**TSQL Procedural**

T-SQL dispune de un limbaj procedural ca o extensie a limbajului neprocedural SQL (Structured Query Language), care constă din declarații și elemente de limbaj care pot fi utilizate pentru a pune în aplicare logica procedurală a aplicațiilor. Instrucțiunile pot fi executate independent sau pot fi incluse în definițiile unor obiecte ale bazei de date, cum ar fi proceduri, funcții, triggere.

Instructiunea PRINT

Se folosește pentru afișarea mesajelor

Sintaxa

Print *mesaj*

Declararea variabilelor utilizator

O variabilă este o zonă de memorie caracterizată printr-un tip și un nume, și permite stocarea unei valori de tipul respectiv. In SQL Server, variabilele trebuie declarate înainte de utilizare. La declarare variabilele primesc valoarea NULL. Pentru a atribui o valoare unei variabile se poate folosi instrucțiunea SELECT sau instrucțiunea SET. Numele variabilelor trebuie să înceapă cu @ și se definesc cu ajutorul instrucțiunii DECLARE conform următoarelor exemple:

1. Declararea unei variabile de tip int

declare @x int

2) Declararea mai multor variabile în cadrul instrucțiunii *declare*

declare @a int, @b int, @c varchar(20)

3) Declararea cu inițializare a variabilelor

declare @pi numeric(3,2)=3.14

print @pi

4) Declararea variabilelor și setarea lor cu valori

- Exemplu de utilizare a instrucțiunii set:

declare @vocale varchar(5)

set @vocale='aeiou'

print @vocale

- Exemplu de utilizare a instrucțiunii select:

declare @catitate int, @pret numeric(5,2)

select @catitate=10, @pret=20.5

print 'cantitate='+str(@catitate)+' pret='+str(@pret,7,2)

1. Declararea variabilelor și utilizarea lor în comenzile select și print

declare @nrClienti int, @nrFacturi int

select @nrClienti=count(distinct codClient),

@nrFacturi=count(nrFact)

from tFacturi

print 'nr clienti=' + str(@nrClienti)+' nrFacturi='+ str(@nrFacturi)

print 'media facturilor pe client='+ str(1.0\*@nrFacturi/@nrClienti,7,2)

declare @codClient char(10)

set @codClient=

( select top 1 codClient

from tFacturi

group by codClient

order by count(nrFact) desc

)

print 'Clientul cu cele mai multe facturi='+@codClient

**Operatori si expresii**

Operatorii sunt simboluri care specifică operaţiile ce se aplică unor variabile sau constante numite operanzi.

O expresie este o construcţie aritmetică sau algebrică care defineşte un calcul prin aplicarea unor operatori asupra unor termeni care pot fi: constante, variabile, funcţii.

Expresiile se evaluează pe baza unui set de reguli care precizează prioritatea şi modul de asociere a operatorilor precum şi conversiile aplicate operanzilor

Dacă o expresie folosește mai mulți operatori, ordinea în care sunt efectuate operațiile sunt determinate prioritatea operatorilor. Tabelul următor prezintă operatorii pe niveluri de precedență. Un operator de pe un nivel mai ridicat este evaluat înaintea unui operator de pe un nivel inferior. Dacă doi operatori au același nivel de prioritate, atunci ei vor fi evaluați de la stânga la dreapta în funcție de poziția lor în expresie.

|  |  |
| --- | --- |
| **Nivel** | **Operatori** |
|  | ( ) |
|  | ~ (Negație, la nivel de bit), + (pozitiv), - (negativ), |
|  | \* (Înmulțire), / (Impărțire), % ( Restul impărțirii) |
|  | + (adunare), + (concatenare),  -(Scădere), & ( ȘI la nivel de bit), ^ ( SAU exclusiv la nivel de bit), | (SAU la nivel de bit) |
|  | Operatori de comparare  =(egal), >(mai mare), <(mai mic), >=(mai mare sau egal), <=(mai mic sau egal), <>(diferit), != (diferit), !> (mai mic sau egal),!< (mai mare sau egal) |
|  | NOT |
|  | AND |
|  | OR, ALL, ANY, BETWEEN, IN, LIKE, SOME |
|  | Operatori de atribuire  =, +=, -=, \*=, /=, %=, &=, ^=, |= |

DECLARE @m Int

SET @m = 2 \* 4 + 5

SELECT @m --produce rezultatul 13.

