Pregrado

Programa de Ingeniería de Sistemas

# GESTIÓN DE DATOS E INFORMACIÓN II

## Sesión 6

**Tema:**

**Procedimientos almacenados**

## Resultado de aprendizaje

Implementa procedimientos almacenados, cursores y triggers sobre una base de datos utilizando T-SQL para resolver las diversas actividades de una organización.

## Evidencia de aprendizaje

Los estudiantes demostrarán su comprensión y habilidades sobre el uso de procedimientos almacenados ,diseñando, implementando y optimizando para abordar necesidades específicas en una base de datos empresarial.

# Contenido

## Nombre del tema

- Procedimientos almacenados
- Transacciones
- Manejo de errores

# Revisa el siguiente video:


https://youtu.be/jTYttzh63Uo?si=d3KGlGH3TC3Nitvt

**Después de haber visualizado el video en la slide anterior, reflexionamos y respondemos las siguientes interrogantes:**

01 ¿Qué es un procedimiento almacenado?

02 ¿Para qué sirve un procedimiento almacenado?

03 ¿Cuales son las ventajas de utilizar un procedimiento almacenado?

Tema

# Procedimientos almacenados

## PROCEDIMIEMTOS ALMACENADOS

Está formado por un conjunto de instrucciones **Transact-SQL** que definen un determinado proceso, puede aceptar parámetros de entrada y devolver un valor o conjunto de resultados. Este procedimiento se guarda en el servidor y puede ser ejecutado en cualquier momento.

- ✓ **SQL Server** incorpora procedimientos almacenados del sistema, se encuentran en la base de datos master y se reconocen por su nombre, todos tienen un nombre que empieza por **sp_**.

- ✓ Permiten recuperar información de las tablas del sistema y pueden ejecutarse en cualquier base de datos del servidor.

## PROCEDIMIEMTOS ALMACENADOS

- ✓ También están los procedimientos de usuario, los crea cualquier usuario que tenga los permisos oportunos.

- ✓ Los **procedimientos almacenados** se diferencian de las instrucciones SQL ordinarias y de los lotes de instrucciones SQL en que están precompilados.

- ✓ La primera vez que se ejecuta un procedimiento, el procesador de **consultas de SQL Server** lo analiza y prepara un plan de ejecución que se almacena en una tabla del sistema.

## PROCEDIMIEMTOS ALMACENADOS

- ✓ Posteriormente, el procedimiento se ejecuta según el plan almacenado. Puesto que ya se ha realizado la mayor parte del trabajo de procesamiento de consultas, los **procedimientos almacenados** se ejecutan casi de forma instantánea por lo que el uso de procedimientos almacenados mejora notablemente la potencia y eficacia del SQL.

- ✓ Se pueden crear también **procedimiento temporales locales** y **globales**.

- ✓ Un procedimiento temporal local se crea por un usuario en una conexión determinada y sólo se puede utilizar en esa sesión, un **procedimiento temporal global** lo pueden utilizar todos los usuarios, cualquier conexión puede ejecutar un **procedimiento almacenado temporal global**.

## PROCEDIMIEMTOS ALMACENADOS

- ✓ Éste existe hasta que se cierra la conexión que el usuario utilizó para crearlo, y hasta que se completan todas las versiones del procedimiento que se estuvieran ejecutando mediante otras conexiones.

- ✓ Una vez cerrada la conexión que se utilizó para crear el procedimiento, éste ya no se puede volver a ejecutar, sólo podrán finalizar las conexiones que hayan empezado a ejecutar el procedimiento.

- ✓ Tanto los **procedimientos temporales** como los **no temporales** se crean y ejecutan de la misma forma, el nombre que le pongamos indicará de qué tipo es el procedimiento.

- ✓ Los procedimientos almacenados se crean mediante la sentencia **CREATE PROCEDURE** y se ejecutan con **EXEC** (o **EXECUTE**).

