Capítulo XI

Transacciones y Procedimientos Almacenados

En este capítulo aprenderemos a definir transacciones, un mecanismo que garantiza la consistencia de la base de datos al momento de ejecutar operaciones con los datos. También aprenderemos a crear procedimientos almacenados para ejecutar código del lado del servidor que entregue resultados a las aplicaciones cliente.

1. ¿QUÉ ES UNA TRANSACCIÓN?

Una **transacción** es un conjunto de modificaciones de datos (operaciones) que debe ser procesado como una unidad.

Por ejemplo, se tiene las tablas GUIA_ENVIO, GUIA_DETALLE y ARTICULO de la base de datos **QhatuPERU**. La tabla ARTICULO registra el nivel de inventario (stock) de cada artículo.

Supongamos que se tiene que registrar el envío a una de las tiendas de 10 unidades del producto ABC. Para ello, la aplicación procede de la siguiente manera:

Operación 1 Registra en la tabla GUIA_ENVIO, los datos de la cabecera de la guía de remisión.

Operación 2 Registra en la tabla GUIA_DETALLE, los detalles del envío.

Operación 3 Actualiza el nivel de inventario del artículo ABC en la tabla ARTICULO.

Pregunta:

¿Qué ocurriría si durante la ejecución de la aplicación se completan las operaciones 1 y 2, y por una falla del sistema, la operación 3 no se lleva a cabo?

Respuesta:

Si no se diseña un mecanismo para corregir el error, la base de datos perdería consistencia. Según las tablas GUIA_ENVIO y GUIA_DETALLE se han despachado 10 unidades del artículo ABC, por lo que en la tabla ARTICULO deberíamos tener 10 unidades menos. Esto último no es cierto al no haberse completado la operación 3.

En otras palabras, las operaciones 1, 2 y 3 forman una transacción, es decir una unidad lógica de procesamiento, y si ésta no se completa, el sistema debe deshacer todas las operaciones.

Una transacción asegura que las operaciones se llevarán a cabo completas, o en caso contrario, la transacción será anulada para garantizar la consistencia de los datos.

2. MODOS DE TRANSACCIÓN PARA UNA SESIÓN

Las transacciones pueden ser de confirmación automática, explícitas, o implícitas.

Transacción de confirmación automática (autocommit)

Cada sentencia es considerada una transacción. Se presenta cuando cada sentencia es confirmada automáticamente cuando se completa. No es necesario especificar ninguna instrucción de control de transacciones.

En este caso, las modificaciones hechas por cada una de las sentencias INSERT, UPDATE o DELETE no podrán ser deshechas o anuladas. Este es el modo predeterminado del motor de base de datos.

Transacción explícita

Es aquella definida explícitamente por el programador utilizando la sentencia BEGIN TRANSACTION.

Consiste en un conjunto de sentencias agrupadas después de BEGIN TRANSACTION, y que culmina con la ejecución de la sentencia COMMIT TRANSACTION que confirma los cambios ejecutados, o con la ejecución de la sentencia ROLLBACK TRANSACTION que anula los cambios.

Transacción implícita

Se presenta cuando se ha ejecutado la instrucción SET IMPLICIT_TRANSACTIONS ON. En este caso no es necesaria la ejecución de la instrucción BEGIN

TRANSACTION para iniciar una transacción. Una nueva transacción se inicia implícitamente después de la ejecución de una sentencia COMMIT TRANSACTION o ROLLBACK TRANSACTION.

2.1. La opción IMPLICIT_TRANSACTIONS

SET IMPLICIT_TRANSACTIONS ON establece el modo de transacciones implícitas. En este modo, siempre y cuando no hay una transacción abierta, la ejecución de cualquiera de las siguientes instrucciones inicia una transacción:

- ALTER TABLE
- BEGIN TRANSACTION
- CREATE
- DELETE
- DROP
- FETCH
- GRANT
- INSERT
- OPEN
- REVOKE
- SELECT
- TRUNCATE TABLE
- UPDATE

Para confirmar o revertir una transacción implícita se debe ejecutar la sentencia COMMIT TRANSACTION o la sentencia ROLLBACK TRANSACTION. De no hacerlo, todos los cambios se revierten cuando la sesión se desconecta.

