
Extensões SQL para programação

Procedimentos Armazenados

Gatilhos

Funções

Sistemas de Informação II (ISEL-ADEETC Walter Vieira)

1

1

A norma ISO SQL 2016

Com a introdução dos PSM (persistent stored modules), a norma também introduziu mecanismos típicos das linguagens de programação tradicionais:

Declaração de variáveis:

```
<SQL variable declaration> ::=  
    DECLARE <SQL variable name list>  
    <data type> [ <default clause> ]  
  
<SQL variable name list> ::=  
    <SQL variable name> [ { <comma> <SQL variable name> }... ]  
  
<data type> ::=  
    <predefined type> | <row type> | <user-defined type>  
    | <reference type> | <collection type>  
  
<default clause> ::= DEFAULT <default option>  
  
<default option> ::= <literal> | <datetime value function> | USER | ...
```

Exemplo: DECLARE v1 integer default 101;

Sistemas de Informação II (ISEL-ADEETC Walter Vieira)

2

2

A norma ISO SQL 2016

Instruções de controlo de fluxo:

```
<SQL procedure statement> ::=
    <SQL executable statement>

<SQL executable statement> ::=
    <SQL schema statement> | <SQL data statement>
    | <SQL control statement> | <SQL transaction statement>
    | <SQL connection statement> | <SQL session statement>
    | <SQL diagnostics statement> | <SQL dynamic statement>

<SQL control statement> ::=
    <call statement> | <return statement>
    | <assignment statement> | <compound statement>
    | <case statement> | <if statement>
    | <iterate statement> | <leave statement>
    | <loop statement> | <while statement>
    | <repeat statement> | <for statement> | ...
```

3

A norma ISO SQL 2016

<call statement> ::= CALL <routine invocation>

<return statement> ::= RETURN <return value>

<assignment statement> ::=

```
    <singleton variable assignment>
    | SET <assignment target> <equals operator> <assignment source>
```

Exemplo: SET v1 = 444

<compound statement> ::=

```
    [ <beginning label> <colon> ]
    BEGIN [ [ NOT ] ATOMIC ]
    [ <local declaration list> ]
    [ <local cursor declaration list> ]
    [ <local handler declaration list> ]
    [ <SQL statement list> ]
    END [ <ending label> ]
```

4

A norma ISO SQL 2016

<while statement> ::=
[<beginning label> <colon>]
WHILE <search condition> DO
<SQL statement list>
END WHILE [<ending label>]

Exemplo:
L1:
WHILE v1 < 1000 DO
...
SET v1 = v1+1
END WHILE L1

<for statement> ::=
[<beginning label> <colon>]
FOR <for loop variable name>
AS [<cursor name> [<cursor sensitivity>] CURSOR FOR]
<cursor specification>
DO <SQL statement list>
END FOR [<ending label>]

Variável do tipo ROW que em cada posição do cursor conterá as colunas associadas

<for loop variable name> ::= <identifier>

...

A norma ISO SQL 2016

<if statement> ::=
IF <search condition>
<if statement then clause>
[<if statement elseif clause>...]
[<if statement else clause>]
END IF

<if statement then clause> ::= THEN <SQL statement list>

**<if statement elseif clause> ::= ELSEIF <search condition>
THEN <SQL statement list>**

<if statement else clause> ::= ELSE <SQL statement list>

Exemplo:
IF v1 = 200 THEN
...
ELSE
...
END IF

A norma ISO SQL 2016

<select statement single row> ::=
SELECT | **<set quantifier>** | **<select list>**
INTO **<select target list>**
<table expression>

<select target list> ::=
<target specification> | { **<comma>** **<target specification>** }... |

<target specification> ::=
<host parameter specification>
| **<SQL parameter reference>**
| **<column reference>**
| ...
| **<SQL variable reference>**

Exemplo:

```
SELECT c1, c2 INTO v1, v2  
FROM T WHERE c3 = 333
```

A norma ISO SQL 2016 – Exemplo MySql

```
create procedure p(px integer)  
begin  
  declare i int default 0;  
  declare x int default px;  
  Lw:  
  while x > 0 do  
    set x = x-1;  
    set i = i+1;  
  end while Lw;  
  select j into x from t where t.i = i;  
  select * from t where t.i = x;  
end  
...  
  
call p(2);
```

T-SQL – Declaração e utilização de variáveis

```
DECLARE @vi int,  
        @vc varchar(10)
```

```
SET @vi = 12  
print @vi
```

```
SELECT @vi = 20, @vc = 'xxxx'  
print @vi  
print @vc
```

```
SELECT @vi = Bi, @vc = NomeAl FROM Aluno  
print @vi  
print @vc
```

Só para afectação. Não produz “result set”

```
UPDATE Aluno SET NomeAl = cast(@vi as varchar) WHERE Bi = @vi
```

T_SQL – Variáveis globais (scalar functions)

São variáveis pré-definidas pelo SQL Server 2005, que não podem ser alteradas, nem necessitam de declaração.

Algumas delas são:

@@IDENTITY – contém o último valor atribuído a uma das colunas identity

@@ERROR – contém o valor 0 se a última instrução executada decorreu com sucesso, ou, caso contrário, um número de erro que identifica o tipo de erro

@@ROWCOUNT – contém o número total de linhas afectadas ou devolvidas pela última instrução executada

T-SQL – Variáveis

T-SQL e conversões de tipos

To:	binary	varbinary	char	nchar	varchar	nvarchar	datetime	smalldatetime	decimal	numeric	float	real	bigint	int(INT4)	smallint(INT2)	tinyint(INT1)	money	smallmoney	bit	timestamp	uniqueidentifier	image	ntext	text	sql_variant
From:	binary	●																							
varbinary	●	●																							
char	●		●	●																					
varchar	●		●	●	●																				
nchar	●		●	●		●																			
nvarchar	●		●	●	●	●																			
datetime	●						●	●																	
smalldatetime	●						●	●																	
decimal	●								●	●															
numeric	●								●	●															
float	●									●	●														
real	●									●	●														
bigint	●											●	●												
int(INT4)	●											●	●												
smallint(INT2)	●											●	●												
tinyint(INT1)	●											●	●												
money	●													●	●										
smallmoney	●													●	●										
bit	●															●	●								
timestamp	●																●	●							
uniqueidentifier	●																		●	●					
image	●																			●	●				
ntext	●																				●	●			
text	●																					●	●		
sql_variant	●																						●	●	
xml	●																							●	●

● Explicit conversion (usar CAST ou CONVERT)
○ Implicit conversion
○ Conversion not allowed

* Requires explicit CAST to prevent the loss of precision or scale that might occur in an implicit conversion

● Implicit conversions between XML datatypes are supported only if the source or target is untyped xml. Otherwise, it has to be explicit.

Sistemas de Informação II (ISEL-ADEETC Walter Vieira)

11

11

T-SQL – Variáveis tabela

DECLARE @tabLocal TABLE (i int primary key, j varchar(10))

insert into @tabLocal values(1, 'uuuu')

select * from @tabLocal

Notar que as tabelas variável só têm existência no batch onde são declaradas, pelo que não devem ser confundidas com tabelas temporárias (com nomes iniciados por #), as quais têm existência durante a ligação corrente.

As tabelas temporárias cujo nome se inicia com um único # dizem-se locais e são vistas apenas na ligação que as criou, sendo eliminadas quando a ligação se quebrar; As cujo nome se inicia com ## dizem-se globais e podem ser vistas por outras ligações, sendo eliminadas apenas quando todas as ligações que as usam forem quebradas

Sistemas de Informação II (ISEL-ADEETC Walter Vieira)

12

12

T-SQL – Instruções de controlo de fluxo

Decisão simples:

```
declare @v bit
set @v = 0

IF @v = 1
BEGIN
    print 'verdade'
    print 'verdade outra vez'
END
ELSE
    print 'mentira'

IF not exists (select * from Alunos where nAl = 6666)
    print 'aluno não existe'

IF 3333 in (select nAl from Alunos) print 'aluno não existe'

IF (select nomeAl from Alunos where nAl = 3333) is null print 'nulo'

IF (select saldo from contas where numero = 1000) > 10000
    print 'está rico'
```

Sistemas de Informação II (ISEL-ADEETC Walter Vieira)

13

13

T-SQL – Instruções de controlo de fluxo

Repetição:

```
declare @v int
set @v = 1

WHILE @v <= 10
BEGIN
    print @v
    set @v = @v+1
    -- BREAK
END
```

Go to:

```
declare @v int
set @v = 1

I1:
IF @v <= 10
BEGIN
    print @v
    set @v = @v+1
    GOTO I1
END
```

Sistemas de Informação II (ISEL-ADEETC Walter Vieira)

14

14

Cursores (norma ISO SQL 2016)

**<declare cursor> ::= DECLARE <cursor name> <cursor properties>
FOR <cursor specification>**

<cursor properties> ::=
[<cursor sensitivity>] [<cursor scrollability>] CURSOR
[<cursor holdability>]
[<cursor returnability>]

<cursor sensitivity> ::= SENSITIVE | INSENSITIVE | **ASENSITIVE**

<cursor scrollability> ::= SCROLL | NO SCROLL

<cursor holdability> ::= WITH HOLD | WITHOUT HOLD

<cursor returnability> ::= WITH RETURN | WITHOUT RETURN

<cursor specification> ::= <query expression> [<updatability clause>]

<updatability clause> ::=
FOR { READ ONLY | UPDATE [OF <column name list>] }

Exemplo: DECLARE curs CURSOR FOR SELECT i,j FROM T WHERE i > -333 FOR READ ONLY;

Sistemas de Informação II (ISEL-ADEETC Walter Vieira)

15

15

Cursores (norma ISO SQL 2016)

<fetch statement> ::=
FETCH [[<fetch orientation>] FROM]
<cursor name> INTO <fetch target list>

<fetch orientation> ::=
NEXT
| PRIOR
| FIRST
| LAST
| { ABSOLUTE | RELATIVE } <simple value specification>

<fetch target list> ::=
<target specification> [{ <comma> <target specification> }...]

<open statement> ::= OPEN <cursor name>

<close statement> ::= CLOSE <cursor name>

Exemplo: FETCH FROM curs INTO v1, v2

Sistemas de Informação II (ISEL-ADEETC Walter Vieira)

16

16

Cursores (norma ISO SQL 2016)

<delete statement: positioned> ::=
DELETE FROM <target table> [[AS] <correlation name>]
WHERE CURRENT OF <cursor name>

<update statement: positioned> ::=
UPDATE <target table> [[AS] <correlation name>]
SET <set clause list>
WHERE CURRENT OF <cursor name>

<target table> ::=
<table name>
| ONLY <left paren> <table name> <right paren>

Exemplo: DELETE FROM T WHERE CURRENT OF Curs

Sistemas de Informação II (ISEL-ADEETC Walter Vieira)

17

17

Cursores (norma ISO SQL 2016)

Exemplo :

```
--#SET TERMINATOR @ -----> Para uso com o IBM Data Studio
BEGIN
  DECLARE v1, v2 integer;
  DECLARE SQLSTATE char(5);
  DECLARE curs CURSOR FOR SELECT i, j FROM T
                        WHERE i > -333 FOR READ ONLY;

  OPEN curs;
  FETCH FROM curs INTO v1, v2;
  WHILE(SQLSTATE = '00000') DO
    -- processar v1 e v2;
    if v1 = v2 then
      insert into t1 values(v1,v2);
    end if;
    FETCH FROM curs INTO v1, v2;
  END WHILE;
  CLOSE curs;
END @
```

Este exemplo foi testado com IBM DB2 Express 11.1

Sistemas de Informação II (ISEL-ADEETC Walter Vieira)

18

18

Cursores (norma ISO SQL 2016)

