PL/SQL - 6

L3 Miage

ISI3 : Bases de données et applications transactionnelles

Objectif du cours

- **Déclencheurs (trigger)**

Distinguer

- Les **déclencheurs de base de données**
  - Enregistrés dans la bd (côté serveur)
  - Contrôlent la dynamique de la bd
  - Par exemple : quand un tuple est inséré dans une table

- Les déclencheurs d'application
  - Enregistrés avec l'application (côté client)
  - Contrôlent la dynamique de l'interface
  - Par exemple : quand l'utilisateur clique sur un bouton, WHEN-BUTTON-PRESSED

Déclencheurs de Base de Données

- Définition
- Pour quoi faire ?
- Comment les :
  - créer
  - activer
  - désactiver
  - supprimer
- Exemples : application d'une règle de sécurité avec des déclencheurs
- Règles d'utilisation des déclencheurs

Définition

Un déclencheur est une règle, dite active, de la forme :

- Evènement-condition-action
- Trigger : procédure stockée qui est déclenchée automatiquement par des événements et ne s'exécutant que lorsqu'une condition est satisfaite
- Déclencheur de base de données: un ensemble d'instructions qui s'exécutent lorsque le contenu d'une table est modifié par l'intermédiaire d'une instruction INSERT, UPDATE ou DELETE

Déclencheurs - intérêts

Les déclencheurs permettent :

- d'implémenter des règles de sécurité et de gestion sans connaître les endroits où la règle va être mise en œuvre
- de contrôler la cohérence des données contenues dans une table lors de leur modification
- d'effectuer des validations complexes en fonctions de données provenant de tables multiples
- de garder trace des modifications en reportant les valeurs modifiées dans une autre table
- de calculer des valeurs de colonnes par défaut
- de décrire le comportement des composants des interfaces graphiques développées dans Developer Suite

Déclencheurs - intérêts

- Ces règles sont implémentées une fois pour toutes: elles n'ont pas à être vérifiées à chaque utilisation des tables sur laquelle elles portent
  - compatibilité avec le paradigme événementiel
    → pas d'invocation explicite de la règle
    → contrôle systématique de chaque modification des tables
  - facilité de mise à jour et de maintenance
    → code de la règle séparé de l'applicatif de traitement normal
    → contrôle centralisé de toutes les modifications d'une table

Manipulation des déclencheurs BD

- créer
- activer
- désactiver
- supprimer

Syntaxe :

```
CREATE [OR REPLACE] TRIGGER Trigger_name
{BEFORE|AFTER} Triggering_event ON table_name
[FOR EACH ROW]
[FOLLOWS another_trigger]
[ENABLE/DISABLE]
[WHEN condition]
DECLARE …
BEGIN …
EXCEPTION …
END;
```

- [WHEN condition]
  - INSERT : insertion d'une nouvelle ligne dans la table
  - UPDATE : modification des attributs d'une ligne de la table
  - DELETE : suppression d'une ligne de la table
  - une combinaison quelconque des trois

Création des déclencheurs - exemple

```
SELECT * FROM student WHERE rownum < 3 ORDER BY student_id desc;

CREATE OR REPLACE TRIGGER student_bi
BEFORE INSERT ON student
FOR EACH ROW
DECLARE
v_student_id STUDENT.STUDENT_ID%TYPE;
BEGIN
SELECT STUDENT_ID_SEQ.NEXTVAL
INTO v_student_id
FROM dual;
:NEW.student_id := v_student_id;
:NEW.created_by := USER;
:NEW.created_date := SYSDATE;
:NEW.modified_by := USER;
:NEW.modified_date := SYSDATE;
END;
```

Maintenant avec le TRIGGER "before insert" de la table STUDENT (student_bi), au lieu de:

```
INSERT INTO student (student_id, salutation,
first_name, last_name, zip,
registration_date, created_by, created_date,
modified_by, modified_date) VALUES
(STUDENT_ID_SEQ.NEXTVAL, 'Mr.', 'Pedro',
'Perez', '00914', SYSDATE, USER, SYSDATE,
USER, SYSDATE);
```

On peut utiliser:

```
INSERT INTO student (salutation, first_name,
last_name, zip, registration_date) VALUES
('Mr.', 'Pedro', 'Perez', '00914', SYSDATE);
```

Cela est plus simple et pourtant il y a moins de possibilités d'erreur.

