LEZIONE-TSQL-02 TRIGGER E PROCEDURE (VER03)

T-SQL PROGRAMMAZIONE PARTE SECONDA

ROUTINE E PROGRAMMAZIONE

In TSQL è possibile definire alcune routine da eseguire lato server per svolgere blocchi di codice che controllano o eseguono più comandi di seguito. I modi più diffusi sono i TRIGGER e le PROCEDURE.

TRIGGER

Il Trigger (letteralmente grilletto, innesco) serve per definire un meccanismo automatico sui dati. Quando una determinata operazione viene effettuata sui dati il sistema verifica se esiste un trigger associato e lo esegue.

Questo meccanismo serve per garantire che le operazioni sui dati siano eseguite correttamente, magari correggendo o completando l'operazione richiesta. Purtroppo anche questo comando non è nello standard SQL92, sebbene sia ampiamente implementato in numerosi motori per database.

I trigger sono utilizzati per diversi scopi nella progettazione di un database, e principalmente:

- per mantenere l'integrità referenziale tra le varie tabelle
- per mantenere l'integrità dei dati della singola tabella
- per monitorare i campi di una tabella ed eventualmente generare eventi ad hoc
- per creare tabelle di auditing per i record che vengono modificati o eliminati

Fino alla versione 7.0 di SQL Server, i trigger possono essere attivati solo su tabelle e non su viste. Ma da SQL Server 2005 è possibile attivarli anche su viste.

Un trigger è un tipo speciale di stored procedure che viene eseguita automaticamente quando si verifica un evento nel server di database. MS SQL SERVER prevede tre tipi di TRIGGER: DDL, LOGON e DML.

I trigger DDL vengono eseguiti in risposta a vari eventi DDL (Data Definition Language) e corrispondono principalmente alle istruzioni CREATE, ALTER e DROP su oggetti del database oppure si legano ad alcune procedure di sistema che eseguono operazioni di tipo DDL. I trigger LOGON vengono attivati in risposta all'evento LOGON generato quando viene stabilita una sessione utente.

In questa dispensa analizzeremo solo i trigger DML. I trigger DML vengono eseguiti quando un utente tenta di agire sui dati di una o più tabelle, mediante un evento DML (Data Manipulation Language). Gli eventi che innescano questi trigger sono le istruzioni INSERT, UPDATE o DELETE eseguite su una tabella o una vista. Questi trigger si attivano quando viene generato un evento valido (senza errori di sintassi e senza errori di validazione dei vincoli), indipendentemente dal fatto che esistano o meno righe di tabella interessate.

CREATE TRIGGER

La sintassi semplificata del comando è la seguente:

```
CREATE TRIGGER [ SCHEMA_NAME . ]TRIGGER_NAME
ON { NOME-TABELLA | NOME-VISTA }
{ FOR | AFTER | INSTEAD OF }
{ [ INSERT ] [ , ] [ UPDATE ] [ , ] [ DELETE ] }
AS
BEGIN
{ SQL_STATEMENT  [ ; ] [ ,...N ]
END
```

CREATE TRIGGER: serve per creare il trigger con un suo nome univoco, eventualmente preceduto dal nome del database;

ON: serve per indicare a quale tabella oppure a quale vista agganciare il trigger;

FOR: serve per indicare che deve essere eseguito prima del comando a cui è legato (per esempio FOR INSERT significa prima di un inserimento);

INSTEAD OF: serve per indicare che deve essere eseguito al posto del comando a cui è legato (per esempio INSTEAD OF INSERT significa in sostituzione di un inserimento);

BEFORE: serve per indicare che deve essere eseguito dopo del comando a cui è legato (da TSQL 2012);

INSERT: serve per indicare che deve essere eseguito quando è eseguito il comando INSERT;

UPDATE: serve per indicare che deve essere eseguito quando è eseguito il comando UPDATE;

DELETE: serve per indicare che deve essere eseguito quando è eseguito il comando DELETE;

AS: serve per indicare cosa deve eseguire il trigger;

Le opzioni FOR, BEFORE e INSTEAD OF sono alternative esclusive (se ne usa una e solo una) :

- AFTER, si usa nel caso di voler effettuare delle azioni prima che i controlli sui vincoli della tabella siano stati controllati. È utile per impostare valori di chiave primaria calcolati o per riempire dei campi lasciati vuoti prima che siano effettuati i normali controlli;
- BEFORE, si usa per effettuare azioni dopo che i controlli sono stati superati (e nel caso che l'operazione abbia avuto successo), magari per modificare altre tabelle collegate o per modificare alcuni valori di alcuni campi.

mentre le opzioni DELETE | INSERT | UPDATE sono facoltative (se ne usa una o due o tre) ed indicano il tipo di operazione che invoca l'esecuzione dei comandi.

ESEMPIO 1.

```
CREATE TRIGGER PROVA_STAMPA
ON PRESTITI AFTER INSERT
AS
PRINT 'STAMPA DOPO L''INSERIMENTO'
```

Il trigger si attiva dopo un inserimento sulla tabella prestiti e mostra un banale avviso.

