

# **BASES DE DATOS**

**Stored  
Procedures**

# Agenda



- ▣ Procedures
- ▣ Functions

# Stored Procedures

- Son bloques de código almacenado en la BD.
- Lenguajes: Transact SQL, PL/SQL, PgSql, etc.
- Existen 2 tipos de procedimientos almacenados: PROCEDURES y FUNCTIONS
- Se almacenan en el Catálogo
  - sys.objects
  - sys.procedures
  - sys.parameters
  - sys.sql\_modules

# Procedures

```
CREATE PROCEDURE nombre (parámetros) AS
```

```
Begin
```

```
...
```

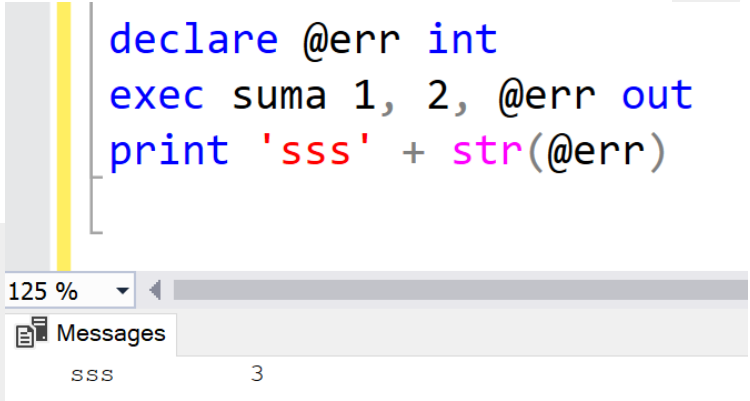
```
end;
```

```
CREATE PROCEDURE suma (@aa int, @bb int, @cc int OUT)  
AS
```

```
Begin
```

```
    set @cc = @aa + @bb
```

```
end
```



```
declare @err int  
exec suma 1, 2, @err out  
print 'sss' + str(@err)
```

The screenshot shows a SQL IDE window with a yellow vertical bar on the left. The code in the editor is: `declare @err int`, `exec suma 1, 2, @err out`, and `print 'sss' + str(@err)`. Below the editor, there is a 'Messages' pane showing the output 'sss' and a status bar showing '3'.

# Procedures

```
ALTER PROCEDURE nombre (parámetros) AS
```

```
Begin
```

```
...
```

```
end;
```

```
DROP PROCEDURE nombre
```

# Functions

CREATE FUNCTION nombre (parámetros) RETURNS (tipo de dato) AS

Begin

...

return valor

end;

ALTER FUNCTION nombre (parámetros) RETURNS (tipo de dato)  
AS

Begin

...

return valor

end;

DROP FUNCTION nombre

# Functions

```
ALTER FUNCTION suma (@aa int, @bb int) returns int AS
```

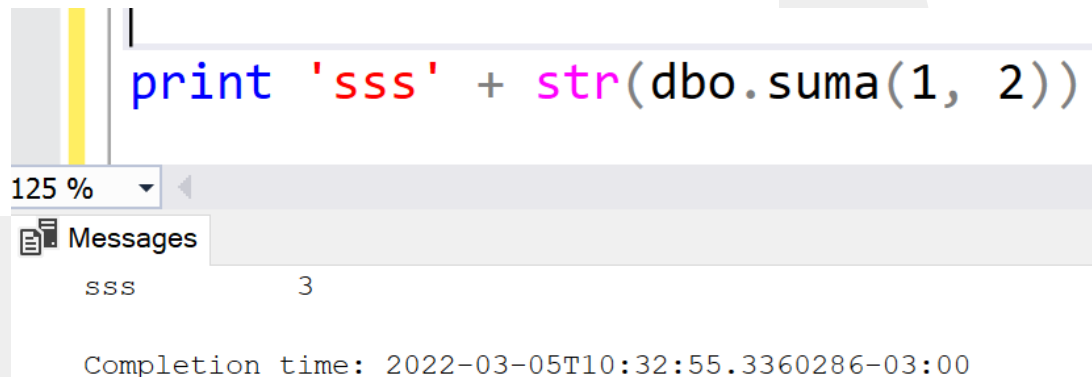
```
Begin
```

```
    declare @cc int
```

```
    set @cc = @aa + @bb
```

```
    return @cc
```

```
end
```