Exemplo :

```
create procedure p1(x integer)
L1: begin
  declare v1,v2 integer;
  declare done integer default FALSE;
  declare curs cursor for select i,j from t where i = x;
  /*DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;*/
  /* ou: */
  DECLARE CONTINUE HANDLER FOR sqlstate '02000' SET done = TRUE;
  open curs;
  FETCH curs INTO v1, v2;
  WHILE not done DO
    -- processar v1 e v2
    if v2 = v1 then
      insert into t1 values(v1,v2);
    end if;
    FETCH curs INTO v1, v2;
  END WHILE;
  close curs;
end l1;
```

Este exemplo foi testado com MySql 5.7

Sistemas de Informação II (ISEL-ADEETC Walter Vieira)

19

19

Cursores (norma ISO SQL 2016)

Exemplo:

```
--#SET TERMINATOR @
BEGIN
  DECLARE v1, v2 integer;

  FOR linha AS curs CURSOR FOR
    SELECT i,j FROM T WHERE i > 333 FOR READ ONLY
  DO
    SET v1 = linha.i;
    SET v2 = linha.j;
    -- processar v1 e v2
    if v1 = v2 then
      insert into t1 values(v1,v2);
    end if;
  END FOR;
END @
```

Este exemplo foi testado com IBM DB2 Express 11.1

Sistemas de Informação II (ISEL-ADEETC Walter Vieira)

20

20

T-SQL – Cursores

À la ISO SQL:

```
DECLARE cursor_name [ INSENSITIVE ] [ SCROLL ] CURSOR
FOR select_statement
[ FOR { READ ONLY | UPDATE [ OF column_name [ ,...n ] ] } ]
```

Extensão SQL Server:

```
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 ] ] ]
```

T-SQL – Cursores

```
DECLARE c1 CURSOR FOR SELECT * FROM T
```

```
OPEN c1
```

```
FETCH NEXT FROM c1
```

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

```
    FETCH NEXT FROM c1
```

```
CLOSE c1
```

```
DEALLOCATE c1
```

```
FETCH
[ [ NEXT | PRIOR | FIRST | LAST
  | ABSOLUTE { n | @nvar }
  | RELATIVE { n | @nvar }
]
FROM
[
  { [ GLOBAL | cursor_name ] | @cursor_variable_name }
  [ INTO @variable_name [ ,...n ] ]
]
```

T-SQL – Cursores

UPDATE

```
[...]  
SET { column_name = { expression | DEFAULT | NULL } | ... } [ ,...n ]  
[ <OUTPUT Clause> ]  
[ FROM { <table_source> } [ ,...n ] ]  
[ WHERE { <search_condition>  
    | {[CURRENT OF {[GLOBAL] cursor_name} | cursor_variable_name}}  
    }  
]  
[ OPTION ( <query_hint> [ ,...n ] ) ]
```

23

T-SQL – Cursores

DELETE

```
[ TOP ( expression ) | PERCENT ] ]  
[ FROM ]  
{ <object> | rowset_function_limited  
  [ WITH ( <table_hint_limited> [ ...n ] ) ]  
}  
[ <OUTPUT Clause> ]  
[ FROM <table_source> [ ,...n ] ]  
[ WHERE { <search_condition>  
    | {[CURRENT OF  
        { {[GLOBAL] cursor_name }  
        | cursor_variable_name  
        }  
    ]  
    }  
]  
[ OPTION ( <Query Hint> [ ,...n ] ) ]
```

24

T-SQL – Cursores

Exemplo:

```
declare curs cursor for select c1, c2 from T where c3 = 333
                        for update
open curs
declare @v1 int, @v2 int
FETCH NEXT FROM curs into @v1,@v2;
WHILE (@@FETCH_STATUS = 0)
BEGIN
    -- processar @v1 e @v2
    -- ou, por exemplo, fazer uma actualização
    if @v1 = 5
        update t set c1 = @v1 + 10 where current of curs
    FETCH NEXT FROM curs into @v1,@v2
END;
CLOSE curs
DEALLOCATE curs
```

25

Cursores e ResultSet em JDBC

```
Statement statement = con.createStatement(ResultSet.TYPE_FORWARD_ONLY,
                                           ResultSet.CONCUR_UPDATABLE);
ResultSet rs = statement.executeQuery("SELECT BI, NomeAl FROM Aluno");

while ( rs.next() ) // FETCH NEXT ...
{
    BigDecimal bi = rs.getBigDecimal( "BI" );
    String nome = rs.getString( "NOMEAL" );
    if (....) {
        rs.updateBigDecimal(1,new BigDecimal(9999998));
        rs.updateString(2,"novo nome");
        rs.updateRow(); // update Aluno set nome = 'novo nome' where current of ...
    }
}

...
```

26

Cursores e ResultSet em JDBC

```
Statement statement = con.createStatement(ResultSet.TYPE_FORWARD_ONLY,  
                                           ResultSet.CONCUR_UPDATABLE);  
ResultSet rs = statement.executeQuery("SELECT BI, NomeAl FROM Aluno");  
  
while ( rs.next( ) ) // FETCH NEXT ...  
{ BigDecimal bi = rs.getBigDecimal( "BI" );  
  String nome = rs.getString( "NOMEAL" );  
  if (....) {  
    rs.deleteRow(); // DELETE FROM Alunos where current of ...  
  }  
}  
...  
...
```

27

Cursores e ResultSet em JDBC

Com rs é possível realizar instruções como as seguintes:

```
rs.absolute(2000); // FETCH ABSOLUTE ...  
rs.last();        // FETCH LAST ...  
rs.first();       // FETCH FIRST ...  
rs.relative(-200); // FETCH RELATIVE ...  
rs.previous();    // FETCH PRIOR ...
```

28

Exceções ou condition handling (norma ISO SQL 2016)

<handler declaration> ::=
DECLARE <handler type> HANDLER
FOR <condition value list>
<handler action>

<handler type> ::= CONTINUE | EXIT | UNDO^(*)

<handler action> ::= <SQL procedure statement>

<condition value list> ::= <condition value> [{ <comma> <condition value> }...]

<condition value> ::=
<sqlstate value> | <condition name> | SQLEXCEPTION | SQLWARNING
| NOT FOUND

<condition declaration> ::=
DECLARE <condition name> CONDITION
[FOR <sqlstate value>]

<compound statement> ::=
[<beginning label> <colon>]
BEGIN [| NOT | ATOMIC]
...
[<local handler declaration list>]
código executado sob controlo dos
handlers
END [<ending label>]

^(*) Exige ATOMIC

Sistemas de Informação II (ISEL-ADEETC Walter Vieira)

29

29

Exceções ou condition handling (norma ISO SQL 2016)

Exemplo: --#SET TERMINATOR @

```
begin
DECLARE v1 integer;
DECLARE cont BOOLEAN DEFAULT TRUE;
DECLARE CondFim CONDITION FOR SQLSTATE '02000';
DECLARE curs CURSOR FOR SELECT i FROM T WHERE j > -333 FOR
                                READ ONLY;

BEGIN NOT ATOMIC
  DECLARE CONTINUE HANDLER FOR CondFim
    SET cont = FALSE;
  OPEN curs;
  FETCH curs INTO v1;
  WHILE cont = TRUE DO
    -- tratar v1
    set v1 = v1*2;
    insert into t1 values(v1,v1+1);
    FETCH curs INTO v1;
  END WHILE;
  CLOSE curs;
END;
```

End @

Este exemplo foi testado com IBM DB2 Express 11.1

Sistemas de Informação II (ISEL-ADEETC Walter Vieira)

30

30

Exceções ou condition handling (norma ISO SQL 2016)

Ideia do funcionamento do <for statement> (só por curiosidade): →

```

BL: BEGIN NOT ATOMIC
FLVN: BEGIN NOT ATOMIC
DECLARE CN CS CURSOR FOR
SELECT ROW ( Q.V1, Q.V2, ..., Q.Vn )
FROM ( FCS ) AS Q
DECLARE FLVN ROW ( V1 DT1, V2 DT2, ..., Vn DTn );

DECLARE AT_END BOOLEAN DEFAULT FALSE;
DECLARE NOT_FOUND CONDITION FOR SQLSTATE '02000';
BEGIN NOT ATOMIC
    DECLARE CONTINUE HANDLER FOR NOT_FOUND
        SET AT_END = TRUE;
    OPEN CN;
    FETCH CN INTO FLVN;
    WHILE NOT AT_END DO
        <instruções>
        BEGIN NOT ATOMIC
            FETCH CN INTO FLVN;
        END;
    END WHILE;
    CLOSE CN;
END;
END FLVN;
END BL
    
```

Adaptado de ANSI/ISO/IEC International Standard (IS) Database Language SQL—Part 4: Persistent Stored Modules (SQL/PSM) (ISO/IEC 9075-4:1999)

Sistemas de Informação II (ISEL-ADEETC Walter Vieira)

31

31

Exceções ou condition handling (norma ISO SQL 2016)

<signal statement> ::=
 SIGNAL <signal value>
 [<set signal information>]

<signal value> ::=
 <condition name>
 | <sqlstate value>

<resignal statement> ::=
 RESIGNAL [<signal value>]
 [<set signal information>]

<set signal information> ::=
 SET <signal information item list>

<signal information item list> ::=
 <signal information item> [{ <comma> <signal information item> }...]

<signal information item> ::=
 <condition information item name> <equals operator>
 <simple value specification>

MESSAGE_TEXT, CURSOR_NAME, SERVER_NAME, TABLE_NAME, ...

A norma prevê a existência de diagnostics áreas nas quais podem ser empilhadas várias informações de erro (conditions) correspondentes à sequência de SIGNAL, RESIGNAL, ... ativa RESIGNAL apenas pode ser usado no código dos handlers. SIGNAL inicia diagnostic area

Sistemas de Informação II (ISEL-ADEETC Walter Vieira)

32

32

Exceções ou condition handling (norma ISO SQL 2016)

```
CREATE PROCEDURE AddOrderItem( in orderNo int,in productCode varchar(45),
                               in qty int, in price double, in lineNo int )
BEGIN
    DECLARE C INT;

    SELECT COUNT(orderNumber) INTO C
    FROM orders
    WHERE orderNumber = orderNo;

    -- check if orderNumber exists
    IF(C != 1) THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Order No not found in orders table';
    END IF;
    -- more code below
    -- ...
END
```

Retirado de: <http://www.mysqltutorial.org/mysql-signal-resignal/>

Sistemas de Informação II (ISEL-ADEETC Walter Vieira)

33

33

Exceções (T-SQL)

```
BEGIN TRY
    { sql_statement | statement_block }
    -- estas instruções correm sob protecção do handler
END TRY
BEGIN CATCH
    { sql_statement | statement_block }
END CATCH
```

- Erros com gravidade ≤ 10 são considerados avisos e não são apanhados em blocos try catch
- Alguns erros com valor ≥ 20 (erros fatais) quebram a ligação e, portanto, também não são apanhados

Sistemas de Informação II (ISEL-ADEETC Walter Vieira)

34

34

Exceções (T-SQL)

```
RAISERROR ( {msg_id | msg_str | local_variable} {, severity, state }  
[ , argument [ ,...n ] ] )  
[ WITH option [ ,...n ] ]
```

RAISERROR ('Erro 1',16,1)

O utilizador só pode indicar valores de gravidade <= 19

Ou, mais elaborado:

```
exec sp_addmessage 50005, 16, 'erro na base de dados %s', @replace = 'replace'  
...  
declare @dbname varchar(128)  
set @dbname = db_name()  
rollback  
raiserror(50005,16,1,@dbname) with log  
print @@error
```

erro na base de dados si_wv
50005

Sistemas de Informação II (ISEL-ADEETC Walter Vieira)

35

35

Exceções (T-SQL)

```
BEGIN TRY  
    DECLARE @DBID INT;  
    SET @DBID = DB_ID();  
    DECLARE @DBNAME sysname --NVARCHAR(128);  
    SET @DBNAME = DB_NAME();  
    RAISERROR (N'The current database ID is:%d, the database name is: %s.', 18,  
        1, @DBID,@DBNAME)  
END TRY  
BEGIN CATCH  
    SELECT  
        ERROR_NUMBER() AS ErrorNumber, ERROR_SEVERITY() AS ErrorSeverity,  
        ERROR_STATE() as ErrorState, ERROR_PROCEDURE() as ErrorProcedure,  
        ERROR_LINE() as ErrorLine, ERROR_MESSAGE() as ErrorMessage;  
    --RAISERROR(N'xxx',18,1)  
END CATCH
```

Sistemas de Informação II (ISEL-ADEETC Walter Vieira)

36

36

Excepções (T-SQL)

```
DECLARE @nTent INT
SET @nTent = 3
WHILE (@nTent > 0)
BEGIN
```

```
    BEGIN TRY
        BEGIN TRANSACTION
        print 'transacção 1 iniciada'
        UPDATE conta set saldo = saldo + 500 where id = 1111
        WAITFOR DELAY '00:00:13'
        UPDATE conta set saldo = saldo + 500 where id = 2222
        SET @nTent = 0
        COMMIT
        print 'transacção 1 terminada com sucesso'
    END TRY
```

```
    BEGIN CATCH
```

```
        IF (ERROR_NUMBER() = 1205) -- deadlock victim
```

```
        begin
```

```
            SET @nTent = @nTent - 1
```

```
            print 'transacção 1 vítima de deadlock'
```

```
        end
```

```
    ELSE
```

```
        SET @nTent = -1;
```

```
    IF XACT_STATE() <> 0 ROLLBACK TRANSACTION -- transacções que seriam abortadas
```

```
    END CATCH
```

```
END
```

Transacção 1

Só para atrasar a trans. de forma a arrancarmos com a outra

Estado transaccional da sessão:

0 – não existe transacção activa

1 – existe transacção activa sem restrições

-1 – existe uma transacção *uncommittable*.