La ejecución de SET IMPLICIT_TRANSACTIONS OFF establece la sesión en el modo de transacciones de confirmación automática.

2.2. Las sentencias BEGIN TRANSACTION y COMMIT TRANSACTION

Sintáxis

```
BEGIN TRANSACTION [ nombre_transacción ]
   sentencias_SQL_transaccionales
COMMIT TRANSACTION [ nombre_transacción ]
```

- La sentencia BEGIN TRANSACTION define que cada una de las sentencias SQL que la siguen forman parte de una transacción.
- Todas las sentencias SQL después de BEGIN TRANSACTION deben ejecutarse para que se considere que la transacción se ha completado, salvo que se decida anular la transacción de manera explícita ejecutando ROLLBACK TRANSACTION.
- La sentencia COMMIT TRANSACTION confirma todas las modificaciones llevadas a cabo por las sentencias SQL anteriores y finaliza la transacción.

Cada transacción es registrada en el log de transacciones de la base de datos para mantener la consistencia de la base de datos y ayudar en la recuperación ante la eventualidad de una falla del sistema.

2.3. Cancelación de una transacción – La sentencia ROLLBACK TRANSACTION

Como una parte del manejo de errores, puede incluir dentro de la transacción, sentencias de control, de manera tal que al producirse un error, la transacción pueda ser deshecha utilizando la sentencia ROLLBACK TRANSACCIÓN.

Sintáxis

```
ROLLBACK TRANSACTION [ nombre_transacción ]
```

Ejercicio 11.1: Transacciones implícitas y explícitas

En este ejercicio estableceremos IMPLICIT_TRANSACTIONS en ON e iniciaremos transacciones implícitas y explícitas. Usaremos la variable de sistema @@TRANCOUNT para establecer las transacciones abiertas.

Digite y ejecute las siguientes instrucciones, una por una, en una ventana de consulta:

```
CREATE DATABASE Pruebas

USE Pruebas
```

```
SET NOCOUNT ON
```

Establezca el modo transacciones de confirmación automática.

```
SET IMPLICIT_TRANSACTIONS OFF

CREATE TABLE Prueba1(
     colA int )

INSERT INTO Prueba1 VALUES(1)
PRINT @@TRANCOUNT
-- 0 transacciones abiertas

-- Inicio de una transacción explícita
BEGIN TRANSACTION
INSERT INTO Prueba1 VALUES(2)
PRINT @@TRANCOUNT
-- 1 transacción abierta

COMMIT TRANSACTION
PRINT @@TRANCOUNT
-- 0 transacciones abiertas
```

Establezca el modo transacciones implícitas.

```
SET IMPLICIT_TRANSACTIONS ON

INSERT INTO Prueba1 VALUES(3)
PRINT @@TRANCOUNT
-- 1 transacción abierta
-- pese a que no se ha ejecutado
-- BEGIN TRANSACTION

COMMIT TRANSACTION
PRINT @@TRANCOUNT
-- 0 transacciones abiertas

BEGIN TRANSACTION
PRINT @@TRANCOUNT
-- 2 transacciones abiertas
-- IMPLICIT_TRANSACTIONS es ON
-- se anida la transacción explícita
```

```
INSERT INTO Prueba1 VALUES(4)

COMMIT TRANSACTION
PRINT @@TRANCOUNT
-- 1 transacción abierta

COMMIT TRANSACTION
-- cerramos la transacción abierta

SET IMPLICIT_TRANSACTIONS OFF
```

Ejercicio 11.2: Error de sintáxis

Digite y ejecute las siguientes instrucciones en una nueva ventana de consulta:

```
USE Pruebas
go

CREATE TABLE PruebaBatch(
   ColA int PRIMARY KEY,
   ColB char(3) not null )
go
```

Ahora digite y ejecute el siguiente batch. La instrucción GO al final del conjunto de sentencias define un batch ó lote de instrucciones a enviar y ejecutar juntas en el servidor. Observe que se ha establecido premeditadamente un error de sintáxis.

```
INSERT INTO PruebaBatch VALUES(1, 'aaa')
INSERT INTO PruebaBatch VALUES(2, 'bbb')
INSERT INTO PruebaBatch VALEUS(3, 'ccc')
-- Error de sintáxis en el último INSERT
go
```

Se recibe el siguiente mensaje:

```
Mens. 102, Nivel 15, Estado 1, Línea 3
Sintaxis incorrecta cerca de 'VALEUS'.
```

Verifique el contenido de la tabla **PruebaBatch**.

```
SELECT * FROM PruebaBatch
go
```

No se ha ejecutado ninguna de las sentencias INSERT.