The screenshot shows a SQL Server query editor window. The query is: `print 'sss' + str(dbo.suma(1, 2))`. The results pane shows the output: `sss` and `3`. The completion time is `2022-03-05T10:32:55.3360286-03:00`.

```
print 'sss' + str(dbo.suma(1, 2))
```

125 %

Messages

sss 3

Completion time: 2022-03-05T10:32:55.3360286-03:00

Notas: Se pueden ejecutar Funciones desde una sentencia SQL.

No puede incluir sentencias de actualización (Sql Server).

# TSql - Variables

```
DECLARE @variable [tipo]
```

```
SET @variable = valor
```

```
SELECT @variable = valor
```

```
SELECT @variable = columna
```

```
FROM tabla
```

```
WHERE condicion
```

## **Tipos de variables (algunos)**

Int

Smallint

Decimal(m, n)

Varchar(n)

Char(n)

Date

Datetime



# TSql - Operadores



+, -, \*, /

+=, -=, \*=, /=

```
SQLQuery1.sql - L...RU31H6\Herni (52))* ✕  
declare @aa int  
set @aa =10  
set @aa /= 5  
print @aa  
110 %  
Messages  
2  
Completion time: 2022-04-05T15:09:29.9466084-03:00
```

# TSql – Sentencias de control

**IF** *condición*

**BEGIN**

*sentencias TSql*

**END**

**[ELSE]**

**BEGIN**

*sentencias TSql*

**END**

Operadores de condición

=, <>, !=, <=, >=, <, >

# TSql – Sentencias de control

```
IF @precio < 1000
```

```
BEGIN
```

```
    @precio = @precio * 1.1
```

```
END
```

```
ELSE
```

```
BEGIN
```

```
    @precio *= 1.05
```

```
END
```

# TSql – Sentencias de control

IF **EXISTS** (select 1 from pedidos where producto = 1010)

BEGIN

*print 'Producto 1010 tiene pedidos'*

END

ELSE

BEGIN

*print 'Producto SIN pedidos'*

END

# TSql – Sentencias de control

```
declare @dia varchar(15), @numero int
set @numero = 3
SET @dia = CASE @numero when 1 then 'Lunes'
                    when 2 then 'Martes'
                    when 3 then 'Miercoles'
                    when 4 then 'Jueves'
                    when 5 then 'Viernes'
                    when 6 then 'Sabado'
                    when 7 then 'Domingo'
                    ELSE 'Invalido'
END
print @dia
```

El **CASE** tiene dos formatos

# TSql – Sentencias de control

“

```
SET @valor = CASE when @valor < 1000 then 'Barato'  
                  when @valor >= 1000 and @valor < 8000 then 'Medio'  
                  when @valor >= 8000 then 'Caro'  
END
```

# TSql – Sentencias de control

**WHILE** *condicion*

**Begin**

...

**[BREAK/CONTINUE]**

...

**End**

**Se ejecuta mientras la condición sea verdadera.**

**BREAK:** Corta la ejecución.

**CONTINUE:** Salta a evaluar la condición.

# TSql – Asignación de columnas

```
declare @fabricante_cod varchar(5), @fabricante_nom varchar(20)
```

```
declare @tiempo_entrega smallint, @provincia_cod char(2)
```

```
select @fabricante_cod = fabricante_cod, @fabricante_nom = fabricante_nom,  
       @tiempo_entrega = tiempo_entrega, @provincia_cod = provincia_cod  
from fabricantes where fabricante_cod = 'CASA'
```

```
print 'Fabricante:' + @fabricante_cod + ' ' + @fabricante_nom;
```

```
print 'T.Entrega:' + cast(@tiempo_entrega as varchar);
```

```
print 'Provincia:' + @provincia_cod
```