Apenas são possíveis leituras seguidas de um rollback

-- tornam-se *uncommittable* dentro de begin try

Sistemas de Informação II (ISEL-ADEETC Walter Vieira)

37

37

Excepções (T-SQL)

```
DECLARE @nTent INT
SET @nTent = 3
WHILE (@nTent > 0)
BEGIN
```

```
    BEGIN TRY
```

```
        BEGIN TRANSACTION
```

```
        print 'transacção 2 iniciada'
```

```
        UPDATE conta set saldo = saldo + 1000 where id = 2222
```

```
        WAITFOR DELAY '00:00:13'
```

```
        UPDATE conta set saldo = saldo + 1000 where id = 1111
```

```
        SET @nTent = 0
```

```
        print 'transacção 2 terminada com sucesso'
```

```
        COMMIT
```

```
    END TRY
```

```
    BEGIN CATCH
```

```
        IF (ERROR_NUMBER() = 1205) -- deadlock victim
```

```
        begin
```

```
            SET @nTent = @nTent - 1
```

```
            print 'transacção 2 vítima de deadlock'
```

```
        end
```

```
    ELSE
```

```
        SET @nTent = -1;
```

```
    IF XACT_STATE() <> 0 ROLLBACK TRANSACTION
```

```
    END CATCH
```

```
END
```

Transacção 2

Sistemas de Informação II (ISEL-ADEETC Walter Vieira)

38

38

Exceções (T-SQL)

No Sql Server 2012:

```
THROW [ { error_number | @local_variable },  
        { message | @local_variable },  
        { state | @local_variable } ] [ ; ]
```

Notas:

- Usado sem parâmetros dentro de blocos catch para enviar a exceção corrente para o nível acima;
- Contrariamente a RAISERROR, error_number não tem de existir em sys.messages
- Não são permitidos “placeholders” do tipo printf em message (em alternativa, deve usar-se a função FORMATMESSAGE)
- A gravidade é sempre 16

Exceções (T-SQL)

Exemplo:

```
EXEC sys.sp_addmessage  
    @msgnum = 60000  
    ,@severity = 16  
    ,@msgtext = N'This is a test message with one numeric parameter (%d), one  
string parameter (%s), and another string parameter (%s).'
```

```
    ,@lang = 'us_english';  
GO  
  
DECLARE @msg NVARCHAR(2048) = FORMATMESSAGE(60000, 500, N'First  
string', N'second string');  
  
THROW 60000, @msg, 1;
```

Msg 60000, Level 16, State 1, Line 2

This is a test message with one numeric parameter (500), one string parameter (First string), and another string parameter (second string).

Exceções (T-SQL)

Exemplo:

```
BEGIN TRY  
  print 1/0;
```

```
END TRY  
BEGIN CATCH  
  -- tratar exceção neste nível
```

```
  THROW; -- enviar a mesma  
         -- exceção para  
         -- nível superior
```

```
END CATCH
```

Em vez de:

```
BEGIN CATCH  
  -- tratar exceção neste nível  
  DECLARE  
    @m NVARCHAR(2048), @sv INT, @st INT  
  
  SELECT  
    @m = ERROR_MESSAGE(),  
    @sv = ERROR_SEVERITY(),  
    @st = ERROR_STATE()  
  
  RAISERROR (@m, @sv, @st )  
END CATCH
```

Exceções (T-SQL)

Exemplo (diferenças):

Com raiserror o n.º de linha associado à mensagem é o do local onde o último RAISERROR é executado e não aquele onde o erro ocorre.

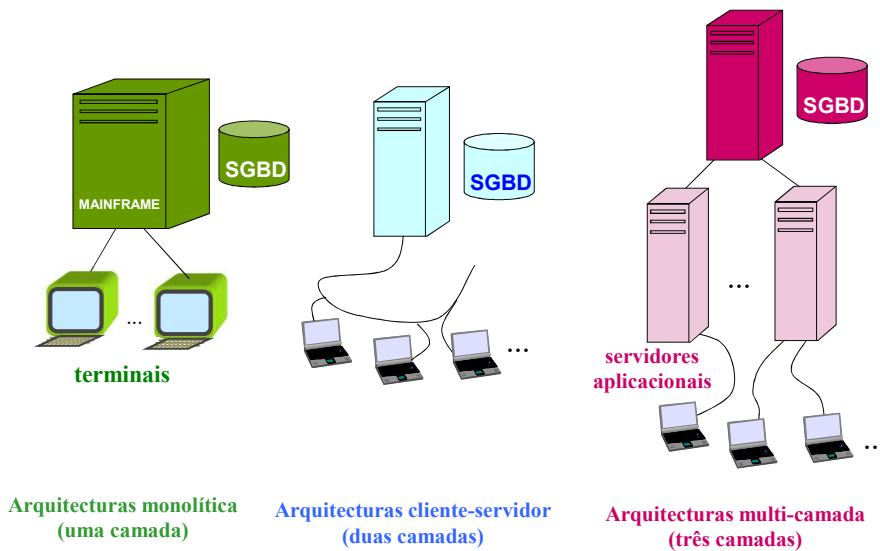
Igualmente, o n.º de erro com RAISERRER é 50000 e não o gerado originalmente (8134).

Throw não permite parametrizar a mensagen

THROW não permite o uso de números de erro abaixo de 50000

RAISERROR não permite o uso de números de erro que não estejam definidos em sys.messages

Arquitecturas de Sistemas de Informação



Sistemas de Informação II (ISEL-ADEETC Walter Vieira)

43

43

Arquitecturas de Sistemas de Informação

Arquitecturas de duas camadas:

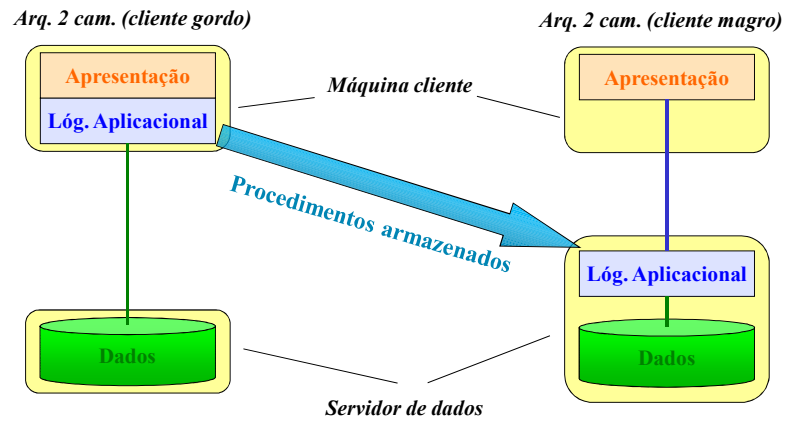
1. **Cliente gordo:**
 - Tiram partido da capacidade de processamento das máquinas cliente (+)
 - Dificultam manutenção (-)
 - Expõem a lógica de negócio aos clientes (-)
 - Podem originar sobrecargas de tráfego na rede (-)
2. **Cliente magro:**
 - Lógica de negócio transportada para o servidor de dados pode sobrecarregar o servidor (-)
 - Permitem ocultar a lógica de negócio (+)
 - Permitem reduzir o tráfego na rede (+)
 - Facilitam a manutenção (+)

Sistemas de Informação II (ISEL-ADEETC Walter Vieira)

44

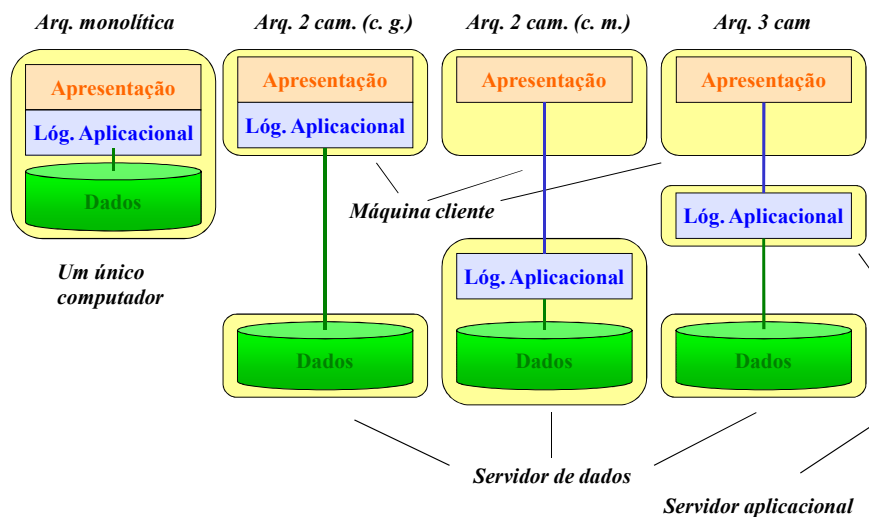
44

Arquitecturas de Sistemas de Informação (2 camadas)



45

Arquitecturas de Sistemas de Informação



46

Procedimentos armazenados

Em geral, não se devem usar os procedimentos armazenados para a implementação da lógica aplicacional (isto é, de processos de negócio).

A sua utilização deve ser restringida a aspectos que tenham a ver com a natureza intrínseca dos dados, como, por exemplo, validação de regras de negócio associados a restrições de integridade dos dados, não suportadas directamente pelo SGBD.

A estrutura e organização dos dados numa organização tendem a ser mais estáveis do que os processos de negócio. Para além disso, muitas vezes, os mesmos dados são usados em processos de negócio bastante distintos, associados, por exemplo, às várias áreas departamentais das empresas.

Naturalmente, não devemos ser fundamentalistas. Em algumas situações, a utilização de procedimentos armazenados para implementar lógica de negócio pode ser a melhor solução. Mas essas situações devem ser objecto de análise rigorosa, de forma a que a opção por procedimentos armazenados possa ser devidamente justificada.