Ejercicio 11.3: Error en tiempo de ejecución

Digite y ejecute las siguientes instrucciones en una ventana de consulta:

```
USE Pruebas
go

INSERT INTO PruebaBatch VALUES(1, 'aaa')
INSERT INTO PruebaBatch VALUES(2, 'bbb')
INSERT INTO PruebaBathc VALUES(3, 'ccc')
-- Error en nombre de tabla
go

SELECT * FROM PruebaBatch
go
```

Observe que el tercer INSERT tiene un error en el nombre de archivo. Pese a ello, los dos primeros INSERT se ejecutaron sin problemas. El motor de base de datos utiliza resolución de nombres diferida; es decir, que no verifica los nombres hasta el momento que va a ejecutar la instrucción.

Adicionalmente, el motor de base de datos se encuentra en el modo transacciones de confirmación automática (autocommit).

Ejercicio 11.4: Transacción implícita

Digite y ejecute las siguientes instrucciones en la ventana de consulta:

```
USE Pruebas
go

CREATE TABLE PruebaBatch2(
   ColA int PRIMARY KEY,
```

```
   ColB char(3) not null )
go
```

Ahora establezca el modo transacciones implícitas.

```
SET IMPLICIT_TRANSACTIONS ON
go
```

Inserte las siguientes filas:

```
INSERT INTO PruebaBatch2 VALUES(1, 'aaa')
INSERT INTO PruebaBatch2 VALUES(2, 'bbb')
INSERT INTO PruebaBatch2 VALUES(3, 'ccc')
go

SELECT * FROM Pruebabatch2
go
```

Observe que las filas se han insertado en la tabla.

Ahora vamos a cancelar la transacción.

```
ROLLBACK TRANSACTION
go

SELECT * FROM PruebaBatch2
go

COMMIT TRANSACTION
go

SET IMPLICIT_TRANSACTIONS OFF
go
```

Note que las filas ya no se encuentran en la tabla. Cuando el modo es transacciones implícitas, una transacción finaliza cuando se la confirma (COMMIT) ó cuando se la cancela (ROLLBACK).

El último COMMIT TRANSACTION es necesario, ya que la instrucción SELECT ejecutada después del ROLLBACK TRANSACTION inicia una segunda transacción.

Ejercicio 11.5: Transacción explícita

Digite y ejecute las siguientes instrucciones en una ventana de consulta nueva:

```
USE Pruebas
go

CREATE TABLE Padre(
   codPadre int PRIMARY KEY,
   nomPadre varchar(15) not null )

CREATE TABLE Hijo(
   codHijo int PRIMARY KEY,
   nomHijo varchar(15) not null,
   codPadre int not null FOREIGN KEY
   REFERENCES Padre )
go
```

Defina y ejecute la siguiente transacción explícita. En ella se ha establecido intencionalmente un error de integridad referencial.

```
BEGIN TRANSACTION
   INSERT INTO Padre VALUES(100, 'Juan')
      IF (@@error <> 0) GOTO hayError
   INSERT INTO Hijo VALUES(101, 'Pedro', 99)
   -- padre 99 no existe
      IF (@@error <> 0) GOTO hayError
COMMIT TRANSACTION
RETURN
hayError:
ROLLBACK TRANSACTION
go
```

Se recibe los mensajes:

```
(1 filas afectadas)
Mens. 547, Nivel 16, Estado 0, Línea 4
Instrucción INSERT en conflicto con
```

```
la restricción FOREIGN KEY
"FK__Hijo__codPadre__173876EA". El conflicto
ha aparecido en la base de datos "Pruebas",
tabla "dbo.Padre", column 'codPadre'.
Se terminó la instrucción.
```

Ahora consulte las tablas. Note que ninguno de los INSERT se ha confirmado.

