	

Procedimiento que recibe y entrega parámetros

Sintaxis

```
CREATE PROCEDURE nombreProcedimiento
    @parámetro1 tipo_dato [ = valor ] ,
    @parametro2 tipo_dato [ = valor ] ,
    @parámetro3 tipo_dato [ = valor ] OUTPUT , ...
AS
sentenciasSQL
```

Ejercicio 130: Procedimiento que recibe y entrega parámetros

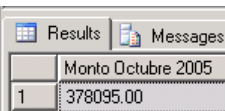
Crear un procedimiento que devuelva el monto mensual total de las guías de remisión correspondientes al mes Z.

```
USE MarketPERU
go

CREATE PROCEDURE usp_MontoGuiasMes
    @año int, @mes int, @monto money OUTPUT
AS
SET @monto = (SELECT
    sum(Guia_detalle.cantidad * Guia_detalle.precioVenta)
    FROM Guia_detalle INNER JOIN Guia
    ON Guia_detalle.idGuia = Guia.idGuia
    WHERE year(Guia.fechaSalida) = @año
    AND month(Guia.fechaSalida) = @mes)
go
```

Para ejecutar el procedimiento

```
DECLARE @miMonto money
EXEC usp_MontoGuiasMes 2005, 10, @miMonto OUTPUT
SELECT 'Monto Octubre 2005' = @miMonto
go
```



Results		Messages	
		Monto Octubre 2005	
1		378095.00	

Ejercicio 131: Ranking de los N productos más solicitados – Uso de TOP

Sintaxis

```
SELECT TOP (expresión) [ PERCENT ]  
    [ WITH TIES ] listaColumnas  
FROM ...
```

- **expresión** indica la cantidad de filas que la consulta debe entregar.
- PERCENT establece que **expresión** indica el porcentaje de filas que la consulta debe entregar respecto al listado resultante.
- WITH TIES indica que en el caso de una consulta ORDER BY, si existen filas adicionales con el mismo valor de ordenamiento que la última fila, entonces las filas adicionales se deben incluir en el resultado.

Crear un procedimiento almacenado que entregue la lista de los N productos más solicitados, donde N es un parámetro que se le entrega al procedimiento.

```
USE MarketPERU  
go  
  
CREATE PROCEDURE usp_RankingProductos  
    @posiciones int  
AS  
SELECT TOP (@posiciones) WITH TIES p.idProducto, p.nombre,  
    sum(gd.cantidad * gd.precioVenta) AS Monto  
FROM Producto p INNER JOIN Guia_detalle gd  
    ON p.idProducto = gd.idProducto  
GROUP BY p.idProducto, p.nombre  
ORDER BY 3 DESC  
go  
  
DECLARE @primeros int  
SET @primeros = 5  
EXEC usp_RankingProductos @primeros  
go
```

Results		Messages	
	idProducto	nombre	Monto
1	130	DETERGENTE LIMON INVICTO	27000.00
2	124	DETERGENTE C/BLANQUEADOR ARIEL	22500.00
3	125	DETERGENTE LIMON ARIEL	22500.00
4	129	DETERGENTE LIMON ARIEL	21150.00
5	64	JABON DOVE BLANCO	21000.00

Obtención de la definición de un procedimiento almacenado

Como con otros objetos de la base de datos, puede usar los procedimientos almacenados del sistema para obtener información sobre la definición de los procedimientos almacenados de su base de datos.

Vista de sistema	Almacena	Procedimiento
sys.sysobjects	Nombre del procedimiento almacenado	sp_help [<i>nombre_procedimiento</i>] sp_stored_procedures
sys.sysdepends	Nombre de los objetos dependientes del procedimiento almacenado	sp_depends <i>nombre_procedimiento</i>
sys.syscomments	Sentencia que definió el procedimiento almacenado	sp_helptext <i>nombre_procedimiento</i>

Procedimientos almacenados anidados

Los procedimientos almacenados se pueden anidar de manera que un procedimiento almacenado llame a otro. Al anidar tenga en cuenta lo siguiente:

- Los procedimientos almacenados se pueden anidarse hasta 32 niveles.
- El nivel de anidamiento actual se almacena en la variable global **@@nestlevel**.
- Si un procedimiento almacenado llama a un segundo procedimiento almacenado, el segundo procedimiento almacenado puede acceder a todos los objetos creados por el primer procedimiento almacenado, incluyendo las tablas temporales.