## PROCEDIMIEMTOS ALMACENADOS

Para ejecutarlo también se puede utilizar el nombre del procedimiento almacenado sólo, siempre que sea la primera palabra del lote. Para eliminar un procedimiento almacenado utilizamos la sentencia **DROP PROCEDURE**.

```
CREATE {PROC|PROCEDURE} [NombreEsquema.]NombreProcedimiento
    [{@parametro tipo} [VARYING] [= valorPredet]  [OUT|OUTPUT] ] [,...n]
    AS {  <bloque_instrucciones> [ ...n] }[;]
<bloque_instrucciones> ::=
{[BEGIN] instrucciones [END] }
```

## PROCEDIMIEMTOS ALMACENADOS

- ✓ Las instrucciones **CREATE PROCEDURE** no se pueden combinar con otras instrucciones **SQL** en el mismo lote.

- ✓ Después del verbo **CREATE PROCEDURE** indicamos el nombre del procedimiento, opcionalmente podemos incluir el nombre del esquema donde queremos que se cree el procedimiento, por defecto se creará en **dbo**. Ya que Sqlserver utiliza el prefijo **sp_** para nombrar los procedimientos del sistema se recomienda no utilizar nombres que empiecen por **sp_**.

- ✓ Como se puede deducir de la sintaxis (no podemos indicar un nombre de **base de datos** asociado al nombre del procedimiento) sólo se puede crear el procedimiento almacenado en la base de datos actual, no se puede crear en otra base de datos.

## PROCEDIMIEMTOS ALMACENADOS

- ✓ Si queremos definir un **procedimiento temporal local** el nombre deberá empezar por una almohadilla (#) y si el procedimiento es temporal global el nombre debe de empezar por **##**.

- ✓ El nombre completo de un procedimiento almacenado o un procedimiento almacenado temporal global, incluidas **##**, no puede superar los 128 caracteres.

- ✓ El nombre completo de un procedimiento almacenado temporal local, incluidas **#**, no puede superar los **116 caracteres**.

- ✓ **Transact-SQL** permite abreviar la palabra reservada **PROCEDURE** por **PROC** sin que ello afecte a la funcionalidad de la instrucción.

## PROCEDIMIEMTOS ALMACENADOS

- ✓ **@parametro:** representa el nombre de un parámetro. Se pueden declarar uno o más parámetros indicando para cada uno su nombre (debe de empezar por arroba) y su tipo de datos, y opcionalmente un valor por defecto (=valorPredet) este valor será el asumido si en la llamada el usuario no pasa ningún valor para el parámetro.

- ✓ Un **procedimiento almacenado** puede tener un **máximo de 2.100** parámetros.

- ✓ Los parámetros son locales para el procedimiento; los mismos nombres de parámetro se pueden utilizar en otros procedimientos. De manera predeterminada, los parámetros sólo pueden ocupar el lugar de expresiones constantes; no se pueden utilizar en lugar de nombres de tabla, nombres de columna o nombres de otros objetos de base de datos.

## PROCEDIMIEMTOS ALMACENADOS

- ✓ **VARYING** Sólo se aplica a los parámetros de tipo cursor. **OUTPUT** | **OUT** (son equivalentes) Indica que se trata de un parámetro de salida. El valor de esta opción puede devolverse a la instrucción **EXECUTE** que realiza la llamada.

- ✓ El parámetro variable **OUTPUT** debe definirse al crear el procedimiento y también se indicará en la llamada junto a la variable que recogerá el valor devuelto del parámetro.

- ✓ El nombre del parámetro y de la variable no tienen por qué coincidir; sin embargo, el tipo de datos y la posición de los parámetros deben coincidir a menos que se indique el nombre del parámetro en la llamada de la forma **@parametro=valor**.

## PROCEDIMIEMTOS ALMACENADOS

- ✓ **VARYING** Sólo se aplica a los parámetros de tipo cursor. **OUTPUT** | **OUT** (son equivalentes) Indica que se trata de un parámetro de salida. El valor de esta opción puede devolverse a la instrucción **EXECUTE** que realiza la llamada.