ESEMPIO 2.

```
CREATE TRIGGER CONTROLLA_DATE
ON PRESTITI AFTER INSERT
AS
```

```
IF EXISTS ( SELECT * FROM PRESTITI WHERE DATAPRESTITO > DATARESTITUZIONE )
ROLLBACK
```

Il trigger si attiva dopo un inserimento sulla tabella prestiti. Esso verifica se è stata inserito un prestito la cui data di restituzione è precedente a quella di rilascio del prestito; se si verifica allora disfa (annulla) le operazioni (del trigger ma anche dell'inserimento che lo ha causato).

ESEMPIO 3.

```
CREATE TRIGGER CTRL_COMPOSIZIONE
ON COMPOSIZIONI
FOR INSERT
AS
BEGIN
  DECLARE @IDPRODOTTOGREZZO INT
  DECLARE @IDPRODOTTOFINITO INT
  SELECT @IDPRODOTTOGREZZO = IDPRODOTTOGREZZO
      FROM INSERTED
  SELECT @IDPRODOTTOFINITO = IDPRODOTTOFINITO
      FROM INSERTED
  DECLARE @PERCENTUALE INT
  SELECT @PERCENTUALE = SUM(PERCENTUALE)
      FROM COMPOSIZIONI
      WHERE (@IDPRODOTTOGREZZO = IDPRODOTTOGREZZO)
        AND (@IDPRODOTTOFINITO = IDPRODOTTOFINITO)
  IF @ PERCENTUALE > 100
    ROLLBACK
END
```

Creo un trigger sulla tabella composizioni che indica le percentuali di composizione di un elemento con un altro. Se la somma delle composizioni percentuali supera il 100% allora disfa tutte le operazioni (del trigger e dell'inserimento).

ESEMPIO 4.

```
CREATE TRIGGER G_ISA_ESCLUSIVA
ON UOMINI
FOR INSERT
AS
BEGIN
  DECLARE @IDDIPENDENTE INT
  SET @IDDIPENDENTE = IDDIPENDENTE
    FROM INSERTED
  IF @IDDIPENDENTE IN (SELECT IDDIPENDENTE FROM DONNE)
    ROLLBACK
END
```

Creo un trigger sulla tabella Uomini col seguente corpo: dichiaro una variabile intera (compatibile con la chiave primaria di uomini e di donne, specializzazioni di una gerarchia ISA di Persone); assegno alla variabile il valore della chiave primaria appena inserita; verifico se la chiave è già nelle Donne e, se lo è, disfo (ROLLBACK disfa quanto fatto da INSERT e TRIGGER).

Per poter scrivere correttamente i trigger è necessario conoscere il linguaggio TSQL con maggiori dettagli (variabili, istruzioni e tabelle di sistema, conferma o disfacimento di azioni) che vedremo in seguito.

PROCEDURE

Le STORED PROCEDURE rappresentano il "nucleo" della programmazione Transact SQL. Le procedure figurano già nelle prime versioni di MS SQL Server e sono pacchetti di istruzioni TSQL preparate nel server in maniera analoga alle subroutine dei linguaggi procedurali e memorizzati nella cache del server per i successivi utilizzi.

Racchiudere il codice SQL all'interno di procedure memorizzate apporta due grossi vantaggi rispetto ai file batch di codice SQL tradizionale:

- Aumento nella velocità di esecuzione del codice SQL e quindi delle performance generali delle applicazioni.
- Aumento della leggibilità e della portabilità del codice e quindi della scalabilità delle applicazioni.

Le procedure possono essere create sia per uso permanente che temporaneo ed inoltre possono essere avviate in modo automatico quando viene avviato SQL Server.

Possiamo scrivere istruzioni SQL in una singola procedura fino a 128MB (una enormità); inoltre possiamo dichiarare fino a 1200 parametri per singola procedura.

Le procedure scritte vengono salvate su una tabella di sistema (dal nome syscomments) propria del database sul quale viene costruita.

MS SQL Server stesso possiede una serie di procedure dette di sistema che vengono generate al momento della sua installazione e sono necessarie ad eseguire una serie fondamentale di compiti che vanno dalla creazione dei databases alla loro manutenzione (utenti, permessi, repliche, backup, restore, ecc...). in questa dispensa ci occuperemo delle procedure create da utenti e amministratori.

CREATE PROCEDURE

Come al solito si usa un comando CREATE per creare una procedura. La sintassi semplificata del comando è la seguente:

```
CREATE PROCEDURE NOME_PROCEDURA  (
      @PARAMETRO1 TIPO_DI_DATO1
      ,@PARAMETRO2 TIPO_DI_DATO2

      ,@PARAMETROK TIPO_DI_DATOK )
AS
. . .
-- ALLA FINE DEL COMANDO PUOI USARE GO PER ESEGUIRE LA PROCEDURA IN CONSOLE
```

CREATE PROCEDURE — È il comando di creazione della nuova procedura; è seguito da nome della procedura, rispettando le regole per gli identificatori e che devono essere univoci all'interno dello schema. È consigliabile evitare di utilizzare il prefisso sp_ nel nome della procedura. Questo prefisso viene utilizzato da SQL Server per contrassegnare stored procedure di sistema. Per creare procedure temporanee (locali o globali), utilizzare un simbolo di cancelletto (#) prima del nome (es. #NomeProcedura) per le procedure temporanee locali e due simboli di cancelletto (##) per le procedure temporanee globali (##NomeProcedura).