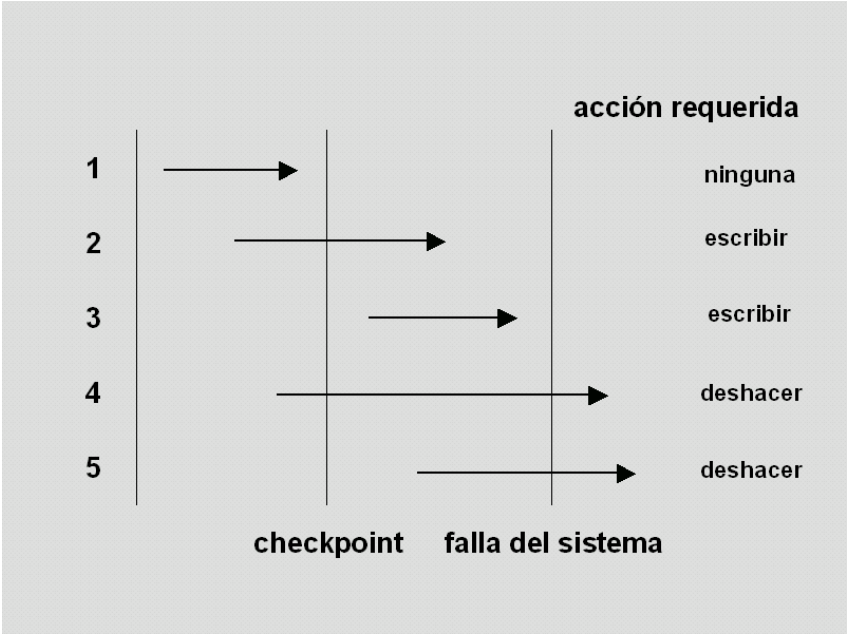
```
SELECT * FROM Padre
SELECT * FROM Hijo
go
```

3. EL PROCESO CHECKPOINT Y LA RECUPERACIÓN DE TRANSACCIONES

El archivo log de transacciones de una base de datos registra todas las transacciones ejecutadas en ella, tanto las transacciones confirmadas (commited transactions) como las no confirmadas (uncommited transactions), lo que garantiza que todas las transacciones confirmadas se reflejarán en la base de datos en la eventualidad de una falla del sistema.

SQL Server ejecuta periódicamente el proceso checkpoint, que consiste en la verificación del log de transacciones para determinar las transacciones confirmadas y escribirlas en la base de datos. Las transacciones no confirmadas son ignoradas y no producen cambios en la base de datos.

El siguiente diagrama resume algunas situaciones que pueden presentarse cuando se produce una falla del sistema.



- Para la transacción 1 se verificó COMMIT antes del checkpoint, por lo tanto ya está confirmada plenamente en la base de datos.
- Para las transacciones 2 y 3 el COMMIT se verificó despues del checkpoint, por lo tanto deben ser reconstruidas desde el log de transacciones.
- Para las transacciones 4 y 5 no se verificó el COMMIT, por lo tanto deben ser deshechas.

3.1. Consideraciones al definir transacciones

Al momento de definir las transacciones tenga en cuenta lo siguiente:

- Las transacciones deben ser cortas. Durante la ejecución de una transacción, los elementos de la base de datos que la transacción está utilizando se bloquean. Si la transacción toma demasiado tiempo se incrementa la probabilidad de que los usuarios no puedan acceder a la data bloqueada.
- Evite que la transacción interactúe con el usuario. Toda interacción con el usuario se debe ejecutar a nivel de la aplicación cliente y antes que la transacción se inicie. Cuando la transacción se inicia, ésta debe finalizar sin la intervención del usuario.
- Las principales sentencias en una transacción deben ser INSERT, UPDATE y DELETE.