Sistemas de Informação II (ISEL-ADEETC Walter Vieira)

47

47

Procedimentos armazenados (norma ISO SQL 2016)

<SQL-invoked routine> ::= <schema routine>

<schema routine> ::= <schema procedure> | <schema function>

<schema procedure> ::= CREATE <SQL-invoked procedure>

<schema function> ::= CREATE <SQL-invoked function>

<SQL-invoked procedure> ::=
 PROCEDURE <schema qualified routine name>
 <SQL parameter declaration list>
 <routine characteristics>
 <routine body>

<SQL parameter declaration list> ::=
 <left paren>
 [<SQL parameter declaration> [{ <comma>
 <SQL parameter declaration> }...]]
 <right paren>

Sistemas de Informação II (ISEL-ADEETC Walter Vieira)

48

48

Procedimentos armazenados (norma ISO SQL 2016)

<SQL parameter declaration> ::=
| <parameter mode> | | <SQL parameter name> |
<parameter type> | RESULT |
| DEFAULT <parameter default> |

<parameter mode> ::=
IN
| OUT
| INOUT

<parameter type> ::=
<data type> | <locator indication> |

<routine body> ::=
<SQL routine spec>
| <external body reference>

Procedimentos armazenados (norma ISO SQL 2016)

<routine characteristic> ::=
<language clause>
| <parameter style clause>
| SPECIFIC <specific name>
| <deterministic characteristic>
| <SQL-data access indication>
| <null-call clause>
| <returned result sets characteristic>
| <savepoint level indication>

<SQL routine spec> ::=
| <rights clause> | <SQL routine body>

<SQL routine body> ::= <SQL procedure statement>

<SQL procedure statement> ::= <SQL executable statement> (ver slides anteriores)

Procedimentos armazenados (norma ISO SQL 2016)

Exemplo:

```
--#SET TERMINATOR @
CREATE PROCEDURE AcimaDaMedia()
LANGUAGE SQL
READS SQL DATA → routine characteristics
DYNAMIC RESULT SETS 1
BEGIN
    DECLARE m REAL;
    DECLARE C1 CURSOR WITH RETURN FOR SELECT i FROM t WHERE i > m;
    set m = (SELECT AVG(j) as med FROM t);
    OPEN C1;

END @
```

Chamada:

```
--#SET TERMINATOR ;
CALL AcimaDaMedia();
```

Este exemplo foi testado com IBM DB2 Express 11.1

Sistemas de Informação II (ISEL-ADEETC Walter Vieira)

51

51

Procedimentos armazenados (norma ISO SQL 2016)

Exemplo:

Mas para se navegar no *result set* tem de se fazer:

```
--#SET TERMINATOR @
begin
    declare v real;
    declare sqlstate char(5);
    DECLARE result RESULT_SET_LOCATOR VARYING;
    delete from t1;
    call AcimaDaMedia();
    ASSOCIATE RESULT SET LOCATOR (result) WITH PROCEDURE
                                                AcimaDaMedia;

    ALLOCATE cur CURSOR FOR RESULT SET result;
    FETCH cur into v;
    while sqlstate = '00000' do
        insert into t1 values(v,v+1);
        FETCH cur INTO v;
    end while;
    close cur;

End @
```

Este exemplo foi testado com IBM DB2 Express 11.1

Sistemas de Informação II (ISEL-ADEETC Walter Vieira)

52

52

Procedimentos armazenados (norma ISO SQL 2016)

Exemplo:

```
--#SET TERMINATOR @  
CREATE PROCEDURE SaldoConta (IN Nc INTEGER, OUT Saldo REAL)  
LANGUAGE SQL  
READS SQL DATA
```

routine characteristics

```
P1: BEGIN  
    SELECT Contas.Saldo INTO Saldo FROM Contas  
    WHERE Contas.NumConta = Nc;  
END P1 @
```

Chamada: --#SET TERMINATOR @
 begin
 declare s real;
 call SaldoConta(2222,s);
 -- processar s
 End @

Este exemplo foi testado com IBM DB2 Express 11.1

Sistemas de Informação II (ISEL-ADEETC Walter Vieira)

53

53

Procedimentos armazenados (T-SQL)

```
CREATE { PROC | PROCEDURE }  
[schema_name.]procedure_name [ ; number ]  
[ { @parameter [ type_schema_name. ] data_type }  
  [ VARYING ] [ = default ] [ [ OUT ] PUT ]  
  ] [ ,...n ]  
[ WITH <procedure_option> [ ,...n ]  
[ FOR REPLICATION ]  
AS { <sql_statement> [ ; ] [ ...n ] | <method_specifier> }
```

RETURN [*integer_expression*]

```
[ { EXEC | EXECUTE } ]  
{  
  [ @return_status = ]  
  { module_name [ ;number ] | @module_name_var }  
  [ [ @parameter = { value | @variable [ OUTPUT ] [ DEFAULT ] } ]  
  ]  
  [ ,...n ]  
  [ WITH RECOMPILE ]  
}
```

Sistemas de Informação II (ISEL-ADEETC Walter Vieira)

54

54

Procedimentos armazenados (T-SQL)

Exemplo:

```
create procedure p @p1 int = 33 , @p2 int = 59
as
    print @p1
    print @p2
```

```
exec p 1
    1
    59
exec p @p1=default, @p2 = 100
    33
    100
exec p 1,7
    1
    7
exec p default, 3900
    33
    3900
exec p @p2 = 88
    33
    88
```

Sistemas de Informação II (ISEL-ADEETC Walter Vieira)

55

55

Procedimentos armazenados (T-SQL)

Exemplo de execução

```
create procedure tCurs @c cursor varying output
as
    set @c = cursor for SELECT * FROM t
    open @c

/* chamar o Stored Procedure e navegar no cursor que ele abre */
declare @x cursor, @v real
exec tCurs @x output
fetch next from @x into @v
while(@@FETCH_STATUS = 0)
begin
    print @v
    fetch next from @x into @v
end
close @x
deallocate @x
```

Sistemas de Informação II (ISEL-ADEETC Walter Vieira)

56

56

Procedimentos armazenados (T-SQL)

Ver também
SET NOCOUNT ON/OFF

Este select destina-se apenas a afectar a variável @nome.

Não contribui com um result set.

Result set 1

id	saldo
1111	1000
2222	2000

Result set 2

nAl	nomeAl
2222	Ze
3333	Ana
4444	Pedro
5555	Rita

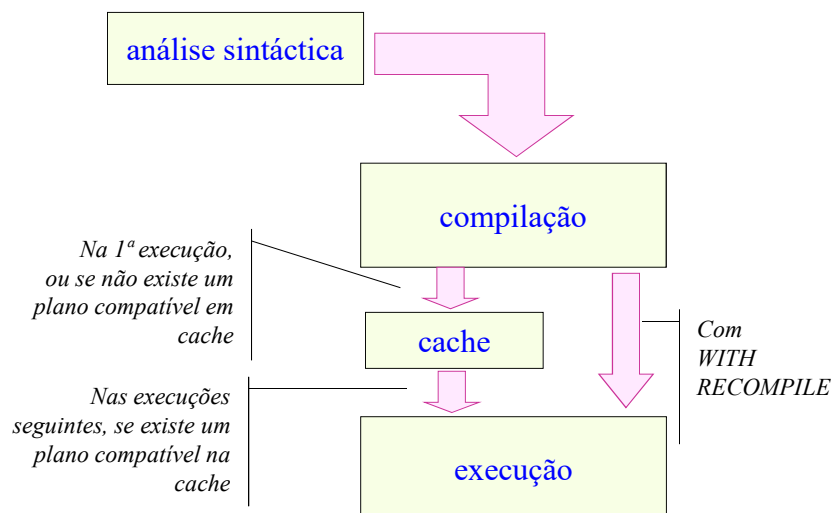
Query execute... sisad_mobile\instancia1 (9.0 RTM) se (53) SL_2 00:00:00 6 rows
Ready Ln 136 Col 1 Ch 1

Sistemas de Informação II (ISEL-ADEETC Walter Vieira)

57

57

Procedimentos armazenados (T-SQL)



Sistemas de Informação II (ISEL-ADEETC Walter Vieira)

58

58

Procedimentos armazenados – principais vantagens

- **maior desempenho**

dados processados no local onde estão armazenados permitem tirar partido das características do SGBD; diminuição do tráfego na rede. Mas, se exagerarmos, também podemos sobrecarregar o SGBD com processamentos não muito fortemente ligados aos dados

- **bom encapsulamento**

A utilização de procedimentos evita que se acedam directamente aos dados, contribuindo para uma maior consistência dos mesmos

- **maiores níveis de reutilização:**

Procedimentos podem ser partilhados por várias aplicações

- **maior segurança:**

Utilizadores podem ter permissão para executar um procedimento e não para aceder directamente aos dados; lógica de negócio oculta para os clientes.

Procedimentos armazenados – Exercício proposto

Considere a existência da tabela criada com a seguinte instrução:

`create table tabela (id int identity primary key, valor real)`

Pretende-se construir o procedimento armazenado **mediaNemN** que recebe dois parâmetros: no primeiro é indicado um passo (n) e no segundo é devolvido o valor da média das colunas de nome valor da tabela tabela, calculada considerando apenas os elementos de n em n.

Não se pode admitir que, por ser identity, a coluna id tenha sempre valores consecutivos a partir de 1, pois podem ser apagados registos

Procedimentos armazenados – Exercício proposto

```
create proc MediaNemN(@passo int, @med float output)
as
    declare @total real
    declare @v real
    declare @nElem int
    declare @conta int
    set @total = 0
    set @nElem = 0
    set @conta = 0
    declare c insensitive cursor for select valor from tabela for read only
    SET TRANSACTION ISOLATION LEVEL ?
    begin tran
    open c
```

61

Procedimentos armazenados – Exercício proposto

```
FETCH NEXT FROM c INTO @v
WHILE @@FETCH_STATUS = 0
BEGIN
    if @conta = 0
    begin
        set @total = @total + @v
        set @nElem = @nElem + 1
    end
    set @conta = (@conta + 1) % @passo
    FETCH NEXT FROM c INTO @v
end
commit
if @nElem = 0
    set @med = null
else
    set @med = @total/@nElem
close c
deallocate c
```

62

Procedimentos armazenados – Exercício proposto

```
create proc MediaNemN_v2(@passo int, @med float output)
as
declare @totalreal
declare @v real
declare @nElem int
set @total= 0
set @nElem = 0
declare c insensitive scroll cursor for select valor from tabela for read only
SET TRANSACTION ISOLATION LEVEL ?
begin tran
open c
FETCH NEXT FROM c INTO @v
WHILE @@FETCH_STATUS = 0
BEGIN
    set @total = @total + @v
    set @nElem = @nElem+1
    FETCH RELATIVE @passo FROM c INTO @v
end
commit
if @nElem = 0 set @med = null else set @med = @total/@nElem
close c
deallocate c
```

Sistemas de Informação II (ISEL-ADEETC Walter Vieira)

63

63

Procedimentos armazenados – Exercício proposto

O cursor deve ser sensitive ou insensitive?

O cursor poderá ser insensitive, dado que assim calculamos a média apenas com os registos existentes na altura em que se abre o cursor. No entanto, para result sets de grandes dimensões isto pode ser proibitivo, dado que os dados têm de ser copiados para uma zona que só é usada pelo cursor. Assim, nestas situações, o cursor deverá ser sensitive.

É necessária uma transação? Com que nível de isolamento?

Com um cursor insensitive não é necessária uma transação explícita (com open é criada a cópia dos dados). O nível de isolamento deverá ser READ COMMITTED.

Com um cursor sensitive, é necessária uma transação explícita. Nesta situação, se pretendermos que os registos acedidos não possam ser alterados até ao fim do cálculo, devemos subir o nível de isolamento para REPEATABLE READ. Se quisermos garantir que os registos a ter em conta no cálculo são os e apenas os existentes no momento do select com os valores dessa altura, devemos optar por SERIALIZABLE (assim estaremos a imprimir ao cursor sensitive um comportamento similar ao conseguido com um cursor insensitive).