Pour tester: SELECT * FROM student WHERE rownum < 2 ORDER BY student_id desc;

Déclencheurs BEFORE et AFTER

- Tout déclencheur BEFORE :
  - est exécuté avant la vérification des contraintes d'intégrité et la mise à jour des valeurs
- Tout déclencheur AFTER :
  - est exécuté après la mise à jour des valeurs
- Si le trigger doit fabriquer une valeur à stocker dans une table il faut utiliser: BEFORE
- Si la modification doit être d'abord terminée il faut utiliser: AFTER

Déclencheur AFTER

Considérons la table STATISTICS: utilisée pour collecter des informations statistiques sur les tables de la BDD.

STATISTICS
TABLE_NAME TRANSACTION_NAME TRANSACTION_USER TRANSACTION_DATE

Le trigger suivante se déclenche après un UPDATE ou un DELETE sur la table INSTRUCTOR

```
CREATE OR REPLACE TRIGGER instructor_aud
      AFTER UPDATE OR DELETE ON INSTRUCTOR
DECLARE
      v_type VARCHAR2(10);
BEGIN
      IF UPDATING THEN
            v_type := 'UPDATE';
      ELSIF DELETING THEN
            v_type := 'DELETE';
      END IF;
      INSERT INTO statistics
      VALUES ('INSTRUCTOR', v_type, USER, SYSDATE);
END;
```

Granularité du trigger

Trigger de niveau instruction

- Déclenché exactement une fois avant (**before**) ou après (**after**) l'exécution complète de l'ordre DML l'ayant provoqué
- Voit la BD avant toute modification si **before** ou après toutes les modifications si **after**
- Un tel trigger voit donc la BD dans un état stable et peut donc consulter toutes les tables y compris celle à laquelle il est attaché

Granularité du trigger

Trigger de niveau ligne

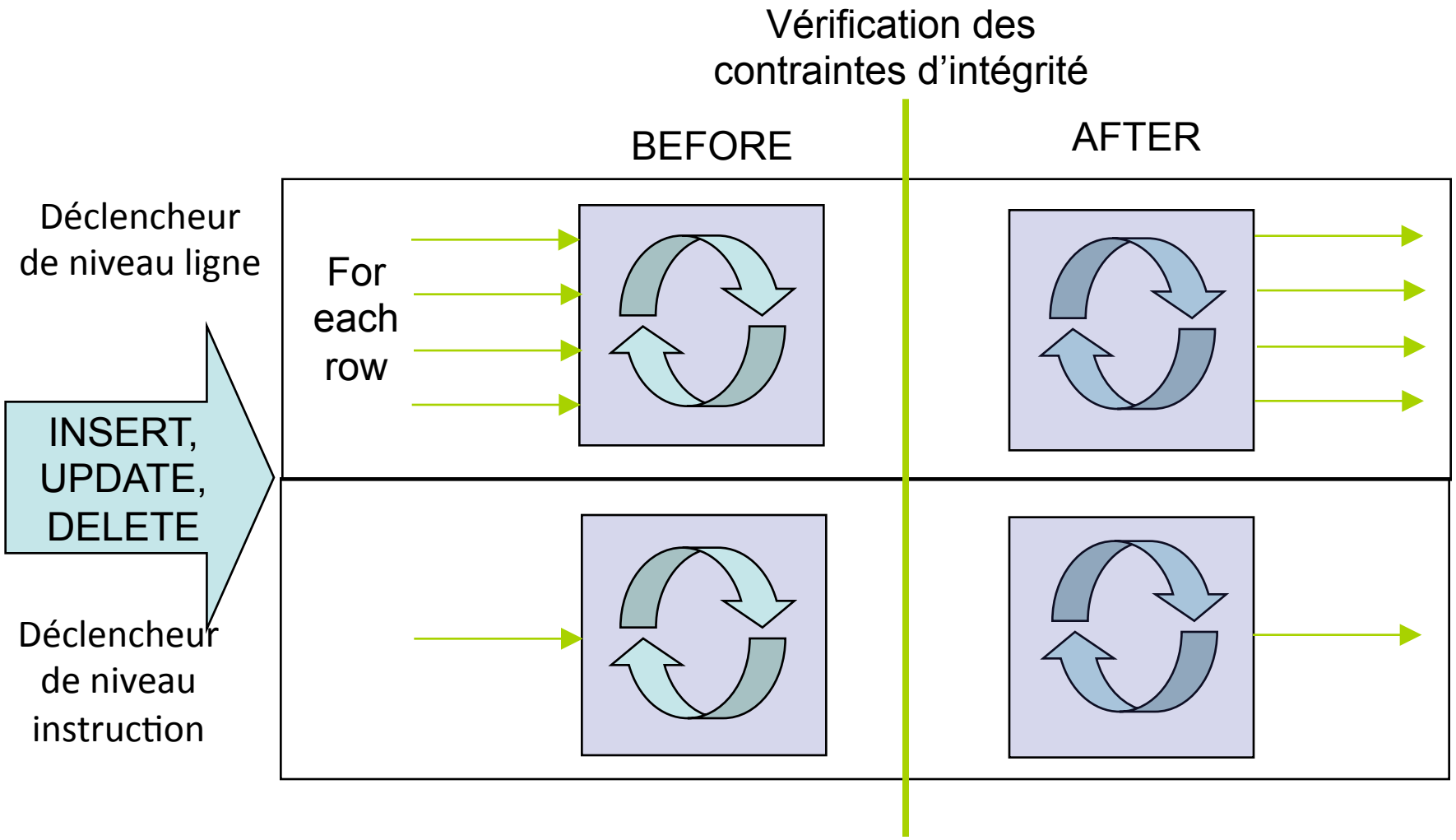
- Déclenché exactement une fois avant (**before**) ou après (**after**) la modification de chaque ligne
- Déclenché autant de fois qu'il y aura de lignes modifiées (éventuellement zéro fois si aucune ligne n'est modifiée)
- Un tel trigger étant exécuté **pendant** l'exécution de l'instruction DML la table en cours de modification est dans un état instable (*mutating table*), le trigger ne peut donc pas la consulter (Oracle déclenche une erreur SQL si on tente de le faire), en revanche il peut consulter toutes les autres tables de la BD

