TUTORIAL

CREAZIONE DI TRIGGER IN ORACLE LITE 8i MEDIANTE CLASSI JAVA

Michele de Nittis

Generalità

Esistono due modi per realizzare un *trigger* in Oracle Lite:
1) Usando il modello di *Caricamento e Pubblicazione* proprio dello sviluppo di una stored procedure;
2) Usando il modello di *Per Allegato* (Attach) di una stored procedure ad una relazione.

Nel seguito del tutorial si farà riferimento ai *trigger* ed alle *stored procedures* indistintamente con il termine 'procedura'.

Bisogna distinguere due tipi di procedure: quelle che intervengono a livello di tabella (*table-level*) e quelle che intervengono a livello di riga (*row-level*). Le prime eseguono il proprio compito su tutta la tabella di applicazione, cioè su tutte le sue righe, mentre le altre eseguono il proprio compito su una sola riga.
Tra le procedure che agiscono a livello di tabella vi sono, ad esempio, quelle che controllano la consistenza dei dati, il rispetto di vincoli di integrità referenziale, il rispetto delle *business rules* ed il calcolo di valori statistici. Tra le procedure che agiscono a livello di riga sono vi sono, ad esempio, quelle che controllano la consistenza dei dati di una singola riga, che calcolano gli attributi derivati, che controllano il rispetto di vincoli tra i valori degli attributi di una stessa riga o record.

A livello pratico, le procedure vengono realizzate come metodi di una classe java ed in particolare quelle che agiscono a livello di tabella vengono realizzate mediante **metodi statici**. Questa particolarità è motivata dal fatto che, mentre per una procedura che agisce a livello di riga viene istanziato un oggetto per ogni riga della tabella interessata da una determinata operazione, per una procedura che agisce a livello di tabella ciò non avviene per mancanza di una specifica riga di applicazione.

Modello di Caricamento e Pubblicazione

Questo modello consente di creare *stored procedures* e *triggers* usando esclusivamente **metodi statici** (dunque solo procedure table-level).

Per creare una procedura con questo modello si deve:
1) Scrivere una classe java con i metodi deputati a realizzare le procedure per la tabella, e compilarla;
2) Caricare la classe Java in OracleLite. Questa operazione può essere fatta attraverso lo speciale comando *loadjava*, eseguibile da linea di comando DOS, o attraverso lo speciale statement SQL "*CREATE JAVA*" da SQLPlus. Nel seguito si approfondirà solo quest'ultima possibilità.
3) Pubblicare la classe java come *SQL Procedure* attraverso lo speciale statement SQL "*CREATE PROCEDURE*";
4) Creare un Trigger, specificando la procedura pubblicata da impiegare ed il tipo di evento che ne provoca l'innesco (*firing)*, usando lo speciale statement SQL "*CREATE TRIGGER*".

La sintassi completa dello statement CREATE JAVA è la seguente:

```
CREATE [OR REPLACE] [AND RESOLVE] [NOFORCE] JAVA
       { CLASS
       [SCHEMA <schema_name>] |
       [RESOURCED NAMED [<schema_name>.]<primary_name>
       }
[<invoker_rights_clause>]
[RESOLVER <resolver_spec>]
USING BFILE ('<dir_path>', '<class_name>');
```

La sintassi è molto articolata e complessa ma per gli scopi di questo tutorial, cioè la creazione di trigger, è sufficiente conoscere solo la parte evidenziata in neretto, come mostra il seguente esempio.

Esempio:

```
CREATE OR REPLACE JAVA CLASS USING
BFILE('D:\ORAWIN95\OLDB40\TESINA_1', 'Idiot.class');
```

La sintassi completa per lo statement CREATE PROCEDURE è la seguente:

```
CREATE [OR REPLACE]
{ PROCEDURE [<schema_name>.]<proc_name> [([<sql_params>])] |
   FUNCTION  [<schema_name>.]<proc_name> [([<sql_params>])]
RETURN <sql_type> }
<invoker_rights_clause>
{IS | AS} LANGUAGE JAVA NAME
```

```
'<java_fullname> ([<java_params>]) [return
<java_type_fullname>]';
/
```

