Come usare lo Statement MERGE (T-SQL)

#### di [Sergio Govoni](http://mvp.microsoft.com/profiles/Sergio.Govoni) - Microsoft MVP

Blog: <http://www.ugiss.org/sgovoni/>

Twitter: <https://twitter.com/segovoni>

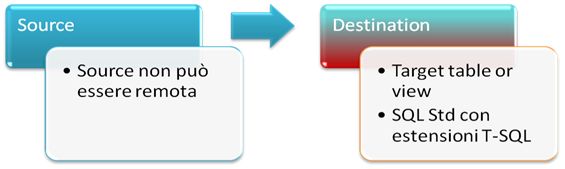
* 1. 

*Febbraio, 2013*

Introduzione

SQL Server 2008 introduce il supporto al comando MERGE. Questo comando permette di distinguere una tabella sorgente (source) ed una tabella destinazione (destination), modificando il contenuto della tabella di destinazione con i dati della tabella sorgente. SQL Server implementa tutte le caratteristiche del comando MERGE standard ANSI SQL-2006 e aggiunge, a queste, alcune importanti estensioni disponibili solo nel linguaggio T-SQL.

La figura seguente schematizza sorgente e destinazione in un comando MERGE.

* 1. 
  2. Figura 1 – Sorgente e Destinazione nel comando MERGE

Ambiti di utilizzo

Il comando MERGE può essere utilizzato in ambienti OLTP, ma anche OLAP. In uno scenario transazionale, possiamo utilizzarlo per eseguire il Merge dei dati da una sorgente esterna ad una tabella esistente. In un data werehouse, il comando MERGE può essere utilizzato per eseguire UPDATE incrementali oppure per processare slowly changing dimensions.

Il comando MERGE viene risolto utilizzando operazioni di Join, che interessano la tabella sorgente e la tabella destinazione. Il predicato ON esprime la condizione di Join che deve essere verificata per collegare le righe della tabella sorgente con le righe della tabella destinazione. Si potrà quindi specificare quale azione avviare quando la riga:

* + Esiste sia nella tabella sorgente che nella tabella destinazione (WHEN MATCHED)
  + Esiste nella tabella sorgente, ma non nella tabella destinazione (WHEN NOT MATCHED [BY TARGET])
  + Esiste nella tabella destinazione, ma non in quella sorgente (WHEN NOT MATCHED [BY SOURCE])

L’ultima clausola WHEN NOT MATCHED [BY SOURCE] rappresenta un’estensione proprietaria del linguaggio T-SQL, non è disponibile nel comando MERGE standard ANSI SQL.

Scenario OLTP: Inventario di magazzino

Vediamo ora come il comando MERGE può aiutarci nel caricamento di un inventario di magazzino.

Ipotizziamo di eseguire l’inventario a rotazione, dei prodotti giacenti a magazzino e di dover aggiornare le giacenze logiche memorizzate nella tabella dbo.ProductInventory del database AdventureWorks2012, con le giacenze fisiche rilevate dagli operatori durante l’inventario. Il comando MERGE sarà di grande aiuto per aggiornare i dati contenuti nella tabella dbo.ProductInventory. Si dovrà quindi prevedere di:

* + Aggiornare la quantità giacente di un prodotto
  + Eliminare, dalla tabella dbo.ProductInventory, i prodotti non presenti a magazzino (rilevazioni con giacenza uguale a zero)
  + Inserire, nella tabella dbo.ProductInventory, i nuovi prodotti rilevati a magazzino (non presenti nella destinazione)

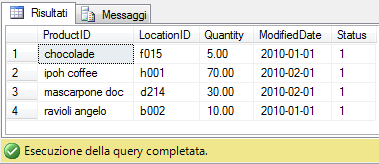
Procediamo con la creazione della tabella dbo.ProductInventory sul database AdventureWorks2012 (scaricabile da CodePlex) e inseriamo alcuni dati di prova.

