

LEZIONE-TSQL-03 PROGRAMMAZIONE (VER02)

T-SQL PROGRAMMAZIONE

PARTE TERZA

VARIABILI, ASSEGNAZIONE, DECISIONI

Tratto da Gregory A. Larsen,





<http://www.databasejournal.com/features/mssql/article.php/3087431/T-SQL-Programming-Part-1---Defining-Variables-and-IFELSE-logic.htm>

Dopo aver appreso alcuni aspetti del T-SQL di Microsoft, è opportuno analizzare altri vari aspetti della programmazione T-SQL. Per costruire un **trigger**, una **stored procedure** (procedura memorizzata) o uno **script** del Query Analyzer, è necessario conoscere le basi della programmazione di T-SQL.

In questa dispensa si discuterà la definizione delle variabili, e l'utilizzo della logica IF ... ELSE.



VARIABILI LOCALI

Come con qualsiasi linguaggio di programmazione, T-SQL consente di definire e impostare le variabili. Una variabile è un contenitore che conserva un pezzo unitario di informazioni, come un numero o una stringa di caratteri. Le variabili possono essere utilizzate per svolgere un numero di cose. Alcuni esempi di utilizzo di variabili comuni sono:

-  Per passare i parametri alle stored procedure o una funzione
-  Per controllare la trasformazione di un ciclo
-  Per verificare una condizione vera o falsa in un IF
-  Per controllare le condizioni di programmazione in WHERE

In SQL Server è tipico l'uso di una variabile nota come «variabile locale», la cui esistenza è limitata ad un blocco o una procedura. L'ambito di una variabile locale è disponibile solo in modalità *batch*, *stored procedure* o *blocco di codice* in cui è definita. Una variabile locale è definita utilizzando il comando T-SQL "DECLARE" che significa «dichiara». Il nome della variabile locale deve necessariamente iniziare con il segno "@" usato come primo carattere del nome.

Una variabile locale può essere dichiarata per essere utilizzata in due modi:

-  come una qualsiasi variabile di sistema
-  come un nuovo tipo di dati definito dall'utente.

La tipica dichiarazione di una variabile è la seguente:

```
DECLARE @NOMEVARIABILE TIPO
```

Ad esempio, una variabile intera denominata @CNT può essere così dichiarata:

```
DECLARE @CNT INT
```



Più di una variabile può essere definita con un singolo DECLARE. Per definire più variabili, con un unico DECLARE, si separa ciascuna definizione di variabile con una virgola, in questo modo:

```
DECLARE @CONTA INT, @ETA TINYINT, @PRESTITO DATE, @NOME CHAR(10);
```

ASSEGNAZIONI E VALORI PER VARIABILI LOCALI

Sopra sono definite 4 variabili locali con un unico DECLARE. Ad una variabile locale viene inizialmente assegnato un valore NULL.

Un valore può essere assegnato ad una variabile locale utilizzando due possibili comandi:

-  **SET** (imposta, che assegna un valore costante o ottenuto con un'operazione)
-  **SELECT** (seleziona, che assegna un valore all'interno di una query SQL)

SET

Nel comando **SET** si specifica la variabile locale e il valore che si desidera assegnare alla variabile locale. Ecco un esempio di dove è definita la variabile @CONTA e poi è inizializzata col valore a 1.

```
INT DECLARE @CONTA
SET @CONTA = 1
```

SELECT

Nel comando **SELECT** si specifica la variabile locale all'interno della prima riga della query con l'espressione da calcolare da assegnare alla variabile locale. Ecco un esempio di come utilizzare l'istruzione **SELECT** per impostare il valore di una variabile locale.

```
DECLARE @CONTEGGIO INT
```

```
SELECT @CONTEGGIO = COUNT (*)
FROM MIODATABASE.MIATABELLA;
```

L'esempio sopra imposta la variabile @ ROWCNT il numero di righe della tabella MioDataBase.MiaTabella.

