STORED PROCEDURE

Una stored procedure (SP) è un insieme di comandi T-SQL compilati, direttamente accessibili da SQL Server.

Tali comandi vengono eseguiti come un'unica unita di lavoro (**batch**) sul server e il beneficio è che il *traffico di rete viene ridotto* limitando la congestione della rete stessa.

CREATE PROC [EDURE] *nome_procedura*
AS *istruzione_sql* [...*n*]

I benefici derivanti dall'utilizzo delle SP sono diversi.

Ad esempio *evitano l'accesso diretto alle tabelle* da parte degli utenti ed inoltre vengono *ottimizzate tramite sistemi di caching* per essere più performanti possibile.

Si può dire che esse costituiscano una sorta di interfaccia tramite cui operare sui dati di un database senza conoscerne la struttura e le relazioni.

- **STORED PROCEDURE (DDL)** se l'*istruzione_sql* definisce strutture (crea/elimina)

- **STORED PROCEDURE: DML** se l'*istruzione_sql* modifica dati (SELECT, UPDATE, INSERT)

STORED PROCEDURE (DDL)
SENZA MEMORIZZARE con NOME la tabella

```
CREATE PROCEDURE Test2
AS
   CREATE TABLE #t(x INT PRIMARY KEY);
   INSERT INTO #t VALUES (2);
   SELECT Test2Col = x FROM #t;
```

Per ESEGUIRE: EXEC *nome* **Exec Test2**

Test2Col
2

Se si tratta della prima istruzione in un batch oppure di uno script **osql** o **sqlcmd**, non è necessario specificare EXEC

select * from sys.procedures

name	object_id	principal_id	schema_id	parent_object_id	type	type_desc	create_date	modify_date
Test2	750625717		1	0	P	SQL_STORED_PROCEDURE	12/13/2015 4:56:27 AM	12/13/2015 4:56:27 AM

```
CREATE PROC TestProc
AS
   CREATE TABLE #t(id INT PRIMARY KEY, Nome varchar (255) );
   INSERT INTO #t VALUES (1, 'Rossi');
   SELECT * FROM #t;
```

Per ESEGUIRE: **TestProc**

id	Nome
1	Rossi

Per eliminare: DROP PROCEDURE <stored procedure name>;

```
DROP PROCEDURE Test2
```

Meglio evitare eventuale errore se non esiste, testando con:

```
IF OBJECT_ID ('Test2') IS NOT NULL
DROP PROCEDURE Test2
```

Evitare di MEMORIZZARE con NOME la tabella: **in tal caso errore quando si riusa la procedura (tabella già creata) vanificando lo scopo principale: il RIUSO**

MEMORIZZANDO con NOME la tabella

```
CREATE PROC TabProc
AS CREATE TABLE Nuova (id INT PRIMARY KEY, Nome varchar (255) );
   INSERT INTO Nuova VALUES (1, 'Rossi');
   SELECT * FROM Nuova;
```

name
Nuova

ESEGUIRE cioè creare la tabella Nuova: TabProc

id	Nome
1	Rossi

effetto
analogo a **select * from Nuova**

Attuali tabelle … da eliminare Nuova *(usata solo per prova)* **con drop table Nuova**

name	object_id	principal_id	schema_id	parent_object_id	type	type_desc	create_date	modify_date
Persons	174623665		1	0	U	USER_TABLE	12/13/2015 03:40:34	12/13/2015 03:40:34
Elenco	245575913		1	0	U	USER_TABLE	11/25/2015 23:52:37	11/25/2015 23:52:37
Amici	277576027		1	0	U	USER_TABLE	11/25/2015 23:52:58	11/25/2015 23:52:58
Ordini	430624577		1	0	U	USER_TABLE	12/13/2015 04:11:19	12/13/2015 04:11:19
Nuova	1438628168		1	0	U	USER_TABLE	12/13/2015 05:51:23	12/13/2015 05:51:23

Attuali stored-procedure da eliminare *(usate solo per prova)*

name	object_id	principal_id	schema_id	parent_object_id	type	type_desc	create_date	modify_date
TestProc	1406628054		1	0	P	SQL_STORED_PROCEDURE	12/13/2015 05:45:46	12/13/2015 05:45:46
TabProc	1422628111		1	0	P	SQL_STORED_PROCEDURE	12/13/2015 05:50:54	12/13/2015 05:50:54

drop proc TabProc
drop proc TestProc → **attualmente nessuna procedura di creazione tabella (DDL)**

STORED PROCEDURE: DML

Oltre a istruzioni SELECT, UPDATE e DELETE una SP può richiamare altre SP, utilizzare istruzioni che controllano il flusso di esecuzione e funzioni di aggregazione.

MS SQL Database:

```
CREATE PROCEDURE SelectAll
AS
SELECT * FROM Elenco
GO;
```

SQL Query Result

MS SQL Database:

```
SelectAll
```

SQL Query Result

Nome	Tel_Abitazione	IDElenco
Paola Notini	010-3346590	1
Sara Notini	010-3346590	2

select * from sys.procedures

name	object_id	principal_id	schema_id	parent_object_id	type	type_desc
SelectAll	1205579333		1	0	P	SQL_STORED_PROCEDURE

E' importante sottolineare che oltre alle SP create dai programmatori esistono centinaia di ***SP di sistema*** all'interno di SQL Server (tutte cominciano con il ***prefisso sp_***).