# TSql – Cursores



- Lee un conjunto de filas y las recorre en forma secuencial
- Se utilizan para realizar procesos sobre las filas leídas.
- Debe ser declarado como tipo.

```
DECLARE cursor_name CURSOR [ LOCAL | GLOBAL ]  
[ FORWARD_ONLY | SCROLL ]  
[ STATIC | KEYSET | DYNAMIC | FAST_FORWARD ]  
[ READ_ONLY | SCROLL_LOCKS | OPTIMISTIC ]  
[ TYPE_WARNING ]  
FOR select_statement  
[ FOR UPDATE [ OF column_name [ ,...n ] ] ]
```

# TSql – Cursores

- Estructura de uso



```
DECLARE cursor_name CURSOR FOR  
    Sentencia select ...;
```

```
OPEN cursor_name
```

```
FETCH cursor_name INTO lista_variables
```

```
WHILE (@@FETCH_STATUS = 0)  
BEGIN
```

```
...
```

```
FETCH cursor_name INTO lista_variables  
END
```

```
CLOSE cursor_name  
DEALLOCATE cursor_name
```

# TSql – Cursores

```
DECLARE @nroCuenta int, @monto decimal(12,2)
DECLARE moviCur CURSOR FOR
    Select nroCuenta, monto from movimientos;

begin
OPEN moviCur
FETCH moviCur INTO @nroCuenta, @monto

WHILE (@@FETCH_STATUS = 0)
BEGIN
    UPDATE saldos SET saldo = saldo + @monto
    WHERE cuenta = @nroCuenta;

    FETCH moviCur INTO @nroCuenta, @monto
END

CLOSE moviCur
DEALLOCATE moviCur
```

# TSql – Transacciones

“

BEGIN TRANSACTION: Marca el comienzo de la transacción

COMMIT: Confirma todas las operaciones.

ROLLBACK: Vuelve atrás todas las operaciones

# TSql – Excepciones

“

```
BEGIN TRY
```

```
...
```

```
-- Ante un error en este bloque la ejecución sigue
```

```
-- en el bloque CATCH
```

```
...
```

```
END TRY
```

```
BEGIN CATCH
```

```
-- Sentencias para el manejo de la excepción
```

```
...
```

```
END CATCH
```

# TSql – Errores

**RAISERROR('Message', Severity, State) ó**

**THROW idMessage, 'Message', State**

'Message': Cadena de mensaje de error

Severity: Severidad del error. Normalmente 16.

idMessage: Número de error (debe ser mayor a 50000)

State: Valor entero de referencia (entre 0 y 255)

RAISERROR estuvo disponible a partir de la versión 2007, mientras que THROW apareció en la versión 2012 y es la que Microsoft recomienda utilizar en los desarrollos nuevos.

# TSql – Errores

-- Ejemplo de RAISERROR

begin

begin try

print 'Entra al try'

raiserror('Error en el try', 16, 1)

print 'Esto no se ejecuta nunca'

end try

begin catch

print 'Entró al catch';

print 'Nro. Error:' + cast(ERROR\_NUMBER() as varchar);

print 'mensaje:' + ERROR\_MESSAGE();

print 'State:' + cast(ERROR\_STATE() as varchar);

raiserror('Error en el catch', 16, 1);

print 'Despues del Raiserror';

end catch

print 'Despues del CATCH'

end

Entra al try

Entró al catch

Nro. Error:50000

mensaje:Error en el try

State:1

Msg 50000, Level 16, State 1, Line 12

Error en el catch

Despues del Raiserror

Despues del CATCH

# TSql – Errores

## -- Ejemplo de THROW

begin

begin try

print 'Entra al try';

throw 50000, 'Disparó el THROW en el TRY', 1

print 'Esto no se ejecuta nunca'

end try

begin catch

print 'Entró al catch';

print 'Nro. Error:' + cast(ERROR\_NUMBER() as varchar);

print 'mensaje:' + ERROR\_MESSAGE();

print 'State:' + cast(ERROR\_STATE() as varchar);

throw 50000, 'Disparó el THROW en el CATCH', 1

print 'Esto no se ejecuta nunca'

end catch

print 'Esto no se ejecuta nunca'

end

Entra al try

Entró al catch

Nro. Error:50000

mensaje:Disparó el THROW en el TRY

State:1

Msg 50000, Level 16, State 1, Line 12

Disparó el THROW en el CATCH





# Ejercicios