Uno degli usi di una variabile è quello di controllare e filtrare i record restituiti da un'istruzione **SELECT**. A tale scopo, è possibile utilizzare una variabile nella clausola **WHERE**. Ecco un esempio che restituisce tutti i record Clienti nel database Northwind in cui la colonna Paese Clienti è uguale a 'Italy':

```
DECLARE @PAESE VARCHAR (25)
SET @PAESE = 'ITALY'
```

```
SELECT *
FROM NORTHWIND.DBO.CUSTOMERS
WHERE COUNTRY = @PAESE ;
```

IF ... ELSE

T-SQL ha la "IF" che permette di aiutare con codice diverso da eseguire sulla base dei risultati di una condizione. Il "IF" consente un T-SQL programmatore per eseguire selettivamente una sola riga o un blocco di codice basato su una condizione booleana. Ci sono due formati per l'istruzione "IF", entrambi sono indicati di seguito:

- Primo Formato: **IF** <condizione>
 < codice **then** da eseguire solo se la condizione rende vero >
- Secondo Formato: **IF** <condizione>
 < codice **then** da eseguire solo se la condizione rende vero >
 ELSE <altro codice da eseguire quando la condizione rende falsa>

In entrambi i formati, la <condizione> è un'espressione booleana o una sequenza di espressioni booleane che restituiscono true o false. Se la condizione restituisce true, allora l'istruzione immediatamente successiva viene eseguita. Il codice da eseguire deve essere una singola istruzione TSQL oppure un blocco di codice che deve essere racchiuso in una dichiarazione **BEGIN** ed **END**.

Per il secondo formato, se la condizione è falsa, allora viene eseguito il codice immediatamente successivo alla clausola **ELSE**.

IF SEMPLICE

Analizziamo meglio il funzionamento del primo formato. L'esempio proposto mostra come scrivere l'istruzione IF per eseguire una singola istruzione, se la condizione valutata è vera. Nell'esempio, banalmente, ci si limita a verificare se una variabile è impostata su un valore specifico ed eventualmente stampare un messaggio appropriato.

```

DECLARE @MESE INT
SET @MESE = 12
IF @MESE = 12 PRINT 'IL MESE È DICEMBRE'
IF @MESE < 12 PRINT 'IL MESE NON È DICEMBRE'

```

Il codice sopra stampa solo la frase «IL MESE È DICEMBRE», perché la condizione del primo IF restituisce true. Poiché la condizione del secondo IF è falsa la rispettiva istruzione PRINT non viene eseguita.

Occorre osservare che la condizione di un IF può anche contenere una intera clausola SELECT. Se viene utilizzata una istruzione SELECT, l'intera query deve essere racchiusa tra parentesi tonde. L'istruzione SELECT avrà bisogno di restituire un valore o anche un insieme di valori che possano essere confrontati in qualche modo. Per esempio:

```

IF ( SELECT COUNT (*)
      FROM DBSCUOLA.STUDENTI
      WHERE ETÀ < 18)
  > 0
PRINT 'HO TROVATO STUDENTI MINORENNI '

```

Oppure, usando la LIKE di T-SQL, posso eseguire:

```

IF ( SELECT COUNT (*)
      FROM MYDB.AUTORI
      WHERE AU_NAME LIKE '[AD]%' )
  > 0
PRINT 'TROVATO AUTORI CON INIZIALE A..D'

```

Qui si stampa il messaggio «TROVATO AUTORI CON INIZIALE A..D» se l'istruzione SELECT ha trovato qualche autore nella tabella AUTORI il cui cognome inizi con una A, B, C o D.

In questi due esempi si è mostrato come eseguire una singola istruzione T-SQL qualora la condizione testata sia vera. Il linguaggio T-SQL consente di eseguire anche un blocco di codice come azione. Un blocco di codice viene definito utilizzando una clausola "BEGIN" prima della prima riga di codice nel blocco di codice, e una dichiarazione "END" dopo l'ultima riga di codice nel blocco di codice.

Ecco qualche esempio che esegue un blocco di codice quando la condizione di istruzione IF restituisce true.

```

IF DB_NAME () = 'MASTER'
BEGIN
PRINT 'SIETE NEL MASTER DATABASE '
PRINT ' '
PRINT 'QUINDI STATE ATTENTI A CIÒ CHE SI ESEGUE '
END

```

La sequenza di clausole "PRINT" saranno eseguite qualora il test dell'IF venga eseguito nel contesto del database master; se il contesto è un altro database, nessuno dei comandi di stampa sarà eseguito.