* 1. use [AdventureWorks2012];
  2. go
  3. ------------------------------------------------------------------------
  4. -- Setup table & insert data
  5. ------------------------------------------------------------------------
  6. -- dbo.ProductInventory
  7. if OBJECT\_ID('dbo.ProductInventory', 'U') is not null
  8. drop table dbo.ProductInventory;
  9. go
  10. create table dbo.ProductInventory
  11. (
  12. ProductID varchar(25) not null
  13. ,LocationID varchar(20) not null
  14. ,Quantity decimal(9, 2) not null
  15. ,ModifiedDate date not null default getdate()
  16. ,Status bit not null default 1
  17. ,constraint PK\_ProductInventory primary key(ProductID, LocationID)
  18. );
  19. insert into dbo.ProductInventory
  20. (ProductID, LocationID, Quantity, ModifiedDate)
  21. values
  22. ('ravioli angelo', 'b002', 10, '20100101'),
  23. ('chocolade', 'f015', 5, '20100101'),
  24. ('ipoh coffee', 'h001', 70, '20100201'),
  25. ('mascarpone doc', 'd214', 30, '20100201');
  26. go

Consultiamo i dati inseriti, che rappresentano le giacenze logiche dei prodotti disponibili in magazzino:

* 1. use [AdventureWorks2012];
  2. go
  3. select \* from dbo.ProductInventory;
  4. go

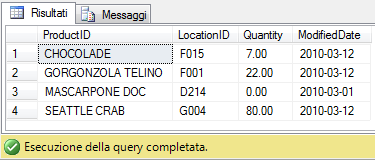
L’output è illustrato in figura 2.

* 1. 
  2. Figura 2 – Giacenze logiche dei prodotti disponibili in magazzino

Ipotizziamo di raccogliere i dati inventariali nella tabella dbo.FrequentInventory, che possiamo creare e popolare con alcuni dati di test attraverso il seguente frammento di codice T-SQL:

* 1. use [AdventureWorks2012];
  2. go
  3. -- dbo.FrequentInventory
  4. if OBJECT\_ID('dbo.FrequentInventory', 'U') is not null
  5. drop table dbo.FrequentInventory;
  6. go
  7. create table dbo.FrequentInventory
  8. (
  9. ProductID varchar(25) not null
  10. ,LocationID varchar(20) not null
  11. ,Quantity decimal(9, 2) not null
  12. ,ModifiedDate date not null default getdate()
  13. ,constraint PK\_FrequentInventory primary key(ProductID, LocationID)
  14. );
  15. go
  16. insert into dbo.FrequentInventory
  17. (ProductID, LocationID, Quantity, ModifiedDate)
  18. values
  19. ('CHOCOLADE', 'F015', 7, '20100612'),
  20. ('GORGONZOLA TELINO', 'F001', 22, '20100612'),
  21. ('SEATTLE CRAB', 'G004', 80, '20100612'),
  22. ('MASCARPONE DOC', 'D214', 0, '20100201');
  23. go

Consultiamo i dati rilevati durante l’inventario, l’output è illustrato in figura 3.

* 1. use [AdventureWorks2012];
  2. go
  3. select \* from dbo.FrequentInventory;
  4. go
  5. 
  6. Figura 3 – Giacenze fisiche rilevate dai magazzinieri durante l’inventario

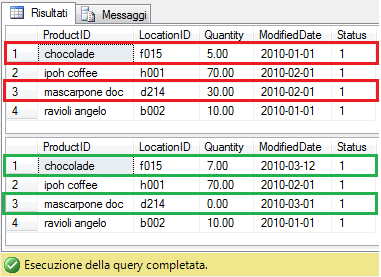
Il comando MERGE in azione

Il primo comando MERGE che esaminiamo è quello che ci permette di aggiornare le righe presenti sia nella tabella sorgente dbo.FrequentInventory che nella tabella destinazione dbo.ProductInventory. Per far questo è sufficiente scrivere un comando di UPDATE nella clausola WHEN MATCHED. Si osservi la presenza della transazione esplicita, dopo aver modificato i dati ed aver interrogato il contenuto della tabella dbo.ProductInventory, la transazione verrà respinta con un ROLLBACK esplicito in modo che si possano utilizzare gli stessi dati anche per i prossimi esempi.