- ✓ El parámetro variable **OUTPUT** debe definirse al crear el procedimiento y también se indicará en la llamada junto a la variable que recogerá el valor devuelto del parámetro.

- ✓ El nombre del parámetro y de la variable no tienen por qué coincidir; sin embargo, el tipo de datos y la posición de los parámetros deben coincidir a menos que se indique el nombre del parámetro en la llamada de la forma **@parametro=valor**.

# PROCEDIMIEMTOS ALMACENADOS (PARAMETROS DE ENTRADA)

```
CREATE PROCEDURE  VerUsuariosPoblacion @pob CHAR(30),@pro CHAR(30)  AS
    SELECT * FROM usuarios WHERE  poblacion=@pob AND provincia = @pro;

GO
EXEC VerUsuariosPoblacion Madrid,  Valencia
```

## PROCEDIMIEMTOS ALMACENADOS (PARAMETROS OPCIONALES)

Para definir un **parámetro opcional** tenemos que asignarle un valor por defecto en la definición del procedimiento.

```
CREATE PROCEDURE VerUsuariosPoblacion2  @pob CHAR(30),@pro CHAR(30)='Madrid'  AS
    SELECT * FROM usuarios WHERE  poblacion=@pob AND provincia = @pro;

    GO
    EXEC VerUsuariosPoblacion2 Madrid
```

## PROCEDIMIEMTOS ALMACENADOS (PARAMETROS DE SALIDA)

Un procedimiento almacenado puede también devolver resultados, definiendo el parámetro como **OUTPUT** o bien utilizando la instrucción **RETURN**.

Para poder recoger el valor devuelto por el procedimiento, en la llamada se tiene que indicar una variable donde guardar ese valor.

```
CREATE PROC ultimo_contrato @ofi  INT, @fecha DATETIME OUTPUT    AS
    SELECT @fecha=(SELECT MAX(contrato)
                   FROM empleados
                    WHERE oficina=@ofi)
    GO
```

## PROCEDIMIEMTOS ALMACENADOS (RETURN)

**RETURN**

Ordena salir incondicionalmente de una consulta o procedimiento, se puede utilizar en cualquier punto para salir del procedimiento y las instrucciones que siguen a **RETURN** no se ejecutan. Además los procedimientos almacenados pueden devolver un valor entero mediante esta orden.

**RETURN [expresion_entera]**

Expresion_entera es el valor entero que se devuelve. A menos que se especifique lo contrario, todos los procedimientos almacenados del sistema devuelven el valor 0. Esto indica que son correctos y un valor distinto de cero indica que se ha producido un error.

## PROCEDIMIEMTOS ALMACENADOS (RETURN)

Cuando se utiliza con un procedimiento almacenado, **RETURN** no puede devolver un valor **NULL**. Si un procedimiento intenta devolver un valor **NULL** (por ejemplo, al utilizar **RETURN @var** si **@var** es **NULL**), se genera un mensaje de advertencia y se devuelve el valor **0**.

```
CREATE PROC trabajadores2 @ofi INT   AS
  RETURN (SELECT COUNT(*) FROM  empleados WHERE oficina=@ofi)
  GO


DECLARE @var INT;
EXEC @var= trabajadores2 12
PRINT @var
```

Tema

# Transacciones

## TRANSACCIONES

Una transacción es un conjunto de operaciones **Transact SQL** que se ejecutan como un único bloque, es decir, si falla una operación **Transact SQL** fallan todas.

- ✓ Si una transacción tiene éxito, todas las modificaciones de los datos realizadas durante la transacción se confirman y se convierten en una parte permanente de la base de datos.

- ✓ Si una transacción encuentra errores y debe cancelarse o revertirse, se borran todas las modificaciones de los datos.

