Il primo comando di ogni Trigger

#### di [Sergio Govoni](http://mvp.microsoft.com/en-us/mvp/Sergio%20Govoni-4029181) - Microsoft MVP

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

Twitter: [@segovoni](https://twitter.com/segovoni)

* 1. Dn269828.7B654F178A3842F7F616A829DC6DF588(it-it,MSDN.10).png

Agosto, 2013

Introduzione

Un oggetto Trigger, di SQL Server, è ottimizzato se la sua durata è breve. I Trigger lavorano sempre in transazione (implicita o esplicita che sia) e i Lock rimarranno al loro posto fino a quando la transazione non verrà confermata (con un COMMIT) o respinta (con un ROLLBACK). Si intuisce facilmente che più lunga sarà la durata di un Trigger, più alta sarà la probabilità che esso blocchi un altro processo.

Ottimizzare un Trigger esistente, che racchiude centinaia di righe di codice, può rivelarsi molto difficile; quindi l’arma migliore è la prevenzione.

Il comando che ogni Trigger dovrebbe avere

Potreste considerarlo banale, ma lo troverete interessante, la prima cosa da fare affinché la durata di un Trigger sia breve è stabilire se il Trigger debba intervenire oppure no. Se non ci sono righe coinvolte nel (precedente) comando che l’ha scatenato, significa **generalmente** che il Trigger in questione non dovrà fare nulla.

L’unica cosa, quindi, che si dovrebbe sempre fare all’interno di un Trigger è verificare il valore della variabile di sistema [@@ROWCOUNT](http://technet.microsoft.com/it-it/library/ms187316.aspx) (o della funzione [ROWCOUNT\_BIG](http://technet.microsoft.com/it-it/library/ms181406.aspx) se il numero di righe è superiore a 2 miliardi) per controllare se nel precedente comando (tipicamente INSERT, UPDATE o DELETE) sono state coinvolte righe. Se ciò non è accaduto (@@ROWCOUNT = 0) non c’è nulla che il Trigger possa fare se non restituire il controllo al chiamate con l’istruzione [RETURN (T-SQL)](http://msdn.microsoft.com/it-it/library/ms174998.aspx).

Il seguente frammento di codice T-SQL dovrebbe quindi comparire all’inizio di ogni Trigger:

* 1. IF (@@ROWCOUNT = 0)
  2. RETURN;

Ecco un esempio di creazione di un Trigger, per il comando UPDATE, sulla tabella Person.Person del database [AdventureWorks2012](http://msftdbprodsamples.codeplex.com/releases/view/55330):

* 1. CREATE TRIGGER TR\_Person\_Person\_Upd ON Person.Person
  2. FOR UPDATE
  3. AS BEGIN
  4. --
  5. IF (@@ROWCOUNT = 0)
  6. RETURN;
  7. -- ...
  8. -- ...
  9. -- ...
  10. END;

La verifica di @@ROWCOUNT permette anche di stabilire se il numero di righe coinvolte è quello che ci si aspetta. In tutti i casi in cui ci si aspetta venga coinvolta solo una riga, dovrà essere eseguito il test:

* 1. IF (@@ROWCOUNT = 1)
  2. <azione>

Dopo aver superato il test riguardante le righe coinvolte, si potrebbe verificare se le colonne di nostro interesse sono state aggiornate (ovviamente solo nel caso di Trigger per il comando UPDATE), utilizzando l’istruzione:

* 1. IF UPDATE(<nome\_colonna>)
  2. <azione>

Nel caso le colonne di nostro interesse non siano state aggiornate, avremmo un’altra occasione per restituire il controllo al chiamante, mantenendo breve la durata del Trigger.

La funzione [UPDATE()](http://technet.microsoft.com/it-it/library/ms187326.aspx) restituisce il valore Booleano True nel caso la colonna sia stata modificata o nel caso di INSERT, ma non permette di sapere, in caso di aggiornamento, se i valori della colonna sono cambiati ovvero se il vecchio valore è diverso dal nuovo. Qualora il Trigger debba eseguire un’azione solo se il vecchio valore, di una colonna, risulti essere diverso dal nuovo, si potranno interrogare le tabelle virtuali INSERTED e DELETED. Il seguente frammento di codice T-SQL implementa un esempio di Trigger, per il comando UPDATE, sulla tabella Sales.SalesOrderDetail del database AdventureWorks2012 dove si desidera eseguire un’azione solo quando viene aggiornata la colonna UnitPrice e il vecchio valore è diverso dal nuovo per almeno una riga.

* 1. CREATE TRIGGER Sales.TR\_SalesOrderDetail\_Upd ON Sales.SalesOrderDetail
  2. AFTER UPDATE AS
  3. BEGIN
  4. IF (@@ROWCOUNT = 0)
  5. RETURN;
  6. SET NOCOUNT ON;
  7. IF UPDATE(UnitPrice)
  8. AND EXISTS (SELECT i.SalesOrderID, i.SalesOrderDetailID
  9. FROM inserted AS i
  10. JOIN deleted AS d ON
  11. (d.SalesOrderID=i.SalesOrderID)
  12. AND (d.SalesOrderDetailID=i.SalesOrderDetailID)
  13. WHERE (d.UnitPrice <> i.UnitPrice))
  14. BEGIN
  15. -- <azione>
  16. END;
  17. END;
  18. GO

Si osservi l’istruzione “SET NOCOUNT ON” eseguita immediatamente dopo aver verificato il valore della variabile @@ROWCOUNT; permette di bloccare l’invio di messaggi al client per ogni istruzione all’interno del Trigger. Nel caso di Trigger contenenti diverse istruzioni e che non restituiscono un’elevata quantità di dati, l'impostazione di SET NOCOUNT su ON può determinare un incremento delle prestazioni significativo grazie alla notevole riduzione del traffico di rete.

Conclusioni

I Trigger sembrano facili da scrivere, ma scrivere Trigger efficienti non è affatto semplice e quando la loro complessità aumenta, talvolta possono presentare effetti collaterali in grado di confondere persino l’autore. L’importante è ricordarsi che un Trigger è ottimizzato se la sua durata è breve.

#### di [Sergio Govoni](http://mvp.microsoft.com/en-us/mvp/Sergio%20Govoni-4029181) - Microsoft MVP

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

Twitter: [@segovoni](https://twitter.com/segovoni)