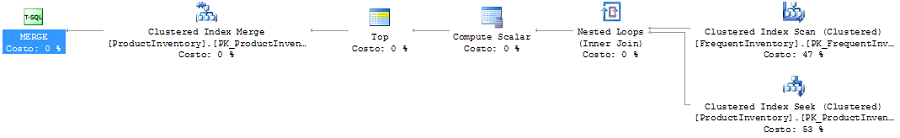
Osserviamo il comando UPDATE, non è stata specificata la tabella dbo.ProductInventory, quest’ultima è infatti già nota by design, è la tabella di destinazione.

* 1. -- MERGE con la clausola WHEN MATCHED
  2. begin tran;
  3. go
  4. select \* from dbo.ProductInventory;
  5. go
  6. merge into dbo.ProductInventory as itarget
  7. using dbo.FrequentInventory as isource
  8. on ((itarget.ProductId = isource.ProductId)
  9. and (itarget.LocationId = isource.LocationId))
  10. when matched then
  11. update /\* dbo.ProductInventory \*/ set
  12. itarget.Quantity = isource.Quantity
  13. ,itarget.ModifiedDate = isource.ModifiedDate;
  14. select \* from dbo.ProductInventory;
  15. go
  16. rollback tran;
  17. go

L’output è illustrato in figura 4, le righe nei riquadri rossi rappresentano i record che verranno aggiornati, mentre quelle nei riquadri verdi rappresentano i record dopo l’esecuzione del comando MERGE.

* 1. 
  2. Figura 4 – Aggiornamenti eseguiti dal comando MERGE con clausola WHEN MATCHED

In presenza della sola clausola WHEN MATCHED, il comando MERGE viene implementato con una INNER JOIN che possiamo vedere nel piano di esecuzione illustrato nella figura seguente.

* 1. 
  2. Figura 5 – Piano di esecuzione per il comando MERGE con clausola WHEN MATCHED

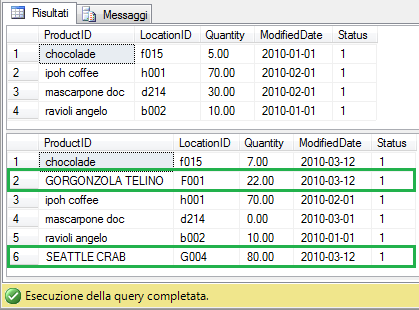
La clausola WHEN NOT MATCHED

Soddisfiamo un altro requisito richiesto, ovvero la possibilità di inserire le righe della tabella sorgente che non trovano corrispondenza nella tabella destinazione. Modifichiamo quindi il precedente comando MERGE per aggiungere la clausola WHEN NOT MATCHED alla quale assoceremo un’operazione di INSERT. Come per il comando di UPDATE, la sintassi del comando INSERT è molto simile a quella che utilizziamo abitualmente ad eccezione del nome della tabella oggetto degli inserimenti, che non viene specificata perché la tabella di destinazione è già nota.

Eseguiamo il seguente frammento di codice, anche in questo caso la transazione viene annullata con un ROLLBACK.