## TRANSACCIONES

La sentencia que se utiliza para indicar el comienzo de una transacción es '**BEGIN TRAN**'. Si alguna de las operaciones de una transacción falla hay que deshacer la transacción en su totalidad para volver al estado inicial en el que estaba la base de datos antes de empezar. Esto se consigue con la sentencia '**ROLLBACK TRAN**'.

Si todas las operaciones de una transacción se completan con éxito hay que marcar el fin de una transacción para que la base de datos vuelva a estar en un estado consistente con la sentencia '**COMMIT TRAN**'.

## TRANSACCIONES Y PROCEDIMIENTOS ALMACENADOS

Cuando trabajamos **transacciones con procedimientos almacenados** debemos recordar que cada procedimiento almacenado es una unidad. Cuando se ejecuta lo hace de manera independiente de quien lo llama. Sin embargo si tenemos un **ROLLBACK TRAN** dentro de un **procedimiento almacenado** cancelaremos la transacción en curso, pero si hay una transacción externa al procedimiento en el que estamos trabajando se cancelará esa transacción externa.

# TRANSACCIONES Y PROCEDIMIENTOS ALMACENADOS

```
CREATE PROCEDURE NUEVOPASAJERO(
    @IDPAS CHAR(5), @NOM VARCHAR(40), @PAI VARCHAR(40),
    @TEL VARCHAR(15), @EMA VARCHAR(40))

AS
    BEGIN TRAN TPASAJERO
        DECLARE @IDPAI CHAR(4)
        SELECT @IDPAI = IDPAIS FROM PAIS WHERE PAIS=@PAI
        INSERT INTO PASAJERO
            VALUES (@IDPAS, @NOM, @IDPAI, @TEL, @EMA)
        IF @@ERROR = 0
        BEGIN
            PRINT 'PASAJERO REGISTRADO CON EXITO'
            COMMIT TRAN TPASAJERO
        END
        ELSE
        BEGIN
            PRINT 'OCURRIO UN ERROR AL INSERTAR'
            ROLLBACK TRAN TPASAJERO
         END
```

Tema

Manejo de errores

## MANEJO DE ERRORES

```
BEGIN TRY
      --code to try
END TRY
BEGIN CATCH
      --code to run if error occurs
--is generated in try
END CATCH
```

Cualquier cosa entre **BEGIN TRY** y **END TRY** es el código que queremos monitorear para detectar un error. Entonces, si se hubiera producido un error dentro de esta sentencia **TRY**, el control se habría transferido inmediatamente a la instrucción **CATCH** y luego habría empezado a ejecutar el código línea por línea.

## MANEJO DE ERRORES

**ERROR_NUMBER** – Devuelve el número interno del error

**ERROR_STATE** – Devuelve la información sobre la fuente

**ERROR_SEVERITY** – Devuelve la información sobre cualquier cosa, desde errores informativos hasta errores que el usuario de DBA puede corregir, etc.

**ERROR_LINE** – Devuelve el número de línea en el que ocurrió un error

**ERROR_PROCEDURE** – Devuelve el nombre del procedimiento almacenado o la función

**ERROR_MESSAGE** – Devuelve la información más esencial y ese es el mensaje de texto del error

# MANEJO DE ERRORES , TRANSACCIONES Y PROCEDIMIENTOS ALMACENADOS

```
CREATE PROCEDURE spAgregaUsuario
    @nom AS VARCHAR(50),
    @ape AS VARCHAR(50),
    @ema AS VARCHAR(30),
    @pas AS VARCHAR(20),
    @idJer AS BIGINT,
    @msg AS VARCHAR(100) OUTPUT
AS
BEGIN
    SET NOCOUNT ON;
    Begin Tran Tadd
    Begin Try
        INSERT INTO dbo.USUARIO_SYS (nombre, apellidos, email, pass, fecha_add) VALUES (@nom, @ape, @ema, @pas, GETDATE())
        INSERT INTO dbo.USUARIO_JERARQUIAS_SYS (id_usuario, id_jerarquia) VALUES (@@IDENTITY, @idJer)
        SET @msg = 'El Usuario se registro correctamente.'
        COMMIT TRAN Tadd
    End try
    Begin Catch
        SET @msg = 'Ocurrio un Error: ' + ERROR_MESSAGE() + ' en la línea ' + CONVERT(NVARCHAR(255), ERROR_LINE() ) + '.'
        Rollback TRAN Tadd
    End Catch
END
```