Questo comando, la cui sintassi è molto articolata, consente di "*mappare*" o "*pubblicare*" un metodo della classe java precedentemente caricata come una *funzione* o *procedura* di OracleLite. La differenza tra funzione (FUNCTION) e procedura (PROCEDURE) consiste nel fatto che la prima restituisce un valore.
Quando si esegue questa *mappatura* si devono specificare le corrispondenze tra i nomi simbolici della procedura e del metodo che la implementa, tra i relativi parametri formali e tra gli eventuali valori restituiti.
Per gli scopi di questo tutorial, cioè la crezione di trigger in OracleLite, non si esaminerà il caso di definizone di funzioni, peraltro concettualmente identico al caso di definizione di procedure.
Si considererà, pertanto, solo la sintassi evidenziata in neretto.
La clausola opzionale OR REPLACE del comando CREATE PROCEDURE specifica ad OracleLite di rimpiazzare o di rigenerare la procedura di ugual nome eventualmente già pubblicata in precedenza.
<schema_name> è il nome dello schema della base di dati nella quale definire la procedura. Se non specificata, OracleLite assume lo schema corrente, cioè quello dell'utente che lancia il comando CREATE.
<proc_name> è il nome della procedura.
<sql_params> è l'elenco dei parametri formali della procedura. Ogni parametro formale deve essere dichiarato con questa sintassi:

<sql_param> = <param_name> [IN | OUT | IN OUT] <sql_datatype>.

Le clausole opzionali IN, OUT, IN OUT vengono introdotte allorquando un procedura debba modificare alcuni dei suoi parametri attuali. Non entreremo in ulteriori dettagli perchè esulano dallo scopo del presente tutorial.

<sql_datatype> rappresenta uno dei tipi di dato SQL supportati da OracleLite.

<java_fullname> è il nome completo (cioè comprensivo del nome della classe di definizione) del metodo della classe java caricata che deve implementare la procedura.
<java_params> è l'elenco dei **tipi java** dei parametri formali del metodo da eseguire quando viene invocata la procedura.
Il comando CREATE deve essere terminato dal carattere *slash* (/).

Esempio:

```
CREATE OR REPLACE PROCEDURE COUNT_IDIOT_TRIGGER AS LANGUAGE JAVA
NAME 'Idiot.Count_Idiot_Trigger(java.sql.Connection)';
/
```

ATTENZIONE
Vi deve essere piena corrispondenza, nel numero e nel tipo, tra i parametri della procedura SQL e quelli del metodo java, con la sola eccezione dell'eventuale parametro java.sql.Connection.

Infatti, quando il metodo java che implementa una procedura deve accedere ad una o più tabelle del database, lo fa impiegando JDBC. Per questo OracleLite passa automaticamente al metodo, come primo argomento, il riferimento ad un oggetto Connection.
Per quanto riguarda la corrispondenza tra i tipi SQL e quelli Java, si faccia riferimento alla tabella in appendice.

La sintassi completa per il comando CREATE TRIGGER è la seguente:

```
CREATE [OR REPLACE] TRIGGER <trigger_name>
      {BEFORE | AFTER} [{INSERT | DELETE | UPDATE}  [OF
<column_list>] [OR]]
      ON  <table_reference>
[FOR EACH ROW] <procedure_reference>(<argument_list>)
```

Oracle Lite consente di definire due tipi di trigger:
- *statement - level;*
- *row-level.*

Un trigger *row-level* viene lanciato per ogni riga affetta dalla modifica di una tabella del database (ad esempio UPDATE <tabella> SET *{<field>=<value>} WHERE <condizione>), mentre un trigger *statement-level* viene lanciato una volta sola, indipendentemente dal numero di righe affette dalla modifica.

I trigger eseguiti sugli eventi BEFORE INSERT ed AFTER DELETE, che non coinvolgono una riga in particolare ma in generale una tabella, devono necessariamente basarsi su procedure di tipo *table-level* (metodi java statici). Gli eventi UPDATE, AFTER INSERT e BEFORE DELETE possono agire su una particolare riga di una tabella (quella modificata o da modificare, quella appena inserita, quella da eliminare) e quindi possono basarsi sia su procedure *row-level*, sia su procedure *table-level*.

Attenzione a non confondere i concetti di procedura *table-level/row-level* e di trigger *statement-level/row-level: table-level/row-level* si riferisce all'ambito di applicazione di una procedura (singola riga, intera tabella), *statement-leve/row-level* si riferisce alla politica di innesco (*trig*) del trigger (una volta per ogni riga affetta dalla modifica apportata alla tabella, una volta per tutta la tabella interessata dalla modifica, ovvero per lo statement).