Pautas para crear procedimientos almacenados

Considere lo siguiente antes de crear los procedimientos almacenados:

- Para evitar situaciones en las que el dueño de un procedimiento almacenado y el dueño de las tablas referenciadas son distintos, se recomienda que los usuarios estén mapeados como dbo para todos los objetos en una base de datos. Como un usuario puede ser un miembro de múltiples roles, siempre especifique al usuario dbo como el dueño al crear el objeto.
 - ✓ También debe tener los permisos apropiados en todas las tablas o vistas que son referenciadas dentro del procedimiento almacenado.
 - ✓ Evite situaciones en las que el dueño de un procedimiento almacenado y el dueño de las tablas referenciadas son distintos.
- Diseñe el procedimiento almacenado de manera tal que ejecute una sola tarea.
- Cree, pruebe, y ponga a punto su procedimiento almacenado en el servidor; luego,, pruébelo en el cliente.
- Para distinguir fácilmente a un procedimiento almacenado del sistema de un procedimiento creado por el usuario, evite el uso del prefijo **sp_** al nombrar el procedimiento.
- Si no desea que los usuarios puedan ver el texto de sus procedimientos almacenados, debe crearlos con la opción de WITH ENCRYPTION. Si no usa WITH ENCRYPTION, los usuarios pueden usar el procedimiento almacenado del sistema **sp_helptext** para ver el texto de procedimientos almacenados en la tabla de sistema **syscomments** o SQL Server Enterprise Manager.

Modificación de un procedimiento almacenado – La instrucción ALTER PROCEDURE

Para modificar un procedimiento almacenado conservando su asignación de permisos, ejecute la sentencia ALTER PROCEDURE. La definición previa del procedimiento será reemplazada por la definición establecida en ALTER PROCEDURE.

Sintaxis

```
ALTER PROCEDURE nombreProcedimiento
    @parámetro1 tipo_dato [ = valor ] ,
    @parametro2 tipo_dato [ = valor ] ,
    @parámetro3 tipo_dato [ = valor ] OUTPUT , ...
AS
sentenciasSQL
```

Eliminación de un procedimiento almacenado – La instrucción DROP PROCEDURE

Use la sentencia DROP PROCEDURE para eliminar los procedimientos almacenados definidos por el usuario de la base de datos actual.

Antes de eliminar un procedimiento almacenado, ejecute el procedimiento almacenado **sp_depends** para determinar los objetos que dependen del procedimiento almacenado.

Sintaxis

```
DROP PROCEDURE nombreProcedimiento
```


Pautas al utilizar los parámetros de entrada

Los parámetros de entrada permiten pasar información a un procedimiento almacenado. Para definir un procedimiento almacenado que acepta parámetros de entrada, declare una o más variables como parámetros en la declaración de la sentencia `CREATE PROCEDURE`.

Tenga en cuenta lo siguiente al especificar parámetros:

- Todos los valores de los parámetros entrantes deben verificarse al principio de un procedimiento almacenado.
- Debe mantener los valores predeterminados apropiados para un parámetro. Si un valor por defecto se define, un usuario puede ejecutar el procedimiento almacenado sin especificar un valor para ese parámetro.
- El número máximo de parámetros en un procedimiento almacenado es de 255.
- El número máximo de variables locales y globales en un procedimiento almacenado está limitado por la memoria disponible.
- Los parámetros son locales en un procedimiento almacenado. Los mismos nombres de parámetro pueden usarse en otros procedimientos almacenados.

La información de la definición de parámetros se guarda en la tabla de sistema `syscolumns`.

Paso de los parámetros por referencia

Cuando en la sentencia `EXECUTE` hacemos referencia a un parámetro con el formato `@parámetro = valor`, éste es pasado por referencia.

Cuando pasa un valor por referencia, los valores de los parámetros pueden especificarse en cualquier orden, y puede omitir parámetros que permiten valores nulos o que tienen un valor por defecto.

Sintáxis

```
[ EXECUTE ] nombreProcedimiento  
[ @parámetro = valor ], [ @parámetro = valor ], ...
```

Paso de los parámetros por posición

Cuando pasamos solamente valores (sin una referencia a los parámetros a los que ellos están asociados) se dice que se están pasando por posición. Cuando especifica solo el valor, los valores de los parámetros deben listarse en el orden en que ellos se definieron en la sentencia CREATE PROCEDURE.

Cuando pasa los parámetros por posición, puede omitir parámetros donde los valores por defecto existen, pero no puede interrumpir la sucesión. Por ejemplo, si un procedimiento almacenado tiene cinco parámetros, puede omitir el cuarto y quinto parámetros, pero no puede omitir el cuarto parámetro y luego especificar el quinto.

Sintaxis

```
[ EXECUTE ] nombreProcedimiento [ valor [ , valor ] , ... ]
```

Pautas al utilizar los parámetros de salida

Los procedimientos almacenados pueden devolver información a través de los parámetros de salida (variables designadas con la palabra clave OUTPUT). Usando los parámetros de salida, los valores retornados por el procedimiento almacenado pueden retenerse, incluso después que el procedimiento almacenado ha completado su ejecución.