Sistemas de Informação II (ISEL-ADEETC Walter Vieira)

64

64

Procedimentos armazenados – Exercício proposto

Mas, com o Sql Server, pode ser assim:

```
create proc MediaNemN_v3(@passo int, @med float output)
as
begin
    select @med = avg(valor) from
        (SELECT ROW_NUMBER() OVER(ORDER BY id ASC) AS NumReg,
            valor
        FROM tabela) as t
    WHERE (NumReg-1)%@passo = 0
end
```

A coluna NumReg tem valores consecutivos (a partir de 1)

Funções (norma ISO SQL 2016)

```
<schema routine> ::=
    <schema procedure>
    | <schema function>

<schema function> ::= CREATE <SQL-invoked function>

<SQL-invoked function> ::=
    { <function specification> | <method specification designator> }
    <routine body>

<function specification> ::=
    FUNCTION <schema qualified routine name>
    <SQL parameter declaration list>
    <returns clause>
    <routine characteristics>
    | <dispatch clause> |

<returns clause> ::= RETURNS <returns type>

<returns type> ::= <returns data type> | <result cast> | <returns table type>

<returns table type> ::= TABLE <table function column list>
```

Idem proc. armazen.

Funções (norma ISO SQL 2016)

```
<routine characteristics> ::= [ <routine characteristic>... ]
```

```
<routine characteristic> ::=
  <language clause>
  | <parameter style clause>
  | SPECIFIC <specific name>
  | <deterministic characteristic>
  | <SQL-data access indication>
  | <null-call clause>
  | <returned result sets characteristic>
  | <savepoint level indication>
```

<deterministic characteristic> ::= DETERMINISTIC | NOT DETERMINISTIC

**<SQL-data access indication> ::= NO SQL | CONTAINS SQL
| READS SQL DATA | MODIFIES SQL DATA**

Funções (norma ISO SQL 2016)

```
CREATE FUNCTION saldoMedio ()
RETURNS REAL
NOT DETERMINISTIC
RETURN ( SELECT AVG(saldo) FROM Contas ) ;
```

Uso: select * from contas where saldo < saldoMedio() ;

```
CREATE FUNCTION funcTabela()
RETURNS TABLE (i INT, j VARCHAR(10))
READS SQL DATA
RETURN SELECT i,j FROM t WHERE i > -10 ;
```

Uso: select * from TABLE(funcTabela());

Estes exemplos foram testados com IBM DB2 Express 11.1

Funções (T-SQL)

Funções escalares:

```
CREATE FUNCTION [ schema_name. ] function_name
( [ { @parameter_name [ AS ] [ type_schema_name. ] parameter_data_type
  [ = default ] }
  [ ,...n ]
]
)
RETURNS return_data_type
[ WITH <function_option> [ ,...n ] ]
[ AS ]
BEGIN
    function_body
    RETURN scalar_expression
END
```

Funções (T-SQL)

Funções escalares:

```
create function dobro(@n int) returns int
as
begin
    return @n*2;
end
```

...

```
print dbo.dobro(4)
```

```
create table tabela2 (
    id int identity primary key,
    valor real,
    dobro as dbo.dobro(valor)
)
```

Nomes de funções escalares têm de incluir sempre o schema

Funções (T-SQL)

Funções que retornam tabelas (inline):

```
CREATE FUNCTION [ schema_name. ] function_name  
( [ { @parameter_name [ AS ] [ type_schema_name. ] parameter_data_type  
  [ = default ] }  
  [ ,...n ]  
]  
)  
RETURNS TABLE  
  [ WITH <function_option> [ ,...n ] ]  
  [ AS ]  
RETURN [ ( [ select_stmt ] ) ]
```

```
CREATE FUNCTION funcTabelas()  
RETURNS TABLE  
AS  
RETURN (SELECT nAl, upper(nomeAl) as nome from alunos)
```

```
select * from funcTabelas()
```

Sistemas de Informação II (ISEL-ADEETC Walter Vieira)

71

71

Funções (T-SQL)

Funções que retornam tabelas – uso correlato:

```
CREATE FUNCTION funcTabelas1(@nAl int)  
RETURNS TABLE  
AS  
RETURN  
  (SELECT nAl, upper(nomeAl) as nome from alunos where nAl > @nAl)
```

```
select * from alunos where not exists (select * from funcTabelas1(alunos.nAl))
```

Sistemas de Informação II (ISEL-ADEETC Walter Vieira)

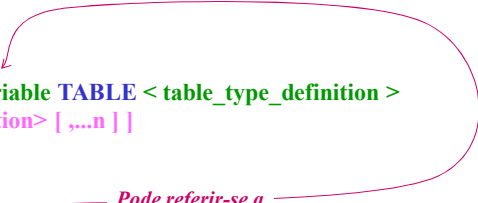
72

72

Funções (T-SQL)

Funções que retornam tabelas (multi-statement):

```
CREATE FUNCTION [ schema_name. ] function_name  
( [ { @parameter_name [ AS ] [ type_schema_name. ] parameter_data_type  
  [ = default ] }  
  [ ,...n ]  
 ]  
)  
RETURNS @return_variable TABLE < table_type_definition >  
  [ WITH <function_option> [ ,...n ] ]  
  [ AS ]  
BEGIN  
    function_body  
RETURN  
END
```



73

Funções (T-SQL)

Funções que retornam tabelas (multi-statement):

```
CREATE FUNCTION funcaoTabMultiStat()  
RETURNS @ret TABLE (i int, j char(2))  
AS  
BEGIN  
    insert into @ret values(1,'aa')  
    insert into @ret values(2,'bb')  
RETURN  
END
```

```
select * from funcaoTabMultiStat()
```

74

Funções (T-SQL)

Exemplo:

Considere-se o seguinte modelo físico:

```
create table Alunos(NumAl int primary key, nome varchar(50))
create table UnidadesCurriculares(codUC char(3) primary key,
                                   design varchar(50))
create table Inscr(NumAl int references Alunos,
                  codUC char(3) references UnidadesCurriculares,
                  Ano int, Nota integer,
                  primary key(NumAl,codUC,Ano))
```

É fácil produzir os pares (NumAl,CodUC) que não correspondem a qualquer inscrição em nenhum dos anos compreendidos entre 2009 e 2010:

```
select NumAl,CodUC FROM Alunos CROSS JOIN UnidadesCurriculares
WHERE NOT EXISTS (select Ano from Inscr as I where
                  Ano between 2009 and 2010 and
                  I.NumAl = Alunos.NumAl and
                  I.CodUC = UnidadesCurriculares.CodUC)
```

Funções (T-SQL)

Exemplo:

Mas já não é tão directo produzir os pares (CodUC, Ano) que não correspondem a qualquer inscrição em nenhum dos anos compreendidos entre 2009 e 2010.

A dificuldade surge porque não existe qualquer tabela que contenha todos os anos no período considerado, o que se pode resolver usando uma função que produza tal tabela:

```
create function AnosIntervalo(@inicio int, @fim int)
returns @r table(Ano int)
as
begin
    declare @i int = @inicio;
    while @i <= @fim
    begin
        insert into @r values(@i);
        set @i = @i+1
    end
    return;
end
```

```
select CodUC,Ano
FROM UnidadesCurriculares CROSS JOIN
(select Ano from AnosIntervalo(2009,2010)) as tAno
WHERE NOT EXISTS (select Ano from Inscr as I where
                  I.CodUC = UnidadesCurriculares.CodUC and
                  I.Ano = tAno.Ano)
```

Funções (T-SQL)

Outra solução:

```
create function ParesUcAno(@inicio int, @fim int)
returns @r table(CodUc char(3), Ano int)
as
begin
    declare @uc char(3);
    declare @i int;
    declare c cursor for select CodUC from UnidadesCurriculares;
    open c;
    fetch next from c into @uc;
    while @@FETCH_STATUS = 0
    begin
        set @i = @inicio;
        while @i <= @fim
        begin
            if not exists (select * from Inscr where CodUc = @uc and Ano = @i)
                insert into @r values(@uc,@i);
            set @i = @i+1
        end
        fetch next from c into @uc;
    end
    close c;
    deallocate c;
    return;
end
```

```
select * from ParesUcAno(2009,2010)
```

Sistemas de Informação II (ISEL-ADEETC Walter Vieira)

77

77

Funções (T-SQL)

Se usarmos duas funções, ficamos com:

```
create function ParesUcAno_v2(@inicio int, @fim int)
returns table
as
return select CodUC,Ano
FROM UnidadesCurriculares CROSS JOIN (select Ano from
AnosIntervalo(2009,2010)) as tAno
WHERE NOT EXISTS (select Ano from Inscr as I
where I.CodUC = UnidadesCurriculares.CodUC and
I.Ano = tAno.Ano)

select * from ParesUcAno_v2(2009,2010)
```

Mas a segunda função pode ser substituída por uma vista.

Sistemas de Informação II (ISEL-ADEETC Walter Vieira)

78

78

Funções (T-SQL)

Mas ainda podemos ter:

```
create function ParesUcAno_v3(@inicio int, @fim int)
returns @r table(CodUc char(3), Ano int)
as
begin
    declare @uc char(3);
    declare @i int = @inicio;
    while @i <= @fim
    begin
        insert into @r select CodUC, @i from UnidadesCurriculares where
            not exists (select * from Inscr WHERE Inscr.Ano = @i and
                        Inscr.CodUc = UnidadesCurriculares.codUC)

        set @i = @i+1
    end
    return;
end

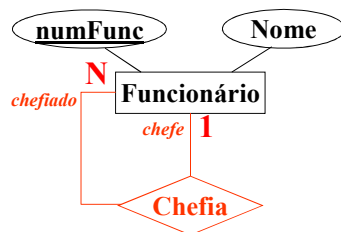
select * from ParesUcAno_v3(2009,2010)
```

Sistemas de Informação II (ISEL-ADEETC Walter Vieira)

79

79

SQL – Fecho transitivo recursivo (revisão)



Será possível obter, de uma forma uniforme, todos os subordinados directos e indirectos de um funcionário?

Recursive queries a partir do SQL 1999

No SQL Server uma query é recursiva se usar uma Recursive CTE (Common Table Expression)

Sistemas de Informação II (ISEL-ADEETC Walter Vieira)

80

80

SQL – Fecho transitivo recursivo (revisão)

Funcionario

nFunc	nome	chefe
1	NumberOne	NULL
2	NumberTwo	1
3	NumberThree	2
11	AnotherNumberOne	NULL
22	AnotherNumberTwo	11

1 NumberOne NULL 0
 2 NumberTwo 1 1
 3 NumberTree 2 2

```
WITH CTE_Chefes (nFunc, nome, chefe, nivel)
AS
(
  -- Âncora (termina a recursão)
  SELECT nFunc, nome, chefe, 0 as nivel
  FROM Funcionario
  WHERE chefe IS NULL and nFunc = 1
  UNION ALL
  -- Recorrência
  SELECT F.nFunc, F.nome, F.chefe,
         nivel + 1
  FROM Funcionario as F
  INNER JOIN CTE_Chefes AS C
  ON F.chefe = C.nFunc
)
-- usar a CTE
SELECT nFunc, nome, chefe, nivel
FROM CTE_Chefes
```

81

SQL – Fecho transitivo recursivo (revisão)

Funcionario

nFunc	nome	chefe
1	NumberOne	NULL
2	NumberTwo	1
3	NumberThree	2
4	AnotherNumberThree	2

1- Seja T_0 o resultado da execução da âncora

T0

nFunc	nome	chefe	nivel
1	NumberOne	NULL	0

2- Seja T_{i+1} o resultado da execução da query de recorrência com base no resultado T_i no lugar de CTE_Chefes. Repetir este ponto até que T_{i+1} seja vazio.