I trigger di tipo *statement-level* non possono avere argomenti. Il sistema reagisce ad ogni tentativo di creare trigger di questo tipo con argomenti stampando questo messaggio di errore (SQLPlus):

ERRORE alla riga 1:
OCA-30021: error preparing/executing SQL statement
[POL-8060] Statement level triggers cannot have arguments

Ogni volta che viene innescato un trigger *row-level*, il sistema istanzia due oggetti, secondo il tipo di evento scatenante: *new* per INSERT ed UPDATE che rappresenta il nuovo record, *old* per UPDATE e DELETE che rappresenta il vecchio record. Le proprietà di questi due oggetti (cioè gli attributi dei record che rappresentano) sono estremamente utili per il passaggio di argomenti ai trigger (vedi esempi sottostanti).

Molte delle clausole dello statement CREATE TRIGGER sono note o facilmente intuibili.
La clausola [OR REPLACE] ha lo stesso significato che ha nello statement CREATE PROCEDURE.
<trigger_name> è il nome del trigger;
La clausola opzionale OF <column_list> della clausola UPDATE è necessaria per avviare l'esecuzione del trigger in seguito all'aggiornamento dell'insieme di colonne specificato in <column_list>.
La clausola OR serve per specificare più eventi capaci di causare l'esecuzione del trigger.
<table_reference> è il nome della tabella su cui viene applicato il trigger.
La clausola opzionale FOR EACH ROW deve essere usata esclusivamente per creare trigger di tipo *row-level* (vedi esempi).
<procedure_reference> è il nome attribuito alla procedura precedentemente pubblicata con il comando CREATE PROCEDURE (vedi <proc_name>).
<argument list> è la lista degli argomenti da passare alla procedura.

Se si usa SQLPlus per definire il trigger, si deve terminare il comando CREATE TRIGGER con i caratteri (.) e (/) posti su righe diverse.

Esempio:

```
CREATE OR REPLACE TRIGGER CountIdiotTrigger BEFORE INSERT ON
Intervento COUNT_IDIOT_TRIGGER();
.
/
```

Esempio di creazione di un trigger con il modello "Caricamento e Pubblicazione"

1) Creazione di una classe Java

```
public class trig_Asserve {
/** funzione statica che controlla se sistem_master è un PC già
caricato nel sistema */
public static void TRG_CK_MasterSlave(Connection conn, String
Master, String Slave) throws SQLException {

 Statement stmnt = null; // inizializzazione dello statement
 ResultSet rset = null; // inizializzazione resultset
 String MySql = null; // inizializzaione della stringa SQL

 // Verifica che Master != Slave

 if(Master.equals(Slave)==true) {
      throw new SQLException("Un sistema non può asservire se
stesso.");
 }

 try{
      stmnt = conn.createStatement();
 } catch(SQLException SQLE) {
      throw new SQLException("difficoltà nella creazione
dell'oggetto Statement.");
      }


 // composizione della query SQL-1: Master è un PC
 MySql = "SELECT * FROM Sistema S WHERE (S.tipo='PC' AND
S.serial_n='" + Master +"')";
 System.out.println(MySql);

 try{
      rset = stmnt.executeQuery(MySql);
 } catch(SQLException SQLE){
 throw new SQLException("difficoltà nell' esecuzione della prima
query.");
 }

 // controllo dell'esistenza del record
 if(rset.next()==false){
      throw new SQLException("Violazione del vincolo sul record
master.");
 }


 // composizione della query SQL-2: Slave non è un PC
```

```
 MySql = "SELECT * FROM Sistema S WHERE (S.tipo !='PC' AND
S.serial_n='" + Slave +"')";
 System.out.println(MySql);
 try{
      rset = stmnt.executeQuery(MySql);
 } catch(SQLException SQLE){
 throw new SQLException("difficoltà nell' esecuzione della
seconda query.");
 }

 // controllo dell'esistenza del record
 if(rset.next()==false){
      throw new SQLException("Violazione del vincolo sul record
slave.");
 }

 rset.close();
 stmnt.close();
} // fine TRG_CK_MasterSlave
} // fine trig_Asserve
```

2) Caricamento della classe Java:

```
CREATE OR REPLACE JAVA CLASS USING
BFILE('D:\ORAWIN95\OLDB40\TESINA_1', 'trig_Asserve.class');
```

3) Pubblicazione della procedura:

```
CREATE OR REPLACE PROCEDURE TRG_CK_MASTERSLAVE (MASTER VARCHAR,
SLAVE VARCHAR) AS LANGUAGE JAVA NAME
'trig_Asserve.TRG_CK_MasterSlave(java.sql.Connection,
java.lang.String, java.lang.String)';
/
```