Para usar un parámetro de salida, la palabra clave OUTPUT debe especificarse tanto en la declaración de CREATE PROCEDURE como en la de EXECUTE. Si la palabra clave OUTPUT se omite cuando se ordena la ejecución del procedimiento almacenado, éste se ejecuta, pero produce una condición del error. Los parámetros de salida tienen las siguientes características:

- Al llamar al procedimiento almacenado, la declaración debe contener el nombre de la variable que recibirá el valor del retorno. No es posible pasar constantes.
- Como consecuencia de lo anterior, puede usar la variable en las declaraciones de SQL adicionales en el lote o al llamar a otro procedimiento almacenado.
- Si especifica los valores de los parámetros en un orden diferente al de la definición del procedimiento almacenado, los parámetros deben pasarse por referencia.
- El parámetro puede ser de cualquier tipo dato excepto texto o imagen.

Manejo de los mensajes de error

La instrucción RAISERROR

RAISERROR genera un mensaje de error que se le envía al usuario. Puede enviar un mensaje de error del sistema leído desde la vista del catálogo **sys.sysmessages** ó puede enviar un mensaje creado dinámicamente en tiempo de ejecución.

Sintaxis

```
RAISERROR( msgID | msgCad | @variableLocal ,  
          severidad , estado )  
          [ WITH LOG | WITH NOWAIT | WITH SETERROR ]
```

- El primer argumento indica que se puede especificar un número de mensaje (**msgID**), una cadena con el mensaje personalizado (**msgCad**), ó una variable conteniendo la cadena con el mensaje. **msgID** es el número del mensaje leído desde **sys.sysmessages**; **msgCad** es la cadena con el mensaje personalizado.
- **severidad** indica el nivel de gravedad del error. Su valor va de 0 a 25. Los niveles del 19 al 25 solo pueden ser especificados por los miembros del rol sysadmin y requieren el uso de WITH LOG. Los mensajes con niveles mayor a 19 son considerados muy graves y provocan que la conexión con la aplicación cliente se interrumpa.
- **estado** es un entero de 1 a 127 y su significado es determinado por el programador.
- WITH LOG permite registrar el error en el log de aplicación del sistema.
- WITH NOWAIT envía inmediatamente el mensaje al cliente.
- SETERROR configura el valor de la variable **@@error** al número del error generado independientemente del nivel de severidad. Por defecto, los mensajes de niveles de 0 a 10 no cambian el valor de **@@error**.

Ejercicio 132: Mensaje de error personalizado

Crear un procedimiento que entregue el monto total despachado para la fecha X.

```
USE MarketPERU
go

-- Monto total despachado en la fecha X
CREATE PROCEDURE usp_MontoGuiasPorFecha
    @fecha char(10), @monto money OUTPUT
AS
IF NOT EXISTS (SELECT fechaSalida
    FROM Guia
    WHERE convert(char(10), fechaSalida, 103) = @fecha)
BEGIN
    RAISERROR('No hay guías para la fecha indicada.',
        16, 1)
    RETURN
END

SET @monto = (SELECT sum(gd.cantidad * gd.precioVenta)
    FROM Guia_detalle gd INNER JOIN Guia g
        ON gd.idGuia = g.idGuia
    WHERE convert(char(10), g.fechaSalida, 103) =
        @fecha)
go
```

Para probar el procedimiento ejecútelo con una fecha para la cual no se han registrado guías de remisión.

```
SET DATEFORMAT dmy
DECLARE @miMonto money
EXEC usp_MontoGuiasPorFecha '15/10/2004', @miMonto OUTPUT
SELECT 'Monto mes' = @miMonto
go
```

Se recibe el siguiente mensaje:

```
Msg 50000, Level 16, State 1, Procedure  
usp_MontoGuiasPorFecha, Line 8  
No hay guías para la fecha indicada.
```

Notas:

Cualquier mensaje generado usando el argumento **msgCad** de RAISERROR tiene como número de error el valor 50000.

Los mensajes de nivel de severidad 10 son solo informativos e indican error en la data ingresada por el usuario.

Los mensajes de los niveles del 11 al 16 indican errores que pueden ser corregidos por el mismo usuario.