* 1. -- MERGE con le clausole WHEN MATCHED e WHEN NOT MATCHED
  2. begin tran;
  3. go
  4. select \* from dbo.ProductInventory;
  5. go
  6. merge into dbo.ProductInventory as itarget
  7. using dbo.FrequentInventory as isource
  8. on ((itarget.ProductId = isource.ProductId)
  9. and (itarget.LocationId = isource.LocationId))
  10. when matched then
  11. update /\* dbo.ProductInventory \*/ set
  12. itarget.Quantity = isource.Quantity
  13. ,itarget.ModifiedDate = isource.ModifiedDate
  14. when not matched then
  15. insert /\* into dbo.ProductInventory \*/
  16. (
  17. ProductId
  18. ,LocationId
  19. ,Quantity
  20. ,ModifiedDate
  21. )
  22. values
  23. (
  24. isource.ProductId
  25. ,isource.LocationId
  26. ,isource.Quantity
  27. ,isource.ModifiedDate
  28. );
  29. select \* from dbo.ProductInventory;
  30. go
  31. rollback tran;
  32. go

Otteniamo l’output illustrato in figura 6, osserviamo che i prodotti “GORGONZOLA TELINO” e “SEATTLE CRAB” sono stati inseriti nella tabella destinazione dbo.ProductInventory per effetto dell’azione specificata nella clausola WHEN NOT MATCHED.

* 1. 
  2. Figura 6 – MERGE con clausole WHEN MATCHED e WHEN NOT MATCHED

La soluzione nelle precedenti versioni di SQL Server (SQL Server 2005 o precedenti)

Come avremmo implementato la soluzione di questo problema in SQL Server 2005 o precedenti? Probabilmente scrivendo due comandi separati, un UPDATE ed un INSERT, come quelli riportati di seguito, che avrebbero permesso di ottenere lo stesso output di figura 6.

* 1. begin tran
  2. go
  3. select \* from dbo.ProductInventory;
  4. go
  5. update
  6. itarget
  7. set
  8. itarget.Quantity = isource.Quantity
  9. ,itarget.ModifiedDate = isource.ModifiedDate
  10. from
  11. dbo.ProductInventory itarget
  12. join
  13. dbo.FrequentInventory isource
  14. on (itarget.ProductID = isource.ProductID) and
  15. (itarget.LocationID = isource.LocationID);
  16. go
  17. insert into dbo.ProductInventory
  18. (
  19. ProductID, LocationID, Quantity, ModifiedDate
  20. )
  21. select
  22. isource.ProductID
  23. ,isource.LocationID
  24. ,isource.Quantity
  25. ,isource.ModifiedDate
  26. from
  27. dbo.FrequentInventory as isource
  28. where
  29. not exists (select \*
  30. from
  31. dbo.ProductInventory as itarget
  32. where
  33. (isource.ProductID = itarget.ProductID)
  34. and (isource.LocationID = itarget.LocationID)
  35. );
  36. go
  37. select \* from dbo.ProductInventory;
  38. go
  39. rollback tran
  40. go

Rispetto alla soluzione tradizionale, uno dei vantaggi che deriva dall’utilizzo del comando MERGE è nell’accesso ai dati, con MERGE i dati vengono acceduti una sola volta, contro il doppio accesso nel caso dei due comandi separati INSERT e UPDATE. Il comando MERGE viene eseguito con una operazione atomica, che non necessità di transazioni esplicite. In aggiunta, l’operazione Merge ci protegge da eventuali UPDATE non deterministici, nel senso che, se la stessa riga verrà aggiornata più di una volta, l’intero comando di MERGE fallirà.

Le clausole WHEN supportano la presenza dell’operatore AND seguito da un predicato, la clausola assumerà quindi la forma:

* 1. WHEN MATCHED AND <predicato> THEN <azione>

Aggiunta dell’operatore AND

Aggiungendo l’operatore AND, l’azione racchiusa nella clausola verrà eseguita solo se risulteranno essere vere entrambe le condizioni, quella specificata nel predicato ON e quella specificata nel predicato aggiuntivo AND.

Il seguente comando MERGE permette di aggiornare le righe nella tabella di destinazione solo se gli attributi non chiave sono diversi da quelli nella sorgente. Con questa tecnica, le performance migliorano, si previene inoltre l’attivazione di eventuali trigger.