Operatorul paranteză rotundă „( )” se utilizează pentru a impune o altă ordine în efectuarea operaţiilor. O expresie inclusă între paranteze rotunde formează un operand.

Parantezele rotunde se utilizează şi la apelul funcţiilor.

DECLARE @m Int

SET @m = 2\*(4+3\*(5-3))

SELECT @m --rezultat expresie este 20.

Operatori la nivel de bit

declare @x smallint

set @x=75

set @x=~@x

select @x -- afișează -76

Reprezentarea binară a lui 75 este de 0000 0000 0100 1011 . Efectuarea operației ~( NOT la nivel de bit) produce 1111 1111 1011 0100 , care în zecimal este -76.

declare @x smallint,@m smallint

set @x=75

set @m=0xFFF0

set @x=@x&@m

select @x --afișează 64

Reprezentarea binară a lui 75 este de 0000 0000 0100 1011  și a lui @m este 1111 1111 1111 0000 . Efectuarea operatiei & (SI la nivel de bit) produce  0000 0000 0100 0000 , care în zecimal este 64.

declare @x smallint,@m smallint

set @x=75

set @m=0x000f

set @x=@x|@m

select @x --afișează 79

Reprezentarea binară a lui 75 este de 0000 0000 0100 1011  și a lui @m este 0000 0000 0000 1111 . Efectuarea operatiei | (SAU la nivel de bit) produce  0000 0000 0100 1111 , care în zecimal este 79.

Operatori de atribuire

declare @d int

set @d=1

set @d+=10

select @d -- afisează valoarea 11

set @d\*=5

select @d -- afisează valoarea 55

Instructiunea compusă

Instrucţiunea compusă este o succesiune de instrucţiuni incluse între Begin si End, succesiune care eventual poate conţine și declaraţii. Sintaxa:

BEGIN

…

END

Exemplu:

begin

declare @x int

set @x=10

print @x

end

Instructiunea if

Instrucţiunea if implementează structura alternativă.

Sintaxa:

IF expresieLogica

    instructiune

[ ELSE

instructiune ]

Exemple:

declare @y int

set @y=5

if @y<7

begin

declare @x int

set @x=10

print @x+1

end

declare @s int, @p int=2000, @c int=10, @r int, @n int

set @s=2500

if @s<=@p

begin

set @r=0

set @n=@s

end

else

begin

set @r=(@s-@p)\*@c/100

set @n=@s-@r

end

print 's='+str(@s)+' r='+str(@r)+' n='+ str(@n)

**Instructiunea while**

Instrucţiunea while implementează structura repetitivă cu test iniţial şi are sintaxa:

WHILE expresieLogica

         instructiune

Exemplu:

declare @s int,@i int

set @S=0;

set @i=0;

WHILE @S<=100

begin

set @i=@i+1;

set @S=@S+@i;

end

print 's='+str(@s)+' i='+str(@i)

**Instructiunea BREAK**

Sintaxa:

BREAK

Realizeaza ieşirea forţată din instrucţiunea repetitivă while

Exemplu:

declare @s int,@i int

set @S=0;

set @i=0;

WHILE 1=1 --ciclu infinit

begin

set @i=@i+1;

set @S=@S+@i;

if @S>100 break;

end

print 's='+str(@s)+' i='+str(@i)

**Instructiunea continue**

Sintaxa:

CONTINUE

Realizează saltul la evaluarea expresiei care decide asupra continuării ciclului while. Secventa de instructiuni ce urmează după cuvântul cheie CONTINUE este ignorată.

declare @s int,@i int

set @S=1;

set @i=2;

WHILE 1=1 --ciclu infinit

begin

set @i=@i+1;

if @s%@i=0

begin

print 's='+str(@s)+' i='+str(@i)

continue

end

set @S=@S+@i;

if @S>100 break;

end

print 's='+str(@s)+' i='+str(@i)

**Instructiunea GOTO**

Prin etichetă înţelegem un nume urmat de două puncte (:)

<*nume*>:

Instrucţiunea goto are formatul

goto <*nume*>;

Ea realizează saltul la instrucţiunea prefixată de <*nume*>:

Exemplu:

declare @s int,@i int

set @S=0

set @i=0

inceput:

set @i=@i+1

set @S=@S+@i

if @S<=100 goto inceput

print 's='+str(@s)+' i='+str(@i)

**Controlul excepțiilor**

TRY...CATCH

Tratarea excepțiilor in Transact-SQL se poate realiza cu ajutorul blocurilor TRY...CATCH, intr-un mod similar similară cu gestionarea erorilor din Microsoft Visual C # șau Java. Un grup de instrucțiuni Transact-SQL poate fi inclus într-un bloc TRY. Dacă apare o eroare în blocul TRY, controlul este transmis grupului de instructiuni inclus în blocul CATCH asociat.

Sintaxa:

BEGIN TRY

 instrucțiune Transact-SQL

END TRY

BEGIN CATCH

  instrucțiune Transact-SQL

END CATCH [ ; ]

Exemple:

declare @a decimal, @b decimal, @c decimal

set @a=10

set @b=0

BEGIN TRY

SET @c = @a / @b;

END TRY

BEGIN CATCH

SELECT ERROR\_NUMBER() AS CodEroare,

ERROR\_LINE() AS LinieEroare,

ERROR\_MESSAGE() AS MesajEroare;

END CATCH

Obtinem:

CodEroare LinieEroare MesajEroare

8134 5 Divide by zero error encountered.

begin TRY

/\* instructiuni SQL \*/

RAISERROR('Eroare:Am lansat o eroare',11,2) -- lansam o eroare

/\* instructiuni SQL \*/

End TRY

Begin Catch

print 'err'+ str(@@error)

print ' Cod Eroare:'+str(ERROR\_NUMBER()) +' Linie Eroare:' +

str(ERROR\_LINE())+

' MesajEroare:'+ERROR\_MESSAGE();

end catch

In pagina Messages obținem:

err 50000

Cod Eroare: 50000 Linie Eroare: 3 MesajEroare: Eroare:Am lansat o eroare

Variabila globala @@ERROR returnează 0 dacă instrucțiunea Transact-SQL anterioară nu a întâmpinat erori, respectiv un număr de eroare dacă dacă instrucțiunea Transact-SQL anterioară a generat o eroare.

Deoarece @@ ERROR este resetată înaintea fiecărei instrucțiuni executate, ea trebuie verificată imediat, eventual o putem salva într-o variabilă locală care poate fi verificată ulterior, ca in exempul urmator:

begin TRY

/\* instructiuni SQL \*/

RAISERROR('Eroare:Am lansat o eroare',11,2) -- lansam o eroare

/\* instructiuni SQL \*/

End TRY

Begin Catch

declare @error int

set @error=@@error

print 'err'+ str(@error)

print 'Cod Eroare:'+str(ERROR\_NUMBER()) +' Linie Eroare:' +

str(ERROR\_LINE())+ ' MesajEroare:'+ERROR\_MESSAGE();

print 'err'+ str(@@error)+' -se observa resetarea variabilei globale @@error'

print 'err'+ str(@error) +' - am afisat valoarea lui @@error memorata in var. locala @error'

end catch

In pagina Messages obținem:

err 50000

Cod Eroare: 50000 Linie Eroare: 3 MesajEroare: Eroare:Am lansat o eroare

err 0 -se observa resetarea variabilei globale @@error

err 50000 - am afisat valoarea lui @@error memorata in var. locala @error