T1

nFunc	nome	chefe	nivel
2	NumberTwo	1	1

T2

nFunc	nome	chefe	nivel
3	NumberThree	2	2
4	AnotherNumberThree	2	2

82

SQL – Fecho transitivo recursivo (revisão)

3- Retornar o resultado de UNION ALL de todos os resultados parciais T0 a Tn., onde Tn é o último resultado parcial obtido.

nFunc	nome	chefe	nivel
1	NumberOne	NULL	0
2	NumberTwo	1	1
3	NumberThree	2	2
4	AnotherNumberThree	2	2

83

SQL – CTE (outro exemplo de recursive query)

Função que devolve tabela com os pares (N,N!) para N desde 0 até a um máximo passado como parâmetro

```
create function factorial(@max int)
returns @r table(n int, fact int)
as
begin
    with cte_fact (n, fact)
    as
    (
        select 0,1
        union all
        select n+1, (n+1)*fact from cte_fact where n < @max
    )
    insert into @r select n, fact from cte_fact
    return
end
```

84

SQL – CTE (outro exemplo de *recursive query*)

Com o código:

```
create view factorialv(n,fact)
as
  with cte_fact (n,fact)
  as
  (
    select 0,1
    union all
    select n+1,(n+1)*fact from cte_fact
  ) select n, fact from cte_fact

select * from factorialv where n < 2
```

Obtemos os seguintes resultados:

n	fact
0	1
1	1

E a seguir a exceção:

Msg 8115, Level 16, State 2, Line 59
Arithmetic overflow error converting expression to data type int.

Mas, se acrescentarmos **where n < 4**
já não aparecerá a exceção.
PORQUÊ?

85

SQL – Outros usos das CTE (revisão)

```
WITH Aux(med)
AS
(
  SELECT AVG(Nota) as med from Inscricao
)
SELECT BI,CodDisc,Ano,Nota FROM Inscricao
where Nota > (SELECT med FROM Aux)
```

86

Funções (T-SQL)

Solução do exemplo do slide 76 com CTEs:

```
create function AnosIntervalo(@inicio int, @fim int)
returns @r table(ano int)
as
begin
    with anos(a)
    as
    (
        select @inicio
        union all
        select a+1 from anos where a < @fim
    )
    insert into @r select * from anos
    return
end
```

```
select CodUC,Ano
FROM UnidadesCurriculares CROSS JOIN
(select Ano from AnosIntervalo(2009,2010)) as tAno
WHERE NOT EXISTS (select Ano from Inscr as I where
    I.CodUC = UnidadesCurriculares.CodUC and
    I.Ano = tAno.Ano)
```

87

Funções (T-SQL)

Mas com CTEs não necessitamos de uma função:

```
with anosIntervaloCTE(ano)
as
(
    select 2009
    union all
    select ano+1 from anosIntervaloCTE where ano < 2010
)
select CodUC,Ano
FROM UnidadesCurriculares CROSS JOIN
(select Ano from anosIntervaloCTE) as tAno
WHERE NOT EXISTS (select Ano from Inscr as I where
    I.CodUC = UnidadesCurriculares.CodUC and
    I.Ano = tAno.Ano)
```

88

Funções (T-SQL)

Algumas limitações das funções definidas pelo utilizador:

- Não podem realizar ações que modifiquem o estado da base de dados (escritas, função rand, etc.)
 - Não podem produzir múltiplos result sets
 - Não podem chamar procedimentos armazenados
 - Não podem usar instruções SET (SET TRANSACTION, SET NOCOUNT, ...)
 - Não suportam TRY...CATCH, @ERROR ou RAISERROR.
- ...

Ver:

<https://docs.microsoft.com/en-us/sql/relational-databases/user-defined-functions/create-user-defined-functions-database-engine>

Sistemas de Informação II (ISEL-ADEETC Walter Vieira)

89

89

Funções (T-SQL)

create function fff() returns datetime

as

begin

declare @d datetime

select @d = getdate()

return @d

end

Command(s) completed successfully.

create function fff() returns float

as

begin

declare @r float

select @r = rand()

return @r

end

**Msg 443, Level 16, State 1, Procedure fff, Line 6 [Batch Start Line 46]
Invalid use of a side-effecting operator 'rand' within a function.**

Sistemas de Informação II (ISEL-ADEETC Walter Vieira)

90

90

SQL – Vistas (revisão)

```
<view definition> ::=  
    CREATE [ RECURSIVE ] VIEW <table name>  
    <view specification>  
    AS <query expression>  
    [ WITH [ <levels clause> ] CHECK OPTION ]  
  
<view specification> ::= <regular view specification> | <referenceable view specification>  
  
<regular view specification> ::= [ <left paren> <view column list> <right paren> ]  
  
<levels clause> ::=  
    CASCADED  
    | LOCAL
```

Exemplo:

```
CREATE VIEW Inscr  
AS  
    SELECT Aluno.Bi, Aluno.NomeAl, Disciplina.DesDisc, Inscricao.Ano  
    FROM Aluno, Disciplina, Inscricao  
    WHERE Aluno.Bi=Inscricao.Bi AND Disciplina.CodDisc = Inscricao.CodDisc
```

Sistemas de Informação II (ISEL-ADEETC Walter Vieira)

91

91

SQL – Vistas Alteráveis (revisão)

Uma vista é potencialmente alterável se:

1. Não especificar **DISTINCT**
2. Todos os itens de seleção na referida (<select list>) representam uma referência única para uma coluna da tabela base.
3. Não contiver imediatamente cláusulas **GROUP BY** ou **HAVING**
4. A cláusula **FROM** não fizer referência a mais do que uma tabela
5. A cláusula **FROM** não fizer referência a uma vista não alterável

Sistemas de Informação II (ISEL-ADEETC Walter Vieira)

92

92

SQL – Vistas Alteráveis (revisão)

A construção CHECK OPTION

Sejam as instruções

```
CREATE TABLE Alunos (Numero integer Primary key,  
                        Nome varchar(20) NOT NULL,  
                        Nota Float NOT NULL  
                        )
```

```
CREATE VIEW AlunosPassados (Num, Nome, Nota)  
AS SELECT Alunos.Numero, Alunos.Nome, Alunos.Nota  
FROM Alunos  
WHERE Alunos.Nota >= 10
```

93

SQL – Vistas Alteráveis (revisão)

O que acontece com:

```
UPDATE AlunosPassados  
SET Nota = 7 WHERE AlunosPassados.Num = 3000
```

se o aluno 3000 tiver nota 14?

O aluno desaparece da vista

E com:

```
INSERT INTO AlunosPassados VALUES (4000, 'xico', 8) ?
```

O aluno 4000 nunca pertenceu à vista

94

SQL – Vistas Alteráveis (revisão)

```
CREATE VIEW AlunosPassados (Num, Nome, Nota)
AS SELECT Alunos.Numero, Aluno.Nome, Alunos.Nota
FROM Alunos
WHERE Alunos.Nota >= 10
WITH CHECK OPTION
```

Agora as inserções e alterações anteriores seriam rejeitadas

Na norma SQL 2016, podemos ter

WITH CASCADED CHECK OPTION – as *search condition* de todas as vistas de que esta depende são testadas

Ou:

WITH LOCAL CHECK OPTION – apenas as *search condition* de vistas de que esta depende tenham **WITH (LOCAL | CASCADED) CHECK OPTION** são testadas

Gatilhos

Gatilhos (Também conhecidos por “event-condition-action rules”)

- **Evento:** *Insert, Update, Delete*
- **Condição:** *em que condições a acção do gatilho é executada*
- **Acção:** *sequência de acções SQL*

Gatilhos (norma ISO SQL 2016)

<trigger definition> ::=

```
CREATE TRIGGER <trigger name>
<trigger action time> <trigger event>
ON <table name>
[ REFERENCING <transition table or variable list> ]
<triggered action>
```

<trigger action time> ::=

```
BEFORE
| AFTER
| INSTEAD OF
```

→ *Quando é que a acção é executada relativamente ao evento*

<trigger event> ::=

```
INSERT
| DELETE
| UPDATE [ OF <trigger column list> ]
```

<trigger column list> ::= <column name list>

Gatilhos (norma ISO SQL 2016)

<triggered action> ::=

```
[ FOR EACH { ROW | STATEMENT } ]
[ <triggered when clause> ]
<triggered SQL statement>
```

gatilhos nível linha ou nível commando (por omissão, nível commando)

<triggered when clause> ::= WHEN

<left paren> <search condition> <right paren>

<triggered SQL statement> ::=

```
<SQL procedure statement>
| BEGIN ATOMIC
  { <SQL procedure statement> <semicolon> }...
END
```

Gatilhos (norma ISO SQL 2016)

<transition table or variable list> ::=
<transition table or variable>...

<transition table or variable> ::=
OLD [ROW] [AS] <old transition variable name>
| NEW [ROW] [AS] <new transition variable name>
| OLD TABLE [AS] <old transition table name>
| NEW TABLE [AS] <new transition table name>

Gatilhos (norma ISO SQL 2016)

Gatilho nível linha:

```
CREATE TRIGGER GatilhoNivelLinha  
    AFTER UPDATE OF nota ON Alunos  
    REFERENCING NEW ROW AS novo OLD ROW AS velho  
    FOR EACH ROW  
    WHEN (novo.nota > velho.nota)  
    INSERT INTO Melhorias VALUES (novo.numero)
```

Este exemplo foi testado com IBM DB2 Express 10.05

Gatilhos (norma ISO SQL 2016)

Gatilho nível comando:

```
CREATE TRIGGER GatilhoNivelComando
AFTER INSERT ON Alunos
REFERENCING NEW TABLE AS novos
FOR EACH STATEMENT
BEGIN ATOMIC
  INSERT INTO Aprovados
    (SELECT numero FROM novos WHERE nota >= 10);
END
```

Este exemplo foi testado com IBM DB2 Express 10.05

Gatilhos (norma ISO SQL 2016)

Mas o primeiro exemplo também poderia ser realizado assim:

```
CREATE or replace TRIGGER Gatilho
AFTER UPDATE OF nota ON Alunos
REFERENCING NEW TABLE AS novos OLD TABLE AS velhos
FOR EACH STATEMENT
--WHEN ((select count(*) from novos) > 0)
  INSERT INTO Melhorias (select novos.Numero from novos, velhos
    WHERE novos.Numero = velhos.Numero and
      velhos.nota < novos.nota)
```

Este exemplo foi testado com IBM DB2 Express 10.05

Mas, neste caso, o código do gatilho executa-se sempre uma vez, mesmo que não existam registos alterados (a menos que se active a condição WHEN). A solução também não funciona se houver alterações da chave e não existirem outras chaves candidatas que não sejam alteradas.

Gatilhos (T-SQL)

*não existem gatilhos
nível linha*

```
CREATE TRIGGER [schema_neme.]trigger_name
ON { table | view }
[ WITH <dml_trigger_option> [,...n]]
{FOR | AFTER | INSTEAD OF}{[INSERT][,][UPDATE][,][DELETE]}
[ WITH APPEND ]      (será abandonado no futuro)
[ NOT FOR REPLICATION ]
AS
{sql_statement [;] [ ...n ] | EXTERNAL NAME <method specifier>}
```

Nota:

FOR e AFTER têm o mesmo significado

Sistemas de Informação II (ISEL-ADEETC Walter Vieira)

103

103

Gatilhos (T-SQL)

Teste para se saber se houve inserções ou alterações:

```
{IF UPDATE ( column )
  [ {AND | OR} UPDATE ( column ) ]
  [ ...n ]
| IF ( COLUMNS_UPDATED ( ) {bitwise_operator} updated_bitmask )
  {comparison_operator} column_bitmask [ ...n ] }
```

create trigger t_tg on tg for insert, update, delete

As

```
declare @n int, @n1 int
select @n = count(*) from inserted
select @n1 = count(*) from deleted
if @n > 0 and @n1 = 0
  print 'inseridos '+CAST(@n as varchar)+' registos'
if (Update(i))
  print 'alterada coluna i'
if (Update(j))
  print 'alterada coluna j'
if @n1 > 0 and @n = 0
  print 'apagados '+CAST(@n1 as varchar)+' registos'
```

Sistemas de Informação II (ISEL-ADEETC Walter Vieira)

104

104

Gatilhos (T-SQL)

Mensagens observadas:

```
insert into tg values(1,1)
    inseridos 1 registos
    alterada coluna i
    alterada coluna j
```

```
update tg set j = j+1
    alterada coluna j
```

```
delete from tg
    apagados 1 registos
```

Sistemas de Informação II (ISEL-ADEETC Walter Vieira)

105

105

Gatilhos (T-SQL)

As tabelas virtuais INSERTED e DELETED:

Não podem ser alteradas!

comando	DELETED	INSERTED
INSERT	-	registos novos
UPDATE	valores antigos	valores novos
DELETE	registos apagados	-

```
create trigger ActualizarHistorico
on Produto
for UPDATE
as
    if update(Preco)
        INSERT INTO ProdHist(IDProd,PrecoAnt)
        SELECT IDProd,Preco FROM DELETED
```

Sistemas de Informação II (ISEL-ADEETC Walter Vieira)

106

106

Gatilhos (T-SQL)

Quando se realiza **INSERT**, **UPDATE** ou **DELETE** sobre uma vista ou tabela que possua um gatilho do tipo **INSTEAD OF**, o SGBD activa o gatilho em vez de executar a acção sobre a tabela ou vista.