Il nome completo non deve superare i 128 caratteri. Il nome completo di una stored procedure temporanea locale, incluso il simbolo #, non deve superare i 116 caratteri.

PARAMETRO — è una specie di variabile, che serve per fare comunicare la procedura con l'esterno; il parametro può essere usato per inviare dati alla procedura, per ricevere risultati dalla procedura e per entrambe le cose insieme

AS — inizia il corpo della procedura dove scrivere istruzioni T-SQL

APPROFONDIMENTI: PARAMETRI

Tutti i Transact-SQL tipi di dati possono essere utilizzati come parametri. È possibile utilizzare un tipo tabella definito dall'utente per dichiarare parametri. Parametri con valori di tabella possono essere solo parametri di ingresso e devono essere accompagnati dalla parola chiave READONLY.

Possono essere dichiarati fino a 2100 parametri per una singola procedura. Il valore di ogni parametro dichiarato deve essere fornito dall'utente quando la procedura viene chiamata salvo che sia stato dichiarato un valore predefinito per il parametro con DEFALUT.

ESEMPIO 5. PROCEDURA SENZA PARAMETRI

```
CREATE PROCEDURE PROC1_VUOTA
AS
PRINT 'PROCEDURA VUOTA'
```

La procedura è senza parentesi tonde poiché non ha parametri. Quando viene invocata mostra una frase a video.

ESEMPIO 6. PROCEDURA CON PARAMETRO

```
CREATE PROCEDURE PROC2_PROVA (
      @PAR1 VARCHAR(5)
)
AS
PRINT @PAR1
```

La procedura usa le parentesi tonde per rinchiudere il parametro. Quando viene invocata mostra il valore del parametro a video. Il tipo è obbligatorio.

ESEMPIO 7. VALORE DI DEFAULT

```
CREATE PROCEDURE PROC3_DEFAULT (
      @PAR1 INT = 13
)
AS
PRINT @PAR1
```

La procedura può essere invocata con un valore intero, ma se invoca senza alcun argomento assegna il valore 13 al parametro; in questo modo il parametro ha sempre un valore definito.

ESEMPIO 8. PARAMETRO IN SOLA LETTURA

```
CREATE PROCEDURE PROC4_SOLALETTURA (
      @PAR1 INT READONLY
)
AS
PRINT @PAR1
```

La procedura **deve** essere invocata con un valore intero, altrimenti genera un errore; tuttavia il parametro non può essere modificato all'interno del corpo, oppure si solleva un errore sintattico; la procedura infine stampa il valore passato come argomento.

ESEMPIO 9. PARAMETRO OUTPUT

```
CREATE PROCEDURE PROC3_DEFAULT (
      @PAR1 INT OUT
)
AS
@PAR1 = 13
PRINT @PAR1
```

La procedura **deve** essere invocata con un valore intero, altrimenti genera un errore; tuttavia il parametro può essere modificato all'interno del corpo e reso come risultato al chiamante.

ESEMPIO 10. PARAMETRI VARI

```
CREATE PROCEDURE PROC3_DEFAULT (
      @PAR1 INT = 17
      , @PAR2 INT OUT
)
AS
DECLARE @TMP
@TMP = @PAR1
@PAR1 =
PRINT @PAR1
```

La procedura può essere invocata con uno o due parametri, altrimenti genera un errore; se non si specifica il primo valore diventa 17 in automatico; dopo il corpo scambio i valori contenuti nei due parametri.

CONCLUSIONI

Per programmare correttamente i TRIGGER e le PROCEDURE occorre studiare il linguaggio T_SQL più approfonditamente

ESERCIZI

ESERCIZIO i)

Si vuole controllare l'inserimento dei voti nella tabella Verifiche; e il voto inserito è inferiore a 1 allora si inserisce 1; se è superiore a 10 si inserisce 10; usare un TRIGGER

ESERCIZIO ii)

La tabella Versamenti registra i KG versati da ciascun camion in una discarica ogni giorno; ma se lo stesso camion versa più di una volta nello stesso giorno allora si cumulano i KG, invece di registrare più versamenti distinti; risolvere il problema con un TRIGGER ma anche con una PROCEDURE.

ESERCIZIO iii)

La tabella Prestiti (IDPrestito, Data, Scadenza, Restituzione, IDSocio, IDTesto) registra prestiti di libri; se si inserisce un prestito e non si specifica la data usare quella odierna e impostare la scadenza dopo 30 giorni

Argomenti Trattati: