SQL et Programmation

- 1 - généralités.
- 2 - description.
- 3 - déclencheurs (triggers).
-

1 - Généralités.

- Unicité de la description des données. Description des contraintes d'intégrité.
- Mécanisme de "vues".
- Interrogation et manipulation "non procédurale", expression d'opérations sur des ensembles.
- Indépendance traitement/stockage (logique/physique).
- Deux enjeux "forts": 1 - le maintien de l'intégrité.

  2 - la concurrence d'accès.

2 - Description.

entreprise(<u>nument</u>, noment, budgetStage);
offre_stage (<u>numoffre</u>, #nument, libelle, salaire)

La contrainte peut être violée par la **création** d'une offre de stage , par la **suppression** d'une **entreprise** ou par la **modification** de l'affectation de l'offre de stage.

Option on delete [cascade | set null]

Destinée à imposer un comportement qui assure le maintien de la contrainte de référence.

La **suppression** d'une entreprise impliquera la **suppression** de toutes les **offres** qui sont affectées à l ’entreprise supprimée. (remplacera par "NULL" dans offre_stage toutes les « #nument" égaux à « nument" l’entreprise).

Pour des raisons de performance, nous verrons cela plus loin, **on n'utilisera pas**, sauf exception, **cette description** pourtant commode.

Option initially [immediate | deferred]

Indique si la contrainte doit être vérifiée :

- "immédiatement" après l'instruction qui peut la violer,

- ou si la vérification doit être "**différée**" à la "confirmation" (au moment du "**commit**").

Le défaut d'ORACLE est "**initially immediate** ".

Option deferrable

Permet de choisir le moment de la vérification pour les transactions. Le défaut d'ORACLE est "**not deferrable".**

set constraint [{*nom_contrainte***,**}… | **all**} [**immediate** | **deferred**]

3 - Déclencheur (trigger).

Certaines contraintes ne peuvent être décrites lors de la création des tables
- **Augmentation de Salaire** : Nouveau salaire est toujours supérieur à l'ancien Salaire.
- **Le salaire du chef est supérieur** aux salaires des employés.
- **Gestion de stock** : Enregistrement des produits à commander

Ces contraintes sont vérifiées seulement lors de **la mise à jour des tables.** C'est à dire à des événements de type **insert**, **delete**, **update.**

Nous allons donc créer des **évènements**(ou **trigger**) ou des actions qui sont **déclenchées** lorsqu'un ***insert***, ***delete*** ou un ***update*** se produit sur la table.
Ces **trigger** sont déclenchés soit **avant**(**before**) ou **après**(**after**) un **insert, delete ou update**

Les actions du **trigger** sont **inséparables** de l'action qui a provoqué le déclenchement. **Tout réussit ou tout échoue**.

Create [**or replace**] **trigger** *nom_déclencheur* [**after** | **before**]
[**insert** | [**or**] **delete** | [**or**] **update** [**of** {*nom_domaine*,}…]]
on *nom_table* [**for each row** [**when** *prédicat de qualification*]]
pl/sql_block

Le trigger peut se déclencher sur une modification d'un attribut d'une table avec l'option of

- **drop trigger** *nom_déclencheur***;**
- **alter trigger** *nom_déclencheur* [**enable** | **disable**]**;**
- **alter table** *nom_table* [**enable** | **disable**] **all triggers;**

Le bloc PL/SQL du **trigger** est enregistré dans le dictionnaire de données.

Au moment de leur enregistrement les **triggers** sont compilés.

Un bloc PL/SQL n'est pas une procédure, on ne peut pas lui "passer" d'argument.

3.1 Trigger de type For each row

Par contre chaque ligne modifiée a un état avant et un état après, auxquels on doit faire référence par

:new.*nom_domaine* et **:old.*nom_domaine***.

Exemple :
entreprise(#nument, noment);
offre_stage (#numoffre, nument, libelle, salaire)

Contrainte : le nouveau salaire est toujours supérieur à l'ancien lors de la modification

Trigger se déclenche après un update
Vérification se faite entre **:new.salaire et :old.salaire**

Offre_stage(Num_Offre,…..#nument, salaire,….) ;

create or replace **TRIGGER** modif_salaire **after update on offre_stage** of **salaire for each row**

BEGIN

if **:new.salaire < :old.salaire** then
raise_application_error
(-20001,'viol de la règle :un salaire ne peut etre diminué') ;
end if ;
END;

Le code erreur est toujours entre **20000 et 20999**.

3.2 Trigger et opération induite

On tire les conséquences de modifications faites **sur une table**. C'est donc l'état après qui nous intéresse. Ce sera donc un **trigger after**. Il en faut un pour toutes les modifications de tables qui ont une (ou plusieurs) opération(s) induite(s).