- Evite anidar transacciones. Puede utilizar la variable global @@TRANCOUNT para determinar cuántas transacciones están abiertas y cuán profundo es el anidamiento. La sentencia BEGIN TRANSACTION incrementa el valor de la variable en uno, y la sentencia ROLLBACK TRANSACTION la configura a cero, salvo en el caso de las transacciones anidadas cuando la sesión se encuentra en el modo transacciones implícitas.
- No se permite la creación de tablas temporales dentro de una transacción. Algunos procedimientos almacenados del sistema no pueden utilizarse porque generan tablas temporales.
- Algunas de las sentencias que tienen que ver con la creación, modificación o eliminación de objetos, no pueden ser utilizadas dentro de una transacción.

4. ¿QUÉ ES UN PROCEDIMIENTO ALMACENADO?

Un procedimiento almacenado es un subprograma que se almacena como un objeto en la base de datos y ejecuta un conjunto 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:

- Realizan operaciones en la base de datos y pueden invocar a otros procedimientos.
- Pueden recibir parámetros de entrada.
- Pueden entregar sus resultados en forma de parámetros de salida.
- Pueden entregar un valor de retorno que puede utilizarse como valor de estado para indicar el éxito o fracaso de la ejecución del procedimiento.

4.1. Creación de procedimiento que no recibe ni entrega parámetros

Sintáxis

```
CREATE PROCEDURE nombreProcedimiento
AS
sentenciasSQL
```

Ejercicio 11.6: Procedimiento sin parámetros

Crear un procedimiento almacenado que genere el catálogo de artículos en la base de datos **QhatuPERU**.

Digite y ejecute el siguiente código en una ventana de consulta:

```
USE QhatuPERU
go

CREATE PROCEDURE usp_CatalogoArticulos
AS
SET NOCOUNT ON;
SELECT LINEA.NomLinea,
     ARTICULO.CodArticulo,
     ARTICULO.DescripcionArticulo,
     ARTICULO.Presentacion,
     ARTICULO.PrecioProveedor
FROM LINEA INNER JOIN ARTICULO
     ON LINEA.CodLinea = ARTICULO.CodLinea;
go
```

Para ejecutar el procedimiento

```
EXEC usp_CatalogoArticulos
go
```

Resultados | Mensajes

	NomLinea	CodArticulo	DescripcionArticulo	Presentacion	PrecioProveedor
1	GOLOSINAS	1	CARAMELOS BASTON VIENA ARCOR	PAQUETE 454 GR	1,50
2	GOLOSINAS	2	CARAMELOS SURTIDO DE FRUTAS	PAQUETE 450 GR	1,00
3	GOLOSINAS	3	CARAMELOS FRUTAS SURTIDA ARCOR	PAQUETE 520 GR	1,50
4	GOLOSINAS	4	CARAMELOS FRUTAS MASTICABLES	PAQUETE 454 GR	1,30
5	GOLOSINAS	5	CHUPETES LOLY AMBROSOLI	KILOGRAMO	1,20
6	GOLOSINAS	6	FRUNA SURTIDA DONOFRIO	PAQUETE X 24 UNIDADES	1,80
7	GOLOSINAS	7	CHOCOLATE DOÑA PEPA FIELD	PAQUETE X 6 UNIDADES	2,20
8	GOLOSINAS	8	CHOCOLATE CUA CUA FIELD	PAQUETE X 6 UNIDADES	1,60
9	GOLOSINAS	9	MELLOWS FAMILIAR FIELD	PAQUETE 454 GR	2,10
10	GOLOSINAS	10	WAFER CHOCOLATE FIELD	PAQUETE X 9 UNIDADES	0,70

4.2. Creación de procedimiento que recibe parámetros

Sintáxis

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

Ejercicio 11.7: Procedimiento que recibe un parámetro

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

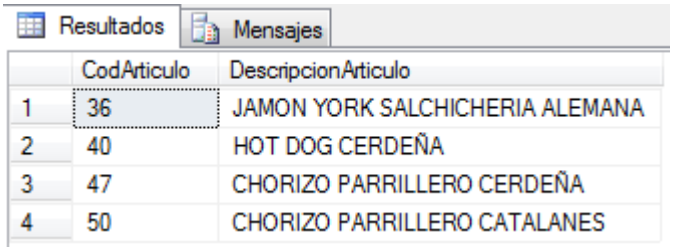
```
USE QhatuPERU
go

CREATE PROCEDURE usp_ArticulosProveedor
     @proveedor varchar(40)
AS
SET NOCOUNT ON;
SELECT ARTICULO.CodArticulo,
     ARTICULO.DescripcionArticulo
FROM PROVEEDOR INNER JOIN ARTICULO
     ON PROVEEDOR.CodProveedor =
          ARTICULO.CodProveedor
WHERE PROVEEDOR.NomProveedor = @proveedor;
go
```