Ejercicio 133: Mensaje de error del sistema

Para invocar a un mensaje de error del sistema utilice RAISERROR especificando el número del mensaje de error a enviar al usuario.

```
RAISERROR(13188, 10, 1)  
SELECT @@error -- 0 (cero)  
go
```

Se recibe el mensaje:

```
The certificate's private key is password protected
```

Note que como el nivel de severidad es 10, el error no se registra en @@error. Si quiere que el número del error se guarde en @@error ejecute:

```
RAISERROR(13188, 10, 1) WITH SETERROR  
SELECT @@error -- 13188  
go
```

Ejercicio 134: Mensaje de error con parámetros sustituibles

Crear un procedimiento que genere un listado de las guías emitidas en la fecha F que contienen al producto P, donde P indica el tipo de producto, por ejemplo QUESO, GALLETA, ó LECHE.

```
USE MarketPERU
go

CREATE PROCEDURE usp_BuscaGuiaProducto
    @fecha char(10), @producto varchar(20)
AS
SELECT gd.idGuia, gd.idProducto, p.nombre
FROM Guia_detalle gd INNER JOIN Producto p
    ON gd.idProducto = p.idProducto
INNER JOIN Guia g
    ON gd.idGuia = g.idGuia
WHERE convert(char(10), g.fechaSalida, 103) = @fecha
    AND p.nombre LIKE '%' + @producto + '%'

IF @@rowcount = 0
    RAISERROR('No existe guías para el producto %s en la
fecha %s.', 16, 1, @producto, @fecha)
go
```

Ejecute el procedimiento para una fecha en la que no se emitieron guías ó para un producto no despachado en dicha fecha.

```
EXEC usp_BuscaGuiaProducto '26/11/2004', 'QUESO'
go
```

Se genera el siguiente mensaje:

```
Msg 50000, Level 16, State 1, Procedure
usp_BuscaGuiaProducto, Line 13
No existe guías para el producto QUESO en la fecha
26/11/2004.
```

La definición de RAISERROR en el procedimiento

```
RAISERROR('No existe guías para el producto %s en la fecha  
%s.', 16, 1, @producto, @fecha)
```

contiene 2 parámetros sustituibles: el primer **%s** que indica que en esa posición debe ir una cadena, y el segundo **%s** que también indica que en esa posición debe ir otra cadena.

El valor de la variable **@producto** sustituye al primer **%s** en el mensaje, y el valor de la variable **@fecha** sustituye al segundo **%s**, generándose el siguiente mensaje:

```
No existe guías para el producto QUESO en la fecha  
26/11/2004.
```

Los parámetros sustituibles que podemos utilizar son:

- **%s** para cadena
- **%d** ó **%i** para entero con signo
- **%u** para entero sin signo

Funciones definidas por el usuario

Tal como en los lenguajes de programación, las funciones definidas por el usuario de MS SQL Server son rutinas que aceptan parámetros, ejecutan un proceso, y retornan el resultado del proceso como un valor. Este valor de retorno puede ser un escalar ó un conjunto de resultados.

Definición de función definida por el usuario

Desde la versión SQL Server 2000 podemos diseñar nuestras propias funciones para complementar y extender las funciones estándar incorporadas con el sistema.

Al igual que una función estándar, una función definida por el usuario puede tener cero, uno ó más parámetros de entrada (argumentos), y debe retornar un valor escalar ó una tabla. Los parámetros de entrada pueden ser de cualquier tipo excepto **timestamp**, **cursor** ó **table**. Las funciones definidas por el usuario no soportan parámetros de salida.

Tipos de funciones definidas por el usuario

SQL Server soporta tres tipos de funciones definidas por el usuario:

Función escalar

Es similar a las funciones estándar del sistema.

Función tabla evaluada multisentencia

Retorna una tabla construida en base a una ó más sentencias Transact-SQL, y es similar a un procedimiento almacenado. A diferencia del procedimiento almacenado, una función de este tipo puede ser referenciada en la cláusula FROM de una sentencia SELECT, tal como si fuera una vista.

Función tabla evaluada en línea

Retorna una tabla que es resultado de la ejecución de una sola sentencia SELECT. Es similar a una vista, pero ofrece más flexibilidad en el uso de parámetros, y extiende las características de las vistas indexadas.

Creación de función – La instrucción CREATE FUNCTION

Una función definida por el usuario se crea de manera similar a una vista ó un procedimiento almacenado.

Sintáxis

```
CREATE FUNCTION nombreFunción(  
    @parámetro1 tipo_dato [ = valor ] ,  
    @parámetro2 tipo_dato [ = valor ] , ...  
    RETURNS tipo_dato_valor_retorno  
[ AS ]  
BEGIN  
    sentenciasSQL  
    RETURN valorRetorno  
END
```

Ejercicio 135: Creación de función

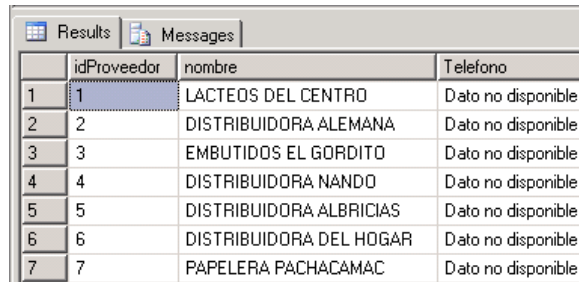
```
USE MarketPERU  
go  
  
CREATE FUNCTION udf_NoData(  
    @dato sql_variant )  
    RETURNS varchar(20)  
BEGIN  
    IF @dato IS NULL  
        SET @dato = 'Dato no disponible'  
    RETURN convert(varchar(20), @dato)  
END  
go
```

Crea una función de nombre **udf_NoData** que recibe como argumento un dato de cualquier tipo soportado por SQL Server, excepto **text**, **ntext**, **image**, **timestamp**, y **sql_variant**.