* 1. -- MERGE con operatore AND nella clausola WHEN MATCHED
  2. begin tran;
  3. go
  4. select \* from dbo.ProductInventory;
  5. go
  6. merge into dbo.ProductInventory as itarget
  7. using dbo.FrequentInventory as isource
  8. on ((itarget.ProductId = isource.ProductId)
  9. and (itarget.LocationId = isource.LocationId))
  10. when matched AND
  11. ((itarget.Quantity <> isource.Quantity)
  12. or (itarget.ModifiedDate <> isource.ModifiedDate)) then
  13. update set
  14. itarget.Quantity = isource.Quantity
  15. ,itarget.ModifiedDate = isource.ModifiedDate
  16. when not matched then
  17. insert
  18. (
  19. ProductId
  20. ,LocationId
  21. ,Quantity
  22. ,ModifiedDate
  23. )
  24. values
  25. (
  26. isource.ProductId
  27. ,isource.LocationId
  28. ,isource.Quantity
  29. ,isource.ModifiedDate
  30. );
  31. select \* from dbo.ProductInventory;
  32. go
  33. rollback tran;
  34. go

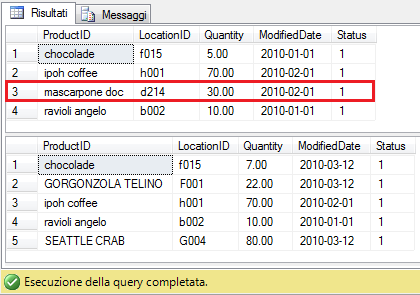
Clausole WHEN MATCHED multiple

Il comando MERGE supporta al più due clausole WHEN MATCHED con l’obbligo di specificare nella prima clausola l’operatore aggiuntivo AND, che opzionalmente può essere specificato anche nella seconda clausola. Quando vengono specificate due clausole WHEN MATCHED, SQL Server esegue l’azione indicata nella prima clausola solo se risultano essere verificate le condizioni specificate nel predicato ON e nel predicato aggiuntivo AND. Se l’espressione logica nel predicato ON risulta essere vera, ma l’espressione nel predicato aggiuntivo AND della prima clausola risulta essere falsa o unknown, SQL Server valuterà la seconda clausola WHEN MATCHED e l’azione indicata nella prima clausola non verrà applicata.

L’utilizzo multiplo delle clausole WHEN MATCHED è di aiuto per implementare il seguente requisito: quando l’inventario rileva un prodotto con giacenza uguale a zero (= non presente a magazzino), la rispettiva riga, nella tabella destinazione, deve essere eliminata. Il seguente comando MERGE implementa anche questo requisito.

* 1. -- MERGE con WHEN MATCHED multipli
  2. begin tran;
  3. go
  4. select \* from dbo.ProductInventory;
  5. go
  6. merge into dbo.ProductInventory as itarget
  7. using dbo.FrequentInventory as isource
  8. on ((itarget.ProductId = isource.ProductId)
  9. and (itarget.LocationId = isource.LocationId))
  10. when matched AND
  11. (isource.Quantity <> 0)
  12. and ((itarget.Quantity <> isource.Quantity)
  13. or (itarget.ModifiedDate <> isource.ModifiedDate)) then
  14. update set
  15. itarget.Quantity = isource.Quantity
  16. ,itarget.ModifiedDate = isource.ModifiedDate
  17. when matched AND
  18. (isource.Quantity = 0) then
  19. delete
  20. when not matched then
  21. insert
  22. (
  23. ProductId
  24. ,LocationId
  25. ,Quantity
  26. ,ModifiedDate
  27. )
  28. values
  29. (
  30. isource.ProductId
  31. ,isource.LocationId
  32. ,isource.Quantity
  33. ,isource.ModifiedDate
  34. );
  35. select \* from dbo.ProductInventory;
  36. go
  37. rollback tran;
  38. go

L’output ottenuto è illustrato in figura 7, in particolare osserviamo che la riga relativa al prodotto “mascarpone doc” è stata eliminata (in dbo.ProductInventory) per effetto dell’azione DELETE specificata nella seconda clausola WHEN MATCHED.