# Autoevaluación

Sesión 6

Pregunta 1

## ¿Cuál es el propósito de un procedimiento almacenado?

- Almacenar datos en la base de datos.
- Optimizar consultas y operaciones en la base de datos.
- Crear nuevas tablas en la base de datos.
- Ninguna.

Pregunta 2

## ¿Por qué es importante optimizar un procedimiento almacenado?

- Para reducir la legibilidad del código.
- Para mejorar el rendimiento y la eficiencia de las operaciones.
- Para aumentar el número de parámetros
- Todas.

Pregunta 3

## ¿Qué función cumplen los parámetros en un procedimiento almacenado?

- Almacenar datos permanentes.
- Facilitar la comunicación con la aplicación.
- Generar informes visuales.
- Todas.

# Autoevaluación

¡Vamos por más logros!

# ¡Felicitaciones!

Ha concluido la autoevaluación

# Conclusiones

El uso de **procedimientos almacenados** en **SQL Server** proporciona una mejora significativa en el rendimiento de las operaciones en la base de datos. Al ejecutarse de forma precompilada, los procedimientos almacenados reducen la carga en el servidor y minimizan el tiempo de ejecución de consultas y operaciones.

Los procedimientos almacenados permiten la reutilización eficiente de código SQL. Al encapsular lógica de negocio y consultas complejas en **procedimientos almacenados**, se facilita su uso en diferentes partes de una aplicación, promoviendo la coherencia y reduciendo la redundancia en el código.

Al centralizar la lógica de negocio en **procedimientos almacenados**, se mejora la mantenibilidad del código. Cualquier cambio o actualización en la lógica se realiza en un único lugar, lo que facilita la gestión y evita la dispersión de la lógica de acceso a datos en toda la aplicación.

Los **procedimientos almacenados** pueden reducir el tráfico de red al ejecutarse en el servidor de base de datos. Al enviar solo la llamada al procedimiento y recibir resultados, en lugar de enviar consultas SQL completas, se minimiza la cantidad de datos transferidos entre la aplicación y la base de datos.

# Aplicando lo aprendido:

**Desarrollar la Guía de Laboratorio N°6**

# Referencias


DURÁN, Luis. Bases de Datos con Visual Basic [en línea]. España: Marcombo, 2007. ISBN 9788426714237.
Disponible en: https://www.digitaliapublishing.com/a/17229

CAPACHO, José y Wilson NIETO. Diseño de Bases de Datos [en línea]. Barranquilla: Universidad del Norte, 2017. ISBN 9789587418255. Disponible en: https://search.ebscohost.com/login.aspx?direct=true&db=e000xww&AN=1690049&lang=es&site=ehost-live

WANUMEN Luis, RIVAS Edwin, Mosquera Darín. Bases de datos en SQL Server [en línea]. Bogotá: Ecoe Ediciones, 2017. ISBN 9789587715705. Disponible en: https://www.digitaliapublishing.com/a/66605

HUESO Luis. Bases de datos [en línea]. Madrid: Rama Editorial, 2014. ISBN 9788499641577.
Disponible en: https://www.digitaliapublishing.com/a/109943

PRIETO, Rafael. SGBD e instalación: administración de bases de datos (UF1469) [en línea]. Antequera : IC Editorial. ISBN 9788416433360. Disponible en: https://www.digitaliapublishing.com/a/86830


# Pregrado