Les Triggers

Un trigger (déclencheur) est un programme qui se déclenche automatiquement suite à un événement. C'est donc un programme qui fait partie du schéma (comme les modules stockés) mais que l'on n'appelle pas explicitement, à la différence d'une procédure stockée. Ils existent dans la plupart des SGBD et sont la traduction d'une vision *active* de la base de données.

Les triggers que nous étudierons dans ce cours sont déclenchés par des instructions DML qui modifient l'instance d'une table :INSERT, DELETE ou UPDATE. (il existe sous Oracle d'autres types de triggers)

Utilisation d'un trigger

Les triggers peuvent servir à :

- vérifier des contraintes que l'on ne peut pas définir de façon déclarative
- gérer de la redondance d'information, après une dénormalisation du schéma.
- collecter des informations sur les mises-à-jour de la base.

Structure d'un trigger

```
CREATE [OR REPLACE] TRIGGER nom_trigger
instant liste_evts
ON nom_table [FOR EACH ROW]
[WHEN ( condition ) ]
corps


instant ::= AFTER | BEFORE
liste_evts ::= evt {OR evt}
evt ::= DELETE | INSERT |
          UPDATE [OF { liste_cols }]
liste_col ::= nom_col { , nom_col }
corps corps de pgme PL/SQL
```

Entête du trigger

On définit :

- la table à laquelle le trigger est lié,
- les instructions du DML qui déclenchent le trigger
- le moment où le trigger va se déclencher par rapport à l'instruction DML (avant ou après)
- si le trigger se déclenche une seule fois pour toute l'instruction (i.e. trigger instruction), ou une fois pour chaque ligne modifiée/insérée/supprimée. (i.e. trigger ligne, avec l'option FOR EACH ROW)
- et éventuellement une condition supplémentaire de déclenchement (clause WHEN)

After ou Before ?

- Si le trigger doit déterminer si l'instruction DML est autorisée, utiliser BEFORE

- Si le trigger doit "fabriquer" la valeur d'une colonne pour pouvoir ensuite la mettre dans la table : utiliser BEFORE
- Si on a besoin que l'instruction DML soit terminée pour exécuter le corps du trigger : utiliser AFTER

Trigger ligne ou instruction ?

Un trigger instruction se déclenche une fois, suite à une instruction DML.

Un trigger ligne (FOR EACH ROW) se déclenche pour chaque ligne modifiée par l'instruction DML.

Par exemple, si une instruction update sur une table T modifie 5 lignes, un trigger instruction lié à cet événement se déclenchera une seule fois (avant ou après la modification en fonction de after/before) et un trigger ligne se déclenchera 5 fois, une fois par ligne modifiée (avant ou après la modification en fonction de after/before).

Dans un **trigger ligne**, on peut faire référence à la ligne courante, celle pour laquelle le trigger s'exécute. Pour cette ligne, on a accès à la valeur avant l'instruction DML (nommée :old) et à la valeur après l'instruction (nommée :new).

	:old	**:new**
insert	null	ligne insérée
delete	ligne supprimée	Null
update	ligne avant modif	ligne après modif

Clause When

On peut définir une condition pour un trigger ligne : le trigger se déclenchera pour chaque ligne vérifiant la condition.

Le trigger suivant insère une ligne dans une table de log, lorsque le salaire d'un employé diminue.

```
Create or replace trigger journal_emp
after update of salary on EMPLOYEE
for each row
when (new.salary < old.salary) -- attention, ici on utilise new et
pas :new
begin
  insert into EMP_LOG(emp_id, date_evt, msg)
  values (:new.empno, sysdate, 'salaire diminué');
end ;
```

Ordre d'exécution des triggers

Pour une instruction du DML sur une table de la base, il peut y avoir 4 familles de triggers possibles selon l'instant (before, after) et le type (instruction ou ligne).

Ces triggers se déclenchent dans l'ordre suivant :

- Triggers instruction BEFORE
- Triggers ligne BEFORE (déclenchés n fois)
- Triggers ligne AFTER (déclenchés n fois)
- Triggers instruction AFTER

Dans une même famille, on ne contrôle pas l'ordre d'exécution des triggers : par exemple, s'il y a plusieurs triggers instructions *after* pour l'instruction update sur une table T, on ne sait pas dans quel ordre ils vont s'exécuter.

