Triggers
Introduction à PL-SQL

Céline Rouveirol

d'après les transparents de Jeff Ullman
http://infolab.stanford.edu/~ullman/

Triggers: Motivation

- Les vérifications de contraintes (attribut ou n-uplet) ont des possibilités limitées
- Un déclencheur (“trigger” en anglais) permet à l'utilisateur de spécifier **quand** effectuer la vérification.
- Il est possible d'associer à un trigger une condition et de lui faire exécuter une séquence d'instructions SQL

Trigger : principe

- A chaque trigger, on associe
  - *un évènement* : un évènement sur la base de données,
    - `Exemple: insert on Vend`
  - *une condition* : toute expression booléenne qui s'évalue à vrai ou faux
  - *Action* : un ensemble d'instructions SQL

CREATE TRIGGER

- CREATE TRIGGER <nom>
- Aussi:

CREATE OR REPLACE TRIGGER <nom>

  - Utile s'il existe un trigger de ce nom que vous voulez modifier.

Exemple de trigger

- Au lieu d'utiliser une contrainte de clé étrangère et de rejeter les insertions dans `Vend(bar, bière, prix)` avec des bières inconnues (dans la Relation `Bières`), un trigger peut ajouter cette bière dans `Bières`, avec une couleur et une origine à `NULL`.

Exemple: Définition d'un trigger

```
CREATE TRIGGER BièreTrig
  AFTER INSERT ON Vend
  REFERENCING NEW AS NTuple
  FOR EACH ROW
  WHEN (NTuple.bière NOT IN
   (SELECT nom FROM Bières))
  BEGIN
   INSERT INTO Bières(nom)
      VALUES(:NTuple.bière);
  END;
```

L'évènement

La condition

L'action

Triggers: partie condition

- AFTER ou BEFORE.
  - INSTEAD OF, si la relation est une vue.
    - Une façon de modifier une vue : avoir un trigger qui effectue les modifications appropriées sur les tables de base
- INSERT, DELETE, UPDATE, UPDATE ON <nom d'attributs>

Triggers: FOR EACH ROW

- Les triggers sont soit *row-level* ou *statement-level*.
- FOR EACH ROW précise que le trigger est row-level; son absence indique que le trigger est “statement-level”
- Les triggers “row level” sont exécutés à chaque modification de nuplets, pour chaque n-uplet modifié.
- Les triggers “statement-level” sont exécutés une fois pour toute à chaque modification, quelque soit le nombre de nuplets modifiés.

Trigger statement level: exemple

```
CREATE OR REPLACE TRIGGER emp_alert_trig
  BEFORE INSERT ON employé
BEGIN
  DBMS_OUTPUT.PUT_LINE('Ajout employé(s)');
END;
```

```
INSERT INTO employé (emp_id, emp_nom, dept_id)
SELECT EmpId + 1000, , emp_nom, 40    FROM employé_tempo
WHERE emp_id BETWEEN 7900 AND 7910;
```

Supposons que la table employé soit vide et que la sous-requête sélectionne 3 employés dans la table employé_tempo. Après le insert précédent (3 nuplets):

Affichage de `'Ajoute employé(s)'` (1 seule fois)

Trigger statement level : exemple

```
CREATE TABLE empauditlog (
    audit_date      DATE,
    audit_user      VARCHAR2(20),
    audit_desc      VARCHAR2(20));
```

Trigger statement level

```
CREATE OR REPLACE TRIGGER emp_audit_trig
   AFTER INSERT OR UPDATE OR DELETE ON employé
DECLARE
   v_action       VARCHAR2(20);
BEGIN
   IF INSERTING THEN
      v_action := ’Employe(s) ajouté(s)';
   ELSIF UPDATING THEN
      v_action := ’Employe(s) mis à jour';
   ELSIF DELETING THEN
      v_action := ’Employe(s) effacés';
   END IF;
   INSERT INTO empauditlog VALUES (SYSDATE, USER,
      v_action);
END;
```

Triggers: REFERENCING

- Les expressions `INSERT` impliquent un nouveau nuplet (row-level) ou un nouvel ensemble de nuplets (statement-level).
- `DELETE` implique un ancien nuplet ou table.
- `UPDATE` implique les deux
- Pour les référencer, on utilise

`[NEW|OLD][TUPLE|TABLE] AS` <nom>

Trigger: la partie Condition

- Une condition booléenne quelconque (attention, sous Oracle, pas de sous-requêtes)
- La condition est évaluée avant ou après l'évènement déclencheur, selon que BEFORE ou AFTER apparaisse dans l'événement
- Accède aux nouveaux/anciens nuplets ou ensembles de nuplets à travers les noms associés à OLD et NEW dans la partie REFERENCING.