Granularité du trigger - autres détails

- Un déclencheur de niveau instruction (*statement trigger*)
  - ne s'exécute qu'une fois pour un événement particulier
  - ne peut accéder aux valeurs des lignes qui pourraient être modifiées par l'événement déclenchant
  - **est adapté pour retrouver l'auteur ou la date de l'événement déclenchant**

- Un déclencheur de niveau ligne (*row trigger*)
  - **doit inclure la clause** FOR EACH ROW dans l'instruction CREATE TRIGGER
  - est déclenché pour chaque ligne modifiée par l'événement déclenchant
  - peut accéder aux anciennes et aux nouvelles valeurs modifiées par cet événement
  - **est adapté pour mettre en application les règles de gestion et de sécurité (compatibilité)**
  - problème de tables mutantes (*mutating table*)

Granularité du trigger



Trigger de niveau ligne

- Accès aux valeurs des attributs de la ligne modifiée → utilisation de deux variables :old et :new

	:old.<nom d'attribut>	:new.<nom d'attribut>
INSERT	Is null	Valeur insérée
UPDATE	Valeur originale	Nouvelle valeur ou valeur originale si pas de mise à jour
DELETE	Valeur originale	Is null

- WHEN
  - le bloc anonyme ne sera exécuté que si la condition est vraie. Si la condition du **when** est *unknown* le trigger n'est pas déclenché.
  - La condition ne peut utiliser de fonction PL/SQL ni contenir de sous-requête et on doit utiliser les préfixes :old et :new pour accéder aux noms de colonnes de la ligne courante.
  - Intérêt: éviter l'exécution du bloc anonyme lorsqu'inutile (car coûteux)

```
CREATE OR REPLACE TRIGGER enrollment_ad
AFTER DELETE ON ENROLLMENT
...
```

Cet trigger se déclenche après un DELETE sur la table ENROLLMENT. Si le DELETE enlève 1 ou N entrées de la table ENROLLMENT, le trigger s'exécute une seule fois.

```
CREATE OR REPLACE TRIGGER course_au
AFTER UPDATE ON COURSE
FOR EACH ROW
...
```

Dans ce fragment de code, le FOR EACH ROW présent dans le CREATE TRIGGER fait que le trigger est un trigger de ligne. Si un UPDATE affecte 20 lignes dans la table COURSE, ce trigger s'exécute 20 fois.