IF .. ELSE

Talvolta si vuole non solo eseguire del codice qualora la condizione sia vera, ma anche eseguire un diverso insieme di istruzioni T-SQL quando la condizione sia falsa. Per questa necessità è opportuno utilizzare il

costrutto IF ... ELSE, indicato sopra come secondo formato. Con questo formato, se la condizione è vera, allora la dichiarazione o il blocco di codice che segue la clausola IF viene eseguito, ma se la condizione restituisce falso allora verrà eseguito l'istruzione o il blocco di codice che segue la clausola ELSE. Vediamo un paio di esempi.

Vediamo il caso di un conferimento di una certa quantità (Kg) di un ortaggio (codice PG01) odierno da parte di un socio (IS47) di una cooperativa agricola; se il socio, nella stessa data, ha già conferito quell'ortaggio allora si tratta di incrementare la quantità già versata; viceversa occorre inserire un nuovo conferimento.

```

DECLARE @QUANTO INT
SET @QUANTO = 345;
IF EXISTS (      SELECT *
                  FROM COOPERATIVA.CONFERIMENTI
                  WHERE IDSOCIO='IS47' AND IDPRODOTTO='PG01' )

BEGIN
UPDATE COOPERATIVA.CONFERIMENTI
SET QUANTITÀ = QUANTITÀ + @QUANTO
WHERE IDSOCIO='IS47' AND IDPRODOTTO='PG01'
END
ELSE
BEGIN
INSERT INTO COOPERATIVA.CONFERIMENTI
          (IDSOCIO, IDPRODOTTO, QUANTITÀ)
VALUES ( 'IS47' , 'PG01' , @QUANTO )
END

```

È anche possibile determinare più condizioni in un IF legate da operatori logici oppure nidificare più IF in cascata, nel caso che sia opportuno agire secondo una logica articolata. Per esempio, vediamo un caso in cui uno script determina se l'ambito della query è nel database 'Northwind' e se la tabella "Clienti" esiste.

La scrittura di questa interrogazione è proposta in due modi diversi, il primo con condizioni multiple su un unico IF, e l'altro usando istruzioni IF nidificate.

```

-- COMANDO IF SINGOLO, CON PIÙ CONDIZIONI
USE NORTHWIND
IF DB_NAME() = 'NORTHWIND' AND
    (SELECT COUNT(*)
     FROM SYSOBJECTS
     WHERE NAME = 'CUSTOMERS') = 1
PRINT 'TABELLA CUSTOMERS ESISTENTE'
ELSE
PRINT 'NON DATABASE NORTHWIND' + 'O TABELLA CUSTOMER INESISTENTE'

```

```

-- COMANDI IF NIDIFICATI
USE NORTHWIND
IF DB_NAME() = 'NORTHWIND'
IF (SELECT COUNT(*)
     FROM SYSOBJECTS
     WHERE NAME = 'CUSTOMERS') = 1
PRINT 'TABELLA CUSTOMERS ESISTENTE'
ELSE
PRINT 'NON DATABASE NORTHWIND'
ELSE
PRINT ' TABELLA CUSTOMER INESISTENTE'

```

ISTRUZIONI ITERATIVE IN T-SQL

Questo è il secondo articolo della sequenza di Gregory A. Larsen sulla programmazione T-SQL.

Questo articolo affronta la costruzione di un ciclo di programma con T-SQL. Oltre a discutere di come costruire un ciclo, si discute anche delle modalità di controllo sulla gestione dei loop, e dei diversi modi per uscire da un ciclo.

Un ciclo di programmazione è un pezzo di codice che viene eseguito più volte iterativamente. Nel ciclo, una certa logica viene eseguita ripetutamente in modo iterativo fino a quando una condizione è verificata, che consente al codice di uscire dal loop. Un esempio di dove si potrebbe utilizzare un ciclo potrebbe essere quella di elaborare una serie di record un record alla volta. Un altro esempio potrebbe essere quello in cui è necessario generare alcuni dati di prova e di un ciclo che permette di inserire un record nella tabella dei dati di test con i valori delle colonne leggermente diversi, ogni volta che il ciclo viene eseguito. In questo articolo si accennerà ai costrutti `time`, `break`, `continue` e `goto`.

WHILE

In T-SQL l'istruzione **WHILE** è il modo più comunemente utilizzato per eseguire un ciclo. La sintassi di base per un ciclo **WHILE** è:

```
WHILE <CONDIZIONE> <BLOCCO>
```

La <CONDIZIONE> è una qualsiasi espressione booleana che può essere valutata e restituire vero o falso.