4) Creazione del Trigger:

```
CREATE OR REPLACE TRIGGER TRG_CK_MASTERSLAVE BEFORE INSERT ON
Asserve FOR EACH ROW TRG_CK_MASTERSLAVE(new.sistema_master,
new.sistema_slave);
.
/
```

Modello Per Allegato (Attachment)

Questo modello è tipico di Oracle Lite ed è stato introdotto per superare la limitazione del modello precedente con il quale è possibile creare procedure esclusivamente mediante metodi java **statici**, e quindi solo procedure *table-level*. Con questo modello, inoltre, è possibile fare riferimento direttamente al metodo java, senza definire una *stored procedure* (pubblicazione) che lo mascheri, avendo cura di prestare particolare attenzione alla corrispondenza tra i tipi di dato Java ed i tipi di dato SQL perchè Oracle Lite non garantisce, per questo modello, alcun meccanismo automatico di conversione (*Java-to-SQL Type Conversion*).

Per creare una procedura con questo modello si devono seguire i seguenti passi:

1) Scrivere una classe java con tutti i suoi metodi, alcuni dei quali diverranno le procedure per il database, e compilarla;

2) Allegare uno o più metodi (statici e non) della predetta classe java alla tabella con lo statement SQL *ATTACH JAVA CLASS*;

Esempio:
```
alter table Intervento ATTACH JAVA CLASS "Idiot" IN
'c:\oraclelite\orant\oldb40\Tesina_1';
```

3) Creare il trigger con lo statement SQL *CREATE TRIGGER*.

La sintassi completa dello statement ATTACH JAVA CLASS è la seguente:

```
ALTER TABLE [<schema>.]<table>
      ATTACH JAVA {CLASS | SOURCE} "<class_or_source_name>"
      IN {DATABASE | '<class_or_source_path>'}
      [WITH CONSTRUCTOR ARGS (<col_name_list>)]
```

Il significato di alcune clausole dello statement è immediato e pertanto non verrà illustrato.
Le clausole {CLASS | SOURCE} servono per specificare se la classe che viene allegata alla tabella <table> in forma di bytecode o di sorgente rispettivamente. Nel secondo caso Oracle Lite provvede a lanciare il compilatore java disponibile (il path del compilatore java deve tovarsi nella variabile ambiente PATH).
<class_or_source_name> è il nome della classe (.class) o del sorgente java (.java) senza l'estensione. Questo nome deve obbligatoriamente essere racchiuso tra doppi apici (") se contiene caratteri minuscoli.
Le clausole {DATABASE | '<class_or_source_path>'} servono per indicare ad Oracle Lite il path della classe (o del file sorgente) specificato nella clausola precedente. Se esso

è già disponibile per effetto di uno statement ATTACH JAVA eseguito precedentemente, si deve specificare DATABASE.
La clausola WITH CONSTRUCTOR ARGS serve nel caso di procedure di tipo *row-level* per invocare un costruttore diverso da quello senza argomenti (invocato per default da Oracle Lite) al momento di istanziare l'oggetto della classe allegata. <col_name_list> è la lista di argomenti da passare al costruttore.

Esempio:
```
ALTER TABLE Intervento ATTACH JAVA CLASS "Idiot" IN
'c:\oraclelite\orant\oldb40\Tesina_1';
```

Lo statement CREATE TRIGGER è stato già illustrato nella sezione precedente. Si precisa che, in questo caso, <procedure_reference> rappresenta il nome del metodo della classe allegata da invocare, specificando anche il nome della tabella di applicazione nel caso di metodo statico.

Esempio:
```
CREATE TRIGGER Idiot_Trigger BEFORE INSERT ON Intervento FOR EACH
ROW Intervento."Idiot_Trigger"();
```

Esempio di creazione di un trigger con il modello per "Allegato"