Trigger: la partie Action

- Il peut y avoir plus d'une instruction SQL dans la partie action.
  - Encadrés par BEGIN . . . END
- Des requêtes n'ont vraiment aucun sens dans une action, il faut donc effectuer des *modifications*.

Un autre exemple

- On utilise la relation Vend(bar, biere, prix) et une relation unaire CheroBars(bar) qui maintient la liste des bars qui ont augmenté au moins une de leur bière de plus de 1 euro.

Exemple Trigger

```
CREATE TRIGGER PrixTrig
  AFTER UPDATE OF prix ON Vend
  REFERENCING
   OLD as o
   NEW as n
  FOR EACH ROW
  WHEN(n.prix > o.prix + 1.00)
  BEGIN
  INSERT INTO CheroBars
   VALUES(:n.bar);
  END;
```

La mise à jour de Vend.prix

On référence ici les nuplets avant et après mise à jour

pour chaque n-uplet mis à jour

Condition

Quand le prix a augmenté significativement, on ajoute à CheroBars

Triggers sur des vues

- Il est impossible de modifier une vue, car elle n'existe pas, les tables sous-jacentes sont modifées.
- Les triggers INSTEAD OF permettent d'effectuer des modifications lorsque la vue est non modifiable.
- Exemple: on définit une vue TripletGagnant de schéma (client, biere, bar) tel que le bar sert une bière, le client fréquente le bar et apprécie la biere.

Exemple

Prend une copie de chaque att.

```
CREATE VIEW TripletGagnant AS
  SELECT Frequente.Client, Apprecie.biere,
         Vend.bar
  FROM Apprecie, Vend, Frequente
  WHERE Apprecie.client = Fréquente.client
   AND Apprecie.biere = Vend.biere
   AND Vend.bar = Frequente.bar;
```

Jointure naturelle de Apprecie Vend, and Fréquente

Insertion dans une vue

- Il n'est pas possible d'insérer dans TripletGagnant --- c'est une vue.

- Pour savoir quels champs d'une vue sont modifiables :

```
SELECT column_name, updatable
   FROM user_updatable_columns
  WHERE table_name ='TripletGagnant';
```

- Aucun pour TripletGagnant...

Insertion dans une vue

- On peut utiliser un trigger `INSTEAD OF` pour remplacer une insertion d'un triplet `(Client, Biere, Bar)` dans la vue par trois insertions de paires, dans `Apprecie`, `Vend` et `Fréquente`.
- `Vend.prix` sera à `NULL`.

Exemple trigger instead of

```
CREATE TRIGGER ViewTrig
   INSTEAD OF INSERT ON TripletGagnant
   REFERENCING NEW AS n
   FOR EACH ROW
   BEGIN
    INSERT INTO Apprecie VALUES(:n.client, :n.biere);
    INSERT INTO Vend (bar, biere)
   VALUES(:n.bar,:n.biere);
    INSERT INTO Frequente VALUES(:n.client,:n.bar);
   END;
```

Contraintes et trucs pour les triggers

- Ne pas oublier à la fin de la définition d'un trigger : un point (.) et le mot clé `run`; Ceci permet au trigger d'être installé dans la base de données (et non exécuté)
- Dans la partie actions, mais pas dans la partie *conditions*, il faut préfixer “new,” etc., par :

Contraintes et trucs pour les triggers

- Quand on pose un trigger sur un évènement de la relation *R:*
  - *R* ne doit subir aucune modification dans l'action
  - Aucune relation reliée à *R* par une chaîne de contraintes étrangères ne doit être modifiée
- Récupérer le message d'erreur du trigger si message d'erreur: Déclencheur créé avec erreurs de compilation.
  - `show errors trigger <nom_trigger>;`
- Visualiser les triggers définis
  - `select trigger_name from user_triggers;`

Derniers trucs sur les triggers

- Pour plus de détails sur un trigger en particulier:
  - `select trigger_type, triggering_event, table_name, referencing_names, trigger_body from user_triggers where trigger_name = '<nom_trigger>';`
- Effacer des Triggers: `drop trigger <nom_trigger_name>;`
- Pour rendre un trigger inactif : `alter trigger <nom_trigger> {disable| enable};`

PL/SQL

- PL/SQL veut dire Procedural Language/SQL. PL/SQL étend SQL en ajoutant des constructions disponibles dans des langages procéduraux
- L'unité de base en PL/SQL est un bloc. Tous les programmes PL/SQL sont faits de blocs. Un bloc effectue une action logique dans un programme.

Bloc en PL/SQL

```
DECLARE
  <déclarations>
BEGIN
  <instructions>
END;
.
run
```

- La section `DECLARE` est optionnelle.