Si el argumento recibido es NULL, entonces la función retorna la cadena 'Dato no disponible'.

Para invocar a la función

```
SELECT idProveedor, nombre, dbo.udf_NoData(telefono) AS  
Telefono  
FROM Proveedor  
go
```



	idProveedor	nombre	Telefono
1	1	LACTEOS DEL CENTRO	Dato no disponible
2	2	DISTRIBUIDORA ALEMANA	Dato no disponible
3	3	EMBUTIDOS EL GORDITO	Dato no disponible
4	4	DISTRIBUIDORA NANDO	Dato no disponible
5	5	DISTRIBUIDORA ALBRICIAS	Dato no disponible
6	6	DISTRIBUIDORA DEL HOGAR	Dato no disponible
7	7	PAPELERA PACHACAMAC	Dato no disponible

Para establecer una referencia a una función definida por el usuario de tipo escalar, debe especificar el dueño de la función usando la sintáxis **dueño.objeto**.

Función escalar

- Es similar a las funciones estándar del sistema. Retorna un valor del tipo definido en RETURNS.
- El tipo del valor de retorno puede ser cualquiera excepto **text**, **ntext**, **image**, **cursor**, o **timestamp**.

Ejercicio 136: Función escalar

Crear una función que retorne el monto de la guía número N.

```
USE MarketPERU
go

CREATE FUNCTION udf_MontoGuia(
    @guia int)
    RETURNS money
BEGIN
    RETURN (SELECT sum(precioVenta * cantidad)
            FROM Guia_detalle
            WHERE idGuia = @guia)
END
go
```

El siguiente batch obtiene el monto de la guía número 87:

```
DECLARE @miGuia int
SET @miGuia = 87
SELECT dbo.udf_MontoGuia(@miGuia)
go
```

Función tabla evaluada multisentencia

- Es una combinación de vista y procedimiento almacenado, y puede utilizarse para retornar una tabla como alternativa al uso de vistas y procedimientos almacenados.
- Puede utilizarse en la cláusula FROM de una sentencia Transact-SQL.
- El tipo de valor de retorno debe ser **table**, y define el nombre y estructura de la tabla.
- El alcance de la variable de retorno es local.

Ejercicio 137: Función tabla evaluada multisentencia

```
USE MarketPERU
go

CREATE FUNCTION udf_ListaGuias(
    @salida varchar(15))
    RETURNS @lista table(
        Numero int NOT NULL PRIMARY KEY,
        Descripcion varchar(75) NOT NULL)
AS
BEGIN
    IF @salida = 'Local'
        INSERT @lista
        SELECT g.idGuia, l.direccion
        FROM Guia g INNER JOIN Local l
        ON g.idLocal = l.idLocal
    ELSE IF @salida = 'Fecha'
        INSERT @lista
        SELECT idGuia, convert(char(11), fechaSalida, 106)
        FROM Guia
    RETURN
END
go
```

La función **udf_ListaGuias** retorna una variable de tipo table con dos columnas: **Numero** y **Descripcion**.

El contenido de la tabla está determinado por el argumento **@salida**. Si **@salida** contiene la cadena **'Local'**, llena la tabla con los números de guía y sus direcciones de envío; si **@salida** contiene la cadena **'Fecha'**, llena la tabla con los números de guía y sus respectivas fechas de emisión.

Para ejecutar la función:

```
SELECT * FROM dbo.udf_ListaGuias('Fecha')
go
```

Tabla evaluada en línea

- Al igual que en una vista, el contenido de la función es una sentencia SELECT.
- El cuerpo de la función no está delimitado por BEGIN y END.
- El tipo del valor de retorno debe ser **table**.