Exemplo:

```
create table produto (  
  cod char(8) primary key,  
  preco decimal(6,2) not null  
)
```

```
create view prodIVA as  
  select cod, 1.19*preco as preco from produto
```

vista não alterável

```
CREATE TRIGGER InsProdIVA ON prodIVA INSTEAD OF INSERT  
AS  
  INSERT INTO produto SELECT cod, preco/1.19 FROM INSERTED
```

Sistemas de Informação II (ISEL-ADEETC Walter Vieira)

107

107

Gatilhos (T-SQL)

```
insert into produto values('p1',10.1)  
insert into produto values('p2',5.5)  
insert into prodIva values('p3',11.9)
```

Select * from produto

p1	10.10
p2	5.50
p3	10.00

select * from prodIva

p1	12.0190
p2	6.5450
p3	11.9000

Executam-se após afectadas as
tabelas virtuais INSERTED e
DELETED

Sistemas de Informação II (ISEL-ADEETC Walter Vieira)

108

108

Gatilhos (T-SQL)

Gatilhos recursivos (recursão directa):

```
create trigger t_tg1 on tg for insert
As
insert into tg values(2,2)
```

```
insert into tg values(1,1)
```

Msg 2627, Level 14, State 1, Line 1

Violation of PRIMARY KEY constraint 'PK__tg__6B24EA82'. Cannot insert duplicate key in object 'dbo.tg'.

The statement has been terminated.

```
ALTER DATABASE SI_2
SET RECURSIVE_TRIGGERS {ON | OFF}
```

Só elimina a recursão directa

ON só funciona se a opção 'nested triggers' (sp_configure) tiver o valor 1

*Com gatilhos **INSTEAD OFF** a recursão directa está sempre desactivada*

Sistemas de Informação II (ISEL-ADEETC Walter Vieira)

109

109

Gatilhos (T-SQL)

Gatilhos encaixados (recursão indirecta):

```
create table tg1(i int primary key, j int)
create table tg2(i int primary key, j int)
```

```
create trigger trig1 on tg1
for insert
AS
insert into tg2 values(2,2)
```

```
create trigger trig2 on tg2
for insert
AS
insert into tg1 values(1,1)
```

Ocorre quando gatilhos diferentes do mesmo tipo (AFTER ou Instead Of) se executam de forma circular (<http://technet.microsoft.com/en-us/library/ms190739.aspx>)

Sistemas de Informação II (ISEL-ADEETC Walter Vieira)

110

110

Gatilhos (T-SQL)

Gatilhos encaixados (podem produzir recursão indirecta):

```
insert into tg1 values(3,3)
```

Msg 2627, Level 14, State 1, Procedure trig1, Line 4

Violation of PRIMARY KEY constraint 'PK__tg2__7F2BE32F'. Cannot insert duplicate key in object 'dbo.tg2'.

The statement has been terminated.

*No máximo, são permitidos 32
níveis de ancadeamento*

```
exec sp_configure 'nested triggers', 0;
```

Reconfigure with override

*Com gatilhos **INSTEAD OFF** a execução
“encaixada” está sempre activada*

Sistemas de Informação II (ISEL-ADEETC Walter Vieira)

111

111

Gatilhos (T-SQL)

Também se considera recursão directa quando o mesmo gatilho é disparado duas vezes tendo de permeio a activação de um ou mais gatilhos de outro tipo:

- Se uma acção causa o disparo de um gatilho **instead of** e este executa uma acção que dispara um gatilho **after** que por sua vez executa uma acção que dispara o primeiro gatilho
- Se uma acção causa o disparo de um gatilho **after** e este executa uma acção que dispara um gatilho **instead of** que por sua vez executa uma acção que dispara o primeiro gatilho

Sistemas de Informação II (ISEL-ADEETC Walter Vieira)

112

112

Gatilhos (T-SQL)

Em <http://technet.microsoft.com/en-us/library/ms190739.aspx> pode ler-se:

•Direct recursion

This recursion occurs when a trigger fires and performs an action that causes the same trigger to fire again. For example, an application updates table T3; this causes trigger Trig3 to fire. Trig3 updates table T3 again; this causes trigger Trig3 to fire again.

Direct recursion can also occur when the same trigger is called again, but after a trigger of a different type (AFTER or INSTEAD OF) is called. In other words, direct recursion of an INSTEAD OF trigger can occur when the same INSTEAD OF trigger is called for a second time, even if one or more AFTER triggers are called in between. Likewise, direct recursion of an AFTER trigger can occur when the same AFTER trigger is called for a second time, even if one or more INSTEAD OF triggers are called in between. For example, an application updates table T4. This update causes INSTEAD OF trigger Trig4 to fire. Trig4 updates table T5. This update causes AFTER trigger Trig5 to fire. Trig5 updates table T4, and this update causes INSTEAD OF trigger Trig4 to fire again. This chain of events is considered direct recursion for Trig4.

Gatilhos (T-SQL)

Em <http://technet.microsoft.com/en-us/library/ms190739.aspx> pode ler-se:

•Indirect recursion

This recursion occurs when a trigger fires and performs an action that causes another trigger of the same type (AFTER or INSTEAD OF) to fire. This second trigger performs an action that causes the original trigger to fire again. In other words, indirect recursion can occur when an INSTEAD OF trigger is called for a second time, but not until another INSTEAD OF trigger is called in between. Likewise, indirect recursion can occur when an AFTER trigger is called for a second time, but not until another AFTER trigger is called in between. For example, an application updates table T1. This update causes AFTER trigger Trig1 to fire. Trig1 updates table T2, and this update causes AFTER trigger Trig2 to fire. Trig2 in turn updates table T1 that causes AFTER trigger Trig1 to fire again.

Only direct recursion of AFTER triggers is prevented when the RECURSIVE_TRIGGERS database option is set to OFF. To disable indirect recursion of AFTER triggers, also set the nested triggers server option to 0.

Gatilhos (T-SQL)

Podemos controlar a activação recursiva/encadeada de forma mais fina:

```
create trigger trig1 on tg1
for insert
AS
begin
    if (trigger_nestlevel() > 1 )
    begin
        RAISERROR(
            'activação recursiva ou encadeada de gatilhos em tg1',16,1)
    end
    ...
end
```

Para teste do nível de um gatilho específico:

```
IF ( (SELECT TRIGGER_NESTLEVEL( OBJECT_ID('tg1') , 'AFTER' , 'DML' ) ) > 1 )
RAISERROR(
    'activação recursiva ou encadeada do gatilho tg1',16,1)
```

IOT para instead of

115

Gatilhos (T-SQL)

ISO SQL 2016	SQL Server
<pre>CREATE TRIGGER GatilhoNivelLinha AFTER UPDATE OF nota ON Alunos REFERENCING NEW ROW AS novo OLD ROW AS velho FOR EACH ROW WHEN (novo.nota > velho.nota) INSERT INTO Melhorias VALUES (novo.numero)</pre>	<pre>CREATE TRIGGER GatilhoNivelLinha ON Alunos FOR UPDATE AS IF UPDATE(nota) BEGIN INSERT INTO Melhorias SELECT novo.numero FROM INSERTED AS novo, DELETED AS velho WHERE (novo.numero = velho.numero) AND (novo.nota > velho.nota) END</pre>

*Em casos mais complexos pode ser necessário utilizar cursores.
TEM DE HAVER, pelo menos, uma chave que não seja alterada*

116

Gatilhos (T-SQL)

Como fazer em SQL Server o equivalente a:

```
CREATE TRIGGER gc
BEFORE UPDATE OF saldo ON Contas
REFERENCING NEW ROW AS novo
FOR EACH ROW
WHEN (novo.saldo >
      (SELECT avg(saldo) FROM Contas))
INSERT INTO Ricos
VALUES (novo.numero)
```

Alguns sistemas limitam muito o tipo de instruções que se podem usar dentro de gatilhos BEFORE

117

Gatilhos (T-SQL)

```
CREATE TRIGGER gc ON Contas
AFTER UPDATE
AS
insert into Ricos
Select numero from Inserted where saldo >
      (select avg(saldo) from
      (select saldo from Contas where numero not in
      (select numero from deleted)
      UNION
      select saldo from deleted) as u)
```

Qual o nível de isolamento?

118

Gatilhos (T-SQL)

Ou:

CREATE TRIGGER gc **ON** Contas

INSTEAD OF UPDATE

AS

insert into Ricos

Select numero **from** Inserted **where** saldo >

(**select** avg(saldo) **from** Contas)

delete from Contas **where** numero **in** (**select** numero **from** inserted)

insert into Contas **select** * **from** inserted

119

Gatilhos (a ter em conta)

- Deve privilegiar-se a utilização de mecanismos de integridade declarativos
NOT NULL, UNIQUE, PRIMARY KEY, FOREIGN KEY, CHECK, ON UPDATE CASCADE, ON DELETE CASCADE, ASSERTIONS, ...
- Os gatilhos devem ser reservados para
 - assegurar a integridade quando tabelas relacionadas estão em máquinas diferentes (SQL Server)
 - aplicar regras de negócio complexas impossíveis de conseguir com os mecanismos declarativos (por exemplo, restrições de domínio complexas)
 - Introdução/remoção dinâmica de funcionalidade de forma transparente para as aplicações (Ex: auditing)
- Em SQL Server, a realização de rollback dentro de um trigger aborta a execução do “batch” de comandos corrente
- Em Sql Server, é conveniente colocar **SET NOCOUNT ON** nos procedimentos armazenados e gatilhos se a informação sobre n.º de linhas afectadas por instruções internas a estes objetos não for relevante nas aplicações cliente.

120

Gatilhos (a ter em conta)

- Os gatilhos executam-se sempre em âmbito transaccional (atomicidade das instruções SQL)
 - Em SQL Server, um rollback dentro de um trigger não anula apenas as acções do trigger, mas sim todas as acções da transacção em curso. Com savepoints (que veremos mais tarde) podemos ter melhor controlo sobre estas acções.
 - Algumas instruções SQL podem ocasionar erros não severos que não fazem abortar a transacção em curso; noutros casos, as instruções podem não produzir efeito. O programador de gatilhos tem de considerar cada uma destas possibilidades para verificar se elas podem dar origem a inconsistências. Nesse caso deverá forçar o rollback e gerar um erro.

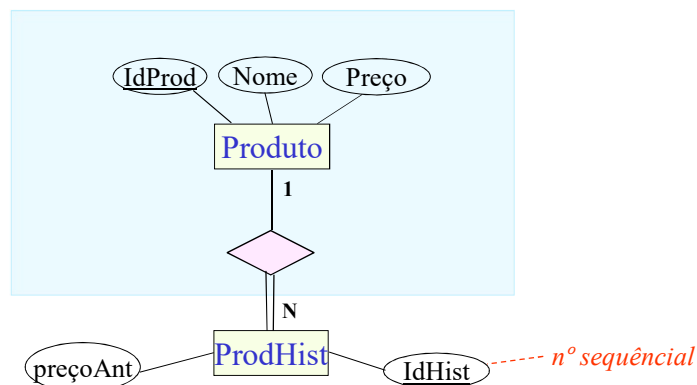
```
create trigger ActualizarHistorico on t1
...
DELETE FROM t2 WHERE ...
if @@ROWCOUNT = 0
begin
    rollback;
    throw ...
```

Sistemas de Informação II (ISEL-ADEETC Walter Vieira)

121

121

Exercício 1



Pretende-se que sempre que se altera o preço de um produto seja colocado em **prodHist** um registo com a indicação do preço anterior.

