

Base de Données II Chap2:

Fonctions et Procédures stockées

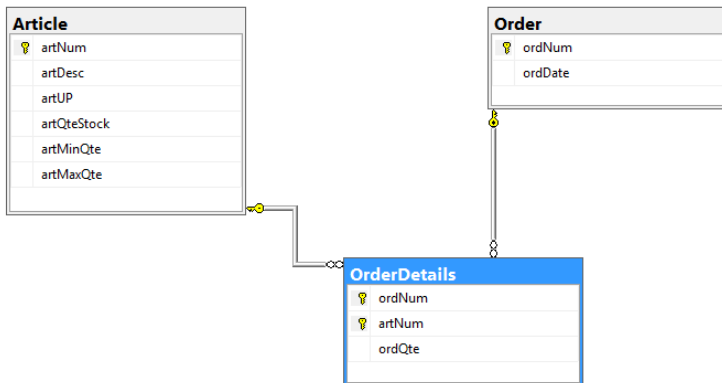
Functions and Stored Procedures



Outline

- 1 T-SQL overview
- 2 Functions
- 3 Stored procedures

Database to use





Transact-SQL syntax

T-SQL: Transact Structured Query Language

- 1 Declaring variables
- 2 Assignment
- 3 Displaying
- 4 Compound statement: begin ... end
- 5 Loops: while loop
- 6 Conditional statements (if-then-else)
- 7 Case statement



T-SQL variables

To declare a variable, we use:

```
1 DECLARE @variable_name datatype [ = initial_value ],  
2         @variable_name datatype [ = initial_value ],  
3         ...;
```

```
1 DECLARE @name varchar(50) = 'Marc';  
2 DECLARE @FirstTable table (RandomInteger int)
```

Declare more than one variable

```
1 DECLARE @name varchar(50) = 'Marc', @age int =30;
```

T-SQL variables



To change the value of a variable, we use the **SET** or **SELECT** statement.

```
1 DECLARE @id int, @name varchar(50)
2 SET @id=100
3 SELECT @name= 'Marc '
4 SELECT @id = MAX(artNum) from article
```



T-SQL displaying



To print on the screen, we use **print**:

Examples:

```
1 PRINT 'Bonjour tout le monde'
2 PRINT GETDATE()
```

```
1 PRINT 'The number of articles existed in the table article:'
2 DECLARE @v int
3 SELECT @v = COUNT(*) FROM article
4 PRINT @v
```



If-Else statements



Usual **if-else** statements are supported.

- 1 if-else
- 2 No else condition
- 3 Nested if-else





If-Else statements

```
1 DECLARE @v int;  
2 SET @v = 15;  
  
4 IF @v < 0  
5     PRINT 'Negative';  
6 ELSE  
7 BEGIN  
8     IF @v > 0  
9         PRINT 'Positive';  
10    ELSE  
11        PRINT 'Zero';  
12 END;  
  
14 GO
```





If-Else statements

Check if the **article number 10 in the stock has reached its minimum threshold**.
If this is the case display the message 'Out of Stock'.

```
1 declare @qs int
2 declare @m int
3
4 select @qs = artQteStock from article where artNum =10
5 Select @m = artMinQte from article where artNum =10
6
7 if @qs<=@m
8     print 'Out of Stock'
9 else
10    print 'Available in Stock'
```



WHEN statement



```
1 CASE
3     WHEN condition_1 THEN result_1
4     WHEN condition_2 THEN result_2
5     ...
6     WHEN condition_n THEN result_n
8     ELSE result
10 END
```





WHEN statement

Display the list of articles (number, description and price) plus a comment column that contains:

- 'Not Available' if the quantity in stock is 0;
- 'Available' if the quantity in stock is higher than the stock minimum;
- 'To Order' otherwise.

```
1 select artNum, artDesc, artUP, 'Observation' =  
2     case  
3         when artQteStock=0 then 'Not Available'  
4         when artQteStock>artMinQte then 'Available'  
5         else 'To Order'  
6     end  
7 from article
```



While loop



```
1 WHILE condition
2 BEGIN
3     Statements
4 END
```



While loop: example



```
1 DECLARE @cnt int = 0;

3 WHILE @cnt < 10
4 BEGIN
5     print 'Bonjour';
6     SET @cnt = @cnt + 1;
7 END;
```





While loop: example

- As the average price of items has not yet reached 20\$
- and the highest price for an article has not yet reached 30\$
- increase 10% the prices and display the list of articles
- After the price modification, display the new price average and the new highest price.

```
1 while ((select avg(artUP) from article)<20) and (select max(artUP) from article
   ) <30)
2 begin
3     update article set artUP=artUP + 0.1 * artUP
4     select * from article
5 end
6
7 declare @av int
8 declare @m int
9 select @av = avg(artUP) from article
10 select @m = max(artUP) from article
11 print @av
12 print @m
```

Outline



1 T-SQL overview

2 **Functions**

- Scalar user-defined functions
- Inline Table-valued UDFs
- Multistatement Table-valued UDFs

3 Stored procedures





Scalar UDFS

Scalar UDFs return a **single (scalar)** value.