On prendra garde à ne pas faire de modification qui ne change rien.

L'opération induite est "transparente" le trigger ne doit donc lever aucune exception particulière (autres que celles levées par des contraintes d’intégrité violées par les actions du trigger)

T1(**a**, …)
T2(b, **#a**,…..) ; a est une clé étrangère de T2 vers T1

```
CREATE or replace TRIGGER Supp_T1 AFTER
DELETE ON T1
BEGIN
DELETE FROM T2 WHERE a   NOT IN
                                (SELECT a FROM T1);
END;
```

La clé étrangère FK_a de T2 vers T1 doit être **Initially deferred**

Exemple : suppression d'une entreprise implique la suppression des offres qui sont liées.

create or replace trigger tri_sup_entreprise after **delete**

on entrepsie

begin

delete from offre_stage where nument not in

(select nument from entreprise);

end;

Ce **trigger** nécessite que la contrainte de clé étrangère nument dans offre de stage soit **deferred**.

Ce **trigger** est préférable à l'option **on delete cascade**.

3.3 Trigger et contrainte dynamique non décrite

Associer aux opérations sur les tables, les opérations "induites", en particulier celles que l'on doit ou que l'on a choisi de faire pour maintenir l'intégrité.

Exemple : On fera ainsi les opérations de type "cumul" (par exemple le nombre d'heures de vol d'un avion, le nombre max d’employés dans un service, etc...).

On veut vérifier que les modifications faites ne "violent" pas une contrainte, c'est donc l'état après qui nous intéresse. Ce sera donc un **trigger after**. Il en faut un pour chaque modification susceptible de "violer" la contrainte.

SELECT * FROM T1

WHERE Condition C1

Nous recherchons les enregistrements de T1 qui ne

respectent pas la condition C1.

```
CREATE OR REPLACE TRIGGER TRIG3 AFTER
insert or update ON Table2
DECLARE
REC_T1 T1%ROWTYPE;
BEGIN
SELECT * INTO REC_T1 FROM T1
WHERE Condition ;
//1 seul enregistrement trouvé en lève une exception
RAISE_APPLICATION_ERROR (numéro, message);//1
EXCEPTION
WHEN NO_DATA_FOUND THEN NULL;//0 enregist
WHEN TOO_MANY_ROWS THEN
RAISE_APPLICATION_ERROR ( … );// 2 ou pl+
END;
```

On va donc par un **select … into ...** rechercher les lignes qui "violent" la contrainte.

Si la contrainte **est respectée**, On lève l'exception prédéfinie **no_data_found.**

Si un enregistrement est trouvé l'exception est levée juste après le select.

Si plusieurs enregistrements sont trouvés, l'exception prédéfinie **too_many_rows est levée**.

service(numserv, nomservi, budgetStage);
offre_stage (numoffre, #numserv, libelle, salaire)

La condition à respecter : sum(salaire) < budgetStage

Ecrire la requête avant de se lancer sur le trigger

```
select *  from entreprise e
where budgetStage < (select sum(salaire)
                     from offre_stage o
  where o.nument = e.nument);
```

```
declare
ligne_entreprise entreprise%rowtype;
begin
select entreprise.* into ligne_entreprise from entreprise e
where budgetStage < (select sum(salaire) from offre_stage o
      where o.nument = e.nument);
```

ligne_entreprise correspond à un seule enregistrement.

Le **select * into ligne_entreprise** est possible que s'il y a un seul e

0 enregistrement trouvé : **when no_data_found**
2 ou plusieurs : **when too_many_rows**

```
create or replace trigger maj_salaire
after insert or update of salaire on offre_stage
declare
ligne_entreprise entreprise%rowtype;
begin
select entreprise.* into ligne_entreprise from entreprise e
where budgetStage < (select sum(salaire) from offre_stage o
        where o.nument = e.nument);
raise_application_error (...);
exception
when no_data_found then null;
when too_many_rows then
raise_application_error (numero_erreur, message);
end;
```

3.4 Trigger et type Alerte

ARTICLE(**numart**, nomart,stock)
ARTICLE_ALERTE(#**numart**)
Ici nous soutaitons alerter les produits à acheter lorsque
le stock est **< à 10**.
Nous inserons dans une table Article_Alerte les produits à acheter.

```
CREATE or REPLACE TRIGGER gestion_stock AFTER
UPDATE OF STOCK ON ARTICLE FOR EACH ROW   WHEN
( NEW.STOCK < 10) // il n'y a pas les : avant new
BEGIN
INSERT INTO ART_ALERTE
VALUES (:NEW.NUART);
END;
```