Ejercicio 138: Función tabla evaluada en línea

```
USE MarketPERU
go

CREATE FUNCTION udf_ProductosPorCategoria(
    @categoria integer)
    RETURNS table
AS
RETURN(
    SELECT idProducto, nombre
    FROM Producto
    WHERE idCategoria = @categoria)
go
```

La función **udf_ProductosPorCategoria** entrega la lista de productos de la categoría que se le entrega como argumento.

```
SELECT * FROM udf_ProductosPorCategoria(2)
GO
```

Results		Messages
	idProducto	nombre
1	26	JAMONADA LAIVE
2	27	JAMONADA ESPECIAL LA SEGOVIANA
3	28	JAMONADA POLACA OTTO KUNZ
4	29	JAMONADA DE POLLO SAN FERNANDO
5	30	JAMONADA ESPECIAL OTTO KUNZ

Creación de función con enlace de esquema

Cuando una función se crea con la opción SCHEMABINDING, es enlazada a los objetos de base de datos a los que hace referencia. Estos objetos no podrán ser modificados (ALTER) ó eliminados (DROP).

Sintaxis

```
CREATE FUNCTION nombre_función(  
    @parámetro1 tipo_dato [ = valor ] ,  
    @parámetro2 tipo_dato [ = valor ] , ...  
    RETURNS tipo_dato_valor_retorno  
WITH SCHEMABINDING  
[ AS ]  
BEGIN  
    sentencias_sql  
    RETURN valor_retorno  
END
```

Una función puede tener enlace de esquema solo si se cumplen las siguientes condiciones:

- Las funciones definidas por el usuario y las vistas referenciadas por la función tienen también enlace de esquema.
- Los objetos a los que la función hace referencia no son referenciados con la forma **dueño.objeto**.
- La función, y los objetos a los que hace referencia pertenecen a la misma base de datos.
- El usuario que crea la función tiene el permiso REFERENCE sobre todos los objetos a los que la función hace referencia.

El operador APPLY

El operador APPLY permite invocar a una función tabla evaluada en línea por cada fila retornada por una consulta externa. La tabla evaluada en línea actúa como la consulta derecha de una subconsulta correlacionada, donde la consulta derecha es evaluada por cada fila de la consulta izquierda, y las filas obtenidas se combinan en el resultado final. El resultado producido por el operador APPLY es el conjunto de columnas en la consulta izquierda seguido del conjunto de columnas de la consulta derecha.

Formas de APPLY

- CROSS APPLY retorna solo las filas de la tabla izquierda para las que la función tabla evaluada en línea produce un conjunto de resultados.
- OUTER APPLY retorna las filas de la tabla izquierda para las que la función tabla evaluada en línea produce un conjunto de resultados, y además las filas de la tabla izquierda que no generan un conjunto de resultados.

Ejercicio 139: Autojoin

Ejecute el siguiente código en el **Code Editor**:

```
USE MarketPERU
go

-- Creación de la tabla EMPLEADO
CREATE TABLE Empleado(
    IdEmpleado int PRIMARY KEY,
    Nombre varchar(40) not null,
    IdSuperior int null )
go

INSERT INTO Empleado
    VALUES(1, 'Matsukawa Maeda, Sergio', NULL)
INSERT INTO Empleado
    VALUES(2, 'Coronel Castillo, Gustavo', 1)
INSERT INTO Empleado
    VALUES(3, 'Marcelo Villalobos, Ricardo', 1)
```

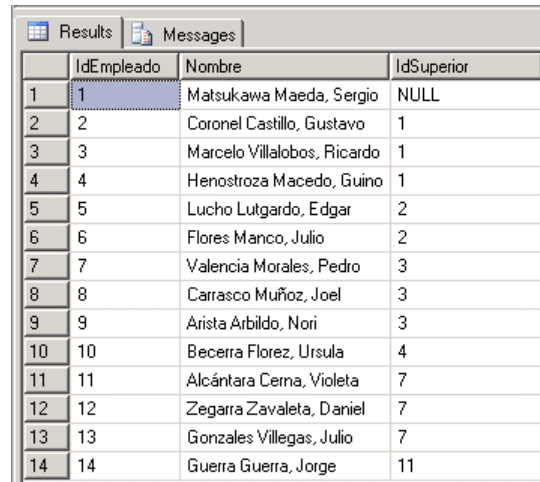
```
INSERT INTO Empleado
VALUES(4, 'Henostroza Macedo, Guino', 1)
INSERT INTO Empleado
VALUES(5, 'Lucho Lutgardo, Edgar', 2)
INSERT INTO Empleado
VALUES(6, 'Flores Manco, Julio', 2)
INSERT INTO Empleado
VALUES(7, 'Valencia Morales, Pedro', 3)
INSERT INTO Empleado
VALUES(8, 'Carrasco Muñoz, Joel', 3)
INSERT INTO Empleado
VALUES(9, 'Arista Arbildo, Nori', 3)
INSERT INTO Empleado
VALUES(10, 'Becerra Florez, Ursula', 4)
INSERT INTO Empleado
VALUES(11, 'Alcántara Cerna, Violeta', 7)
INSERT INTO Empleado
VALUES(12, 'Zegarra Zavaleta, Daniel', 7)
INSERT INTO Empleado
VALUES(13, 'Gonzales Villegas, Julio', 7)
INSERT INTO Empleado
VALUES(14, 'Guerra Guerra, Jorge', 11)
go

-- Creación de la tabla DEPARTAMENTO
CREATE TABLE Departamento(
    idDepartamento int PRIMARY KEY,
    nombre varchar(30) not null,
    idResponsable int null FOREIGN KEY
    REFERENCES Empleado )
go

INSERT INTO Departamento VALUES(1, 'Recursos Humanos', 2)
INSERT INTO Departamento VALUES(2, 'Marketing', 7)
INSERT INTO Departamento VALUES(3, 'Finanzas', 8)
INSERT INTO Departamento VALUES(4, 'Investigación', 9)
INSERT INTO Departamento VALUES(5, 'Capacitación', 4)
INSERT INTO Departamento VALUES(6, 'Logística', NULL)
go
```