* 1. 
  2. Figura 7 – MERGE con clausole WHEN MATCHED multiple

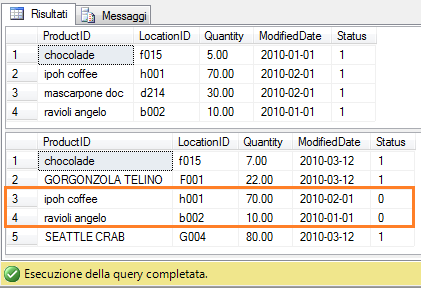
Estensioni proprietarie del linguaggio T-SQL

Il comando MERGE, solo nel linguaggio T-SQL, supporta una terza clausola chiamata WHEN NOT MATCHED [BY SOURCE], che permette di indicare una azione da eseguire quando ad una riga nella tabella destinazione non corrisponde nessuna riga nella tabella sorgente.

Questa terza clausola, WHEN NOT MATCHED [BY SOURCE], ci aiuta ad implementare un altro requisito richiesto nella soluzione: quando una riga nella tabella destinazione non trova corrispondenza nella tabella sorgente aggiornare il campo Status impostando il valore zero. Il seguente frammento di codice T-SQL rappresenta un esempio di utilizzo della clausola WHEN NOT MATCHED [BY SOURCE].

* 1. -- MERGE con terza clausola WHEN NOT MATCHED BY SOURCE
  2. begin tran;
  3. go
  4. select \* from dbo.ProductInventory;
  5. go
  6. merge into dbo.ProductInventory as itarget
  7. using dbo.FrequentInventory as isource
  8. on ((itarget.ProductId = isource.ProductId)
  9. and (itarget.LocationId = isource.LocationId))
  10. when matched AND
  11. (isource.Quantity <> 0)
  12. and ((itarget.Quantity <> isource.Quantity)
  13. or (itarget.ModifiedDate <> isource.ModifiedDate)) then
  14. update set
  15. itarget.Quantity = isource.Quantity
  16. ,itarget.ModifiedDate = isource.ModifiedDate
  17. when matched AND
  18. (isource.Quantity = 0) then
  19. delete
  20. when not matched then
  21. insert
  22. (
  23. ProductId
  24. ,LocationId
  25. ,Quantity
  26. ,ModifiedDate
  27. )
  28. values
  29. (
  30. isource.ProductId
  31. ,isource.LocationId
  32. ,isource.Quantity
  33. ,isource.ModifiedDate
  34. )
  35. when not matched by source then
  36. update set
  37. status = 0;
  38. go
  39. select \* from dbo.ProductInventory;
  40. go
  41. rollback tran;
  42. go

L’output ottenuto è illustrato in figura 8, in particolare osserviamo i valori della colonna Status per i prodotti “ipoh coffee” e “ravioli angelo” non presenti nella tabella dbo.FrequentInventory.

* 1. 
  2. Figura 8 – MERGE con clausola WHEN NOT MATCHED [BY SOURCE]

Conclusioni

Il comando MERGE permette di eseguire, nella destinazione, operazioni DML multiple (INSERT/UPDATE/DELETE) in funzione delle condizioni logiche specificate nelle clausole del comando. Come si è potuto osservare negli esempi, MERGE è un’istruzione molto potente, è quindi necessario prestare attenzione alle condizioni specificate nelle clausole WHEN ed effettuare gli opportuni test prima di utilizzarlo in produzione.

#### di [Sergio Govoni](http://mvp.microsoft.com/profiles/Sergio.Govoni) - Microsoft MVP

Blog: <http://www.ugiss.org/sgovoni/>

Twitter: <https://twitter.com/segovoni>