1) Creazione della Classe Java

```
public class trig_composizione{
 /** Questo metodo statico implementa un trigger che verifica che
  * l'identificativo del sistema corrisponda ad un PC.
  * L'associazione Composizione, infatti, correla un PC con i
componenti
  * interni.
  */
 public static void TRG_CK_PC(Connection conn, String Sistema)
throws SQLException {

 String MySql = null;
 Statement Stmnt = null;
 ResultSet MyResSet = null;

 try{
      Stmnt = conn.createStatement();
 } catch (SQLException SQLE) {
      throw new SQLException("difficoltà nella creazione dello
statement");
 }

 MySql="SELECT * FROM Sistema WHERE serial_n='" + Sistema + "'
AND tipo = 'PC'";
 System.out.println(MySql);

 try{
      MyResSet = Stmnt.executeQuery(MySql);
 } catch (SQLException SQLE) {
      throw new SQLException("difficoltà nell'esecuzione della
query");
 }

 if(MyResSet.next()==false){
      throw new SQLException("Violato il vincolo: il sistema non
è un PC.");
 }
 // chiusura
 MyResSet.close();
 Stmnt.close();
 } // fine trigger
```

2) Allegamento della classe trig_composizione alla tabella Composizione

```
ALTER TABLE Composizione ATTACH JAVA CLASS "trig_composizione" IN
'd:\orawin95\oldb40\Tesina_1';
```

3) Creazione del trigger TRG_CK_PC

```
CREATE OR REPLACE TRIGGER TRG_CK_PC BEFORE INSERT ON Composizione
FOR EACH ROW Composizione."TRG_CK_PC"(new.sistema);
.
/
```

dove la tabella Composizione ha la seguente definizione

```
CREATE TABLE Composizione (
 sistema varchar(50) UNIQUE REFERENCES Sistema(serial_n),
 componenti integer REFERENCES Componenti(cod_comp),
 CONSTRAINT FK_COPOSIZIONE PRIMARY KEY(sistema, componenti)
);
```

Rimozione delle procedure e dei trigger

Per disabilitare un trigger senza rimuoverlo si deve impiegare il comando ALTER TRIGGER con la clausola DISABLE

```
ALTER TRIGGER <trigger_name> {ENABLE | DISABLE}
```

Per disabilitare tutti i trigger che agiscono su una data tabella si deve impiegare la clausola DISABLE ALL TRIGGERS dello statement ALTER TABLE

```
ALTER TABLE <table_name>  {ENABLE | DISABLE} ALL TRIGGERS
```

Per rimuovere un trigger si deve usare il comando DROP TRIGGER

```
DROP TRIGGER [<schema_name>.]<trigger_name>
```

Per rimuovere da una tabella una classe precedentemente allegata, si deve utilizzare la clausola DETACH dello statement ALTER TABLE. Questo comando, tuttavia, non elimina la classe dal database. Per far ciò si deve impiegarre anche la clausola opzionale AND DELETE.

```
ALTER TABLE [<schema_name>.]<table_name> DETACH [AND DELETE] JAVA
CLASS "<java_class_name>"
```

Per rimuovere una procedura precedentemente pubblicata si deve usare il seguente comando:

```
DROP {FUNCTION | PROCEDURE} [<schema_name>.][<proc_name>]
```

Per eliminare una classe dal database, in generale, si deve usare il comando DROP JAVA che ha la seguente sintassi:

```
DROP JAVA {CLASS | RESOURCE} "<class_name>"
```

dove <class_name> è il nome della classe senza estensioni.

Attenzione:
Ogni volta che si apportano delle modifiche al codice di un metodo che implementa una procedura, si deve rimuovere la corrispondente classe java, precedentemente caricata (DROP JAVA), e ricaricarla di nuovo. Solo in questo modo saranno visibili le modifiche apportate.

Esempio:

```
prompt rimozione Trigger TRG_CK_SWPC;
DROP TRIGGER TR_CK_SWPC;
DROP PROCEDURE TRG_CK_SWPC;
DROP JAVA CLASS "Trig_Installazione";
```

```
Esempio:
prompt rimozione Trigger TRG_CK_Esito;
DROP TRIGGER TR_CK_ESITO;
ALTER TABLE Termine DETACH AND DELETE JAVA CLASS "Trig_Termine";
```

Tipi di dati in Java	Tipo di Dati in Oracle
Byte[], byte[]	BINARY, RAW, VARBINARY
Boolean, boolean	BIT
String, String[]	CHAR, VARCHAR, VARCHAR2
Short, Short[], short, short[]	SMALLINT
Integer,Integre[], int, int[]	INT
Float, Float[], float, float[]	REAL
Double, Double[], double, double[]	DOUBLE, REAL (senza precisione)
BigDecimal, BigDecimal[]	NUMBER(n)
java.sql.Date, java.sql.Date[]	DATE
java.sql.Time, java.sql.Time[]	TIME
java.sql.Timestamp, java.sql.Timestamp[]	TIMESTAMP
java.sql.Connection	JDBC CONNECTION