Sistemas de Informação II (ISEL-ADEETC Walter Vieira)

122

122

Exercício 1

Suponhamos as tabelas:

```
CREATE TABLE Produto(  
    IdProd int primary key,  
    Nome varchar(50) not null,  
    Preco float not null  
)  
  
CREATE TABLE ProdHist(  
    IdHist int identity primary key,  
    PrecoAnt float not null,  
    IdProd int NOT NULL references Produto  
)
```

123

Exercício 1

Solução com procedimento armazenado:

```
CREATE PROC AlterarPrecoProd(@IdProd int,  
                             @novoPreco float)  
AS  
BEGIN  
    set transaction isolation level repeatable read  
    begin transaction  
    declare @precoAnt float  
    select @precoAnt = preco from produto where IdProd = @IdProd  
    update produto set preco = @novoPreco where IdProd = @IdProd  
    insert into prodHist values(@precoAnt, @IdProd)  
    commit  
END
```

Quando o SP terminar 'recupera-se o nível de isolamento anterior'

Vantagem: melhor controlo e melhor desempenho

Desvantagem: Impede a utilização normal da linguagem SQL nas aplicações.

Não pode ser aplicada se já existirem aplicações a usar estes dados (teriam de ser refeitas)

Completar com testes de consistência

124

Exercício 1

Solução com gatilho:

```
CREATE TRIGGER GatilhoProd ON Produto
AFTER UPDATE
AS
    IF UPDATE(Preco)
    begin
        insert into prodHist
            select Preco, IdProd from DELETED
    end
```

Vantagem: Não impede a utilização normal da linguagem SQL nas aplicações.
Pode ser aplicada se já existirem aplicações a usar estes dados (não terão de ser refeitas)

Desvantagem: Pior controlo e pior desempenho

125

Exercício 2

Considere a seguinte criação da tabela produto:

```
CREATE TABLE Produto (
    cod varchar(10) PRIMARY KEY,
    preco float NOT NULL,
    NumOrdem int identity
)
```

Pretende-se garantir que os novos produtos apenas são inseridos se o seu preço não for inferior a 60% da média dos preços dos produtos já existentes.

126

Exercício 2

Solução com procedimento armazenado:

```
CREATE PROC InsProduto(@cod varchar(10), @preco float)
```

```
AS
```

```
declare @m float
```

```
set transaction isolation level SERIALIZABLE
```

```
begin tran
```

```
select @m = avg(preco) from produto
```

```
if (@m is null OR @preco >= 0.6*@m)
```

```
insert into Produto values(@cod,@preco)
```

```
commit
```

Quando o SP terminar 'recupera-se o nível de isolamento anterior'

se não for **SERIALIZABLE**, outra transacção pode inserir um registo sem contar com o que esta está a inserir no cálculo da média e vice-versa (por exemplo se 2 sessões concorrentes executarem este código)

127

Exercício 2

Solução com gatilho:

```
CREATE TRIGGER gp ON Produto AFTER INSERT
```

```
AS
```

```
set transaction isolation level SERIALIZABLE
```

```
delete from produto
```

```
where preco < 0.6*(select avg(preco) from produto
```

```
where cod not in (select cod from INSERTED))
```

```
and cod in (select cod from INSERTED)
```

Quando o gatilho terminar 'recupera-se o nível de isolamento anterior'

cuidado com recursividade se se tiverem triggers para DELETE

128

Exercício 2

Ou:

```
CREATE TRIGGER gp ON Produto INSTEAD OF INSERT
AS
set transaction isolation level SERIALIZABLE
insert into produto
select cod, preco from INSERTED where
    preco >= 0.6*(select avg(preco) from produto)
    or (select avg(preco) from produto) is null
```

Porque é que, neste caso, isto é necessário?

Notar que com gatilhos instead of, não há invocação recursiva

129

Exercício 2

Ou, ainda melhor:

```
CREATE TRIGGER gp ON Produto INSTEAD OF INSERT
AS
declare @med float
set transaction isolation level SERIALIZABLE
select @med=avg(preco) from produto
insert into produto
select cod, preco from INSERTED where
    preco >= 0.6*@med
    or @med is null
```

130

Exercício 3

Uma instituição bancária faz sorteios entre os seus clientes cujo saldo das contas aumenta. Para determinação da bonificação a existir foi desenvolvido o procedimento armazenado seguinte:

```
create proc pContas @num int, @ds real, @saldo real output
as
if exists (select numero from sortudos where numero = @num) and @ds > 0
    set @saldo = @saldo + @ds*1.1
else
    set @saldo = @saldo + @ds
```

Este procedimento recebe o número de uma conta, a diferença de saldo resultante de uma atualização da conta e o saldo anterior da conta e altera o parâmetro onde recebe o saldo anterior de forma a manter a alteração, ou a bonificá-la no caso de ser um depósito e o cliente ser um dos sorteados.

Desenvolva uma solução que permita aplicar as possíveis bonificações quando existem atualizações dos saldos das contas

Sistemas de Informação II (ISEL-ADEETC Walter Vieira)

131

131

Exercício 3

create trigger ts on contas instead of update

As

set transaction isolation level READ COMMITTED

if(update(saldo))

begin

declare @num int, @saldo real, @dsaldo real

declare c cursor for

select i.numero, d.saldo, i.saldo - d.saldo from inserted as i, deleted as d
where i.numero = d.numero

open c

fetch next from c into @num, @saldo, @dsaldo

while @@FETCH_STATUS = 0

begin

exec pContas @num, @dsaldo, @saldo output

update contas set saldo = @saldo where numero = @num

fetch next from c into @num, @saldo, @dsaldo

end

close c

deallocate c

end

Admitindo que é importante que as aplicações executem diretamente a instrução UPDATE

Cuidado: Se a tabela contas tivesse outras colunas, como o gatilho é instead of, a sua alteração não se consumiria fisicamente. Nesse caso talvez fosse preferível usar um gatilho AFTER

Sistemas de Informação II (ISEL-ADEETC Walter Vieira)

132

132

Exercício 3

Se em vez do procedimento armazenado se usasse uma função, poderíamos ter:

```
create function fContas(@num int, @ds real, @saldo real) returns real
as
begin
if exists (select numero from sortudos where numero = @num) and @ds > 0
    set @ds = @ds*1.1
return @saldo + @ds
End
```

```
create trigger ts_v2 on contas instead of update
As
set transaction isolation level READ COMMITTED
if(update(saldo))
begin
    -- usando a variante UPDATE FROM do T-SQL:
    update contas set saldo = dbo.fContas(i.numero,i.saldo-d.saldo,d.saldo) from
        inserted i join deleted d on i.numero = d.numero
        where contas.numero = i.numero
end
```

Cuidado: Se a tabela contas tivesse outras colunas, como o gatilho é instead of, a sua alteração não se consumiria fisicamente. Nesse caso talvez fosse preferível usar um gatilho AFTER

Exercício 4

A empresa SoTurbos, Lda dedica-se à venda de viaturas. Atualmente existem aplicações a correr sobre uma base de dados que contém a tabela **viatura(matricula, ano, preço)**. A empresa chegou à conclusão que no futuro também se deverá incluir informação sobre a **cilindrada da viatura**. Esta informação **não poderá ter o valor NULL**, deverá ser 0 para as viaturas manipuladas da forma anterior à alteração e **as novas aplicações têm de a indicar na altura das inserções**. A empresa também pretende que todas as aplicações existentes continuem a funcionar, de modo a que a sua alteração para suportar os novos dados possa ser escalonada ao longo do tempo sem prejudicar a atividade da empresa.

Apresente uma solução (incluindo o código TSQL necessário) que, em seu entender, resolva adequadamente este problema.

Exercício 4

set transaction isolation level SERIALIZABLE

begin tran

create table viaturaNova(matricula **char**(8) **primary key**,
ano **int**,
preco **float**)

insert into viaturaNova **select** * **from** viatura

alter table viaturaNova **add** cilindrada **int**

update viaturaNova **set** cilindrada = 0

alter table viaturaNova **add constraint** cc **CHECK**
(cilindrada IS NOT NULL)

135

Exercício 4

drop table viatura

GO

create view viatura **as select** matricula,ano,preco **from**
viaturaNova

GO

create trigger tViatura **on** viatura

INSTEAD OF INSERT

AS

insert into viaturaNova **select** matricula, ano, preco, 0 **as**
cilindrada **from** INSERTED

GO -- atação a este GO (porque é necessário?)

commit

136

Sequências – Sql Server

São mais gerais do que as colunas identity

- Permitem gerar sequências crescentes ou decrescentes dos tipos tinyint, smallint, int, bigint e decimal ou numeric com escala 0.
- Podem ser gerados independentemente de operações insert

```
CREATE SEQUENCE [schema_name . ] sequence_name
[ AS [ built_in_integer_type | user-defined_integer_type ] ]
[ START WITH <constant> ]
[ INCREMENT BY <constant> ]
[ { MINVALUE [ <constant> ] } | { NO MINVALUE } ]
[ { MAXVALUE [ <constant> ] } | { NO MAXVALUE } ]
[ CYCLE | { NO CYCLE } ]
[ { CACHE [ <constant> ] } | { NO CACHE } ]
[ ; ]
```

Não podem ser usadas dentro de funções

Também existem na norma ISO 2016

Sistemas de Informação II (ISEL-ADEETC Walter Vieira)

137

137

Sequências – Sql Server

```
CREATE SEQUENCE Seq
AS decimal(6,0)
START WITH 500000
INCREMENT BY -3
MINVALUE 499997
```

```
declare @x decimal(6,0)
set @x = NEXT VALUE FOR Seq
print @x
set @x = NEXT VALUE FOR Seq
print @x
set @x = NEXT VALUE FOR Seq
print @x
```

500000
499997

Msg 11728, Level 16, State 1, Line ...
The sequence object 'Seq' has reached its minimum or maximum value.
Restart the sequence object to allow new values to be generated.

Porque não se indicou a opção cycle

Sistemas de Informação II (ISEL-ADEETC Walter Vieira)

138

138

Sequências – Sql Server

Algumas limitações:

- Não se pode usar NEXT VALUE FOR em
check constraints, default objects, computed columns, views, user-defined functions, user-defined aggregates, user-defined table types, sub-queries, common table expressions, derived tables e return statements.
- Dentro do mesma instrução, usos diferentes de NEXT VALUE FOR produzem os mesmos valores para cada linha
update Ex1.t set c1 = NEXT VALUE FOR Ex1.Seq,
c2 = NEXT VALUE FOR Ex1.Seq produz valores repetidos para as colunas c1 e c2 em cada linha
- Tal como com colunas identity, se uma transação que use NEXT VALUE FOR abortar, o valor da sequência não é repostado (podem surgir gaps).

Sequências – Sql Server

```
ALTER SEQUENCE [schema_name.] sequence_name  
[ RESTART [ WITH <constant> ] ]  
[ INCREMENT BY <constant> ]  
[ { MINVALUE <constant> } | { NO MINVALUE } ]  
[ { MAXVALUE <constant> } | { NO MAXVALUE } ]  
[ CYCLE | { NO CYCLE } ]  
[ { CACHE [ <constant> ] } | { NO CACHE } ] [ ; ]
```

Bibliografia

Microsoft Transact-SQL statements (<https://docs.microsoft.com/en-us/sql/t-sql/statements/statements>)

**ANSI/ISO/IEC International Standard (IS)
Database Language SQL—Part 2: Foundation (SQL/Foundation)
(ISO/IEC 9075-2:2016)**

**ANSI/ISO/IEC International Standard (IS)
Database Language SQL—Part 4: Persistent Stored Modules (SQL/PSM)
(ISO/IEC 9075-4:2016)**

Ramez Elmasri and Shamkant B. Navathe, Fundamentals of Database Systems, Addison Wesley