Example: define a function which accepts an order ID as input and returns the number of articles for the input order.

```
1 create function getNB(@idOrd int)
2   returns int
3 as
4 begin
5   declare @nb int;
6   select @nb= count(*) from OrderDetails where ordNum=@idOrd
7   return @nb
8 end
```



Scalar UDFS



To test the getNB function, run the following query:

```
1 PRINT getNB(100)
```



Scalar UDFS and constraints



Add the constraints:

- 1 Each order ID in OrderDetails should be in Order
- 2 The ordered quantity for a given article should be $< \text{artstockQte} - \text{artMinQte}$





Scalar UDFS and constraints

Each order ID in OrderDetails should be in Order

```
1 create function orderIn(@idOrd int)
2     returns bit
3 as
4 begin
5     if(exists(select ordNum from Order where ordNum=@idOrd))
6         return 1
7     return 0
8 end
10 alter table OrderDetails add constraint ck_orderID check(orderIn(ordNum)=1)
```





Scalar UDFS and constraints

The ordered quantity for a given article should be $< \text{stockQte} - \text{minimum}$

```
1 create function checkQte(@q int,@idArt int)
2   returns bit
3 as
4 begin
5   declare @res bit
6   if((select count(*) from article where artNum=@idArt and @q<(artQteStock -
7     artMinQte))>1)
8     set @res=1
9   else
10    set @res=0
11  return @res
12 end
13 alter table ordersDetails add constraint ck_qte check(checkQte(artQteStock,
    numArt)=1)
```

Scalar UDFS and constraints



```
1 artNum=2
2 artQteStock=20
3 artMinQte=5
4 insert into OrderDetails values (1,2,10) — OK
5 insert into OrderDetails values (1,2,7) — Not OK
```



Table-valued UDFS



- Table-valued UDFs are UDFs that return a table;
- Are typically specified in the FROM clause of an outer query
- Two forms: **inline table-valued UDFS** and **multistatement table-valued UDFS**





Inline Table-valued UDFs

Syntax:

```
1 create function name_function(  
2 @param1 type1,  
3 @param2 type2,  
4 .  
5 .  
6 .  
7 )  
8 returns table  
  
10 as  
11 return  
12 (  
13     select statement with parameter references here  
14 )
```



Inline Table-Valued UDFs



- The **returns** clause contains only the keyword **table**;
- Unlike scalar and multistatement table-valued UDFs, there is no **function_body** delimited by **BEGIN/END** block;
- The **return** clause contains a single **select** statement in parentheses.





Inline Table-valued UDFs: example

Write the `getArticles` function, which accepts an order ID as an input and returns the article of that order:

```
1 create function getArticles(@idOrd int)
2   returns table
3 as
4   return
5     (select artNum, ordQte
6      from OrderDetails
7      where ordNum = @idOrd
8      )
9 GO
```

Usage:

```
1 select artNum
2 from getArticles(1)
```



Multistatement Table-valued UDFs

Syntax:

```
1 create function name_function(  
2 @param1 type1,  
3 .  
4 .  
5 .  
6 )  
7 returns @name_table table  
8 (  
9 column1 type1,  
10 .  
11 .  
12 .  
13 )  
14 as  
15 begin  
16     Filling the table variable with the rows for your result set  
17     return  
18 end
```



Multistatement Table-valued UDFs: example

Write the `detailsArticles` function, which accepts an article ID as input and returns the details (ordNum, ordDate, ordQte) of the order of that article.

```
1 create function detailsOf(@idArt int)
2   returns @t table(idOrd int,
3                   dateOrd datetime,
4                   qte integer
5                   )
6 as
7 begin
8   insert @t select o.ordNum, od.ordDate, od.ordQte
9             from OrderDetails od, Order o
10            where od.ordNum = o.ordNum and od.artNum=@idArt
11   return
12 end
```



Multistatement Table-valued UDFs: example



```
1 create function randTable()  
2     returns @t table(randInt INT))  
3 as  
4 begin  
5     DECLARE @i int  
6     SET @i = 0  
  
8     WHILE @i < 100000  
9         BEGIN  
  
11             INSERT INTO @t VALUES (RAND() * 10000)  
12             SET @i = @i + 1  
13         END  
14     return  
15 end
```





Multistatement Table-valued UDFs: example

```
1 create function randTable()  
2     returns @t table(randInt INT))  
3     as  
4     begin  
5         DECLARE @i int  
6         SET @i = 0  
  
8         WHILE @i < 100000  
9             BEGIN  
  
11                 INSERT INTO @t VALUES (RAND() * 10000)  
12                 SET @i = @i + 1  
13             END  
14     return  
15 end
```

ERROR: invalid use of a side-effecting operator rand() within a **function**
(values must be deterministic, no randomness must be involved).