Corps du trigger

Le corps du trigger est écrit en PL/SQL. Par rapport à une procédure stockée, on peut utiliser des prédicats qui permettent de savoir quel événement a déclenché le trigger : prédicat INSERTING (resp. UPDATING, DELETING) qui vaut vrai ssi le trigger a été déclenché par un INSERT (resp UPDATE, DELETE).

Nous avons vu précédemment que dans le corps d'un trigger ligne, on peut faire référence aux valeurs de la ligne courante (avant et après modif) par :old et :new.

Il ne faut pas, dans un trigger ligne, interroger une table qui est en cours de modification (pb mutating table). En effet, l'état de la table est *instable* puisque l'instruction SQL est en cours d'exécution.

Exemple : génération automatique d'une clef primaire.

```
create table T1(
  c1 number(3)
  constraint t1_pkey primary key,
  c2 varchar2(20)
);

create sequence seq
increment by 1
start with 1 ;

create or replace trigger clef_auto
before insert
on T1
for each row
when (new.c1 is null)
declare
  la_clef number ;
begin
  select seq.NEXTVAL
  into la_clef from dual ;
  :new.c1 := la_clef ;
end ;

insert into T1(c2) values ('coucou') ;

select * from T1;
C1   C2
----------------------------
1    coucou
```

Ca marche mais rien n'empêche l'utilisateur de mettre une valeur "à lui" pour la clef. Pour éviter ce problème, on peut enlever la clause When. Dans ce cas, on utilisera la séquence quoiqu'il arrive.

On peut aussi imaginer de calculer la clef en fonction des valeurs déjà présentes (sans utiliser de séquence). C'est ce que propose le trigger suivant.

Si on veut calculer la clef en fonction des valeurs déjà présentes :

```
create or replace trigger clef_auto
before insert
on T1
for each row
when (new.c1 is null)
declare
  la_clef number ;
begin
  select nvl(max(c1),0)+1 into la_clef from T1 ;
  :new.c1 := la_clef ;
end ;
Mais cette solution entraîne un problème de table en mutation :
insert into T1(c2) values ('abc');
--> ligne (2,'abc')
insert into T1 values (10,'def');
--> ligne(10,'def')
insert into T1(c2) values ('ghi');
--> ligne(11,'ghi')

insert into T1(c2)
  select c2 from T1bis;

ORA-04091: la table CARON.T1
est en mutation ;
le déclencheur ou la fonction
ne peut la voir
...
```

Voici un exemple de gestion d'un attribut calculé :

```
create table T2(
  clef number(3) constraint t2_pkey primary key,
  le_T1 number(3) constraint t2_t1_fkey references T1,
  blabla varchar2(10)
  );

 -- redondance : compteur du nombre de lignes de T2
 --              qui font référence à cette ligne de T1
alter table T1 add (nb_t2 number(5) default 0 not null);
update T1 set nb_t2 = 0;

select * from t1 ;
C1  C2          NB_T2
----------------------------
1   coucou    0
2   recoucou  0
10  fghfg     0
11  ghgfhkl   0

create or replace trigger coherencet1_nb_t2
before insert
on t2
for each row
begin
  update t1 set nb_t2 = nb_t2+1 where :new.le_t1 = c1 ;
end ;
```

```
insert into t2 values (1,10,'jhgfdlkgjh');

select * from t1;
C1  C2          NB_T2
----------------------------
1   coucou      0
2   recoucou    0
10  fghfg       1
11  ghgfhkl     0

-- faire la meme chose pour delete et update
```

Les instructions permettant de gérer les transactions (commit, rollback, savepoint) sont interdites dans le corps d'un trigger.

Synthèse :

- programmes dont l'exécution est déclenchée par les instruction du DML
- pratiques pour vérifier des contraintes que l'on ne peut pas déclarer dans le schéma.
- Ne pas en abuser :
    - interdépendance entre triggers qui peut être difficile à gérer, cascade de triggers.
    - coût