Ahora, consulta la tabla **Empleado**.

```
SELECT * FROM Empleado  
go
```



The screenshot shows a SQL Server query results window with two tabs: 'Results' and 'Messages'. The 'Results' tab is active, displaying a table with four columns: 'IdEmpleado', 'Nombre', and 'IdSuperior'. The table contains 14 rows of data. The first row (IdEmpleado 1) has a NULL value for IdSuperior. The subsequent rows show a hierarchy where each employee's IdSuperior is the IdEmpleado of their superior.

	IdEmpleado	Nombre	IdSuperior
1	1	Matsukawa Maeda, Sergio	NULL
2	2	Coronel Castillo, Gustavo	1
3	3	Marcelo Villalobos, Ricardo	1
4	4	Henostroza Macedo, Guino	1
5	5	Lucho Lutgardo, Edgar	2
6	6	Flores Manco, Julio	2
7	7	Valencia Morales, Pedro	3
8	8	Carrasco Muñoz, Joel	3
9	9	Arista Arbildo, Nori	3
10	10	Becerra Florez, Ursula	4
11	11	Alcántara Cerna, Violeta	7
12	12	Zegarra Zavaleta, Daniel	7
13	13	Gonzales Villegas, Julio	7
14	14	Guerra Guerra, Jorge	11

Observe que para cada empleado se registra el código de su superior inmediato (**idSuperior**), el que a su vez también está registrado como empleado.

Se desea generar una lista que muestre para cada empleado, el nombre de su superior inmediato. Para ello, haciendo uso de los alias de tabla y de JOIN combinaremos la tabla **Empleado** consigo misma, ya que hay una relación entre las columnas **idSuperior** e **idEmpleado** de la tabla.

```
SELECT e1.idEmpleado, e1.nombre, e1.idSuperior, e2.nombre  
FROM Empleado e1 LEFT OUTER JOIN Empleado e2  
ON e1.idSuperior = e2.idEmpleado  
go
```

	idEmpleado	nombre	idSuperior	nombre
1	1	Matsukawa Maeda, Sergio	NULL	NULL
2	2	Coronel Castillo, Gustavo	1	Matsukawa Maeda, Sergio
3	3	Marcelo Villalobos, Ricardo	1	Matsukawa Maeda, Sergio
4	4	Henostroza Macedo, Guino	1	Matsukawa Maeda, Sergio
5	5	Lucho Lutgardo, Edgar	2	Coronel Castillo, Gustavo
6	6	Flores Manco, Julio	2	Coronel Castillo, Gustavo
7	7	Valencia Morales, Pedro	3	Marcelo Villalobos, Ricardo
8	8	Carrasco Muñoz, Joel	3	Marcelo Villalobos, Ricardo
9	9	Arista Arbildo, Nori	3	Marcelo Villalobos, Ricardo
10	10	Becerra Florez, Ursula	4	Henostroza Macedo, Guino
11	11	Alcántara Cerna, Violeta	7	Valencia Morales, Pedro
12	12	Zegarra Zavaleta, Daniel	7	Valencia Morales, Pedro
13	13	Gonzales Villegas, Julio	7	Valencia Morales, Pedro
14	14	Guerra Guerra, Jorge	11	Alcántara Cerna, Violeta