Para ejecutar el procedimiento

```
DECLARE @elProveedor varchar(40)
SET @elProveedor = 'Embutidos El Gordito'
EXEC usp_ArticulosProveedor @elProveedor
go
```

También,

```
EXEC    usp_ArticulosProveedor    'Embutidos    El
Gordito'
go
```



Resultados | Mensajes

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

Ejercicio 11.8: Procedimiento que recibe más de un parámetro

Crear un procedimiento que entregue una lista de las guías de envío emitidas en el mes M del año A y enviadas a la tienda ubicada en el distrito de Pueblo Libre.

```
USE QhatuPERU
go

CREATE PROCEDURE usp_GuiasFechaTienda
     @tienda varchar(20),
     @año smallint,
     @mes smallint
AS
SET NOCOUNT ON;
SELECT GUIA_ENVIO.NumGuia,
     GUIA_ENVIO.FechaSalida,
     GUIA_ENVIO.CodTienda
FROM TIENDA INNER JOIN GUIA_ENVIO
     ON TIENDA.CodTienda = GUIA_ENVIO.CodTienda
WHERE TIENDA.Distrito = @tienda
     AND YEAR(GUIA_ENVIO.FechaSalida) = @año
     AND MONTH(GUIA_ENVIO.FechaSalida) = @mes;
go
```

Para ejecutar el procedimiento entregando los parámetros por posición

En este caso los parámetros se deben entregar al procedimiento en el mismo orden en que fueron definidos.

```
--  entrega de parámetros por posición
DECLARE @laTienda varchar(20)
DECLARE @elAño smallint, @elMes smallint
```

```
SET @laTienda = 'Pueblo Libre'
SET @elAño = 2013
SET @elMes = 3
EXEC usp_GuiasFechaTienda @laTienda,
     @elAño, @elMes
go
```

También,

```
EXEC usp_GuiasFechaTienda 'Pueblo Libre', 2013, 3
go
```

Para ejecutar el procedimiento entregando los parámetros por referencia

En este caso los valores de los parámetros se pueden pasar en cualquier orden, pero hay que especificar a qué parámetro se asigna cada valor.

```
-- entrega de parámetros por referencia
EXEC usp_GuiasFechaTienda
     @año = 2013,
     @mes = 3,
     @tienda = 'Pueblo Libre'
go
```

Resultados | Mensajes

	NumGuia	FechaSalida	CodTienda
1	2	2013-03-25 20:38:04.473	2
2	7	2013-03-25 20:38:04.577	2
3	12	2013-03-25 20:38:04.650	2
4	17	2013-03-25 20:38:04.693	2
5	22	2013-03-25 20:38:04.753	2
6	27	2013-03-25 20:38:04.813	2
7	32	2013-03-26 20:38:04.863	2
8	37	2013-03-26 20:38:04.897	2
9	42	2013-03-26 20:38:04.943	2
10	47	2013-03-26 20:38:05.010	2

4.3. Creación de procedimiento que recibe y entrega parámetros

Sintáxis

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

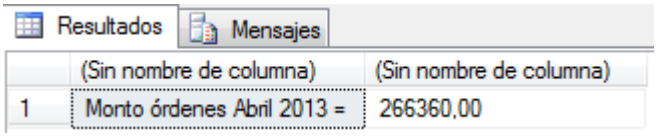
Ejercicio 11.9: Procedimiento que recibe y entrega parámetros