Les **triggers d'instruction** doivent être utilisés quand les opérations faites par le trigger ne dépendent pas des données de chaque ligne. Par exemple: si vous voulez faire qu'une table soit accessible que à certaines heures et certains jours.

```
CREATE OR REPLACE TRIGGER instructor_biud
BEFORE INSERT OR UPDATE OR DELETE ON INSTRUCTOR
DECLARE
      v_day VARCHAR2(10);
BEGIN
      v_day := RTRIM(TO_CHAR(SYSDATE, 'DAY'));
      IF v_day LIKE ('S%') THEN
            RAISE_APPLICATION_ERROR
            (-20000, 'Pas de modif. les weekends');
      END IF;
END;
```

Dans un **trigger composé**, on peut combiner:

1. Un trigger d'instruction qui s'exécute avant un STATEMENT
2. Un trigger d'instruction qui s'exécute après l'exécution d'un STATEMENT
3. Un trigger de ligne qui s'exécute avant l'affectation de chaque ligne
4. Un trigger de ligne qui s'exécute après l'affectation de chaque ligne

Dans un trigger composé les BEFORE et les AFTER vont dans les sub-triggers et pas dans le trigger principal (on utilise FOR insert, …)

```
CREATE [OR REPLACE] TRIGGER t_name
triggering_event ON table_name
COMPOUND TRIGGER

declaration statements

    BEFORE STATEMENT IS
    BEGIN
        executable statements
    END BEFORE STATEMENT;

    AFTER STATEMENT IS
    BEGIN
        executable statements
    END AFTER STATEMENT;

    BEFORE EACH ROW IS
    BEGIN
        executable statements
    END BEFORE EACH ROW;

    AFTER EACH ROW IS
    BEGIN
        executable statements
    END AFTER EACH ROW;

END t_name;
```

```
CREATE OR REPLACE TRIGGER student_compound
FOR INSERT ON STUDENT
COMPOUND TRIGGER
  v_day VARCHAR2(10);
BEFORE STATEMENT IS
BEGIN
  v_day := RTRIM(TO_CHAR(SYSDATE, 'DAY'));
  IF v_day LIKE ('S%') THEN
  RAISE_APPLICATION_ERROR
  (-20000, 'A table cannot be modified during off hours');
  END IF;
END BEFORE STATEMENT;
BEFORE EACH ROW IS
BEGIN
  :NEW.student_id := STUDENT_ID_SEQ.NEXTVAL;
  :NEW.created_by := USER;
  :NEW.created_date := SYSDATE;
  :NEW.modified_by := USER;
  :NEW.modified_date := SYSDATE;
END BEFORE EACH ROW;
END student_compound;
```

Activation / désactivation / suppression des déclencheurs

- Désactivation de déclencheur - Syntaxe :
  ALTER TRIGGER <nom du déclencheur> DISABLE ;

- Activation de déclencheur - Syntaxe :
  ALTER TRIGGER <nom du déclencheur> ENABLE ;

- Suppression de déclencheur - Syntaxe :
  DROP TRIGGER <nom du déclencheur> ;

Quelques remarques sur les déclencheurs

- Déclencheurs en cascade : un déclencheur réalisant INSERT, UPDATE ou DELETE peut générer des événements amenant à l'exécution d'un ou plusieurs autres déclencheurs, et ainsi de suite; on parle alors de déclencheurs en cascade. ***Evitez plus de deux niveaux de cascade: suivi difficile et risque de boucle infinie***.

- Mutation : tant que les modifications (INSERT, UPDATE ou DELETE) d'une table ne sont pas terminées, celle-ci est dite en cours de mutation. ***Un déclencheur de niveau ligne ne peut lire ou modifier une table en cours de mutation*** (car elle est instable).

- Validation : un déclencheur ne peut ***ni*** exécuter d'instruction COMMIT ou ROLLBACK, ***ni*** appeler de fonction, procédure ou sous-programme de package invoquant ces instructions.

Quelques remarques sur les déclencheurs

- Les déclencheurs ne doivent pas être utilisés lorsqu'il est possible de mettre en place des contraintes d'intégrité (`check`) qui permettent une vérification plus rapide.

- La suppression d'une table entraine la suppression des triggers qui lui sont attachés.

- Visualiser les triggers de votre base de données

```
SQL> select object_type, object_name
    from user_objects
    where object_type = 'TRIGGER';
```