Syntaxe d'une procédure PL/SQL

```
CREATE OR REPLACE PROCEDURE
  <nom> (<arguments>) AS
  <déclarations optionnelles>
  BEGIN
   <PL/SQL statements>
  END;
```

AS est obligatoire ici

```
.
run
```

Obligatoire pour stocker la procédure dans la base de données

Déclarations et affectations en PL/SQL

- Le mot-clé `DECLARE` n'apparaît pas en tête des déclarations locales, qui sont juste formées des noms de variables et de leur types

- Affectation
  - Example: x := y;

Déclarations et affectations en PL/SQL

- Une variable peut avoir pour type
  - classique en SQL (VARCHAR, INTEGER, ...)
  - Un type générique en SQL: NUMBER
  - Du même type qu'un attribut d'une table de la base de données
  - Du même type qu'un nuplet dans une table

```
DECLARE
    prix  NUMBER;
    maBiere VARCHAR(20);
    maBiere Biere.nom%TYPE;
    biereNuplet Biere%ROWTYPE;
```

Exemple:Le menu de Joe

- Soit la procédure JoeMenu(b,p) qui ajoute la bière *b* au prix *p* à toutes les bières vendues par Joe dans la relation Vend.

- CREATE OR REPLACE PROCEDURE JoeMenu (
  b IN Vend.biere%TYPE,
  p IN Vend.prix%TYPE) AS

  BEGIN
  INSERT INTO Vend VALUES (’Joe”s Bar’, b, p);
  END;

.

Instructions de branchement en PL/SQL

- IF <condition> THEN <liste-instructions> ELSE <liste-instructions> END IF;
- La partie ELSE est optionnelle.

```
DECLARE a NUMBER; b NUMBER;
BEGIN
   SELECT e,f INTO a,b FROM T1 WHERE e>1;
   IF b=1 THEN
      INSERT INTO T1 VALUES(b,a);
   ELSE
   INSERT INTO T1 VALUES(b+10,a+10);
   END IF;
END;
```

Boucles PL/SQL

- `LOOP ... END LOOP.`
- `EXIT WHEN` <condition>
- Condition: le curseur cursor *c* n'a trouvé aucun nuplet : `c%NOTFOUND`

```
DECLARE
   i NUMBER := 1;
BEGIN
   LOOP
      INSERT INTO T1 VALUES (i,i);
      i := i+1;
      EXIT WHEN i>100;
   END LOOP;
END;
```

Curseurs en PL/SQL

- Déclaration de la forme :

```
CURSOR <name> IS <query>;
```

- Pour déplacer un curseur c,
  FETCH c INTO <variable(s)>;

Exemple de curseur

- JoeAugmente() envoie un curseur sur les nuplets de Joe dans Vend et augmente de 1 le prix de chaque bière que Joe vend, si ce prix est initialement de moins de 3.

JoeAugmente(): Déclarations

```
CREATE OR REPLACE PROCEDURE
   JoeAugmente() AS
  LaBiere Vend.biere%TYPE;
  LePrix Vend.prix%TYPE;
  CURSOR c IS
   SELECT biere, prix FROM Vend
   WHERE bar = 'Joe''s Bar';
```

Exemple:le corps de la procédure JoeAugmente()

```
BEGIN
   OPEN c;
   LOOP
    FETCH c INTO LaBiere,LePrix;
    EXIT WHEN c%NOTFOUND;
    IF LePrix < 3.00 THEN
       UPDATE Vend SET prix = LePrix + 1.00;
       WHERE bar = 'Joe"s Bar' AND biere = LaBiere;
    END IF;
   END LOOP;
   CLOSE c;
END;
```

Sortie de boucle

Attention, mise à jour et non assignation.

Variables Nuplets

- PL/SQL permet qu'une variable *x* ait le type d'un nuplet.
- `x R%ROWTYPE` donne à *x* le type des nuplets de R
- *R* peut être une relation ou un curseur.
- x.a donne la valeur de l'attribut *a* dans le nuplet *x*.

JoeAugmente, version Nuplet

- ```
  CREATE OR REPLACE PROCEDURE
   JoeAugmente() AS
   CURSOR c IS
   SELECT biere, prix FROM Vend
   WHERE bar = 'Joe''s Bar';
   bp c%ROWTYPE;
  ```
  ```

JoeAugmente () version nuplet

```
BEGIN
   OPEN c;
   LOOP
    FETCH c INTO bp;
    EXIT WHEN c%NOTFOUND;
    IF bp.prix < 3.00 THEN
       UPDATE Vend SET prix = bp.prix + 1.00
       WHERE bar = 'Joe"s Bar' AND biere =bp.biere;
    END IF;
   END LOOP;
   CLOSE c;
END;
```