Crear un procedimiento que entregue el monto mensual total de las órdenes de compra correspondientes al mes M del año A.

```
USE QhatuPERU
go

CREATE PROCEDURE usp_MontoOrdenesAñoMes
     @año smallint,
     @mes smallint,
     @monto money OUTPUT
AS
SET NOCOUNT ON;
SET @monto =
     ( SELECT SUM(ORDEN_DETALLE.PrecioCompra *
          ORDEN_DETALLE.CantidadRecibida)
     FROM ORDEN_COMPRA INNER JOIN ORDEN_DETALLE
          ON ORDEN_COMPRA.NumOrden =
               ORDEN_DETALLE.NumOrden
     WHERE YEAR(ORDEN_COMPRA.FechaIngreso) = @año
          AND MONTH(ORDEN_COMPRA.FechaIngreso) =
               @mes);
go
```

Para ejecutar el procedimiento

```
DECLARE @elMonto money
EXEC usp_MontoOrdenesAñoMes 2013, 4,
     @elMonto OUTPUT
SELECT 'Monto órdenes Abril 2013 = ', @elMonto
go
```



5. ¿PORQUÉ USAR PROCEDIMIENTOS ALMACENADOS?

Las siguientes son razones por la que deberíamos programar procedimientos almacenados para ejecutar las operaciones en el servidor:

- Se reduce el tráfico de red entre el cliente y el servidor ya que el procedimiento encapsula una serie de instrucciones que se ejecuta como un lote en el servidor; si no usaramos un procedimiento tendríamos que enviar las instrucciones una por una al servidor.
- Refuerza la seguridad ya que no es necesario conceder a los usuarios permisos explícitos sobre los objetos que el procedimiento utiliza. Los usuarios acceden a dichos objetos a través del procedimiento aún si no tienen permisos sobre ellos.
- La lógica de la aplicación puede ser compartida con otras aplicaciones al estar almacenada en una ubicación centralizada.
- El mantenimiento es más simple, ya que al encontrarse la lógica de negocios almacenada en el servidor, cualquier cambio se realiza en el servidor sin afectar a las aplicaciones de los usuarios.
- Cuando enviamos una instrucción SQL al servidor, el motor de datos debe interpretarla (verificar sintáxis y resolver las referencias a identificadores de los objetos), compilarla, crear el plan de ejecución, cargar el plan en la memoria, y ejecutarla. Con un procedimiento almacenado el rendimiento mejora ya que el procedimiento se guarda precompilado y con su plan de ejecución ya definido.

6. OBTENCIÓN DE LA METADATA DE UN PROCEDIMIENTO ALMACENADO

Podemos utilizar procedimientos almacenados del sistema para averiguar cómo se ha definido un procedimiento almacenado.

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

7. MODIFICACIÓN Y ELIMINACIÓN DE UN PROCEDIMIENTO ALMACENADO

Utilice la sentencia ALTER PROCEDURE para modificar un procedimiento almacenado. La definición previa del procedimiento será reemplazada por la definición establecida en ALTER PROCEDURE.

Sintáxis

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

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

Ejecute el procedimiento almacenado **sp_depends** antes de eliminar un procedimiento almacenado para examinar qué objetos dependen del procedimiento almacenado.

Sintáxis

```
DROP PROCEDURE nombreProcedimiento
```

8. EJERCICIOS PROPUESTOS

1. Escriba un procedimiento que entrega una lista de los artículos de la línea L, donde L es el nombre de la línea de artículos.
2. Escriba un procedimiento que entrega una lista de los artículos del proveedor P y de la línea L, donde P y L son el nombre del proveedor y el nombre de la línea respectivamente.
3. Crear un procedimiento que entregue el monto total de los artículos enviados a la tienda T, donde T es el código de la tienda.
4. Crear un procedimiento que entregue el monto total de los artículos enviados a la tienda T durante el mes M del año A.
5. Crear un procedimiento que entregue la lista de los N artículos más enviados a las tiendas; N es un entero.
6. Crear un procedimiento que entregue el balance entrada/salida del artículo A. El procedimiento debe entregar el total de unidades que ingresaron, y el total de unidades que salieron del artículo A.
7. Crear un procedimiento que entregue el monto total adquirido al proveedor P durante el año A, donde P es el código del proveedor.
8. Crear un procedimiento que en la tabla ARTICULO permita actualizar el precio del artículo A, donde A es el código del artículo.
9. Crear un procedimiento que permita registrar los datos de un artículo nuevo.
10. Crear un procedimiento que calcule el monto promedio mensual enviado a la tienda T, donde T es el código de la tienda.