Ejercicio 140: Uso del operador APPLY

Se desea obtener una lista que muestre para cada departamento a su responsable, y a los subordinados de cada responsable.

```
-- Función tabla evaluada en línea que retorna todos
-- los subordinados del empleado que se le entrega como
-- argumento
CREATE FUNCTION udf_ListaSubordinados(
    @empleado AS INT )
    RETURNS @lista TABLE(
        idEmpleado int not null,
        nombre varchar(40)not null,
        idSuperior int null,
        nivel int not null )
AS
```

```

BEGIN
    WITH Arbol_Empleados(idEmpleado, nombre,
        idSuperior, nivel)
    AS
    (
        -- Miembro ancla
        SELECT idEmpleado, nombre, idSuperior, 0
        FROM Empleado
        WHERE idEmpleado = @empleado

        UNION all

        -- Miembro recursivo
        SELECT e.idEmpleado, e.nombre, e.idSuperior, ae.nivel+1
        FROM Empleado e INNER JOIN Arbol_Empleados ae
            ON e.idSuperior = ae.idEmpleado
    )
    INSERT INTO @lista
    SELECT * FROM Arbol_Empleados
    RETURN
END
go

```

Ejecute la siguiente instrucción para obtener la lista de subordinados del empleado cuyo **idEmpleado** es 1.

```

SELECT * FROM udf_ListaSubordinados(1)
go

```

Results Messages				
	idEmpleado	nombre	idSuperior	nivel
1	1	Matsukawa Maeda, Sergio	NULL	0
2	2	Coronel Castillo, Gustavo	1	1
3	3	Marcelo Villalobos, Ricardo	1	1
4	4	Henostroza Macedo, Guino	1	1
5	10	Becerra Florez, Ursula	4	2
6	7	Valencia Morales, Pedro	3	2
7	8	Carrasco Muñoz, Joel	3	2
8	9	Arista Arbildo, Nori	3	2
9	11	Alcántara Cerna, Violeta	7	3
10	12	Zegarra Zavaleta, Daniel	7	3
11	13	Gonzales Villegas, Julio	7	3
12	14	Guerra Guerra, Jorge	11	4
13	5	Lucho Lutgardo, Edgar	2	2
14	6	Flores Manco, Julio	2	2

Observe que la función muestra a todos los subordinados indicando para cada subordinado, cuántos niveles se encuentra por debajo del empleado que se le entregó como argumento.

Para obtener la lista de responsables por departamento con sus subordinados, ejecute la siguiente consulta APPLY que invoca a la función **udf_ListaSubordinados**.

```
-- Subordinados del responsable de cada departamento
SELECT * FROM Departamento d
      CROSS APPLY udf_ListaSubordinados(d.idResponsable) ls
go
```

	idDepartamento	nombre	idResponsable	idEmpleado	nombre	idSuperior	nivel
1	1	Recursos Humanos	2	2	Coronel Castillo, Gustavo	1	0
2	1	Recursos Humanos	2	5	Lucho Lutgardo, Edgar	2	1
3	1	Recursos Humanos	2	6	Flores Manco, Julio	2	1
4	2	Marketing	7	7	Valencia Morales, Pedro	3	0
5	2	Marketing	7	11	Alcántara Cerna, Violeta	7	1
6	2	Marketing	7	12	Zegarra Zavaleta, Daniel	7	1
7	2	Marketing	7	13	Gonzales Villegas, Julio	7	1
8	2	Marketing	7	14	Guerra Guerra, Jorge	11	2
9	3	Finanzas	8	8	Carrasco Muñoz, Joel	3	0
10	4	Investigación & D...	9	9	Arista Arbildo, Nori	3	0
11	5	Capacitación	4	4	Henostroza Macedo, G...	1	0
12	5	Capacitación	4	10	Becerra Florez, Ursula	4	1

En el resultado no aparece el departamento Logística ya que éste no tiene registrado a su responsable. Para incluirlo en el resultado ejecute OUTER APPLY.

```
SELECT * FROM Departamento d
        OUTER APPLY udf_ListaSubordinados(d.idResponsable) ls
go
```

	idDepartamento	nombre	idResponsable	idEmpleado	nombre	idSuperior	nivel
1	1	Recursos Humanos	2	2	Coronel Castillo, ...	1	0
2	1	Recursos Humanos	2	5	Lucho Lutgardo, ...	2	1
3	1	Recursos Humanos	2	6	Flores Manco, Julio	2	1
4	2	Marketing	7	7	Valencia Morales,...	3	0
5	2	Marketing	7	11	Alcántara Cerna, ...	7	1
6	2	Marketing	7	12	Zegarra Zavaleta,...	7	1
7	2	Marketing	7	13	Gonzales Villegas...	7	1
8	2	Marketing	7	14	Guerra Guerra, Jo...	11	2
9	3	Finanzas	8	8	Carrasco Muñoz, ...	3	0
10	4	Investigación & D...	9	9	Arista Arbildo, Nori	3	0
11	5	Capacitación	4	4	Henostroza Mace...	1	0
12	5	Capacitación	4	10	Becerra Florez, Ur...	4	1
13	6	Logística	NULL	NULL	NULL	NULL	NULL

Esta página se ha dejado en blanco intencionalmente.