Il <BLOCCO> è il frammento di programma che si intende eseguire nel caso la condizione sia vera.

Vediamo un semplice esempio reale. Si desidera incrementare un contatore da 1 a 10 e visualizzare il valore del contatore ad ogni iterazione del ciclo **WHILE**.

```
DECLARE @CONTA INT
SET @CONTA = 0
WHILE (@CONTA < 10)
BEGIN
SET @CONTA = @CONTA + 1
PRINT 'CONTA VALE ' + CAST(@CONTA AS CHAR)
END
```

Qui il codice esegue l'istruzione **WHILE** a condizione che il @ variabile contatore intero è inferiore a 10, questa è l'espressione booleana del ciclo while. La variabile @ contatore inizia a zero, e ogni volta attraverso il ciclo **WHILE** viene incrementato di 1. L'istruzione **PRINT** visualizza il valore della variabile @ contatore ogni iterazione del ciclo **WHILE**. L'uscita da questo campione è simile al seguente:

```
CONTA vale 1
CONTA vale 2
CONTA vale 3
CONTA vale 4
CONTA vale 5
CONTA vale 6
CONTA vale 7
CONTA vale 8
CONTA vale 9
CONTA vale 10
```

Oltre a poter utilizzare un singolo ciclo **WHILE**, è possibile nidificare più cicli scrivendoli uno dentro l'altro. I motivi per usare l'annidamento dei cicli sono diversi; comunemente si usa la nidificazione di cicli **WHILE** per generare dati di test nelle prove del sistema. Il seguente esempio propone un ciclo **WHILE** per generare i record di una tabella di prova **PROVA**. Un record in **PROVA** è formato da un identificatore **PROVA_ID**, e da un **CATEGORIA_ID**; infine c'è un campo **DETTAGLI**.

Per ogni **Part_Id** ci sono tre diversi **category_id** del. Nell'esempio si generano 6 record differenti per la tabella **ARTICOLI** utilizzando un ciclo nidificato **WHILE**.

```

DECLARE @PROVA_ID INT
DECLARE @CATEGORIA_ID INT
DECLARE @DESC VARCHAR(50)
CREATE TABLE PROVA (
    PROVA_ID INT,
    CATEGORIA_ID INT,
    DETTAGLI VARCHAR(50)
)
SET @PROVA_ID = 0
SET @CATEGORIA_ID = 0
WHILE @PROVA_ID < 2
BEGIN
    SET @PROVA_ID = @PROVA_ID + 1
    WHILE @CATEGORIA_ID < 3
    BEGIN
        SET @CATEGORIA_ID = @CATEGORIA_ID + 1
        SET @DESC = 'PROVA_ID: ' + CAST(@PROVA_ID AS CHAR(1)) +
            'CATEGORIA_ID: ' + CAST(@CATEGORIA_ID AS CHAR(1))
        INSERT INTO PROVA
        VALUES (@PROVA_ID, @CATEGORIA_ID, @DESC)
    END
    SET @CATEGORIA_ID = 0
END
SELECT *
FROM PROVA
DROP TABLE PROVA

```

Ecco l'output del SELECT che è proposta nella sezione finale dell'esempio prima di eliminare l'intera tabella (eliminare la struttura, non solo i record, con DROP).

PROVA_ID	CATEGORIA_ID	DESCRIZIONE
1	1	PROVA_ID: 1 CATEGORIA_ID: 1
1	2	PROVA_ID: 1 CATEGORIA_ID: 2
1	3	PROVA_ID: 1 CATEGORIA_ID: 3
2	1	PROVA_ID: 2 CATEGORIA_ID: 1
2	2	PROVA_ID: 2 CATEGORIA_ID: 2
2	3	PROVA_ID: 2 CATEGORIA_ID: 3

Argomenti Trattati:

T-SQL PROGRAMMAZIONE PARTE SECONDA	1
VARIABILI, ASSEGNAZIONE, DECISIONI	1
Variabili locali	1
Assegnazioni e valori per variabili locali	1
SET	2
SELECT	2
IF ... ELSE	2
IF semplice.....	3
IF . . ELSE.....	3
ISTRUZIONI ITERATIVE IN T-SQL.....	5
WHILE	5