Per visualizzare tutti gli oggetti di Sistema SELECT * FROM sysobjects WHERE Type='S'
oppure SELECT * FROM sysobjects WHERE name LIKE 'sys%'

name	id	xtype	uid	info	status	base_schema_ver	replinfo	parent_obj	crdate	ftcatid	schema_ver	stats_schema_ver	type
sysrscols	3	S	4	0	0	0	0	0	2/10/2012 8:16:00 PM	0	0	0	S
sysrowsets	5	S	4	0	0	0	0	0	4/13/2009 12:59:11 PM	0	0	0	S

La ***sintassi generale*** per la creazione di una SP :

```
CREATE PROCEDURE procedure_name
[ { @parameter_name} datatype [= default_value] [OUTPUT]]
[ { WITH [RECOMPILE | ENCRYPTION | RECOMPILE, ENCRYPTION } ]
AS
[BEGIN]
   statements
[END]
```

Dopo aver inserito il **nome** occorre specificare i **parametri** (se necessari e comunque **preceduti dalla @**) che la SP richiede, con **relativo tipo** di dati ed eventualmente valore di default.

E' anche possibile ritornare uno o più valori o una tabella di dati utilizzando un parametro per trasmettere tali informazioni. Tale parametro deve essere seguito dalla parola riservata OUTPUT e per esso non può essere definito un valore di default.

Nella sintassi generale possiamo notare anche le opzioni RECOMPILE e ENCRYPTION.

La prima indica a SQL Server di ricompilare la SP ogni volta che viene eseguita per forzare ogni volta la rigenerazione del piano di esecuzione, al fine di *migliorare le prestazioni*.

La seconda comporta la *crittografia del contenuto* della SP in modo che esso non sia visibile e comprensibile da chi non ha diritto di farlo.

Dopo la parola riservata AS comincia poi un blocco BEGIN-END (*opzionali*) dove possiamo inserire le nostre istruzioni T-SQL.

Di seguito un esempio[1] di una SP

CREATE PROCEDURE ProceduraTest **@id_cliente int** **AS** **SELECT LastName, AVG(prezzo) AS spesa_media** **FROM ordini, Persons** **Where PersonID =@id_cliente** **GROUP BY LastName**	**CREATE PROCEDURE ProceduraTest** **@id_cliente int** **AS** **begin** **SELECT LastName, AVG(prezzo) AS spesa_media** **FROM ordini, Persons** **Where PersonID =@id_cliente** **GROUP BY LastName** **End**

SELECT * FROM sys.procedures

name	object_id	principal_id	schema_id	parent_object_id	type	type_desc	create_date	modify_date
ProceduraTest	1726629194		1	0	P	SQL_STORED_PROCEDURE	12/13/2015 06:43:00	12/13/2015 06:43:00

[1] Esempi tratti da **http://www.mrwebmaster.it/sql-server/stored-procedure-funzioni_10512.html**

Per eseguire, se ci sono parametri

EXEC *nome* *@nomeParamentro* = 25

EXEC *nome* 25;

ProceduraTest 2

LastName	spesa_media
Dori	370

ProceduraTest @id_cliente =1

LastName	spesa_media
Perlo	370

Nb: se più parametri, è essenziale il corretto ordine

È possibile usare lo **stesso nome** per **parametro** e **campo:**

```
CREATE PROCEDURE ProceduraOK
@IdElenco int
AS
SELECT *
FROM Elenco
WHERE IdElenco = @IdElenco
```

ProceduraOK 1

MS SQL Database: dbDesign

```
ProceduraOK 1
```

SQL Query Result

Nome	Tel_Abitazione	IDElenco
Paola Notini	010-3346590	1

NB: vedremo poi la possibilità di creare FUNZIONI DEFINITE dall'UTENTE

Funzioni

Una SP è associata ad un'istruzione di SELECT: completa il proprio lavoro e termina la propria esecuzione. Questo è proprio uno scenario in cui le **funzioni** sono utili poichè sono simili alle SP ma è possibile utilizzarle all'interno di una query per **sfruttare i dati che esse restituiscono**.

Le funzioni possono essere di due tipi: **scalari** o **tabellari**.

Le scalari restituiscono un **singolo valore** con sintassi :

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

Anche le funzioni possono richiedere **parametri**, modificabili all'interno delle stesse se nella loro definizione dopo il tipo di dati non viene messo il valore READONLY. Diverso rispetto alle SP: l'istruzione **RETURNS** che serve a specificare il tipo di dati del valore scalare che la funzione restituirà e l'istruzione **RETURN** che fa sì che la funzione restituisca tale valore.

Le funzioni **tabellari** sono del tutto simili ma restituiscono una **tabella di dati** con sintassi

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

L'espressione **select_stmt** indica una query che restituirà i dati richiesti in forma tabellare.

Funzioni VS Stored Procedure

Le **funzioni** devono essere **deterministiche** e non possono essere utilizzate per apportare modifiche al database, mentre le stored procedure consentono di eseguire inserimenti e aggiornamenti, ecc.
Si condiglia di limitare l'uso delle funzioni, poiché rappresentano un enorme **problema di scalabilità** per query complesse. Diventano una specie di "scatola nera" per Query Optimizer con enormi differenze nelle prestazioni tra l'uso di funzioni e il semplice inserimento di codice in una query. Utili per i ritorni a valori di tabella in casi molto specifici.