Outline



- 1 T-SQL overview
- 2 Functions
- 3 Stored procedures**
 - User-defined stored procedures
 - System stored procedures



Procedural Extensions and Stored Procedures



Les procédures stockées sont des ensembles de requêtes SQL enregistrés dans les bases de données.

- SQL provides a module language:
 - Permits definition of procedures in SQL, with if-then-else statements, for and while loops, etc
- Stored Procedures:
 - Can store procedures in the database;
 - Then execute them using the call statement;
 - Permit external applications to operate on the database without knowing about internal details:
 - Stored procedures can be used as a security layer. You can control access



Stored procedures in TSQL



Two kinds of stored procedures are supported:

- 1 **User-defined** stored procedures;
- 2 **System** stored procedures.





Stored procedures in TSQL

- A **user-defined procedure** is a stored program that you can pass parameters into;
- It does not return a value like a function does.

Syntax:

```
1 Create Procedure procedure_name (  
2 input_param1 data_type = defaultValue,  
3 input_param2 data_type = defaultValue,  
4 output_param1 data_type Output,  
5 output_param2 data_type Output...)  
6 as  
8 Instructions
```



User-defined Stored Procedures



To invoke a user-defined stored procedure, we use the **EXEC** (or **EXECUTE**) command:

```
2 Exec procedure_name value_param1, value_Param2, value_Param3 output, ...
```

```
4 or
```

```
6 Exec procedure_name param1_name = value_param1, param2_name = value_Param2 ...
```



User-defined Stored Procedures



Get all the articles of a given order

```
1 create procedure getArticles
2   (@idOrd int
3   )
4 as
5 select a.artNum, artDesc, artUP, orderedQte
6 from article a, OrderDetails od
7 where a.artNum=od.artNum and od.ordNum=@idOrd
```





User-defined Stored Procedures

Get all the articles of a given order

```
1 create procedure getArticles
2   (@idOrd int
3   )
4 as
5 select a.artNum, artDesc, artUP, orderedQte
6 from article a, OrderDetails od
7 where a.artNum=od.artNum and od.ordNum=@idOrd
```

To execute the procedure getArticles:

```
1 EXECUTE getArticles 100
2 or
3 EXECUTE getArticles @idOrd=100
```





User-defined Stored Procedures

```
1 create procedure insertEmployee
2   (@fn varchar(15),
3    @ln varchar(15),
4    @salary int,
5    @hireDate datetime
6   )
7 as
8 declare @count int
9 select @count=0
10 select @count= count(*) from employee where fn=@fn and ln=@ln
11 if(@count=0)
12   begin
13     insert into employee(fn,ln,salary,hireDate) values(@fn, @ln,@salary,
14     @hireDate)
15     print 'Employee record inserted'
16   end
17 else
18   print 'Employee record already exists'
```



User-defined Stored Procedures



```
1 EXECUTE insertEmployee @fn='Marc', @ln='Aiguier', @salary=2000, @hireDate='  
    01-02-2008'
```





User-defined Stored Procedures

Get the number of articles of a given order.

```
1 create procedure nbOrders
2 (@idOrd int,
3 @n int output
4 )
5 as
6 select @n= count(*) from OrderDetails where ordNum=@idOrd
7 GO
```

```
1 declare @n int, @idOrd int
2 set @idOrd=1
3
4 EXECUTE nbOrders @idOrd, @n Output
5 print 'The number of articles associated to the order number' + convert(varchar
    ,@idOrd) + ' is: ' + convert(varchar,@n)
```



System stored procedures



- Uses the **sp_prefix** to designate system stored procedures;
- Resided in the **master database**;
- Large list of system stored procedures



System stored procedures: examples



- 1 sp_tables
- 2 sp_databases
- 3 sp_server_info
- 4 sp_help
- 5 sp_who
- 6 sp_rename





Stored procedure: sp_rename

sp_rename is used to change the name of a user-created object in the current database such as a table, index, column, etc.

Examples:

- 1 Rename the 'table1' table to 'table2'

```
EXEC sp_rename 'table1', 'table2';
```

- 2 Rename the column 'id' in the 'table1' to 'idBis'

```
EXEC sp_rename 'table1.id', 'idBis', 'COLUMN';
```

- 3 Rename the check constraint 'cons1' to 'cons2'

```
EXEC sp_rename 'cons1', 'cons2';
```